A TOUR OF SOFTWARE DESIGN

CSC207 SOFTWARE DESIGN

SOFTWARE DEVELOPMENT TEAM

- Developer (you): build the product
  - Design the architecture
    - How the parts of the program will be organized
    - Where persistent data is stored
    - How data passes between the parts of the program
  - Create the screens
    - Match a high-fidelity prototype that someone else created
    - Add functionality (what happens when a button is clicked?)
  - Test

Stuff to ponder

- How do developers know what user interfaces to create?
- How do developers know what data to keep track of?
- How do developers know that they are creating what the client wants?

SOFTWARE DEVELOPMENT TEAM

- **Product manager**: mini-CEO for a project
  - High level focus: understand client needs, turn their idea into reality
  - Stakeholder management: dev company, client, end users, dev team
  - Product success: define Minimum Viable Product (MVP), measure success (user surveys, client interviews), fine tune the product
- **Project manager**: in charge of dev team day-to-day details
  - Identifies *use cases*: what will users need to do with the application?
  - Understand high-level requirements, translate to step-by-step dev plan
  - Liaise between product manager, stakeholders, and dev team

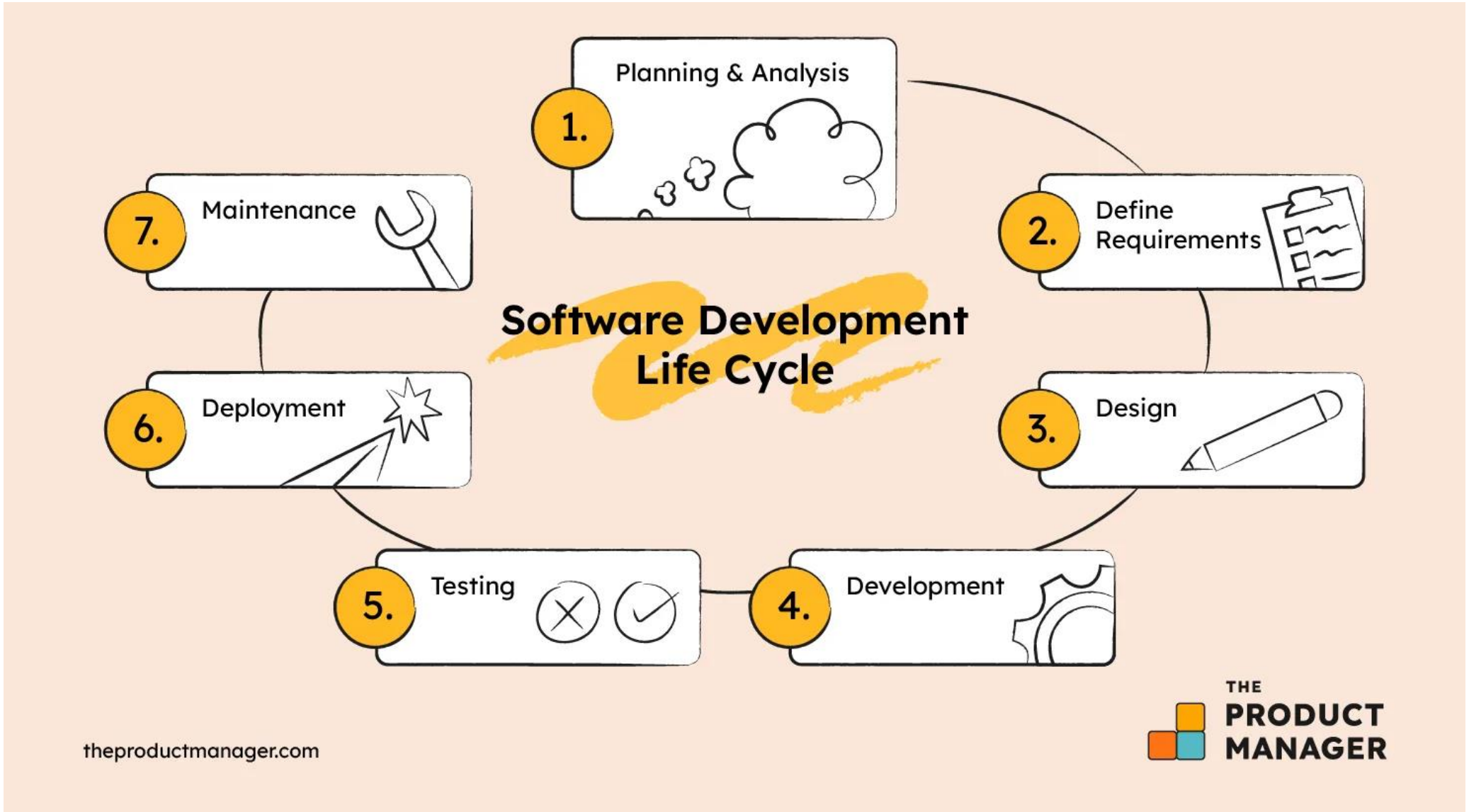
SOFTWARE DEVELOPMENT TEAM (CONTINUED)

