Software Engineering

Unit-1: Introduction to Software Engineering & Generic View of Process

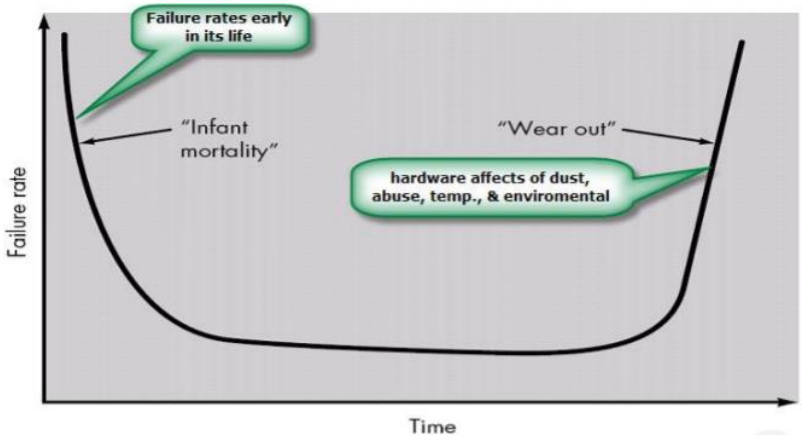
1Q) What is Software? Explain its characteristics & nature.

Definition:

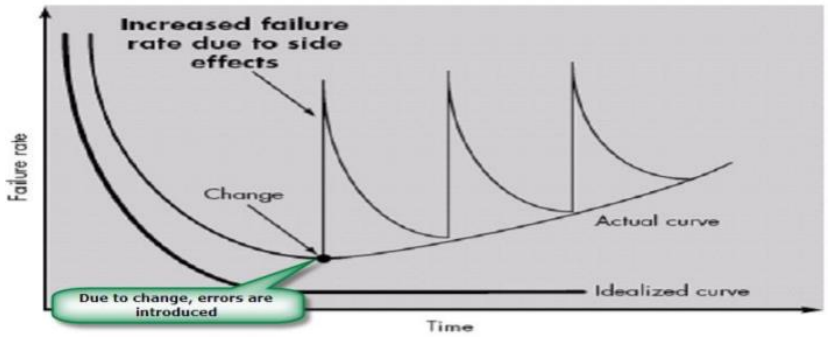
- Software is a set of instructions/programs that when executed provides desired features, functions & performance.
- It is a document that describes the operation & the use of programs.

Characteristics:

1. Software is developed or engineered; it is not manufactured in the classical sense.
   - Concept of raw material is non-existent here.
   - Manufacturing phase for hardware can introduce quality problems that are Non-Existing for software.
2. Software doesn't wear out.
   - Failure curve for Hardware:



     - ✓ Above Bathtub curve indicates that hardware exhibits relatively high failure rates in its early stage.
     - ✓ Defects are then corrected, & failure rate drops to steady-state level for some period of time.
     - ✓ As time passes the failure rate raises again due to external effects like dust, temperature, etc...,
   - Failure curve for Software:



     - ✓ In theory the failure rate curve for software should take the form of "Idealized curve".

- ✓ Software undergoes many changes & updates in its life which leads to spikes in the curve.
- ✓ Before the curve reaches to its steady-state another change is requested.
- ✓ Slowly software becomes worse due to changes.

3. Although the industry is moving towards component-based construction, most software continues to be custom-built.

Nature:

1. **System Software:**
   It is a collection of programs written to service other programs.
   Ex: Compilers, Operating System.
2. **Application Software:**
   It is a program that solves a specific business need.
   Ex: Transaction Processing.
3. **Engineering/Scientific Software:**
   Applications range from astronomy to volcano logy, from automotive stress analysis to space shuttle orbital dynamics, and from molecular biology to automated manufacturing.
   Ex: Computer Aided Design (CAD).
4. **Embedded Software:**
   Software that resides within a product or system and is used to implement & control features & limited functions for the end-user for the system itself.
   Ex: Keypad control for a micro-oven.
5. **Product-Line Software:**
   Designed to provide a specific capability for use by many different customers.
   Ex: Spreadsheet.
6. **Artificial Intelligence Software:**
   It makes use of non-numerical algorithms to solve complex problems that are not easy to compute or straight forward analysis.
   Ex: Used in Robotics, Neural Networks, Pattern Recognition.

2Q) Explain Software Myths?

Definition:

Beliefs about software & the process used to build it. Misleading Attitudes causes serious problem for managers & technical people.

Management Myths:

Managers are often under pressure to maintain budgets, keep schedules on time & improve quality.

Myth-1:

We already have a book that's full of standards and procedures for building software, won't that provide my people with everything they need to know?

Reality:

Whether people know about that book or even if they know have the studied it?

Myth-2:

If we get behind schedule, we can add more programmers and catch up.

Reality:

How well the new added programmers synchronize with the current programmers & understand the flow of code till now?

Myth-3:

If I decide to outsource the software project to a third party, I can just relax and let that firm build it.

Reality:

What if third party does the same or what if they deploy late than the project completion time?

Customer Myths:

Customer may be a person from inside or outside the company that has requested software under contract.

Myth-1:

A general statement of objectives is sufficient to begin writing programs. We can fill in the details later.

Reality:

Incomplete knowledge of user requirements may lead to any other application rather than the required one.

Myth-2:

Project requirements continually change, but change can be easily accommodated because software is flexible.

Reality:

What if the required new change leads to complete change in software structure?

Practitioner's Myths:

Practitioner's are the one who make day-to-day design decisions about the project.

Myth-1:

Once we write the program & get it to work, our job is done.

Reality:

The sooner you begin writing the code, the longer it will take you to get done.

Myth-2:

Software engineering will make us create unnecessary documentation & will slow us down.

Reality:

Software engineering is not about creating documents. It is about creating quality. Better quality leads to reduced rework. And reduced rework results in faster delivery times.

Myth-3:

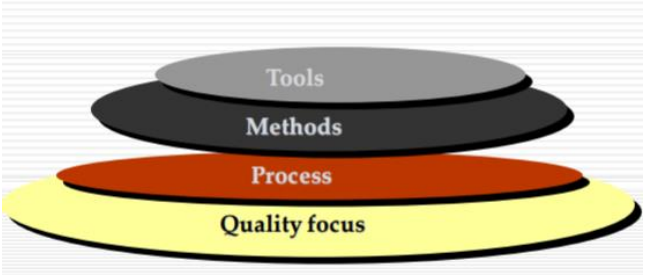
Until I get the program running, I have no way of accessing its quality.

Reality:

One of the most effective software quality assurance mechanisms can be applied from the inception of a project.

3Q) Explain Software Engineering – A Layered Technology?

Software engineering is the establishment and use of sound engineering principles in order to obtain a software that is economically reliable & works efficiently on real machines.



Quality Focus:

- Every organization is rest on its commitment to quality.
- The bedrock that supports software engineering is a quality focus.
- Total quality management, six sigma etc…, methods leads to effective development.

Process:

- The processes define the tasks to be performed and the order in which they are to be performed
- It's a foundation layer for software engineering.
- It's a framework for a set of Key Process Areas (KRA) to effectively manage & deliver quality software in a cost-effective manner.

Methods:

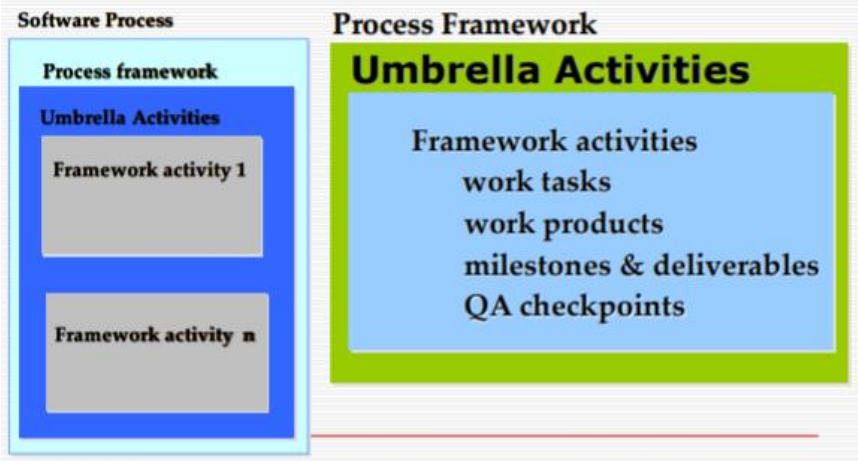
- It provides the technical how-to's for building software.
- There could be more than one technique to perform a task and different techniques could be used in different situations.
- Methods encompass a broad array of tasks that includes:
    1. Requirements analysis
    2. Design
    3. Program construction
    4. Testing
    5. Support

Tools:

- Provide automated or semi-automated support for the process, methods and quality control.
- When tools are integrated so that information created by one tool can be used by another

4Q) Explain Process Framework?

It establishes the foundation for a complete software process by identifying a small number of "framework activities" that are applicable to all software projects, regardless of their size or complexity.



Process Model Adaptability:

The task for each activity will vary based on:

- Type of project
- Characteristics of project
- Common sense judgement
- Concurrence of the project team

Generic Process Framework Activities:

- **Communication:**
  - Heavy communication with customers, stakeholders, team.
  - Encompasses requirements gathering & related activities.
- **Planning:**
  - Workflow that is to follow.
  - It describes: Technical tasks to be conducted, Risks that are likely to occur, Resources that will be required, work schedule.
- **Modelling:**
  - Creation of models that allows the developer & customer to better understand S/W requirements & design of software.
- **Construction:**
  - Code generation: either manual or automated or both.
  - Testing: to uncover error in the code.
- **Deployment:**
  - Delivery to the customer for evaluation.
  - Customer provides feedback.

Umbrella Activities:

- **Software project tracking & control:**
  - Assessing progress against the project plan.
  - Take adequate action to maintain schedule.

- **Formal Technical Reviews:**
  - Flow of developing is validated by technical reviews.
- **Software Quality Assurance:**
  - Define & conducts the activities required to ensure software quality.
- **Software Configuration Management:**
  - Manages the effects of change.
- **Document Preparation & Production:**
  - Help to create work products such as models, documents, logs, form & list.
- **Reusability Management:**
  - Configuration, maintenance, upgrade issues should be explicitly managed in proper reusable point of view.
- **Measurement:**
  - Define and collects process, project, and product measures
  - Assist the team in delivering software that meets customer's needs.
- **Risk Management:**
  - Assesses risks that may affect that outcome of project or quality of product (i.e. software)

5Q) Explain CMMI Levels?

The **Capability Maturity Model Integration** developed by SEI defines each process area in terms of "specific goals" & "specific practises" required to achieve these goals.

Level 0 (Incomplete):

- Process are not performed or not achieve all the goals and objectives defined by the CMMI for level-1 capability.

Level 1 (Performed):

- All specific goals are performed as per defined by CMMI.

Level 2 (Managed):

- All level-1 criteria have been satisfied
- In addition to Level-1:
  - People doing work have access to adequate resources to get job done.
  - Stakeholders are actively involved.
  - Work tasks and products are monitored, controlled, reviewed, and evaluated for conformance to process description.

Level 3 (Defined):

- All level-2 criteria have been achieved.
- In addition to Level-2:
  - Management and engineering processes documented.
  - Standardized and integrated into organization-wide software process.

Level 4 (Quantitatively Managed):

- All level-3 criteria have been satisfied.
- Software process and products are quantitatively understood.
- Controlled using detailed measures and assessment.

Level 5 (Optimized):

- All Level-4 criteria have been satisfied.
- Continuous process improvement is enabled by quantitative feedback from the process and testing innovative ideas.

