COMP9900 Information Technology Project
COMP3900 Computer Science Project

Software development methods overview

and

Unified Modelling Language

Week 3

GitHub Classroom

COMP3900/COMP9900 Term 3 2019

“https://classroom.github.com/g/vbcJ-pn3”

- The link was sent to all Scram masters
- After crating a group, the repository is automatically created
- Each Scram master invites his members to join
- Invitation to all students – webCMS3

GitHub Classroomå

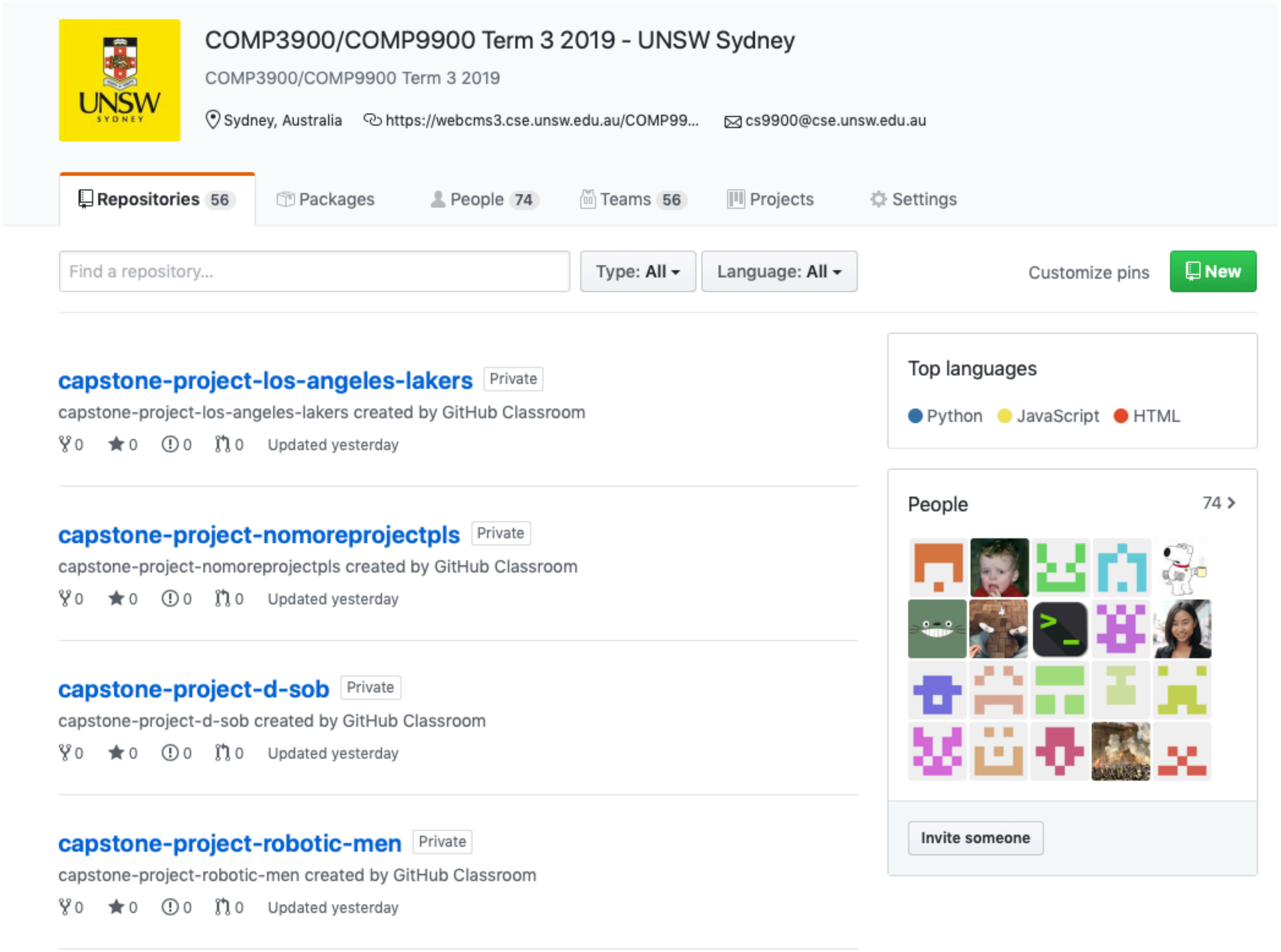
COMP3900/COMP9900 Term 3 2019

https://classroom.github.com/g/vbcJ-pn3

Using your GitHub account (you need to create one if you do not have one) to join the classroom roster.

Please do NOT skip the step of matching your GitHub account with your zID.

You will then be asked to accept the "Capstone Project" assignment I created.


COMP3900/COMP9900 Term 3 2019 - UNSW Sydney

COMP3900/COMP9900 Term 3 2019

Sydney, Australia https://webcms3.cse.unsw.edu.au/COMP99... cs9900@cse.unsw.edu.au

Repositories 56 | Packages | People 74 | Teams 56 | Projects | Settings

Find a repository... | Type: All | Language: All | Customize pins | New

capstone-project-los-angeles-lakers Private
capstone-project-los-angeles-lakers created by GitHub Classroom
0 ★ 0 0 0 Updated yesterday

capstone-project-nomoreprojectpls Private
capstone-project-nomoreprojectpls created by GitHub Classroom
0 ★ 0 0 0 Updated yesterday

capstone-project-d-sob Private
capstone-project-d-sob created by GitHub Classroom
0 ★ 0 0 0 Updated yesterday

capstone-project-robotic-men Private
capstone-project-robotic-men created by GitHub Classroom
0 ★ 0 0 0 Updated yesterday

Top languages
● Python ● JavaScript ● HTML

People 74 >

Invite someone

Software Development Process

A software development process is a recipe used for constructing software determining the capabilities it has, how it is constructed, who works on what, and the time frames for all activities.

Processes aim to bring discipline and predictability to software development, increasing the chance of success of a project.

Unified Modelling Language (UML)

UML is the language for modelling your software, it's an important part of the software development process.

Modelling consists of building an abstraction of reality.

Abstractions are simplifications that

- They ignore irrelevant details and
- They only represent the relevant details.

What is relevant or irrelevant depends on the purpose of the model.

Unified Modelling Language (UML)

UML is not a method, methodology or software development process.

Different Types of Software Development Process

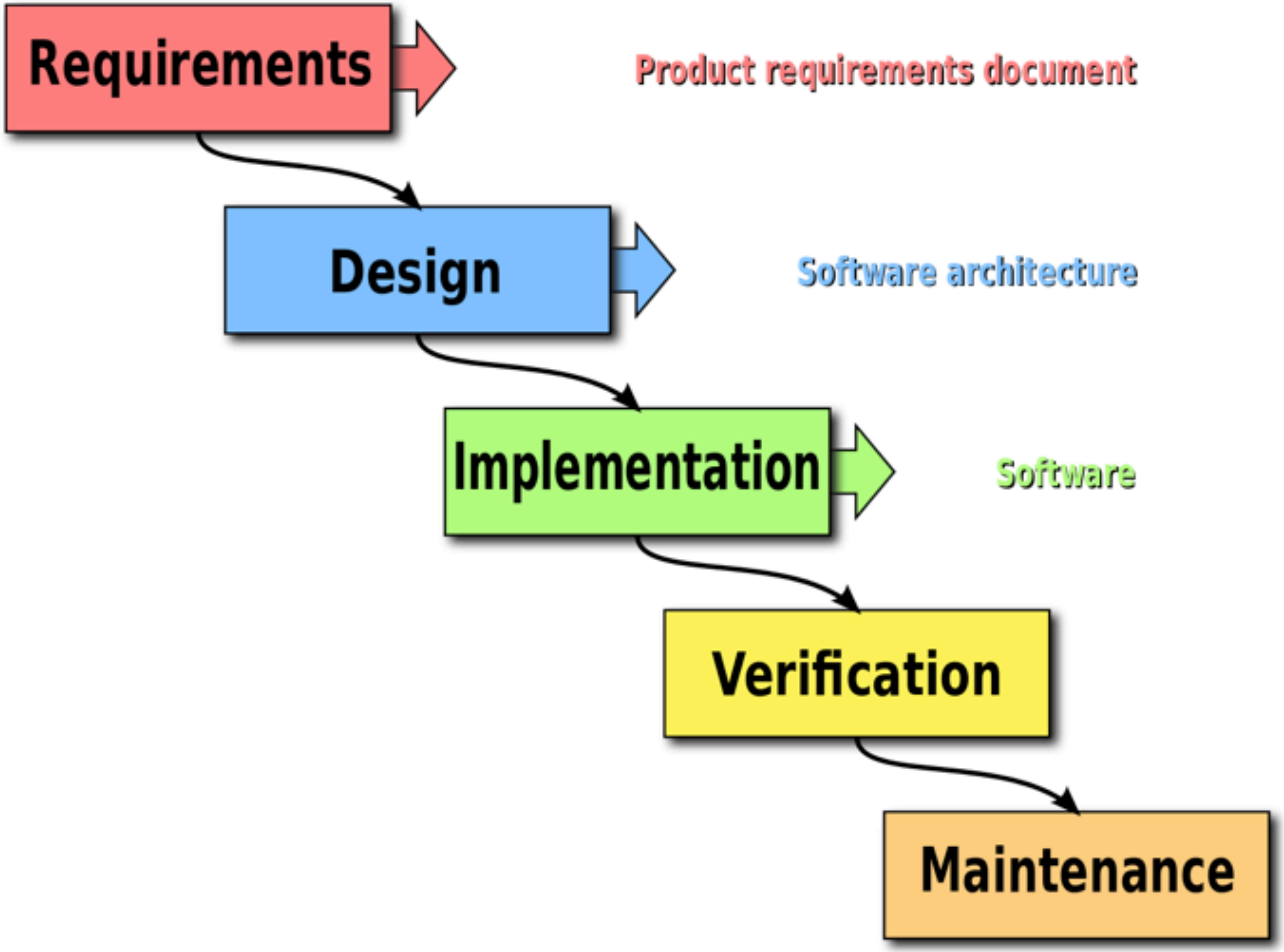
Different Types of Software Development Process

Waterfall — The waterfall method attempts to pin down the requirements early in the project life cycle.

After gathering requirements, software design is performed in full.

Once the design is complete, the software is implemented.

The problem with this method is that if a change in requirements occurs, the impact can be devastating.


Requirements
Product requirements document
Design
Software architecture
Implementation
Software
Verification
Maintenance

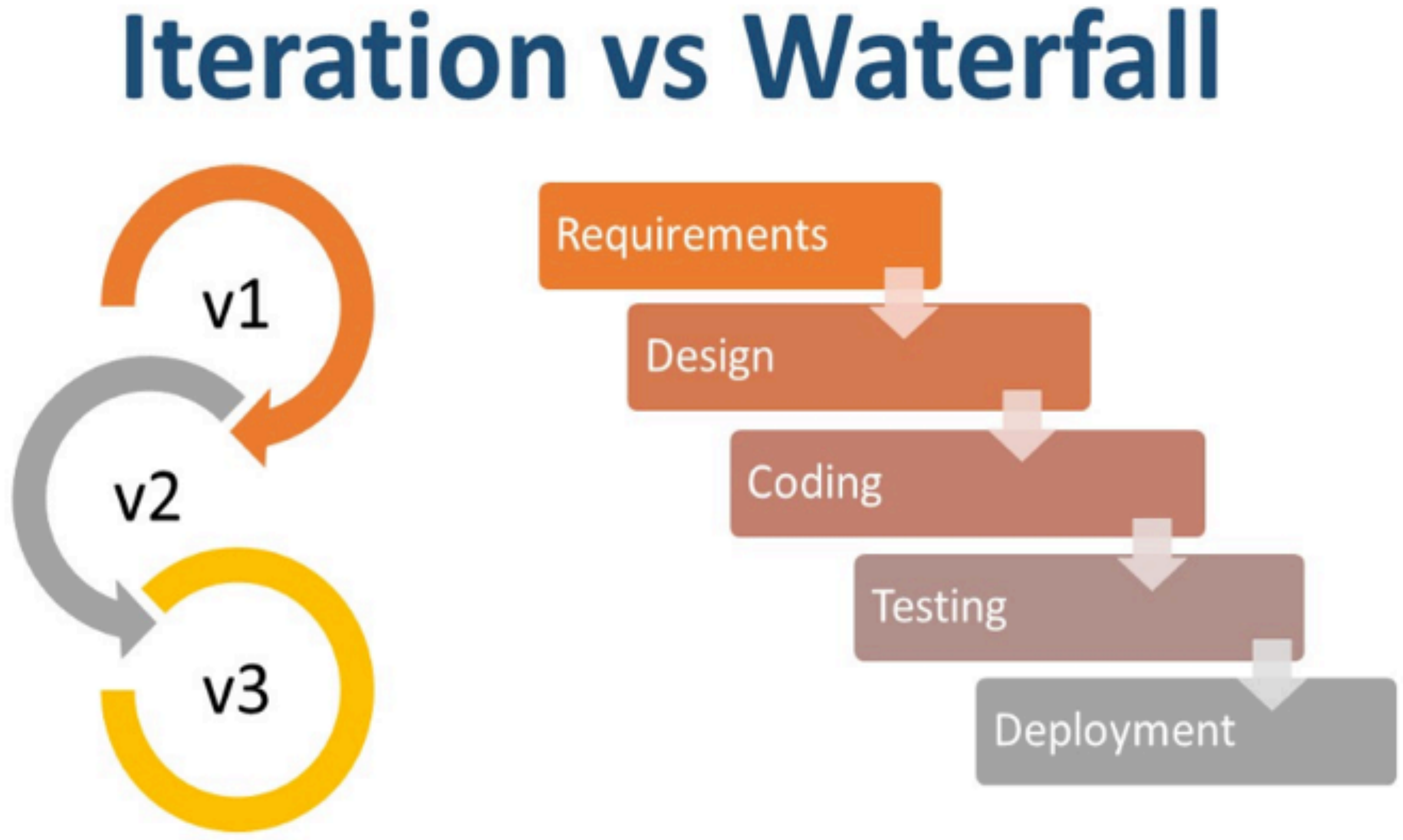
Different Types of Software Development Process

Iterative — Iterative methods attempt to address the shortcomings of the waterfall approach by accepting that change will happen and, in fact, embracing it.

The Unified Process - a well-known iterative process, consists of multiple phases, each phase containing some amount of the following activities: requirements, design, and implementation (coding).

Iterative methods encompass a wider range of approaches (e.g., agile iterative processes), and they can range from using UML as sketch to using UML as blueprint.

