Unit 1
Software Engineering
Lecture 1 and 2

Outline

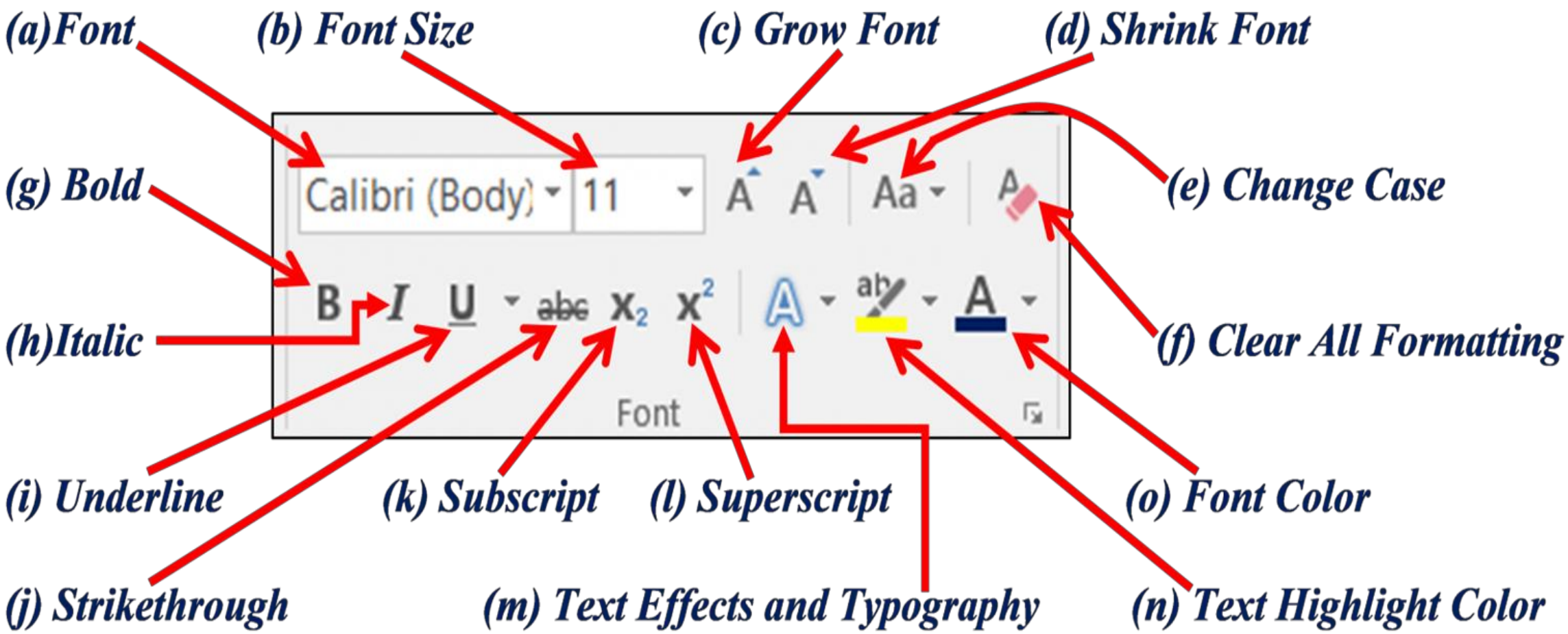
- What is Software?
- Example
- Observations
- Software Engineering
- Software Engineering Effects
- Take away

What is a SOFTWARE?

A Software essentially is a set of programs which enable the users to perform some specific task.

Why mention a SOFTWARE —

Set of Programs for specific task(s)



OBSERVATIONS / KEY FINDINGS

1. Each program corresponds to a special attribute.

2. We need to identify the special attributes.

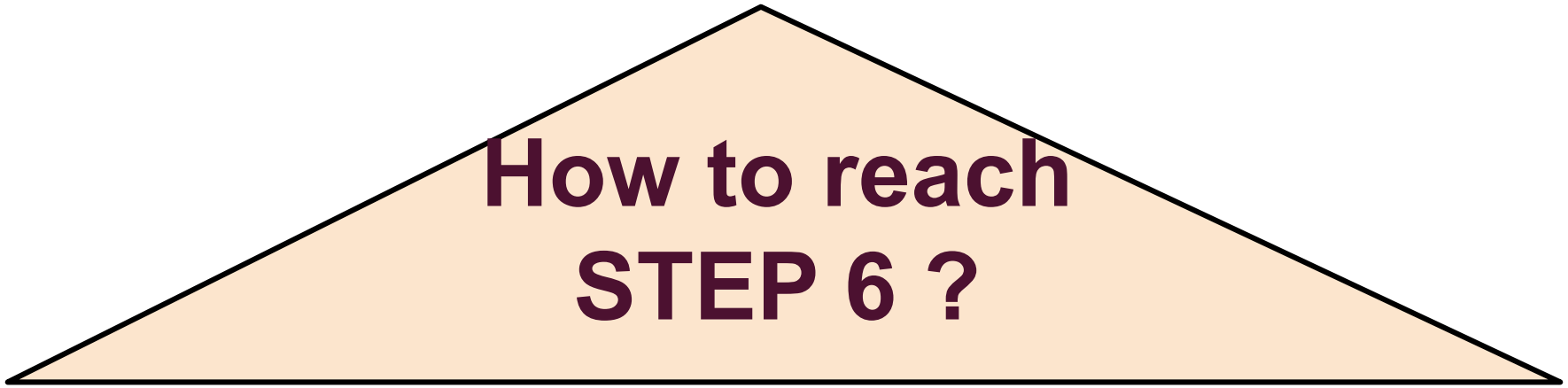
3. We need to find a way to arrange the attributes visually for user.

4. We need to discover a way to integrate all the attributes.

OBSERVATIONS / KEY FINDINGS

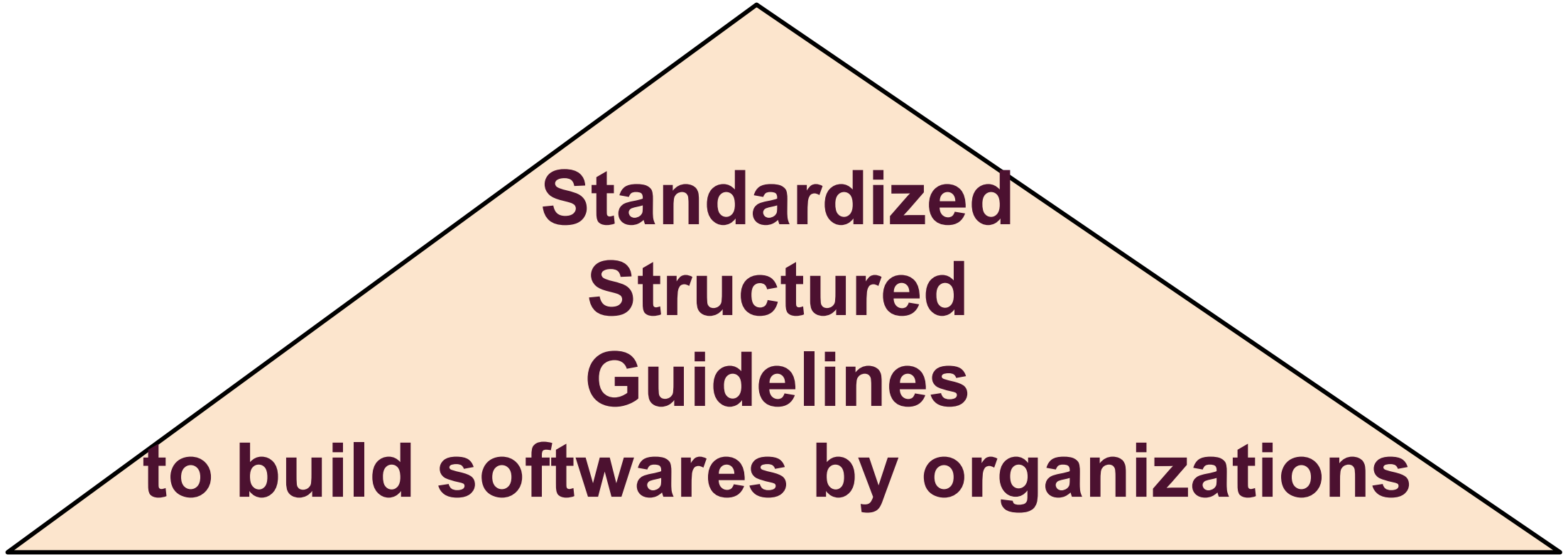
5. We need to verify whether the integration of the attributes is a success or failure.

6. We need to scrutinize whether the software is at par with standards as a product.



SOFTWARE ENGINEERING

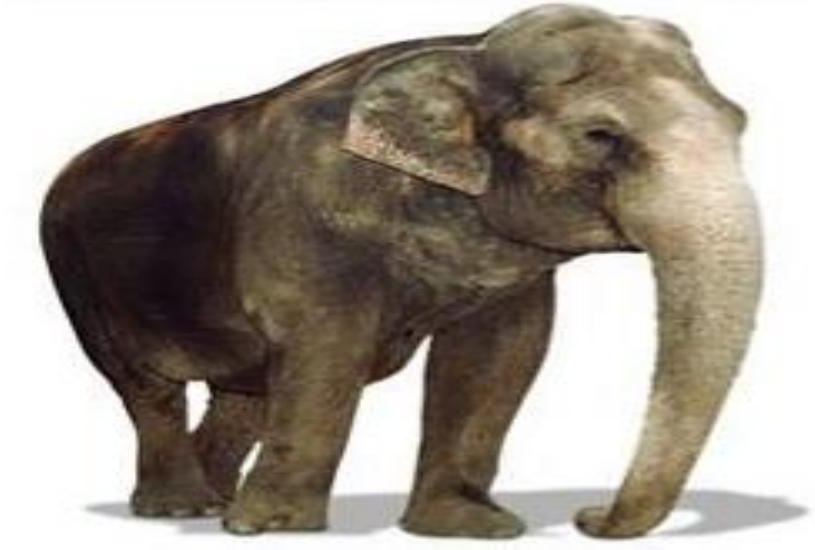
It is the study and practice of engineering to design, build, develop, maintain and retrieve software.

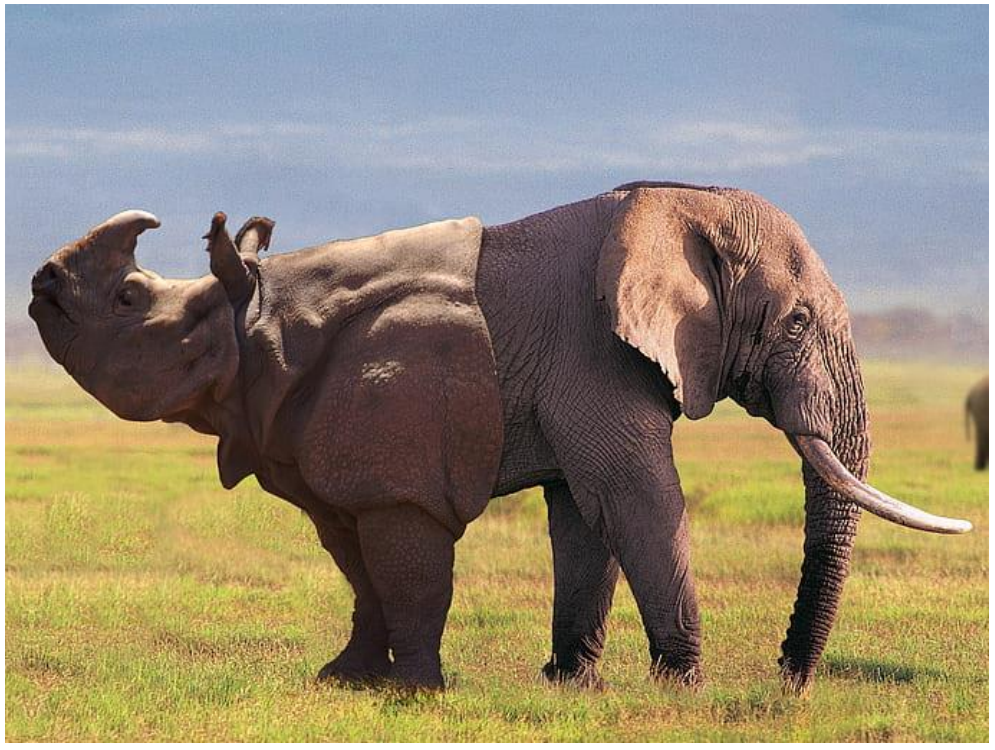


Let's see … Life of a software without Software Engineering

Customer Requirement

- Have one trunk
- Have four legs
- Heavy and provides strength
- Black in color
- Big eyes



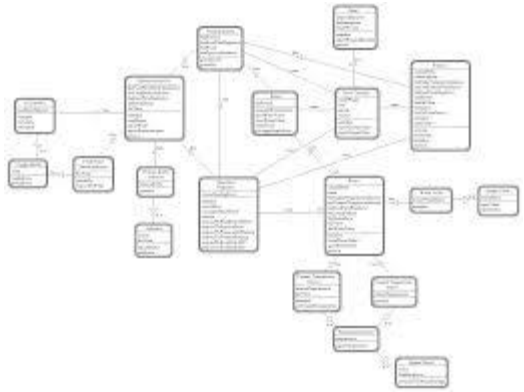


Delivered:
Our value added, have horn with extra strength

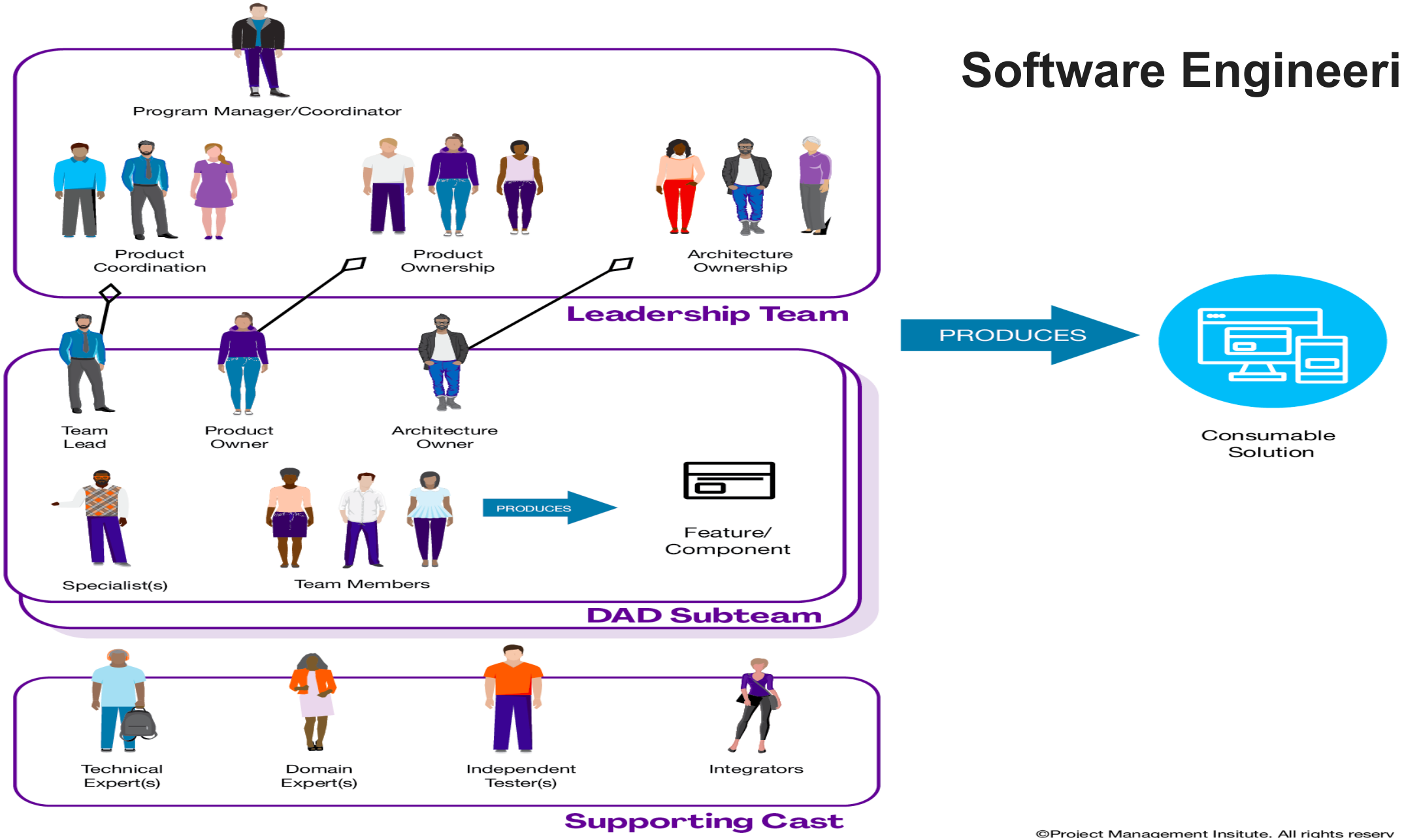
What do we will study in Software Engineering?

- ❑ Good software engineering practices
- ❑ Discipline of how a company develops a software
- ❑ How a software will fulfil its operations i.e meeting its KPIs (key performance indicators)
- ❑ How a software will be maintained

Why Software Engineering is called as an Engineering

Engineering			
	Design	Build	Product
Software Engineering			

Software Engineering



Set of teams are working with some guided principals and standards

Need of Software Engineering

The software development **process** needs to be **engineered** to

avoid the **communication gap** & to **meet the actual requirements**

of customer within **stipulated budget** & **time.**

1. Ola Money - A Bad Design Poorly Implemented --- 2016 – design issue

The cabs are booked via app, driver can start the trip, driver can end the trip, OLA money gets deducted for the travel. The base assumption in this travel is, "the drivers starts and ends the trip in front of the customer". The conceptualization of idea seems very good, but the implementation sucks.

The driver does not reach you, he starts the trip as soon as booking is received and your app shows your trip is "ongoing". And the beauty is, the customer is not even given an option to cancel/monitor the start/stop of the trip. You can watch the driver driving the cab away from you in maps with charges being deducted from your OLA money account, when he marks the trip is complete.

The underlying design where by the driver is given the authority to start the trip could have been articulated better. It leaves gaps in the design. The customer should be involved in making a decision onto starting of the trip.

Later solution

The driver can only start the trip only when customer shares a correct token/OTP/customer pass phrase.

2. Prison Break
– implementation issue

A glitch occurred in December 2015 led to over 3,200 US prisoners being released before their declared date.

The software was designed to monitor the behaviour of prisoners and was introduced in 2002. The problem was occurring for about 13 years and on an average prisoners were released almost 49 days in advance.

The software bug was introduced during the implementation of a change to the "good time" credit system, which allowed inmates to claim credits for good behaviour not only while in state prisons, but also in county jails.

Prison sentences often consist of two parts: the regular one and an "enhancement" added for crimes committed under certain circumstances.

The "good time" credits are meant to only to be applied to the regular part of the sentence, but due to an error in the code they were applied to both..

3. UK government's new online farming payments system gets delayed

---- deadline issues

In March 2015, the UK government delayed the launch of £154 million rural payments system.

The system is an online service for farmers to apply for Common Agricultural Policy payments from the EU.

This online service that was supposed to be up and running by May 2015 got delayed due to integration issues between the portal and the rules engine software.

It was then not expected to be up even by 2016.

4. Glitches in income tax portal
--- reputation loss

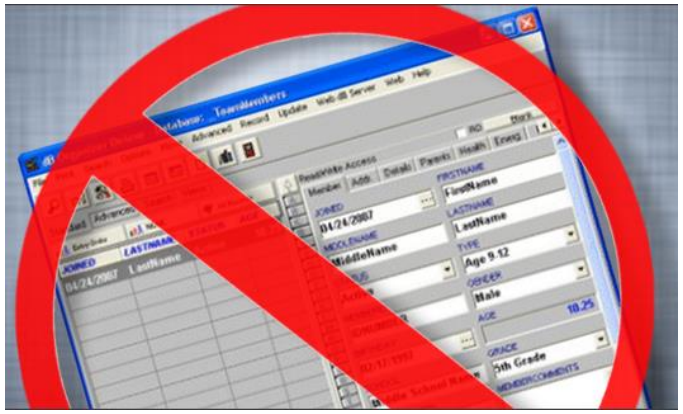
Infosys was in 2019 awarded the contract to develop the new system to reduce processing time for returns from 63 days to one day and expedite refunds.
But the portal developed by Infosys has faced issues in generating passwords, linking data for past returns, and filing of returns. After over two-and-a-half months of continuing glitches in the revamped e-filing portal of the Income Tax Department caused a large financial loss to Indian government.

Within hours of its June 7 launch, the portal started to face issues such as inability to generate an OTP for Aadhaar validation, password generation glitches, failure to link old data for past returns, and problems in filing returns.

The problems have now expanded to include errors in interest calculation, incorrect capturing of details from Form 16, and inability to add details for tax exemption for trusts. Individual taxpayers also complained about their ITR-1 not being accepted even days after submission, inability to e-verify the ITR after filing, Form 26AS details not getting automatically populated, and the portal not having a secure connection and taking time to load properly.

Software is dead…..!

- The **old School view of Software**
  - You buy it
  - You own it &
  - It's your job to manage it
  - That is coming to an end



- Because of **web 2.0** & extensive **computing power**, there is a different generation of software
  - It is delivered via Internet
  - It looks exactly like it's residing on each user's computing device
  - Actually it reside on far away server

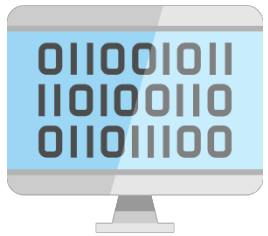
Google™ Apps

Office 365 Business Premium

What is Software?

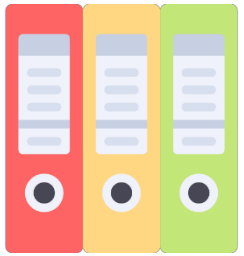
Software is

1) **Computer program** that when executed provide desired features, function & performance
2) **Data Structure** that enable programs to easily manipulate information
3) **Descriptive information** in both hard and soft copy that describes the operation and use of programs



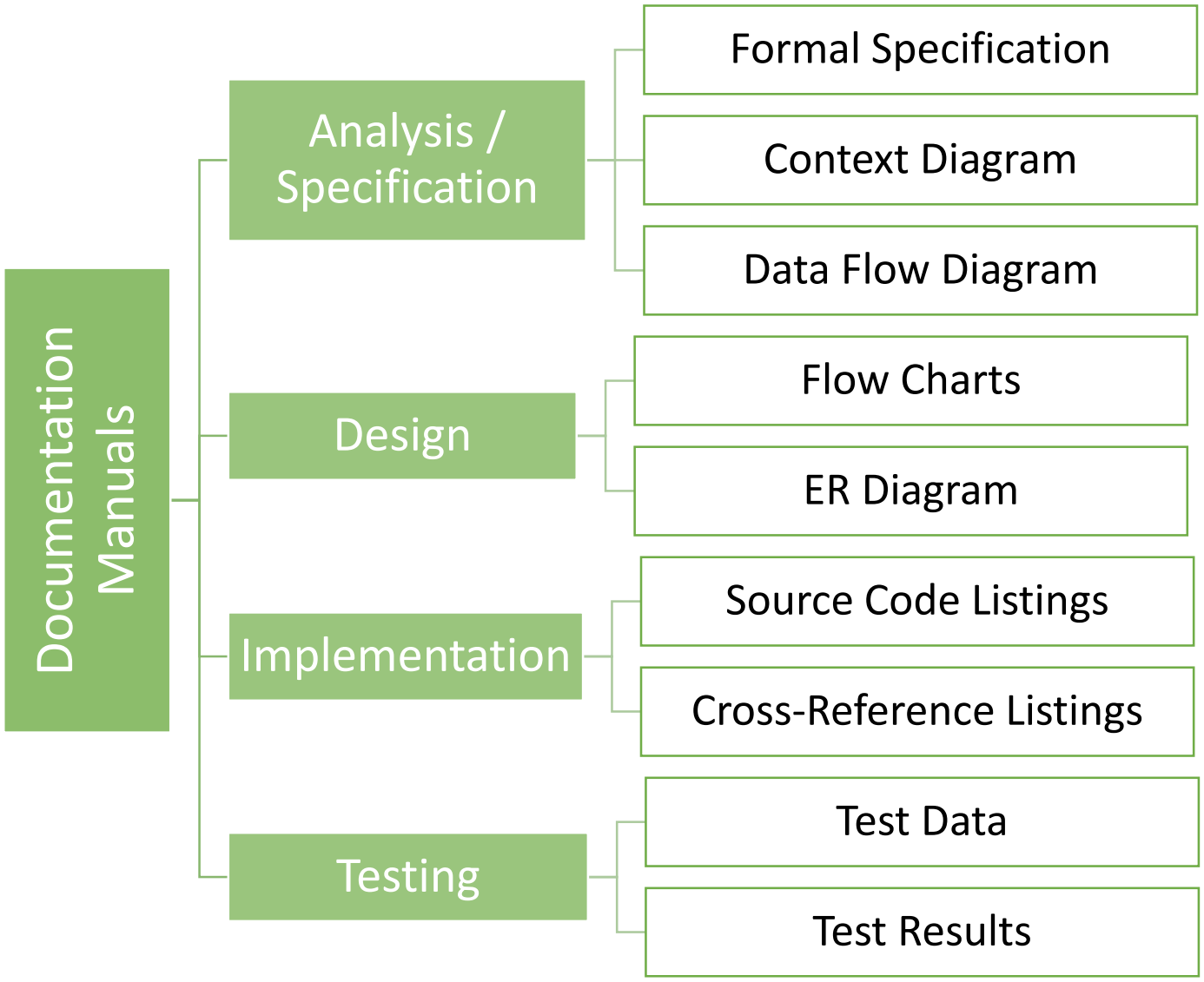
Computer Program

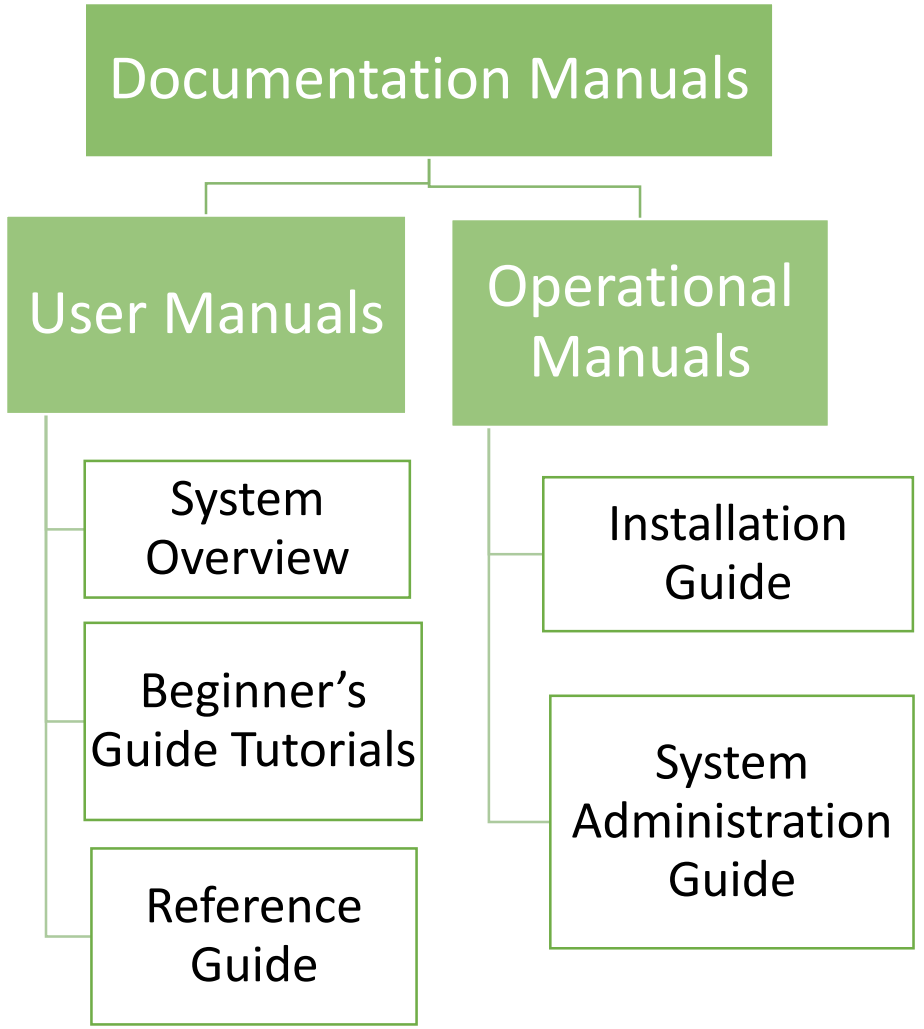
Data Structure



Documents Soft & Hard

List of documentation & manuals





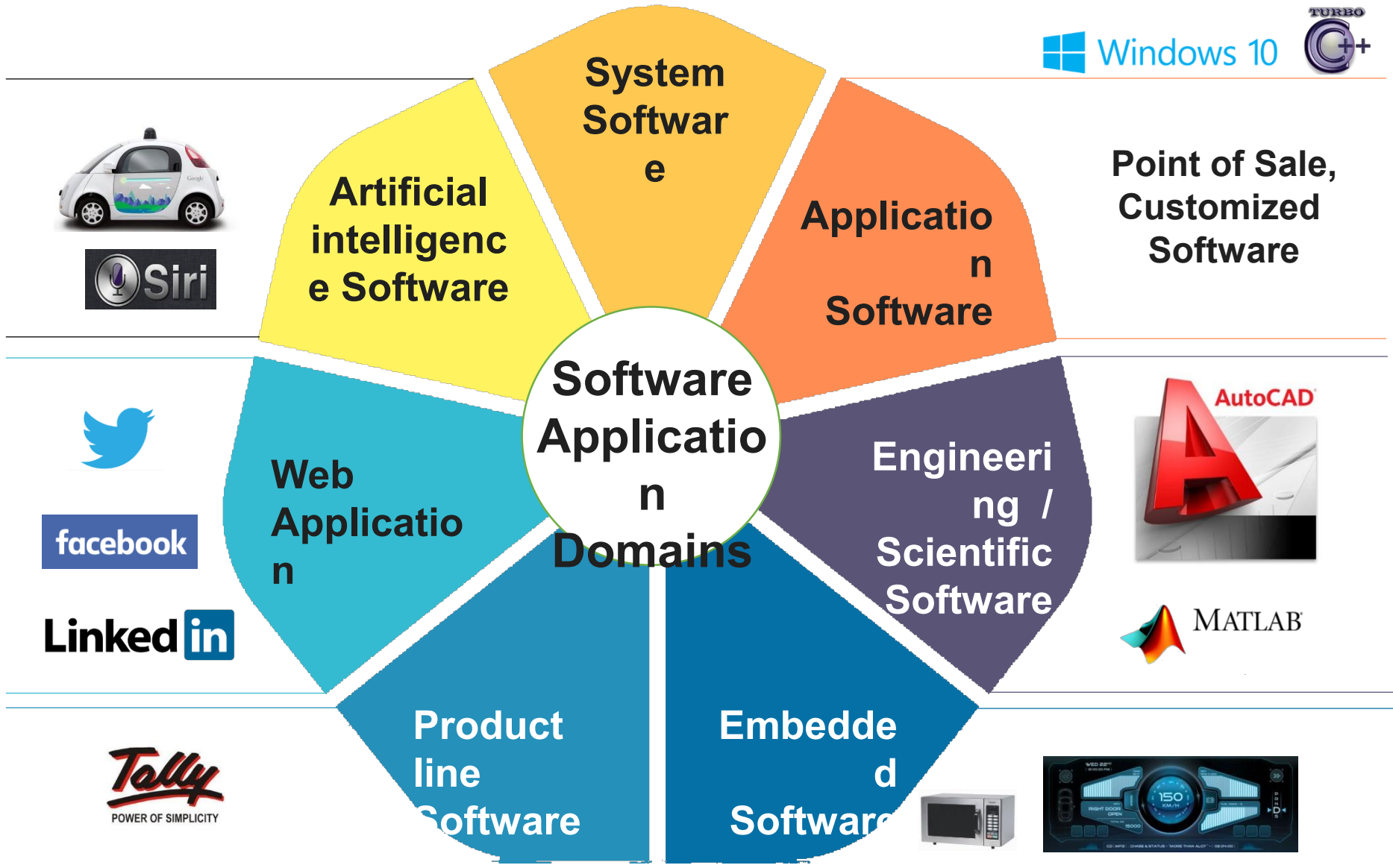
Take Away

- A software is a collection on programs.
- We need standard guidelines and approaches to convert a customer need to a product.
- Good software engineering practices leads to better software development

Next Lecture: Domains of Software, its myths and related models

Software Application Domains

- System Software
- Application Software
- Engineering / Scientific Software
- Embedded Software
- Product line Software
- Web Application



Software Engineering

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

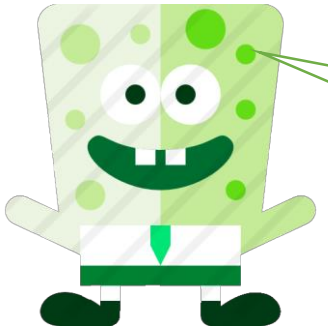
Software Engineering is the science and art of building (designing and writing programs) a software systems that are:

1) on **time**
2) on **budget**
3) with acceptable **performance**
4) with **correct operation**

Software Myths

Beliefs about software and the process used to build it.

