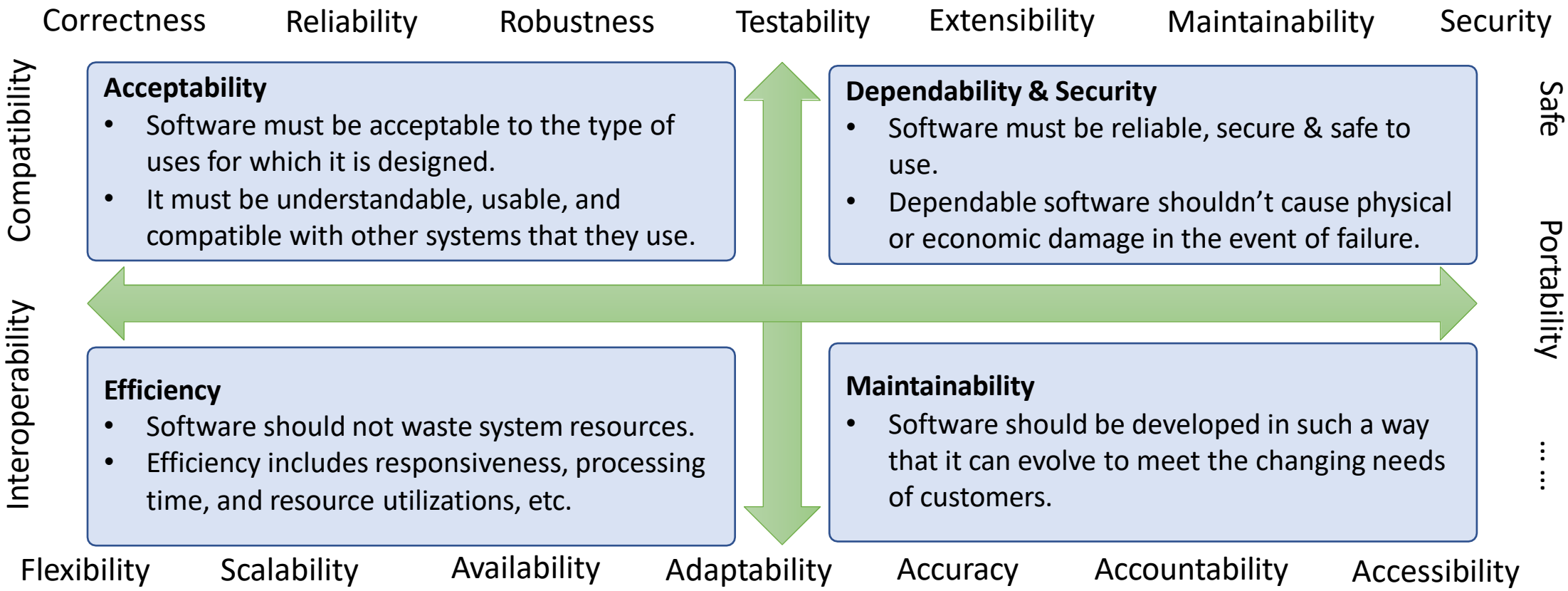
Software Engineering CS335FZ

1. The style of the final exam
2. Reviewing L1-L11
3. Q&A

L1

Quality of Software Product



Software quality is not just concerned with what the software does, it also includes the software's behavior while it is running and the structure and organization of the system programs and associated documentation.

L1

Types of Software (1)

- Stand-alone applications
  - Applications that run on personal computers, mobile devices or mainframes
  - For example, Adobe Photoshop, Windows Calculator, etc.
- Interactive transaction-based applications
  - Applications that run on a remote computer, but accessed by users from their own computers
  - For example, Web applications, eBay Apps, cloud-based services, etc.
- Embedded control systems
  - Software that control and manage hardware devices such as refrigerator temperature control, microware cooking functions, oil pump control, etc.



L1

Types of Software(2)

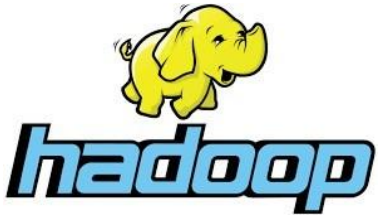
- Batch processing systems
  - Application systems that process data in large batches
  - For example, cell phone billing systems, staff salary payment systems, bank transaction processing systems, etc.

- Entertainment systems
  - Most of these systems are games that can run on personal computers or special console hardware, e.g., Xbox, PS4, Nintendo Switch, etc.

- System for modeling and simulation
  - Application systems developed for scientists and engineers to model and simulate physical processes, chemical reactions, protein folding, visualization, etc.

L1

Types of Software(3)



- Data collection and analysis systems
  - Software systems that collect data from their environment and send that data to other systems for processing
  - For example, data warehouse, data lake, big data analytics systems, sensor data processing systems, etc.
- Systems of systems
  - Systems or subsystems used in the large-scale enterprise or organizations.
  - For example, Enterprise Resource Planning (ERP) system, or systems that are composed from other discrete systems.

NOTE: there are on clear boundaries between these types of software. As the world is becoming more and more software controlled, other types of software may emerge, thus new software processes are likely to be developed.

L1

The Importance of Software Engineering

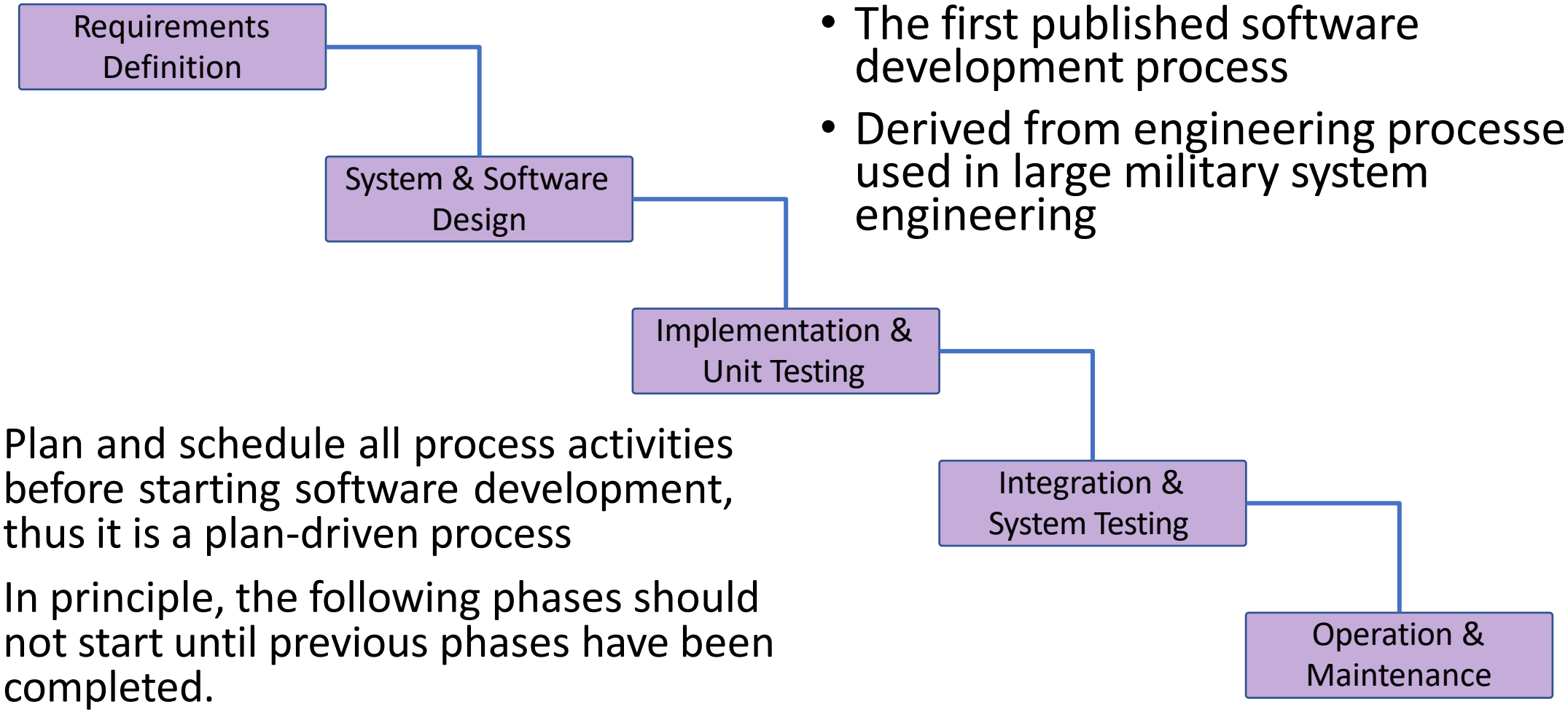
To be able to produce reliable and trustworthy systems economically and quickly.

It is usually cheaper, in the long run, to use software engineering methods and techniques.

- How to plan software development activities across the long term?
  - E.g., The electronic trading platform project (TAURUS) for the UK stock exchange lasted for more than 14 years
- How to allocate the budget?
  - E.g., The air traffic control project (FAA Advanced Automation System) for US federal aviation administration received 3 – 6 Billion US dollars
- How to deal with hundreds and thousands of requirements gathered from users? And how to prioritize them?
- How to deal with conflict requirements from different stakeholders?
- How to design an architecture that can scale to thousands of servers concurrently?
- How to manage hundreds and thousands of source code files and their dependencies?
- How to test, deploy, configure, and manage software products?
- How to …

L2

The Waterfall Model



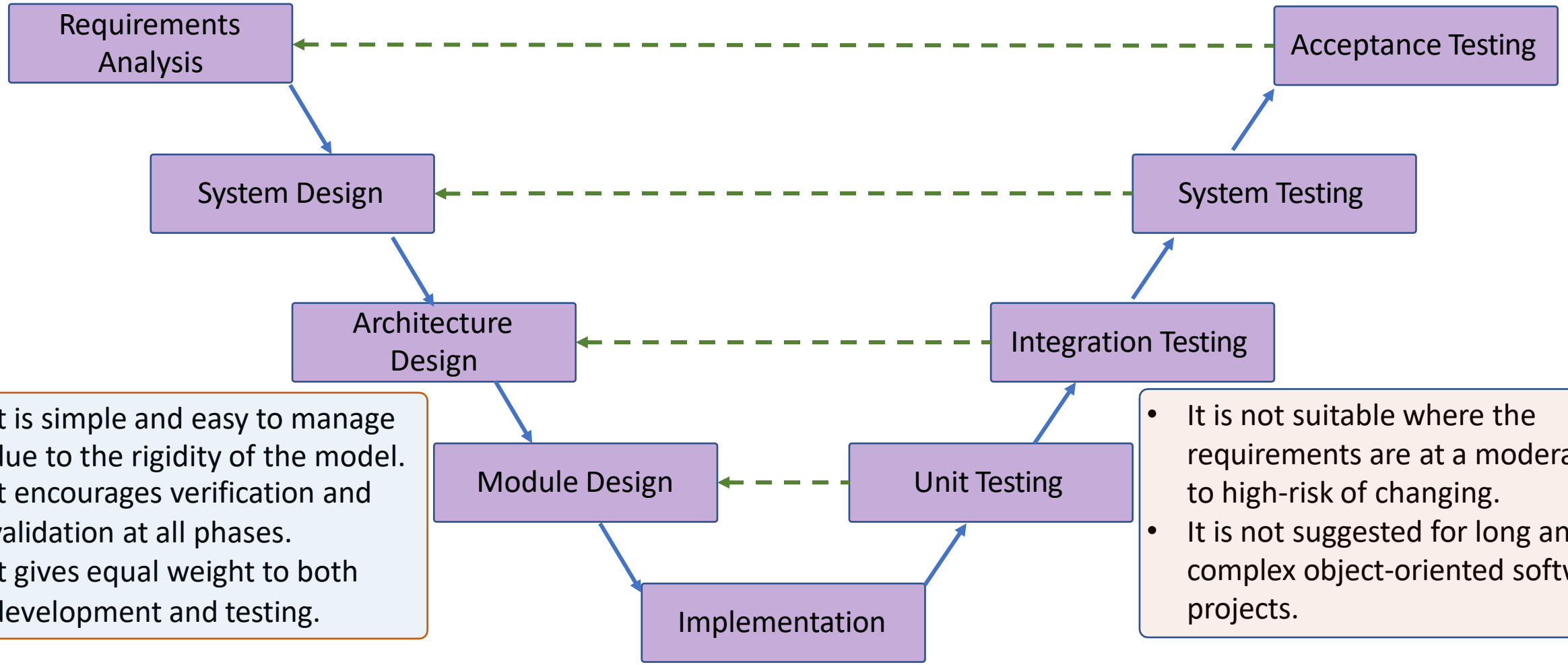
- The first published software development process
- Derived from engineering processes used in large military system engineering

- Plan and schedule all process activities before starting software development, thus it is a plan-driven process
- In principle, the following phases should not start until previous phases have been completed.