Different Types of Software Development Process



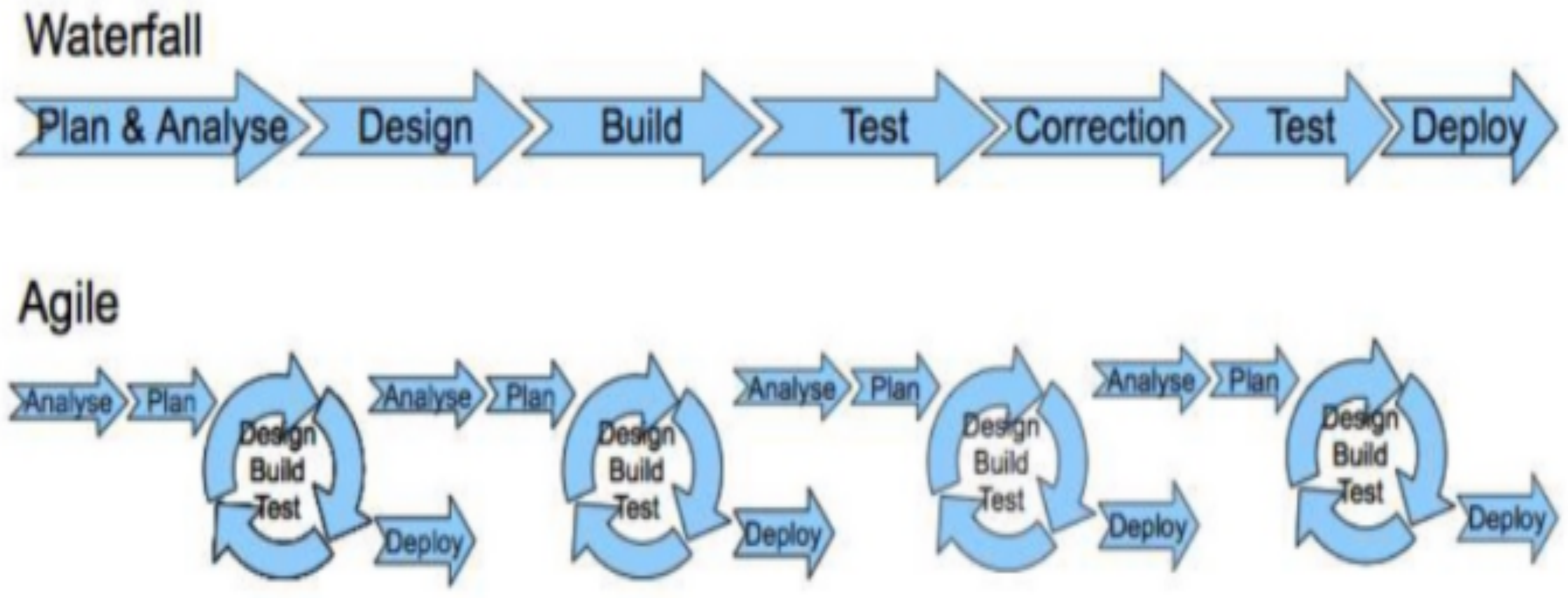
Different Types of Software Development Process

Agile methods — Agile methods use iterations in extremely short bursts and attempt to minimize risk by always having a working system of expanding capabilities.

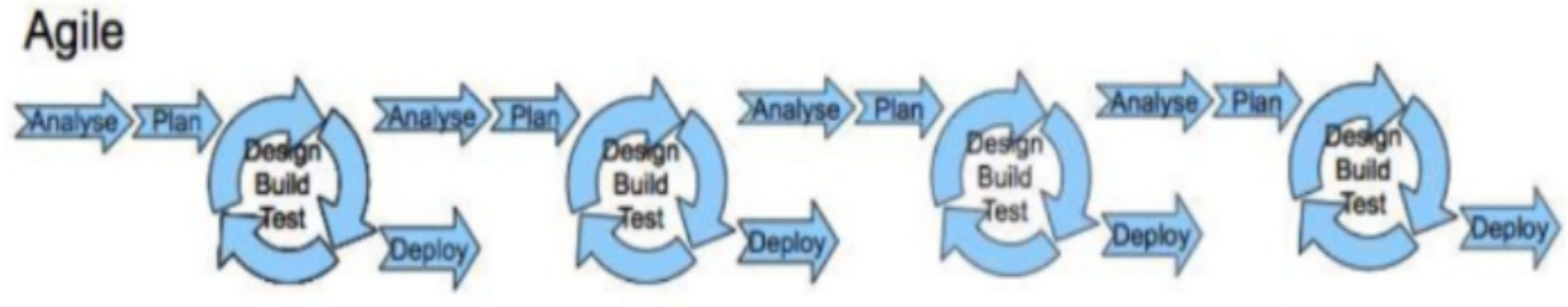
Methodologies under this category have introduced some of the more interesting development practices.

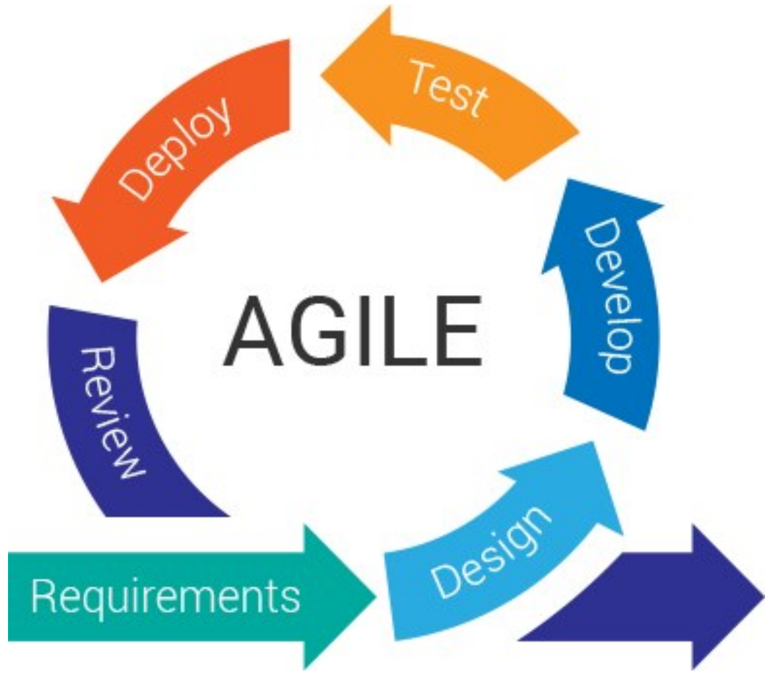
Agile methods emphasize using UML as a sketch.

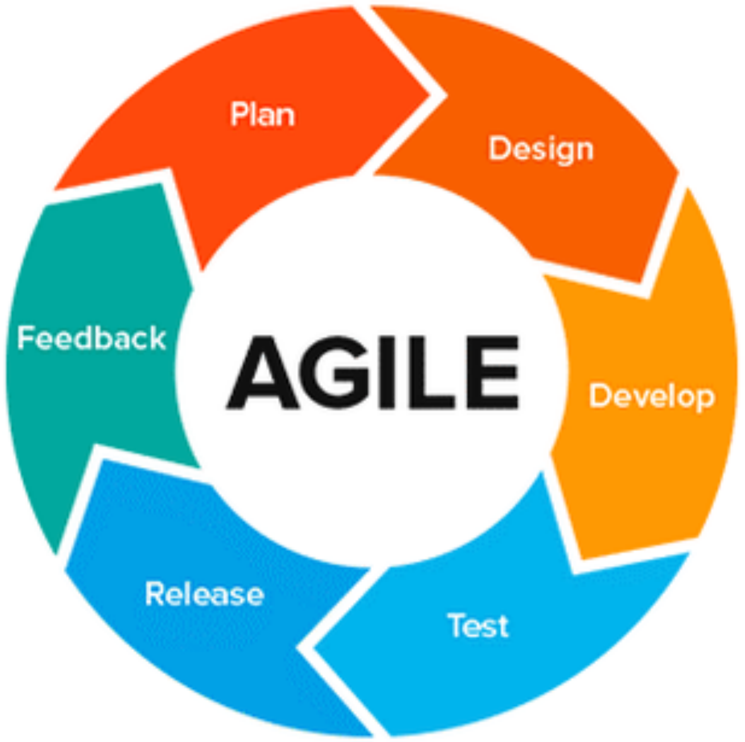
Different Types of Software Development Process



Different Types of Software Development Process







Good resources
Agile and SCRUM

Agile Alliance

https://www.agilealliance.org

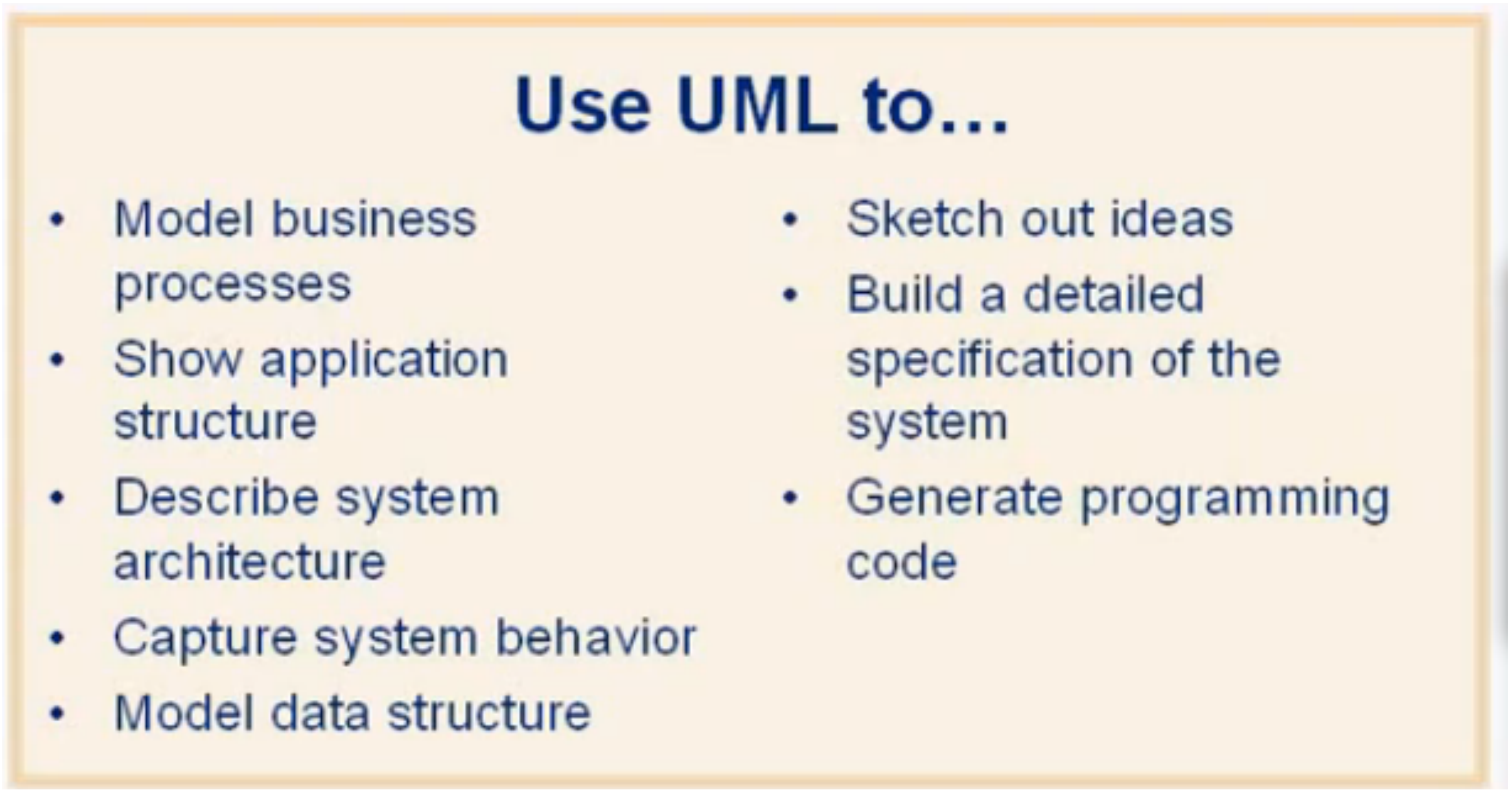
https://www.agilealliance.org/glossary/scrum/

Unified Modelling Language UML

UML (Unified Modelling Language) is

- a standard language for specifying, visualizing, constructing, and documenting the artefacts of software systems.

What can we use UML for?



Use UML to…

- Model business processes
- Show application structure
- Describe system architecture
- Capture system behavior
- Model data structure
- Sketch out ideas
- Build a detailed specification of the system
- Generate programming code

Unified Modelling Language UML

- UML was created by the Object Management Group (OMG) and UML 1.0 specification draft was proposed to the OMG in January 1997.

- It was initially started to capture the behaviour of complex software and non-software system and now it has become an OMG standard.

Goals of UML

A picture is worth a thousand words, this idiom absolutely fits describing UML.

There are a number of goals for developing UML but the most important is to define some general purpose modelling language, which all modellers can use and it also needs to be made simple to understand and use.

Goals of UML

UML diagrams are not only made for developers but also for business users, common people, and anybody interested to understand the system.

The system can be a software or non-software system.

- UML is not a development method rather it accompanies with processes to make it a successful system.

Goals of UML

The goal of UML can be defined as

a simple modelling mechanism to model all possible practical systems in today’s complex environment.

UML - Architecture

Any real-world system is used by different users.

- The users can be developers, testers, business people, analysts, and many more.
- Before designing a system, the architecture is made with different perspectives in mind.
- The most important part is to visualize the system from the perspective of different viewers. The better we understand the better we can build the system.

UML - Architecture

The center is the **Use Case** view which connects all these four. A **Use Case** represents the functionality of the system. Hence, other perspectives are connected with use case.

UML - Architecture

Design of a system consists of classes, interfaces, and collaboration. UML provides class diagram, object diagram to support this.

Implementation defines the components assembled together to make a complete physical system. UML component diagram is used to support the implementation perspective.

UML - Architecture

Process defines the flow of the system. Hence, the same elements as used in Design are also used to support this perspective.

Deployment represents the physical nodes of the system that forms the hardware. UML deployment diagram is used to support this perspective.

Architectural Modelling

Architectural model represents the overall framework of the system.

It contains both structural and behavioural elements of the system.

Architectural model can be defined as the blueprint of the entire system. Package diagram comes under architectural modelling.

Structural Modelling

Structural modelling captures the static features of a system. They consist of the following −

- Classes diagrams
- Objects diagrams
- Deployment diagrams
- Package diagrams
- Composite structure diagram
- Component diagram

Structural Modelling

Structural model represents the framework for the system and this framework is the place where all other components exist.

- the class diagram,
- component diagram and
- deployment diagrams are part of structural modeling.

They all represent the elements and the mechanism to assemble them.

The structural model never describes the dynamic behavior of the system.

Class diagram is the most widely used structural diagram.

Behavioural Modelling

Behavioural model describes the interaction in the system.

It represents the interaction among the structural diagrams.

Behavioural Modelling

Behavioural modelling shows the dynamic nature of the system.

They consist of the following −

- Activity diagrams
- Interaction diagrams
- Use case diagrams

All the above show the dynamic sequence of flow in a system.

UML - Standard Diagrams

Any complex system is best understood by making some kind of diagrams or pictures.

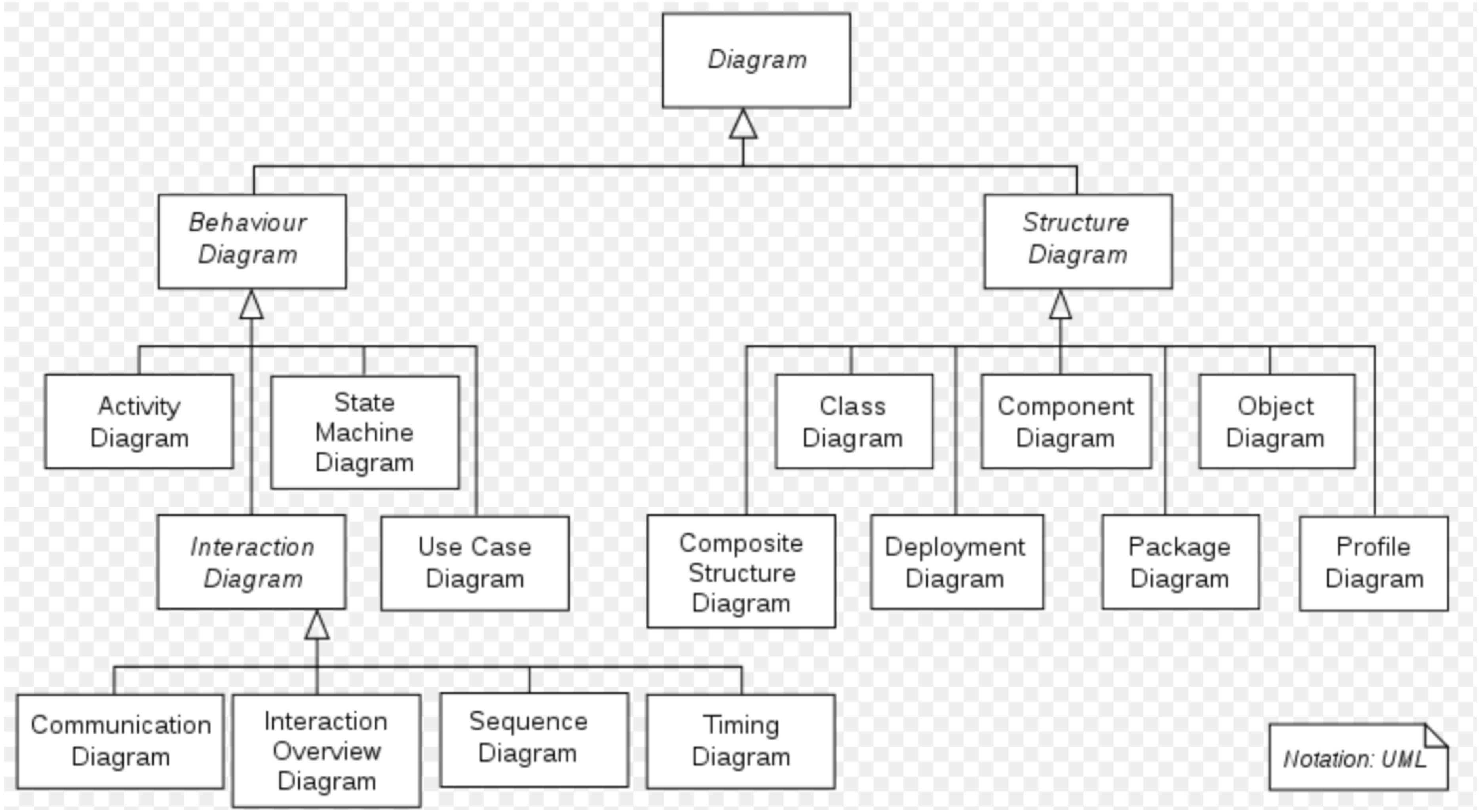
We prepare UML diagrams to understand the system in a better and simple way. A single diagram is not enough to cover all the aspects of the system.