Software Engineering

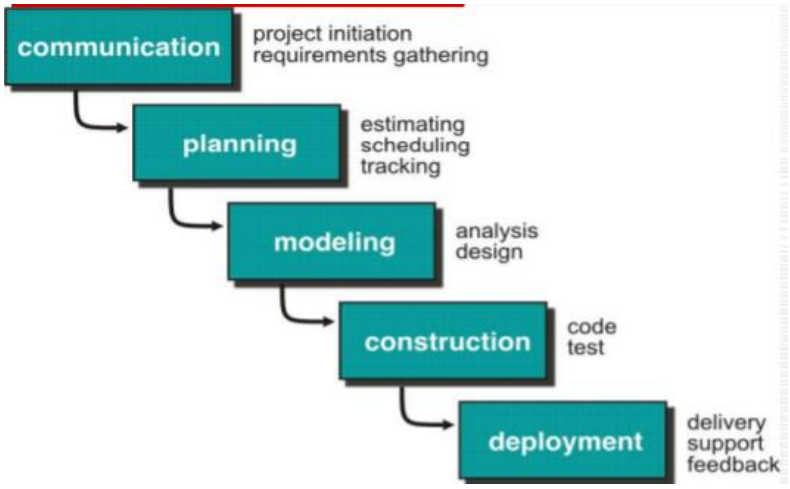
Unit-2: Process Models, Software Requirements & Req.Engg Process

1Q) Describe Process Models?

- Process models defines a distinct set of activities, actions, tasks, milestones, work products that are required to engineer high-quality software.
- Though these process models are not perfect, they provide a useful road map for software engineering work.

Types of Models:

1. **Waterfall Model:**
   - It is also called as classical life cycle.
   - It suggests a systematic sequential approach to software development.



   - **Limitations:**
     - The model implies that present stage must be completed before moving into next stage.
     - Customers cannot use until the entire system is complete.
     - Surprises at the middle or end are very expensive.
     - Some teams sit ideal for other teams to finish their part.
2. **Incremental Process Model:**
   - 2.1. **Incremental Process Model:**
     - Rather than delivering the system as a single delivery, the development and delivery is broken down into increments with each increment delivering part of the required functionality.
     - User requirements are prioritised and the highest priority requirements are included in early increments.
     - First Increment is often core product
       - Includes basic requirement
       - Many supplementary features (known & unknown) remain undelivered
     - A plan of next increment is prepared.
       - Modifications of the first increment.
       - Additional features of the first increment

- **Limitations:**
  - Particularly used when enough staffing is not available & deadline is given.
  - Difficult for the customer to state all the requirements explicitly.
  - Focus more on delivery of operation product with each increment.
  - Once the development is started, requirements are freeze.
  - Customer must have patience.

2.2. RAD Model:

- It is an incremental software process model that emphasizes a short development cycle.
- It enables a development team to create a fully functional system within a very short time period.
- Communication & planning are initially done by the whole group.
- Once planning is done, modelling & construction are done by splitting into several teams.
- Each team works on developing a separate functionality.
- Before deployment all the functionalities are Integrated, validated & deployed as a whole system.
- **Limitations:**
  - Sufficient Human Resources are required.
  - If the developers and customers are not committed to rapid frame, RAD projects will fail.
  - System may not be properly modularized.
  - RAD may not be applicable when technical risks are high (heavy use of new technology).

3. Evolutionary Model:

- These are iterative.
- Produce an increasingly more complete version of the software with each iteration.

3.1. Prototyping Model:

- It is used when objectives defined by customer are general but does not have details like input, processing, output requirement.
- It is also used when developer is unsure of the efficiency of the algorithm.
- Procedure includes developing a prototype which as a whole represents the working model of original application.
- Once the prototype meets the user requirements original software is constructed & deployed.
- **Limitations:**
  - The customer sees for a working version of software without considering overall S/W quality.
  - The developer often makes implementation compromises in order to get prototype working quickly.

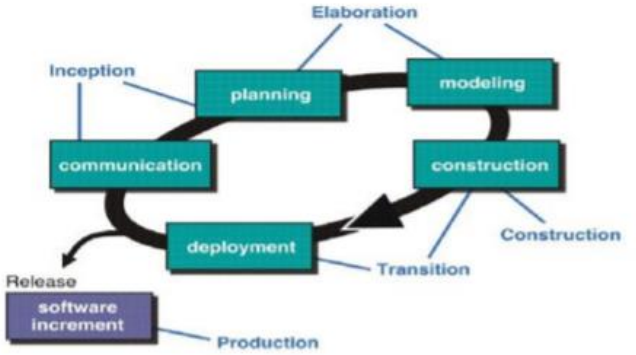
3.2. Spiral Model:

- This model is similar to the Incremental Model with more emphasis placed on risk analysis.
- It has 4 phases: Planning, Risk analysis, Engineering, Evaluation.
- Software project repeatedly passes through these 4 phases in iterations (called spirals).
- Each subsequent spiral upgrades the features based on baseline spiral.
- **Limitations:**
  - Used when project is large.
  - Not applicable when there is fixed budget.
  - If major risks are not encountered & managed, problems will undoubtedly occur in further spirals also.

3.3. Concurrent – Development Model:

- It represented schematically as series of major technical activities, tasks, and their associated states.
- It is often more appropriate for system engineering projects where different engineering teams are involved.
- Each activities, actions and tasks on the network exists simultaneously with other activities, actions and tasks.

4. Unified Process:



- Today, the trend in software is towards bigger & more complex systems.
- We want the software that is better adapted to our needs but that in turn makes the software more complex.
- **Phases Involved:**
  1. **Inception Phase:**
     Deals with customer communication, planning activities & emphasize the development & refinement of use-cases as a primary model.
  2. **Elaboration Phase:**
     Deals with modelling activities focusing on the creation of analysis & design models on class definitions & architectural representations.
  3. **Construction Phase:**
     Refines and translates the design model into implemented software components.

4. **Transition Phase:**
   Transfers the software from the developer to the end-user for beta testing and acceptance.
5. **Production Phase:**
   A production phase in which on-going monitoring and support are conducted.

2Q) Describe Functional & Non-Functional Requirements?

Functional Requirements:

- Describes what the system should do.
- Describes the Inputs & Outputs.
- What are the computations needed to be done?
- What data the system should store?
- Functional requirements must be complete (i.e., all the services req. by the user should be defined).
- Functional requirements must be consistent (i.e., requirements should not have contradictory definitions.
- In practical it is impossible to achieve requirements completeness & consistency.
- Ex: **LIBRARY SYSTEM**
  - ✓ The user shall be able to search either all of the initial set of databases or select a subset from it.
  - ✓ Every order shall be allocated by a unique identifier [order_id].

Non-Functional Requirements:

- Constraints that must be accomplished like:
  - Response Time
  - Throughput
  - Reliability
  - Availability
  - Security
  - Recover from failure
  - Technology to be used
  - Cost
- Classifications:
  1. **Product requirements:**
     Requirements which specify that the delivered product must behave in a particular way
     Ex: Execution speed, Reliability.
  2. **Organisational Requirements:**
     Requirements which are a consequence of organisational policies and procedures
     Ex: Process standards used, Implementation requirements.

3. **External Requirements:**
   Requirements which arise from factors which are external to the system and its development process
   Ex: Interoperability requirements, Legislative requirements.

- Ex: **LIBRARY SYSTEM**
  - Product Requirements:
    Software shall be implemented as simple HTML without frames or Java applets.
  - Organisational Requirements:
    Document shall confirm the process & deliverables according to standards.
  - External Requirements:
    System shall not dispose any personal info about system users apart from their name & library reference number to the staff.

3Q) Describe User & System Requirements?

User Requirements:

- Should describe functional & non-functional requirements so that they are understandable by system users who don't have detailed technical knowledge.
- These are defined using natural language, tables & diagrams.

System Requirements:

- More detailed specifications of user requirements.
- Serves as a basis for designing the system.
- May be used as part of the system contract.
- May be expressed using system models.

Guidelines for writing requirements:

- Invent a standard format & use it for all requirements.
- Use language in a consistent way.
- Use "shall" for mandatory requirements.
- Use "should" for desirable requirements.
- Use text-highlighting to identify key parts of the requirement.
- Avoid the use of computer jargon.

Problems with Natural Language:

- **Lack of Clarity:**
  Precision is difficult without making the document difficult to read.
- **Requirements Confusion:**
  Functional & Non-functional requirements tend to be mixed up.
- **Requirements amalgamation:**
  Several different requirements may be expressed together.

- **Lack of Modularisation:**
  Inadequate to structure system requirements.
- **Over-flexibility:**
  The same thing may be said in a number of different ways in the specification.

Alternatives to Natural Language:

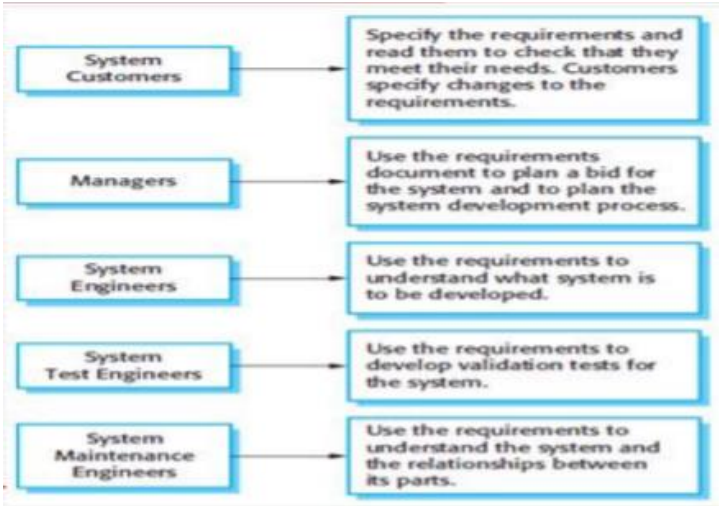
Notation	Description
Structured natural language	This approach depends on defining standard forms or templates to express the requirements specification.
Design description languages	This approach uses a language like a programming language but with more abstract features to specify the requirements by defining an operational model of the system.
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system. Use-case descriptions (Jacobsen, Christerson et al., 1993) are one technique.
Mathematical specifications	These are notations based on mathematical concepts such as finite-state machines or sets. These unambiguous specifications reduce the arguments between customer and contractor about system functionality. However, most customers don't understand formal specifications and are reluctant to accept it as a system contract.

4Q) Describe Software Requirements Document (SRS)?

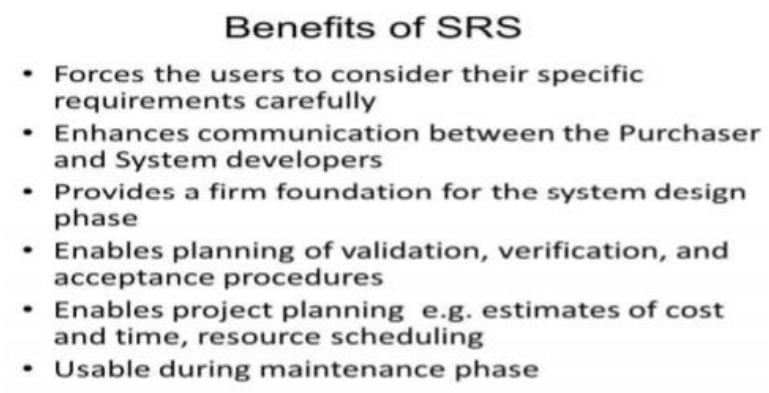
Definition:

- The requirements document is the official statement of what is required of the system developers.
- Should include both a definition and a specification of requirements.
- It is not a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it

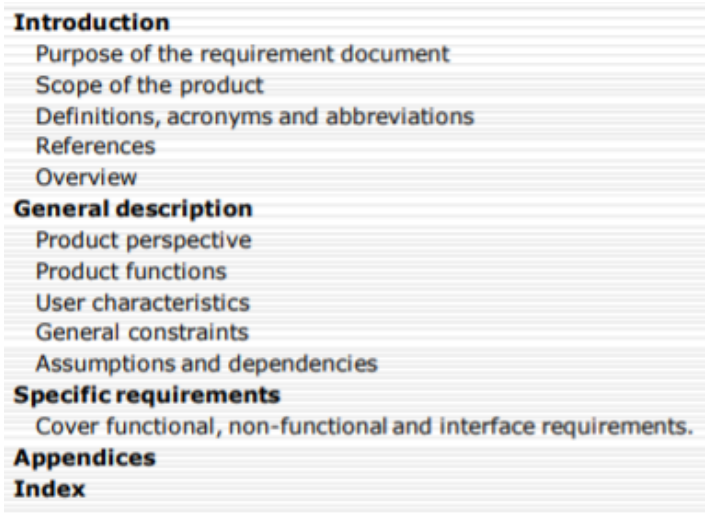
Users of SRS:



Benefits of SRS:



IEEE Requirements Standard:



5Q) Explain Requirement Analysis, Requirement Elicitation, Requirement Validation, Requirement Management, Feasible Studies?

