# SIT771 – Lecture 6

## Responsibilities and design considerations

## Further reading

- Bernhard Rumpe (2016). Modeling with UML: Language, concepts, methods. Switzerland Springer. Ebook on Deakin Library – Chapter 6.

## Outline

### In this lecture...

- Responsibility-driven design (RDD)
  - Centralized
  - Distributed
- OOP system testing
- UML sequence diagrams
- Communication of design

## RESPONSIBILITY-DRIVEN DESIGN (RDD)

# OO design considerations

## First glance

- When you start designing a software solution using OO principles, there are two fundamental questions that you will need to answer:
  - What classes/objects to have in a program?
  - How should these classes/objects collaborate with each other to fulfill functionalities of a program?

- These considerations will lead to **responsibility-driven design**, which is…
  - A design approach that focuses on assigning responsibilities to objects within a system. It emphasizes the behaviour of objects rather than their data.

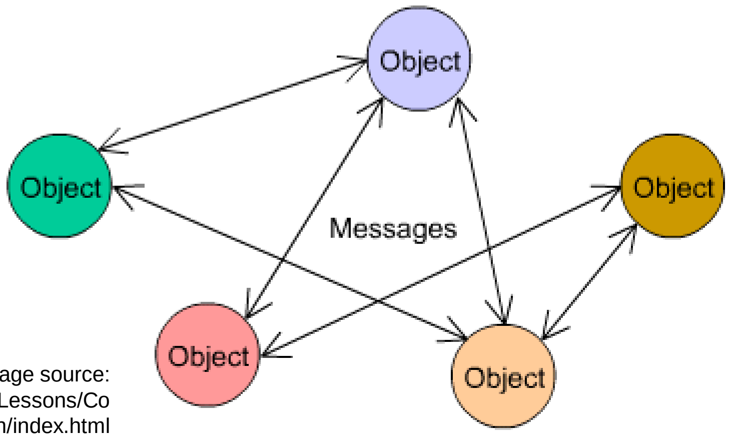


Image source: https://courses.cs.vt.edu/csonline/SE/Lessons/Comparison/index.html

# OO design considerations

## Second thought

- The aim of OO design is to identify…
  - The **roles** within your software solution
  - The associated **responsibilities** of each role

- When implemented in OO programs...
  - **Roles** will become **classes**
  - **Responsibilities** will be manifested within **methods** (and properties if needed)

Image source: https://www.hse-learning.com/

# OO design considerations

## Third view – Identifying collaborations

- Think about each object and…
  - Identify what other objects it needs to access
  - Discover additional roles and responsibilities for each object

- Example: Identifying collaborations between a **Car** and a **Buyer**…
  - Car will need to also have a **current location** as a field/property
  - Car will need to have a behavior to provide its test-drive available dates/times
  - Buyer will also need to have an **interest** as a field/property
  - *What OOP principle does this remind you of?* **Abstraction!**

Image source: https://keyautocompany.com/

# Core principles

## Summary

- RDD has three core principles…
  - **P1 – Classes/objects**, that are seen as abstractions resulting in active and well-defined roles
  - **P2 – Responsibilities**, that are well-defined and assigned to each object to carry specific knowledge and/or actions.
  - **P3 – Collaborations**, that puts responsible objects in interactions with other objects creating a network of responsibilities and collaboration.

Image source: https://cimatri.com/

# Key objectives

## Summary (cont.)

- RDD has three key objectives…
  - **Optimize abstraction**: Keeping the focus on how the objects should be represented
  - **Distribute behaviors**: Ensuring each object is "smart"…
    - has the data it needs (only)
    - can perform associated tasks (only)
  - **Preserve flexibility**: Making it easy to change the internal implementation details (e.g., within a class) with least effect on other parts of the solution (e.g., other classes)

Image source: https://hbr.org/2022/07/are-you-too-responsible

## Procedure

### Steps to take with RDD include...

- Identify candidate roles/objects (P1)
  - through an understanding of the domain
  - nouns in a problem statements

- Determine the responsibilities of each role/object (P2)
  - what the object should be able to do
  - what the object should know about itself

- Understand the system through role-play (P3)
  - what are the interactions between objects
  - can use UML sequence diagrams (TBD later in this lecture)

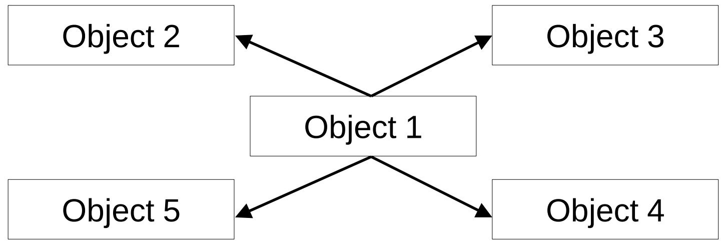
Reference: Introducing Object-Oriented Design with Active Learning, Rick Mercer , Consortium for Computing in Small Colleges, 2000

**Sample problem statement**
*The emergency department has several triage nurses in each shift, and they see several patients during their shifts. Patients come with chief complaints and are assigned triage categories according to the severity of their symptoms. Patients may wait for a period of time before they can see an emergency doctor, who may then prescribe tests to diagnose conditions or refer the patient to a specialist clinic for surgery...*

## Possible structures

### Centralized RDD...

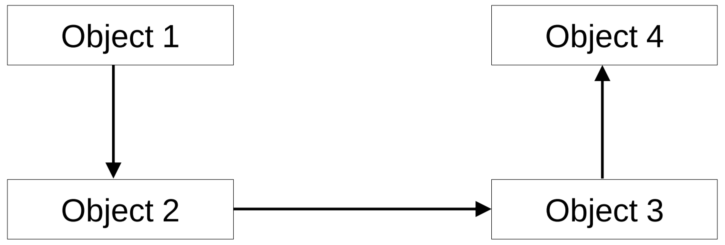
- Few objects contain all the functionalities and other objects provide little functionalities. The central authorities usually determine the system's overall state and actions, store and manage critical data, and interacts with other components as needed. (simple to implement and efficient for small-scale systems). Often leads to...
  - Tightly coupled components (classes)
  - Decreased reusability of central components (classes)