UML defines various kinds of diagrams to cover most of the aspects of a system.

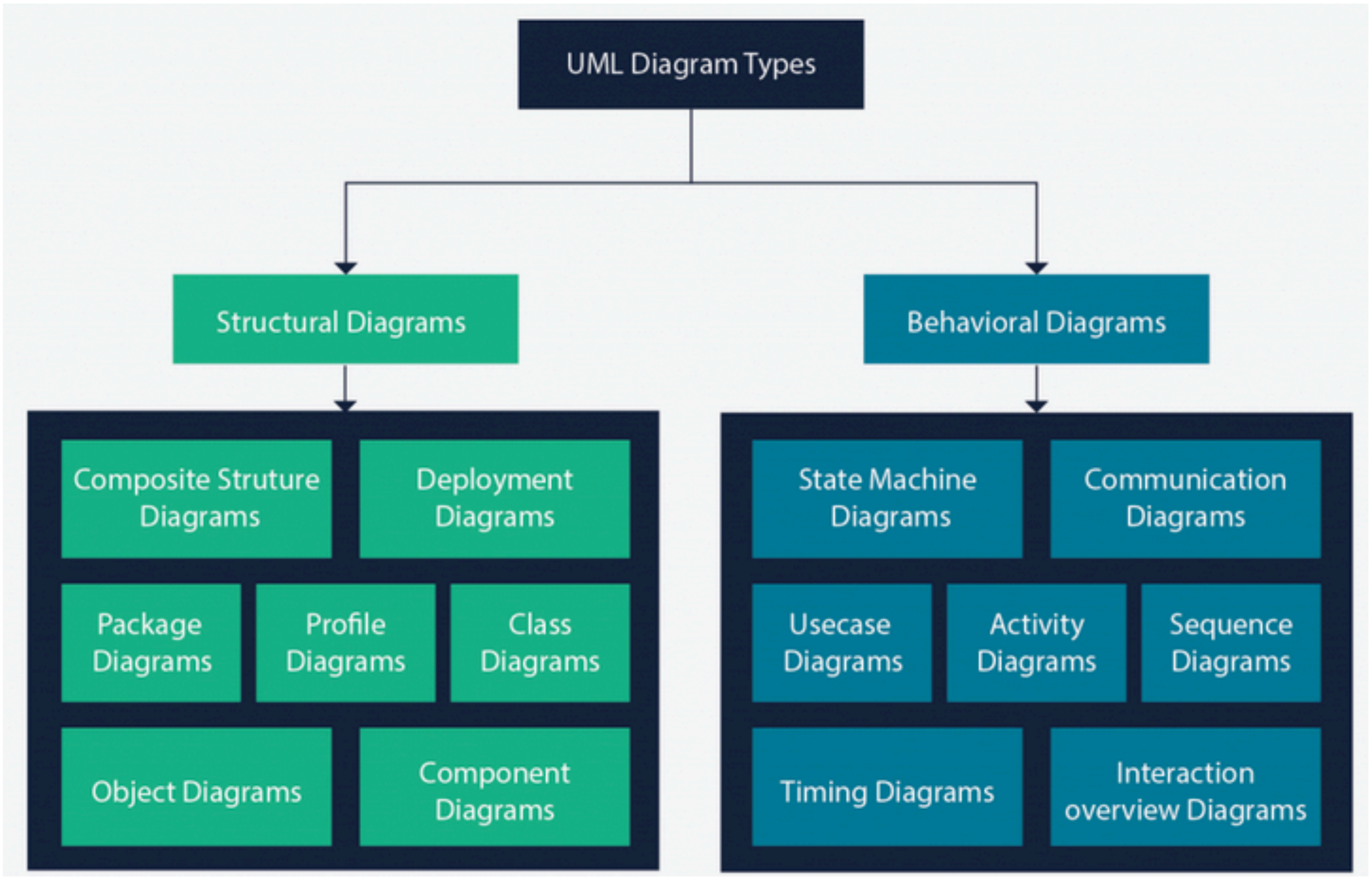
You can also create your own set of diagrams to meet your requirements.

Diagrams are generally made in an incremental and iterative way.

UML Diagram Types



UML Diagrams



UML - Standard Diagrams

UML defines various kinds of diagrams to cover most of the aspects of a system.

There are two broad categories of diagrams and they are again divided into subcategories −

- Structural Diagrams
- Behavioural Diagrams

UML diagrams - I

- Use case diagrams
- Class and object diagrams
- Sequence diagrams
- Statechart diagrams
- Activity diagrams (“workflow”)

Used 80%

UML diagrams - II

- Collaboration diagrams
- Component diagrams
- Deployment diagrams

Used 20%

Structural Diagrams

The structural diagrams represent the static aspect of the system.

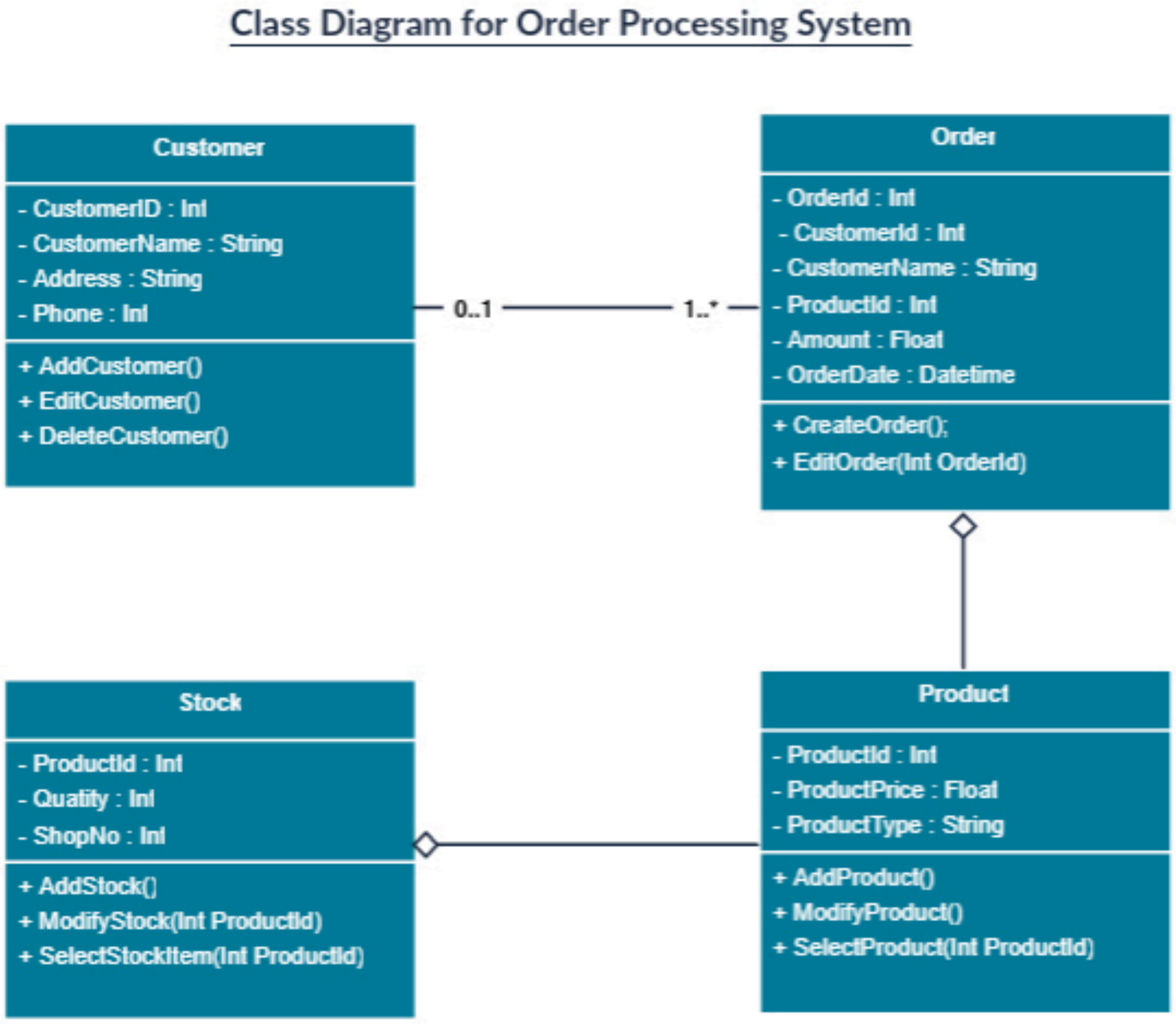
These static aspects represent those parts of a diagram, which forms the main structure and are therefore stable.

These static parts are represented by classes, interfaces, objects, components, and nodes. The four structural diagrams are :

- Class diagram
- Object diagram
- Component diagram
- Deployment diagram

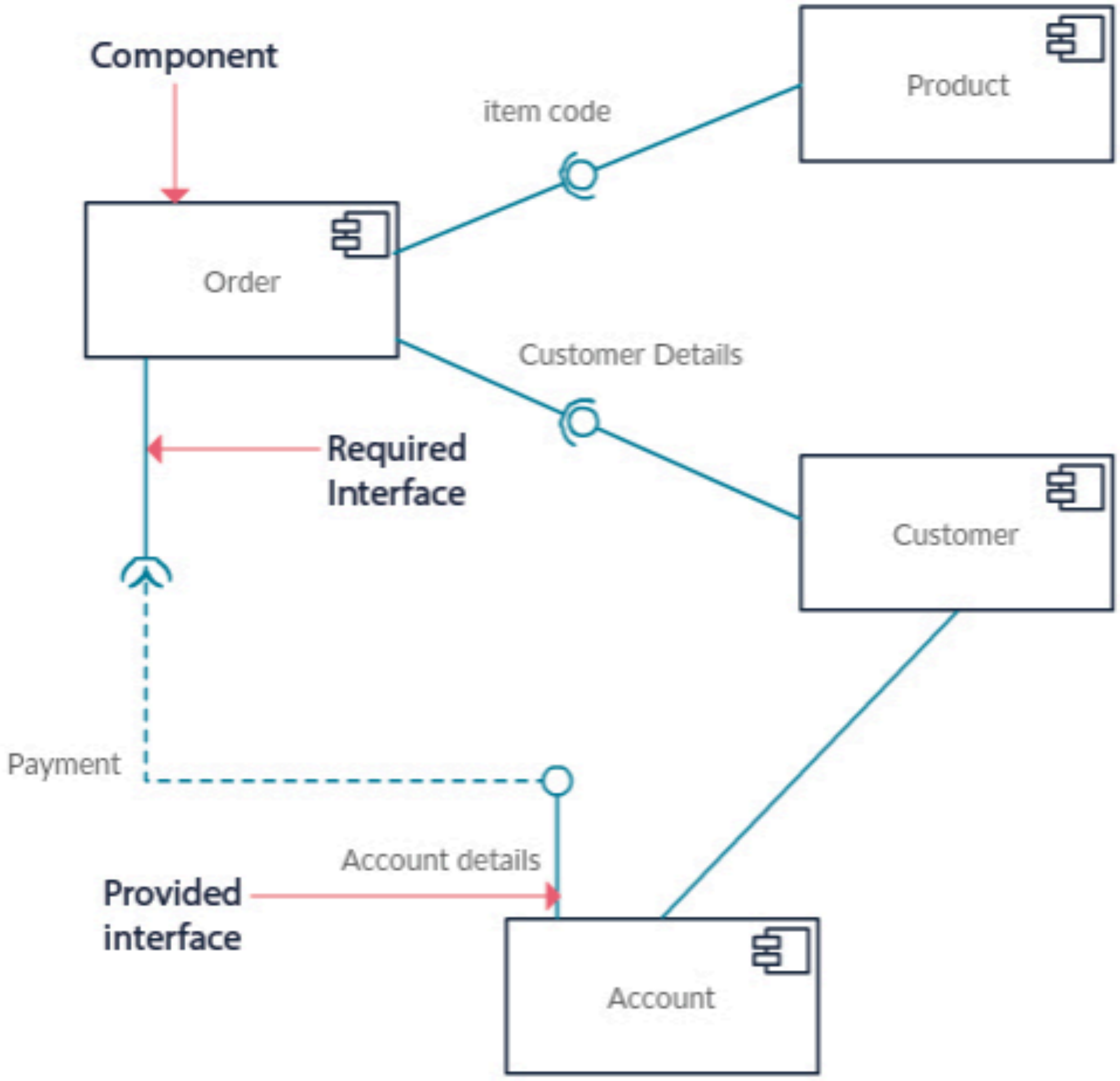
Class diagram

Class diagrams are the main building block of any object-oriented solution.

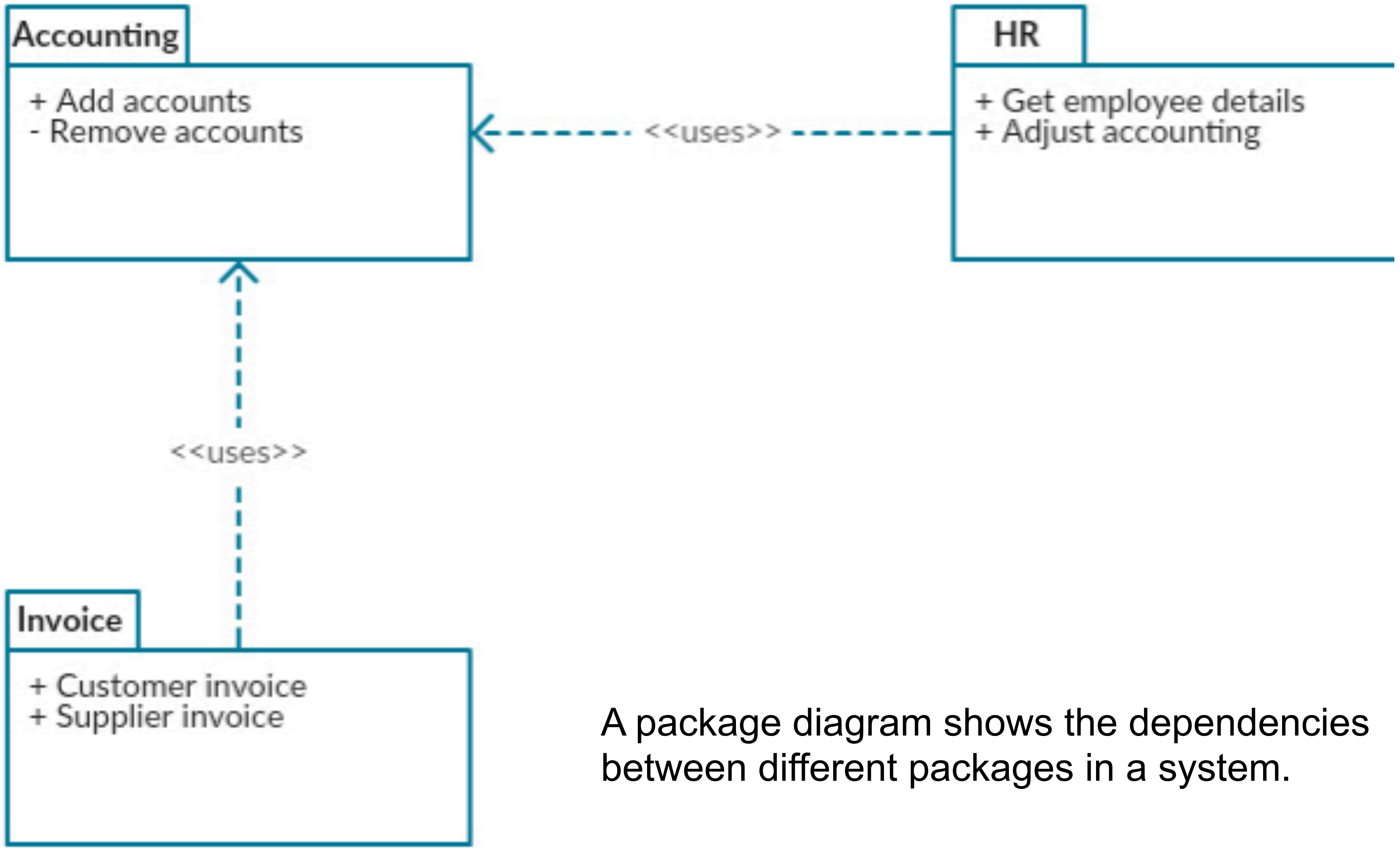


Component Diagram

A component diagram displays the structural relation ship of components of a software system.