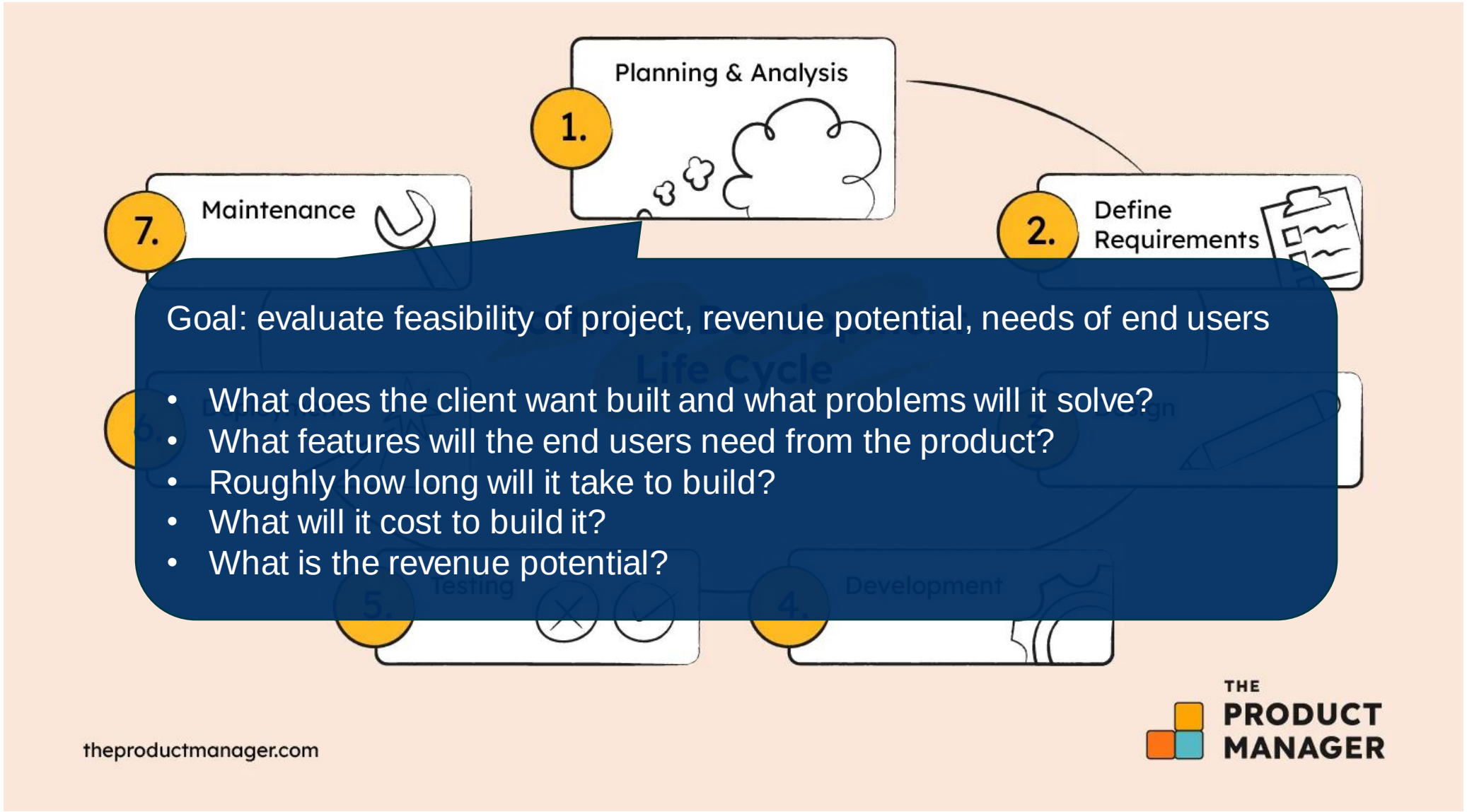
- **Designer**: User eXperience (UX) and a pretty User Interface (UI)
  - UX — how the user uses the app, navigating between screens
    - Draw high-level wireframes: focus on usability and user flow
      - No colours or other such details
    - Can the end user accomplish all the use cases?
    - Allows software developers to start planning
  - UI — Draw high-fidelity prototype
    - Based on wireframes, create fully-branded UI
    - Hand to devs to create

