

Requirement Engineering & Analysis

Requirement Engineering

- The broad spectrum of tasks and techniques that leads to an understanding of requirements is called requirement engineering.

Requirements Engineering-I

- **Inception**—ask a set of questions that establish ...
 - basic understanding of the problem
 - the people who want a solution
 - the nature of the solution that is desired, and
 - the effectiveness of preliminary communication and collaboration between the customer and the developer
- **Elicitation**—elicit requirements from all stakeholders
- **Elaboration**—create an analysis model that identifies data, function and behavioral requirements
- **Negotiation**—agree on a deliverable system that is realistic for developers and customers

Requirements Engineering-II

- **Specification**—can be any one (or more) of the following:
 - A written document
 - A collection of user scenarios (use-cases)
 - A prototype
- **Validation**—a review mechanism that looks for
 - errors in content or interpretation
 - areas where clarification may be required
 - missing information
 - inconsistencies (a major problem when large products or systems are engineered)
- **Requirements management**

Inception

- Identify stakeholders
 - “who else do you think I should talk to?”
- Recognize multiple points of view
- Work toward collaboration
- The first questions
 - Who is behind the request for this work?
 - Who will use the solution?
 - What will be the economic benefit of a successful solution
 - Is there another source for the solution that you need?

Eliciting Requirements

- meetings are conducted and attended by both software engineers and customers
- rules for preparation and participation are established
- a "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
- a "definition mechanism" (can be work sheets, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used
- the goal is
 - to identify the problem
 - propose elements of the solution
 - negotiate different approaches, and
 - specify a preliminary set of solution requirements

Quality Function Deployment

- Normal Requirements
- Expected Requirements
- Exciting Requirements

Elicitation Work Products

- a statement of need and feasibility.
- a bounded statement of scope for the system or product.
- a list of customers, users, and other stakeholders who participated in requirements elicitation
- a description of the system's technical environment.
- a list of requirements (preferably organized by function
- a set of usage scenarios that provide insight into the use of the system or product under different operating conditions.
- any prototypes developed to better define requirements.

Negotiating Requirements

- Identify the key stakeholders
 - These are the people who will be involved in the negotiation
- Determine each of the stakeholders “win conditions”
 - Win conditions are not always obvious
- Negotiate
 - Work toward a set of requirements that lead to “win-win”

Validating Requirements-I

- Is each requirement consistent with the overall objective for the system/product?
- Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each requirement bounded and unambiguous?
- Do any requirements conflict with other requirements?

Validating Requirements-II

- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function and behavior of the system to be built.
- Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system.

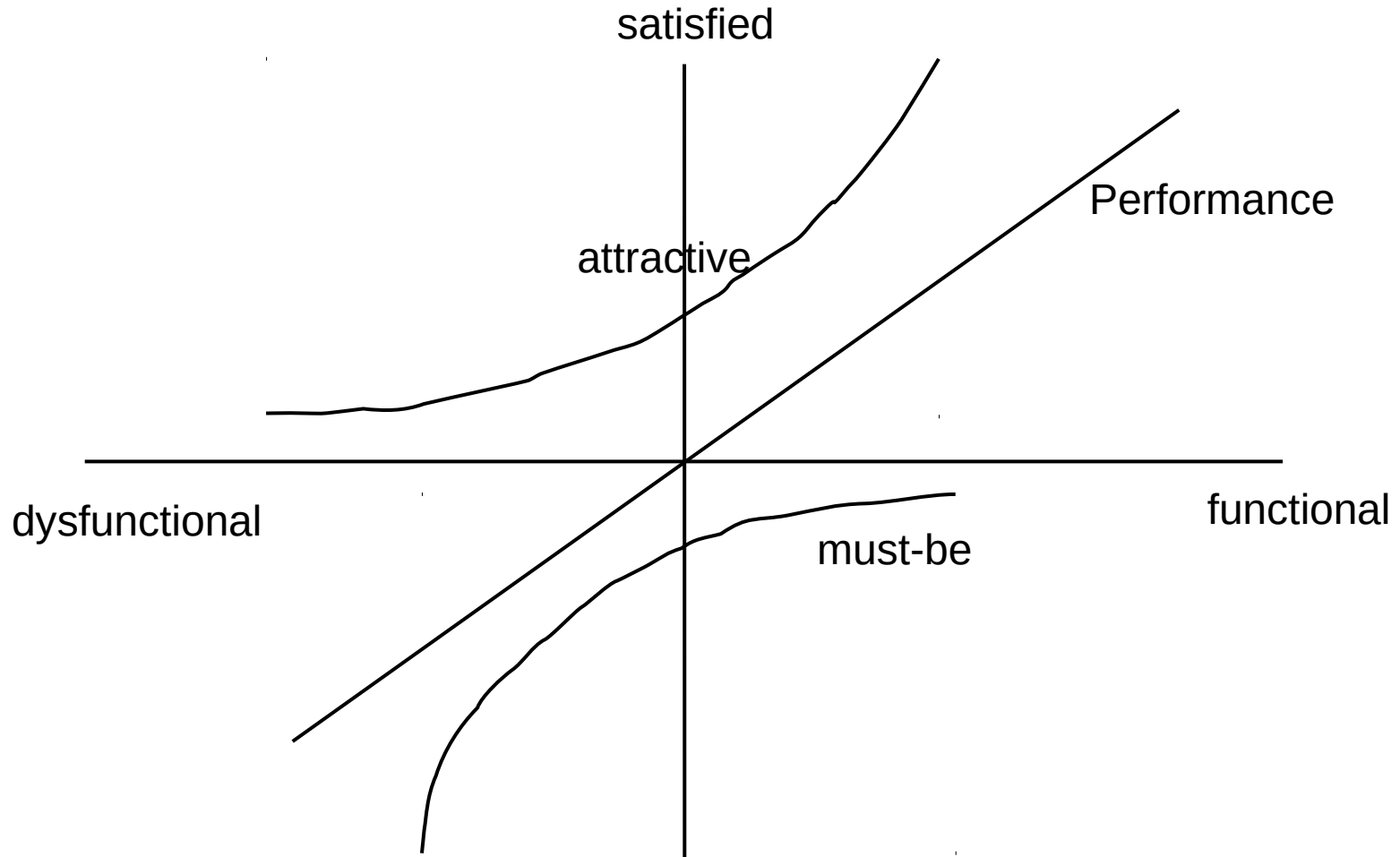
Prioritizing requirements (Kano model)

- **Must-be**: dissatisfied when -, at most neutral
- **One-dimensional**: satisfaction proportional to number
- **Attractive**: more satisfied if +, not less satisfied if -
- **Indifferent**: don't care
- **Reverse**: opposite of what analyst thought
- **Questionable**: preferences not clear

Kano Model

- Must be
- Performance
- Attractive

Kano Model



Kano Model

- Must be
- Performance
- Attractive

Requirement	Prioritization
Register	Must be
Login	Must be
Book Ticket	Must be
Seat Selection	Performance
Make Payment	Performance
Buy Refreshment	Attractive

Kano Model

- How to identify the category
- Ask Customer

How would you feel if product has the feature?

How would you feel if product doesn't has the feature?

Kano Model

Customer Requirements		Negative Questions / Dysfunctional				
		Like	Must be	Neutral	Live with	Dislike
Positive questions/ Functional	Like	Q	A	A	A	O
	Must be	R	I	I	I	M
	Neutral	R	I	I	I	M
	Live with	R	I	I	I	M
	Dislike	R	R	R	R	Q

Kano Model

- Must be
- One-Dimensional
- Attractive
- Indifferent
- Reverse
- Questionable

Agile Requirements User-stories

- User of customer need
- Product description
- Used for Planning
- Conversation

User Stories

- Easy to Understand
- Written by the Customer
- Small and Estimable
- Testable
- Becomes more detailed over the time

User story template

- *As a <type of user>, I want <to perform some task> so that I can <achieve some goal/benefit/value>.*

User story example

- **User story title:** Customer withdraws cash.