Reading (The Waterfall Model): *Royce, W.W., 1987, March. Managing the development of large software systems: concepts and techniques. In Proceedings of the 9th international conference on Software Engineering (pp. 328-338).*

L2

The V-Model



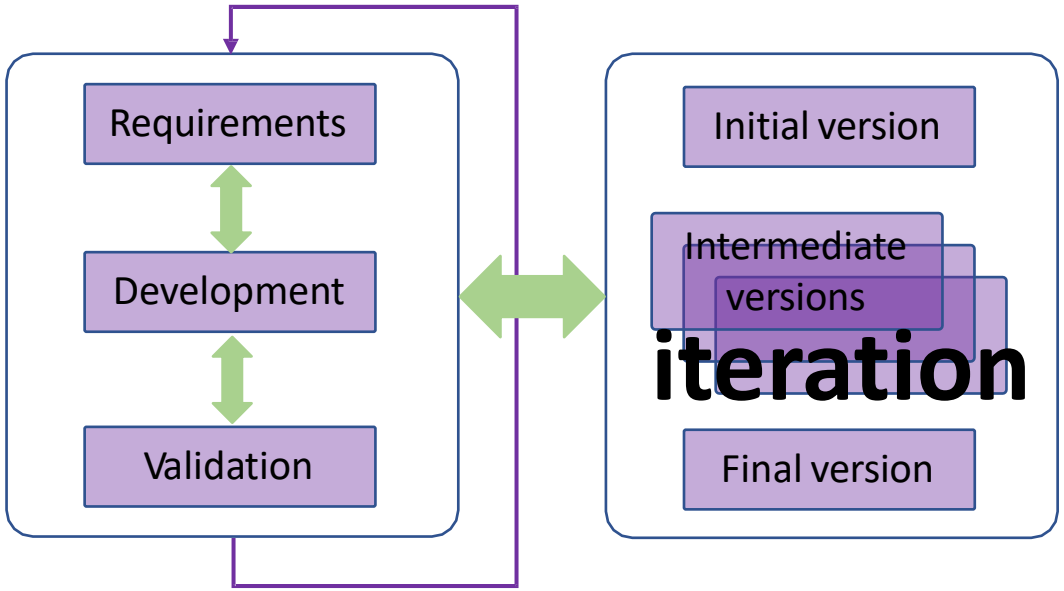
- It is simple and easy to manage due to the rigidity of the model.
- It encourages verification and validation at all phases.
- It gives equal weight to both development and testing.

- It is not suitable where the requirements are at a moderate to high-risk of changing.
- It is not suggested for long and complex object-oriented software projects.

Reading (The V-Model): *Forsberg, K. and Mooz, H., 1991, October. The relationship of system engineering to the project cycle. In INCOSE International Symposium (Vol. 1, No. 1, pp. 57-65).*

L2

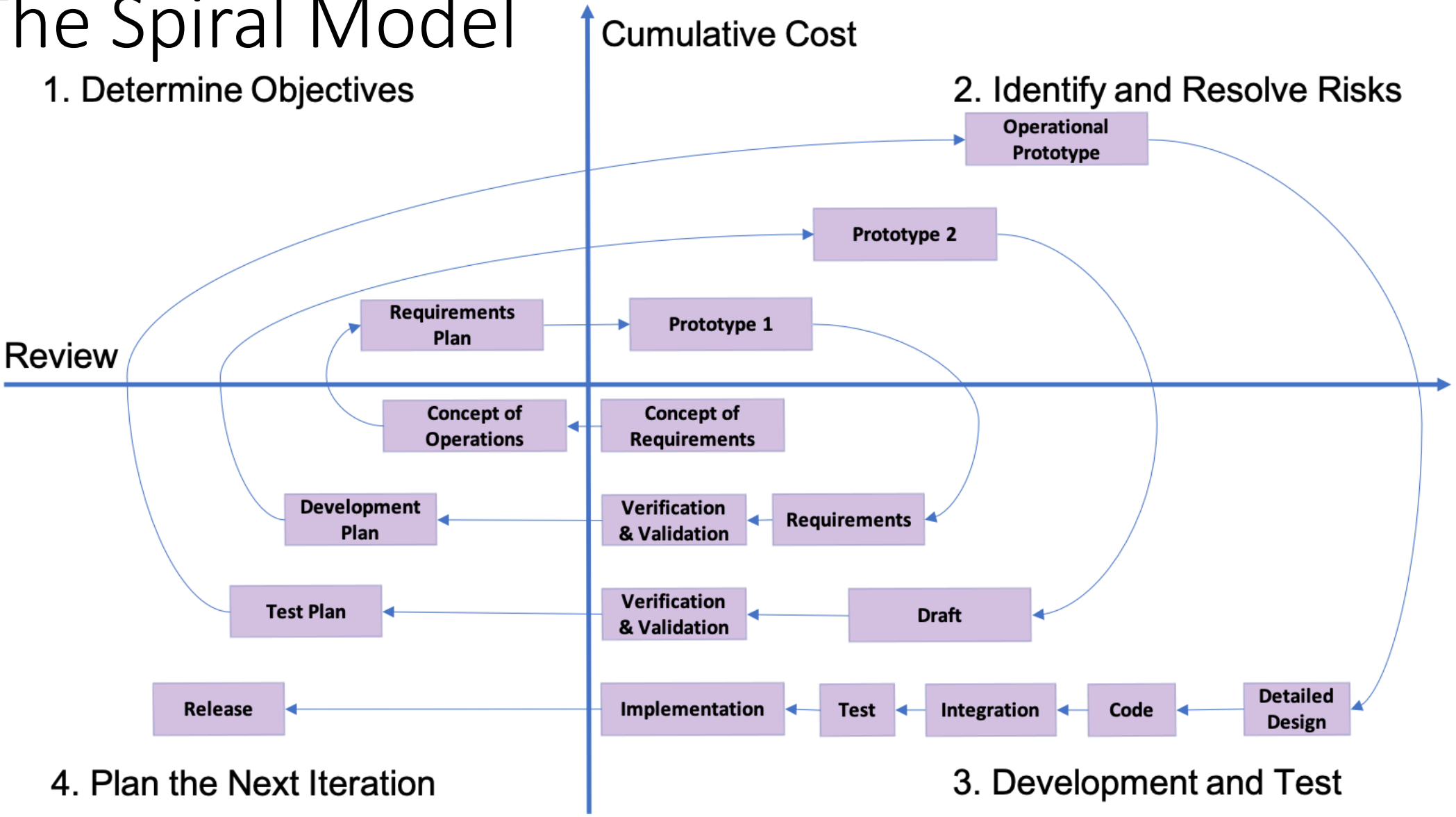
Incremental Development



- Begins with a simple implementation of a part of the software system.
- With each increment the product evolves with enhancements being added every time until the final version is completed.
- Code is written and tested in smaller pieces, thus reduces risks associated with the process.
- It allows changes to be included easily along the development process.

L2

The Spiral Model



Reading (The Spiral Model): *Boehm, B.W., 1988. A spiral model of software development and enhancement. Computer, 21(5), pp.61-72.*

L3

XP Summary

Values:

1. Communication
2. Simplicity
3. Courage
4. Feedback
5. Respect

Practices:

1. Energized Work
2. Pair Programming
3. Stories
4. Weekly Cycle
5. Quarterly Cycle
6. Slack
7. Ten-Minute Build
8. Continuous Integration
9. Test-First Programming
10. Incremental Design
11. Sit Together
12. Whole Team
13. Informative Workspace

Roles:

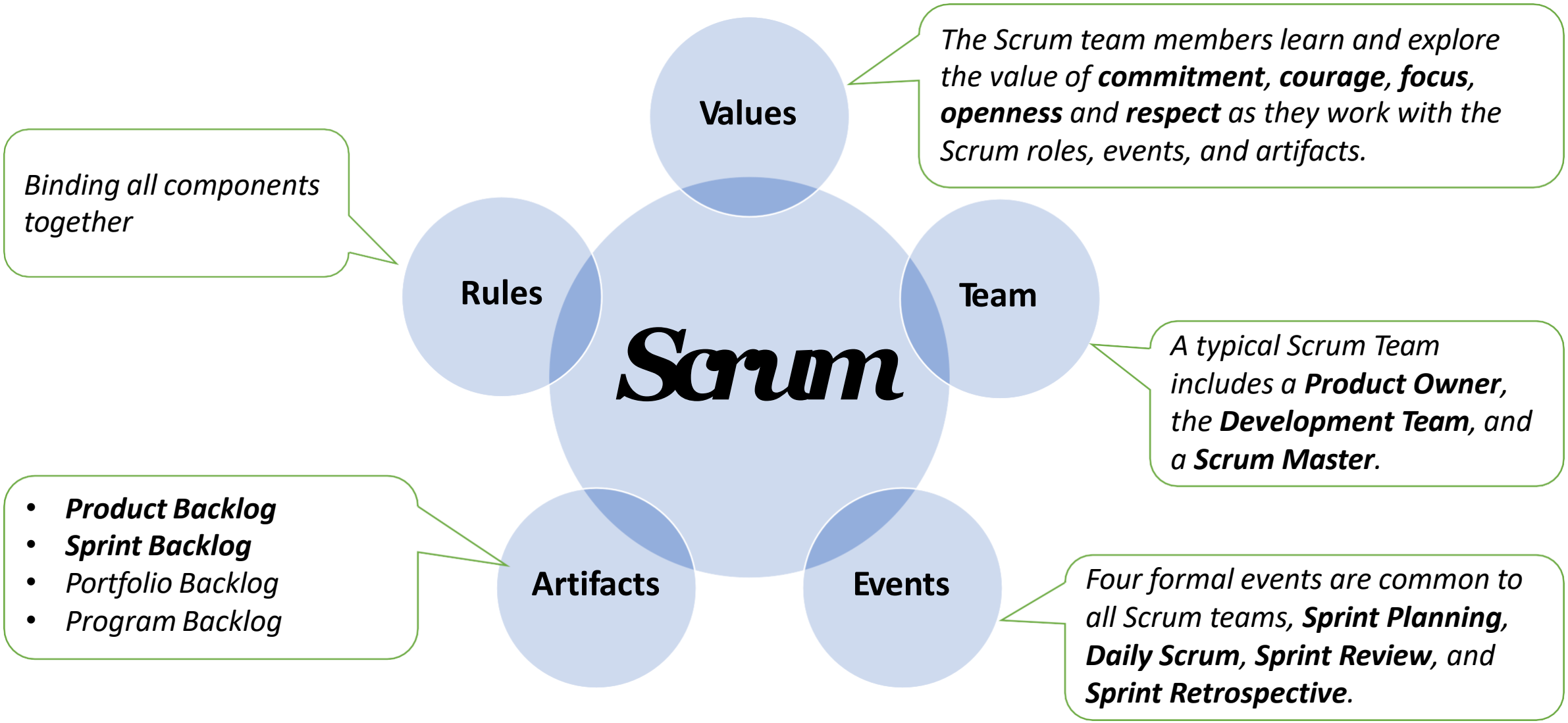
1. Tester
2. Project Manager
3. Product Manager
4. User
5. Programmer
6. Tracker
7. Coach
8. Interaction Designer
9. Architect
10. Executive
11. Technical Writer
12. Human Resource

Principles:

1. Humanity
2. Economics
3. Mutual Benefit
4. Self-Similarity
5. Improvement
6. Diversity
7. Reflection
8. Flow
9. Opportunity
10. Redundancy
11. Failure
12. Quality
13. Baby Steps
14. Accepted Responsibility