Requirement Analysis:

- It is essential that the software engineering team understand the requirements of a problem before the team tries to solve the problem.
- RE is software engineering actions that start with communication activity and continues into the modelling activity.
- RE establishes a solid base for design and construction. Without it, resulting software has a high probability of not meeting customer needs.
- Characteristics of a good requirement:
    - Clear & Unambiguous
    - Correct
    - Understandable
    - Verifiable
    - Complete
    - Consistent
    - Traceable

Software Engineering

Requirement Elicitation:

- Requirements elicitation is the practice of collecting the requirements of a system from users, customers and other stakeholders.
- Requirement Elicitation Techniques:
    - **Viewpoint-Oriented Elicitation:**
      There is no single correct way to analyse the requirements. Hence it is required to look at a problem in different ways.
    - **Ethnography:**
      Identifies implicit system requirements.
    - **Scenarios:**
      People can relate to these more easily than an abstract statement of what they require from a system.
    - **Structured Analysis Methods:**
      Consists of structured & systematic analysis of requirements.
    - **Prototyping:**
      Requirements are tested by a prototype before building original software.

Requirement Validation:

- It is a process in which it is checked that whether the gathered requirements represent the same system that customer really wants.
- Errors are fixed.
- Types:
    - Validity
    - Consistency
    - Completeness
    - Realism
    - Verifiability
- Techniques:
    - **Requirement Reviews:**
      Systematic manual analysis of the requirements.
    - **Prototyping:**
      Using an executable model of the system to check requirements.
    - **Test-Case generation:**
      Developing tests for requirements to check testability.

Requirement Management:

- Requirements management is the process of managing changing requirements during the requirements engineering process and system development.
- Requirements are inevitably incomplete and inconsistent.
- Requirements begin with identification.
- Each requirement is assigned a unique identifier.
- Once requirement have been identified, traceability table are developed.

Software Engineering

- There are 3 principal changes to a change management process.
  - Problem analysis & change specification
  - Change analysis & costing
  - Change Implementation

Feasible Studies:

- It is an analysis of the viability of an idea from its logical beginning to logical end.
- Types:
  - **Technical Feasibility:**
    - Does our current Technology support the proposed solution?
    - Can we support & maintain the system when it is in use?
  - **Legal & Regulatory Feasibility:**
    - What restrictions have been introduced by company law, auditors?
    - What laws must be observed in terms of health & safety, data protection & working hours?
  - **Organizational Feasibility:**
    - Is the organization capable of adopting new process?
    - Is the organization capable of accepting changes involved in making decisions?
  - **Social Feasibility:**
    - Are employees willing to accept changes in work conditions?
    - Are employees willing to accept changes in power structure?
  - **Economic Feasibility:**
    - Can we afford the system?
    - What economic benefits will the system provide?

Software Engineering

Unit-4: Design Engineering & Testing Strategies

1Q) Explain Design Process & Design Quality

Significance:

- Design creates a representation or model of the software.
- Quality is established during design.
- Design sets the stage for construction.
- Design model provides details about S/W architecture, interfaces & components that are necessary to implement the system.

Quality Guidelines:

- Uses recognizable architectural styles or patterns.
- Modular.
- Appropriate Data Structure for the classes to be implemented.
- Interfaces that reduces complexity of connection.
- Repeatable method.

Functionality:

- Set of features & capabilities of programs.
- Security of the overall system.

Usability:

- User-friendliness
- Aesthetics
- Consistency
- Documentation

Reliability:

- Evaluated by measuring the frequency and severity of failure
- MTTF (Mean Time to Failure)
- MTTR (Mean Time to Repair)

Performance:

- Measured by:
    - Processing speed
    - Response time
    - Resource consumption
    - Throughput
    - Consumption

Supportability:

- Extensibility
- Adaptability
- Serviceability

Software Engineering

2Q) Explain Design Concepts?

1. **Abstraction:**
   - **Highest level of Abstraction:**
     Solution is stated in broad terms using the language of the problem environment.
   - **Lower levels of Abstraction:**
     More detailed description of the solution is provided.
   - **Procedural Abstraction:**
     Refers to a sequence of instructions for a specific task.
   - **Data Abstraction:**
     Named collection of data that describes a data object.
2. **Architecture:**
   - Structure organization of program components & their interconnection.
   - Models:
     - **Structural Models:**
       Organized collection of program components.
     - **Framework Models:**
       Represent the design in more abstract way.
     - **Dynamic Models:**
       Represents the behaviour aspects.
     - **Process Models:**
       Focus on the design of technical process.
3. **Patterns:**
   - Helps to determine the following:
     - Whether the pattern is applicable to the current work?
     - Whether the pattern can be reused?
4. **Modularity:**
   - Divides software into separately named & addressable components called modules.
   - Modules are integrated to satisfy problems.
   - The complexity of two problems when they are combined is often grater than the sum of the perceived complexity when each is taken separately.
   - Based on Divide & Conquer strategy it is easier to solve a complex problem when broken into sub-modules.
5. **Information Hiding:**
   - Information contained within a module is inaccessible to other modules who do not need such information.
   - Provides the greatest benefits when modifications are req. during testing.
   - Errors introduced during modification are less likely to propagate to other locations.
6. **Functional Independence:**
   - Independence is assessed by two quantitative criteria:

- **Cohesion:** Performs a single task requiring little interaction with other components
- **Coupling:** Measure of interconnection among modules.

- Coupling should be low & cohesion should be high for good design.

7. **Refinement:**
   - Process of elaboration from high level abstraction to the lowest level abstraction.
   - It causes the designer to elaborate providing more & more details at successive level of abstractions.
8. **Refactoring:**
   - Organization technique that simplifies the design of components.
   - Examines for redundancy, unused design elements & inefficient or unnecessary algorithms.
9. **Design Classes:**
   - Class represents a different layer of design architecture.
   - There are 5 types of Design classes
     - User-Interface Class
     - Business Domain Class
     - Process Class
     - Persistent Class
     - System Class

3Q) Explain Design Model?

1. **Data Design Elements:**
   - It creates a model of data that is represented at a high level of abstraction.
   - Translation of data model into a data base is pivotal.
2. **Architectural Design Elements:**

   Derived from 3 sources:
   - Info about the application domain of the software
   - Architectural pattern & analysis
   - Dataflow diagrams.
3. **Interface Design Elements:**

   Set of detailed drawings constituting:
   - User Interface
   - External Interface
   - Internal Interface
4. **Component Level Design Elements:**
   - Fully describes the internal details of each software component
   - UML diagrams can be used
5. **Deployment Level Design Elements:**
   - Indicated software functionality & physical computing environment.
   - UML deployment diagram is developed.

Software Engineering

4Q) Explain Testing Strategies?

- **Unit Testing:**
  It focuses on each unit or module which is implemented in source code.
- **Integration Testing:**
  It focuses on the construction & design of the software.
- **Validation Testing:**
  Validates the functional, behavioural & performance of system against its construction. It includes:
  1. Validation Test Criteria
  2. Configuration Review
  3. Alpha & Beta Testing
- **System Testing:**
  It focuses on performance of entire system. It includes:
  1. Recovery Testing
  2. Security Testing
  3. Stress Testing
  4. Performance Testing
- **Testing Approaches:**
  - When functionality is being tested without taking the actual implementation in concern is known as **Functionality Testing or Black-Box Testing.**
  - When both functionality & implementation is analysed it is known as **Implementation Testing or White-Box Testing.**
- **Black-Box Testing:**
  - It is carried out to test functionality of the program.
  - It is also called Behavioural Testing.
  - The test in this case has a set of input values & respected desired results. When output values are matched with desired results the program is validated as "OK" or else it validates as "Problematic".
  - In this method, the design & structure of the code are not known to the tester.
  - Techniques:
    1. **Equivalence Class:**
       The input is divided into similar classes. If one element of a class passes the test, it is assumed that all the classes are passed.
    2. **Boundary Values:**
       The input is divided into higher & lower end values. If these values pass the test, it is assumed that all the values in between them may pass too.
    3. **Cause-Effect Graphing:**
       Cause (Input) – Effect (Output) is a testing technique, where combinations of input values are tested in systematic way.
    4. **Pair-Wise Testing:**
       In this testing multiple parameters are tested pair-wise for their different values.

5. **State based Testing:**

   The system changes state on provision of input. These systems are tested based on their states & input.

- **White-Box Testing:**
  - It is carried out to test both functionality & implementation.
  - It is also known as Structural Testing.
  - In this method, the design & structure of the code are known to the tester.
  - Techniques:
    1. **Control-Flow Testing:**
       - The purpose of this testing is to set up test cases which covers all the statements & branch conditions.
       - The branch conditions are tested for both being true & false, so that all statements can be covered.
    2. **Data-Flow Testing:**

       This technique emphasis to cover all the data variables included in the program.
- **Art of Debugging:**

  It occurs as a consequence of successful testing. It includes:
  1. Brute Force method
  2. Back Tracking
  3. Cause Elimination

Software Engineering

Unit-5: Product Metrics & Metrics for Process & Projects

1Q) Explain Software Quality Metrics.

Software metrics can be classified into 3 categories:

1. **Product Metrics:**
   Describes the characteristics of the product such as size, complexity, design features, performance, and quality level.
2. **Process Metrics:**
   These characteristics can be used to improve the development and maintenance activities of the software.
3. **Project Metrics:**
   This metrics describe the project characteristics and execution.

Software quality metrics:

- subset of software metrics that focus on the quality aspects of the product, process, and project.
- These are more closely associated with process and product metrics than with project metrics.
- It is further classified into 3 categories:
  - **Product Quality Metrics:**
    - ✓ **Mean Time to Failure:**
      It is the time between failures.
      Ex: airline, traffic control systems, weapons.
    - ✓ **Defect Density:**
      It measures the defects relative to the software size expressed as lines of code or function point, etc. i.e., it measures code quality per unit.
      Ex: Commercial software systems.
    - ✓ **Customer Problems:**
      It measures the problems that customers encounter when using the product.
    - ✓ **Customer Satisfaction:**
      Customer satisfaction is often measured by customer survey data through the five-point scale
      - Very satisfied
      - Satisfied
      - Neutral
      - Dissatisfied
      - Very dissatisfied
  - **In-Process Quality Metrics:**
    In-process quality metrics deals with the tracking of defect arrival during formal machine testing for some organizations.

This metric includes:

- Defect density during machine testing
- Defect arrival pattern during machine testing
- Phase-based defect removal pattern
- Defect removal effectiveness

- **Maintenance Quality Metrics:**

  Although much cannot be done to alter the quality of the product during this phase, following are the fixes that can be carried out to eliminate the defects as soon as possible with excellent fix quality.

