BBM384 SOFTWARE ENGINEERING LABORATORY INTRODUCTION

R.A. BURCU YALÇINER
R.A. FEYZA NUR KILIÇASLAN

CLASS SCHEDULE

	BBM384				Deliverables (see below)
Week #	**Date**	**Content**	**Ch's**	**Other References**	**Return**
1	February 24, 2020	Introduction	1.2	SDLC & Dev.Schedule	
2	March 2, 2020	UML & Tool Intro, System Description	2.3	Sw.Vision & Prj.Plan Temp.s	
3	March 9, 2020	UML Modeling (Context, Use Case, Activity D.)	4	SRS Template	
4	March 16, 2020	Example Requirements Modeling with UML	4	Sw.Vision & Prj.Plan Examples	DEL #1 (on March 14)
5	March 23, 2020	UML Modeling (Package, Component, Deployment D.)	4	SRS example (UCD, GUI, Data model)	
6	March 30, 2020	Example Arch. & HL Design Modeling with UML	5.6	Sys.Test Case Temp, Arch.Notebook Temp.	DEL #2
7	April 6, 2020	UML Modeling (Statechart, Class, Sequence D.)	7	Example Coding Std.	DEL #3
8	April 13, 2020	Example Design & Impl. Modeling with UML	8	SDD Template	demo
9	April 20, 2020	demo1 - DEL3	9	Sys.Test Report Temp.	
10	April 27, 2020	No class			
11	May 4, 2020	demo2 - DEL4			DEL #4
12	May 11, 2020	Project presentations			demo
13	May 18, 2020	Project presentations			
14	May 25, 2020	Project presentations			DEL #5

GRADING

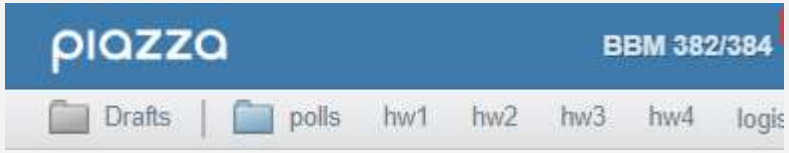
Project deliveries	Documentation	Software / Code	Mark %	Total milestone mark
DEL #1	Software Vision		4%	10%
	Project Plan		6%	
DEL #2	Software Requirements Document		15%	15%
DEL #3	Architectural Notebook		5%	20%
	List of System Test Case Definitions		3%	
		Prototype 1: demo of a single use case	5%	
	Risks management report (see tab "Writing risk mng report")		3%	
	Configuration/change management report (see tab "Writing change mng report")		4%	
DEL #4	Software Design document (UML models)		10%	20%
	Coding Standard		2%	
		Prototype 2: demo at least 6 use cases (half of system)	8%	
DEL #5	Software Test RESULT Report		5%	35%
	Risks management report (see tab "Writing risk mng report")		5%	
	Configuration/change management report (see tab "Writing change mng report")		5%	
	Presentation (EACH STUDENT should say what s/he has done)		5%	
		Release: Final demo of software product (all use cases)	15%	
			100%	100%

GRADING

A minimum of 80% attendance to laboratory sessions are compulsory

COMMUNICATION

The course excel sheet will be updated regularly throughout the semester with lecture notes, laboratory notes, reading assignments, templates and important deadlines. All other communications will be carried out through Piazza. Please enroll it by following the links



The link of piazza

https://piazza.com/hacettepe.edu.tr/spring2020/bbm382/384

The Office hours of research assistants will be announced later

The link of the course excel sheet

https://docs.google.com/spreadsheets/d/1SvdSWxMNS6vWVwkJ-3cblakiNIrvuvsQayM3golS4Gg/edit?ts=5e4fc127#gid=1520303974

LABORATORY GROUPS

Students should form groups of 5 people and let us know it via the link below until March 7 Thursday

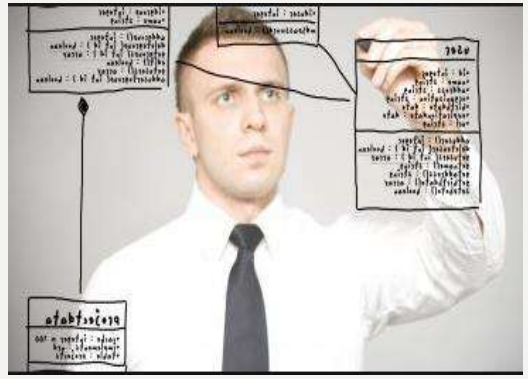
https://drive.google.com/open?id=1932XG4woZwsJDA3VHFViaUrfexrO5xUYwRhBRXHKhRo

SPECIALIZED TEAM ROLES

Software Project Manager

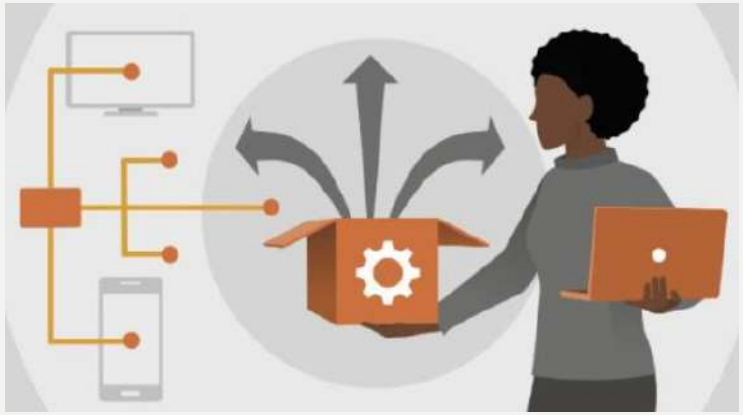
In addition to

Software Developer



Software Architect

Software Analyst



Software Configuration Manager

Software Tester

THE SOFTWARE DEVELOPER

Software Developer duties and responsibilities

- Researching, designing, implementing and managing software programs
- Testing and evaluating new programs
- Identifying areas for modification in existing programs and subsequently developing these modifications
- Writing and implementing efficient code
- Determining operational practicality
- Developing quality assurance procedures
- Deploying software tools, processes and metrics
- Maintaining and upgrading existing systems
- Training users
- Working closely with other developers, UX designers, business and systems analysts

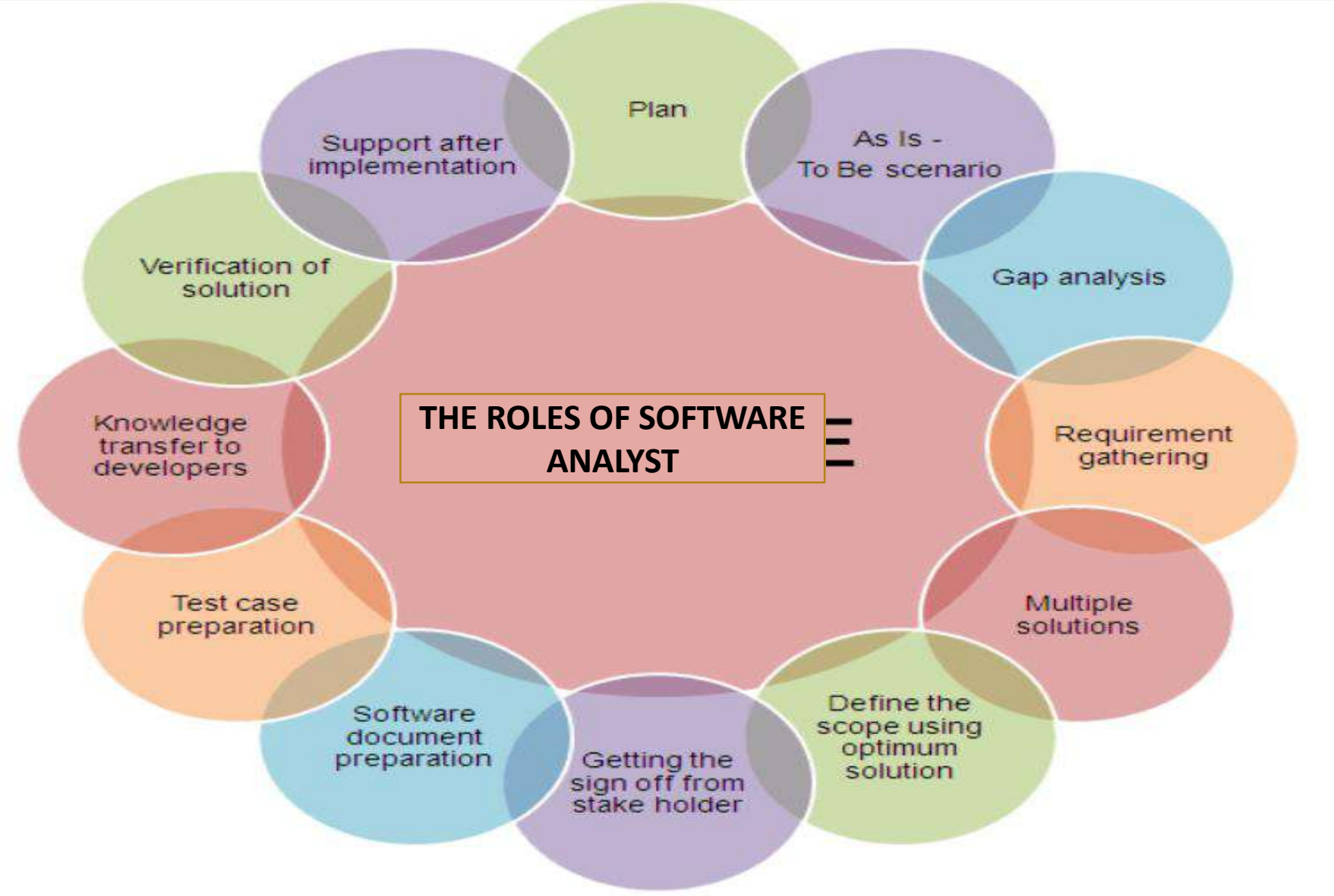
THE SOFTWARE PROJECT MANAGER

Responsibilities of a Project Manager

- Activity and resource planning
- Organizing and motivating a project team
- Controlling time management
- Cost estimating and developing the budget
- Ensuring customer satisfaction
- Analyzing and managing project risk
- Monitoring progress
- Managing reports and necessary documentation

THE SOFTWARE ARCHITECT?

THE SOFTWARE ANALYST



THE CONFIGURATION MANAGER?

- Requirement Documents
- Design Documents
- Test Documents
- Source Code
- Executables
- Databases
- Test Data
- Bug Reports
- Build
- Servers

Identify configuration items

Define policies and procedures for change management

Define versioning, baselining, build and release procedures

Prepare configuration management plan

Configuration control

Configuration audit

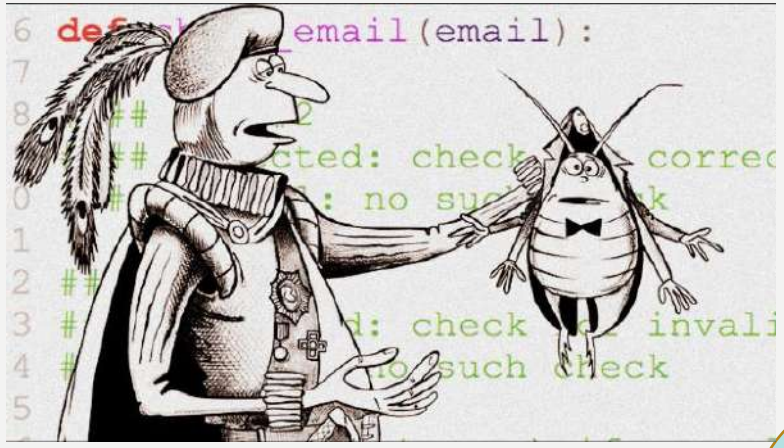
Configuration reporting

THE SOFTWARE TESTER

ROLES AND RESPONSIBILITIES

Executing tests on software usability

Reviewing software requirements and preparing test scenarios



Analyzing test results on database impacts, errors or bugs, and usability

Preparing reports on all aspects related to the software testing carried out and reporting to the design team

Participating in design reviews and providing input on requirements, product design, and potential problems

Interacting with clients to understand product requirements


THANK YOU
for your
ATTENTION!

UML & Tool Introduction

Context

- Introduction of the Project and Rolles
- UML Introduction
- UML Diagram Types
- UML Tool Introduction
- About Vision and Project Plan Templates (with OpenUp)

About the Project

- Sports Center Membership System (SCMS)

UML

- The Unified Modeling Language (UML) is a graphical notation for drawing diagrams of software concepts.
- A language for specifying, visualizing, constructing, and documenting the artifacts of software systems
- *Conceptual (a problem domain)*, *Specification (a proposed software design)*, and *Implementation (software implementation)*

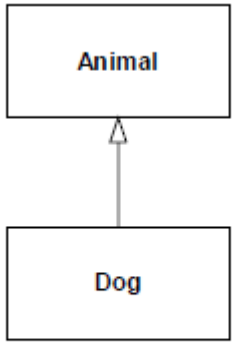


Figure 1-1
A Dog is an Animal

```
public class Animal {}
Public class Dog extends Animal {}
```

UML – The Basics

- Abstraction (A simplification or model of a complex concept, process, or real-World object – Help people understand something at an appropriate level)
- Encapsulation (Higlight the important aspects of an object – Hide the cumbersome internal details of the object – make the system easier to understand and reuse – make a system more extendible)
- Entities (object, class) and relationships between objects

UML Diagram Types

- UML has three main kinds of diagrams
  - Static diagrams describe the unchanging logical structure of software elements by depicting classes, objects, and data structures; and the relationships that exist between them.
  - Dynamic diagrams show how software entities change during execution by depicting the flow of execution, or the way entities change state.
  - Physical diagrams show the unchanging physical structure of software entities by depicting physical entities such as source files, libraries, binary files, data files, etc., and the relationships that exist between them.

UML Diagram Types

- The current UML standards call for 13 different types of diagrams
- These diagrams are organized into two distinct groups: structural diagrams and behavioral or interaction diagrams.

Structural UML diagrams

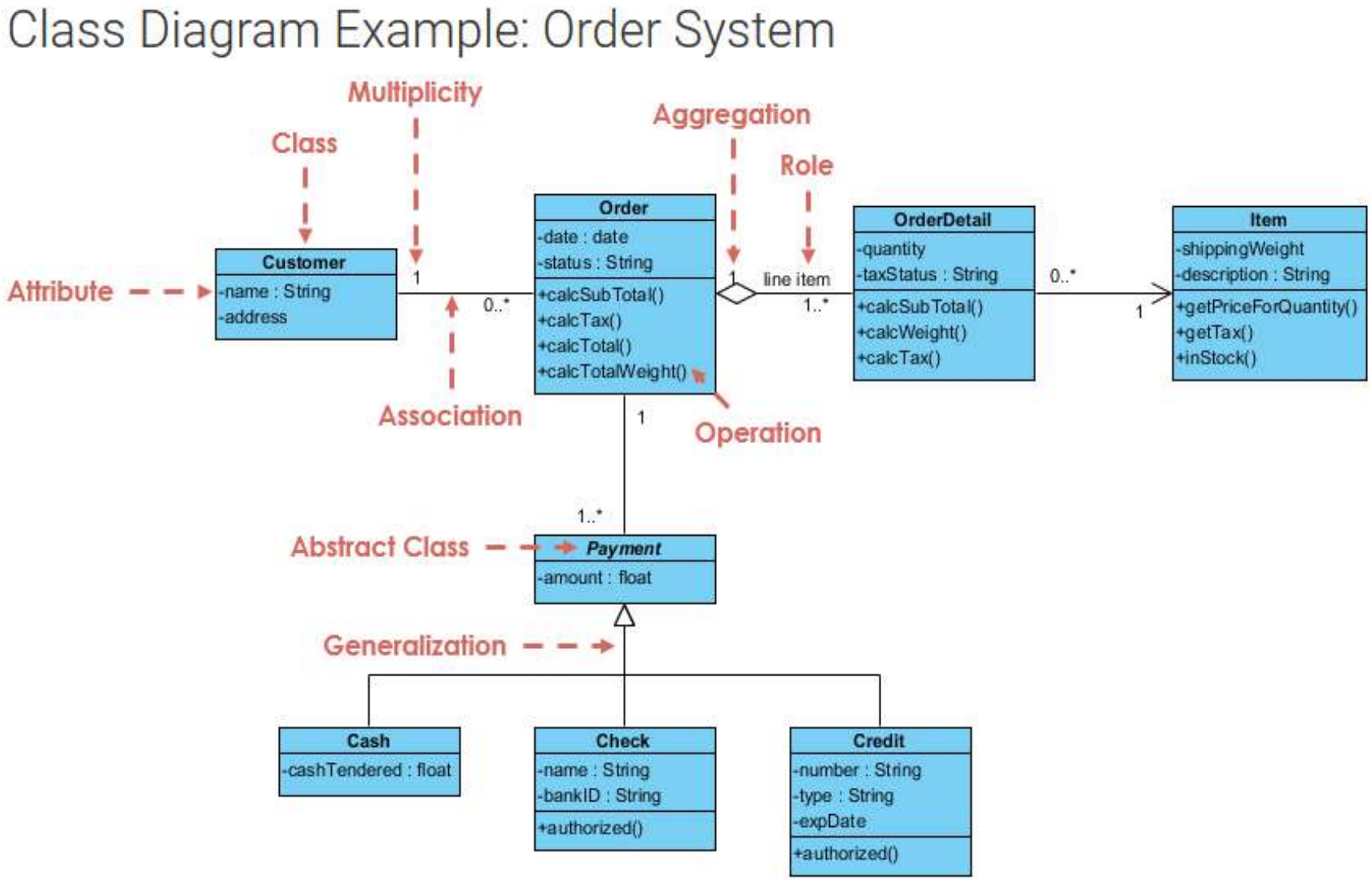
- **Class diagram**
- Package diagram
- Object diagram
- Component diagram
- Composite structure diagram
- Deployment diagram

Behavioral UML diagrams

- **Activity diagram**
- Sequence diagram
- **Use case diagram**
- State diagram
- Communication diagram
- Interaction overview diagram
- Timing diagram

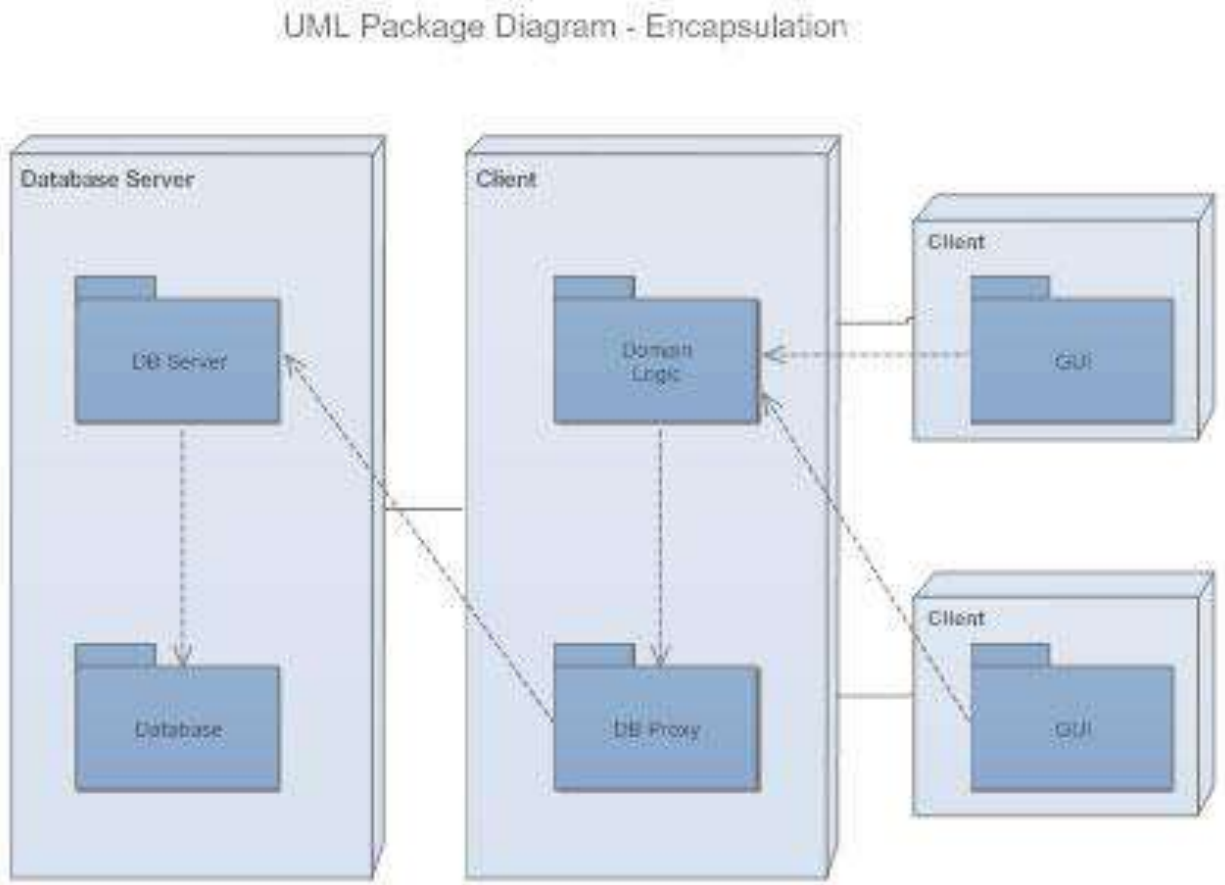
UML Diagram Types – Class Diagram

A class diagram models the static structure of a system. It shows relationships between classes, objects, attributes, and operations.



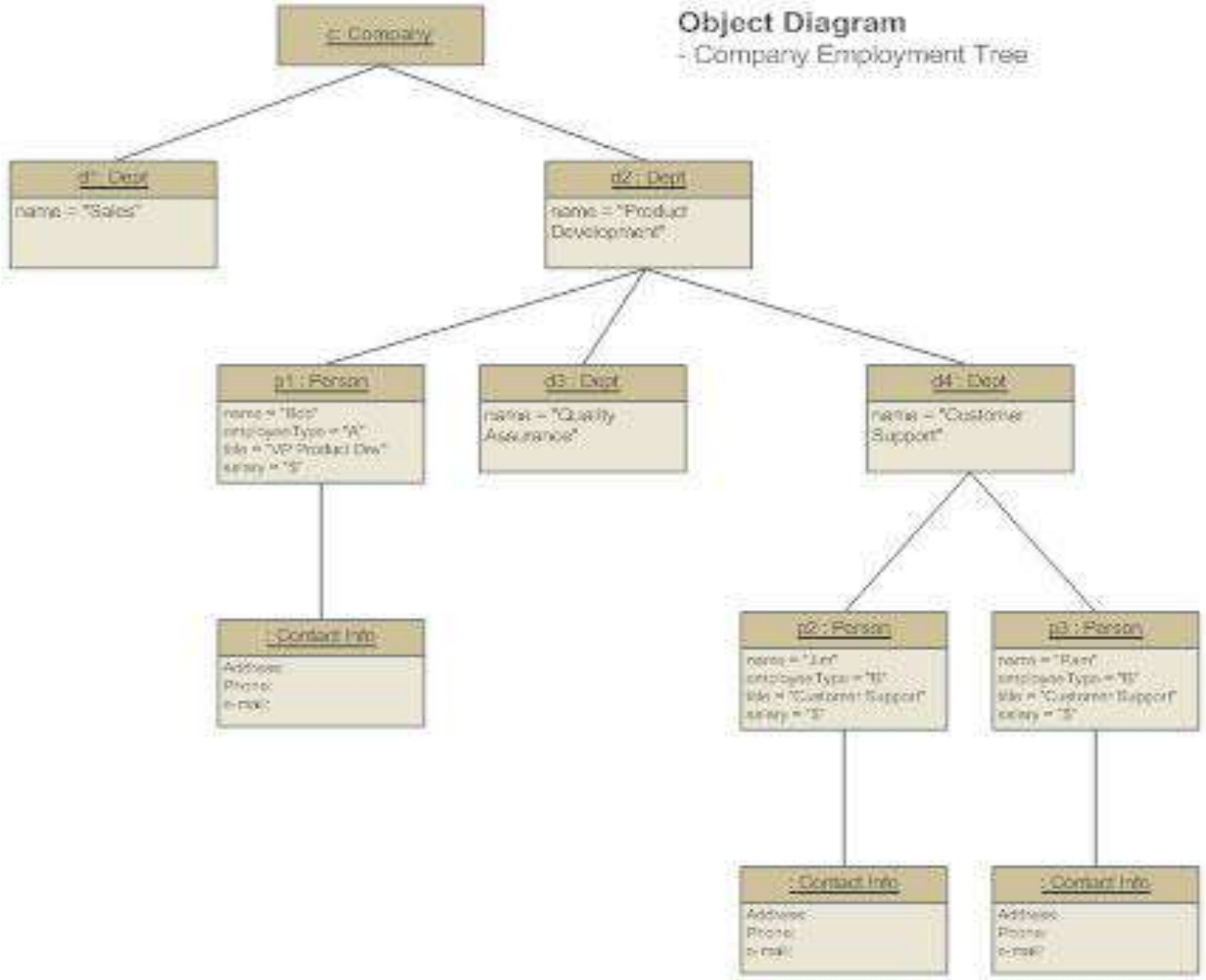
UML Diagram Types – Package Diagram

Package diagrams are a subset of class diagrams, but developers sometimes treat them as a separate technique. Package diagrams organize elements of a system into related groups to minimize dependencies between packages.



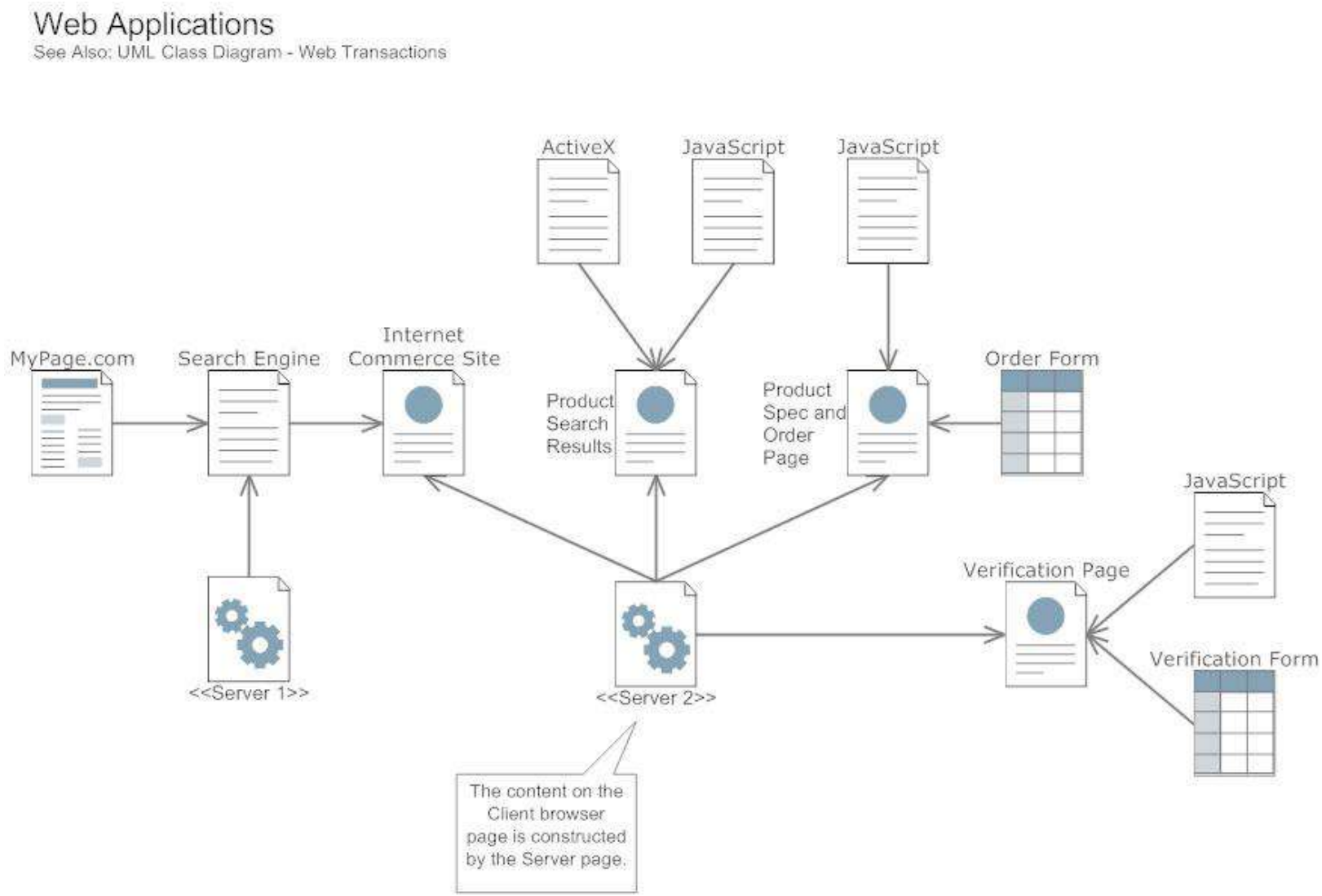
UML Diagram Types – Object Diagram

Object diagrams describe the static structure of a system at a particular time. They can be used to test class diagrams for accuracy.



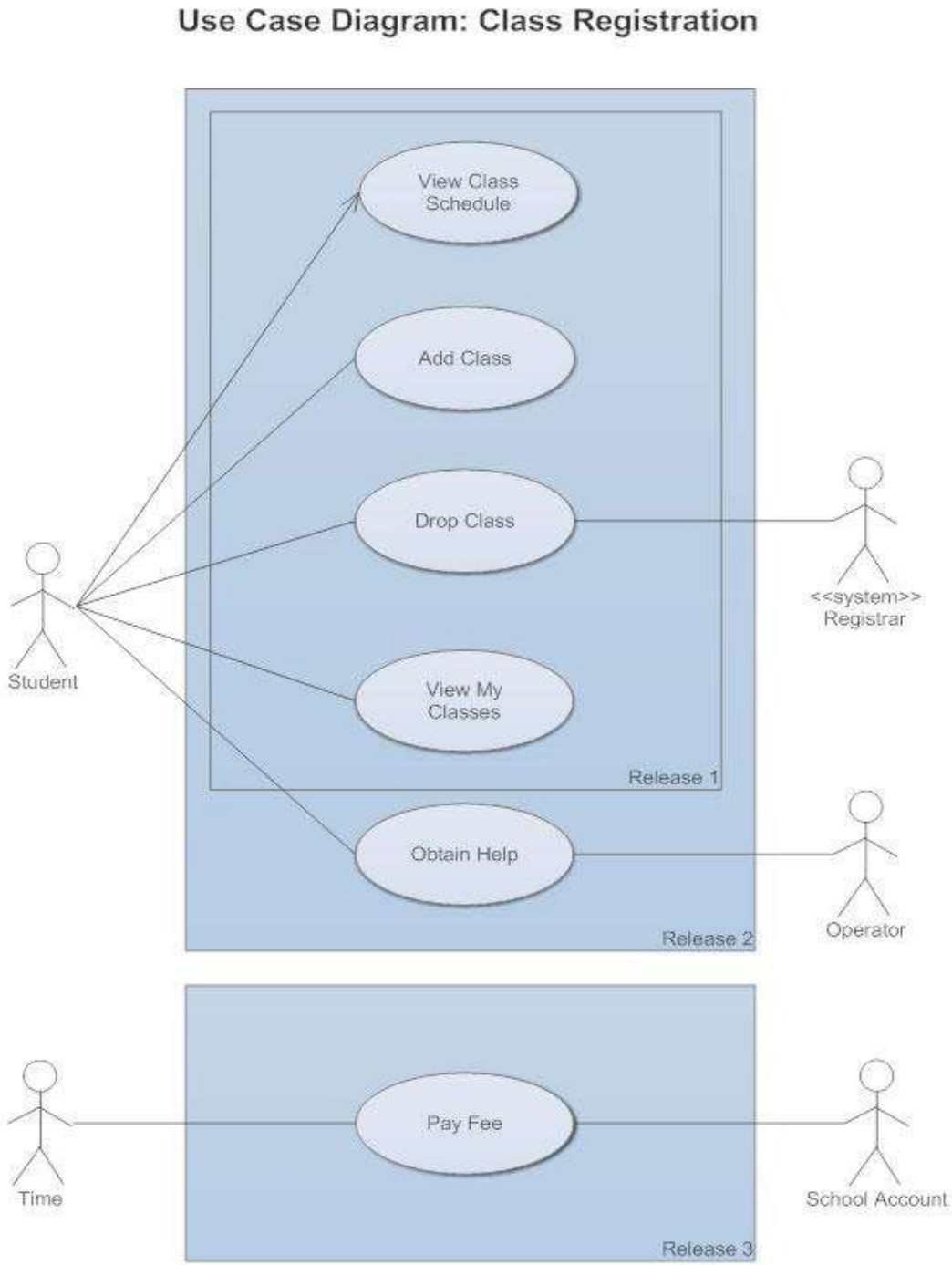
UML Diagram Types – Component Diagram

Component diagrams describe the organization of physical software components, including source code, run-time (binary) code, and executables.



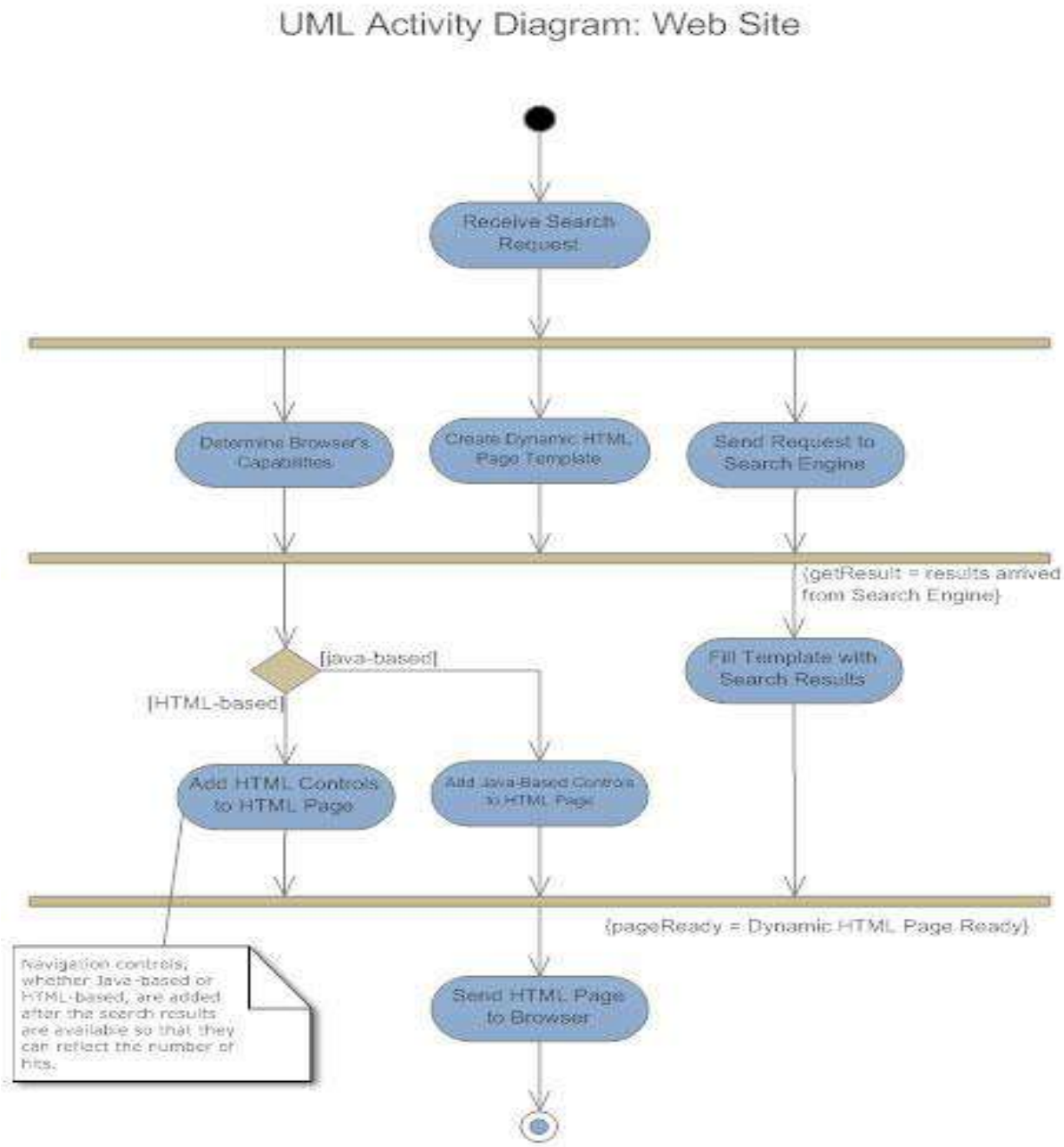
UML Diagram Types – UseCase Diagram

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform.