"**Misleading Attitudes that cause serious problem**" are myths.



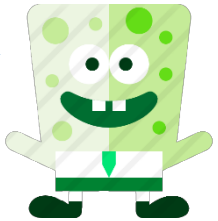
Management Myths

Customer Myths

Practitioner's (Developer) Myths

Management myth - 1

We **have standards and procedures** to build a system, which is enough.



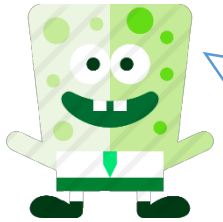
Reality

- Are software **practitioners aware** of standard's existence?
- Does it **reflect modern software engineering** practice?
- Is it **complete**? Is it **adaptable**?
- Is it streamlined to **improve time to delivery** while **still maintaining a focus on quality**?
- In many cases, the answer to all of these questions is "no.“

Management myth - 2 & 3

Mongolian horde

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



I **outsourced the development** activity, now I **can relax** and **can wait** for the **final** working **product**.

Reality

- **People** who were **working** must **spend time educating** the newcomers.
- People can be added but only **in a planned and well-coordinated** manner.

Reality

- If an **organization** does **not understand how to manage** and **control** software projects internally, it will invariably struggle when it outsources software projects.

Customer myth - 1 & 2

A **general statement of objectives** (requirements) is **sufficient** to start a development.

Requirement Changes can be **easily accommodated** because software is very flexible.

Reality

- Comprehensive (**detailed**) **statements** of requirements is not always possible, an **ambiguous** (unclear) "**statement of objectives**" can lead to disaster.
- Unambiguous (clear) requirements can be gathered only through effective and continuous communication between customer and developer.

Reality

- It is true that software **requirements change**, but the **impact** of change **varies with the time** at which it is introduced.
- When requirements changes are requested early the cost impact is relatively small.

The customer believes myths about software because software managers and practitioners do little to correct misinformation.
Myths lead to false expectations (by the customer) and, ultimately, dissatisfaction with the developer.

Practitioner's (Developer) myth – 1 & 2

Once we **write** the **program**, our **job is done**.

I **can't** access **quality until** it is **running**.

Reality

- Experts say **"the sooner you begin 'writing code', the longer it will take you to get done."**
- Industry data indicates that 60 to 80 % effort expended on software will be after it is delivered to the customer for the first time.

Reality

- One of the most effective software **quality assurance mechanisms** can be **applied from** the **beginning** of a project using Walkthrough, Checkpoint Review, Prototype Testing.
- Ask **pilot software reviews**. These are more effective "quality filter" than testing for finding software defects.

Practitioner's (Developer) myth – 3 & 4

Working **program** is the **only deliverable** work **product**.

Software engineering is about **unnecessary** documentation.

Reality

- **A working program** is only one **part of** a **software configuration**.
- A variety of work products (e.g., **models, documents, plans**) provide a foundation for successful engineering and, more important, guidance for software support.
- Proper Documentation is required for customer and developers.

Reality

- Software engineering is not about creating documents. It is about **creating a quality product**.
- Better quality leads to reduced re-work. And reduced rework results in faster delivery times.

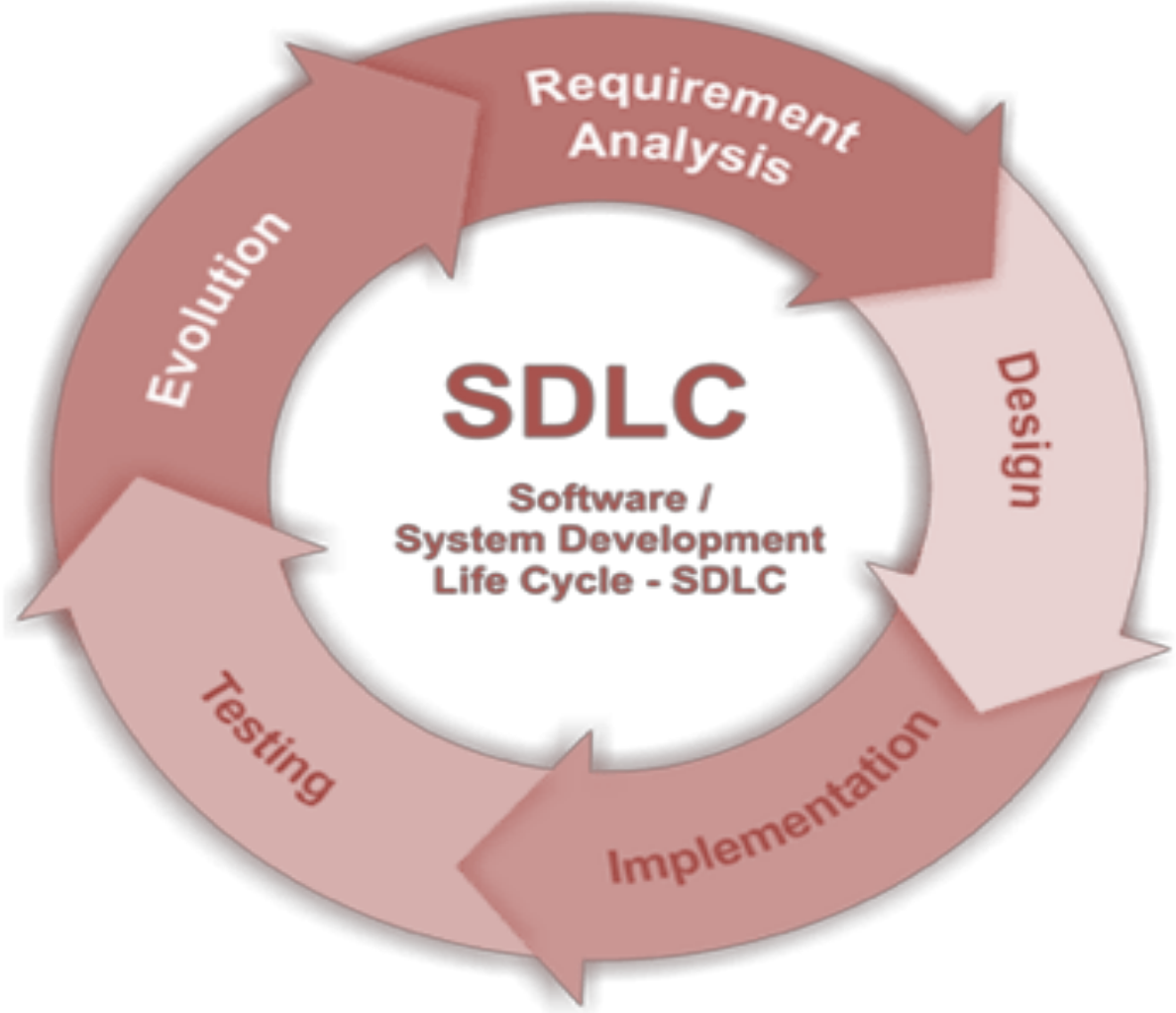
Software Development Process

The life cycle of a software represents the series of identifiable stages through which it evolves during its life time.

Software life cycle has been defined to imply the different stages (or phases) over which a software evolves from the initial customer request (inception) for it, to a fully developed software (phases), and finally to a stage where it is no longer useful (maintenance) to any user, and is discarded.

Software development life cycle (SDLC)

Describes the different activities that need to be carried out for the software to evolve in its life cycle.



Also called *Software life cycle model* and *Software development process model*

Software development life cycle: Definition and Goal

The Software Development Life Cycle (SDLC) is a structured process that enables the production of high-quality, low-cost software, in the shortest possible production time.

The goal of the SDLC is to produce superior software that meets and exceeds all customer expectations and demands.

Different SDLC Models

- **Which type of SDLC model to adopt?**

- Process model is selected based on different parameters:
  - Type of the project & people
  - Complexity of the project
  - Size of team
  - Expertise of people in team
  - Working environment of team
  - Software delivery deadline

TYPES

- Waterfall Model (Linear Sequential Model)
- Incremental Process Model
- Prototyping Model
- The Spiral Model
- Rapid Application Development Model
- Agile Model

The Waterfall Model

When to use

- **Requirements** are very well **known**, **clear** and **fixed**
- Product **definition** is **stable**
- **Technology** is **understood**
- There are **no ambiguous** (unclear) **requirements**
- Ample (**sufficient**) **resources** with required **expertise** are **available** freely
- The **project** is **short**

Advantages

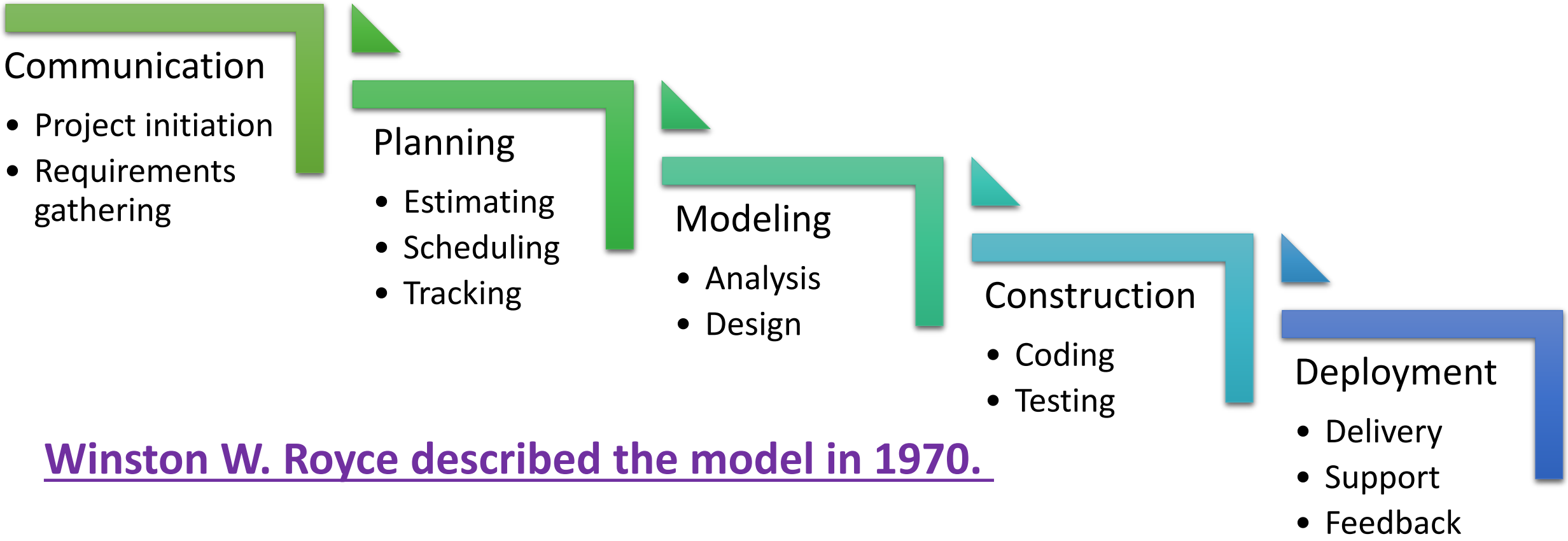
- **Simple to implement** and manage

Drawbacks

- **Unable to accommodate changes** at **later stages**, that is required in most of the cases.
- **Working version** is **not available** during development. Which can lead the development with major mistakes.
- **Deadlock can occur** due to delay in any step.
- **Not suitable** for **large projects**.

The Waterfall Model

Classic life cycle or linear sequential model



Winston W. Royce described the model in 1970.

When **requirements** for a problems are **well understood** then this model is used in which **workflow** from communication to deployment in **linear manner**

The Waterfall Model

Larger teams take control of the project, in some cases can consist of more than 15 people in non-interchangeable groups. These individuals involved abide by a strict hierarchy, with the lead role going to the project manager. The hierarchy is defined as:

1) **PROJECT MANAGER – THE LEADER & DELEGATOR --** principal responsibility is to ensure the project execution respecting its scope, cost and time

2) **BUSINESS ANALYST --** primary responsibilities include liaising between IT and the executive branch, improving the quality of IT services, and analyzing business needs.

3) **DEVELOPERS – THE CODEMAKERS –** technical responsibilities

4) **TESTERS – THE CODEBREAKERS ---** Validation and verification responsibilities

5) **OTHER ROLES**

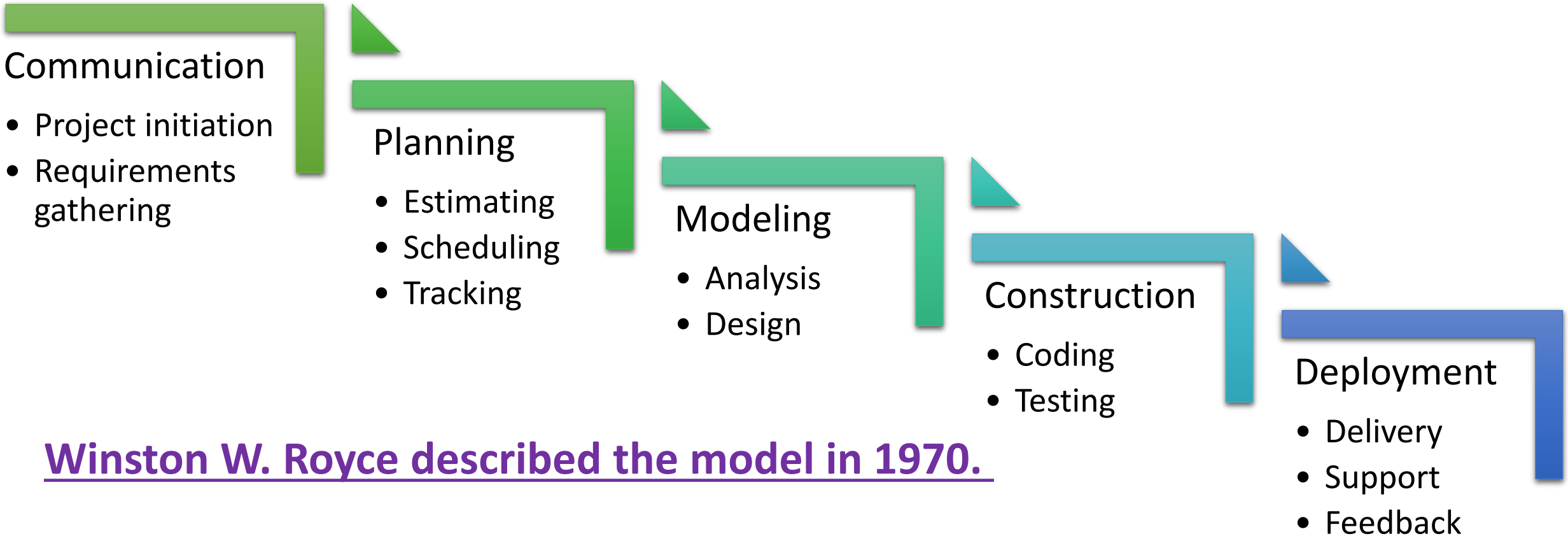
a) **TECHNICAL ARCHITECT / SOLUTIONS ARCHITECT** : their role entails designing the structure of IT systems, and oversight of business-optimizing programs.
b) **SYSTEM ADMINISTRATION AND HELPDESK** –Their job is to organize, install, and support an organization's hardware and computer systems, including LAN and Intranet.
c) **QUALITY MANAGER** – Role is responsible for the final quality of software and ensures that the project is performed according to defined processes.

Next Lecture: Other models and their differentiating aspects

Unit 1
Software Engineering
Lecture 3

The Waterfall Model

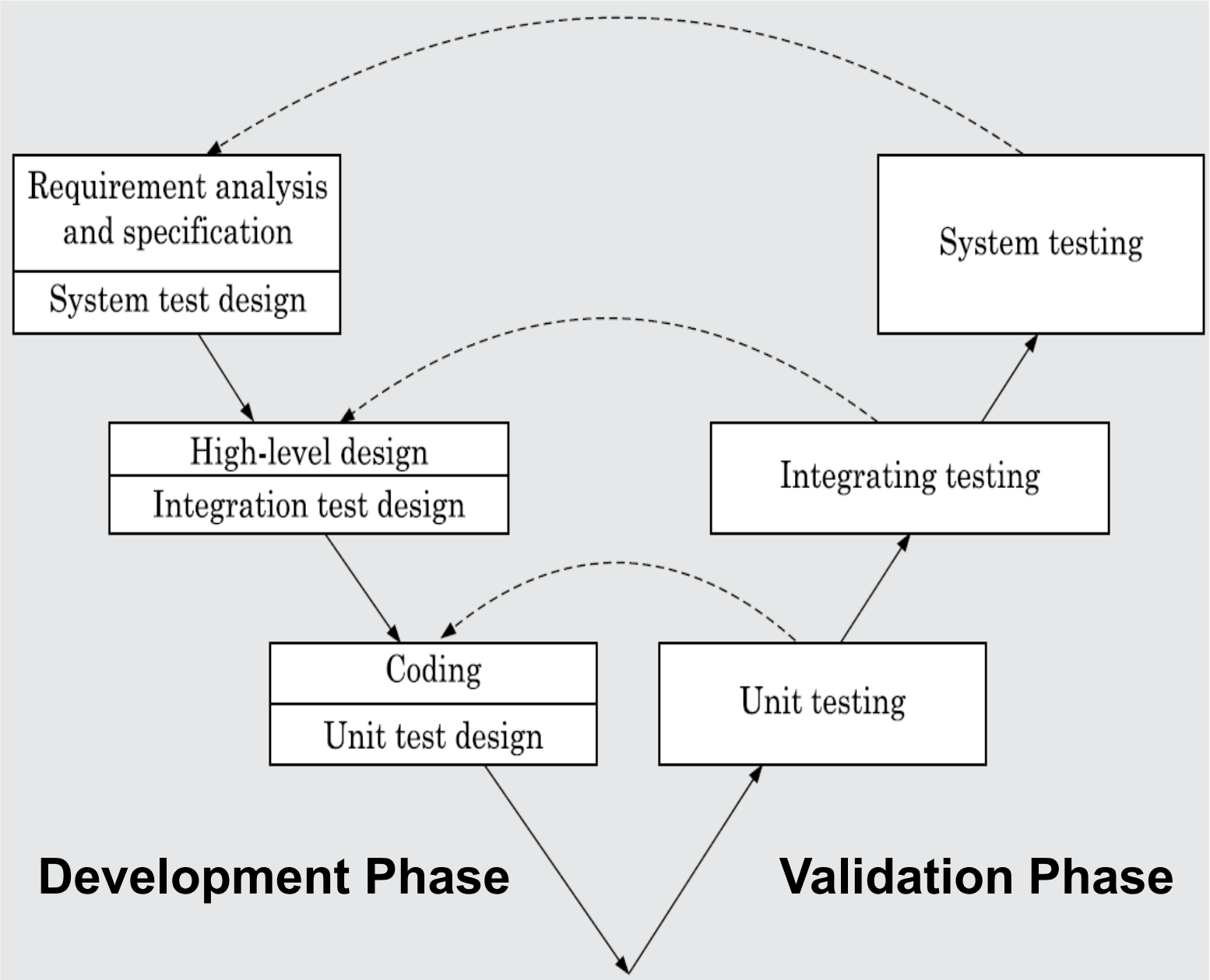
Classic life cycle or linear sequential model



Winston W. Royce described the model in 1970.

When **requirements** for a problems are **well understood** then this model is used in which **workflow** from communication to deployment in **linear manner**

V-Model

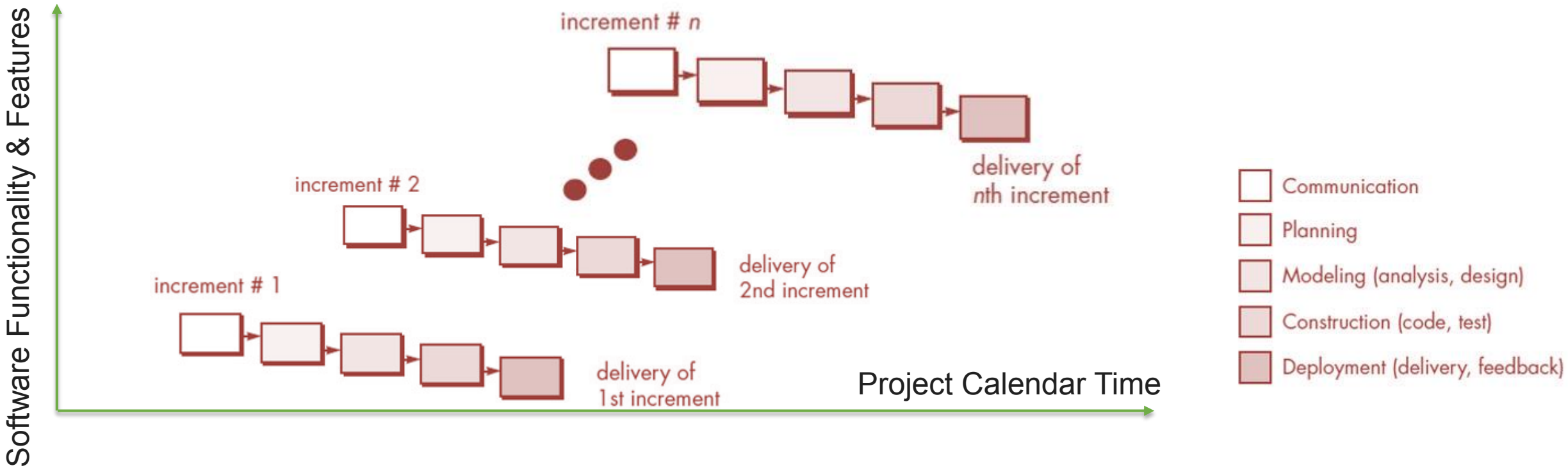


- V-model is a variant of the waterfall model.
- Verification and validation activities are carried out throughout the development life cycle, therefore the chances bugs in the work products reduce.
- Suitable for use in projects concerned with development of safety-critical software that are required to have high reliability.

Incremental Process Model

- There are many situations in which **initial software requirements** are reasonably **well defined**, but the **overall scope of the development** effort prevent a purely linear process.
- In addition, there may be a **compelling need** to provide a **limited set of software functionality** to users **quickly** and then **refine and expand on that functionality** in later software releases.
- In such cases, there is a need of a process model that is designed to produce the software in increments.

Incremental Process Model



- The incremental model **combines** elements of **linear** and **parallel** process flows.
- This model applies linear sequence in an iterative manner.
- Initially **core working product** is **delivered**.
- **Each** linear **sequence** produces deliverable **"increments"** of the software.

Incremental Process Model

e.g., **word-processing software** developed **using** the **incremental model**

- It might deliver basic file management, editing and document production functions in the first increment
- more sophisticated editing in the second increment;
- spelling and grammar checking in the third increment; and
- advanced page layout capability in the fourth increment.

When to use

- When the **requirements** of the **complete** system are clearly **defined** and understood but **staffing is unavailable** for a **complete implementation** by the business deadline.

Advantages

- Generates **working software quickly** and early during the software life cycle.
- It is **easier to test** and debug during a smaller iteration.
- **Customer** can **respond** to each built.
- **Lowers initial** delivery **cost**.
- **Easier** to **manage risk** because risky pieces are identified and handled during iteration.

Evolutionary Process Models

- **Design a little, build a little, test a little, deploy a little model**

- In the incremental development model, complete requirements are first developed and the requirement document is prepared.
- Designed to accommodate **product** that **evolve with time**.
- **In the evolutionary model, the requirements, plan, estimates, and solution evolve over the iterations, rather than fully defined and frozen in a major up-front specification effort before the development iterations begin.**

- Evolutionary models are **iterative**.
- Evolutionary models are :
  - **Prototyping model**
  - **Spiral model**

Prototyping model

When to use

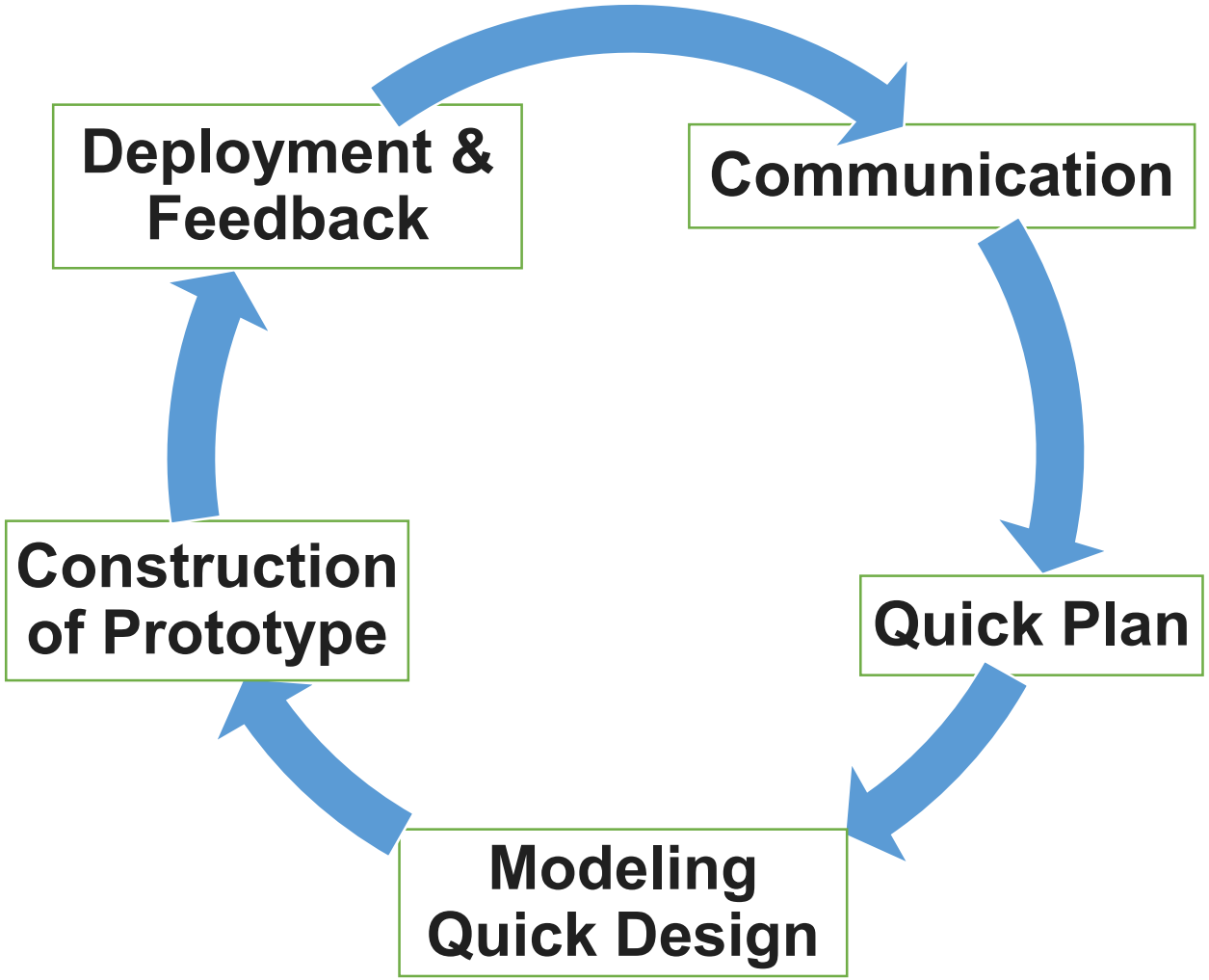
- **Customers have** general **objectives of software** but **do not have detailed requirements** for functions & features.
- **Developers** are **not sure** about **efficiency of an algorithm** & **technical feasibilities**.

- Suggests building a working prototype of the system, before development of the actual software.
- Prototype can be served as "**the first system**".
- **"Quick and dirty"** – quality not important, scripting etc.
- Things like exception handling, recovery, standards are **omitted**
- A prototype is a toy and crude implementation of a system. It has limited functional capabilities, low reliability, or inefficient performance as compared to the actual software.
- Rapid Prototyping like simulation tool application.

Prototyping model cont.

Example: 1 Design the compiler for a small language and then extend it for full language.
2. You train data for small set, verify and then do for the large problem.

It works as



- **Communicate** with stockholders & **define objective** of Software
- **Identify requirements** & design **quick plan**
- **Model** a quick **design** (focuses on visible part of software)
- **Construct Prototype** & deploy
- Stakeholders **evaluate** this **prototype** and provides **feedback**
- Iteration occurs and **prototype** is **tuned** based on **feedback**

Prototyping model cont.

Problem Areas

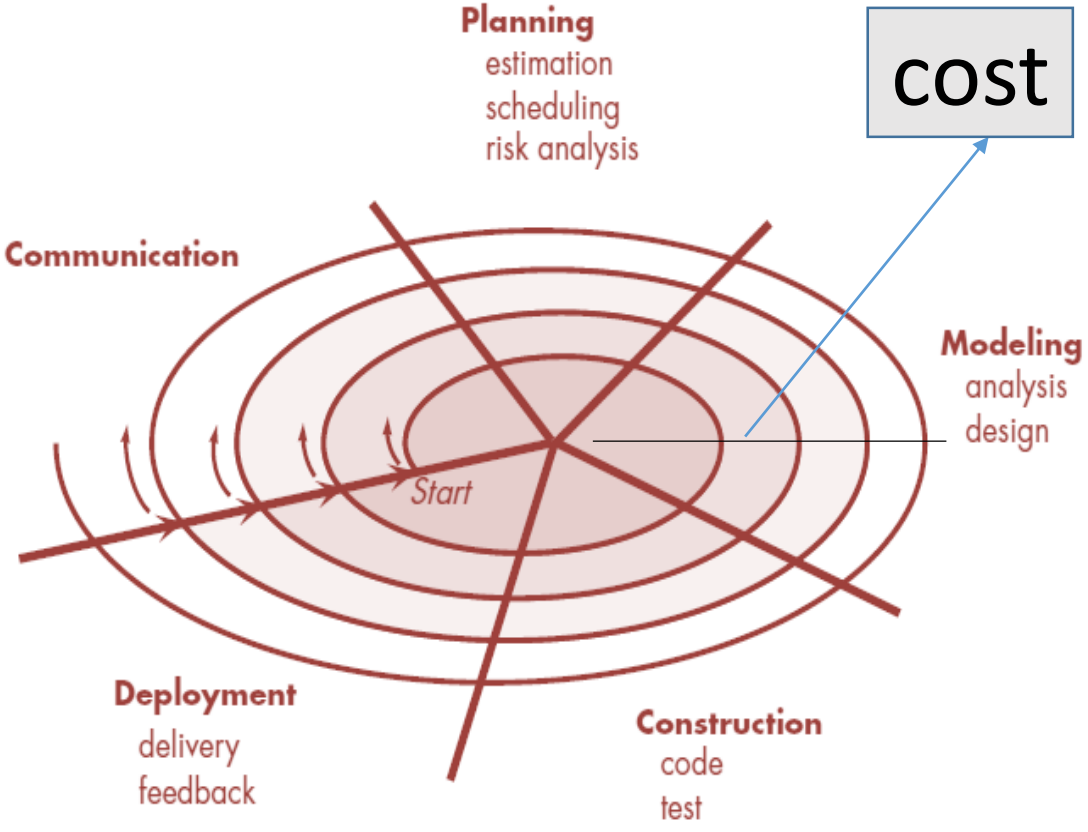
- Potential hit on **cost** and **schedule**
- **Customer demand** that "**a few fixes**" be applied to **make** the **prototype a working product**, due to that software quality suffers as a result
- **Developer** often makes **implementation** in order to get a prototype working quickly; **without considering other factors** in mind like OS, Programming language, etc.
- Potential false sense of security if prototype does not focus on key (**high risk**) issues

Advantages

- **Users** are actively **involved** in the **development**
- Since in this methodology a working model of the system is provided, the **users get a better understanding** of the **system** being developed
- **Errors** can be **detected** much **earlier**

The Spiral Model

Proposed by Barry Boehm in 1986



Incremental Risk Oriented Project Model

- It provides the **potential** for **rapid development**.
- Software is developed in a series of evolutionary releases that mainly focuses on **Risk Handling**.
- **Early iteration** release might be **prototype** but **later iterations** provides more **complete version of software**.
- It is divided into framework activities (C,P,M,C,D). Each activity represent one segment of the spiral
- **Each pass** through the **planning** region results in **adjustments** to
  - the **project plan**
  - **Cost** & **schedule** based on feedback

Next Lecture: RAD case study and AGILE Models

Unit 1
Software Engineering
Lecture 4

The Spiral Model Cont. Meta Model

When to use Spiral

- For development of **large scale / high-risk projects**.
- When costs and **risk evaluation is important**.
- Users are **unsure** of their **needs**.
- **Requirements** are **complex**.
- New product line.
- Significant (**considerable**) **changes** are expected.