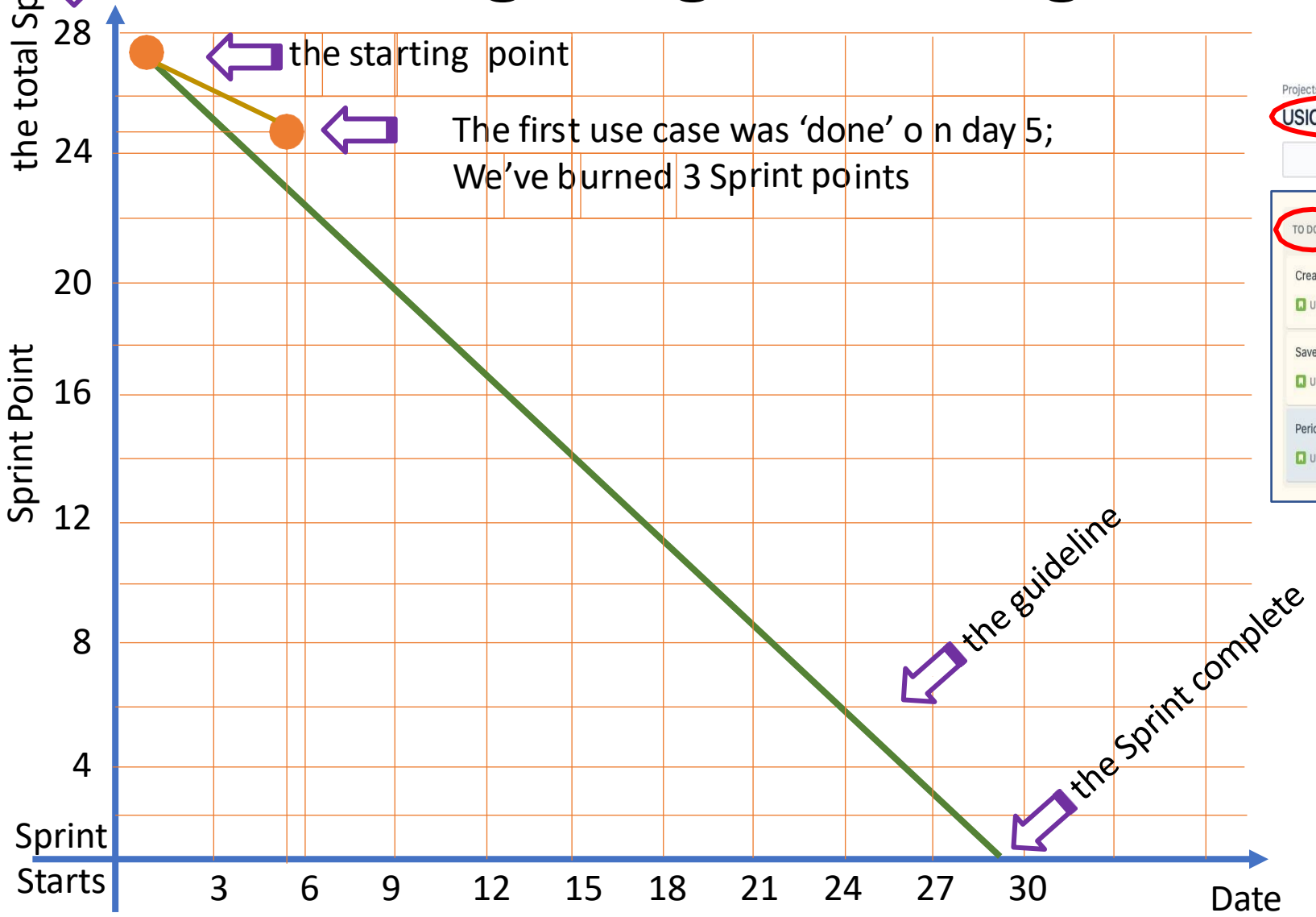
L3

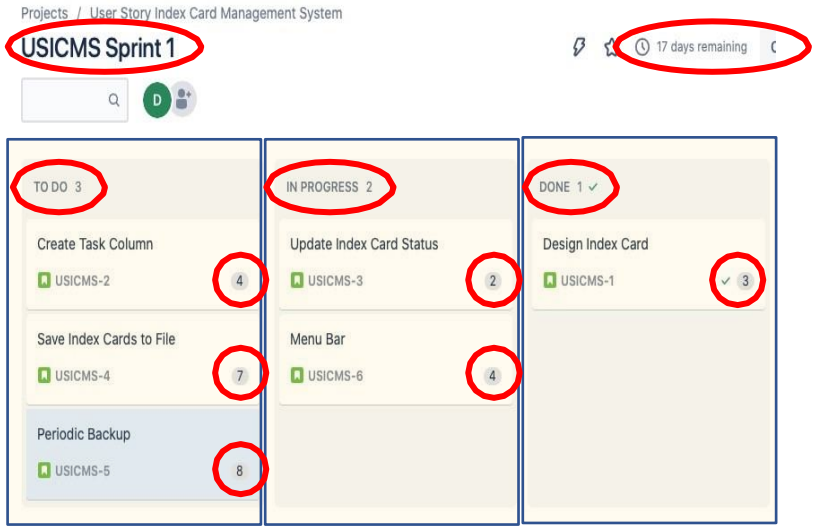
The Components of Scrum Framework



L3

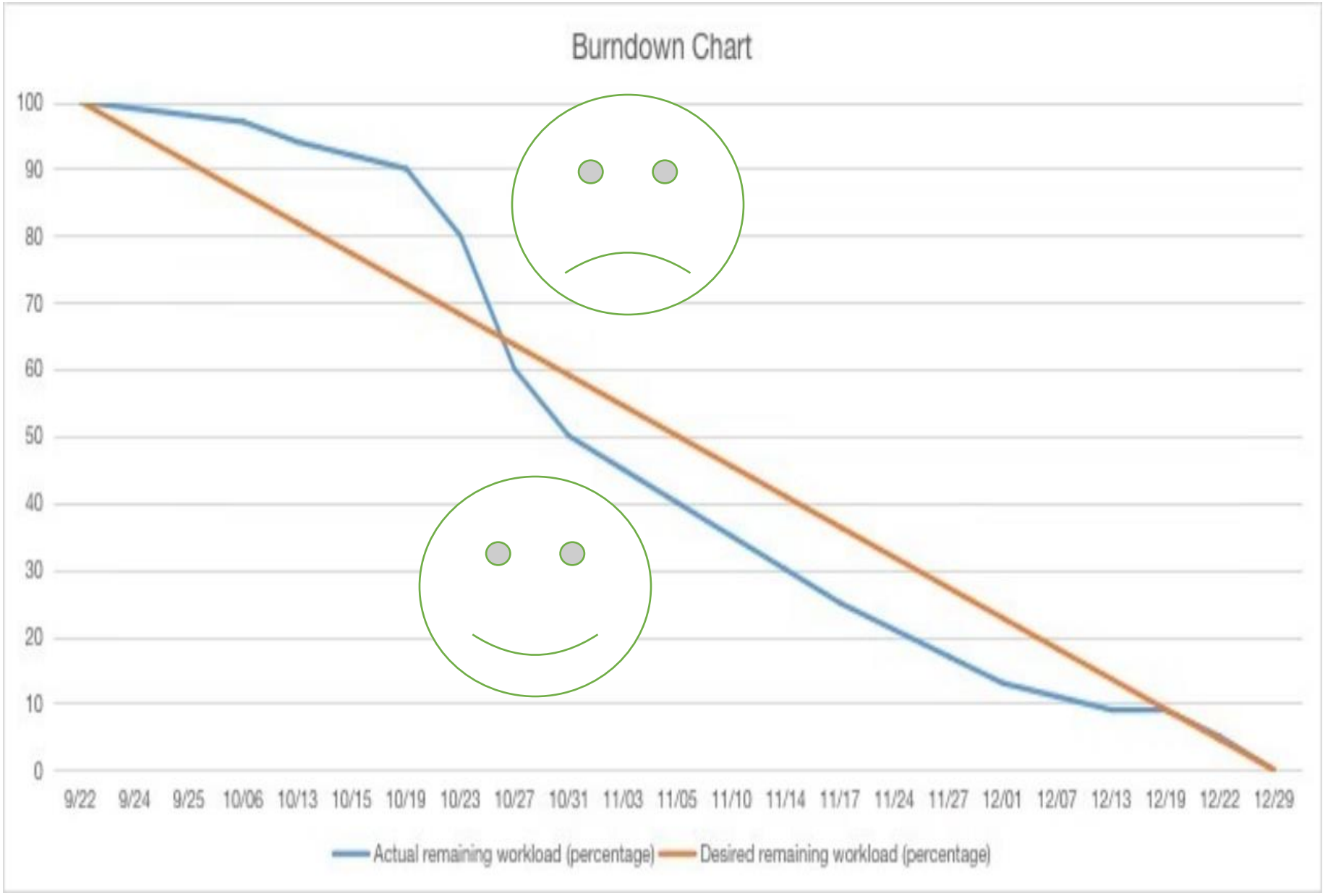
Monitoring Progress using a Burndown Chart





- The first iteration
- 4 Weeks
- 17 Days remaining
- 6 User Stories
- 28 Sprint points

L3


Burndown Chart
100
90
80
70
60
50
40
30
20
10
0
9/22
9/24
9/25
10/06
10/13
10/15
10/19
10/23
10/27
10/31
11/03
11/05
11/10
11/14
11/17
11/24
11/27
12/01
12/07
12/13
12/19
12/22
12/29
Actual remaining workload (percentage)
Desired remaining workload (percentage)

L3

Ideally, the actual effort would line up with the ideal effort. In reality, some work will be marked **below the ideal effort** line (indicating that the team **is ahead of schedule**), and other work will be marked **above the line** (indicating that the team is **behind schedule**).

L3

Practical Issues with Agile Methods

- Agile methods may not be suitable for embedded systems engineering or the development of large and complex systems
  1. The informality of agile is incompatible with the legal approach that is commonly used in large enterprises (Contractual Issues)

  2. Agile are commonly used for new software systems development, rather than for software maintenance

  3. Agile methods are designed for small, co-located teams. For large project with multiple geographically distributed teams, the management and coordination complexity increase significantly, thus the effectiveness of the Agile methods becomes questionable

L4

What is a Design Pattern?

"Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice."

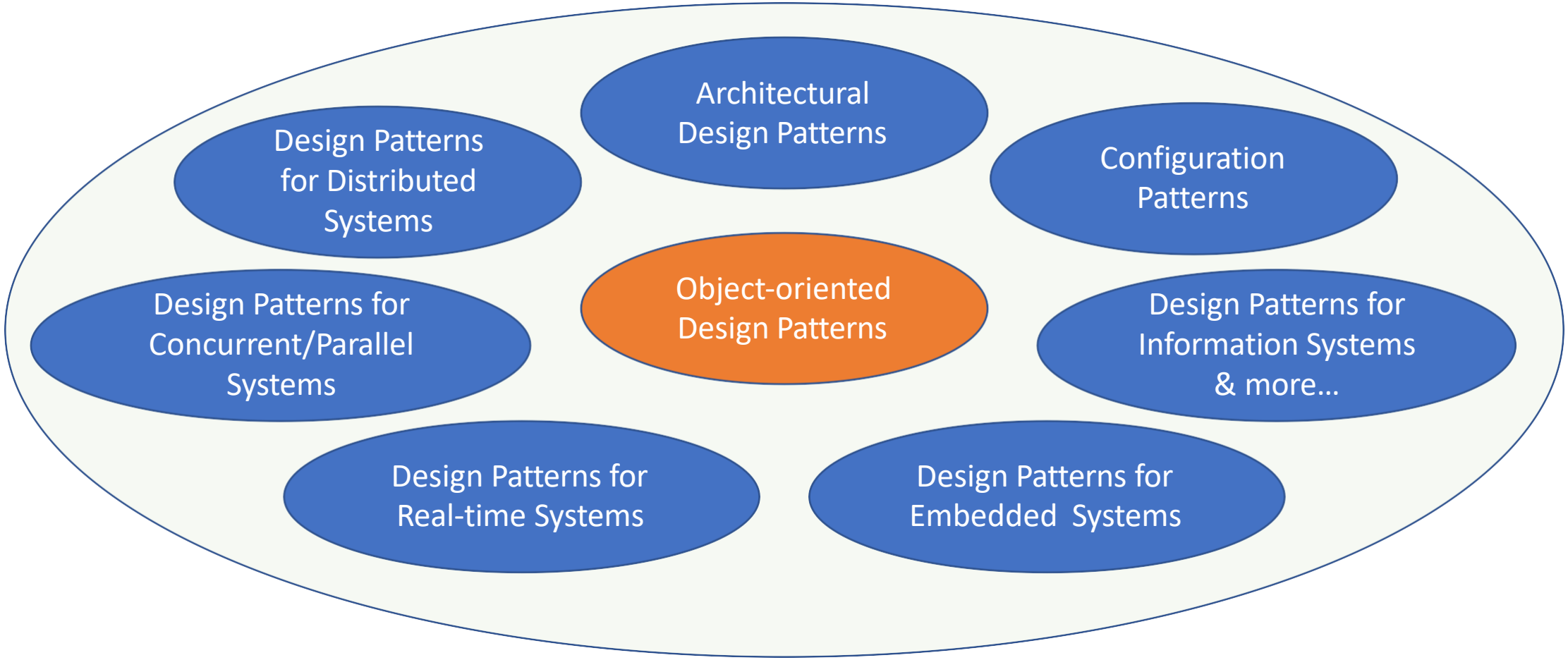
-- Alexander, C., 1977. A pattern language: towns, buildings, construction. Oxford university press.