As a customer,
I want to withdraw cash from an
ATM

So that I don't have to wait in line
at the bank.

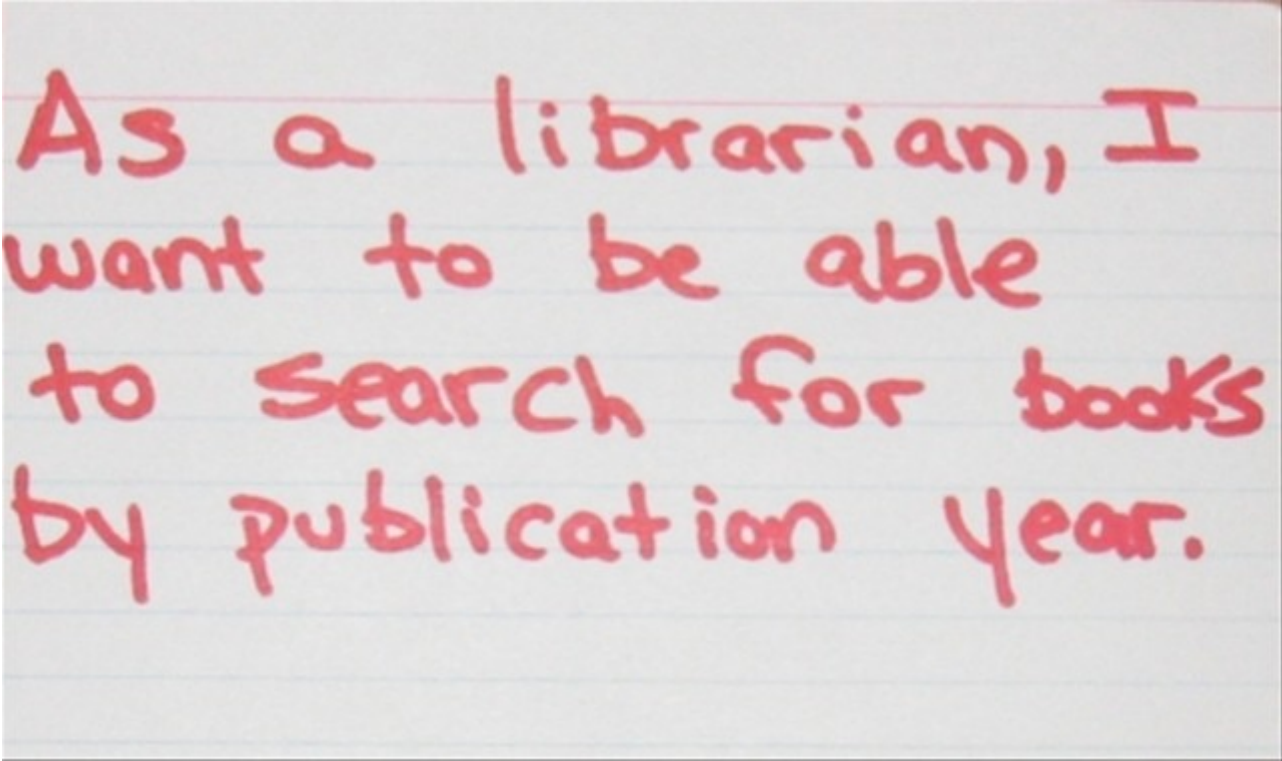
User story example

ITB

As a student I want to purchase
a parking pass so that I can
drive to school

Priority: ~~High~~ Should
Estimate: 4

User story example



As a librarian, I
want to be able
to search for books
by publication year.

User Stories Acceptance Criterion

Acceptance Criterion 1:

Given that the account is creditworthy

And the card is valid

And the dispenser contains cash,

When the customer requests the cash

Then ensure the account is debited

And ensure cash is dispensed

And ensure the card is returned.

User Stories Acceptance Criterion

- **Acceptance Criterion 2:**

Given that the account is overdrawn

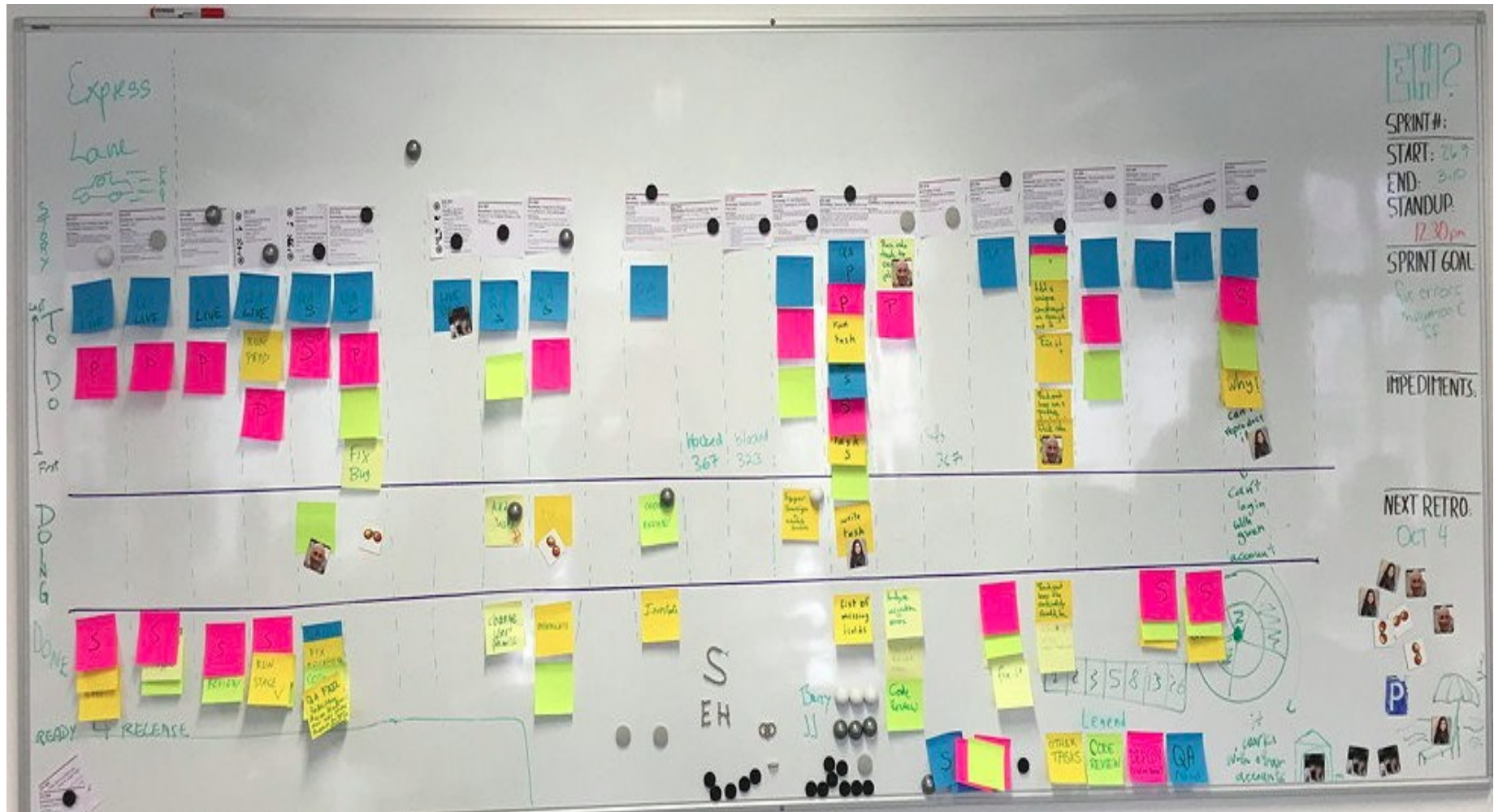
And the card is valid,

When the customer requests the cash

Then ensure the rejection message is displayed

And ensure cash is not dispensed.