Examples: Mission and Applicability Critical Software

Advantages

- High amount of risk analysis hence, **avoidance of Risk** is enhanced.
- **Strong approval** and **documentation** control.
- **Additional functionality** can be **added** later.
- **Software** is **produced early**

Disadvantages

- Can be **a costly model** to use.
- Risk analysis **requires highly specific expertise**.
- Project's success is highly dependent on the risk analysis phase.
- Doesn't work well for smaller projects.

Rapid Application Development Model (RAD)

RAD is based on four core concepts:

1. **Reusable code in an accessible repository**
2. **Quick prototyping**
3. **Constant client feedback**
4. **Building usable software as quickly as possible**

The client and developer work together to create an end product that functions exactly as the client wants.

Rapid Application Development Model (RAD)

Working

Development takes place in a series of short cycles or iterations.

At any time, the development team focuses on the present iteration only, and therefore plans are made for one increment at a time.

The time planned for each iteration is called a *time box i.e deadlines*.

Each iteration is planned to enhance the implemented functionality of the application by only a small amount.

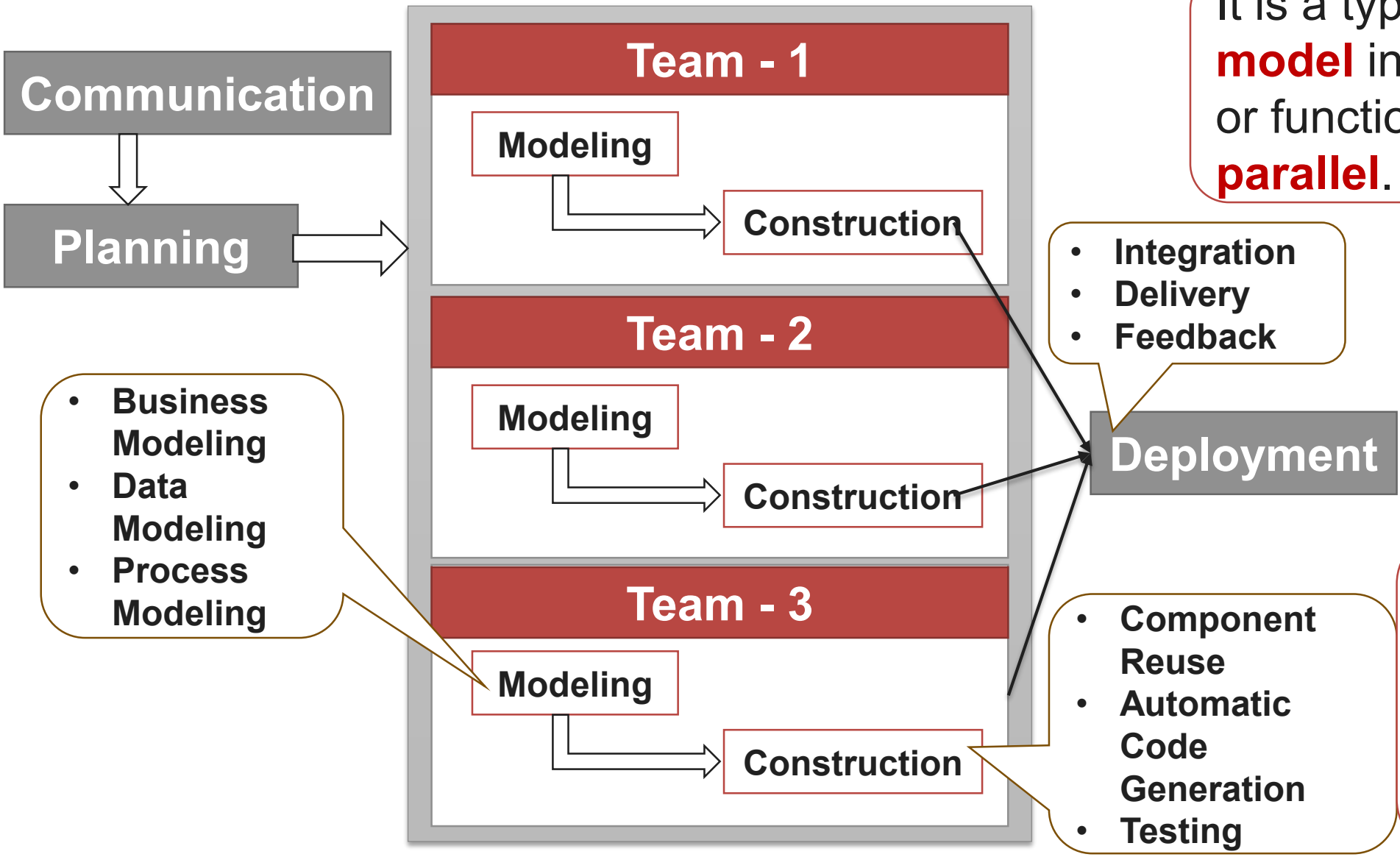
During each time box, a quick-and-dirty prototype-style software for some functionality is developed.

The customer evaluates the prototype and gives feedback on the specific improvements that may be necessary. The prototype is not meant to be released to the customer for regular use though.

The prototype is refined based on the customer feedback.

Rapid Application Development Model (RAD)

It is a type of **incremental model** in which; **components** or functions are **developed in parallel**.



- Business Modeling
- Data Modeling
- Process Modeling

- Integration
- Delivery
- Feedback

- Component Reuse
- Automatic Code Generation
- Testing

This can **quickly give** the customer **something to see** and use and to provide feedback.

Rapid Application Development Model (RAD) Cont.

Communication

- This phase is used to understand business problem.

Planning

- Multiple software teams work in parallel on different systems/modules.

Modeling

- **Business Modeling:** *Information flow* among the business. Eg. Class dia
  - Ex. What kind of information drives (moves)?
  - Who is going to generate information?
  - From where information comes and goes?
- **Data Modeling:** Information refine into set of *data objects* that are *needed* to support business. Eg. DFD, Flowchart
- **Process Modeling:** *Data object* transforms **to** *information flow* necessary to implement business. Eg. State dig, sequence dia, use case

Construction

- It highlighting the *use of pre-existing software component*.

Deployment

- **Integration of modules** developed by parallel teams, **delivery** of integrated software and **feedback** comes under deployment phase.

Rapid Application Development Model (RAD) Cont.

When to Use?

- There is a need to create a **system** that can be **modularized in 2-3 months** of time.
- **High availability** of **designers** and **budget** for modeling along with the cost of automated code generating tools.
- **Resources** with **high** business **knowledge** are available.

Rapid Application Development Model (RAD) Cont.

Advantages

- **Reduced** development **time**.
- **Increases reusability** of components.
- **Quick** initial **reviews** occur.
- **Encourages** customer **feedback**.
- Integration from very beginning **solves** a lot **of integration issues**.
- **Bridges communication** gap between users and developers.

Drawback

- For **large** but scalable **projects**, RAD **requires sufficient human resources**.
- Projects **fail if developers** and **customers** are **not committed** in a much-shortened time-frame.
- **Problematic** if system **can not be modularized**.
- **Not appropriate when technical risks are high** (heavy use of new technology).

Agility

Agility is **ability to move quickly and easily**.

It is an ability of **quickness**, **lightness**, & **ease of movement**.

- The ability to **create** and **respond to change** in order to profit in an unstable global business environment.
- The ability to **quickly reprioritize use of resources** when requirements, technology, and knowledge shift.
- A very **fast response to sudden** market **changes** and emerging threats by intensive customer interaction.
- Use of **evolutionary**, **incremental**, and **iterative delivery** to converge on an optimal customer solution.
- Maximizing **BUSINESS VALUE** with **right sized**, **just- enough**, and **just-in-time processes** and **documentation**.

VALUES OF AGILE

PEOPLE OVER PROCESSES AND TOOLS

WORKING SOFTWARE OVER COMPREHENSIVE DOCUMENTATION

CUSTOMER COLLABORATION OVER RIGID CONTRACTS

RESPONDING TO CHANGE RATHER THAN FOLLOWING A PLAN

Agile Process

- Agile software process addresses **few assumptions**

Difficulty in predicting changes of requirements and customer priorities.

For many types of software; **design** and **construction** are **interleaved** (mixed).

Analysis, **design**, **construction** and **testing** are **not** as **predictable** as we might like.

- An agile **process** must **adapt** incrementally.
- To accomplish incremental adaptation, an agile team **requires customer feedback** (so that the appropriate adaptations can be made).

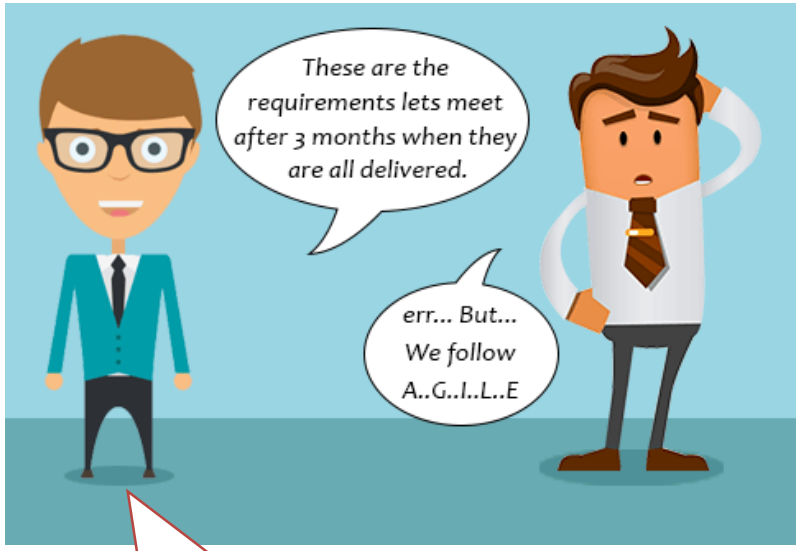
PRINCPILES OF AGILE

- Satisfy the Customer
- Welcome Changing Requirements
- Deliver Working Software Frequently
- Frequent Interaction with Stakeholders
- Motivated Individuals
- Face-to-Face Communication
- Measure by working software
- Maintain constant pace
- Sustain technical excellence and good design
- Keep it simple
- Empower self-organizing teams
- Reflect and Adjust continuously

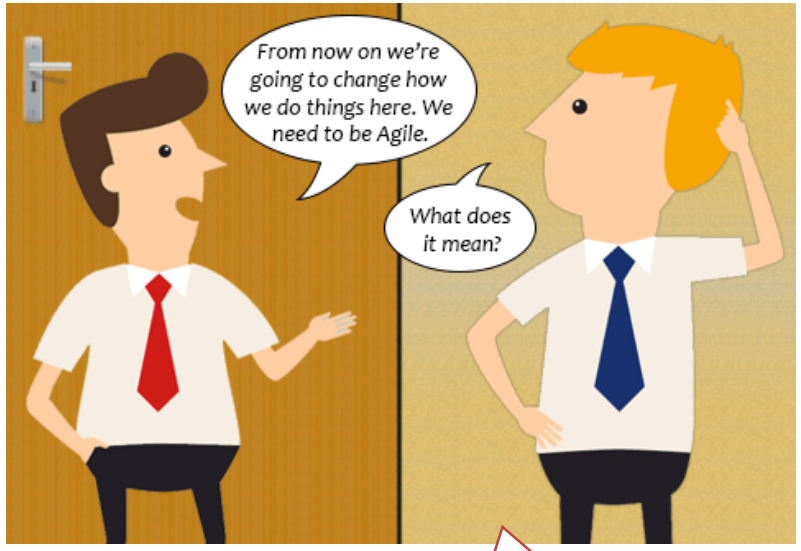
Agility Principles

- **Highest priority** is to **satisfy** the **customer** through early & **continuous delivery** of **software**
- **Welcome changing** requirements
- **Deliver** working **software frequently**
- **Businesspeople** & **developers** must **work together**
- **Build** projects **around motivated** individuals
- Emphasize **face-to-face conversation**
- **Working software** is the **measure** of **progress**
- Continuous **attention** to **technical excellence** and **good design**
- **Simplicity** – the art of maximizing the amount of work done
- The best designs emerge from **self-organizing teams**
- The **team tunes** and **adjusts** its **behaviour** to become more effective

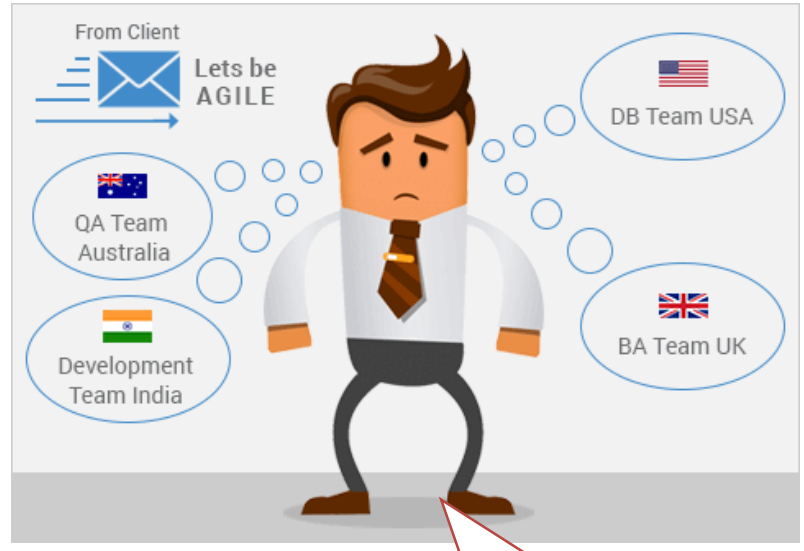
Where agile methodology not work



Project plan & requirements are clear & unlikely to change



Unclear understanding of Agile Approach among Teams



Big Enterprises where team collaboration is tough

Agile development projects usually deploy pair programming.

In pair programming, two programmers work together at one work station. One types in code while the other reviews the code as it is typed in. The two programmers switch their roles every hour or so.

Organizations Using : Amazon, Netflix, Hotstar etc..

Agile is implemented by any of the following frameworks :

Next Lecture

1. SCRUM
2. KANBAN
3. LEAN
4. eXtreme Programming
5. CRYSTAL

Thank You

Unit 1
Software Engineering
Lecture 5

Agile development projects usually deploy pair programming.

In pair programming, two programmers work together at one work station. One types in code while the other reviews the code as it is typed in. The two programmers switch their roles every hour or so.

Agile is implemented by any of the following frameworks :

1. SCRUM
2. KANBAN
3. LEAN
4. eXtreme Programming
5. CRYSTAL

What is Scrum?

Scrum is an agile process model which is used for **developing** the **complex software** systems.

A scrum is a method of restarting play in rugby that involves players packing closely together with their heads down and attempting to gain possession of the ball.

Systematic Customer Resolution Unraveling Meeting.

It is a **lightweight process framework**.

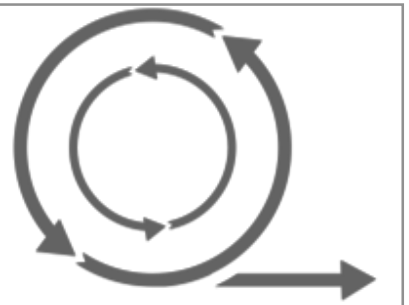
Lightweight means the **overhead of the process is kept as small** as possible in order to maximize the productivity.

Product Backlog	Product Owner
	

Product

Daily Scrum Meeting

Sprint

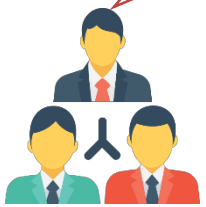


Scrum framework at a glance

Inputs from Customers, Team, Managers

Product Owner

Team Selects starting at top as much as it can commit to deliver by end of sprint

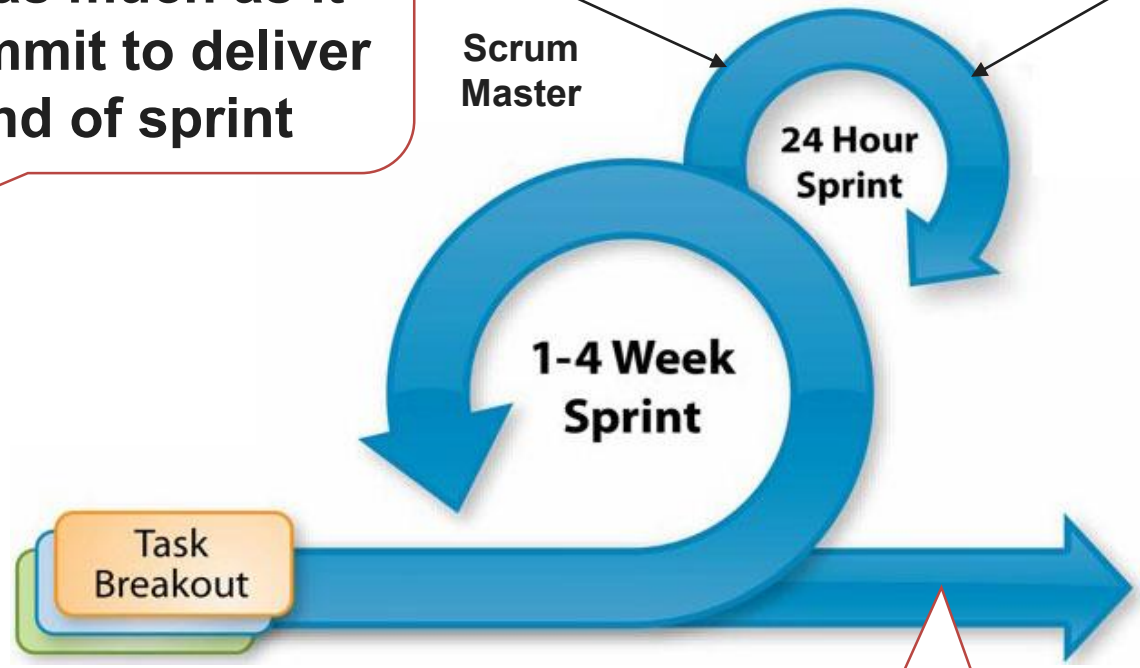


Product Backlog

Sprint Planning Meeting

Scrum Master

Daily Scrum Meetings



Sprint Backlog

Sprint Review

Finished Work

Sprint Retrospective

Prioritized list of what is required: features, bugs to fix...

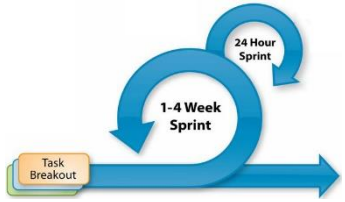
Sprint end date and team deliverable do not change

Scrum cont.

Backlog

- It is a **prioritized list of project requirements** or features that must be provided to the customer.
- The **items can be included** in the backlog at **any time**.
- The **product manager analyses** this **list** and **updates** the **priorities** as per the requirements.

Sprint



- These are the **work units** that are needed **to achieve** the requirements mentioned in the backlogs.
- Typically the sprints have **fixed duration** or time box (of **2 to 4 weeks,**
- **Change** are **not introduced** during the **sprint**.
- Thus sprints allow the team **members** to **work** in **stable** and **short-term environment**

Scrum cont.

- There are **15 minutes daily meetings** to **report** the **completed** activities, **obstacles** and **plan** for **next** activities.
- Following are three questions that are mainly discussed during the meetings.
  1. **What** are the **tasks done** since **last meeting ?**
  2. **What** are the **issues** that team is **facing ?**
  3. **What** are the **next activities** that are **planned?**

- The **scrum master** leads the meeting and **analyses the response** of each team member.
- Scrum meeting **helps** the **team** to **uncover potential problems** as early as possible
- It leads to **“knowledge socialization”** & promotes **“self-organizing team structure”**

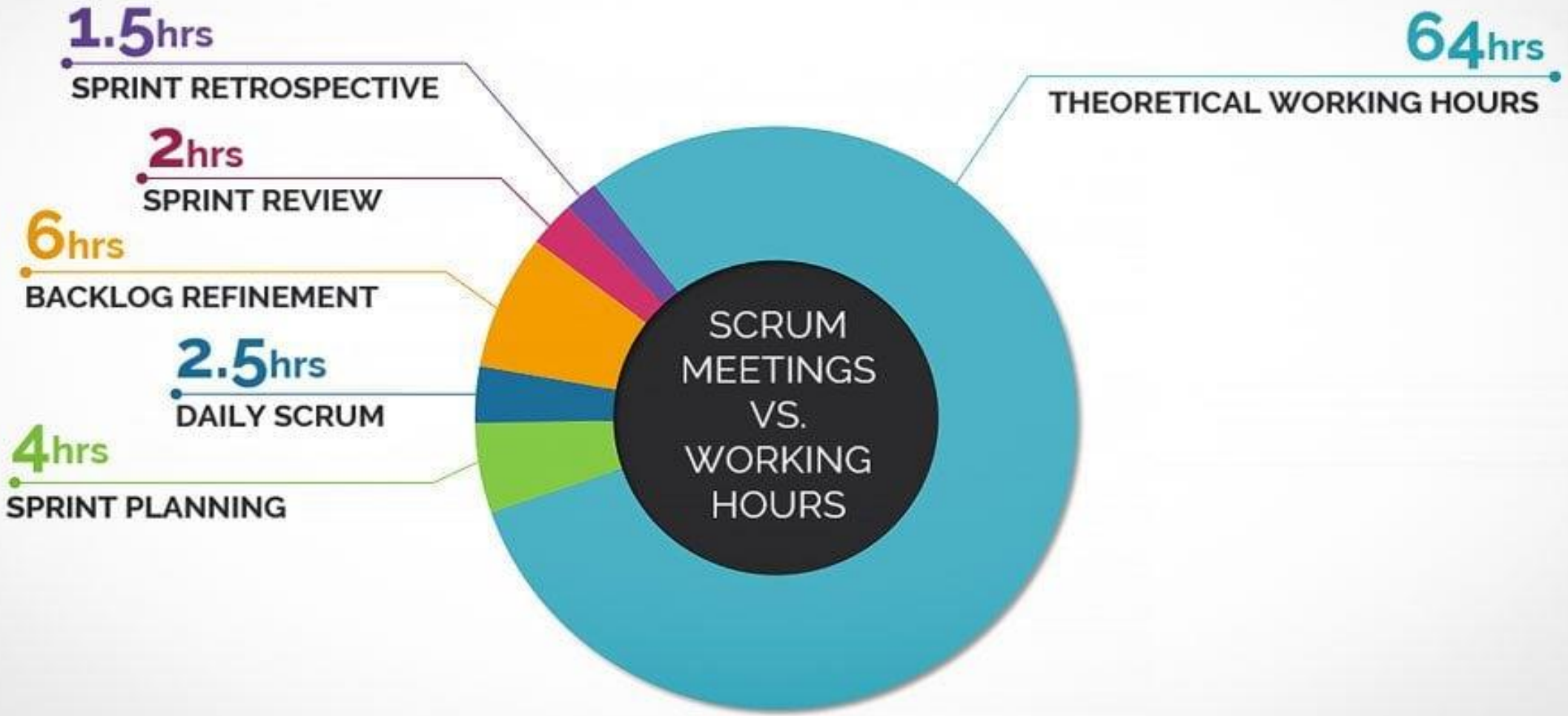
Demo

- Deliver **software increment** to customer
- Implemented functionalities are **demonstrated** to the customer

Real-world examples of companies using SCRUM

- ❑Netflix
- ❑Adobe
- ❑Vodafone
- ❑INTEL
- ❑ING
- ❑Lego
- ❑Deloitte

TOO MANY MEETINGS IN SCRUM?



WWW. VITALITYCHICAGO.COM

**Breakdown based on 2-week sprint and 40-hour work week*

GREVIENCES

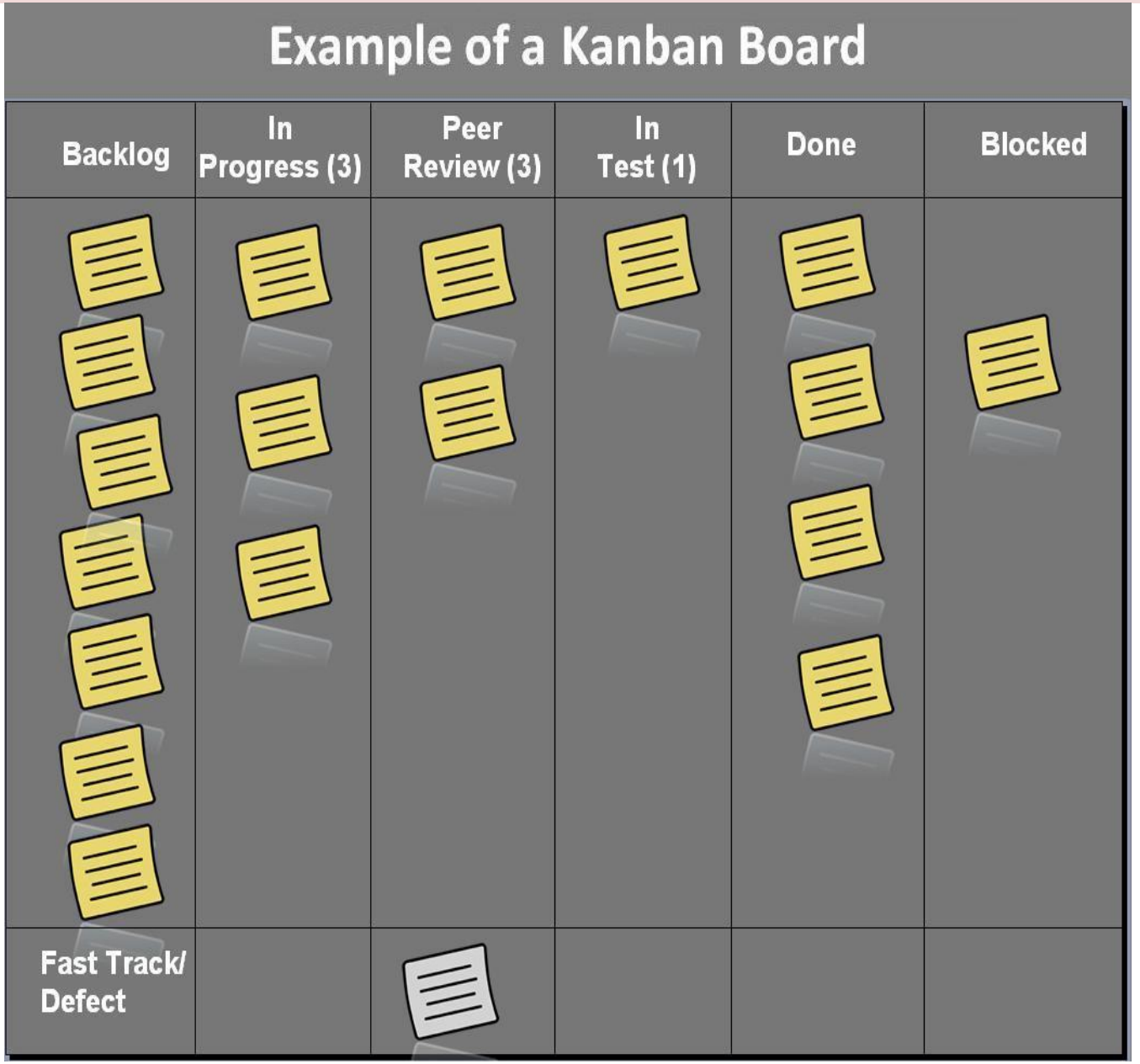

I'M CAROL & I 'M 100% ASSIGNED TO THIS PROJECT.
I'M ALEX & I'M 40% ASSIGNED TO THIS PROJECT.
I'M LIZ & I'M 35% ASSIGNED TO THIS PROJECT.
I'M KENT & I'M 50% ASSIGNED TO THIS PROJECT.
I'M VICTOR & I'M 45% ASSIGNED TO THIS PROJECT.
I'M JANE & I 'M 50% ASSIGNED TO THIS PROJECT.
I'M MAX & I 'M 60% ASSIGNED TO THIS PROJECT.
I'M JIM & I'M 25% ASSIGNED TO THIS PROJECT.

What is KANBAN?

Kanban is a framework that falls under the Agile methodology. It was developed in the late 1940s by a Japanese engineer named Taiichi Ohno.

It's a Japanese word meaning billboard firstly adopted by Toyota.

Agile Kanban Framework focuses on visualizing the entire project on boards in order to increase project transparency and collaboration between team members.

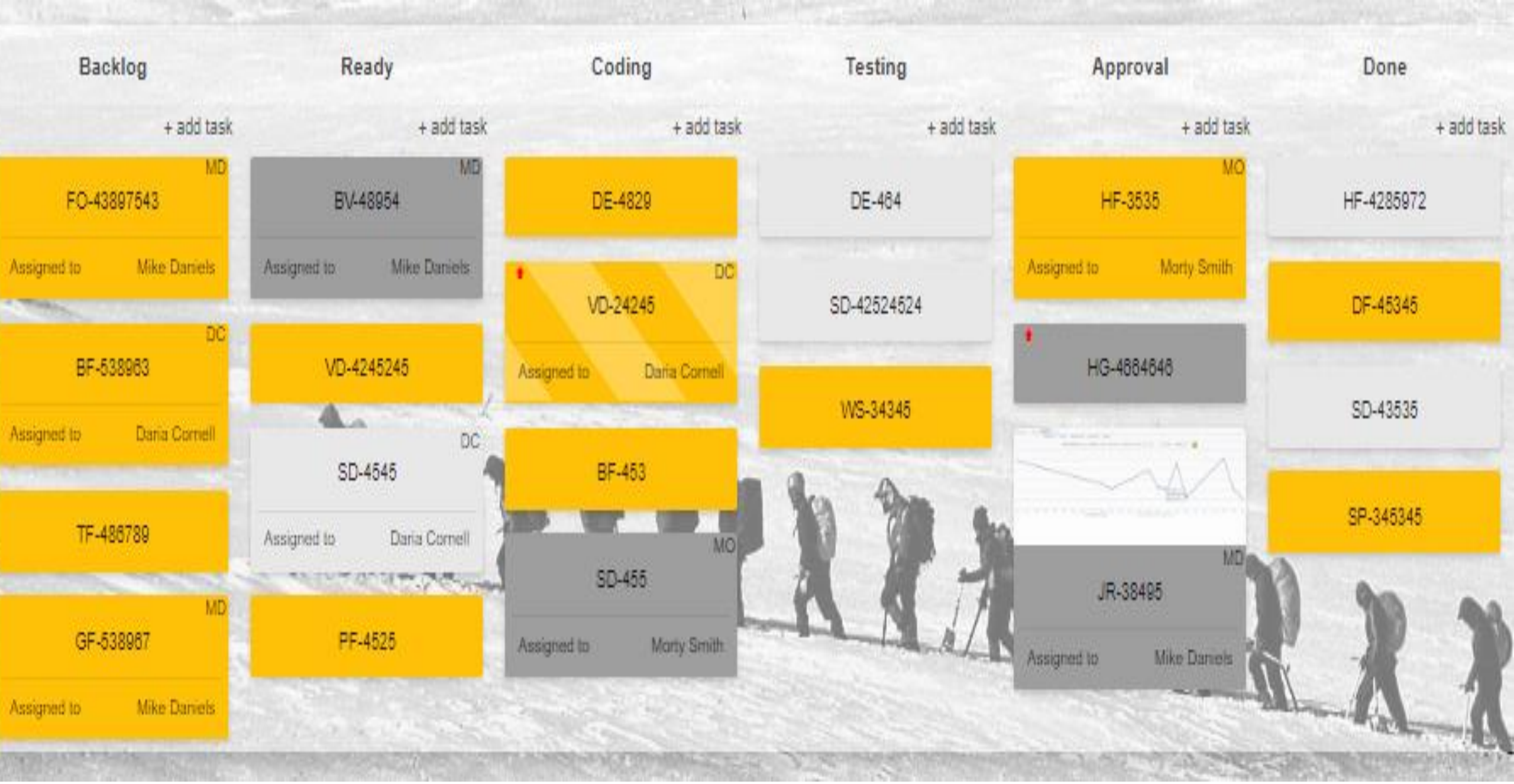


How does Kanban work?