"Design patterns make it easier to reuse successful designs and architectures."

-- Gamma, E., 1995. Design patterns: elements of reusable object-oriented software. Pearson Education.

L4

Design Patterns in Software Engineering



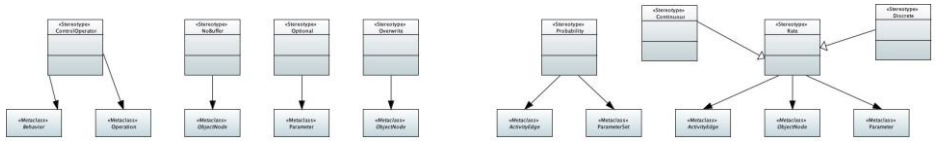
L4

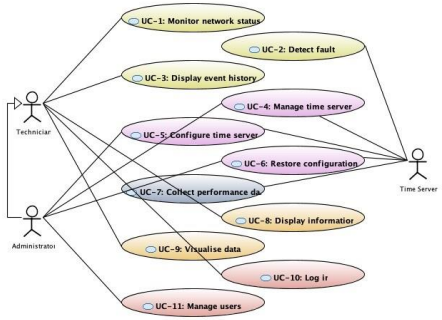
How to Select a Design Pattern?

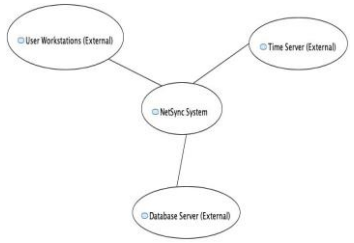
- Consider how design patterns solve a design problem.
  - How to identify appropriate objects,
  - How to determine the granularity of an object,
  - How to specify object interfaces,
  - How to specify object implementations,
  - How to balance between class inheritance and interface inheritance,
  - Consider programming to an interface, not an implementation.
- Look up the intents of the design patterns to find a match.
- Know the inter-relationships between design patterns.
- Think about how to make your design reusable.

L5

UML Diagrams



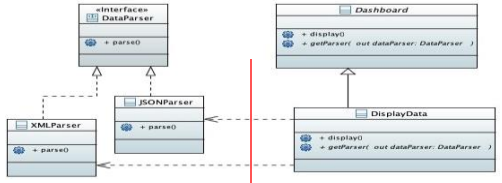




UML 2.5 Diagram

Structure

Behavior



Class

Component

Sequence

Activity

Object

Deployment

Communication

State Machine

Composite Structure

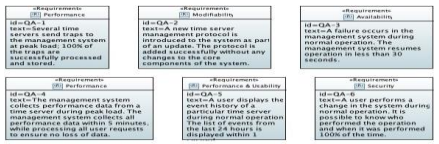
Package

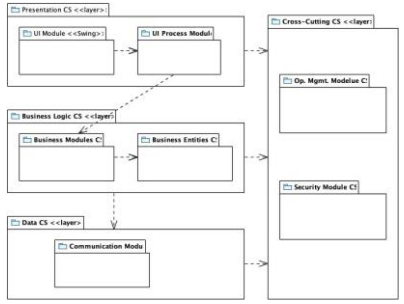
Timing

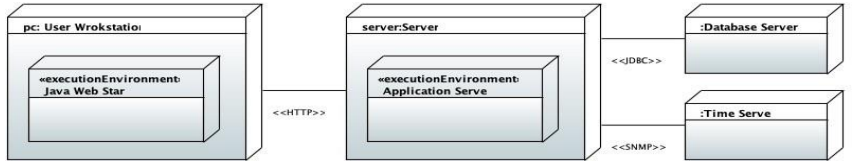
Use Case

Profile

Interaction Overview







UML Diagrams: *Object Management Group. OMG Unified Modeling Language (UML), Superstructure. Technical Report Version 2.5.1, OMG, 2017.*

L5

Elements of Activity Diagram

"An Activity is a kind of Behavior that is specified as a graph of nodes interconnected by edges. The flow of execution is modeled as ActivityNodes connected by ActivityEdges"

--OMG, O., 2017. OMG Unified Modeling Language (OMG UML) Version 2.5. 1.

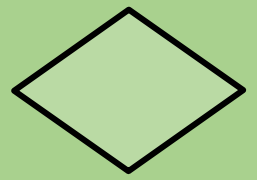
Elements of activity diagram

Action/Executable Node: *carry out the desired behavior.*

Object Node: *hold object tokens.*

Control Nodes: *managing the flow of actions.*

join/fork



decision/merge

initial node

flow final

final node

ActivityNodes

Activity edge

Activity edge for interruptible regions

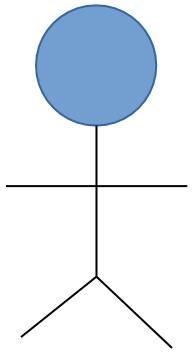
EdgeNodes

NOTE: There are many other advanced notations defined but not presented here.

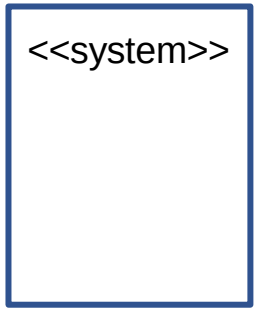
L5

Use Case Modeling Using UML

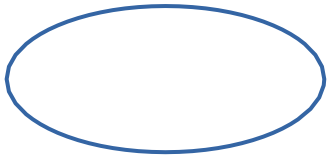
The elements of Use Case diagram:



Actor



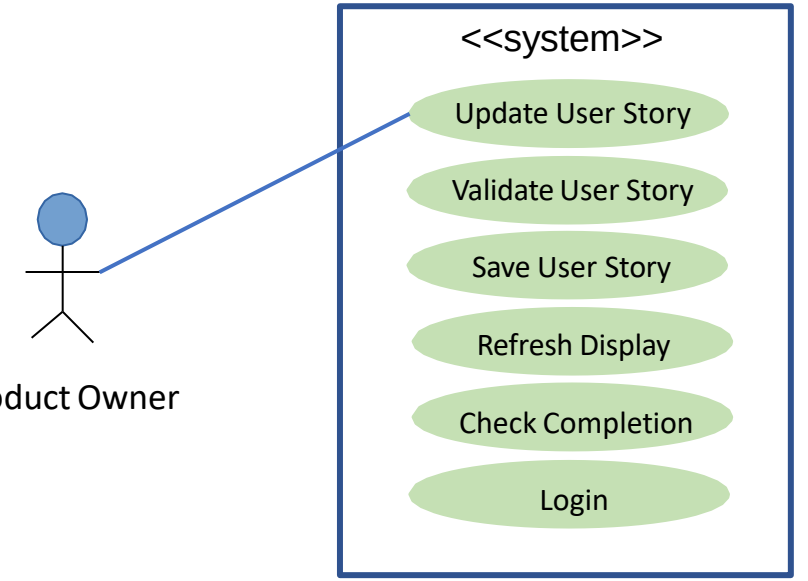
System boundary (<<system>>, <<subsystem>> or <<subject>>)



Use Case

Connection (<<Association>>)

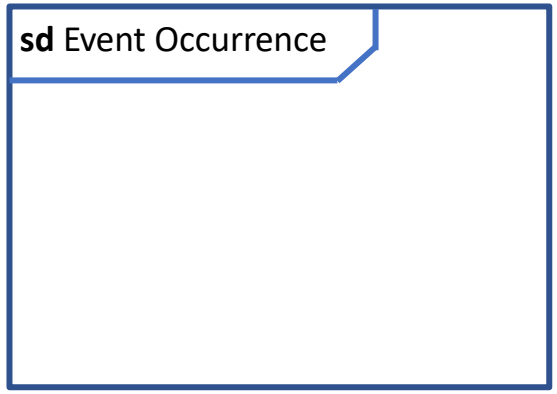
User Story Index Card Management System: Update User Story	
Actors	Product Owner
Description	A product owner may modify an existing user story in the system. By modifying a user story, the newly updated information must be validated, for example, Sprint Point must be a numerical value. At the same time, all mandatory fields must be completed, and then saved on persistent storage. The updated user story must be visible to the user immediately.
Data	An existing user story in the system
Stimulus	User command issued by product owner
Response	Confirmation that the user story has been validated, updated and saved
Comments	The person who wants to make changes to the user story must first login as a Product Owner role; the project file must already be loaded in the system.