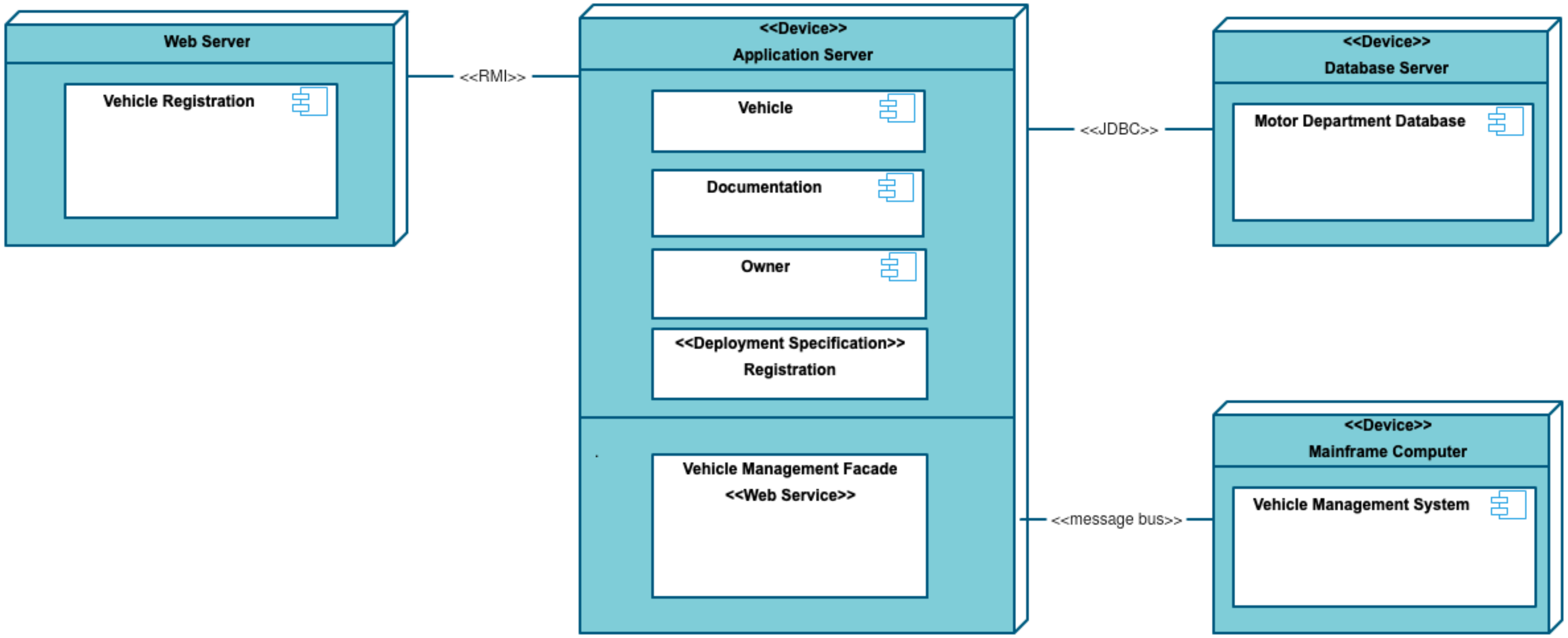


Package Diagram



A package diagram shows the dependencies between different packages in a system.

Deployment Diagram



A deployment diagram shows the hardware of your system and the software in that hardware. Deployment diagrams are useful when your software solution is deployed across multiple machines with each having a unique configuration.

Behavioural Diagrams

Any system can have two aspects, static and dynamic. So, a model is considered as complete when both the aspects are fully covered.

Behavioural diagrams basically capture the dynamic aspect of a system. Dynamic aspect can be further described as the changing/moving parts of a system.

UML has the following five types of behavioural diagrams −

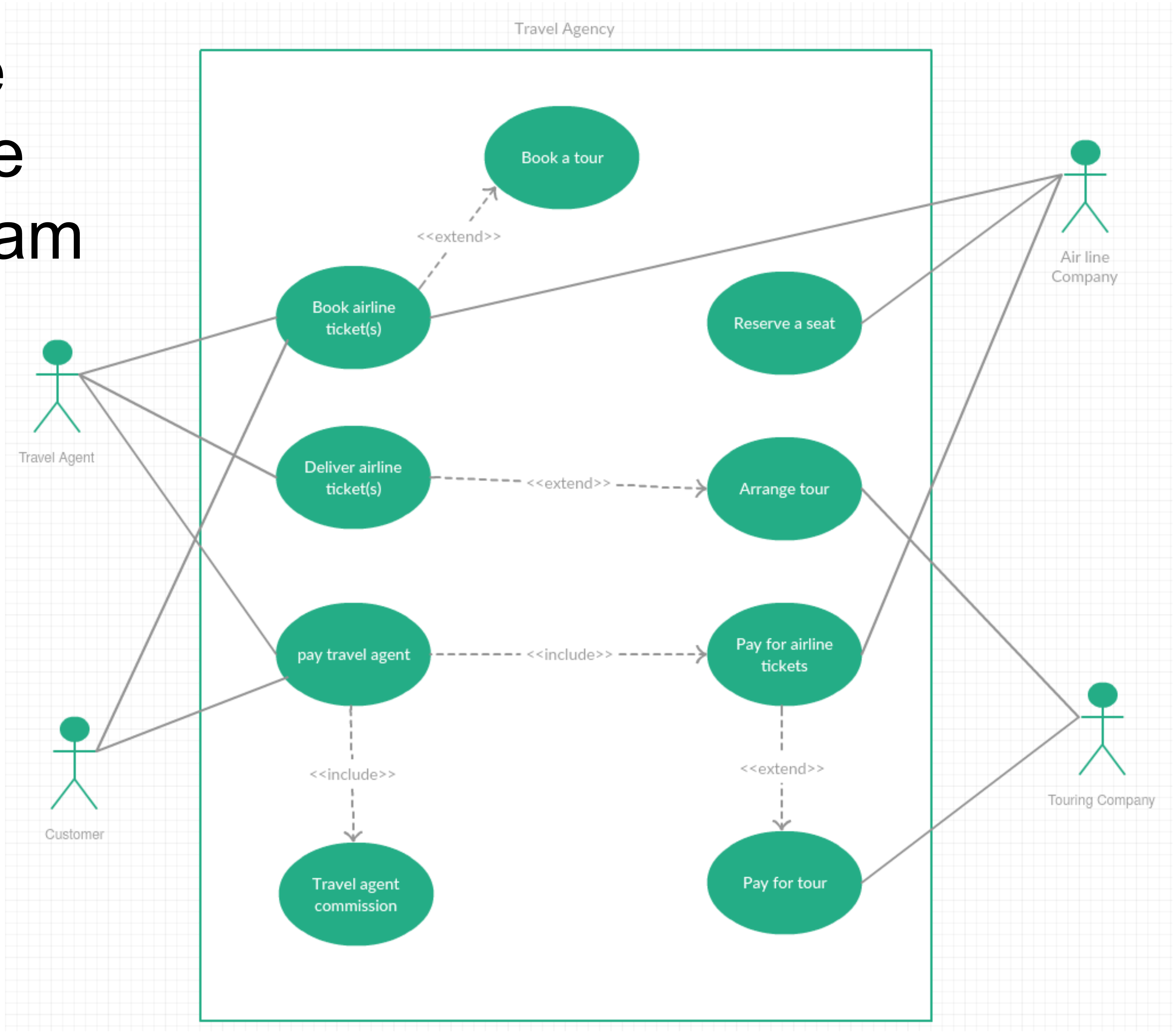
- Use case diagram
- Sequence diagram
- Collaboration diagram
- Statechart diagram
- Activity diagram

Sequence Diagram

A sequence diagram is an interaction diagram. From the name, it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another.

Interaction among the components of a system is very important from implementation and execution perspective. Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.

Use Case Diagram



Activity Diagram

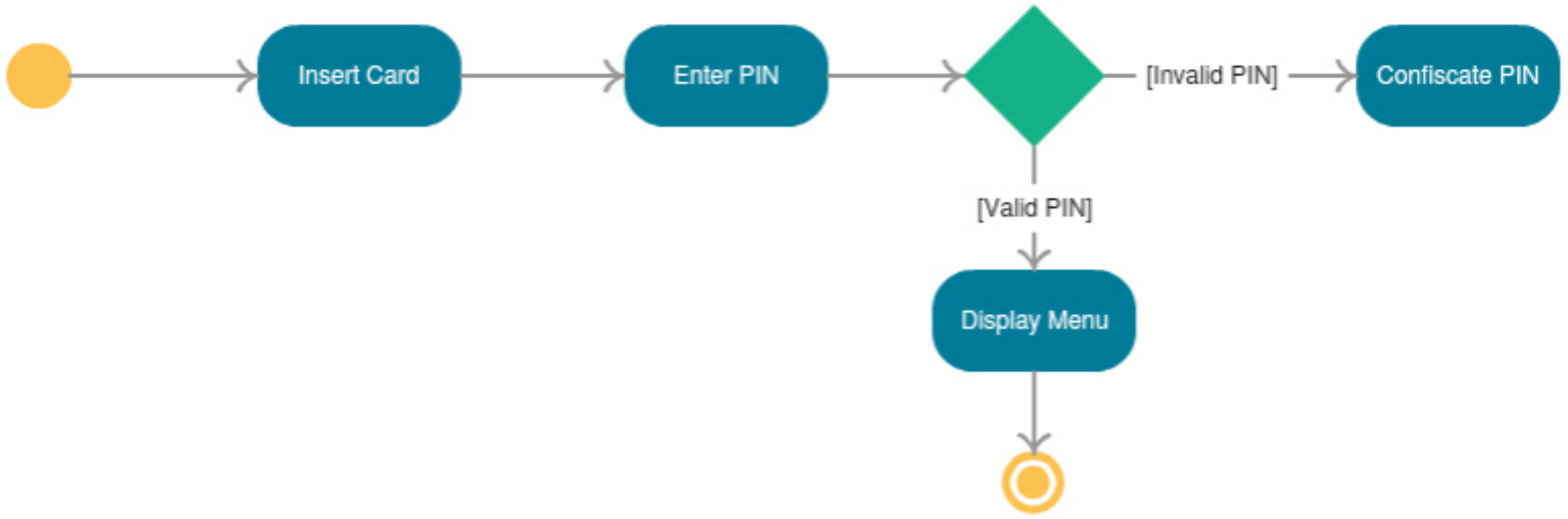
Activity diagram describes the flow of control in a system. It consists of activities and links. The flow can be sequential, concurrent, or branched.

Activities are nothing but the functions of a system. Numbers of activity diagrams are prepared to capture the entire flow in a system.

Activity diagrams are used to visualize the flow of controls in a system. This is prepared to have an idea of how the system will work when executed.

Activity Diagram

Activity diagrams represent workflows in a graphical way. They can be used to describe the business workflow or the operational workflow of any component in a system.