User Stories board



Source: <https://age-of-product.com/wp-content/uploads/Agile-Transition-How-Create-Offline-Board-1024-1024x768-1.jpg>

User stories board (Software)

The screenshot shows a Jira interface for the 'Teams in Space' project. The left sidebar contains navigation links: Backlog, Board (selected), Reports, Releases, Components, Issues, Repository, Add item, and Settings. The main area displays a Kanban board with four columns: TO DO (5 items), IN PROGRESS (5 items), CODE REVIEW (2 items), and DONE (8 items). Each column contains user story cards with titles, priority labels, status icons, and assignees.

Column	Item	Title	Priority	Status	Assignee
TO DO (5)	1	Engage Jupiter Express for outer solar system travel	SPACE TRAVEL PARTNERS	Not Started	TIS-25
	2	Create 90 day plans for all departments in the Mars Office	LOCAL MARS OFFICE	Not Started	TIS-12
	3	Engage Saturn's Rings Resort as a preferred provider	SPACE TRAVEL PARTNERS	Not Started	TIS-17
IN PROGRESS (5)	4	Requesting available flights is now taking > 5 seconds	SEESPACEEZ PLUS	In Progress	TIS-8
	5	Engage Saturn Shuttle Lines for group tours	SPACE TRAVEL PARTNERS	In Progress	TIS-15
	6	Establish a catering vendor to provide meal service	LOCAL MARS OFFICE	In Progress	TIS-15
CODE REVIEW (2)	7	Register with the Mars Ministry of Revenue	LOCAL MARS OFFICE	Code Review	TIS-11
	8	Draft network plan for Mars Office	LOCAL MARS OFFICE	Code Review	TIS-15
DONE (8)	9	Homepage footer uses an inline style - should use a class	LARGE TEAM SUPPORT	Done	TIS-68
	10	Engage JetShuttle SpaceWays for travel	SPACE TRAVEL PARTNERS	Done	TIS-23
	11	Engage Saturn Shuttle Lines for group tours	SPACE TRAVEL PARTNERS	Done	TIS-15

3 C's of User-Stories

- Card
- Conversation
- Confirmation

Software Requirement Specification

Software Requirements Specification Template

(adapted from IEEE Standard 830-1998)

1. Introduction

- 1.1 Purpose
- 1.2 Document Conventions
- 1.3 Intended Audience & Reading Suggestions
- 1.4 Project Scope
- 1.5 References

2. Overall Description

- 2.1 Product Perspective
- 2.2 Product Features
- 2.3 User Classes and Characteristics
- 2.4 Operating Environment
- 2.5 Design & Implementation Constraints
- 2.6 User Documentation
- 2.7 Assumptions and Dependencies

3. System Features

- 3.x System Feature X
 - 3.x.1 Description and Priority
 - 3.x.2 Stimulus/Response Sequences
 - 3.x.3 Functional Requirements

4. External Interface Requirements

- 4.1 User Interfaces
- 4.2 Hardware Interfaces
- 4.3 Software Interfaces
- 4.4 Communications Interfaces

5. Other Nonfunctional Requirements

- 5.1 Performance Requirements
- 5.2 Safety Requirements
- 5.3 Security Requirements
- 5.4 Software Quality Attributes