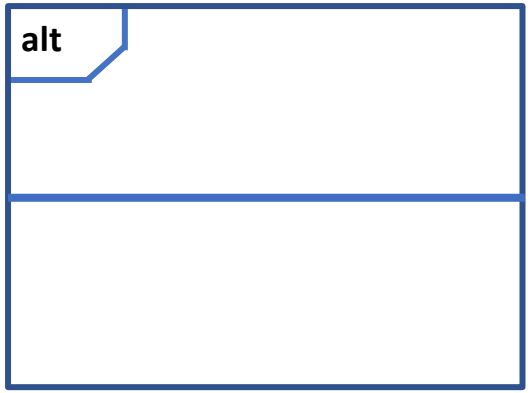
L5

Sequence Diagrams

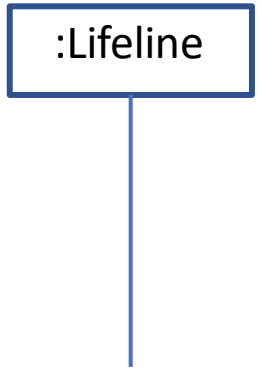
Graphic Nodes



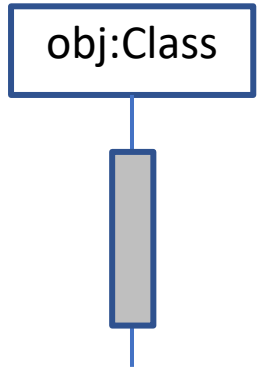
Frame for Interaction



Alternative Combined Fragment



Interaction Lifeline



Execution Specification

Destruction Occurrence Specification

Graphic Paths

Asynchronous Message

Synchronous Message

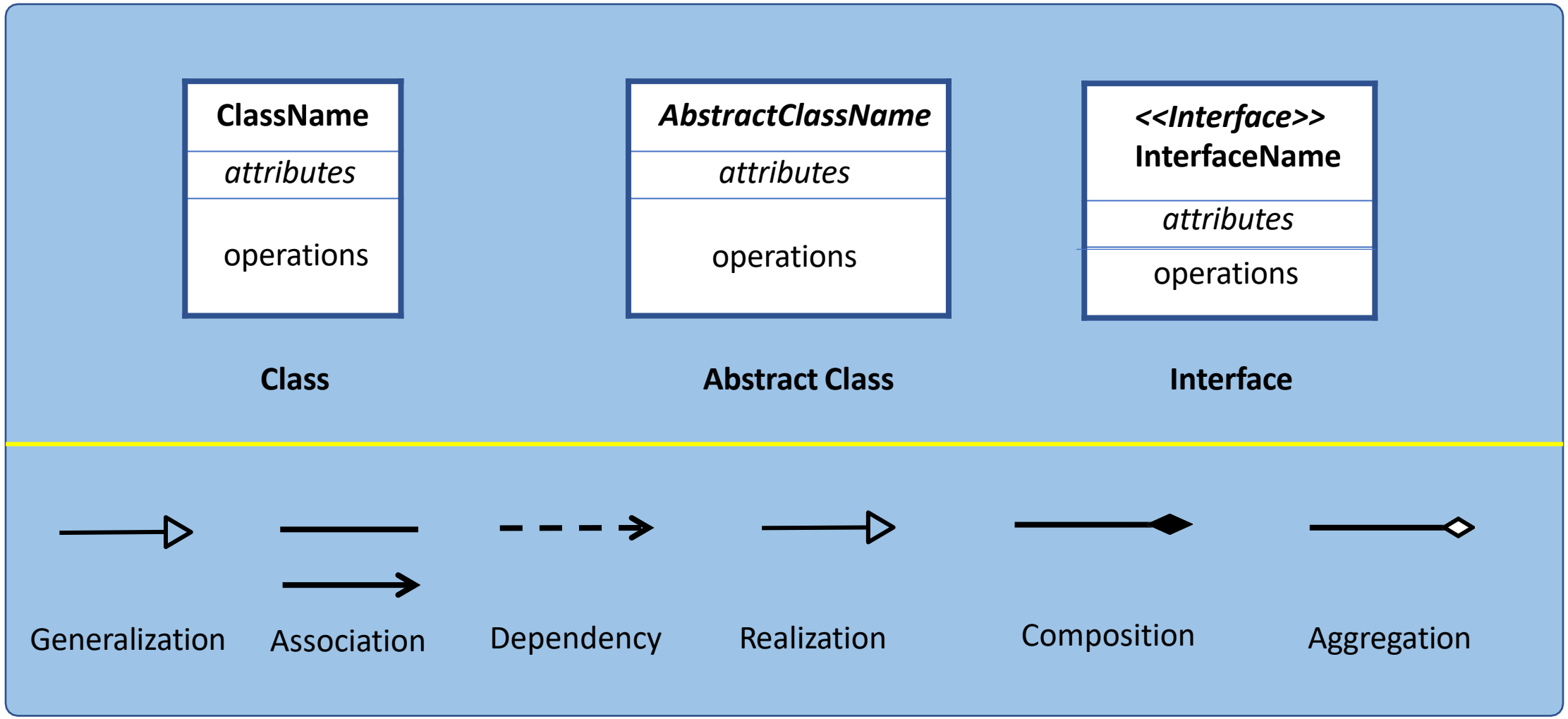
Response Message

Found Message

Lost Message

L5

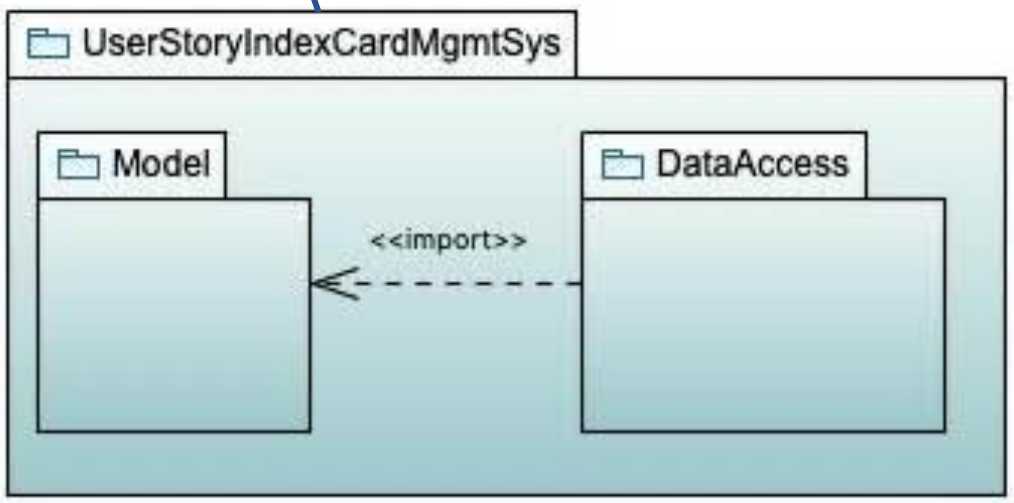
Class Diagram



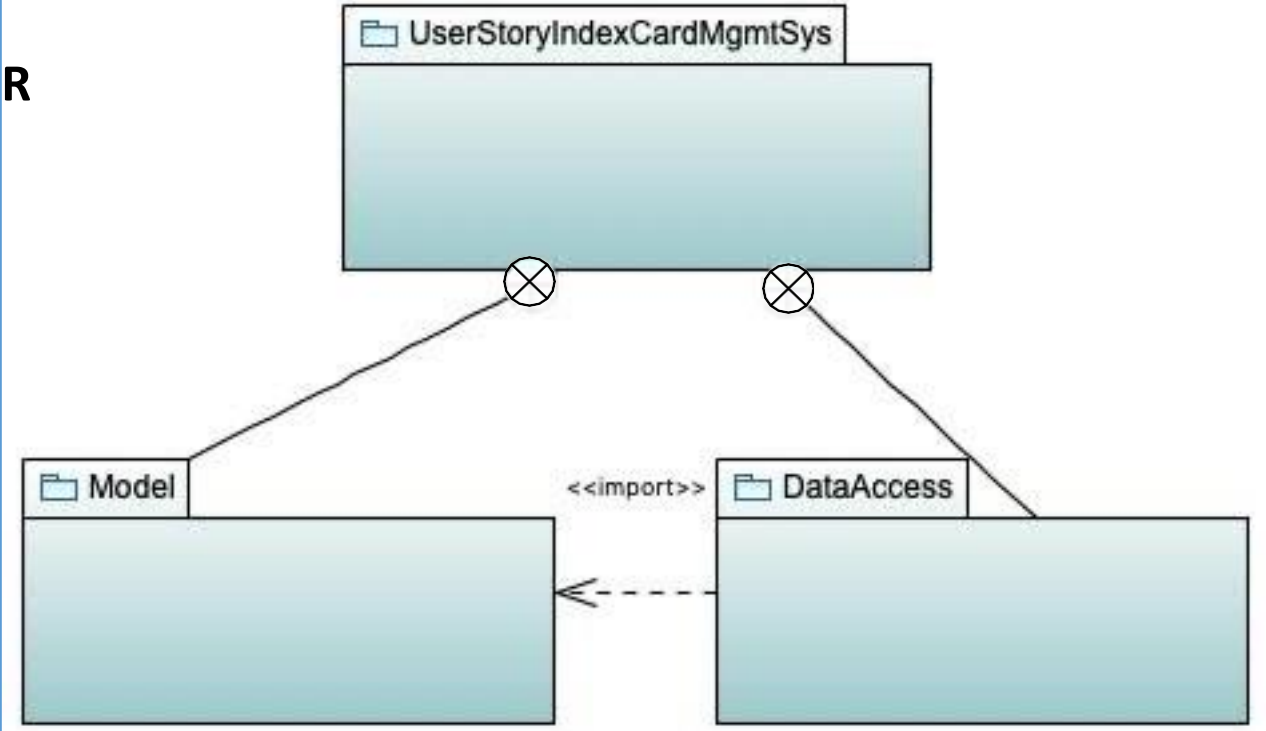
L5

Package Diagram Examples

A ***Package*** is a namespace for its members, which comprise those elements that are owned or contained and those imported.



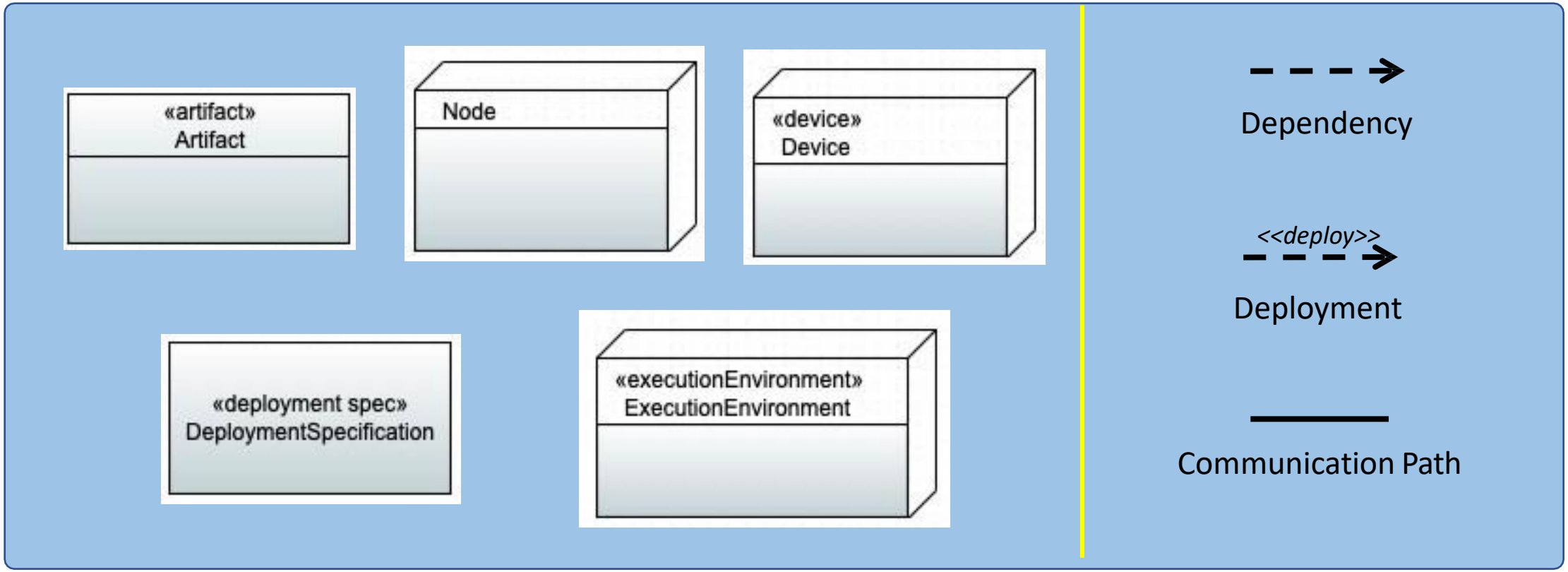
OR



L5

Deployment Diagrams

- Deployment diagram show the relationships between logical and/or physical elements of systems and assets assigned to them.



L5

Exercise:

1. Develop a sequence diagram showing the interactions involved when a student registers for a course at a university. Courses may have limited enrolment, so the registration process must include checks that places are available. Assume the student accesses an electronic course catalogue to learn about available courses.

2. According to the following code fragments, draw a class diagram:

```
package cn.fzu.miec.doc;
public class Report {
  private String title;

  public Report(String title) {
    this.title = title;
  }

  public String getTitle() {
    return title;
  }
}
```

```
package cn.fzu.miec.device;
import cn.fzu.miec.doc;
public class Printer {
    public void print(Report report) {
    System.out.println(report.getTitle());
  }
}
```

```
package cn.fzu.miec.device;
import cn.fzu.miec.doc;
public class HPPrinter extends Printer {

}
```

L6

Summary: Requirements Elicitation and Analysis

Depict the *big picture* of the system and the application domain.

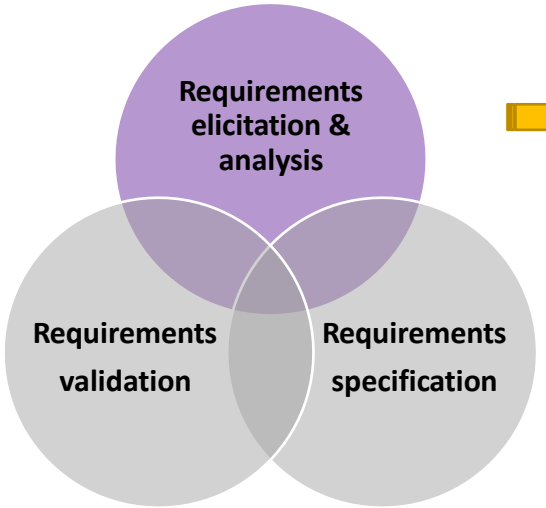
Discover the implicit requirements and the *actual* workflow

Describe the user and system expectations, & workflows, etc. Brainstorming+Innovations

Interview

Ethnography

Stories & Scenarios



Requirements discovery & understanding

Requirements classification & organization

Requirements prioritization & negotiation

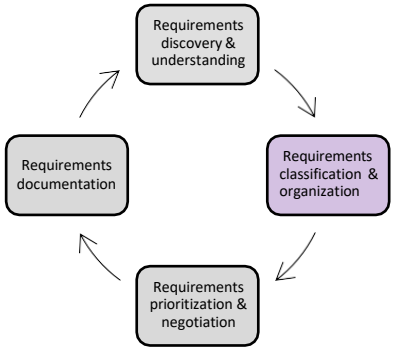
Requirements documentation

Pedigree Attribute eLicitation Method (PALM)

Quality Attribute Workshop (QWA)

L6

User Requirements vs. System Requirements



- User Requirements
  - Often written in natural langue with diagrams
  - High-level, abstract statement of a service that a system should provide or a constraint on a system
  - Usually written for non-technical people
- System Requirements
  - Detailed description of software system's services and operational constraints
  - Define exactly what is to be implemented
  - It may be part of the contract between system investors and software developers
  - Usually written for technical people

- Client managers
- Contractor managers
- System end-users
- Client engineers
- …

- System architects
- Software developers
- Product Managers
- Project managers
- …

Reading (The inconsistent use of the term 'requirement'): Davis, A.M., 1993. *Software requirements: objects, functions, and states*. Prentice-Hall, Inc..