1. **Kanban method revolves around the kanban board.**
2. **It is a tool that visualizes the entire project to track the flow of their project.**
3. **Through this graphical approach of Kanban boards, a new member or an external entity can understand what's happening right now, tasks completed and future tasks.**

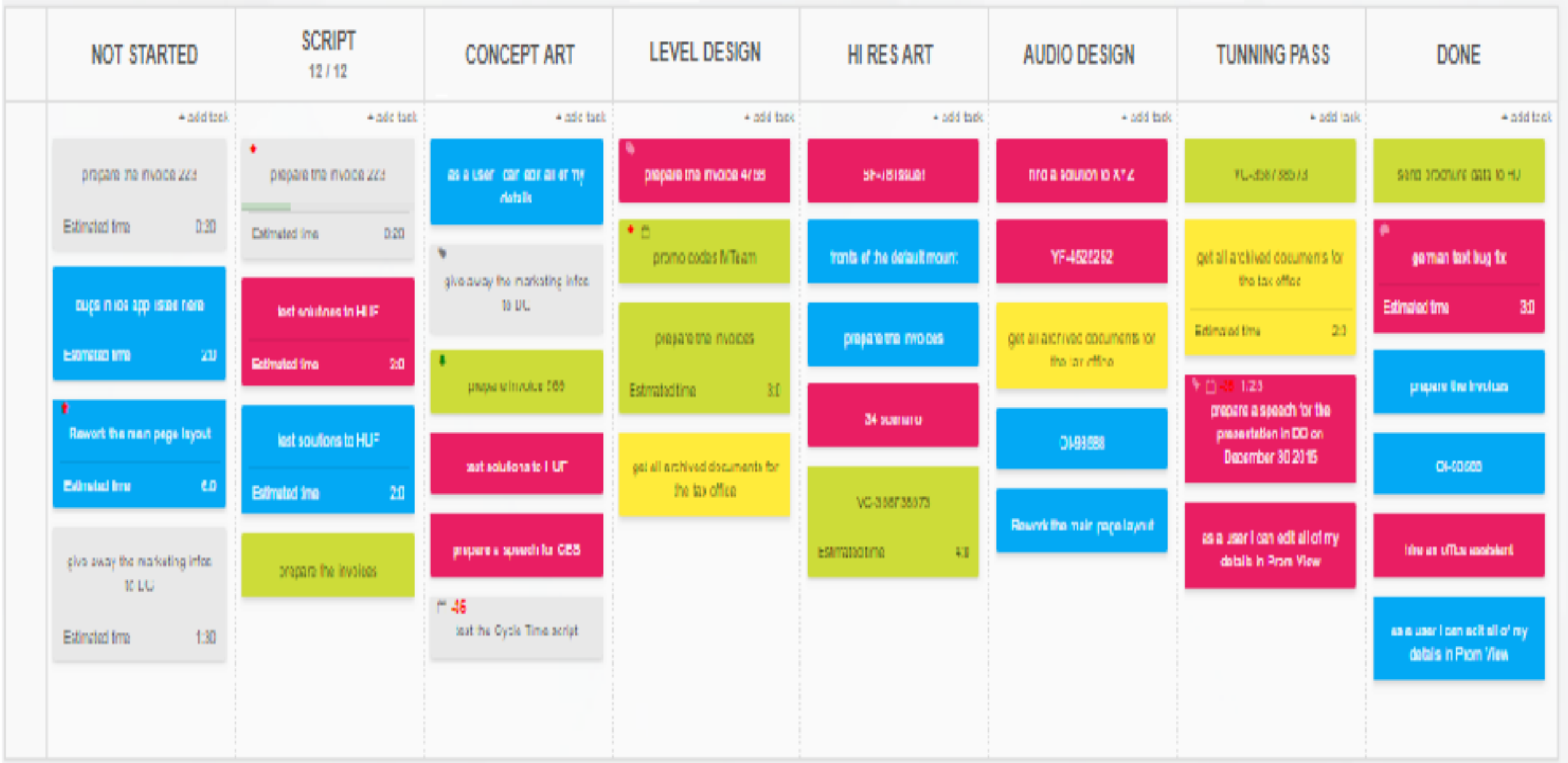
Kanban board indicates

- ❑ **the current tasks that are being performed**
- ❑ **the tasks to do in the future**
- ❑ **the tasks that are completed**



Advanced Kanban board for software development processes

Heijunka board : special kind of a Kanban board often used by game developers



8 real-world examples of companies using Kanban

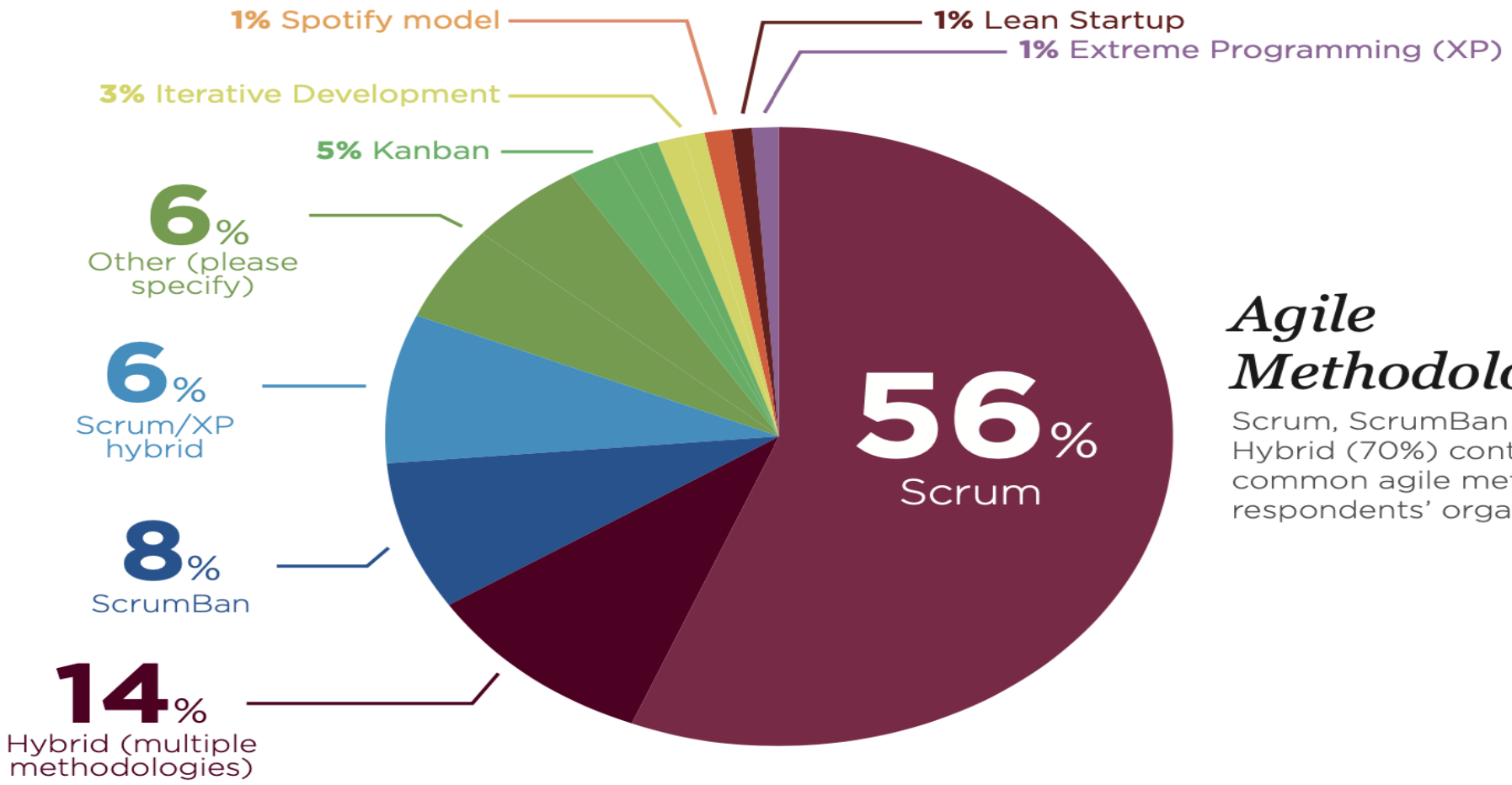
- ❑Nike’s use of Kanban
- ❑Jaguar’s use of Kanban to improve efficiency
- ❑Pixar’s use of Kanban to ‘manufacture’ animation
- ❑NextPhase Medical Devices’ Kanban-based warehousing
- ❑Spotify’s Kanban project planning
- ❑Zara’s fashion Kanban workflow
- ❑Seattle Children’s Hospital: solving supply issues with Kanban
- ❑Dynisco’s inventory reduction using Kanban

Key reasons for adopting KANBAN in SE

- Accelerates software delivery.
- Helps manage changing requirements and priorities. Kanban welcomes changing requirements, even late in development and improves scope control because customers and stakeholders can add new requirements, shift priorities, or rethink requirements.
- Pays attention to software processes to achieve higher quality code.
- Enhance delivery predictability. Effective software development teams can accurately predict their delivery in time, scope, and total quality while continuously finding ways to improve their productivity with Kanban.
- Improves project visibility. Now everyone know what needs to be done, what is in progress and what was done.
- Significantly reduces project risk.
- Boosts the team happiness. Kanban is an investment in a happiness work culture.
- Reduces project total cost.
- Helps to manage distributed teams more efficiently.

the 12th annual STATE OF AGILE™ REPORT

AGILE METHODS AND PRACTICES



Agile Methodologies Used

Scrum, ScrumBan and Scrum/XP Hybrid (70%) continue to be the most common agile methodologies used by respondents' organizations.

Thank You

Unit 1
Software Engineering
Lecture 6

Requirement Engineering

- Discovering
- Documenting

] Requirements

Divisions of Requirements Types

- User
- System
- Functional
- Non Functional

Requirement Engg Activities

- Elicitation
- Analysis
- Validation
- Relationships

Requirements analysis is hard

"The **hardest** single **part** of building a software system is **deciding what to build**. No part of the work so cripples the resulting system if done wrong."

"Fred" Brooks Jr. is an American computer architect, software engineer, and computer scientist, best known for managing the development of IBM's System/360 family of computers

"The **seeds** of major software **disasters** are usually **sown** in the **first three months** of commencing the software project."

Capers Jones is an American specialist in software engineering methodologies

What is Requirement Engineering?

Tasks and **techniques** that lead to an **understanding** of **requirements** is called **requirement engineering**.

If a company wishes to let a contract for a large software development project, it must define its needs in a sufficiently abstract way that a solution is not predefined.

The requirements must be written so that several contractors can bid for the contract, offering, perhaps, different ways of meeting the client organization's needs.

Once a contract has been awarded, the contractor must write a system definition for the client in more detail so that the client understands and can validate what the software will do.

Both of these documents called the requirements document for the system.

User requirements **to mean the high-level abstract requirements**

System requirements **to mean the detailed description of what the system should do.**

Refer the following pdf for examples: user_sys_req.pdf

User Requirements

- natural language statements
- diagrams
- services expected by the system
- operational constraints
- precise descriptions of the system functionality

System Requirements

- detailed descriptions of system functionality
- describes operations, functions and services in compliance to the constraints
- defines exactly what is to be implemented

system Requirement is a contract between the buyer and the developer.

User requirements definition

1. The Mentcare system shall generate monthly management reports showing the cost of drugs prescribed by each clinic during that month.

System requirements specification

1.1 On the last working day of each month, a summary of the drugs prescribed, their cost and the prescribing clinics shall be generated.
1.2 The system shall generate the report for printing after 17.30 on the last working day of the month.
1.3 A report shall be created for each clinic and shall list the individual drug names, the total number of prescriptions, the number of doses prescribed and the total cost of the prescribed drugs.
1.4 If drugs are available in different dose units (e.g. 10mg, 20mg, etc.) separate reports shall be created for each dose unit.
1.5 Access to drug cost reports shall be restricted to authorized users as listed on a management access control list.

Feasibility studies

A feasibility study is a short, focused study that should take place early in the RE process. It should answer three key questions:

(1)Does the system contribute to the overall objectives of the organization?

(2)Can the system be implemented within schedule and budget using current technology?

(3) Can the system be integrated with other systems that are used?

If the answer to any of these questions is no, you should probably not go ahead with the project.

Requirement Engineering

It provides the appropriate mechanism for understanding

What customer wants

Analyzing needs | Assessing feasibility

Negotiating a reasonable solution

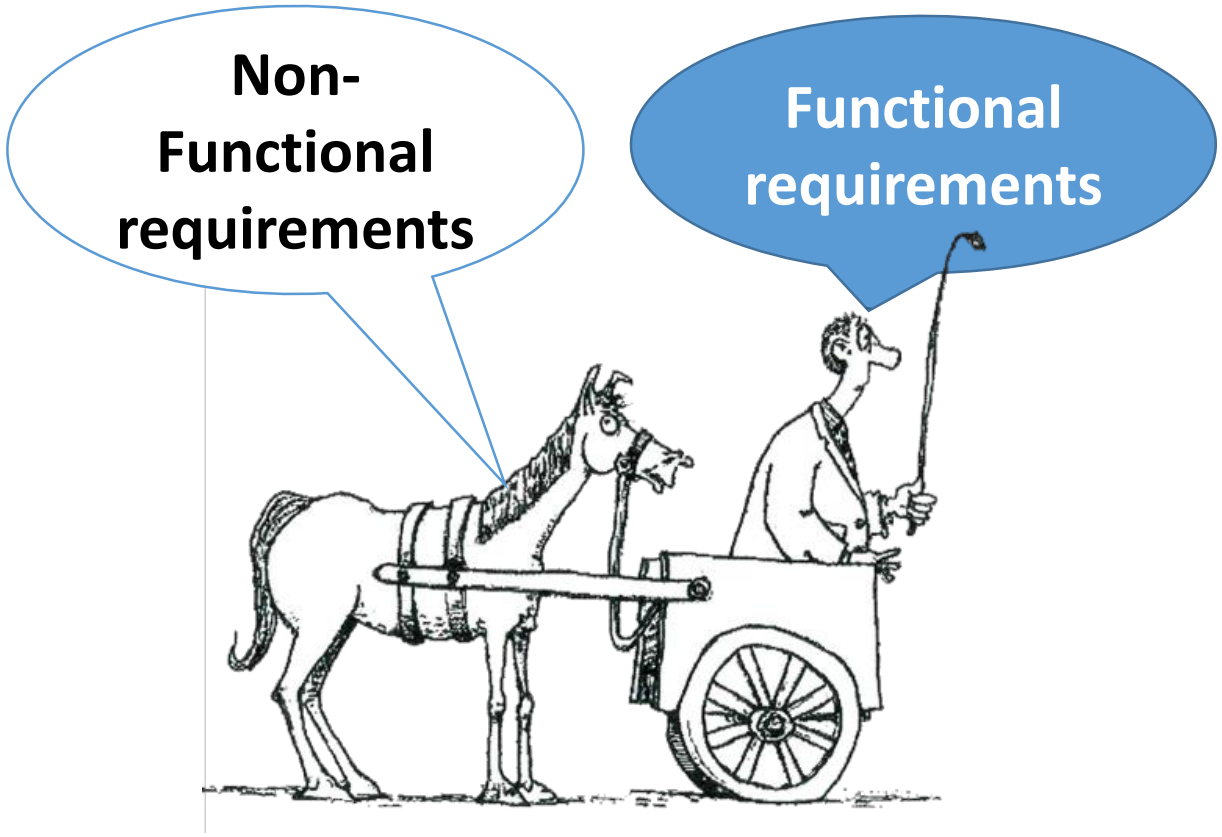
Specifying solution unambiguously

Validating the specification

Managing requirements

Requirements fall into two types

Functional requirements	Non-Functional requirements



Don't put what you want to do - before how you need to do it

Thank You

Unit 1
Software Engineering
Lecture 7

Requirement Engineering

- Discovering
- Documenting

] Requirements

Divisions of Requirements Types

- User
- System
- Functional
- Non Functional

Requirement Engg Activities

- Elicitation
- Analysis
- Validation
- Relationships

Requirement Engineering

It provides the appropriate mechanism for understanding

What customer wants

Analyzing needs | Assessing feasibility

Negotiating a reasonable solution

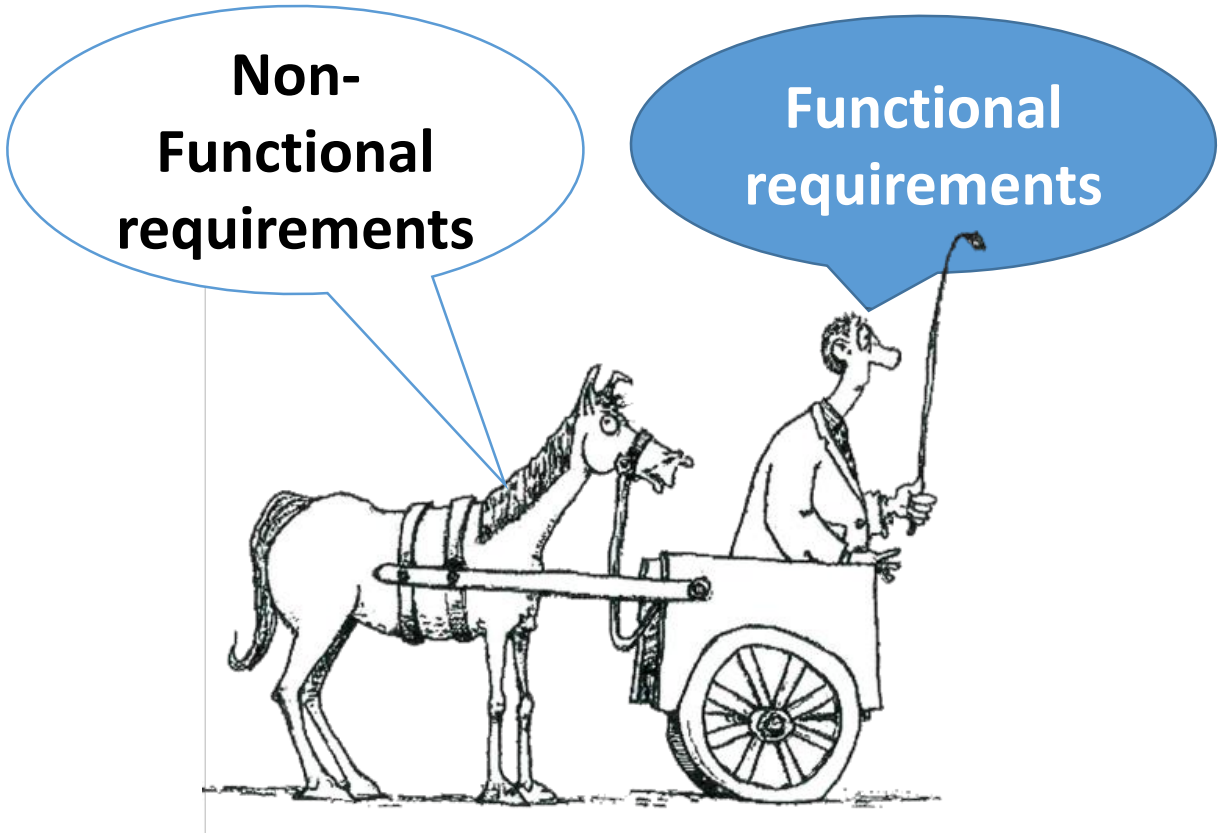
Specifying solution unambiguously

Validating the specification

Managing requirements

Requirements fall into two types

Functional requirements	Non-Functional requirements



Don't put what you want to do - before how you need to do it

Functional requirements are product features or functions that developers must implement to enable users to accomplish their tasks. It's important to make them clear both for the development team and the stakeholders. For example:

- *The system sends an approval request after the user enters personal information.*
- *A search feature allows a user to hunt among various invoices if they want to credit an issued invoice.*
- *The system sends a confirmation email when a new user account is created.*

Nonfunctional requirements, not related to the system functionality, rather define how the system should perform. Some examples are:

- *The website pages should load in 3 seconds with the total number of simultaneous users <5 thousand.*
- *The system should be able to handle 20 million users without performance deterioration.*

Functional requirements

Any requirement which specifies what the system Should do.

A functional requirement will describe a particular behavior of function of the system when certain conditions are met, for example: “Send email when a new customer signs up” or “Open a new account”.

Typical functional requirements

- Business Rules
- Administrative functions
- Historical Data
- Transaction corrections, adjustments and cancellations
- Legal or Regulatory Requirements
- External Interfaces
- Reporting Requirements
- Authentication
- Authorization levels
- Audit Tracking

Non-functional requirements

Any requirement which specifies how the system performs a certain function.

A non-functional requirement will describe how a system should behave and what limits there are on its functionality.

Typical non-functional requirements

Response time	Availability	Regulatory
Throughput	Reliability	Manageability
Utilization	Recoverability	Environmental
Static Volumetric	Maintainability	Data Integrity
Scalability	Serviceability	Usability
Capacity	Security	Interoperability

Library Management System

Function Requirements

- **Add Article:** New entries must be entered in database
- **Update Article:** Any changes in articles should be updated in case of update
- **Delete Article:** Wrong entry must be removed from system
- **Inquiry Members:** Inquiry all current enrolled members to view their details
- **Inquiry Issuance:** Inquiry all database articles
- **Check out Article:** To issue any article must be checked out
- **Check In article:** After receiving any article system will reenter article by Checking
- **Inquiry waiting for approvals:** Librarian will generate all newly application which is in waiting list
- **Reserve Article:** This use case is used to reserve any book with the name of librarian, it can be pledged

Non-function Requirements

- **Safety Requirements:** The database may get crashed at any certain time due to virus or operating system failure. So, it is required to take the database backup.
- **Security Requirements:** We are going to develop a secured database for the university. There are different categories of users namely teaching staff, administrator, library staff ,students etc., Depending upon the category of user the access rights are decided.
- **Software Constraints:** The development of the system will be constrained by the availability of required software such as database and development tools. The availability of these tools will be governed by

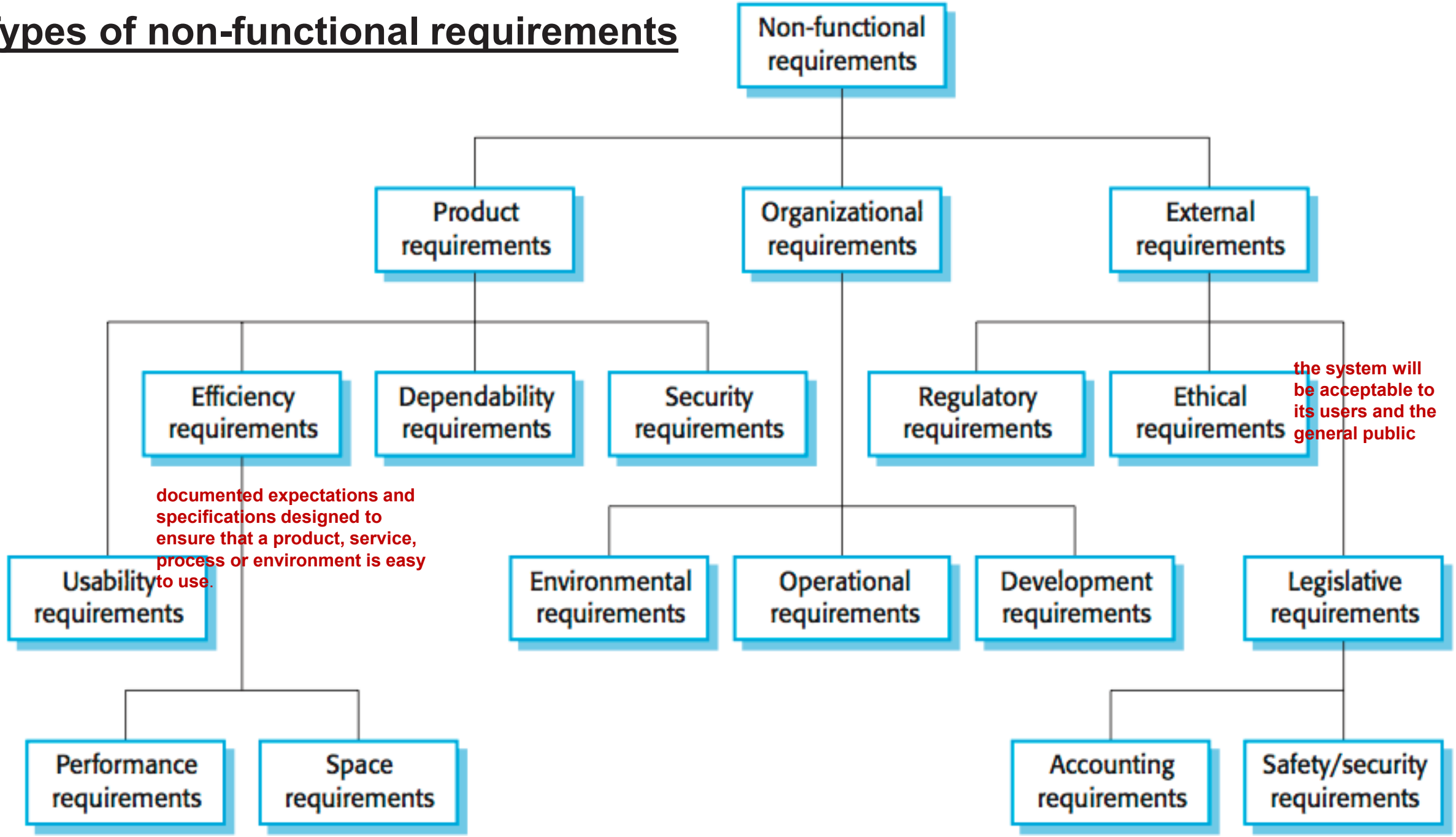
Thank You

Unit 1
Software Engineering
Lecture 8

1. The **functional requirements** for a system describe what the system should do.
2. These requirements depend on the type of software being developed, the expected users of the software, and the general approach taken by the organization when writing requirements.
3. When expressed as user requirements, functional requirements should be written in natural language so that system users and managers can understand them.
4. Functional system requirements expand the user requirements and are written for system developers. They
5. should describe the system functions, their inputs and outputs, and exceptions in detail.

1. **Non-functional requirements** are requirements that are not directly concerned with the specific services delivered by the system to its users.
2. The non-functional requirements usually specify or constrain characteristics of the system as a whole. They may relate to emergent system properties such as reliability, response time, and memory use. Alternatively, they may define constraints on the system implementation, such as the capabilities of I/O devices or the data representations
3. used in interfaces with other systems.
4. Non-functional requirements are often more critical than individual functional requirements.
5. Failing to meet a non-functional requirement can mean that the whole system is unusable.

Types of non-functional requirements



Requirements Engineering Tasks

These are discovering requirements by interacting with stakeholders (elicitation and analysis);
converting these requirements into a standard form (specification);
checking that the requirements actually define the system that the customer wants (validation).

Requirements engineering encompasses seven distinct tasks: inception, elicitation, elaboration, negotiation, specification, validation, and management.

Stakeholder

A *stakeholder* as "anyone who benefits in a direct or indirect way from the system which is being developed."

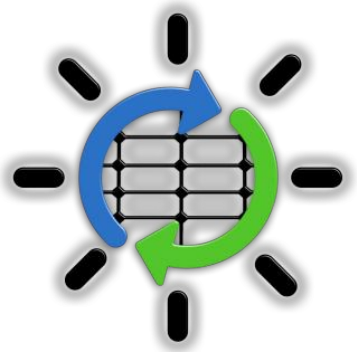
These can be:

Business operations managers, product managers, marketing people, internal and external customers, end users, consultants, product engineers, software engineers, support and maintenance engineers.

Each stakeholder has a different view of the system, achieves different benefits when the system is successfully developed, and is open to different risks if the development effort should fail.

Requirements Engineering Tasks

1 Inception



Roughly define scope

A basic understanding of a problem, people who want a solution, the nature of solution desired , effective preliminary communication and collaboration between the other stakeholders and the software team.

2 Elicitation (Requirement Gathering)

Define requirements

The practice of collecting the requirements of a system from users, customers and other stakeholders. Eg. distributing a questionnaire that elicits a list of questions related to the requirements/needs.

3 Elaboration

Further define requirements

Expand and refine requirements obtained from inception & elicitation

Creation of User scenarios, extract analysis class and business domain entities

11 Requirements Gathering Techniques for Agile Product Teams

- Interviews
- Questionnaires or Surveys
- User Observation
- Document Analysis
- Interface Analysis
- Workshops
- Brainstorming
- Role-Play
- Use Cases and Scenarios
- Focus Groups
- Prototyping

A good idea to take a multi-faceted approach to requirements gathering

A number of problems that are encountered as elicitation occurs

- Problems of scope occur when the boundary of the system is ill-defined or the customers and users specify unnecessary technical detail that may confuse, rather than clarify, overall system objectives.

- Problems of understanding are encountered when customers and users are not completely sure of what is needed, have a poor understanding of the capabilities and limitations of their computing environment, don't have a full understanding of the problem domain, have trouble communicating needs, omit information that is believed to be "obvious," specify requirements that conflict with the needs of other customers and users, or specify requirements that are ambiguous or untestable.

- Problems of volatility occur when the requirements change over time.

We must approach the requirements-gathering activity in an organized manner.

- Elaboration is driven by the creation and refinement of user scenarios that describe how the end user will interact with the system.

- Each user scenario is parsed to extract analysis i.e. classes—business domain entities that are visible to the end user.

- The attributes of each analysis class are defined, and the services that are required by each class are identified.

- The relationships and collaboration between classes are identified, and a variety of supplementary diagrams are produced.

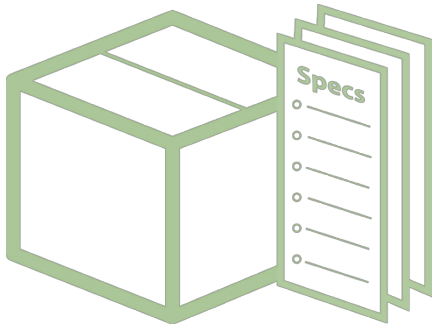
Requirements Engineering Tasks Cont.

4 Negotiation

Reconcile conflicts

Agree on a deliverable **system** that is **realistic for developers** and **customers.** Customers, users, and other stakeholders rank the needs and then discuss conflicts in priority.

5 Specification



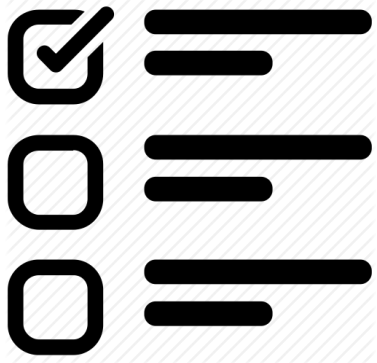
Create analysis model

It may be **written document**, set of **graphical models**, formal **mathematical model**, collection of **user scenarios**, **prototype** or collection of all these.

SRS (Software Requirement Specification) is a document that is created when a detailed description of all aspects of software to build must be specified before starting of project.

Requirements Engineering Tasks Cont.

6 Validation



Ensure quality of requirements

Review the requirements specification for errors, ambiguities, omissions (absence) and conflicts.

The technical review team that validates requirements includes software engineers, customers, users, and other stakeholders

7 Requirements Management

It is a set of activities to identify, control & trace requirements & changes to requirements (Umbrella Activities) at any time as the project proceeds.

Requirements Validation Checklist

It is often useful to examine each requirement against a set of checklist questions. Here is a small subset of those that might be asked:

- **Are requirements stated clearly? Can they be misinterpreted?**
- **Is the source (e.g., a person, a regulation, a document) of the requirement identified? Has the final statement of the requirement been examined by or against the original source?**
- **Is the requirement bounded in quantitative terms?**
- **What other requirements relate to this requirement? Are they clearly noted via a cross-reference matrix or other mechanism?**
- **Does the requirement violate any system domain constraints?**
- **Is the requirement testable? If so, can we specify tests (sometimes called validation criteria) to exercise the requirement?**
- **Is the requirement traceable to any system model that has been created?**
- **Is the requirement traceable to overall system/ product objectives?**
- **Is the specification structured in a way that leads to easy understanding, easy reference, and easy translation into more technical work products?**
- **Has an index for the specification been created?**
- **Have requirements associated with performance, behavior, and operational characteristics been clearly stated? What requirements appear to be implicit?**

Elicitation is the Hardest Part!

- **Problems of scope**
  - System boundaries are ill-defined
  - Customers will provide irrelevant information
- **Problems of understanding**
  - Customers never know exactly what they want
  - Customers don't understand capabilities and limitations
  - Customers have trouble fully communicating needs
- **Problems of volatility**
  - Requirements always change

Project Inception

During the **initial project meetings**, the following tasks should be accomplished

- **Identify the project stakeholders**
  - These are the folks we should be talking to
- **Recognize multiple viewpoints**
  - Stakeholders may have different (and conflicting) requirements
- **Work toward collaboration**
  - It's all about reconciling conflict
- **Ask the first questions**
  - Who? What are the benefits? Another source?
  - What is the problem? What defines success? Other constraints?
  - Am I doing my job right?

Collaborative Elicitation

One-on-one Q&A sessions rarely succeed in practice; collaborative strategies are more practical

The job of a requirement engineer is to identify areas of commonality (i.e., requirements on which all stakeholders agree) and areas of conflict or inconsistency.

To resolve conflicting requirements and understand the relative importance of all requirements is to use a "voting" scheme based on priority points.