## Possible structures

### Distributed RDD…

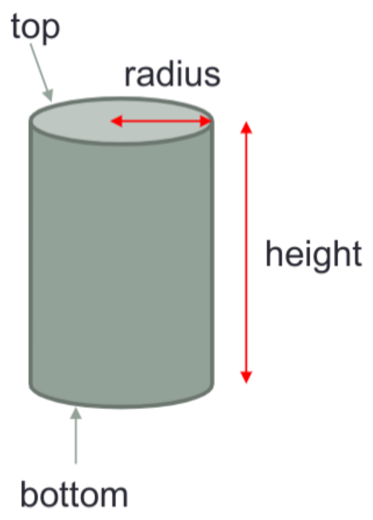
- The responsibilities are evenly distributed and shared among all objects. These components can operate independently and communicate with each other to achieve the system’s goals. (more fault tolerant and scalable). This leads to…
  - Highly cohesive and less complex single components (classes)
  - Increased reusability of cohesive components (classes)
  - Enhanced isolation of changes

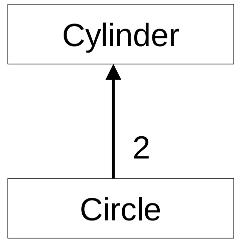


# Example RDD

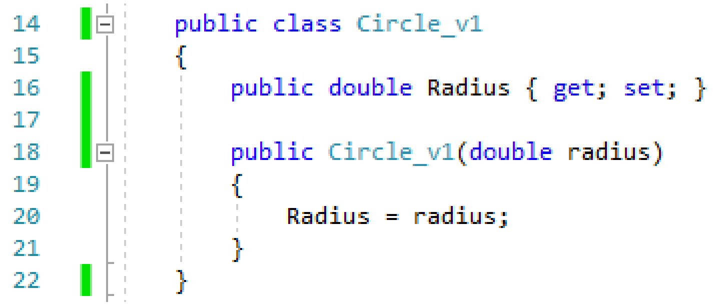
## Calculate cylinder measures

- Design two classes Circle and Cylinder – **v1.0**

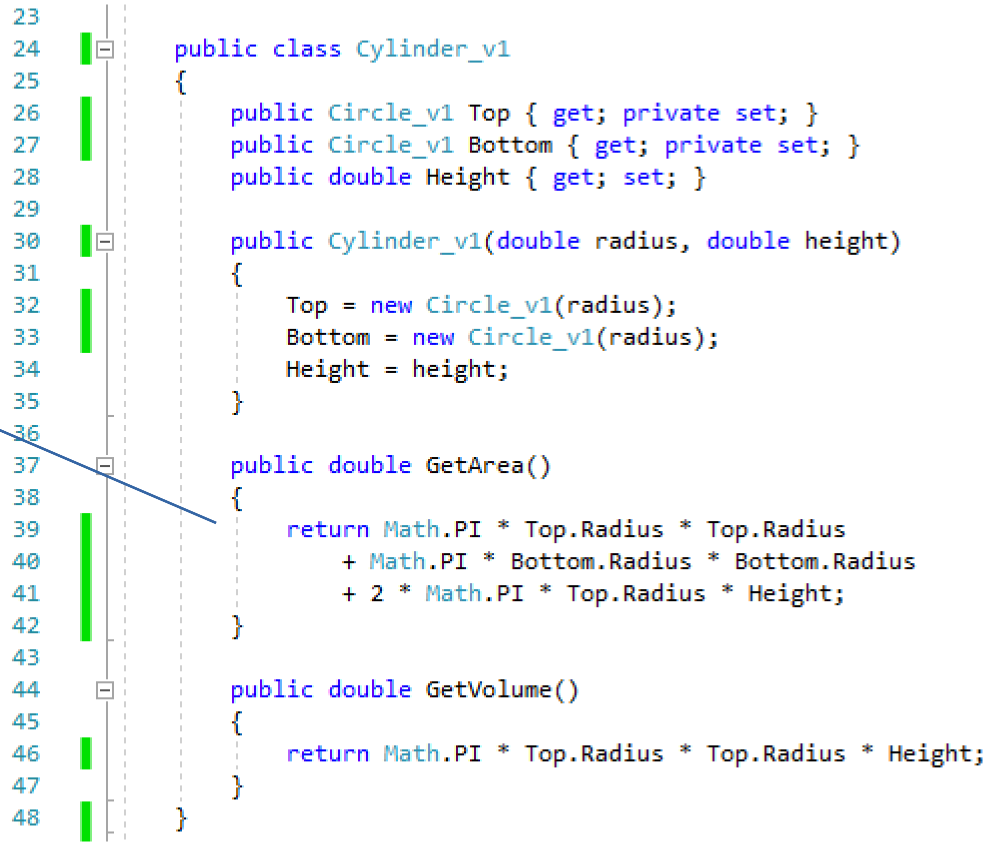




**Cylinder** implements all main functionalities while **Circle** does almost nothing!



```
public class Circle_v1
{
    public double Radius { get; set; }

    public Circle_v1(double radius)
    {
        Radius = radius;
    }
}
```