L6

Functional Requirements and Non-functional Requirements

- Functional Requirements
  - The statements of services that the system should provide
  - How the system should react to particular inputs
  - How the system should behave in a particular situation
  - *Example 1:* The User Story Index Card Management system shall provide an export function allowing users to save all index cards in a PDF file
  - *Example 2:* Data scientists and analysts shall be able to perform ad-hoc data analysis through SQL-like queries to find specific data patterns and correlations to improve infrastructure capacity planning
- Non-functional Requirements
  - The constraints on the services, e.g., timing constraints, resource constraints, constraints imposed by standards
  - *Example 1:* When the system is under peak load, no emergency messages received from the mobile clients shall be dropped, and the cached messages shall be processed as soon as possible and be multi-casted to preselected receivers with no further delay.
  - *Example 2:* The system shall collect up to 15,000 even/sec from approximately 300 web servers.

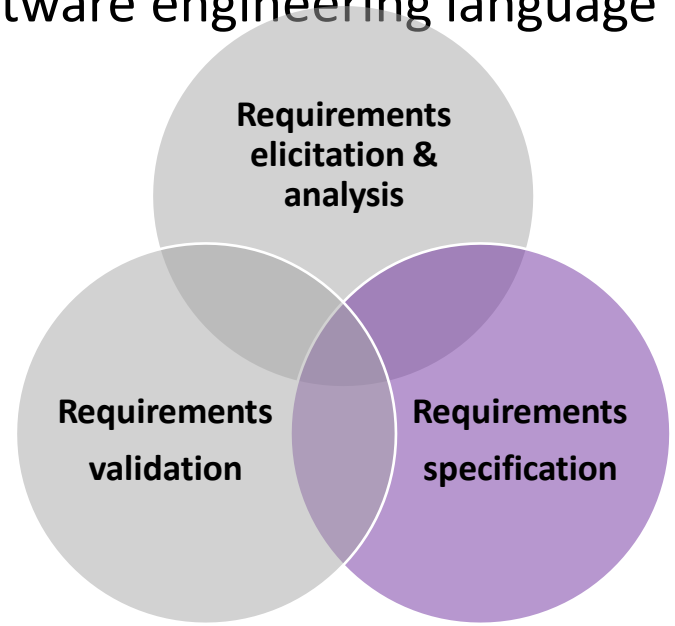
PDF (Portable Document Format): a cross-platform file format developed by Adobe.

SQL (Structured Query Language, An Interesting fact): Chamberlin, D. D. (Donald Dean). (2001). Oral history interview with Donald D. Chamberlin. Charles Babbage Institute. Retrieved from the University of Minnesota Digital Conservancy, https://hdl.handle.net/11299/107215.

L6

Requirements Specification

- Structured natural language
  - Requirements are written in natural language in standard form or template
  - Use language consistently to distinguish between mandatory and desirable requirements.
  - Do NOT assume that readers understand technical or software engineering language
- Graphical notations
  - Unified Modeling Language (UML) diagrams
    - Use case diagrams
    - Sequence diagrams
- Mathematical specification
  - Using notations based on mathematical concepts
    - E.g., Finite-state machines or Formal Methods



L7

What is Software System Architecture?

"The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both."

-- Bass, L. Clements, P. and Kazman, R. Software architecture in practice, 3ed, 2013.

"The architecture of a system is the set of fundamental concepts or properties of the system in its environment, embodied in its elements, relationships, and the principles of its design and evolution."

-- Rozanski, N. and Woods, E., 2012. Software systems architecture: working with stakeholders using viewpoints and perspectives. Addison-Wesley.

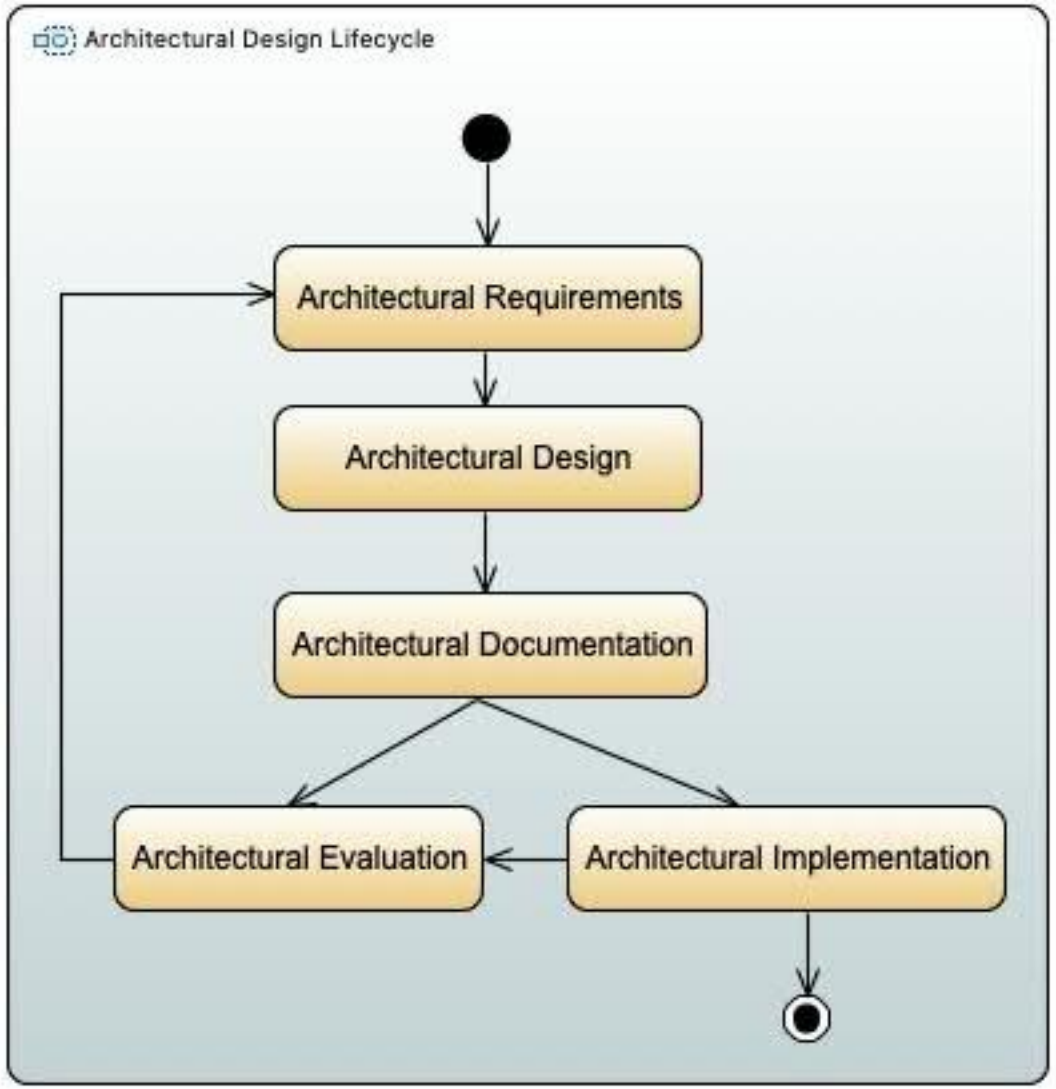
"Architectural design is concerned with understanding how a software system should be organized and designing the overall structure of that system."

-- Sommerville, I., 2016. Software engineering., 10th Edition. Pearson Education.

L7

Architecture Design Lifecycle

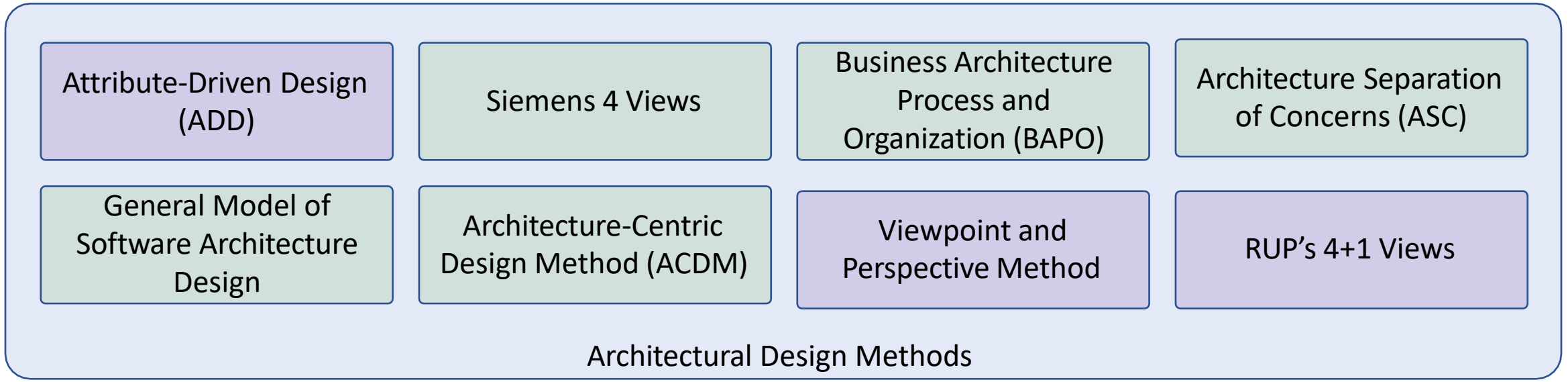
- Among all requirements there are a few that have special importance for the architecture, known as Architecturally Significant Requirements (ASRs):
  - E.g., the most important functionality of the system, the constraints and the quality attributes such as high performance and high availability, etc.
- Design is a translation, from requirements to solutions, which can be structures composed of code, frameworks, and components.
- Preliminary documentation (*sketches*) of the structures should be created as part of architectura design.
- If the project under development is non-trivial, then the design should be evaluated to ensure that the decisions made are appropriate to address the ASRs.
- An architect's responsibility during implementation is to ensure *conformance* of the code to the design.



L7

Principled Methods

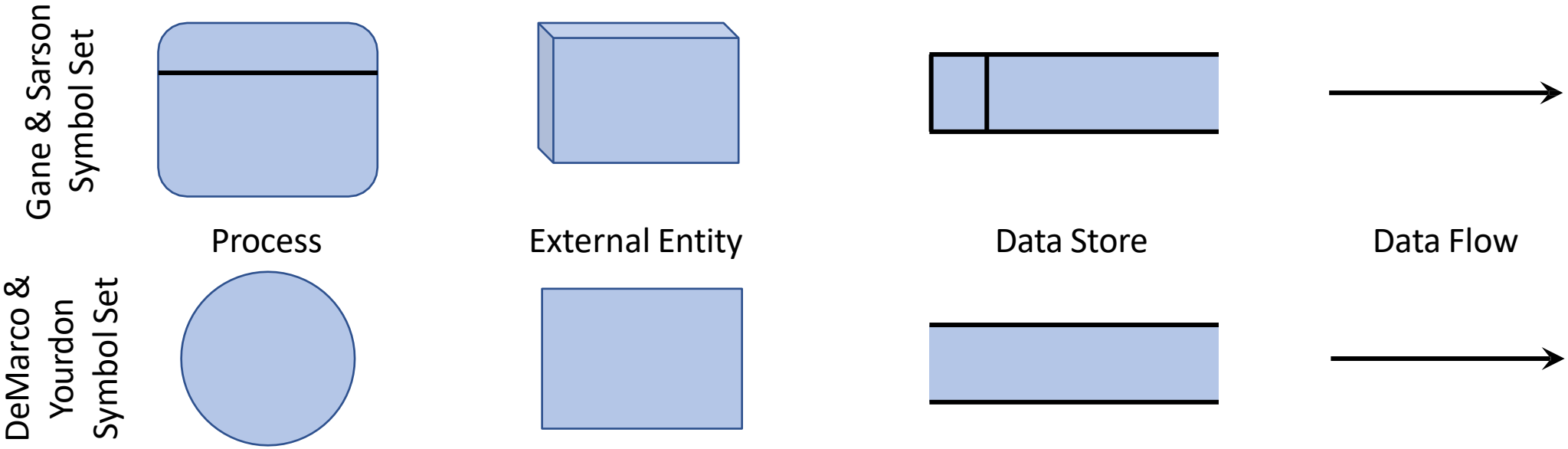
- How do we actually perform a design?
- Performing design to *ensure* that the business requirements are satisfied requires a principled method.
- A method provides guidance.