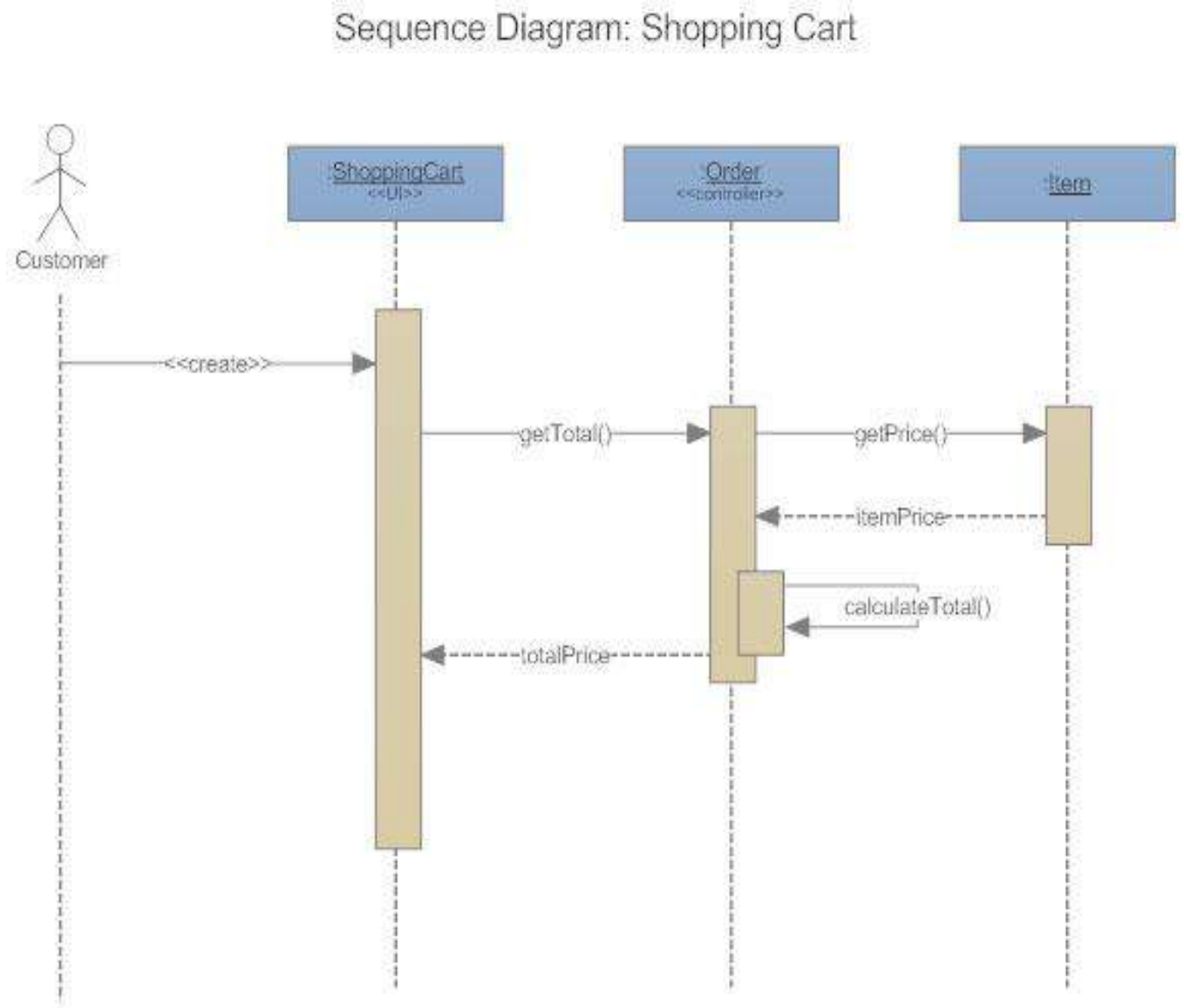
UML Diagram Types – Activity Diagram

Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation



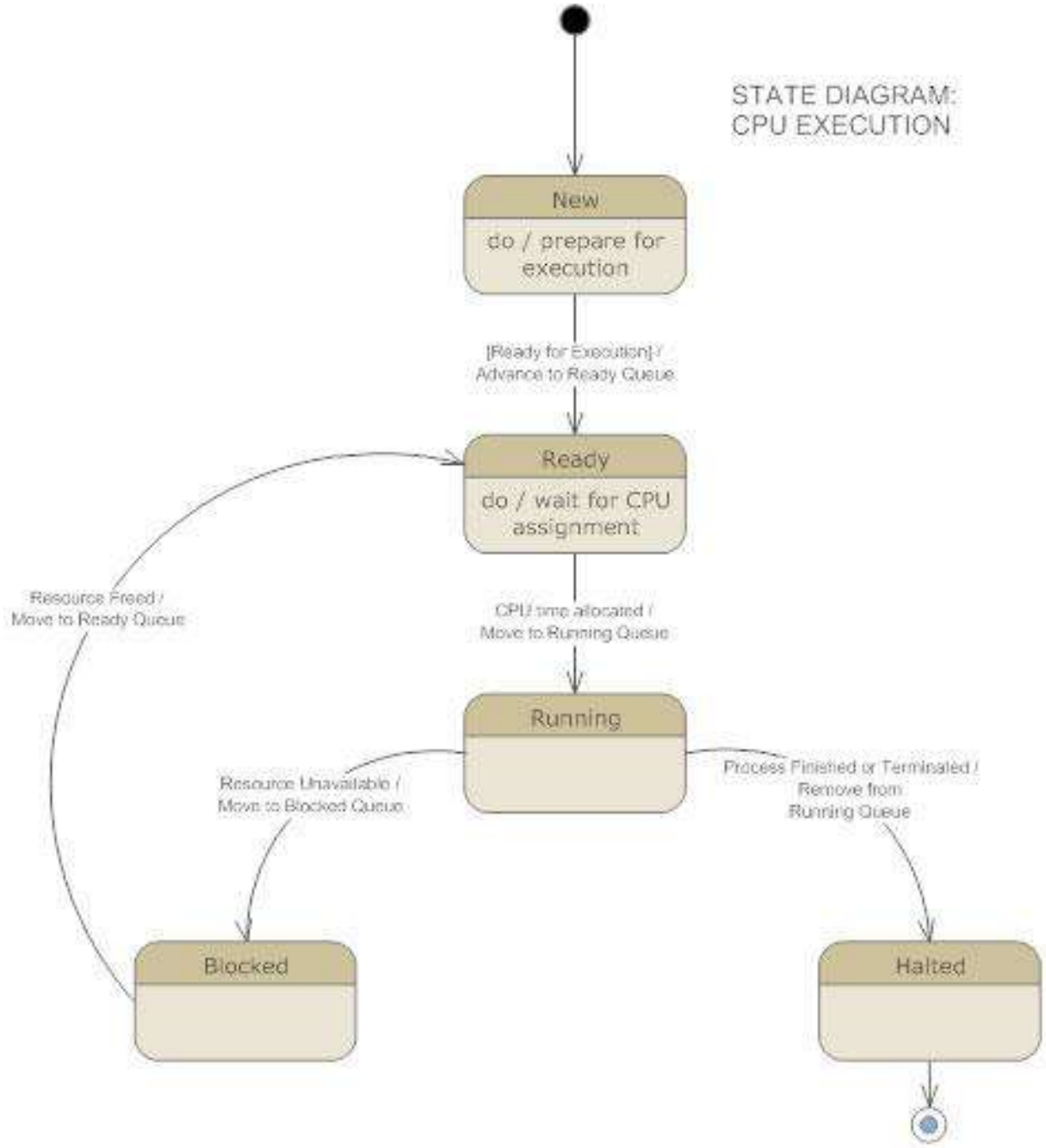
UML Diagram Types – Sequence Diagram

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time



UML Diagram Types – State Diagram

Statechart diagrams, now known as state machine diagrams and state diagrams describe the dynamic behavior of a system in response to external stimuli. State diagrams are especially useful in modeling reactive objects whose states are triggered by specific events



UML Tool

- Visual Paradigm Community Edition https://www.visual-paradigm.com/download/community.jsp
  - Visual Paradigm is a powerful, cross-platform and yet easy-to-use design and management tool for IT systems. Visual Paradigm provides software developers the cutting edge development platform to build quality applications faster, better and cheaper.
- SmartDraw https://www.smartdraw.com/
- MS Visio

Vision and Project Plan Templates (with OpenUp)

- Vision
  - http://epf.eclipse.org/wikis/openup/core.tech.common.extend_supp/guidances/templates/resources/vision_tpl.dot
  - This artifact defines the view of the stakeholders of the technical solution to be developed. This definition is specified in terms of the key needs and features of the stakeholders. The vision contains an outline of the envisioned core requirements for the system.

Vision and Project Plan Templates (with OpenUp)

- Project Plan Template
  - http://epf.eclipse.org/wikis/openup/practice.mgmt.release_planning.base/guidances/templates/resources/project_plan_tpl.dot
  - A collaborative task that outlines an initial agreement on how the project will achieve its goals. The resulting project plan provides a summary-level overview of the project.

Product of the Lab. Project

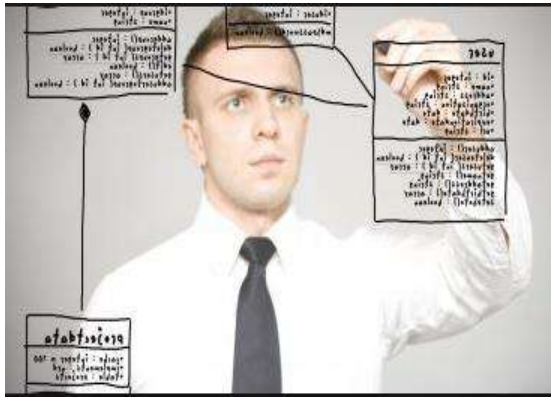
- **System to be developed:**

Sports Center Membership System (SCMS)

Specialized Team Roles

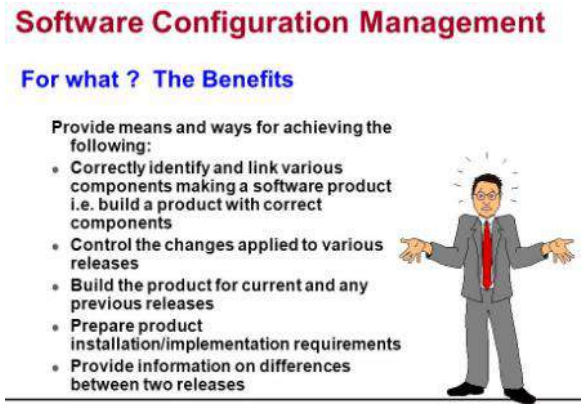
Software Project Manager

In addition to
Software Developer



Software Architect

Software Analyst

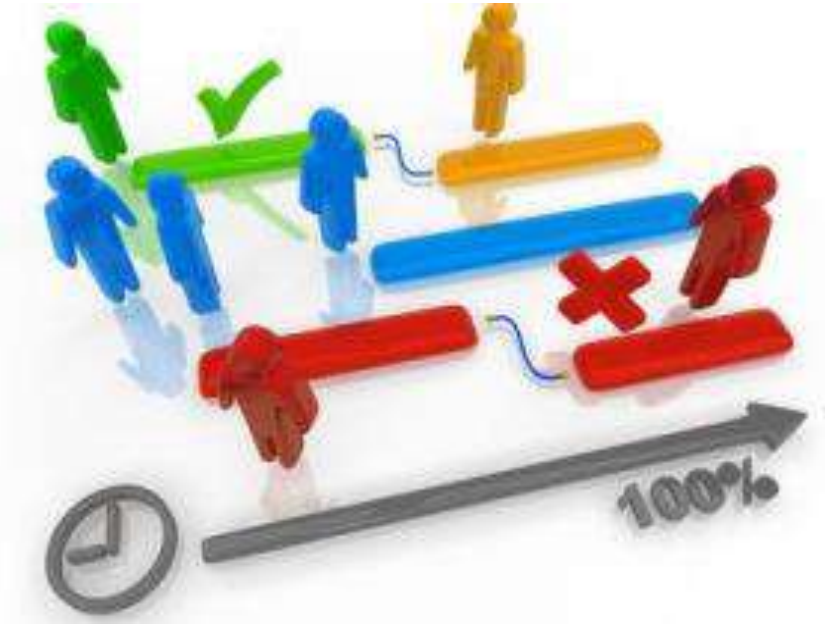


Software Configuration Manager

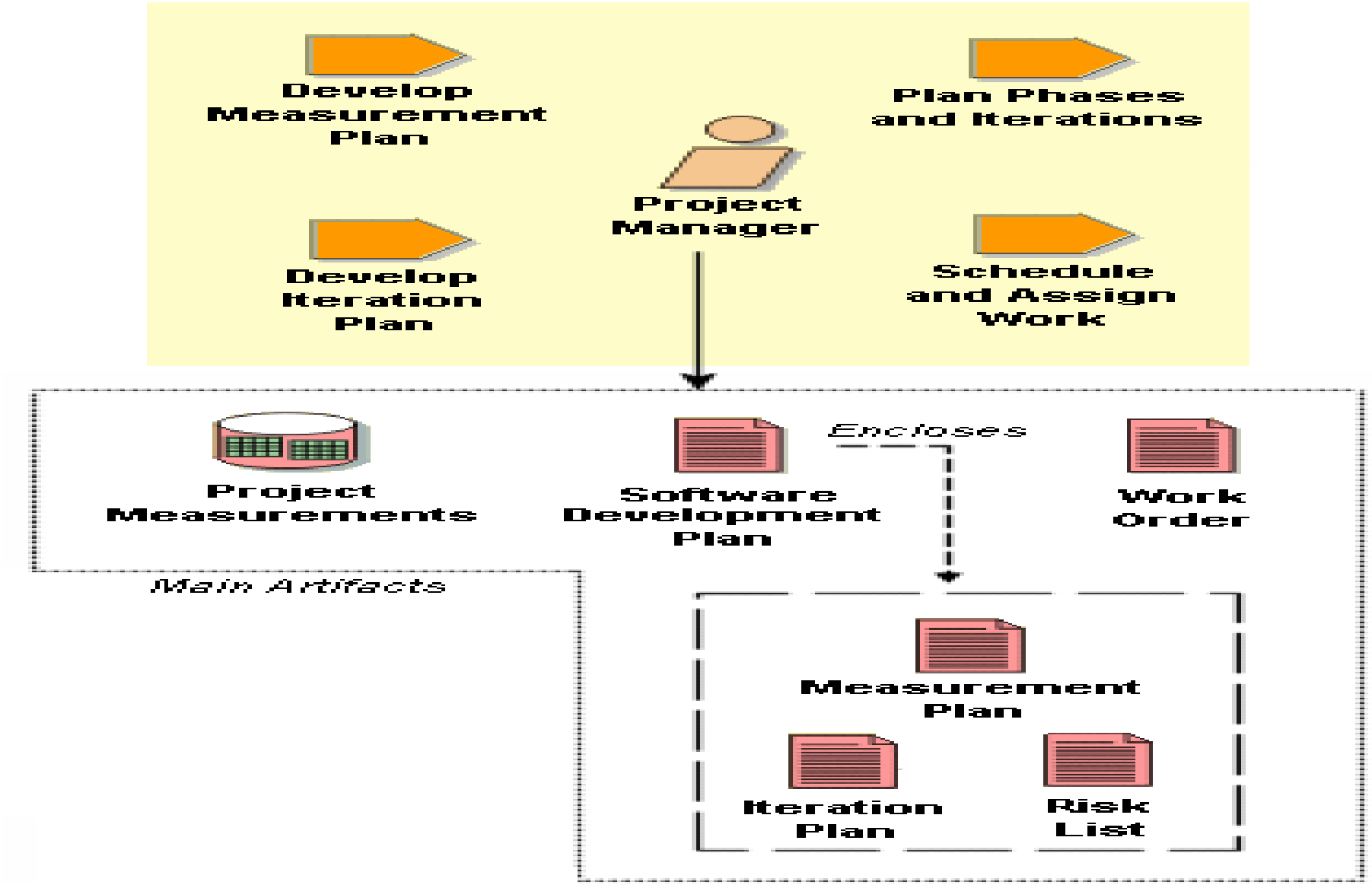
Software Tester

Project Manager

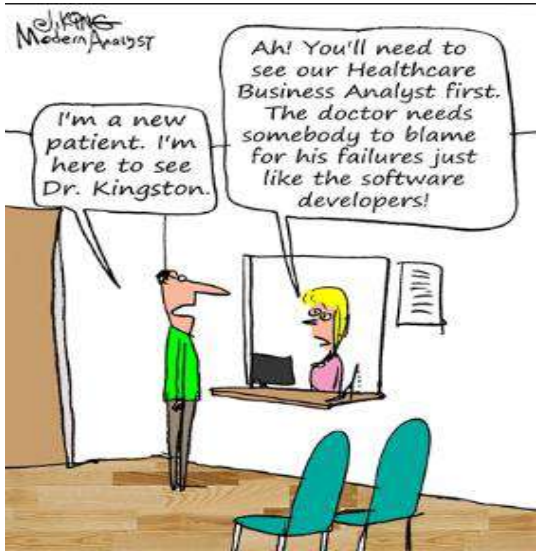
- The **project manager role** allocates resources, shapes priorities, coordinates interactions with customers and users, and generally keeps the project team focused on the right goal. The project manager also establishes a set of practices that ensure the integrity and quality of project artifacts.
- **<u>Responsibilities of a Project Manager</u>**
- **Planning** – scope, activity schedules, gantt chart, potential risks etc.
- **Setting goals -** For example: Complete the project within six months from start date in the budget of xxx amount.
- **Time Management**
- **Budget Allocation and Cost Estimates**
- **Implementation and Monitoring**



Project Manager

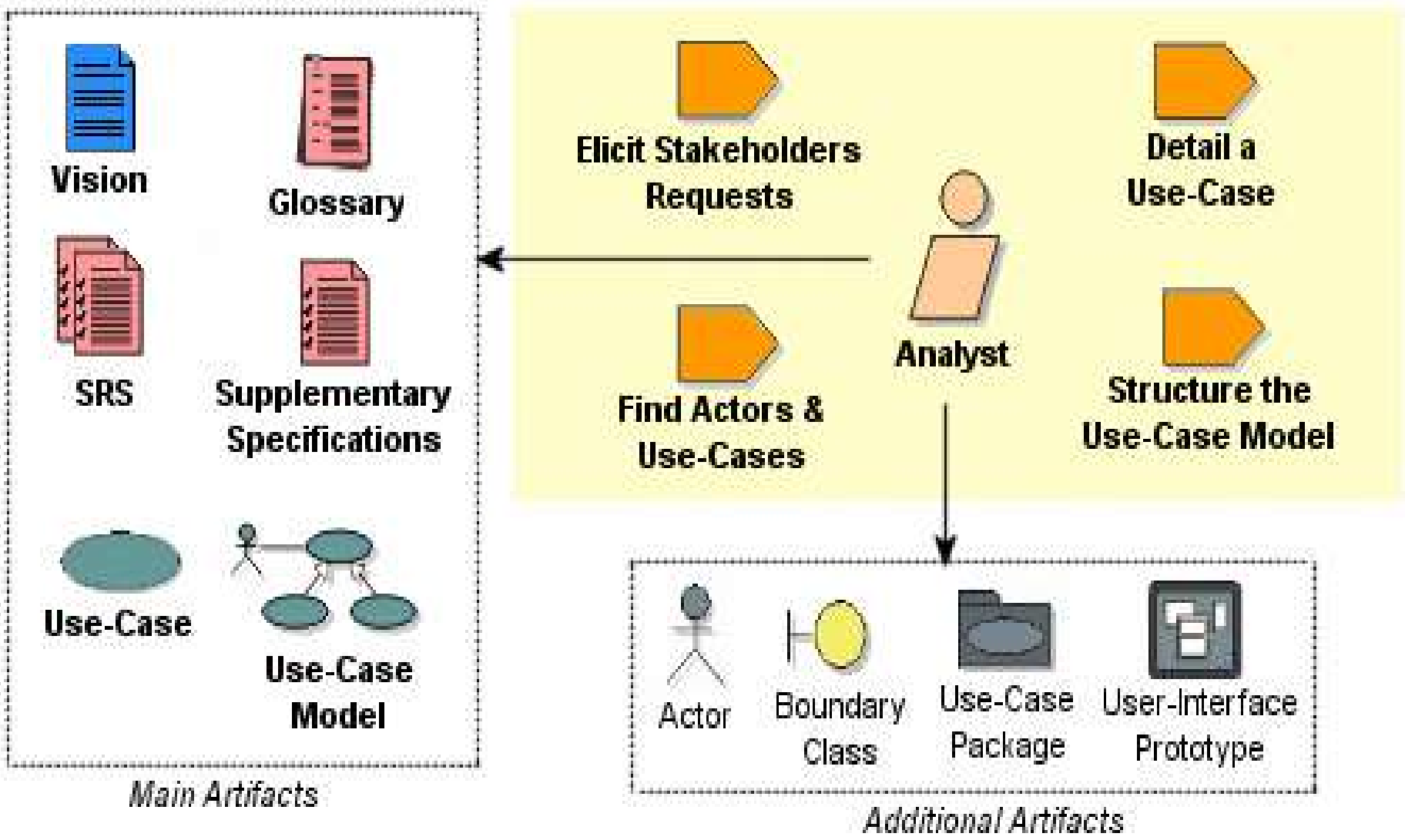


Software Analyst



- In a software development team, a **software analyst** is the person who studies the software application domain, prepares software requirements, and specification (Software Requirements Specification) documents.
The software analyst is the seam between the software users and the software developers.
- S/he is responsible for
  - Performing complex analysis, designing and programming to meet business requirements.
  - Maintaining, managing and modifying all software systems and applications.
  - Defining specifications for complex software programming applications.
  - Interfacing with end-users and software consultants.
  - Developing, maintaining and managing systems, software tools and applications.
  - Resolving complex issues relating to business requirements and objectives.
  - Coordinating and supporting software professionals in installing and analyzing applications and tools.
  - Analyzing, developing and implementing testing procedures, programming and documentation.
  - Training and developing other software analysts.
  - Analyzing, designing and developing modifications and changes to existing systems to enhance performance.

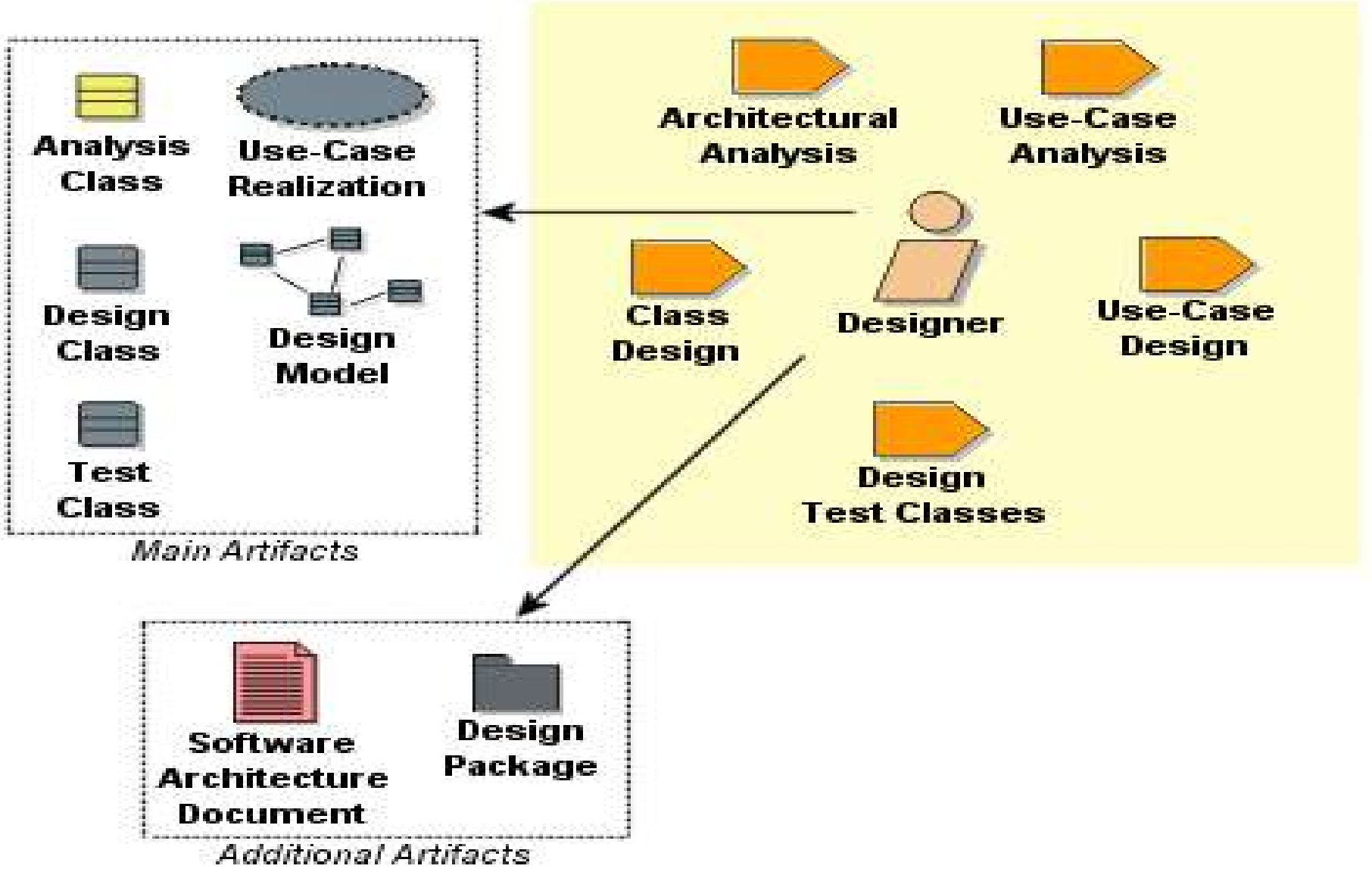
Software Analyst



Software Architect

- A **software architect/designer** is a software expert who makes high-level design choices and dictates technical standards, including software coding standards, tools, and platforms.
- Examples of Software Architect responsibilities
  - Design, develop and execute software solutions to address business issues.
  - Provide architectural blueprints and technical leadership to our IT team.
  - Evaluate and recommend tools, technologies and processes to ensure the highest quality product platform.
  - Collaborate with peer organizations, quality assurance and end users to produce cutting-edge software solutions.
  - Interpret business requirements to articulate the business needs to be addressed.
  - Troubleshoot code level problems quickly and efficiently.

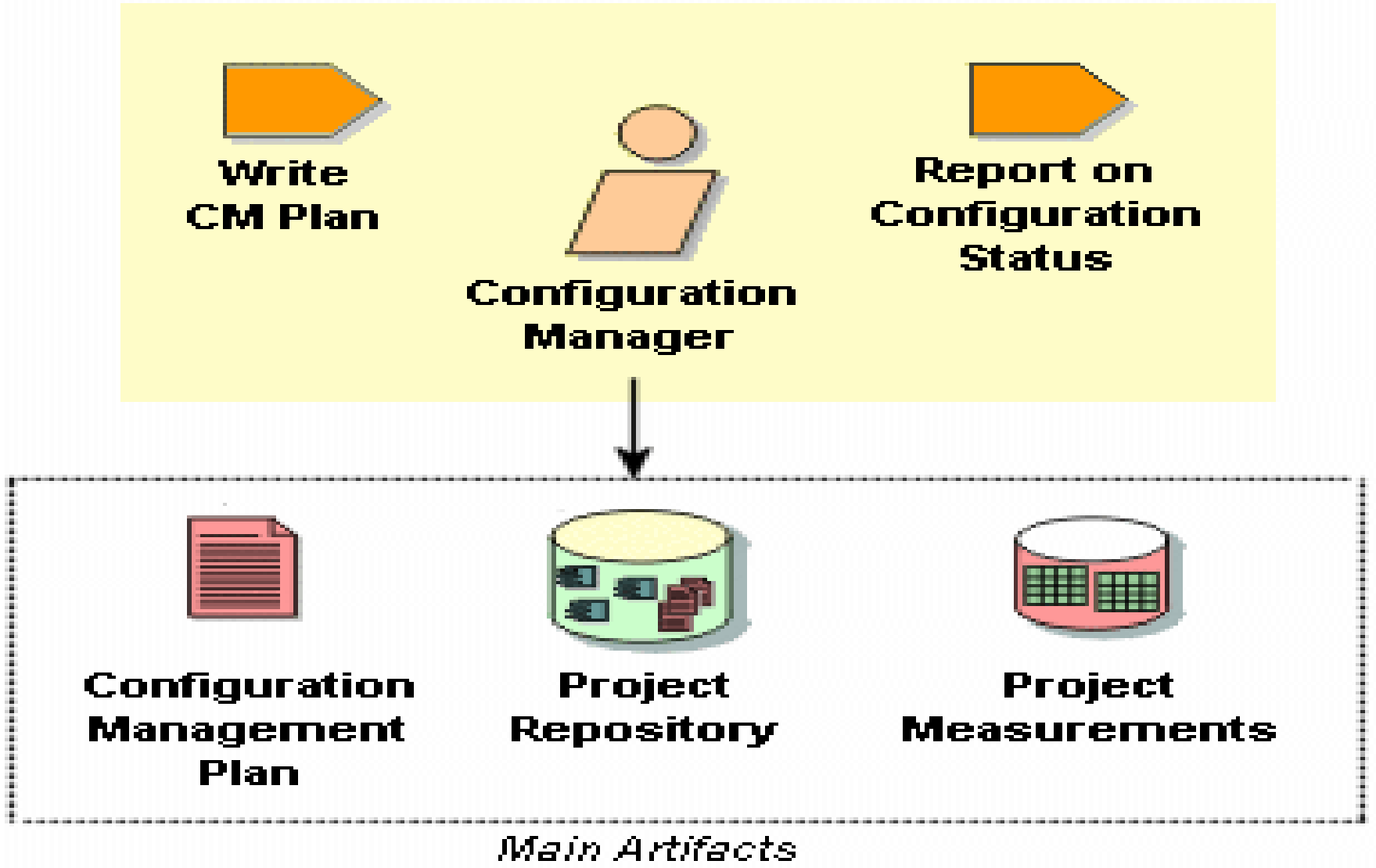
Software Architect



Software Configuration Manager

- The **configuration manager** provides the overall Configuration Management (CM) infrastructure and environment to the product development team. The CM role supports the product development activity so that developers and integrators have appropriate workspaces to build and test their work, and so that all artifacts are available for inclusion in the deployment unit as required. The configuration manager also has to ensure that the CM environment facilitates product review, and change and defect tracking activities. The configuration manager is also responsible for writing the CM Plan and reporting progress statistics based on change requests.

Software Configuration Manager



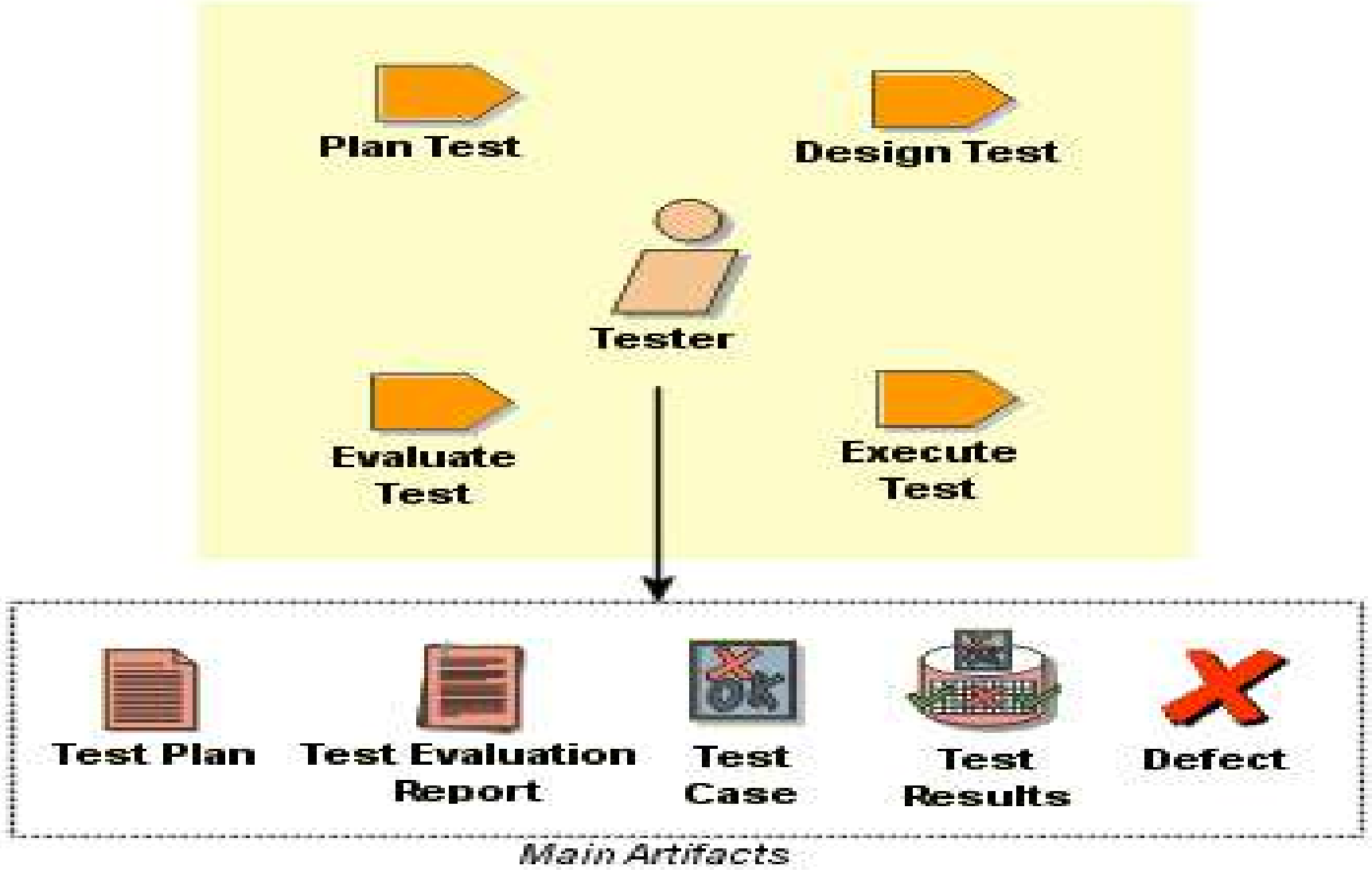
Software Tester

- A **software tester** is an individual that tests software for bugs, errors, defects or any problem that can affect the performance of computer software or an application.
- Software testers are part of a software development team and perform functional and non-functional testing of software using manual and automated software testing techniques.
- S/he is responsible for
  - Identifying the most appropriate implementation approach for a given test
  - Implementing individual tests
  - Setting up and executing the tests
  - Logging outcomes and verifying test execution
  - Analyzing and recovering from execution errors

Some of the techniques software testers must have experience with include:

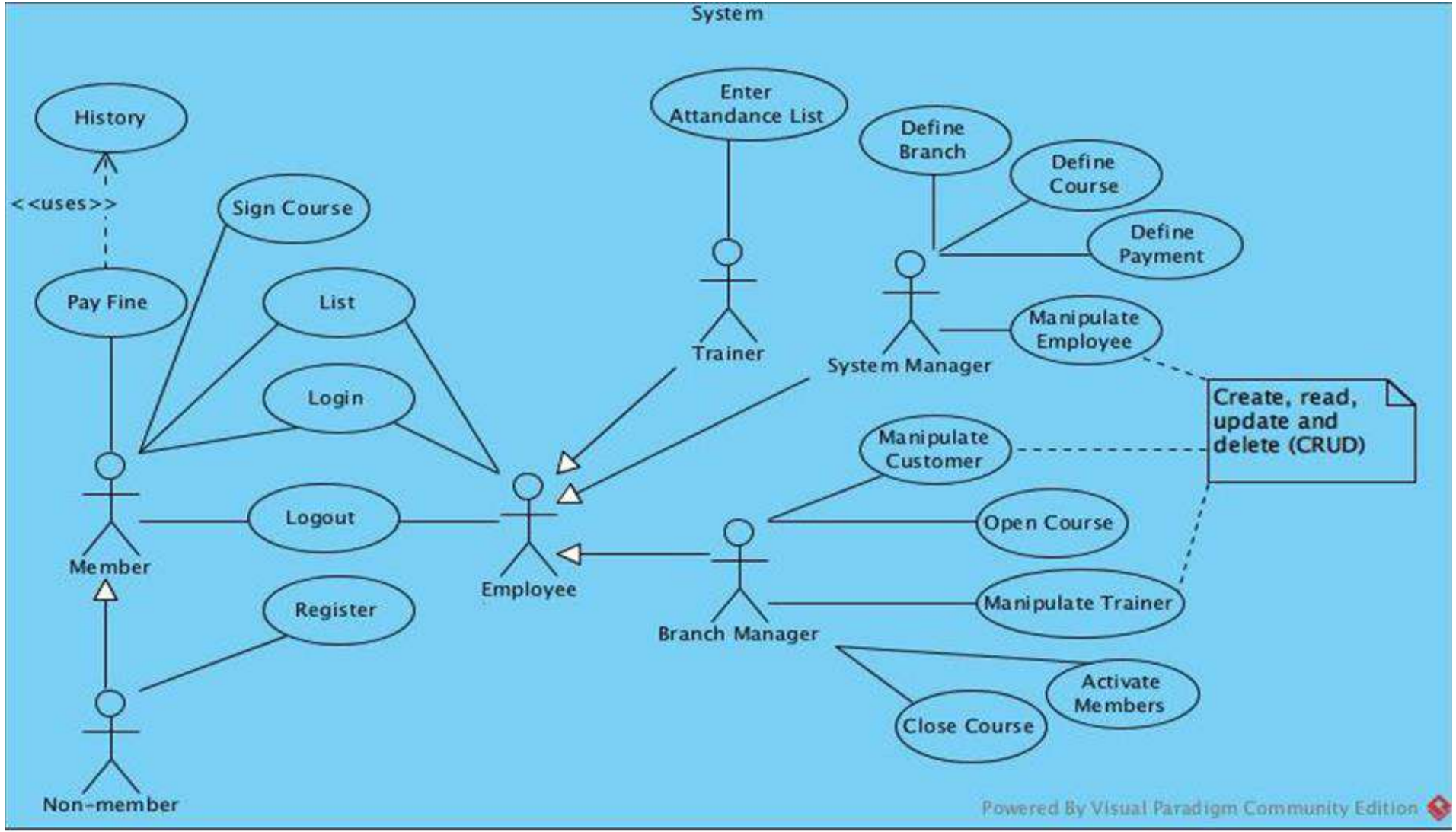
- Unit Testing
- System Testing
- Black box Testing
- Load Testing
- User Acceptance Testing
- Scalability Testing

Software Tester



Sports Center Membership System (SCMS) – Requirements

UML Use Case Diagram for SCMS



See you next week…

UML & TOOL INTRODUCTION

R.A. BURCU YALÇINER

R.A. FEYZA NUR KILIÇASLAN

CONTEXT

- UML Introduction
- UML Diagram Types
- UML Tool Introduction
- About Vision and Project Plan Templates
- Software System to be developed

WHAT IS UML?

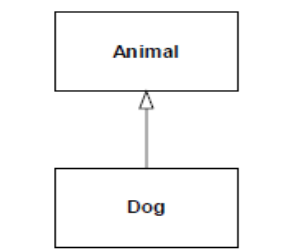


Figure 1-1
A Dog is an Animal

public class Animal {}
Public class Dog extends Animal {}

- **The Unified Modeling Language (UML)** is a graphical notation for drawing diagrams of software concepts.
- A language for specifying, visualizing, constructing, and documenting the **artifacts** of software systems.
- **Conceptual** (a problem domain), **Specification** (a proposed software design), and **Implementation** (software implementation).

An **artifact** is one of many kinds of tangible by-products produced during the development of software.

- Some artifacts which are requirements, design documents, use cases, class diagrams, sequence diagrams and other UML models etc. are used to utilize describing the function, architecture and design of the software developed.
- Other artifacts are concerned with the process of development itself—such as project plans, business cases, and risk assessments.

UML – THE BASICS

- **Abstraction** (A simplification or model of a complex concept, process, or real-world object – help people understand something at an appropriate level)
- **Encapsulation** (Higlight the important aspects of an object – Hide the cumbersome internal details of the object – make the system easier to understand and reuse – make a system more extendible)
- **Entities** (object, class) and **relationships** between objects

UML DIAGRAM TYPES

- UML has three main kinds of diagrams
  - **Static diagrams** describe the unchanging logical structure of software elements by depicting classes, objects, and data structures; and the relationships that exist between them.
  - **Dynamic diagrams** show how software entities change during execution by depicting the flow of execution, or the way entities change state.
  - **Physical diagrams** show the unchanging physical structure of software entities by depicting physical entities such as source files, libraries, binary files, data files, etc., and the relationships that exist between them.

UML DIAGRAM TYPES

- The current UML standards call for 14 different types of diagrams
- These diagrams are organized into two distinct groups: **structural diagrams** and **behavioral (interaction) diagrams.**

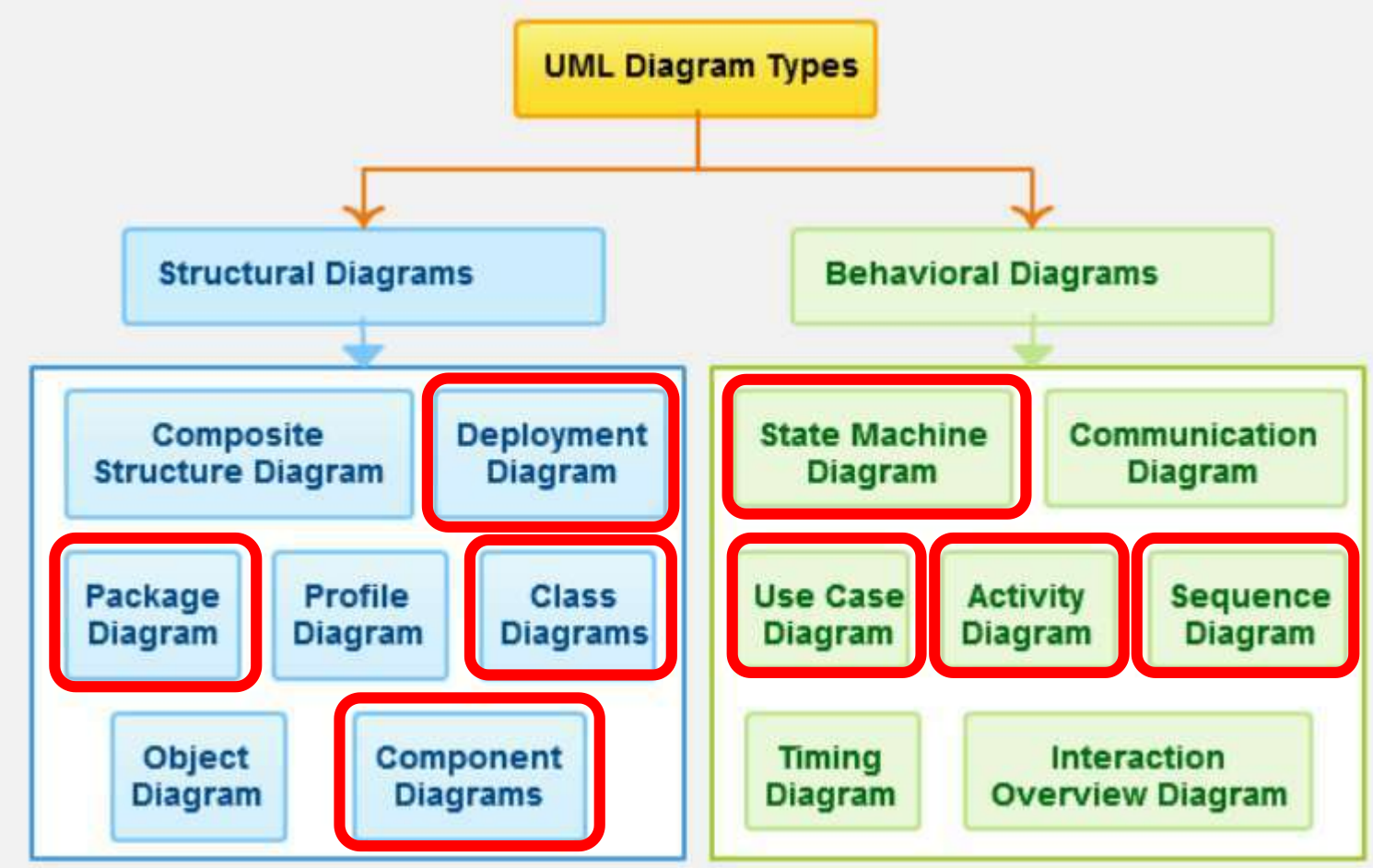


Figure 2. UML Diagram Types

UML TOOL

- **Visual Paradigm Community Edition**

  https://www.visual-paradigm.com/download/community.jsp

  - **Visual Paradigm** is a powerful, cross-platform and yet easy-to-use design and management tool for IT systems. **Visual Paradigm** provides software developers the cutting edge development platform to build quality applications faster, better and cheaper.