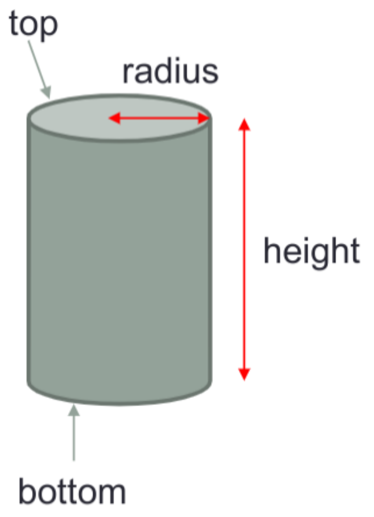
```

public class Cylinder_v1
{
    public Circle_v1 Top { get; private set; }
    public Circle_v1 Bottom { get; private set; }
    public double Height { get; set; }

    public Cylinder_v1(double radius, double height)
    {
        Top = new Circle_v1(radius);
        Bottom = new Circle_v1(radius);
        Height = height;
    }

    public double GetArea()
    {
        return Math.PI * Top.Radius * Top.Radius
            + Math.PI * Bottom.Radius * Bottom.Radius
            + 2 * Math.PI * Top.Radius * Height;
    }

    public double GetVolume()
    {
        return Math.PI * Top.Radius * Top.Radius * Height;
    }
}
```

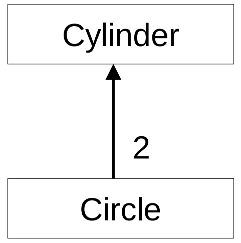
## Example RDD

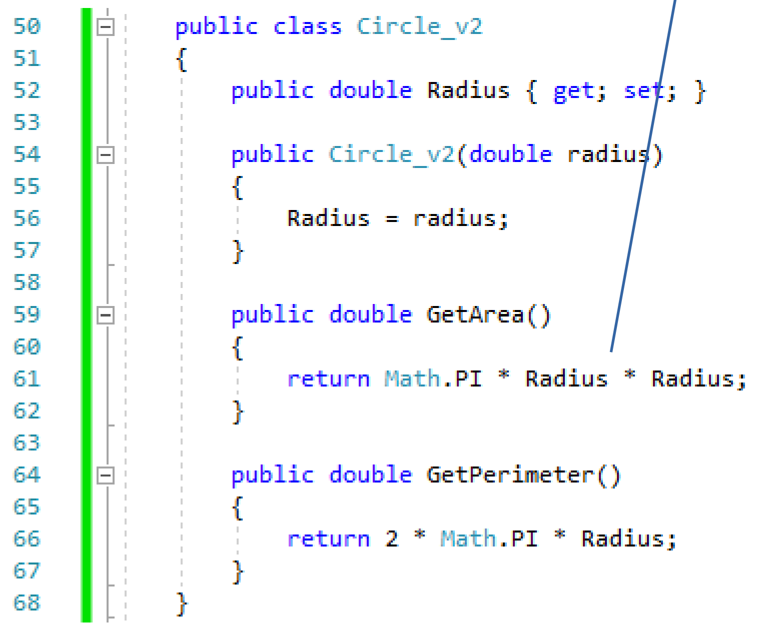
### Calculate cylinder measures

- Design two classes Circle and Cylinder – **v2.0**

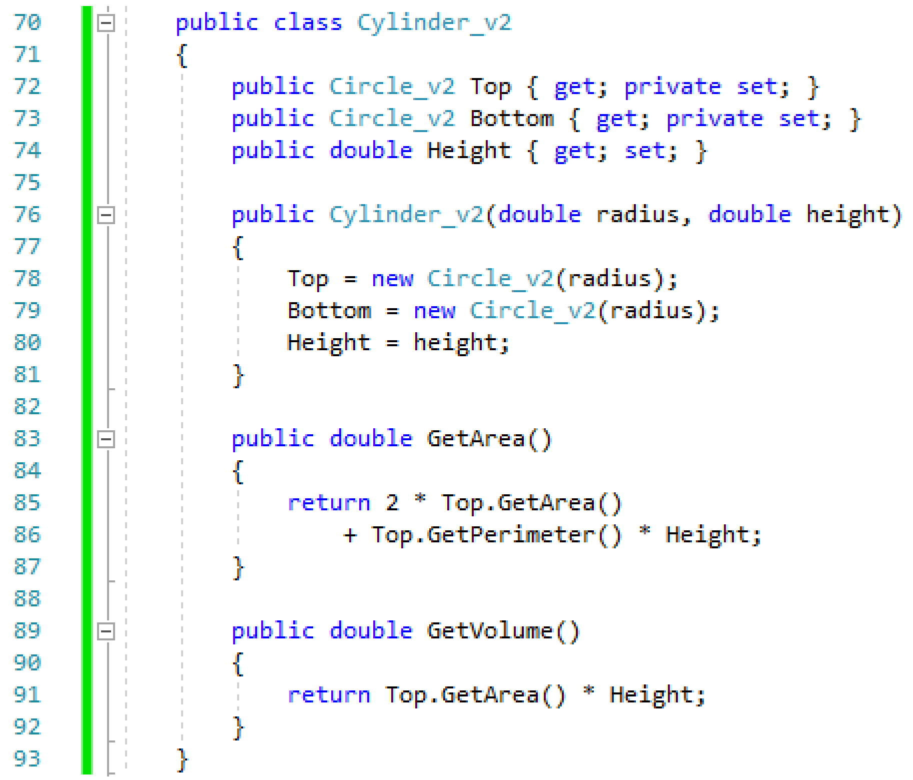
This is better responsibility sharing between **Circle** and **Cylinder**.







```
public class Circle_v2
{
    public double Radius { get; set; }

    public Circle_v2(double radius)
    {
        Radius = radius;
    }

    public double GetArea()
    {
        return Math.PI * Radius * Radius;
    }

    public double GetPerimeter()
    {
        return 2 * Math.PI * Radius;
    }
}
```



```
public class Cylinder_v2
{
    public Circle_v2 Top { get; private set; }
    public Circle_v2 Bottom { get; private set; }
    public double Height { get; set; }

    public Cylinder_v2(double radius, double height)
    {
        Top = new Circle_v2(radius);
        Bottom = new Circle_v2(radius);
        Height = height;
    }

    public double GetArea()
    {
        return 2 * Top.GetArea()
            + Top.GetPerimeter() * Height;
    }

    public double GetVolume()
    {
        return Top.GetArea() * Height;
    }
}
```