  - Fix backlog and backlog management index
  - Fix response time and fix responsiveness
  - Percent delinquent fixes
  - Fix quality

2Q) Explain Metrics for analysis model, design model, source code.

- **Metrics for Analysis Model:**

  These metrics are used to examine the analysis model with the objective of predicting the size of the resultant system. Size acts as an indicator of increased coding, integration, and testing effort; sometimes it also acts as an indicator of complexity involved in the software design.

  Types:

  a. **Function Point Metric:**

     Used to measure the functionality delivered by the system, estimate the effort, predict the number of errors, and estimate the number of components in the system.

  b. **Lines of Code:**

     LOC can be defined as the number of delivered lines of code, excluding comments and blank lines.

- **Metrics for Design Model:**

  Metrics for design modelling allows developers or software engineers to evaluate or estimate quality of design and include various architecture and component-level designs.

  Types:

  a. **Metrics by Glass & Card:**

     It is further classified into 3 types:

     1. **Structural Complexity:**

        Structural complexity depends upon fan-out for modules.

     2. **Data Complexity:**

        Data complexity is complexity within interface of internal module.

     3. **System Complexity:**

        System complexity is combination of structural and data complexity.

Software Engineering

- **Metrics for Source Code:**

  Source code metrics are essential components in the software measurement process. They are extracted from the source code of the software, and their values allow us to reach conclusions about the quality attributes measured by the metrics.

Software Engineering

Unit-6: Risk Management & Quality Management

1Q) Reactive vs Proactive Risk management?

Reactive Risk Management:

- Reactive risk management is often compared to a firefighting scenario.
- The reactive risk management kicks into action once an accident happens, or problems are identified after the audit.
- The accident is investigated, and measures are taken to avoid similar events happening in the future.
- Further, measures will be taken to reduce the negative impact the incident could cause on business profitability and sustainability.
- Reactive risk management can cause serious delays in a workplace due to the unpreparedness for new accidents.
- The unpreparedness makes the resolving process complex as the cause of accident needs investigation and solution involve high cost, plus extensive modification.

Proactive Risk Management:

- Contrary to reactive risk management, proactive risk management seeks to identify all relevant risks earlier, before an incident occurs.
- Proactive risk management can also be defined as Adaptive, closed loop feedback control strategy.
- Though, humans are the source of error, they can also be a very important safety source as per proactive risk management.
- Further, the closed loop strategy refers to setting up of boundaries to operate within.
- These boundaries are considered to have safe performance level.

Differences:

Reactive Risk Management	Proactive Risk Management
Definition: Strategy that takes action after the accident occurs.	**Definition:** Strategy that takes action before the accident occurs.
Purpose: Reactive risk management attempts to reduce the tendency of the same or similar accidents which happened in past being repeated in future.	**Purpose:** Proactive risk management attempts to reduce the tendency of any accident happening in future by identifying the boundaries of activities before lead to an accident.
Timeframe: Reactive risk management solely depends on past accidental analysis and response.	**Timeframe:** Proactive risk management combines a mixed method of past, present and future prediction before finding solutions to avoid risks.

Flexibility: Reactive risk management does not accommodate prediction, creativity, and problem-solving ability of humans in its approach which makes it less flexible to changes and challenges.	**Flexibility:** Reactive risk management accommodate prediction, creativity, and problem-solving ability of humans in its approach which makes it more flexible to changes and challenges.

2Q) Explain Software Risk?

- It involves 2 characteristics:
  - **Uncertainty:** Risk may or may not happen
  - **Loss:** Risk becoming a reality
- Types:

1. **Project Risks:**
   Threaten the project plan and affect schedule and resultant cost.
   Ex: Project schedule slips away.
2. **Technical Risks:**
   Threaten the quality and timeliness of software to be produced.
   Ex: Maintenance problems, Interface problems.
3. **Business Risk:**
   These risks effect the success of software.
   Types:
   - Market Risk
   - Strategic Risk
   - Sales Risk
   - Management Risk
   - Budget Risk
4. **Known Risk:**
   These risks can be recovered from careful evaluation.
   Ex: Time complexity of an algorithm.
5. **Predictable Risk:**
   Risks are identified by past project experience.
   Ex: poor communication with customer.
6. **Unpredictable Risk:**
   Risks that occur and may be difficult to identify.
   Ex: Failure of whole software.

3Q) Explain Risk Identification, Risk projection, Risk refinement?

Risk Identification:

- Risk Identification is an attempt to specify threats to the project plan.
- By identifying known & predictable risks the project manager takes steps for controlling them as well as avoiding them.
- One of the methods for identifying risks is to create a risk item check list. Check list includes:
  - Product size

- Business Impact
- Customer Characteristics
- Development environment
- Technology to be built
- Staff size experience

Risk Projection:

- Steps in Risk projection

1. Estimate L_i for each risk
2. Estimate the consequence X_i
3. Estimate the impact
4. Draw the risk table
5. Ignore the risk where the management concern is low i.e., risk having impact high or low with low probability of occurrence
6. Consider all risks where management concern is high i.e., high impact with high or moderate probability of occurrence or low impact with high probability of occurrence

30

Risk Refinement:

- Also called Risk Assessment.
- Refines based on the following three factors:
  1. Nature
  2. Scope
  3. Timing
- It is based on Risk Elaboration.
- **Formula: Risk Exposure (RE) = P*C**
  Where; P = Probability, C = Cost of project if risk occurs.

4Q) Explain RMMM?

- RMMM stands for Risk Mitigation Monitoring and Management.
- Its goal is to assist project team in developing a strategy for dealing with risk.
- **Mitigation:**
  - How can we avoid the risk?
  - Proactive planning for risk avoidance.
- **Monitoring:**
  - What factors enable us to determine if the risk is becoming more or less likely?
  - Assessing whether predicted risk occur or not?
  - Ensuring risk aversion steps are being properly applied.
  - Collection of info for future risk analysis.
  - Determine which risks caused which problems.
- **Management:**
  - What contingency plans do we have if the risk becomes a reality?
  - Contingency planning

- Devise RMMP

- **RMMM Plan:**
  1. It documents all work performed as a part of risk analysis.
  2. Each risk is documented individually by using a Risk Information Sheet.
  3. RIS is maintained by using a database system.

5Q) Explain Quality Management?

- Variation control is the heart of quality control.
- **Quality of Design:**
  Refers to the characteristics that designers specify for the end product.
- **Quality of Conformance:**
  Degree to which design specifications are followed in manufacturing.
- **Quality Control:**
  Series of inspections, reviews & tests used to ensure conformance of a work product to its specifications.
- **Quality Assurance:**
  Consists of a set of auditing & reporting functions that assess the effectiveness & completeness of quality control activities.
- **Cost of Quality:**
  Prevention costs:
  Quality planning, formal technical reviews, test equipment, training.
  Appraisal costs:
  In-process and inter-process inspection, equipment calibration and maintenance, testing.
  Failure costs:
  rework, repair, failure mode analysis.
  External failure costs:
  Complaint resolution, product return and replacement, help line support, warranty work

6Q) Explain Software Quality Assurance (SQA)?

- Software quality assurance (SQA) is the concern of every software engineer to reduce cost and improve product time-to-market.
- SQA activities are performed on every software project.
- A Software Quality Assurance Plan is not merely another name for a test plan, though test plans are included in an SQA plan.
- Use of metrics is an important part of developing a strategy to improve the quality of both software processes and work products.
- **Objectives:**
  - Conformance to software requirements is the foundation from which software quality is measured.
  - Specified standards are used to define the development criteria that are used to guide the manner in which software is engineered.

- Software must conform to implicit requirements (ease of use, maintainability, reliability, etc.) as well as its explicit requirements.

Software Engineering

Unit-3: Modelling With UML

AIM:

General study of UML

DESCRIPTION:

The heart of object-oriented problem solving is the construction of a model. The model abstracts the essential details of the underlying problem from its usually complicated real world. Several modeling tools are wrapped under the heading of the UML ,which stands for Unified Modeling Language. The purpose of this course is to present important highlights of the UML.

The Major components of Unified Modeling Language

UML includes the following 9 diagrams:

1. Class diagram: These diagrams depict the behavioral pattern of the system, i.e. how each and every class is inter-related to the other one, which relationship exists among each of the classes, etc. There would be only one class diagram possible for a single system. Class diagrams of one system can be linked to the class diagrams of another system, provided, there is a multi-system requirement.

2. Object diagram: Object diagram is similar to the above mentioned class diagram and is said to be a real entity or an instance of the Class used to mention the extra properties of an entity in addition to the properties depicted by the class.

3. Use case diagram: Use case diagram comprises of use cases and actors such that there would be various kinds of relationships among the use cases and the actors. A use case diagram shows all the actions that a particular actor needs to perform throughout the system at every and any point of time. There would be only one use case diagram per each system.

4. Sequence diagram: This diagram, as the name suggests, contains the sequence of flow of actions that are processed through a system and the life lines of the entities, when and how are they accessed. It also contains the security like which entity can process which entity and which one is visible, etc. There can be many number of sequence diagrams per each activity being done.

5. Collaboration diagram: This diagram is a polymorphic form of the sequence diagram in which the representation is different but application is the same. If we are able to create one sequence diagram, then it's very simple to create its collaboration diagram with a single key click that varies from to software. There can be many number of collaboration diagrams per each activity being done because there can be many number of sequence diagrams.

6. Activity diagram: This diagram denotes the structural flow of the activities in the form of flow chart with decision boxes enhanced and hence is also used for troubleshooting like raising exceptions when a particular action is done and the alternative to be done when something abnormal is done. There can be only one activity diagram for the entire system including all the activities that a system can perform.

7. State chart diagram: This diagram is a polymorphic form of the activity diagram in which the representation is different but application is the same. It looks similar to a finite state machine state transition diagrams.

8. Deployment diagram: Deployment diagram is employed when we need to deploy the application we developed. A single deployment diagram is possible for a single system.

9. Component diagram: Component diagram represents the components in which the particular application needs to be installed or implemented on. It also shows the type of relation that exists among the various components that are represented. Hence, only a single component diagram representing all the components and their relations is needed for the entire system.

This article emphasizes on 8 most prominent diagrams out of the given 9 diagrams, excluding object diagram which has no greater importance than the rest of them.