- SmartDraw https://www.smartdraw.com/

- MS Visio

SOFTWARE VISION AND PROJECT PLAN TEMPLATES

- **Software Vision**

  - ✓ **Software Vision** is **the artifact** that defines the view of the stakeholders of the technical solution to be developed. This definition is specified in terms of the key needs and features of the stakeholders. The vision contains an outline of the envisioned core requirements for the system.

https://piazza.com/hacettepe.edu.tr/spring2020/bbm382384/resources

SOFTWARE VISION AND PROJECT PLAN TEMPLATES

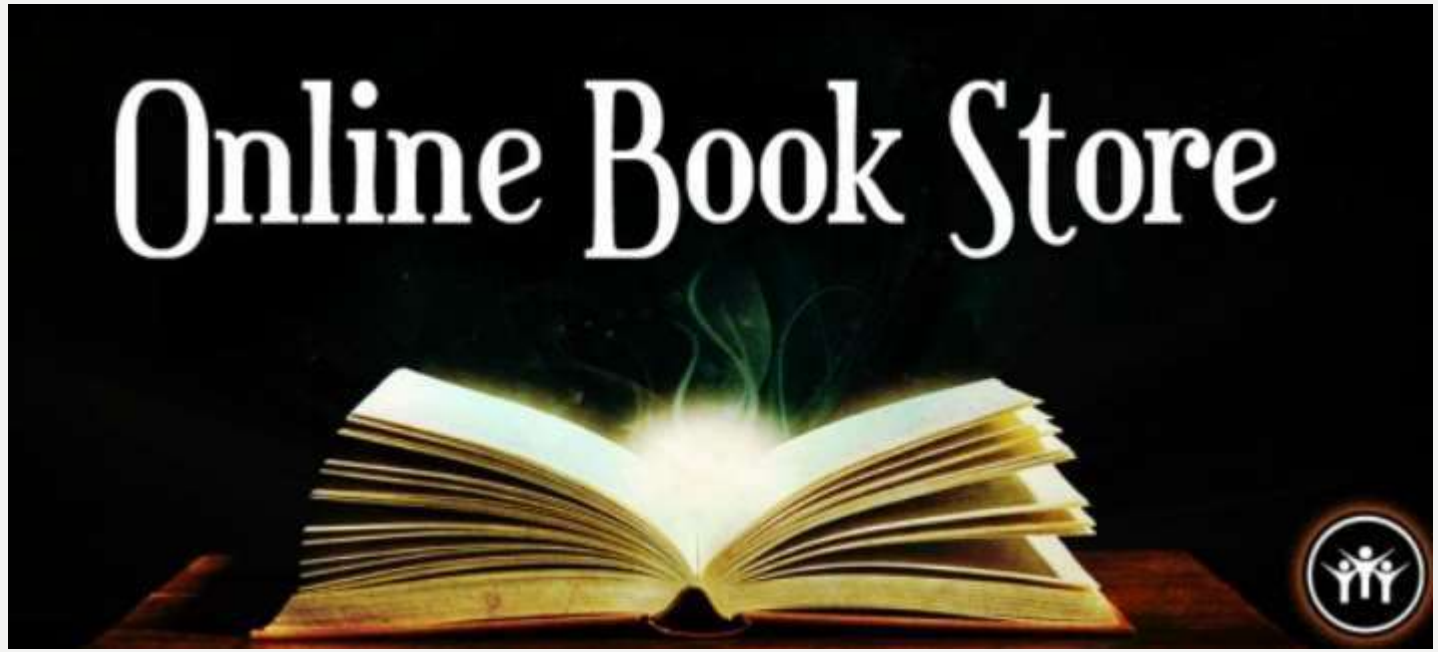
- **Project Plan**

  - ✓ **Project Plan** is the **artifact** that is a collaborative task outlining an initial agreement on how the project will achieve its goals. The resulting project plan provides a summary-level overview of the project.

https://piazza.com/hacettepe.edu.tr/spring2020/bbm382384/resources

PRODUCT OF THE LAB. PROJECT

System to be developed:



SEE YOU NEXT WEEK…

UML & Tool Introduction

Context

- Introduction of the Project and Rolles
- UML Introduction
- UML Diagram Types
- UML Tool Introduction
- About Vision and Project Plan Templates

About the Project

- Online Shopping (such as Morhipo,Trendyol,Hepsi Burada etc.)

UML

- The Unified Modeling Language (UML) is a graphical notation for drawing diagrams of software concepts.
- A language for specifying, visualizing, constructing, and documenting the artifacts of software systems
- *Conceptual (a problem domain)*, *Specification (a proposed software design)*, and *Implementation (software implementation)*

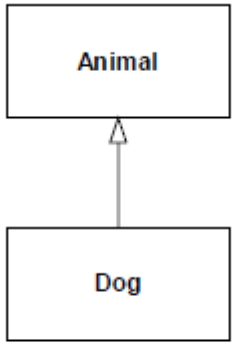


Figure 1-1
A Dog is an Animal

```
public class Animal {}
Public class Dog extends Animal {}
```

UML – The Basics

- Abstraction (A simplification or model of a complex concept, process, or real-World object – Help people understand something at an appropriate level)
- Encapsulation (Higlight the important aspects of an object – Hide the cumbersome internal details of the object – make the system easier to understand and reuse – make a system more extendible)
- Entities (object, class) and relationships between objects

UML Diagram Types

- UML has three main kinds of diagrams
  - Static diagrams describe the unchanging logical structure of software elements by depicting classes, objects, and data structures; and the relationships that exist between them.
  - Dynamic diagrams show how software entities change during execution by depicting the flow of execution, or the way entities change state.
  - Physical diagrams show the unchanging physical structure of software entities by depicting physical entities such as source files, libraries, binary files, data files, etc., and the relationships that exist between them.

UML Diagram Types

- The current UML standards call for 13 different types of diagrams
- These diagrams are organized into two distinct groups: structural diagrams and behavioral or interaction diagrams.

Structural UML diagrams

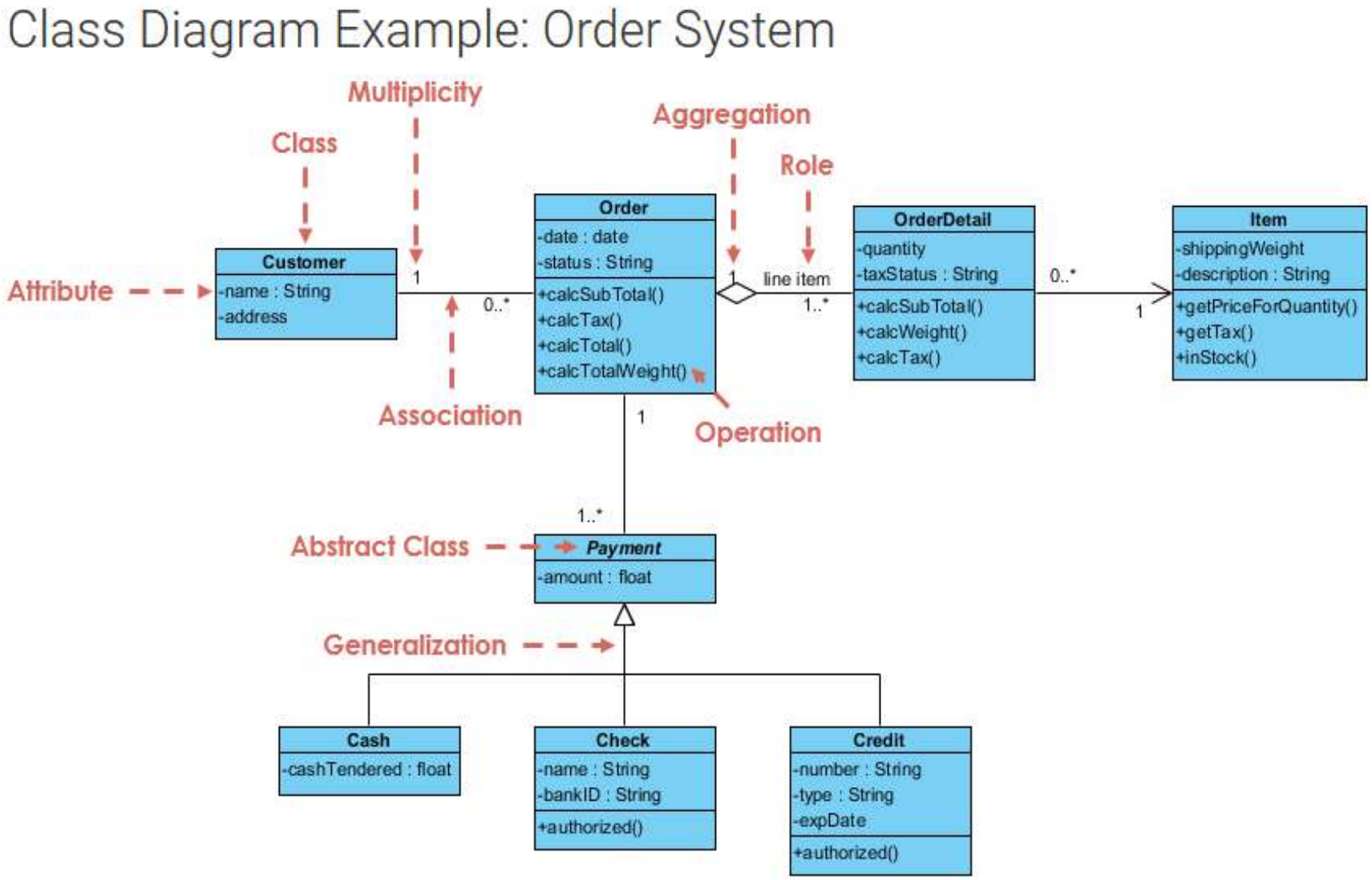
- **Class diagram**
- Package diagram
- Object diagram
- Component diagram
- Composite structure diagram
- Deployment diagram

Behavioral UML diagrams

- **Activity diagram**
- Sequence diagram
- **Use case diagram**
- State diagram
- Communication diagram
- Interaction overview diagram
- Timing diagram

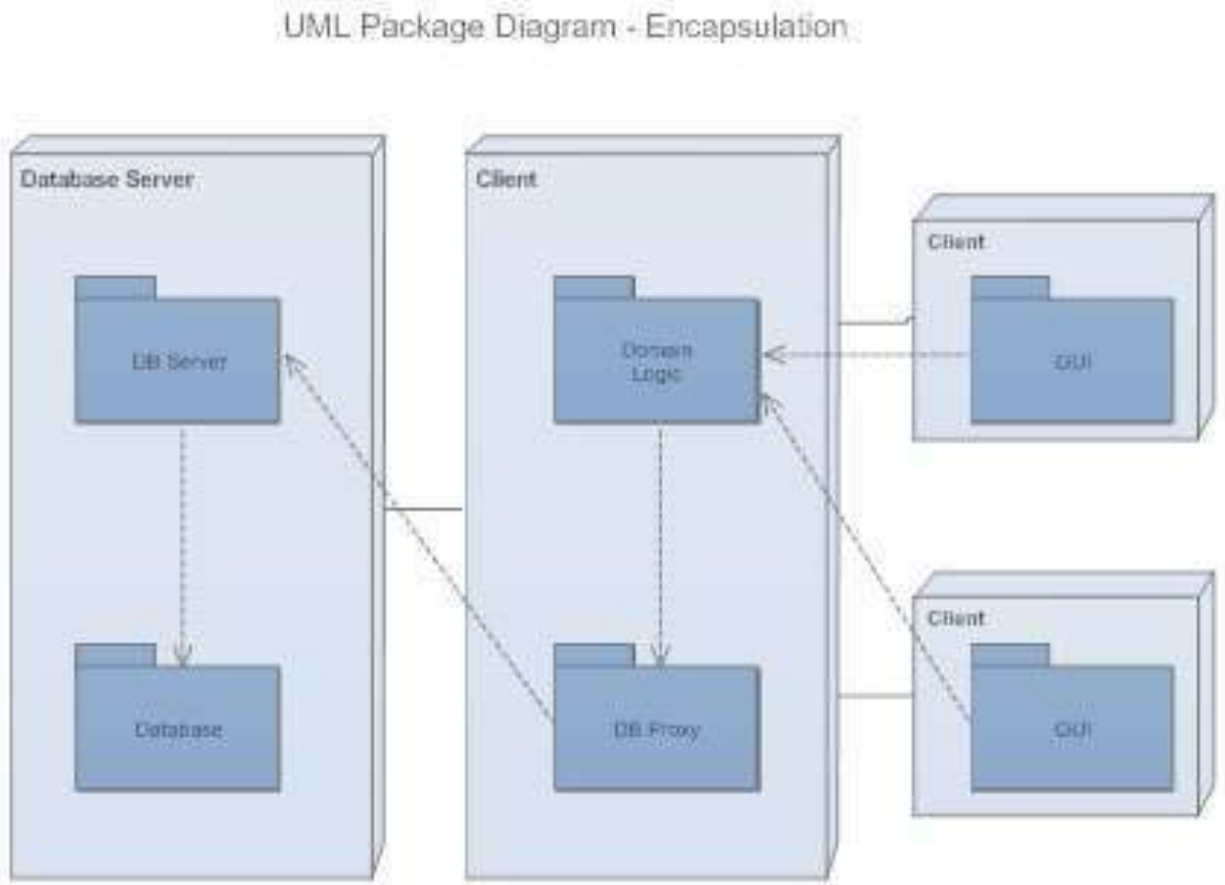
UML Diagram Types – Class Diagram

A class diagram models the static structure of a system. It shows relationships between classes, objects, attributes, and operations.



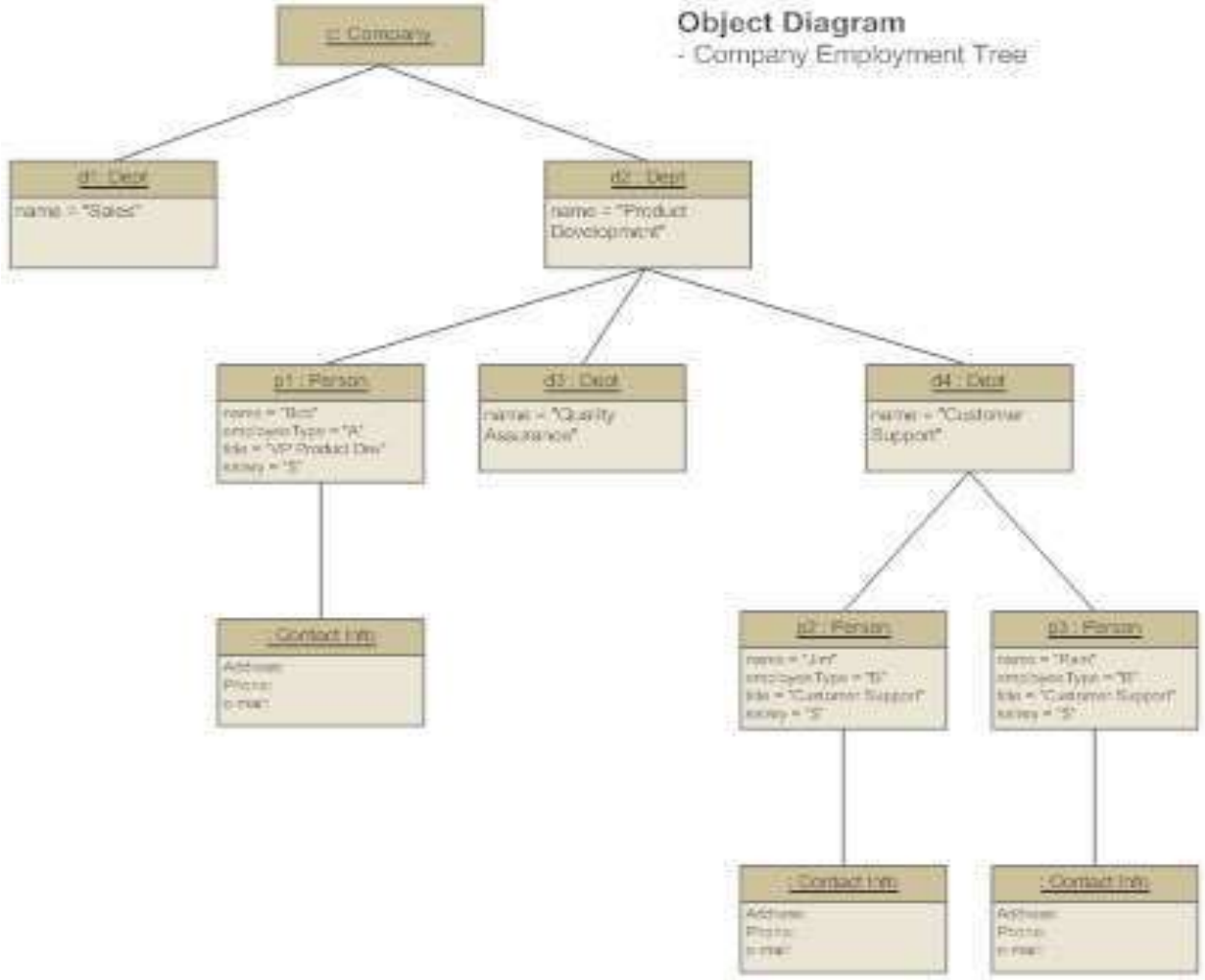
UML Diagram Types – Package Diagram

Package diagrams are a subset of class diagrams, but developers sometimes treat them as a separate technique. Package diagrams organize elements of a system into related groups to minimize dependencies between packages.



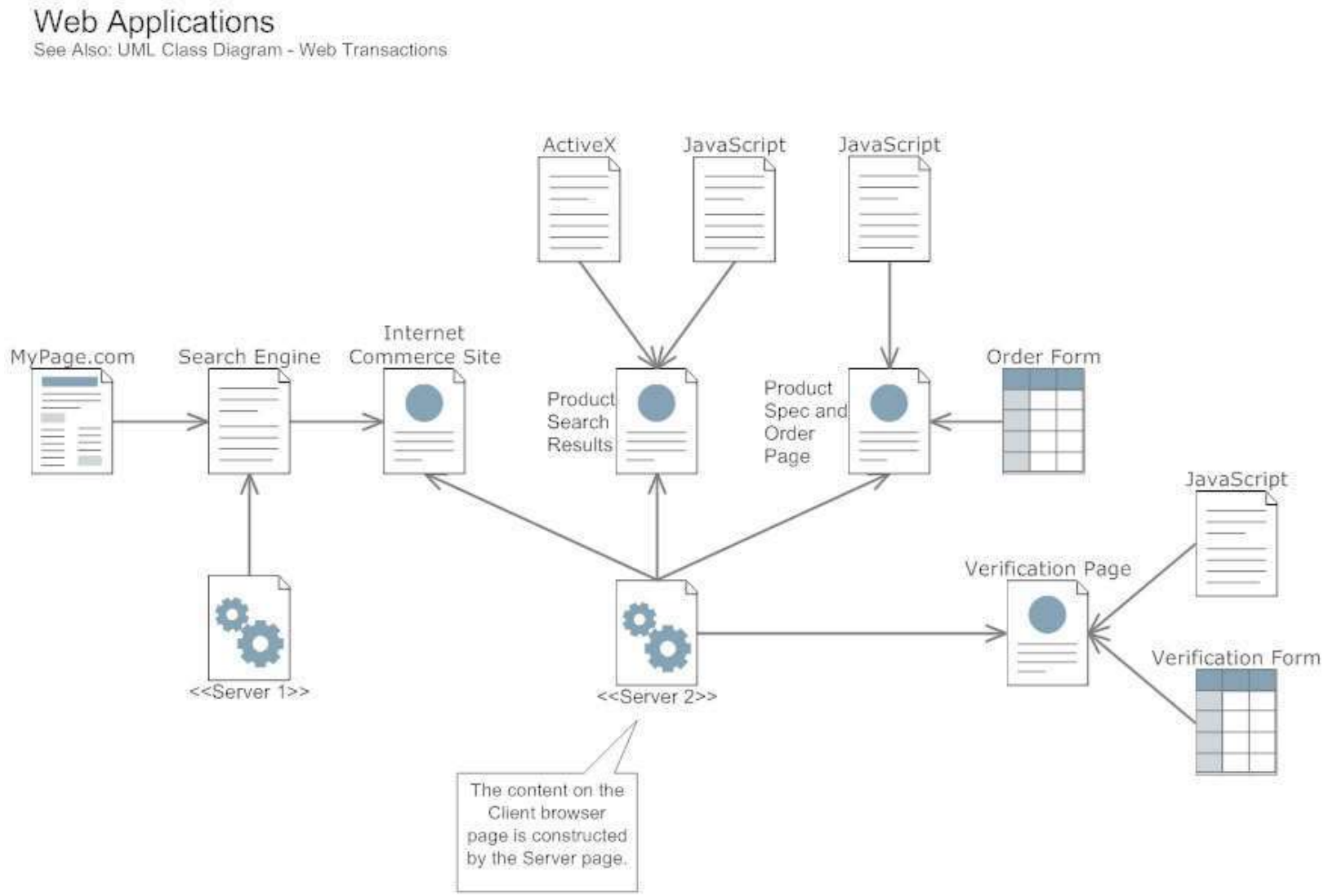
UML Diagram Types – Object Diagram

Object diagrams describe the static structure of a system at a particular time. They can be used to test class diagrams for accuracy.



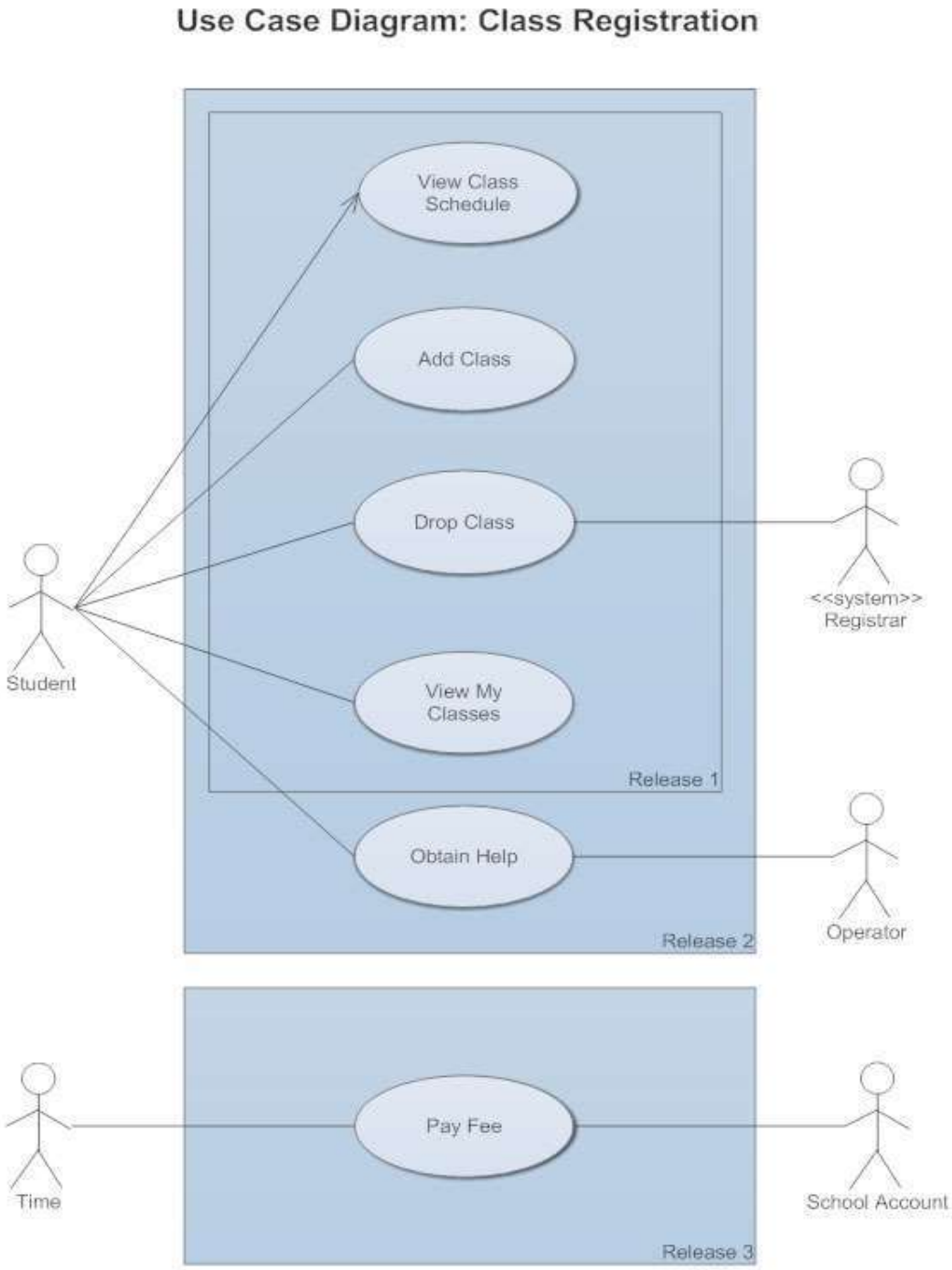
UML Diagram Types – Component Diagram

Component diagrams describe the organization of physical software components, including source code, run-time (binary) code, and executables.



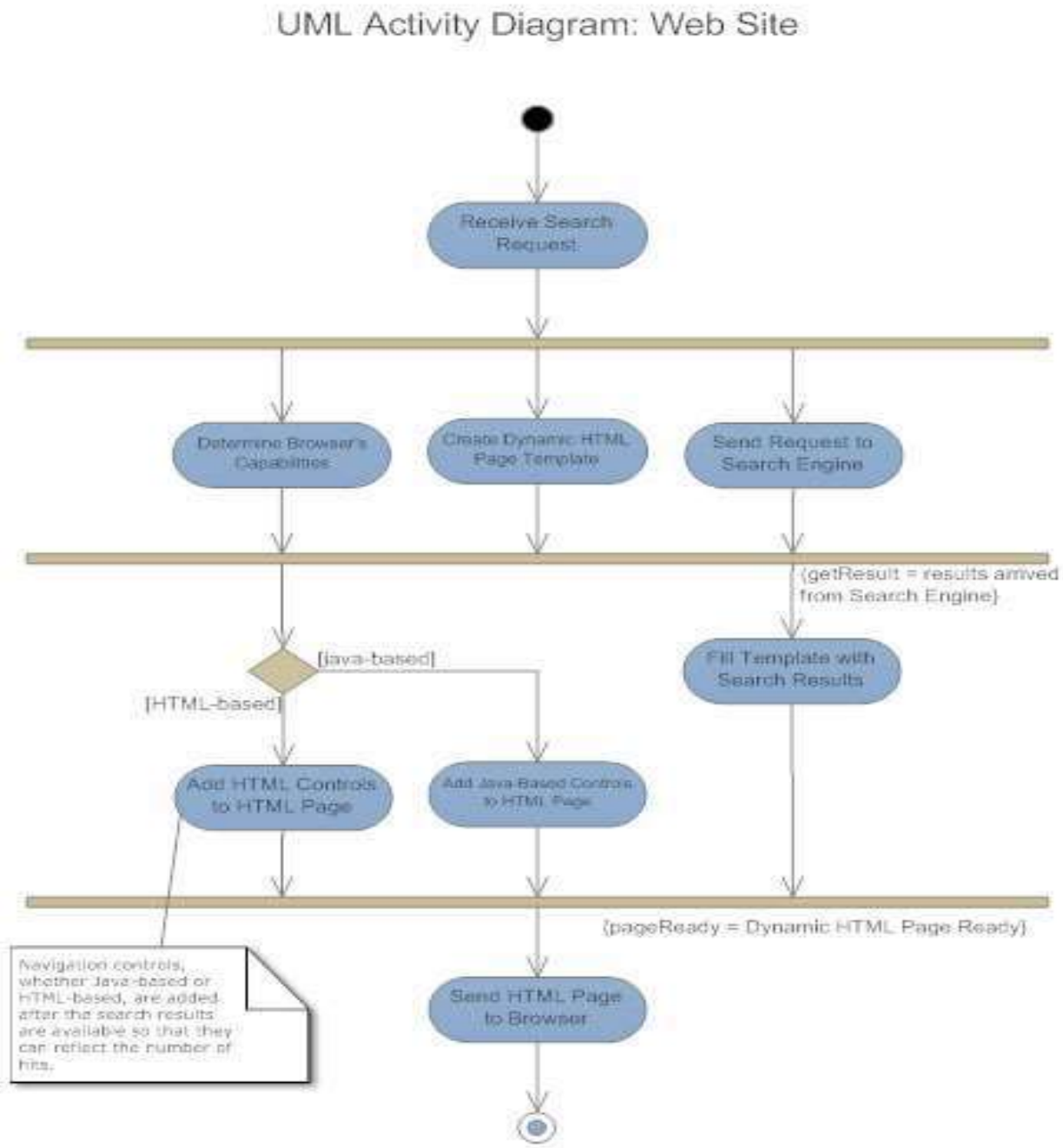
UML Diagram Types – UseCase Diagram

A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform.



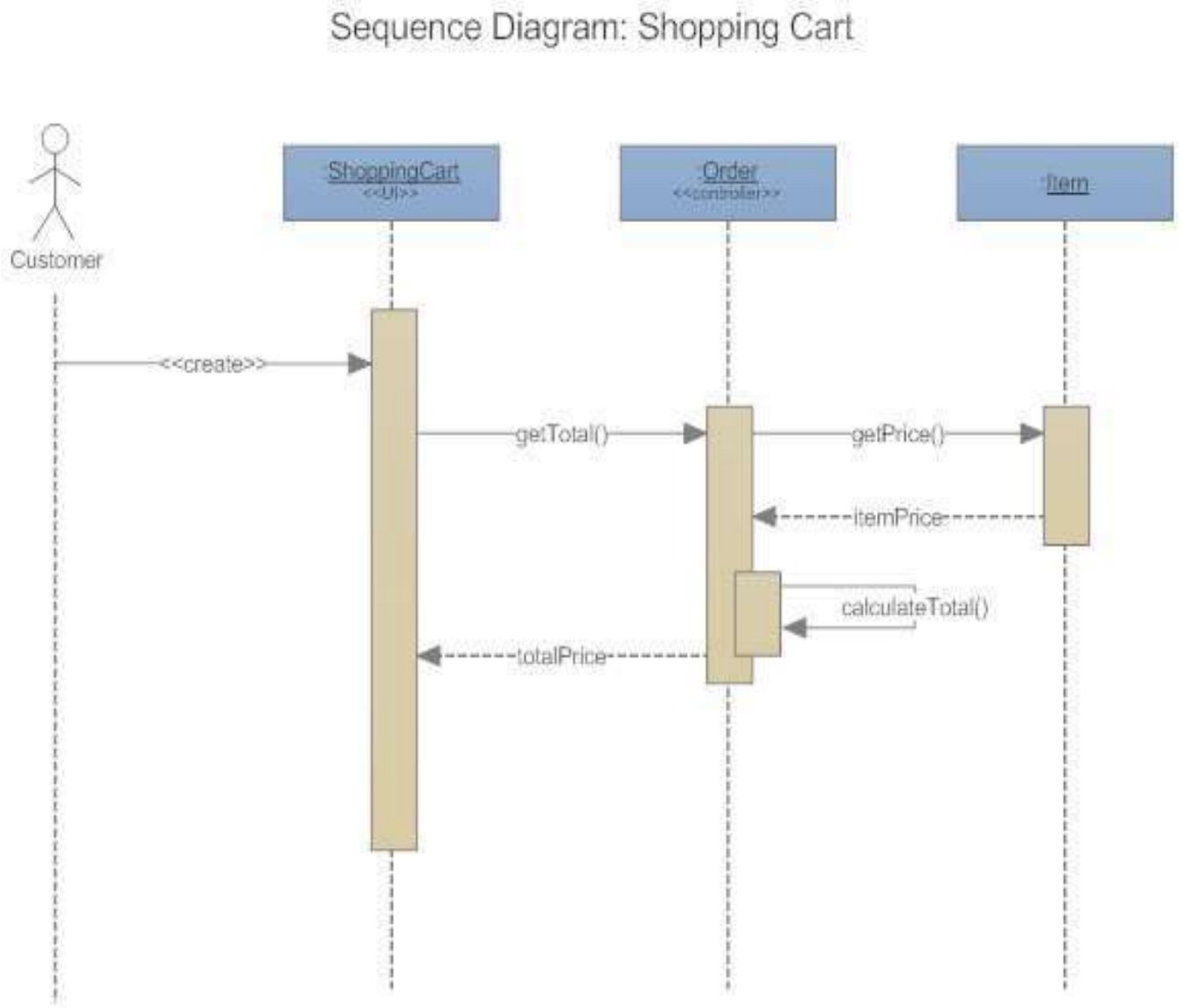
UML Diagram Types – Activity Diagram

Activity diagrams illustrate the dynamic nature of a system by modeling the flow of control from activity to activity. An activity represents an operation on some class in the system that results in a change in the state of the system. Typically, activity diagrams are used to model workflow or business processes and internal operation



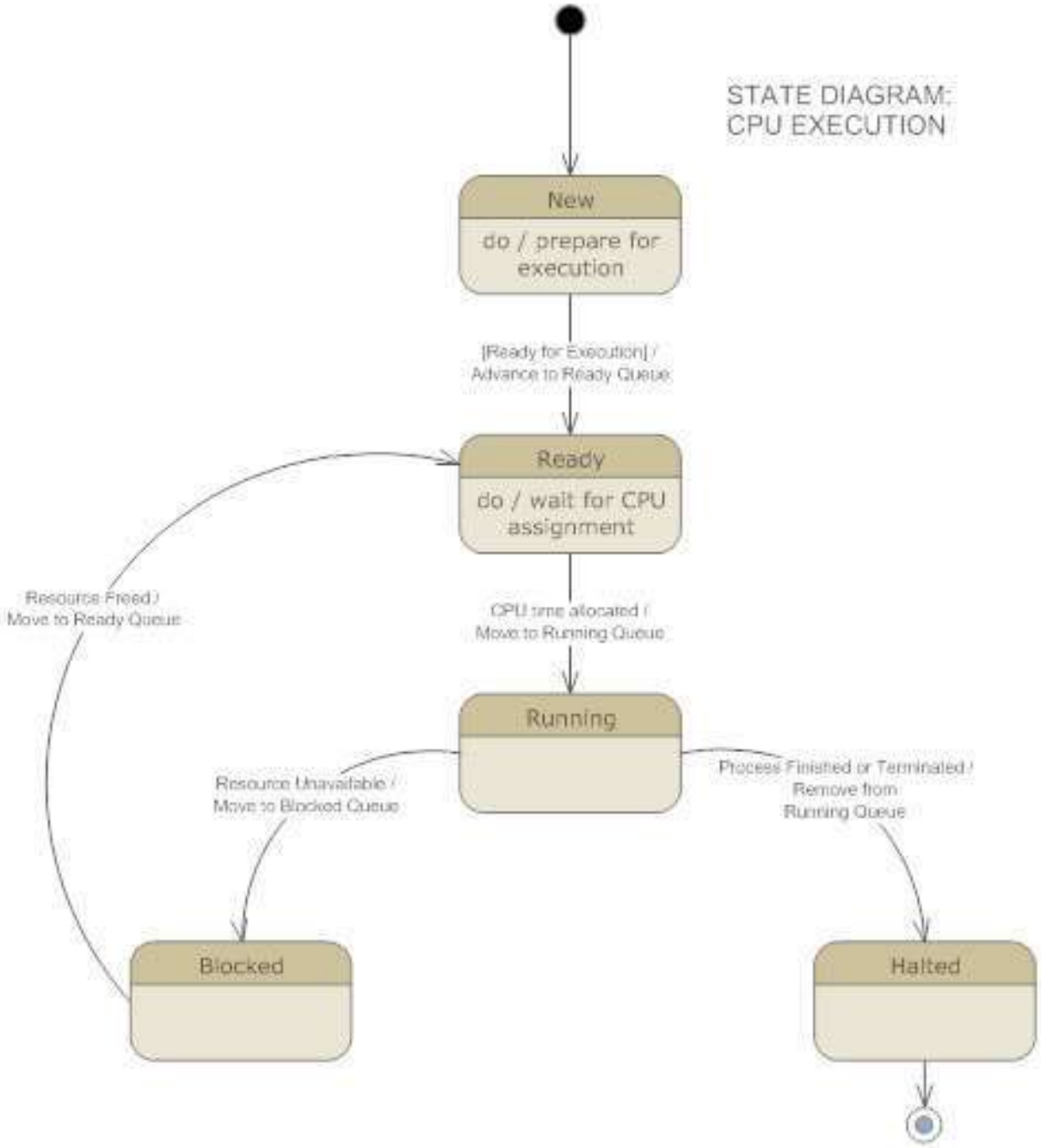
UML Diagram Types – Sequence Diagram

Sequence diagrams describe interactions among classes in terms of an exchange of messages over time



UML Diagram Types – State Diagram

Statechart diagrams, now known as state machine diagrams and state diagrams describe the dynamic behavior of a system in response to external stimuli. State diagrams are especially useful in modeling reactive objects whose states are triggered by specific events



UML Tool

- Visual Paradigm Community Edition https://www.visual-paradigm.com/download/community.jsp
  - Visual Paradigm is a powerful, cross-platform and yet easy-to-use design and management tool for IT systems. Visual Paradigm provides software developers the cutting edge development platform to build quality applications faster, better and cheaper.
- SmartDraw https://www.smartdraw.com/
- MS Visio

Vision and Project Plan Templates

- Vision
  - https://piazza.com/hacettepe.edu.tr/spring2019/bbm382384/resources
  - This artifact defines the view of the stakeholders of the technical solution to be developed. This definition is specified in terms of the key needs and features of the stakeholders. The vision contains an outline of the envisioned core requirements for the system.

Vision and Project Plan Templates (with OpenUp)

- Project Plan Template
  - https://piazza.com/hacettepe.edu.tr/spring2019/bbm382384/resources
  - A collaborative task that outlines an initial agreement on how the project will achieve its goals. The resulting project plan provides a summary-level overview of the project.