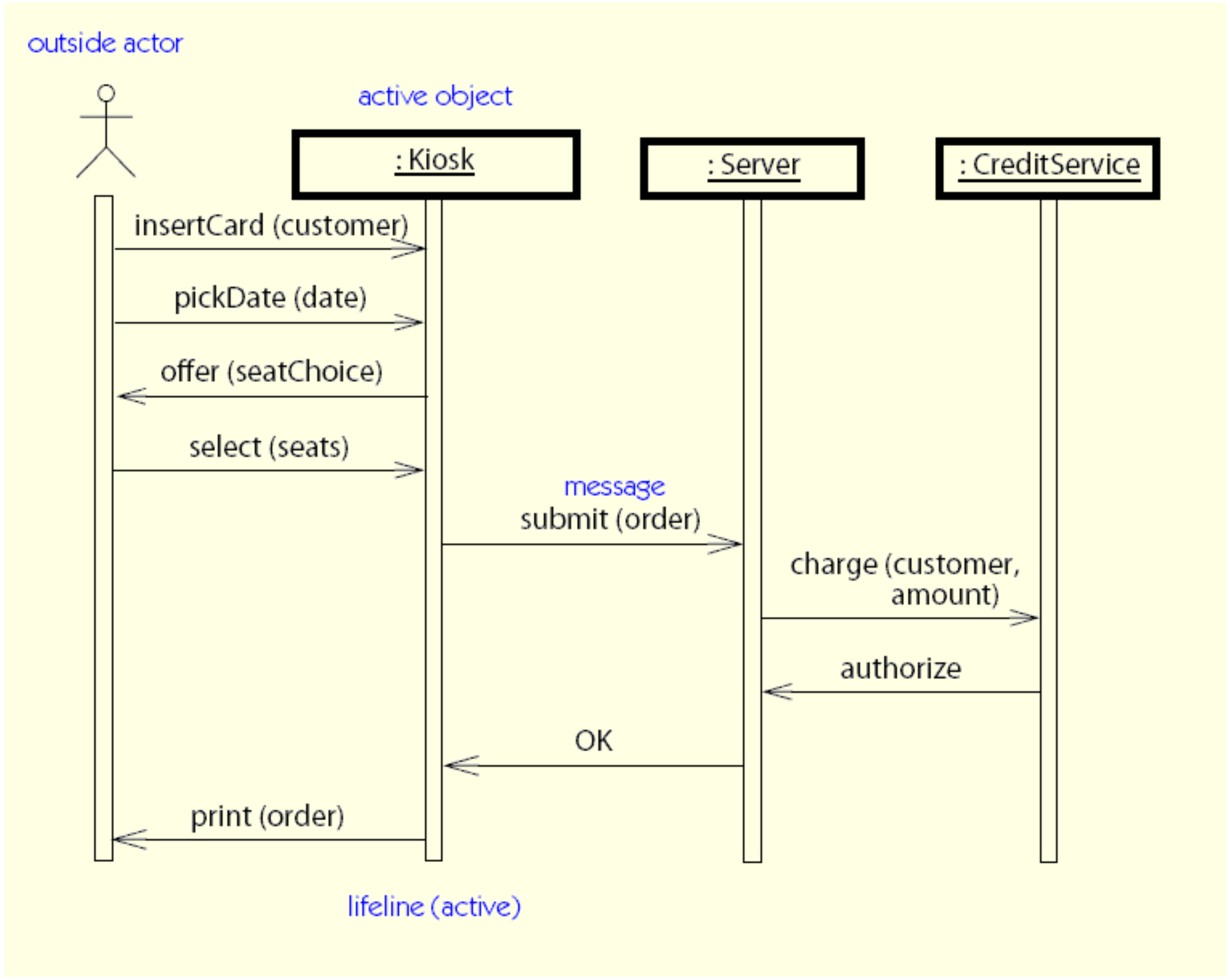


Sequence Diagram

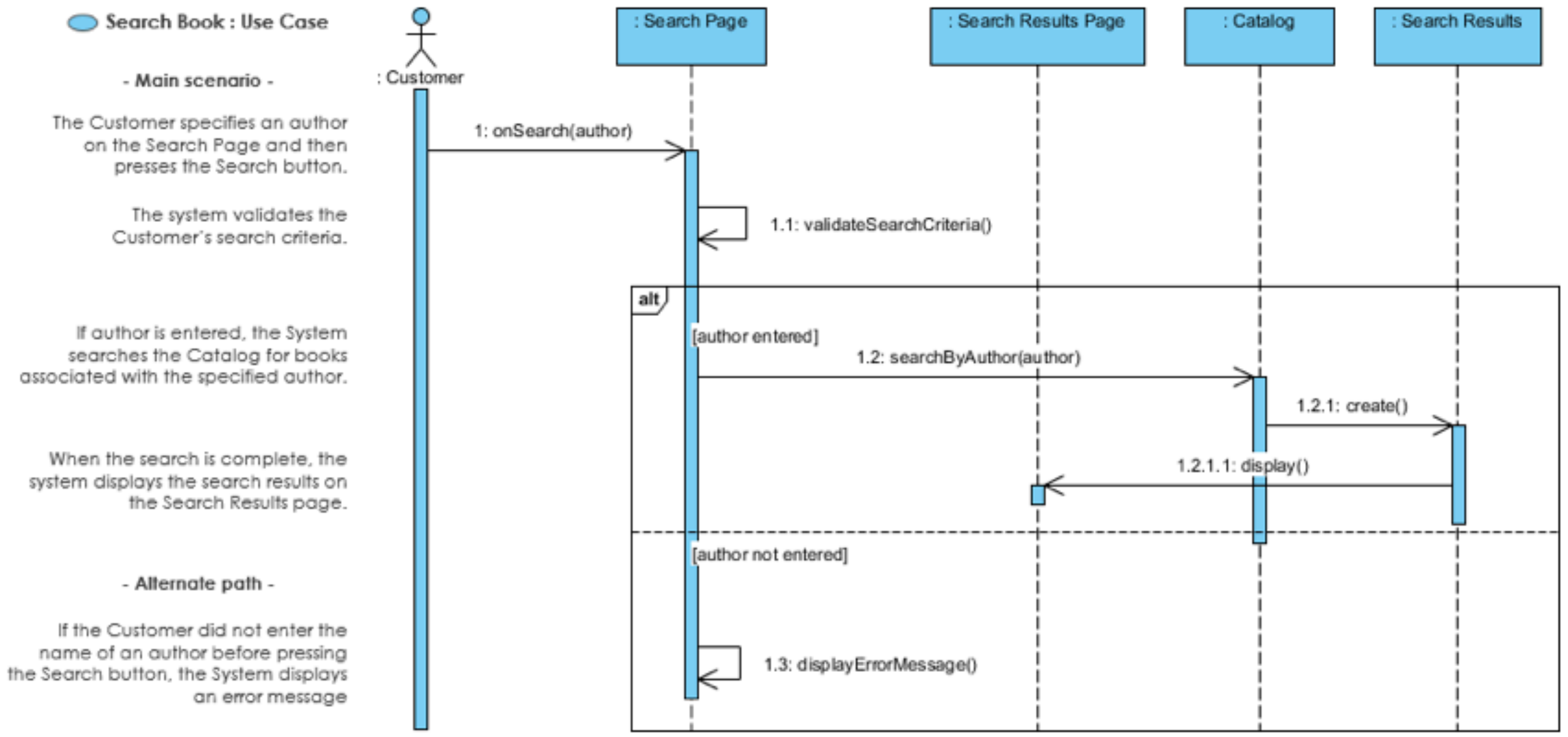
A sequence diagram is an interaction diagram. From the name, it is clear that the diagram deals with some sequences, which are the sequence of messages flowing from one object to another.

Interaction among the components of a system is very important from implementation and execution perspective. Sequence diagram is used to visualize the sequence of calls in a system to perform a specific functionality.

Sequence diagram



Sequence diagram



The Customer specifies an author on the Search Page and then presses the Search button.

The system validates the Customer's search criteria.

If author is entered, the System searches the Catalog for books associated with the specified author.

When the search is complete, the system displays the search results on the Search Results page.

If the Customer did not enter the name of an author before pressing the Search button, the System displays an error message

Additional resources

UML tutorial – **Practical UML : A Hands-On Introduction for Developers**

A very useful tutorial introduction to UML with some very relevant examples and self-testing built in: http://dn.codegear.com/article/31863

Class Diagram Overview- UML 2 Class Diagrams: An Agile Introduction

An overview of the notation for class diagrams with some good examples: http://www.agilemodeling.com/artifacts/classDiagram.htm

Visual Paradigm Tutorials

https://www.visual-paradigm.com/tutorials/

Additional resources

UML Diagram Tool for Agile Software Teams

https://www.visual-paradigm.com/tw/solution/uml/agile-uml-diagram-tool/?gclid=Cj0KCQjwz8bsBRC6ARIsAEyNnvpMElGHjZXs8M1wEoYMuI_WVc2ZosIB9sB5oot-_rgcOmZSYtEkQykaArZmEALw_wcB

Collaboration Diagram

Collaboration diagram is another form of interaction diagram. It represents the structural organization of a system and the messages sent/received. Structural organization consists of objects and links.

The purpose of collaboration diagram is similar to sequence diagram. However, the specific purpose of collaboration diagram is to visualize the organization of objects and their interaction.

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.

State chart diagram is used to visualize the reaction of a system by internal/external factors

Use Case Diagrams

What is a use case?

- A requirements analysis concept
- A case of a use of the system/product
- Describes the system's actions from a the point of view of a user
- Tells a story
    - A sequence of events involving
    - Interactions of a user with the system
- Specifies one aspect of the behavior of a system, without specifying the structure of the system
- Is oriented toward satisfying a user's goal

How do we describe use cases?

- Textual or tabular descriptions
- User stories
- Diagrams

Use Case Descriptions

actors - something with a behavior or role, e.g., a person, another system, organization.

scenario - a specific sequence of actions and interactions between actors and the system, a.k.a. a use case instance

use case - a collection of related success and failure scenarios, describing actors using the system to support a goal.

What is an Actor?

Include all user roles that interact with the system

Include system components only if they responsible for <u>initiating/triggering </u>a use case.

- For example, a timer that triggers sending of an e-mail reminder

primary - a user whose goals are fulfilled by the system

- importance: define user goals

supporting - provides a service (e.g., info) to the system

- importance: clarify external interfaces and protocols

offstage - has an interest in the behavior but is not primary or supporting, e.g., government

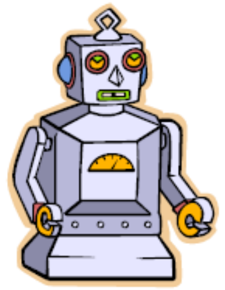
- importance: ensure all interests (even subtle) are identified and satisfied

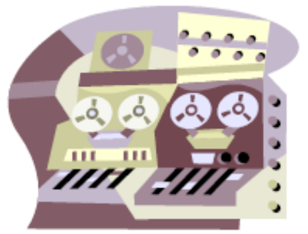
Finding Actors [1]

External objects that produce/consume data:

- Must serve as sources and destinations for data
- Must be external to the system

Humans


Machines


External systems

Organizational Units

Sensors

Finding Actors [2]

Ask the following questions:

- Who are the system’s primary users?
- Who requires system support for daily tasks?
- Who are the system’s secondary users?
- What hardware does the system handle?
- Which other (if any) systems interact with the system in question?
- Do any entities interacting with the system perform multiple roles as actors?
- Which other entities (human or otherwise) might have an interest in the system's output?

What is a user story?

- An abbreviated description of a use case
- Used in agile development

Answers 3 questions:

1.Who?	*As a <type of user>,*
2.Does what?	*I want <some behavior from the system>*
3.And why?	*so that <some value is achieved>*

EXTREME PROGRAMMING
I CAN'T GIVE YOU ALL OF THESE FEATURES IN THE FIRST VERSION.
AND EACH FEATURE NEEDS TO HAVE WHAT WE CALL A "USER STORY."
OKAY, HERE'S A STORY: YOU GIVE ME ALL OF MY FEATURES OR I'LL RUIN YOUR LIFE.
www.dilbert.com scottadams@aol.com
© 2002 United Feature Syndicate, Inc.

Use Case Diagrams

- A picture
  - describes how actors relate to use cases
  - and use cases relate to one another
- Diagrams are not essential
- They are helpful in giving an overview, but only secondary in importance to the textual description
- They do not capture the full information of the actual use cases
- In contrast, text is essential

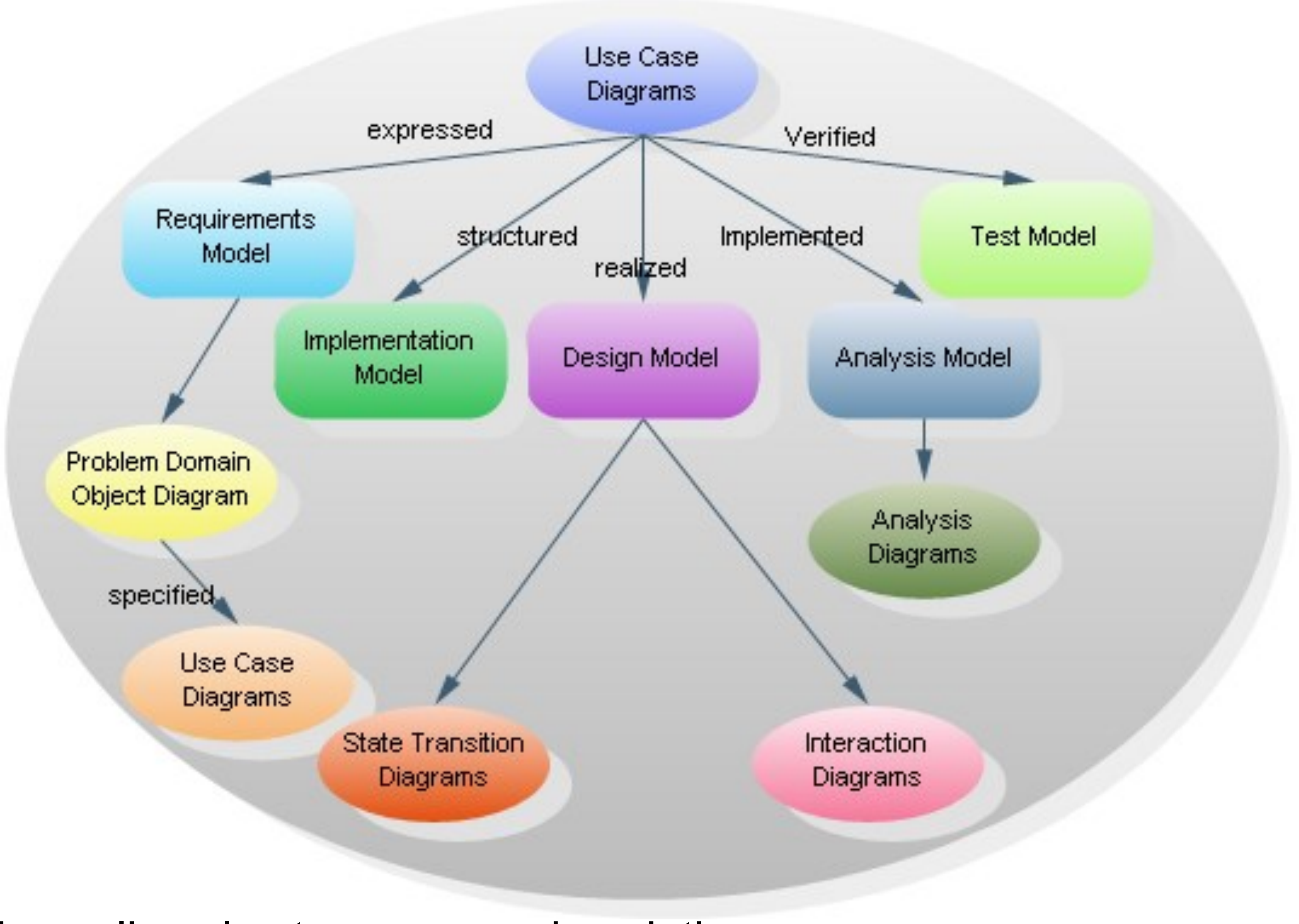
Use Case Diagram Objective

Built in early stages of development

Purpose

- Specify the context of a system
- Capture the requirements of a system
- Validate a systems architecture
- Drive implementation and generate test cases
- Developed by analysts and domain experts

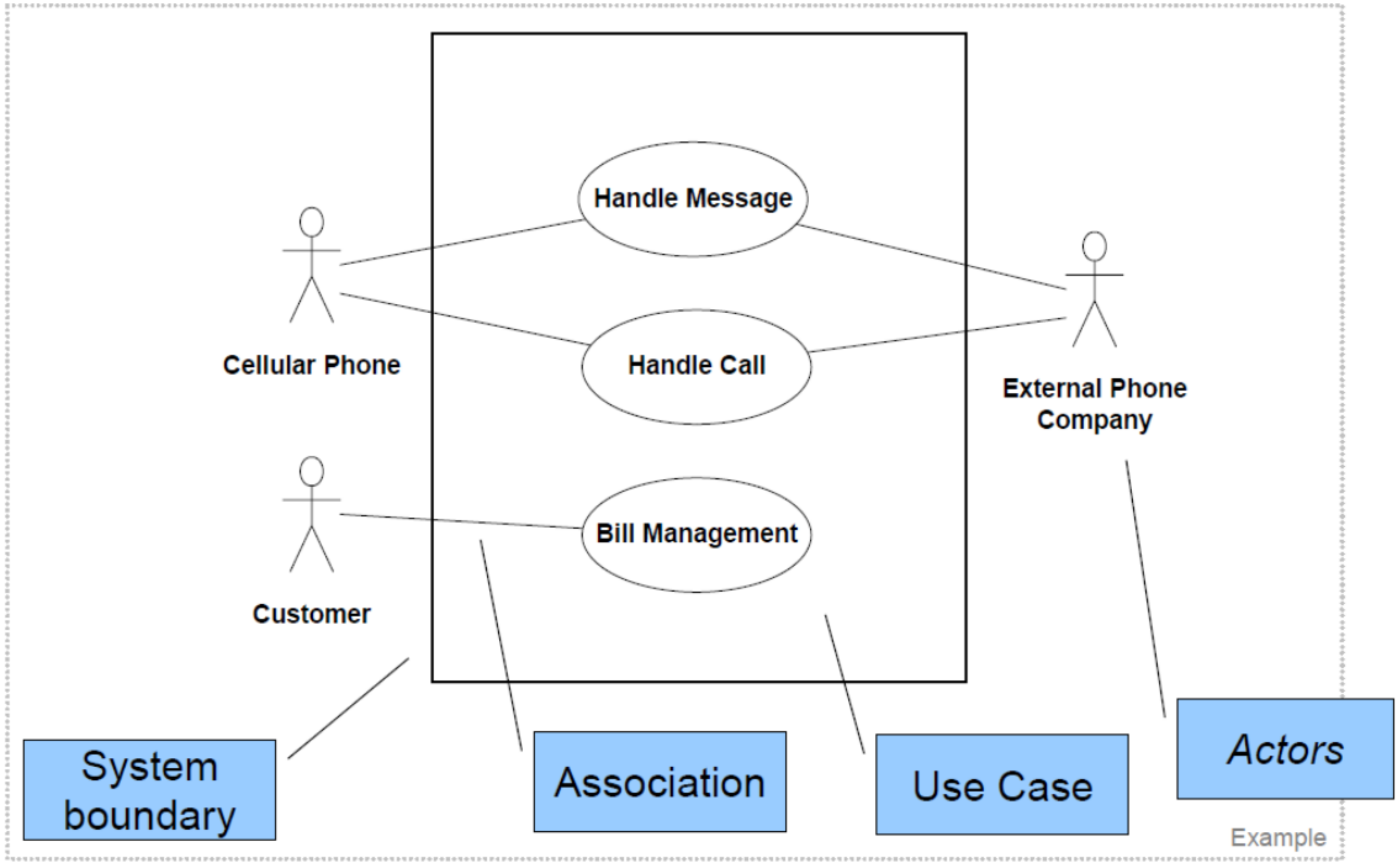
How do use case diagrams fit in?