Elicitation work products

Collaborative elicitation should result in several work products

A bounded statement of scope

A list of stakeholders

A description of the technical environment

A list of requirements and constraints

Any prototypes developed

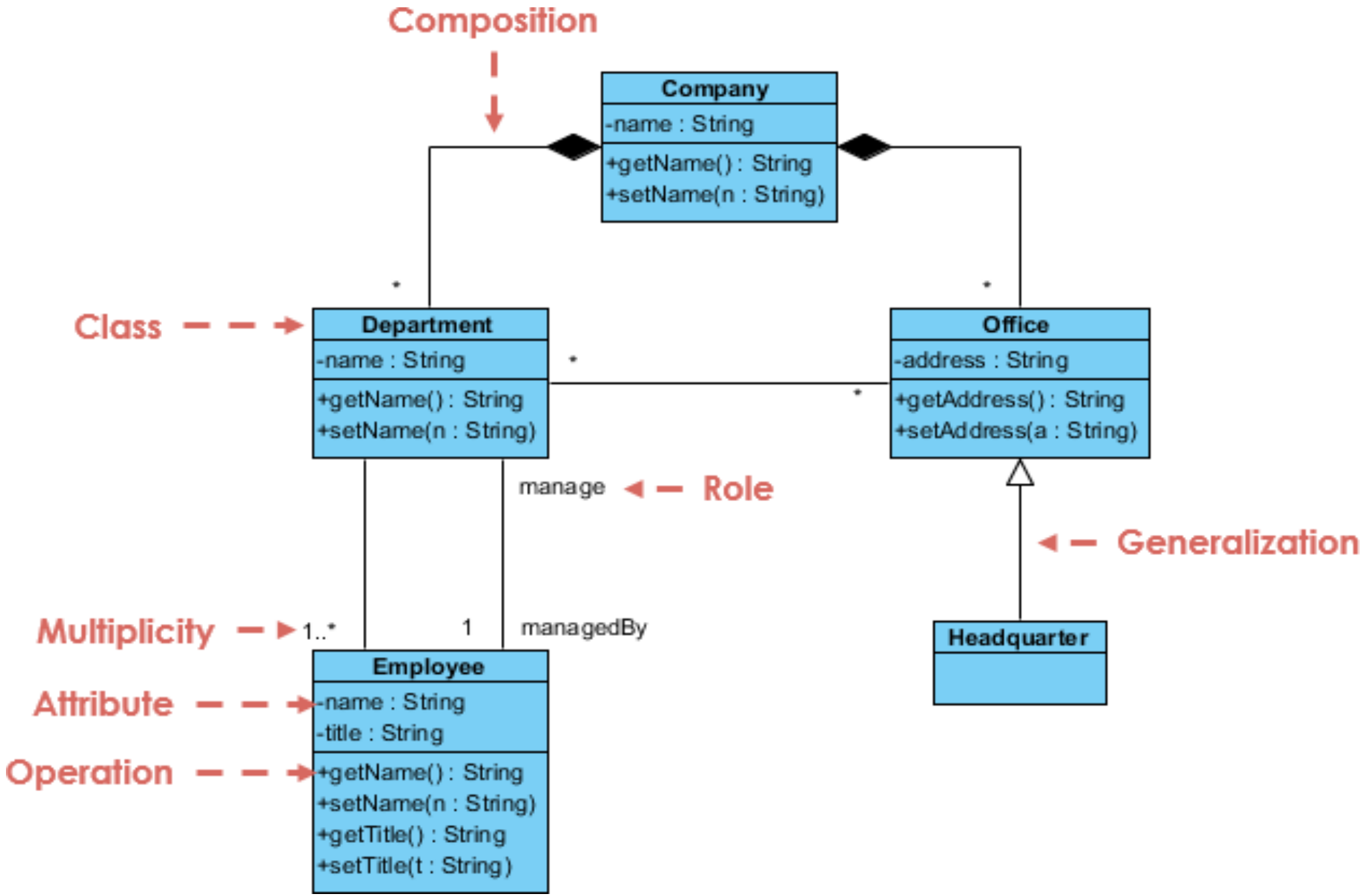
A set of use cases

- Characterize how users will interact with the system
- Use cases tie functional requirements together

Thank You

Unit 2
Software Engineering
Lecture 9

- Class Diagram



A company consists of departments. Departments are located in one or more offices. One office acts as a headquarter. Each department has a manager who is recruited from the set of employees. Your task is to model the system for the company.

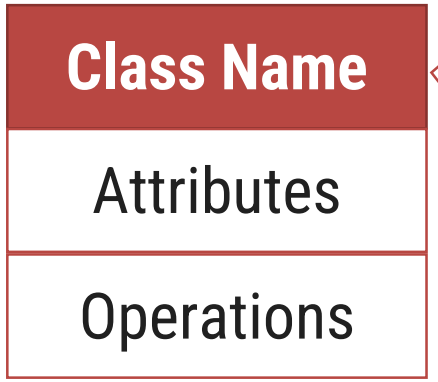
Class diagram

The purpose of class modeling is to describe objects in systems and different types of relationships between them.

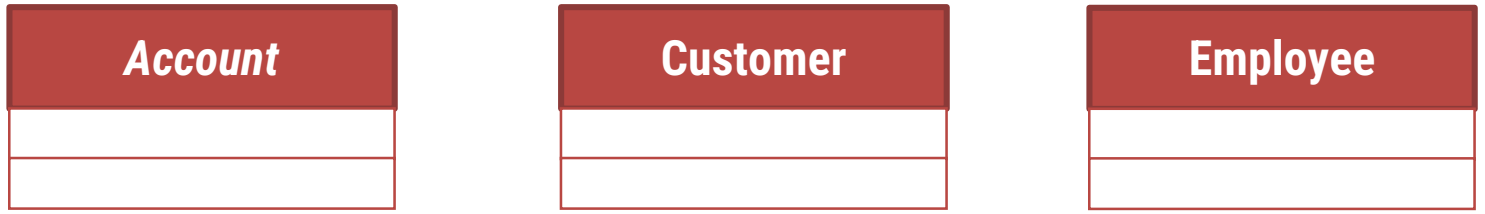
The class diagram is used to construct and visualize object-oriented systems.

- Class modeling is used to specify the structure of the objects, classes, or components that exist in the problem domain or system.
- Class diagram provides a graphic notation for modeling classes and their relationships.
- Class is a blueprint of an object.
- An object is a concept, abstraction, or thing with an identity that has meaning for an application.
- Class diagrams represent an overview of the system like classes, attributes, operations, and relationships.
- In a given problem statement we should identify nouns that will build up classes.

Elements of Class Diagram (Class Name)



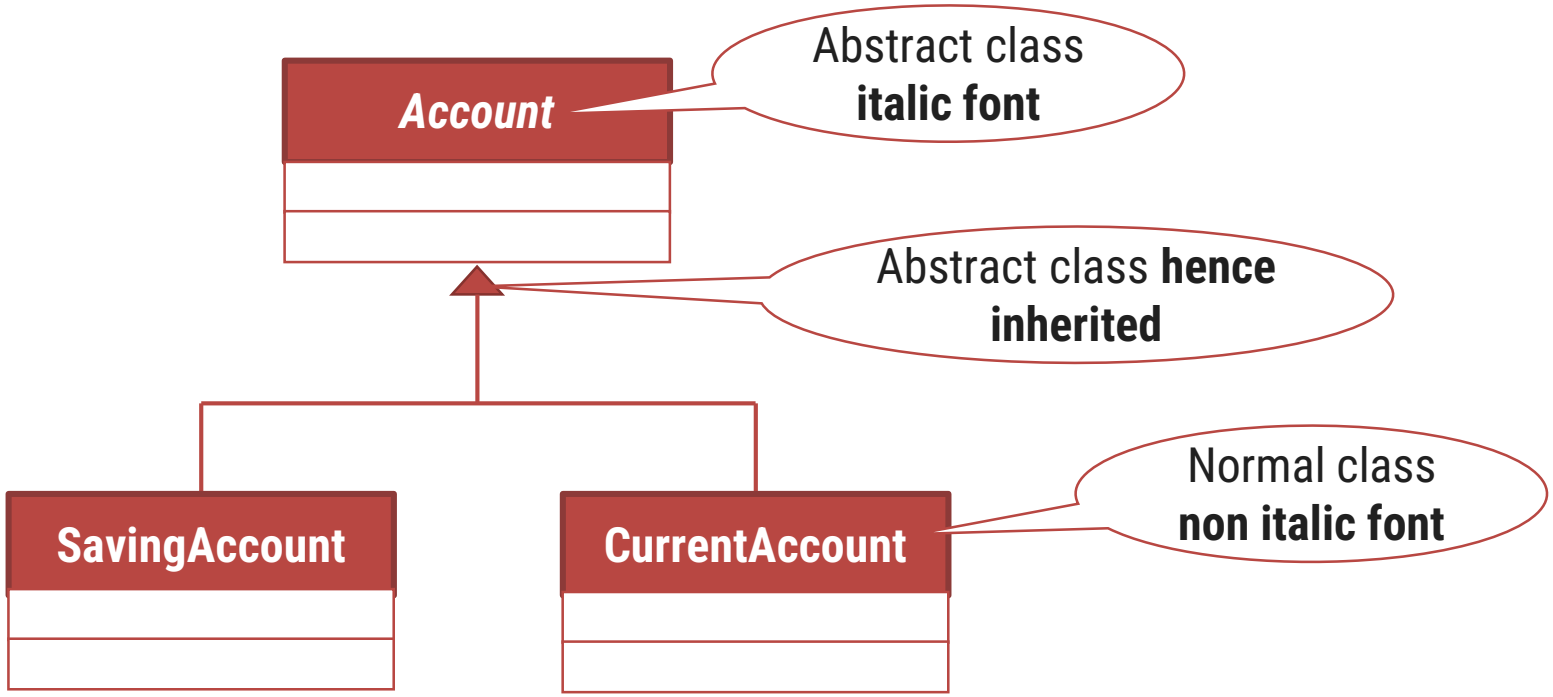
- The **name** of the class appears in the **upper section**.
- Class name should be **meaningful**.
- Class name should always be aligned **center** of the upper section.
- Class name should **start with capital letters**, and **intermediate letter is a capital.**
- Class name should be always **bold format**.
- For e.g.:



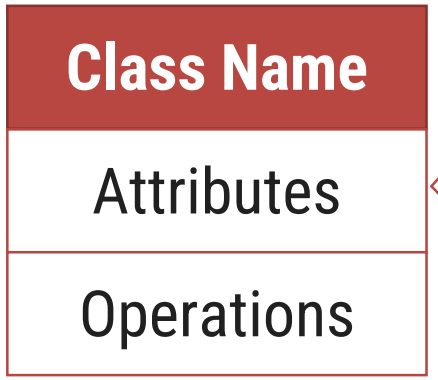
- Abstract class name should be written in **italic format**.

Elements of Class Diagram (Class Name) Cont.

- For e.g. in the banking system, there are two types of accounts; one is a saving account and another is a current account.
- Account is an abstract class and saving account and the current account is a subclass of Account.
- The system can't directly access the Account class. It is accessible by only saving accounts and current accounts.



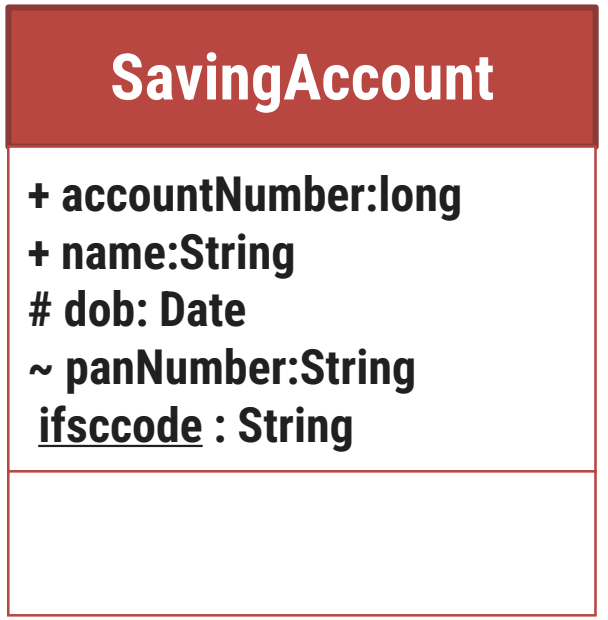
Elements of Class Diagram (Attributes)



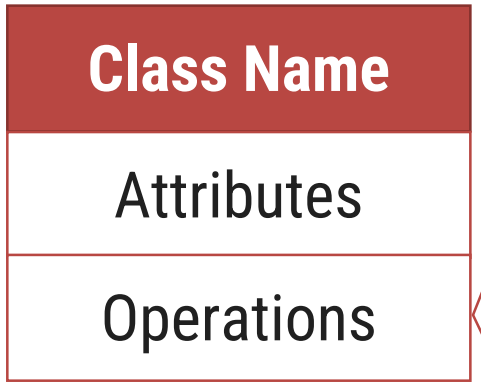
- An attribute is a named property of a class that describes a value held by each object of the class.
- The UML notation lists attributes in the second compartment of the class box.
- The attribute name should be in the regular face, left align in the box & use the lowercase letters for the first character.
- The data type for the attribute should be written after the colon.
- Accessibility of attribute must be defined using a member access modifier.
- Syntax : accessModifier attributeName:dataType=defaultValue
- For e.g. *in this example '–' represents private access modifier*

Elements of Class Diagram (Access Modifiers)

- **Public (+):** Member accessible by **all classes**, whether these classes are in the same package or in another package.
- **Private (-):** Member **cannot be accessed outside** the enclosing/declaring class.
- **Protected (#):** Member can be **accessed only by subclasses** and within a class.
- **Package (~):** Member can be accessible by all classes, **within the package**. Outside package member not accessible.
- **Static (underlined) :** Member can be **accessed** using **class name only**.
- In example you can see how to use access specifier

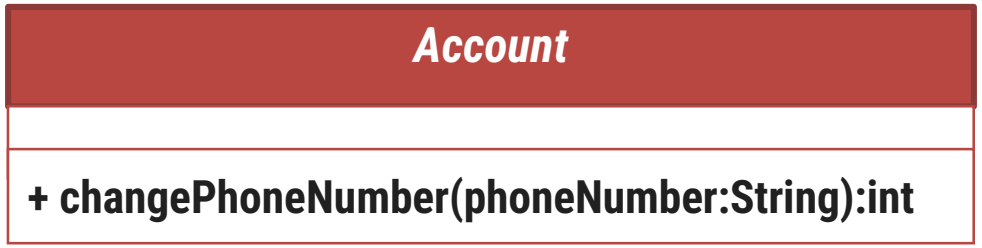


Elements of Class Diagram (Operation)



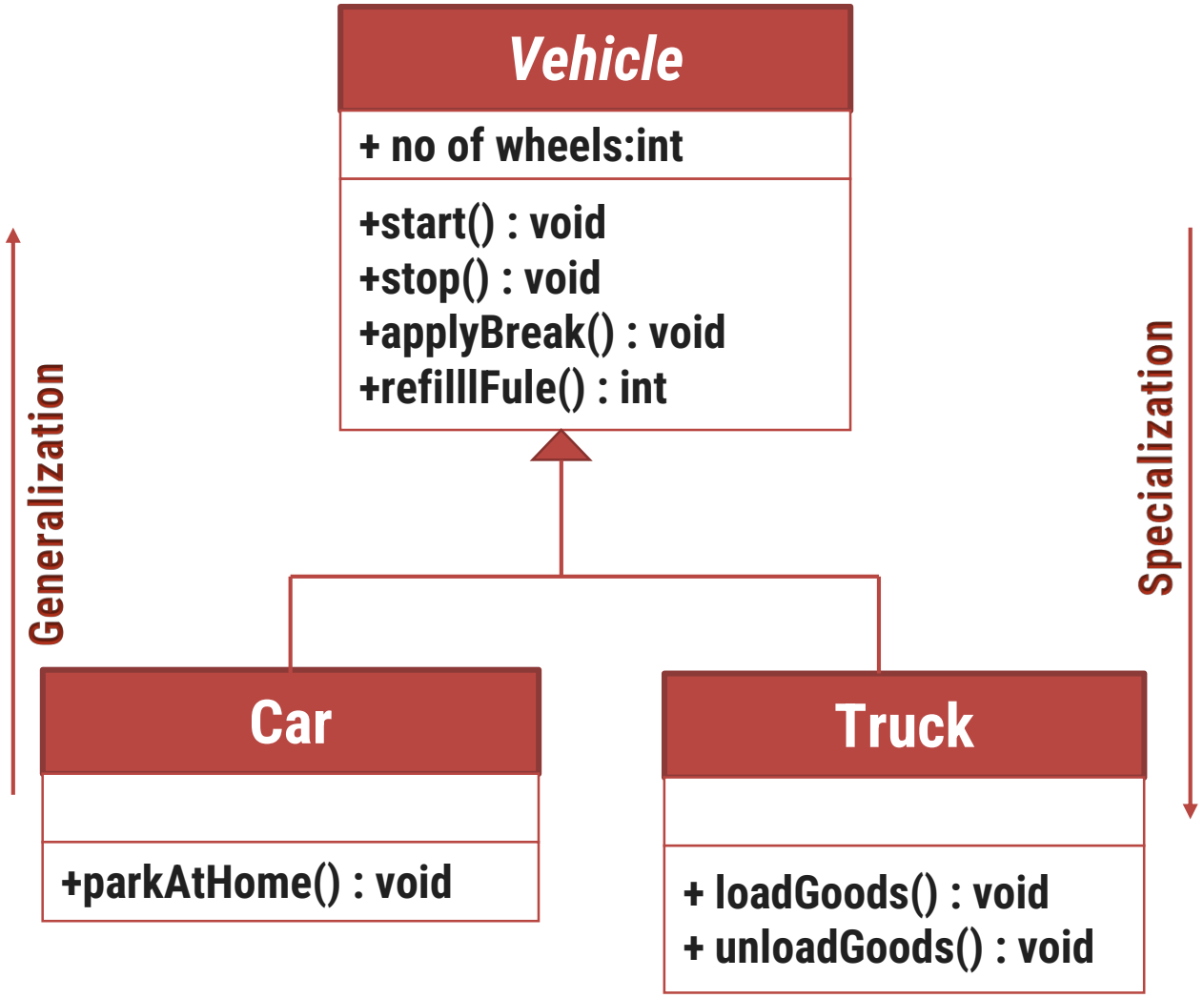
- The operation is a **function or procedure** that may be applied to objects in a class.
- The UML notation is to list operations in the **third compartment** of the class box.
- The operation name in the **regular face**, **left align** the name in the box, and use a **lowercase letter** for the **first character**.
- Optional detail, such as an argument list and result type, may follow each operation name.
- The **return type** of method should be written after colon.
- **Accessibility** of operation must be defined using a member access modifier.
- Syntax : **accessModifier methodName(argumentList):returnType**

For e.g.: you can see **change phone number** is a **method** that accepts **phone number** as an **argument** and **return** the **int value** as a response.



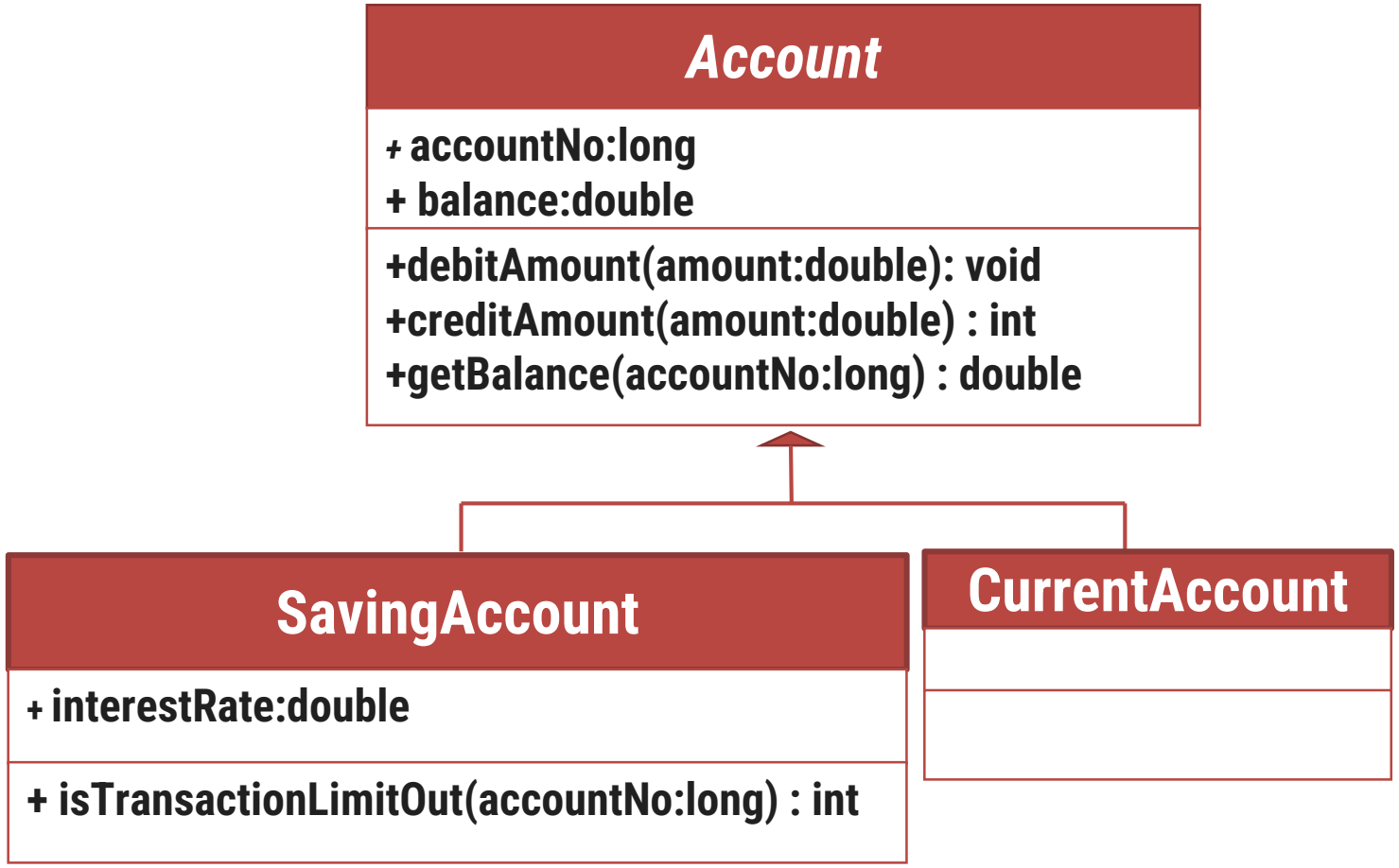
Generalization & Specialization

- Generalization is the process of extracting shared characteristics from two or more classes and combining them into a generalized superclass
- Shared characteristics can be attributes or methods.
- Represents an "is-a" relationship
- For example, a car is a vehicle, and a truck is a vehicle. In this case, vehicle is the general thing, whereas car and truck are the more specific things.
- Specialization is the reverse process of Generalization means creating new sub-classes from an existing class.



Generalization & Specialization

- For example, in a bank, any Customer opens an account.
- The account can be either a savings account or a current account. In saving account, customer earns fixed interest on the deposit. But this facility is not available in the current account.



Link and Association Concepts

- Link and associations are the means for establishing relationships among objects and classes.
- A link is a physical or conceptual connection among objects.
- An association is a description of a group of links with common structure and common semantic & it is optional.
- Aggregation and Composition are the two forms of association. It is a subset of association.
- Means they are specific cases of association.
- In both aggregation and composition object of one class "owns" object of another class, but there is a minor difference.

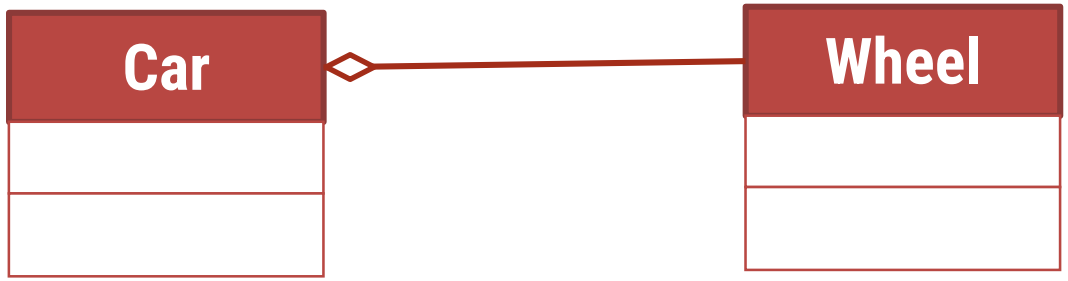
Aggregation

Aggregation is a subset of association. it is a collection of different things.
It is more specific than an association.

It represents 'has a' relationship.

Aggregation implies a relationship where the child is independent of its parent.

- For e.g.: Here we are considering a car and a wheel example. A car cannot move without a wheel.
- But the wheel can be independently used with the bike, scooter, cycle, or any other vehicle.
- The wheel object can exist without the car object, which proves to be an aggregation relationship.



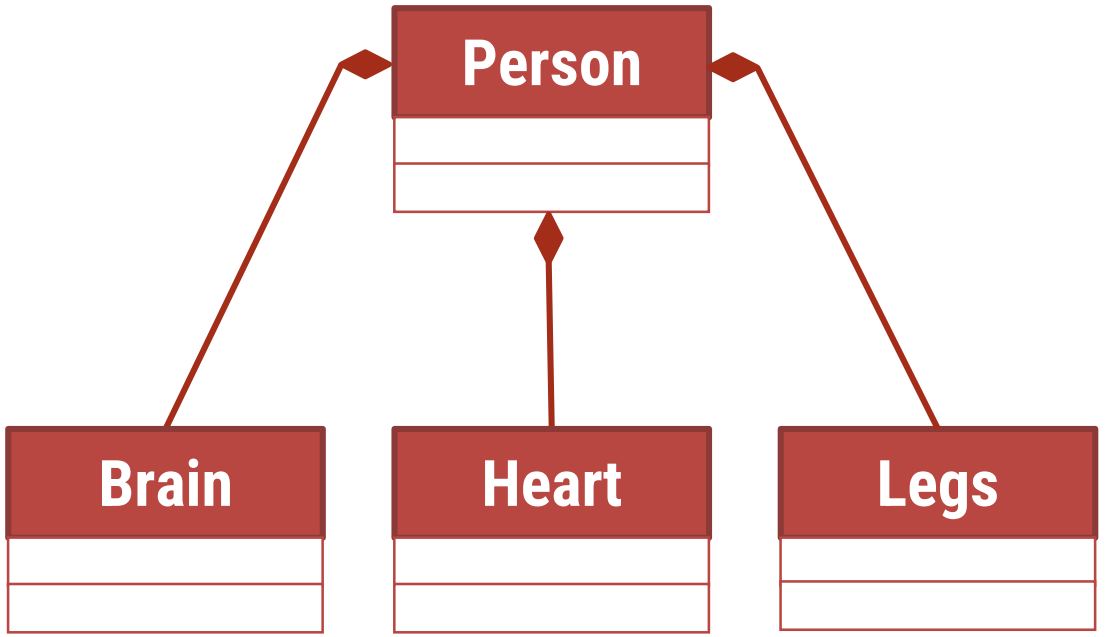
Composition

The composition is a part of the aggregation. It represents the dependency between a parent and its children, which means if the parent is discarded then its children will also be discarded.

It represents 'part-of' relationship.

In composition, both the entities are dependent on each other.

- For e.g.: Person class with Brain class, Heart class, and Legs class.
- If the person is destroyed, the brain, heart, and legs will also get destroyed.



Multiplicity

- **Multiplicity** is the **specification** of the number of **instances of one class** that may be **related** to the instance of another class.
- Multiplicity constrains the **number of a related object**.
- You can use multiple associations between objects.
- Some typical type of multiplicity:

Multiplicity	Option	Cardinality
0..1		No instances or one instance
1..1	1	Exactly one instance
0..*	*	Zero or more instances
1..*		At least one instance
5..5	5	Exactly 5 instances
m..n		At least m but no more than n instances

Example Of Multiplicity

One to One Association

One account holder has one cheque book

Many to Zero or One Association

An account holder can issue at most one debit card.
Or
An account holder can ask for at most one debit card or none.

Many to Many Association

Every account holder can withdraw money from all ATMs.
Or
All account holders can withdraw money from all ATMs.

One to One or Many Association

The bank should have at least one branch.

Role Names or Association End Names

- A role is one end of an association.
- A binary association has 2 roles, each of which may have a role name.
- A role name is a name that uniquely identifies one end of an association.
- **Roles often appear as nouns in problem descriptions.**
- Use of role name is optional.
- Association end names are necessary for associations between two objects of the same class.

Role names for an association

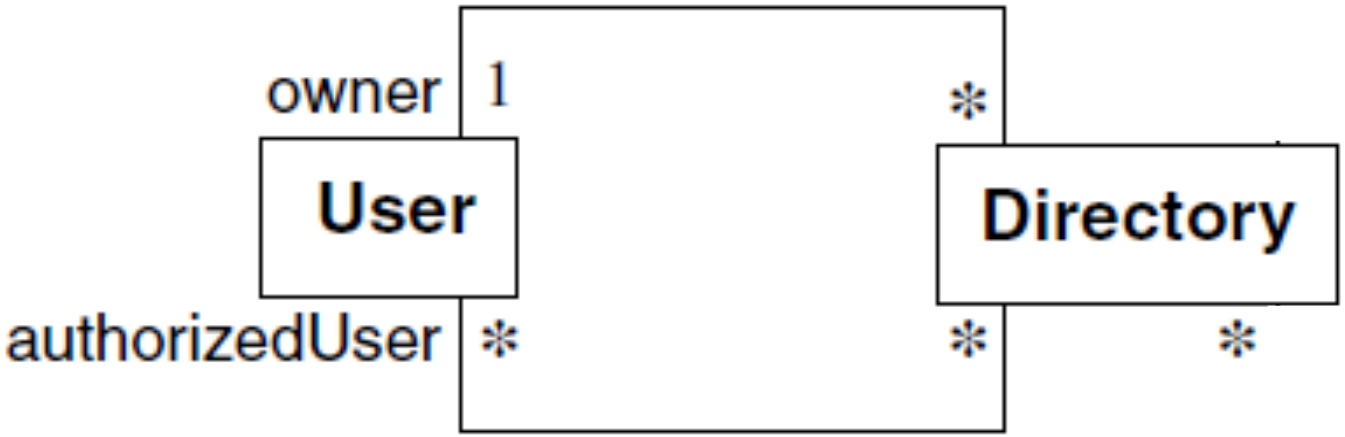
Employee	Employer
Ram	TCS
Mohan	Wipro

A ***reference*** is an attribute in one object that refers to another object.

For example, a data structure for *Person* might contain an attribute *employer* that refers to a *Company* object, and a *Company* object might contain an attribute *employees* that refers to a set of *Person* objects.

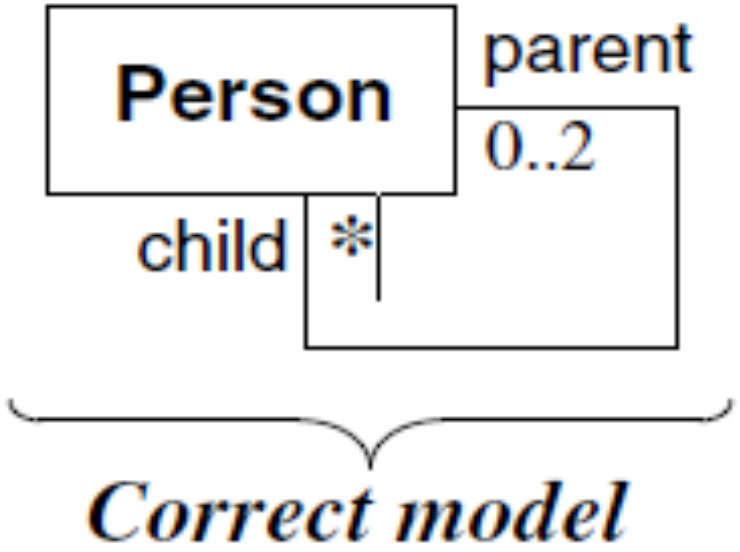
Advantages of Association End Names

Association end names are necessary for associations between two objects of the same class. They can also distinguish multiple associations between a pair of classes.



Container and *contents* distinguish the two usages of *Directory* in the self-association.
Each directory has exactly one user who is an owner and many users who are authorized to use the directory.

Advantages of Association End Names



One person instance participates in two or more links, twice as a parent and zero or more times as a child.

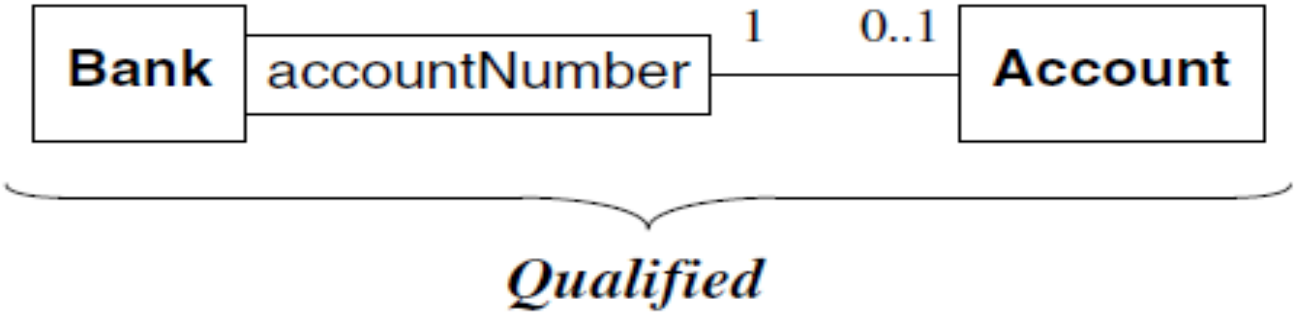
Qualification/Qualified Associations

- A qualified association is an association in which an attribute called the qualifier disambiguates the objects for a “many” association end.