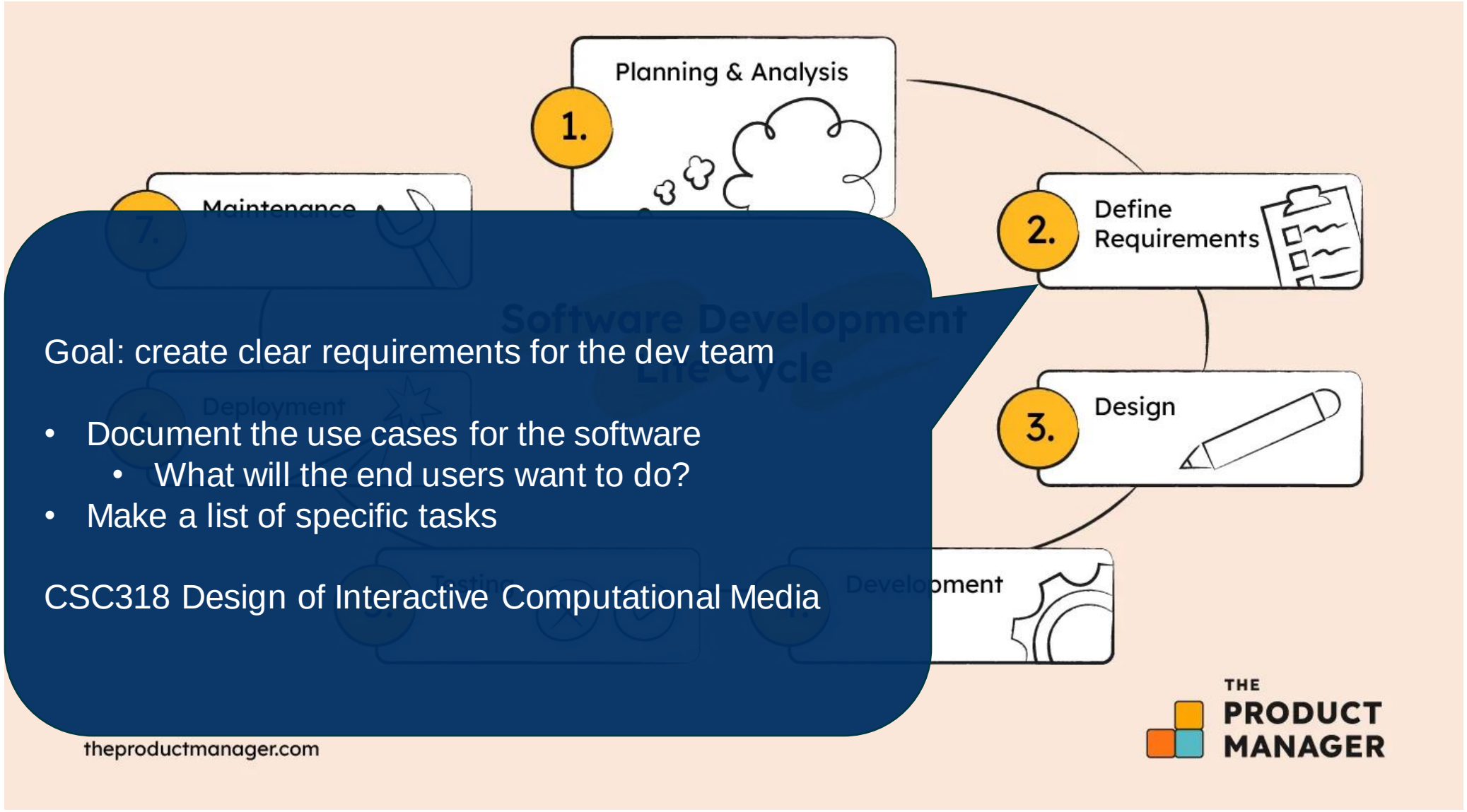
https://youtu.be/CbIMO5EcCD8

SOFTWARE DEVELOPMENT TEAM (CONTINUED)

- Quality Assurance: test the product (also you)
  - Review specification and ensure adherence
  - Test software on different browsers, screen sizes, network conditions
  - Try to break the software!