This applies also to use case descriptions.

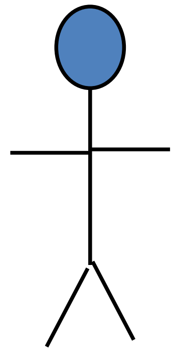
Diagram reproduced from www.edrawsoft.com.

Example Use-Case Diagram



A standard form of use case diagram is defined in the Unified Modeling Language.

Elements of use case diagram: Actor



Name

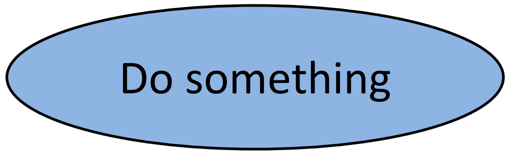
- Actor is someone interacting with use case (system function). Named by noun.
- Similar to the concept of user, but a user can play different roles; (example: a prof. can be instructor and researcher – plays 2 roles with two systems).

- Actor triggers use case.
- Actor has responsibility toward the system (inputs), and Actor have expectations from the system (outputs).

Elements of use case diagram: Use Case

- Actor triggers use case.
- Actor has responsibility toward the system (inputs), and Actor have expectations from the system (outputs).

Elements of use case diagram: Use Case



- System function (process – automated or manual).
- Named by verb.
- Each Actor must be linked to a use case, while some use cases may not be linked to actors.

USER/ACTOR	USER GOAL = Use Case
Order clerk	Look up item availability Create new order Update order
Shipping clerk	Record order fulfillment Record back order
Merchandising manager	Create special promotion Produce catalog activity report

Elements of use case diagram: Other details

——— Connection between Actor and Use Case

▭ Boundary of system

$\xrightarrow{\text{<<include>>}}$

Include relationship between Use Cases (one UC must call another; e.g., Login UC includes User Authentication UC)

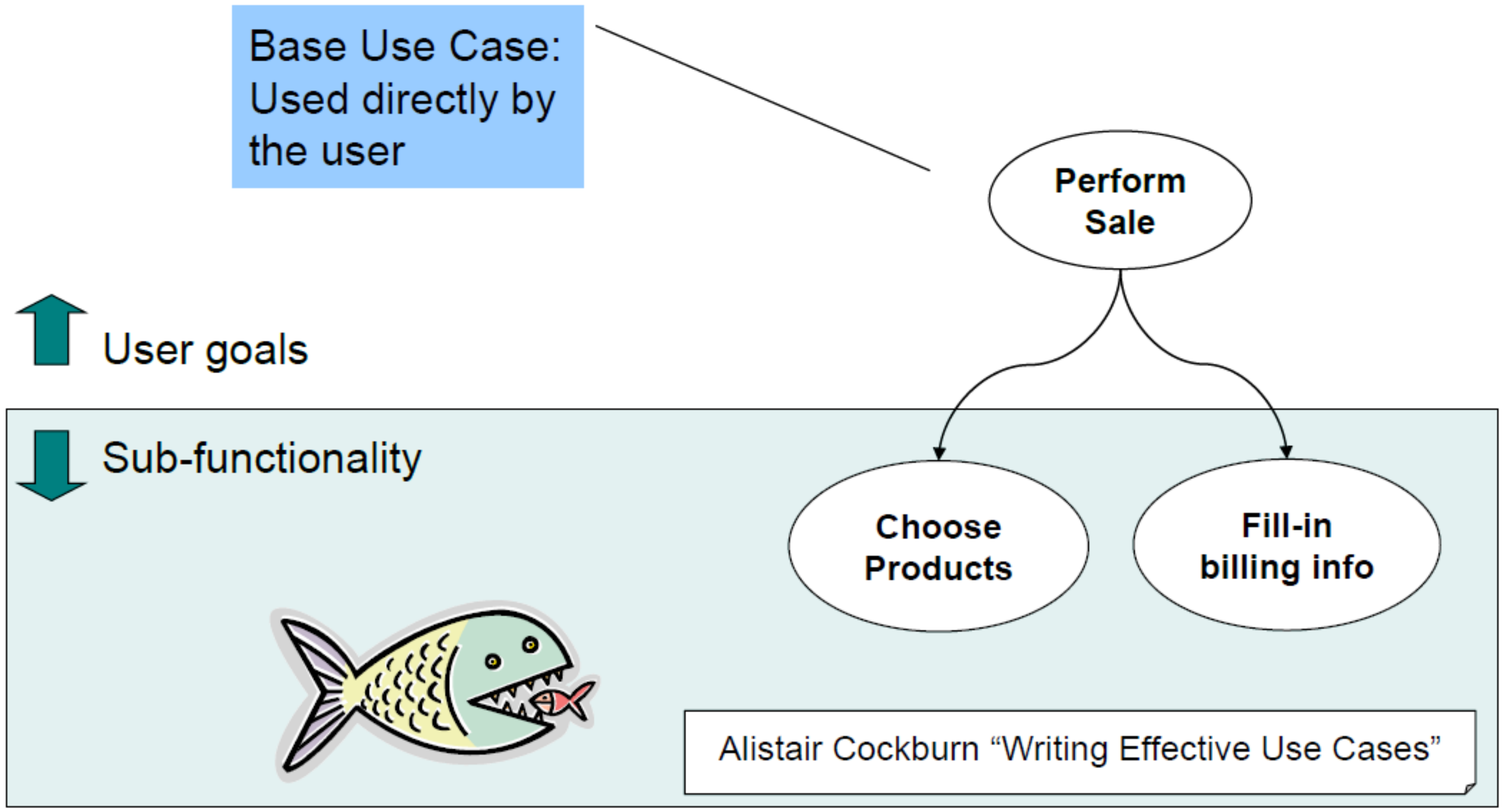
$\xrightarrow{\text{<<extend>>}}$

Extend relationship between Use Cases (one UC calls Another under certain condition; think of if-then decision points)

Linking Use Cases

- Association relationships
- Generalization relationships
  - One element (child) "is based on" another element (parent)
- Include relationships
  - One use case (base) includes the functionality of another (inclusion case)
  - Supports re-use of functionality
- Extend relationships
  - One use case (extension) extends the behavior of another (base)

Use Case Levels



Alistair Cockburn “Writing Effective Use Cases”

How to create use case diagram

1 List main system functions (use cases) in a column:
  - think of business events demanding system’s response
  - users’ goals/needs to be accomplished via the system
  - Create, Read, Update, Delete (CRUD) data tasks
  - Naming use cases – user’s needs usually can be translated in data tasks

2. Draw ovals around the function labels
3. Draw system boundary
4. Draw actors and connect them with use cases (if more intuitive, this can be done as step 2)
5. Specify include and extend relationships between use cases (yes, at the end - not before, as this may pull you into process thinking, which does not apply in UC diagramming).

Use-Case Diagram Case Study [1]

Vending Machine

After client interview the following system scenarios were identified:

- A customer buys a product
- The supplier restocks the machine
- The supplier collects money from the machine

On the basis of these scenarios, the following three actors can be identified:

Use-Case Diagram Case Study [1]

Vending Machine

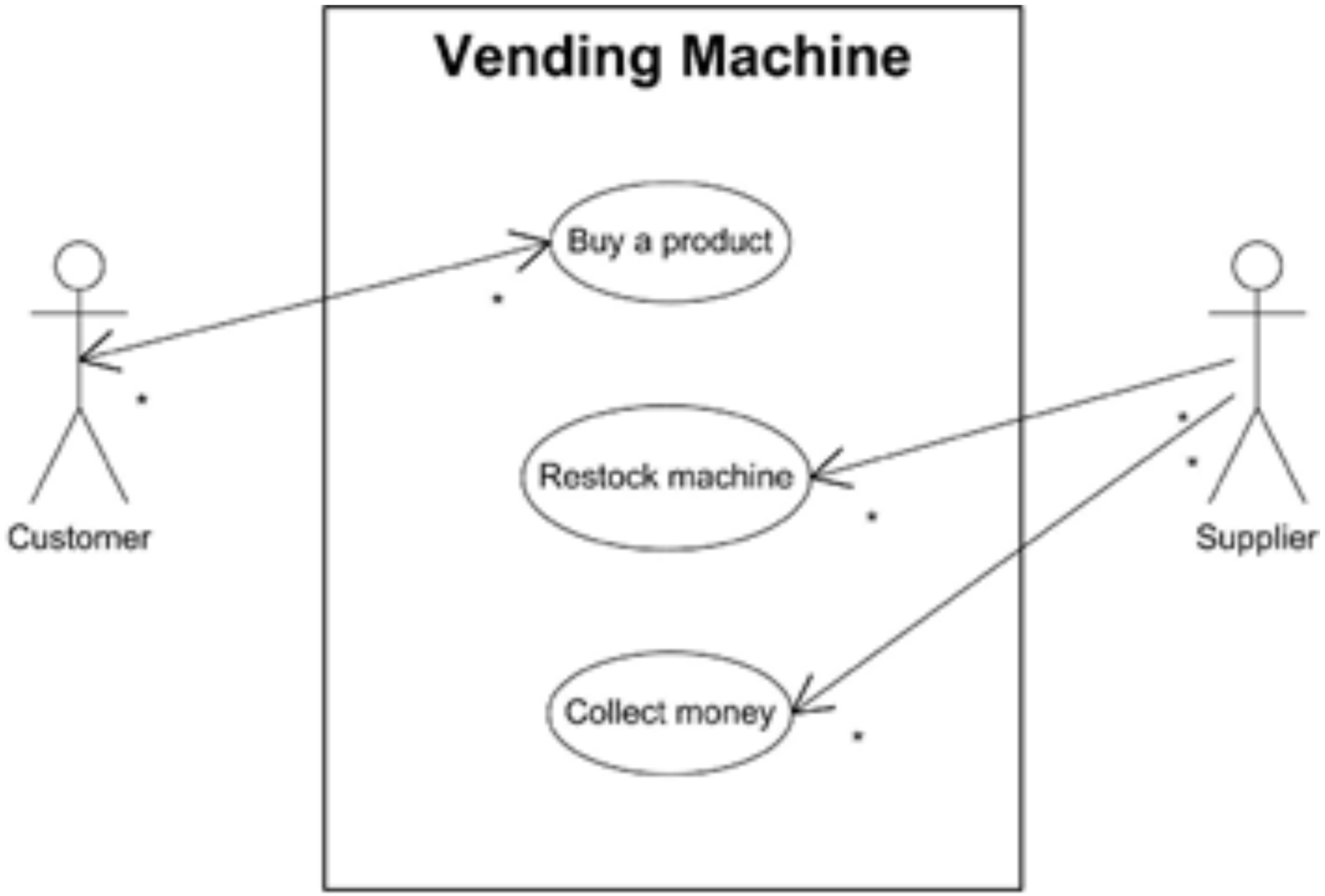
After client interview the following system scenarios were identified:

- A customer buys a product
- The supplier restocks the machine
- The supplier collects money from the machine

On the basis of these scenarios, the following three actors can be identified:

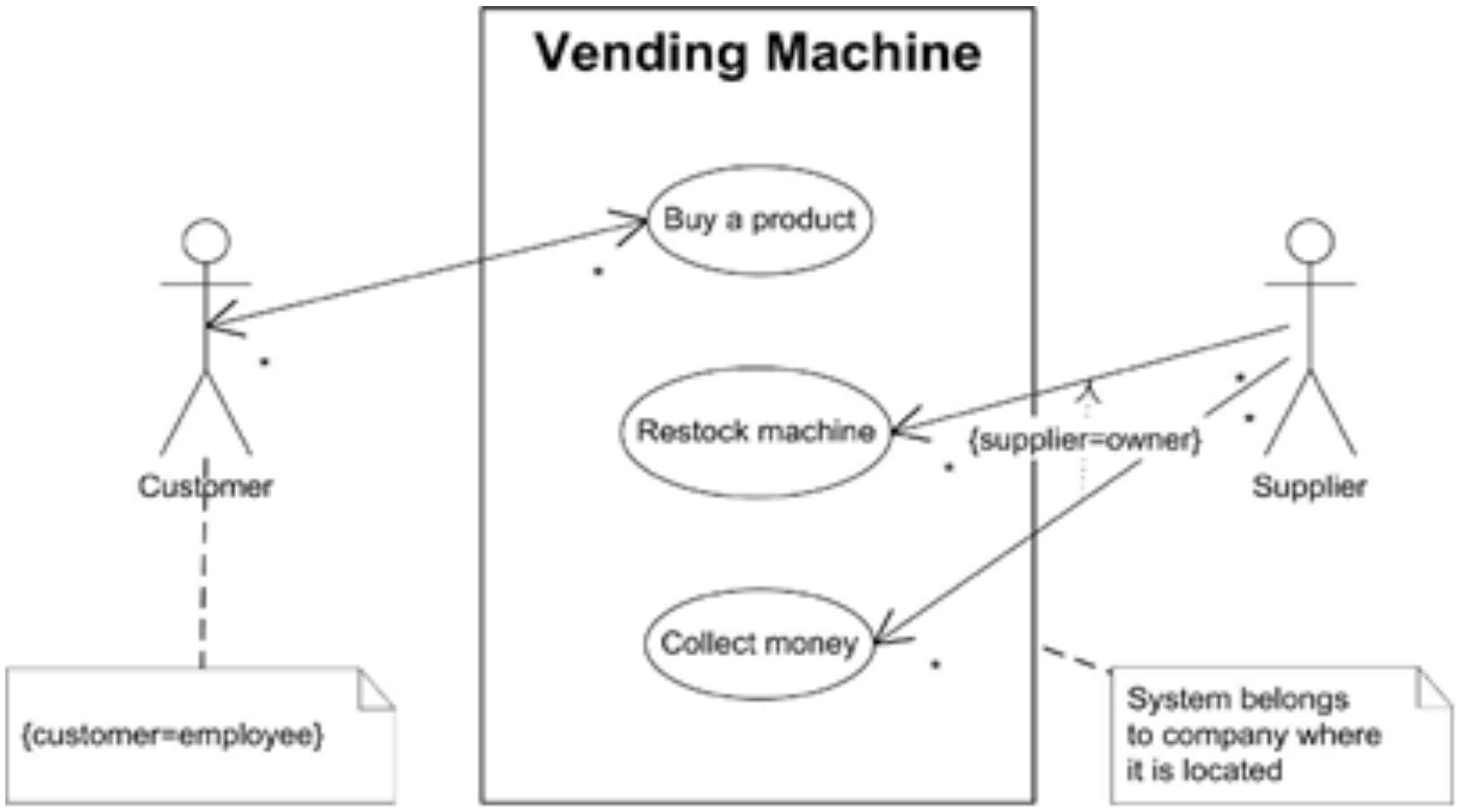
Customer; Supplier; Collector (in this case Collector=Supplier)

Use-Case Diagram Case Study [2]



Use-Case Diagram Case Study [3]

Introducing annotations (notes) and constraints.