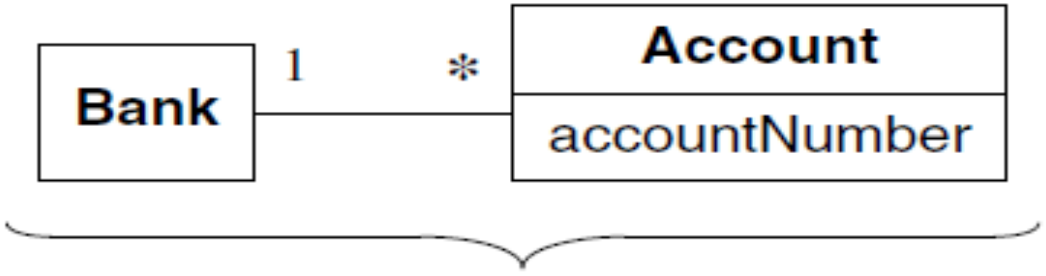
- **Qualification reduces the effective multiplicity of this association from one-to-many to one-to-one.**

- The notation for a qualifier is a small box on the end of the association line near the source class. The qualifier box may grow out of any side (top, bottom, left, right) of the source class. The source class plus the qualifier yields the target class.

- *A qualified association can also be considered a form of ternary association.*



Qualified

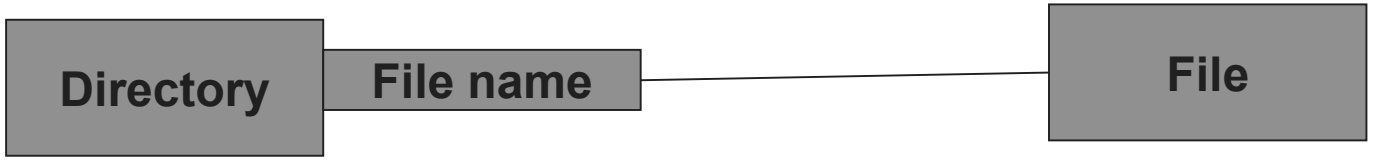


Not qualified

A bank services multiple accounts. An account belongs to a single bank.

Within the context of a bank, the account number specifies a unique account.

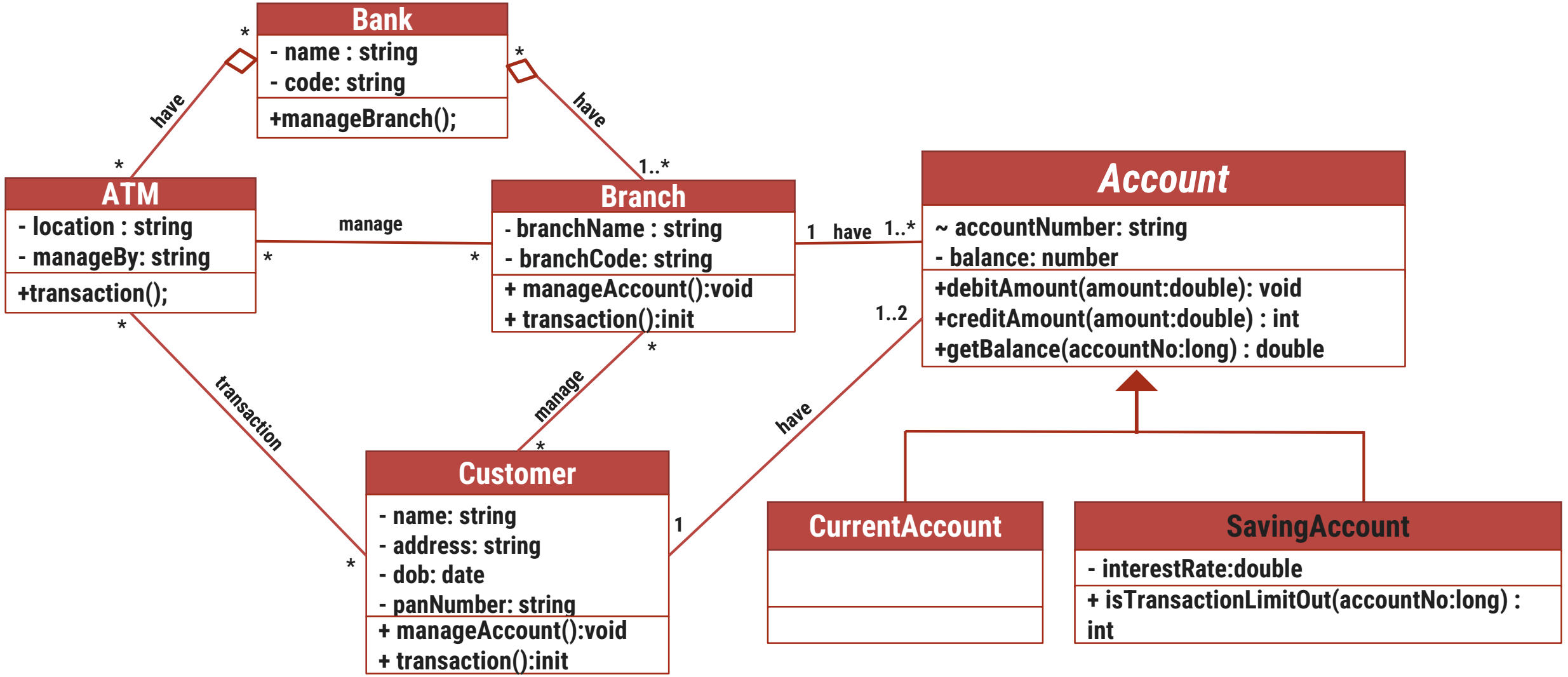
***Bank* and *Account* are classes and *account Number* is the qualifier.**



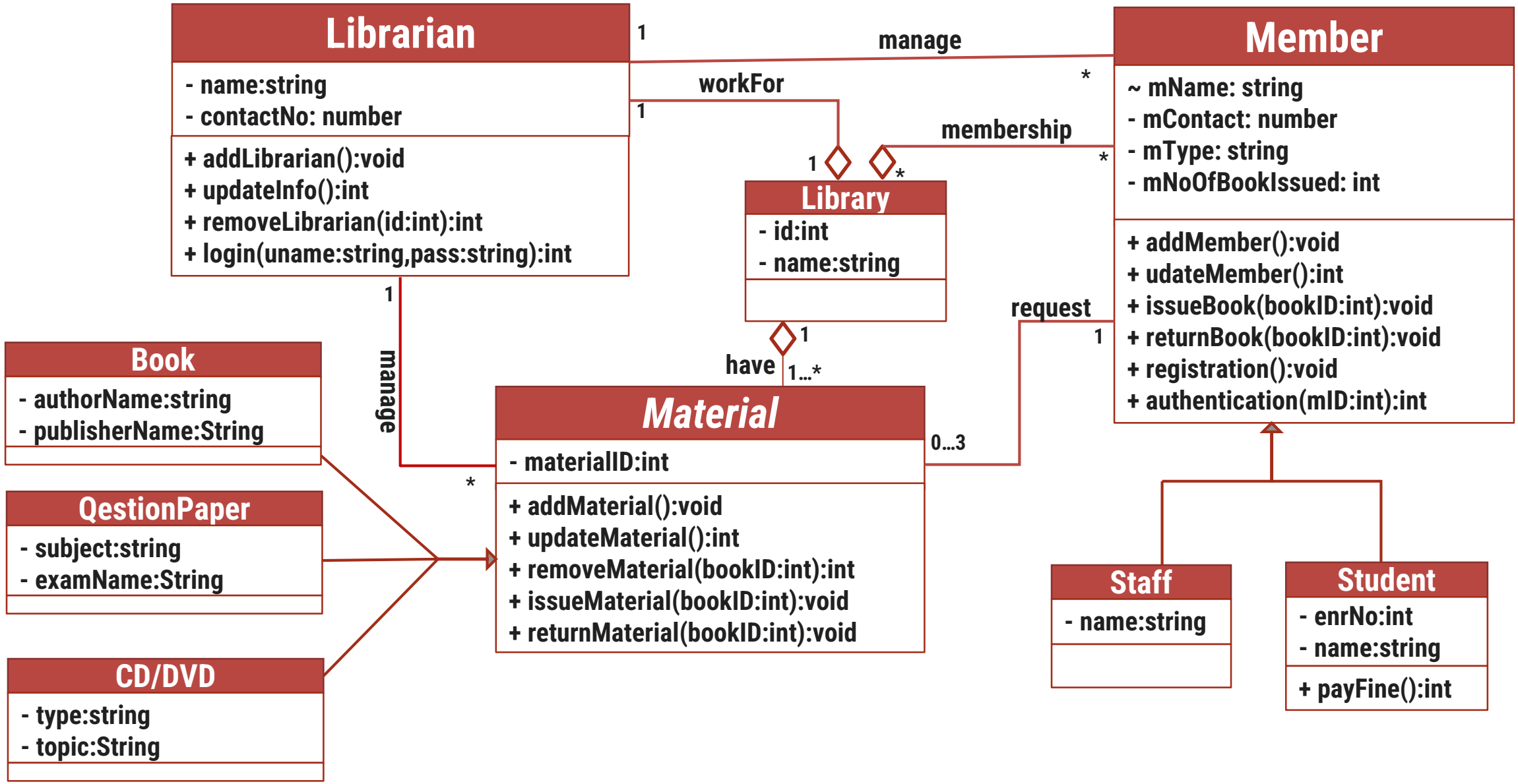
A directory plus a unique file name yields a file

A qualified association

Class Diagram Of Bank Management System



Class Diagram Of Library Management System



Question : Draw Class Diagram for the given Problem

There exists some instructors that are professors, while others have job title adjunct

Departments offer many courses, but a course may be offered by >1 department

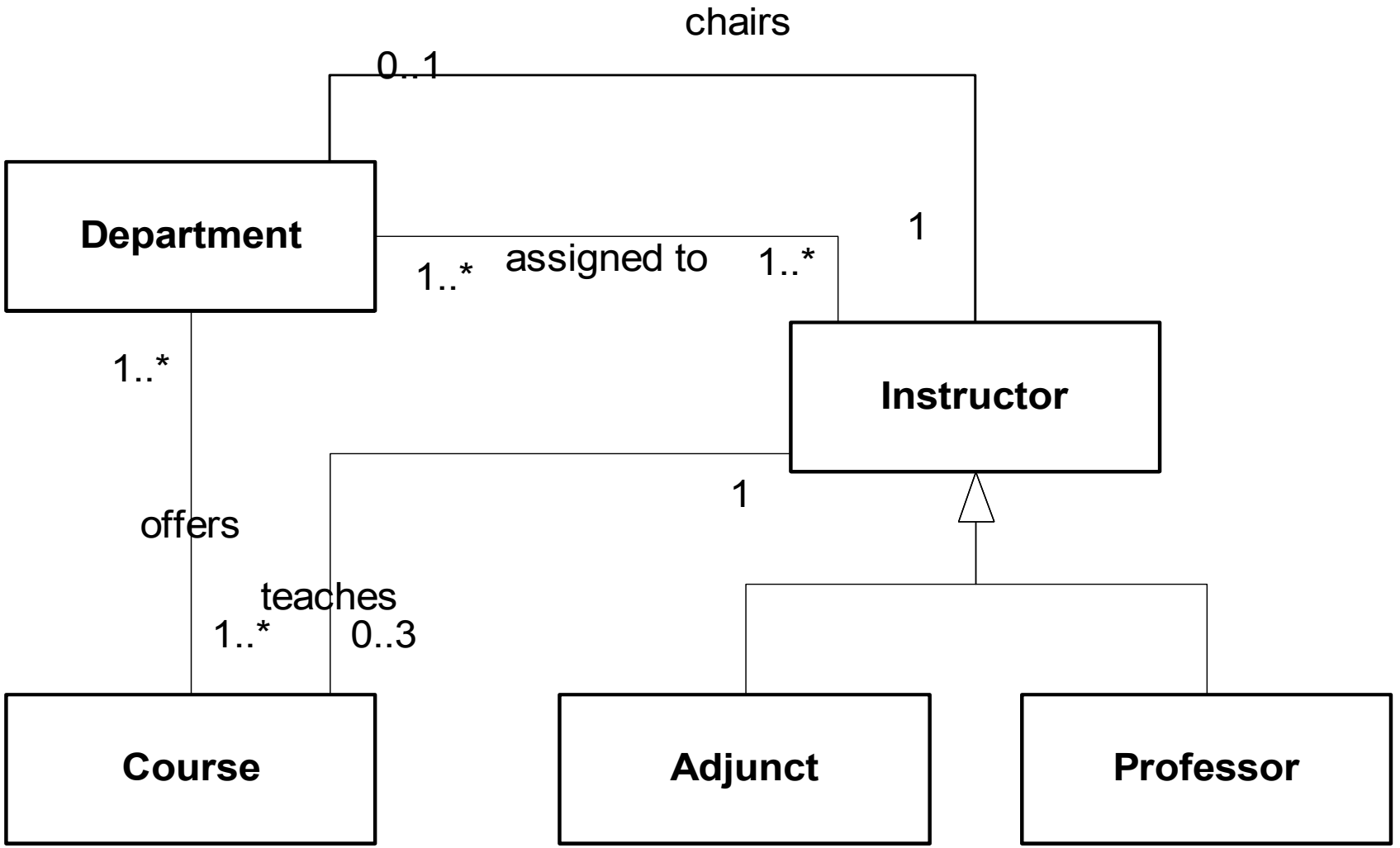
Courses are taught by instructors, who may teach up to three courses

Instructors are assigned to one (or more) departments

One instructor also serves a department chair

Make appropriate variables and operations wherever required.

Class Diagram for Univ. Courses



Question : Draw Class Diagram for the given Problem

An bank institution may issue at most a single credit card accounts, each identified by an account number.

Each account has a maximum credit limit, a current balance, and a mailing address.

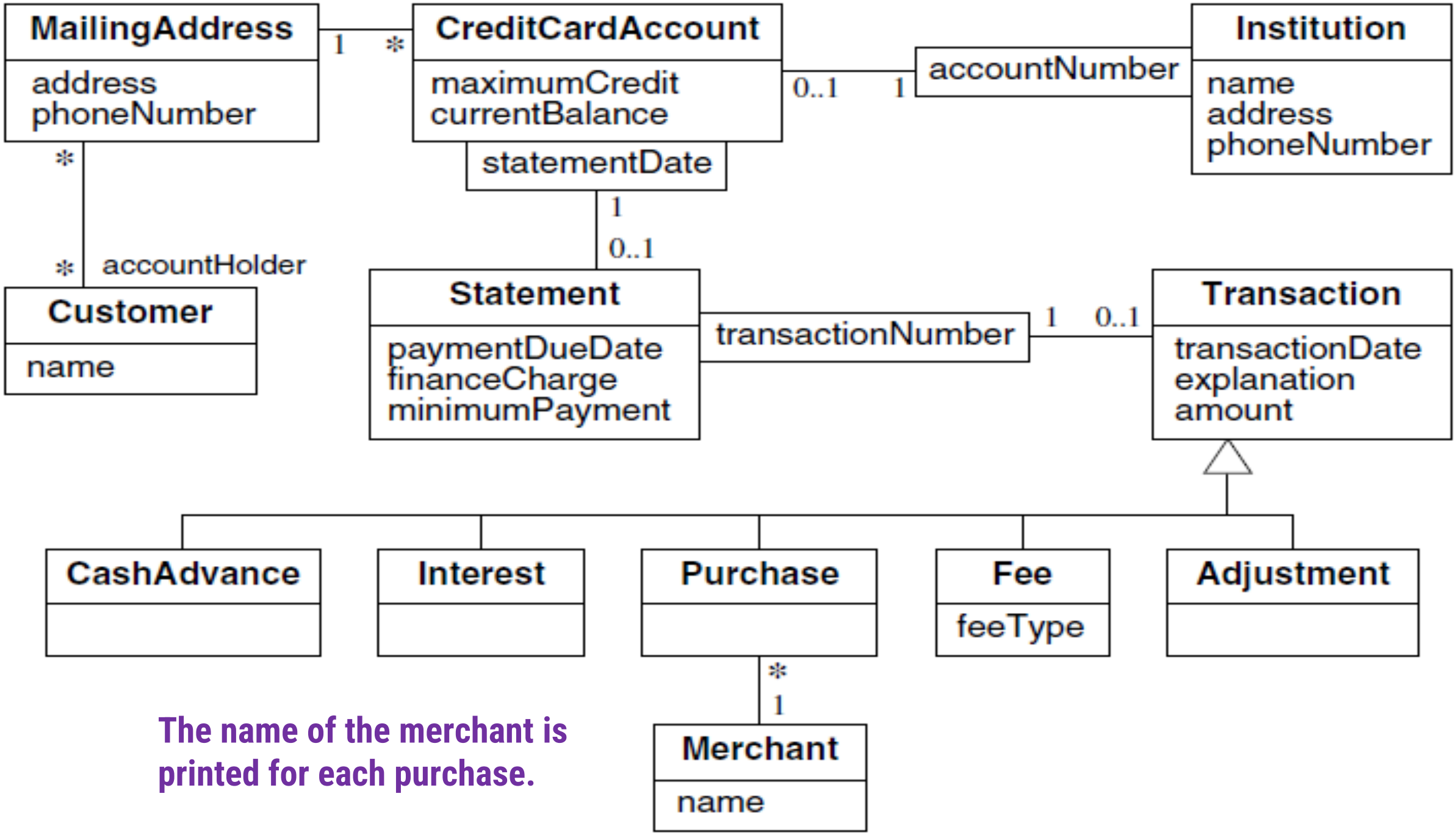
The account serves one or more customers who reside at the mailing address.

The institution periodically issues a statement for each credit-account.

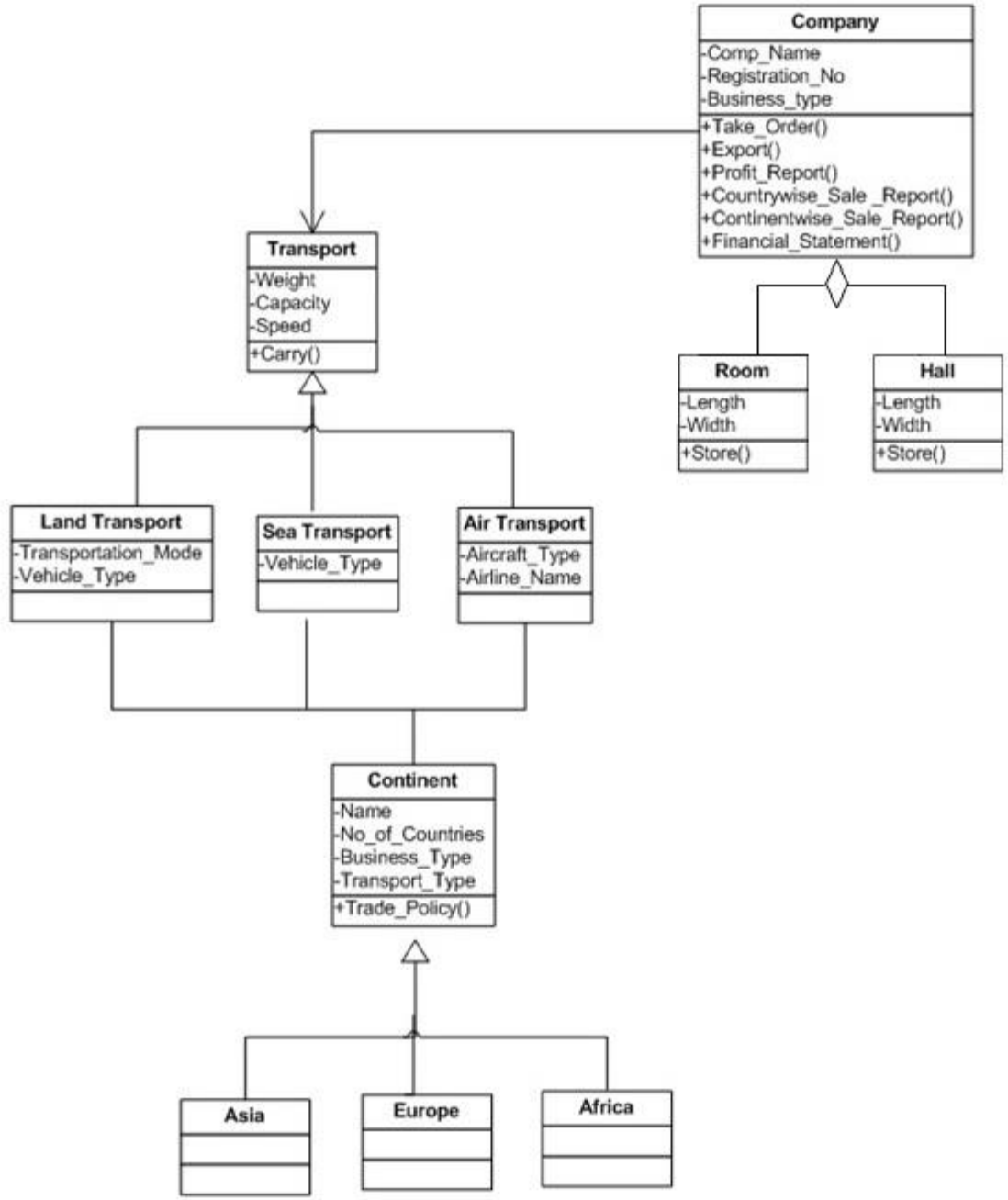
The statement lists a payment due date, finance charge, and minimum payment.

The statement itemizes various transactions that have occurred throughout the billing interval: cash advances, interest charges, purchases, fees, and adjustments to the account.

The name of the merchant is printed for each purchase.



The name of the merchant is printed for each purchase.


Company
-Comp_Name
-Registration_No
-Business_type
+Take_Order()
+Export()
+Profit_Report()
+Countrywise_Sale _Report()
+Continentwise_Sale_Report()
+Financial_Statement()
Transport
-Weight
-Capacity
-Speed
+Carry()
Room
-Length
-Width
+Store()
Hall
-Length
-Width
+Store()
Land Transport
-Transportation_Mode
-Vehicle_Type
Sea Transport
-Vehicle_Type
Air Transport
-Aircraft_Type
-Airline_Name
Continent
-Name
-No_of_Countries
-Business_Type
-Transport_Type
+Trade_Policy()
Asia
Europe
Africa

Thank You

Unit 2
Software Engineering
Lecture 10

- Usecase Diagram

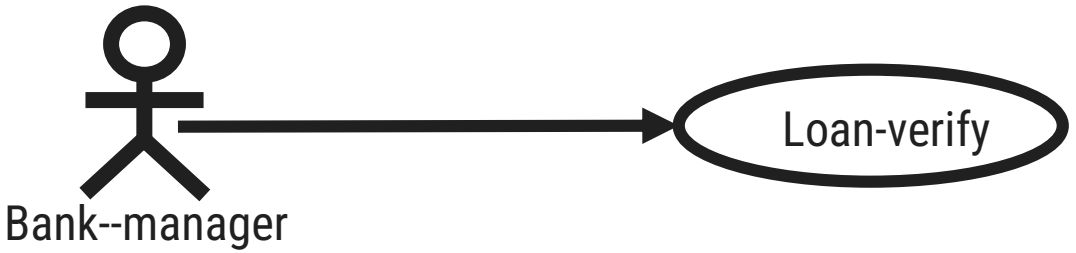
Use Case diagrams

- Use case model is a representation of sequence of transactions, called use cases which are initiated by the user, called here as actor from outside the system

- These diagrams are graphical representation of users view and developers understanding of transactions/methods that are performed.

- ***Use case*** → a sequences of actions (it must be a ***verb***) & it is represented by an ***oval/eclipse.*** A use case is a single unit of meaningful work. It provides a high-level view of behavior observable to someone or something outside the system.

- **Actors:** Used to represent someone or something outside the system that interacts with the system. An actor is usually drawn as a named stick figure.



The arrowhead shows the direction of control. The above diagram indicates that the actor "Bank—manager " uses the "Loan-verify" use case.

The actors are of two types: primary (initiates the use of the system e.g customer) placed at left side and secondary (reactionary e.g bank) at the right.

- ***Relationship*** →illustrates a connection among model elements. They can be association, generalization, extend and include. Each actor has to interact with at least one use case.

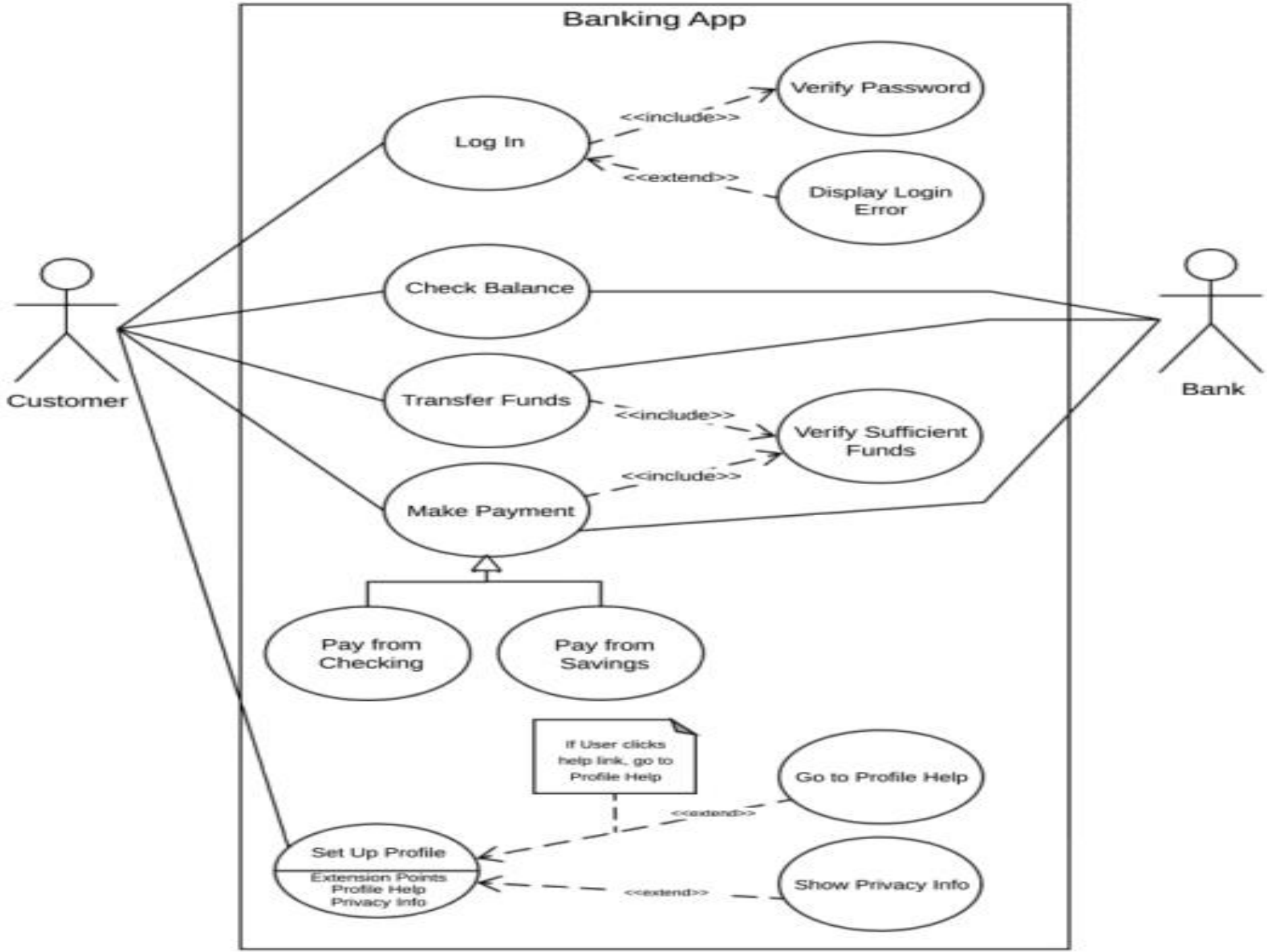
_____ Association **{basic/direct interaction or communication between use case and actor}**

An include relationship is a relationship in which one use case (the base use case) includes the functionality of another use case (the inclusion/included use case). So when base is executed the included is also executed. It is drawn with dashed line arrow pointing to include case with <<include>> written on it.

An extend relationship to specify that one use case (extension/extended) extends the behavior of another use case (base). When base is executed the extended may or may not is executed. It is drawn with dashed line arrow pointing to base case with <<extend>> written on it.

Multiple base cases may connect to single extended or included cases.

- **System:** Whatever we are developing is a system e.g. s/w, business process, website, application. It is represented by a **rectangle** and name of the system is written on top. It defines the **scope** of the system.


Banking App
Verify Password
Log In
<<include>>
<<extend>>
Display Login Error
Check Balance
Customer
Bank
Transfer Funds
<<include>>
Verify Sufficient Funds
<<include>>
Make Payment
Pay from Checking
Pay from Savings
If User clicks help link, go to Profile Help
Go to Profile Help
<<extend>>
Set Up Profile
Extension Points
Profile Help
Privacy Info
<<extend>>
Show Privacy Info

Thank You

Unit 2
Software Engineering
Lecture 11

- Activity & Swimlane Diagram

Activity Diagram

An activity diagram visually presents a series of operation or flow of control in a system similar to algorithm or a flowchart.

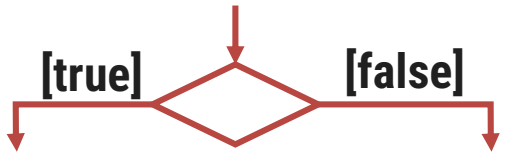
- **An activity diagram is like a traditional flowchart in that it show the flow of control from step to step.**
- **An activity diagram can show both sequential and concurrent flow of control.**
- **Activity diagram mainly focus on the sequence of operation rather than on objects.**
- **Activity diagram represent the *dynamic behavior* of the system or part of the system.**
- **An activity diagram shows 'How' system works.**
- **Activity diagram are most useful during early stages of designing algorithms and workflows.**

Elements of Activity Diagram

Activity

- The **main element** of an activity diagram is the activity itself.
- An activity is a **function/operation performed by the system.**
- The elongated **ovals** show activities.
- An unlabeled **arrow from one activity to another** activity, that indicates that the **first activity must complete before the second activity begin**.

Branches/ Decision



- If there is more than one successor to an activity, each arrow may be labeled with a condition in square brackets. For e.g. *[failure]*
- As a notational convenience, a **diamond shows a branch** into multiple successors.
- The diamond has one incoming arrows and two or more outgoing arrows. Each with condition.
- Shows mutual exclusive options.

Elements of Activity Diagram Cont.

Initiation

- A solid circle with an outgoing arrow shows the starting point of an activity diagram.
- When an activity diagram is activated, control starts at the solid circle and proceeds via the outgoing arrow toward the first activities.

Termination

- A bull's eye – a solid circle surrounded by a hollow circle shows the termination point.
- The symbol only has incoming arrows.
- When control reaches a bull's eye, the overall activity is complete and execution of the activity diagram ends.

Control Flow

- A solid line with an arrow represent the direction of flow of the activities.
- The arrow points to the direction of the progression activities.