6. Other Requirements

Appendix A: Glossary

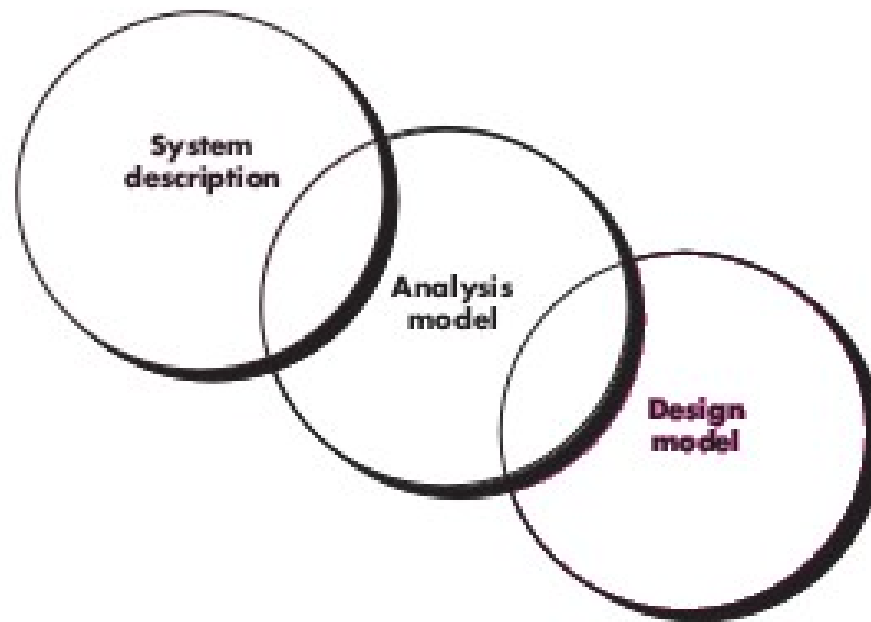
Appendix B: Analysis Models

Appendix C: Issues List

➤ IEEE Software Engineering standards: <http://standards.ieee.org/software>

Requirement Analysis

Requirement Model

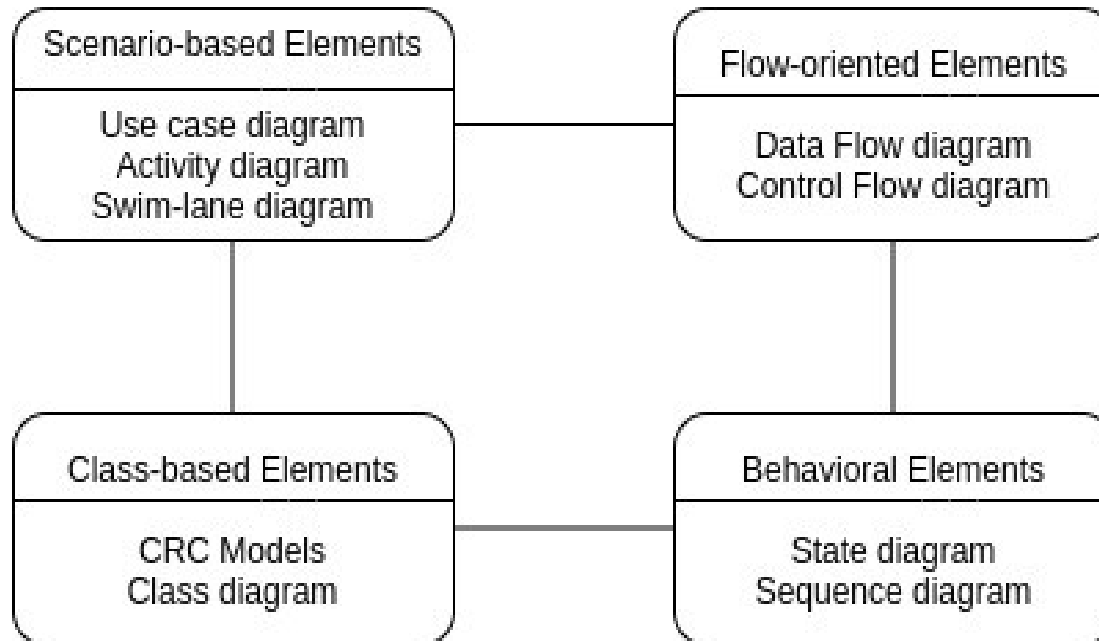


Source: Software Engineering A Practitioner's Approach, Roger S. Pressman

Requirement Model- Analysis Rules of Thumb

- The model should focus on requirements that are visible within the problem or business domain. The level of abstraction should be relatively high
- Each element of the requirements model should add to an overall understanding of software requirements and provide insight into the information domain, function, and behavior of the system
- Delay consideration of infrastructure and other nonfunctional models until design
- Minimize coupling throughout the system
- Be certain that the requirements model provides value to all stakeholders
- Keep the model as simple as it can be. Don't create additional diagrams when they add no new information.

Requirement Model



Ref: Software Engineering A Practitioner's Approach, Roger S. Pressman

Use-Cases

- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way
- Each scenario answers the following questions:
 - Who is the primary actor, the secondary actor (s)?
 - What are the actor’s goals?
 - What preconditions should exist before the story begins?
 - What main tasks or functions are performed by the actor?
 - What information does the actor desire from the system?
 - Does the actor wish to be informed about unexpected changes?

Use Case Diagram

- Creating Preliminary Use Case
- Refining Preliminary Use Case
- Writing Formal Use Case

Crating Preliminary Use Case

To begin developing a set of use cases, list the functions or activities performed by a specific actor

- Log in
- Change Pin
- Withdraw Cash
- Fund Transfer
- Check Balance

Refining Preliminary Use Case

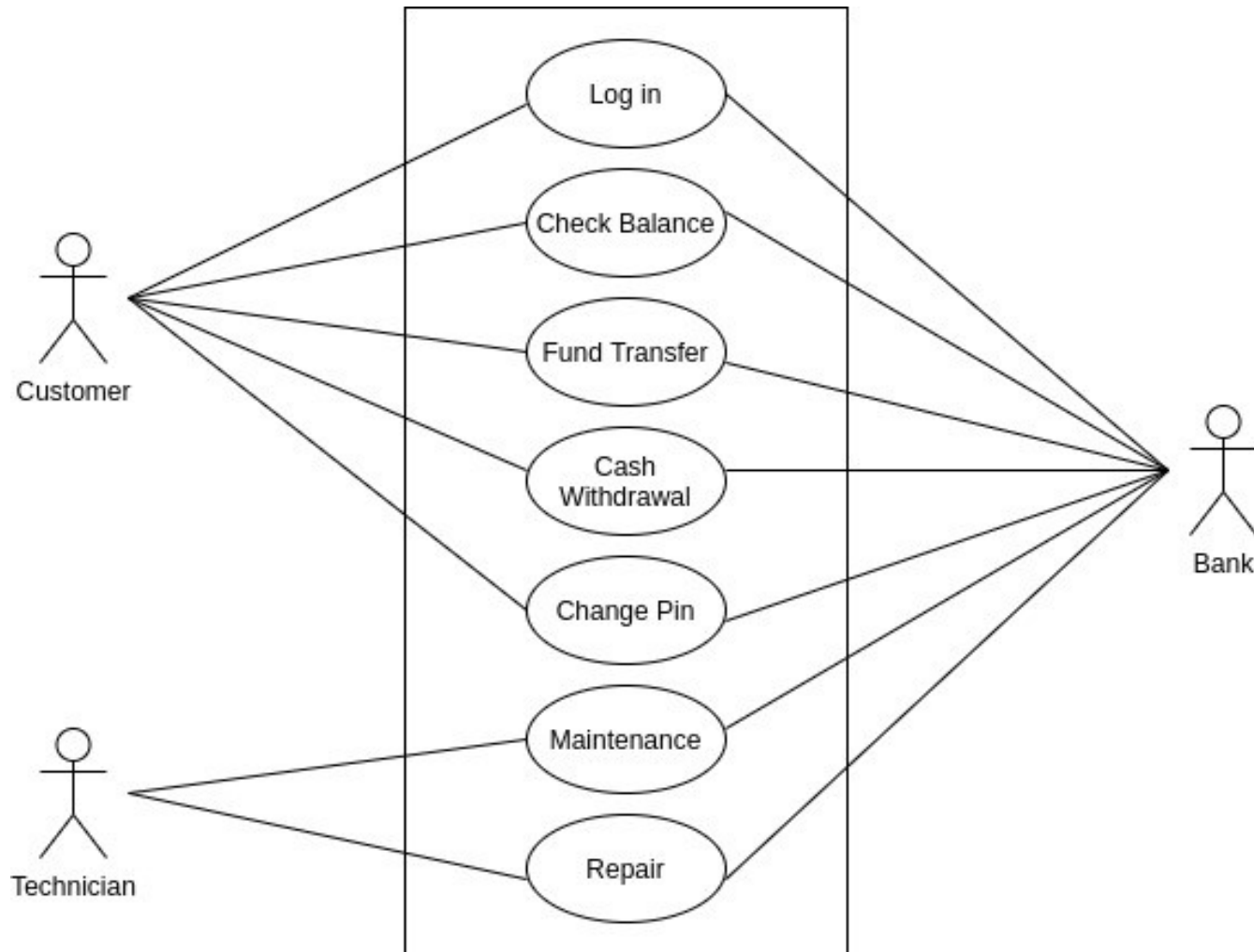
A description of alternative interactions is essential for a complete understanding of the function that is being described by a use case.

- Can the actor take some other action at this point?
- Is it possible that the actor will encounter some error condition at this point? If so, what might it be?
- Is it possible that the actor will encounter some other behavior at this point