Good resources

The Unified Modeling Language

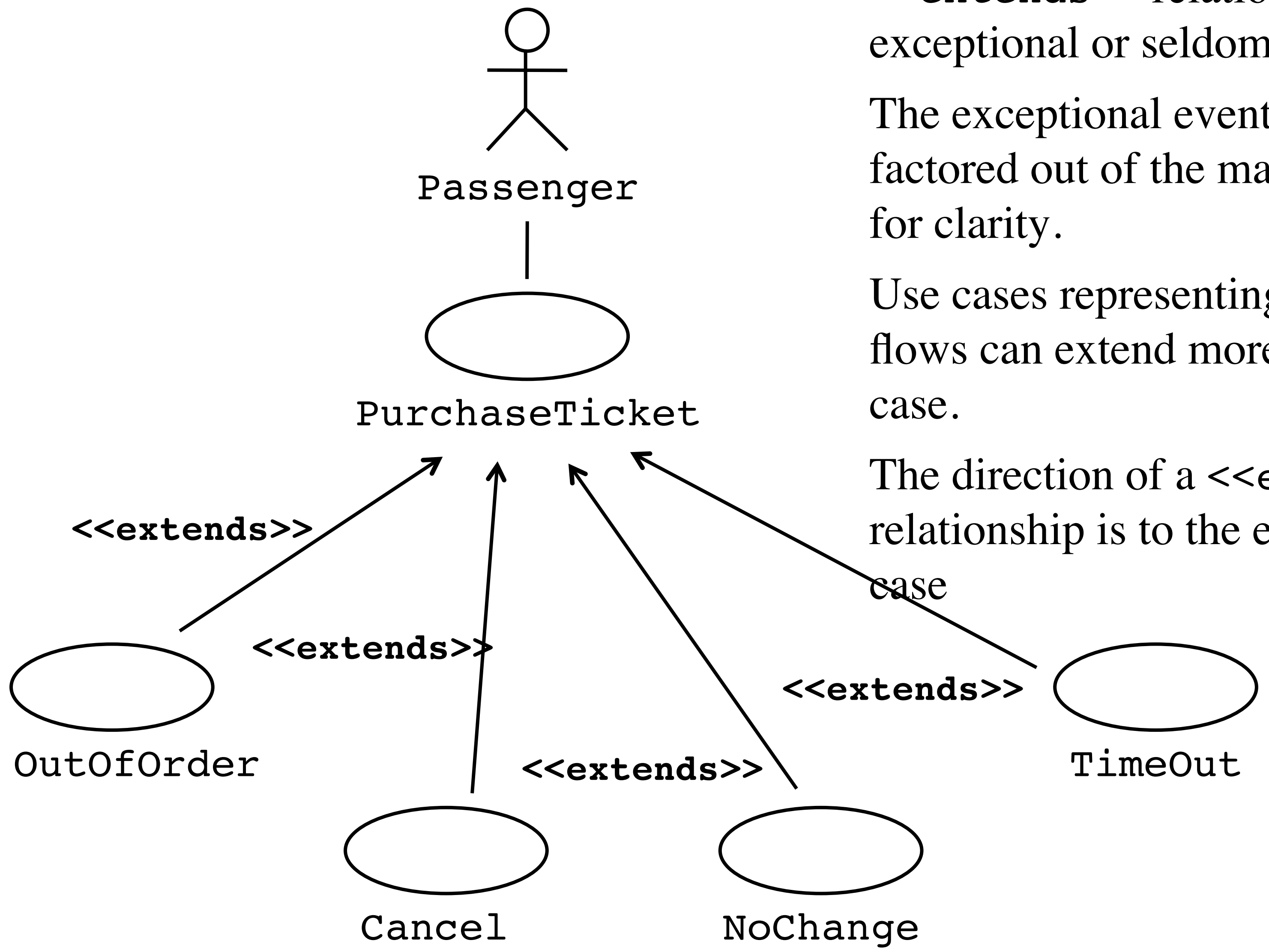
http://www.uml-diagrams.org

UML Diagram Types Guide: Learn About All Types of UML Diagrams with Examples

https://creately.com/blog/diagrams/uml-diagram-types-examples/#ClassDiagram

UML diagrams examples with explanation

The <<extends>> Relationship



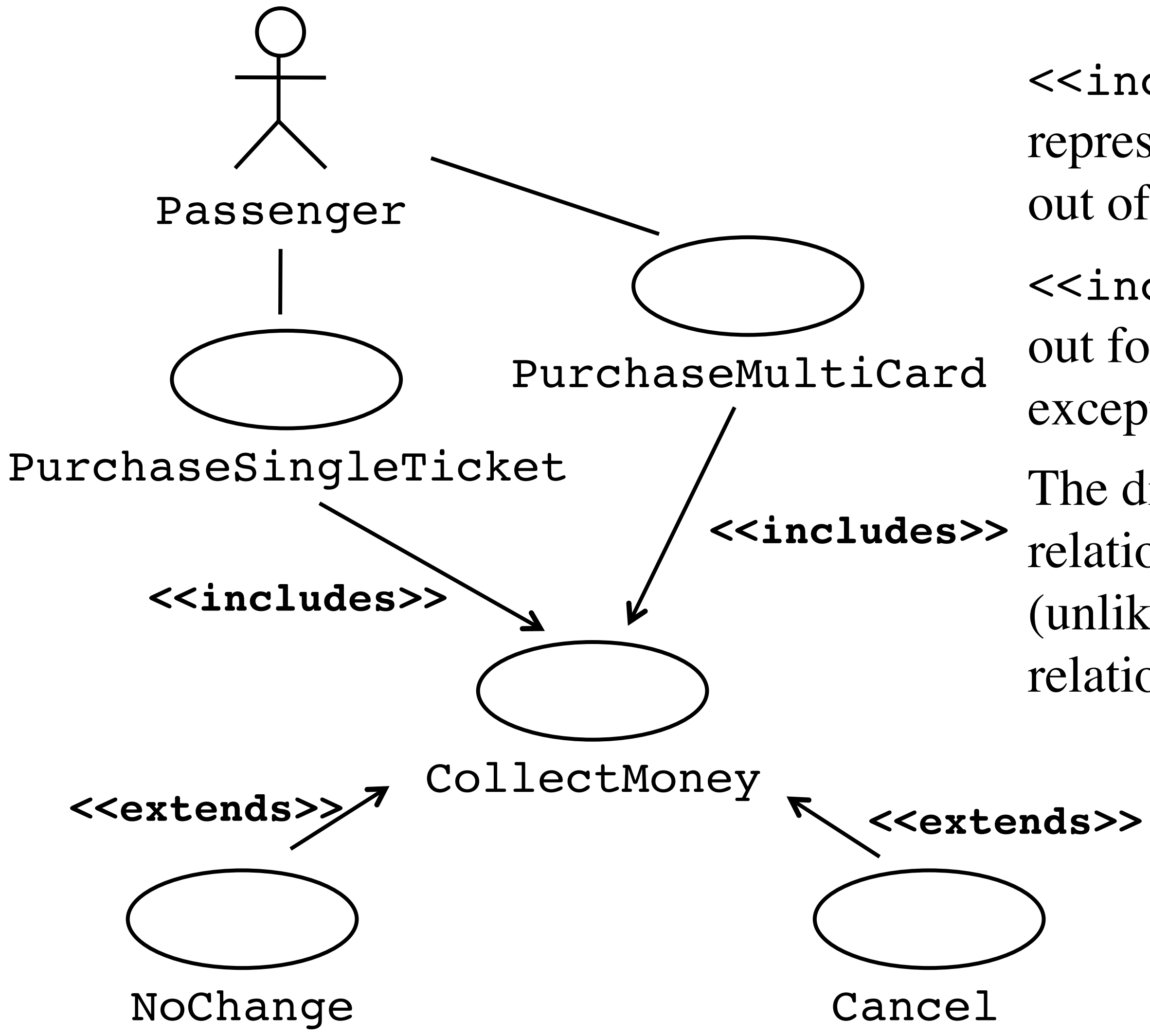
<<extends>> relationships represent exceptional or seldom invoked cases.

The exceptional event flows are factored out of the main event flow for clarity.

Use cases representing exceptional flows can extend more than one use case.

The direction of a <<extends>> relationship is to the extended use case

The <<includes>> Relationship

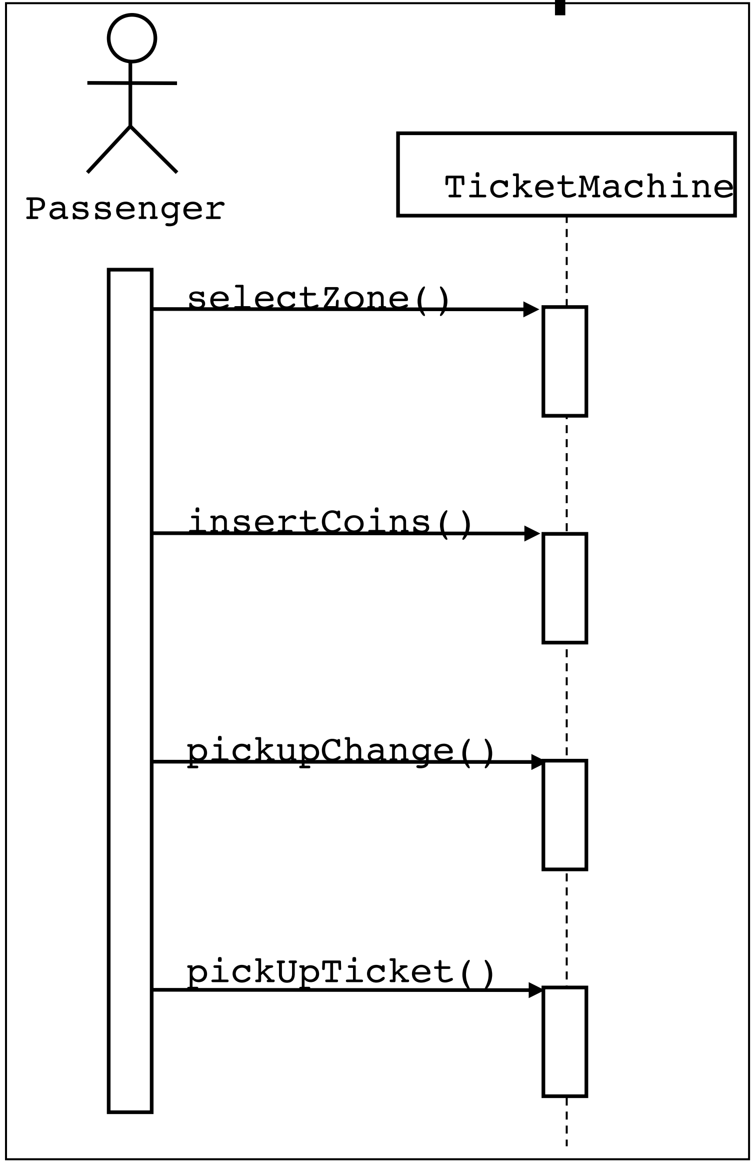


`<<includes>>` relationship represents behavior that is factored out of the use case.

`<<includes>>` behavior is factored out for reuse, not because it is an exception.

The direction of a `<<includes>>` relationship is to the using use case (unlike `<<extends>>` relationships).

UML sequence diagrams



- Used during requirements analysis
  - To refine use case descriptions
  - to find additional objects (“participating objects”)
- Used during system design
  - to refine subsystem interfaces
- ***Classes*** are represented by columns
- ***Messages*** are represented by arrows
- ***Activations*** are represented by narrow rectangles
- ***Lifelines*** are represented by dashed lines

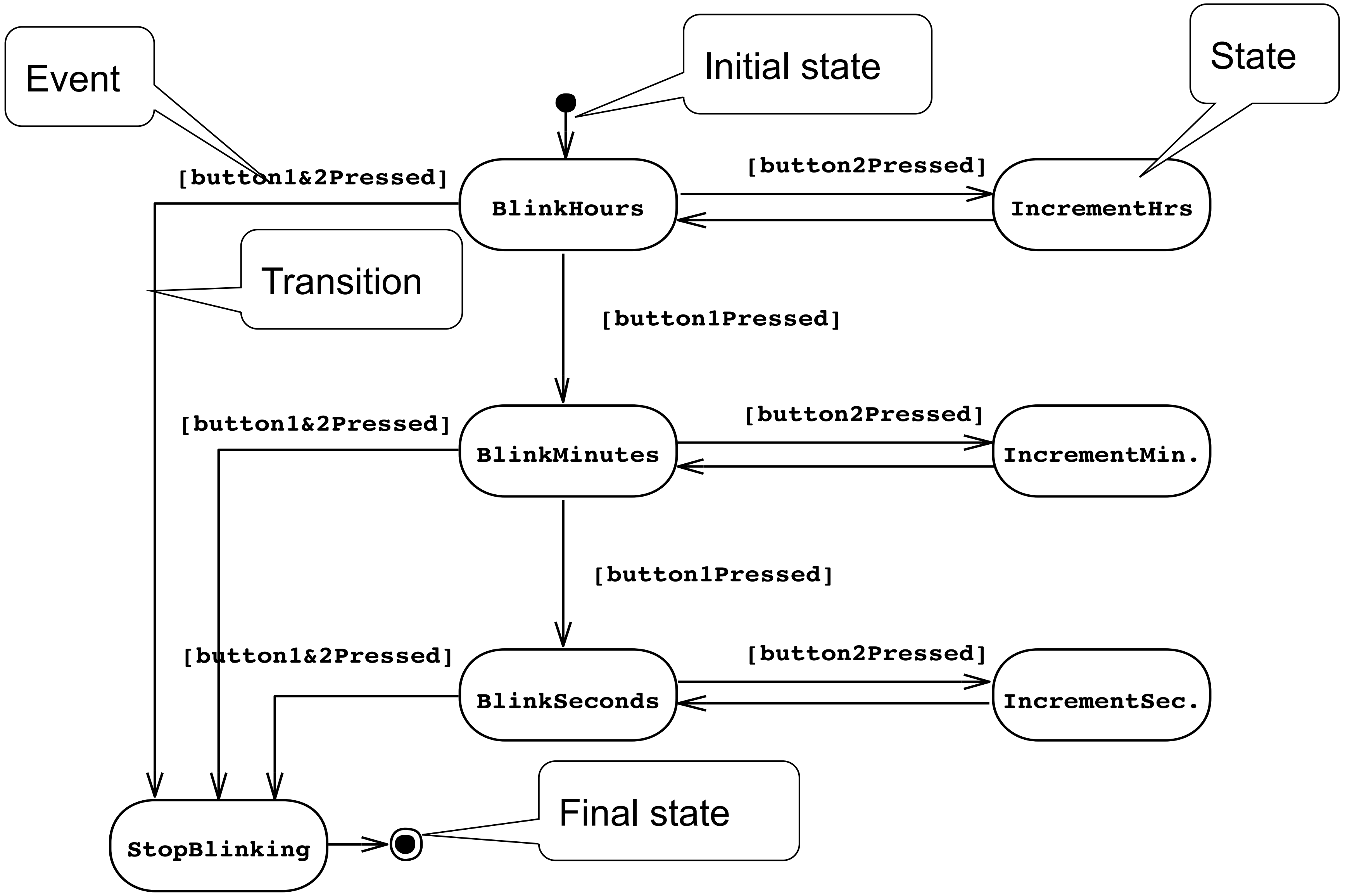
UML sequence diagram represent behavior in terms of interactions.

Useful to find missing objects.

Time consuming to build but worth the investment.

Complement the class diagrams (which represent structure).