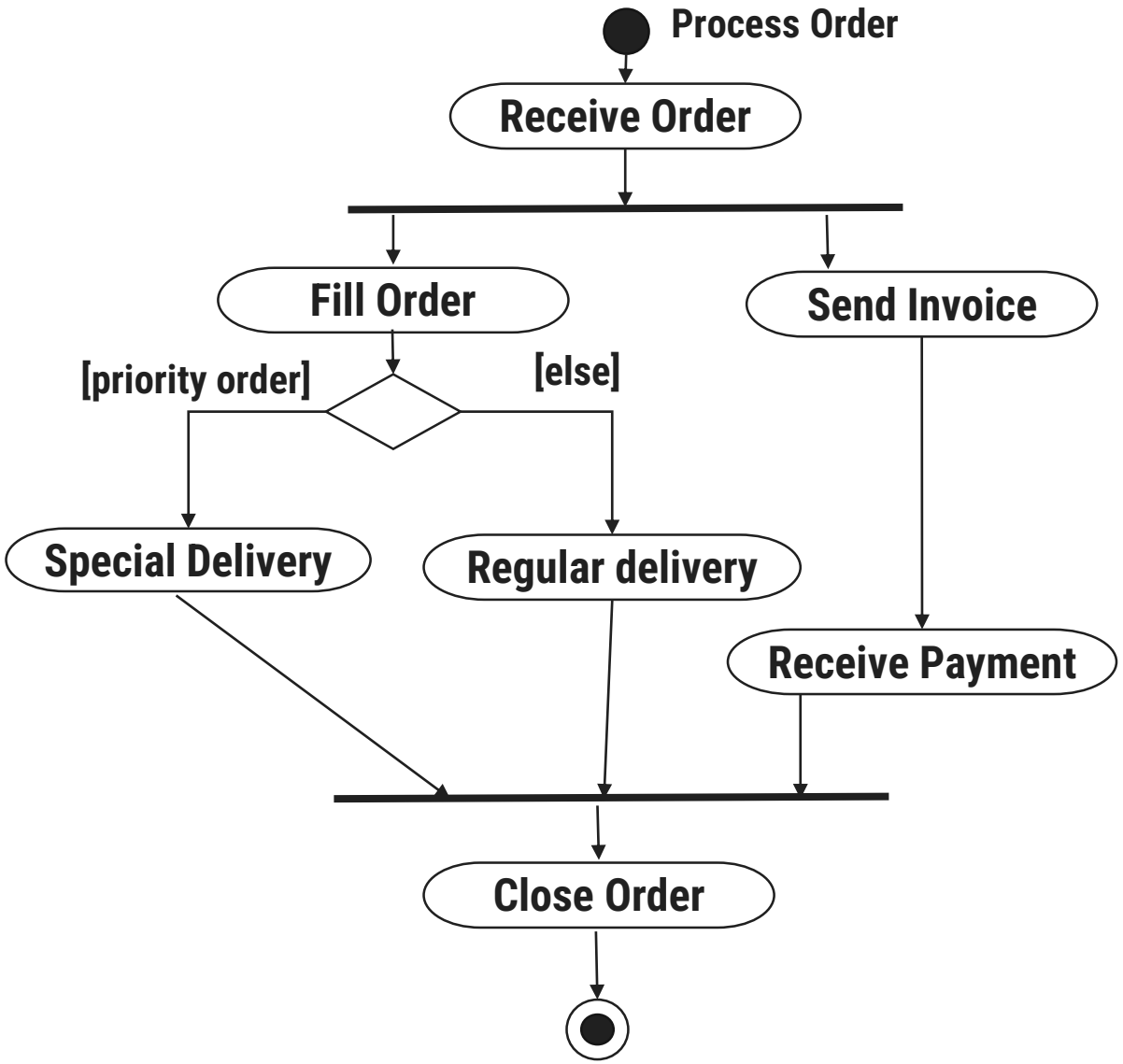
Elements of Activity Diagram Cont.

Concurrent Activities

- System can perform more than one activity at a time.
- For e.g. one activity may be followed by another activity, then split into several concurrent activities (a fork of control), and finally be combined into a single activity (a merge/join of control).
- A fork or merge is shown by a synchronization bar –a heavy line with one or more input arrows and one or more output arrows.

Example of Fork & Join

- An example of business flow activity of order processing, based on the Example order is input parameter of the activity.
- After order is accepted and all required information is filled in, payment is accepted, and order is shipped.
- Note, that this business flow allows order shipment before invoice is sent or payment is confirmed.



Guideline for Activity Diagram

- Activity diagram elaborate the details of computation, thus documenting the steps needed to implement an operation or a business process.
- Activity diagram can help developers to understand complex computations by graphically displaying the progression through intermediate execution steps.
- Here is some advice for activity diagram.

Don’t misuse activity diagram

- Activity diagrams are intended to elaborate use case and sequence models so that a developer can study algorithms and workflow.
- Activity diagrams supplement the object-oriented focus of UML models and should not be used as an excuse to develop software via flowchart.

Guideline for Activity Diagram

Level diagrams

- Activities on a diagram should be at a consistent level of details.
- Place additional details for an activity in a separate diagram.

Be careful with branches and conditions

- If there are conditions, at least one must be satisfied when an activity completes, consider using an *[else]* condition.
- It is not possible for multiple conditions to be satisfied otherwise this is an error condition.

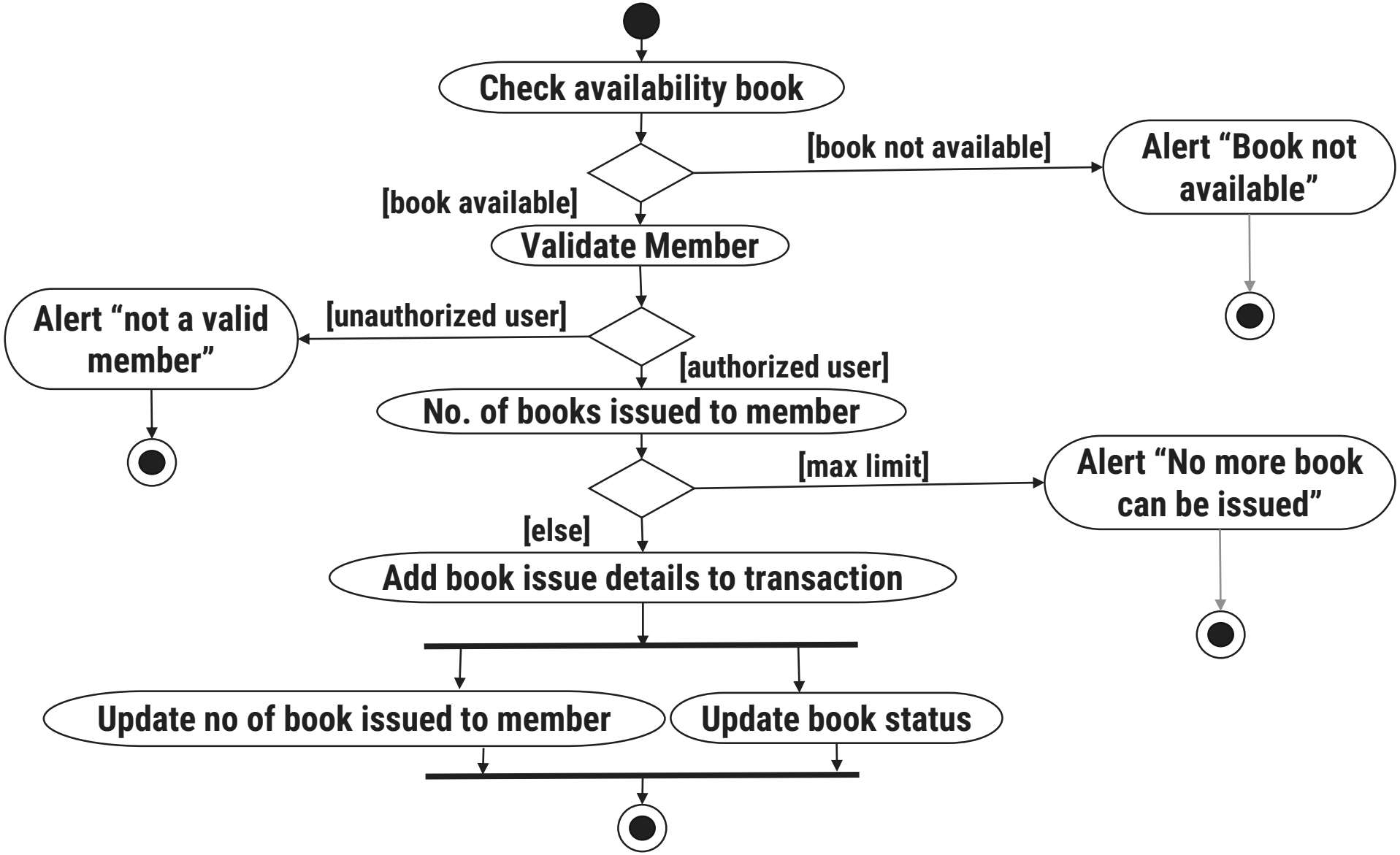
Be careful with concurrent activities

- Means that the activities can complete in any order and still yield an acceptable result.
- Before a merge can happen, all inputs must first complete

How to Draw an Activity Diagram

- **Step 1: Identify the various activities and actions your business process or system**
- **Step 2: Find a flow among the activities**
- **For e.g. in library management system, book issue is a one business process or a function. Show we prepare a activity diagram for Book issue.**
- **Various activity in book issue process like…**
  - **Check availability of book**
  - **Validate the member**
  - **Check No. of books issued by member**
  - **Add book issue details to transaction**
  - **Update no of book issued by member**
  - **Update book status.**

Activity Diagram for Issuing Book from the library



Activity Diagram for Issuing e-Book from the e-library

1. The person has to login first so he will register or not.
2. Do he need to return any book if member.
3. Whether any fine.
4. If all denied then look out for the book.
5. If available then view it and download else search more and exit after some time.

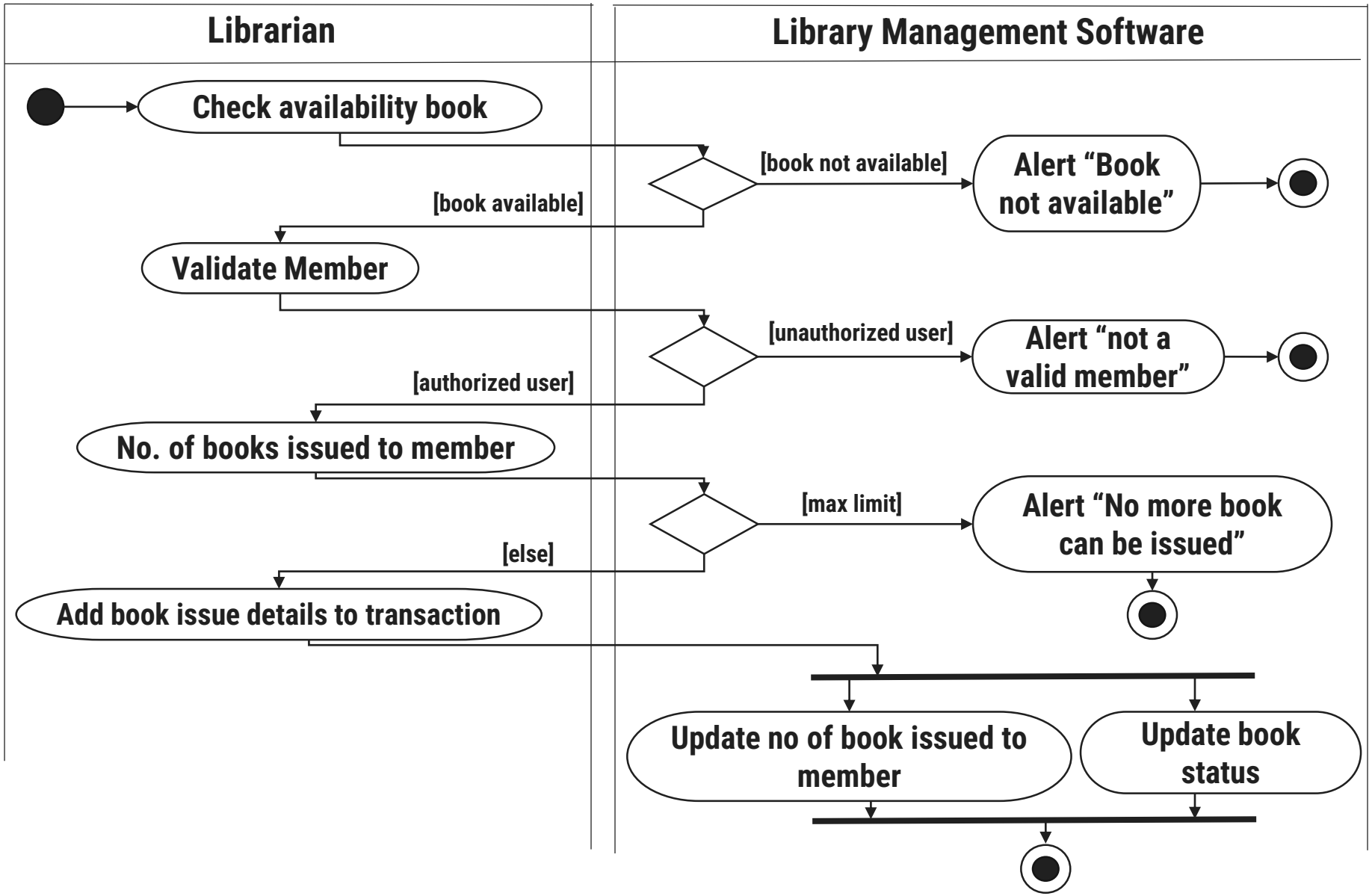
Swimlane Diagram : which role is performing the underlying activity

- In a business model, it is often useful to know which human department is responsible for an activity.
- When design of the system is complete, the activity will be assigned to a person/department, but at a high level it is sufficient to partition the activities among departments.
- You can show such a partitioning with an activity diagram by dividing in to columns and lines.
- Each column is called swim-lane by analogy to a swimming pool.
- Placing an activity within a particular swim-lane indicates that is performed by a person/department.
- Lines across swim-lane boundaries indicate interaction among different person/department.

How to Draw a Swimlane Diagram

- **Step 1:** Identify the various activities and actions your business process or system
- **Step 2:** Figure out which person/departments are responsible for the competition of activity.
- **Step 3:** Figure out in which order the actions are processed.
- **Step 4:** Figured out who is responsible for each action and assign them a swimlane and group each action they are responsible for under them

Swimlane Diagram for Book Issue



Thank You

Unit 2
Software Engineering
Lecture 12

- Sequence Diagram

The Sequence diagram shows the message flow from one object to another object.
The Activity diagram shows the message flow from one activity to another.

Sequence Diagram

A Sequence diagram shows the participants (Objects) in an interaction and the sequence of message among them.

- A sequence diagram shows the interaction of a system with its actors to perform all or part of a use case.
- Sequence diagram represent the dynamic communication between object during execution of task.
- Each use case requires one or more sequence diagram to describe its behavior.
- Each sequence diagram shows a particular behavior sequence of the use case.
- Graphically, a sequence diagram is a table that shows objects arranged along the X axis and messages, ordered in increasing time, along the Y axis.
- A *sequence diagram* is an interaction diagram that emphasizes the time ordering of messages.

Components of Sequence Diagram

Object - Class Roles or Participants

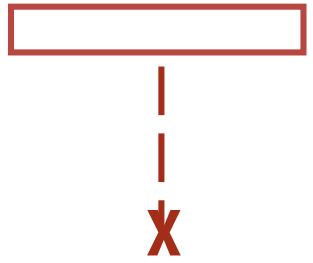
Object : Class

- Class roles describe the way an object will behave in context.
- Use the UML object symbol to illustrate class roles, but don't list object attributes.

Activation or Execution Occurrence

- Activation boxes represent the time an object needs to complete a task.
- When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.

Lifeline



- A lifeline represents an Object in an interaction i.e. the existence object at a particular time.
- When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

Components of Sequence Diagram Cont.

Messages

- Messages are arrows that represent communication between objects.
- Use the following arrows and message symbols to show how information is transmitted between objects.

Synchronous message 

- Represented by a solid line with a solid arrowhead.
- This symbol is used when a sender wait for a response to a message before it continues.
- The diagram should show both the call and the reply.

Asynchronous message 

- Represented by a solid line with a lined arrowhead.
- Asynchronous messages don't require a response before the sender continues.
- Only the call should be included in the diagram.

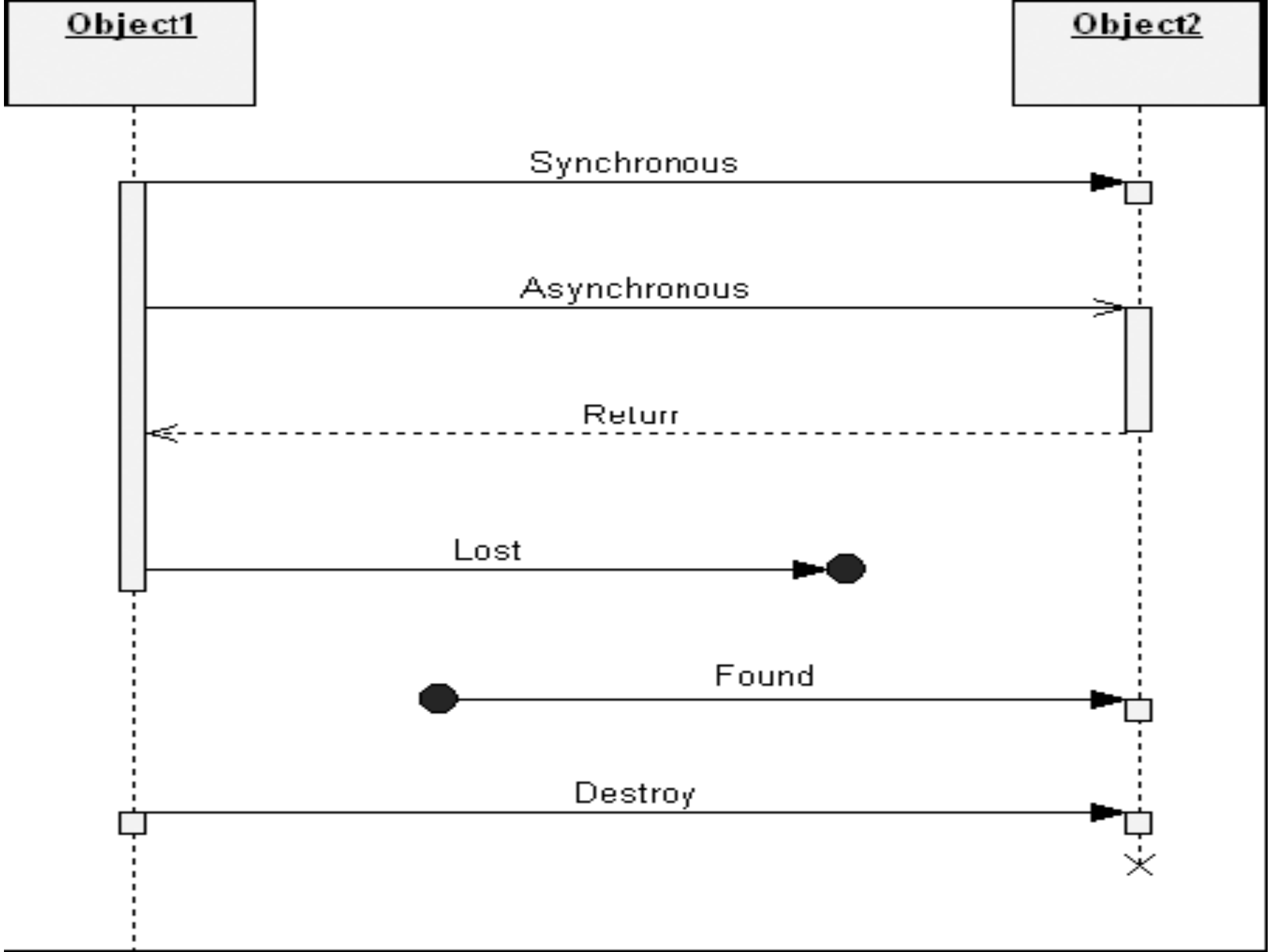
Components of Sequence Diagram Cont.

Reply message

- Represented by a **dashed line** with a lined arrowhead.
- these messages are **replies to calls**.

Delete message

- Represented by a **solid line** with a solid arrowhead, followed by an **X**.
- This message **destroys an object**.


Object1
Object2
Synchronous
Asynchronous
Returr
Lost
Found
Destroy

Guideline for Sequence Diagram

Prepare at least one scenario per use case

- The steps in the scenario should be logical commands, not individual button clicks.
- You can specify the exact syntax of input.
- Start with the simplest mainline interaction - no repetitions, one main activity, and typical values for all parameters.
- If there are substantially different mainline interactions, write a scenario for each.

Abstract the scenarios into sequence diagrams.

- The sequence diagrams clearly show the contribution of each actor.
- It is important to separate the contribution of each actor as a prelude to organizing behavior about objects.

Divide complex interactions

- Break large interactions into their constituent tasks and prepare a sequence diagram for each of them.

Prepare a sequence diagram for each error condition.

- Show the system response to the error condition.

Steps to Draw a Sequence Diagram

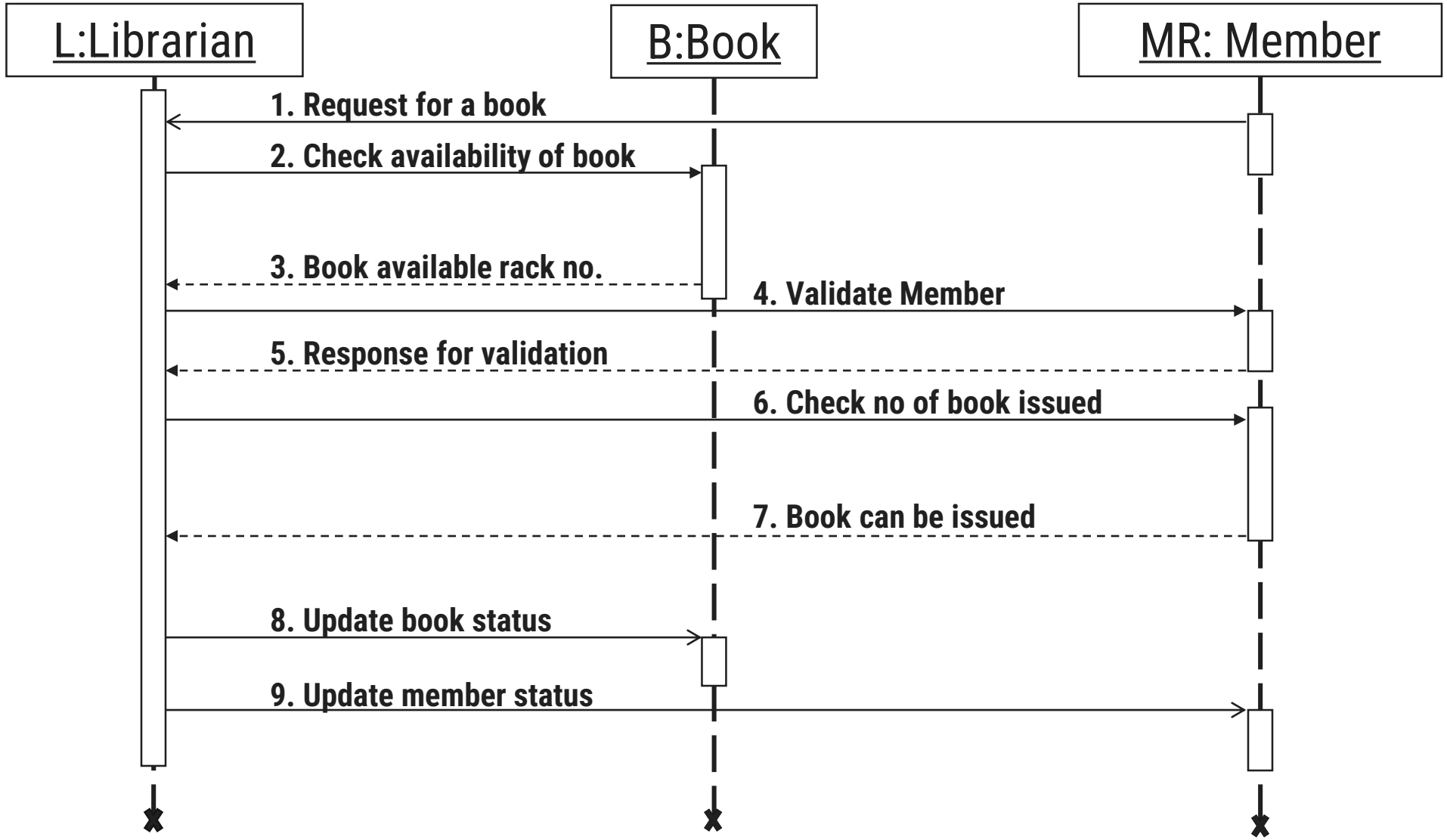
Step-1	Select one scenario
Step-2	Identify the necessary set of the objects. Who is taking part ?
Step-3	Identify the necessary interactions/steps.
Step-4	Describe the message exchange between object.
Step-5	Identify the sequence of interactions and who starts Interactions.

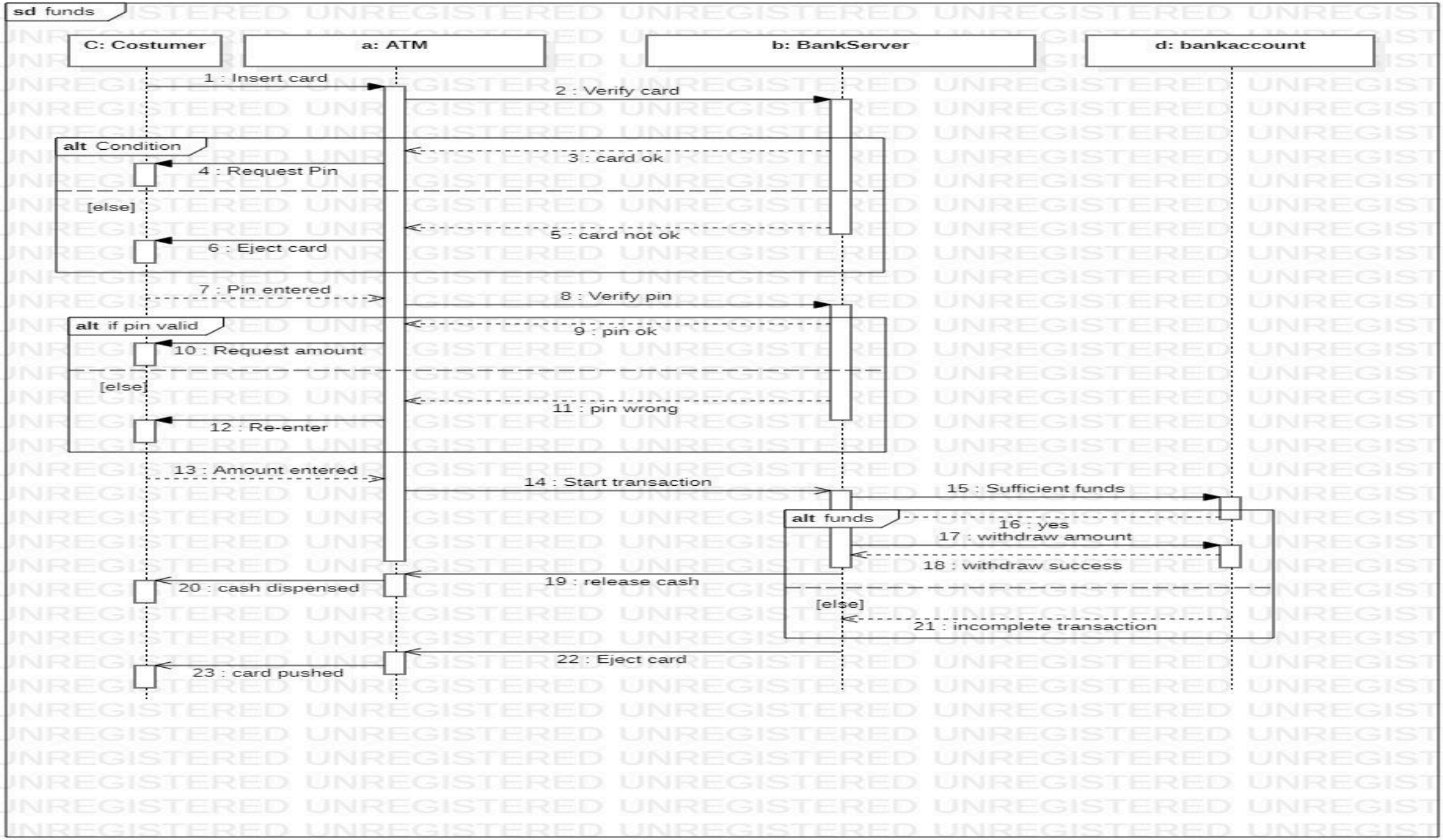
Example: Sequence Diagram for Book Issue

- **Book issue** is a one business process or a **function** in Library Management System.
- Necessary objects for book issue process are **Librarian**, **Book**, **Member** and **Transaction** .
- **Member class object starts the interaction.**
- Various interactions in book issue process are

1	Request for a book
2	Check availability of book
3	Validate the member
4	Check No. of books issued by member
5	Add book issue details to transaction
6	Update no of book issued by member
7	Update book status

Sequence Diagram for Book Issue




sd funds
C: Costumer
a: ATM
b: BankServer
d: bankaccount
1 : Insert card
2 : Verify card
alt Condition
3 : card ok
4 : Request Pin
[else]
5 : card not ok
6 : Eject card
7 : Pin entered
8 : Verify pin
alt if pin valid
9 : pin ok
10 : Request amount
[else]
11 : pin wrong
12 : Re-enter
13 : Amount entered
14 : Start transaction
15 : Sufficient funds
alt funds
16 : yes
17 : withdraw amount
18 : withdraw success
19 : release cash
20 : cash dispensed
[else]
21 : incomplete transaction
22 : Eject card
23 : card pushed

Thank You

Unit 2
Software Engineering
Lecture 13

- State Diagram

The Activity diagram shows the message flow from one activity to another. When an activity gets completed we transit from one state to another. Sequence diagram tells the coordination of the events in an activity that triggers a state change.

State diagram

A state diagram is a graph whose nodes are states and whose directed arcs are transitions between states.

A state diagram specifies the state sequences caused by event sequences.

- The state diagram is a standard computer science concept that relates events and state.
- Events represent external stimuli.
- States represent value of objects.
- All objects in a class execute the state diagram for that class, which models their common behavior.
- The UML notation for a state diagram is a rectangle with object name in a small pentagonal tag in the upper left corner.

Components of state diagram

Initial State

- A solid circle with an outgoing arrow shows the initial state.

Final State

- A bull's eye – a solid circle surrounded by a hollow circle/encircled X shows the termination point.

State

- Drawn as a rounded box containing the name of the state.
- State names must be unique within the scope of the state diagram.
- Our convention is to list state name in boldface, center the name near the top of the box, and capitalize the first letter.

Transition/Event

- Drawn as a line from the origin state to the target state.
- An arrowhead points to the target state.

Components of state diagram

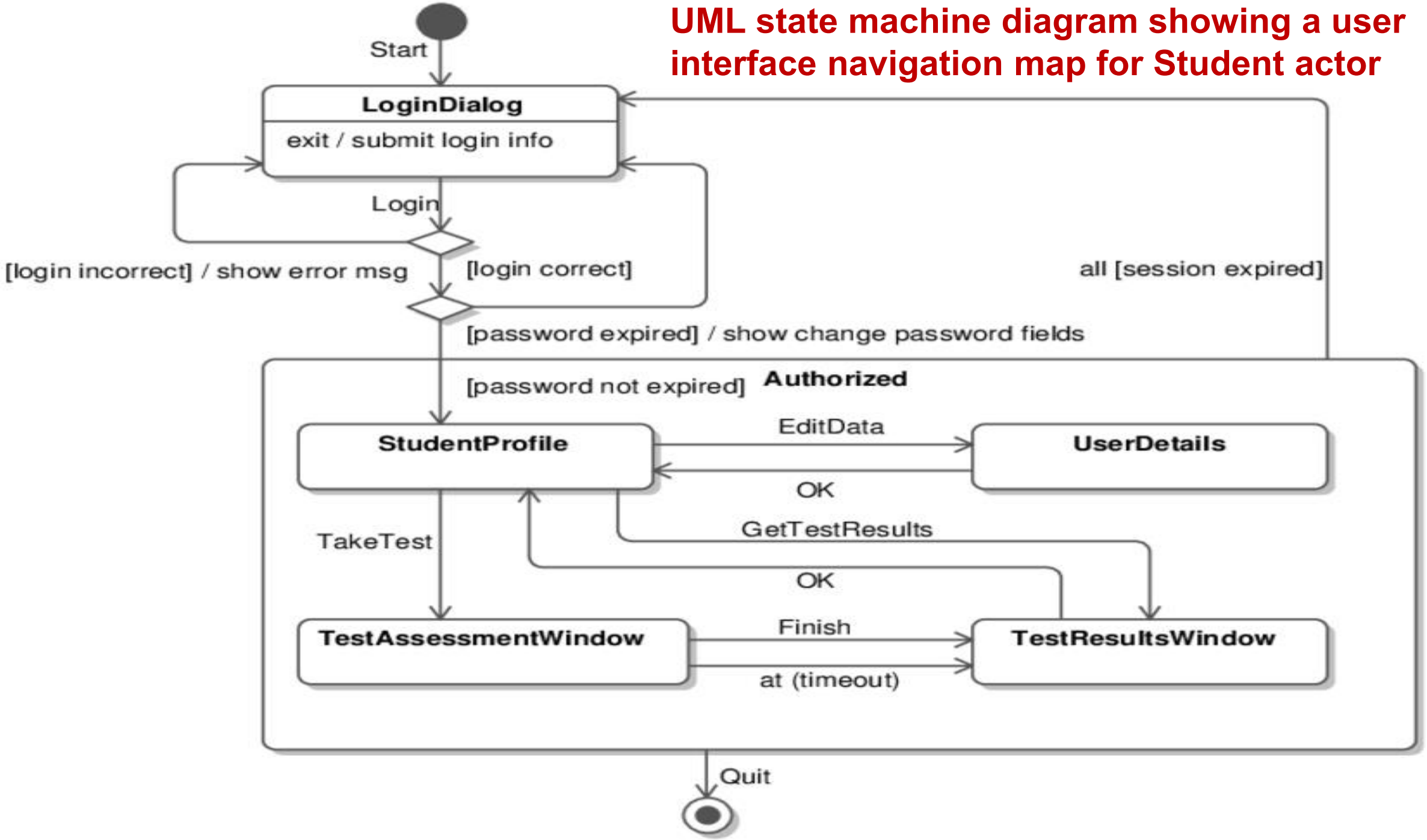
Guard condition

- A guard condition is a Boolean expression that must be true in order for a transition to occur.
- A guarded transition fires when its event occurs.
- Optionally listed in square brackets after an event.