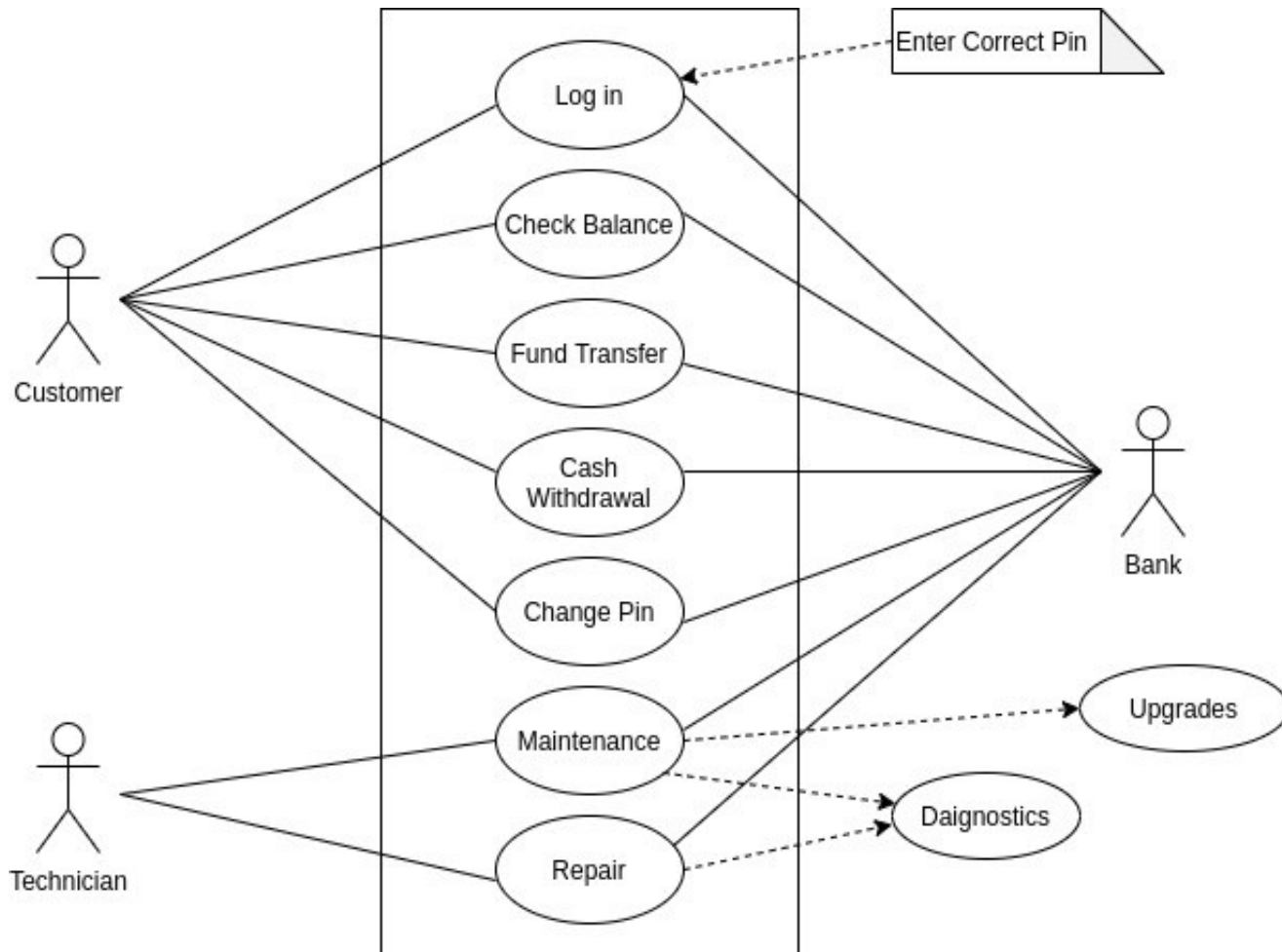
Writing Formal Use Case

- Use Case: **ATM Operations**
- Primary Actor: **Customer**
- Goal in Context:
- Preconditions:
- Scenarios:
- Exceptions:
- Secondary Actor:

Use Case- ATM



Use Case- ATM



Requirement Model

Scenario-based Elements

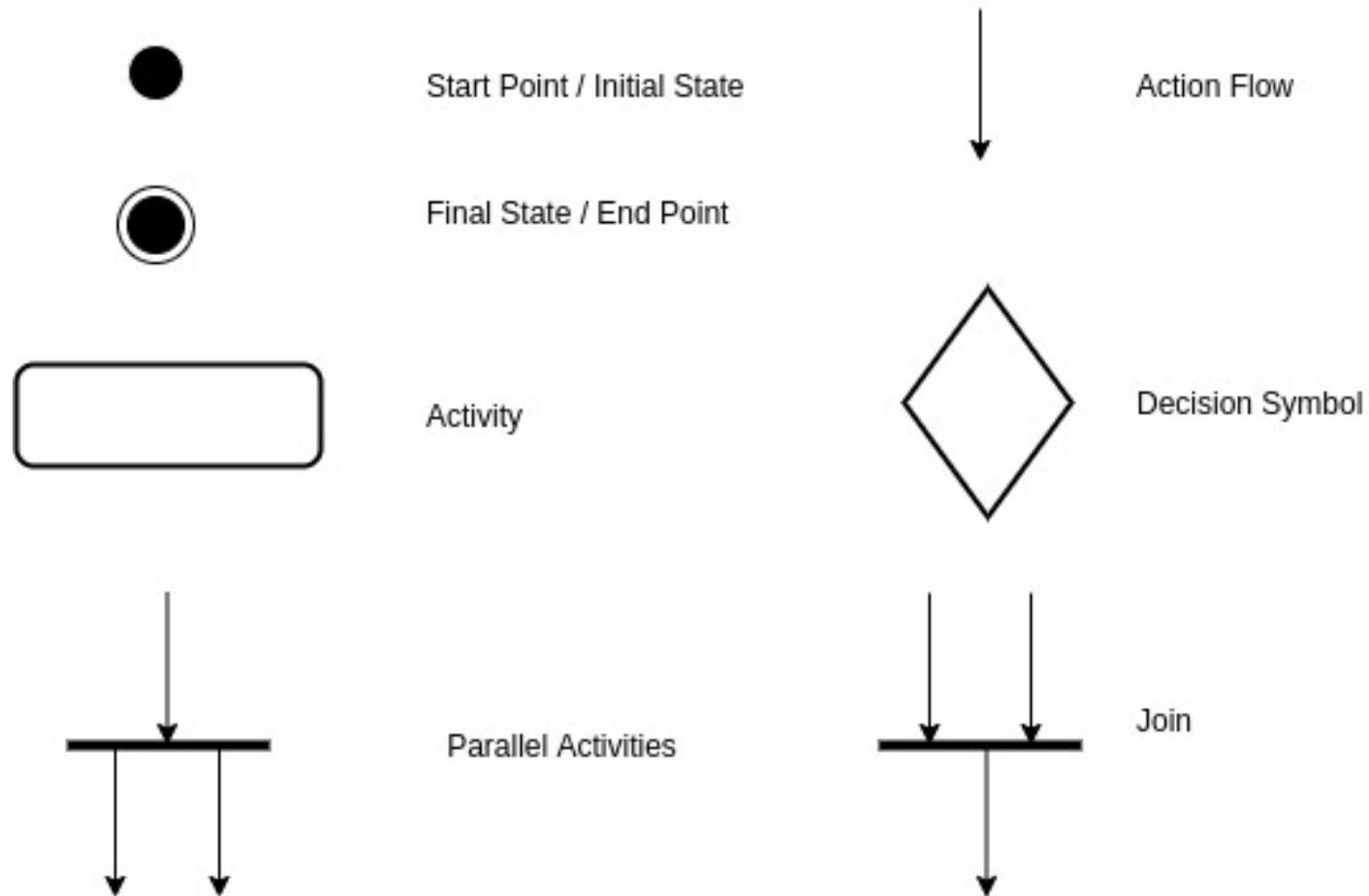
Objectives

- Activity Diagram
- Swimlane Diagram

Activity Diagram

- The UML activity diagram supplements the use case by providing a graphical representation of the flow of interaction within a specific scenario
- Each A UML activity diagram represents the actions and decisions that occur as some function is performed.