TRENDYOL

Product of the Lab. Project

MORHIPO

- **System to be developed:**

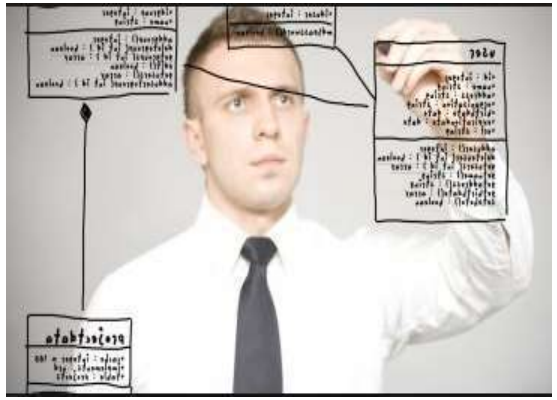
Online Shopping

GİTTİ GİDİYOR

Specialized Team Roles

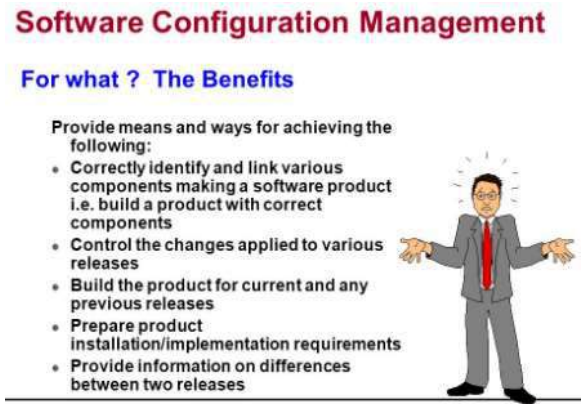
Software Project Manager

In addition to
Software Developer



Software Architect

Software Analyst

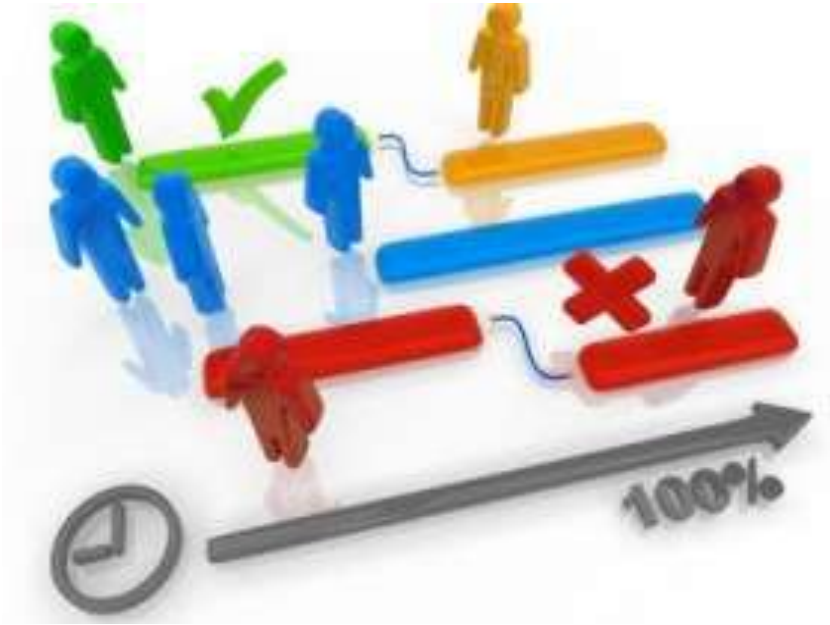


Software Configuration Manager

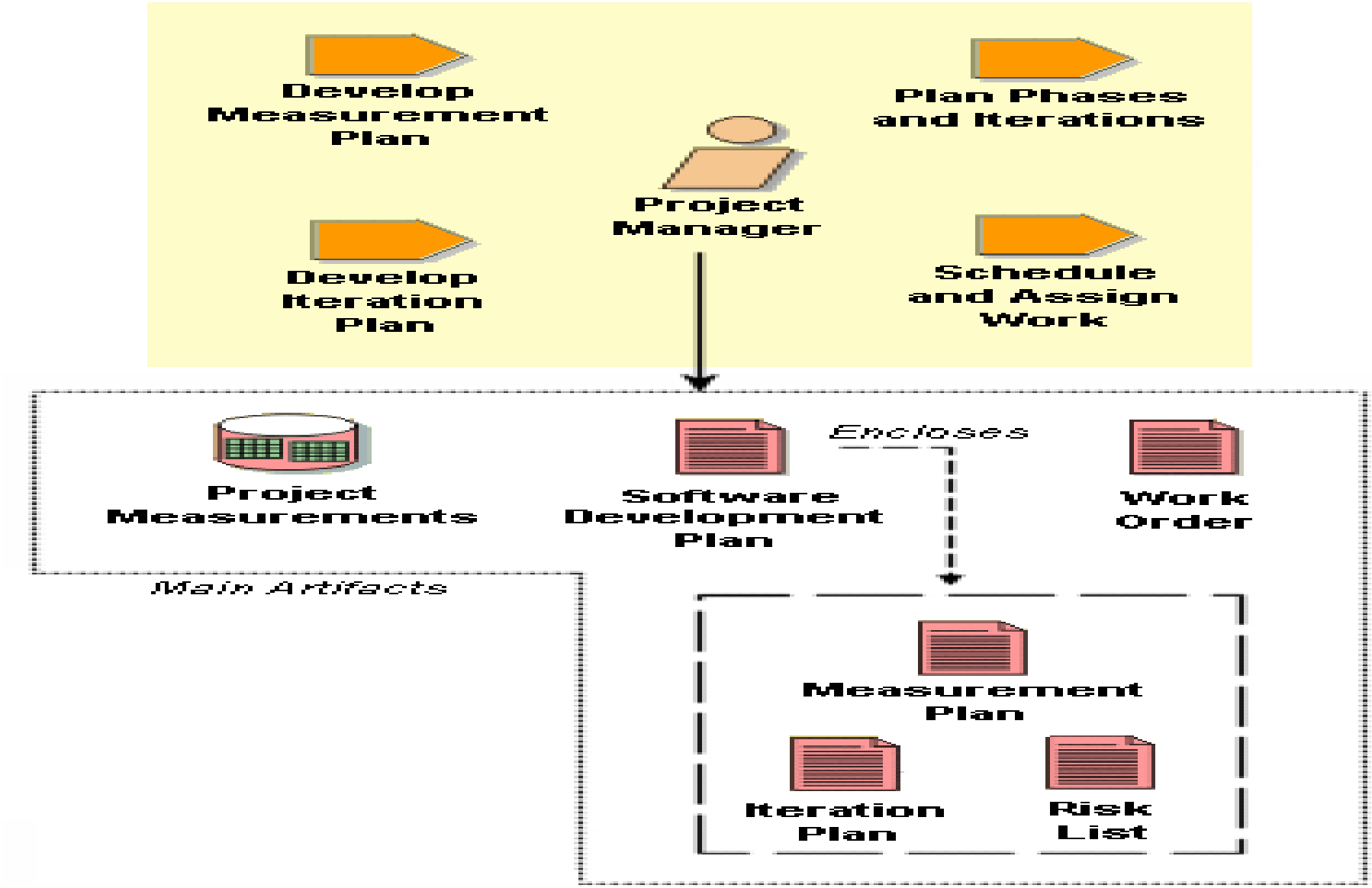
Software Tester

Project Manager

- The **project manager role** allocates resources, shapes priorities, coordinates interactions with customers and users, and generally keeps the project team focused on the right goal. The project manager also establishes a set of practices that ensure the integrity and quality of project artifacts.
- **<u>Responsibilities of a Project Manager</u>**
- **Planning** – scope, activity schedules, gantt chart, potential risks etc.
- **Setting goals -** For example: Complete the project within six months from start date in the budget of xxx amount.
- **Time Management**
- **Budget Allocation and Cost Estimates**
- **Implementation and Monitoring**



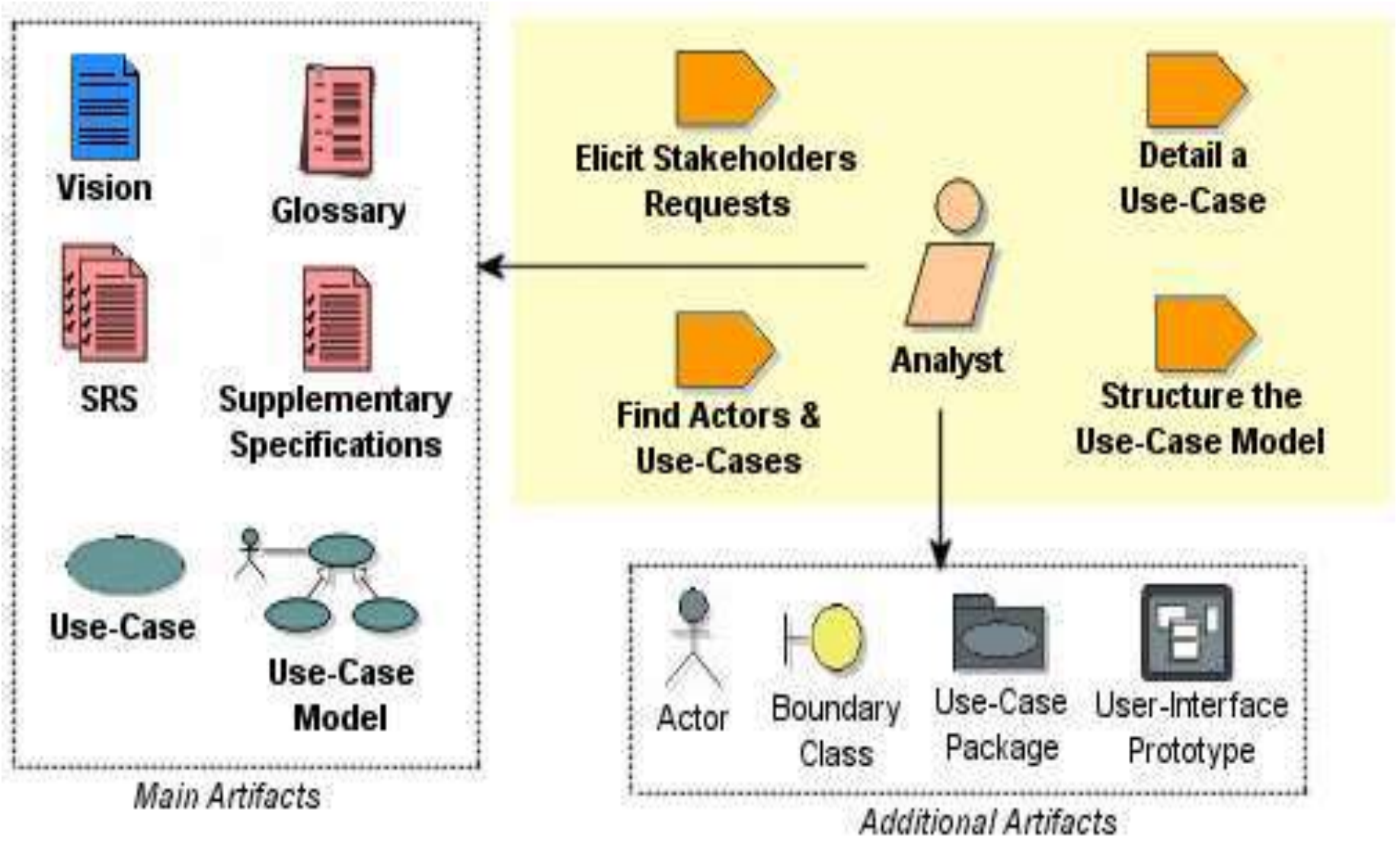
Project Manager



Software Analyst

- In a software development team, a **software analyst** is the person who studies the software application domain, prepares software requirements, and specification (Software Requirements Specification) documents.
The software analyst is the seam between the software users and the software developers.
- S/he is responsible for
  - Performing complex analysis, designing and programming to meet business requirements.
  - Maintaining, managing and modifying all software systems and applications.
  - Defining specifications for complex software programming applications.
  - Interfacing with end-users and software consultants.
  - Developing, maintaining and managing systems, software tools and applications.
  - Resolving complex issues relating to business requirements and objectives.
  - Coordinating and supporting software professionals in installing and analyzing applications and tools.
  - Analyzing, developing and implementing testing procedures, programming and documentation.
  - Training and developing other software analysts.
  - Analyzing, designing and developing modifications and changes to existing systems to enhance performance.

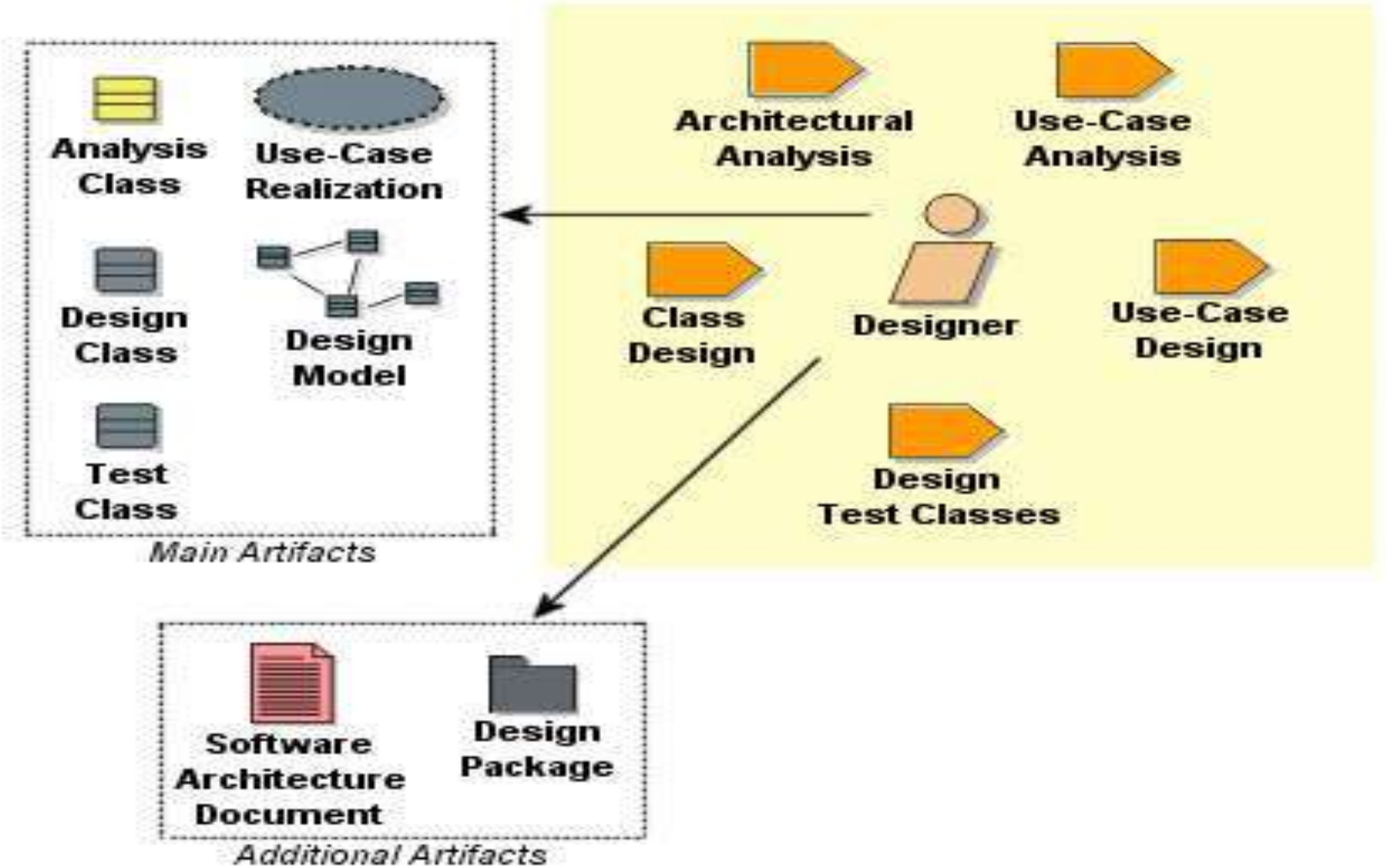
Software Analyst



Software Architect

- A **software architect/designer** is a software expert who makes high-level design choices and dictates technical standards, including software coding standards, tools, and platforms.
- Examples of Software Architect responsibilities
  - Design, develop and execute software solutions to address business issues.
  - Provide architectural blueprints and technical leadership to our IT team.
  - Evaluate and recommend tools, technologies and processes to ensure the highest quality product platform.
  - Collaborate with peer organizations, quality assurance and end users to produce cutting-edge software solutions.
  - Interpret business requirements to articulate the business needs to be addressed.
  - Troubleshoot code level problems quickly and efficiently.

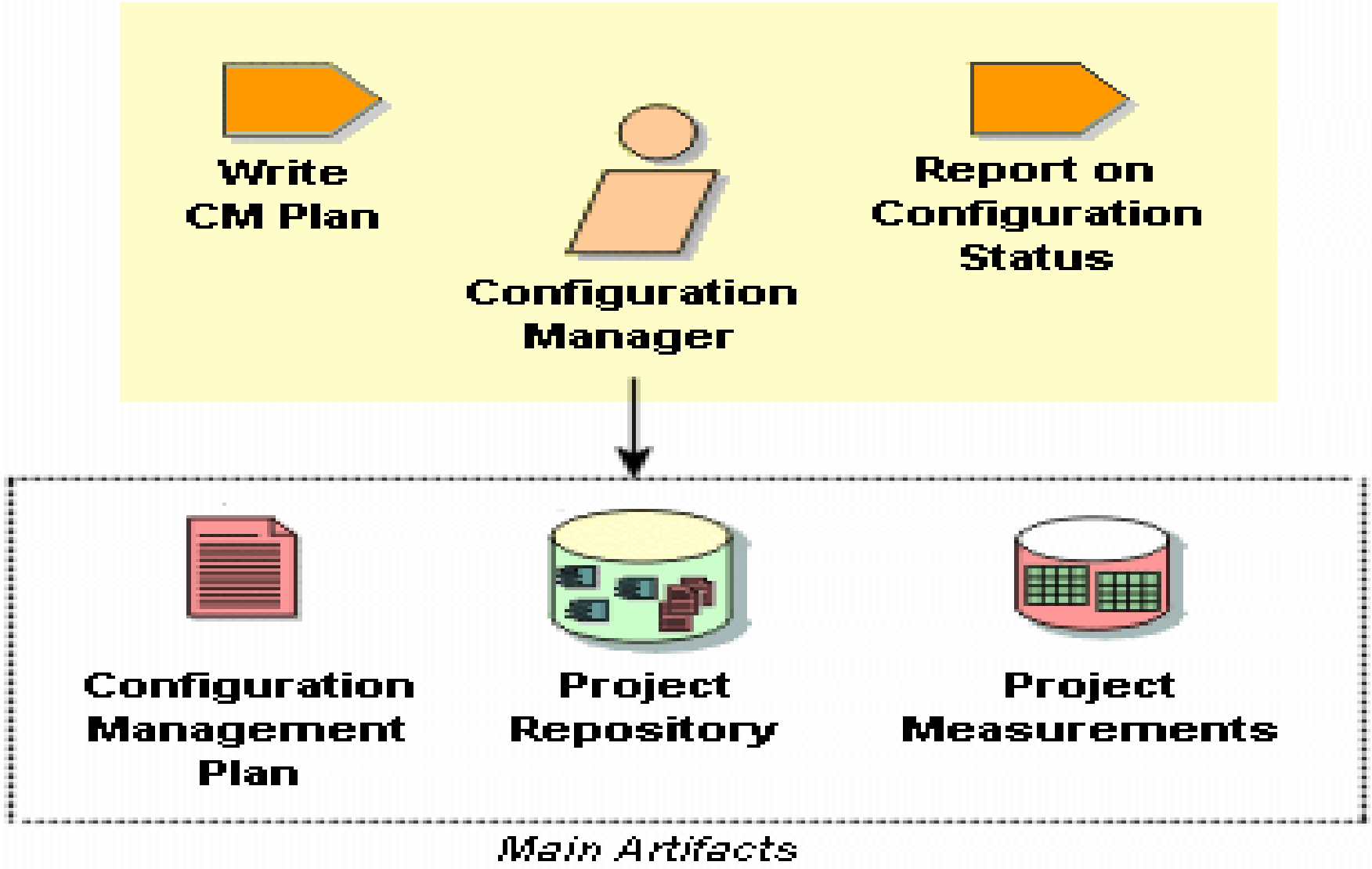
Software Architect



Software Configuration Manager

- The **configuration manager** provides the overall Configuration Management (CM) infrastructure and environment to the product development team. The CM role supports the product development activity so that developers and integrators have appropriate workspaces to build and test their work, and so that all artifacts are available for inclusion in the deployment unit as required. The configuration manager also has to ensure that the CM environment facilitates product review, and change and defect tracking activities. The configuration manager is also responsible for writing the CM Plan and reporting progress statistics based on change requests.

Software Configuration Manager



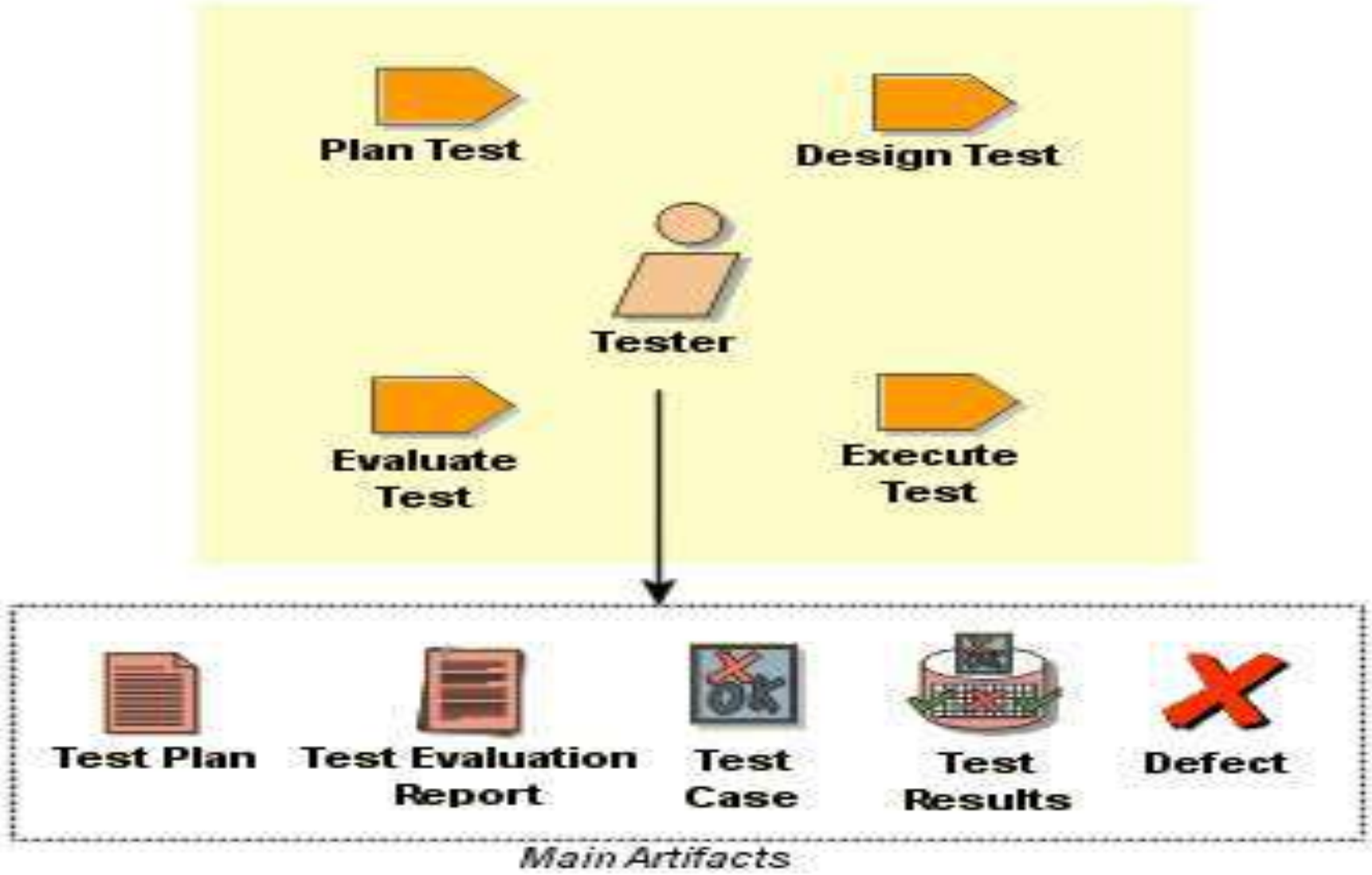
Software Tester

- A **software tester** is an individual that tests software for bugs, errors, defects or any problem that can affect the performance of computer software or an application.
- Software testers are part of a software development team and perform functional and non-functional testing of software using manual and automated software testing techniques.
- S/he is responsible for
  - Identifying the most appropriate implementation approach for a given test
  - Implementing individual tests
  - Setting up and executing the tests
  - Logging outcomes and verifying test execution
  - Analyzing and recovering from execution errors

Some of the techniques software testers must have experience with include:

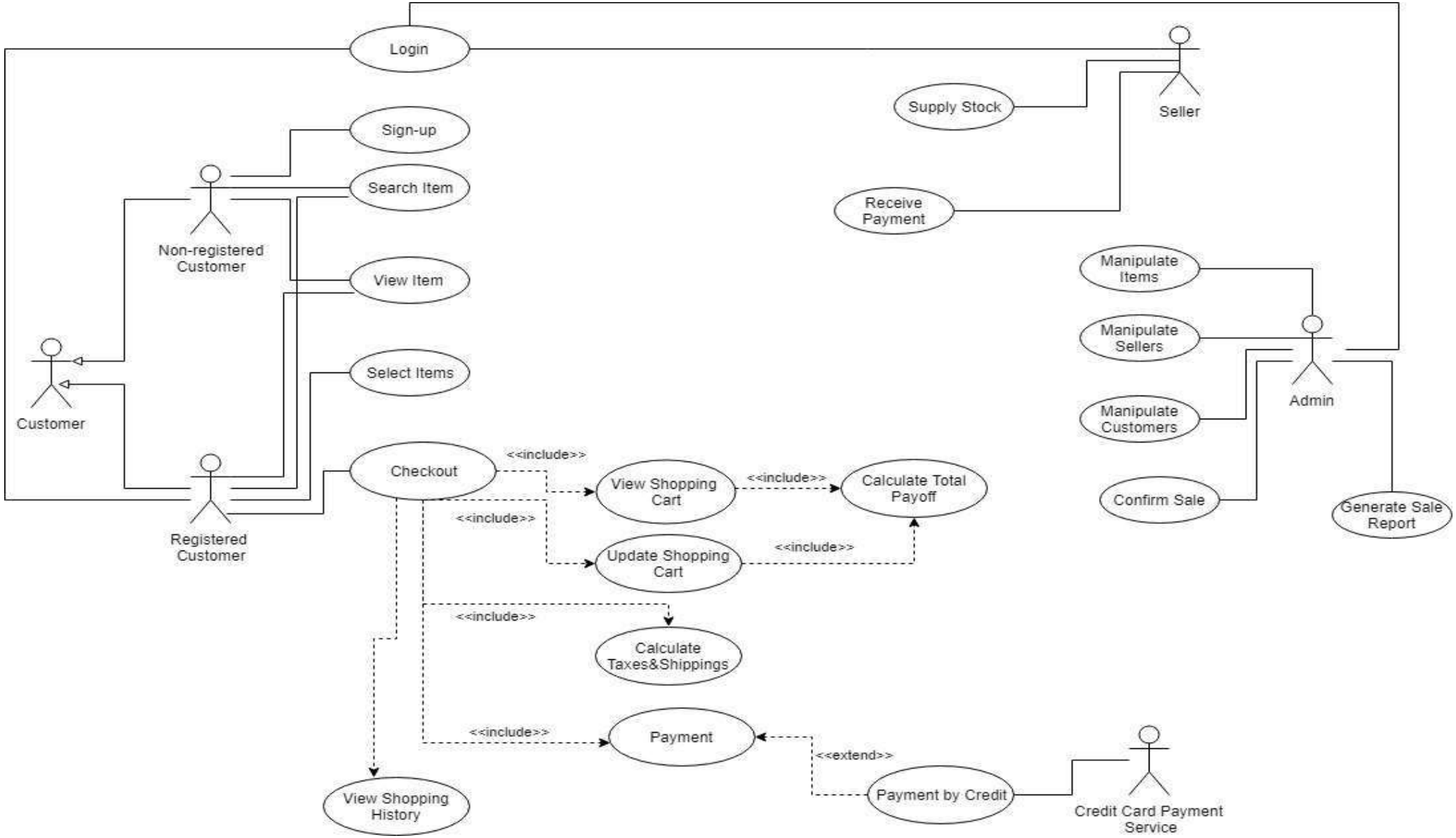
- Unit Testing
- System Testing
- Black box Testing
- Load Testing
- User Acceptance Testing
- Scalability Testing

Software Tester



Sports Center Membership System (SCMS) – Requirements

UML Use Case Diagram for Online Shopping



See you next week…

UML Modelling (Context, UseCase, Activity Diagrams)

Context

- Working with Diagrams
- Context – Data Flow Diagram
- Use Case Diagram
- Activity Diagram
- System-Wide Requirements (SRS) Template

Working with Diagrams

- Before we explore the details of UML, it would be wise to talk about when and why we use it. Much harm has been done to software projects through the misuse and overuse of UML.

- Why do engineers build models? Why do structural engineers build models of bridges?

Working with Diagrams

- Before we explore the details of UML, it would be wise to talk about when and why we use it. Much harm has been done to software projects through the misuse and overuse of UML.

- Why do engineers build models? Why do structural engineers build models of bridges?
- Engineers build models to find out if their designs will work.
- Structural engineers build models of bridges to see if they will stand.

Working with Diagrams

- Before we explore the details of UML, it would be wise to talk about when and why we use it. Much harm has been done to software projects through the misuse and overuse of UML.

- Why don’t structural engineers just build the bridge and then see if it stands?

Working with Diagrams

- Before we explore the details of UML, it would be wise to talk about when and why we use it. Much harm has been done to software projects through the misuse and overuse of UML.

- Why don’t structural engineers just build the bridge and then see if it stands?
- Because bridges are a *lot* more expensive than the models.

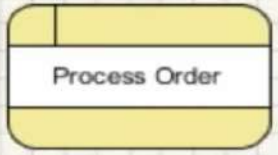
Context Diagram (Data Flow Diagram - DFD)

- A data flow diagram (DFD) illustrates how data is processed by a system in terms of inputs and outputs. As its name indicates its focus is on the flow of information, where data comes from, where it goes and how it gets stored.

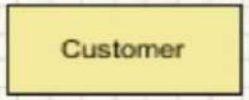
- It is usually beginning with a context diagram as the level 0 of DFD diagram, a simple representation of the whole system.

Context Diagram (Data Flow Diagram - DFD)

Building Blocks of DFD



Process
A business activity or function where the manipulation and transformation of data takes place.



External Entity
An external entity can represent a human, system or subsystem. It is where certain data comes from or goes to.

Data Store
A data store represents the storage of persistent data required and/or produced by the process.

Data Flow
A data flow represents the flow of information. The arrows indicate the direction of data flow from beginning to the end.

Context Diagram (Data Flow Diagram - DFD)

Watch the video

→ https://www.visual-paradigm.com/tutorials/data-flow-diagram-dfd.jsp

UseCase Diagram

- A use case diagram is a dynamic or behavior diagram of a system.
- Use case diagram provides a graphical overview of goals (modeled by use cases) users (represented by actors) want to achieve by using the system (represented by system boundary optionally).
- Use case diagrams model the functionality of a system using actors and use cases.
- Use cases are a set of actions, services, and functions that the system needs to perform

UseCase Diagram

- **System**
  Draw your system's boundaries using a rectangle that contains use cases. Place actors outside the system's boundaries.

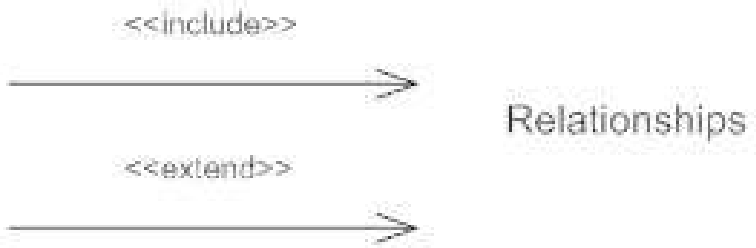
- **Use Case**
  Draw use cases using ovals. Label the ovals with verbs that represent the system's functions.

- **Actors**
  Actors are the users of a system. When one system is the actor of another system, label the actor system with the actor stereotype.

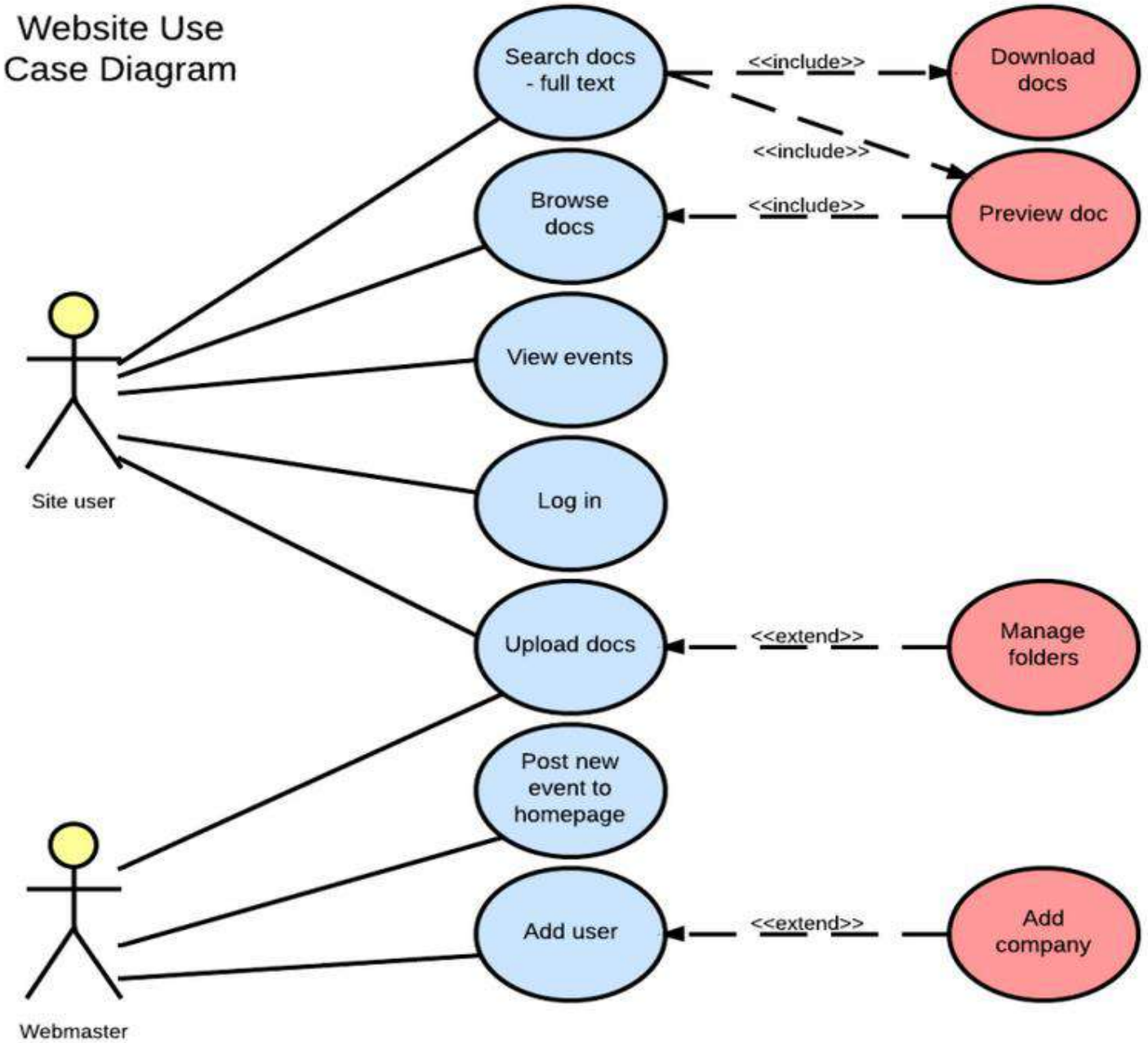
UseCase Diagram

Relationships

Illustrate relationships between an actor and a use case with a simple line. For relationships among use cases, use arrows labeled either "uses" or "extends." A "uses" relationship indicates that one use case is needed by another in order to perform a task. An "extends" relationship indicates alternative options under a certain use case.



UseCase Diagram

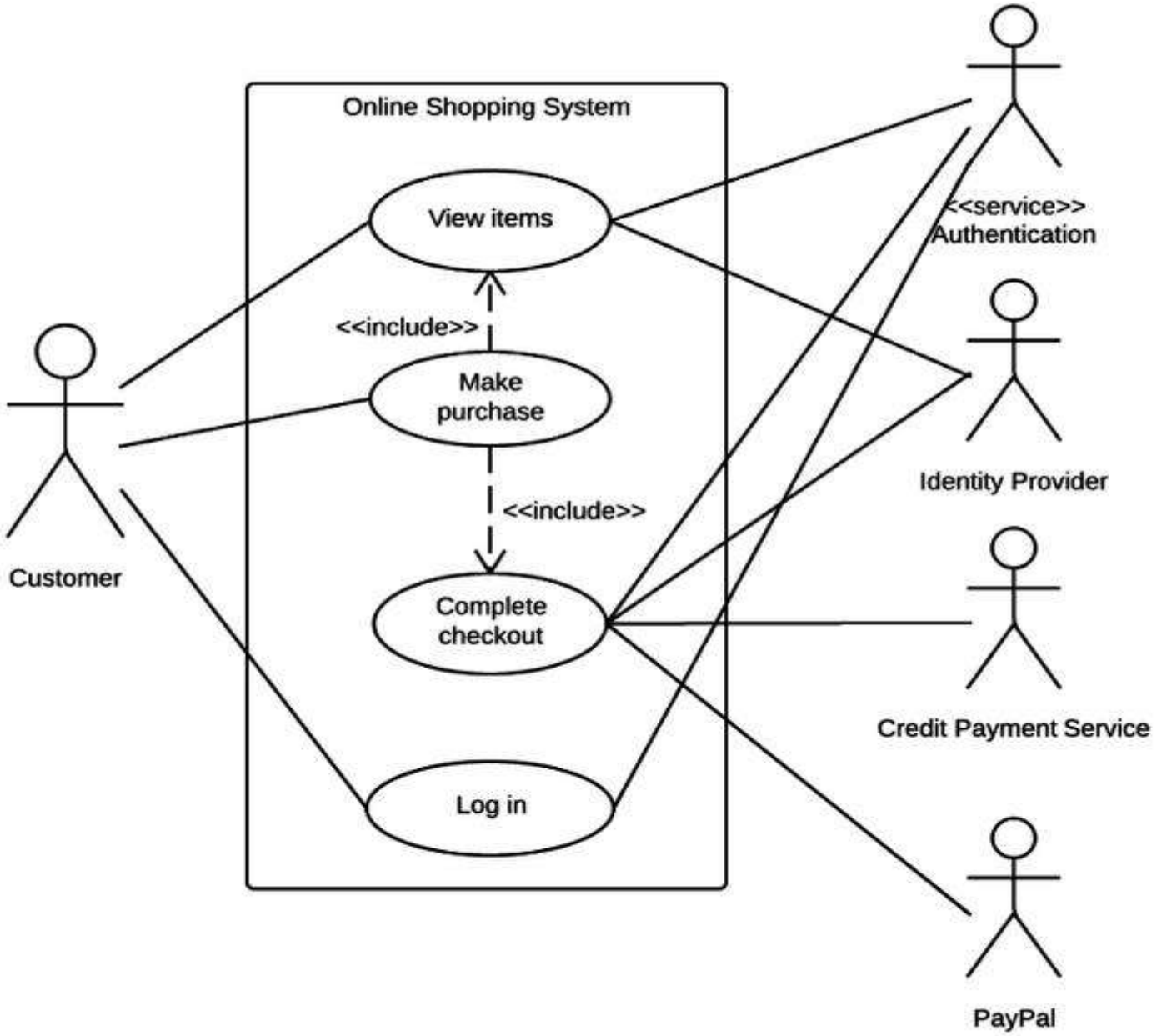


UseCase Diagram

- It’s your turn!
  - Create a basic UseCase Diagram for Online Shopping System

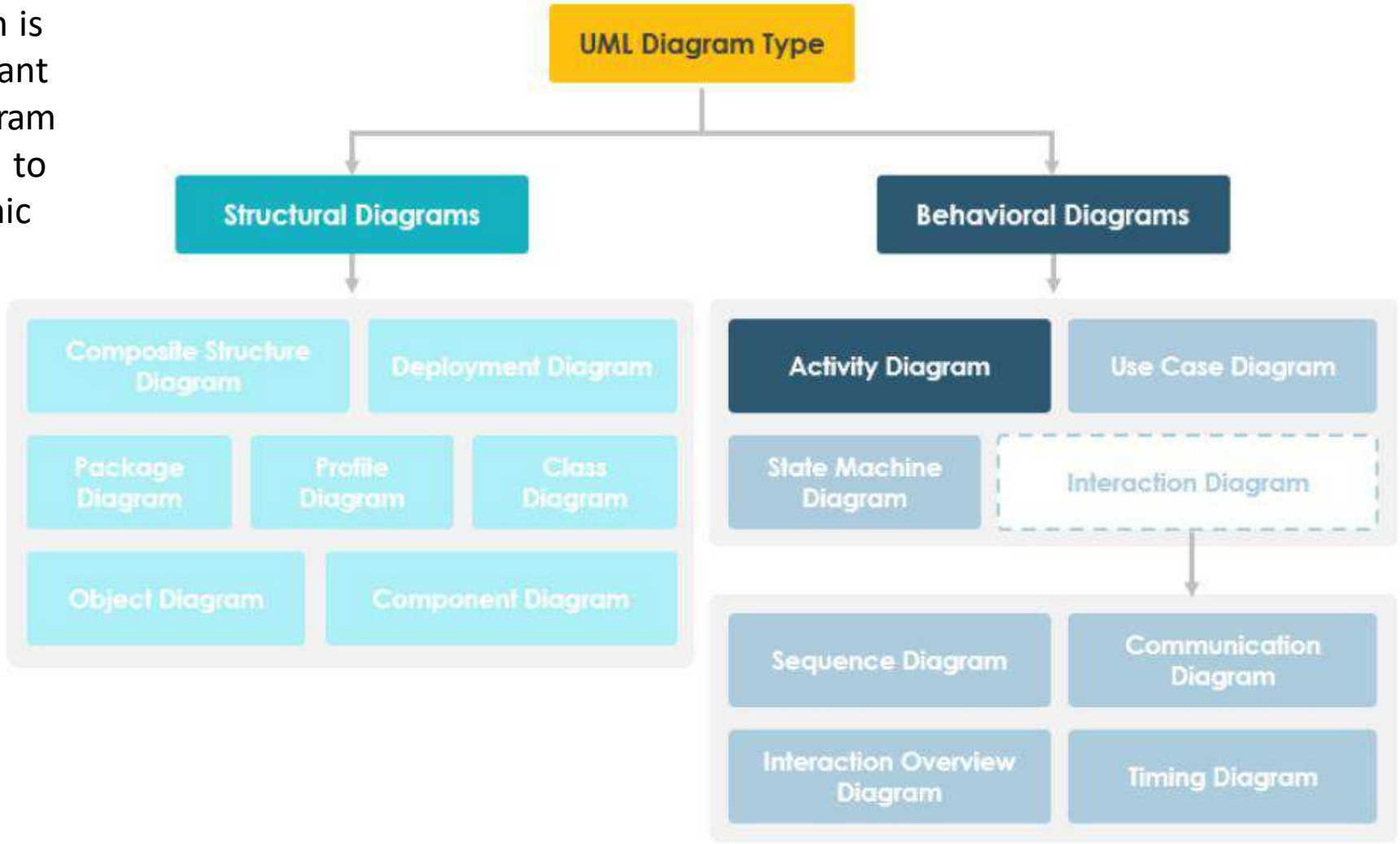
UseCase Diagram

- Online Shopping System



Activity Diagram

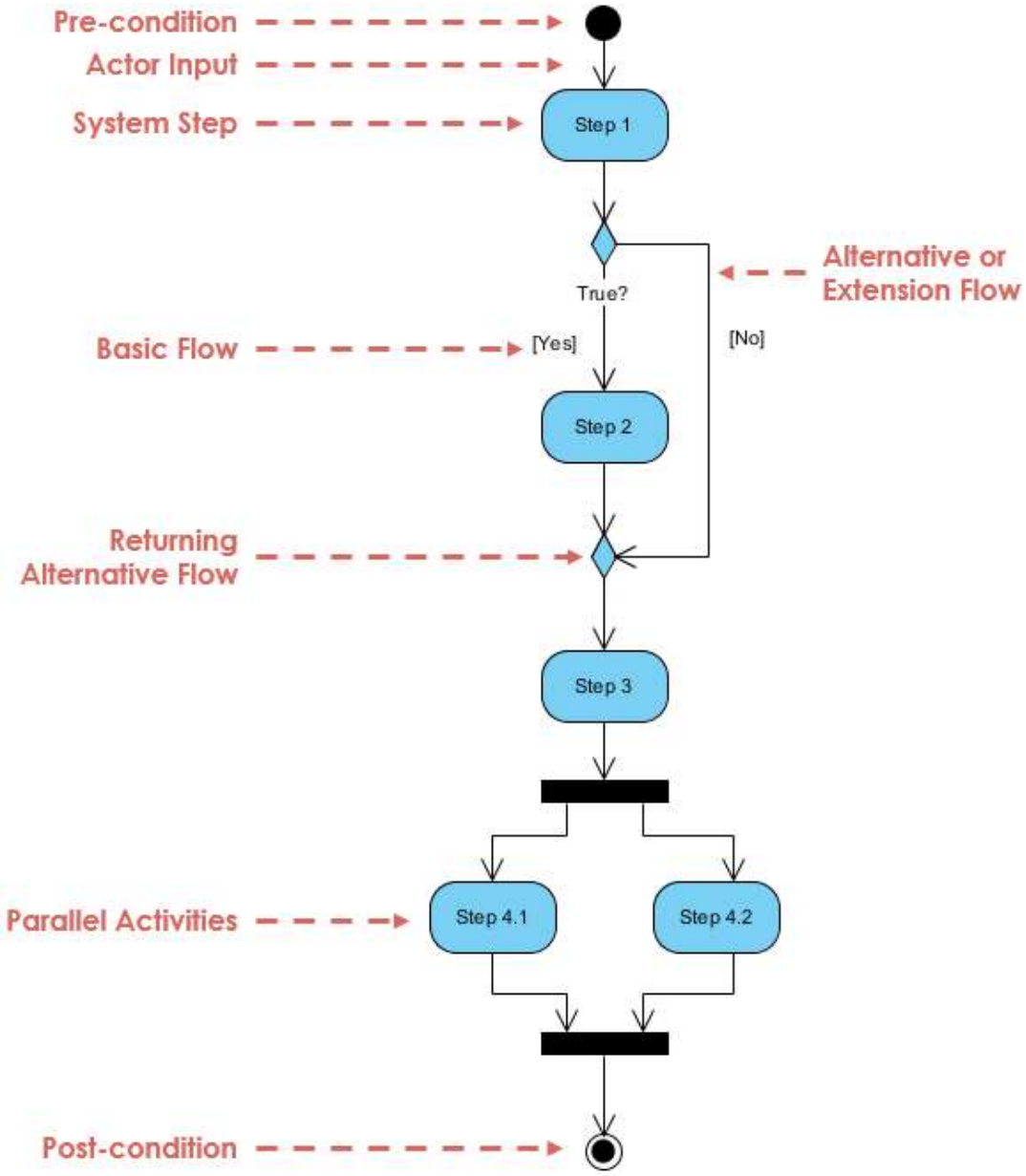
Activity diagram is another important behavioral diagram in UML diagram to describe dynamic aspects of the system.