Software Engineering

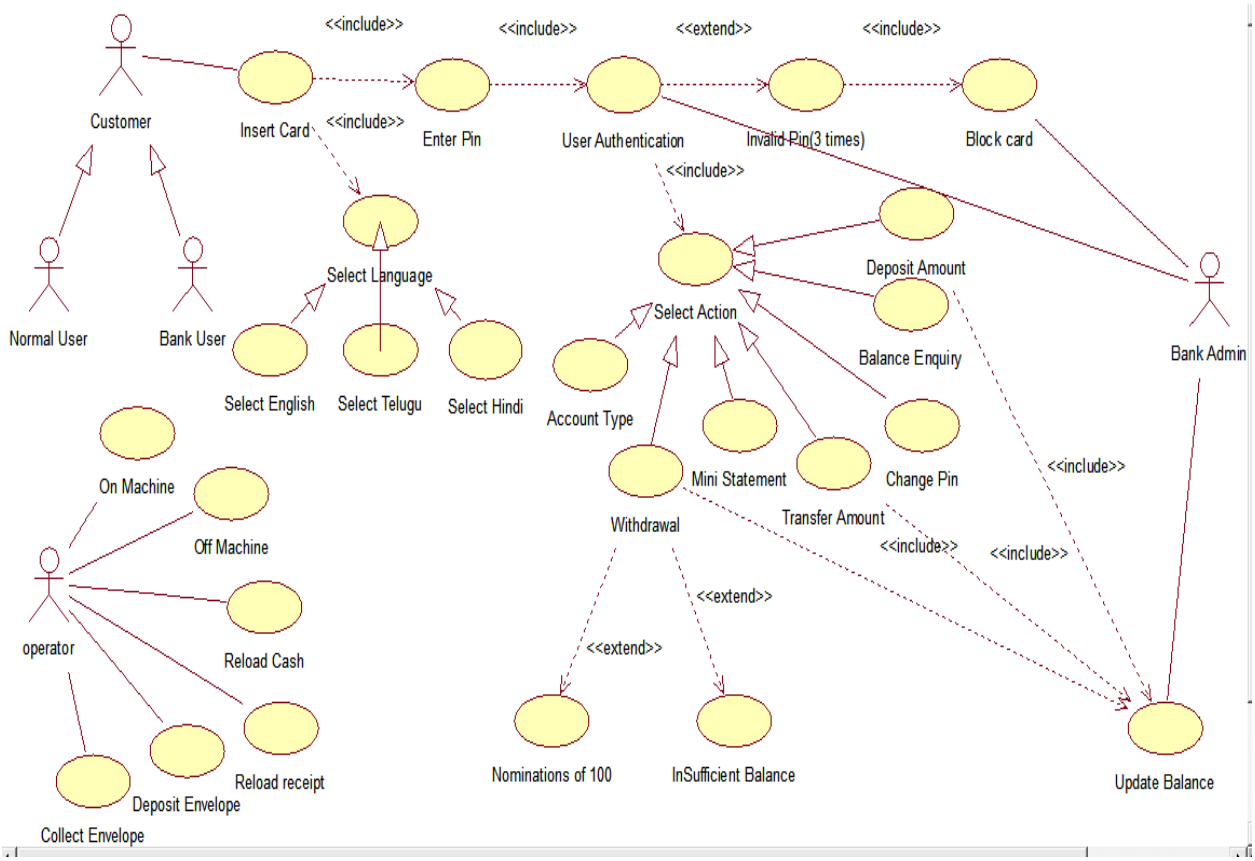
CASE STUDY

ATM Machine and System

ATM or rather Automated Teller Machine is also called as ANY TIME MONEY by many. The ATM system is connected to banks.

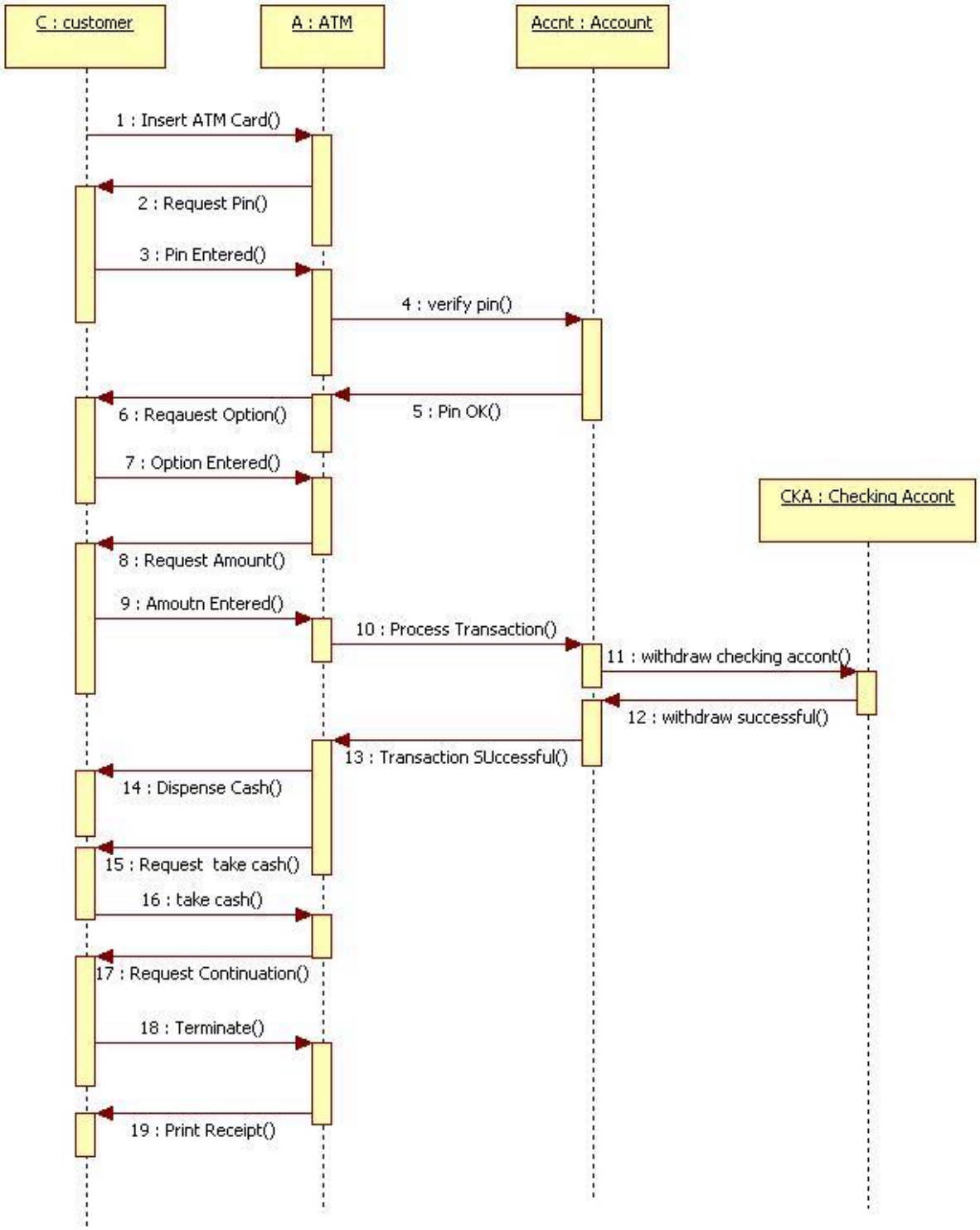
The ATM is given the utmost security in terms of technology because its a standalone system and easily prone to malicious attacks.