Planning & Analysis
1.
2.
Define Requirements
3.
Design
4.
Development
5.
Testing
6.
Deployment
7.
Maintenance
Software Development Life Cycle
theproductmanager.com
THE PRODUCT MANAGER


Planning & Analysis
1.
Define Requirements
2.
Design
3.
Development
4.
Testing
5.
Deployment
6.
Maintenance
7.
Life Cycle
Goal: evaluate feasibility of project, revenue potential, needs of end users
• What does the client want built and what problems will it solve?
• What features will the end users need from the product?
• Roughly how long will it take to build?
• What will it cost to build it?
• What is the revenue potential?
theproductmanager.com
THE PRODUCT MANAGER



Goal: create clear requirements for the dev team

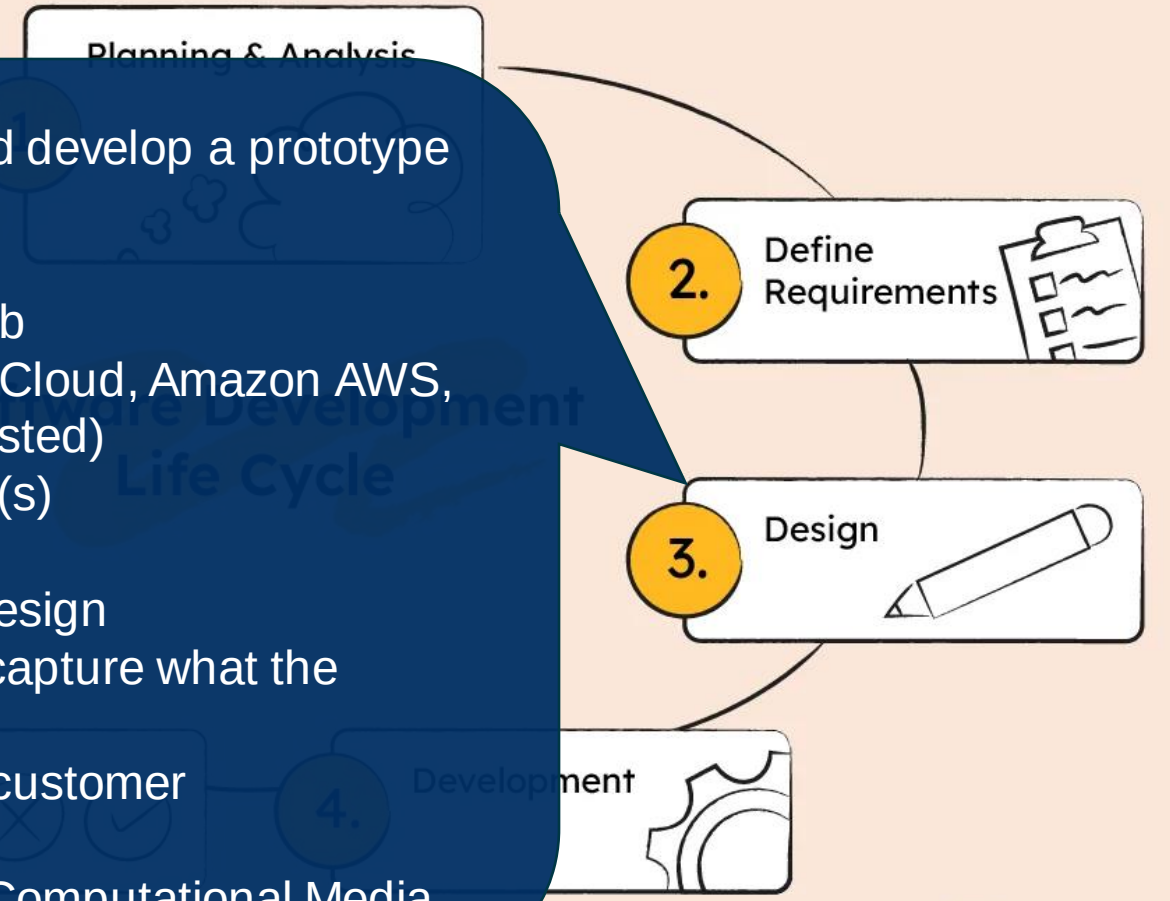
- Document the use cases for the software
  - What will the end users want to do?
- Make a list of specific tasks