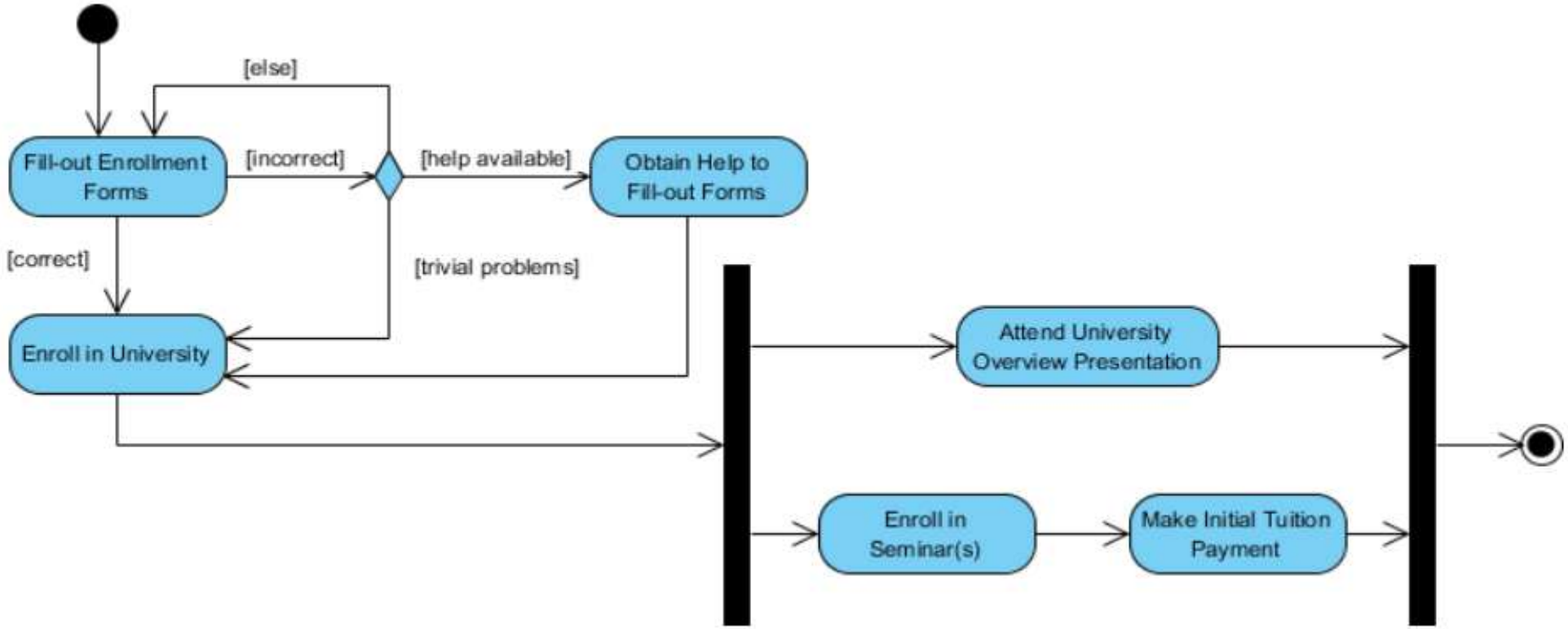
Activity Diagram

Activity diagram is essentially an advanced version of flow chart that modeling the flow from one activity to another activity.

- Identify candidate use cases, through the examination of business workflows
- Identify pre- and post-conditions (the context) for use cases
- Model workflows between/within use cases
- Model complex workflows in operations on objects
- Model in detail complex activities in a high level activity Diagram



- This UML activity diagram example describes a process for student enrollment in a university as follows:
  - An applicant wants to enroll in the university.
  - The applicant hands a filled out copy of Enrollment Form.
  - The registrar inspects the forms.
  - The registrar determines that the forms have been filled out properly.
  - The registrar informs student to attend in university overview presentation.
  - The registrar helps the student to enroll in seminars
  - The registrar asks the student to pay for the initial tuition.



System-Wide Requirements (SRS) Template

- This artifact captures the quality attributes and constraints that have system-wide scope. It also captures system-wide functional requirements.

→http://epf.eclipse.org/wikis/openup/core.tech.common.extend_supp/guidances/templates/systemwide_requirements_specification_B621CDD6.html

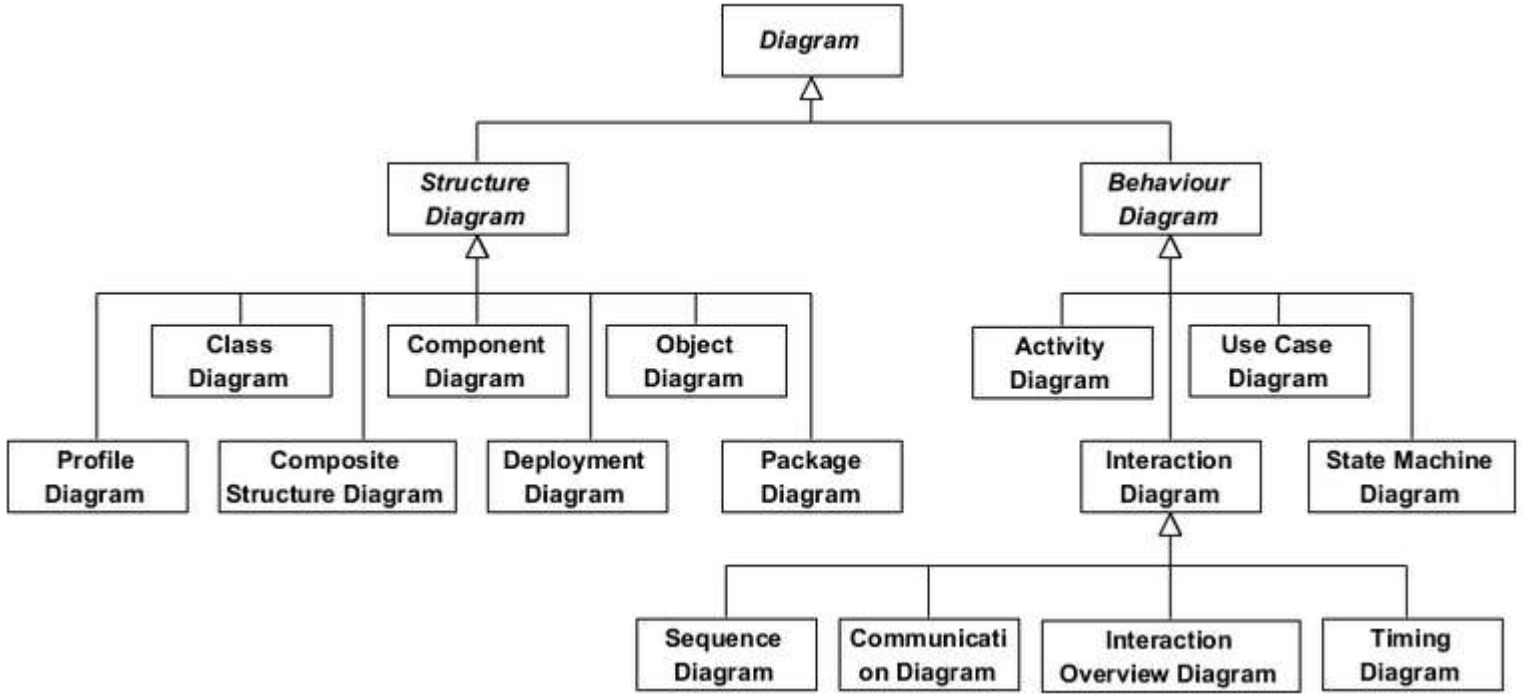
See you next week.

BBM 384
SOFTWARE ENGINEERING LAB.

UNIFIED MODELING LANGUAGE (UML)

- UML (Unified Modeling Language) is a standardized modeling language consisting of an integrated set of diagrams, developed to help system and software developers for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.
- The UML uses mostly graphical notations to express the design of software projects.

UNIFIED MODELING LANGUAGE (UML)



DIAGRAMS

Types of Diagrams

- Context diagram
- Use Case diagram
- Activity diagram

CONTEXT DIAGRAM (CD)

- A Context Diagram is a component of Functional Modelling that stands on its own as a valuable tool.
- It allows a team or an individual to produce a high-level model of an existing or planned system that defines the boundary of the system of interest and its interactions with the critical elements in its environment.

CONTEXT DIAGRAM (CONT.)

- SYSTEM of interest transforms input flows into output flows
- TERMINATOR an external function or object that is the source or destination of flows external to the system of interest
- FLOW an input or output quantity (information, control, material or energy)

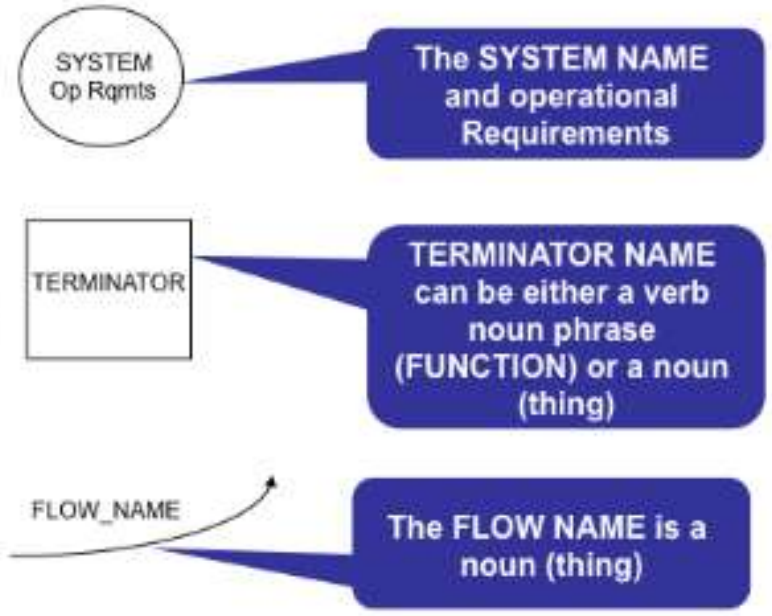
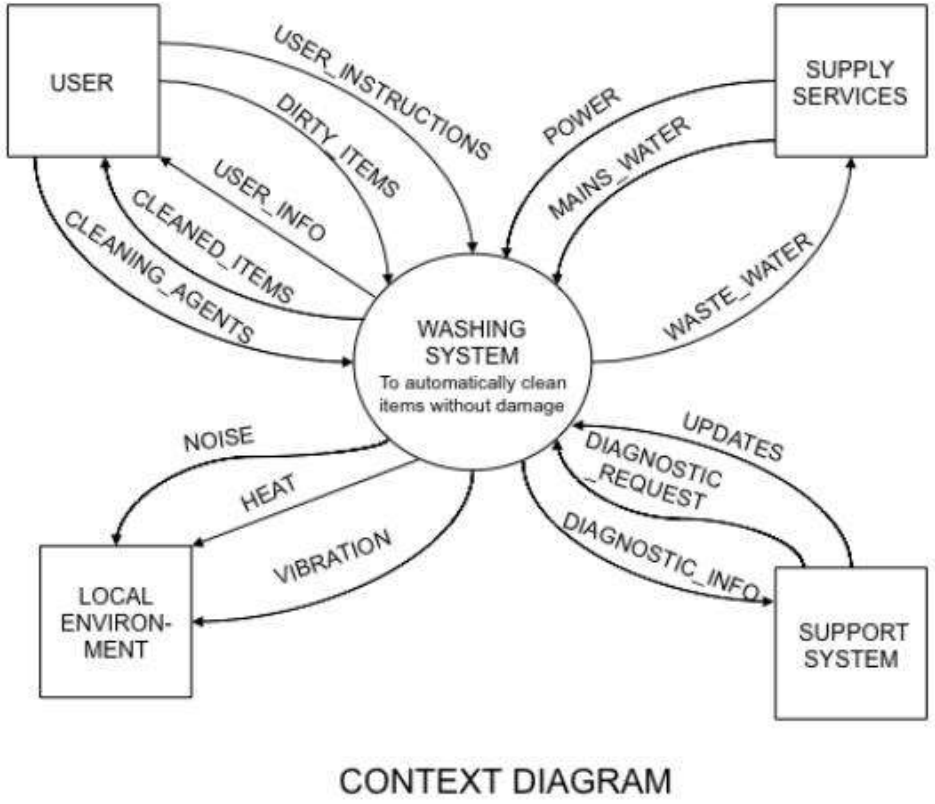


Figure 1: Context Diagram Notations

CONTEXT DIAGRAM (CONT.)



CONTEXT DIAGRAM

Figure 2: Context Diagram for a Domestic water-based Washing Machine

USE CASE DIAGRAM

- Use case diagrams model behavior within a system and helps the developers understand of what the user require. The stick man represents what's called an actor.
- Use case diagram can be useful for getting an overall view of the system and clarifying who can do and more importantly what they cant do.
- Use case diagram consists of use cases and actors and shows the interaction between the use case and actors.
- The purpose is to show the interactions between the use case and actors.
- To represent the system requirements from user's perspective.
- An actor could be the end-user of the system or an external system.

USE CASE DIAGRAM

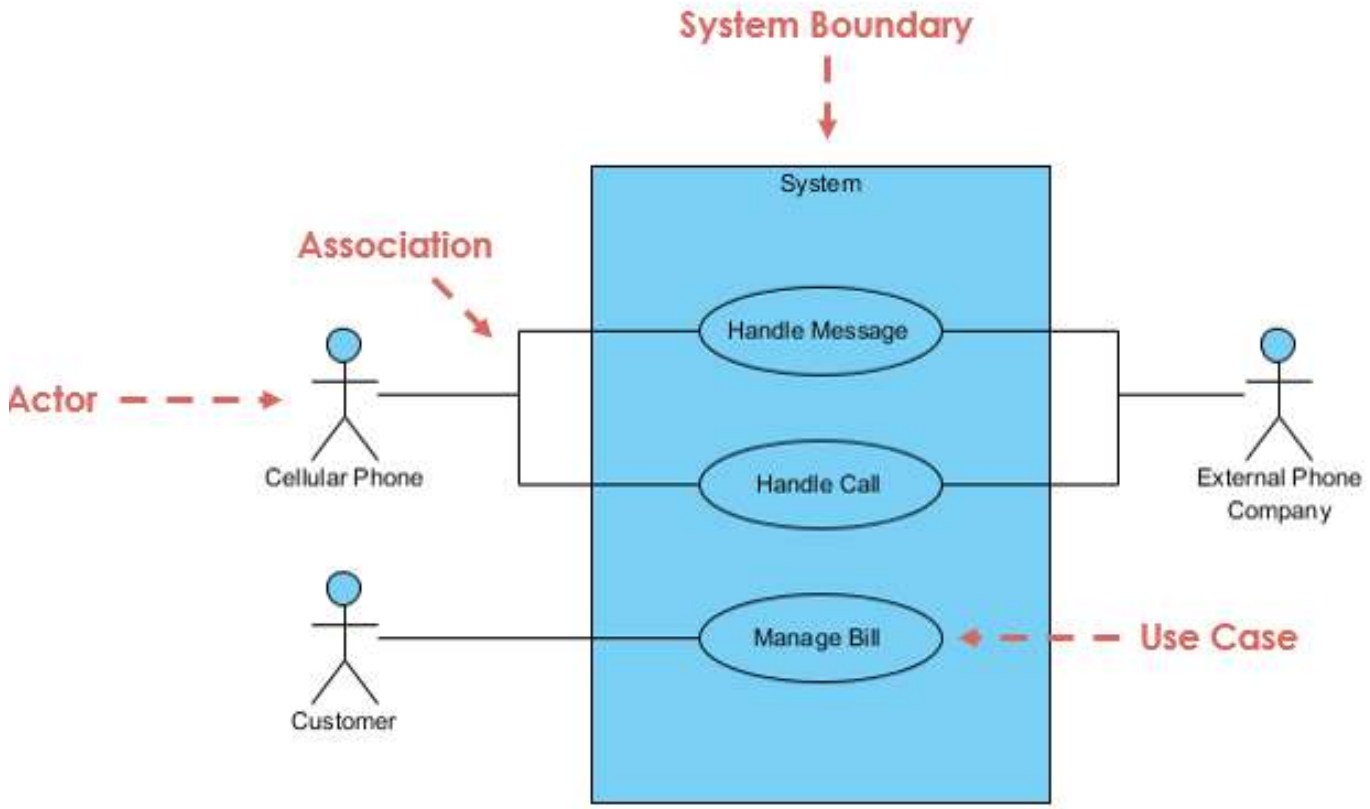


Figure 3:Use Case Diagram at a Glance

USE CASE DIAGRAM

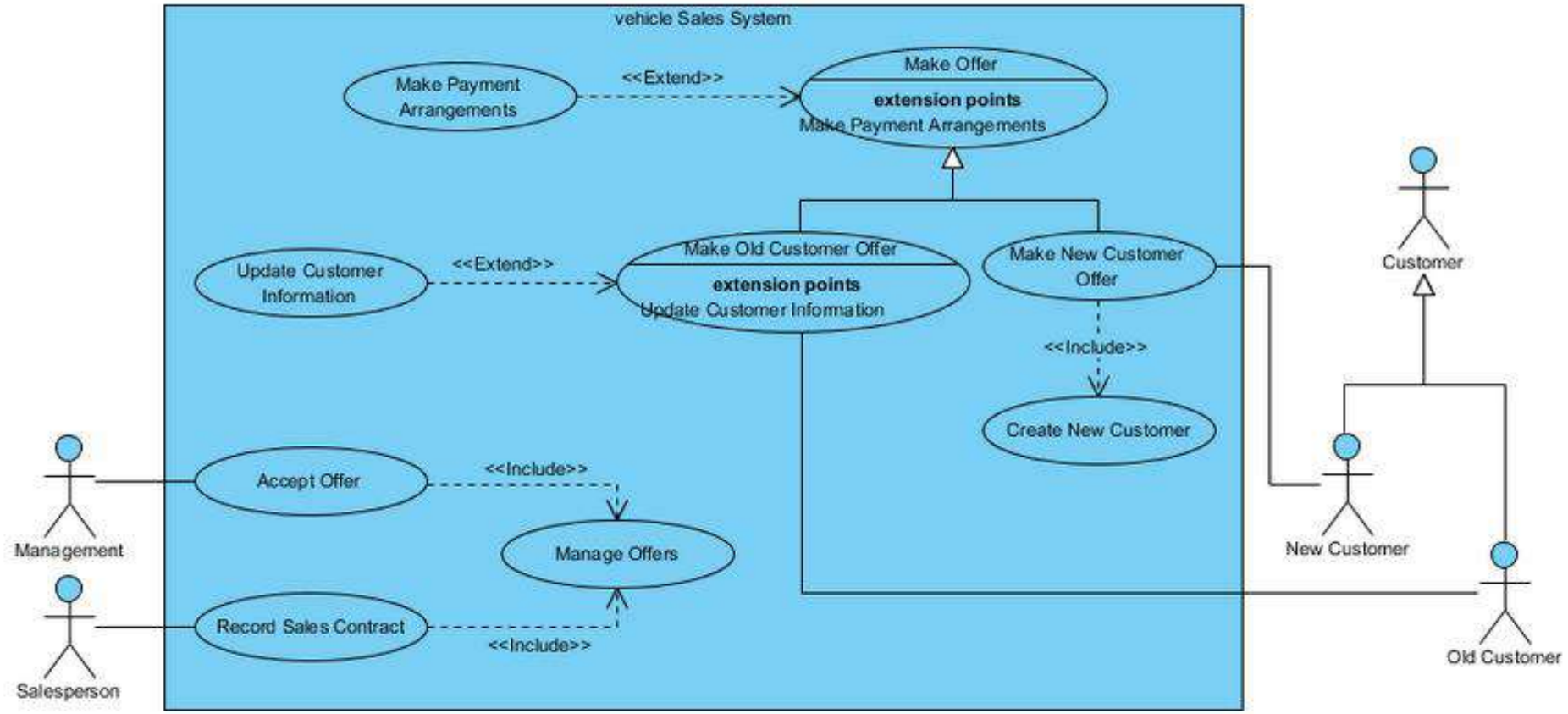


Figure 4: Vehicle Sales Systems

ACTIVITY DIAGRAM

An activity diagram visually presents a series of actions or flow of control in a system similar to a flowchart or a data flow diagram

The components can be:

- Initial node
- Activity final node
- Action
- Decision node

ACTIVITY DIAGRAM

- Identify candidate use cases, through the examination of business workflows
- Identify pre- and post-conditions (the context) for use cases
- Model workflows between/within use cases
- Model complex workflows in operations on objects
- Model in detail complex activities in a high level activity Diagram

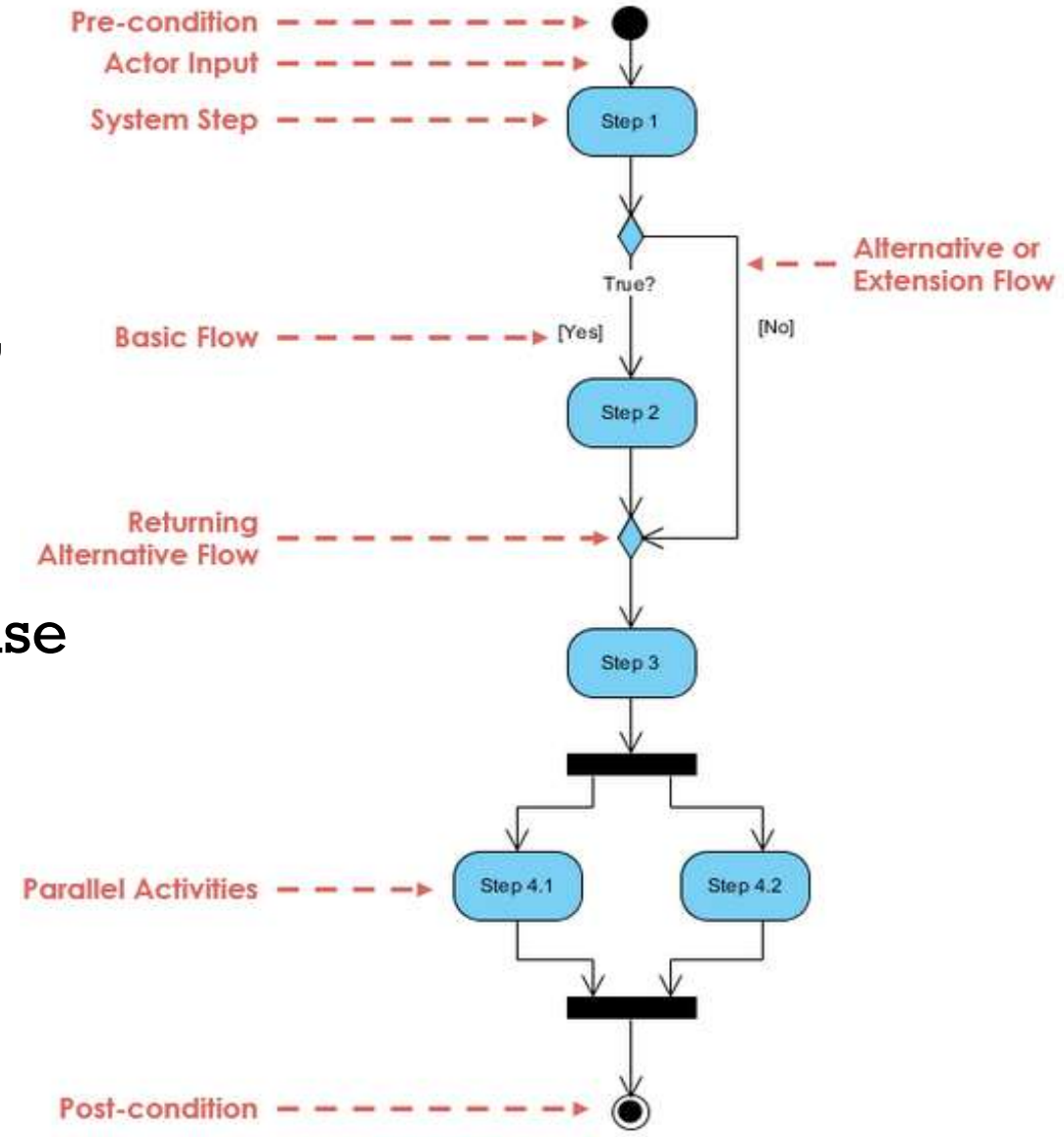


Figure 5: A basic activity diagram

ACTIVITY DIAGRAM

The activity diagram example describes the workflow for a word process to create a document

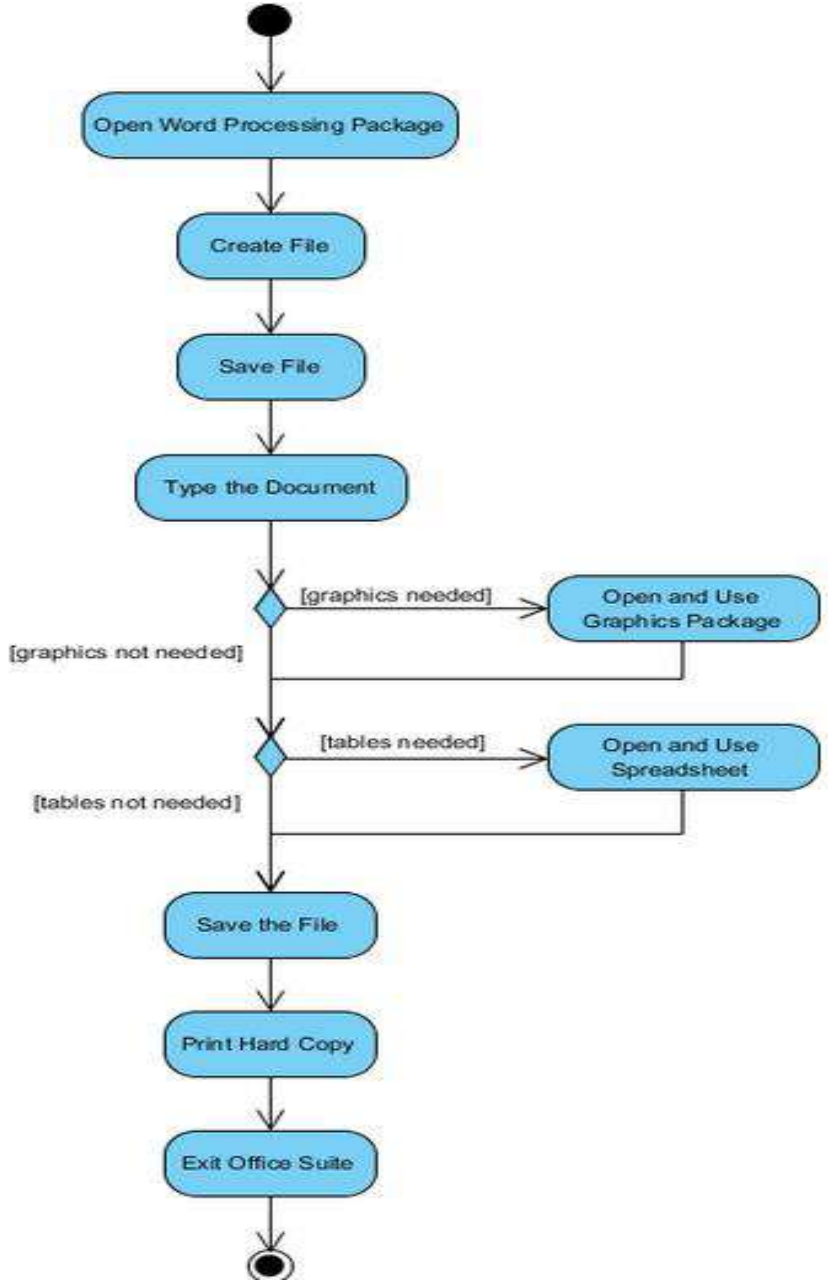


Figure 6: Activity Diagram Word Processor

EXERCISE

- The aim of this exercise is to analyze the requirements of a given system and draw three UML diagrams in the requirements phase. (Context , use case, and activity diagrams).
- The hospital management system (HMS) is designed for Any Hospital to replace their existing manual, paper based system. It is a large system including several subsystems or modules providing variety of functions. The new system is to control the following information; patient information, bed availability, appointment and admission schedules, file operations and etc. These services are to be provided in an efficient, cost effective manner, with the goal of reducing the time and resources currently required for such tasks.

SCOPE

- Can be used in any Hospital or Clinic for maintaining patient details and their test results.

PROBLEMS WITH CONVENTIONAL SYSTEM

- Lack of immediate retrievals
- Lack of immediate information storage
- Lack of prompt updating
- Error prone manual calculation
- Preparations of accurate and prompt reports

ADVANTAGES OF HMS

- Immediate access of data
- Friendly user interface
- Time saving
- Saving paper work

REQUIREMENTS

- **Hospital Receptionist** schedules patient's appointments and admission to the hospital, collects information from patient upon patient's arrival and/or by phone. For the patient that will stay in the hospital ("inpatient") she or he should have a bed allotted in a ward. Receptionists might also receive patient's payments, record them in a database and provide receipts, file insurance claims and medical reports.
- **Hospital Doctor** performs patient examination and views test results.
- **Patient** can view his/her test results.
- **Laborant** can do tests, view test reports and send them to doctor and receptionist.

PURPOSE 1

- Draw the context diagram of HMS.

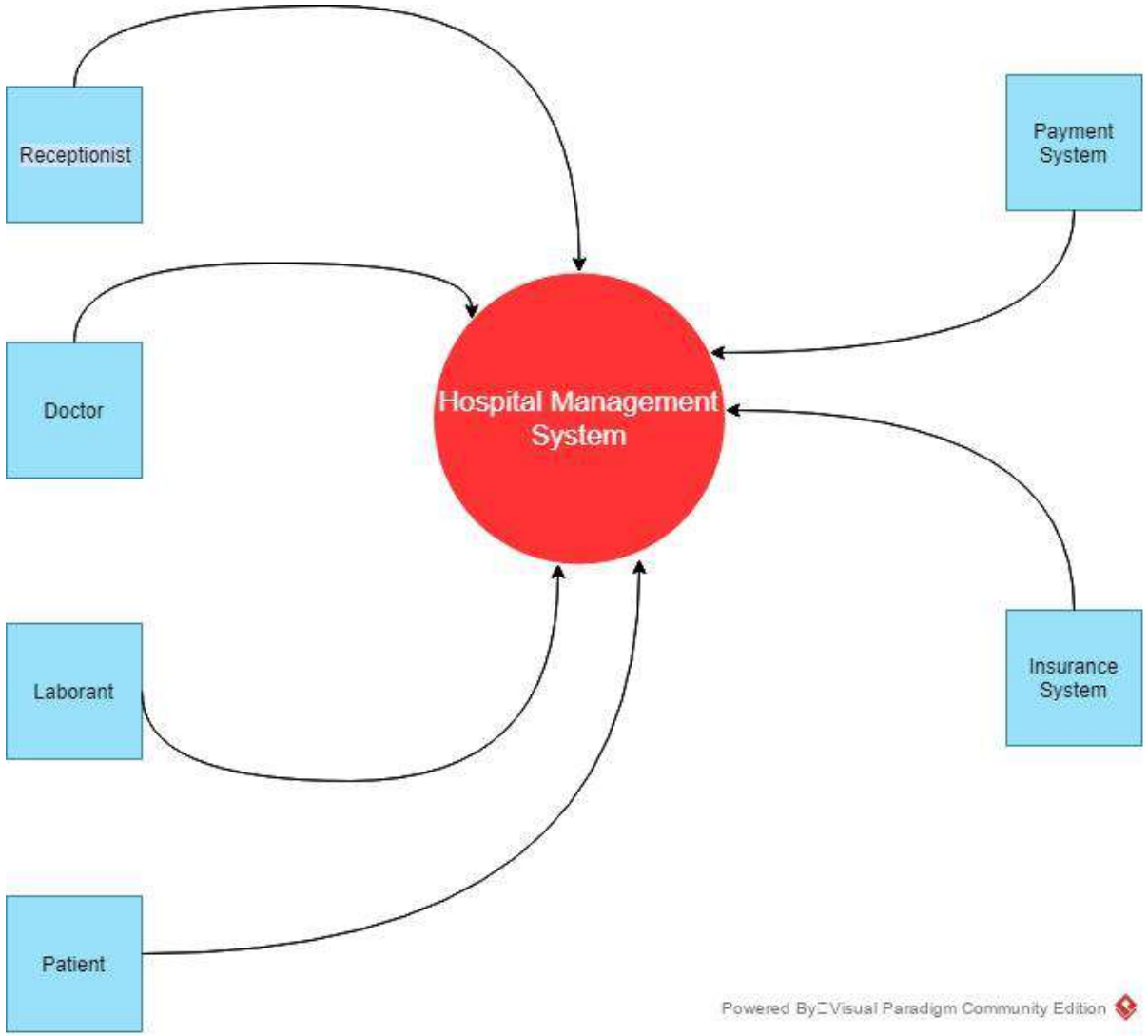


Figure 7. Context diagram of HMS

PURPOSE 2

- Describe major services provided by a hospital's reception, doctor, patient and laborant by using Use Case diagram. Shows actors, use cases and relations in your diagram.