## Quiz

### Consider a system where…

- A single object handles complex calculations, user interface interactions, and data persistence. Which of the following RDD principles or objectives is most violated?
    - A) High cohesion
    - B) Low coupling
    - [✓ C) Single responsibility principle]
    - D) Collaboration

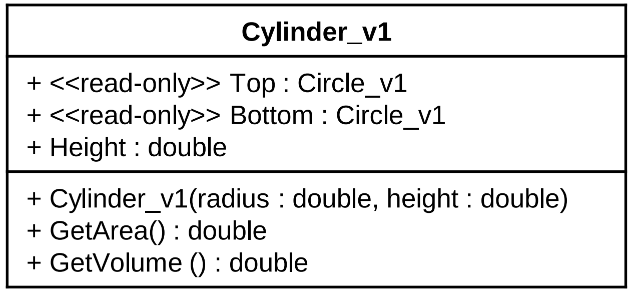
## OOP SYSTEM TESTING

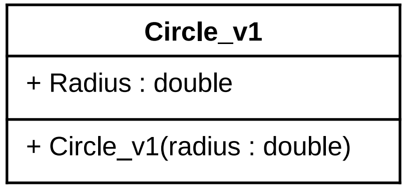
## Software test types

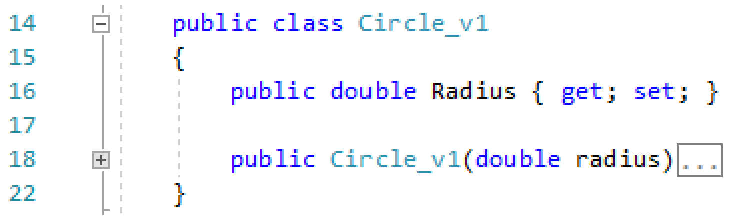
### 1- Unit testing

- Is a software testing method that focuses on validating the smallest testable parts of an application, known as units. These units are typically individual functions, methods, or classes. The goal of unit testing is to ensure that each unit works as expected in isolation before integrating it into the larger system.
- Is usually done by application developer or engineer.

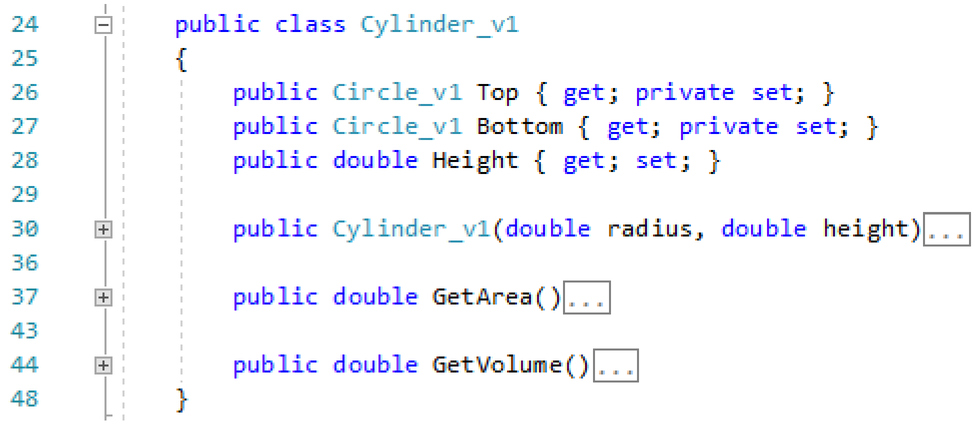
Check the correctness of the implementation of the two classes in **isolation**.


Cylinder_v1
+ <<read-only>> Top : Circle_v1
+ <<read-only>> Bottom : Circle_v1
+ Height : double
+ Cylinder_v1(radius : double, height : double)
+ GetArea() : double
+ GetVolume () : double





```
public class Circle_v1
{
    public double Radius { get; set; }

    public Circle_v1(double radius)...
}
```

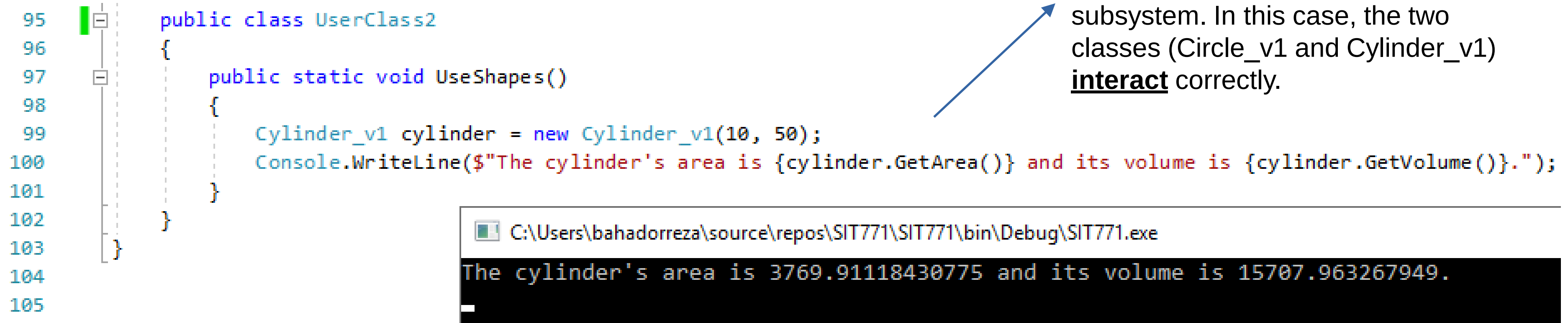


```
public class Cylinder_v1
{
    public Circle_v1 Top { get; private set; }
    public Circle_v1 Bottom { get; private set; }
    public double Height { get; set; }

    public Cylinder_v1(double radius, double height)...

    public double GetArea()...

    public double GetVolume()...
}
```