Activity Diagram- Notations/ Symbols



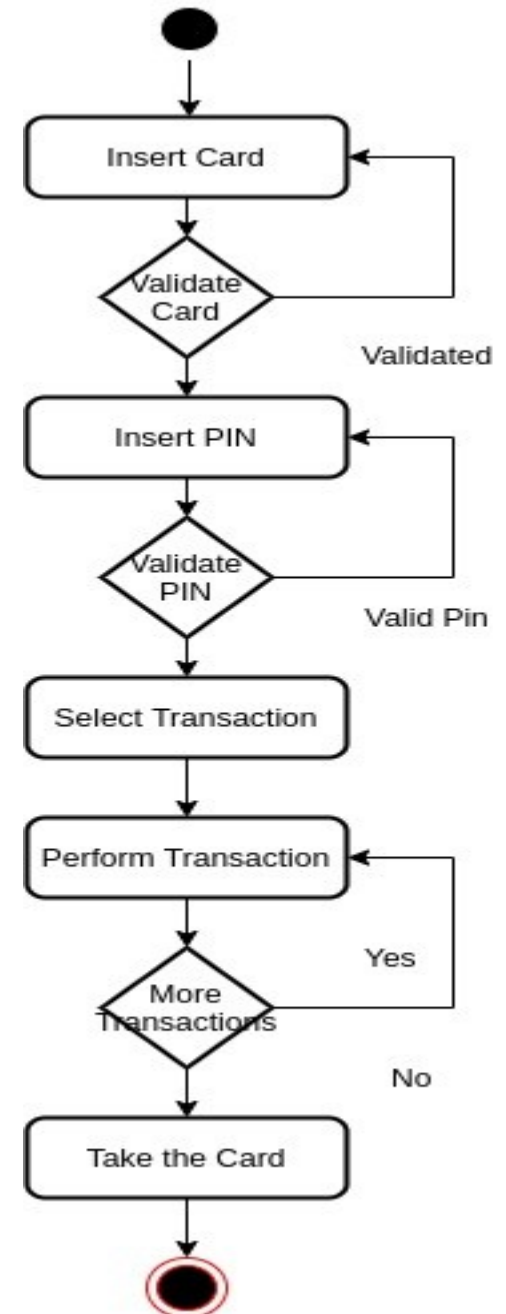
Activity Diagram-Overall ATM Operations

- Insert Card
- If card is valid then ask for the pin
- If pin is valid the ask to select the operation
- Once operation is selected, perform the selected operation
- If no more operations is to be performed take out the card and end the process