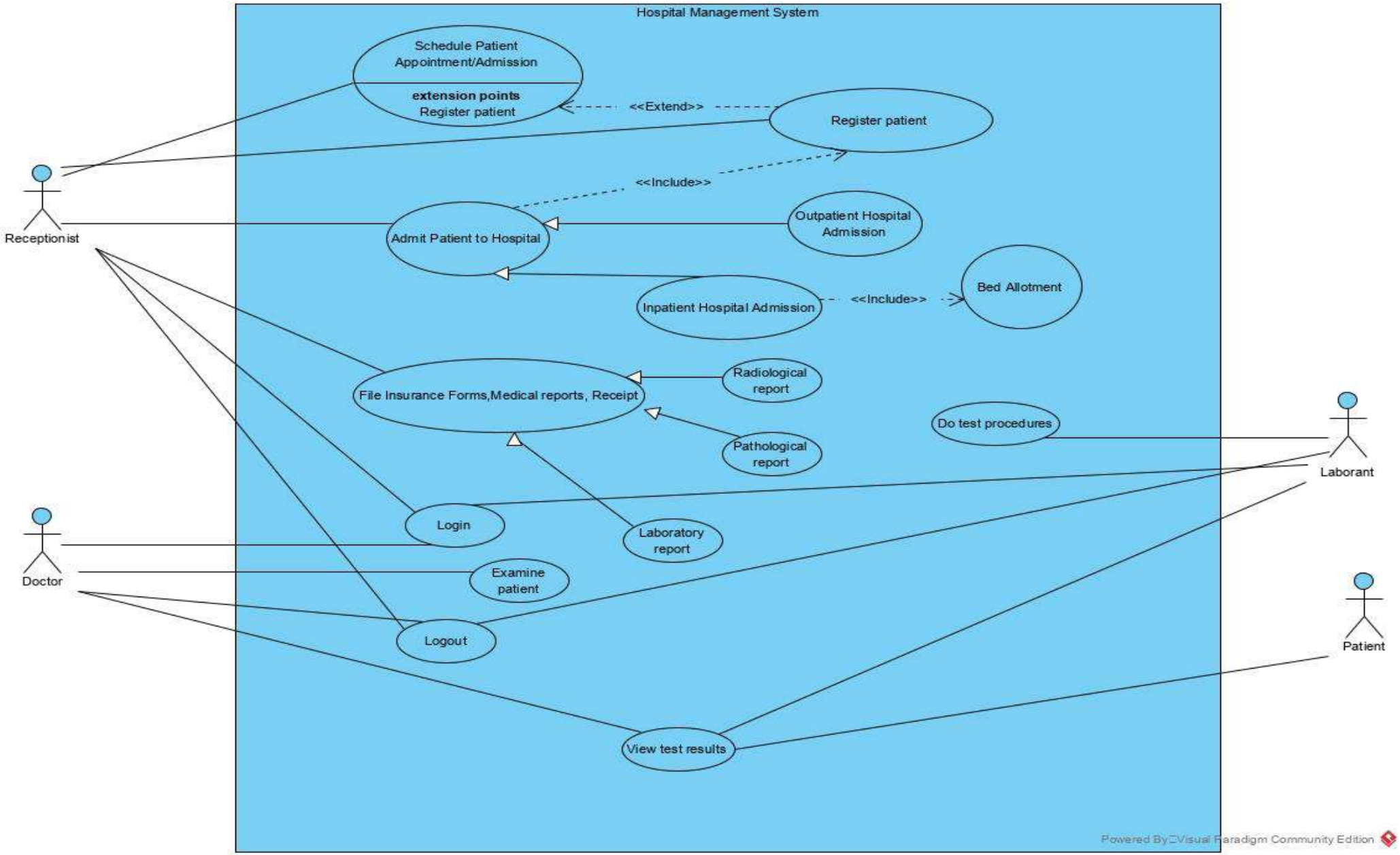


Figure 8. Use case diagram of HMS

PURPOSE 3

- Draw the activity diagram of the doctor's examination process for outpatient.

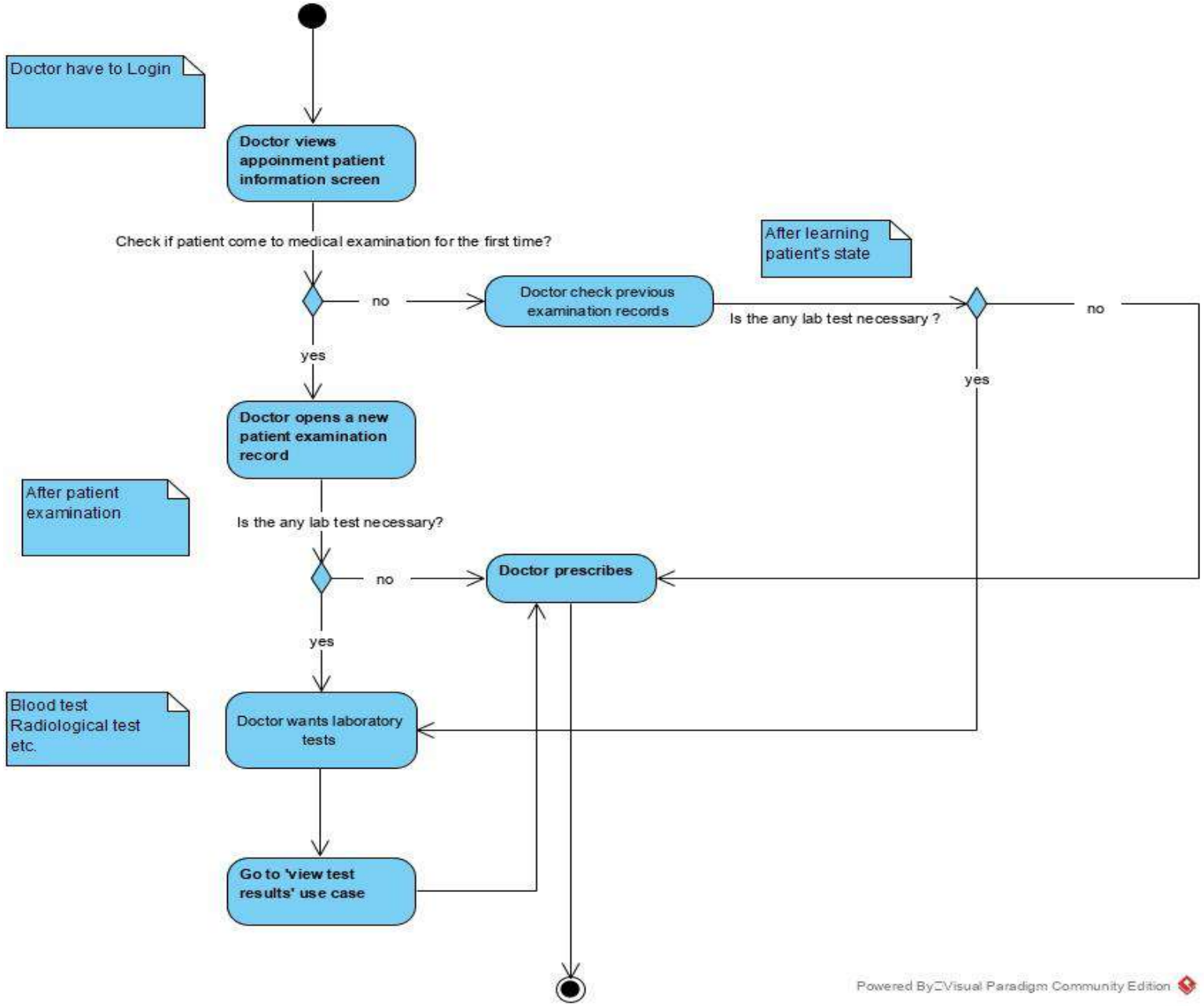


Figure 9. Activity diagram of patient examination

PURPOSE 4

- Write tabular descriptions of Patient Examination use case.

PURPOSE 4

- Structured Natural Language
- The requirements are written using numbered sentences in natural language.
- Only outside behaviour of the systems as observed by the user(s).

Use case:	
Actors	
Pre-condition	
Post-condition	
Main (happy) path:	
Alternative path:	

Use case: Examine patient	
Actors	Doctor
Pre-condition	Appointment patient screen is not opened.
Post-condition	Appointment patient screen is opened.
Main (happy) path:	1-Doctor views the appointment patient information screen 2-Doctor opens a new patient examination record 3-After examination, doctor prescribe.
Alternative path:	1-After examination, if laboratory procedures are necessary, doctor wants laboratory tests 2-Doctor views test results of the patient 3-*continue with step-3 in main path*

Table 1. Tabular description of patient examination

BBM 384

Software Engineering Lab, 2019

Week 3&4

T.A. Nebi YILMAZ, Burcu YALÇINER and Pelin CANBAY

Introduction

1. Purpose

- The aim of this exercise is to analyze the requirements of a given system and draw three UML diagrams in the requirements phase. (Context , use case, and activity diagrams).

The hospital management system (HMS) is designed for Any Hospital to replace their existing manual, paper based system. It is a large system including several subsystems or modules providing variety of functions. The new system is to control the following information; patient information, bed availability, appointment and admission schedules, file operations and etc. These services are to be provided in an efficient, cost effective manner, with the goal of reducing the time and resources currently required for such tasks.

Scope

- Can be used in any Hospital or Clinic for maintaining patient details and their test results.

Problems with conventional system

- Lack of immediate retrievals
- Lack of immediate information storage
- Lack of prompt updating
- Error prone manual calculation
- Preparations of accurate and prompt reports

Advantages of HMS

- Immediate Access of data
- Friendly user interface
- Time saving
- Saving paper work

Requirements

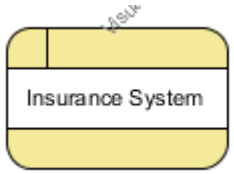
- **Hospital Receptionist** schedules patient's appointments and admission to the hospital, collects information from patient upon patient's arrival and/or by phone. For the patient that will stay in the hospital ("inpatient") she or he should have a bed allotted in a ward. Receptionists might also receive patient's payments, record them in a database and provide receipts, file insurance claims and medical reports.
- **Hospital Doctor** performs patient examination and views test results.
- **Patient** can view his/her test results.
- **Laborant** can do tests, view test reports and send them to doctor and receptionist.

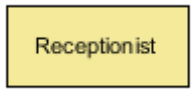
Diagrams

- Types of Diagrams
  - Context diagram
  - Use Case diagram
  - Activity diagram

Context diagram

The Context Diagram shows the system under consideration as a single high-level process and then shows the relationship that the system has with other external entities (systems, organizational groups, external data stores, etc.).

- The elements can be;
  - Process 

  - External entity 

  - Data store 

  - Connector

Purpose 1

- *Draw the context diagram of HMS.*

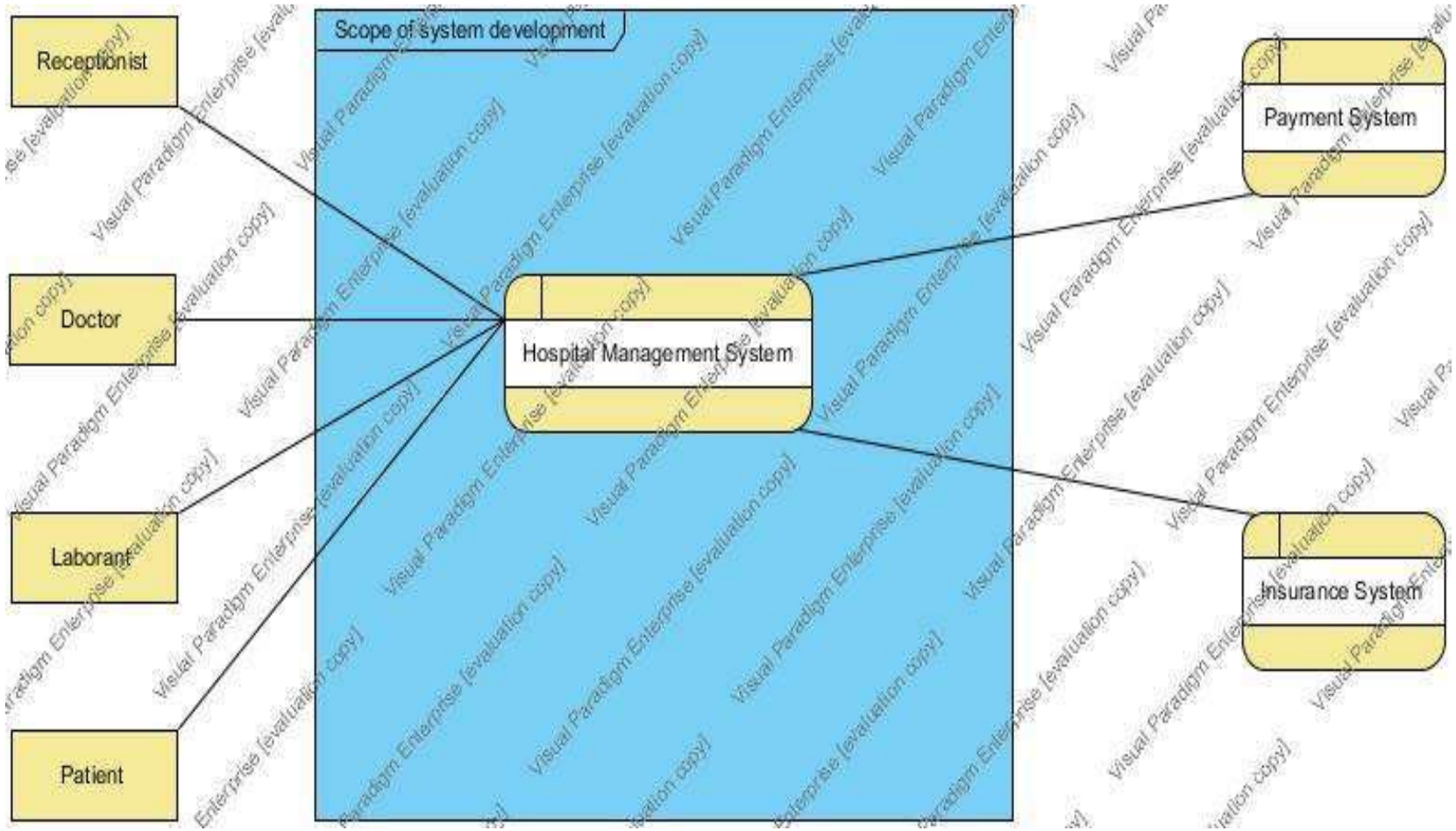


Figure 1. Context diagram of HMS

Purpose 2

- *Describe major services provided by a hospital's reception, doctor, patient and laborant by using Use Case diagram. Shows actors, use cases and relations in your diagram.*

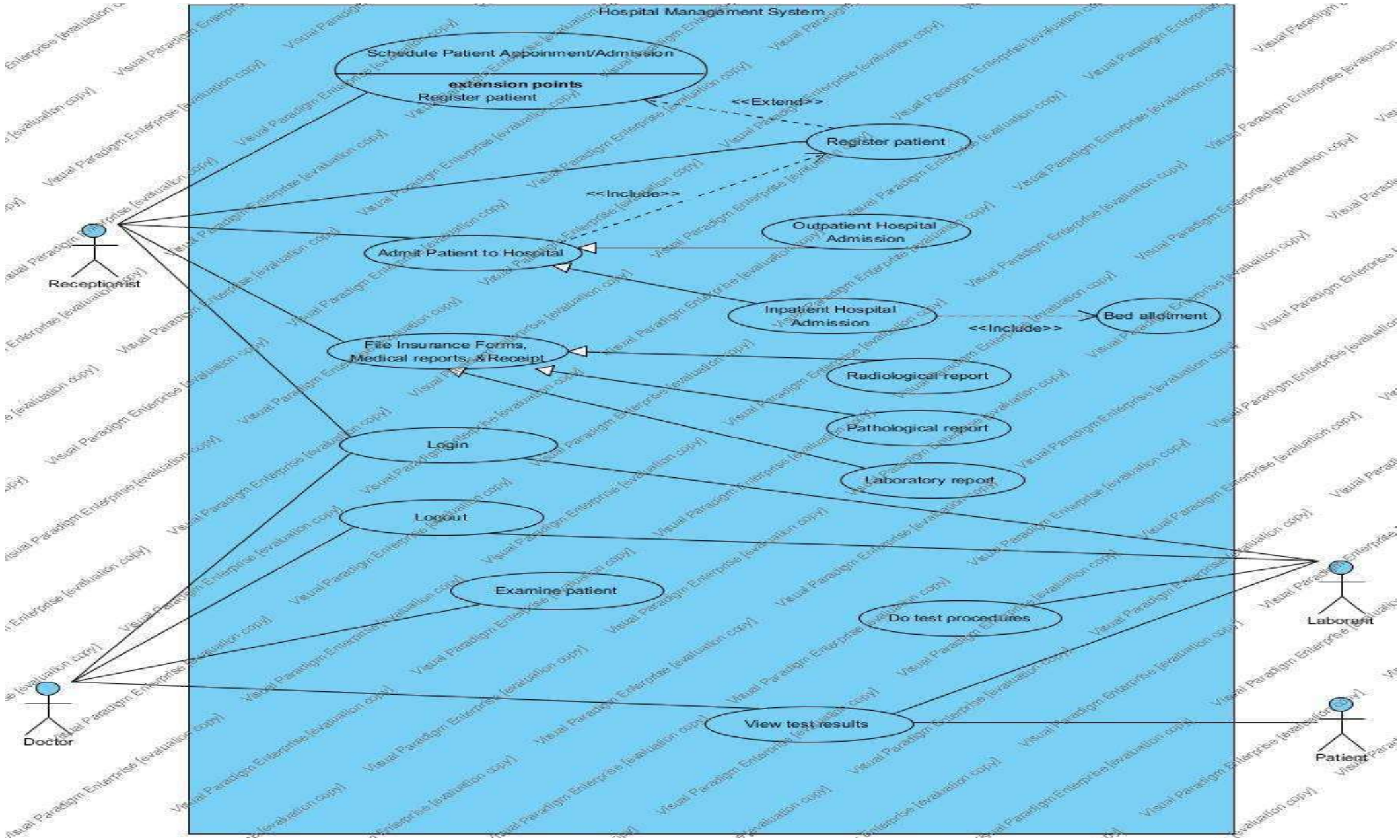
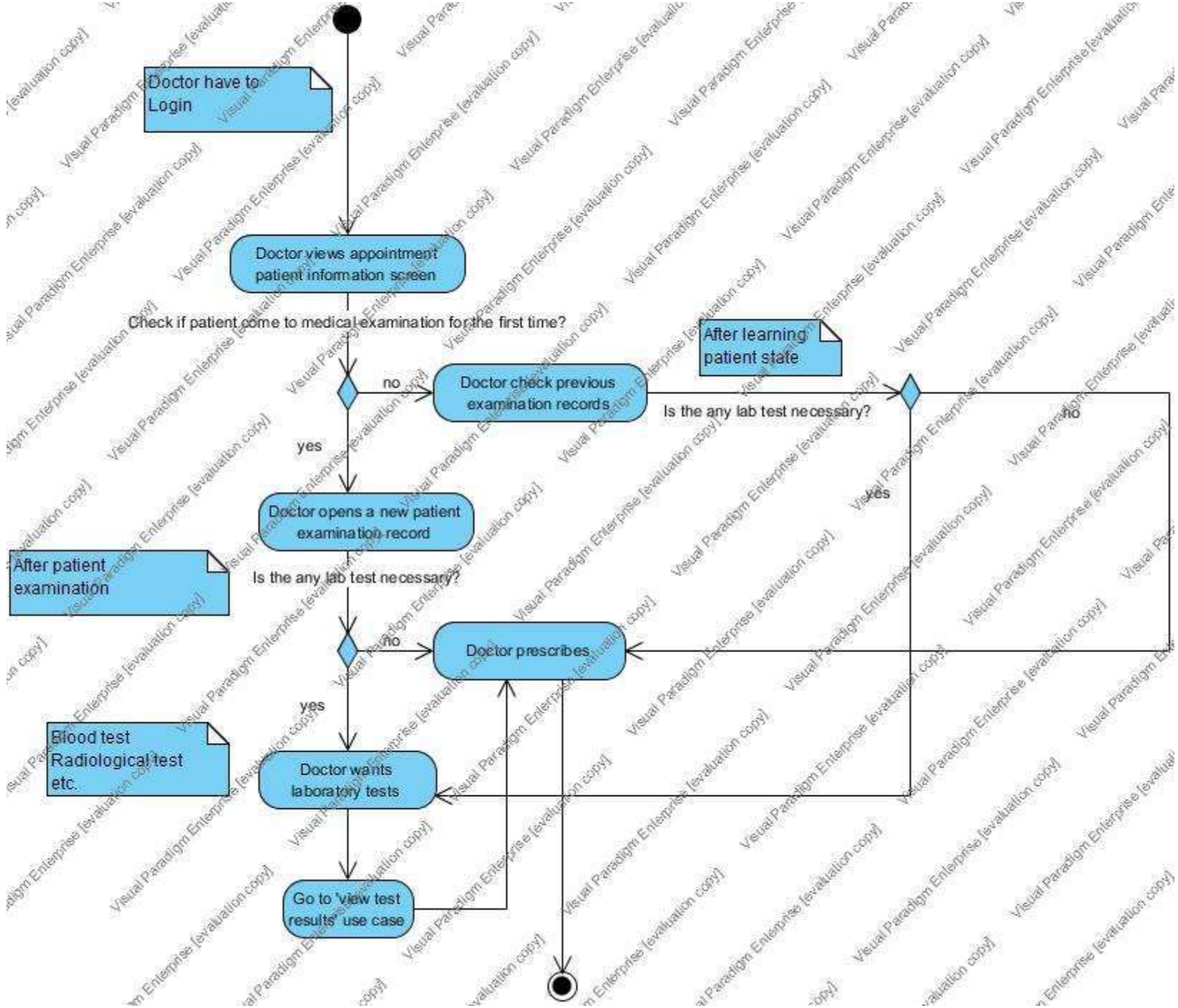


Figure 2. Use case diagram of HMS

Purpose 3

- *Draw the activity diagram of the doctor's examination process for outpatient.*


Doctor have to Login
Doctor views appointment patient information screen
Check if patient come to medical examination for the first time?
no
Doctor check previous examination records
After learning patient state
Is the any lab test necessary?
no
yes
yes
Doctor opens a new patient examination record
After patient examination
Is the any lab test necessary?
no
Doctor prescribes
yes
Blood test
Radiological test
etc.
Doctor wants laboratory tests
Go to 'view test results' use case

Purpose 4

- *Write tabular descriptions of Patient Examination use case.*

Purpose 4

- Structured Natural Language
- The requirements are written using numbered sentences in natural language.
- Only outside behaviour of the systems as observed by the user(s).

Use case:	
Actors	
Pre-condition	
Post-condition	
Main (happy) path:	
Alternative path:	

Use case: Examine patient	
Actors	Doctor
Pre-condition	Appointment patient screen is not opened.
Post-condition	Appointment patient screen is opened.
Main (happy) path:	1-Doctor views the appointment patient information screen 2-Doctor opens a new patient examination record 3-After examination, doctor prescribe.
Alternative path:	1-After examination, if laboratory procedures are necessary, doctor wants laboratory tests 2-Doctor views test results of the patient 3-*continue with step-3 in main path*

Table 1. Tabular description of patient examination

BBM384
Software Engineering Laboratory

Week 10 & 11

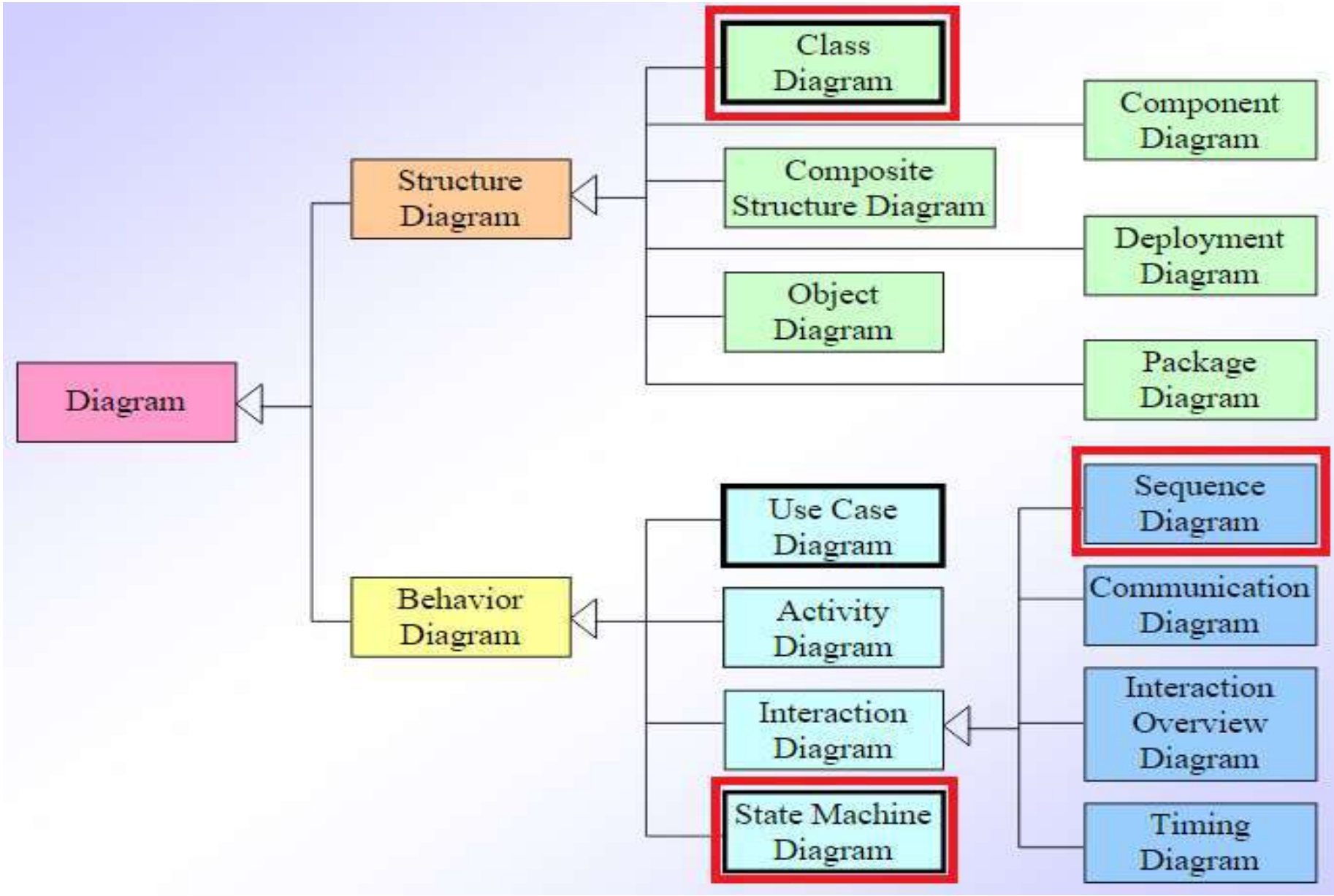
T.A. Burcu YALÇINER

Spring, 2018

UML MODELLING

(Statechart, Class, Sequence Diagrams)

UML DIAGRAMS



STATECHART DIAGRAM

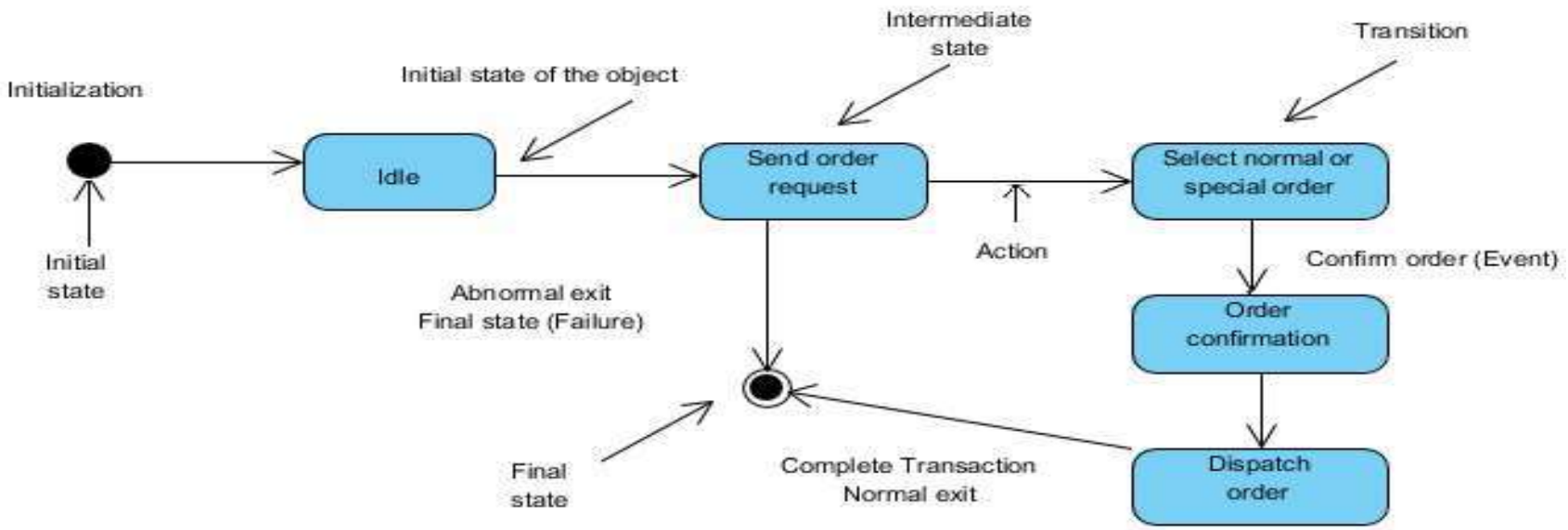
- Any real time system is expected to be reacted by some kind of internal/external events. These events are responsible for state change of the system. Statechart diagram is used to represent the event driven state change of a system. It basically describes the state change of a class, interface etc. Statechart diagram is used to visualize the reaction of a system by internal/external factors.

The main usage can be described as

- To model the object states of a system.
- To model the reactive system. Reactive system consists of reactive objects.
- To identify the events responsible for state changes.
- Forward and reverse engineering.

STATECHART DIAGRAM

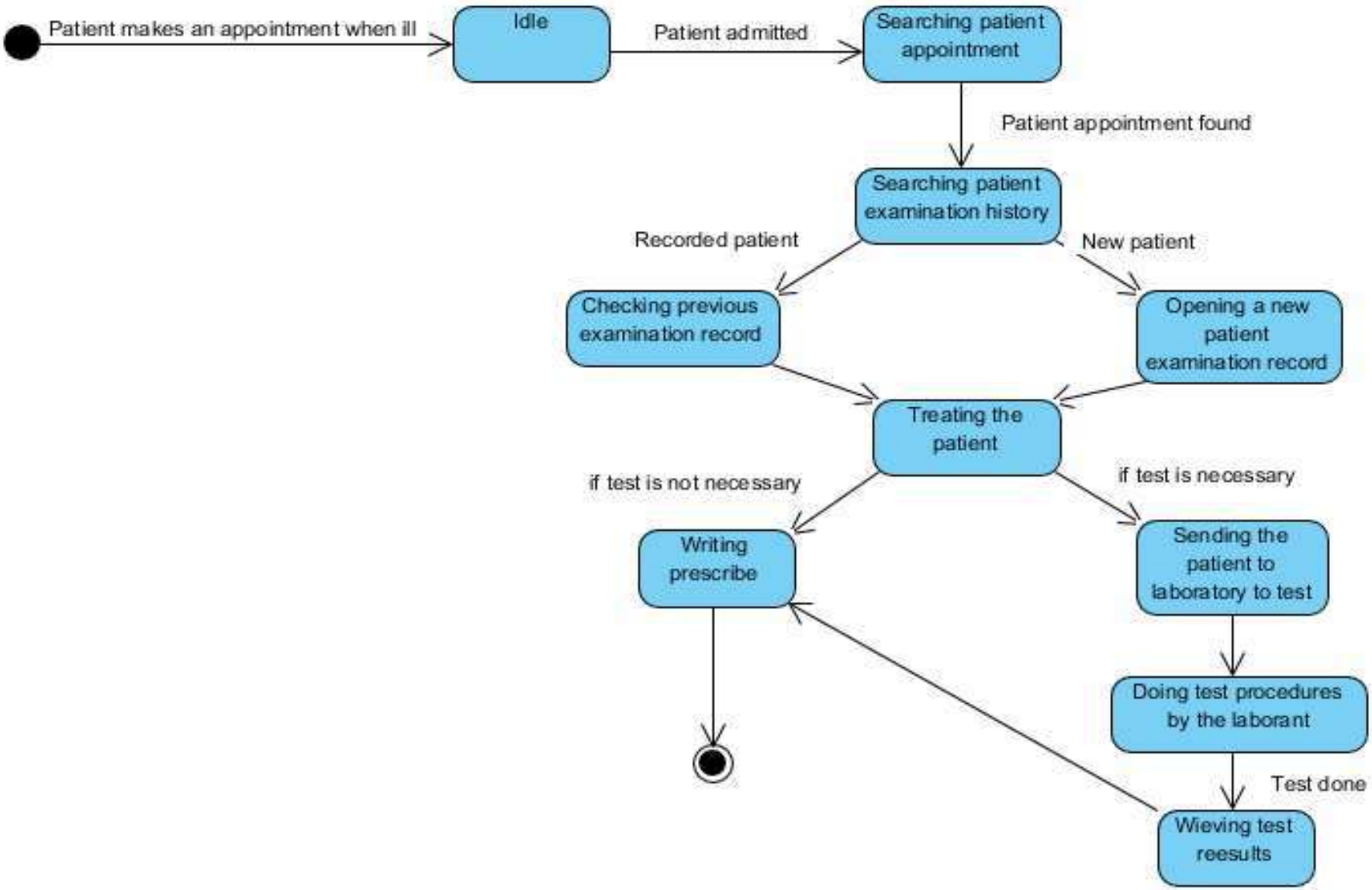
- Before drawing a Statechart diagram we should clarify the following points
  - Identify the important objects to be analyzed.
  - Identify the states.
  - Identify the events.
- Following is an example of a Statechart diagram where the state of Order object is analyzed



https://www.tutorialspoint.com/uml/uml_statechart_diagram.htm

Draw the statechart diagram for HMS

STATECHART DIAGRAM FOR HMS



CLASS DIAGRAM

- A static diagram
- Used for describing structure and behavior in the use cases
- Provide a conceptual model of the system in terms of entities and their relationships
- Class is represented with boxes which contain three parts:
  - Name → The top part contains the name of the class. It is printed in BOLD, centered and the first letter capitalized.
  - Attributes → The middle part contains the attributes of the class. They are left aligned and the first letter is lowercase.
  - Operations → The bottom part gives the methods or operations the class can take or undertake. They are also left aligned and the first letter is lowercase

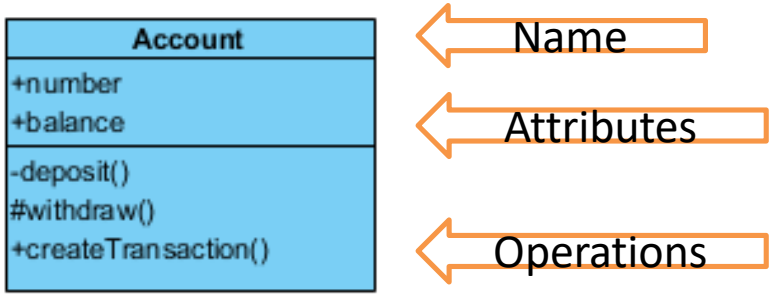
- Modifiers are used to indicate visibility of attributes and operations
  - ❖ **'+'** → **Public** visibility (everyone)
  - ❖ **'-'** → **Private** visibility (no one)
  - ❖ **'#'** → **Protected** visibility (friends and derived)

By default, attributes are hidden and operations are visible.

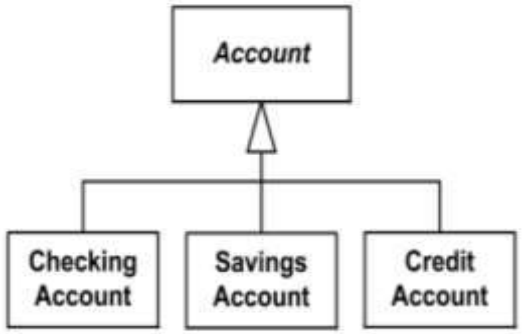
You may omit the last two compartments to simplify the class diagrams

CLASS DIAGRAM

- An example of class



- There are two kinds of Relationships
  1. Generalization
     - A parent/child relationship among related classes
     - Used for abstracting details in several layers
  2. Association
     - Relationships between instances of classes
     - A link connecting two classes
     - There are two types of association:
       - Bi-directional association
       - Uni-directional association
     - It can be further classified as:
       - Aggregation
       - Composition



In bidirectional association, two objects are related and both objects know about the relation.

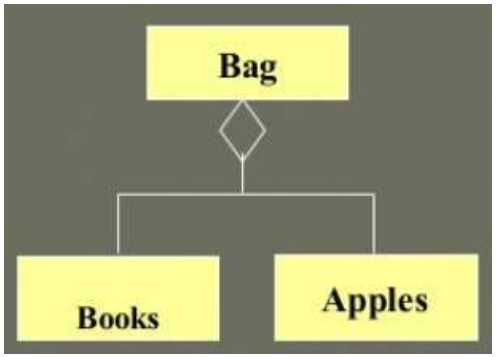
In unidirectional association, two objects are related but only one object know about the relation.

CLASS DIAGRAM

❖ Aggregation

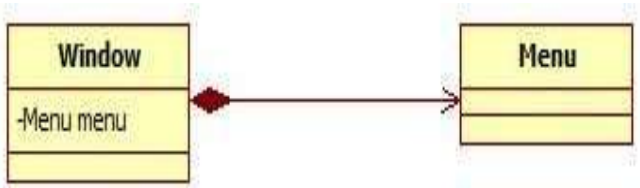
- ✓ A relationship among instances of related classes - A specific kind of **Container-Containee** relationship.
- ✓ A relationship where an instance of the Container-class has the responsibility to **hold and maintain** instances of each Containee-class that have been created outside the auspices of the Container-class.

Steering is part of car. At the same time steering has it's own existence.

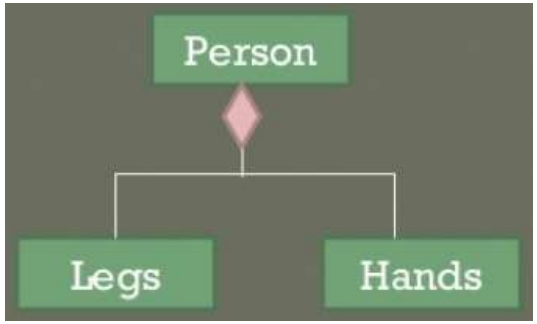


❖ Composition

- ✓ A relationship among instances of related classes - A specific kind of **Whole-Part** relationship
- ✓ A relationship where an instance of the Whole-class has the responsibility to **create and initialize** instances of each Part-class

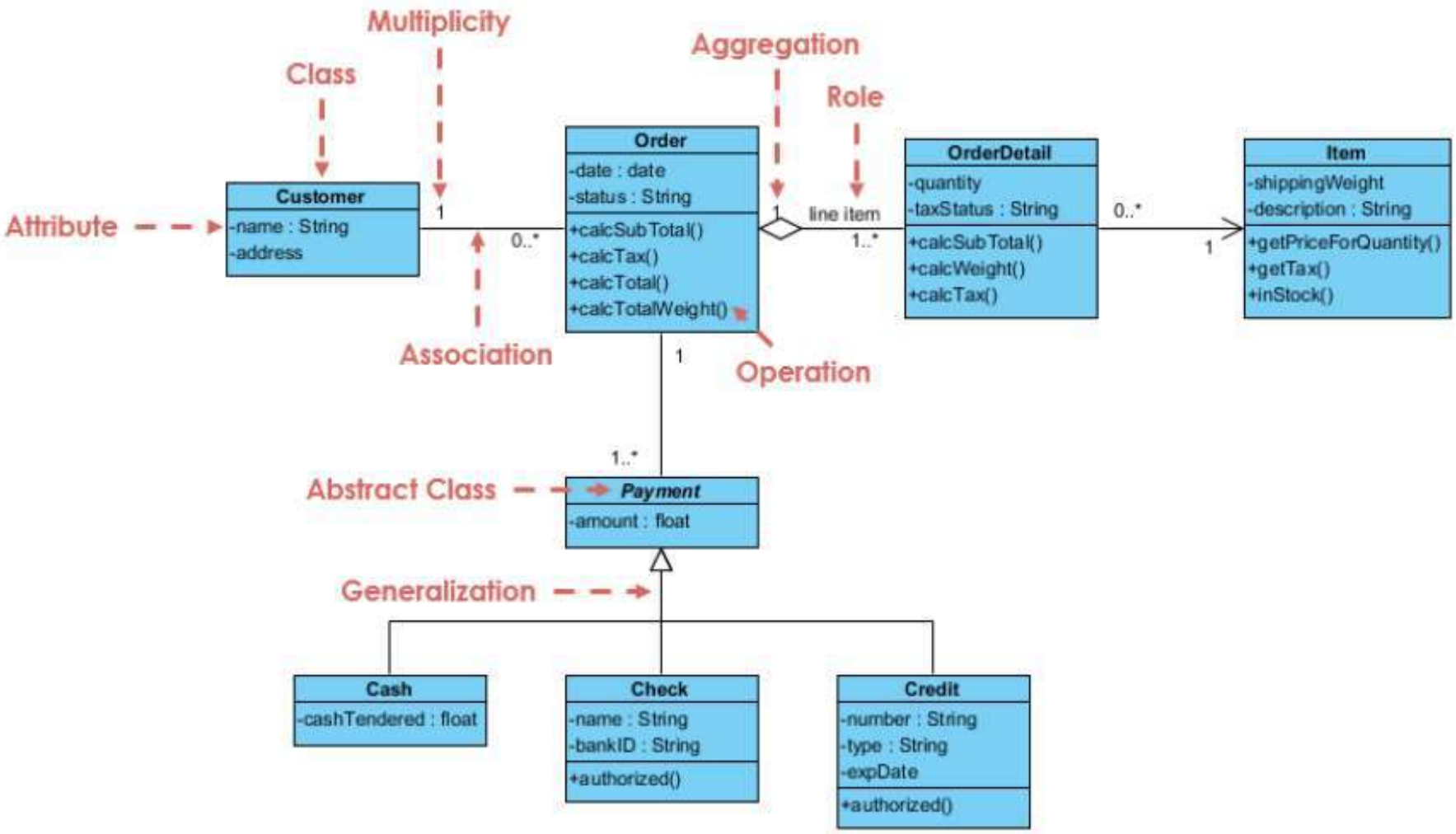


Menu is part of window. Menu exist only if windows exist. Once the window is closed the menu also get destructed.