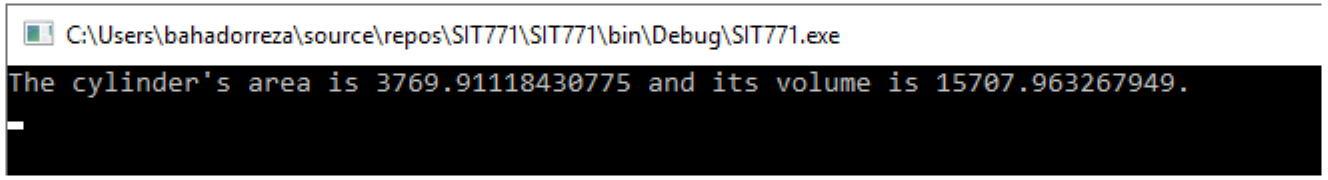
## Software test types

### 2- Integration testing

- Tests software or tools that are designed to facilitate the process of ensuring different components of a software application interact correctly. They provide features to help you identify and resolve issues that arise when these components are combined.
- Is also called <u>subsystem testing</u>.
- Is most often done by the subsystem lead.



```
public class UserClass2
{
    public static void UseShapes()
    {
        Cylinder_v1 cylinder = new Cylinder_v1(10, 50);
        Console.WriteLine($"The cylinder's area is {cylinder.GetArea()} and its volume is {cylinder.GetVolume()}.");
    }
}
```



C:\Users\bahadorreza\source\repos\SIT771\SIT771\bin\Debug\SIT771.exe

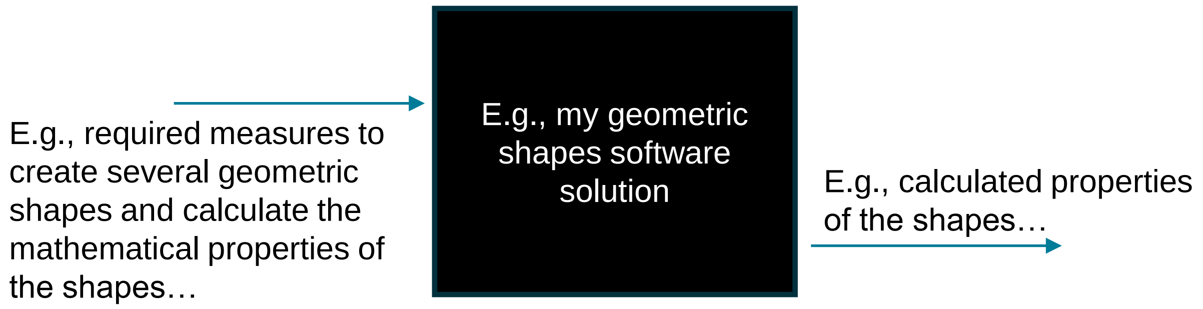
```
The cylinder's area is 3769.91118430775 and its volume is 15707.963267949.
```

Check the correctness of interactions among classes within a subsystem. In this case, the two classes (Circle_v1 and Cylinder_v1) **<u>interact</u>** correctly.

## Software test types

### 3- System testing

- Encompasses all aspects of the software, including integration and unit testing. This is a level of software testing that validates the complete and fully integrated software product. It evaluates the end-to-end system specifications to ensure it meets the specified requirements.
- Focuses on the non-functional and functional requirements.
- Is also known as black-box testing since it only focuses on the external behaviour of the system without considering internal structure.
- Is the responsibility of the quality assurance team.


E.g., required measures to create several geometric shapes and calculate the mathematical properties of the shapes…
E.g., my geometric shapes software solution
E.g., calculated properties of the shapes…

## Software test types

### 4- Regression testing

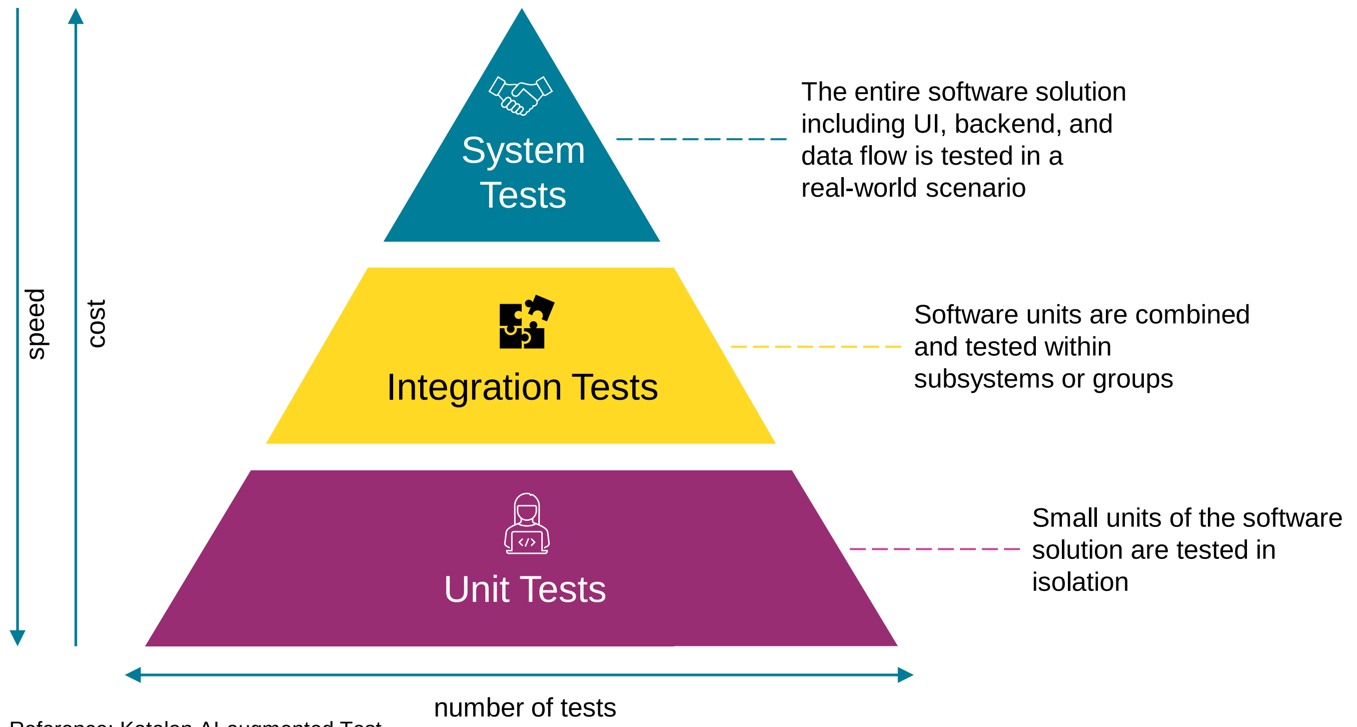
- Is a software testing technique used to ensure that previously developed and tested software still performs correctly (not regressed) after changes, such as bug fixes, new feature implementations, or code modifications. It involves re-executing a subset of previously executed test cases to identify any unintended side effects or regressions caused by the changes. Has different approaches:
  - **Retesting all:** to re-execute all test cases, which is time consuming but guarantees complete coverage.
  - **Selective testing:** to test a specific subset of components again, which is efficient but cannot guarantee complete coverage.
  - **Prioritization testing:** to prioritize test cases based on risk or impact.