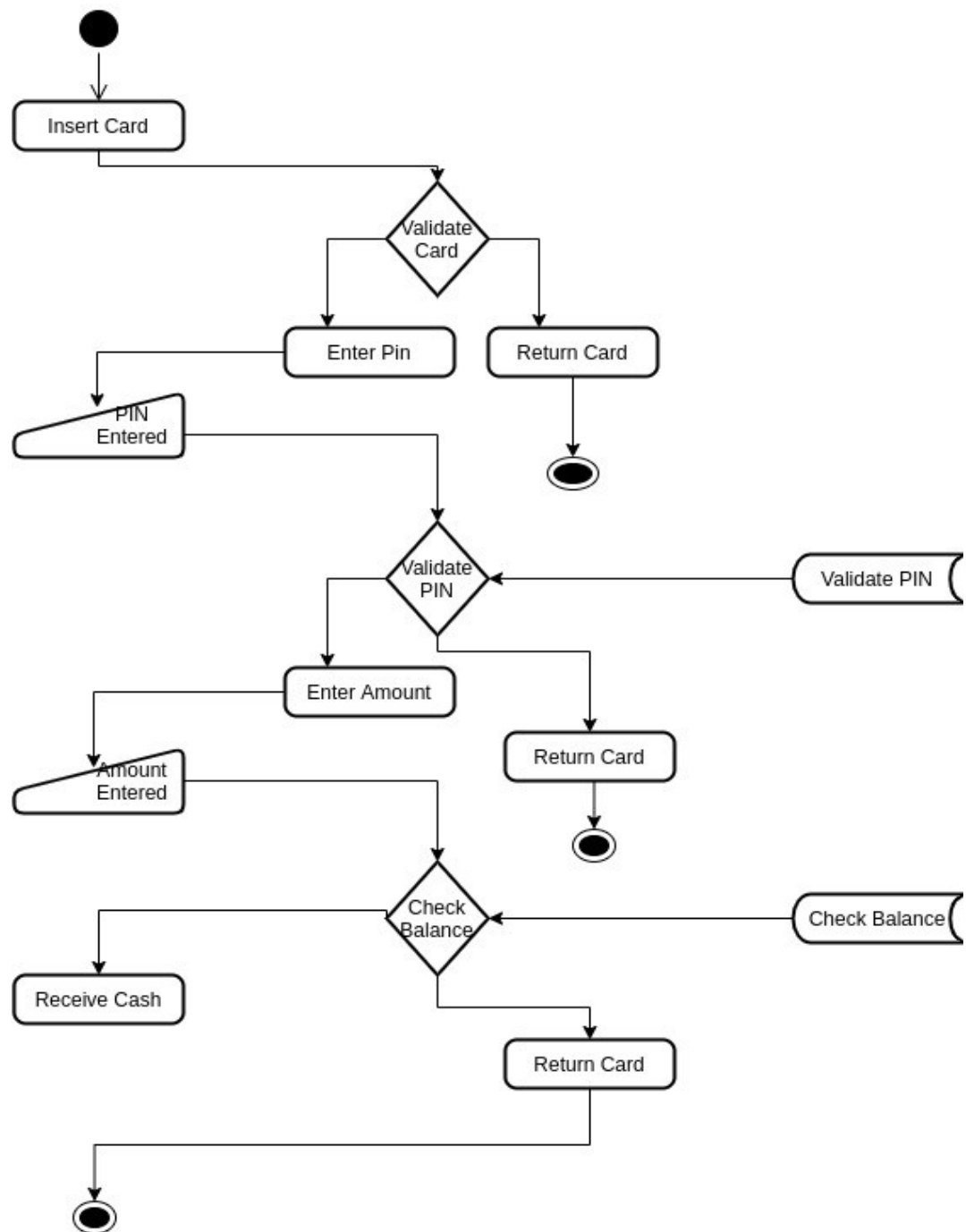
Activity Diagram-

Overall ATM Operations

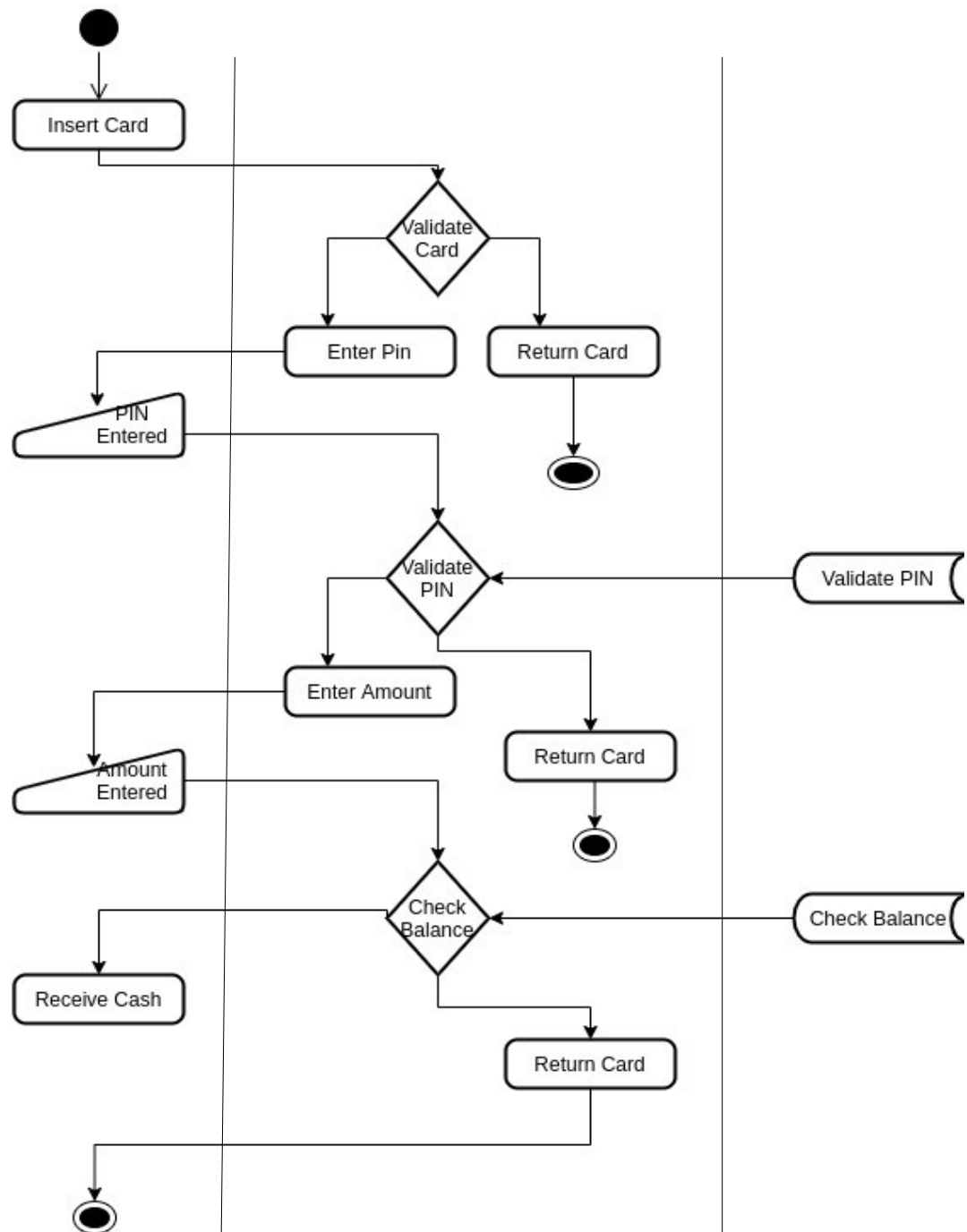
- Insert Card
- If card is valid then ask for the pin
- If pin is valid then ask to select the operation
- Once operation is selected, perform the selected operation
- If no more operations are to be performed take out the card and end the process



Activity Diagram- Withdrawal Operation



Swimlane Diagram- Withdrawal Operation



Requirement Model

Behavioral & Class based Elements

Objectives

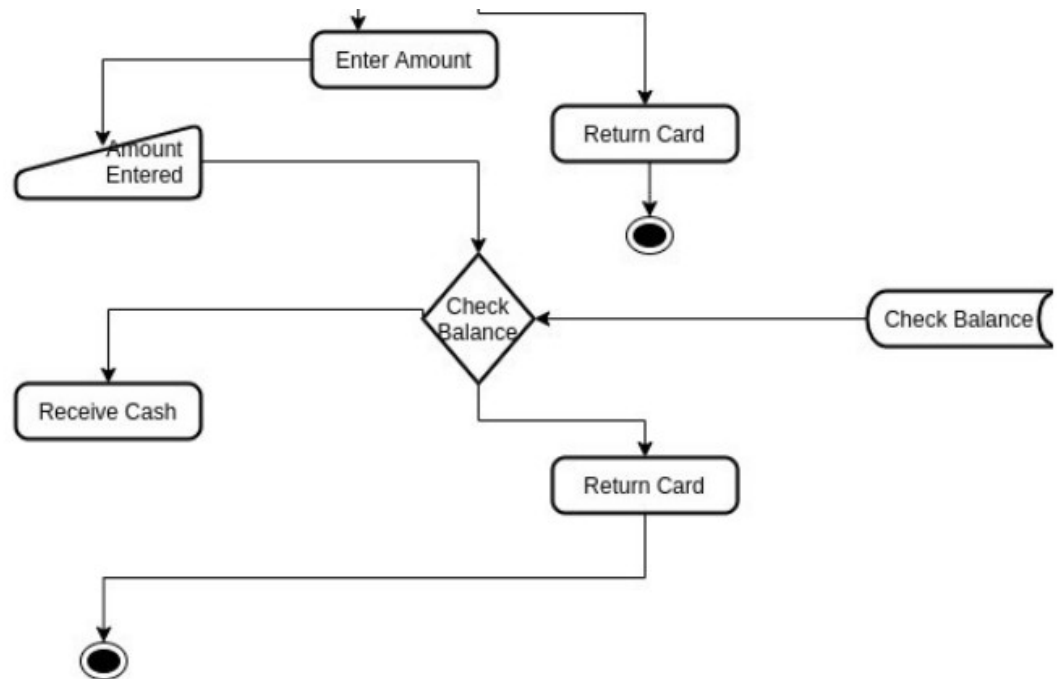
- State Diagram
- Class Diagram

State Diagram

- The UML state machine diagrams depict the various states that an object may be in and the transitions between those states.
- Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered.
- The most important purpose of Statechart diagram is to model lifetime of an object from creation to termination.

State Diagram

- The UML state machine diagrams depict the various states that an object may be in and the transitions between those states.
- Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered.



State Diagram

- The UML state machine diagrams depict the various states that an object may be in and the transitions between those states.
- Statechart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered.
- The most important purpose of Statechart diagram is to model lifetime of an object from creation to termination.