L8

Structured Analysis – Process Modeling

- System analysts use Data Flow Diagram (DFD) to show how data moves through an information system.
- Data Flow Diagrams show the processes that change or transform data.
- A set of data flow diagrams provides a logical model that shows what system does, but not how it does it, i.e., data and process modelling.



L8

Conceptual Data Modelling

"A detailed model that captures the overall structure of organizational data and is independent of any database management system or other implementation considerations."

--Valacich, J.S., George, J.F. and Valacich, J.S., 2017. Modern systems analysis and design. Boston: Pearson.

- Data modelling develops the definition, structure, and relationships within the data.
- A data model explains what the organization does and the rules that govern the work performed in the organization.
- A data model does not care how or when data are processed or used.
- Conceptual data modelling is often performed with other requirements analysis and systems analysis activities, such as process modeling and logic modelling.
  - The works and activities are commonly coordinated and shared through project dictionary or repository maintained by a common Computer-Aided Software Engineering (CASE) software tool.
- The most common technique used for data modelling is Entity-Relationship (E-R) diagramming.

L8

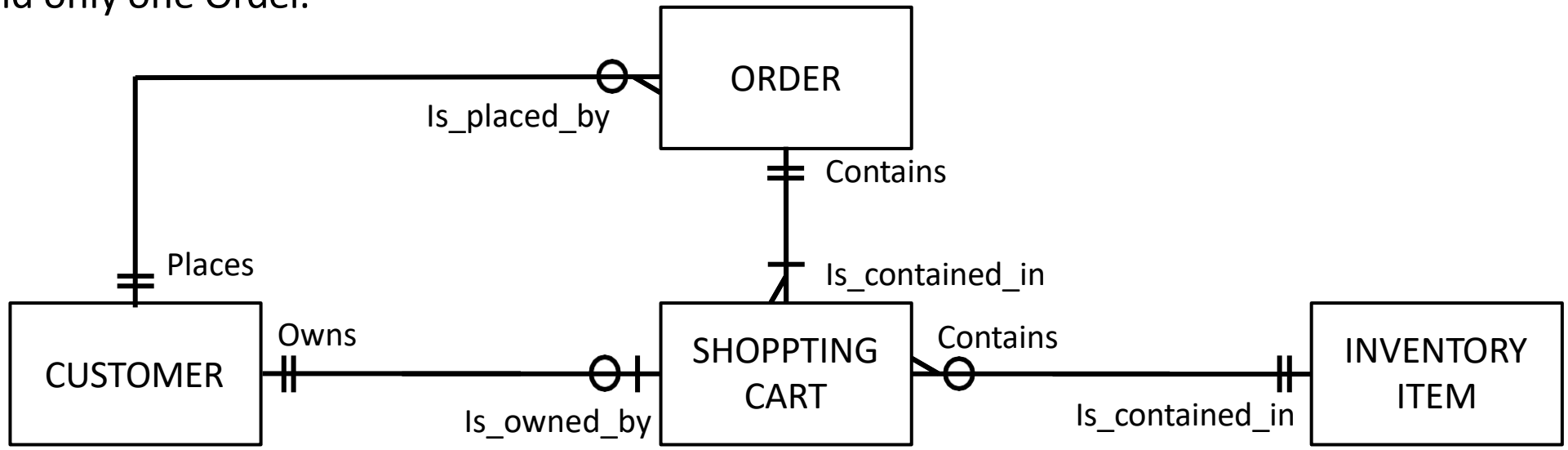
E-R Model – Entities

- An entity is a person, place, object, event, or concept in the user environment.
  - Person: STUDENTS, LECTURER, CLEARK, etc.
  - Place: STORE, WAREHOUSE, DEPARTMENT, etc.
  - Object: SENSOR, PRODUCT, CAR, etc.
  - Event: SALE, REGISTRATION, PURCHASE, etc.
  - Concept: COURSE, ACCOUNT, STOCK, etc.
- An entity has its own identity that distinguishes it from other entity.
- An entity type (or entity class) is a collection of entities that share common properties or characteristics (similar to the *class* in OOD).
- An entity instance is a single occurrence of an entity type (similar to the *object* in OOD)
  - E.g., there is one LECTURER entity type in university, but there may be hundreds (or thousands) of instances of this type stored in the university's staff management database.

L8

Data Stores and External Entities to E-R Model

1. Each Customer *owns* zero or one Shopping Cart instances; each Shopping Cart instance *is owned by* one and only one Customer.
2. Each Shopping Cart instance *contains* one and only one Inventory item; each Inventory item *is contained in* zero or many Shopping Cart instances.
3. Each Customer *places* zero, or one, or many Orders; each Order *is placed by* one and only one Customer.
4. Each Order *contains* one to many Shopping Cart instances; each Shopping Cart instance *is contained in* one and only one Order.



Reference: Valacich, J.S., George, J.F. and Valacich, J.S., 2017. Modern systems analysis and design. Boston: Pearson.

L8 Cardinalities in Relationships

The number of instances of entity ***B*** that can (or must) be associated with each instance of entity ***A***.

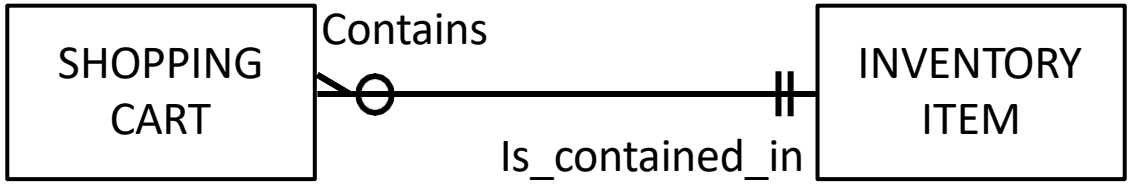
Mandatory Many
(One or many, 1..*)

Optional One
(Zero or one, 0..1)

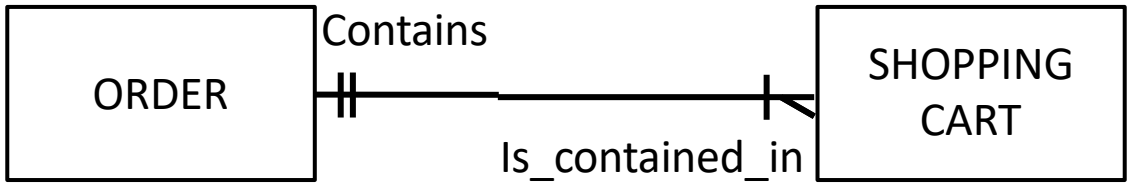
Mandatory One
(One and only one, 1)

Optional Many
(Zero, or one, or many, 0..*)

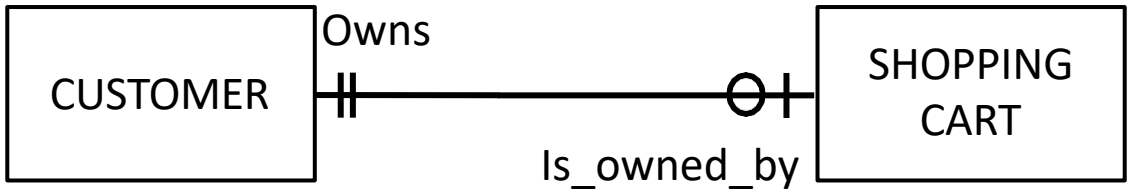
The Crow's Foot Symbols



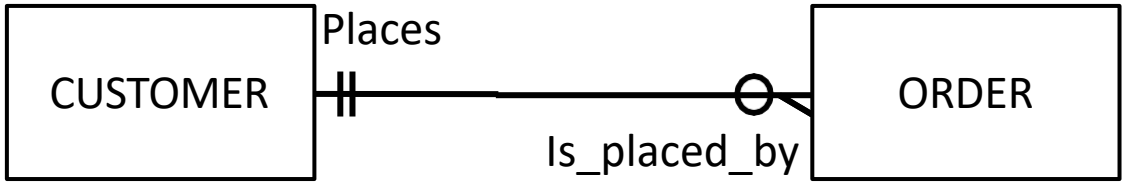
An inventory item can be contained in zero, or one, or many shopping cart instances.



An order must contain one or more shopping cart instances.



A customer can own zero or one shopping cart instance.



A customer can place zero, or one, or many orders

L8

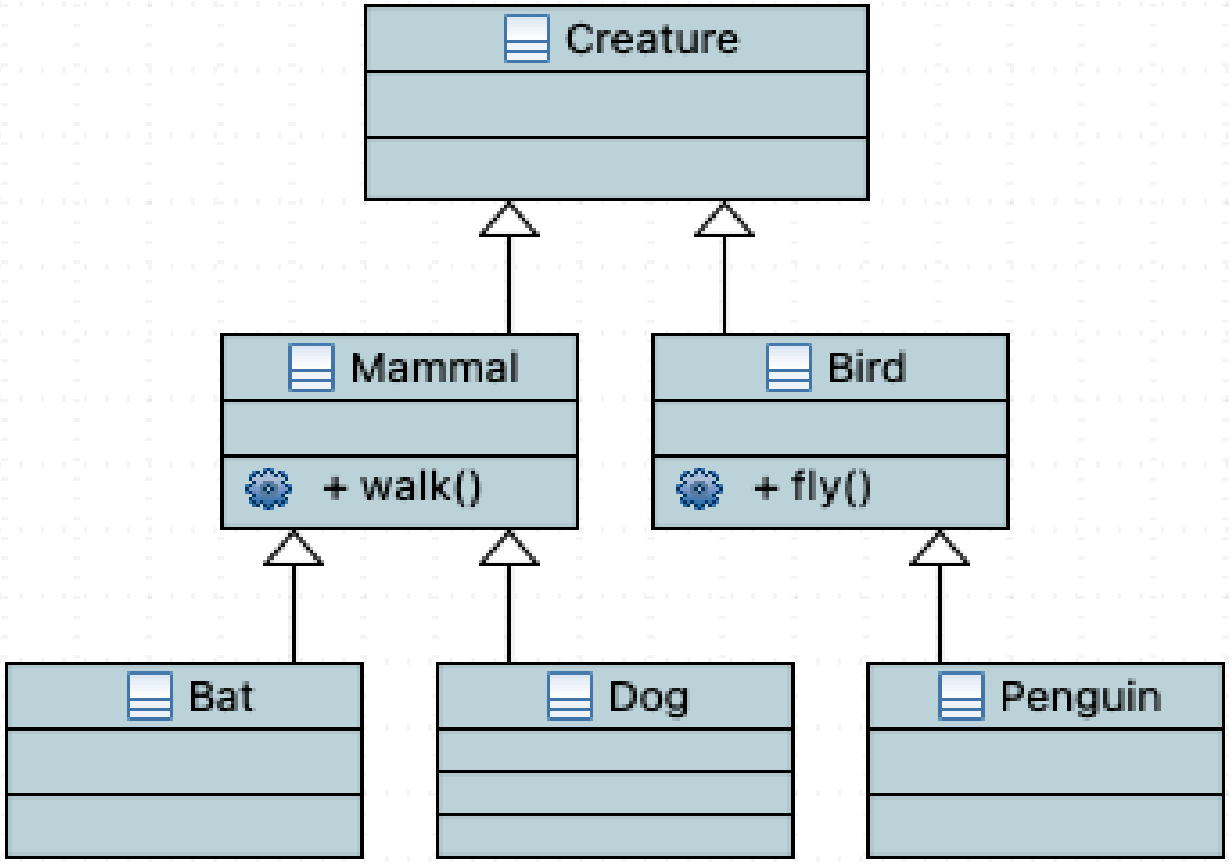
Summary

- The GRASP principles can be used as a tool to help master the basics of OOD and understanding responsibility assignment in object design.
- GRASP follows the idea of responsibility-driven design, thinking about how to assign responsibilities to collaborating objects.
- GRASP and SOLID largely overlap.