- Regression testing is the responsibility of the quality assurance team.

## Software test types

### Put together…

- Notes:
  - Unit tests are the fastest and least costly tests as they are performed on very small units of software.
  - Regression testing may be triggered by the change at any level and may initiate any or all the other three tests that are shown in the diagram.


speed
cost
System Tests
The entire software solution including UI, backend, and data flow is tested in a real-world scenario
Integration Tests
Software units are combined and tested within subsystems or groups
Unit Tests
Small units of the software solution are tested in isolation
number of tests

Reference: Katalon AI-augmented Test Automation Platform

# Quiz

## Consider a complex software system with…

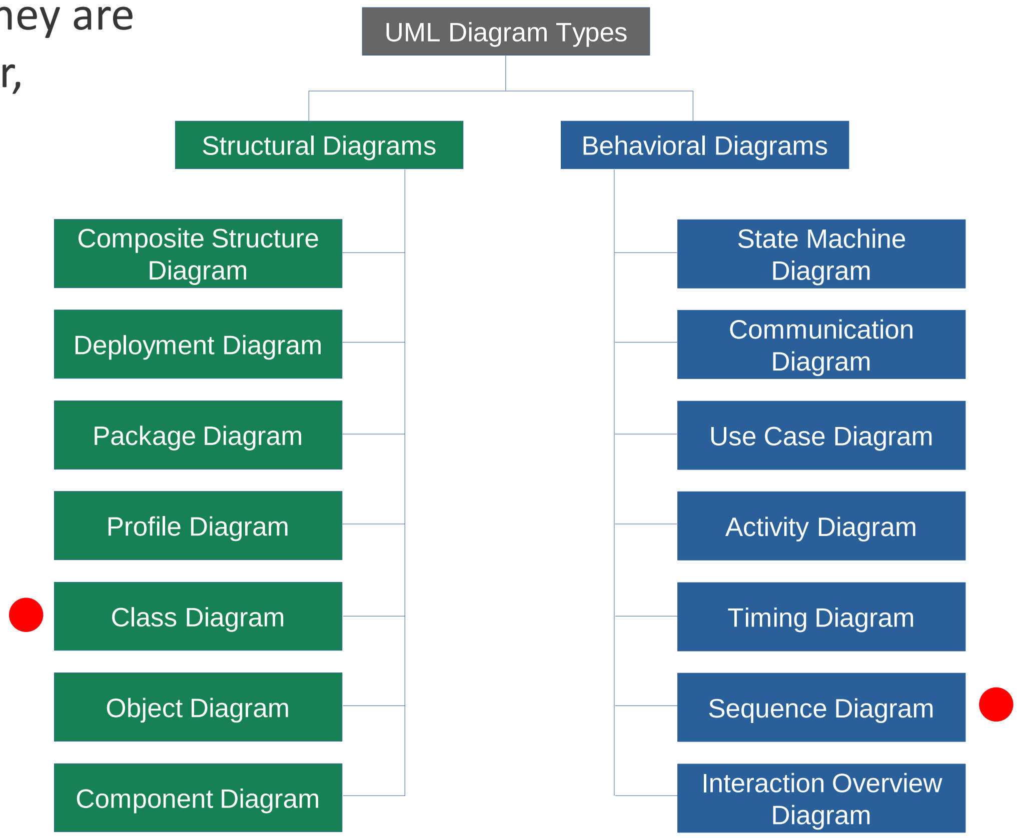
- Interdependent modules. Which testing approach would be most effective in isolating defects caused by interactions between these modules, while also ensuring that previously working functionalities remain intact after code modifications?
  - A) Unit testing, focusing on granular code coverage
  - [✓ B) Regression testing, combined with a robust test suite and selective execution]
  - C) System testing, emphasizing end-to-end functionality and performance
  - D) Integration testing, employing a top-down approach