How to draw a state diagram

- **Step 1:** Identify the important objects.
- **Step 2:** Identify the possible states in which the object can exist.
- **Step 3:** Identify the initial state and the final terminating states.
- **Step 4:** Label the events which trigger these transitions.

UML state machine diagram showing a user interface navigation map for Student actor



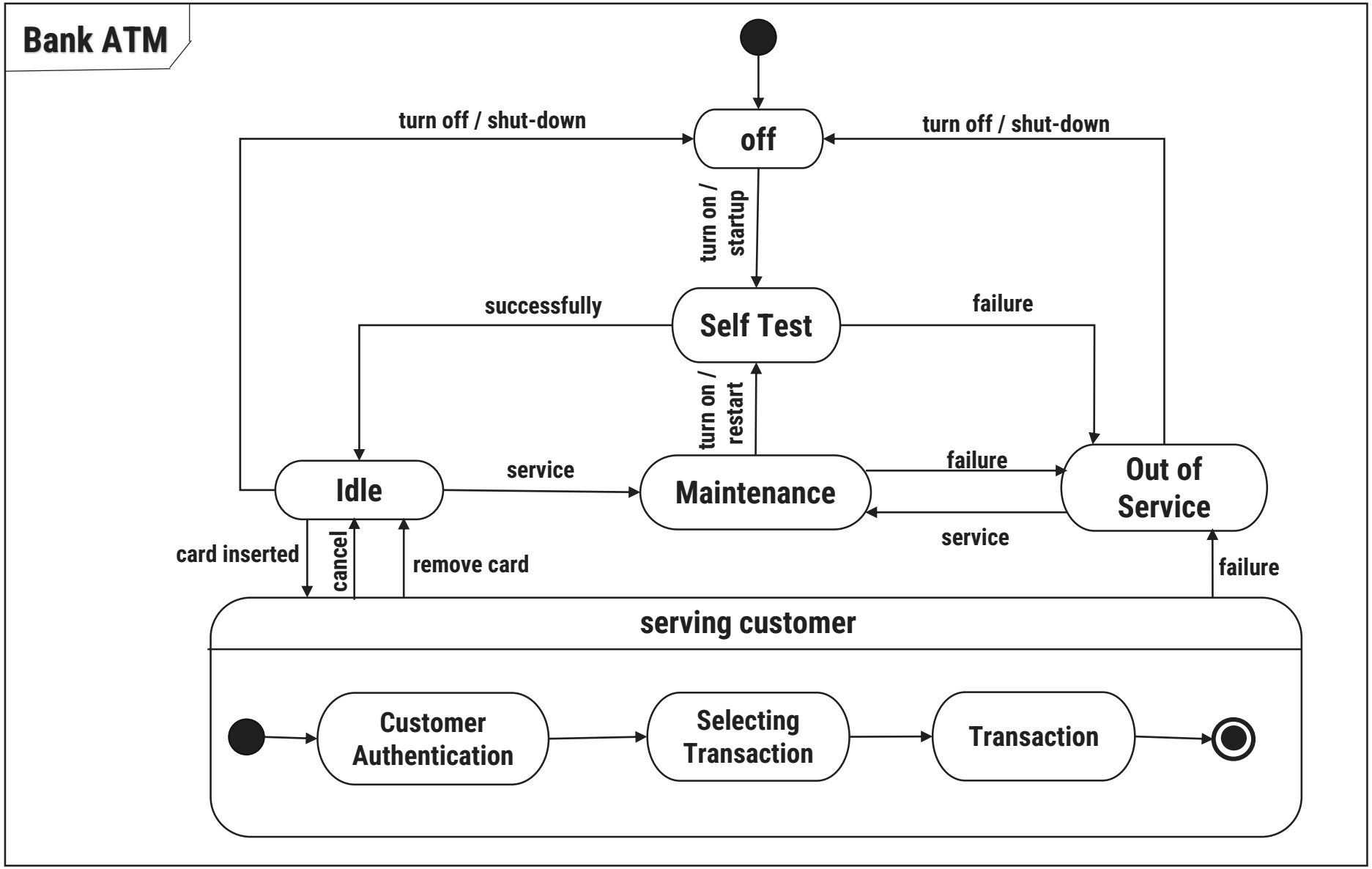
State diagram for library management system

- Identify the important objects
  - Book
  - CD/DVD
  - News Paper
  - Librarian
  - Member
- Identify the states of Book's Object
  - Available
  - Issue
  - Return
  - Renew
- Identify the events / transition
  - Book issued to user
  - Submit the book
  - Request to issue same book
  - Completion of exam / end of the Semester

Example: State diagram of Bank Automated Teller Machine (ATM)

- This is an example of UML behavioral state diagram showing Bank Automated Teller Machine (ATM) top level state machine.
- ATM is initially turned off. After the power is turned on, ATM performs startup action and enters Self Test state.
- If the test fails, ATM goes into Out of Service state, otherwise there is trigger less transition to the Idle state. In this state ATM waits for customer interaction.
- The ATM state changes from Idle to Serving Customer when the customer inserts banking or credit card in the ATM's card reader.
- On entering the Serving Customer state, the entry action readCard is performed.
- Note, that transition from Serving Customer state back to the Idle state could be triggered by cancel event as the customer could cancel transaction at any time.
- Serving Customer state is a composite state with sequential substates Customer Authentication, Selecting Transaction and Transaction.

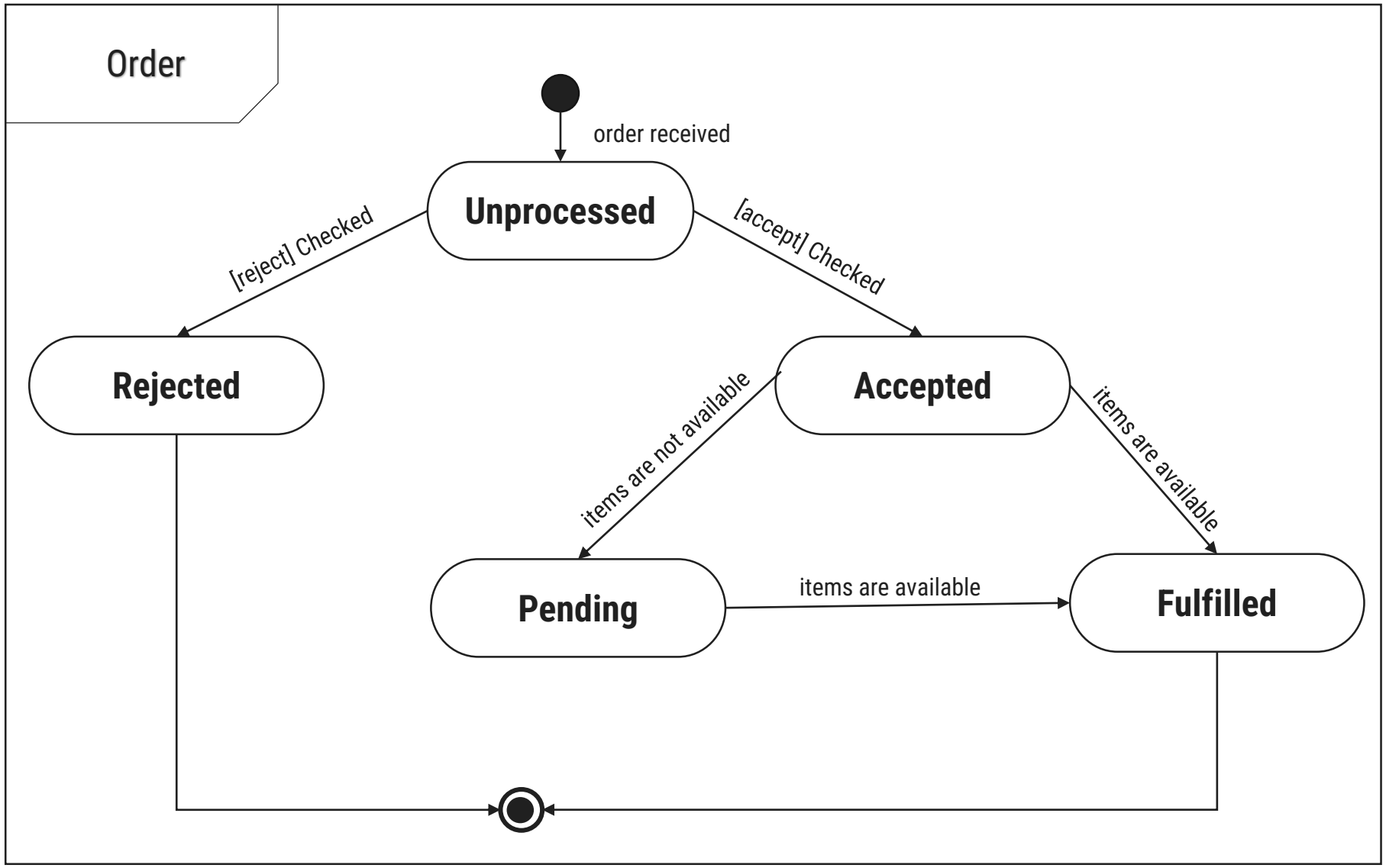
State diagram of Bank Automated Teller Machine (ATM)

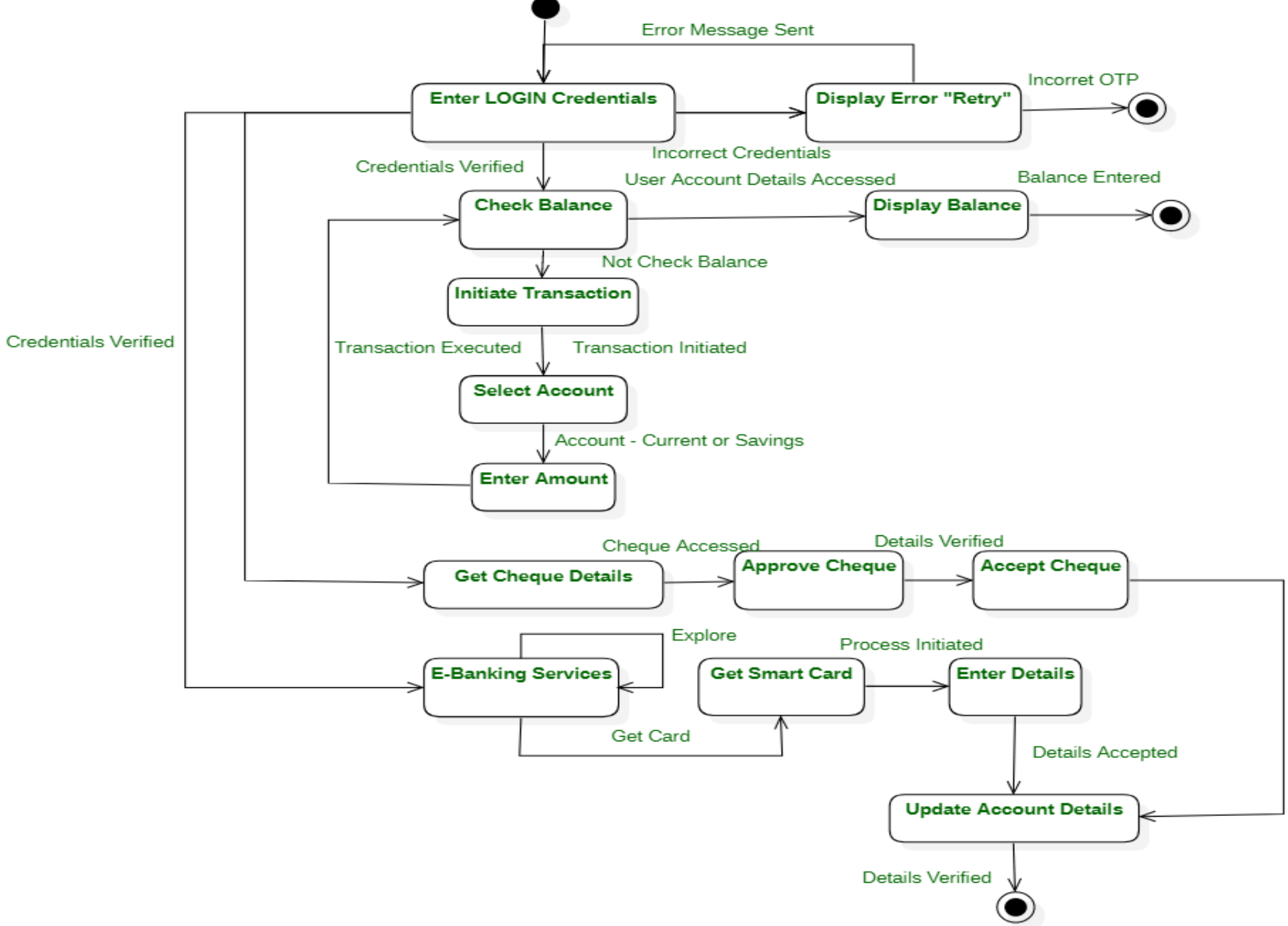


Example: State diagram of Online Order

- Here is just another example of how an online ordering system might look like.
- On the event of an order being received, we transit from initial state to Unprocessed order state.
- The unprocessed order is checked.
- If the order is rejected, we transit to the Rejected order state.
- If the order is accepted and we have the items available, we transit to the fulfilled order state.
- However if the items are not available, we transit to the Pending order state.
- After the order is fulfilled, we transit to the final state. In this example, we merge the two states i.e. Fulfilled order and Rejected order into one final state.

State diagram of Online Order




Error Message Sent
Enter LOGIN Credentials
Display Error "Retry"
Incorret OTP
Incorrect Credentials
Credentials Verified
User Account Details Accessed
Balance Entered
Check Balance
Display Balance
Not Check Balance
Initiate Transaction
Credentials Verified
Transaction Executed
Transaction Initiated
Select Account
Account - Current or Savings
Enter Amount
Cheque Accessed
Details Verified
Get Cheque Details
Approve Cheque
Accept Cheque
Explore
Process Initiated
E-Banking Services
Get Smart Card
Enter Details
Get Card
Details Accepted
Update Account Details
Details Verified

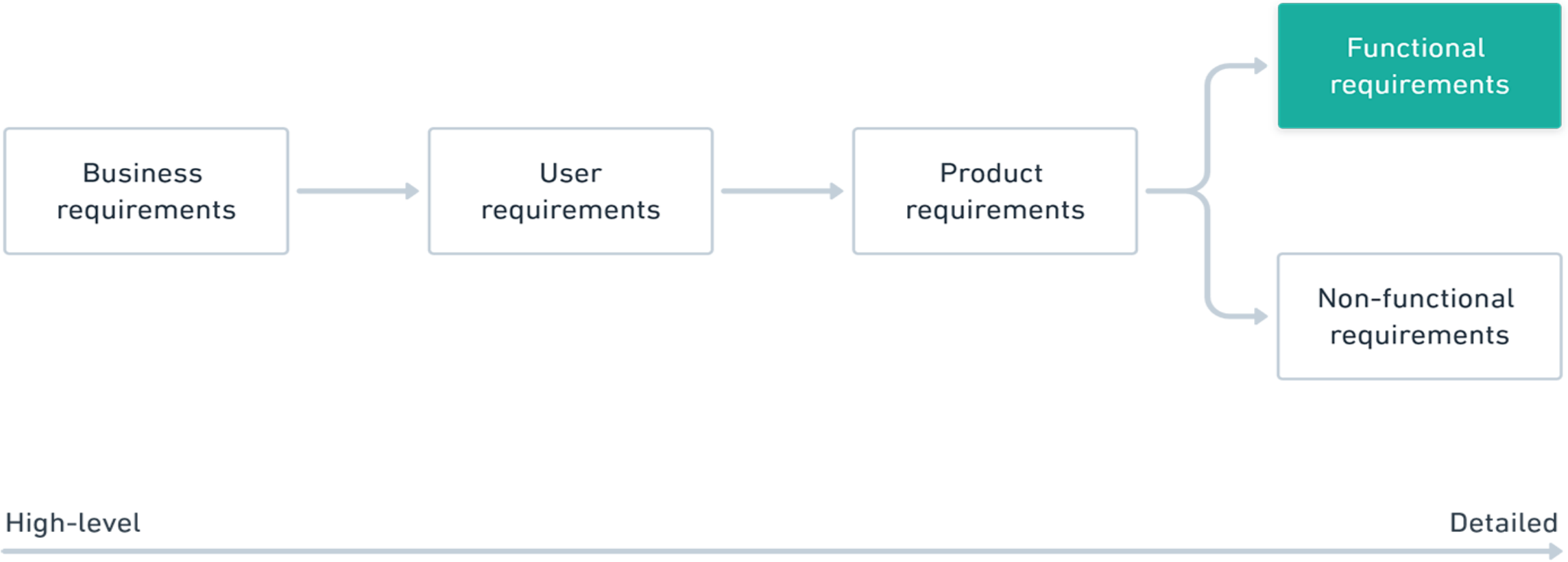
Thank You

Unit 1
Software Engineering
Lecture 14

Requirements Engineering Process

What are functional requirements?

Functional requirements are product features that developers must implement to enable the users to achieve their goals. They define the basic system behavior under specific conditions.



What are non-functional requirements?

Functional requirements or NFRs are a set of specifications that describe the system’s operation capabilities and constraints and attempt to improve its functionality. These are basically the requirements that outline how well it will operate including things like speed, security, reliability, data integrity, etc.

Business requirements describe the high-level business needs, such as carving a market share, reducing customer churn, or improving the customers' lifetime value.

User requirements cover the different goals your users can achieve using the product and are commonly documented in the form of user stories, use cases, and scenarios.

Product requirements describe how the system needs to operate to meet the business and user requirements. They include **functional requirements** and **non-functional requirements.**

Examples of well-written functional requirements:

- **The system must send a confirmation email whenever an order is placed.**
- **The system must allow blog visitors to sign up for the newsletter by leaving their email.**
- **The system must allow users to verify their accounts using their phone number.**

User story: **As an existing user, I want to be able to log into my account.**

Functional requirements:

1. **The system must allow users to log into their account by entering their email and password.**
2. **The system must allow users to log in with their Google accounts.**
3. **The system must allow users to reset their password by clicking on "I forgot my password" and receiving a link to their verified email address.**

Non Functional Requirements Examples

- Each page must load within 2 seconds.
- The process must finish within 3 hours so data is available by 8 a.m. local time after an overnight update.
- The system must meet Web Content Accessibility Guidelines (WCAG 2.1).
- Database security must meet HIPAA (Health Insurance Portability and Accountability Act of 1996) requirements.
- Users shall be prompted to provide an electronic signature before loading a new page.

All of these add more specific restrictions or instructions to what would be functional requirements.

A distinguishing aspect FR vs NFR

Where the functional requirement defines the "what," it often needs a NFR to define the "how."

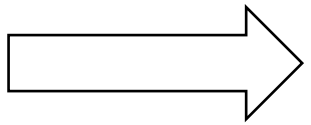
Functional requirement: When an order is fulfilled, the local printer shall print a packing slip.
Non Functional Requirement: Packing slips shall be printed on both sides of 4"x 6" white paper, the standard size for packing slips used by local printers.

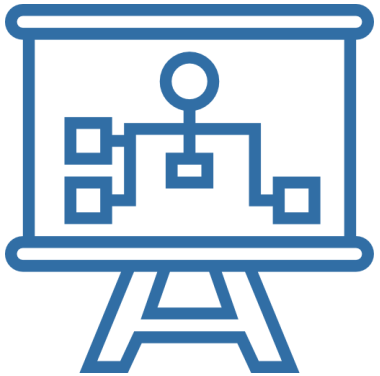
Altexsoft is a AI-driven solution startup that also takes requirement engineering contracts for others. Refer to the following blog to understand and learn more about how to write and identify non-functional requirements: https://www.altexsoft.com/blog/non-functional-requirements/

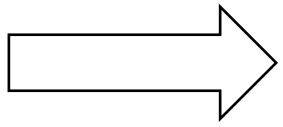
Q: Identify and list out the functional and non-functional requirements for the given problem statement

Milton Jewels has specialized in online jewelry retail since 1998, selling wonderful ranges of both children's and women's jewelry. A customer can register online so that he/she can check the status of the placed order. A customer can purchase any jewelry item online either by using his/her existing account or as an anonymous user specifying shipping address and contact information. Customer can only check the status of his/her order if he/she creates an account. The customer will pay online through credit card or debit card and the order will be delivered on the shipping address within one week.

The requirement analysis model

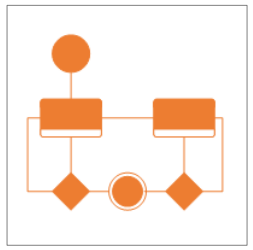


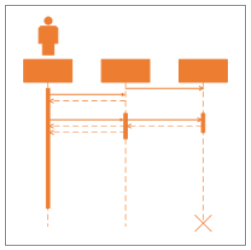


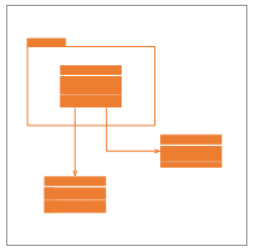


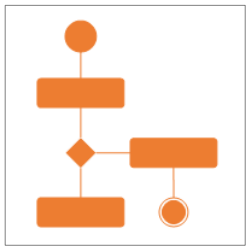
Purpose
Describe what the customer wants to built
Establish the foundation for the software design
Provide a set of validation requirements

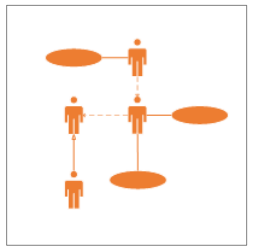
- System Information
- System Function
- System Behaviors











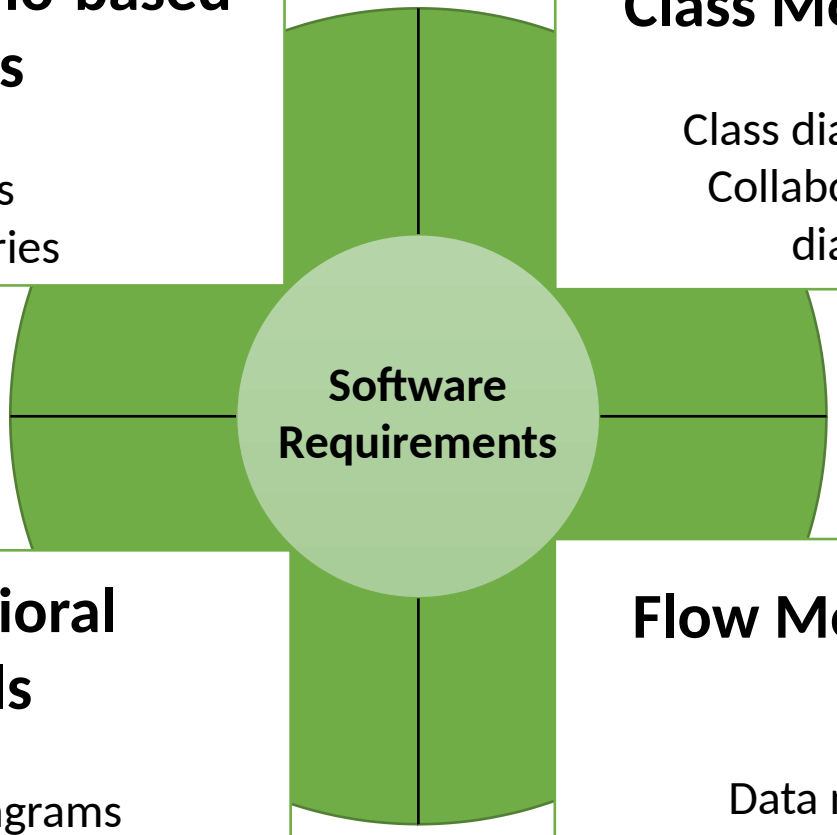
Monologue

1. There are many different ways to look at the requirements for a computer-based system.
2. Some software people argue that it’s best to select one mode of representation (e.g., the use case) and apply it to the exclusion of all other modes.
3. Other practitioners believe that it’s worthwhile to use a number of different modes of representation to depict the analysis model.
4. It is advised to use different modes of representation to consider requirements from different viewpoints—i.e. an approach that has a higher probability of uncovering omissions, inconsistencies, and ambiguity.

Analysis rule of Thumb

- Make **sure all points** of view are **covered**
- Every **element** should **add value**
- **Keep** it **simple**
- **Maintain** a high level of **abstraction**
- **Focus** on the **problem domain**
- **Minimize** system **coupling**
- **Model** should **provide value to all stakeholders**

Elements of the Requirements Model



Elements of the Requirements Model Cont.

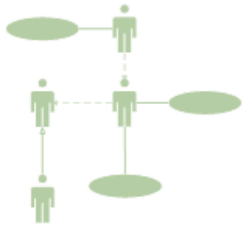
Scenario-based elements

Describe the **system** from the **user's point of view** using scenarios that are depicted (stated) in **use cases** and **activity diagrams**

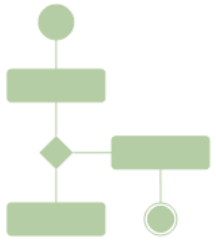
Class-based elements

Identify the **domain classes** for the **objects** manipulated by the actors, the attributes of these classes, and how they interact with one another; which utilize **class diagrams** to do this.

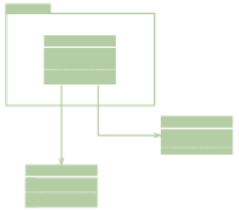
Use Case Diagram



Activity



Class Diagram



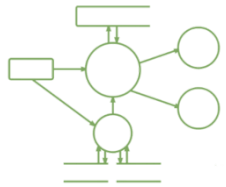
Behavioral elements

Use **state diagrams** to **represent** the **state of the system**, the events that cause the system to change state, and the actions that are taken as a result of a particular event. This can also be applied to each class in the system.

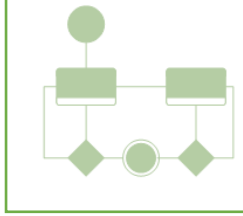
Flow-oriented elements

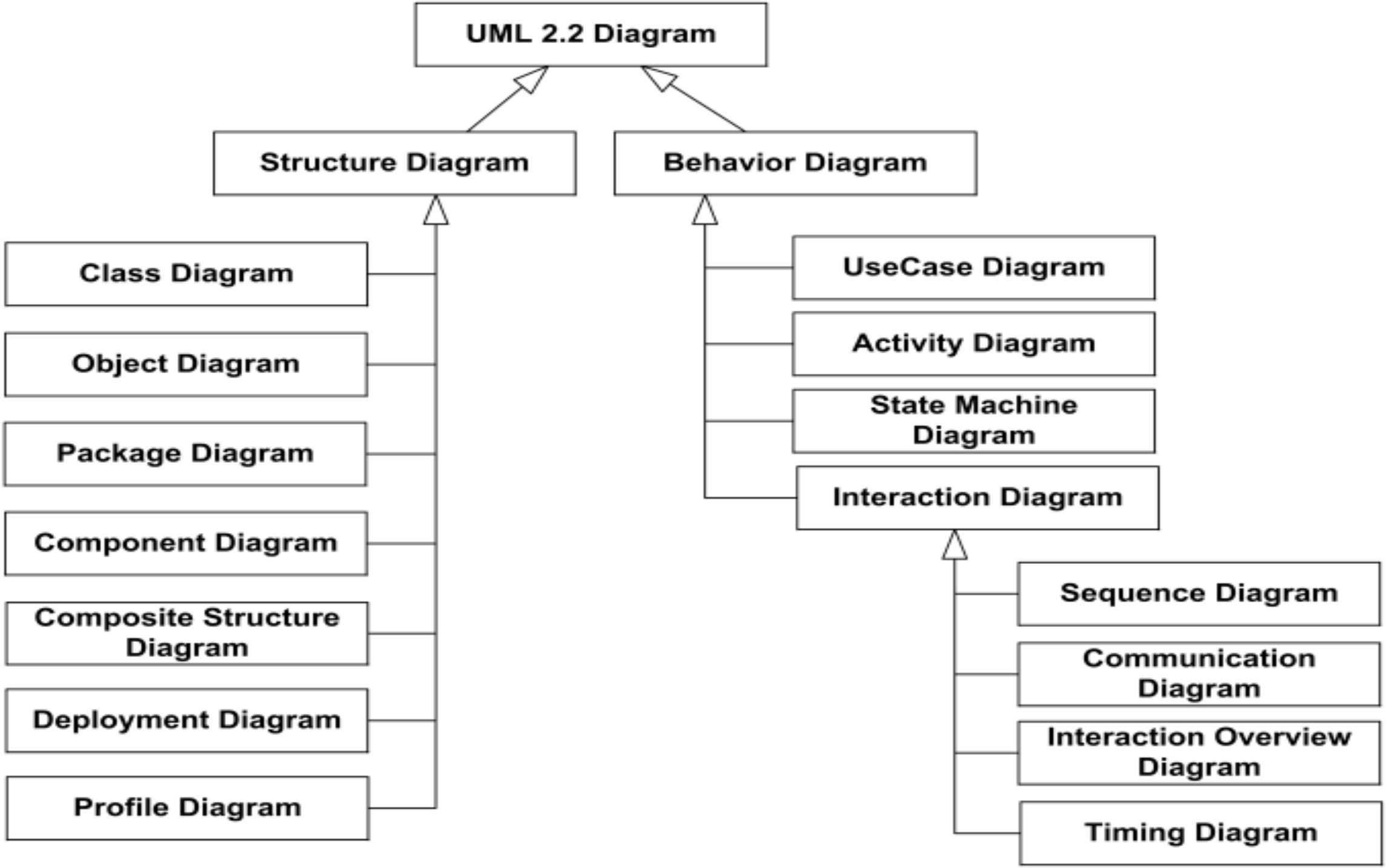
Use **data flow diagrams** to **show** the **input** data that comes into a system, what **functions** are **applied** to that data to do transformations, and what resulting **output** data are produced.

Data Flow Diagram



State Diagram




UML 2.2 Diagram
Structure Diagram
Behavior Diagram
Class Diagram
Object Diagram
Package Diagram
Component Diagram
Composite Structure Diagram
Deployment Diagram
Profile Diagram
UseCase Diagram
Activity Diagram
State Machine Diagram
Interaction Diagram
Sequence Diagram
Communication Diagram
Interaction Overview Diagram
Timing Diagram

SRS (Software Requirements Specification)

Software Requirement Specification (SRS) is a document that completely describes what the proposed software should do without describing how software will do it.

SRS is also helping the clients to understand their own needs.

SRS Contains

A complete information description

A detailed functional description

A representation of system behaviour

An indication of performance requirements and design constraints

Appropriate validation criteria

Other information suitable to requirements

Characteristics of a Good SRS

- SRS should be accurate, complete, efficient, and of high quality, so that it does not affect the entire project plan.
- An SRS is said to be of high quality when the developer and user easily understand the prepared document.
- Characteristics of a Good SRS:

Correct

SRS is correct when all user requirements are stated in the requirements document.

Note that there is no specified tool or procedure to assure the correctness of SRS.

Unambiguous

SRS is unambiguous when every stated requirement has only one interpretation.

Complete

SRS is complete when the requirements clearly define what the software is required to do.

Characteristics of a Good SRS Cont.

Ranked for Importance/Stability

All requirements are not equally important, hence each requirement is identified to make differences among other requirements.

Stability implies the probability of changes in the requirement in future.

Modifiable

The requirements of the user can change, hence requirements document should be created in such a manner that those changes can be modified easily.

Traceable

SRS is traceable when the source of each requirement is clear and facilitates the reference of each requirement in future.

Verifiable

SRS is verifiable when the specified requirements can be verified with a cost-effective process to check whether the final software meets those requirements.

Consistent

SRS is consistent when the subsets of individual requirements defined do not conflict with each other.

Problems Without SRS

- Without developing the SRS document, the system would not be properly implemented according to customer needs.
- Software developers would not know whether what they are developing is what exactly is required by the customer.
- Without SRS, it will be very difficult for the maintenance engineers to understand the functionality of the system.
- It will be very difficult for user document writers to write the users' manuals properly without understanding the SRS.

Standard Template for writing SRS

1. Introduction

- 1.1 Purpose
- 1.2 Document Conventions
- 1.3 Intended Audience and 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 and Implementation Constraints
- 2.6 User Documentation
- 2.7 Assumptions and Dependencies

3. System Features

- 3.1 System Feature 1
- 3.2 System Feature 2 (and so on)

4. External Interface Requirements

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

5. Other Non-functional 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

Thank You