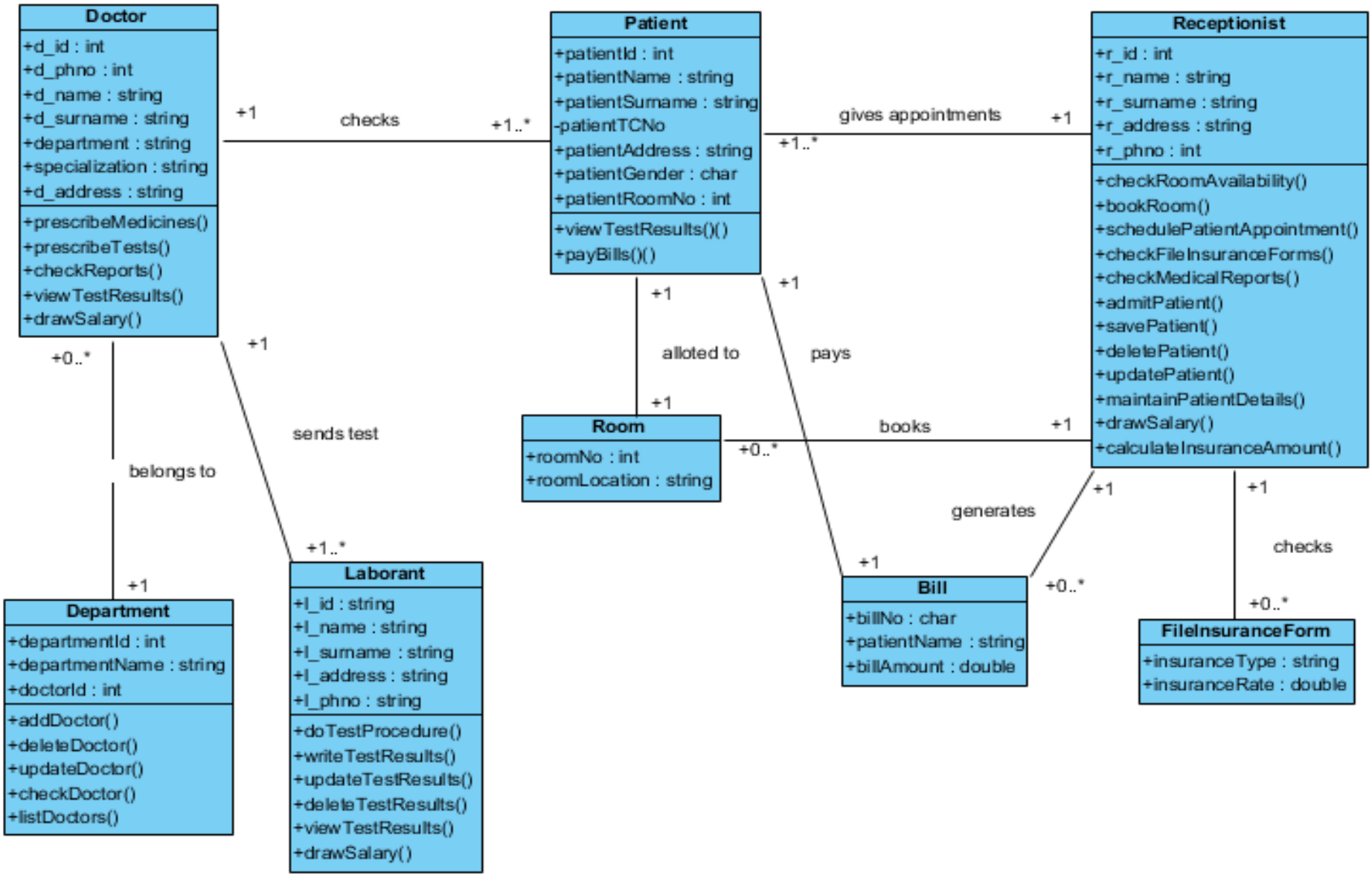
CLASS DIAGRAM

❖ **Class Diagram Example: Order System**



Draw the class diagram for HMS

CLASS DIAGRAM FOR HMS

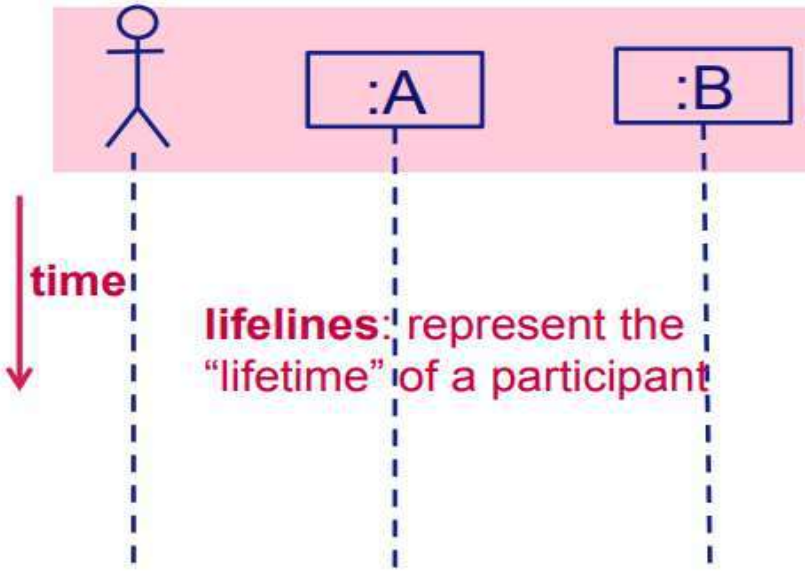


SEQUENCE DIAGRAM

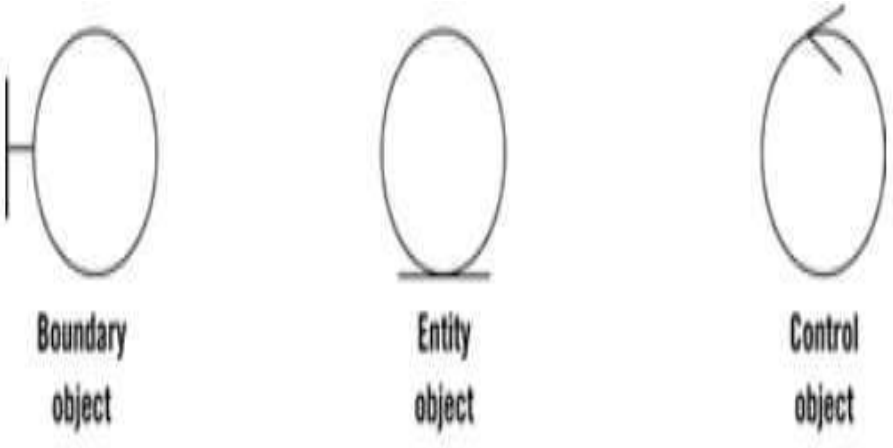
- The most common kind of Interaction Diagrams
  - shows how actors and objects interact to realize a use case scenario
  - focuses on the Message interchange between a number of Lifelines

- You draw a sequence diagram if, e.g.:
  - you have a use case diagram, to describe how the main components of the system interact
  - you have identified messages arriving at an interface of a component, to describe how the internal parts of the component interact.

SEQUENCE DIAGRAM



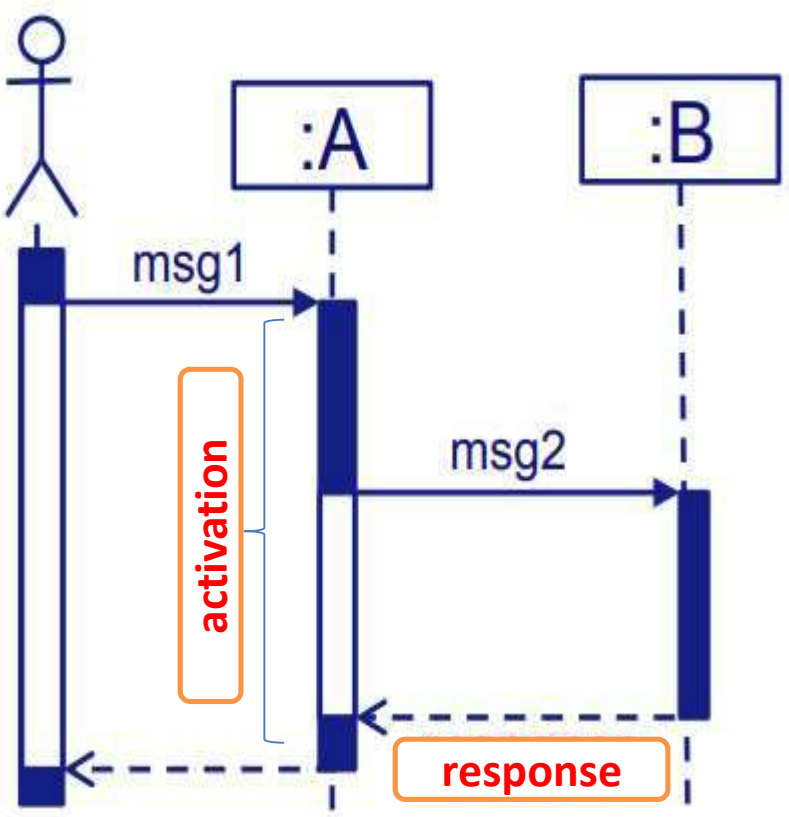
- **Participants:** actors, objects, etc. involved in the use case.
- **Kinds of participants:**
  - actor, class, database
  - boundary, entity, control
- **Notation** object:Class
  - :Class = anObject:Class



- **Boundary objects** interface with actors.
- **Entity objects** represent system data, often from the domain.
- **Control objects** glue boundary elements and entity elements, implementing the logic required to manage the various elements and their interactions.

!!! Not every system should have boundary/entity/control objects.

SEQUENCE DIAGRAM



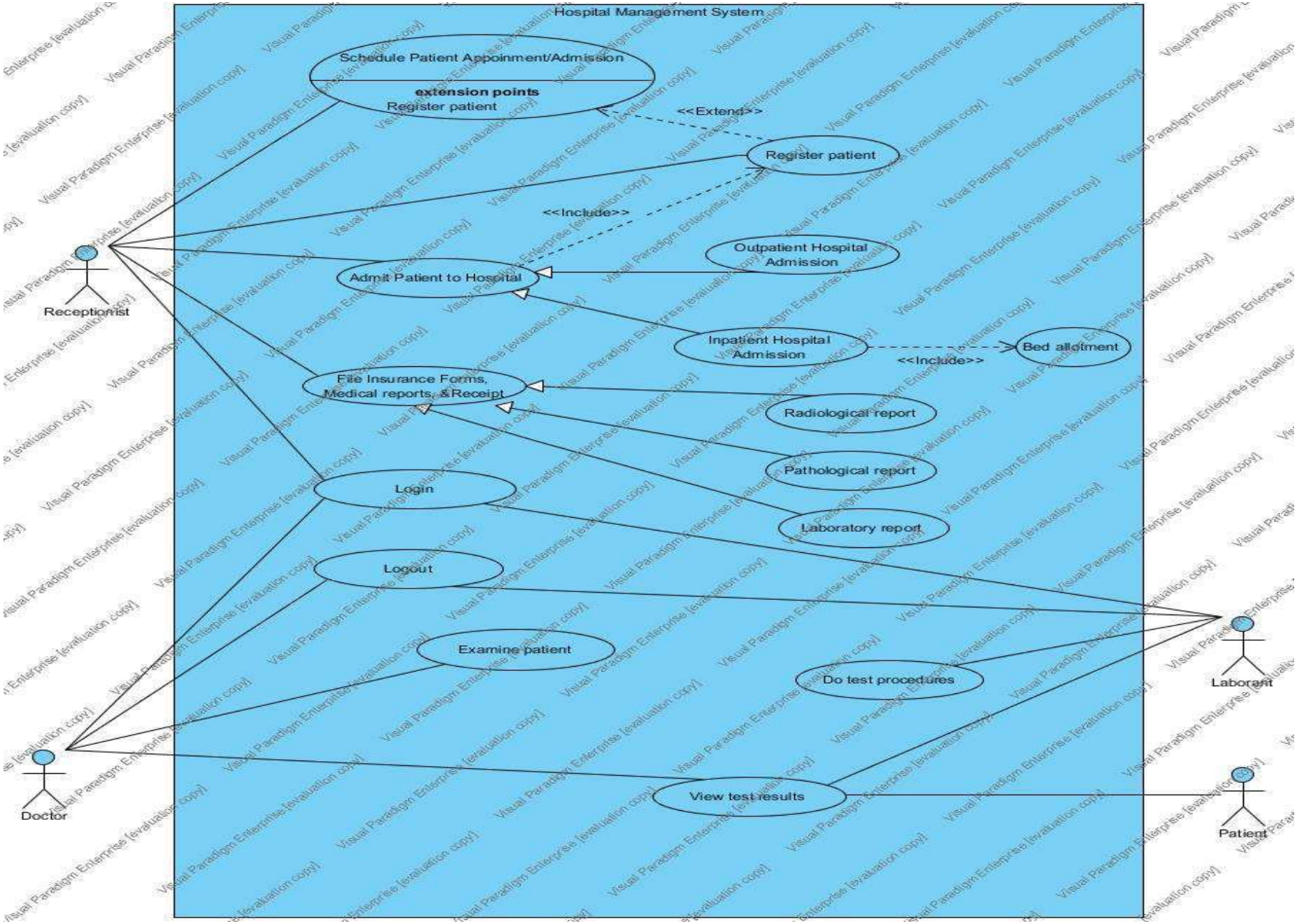
- Order of participants = order of the first participation in the interaction
- Most of the message sends on arrows from left to right
- Intuition: Messages are sent to objects of classes with the corresponding operations.
  - Class A has method msg1
  - Class B has method msg2
- When a message is received by an object, a new activation is started.
- Activation is also known as execution specification.

Draw the sequence diagram for HMS

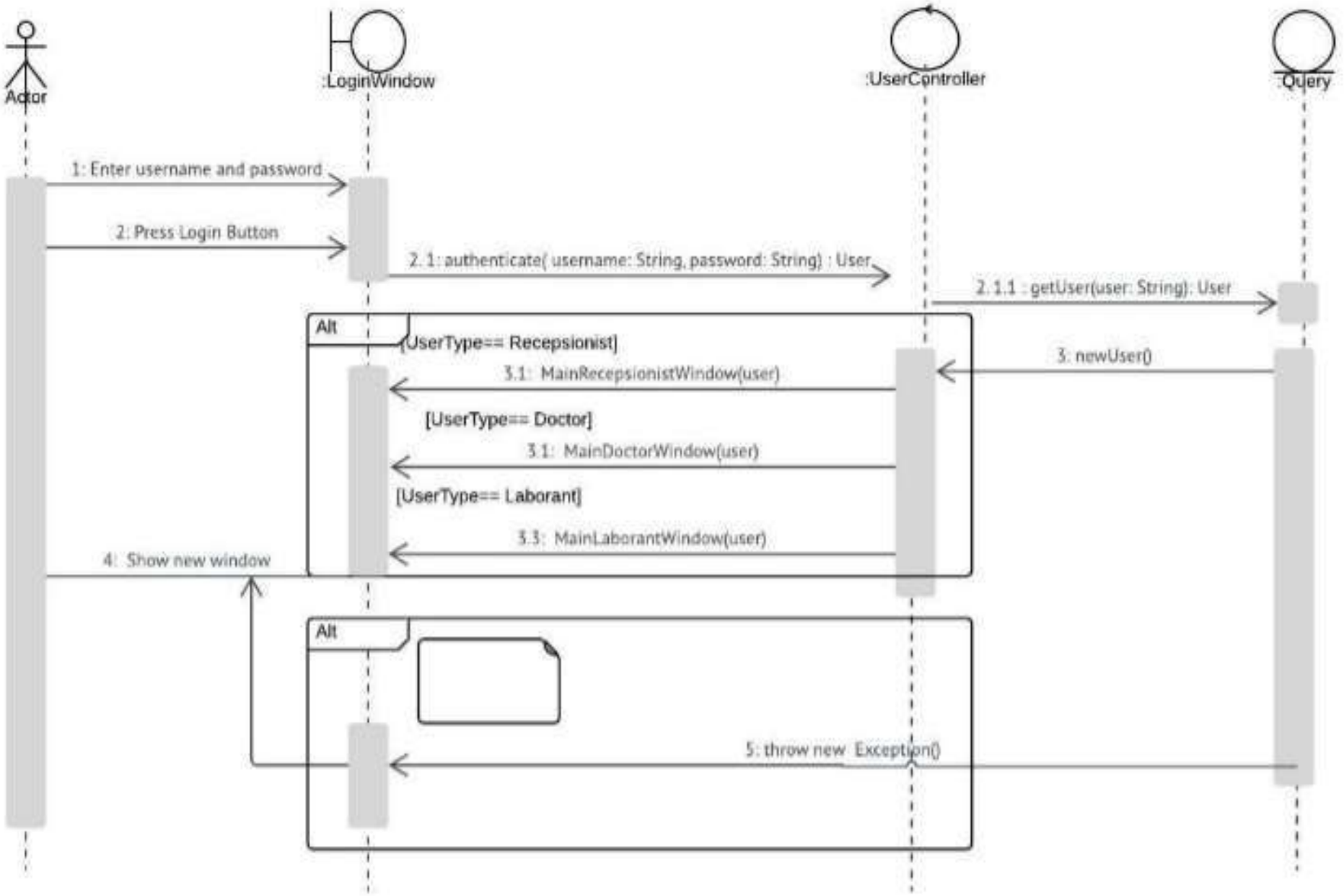
SEQUENCE DIAGRAM FOR HMS

You can draw a sequence diagram if you have a use case diagram

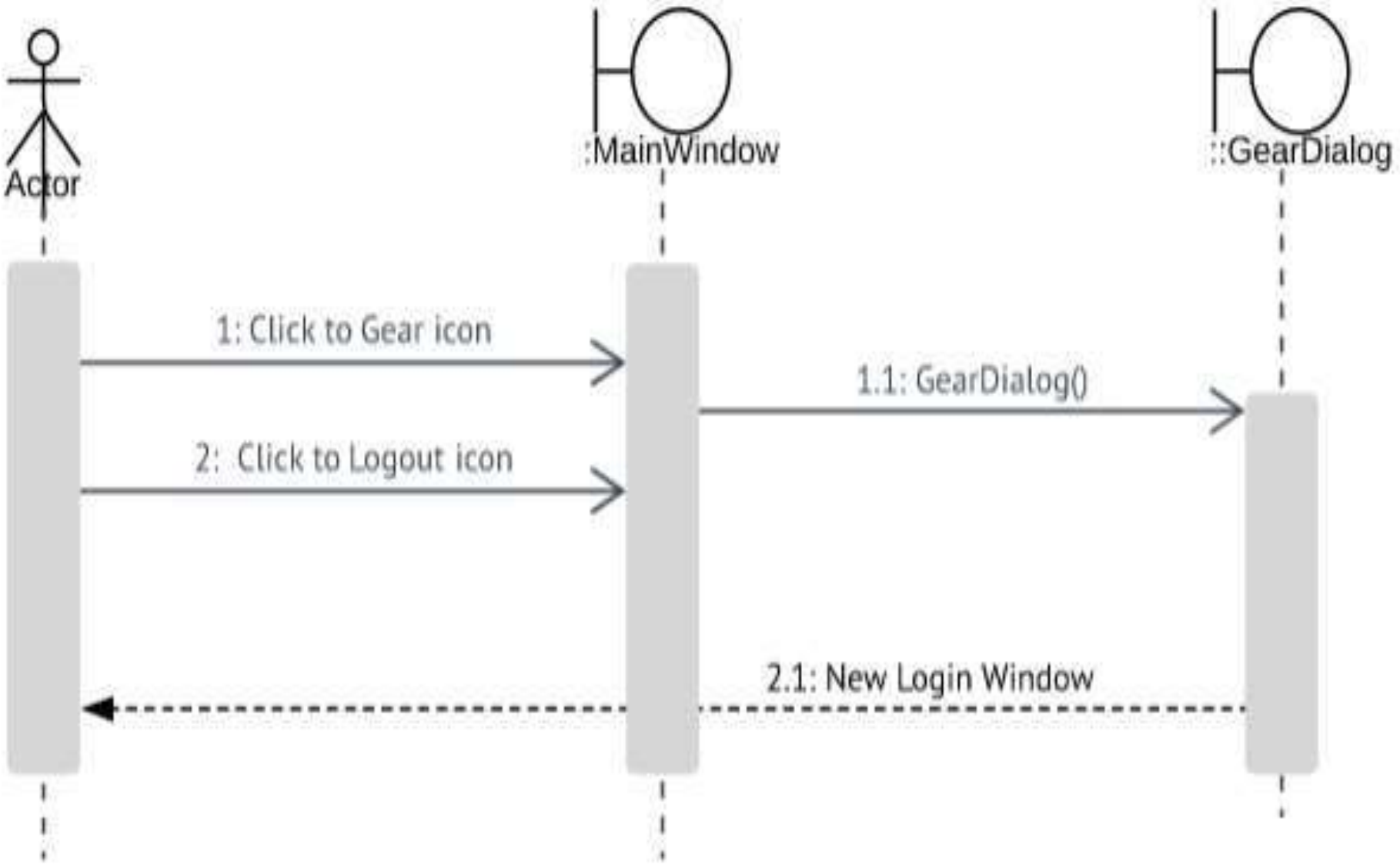
USE CASE DIAGRAM OF THE HOSPITAL MANAGEMENT SYSTEM



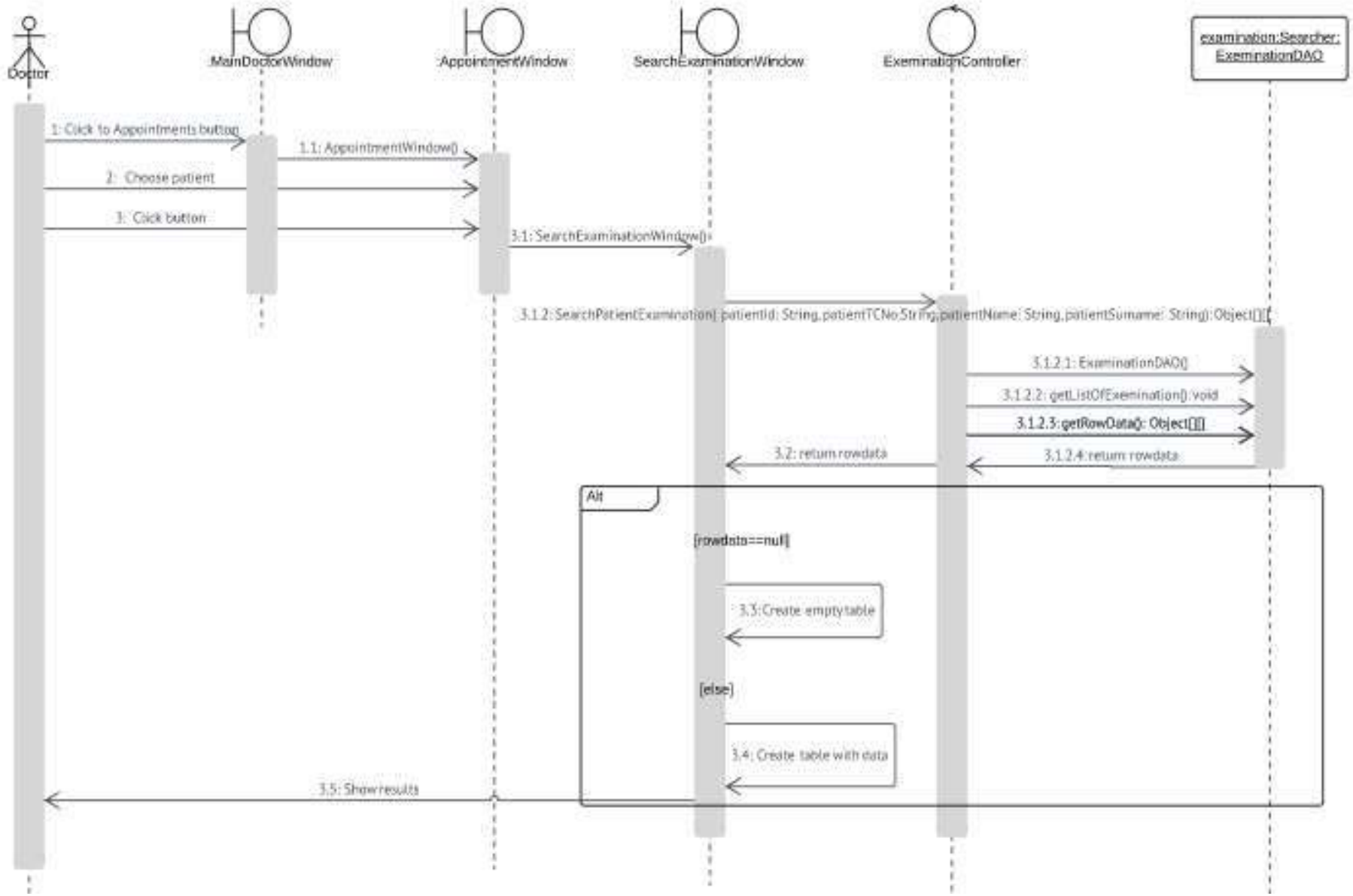
Sequence Diagram for Login Use-Case



Sequence Diagram for Logout Use-Case



Sequence Diagram for Examine Patient Use-Case



BBM 384
SOFTWARE ENGINEERING LAB.

CONTENTS

- Schema of UML Diagram Types
- Package Diagram
- Component Diagram
- Deployment Diagram
- Example Architectural & High Level Design Modeling with UML
- Software Requirements Specification (SRS)

UML DIAGRAM TYPES

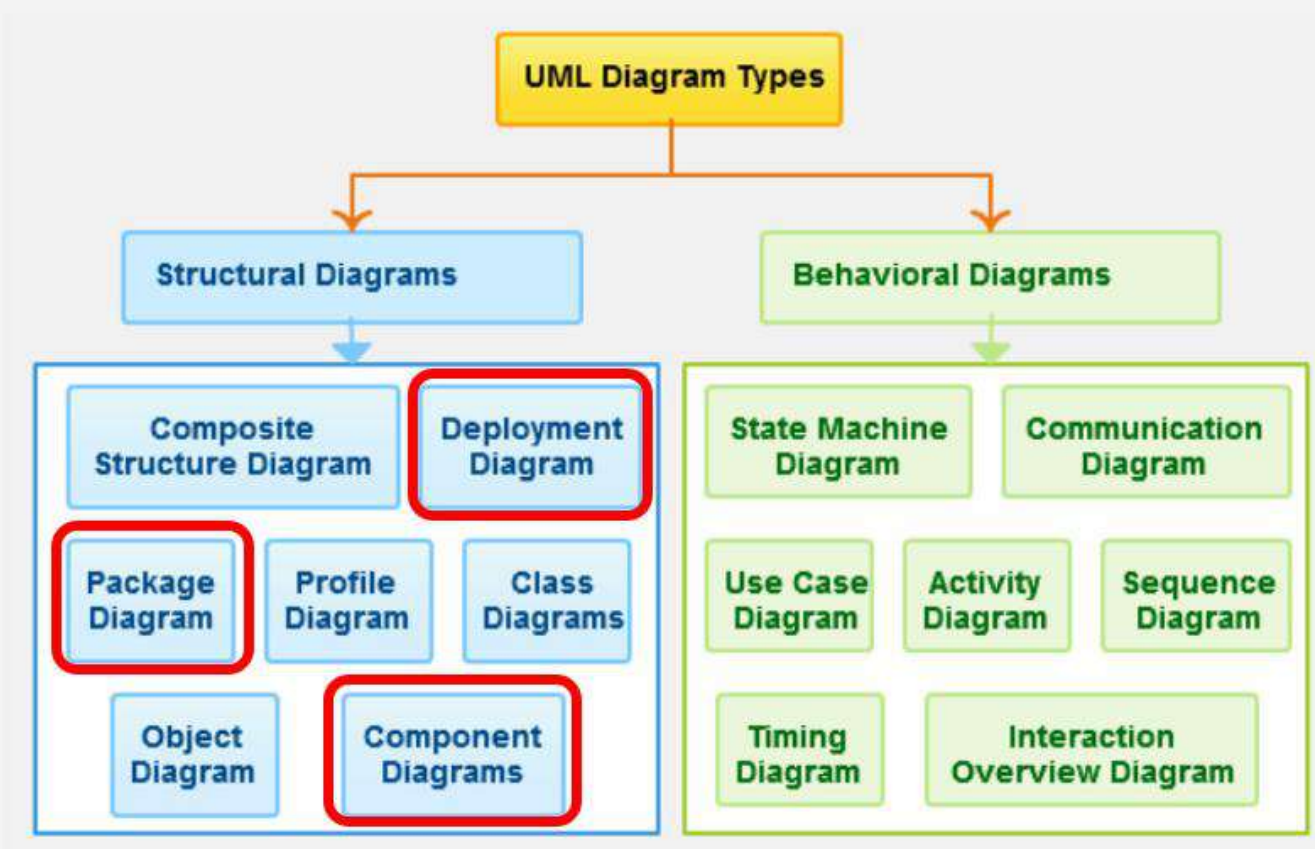


Figure 1: The schema of UML Diagram Types

PACKAGE DIAGRAM

- Package diagrams are used to structure high level system elements. Packages are used for organizing large system which contains diagrams, documents and other key deliverables.
- Package Diagram can be used to simplify complex class diagrams, it can group classes into packages.
- A package is a collection of logically related UML elements.
- Packages are depicted as file folders and can be used on any of the UML diagrams.

THE PACKAGE NOTATION

- UML 2.0 specifies that the name of the package is placed in the interior of the rectangle if the package contains no UML elements. If it does contain elements, the name should be placed within the tab.

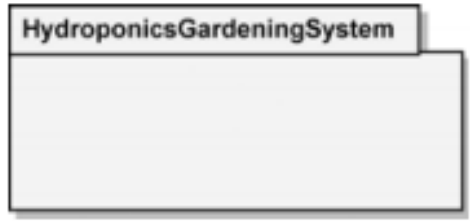


Figure 2: The Package Notation for HydroponicsGardeningSystem

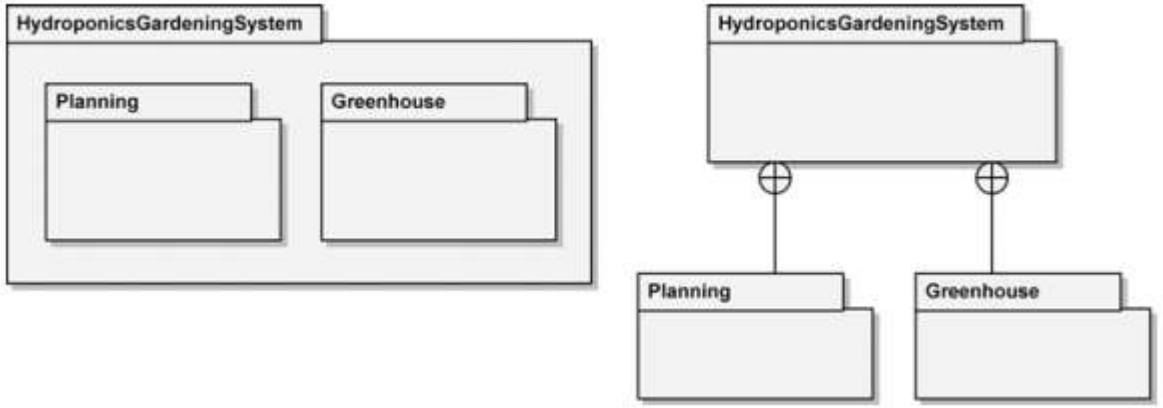


Figure 3: The Package Notation for Contained Elements

VISIBILITY OF ELEMENTS

- The visibility of the elements, defined by the containing package to be either public or private.
- The package provides the namespace, every contained element has a unique name, at least among other elements of its type.
- Public (+): Visible to elements within its containing package, including nested packages, and to external elements
- Private (-): Visible only to elements within its containing package and to nested packages

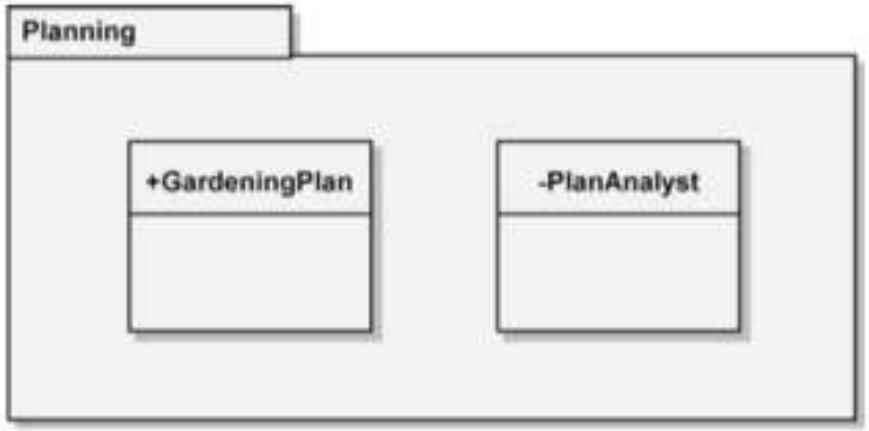


Figure 4: The Visibility of Elements within Planning Package

THE DEPENDENCY RELATIONSHIP

- A ***dependency*** shows that an element is dependent on another element as it fulfills its responsibilities within the system.
- Dependencies may be labeled to highlight the type of dependency between the elements by placing the dependency type—denoted by a keyword—within guillemets (« »),for example, ***«import»***.
- Package-specific dependencies include ***import, access,*** and ***merge***

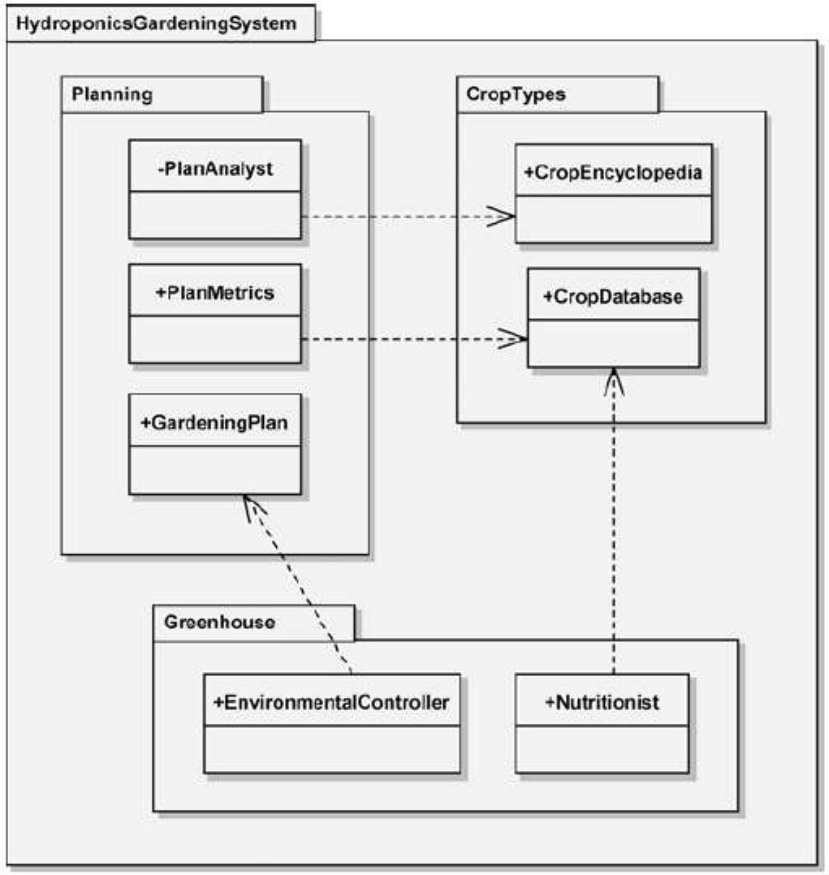


Figure 5: The Dependency Notation for HydroponicsGardeningSystem

THE DEPENDENCY RELATIONSHIP

- If multiple contained element dependencies exist between packages, these dependencies are aggregated at the package level.
- A package-level dependency may be labeled with a keyword, denoting type, inside guillemets (« »); however, if the contained dependencies are of different types, the package-level dependency is not labeled.

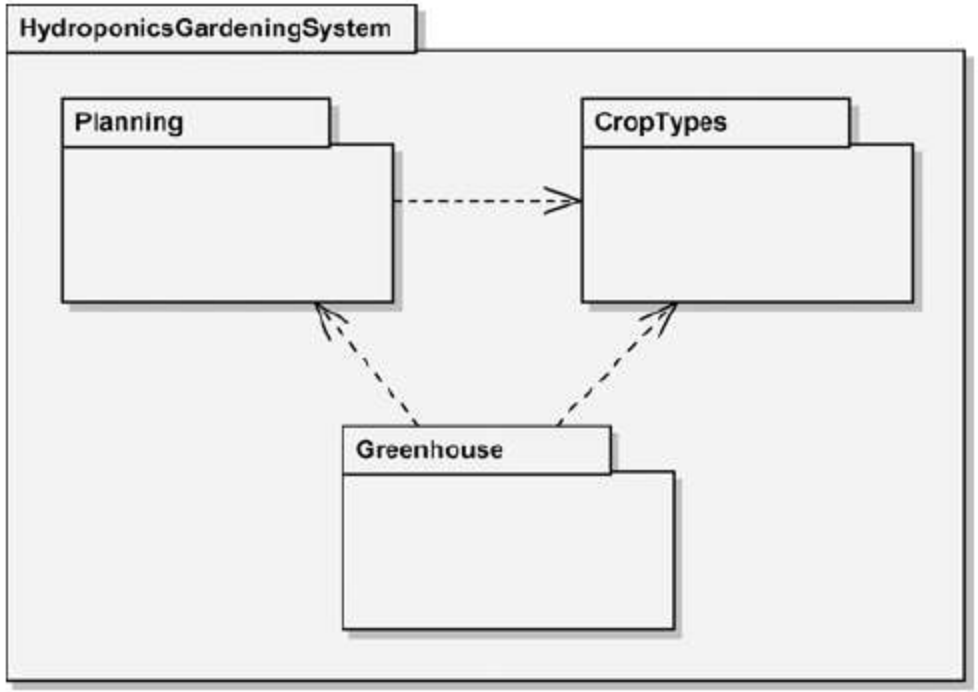


Figure 6: Aggregation of Contained Element Dependencies

IMPORT AND ACCESS

- In an import, other elements that have visibility into the importing package can see the imported items.
- But, when a package performs an access, no other elements can see those elements that have been added to the importing package's namespace. These items are private; they are not visible outside the package that performed the access

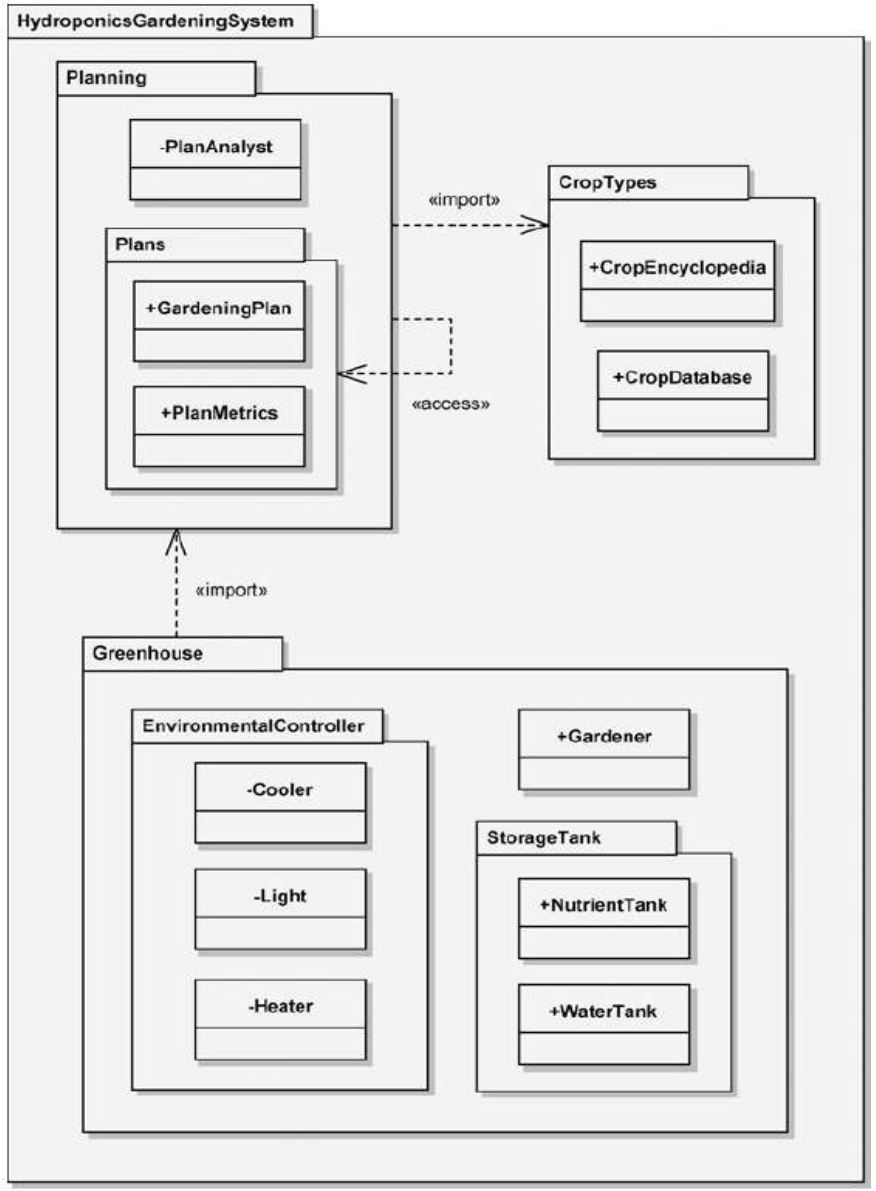


Figure 7:Package Import in the HydroponicsGardeningSystem

COMPONENT DIAGRAM

- A component represents a reusable piece of software that provides some meaningful aggregate of functionality.
- Each class in the system must live in a single component or at the top level of the system.
- Components are a type of structured classifier whose collaborations and internal structure can be shown on a component diagram.
- Components may be used to hierarchically decompose a system and represent its logical architecture.