## UML SEQUENCE DIAGRAMS

# Introduction

## Sequence diagrams…

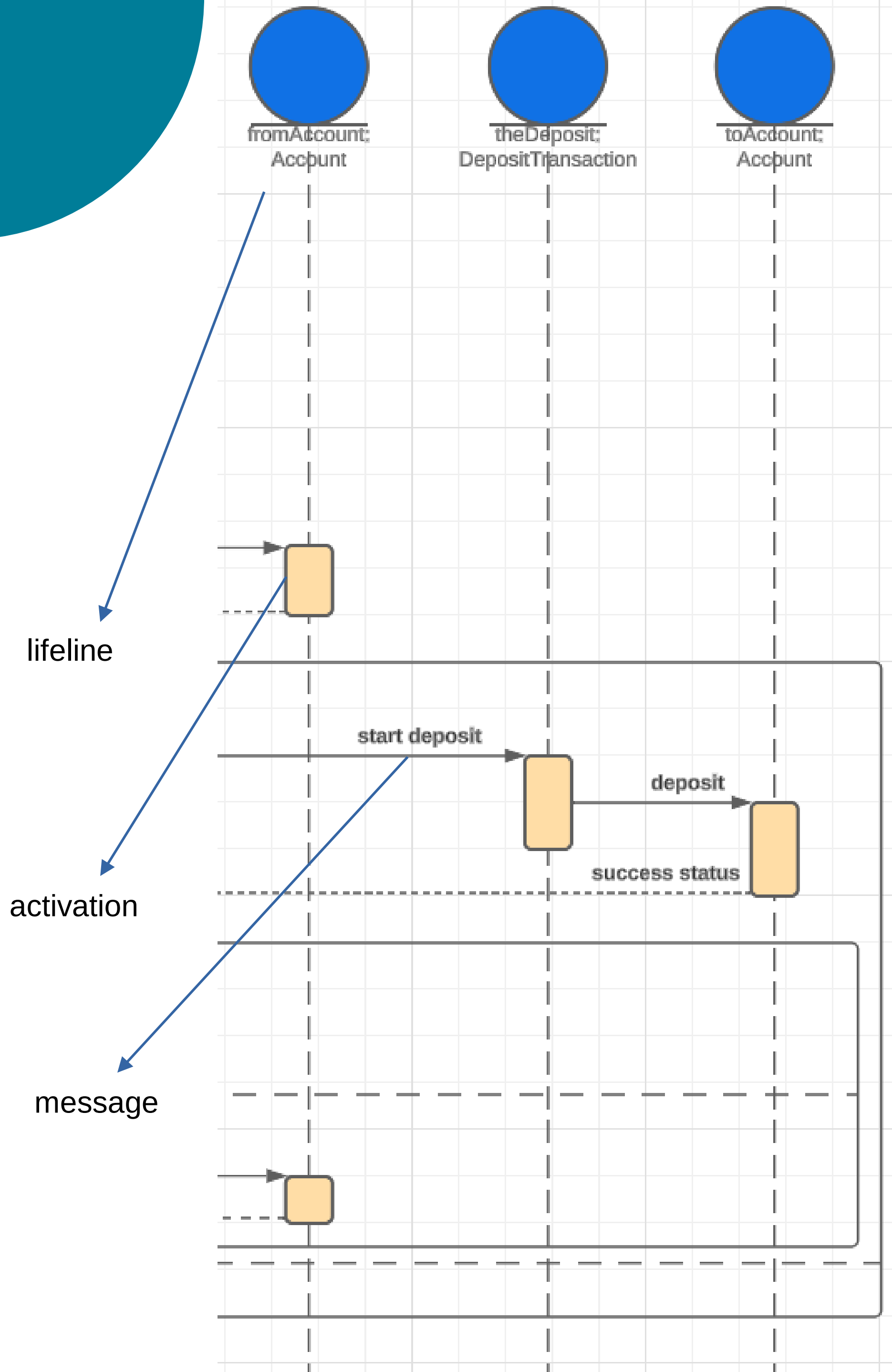
- Are powerful tools used in software development to visualize the interactions between different components of a system over time. They are particularly useful for understanding the flow of a system's behaviour, identifying potential bottlenecks, and clarifying requirements.
- Model the runtime interactions between:
  - objects/classes
  - subsystems within a system
  - a system and another system
  - a user and a system
- Show the sequence of method invocations in runtime



# Introduction

## Structure

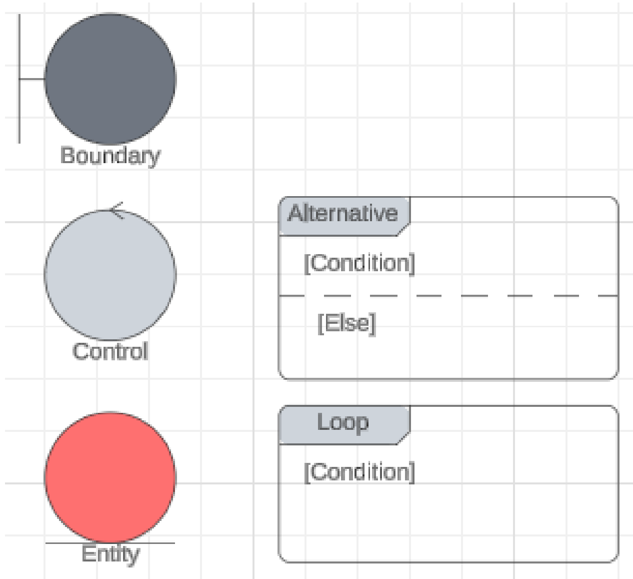
- Sequence diagrams show program elements that interact over time, thus:
  - horizontally, they represent objects
  - vertically down the page, they represent time
- Key elements include…
  - **Lifelines:** Represent the participants in the interaction, such as objects, actors, or systems. They are depicted as vertical lines and include different types of actors (see next slide).
  - **Messages:** Show the communication between lifelines. They are represented by arrows with labels indicating the message content.
  - **Activations:** Indicate when an object is active, performing a task. They are represented by rectangles on the lifeline.