Use Case Diagram ATM Machine



Software Engineering

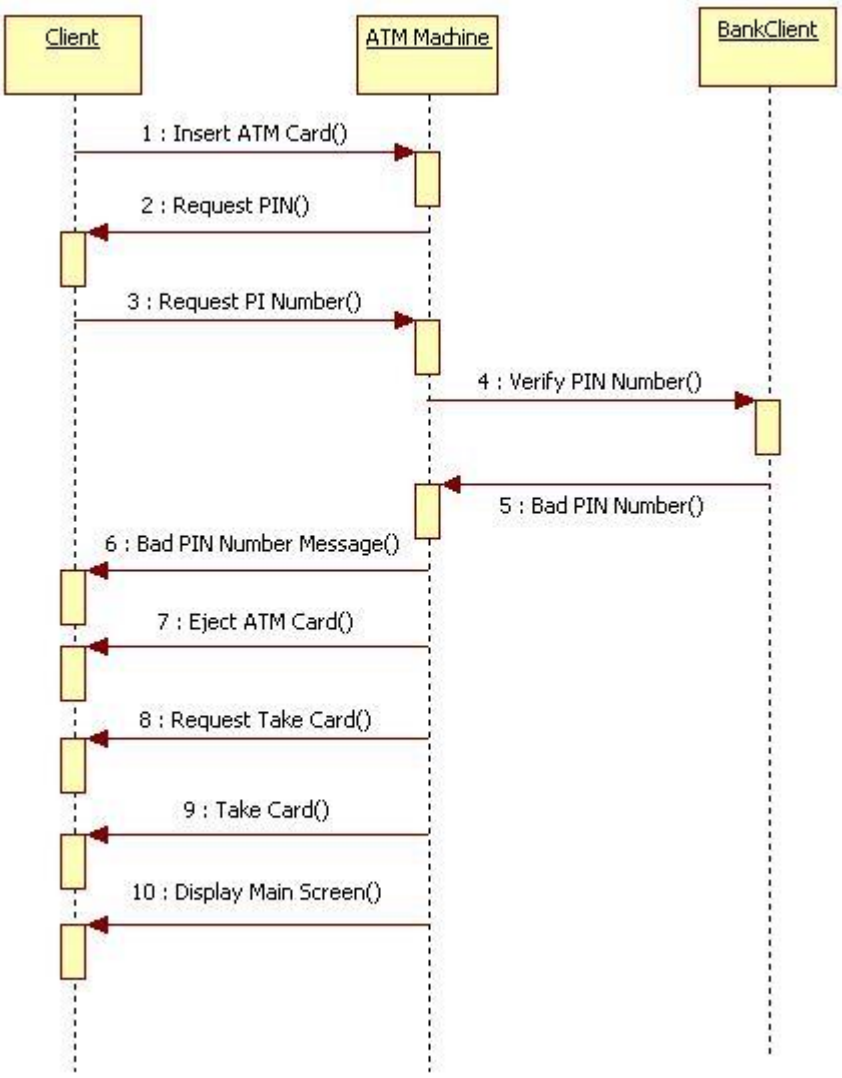
Sequence Diagram for ATM Machine :-



■ Naveen Anagandula

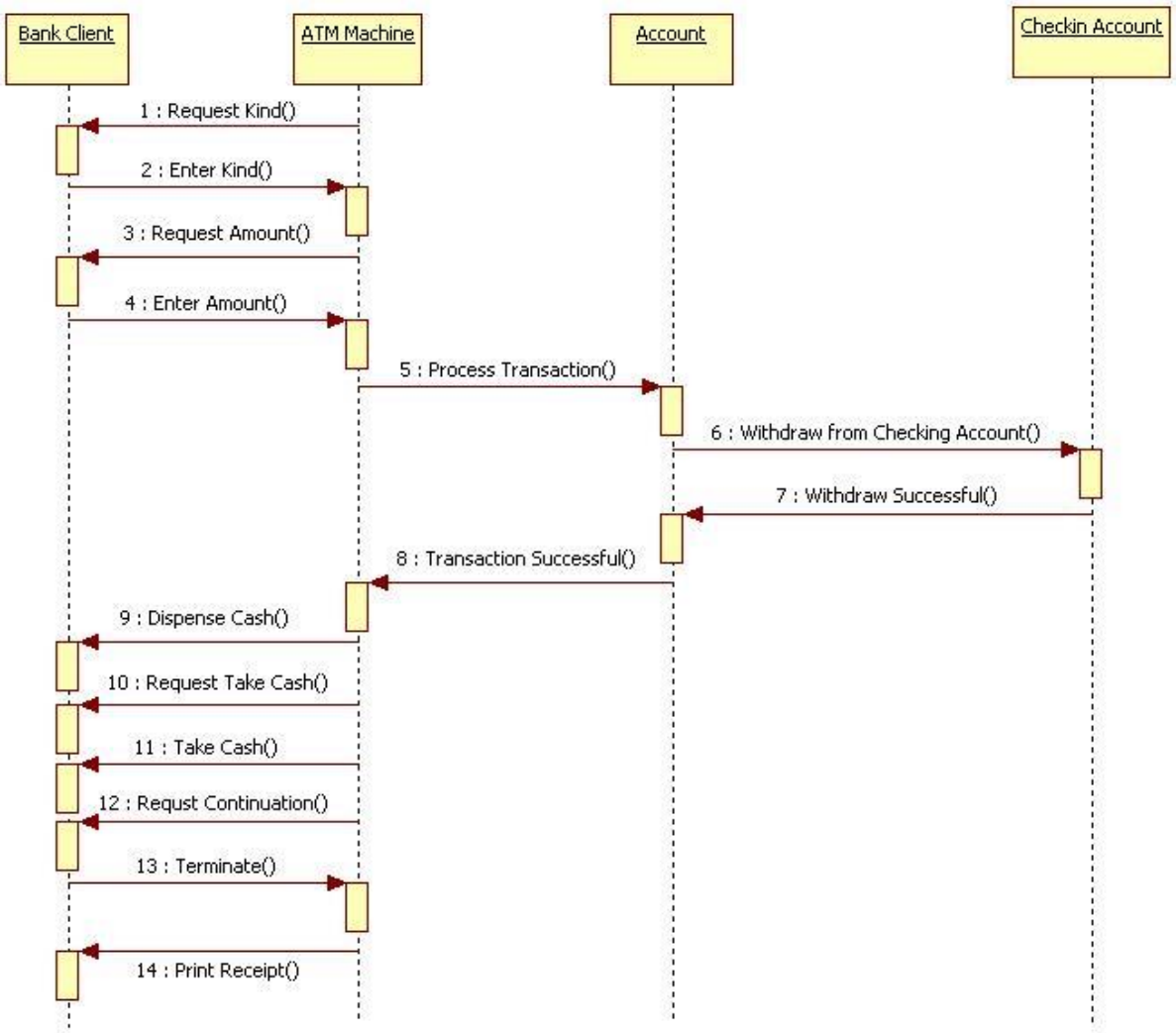
Software Engineering

Sequence Diagram for Invalid Pin ATM Machine:-



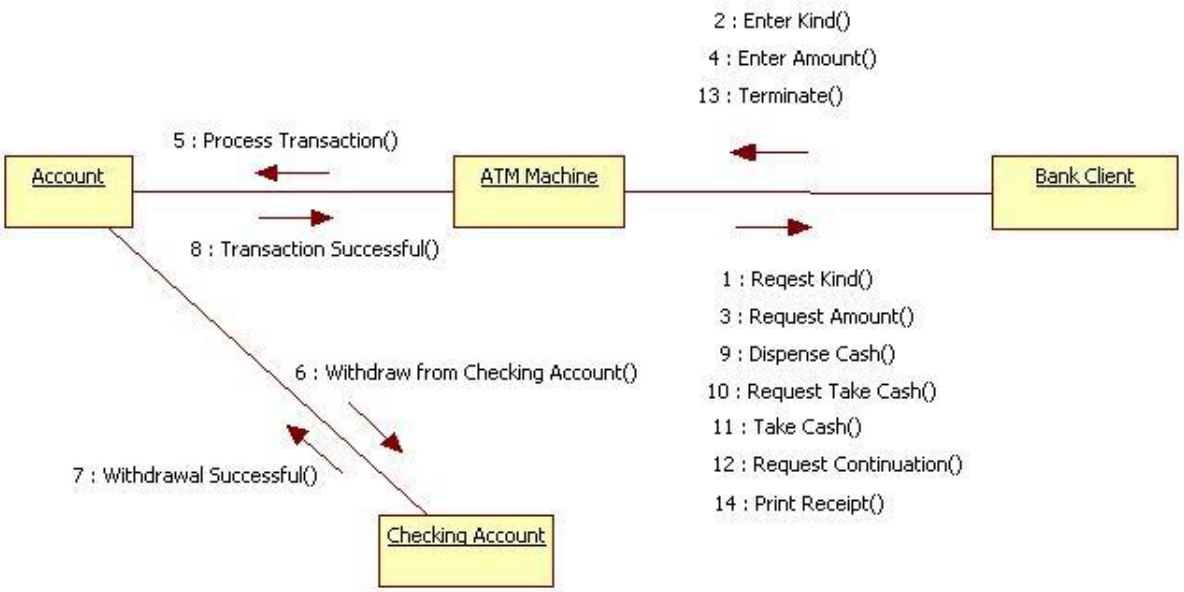
Software Engineering

Sequence Diagram ATM withdrawal:-

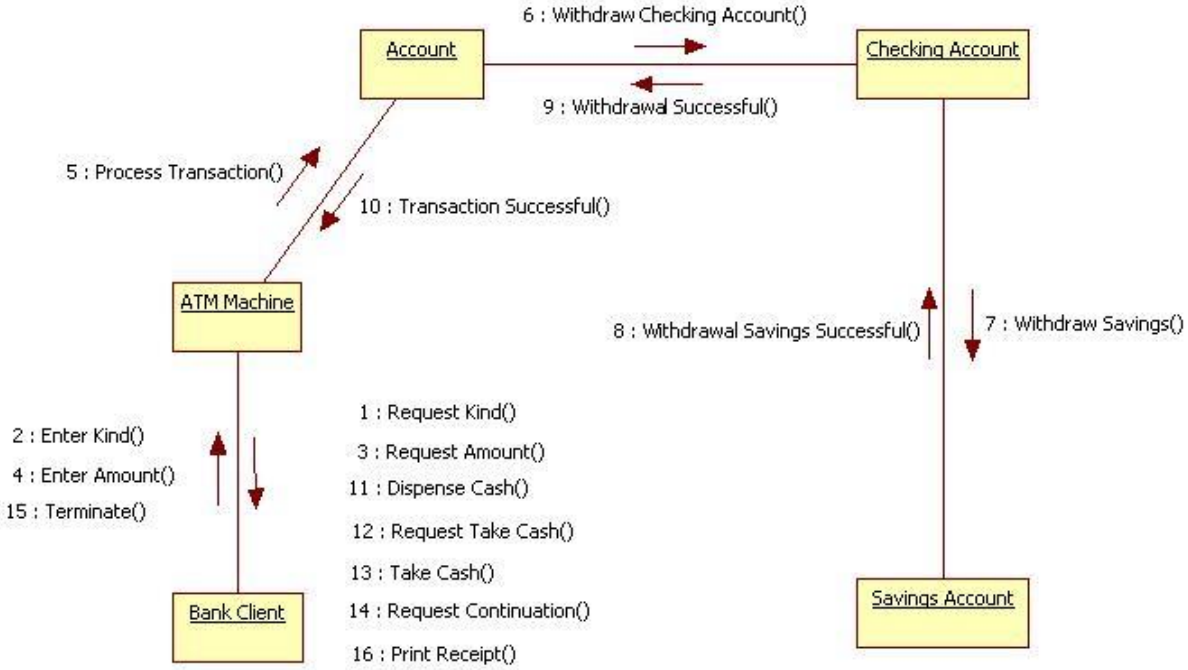


Software Engineering

Collaboration Diagram For ATM machine:-