Favour composition over inheritance;
Program to Interface, not Implementation.

L8 Practice

In Object-Oriented design, modelling complex relationships between entities is hard. The following class diagram models creatures using inheritance. It is technically correct, but logically incorrect, as a bat is a mammal but can fly, and a penguin is a bird, but cannot fly. Redesign the class diagram using class composition so that it is logically and technically correct.



Testing and Debugging

- Defect testing and debugging are distinct processes
- Defect testing is concerned with confirming the presence of errors
- Debugging is concerned with locating and repairing these errors
- Debugging involves formulating a hypothesis about program behavior then testing these hypotheses to find the system error

Errors, Faults and Failures

1. Errors: these are mistakes made by software developers. They exist in the mind, and can result in one or more faults in the software.

2. Faults: these consist of incorrect material in the source code, and can be the product of one or more errors. Faults can lead to failures during program execution.

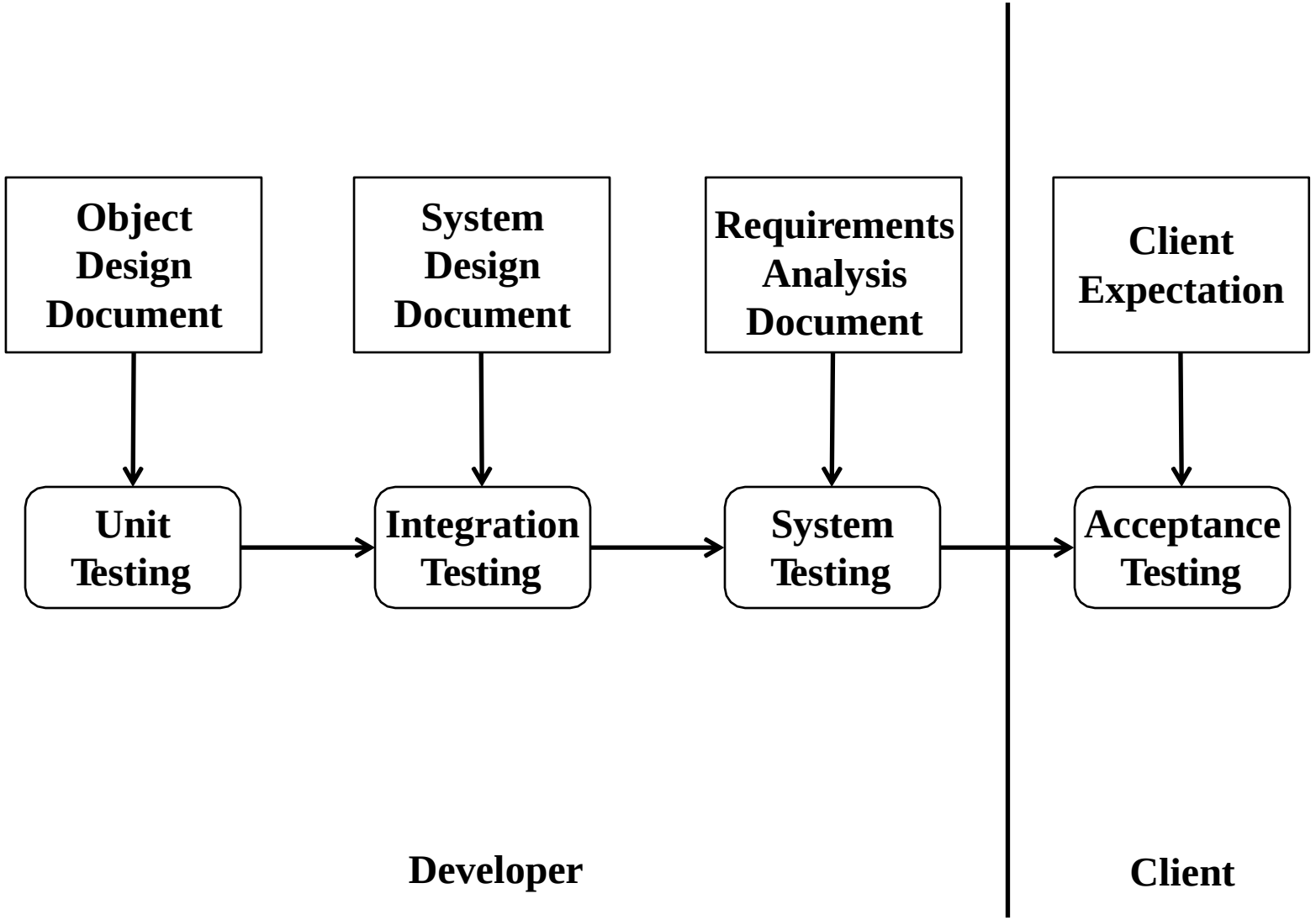
3. Failures: these are symptoms of a fault, and consist of incorrect, or out-of specification behaviour by the software. Faults may remain hidden until a certain set of conditions are met which reveal them as a failure in the software execution.

Testing in the development process

- Software has three key characteristics
  - User requirements that state the user's needs
  - A functional specification stating what the software must do
  - A number of modules that are integrated to form the final system

- These must be verified using the following four test activities

L9


Object Design Document
System Design Document
Requirements Analysis Document
Client Expectation
Unit Testing
Integration Testing
System Testing
Acceptance Testing
Developer
Client

L9

Black and White Box Testing

- Black Box testing is based entirely on the program specification and aims to verify that the program meets the specified requirements
- White box testing uses the implementation of the software to derive the tests. The tests are designed to exercise some aspect of the program codes

Which to choose ???

L10

Principles of Interactive UI Design

- Computers, and interfaces should be functional, easy to use, and intuitive.

- The gulf of execution and the gulf of evaluation:
  - Gulf of Execution: This is the gap between the intention of the user's action and how easily the system will allow them to achieve it.
  - Gulf of evaluation: This is the degree of ease with which a user can perceive and interpret whether, or not, the action they performed was successful.

The Design of Everyday Things: Revised and Expanded Edition -
By D. Norman

L10

Interface Design Errors

Typical Design Errors

- Lack of consistency
- Too much memorization ? No guidance / help
- No context sensitivity
- Poor response
- Unfriendly

L10

User Experience (UX) Design

- User experience (abbreviated as UX) is how a person feels when interfacing with a system. For it to be a success, no matter how good its functionality, users must enjoy using and navigating through it

- The system could be a website, a web application or desktop software and, in modern contexts, is generally denoted by some form of human-computer interaction (HCI).

L10

UX Limitations & Difficulties

- Cannot yet have the one ideal UX design that works for everyone, some compromise has to happen
- Difficult to measure the effectiveness of UX objectively
- Traditional web page assessment metrics such as ‘pages viewed’ cannot be used to assess the effectiveness of a UX design
- UX designers may not necessarily have a technical background and this can lead tensions with the developers over design recommendations

L11

Project Planning Progress

A UML activity diagram that shows important steps needed when planning a project.

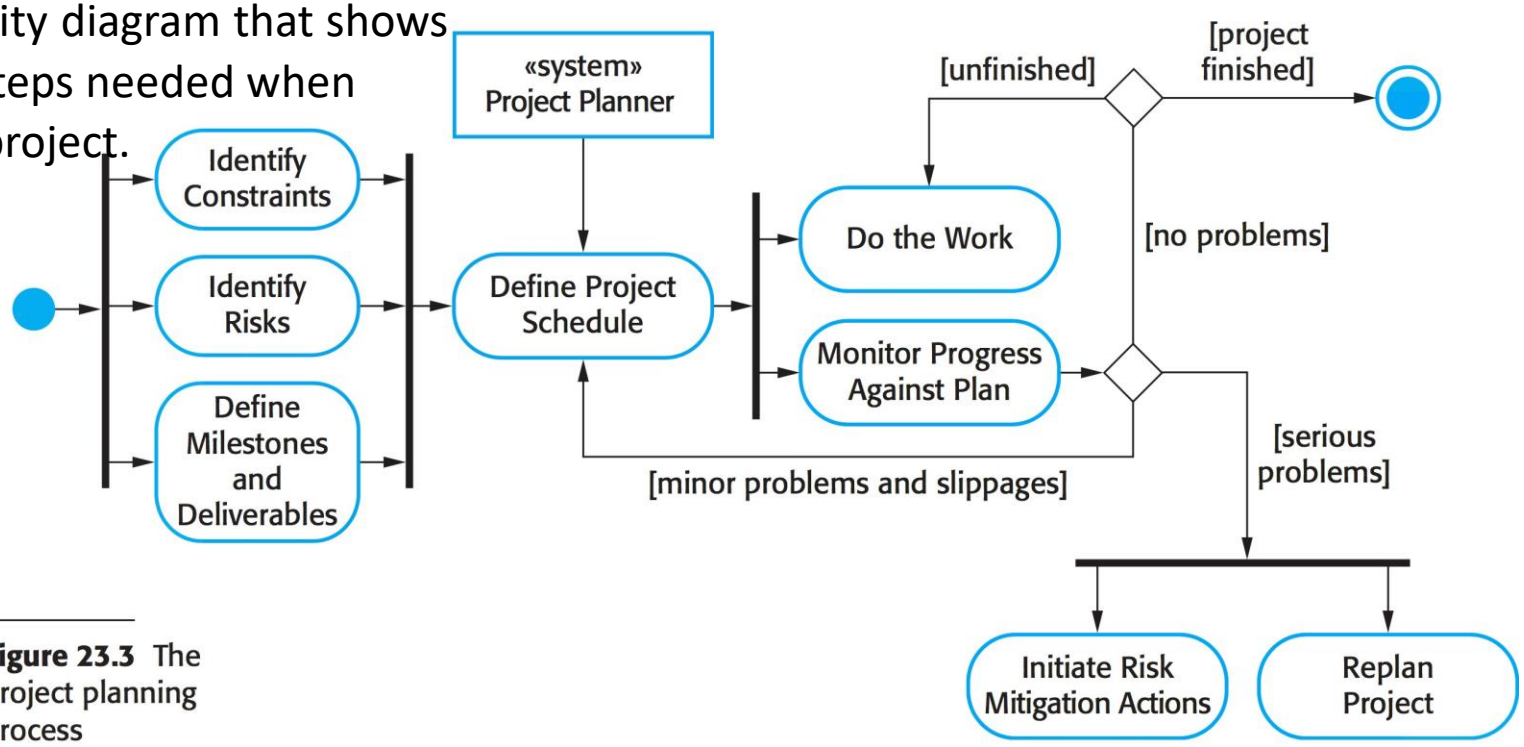


Figure 23.3 The project planning process

See: Sommerville, p. 625

L11

Building an Activity Network

- Develop activity network (with milestones, tasks, or task dependencies)
- Identify critical path (the longest path)
- Adjust milestones to match deadline
- Adjust task by changing / adding people

L11

Activity Network: Example

Activity / Task	Duration (days)	Dependencies
T1	8	
T2	15	
T3	15	T1
T4	10	
T5	10	T2, T4
T6	5	T1, T2
T7	20	T1
T8	25	T4
T9	15	T3, T6
T10	15	T5, T7
T11	7	T9
T12	10	T11

L11

Critical Path

- Critical path is the longest path in the activity network
- This critical path cannot effort any delays
  - Any delay in critical path causes project delay

L11

Risk Management

- What is risk management?
  - Identifying risks and drawing plans to minimise their negative effects
  - A risk is a probability that some adverse circumstance will occur
    1. **Project risks** affect project schedule and resources, e.g. a colleague leaves and is hard to be replaced.
    2. **Product risks** affect the quality or performance of software under development, e.g. a purchased component does not work as expected
    3. **Business risks** affects the organisation that produces the software, e.g. competitor introduces similar software.

Q&A?