THE COMPONENT NOTATION

- A ***component*** can be shown with a composite structure using parts, ports, and connectors.
- The component tags should be included: the keyword label ***«component»*** and the component icon shown in the upper right-hand corner of the classifier rectangle.
- On the boundary of the classifier rectangle, we have seven ***ports***, which are denoted by small squares.

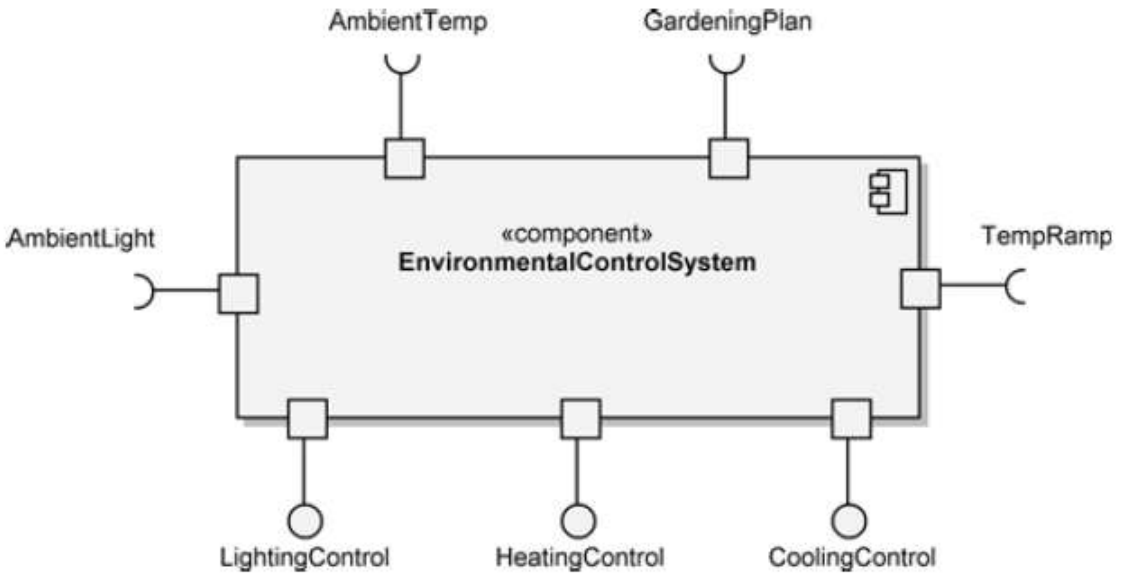


Figure 8: The Component Notation for EnvironmentalControlSystem

THE COMPONENT NOTATION

- ***Ports*** are used by the component for its interactions with its environment, and they provide encapsulation to the structured classifier.
- The ***interfaces*** are shown in the ball-and-socket notation.
- ***Provided interfaces*** use the ball notation to specify the functionality that the component will provide to its environment. (e.g LightingControl)
- ***Required interfaces*** use the socket notation to specify the services that the component requires from its environment (e.g AmbientTemp)

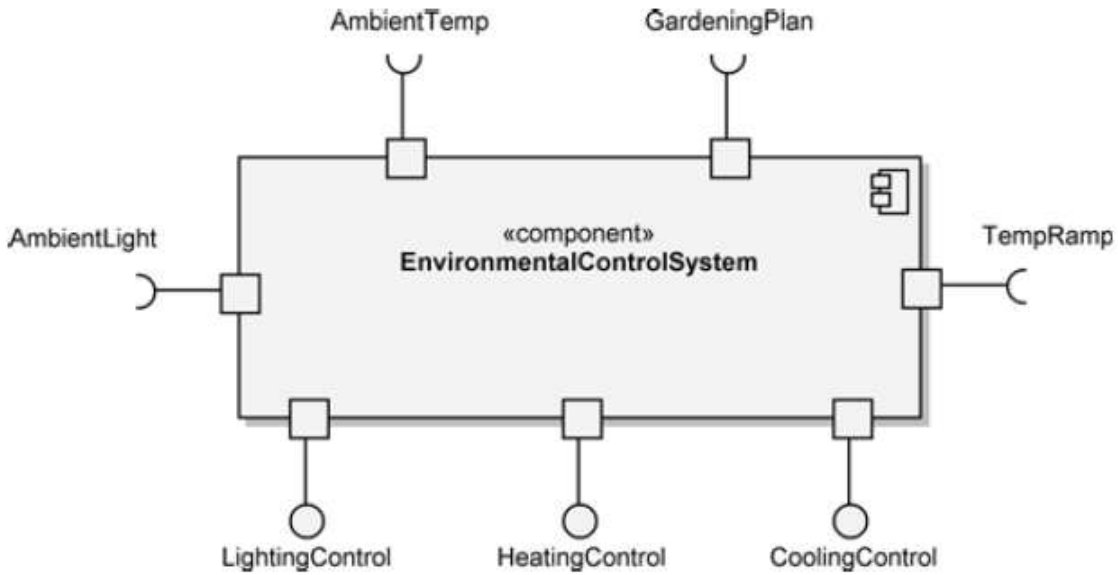


Figure 8: The Component Notation for EnvironmentalControlSystem

THE COMPONENT NOTATION

- ***Ports*** can be used to group ***interfaces*** to provide clarity in a very complex diagram or to represent the intention of having one port through which certain types of interactions will take place.

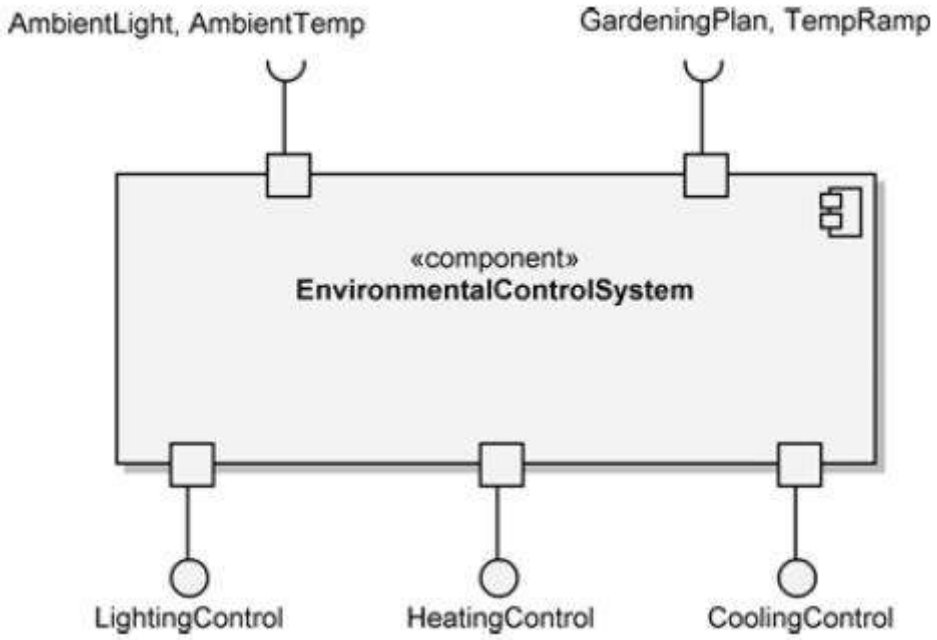


Figure 9: The Component Notation with Interface Grouping

COMPONENT INTERFACES

- EnvironmentalController realizes the CoolControl interface; this means that it provides the functionality specified by the interface.
- Figure 10 shows the dependency of the EnvironmentalController component on the AmbientTemp interface.
- EnvironmentalController acquires the ambient temperature that it requires to fulfill its responsibilities within the EnvironmentalControlSystem component.

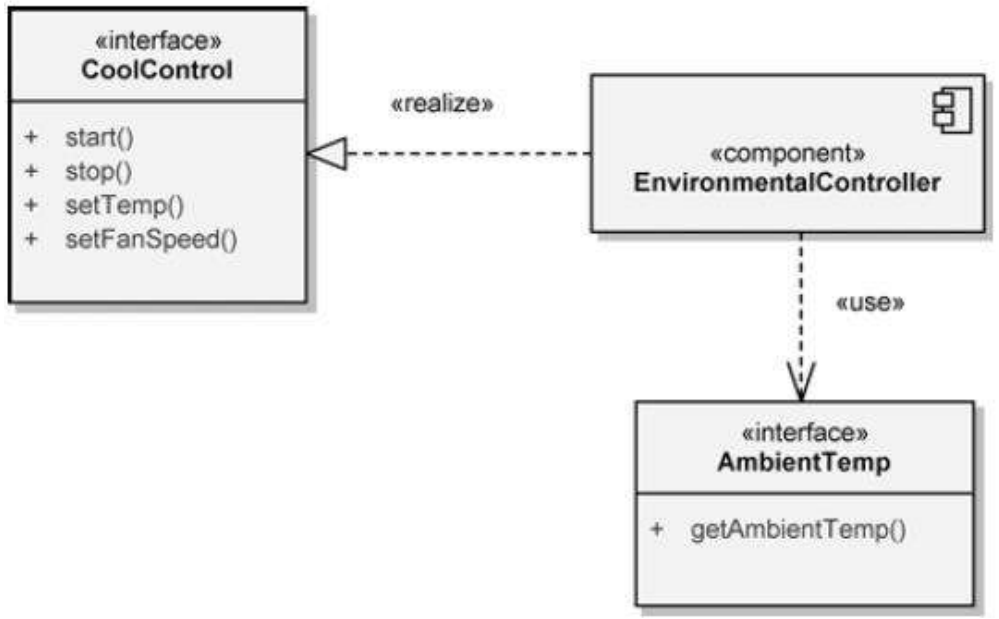


Figure 10: The Specification of Two Interfaces for EnvironmentalController

COMPONENT INTERFACES

- In Figure 11, we show an alternate notation for the interfaces of EnvironmentalController.
- Three provided interfaces listed under the heading ***«provided interfaces».***

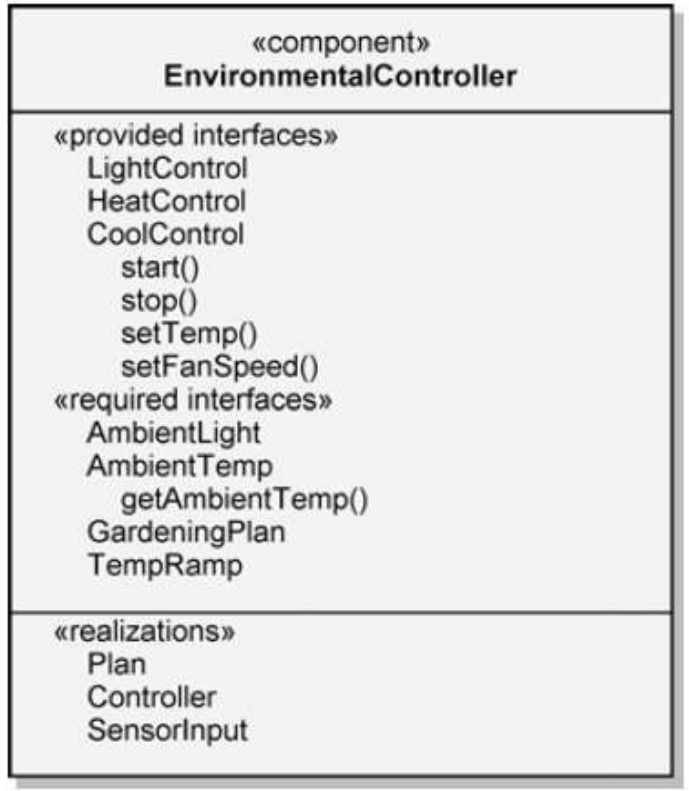


Figure 11: An Alternate Notation for EnvironmentalController's Interfaces and Realizations

COMPONENT REALIZATIONS

- A realization dependency from each of the classes to EnvironmentalController.
- The realization relationship between the EnvironmentalController component and the Plan, Controller, and SensorInput classes is shown in Figure 12.

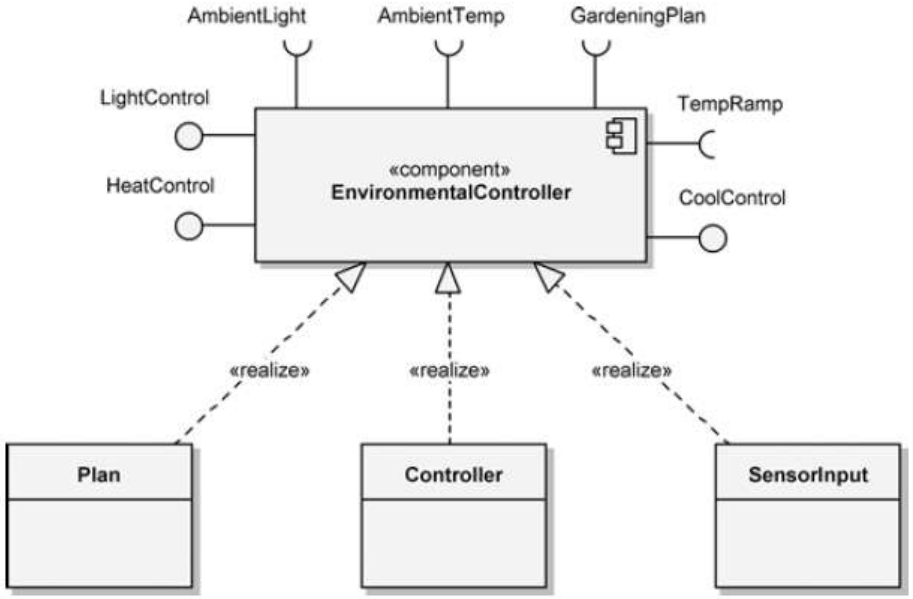


Figure 12: The Realization Dependencies for EnvironmentalController

A COMPONENT'S INTERNAL STRUCTURE

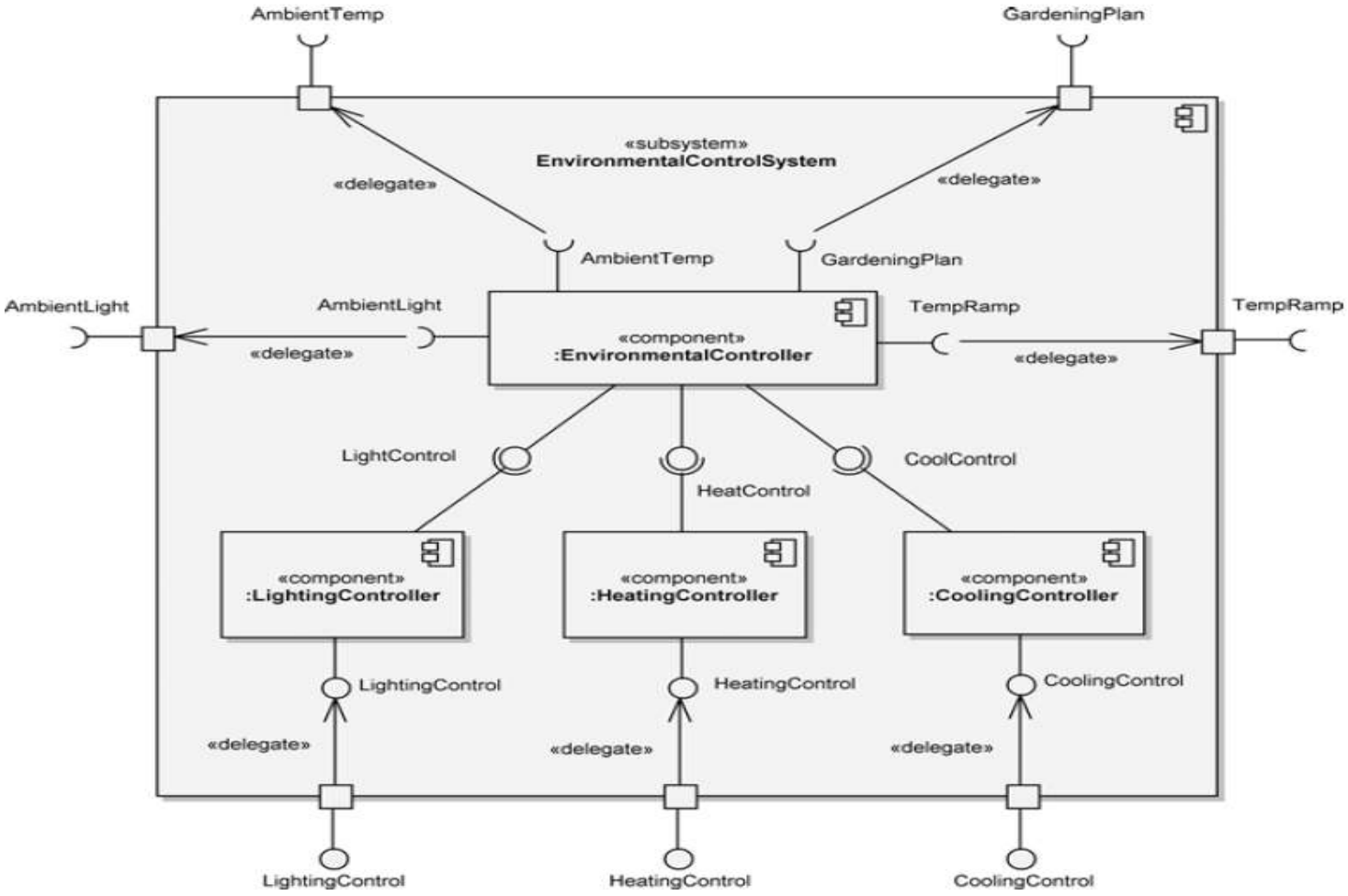


Figure 13: The Internal Structure of EnvironmentalControlSystem

DEPLOYMENT DIAGRAM

- A ***deployment diagram*** is used to show the allocation of artifacts to nodes in the physical design of a system.
- A single deployment diagram represents a view into the artifact structure of a system.
- Deployment diagrams are to used to indicate the physical collection of nodes that serve as the platform for execution of system.
- The three essential elements of a deployment diagram are ***artifacts***, ***nodes***, and ***their connections***.

THE ARTIFACT NOTATION

- An ***artifact*** is typically software code (executable) but could also be a source file, a document, or another item related to the software code.
- The notation for an artifact consists of a class rectangle containing the name of the artifact, the keyword label ***«artifact»,*** and an optional icon that looks like a sheet of paper with the top right-hand corner folded over.
- ***«manifest»*** means that it physically implements the component, thereby connecting the implementation to the design.

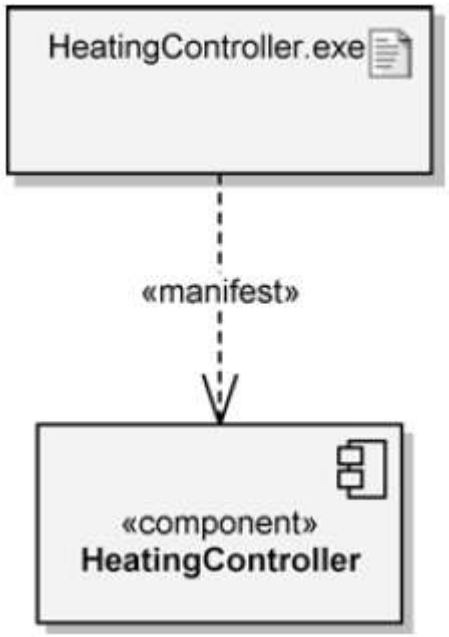


Figure 14: The Artifact Notation for HeatingController.exe

THE NODE NOTATION

- A ***node*** is a computational resource, typically containing memory and processing, on which artifacts are deployed for execution.
- A ***device node*** is a piece of hardware that provides computational capabilities, such as a computer, a modem, or a sensor.
- An ***execution environment node*** is software that provides for the deployment of specific types of executing artifacts; examples include «database» and «J2EE server».

THE NODE NOTATION

- Nodes communicate with one another, via messages and signals, through a communication path indicated by a solid line.
- Communication paths are usually considered to be bidirectional, although if a particular connection is unidirectional, an arrow may be added to show the direction.

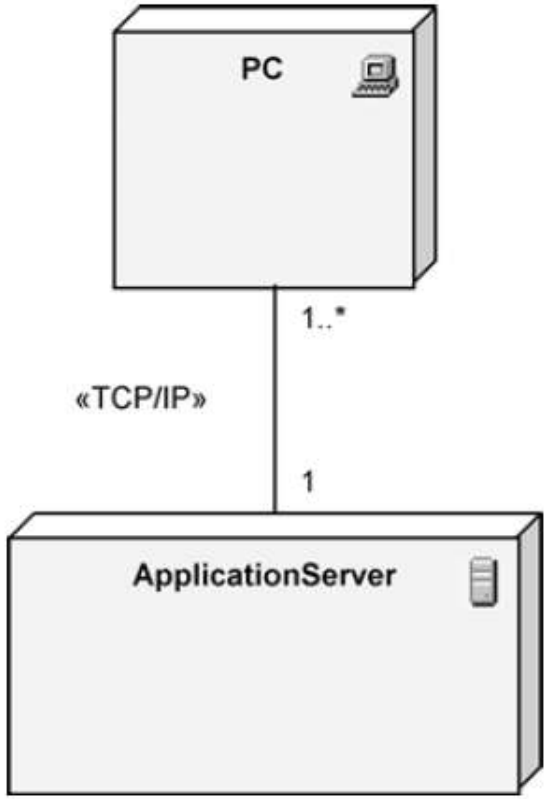


Figure 15: Notations for Two Nodes

DEPLOYMENT DIAGRAM

The deployment of the EnvironmentalController.exe, LightingController.exe, HeatingController.exe, and CoolingController.exe artifacts on the ApplicationServer node is indicated by containment

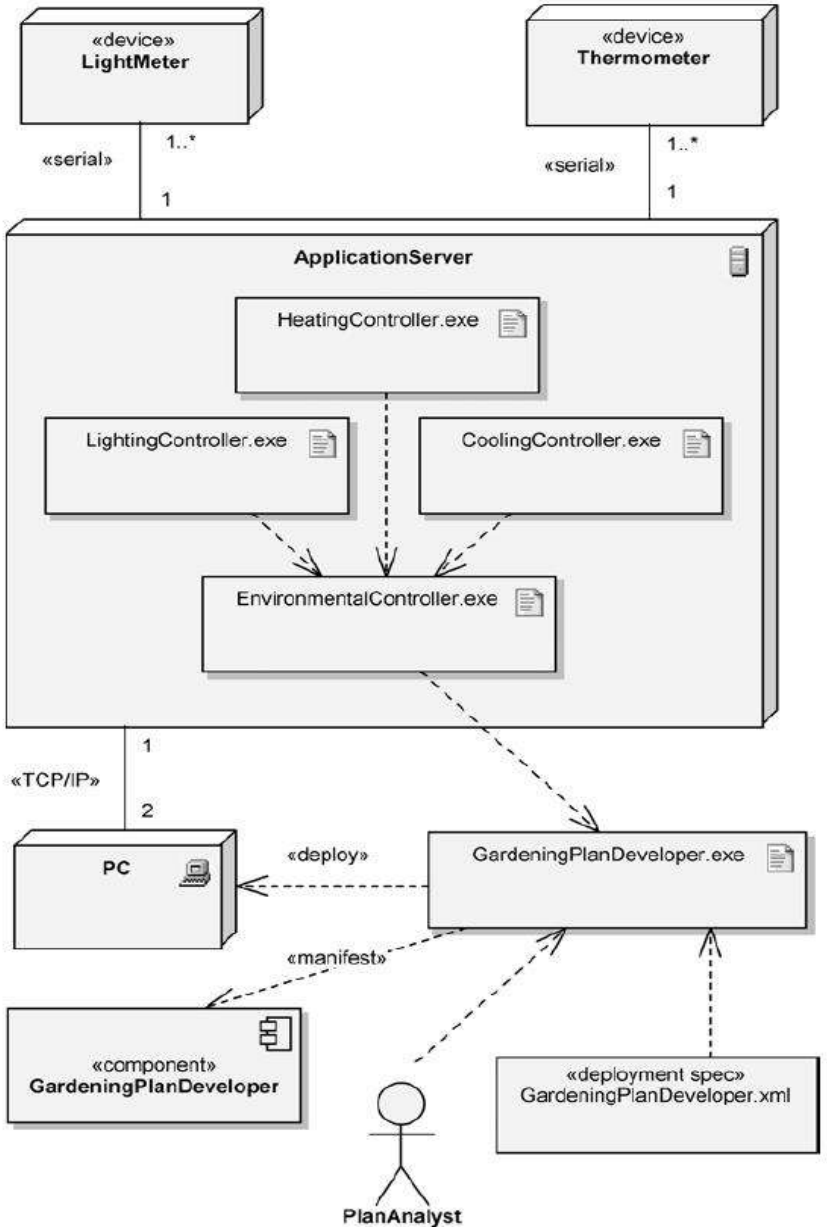


Figure 16: The Deployment Diagram for EnvironmentalControlSystem

EXAMPLE ARCHITECTURAL & HIGH LEVEL DESIGN MODELING WITH UML

- **PURPOSE 1**
  - Draw the package diagram for Hospital Management System (HMS).

EXAMPLE ARCHITECTURAL & HIGH LEVEL DESIGN MODELING WITH UML

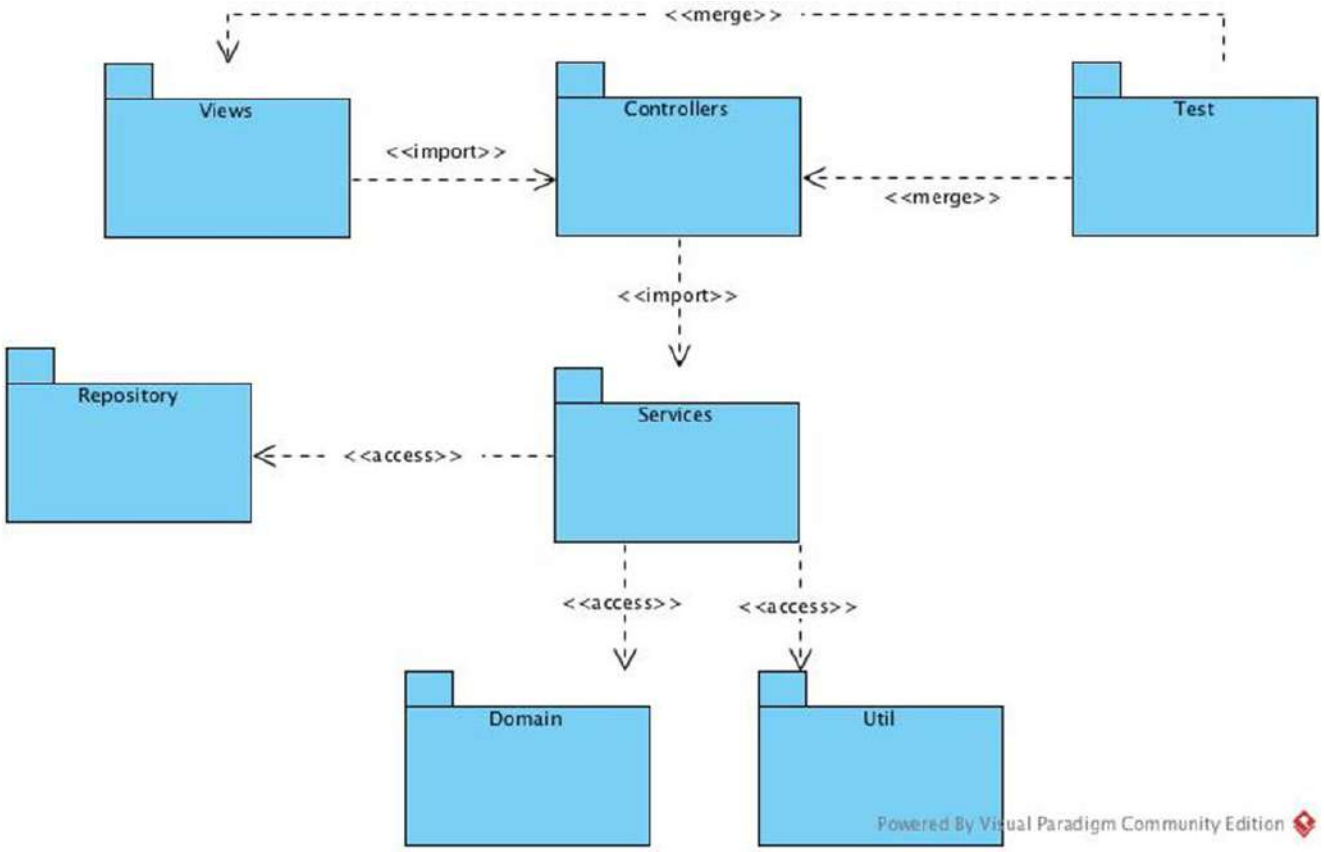


Figure 17: The package diagram for HMS.

EXAMPLE ARCHITECTURAL & HIGH LEVEL DESIGN MODELING WITH UML

- **PURPOSE 2**
  - Draw the component diagram of Hospital Management System (HMS).

EXAMPLE ARCHITECTURAL & HIGH LEVEL DESIGN MODELING WITH UML

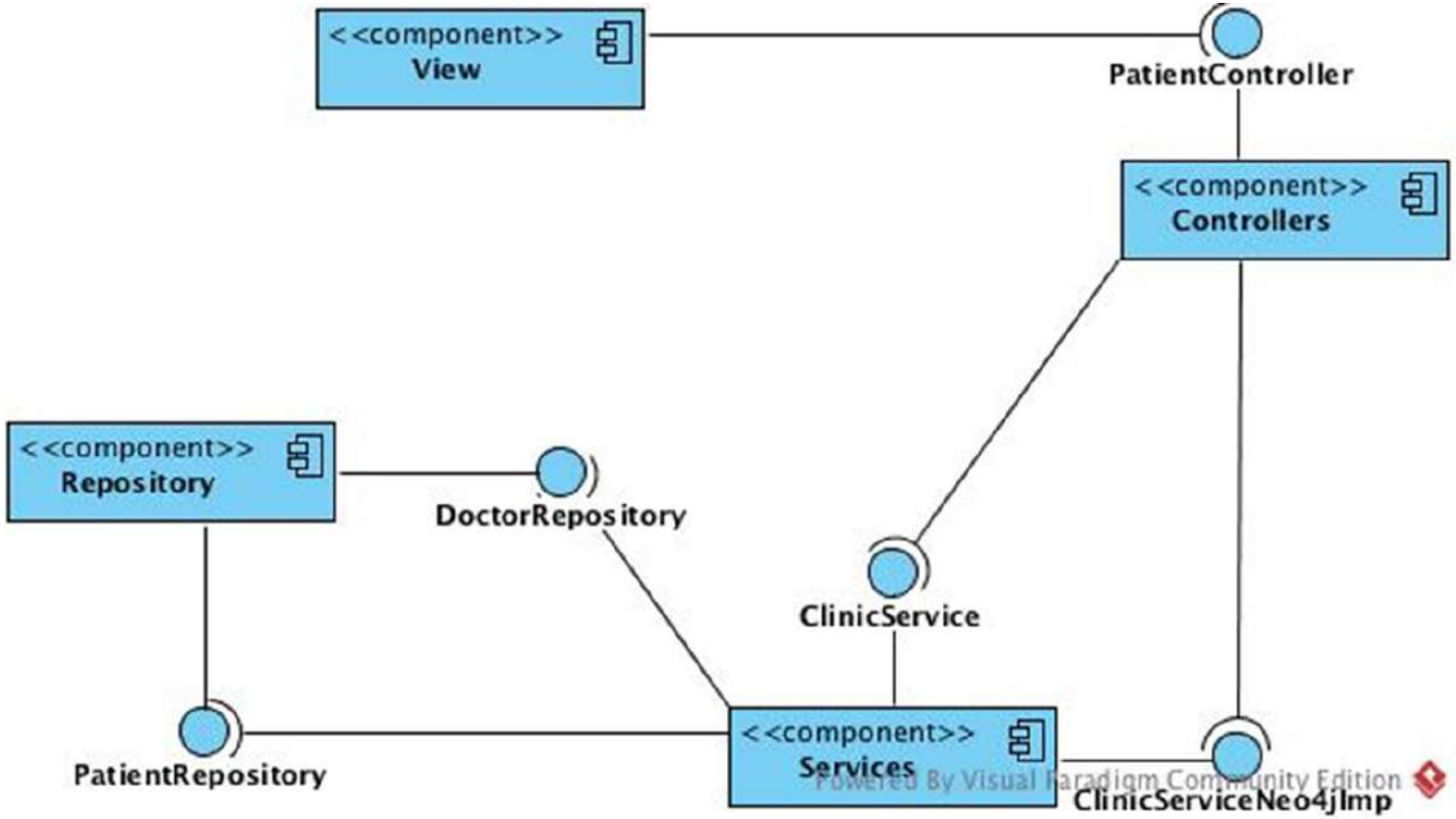


Figure 18: The component diagram for HMS.

EXAMPLE ARCHITECTURAL & HIGH LEVEL DESIGN MODELING WITH UML

- **PURPOSE 3**
  - Draw the deployment diagram of Hospital Management System (HMS).

EXAMPLE ARCHITECTURAL & HIGH LEVEL DESIGN MODELING WITH UML

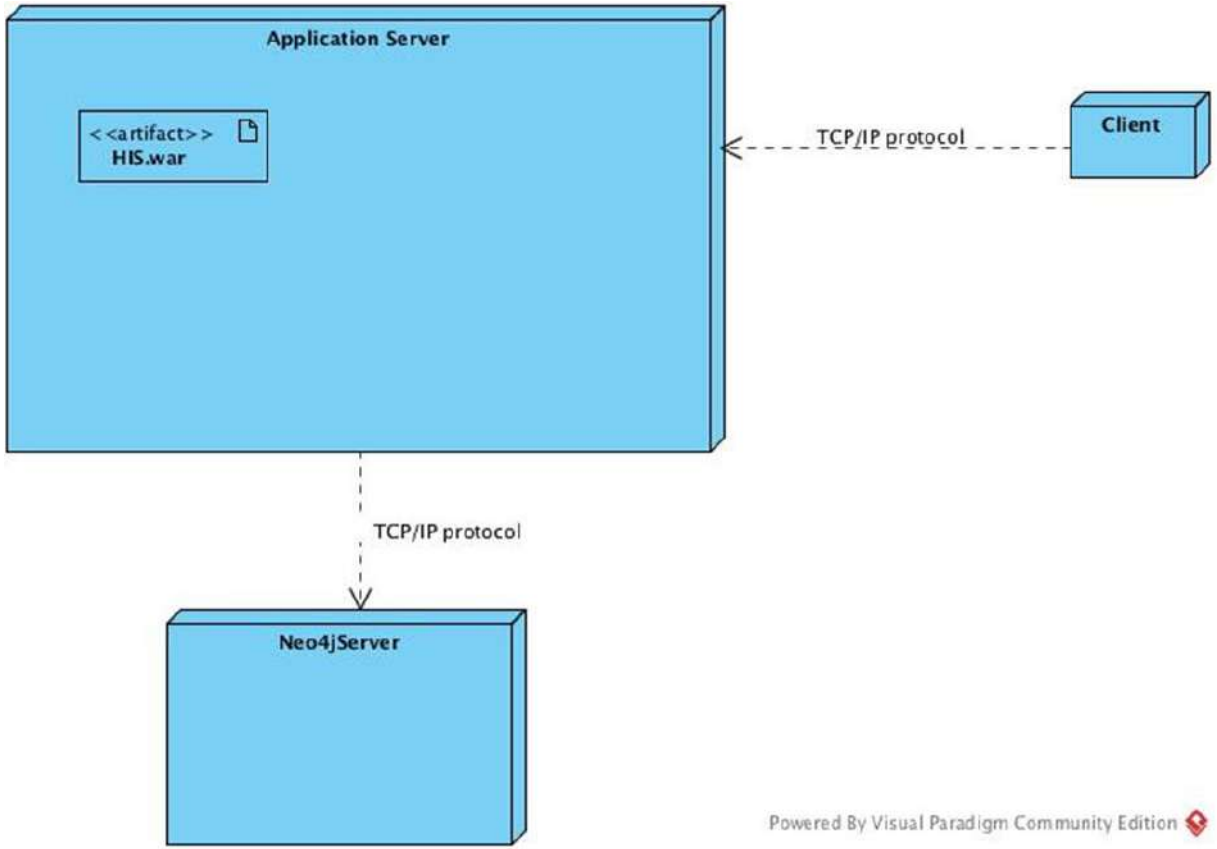


Figure 19: The deployment diagram for HMS.

SOFTWARE REQUIREMENTS SPECIFICATION (SRS)

- **Software Specifications Requirements**

  - ✓**An SRS** is the ***artifact*** that describes what the software will do and how it will be expected to perform
  - ✓An **SRS** describes the functionality the product needs to fulfill all stakeholders (business, users) needs.
  - ✓A typical SRS includes:
    - A purpose
    - An overall description
    - Specific requirements

https://piazza.com/hacettepe.edu.tr/spring2020/bbm382384/resources

HOW TO WRITE AN SRS DOCUMENT

- **Step 1:** Create an outline for your software requirements specification by using the template put in the piazza under Resources
- **Step 2:** Start with a purpose. In the **'1.Introduction'** part of the SRS template, introducing your SRS is very important. You should write the expectation for the system to be developed.
- **Step 3:** Give an detailed specification of What You'll Build. In the **' 2. System-Wide Functional Requirements'** part of the SRS template, you are expected to set user needs-functional requirements for the system developed.
- **Step 4:** Set the quality expectations for your system. In the **' 3. System Qualities'** part of the SRS template, you are expected to set non-functional requirements such as performance,usability etc. for the system to be developed.
- **Step 5:** Define system interfaces. In the **' 4. System Interfaces'** part of the SRS template, you are expected to define **User Interfaces** and **Interfaces to External Systems or Devices.** User Interfaces are the interfaces to be implemented by the software. **Interfaces to External Systems or Devices** outline how your product will interface with other components.
- **Step 6:** Set the Business Rules. In the **'5. Business Rules'** part of the SRS template, you should specify the execution of your systems' actions.
- **Step 7:** Define the System Constraints. In the **'6. System Constraints'** part of the SRS template, you are expected to describe any design; implementation or deployment constraints of the system to be developed.
- **Step 8:** In the **'7. System Compliance'** part of the SRS template, you are expected to specify licencing requirement, legacy and applicable standard which guide you while develping your system such as IEEE Software Engineering Standards.
- **Step 9:** Define system documentation. In the **'8. System Documentation'** part of the SRS template, you are expected to write which documents you will prepare for users to help them to use the system.

RESOURCES

- Object-Oriented Analysis and Design with Applications -Third Edition
- https://www.visual-paradigm.com/guide/uml-unified-modeling-language