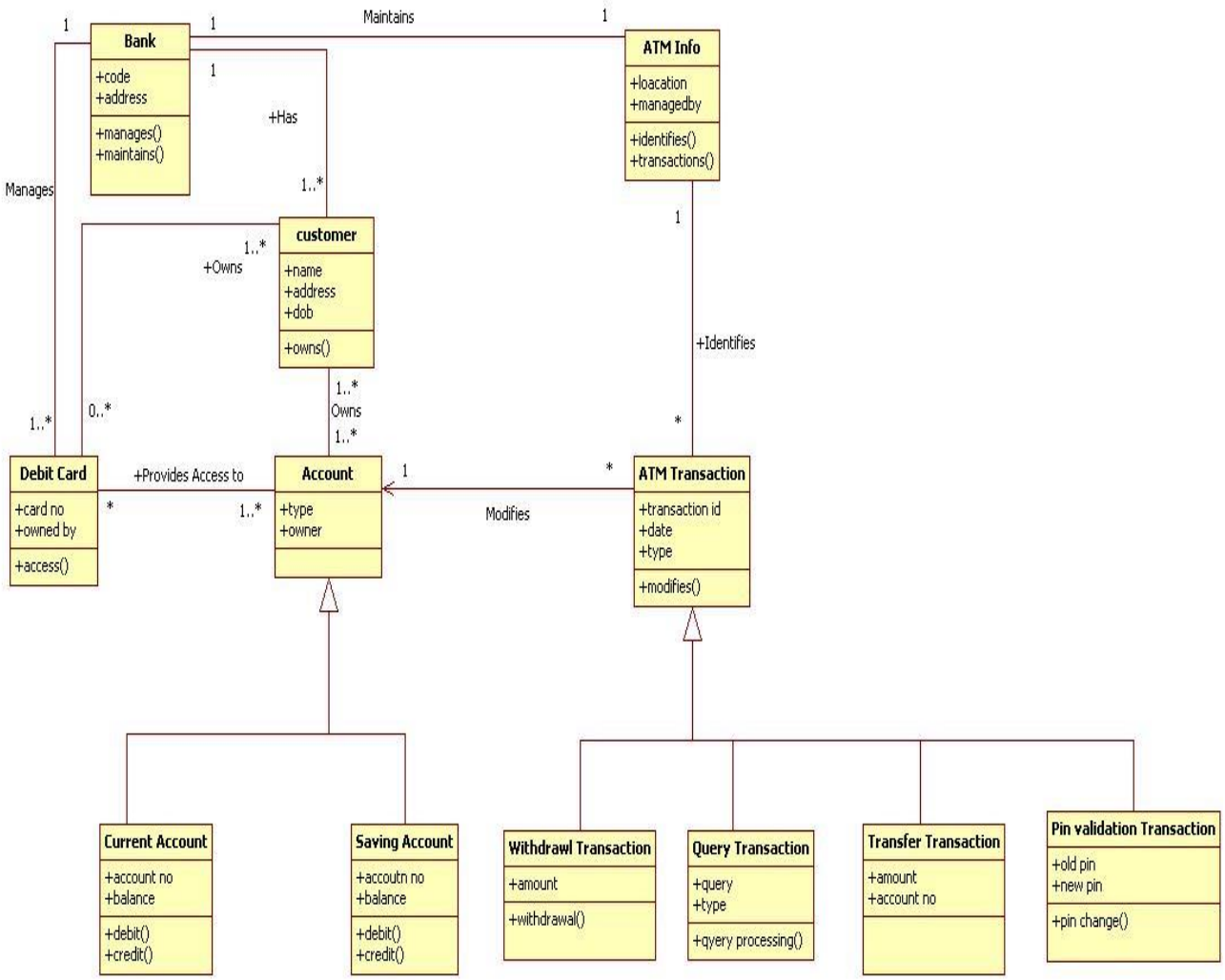


Collaboration Diagram for ATM withdrawal :-



Software Engineering

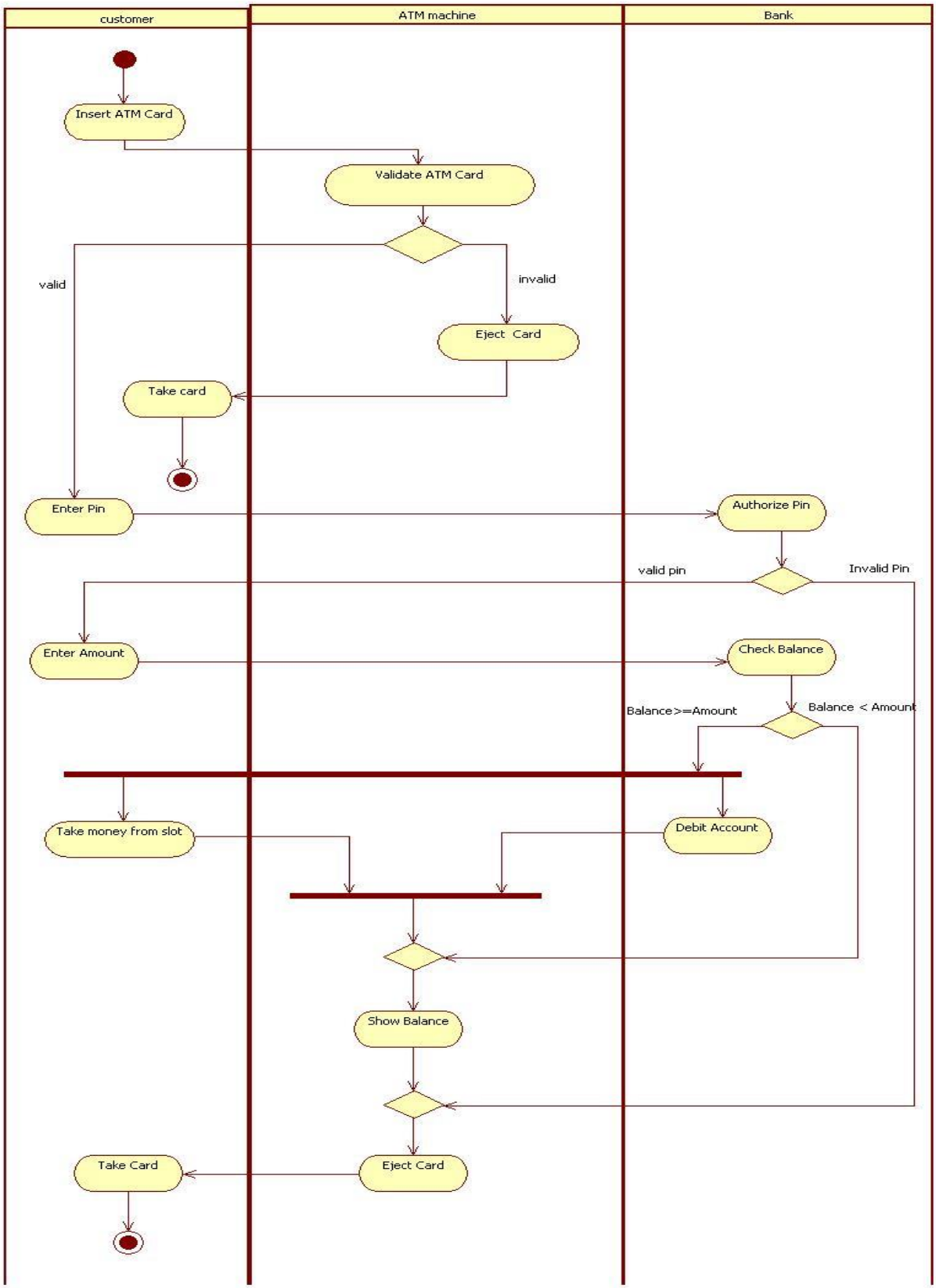
Class Diagram for ATM Machine



■ Naveen Anagandula

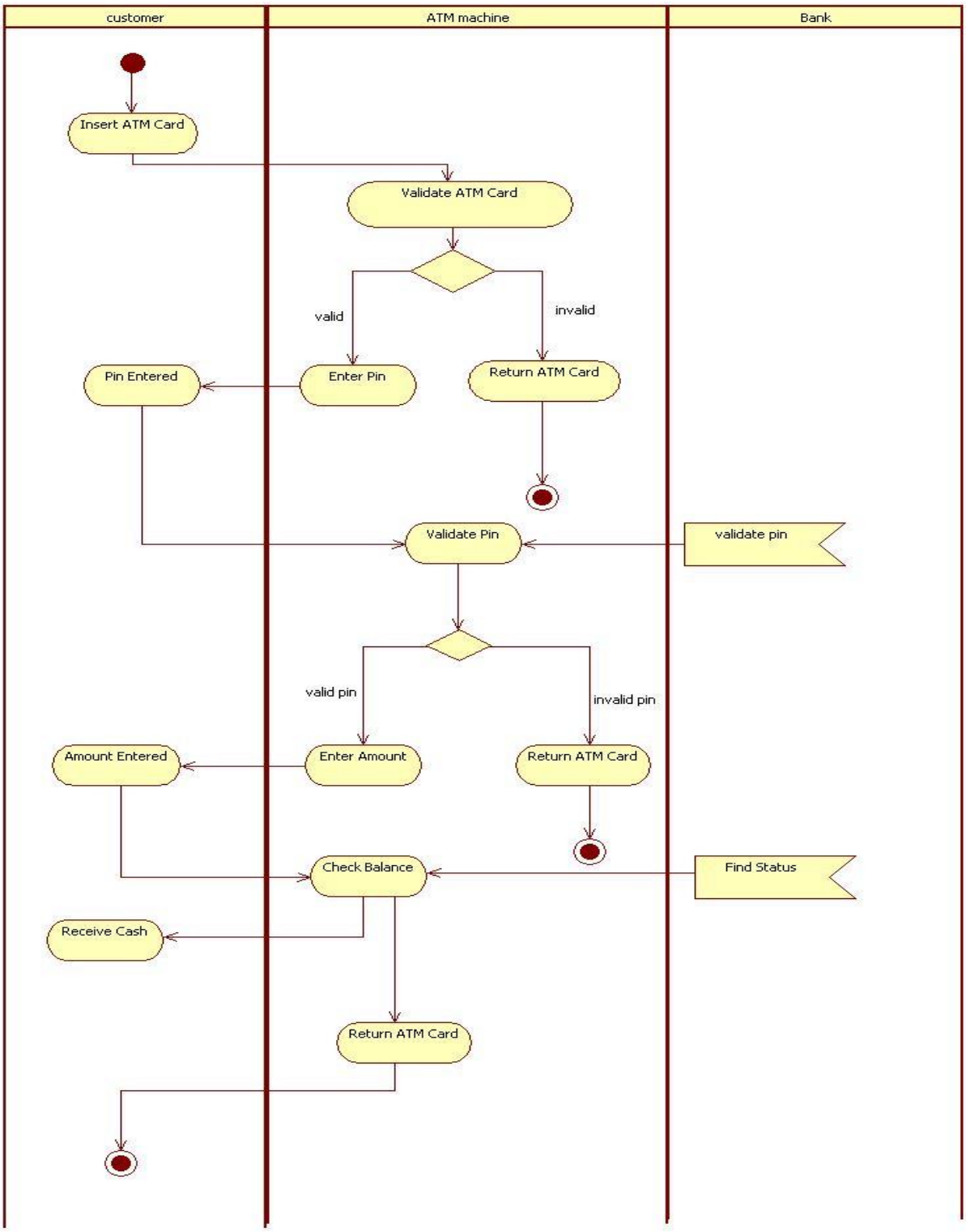
Software Engineering

Activity Diagram for ATM Machine 1



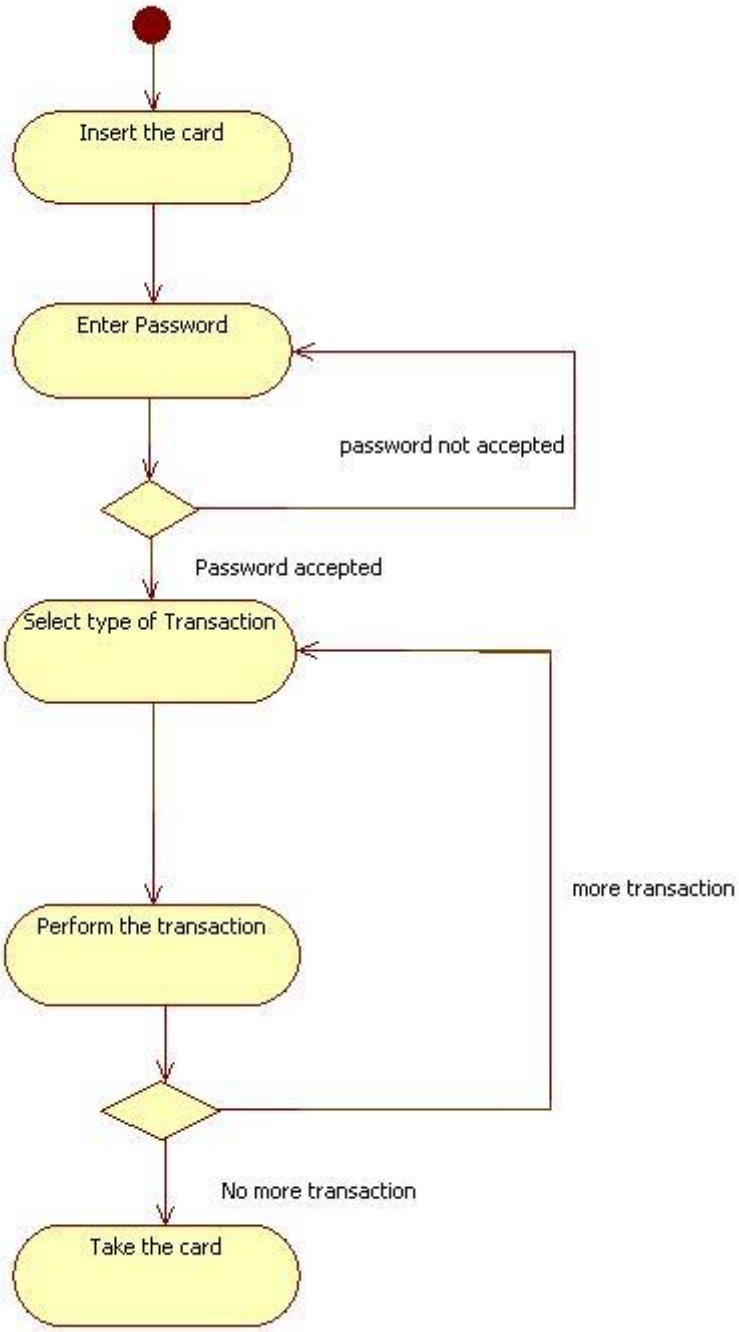
Software Engineering

Activity Diagram for ATM Machine 2



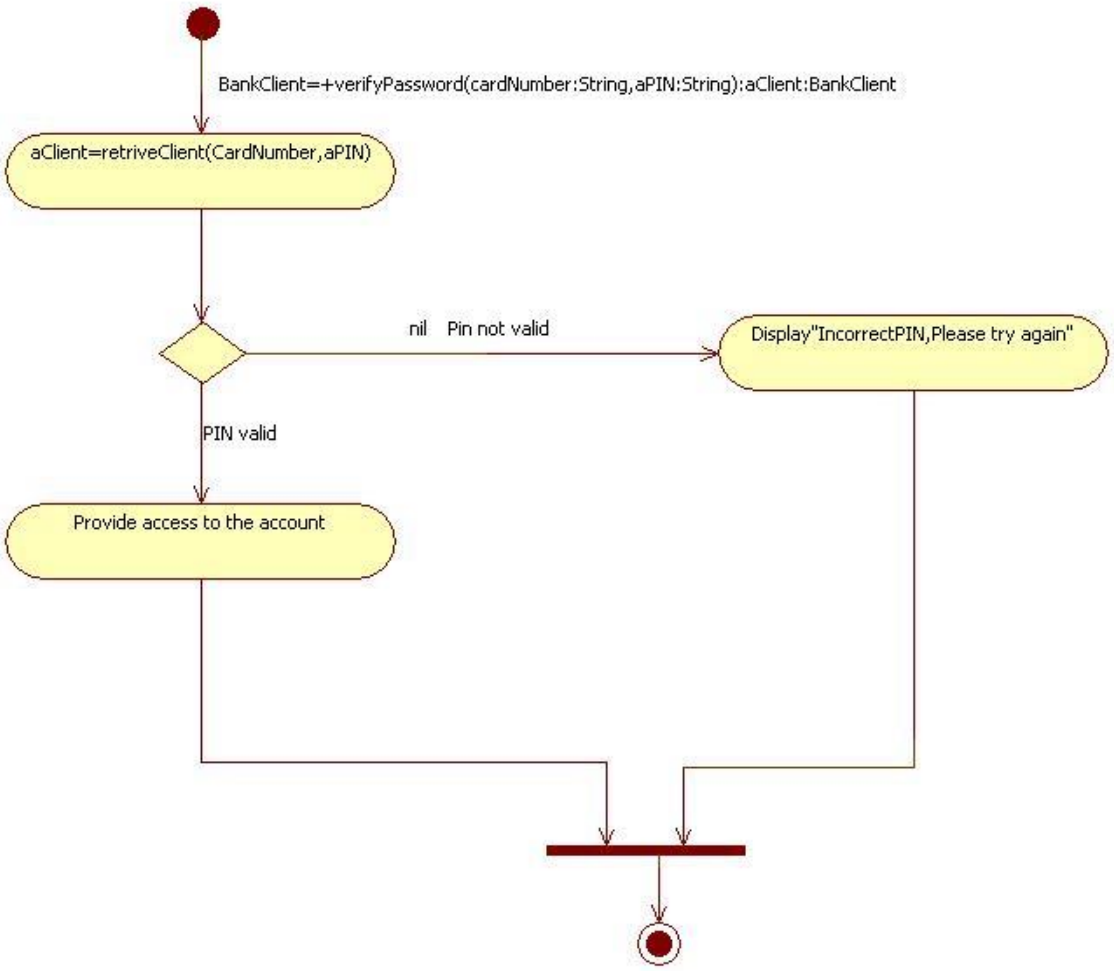
Software Engineering