# Introduction

## Structure (cont.)

- Lifelines in sequence diagrams can be of different types, the most frequently used of which are...
  - Boundary
    - Is a stereotyped object that is used to represent the interface through which some actions start. It interfaces with the user (or a user system), e.g., a terminal, a user interface, or a DBMS.
  - Control
    - Represents the object that controls the flow of some actions in the sequence. It (usually) mediates between boundary and entity objects, e.g., a menu selector, or a main function.
  - Entity
    - Is any object that holds the system data and that is responsible for the implementation of one or more actions, e.g., a player object, a robot object, or a window object.
- Other structural components represent selection (Alternative) and iteration(Loop).



# Example

## Medical operations scenario

- A **Patient** is seen by a **Doctor** who prescribes **Tests** and then an **Operation**...
  - The objects involved are at the top
  - The interactions are shown by the arcs
  - The time dimension is the inverse of the height

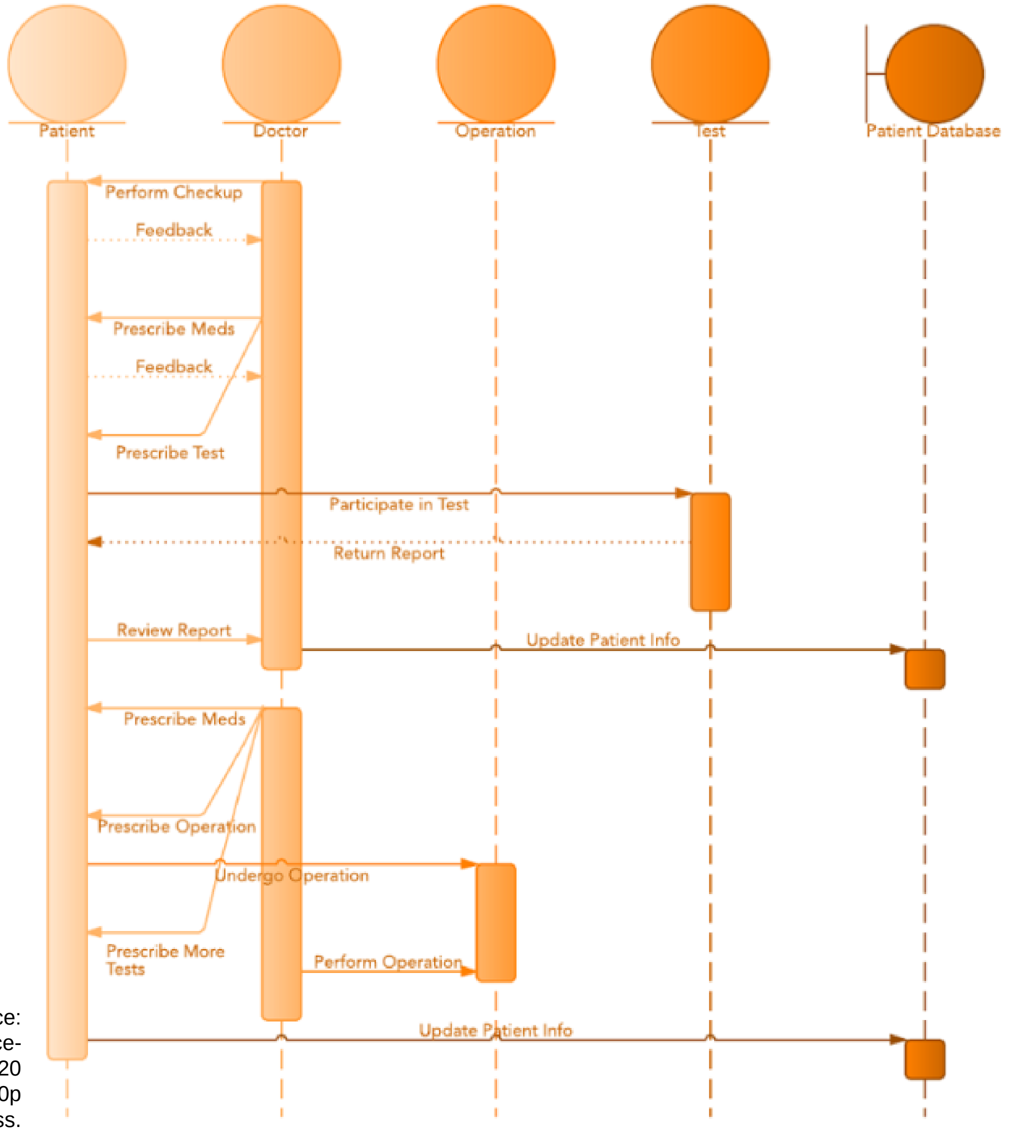


Image source: https://www.lucidchart.com/pages/uml-sequence-diagram#:~:text=A%20sequence%20diagram%20is%20a,to%20document%20an%20existing%20process.

## COMMUNICATION OF DESIGN

# Tips

## Designer-developer agreement

- To reduce the risk of mistakes/overheads...
  - A **short paragraph** (2-3 sentences) to communicate the core aspects of a role.
  - **Overview class diagrams** can be useful to help people get a feel for the solution.
  - Ideally, the **names** you assign to **roles** and **responsibilities** should communicate their **purpose**.
    - e.g., MyClass vs. Engine
    - e.g., Run() vs. MoveObjectDown()
  - Pick tricky or **informative interactions** to illustrate in sequence diagrams.
  - **Less** is often **better**, focus on what you are aiming to communicate.
  - In all cases, feel free to **leave out details** that are **not important**.

Image source: https://www.ekreative.com/

## Epilogue

THERE IS A SAYING IN THE SOFTWARE DESIGN INDUSTRY: “GOOD, FAST, CHEAP. PICK TWO.”

LARRY WALL