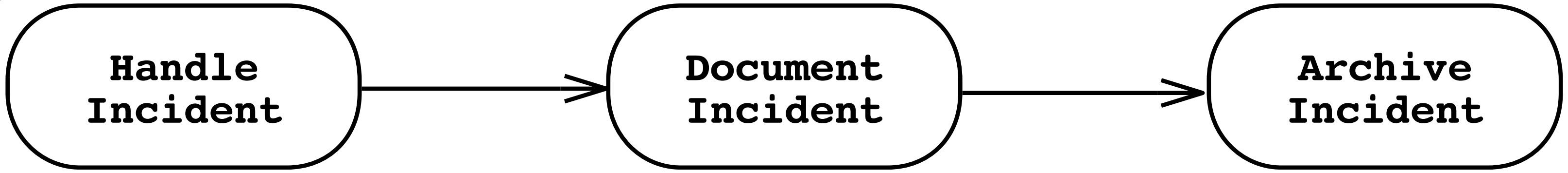
State Chart Diagrams



Represent behavior as states and transitions

Activity Diagrams

An activity diagram shows flow control within a system



An activity diagram is a special case of a state chart diagram in which states are activities (“functions”)

Two types of states:

Action state:

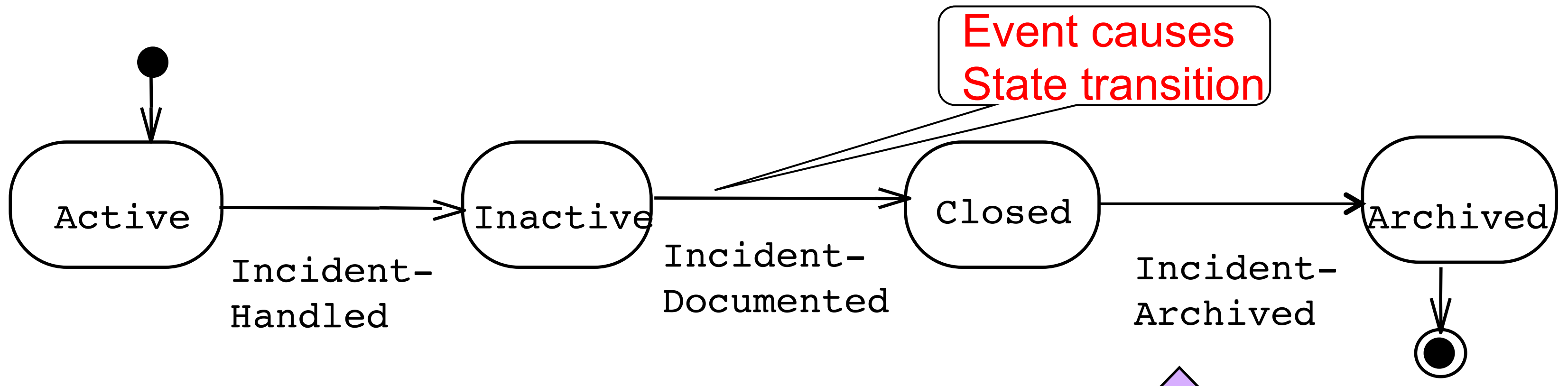
- Cannot be decomposed any further
- Happens “instantaneously” with respect to the level of abstraction used in the model

Activity state:

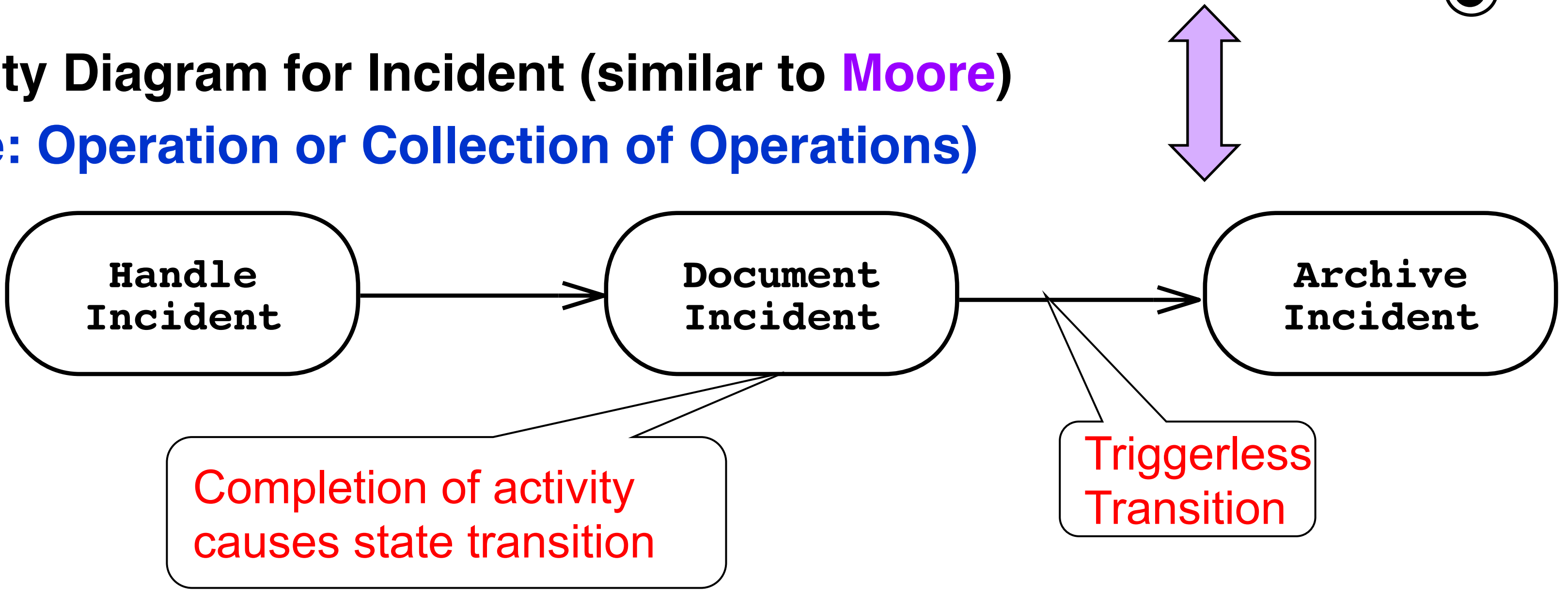
- Can be decomposed further
- The activity is modeled by another activity diagram

Statechart Diagram vs. Activity Diagram

Statechart Diagram for Incident (similar to Mealy Automaton)
(State: Attribute or Collection of Attributes of object of type Incident)



Activity Diagram for Incident (similar to Moore)
(State: Operation or Collection of Operations)



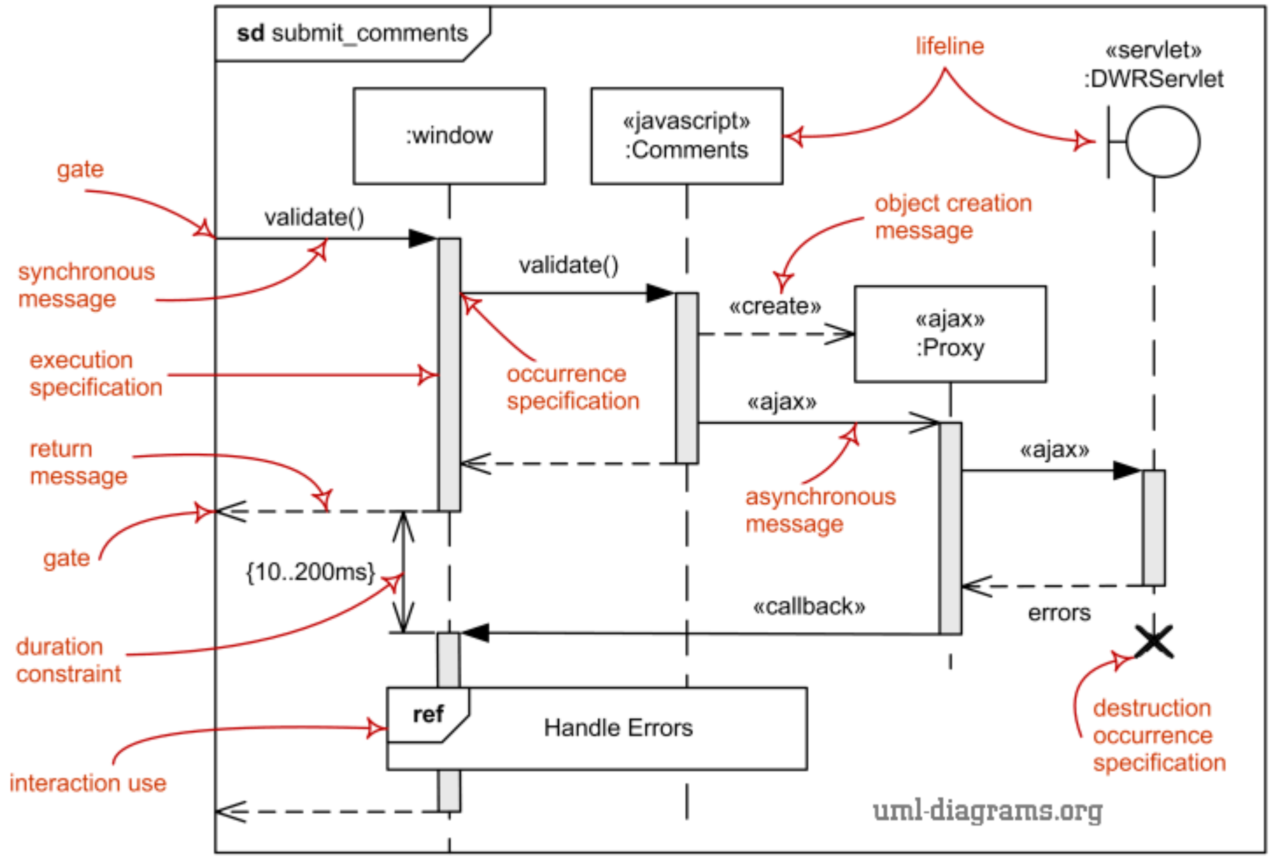
UML Sequence Diagrams

Sequence diagram is the most common kind of **interaction diagram**, which focuses on the **message** interchange between a number of **lifelines**.

Sequence diagram describes an interaction by focusing on the sequence of messages that are exchanged, along with their corresponding occurrence specifications on the lifelines.

The following nodes and edges are typically drawn in a **UML sequence diagram**: **lifeline**, **execution specification**, **message**, **combined fragment**, **interaction use**, **state invariant**, **continuation**, **destruction occurrence**.

Major elements of the sequence diagram are shown on the picture below.



IP

Major elements of UML sequence diagram.

Good resources
Agile and SCRUM

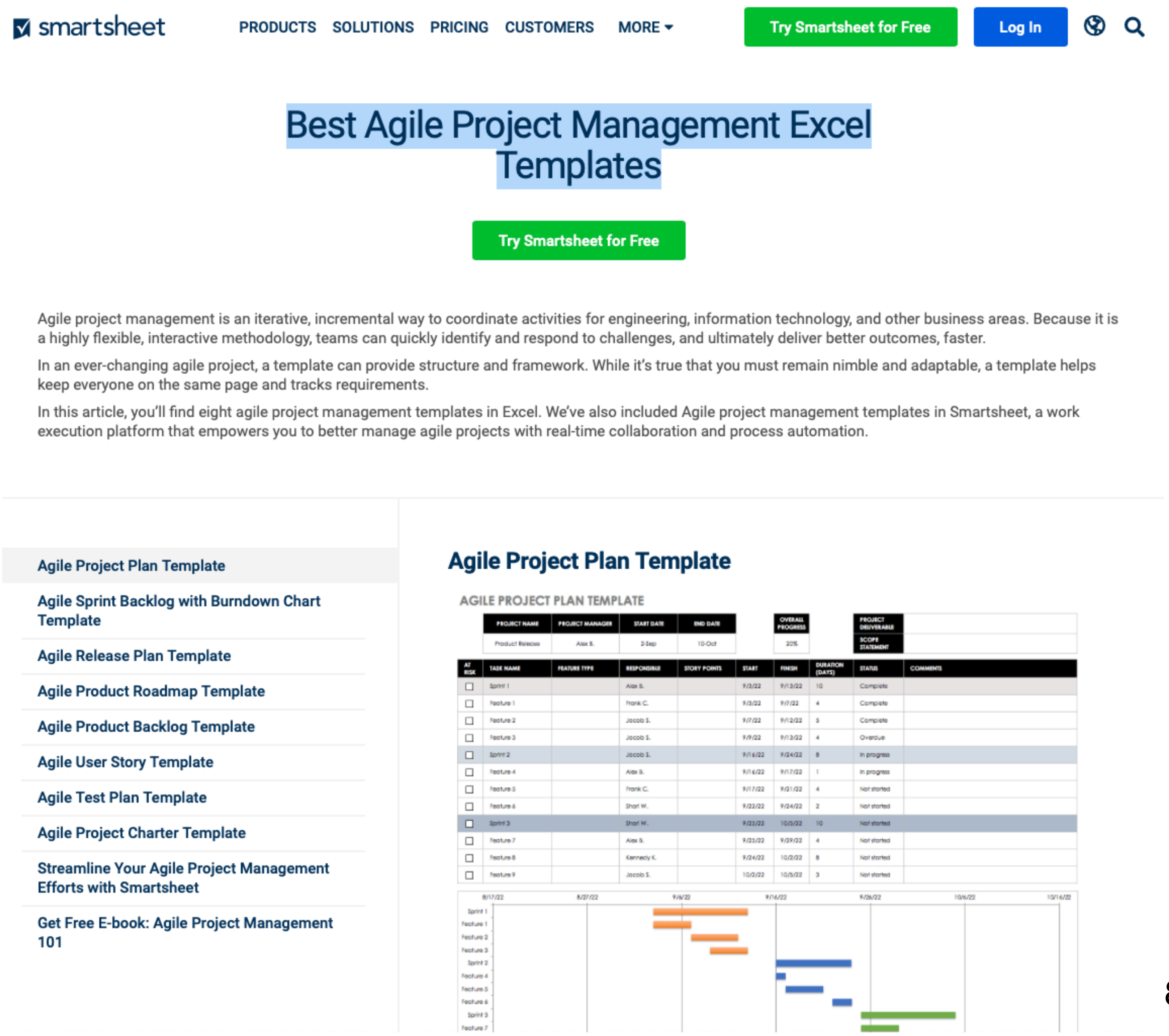
Agile Alliance

https://www.agilealliance.org

https://www.agilealliance.org/glossary/scrum/

Best Agile Project Management Excel Templates

https://www.smartsheet.com/agile-project-management-excel-templates



smartsheet PRODUCTS SOLUTIONS PRICING CUSTOMERS MORE Try Smartsheet for Free Log In

Best Agile Project Management Excel Templates

Try Smartsheet for Free

Agile project management is an iterative, incremental way to coordinate activities for engineering, information technology, and other business areas. Because it is a highly flexible, interactive methodology, teams can quickly identify and respond to challenges, and ultimately deliver better outcomes, faster.

In an ever-changing agile project, a template can provide structure and framework. While it's true that you must remain nimble and adaptable, a template helps keep everyone on the same page and tracks requirements.

In this article, you'll find eight agile project management templates in Excel. We've also included Agile project management templates in Smartsheet, a work execution platform that empowers you to better manage agile projects with real-time collaboration and process automation.

Agile Project Plan Template

AGILE PROJECT PLAN TEMPLATE

PROJECT NAME	PROJECT MANAGER	START DATE	END DATE
Product Release	Alex B.	2-Sep	10-Oct

OVERALL PROGRESS
20%

PROJECT DELIVERABLE	
SCOPE STATEMENT	

AT RISK	TASK NAME	FEATURE TYPE	RESPONSIBLE	STORY POINTS	START	FINISH	DURATION (DAYS)	STATUS	COMMENTS
☐	Sprint 1		Alex B.		9/3/22	9/13/22	10	Complete	
☐	Feature 1		Frank C.		9/3/22	9/7/22	4	Complete	
☐	Feature 2		Jacob S.		9/7/22	9/12/22	5	Complete	
☐	Feature 3		Jacob S.		9/9/22	9/13/22	4	Overdue	
☐	Sprint 2		Jacob S.		9/16/22	9/24/22	8	In progress	
☐	Feature 4		Alex B.		9/16/22	9/17/22	1	In progress	
☐	Feature 5		Frank C.		9/17/22	9/21/22	4	Not started	
☐	Feature 6		Shari W.		9/22/22	9/24/22	2	Not started	
☐	Sprint 3		Shari W.		9/25/22	10/5/22	10	Not started	
☐	Feature 7		Alex B.		9/25/22	9/29/22	4	Not started	
☐	Feature 8		Kennedy K.		9/24/22	10/2/22	8	Not started	
☐	Feature 9		Jacob S.		10/2/22	10/5/22	3	Not started	

References

1. UML Tutorial

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

2. Use Case Diagrams

http://csis.pace.edu/~marchese/CS389/L9/Use%20Case%20Diagrams.pdf

3. Grady Booch, James Rumbaugh, Ivar Jacobson, The Unified Modeling Language User Guide, 2000, Addison Wesley

4. The Unified Modeling Language

http://www.uml-diagrams.org

5. Lecture for Chapter 2, Modeling with UML