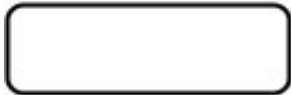
State Diagram- Notations / Symbols



Start Point / Initial State



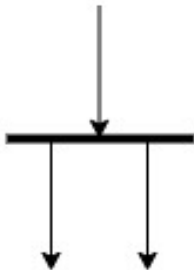
Final State / End Point



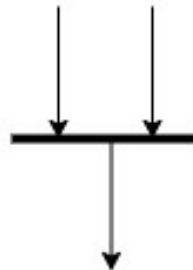
Activity



Action Flow

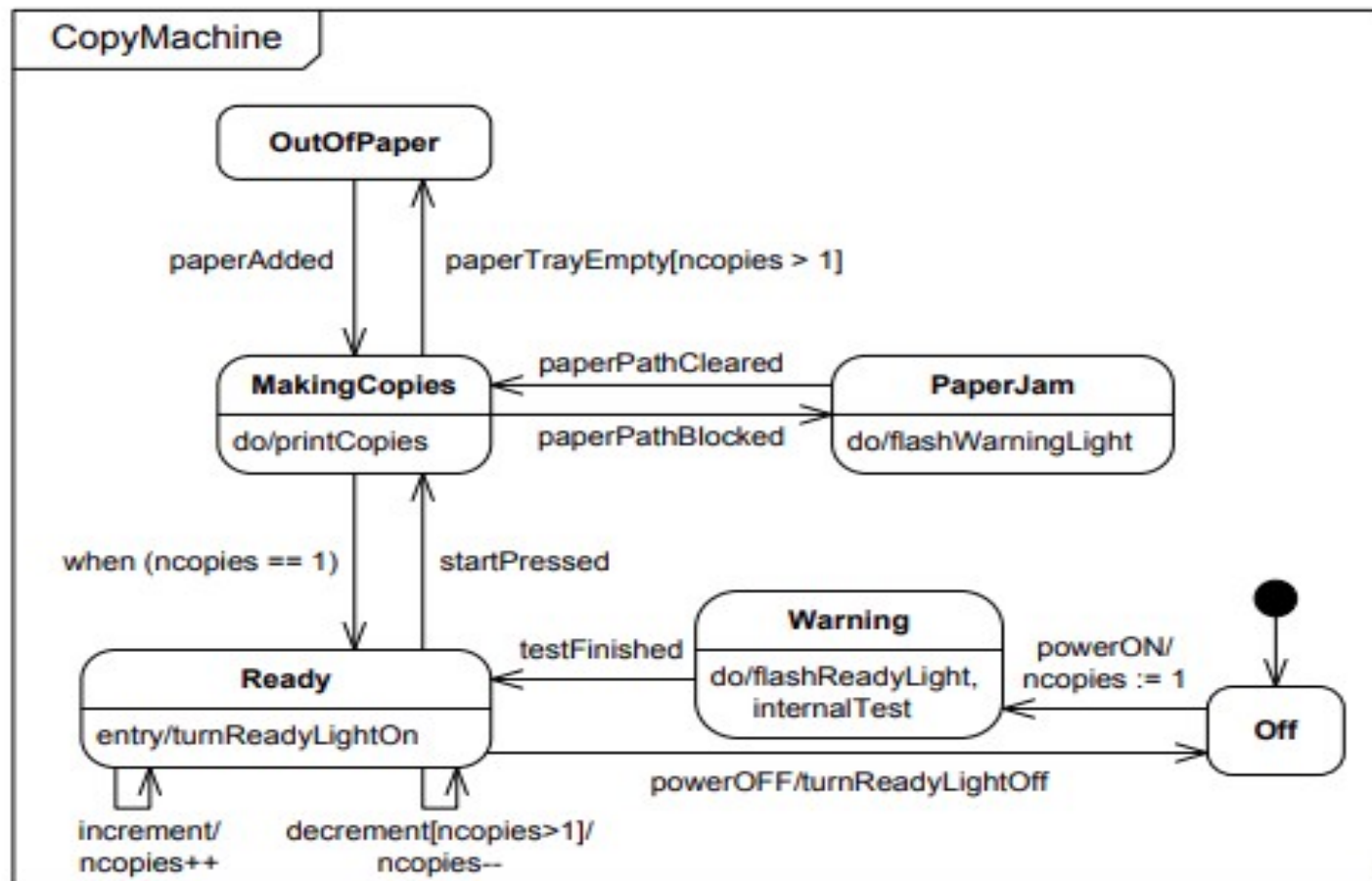


Parallel Activities



Join

State Diagram- Photocopy Machine



Source: <https://courses.cs.ut.ee/2010/sm/uploads/Main/Fall2009Exam.pdf>

Class Diagram

- In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
- A UML class diagram is made up of:
 - A set of classes and
 - A set of relationships between classes

Class Diagram- Class Notations..

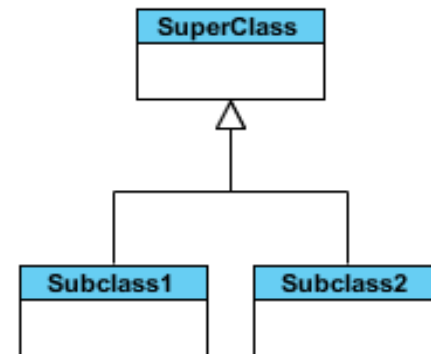
- Class Name
- Class Attributes
- Class Operations

Classname
+ field: type
+ method(type): type

- **Visibility of Class Attributes**
 - + denotes public attributes or operations
 - - denotes private attributes or operations
 - # denotes protected attributes or operations
 - ~ denotes package attributes or operations

Class Diagram- Class Relationships..

- Generalization
 - inheritance



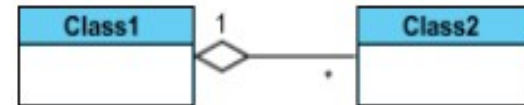
- Simple Association
 - Association between two classes
 - Shown with simple line



Class Diagram- Class Relationships..

- Aggregation

- Class 2 is part of class 1

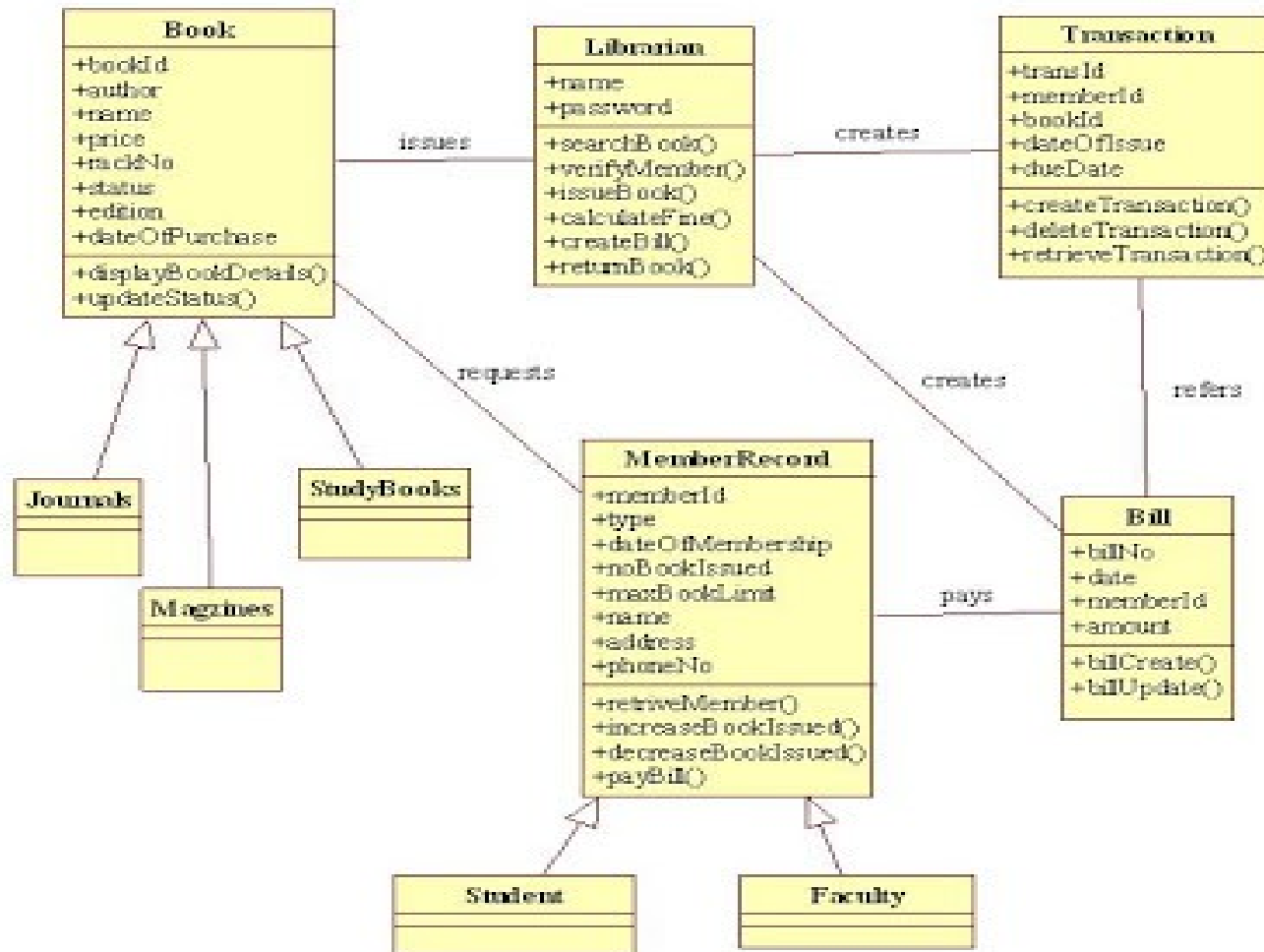


- Dependency

- Class 1 depends on class2
- Any change to class 2 may initiate the change to class 1



Class Diagram- Library System



Thank You