Activity Diagram for Overall ATM Machine:-



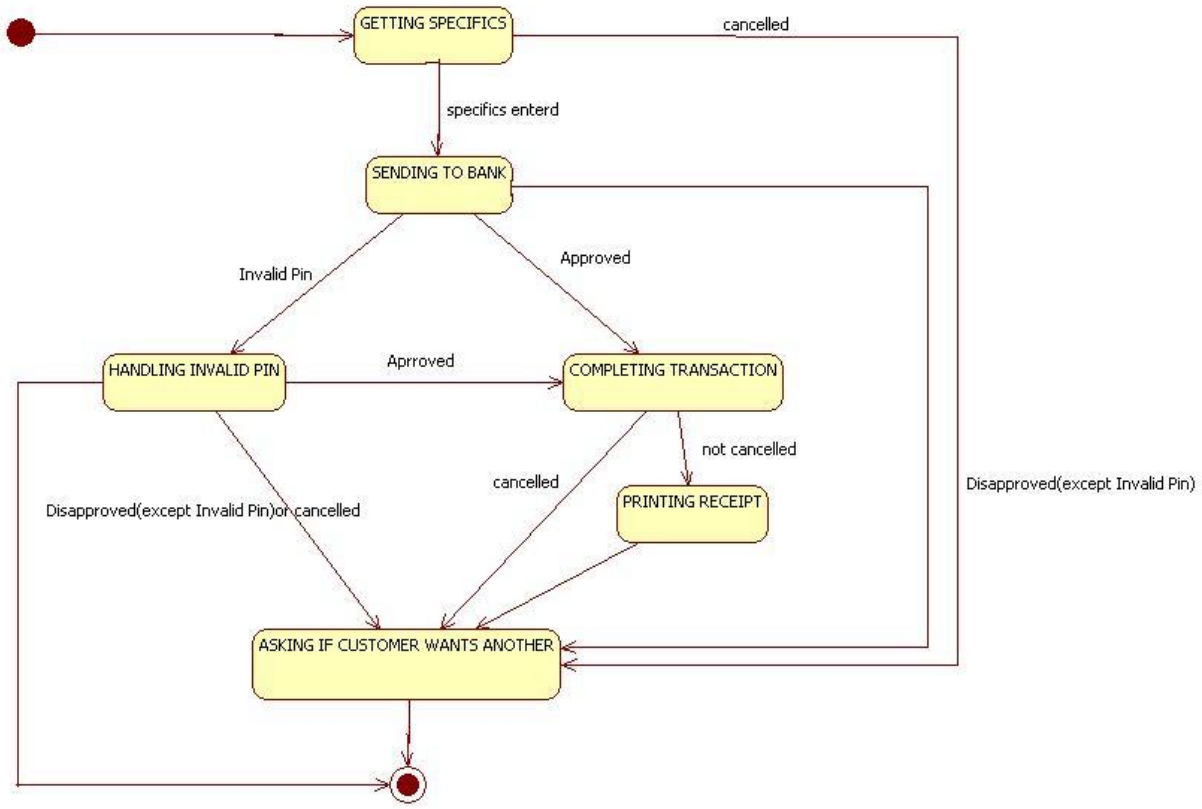
Software Engineering

Activity Diagram Verify Password ATM Machine:-



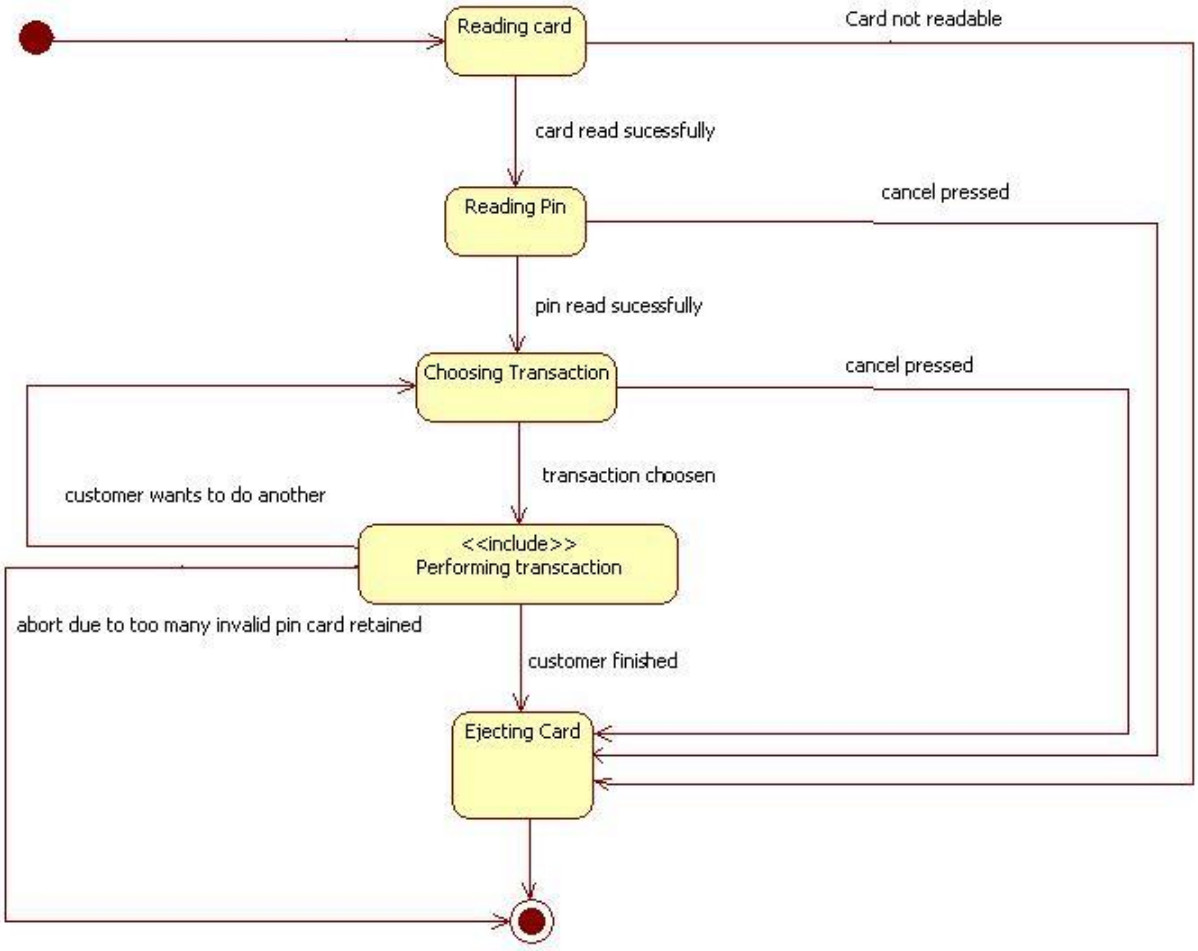
Software Engineering

State Diagram for One Transaction ATM machine:-



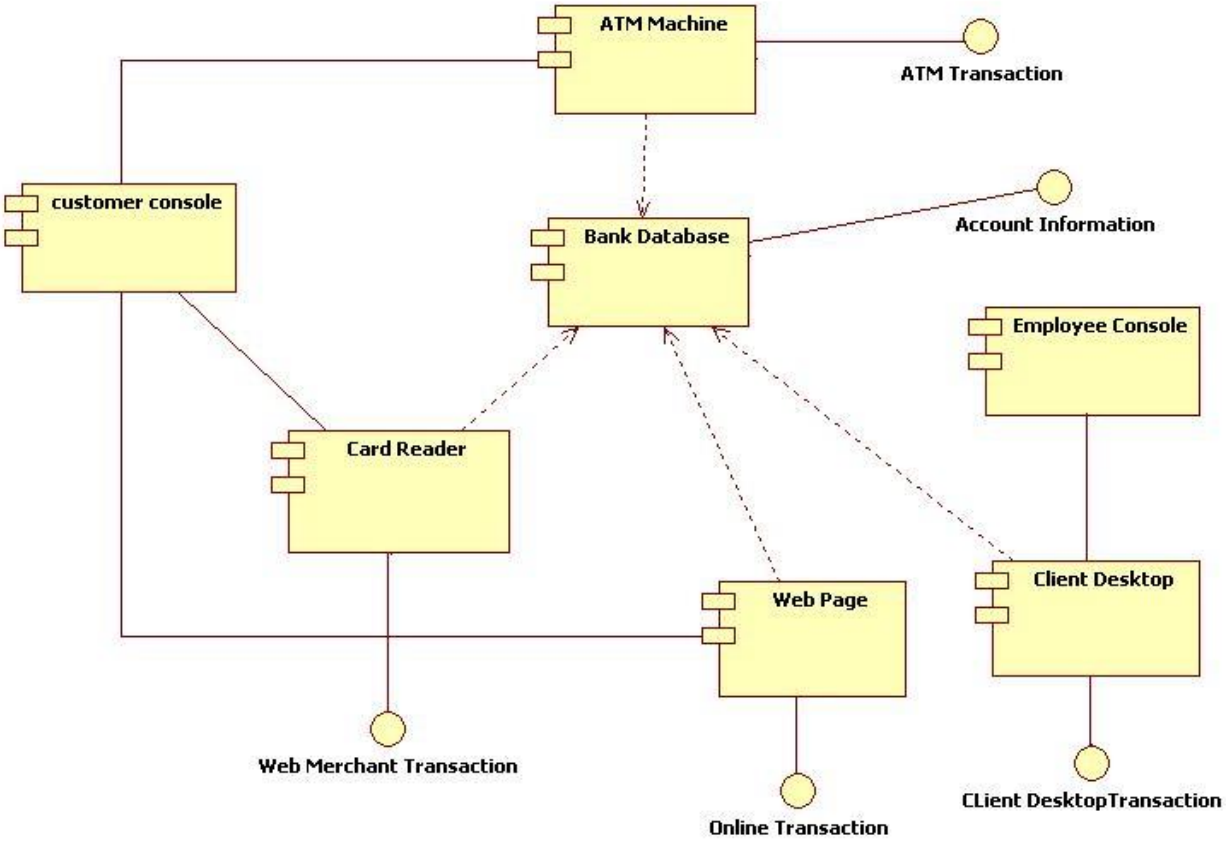
Software Engineering

State Diagram one Session ATM Machine:-



Software Engineering

Component Diagram ATM Machine:-



Software Engineering

Deployment Diagram ATM Machine :-