CSC318 Design of Interactive Computational Media

Goal: decide on tech needs and develop a prototype

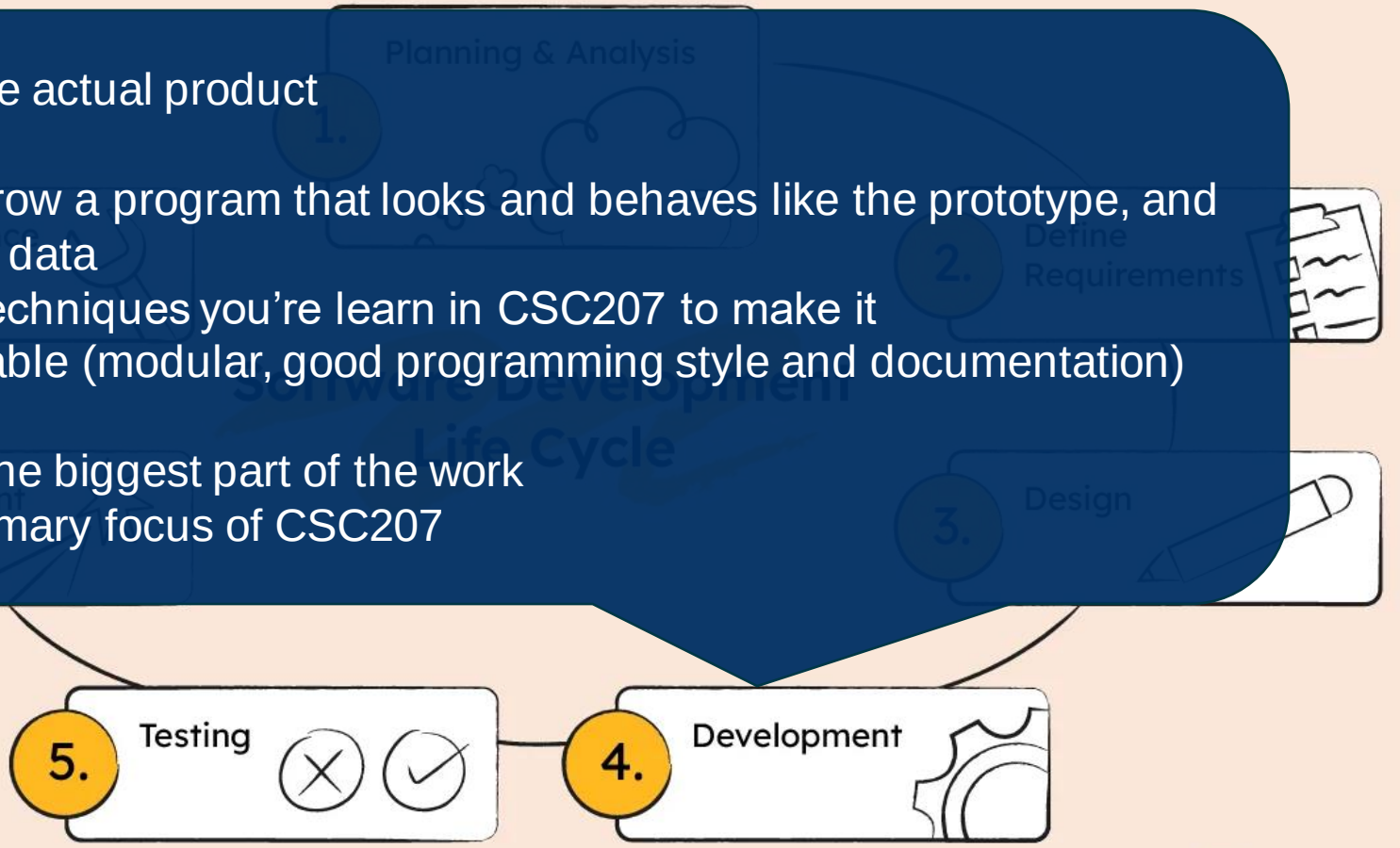
- Decide on your “stack”
  - iOS, Android, and/or web
  - Server hosting (Google Cloud, Amazon AWS, Microsoft Azure, self-hosted)
  - Programming language(s)
- Develop a prototype
  - No programming, just design
  - Draw some pictures to capture what the screens will look like
  - Validate prototype with customer

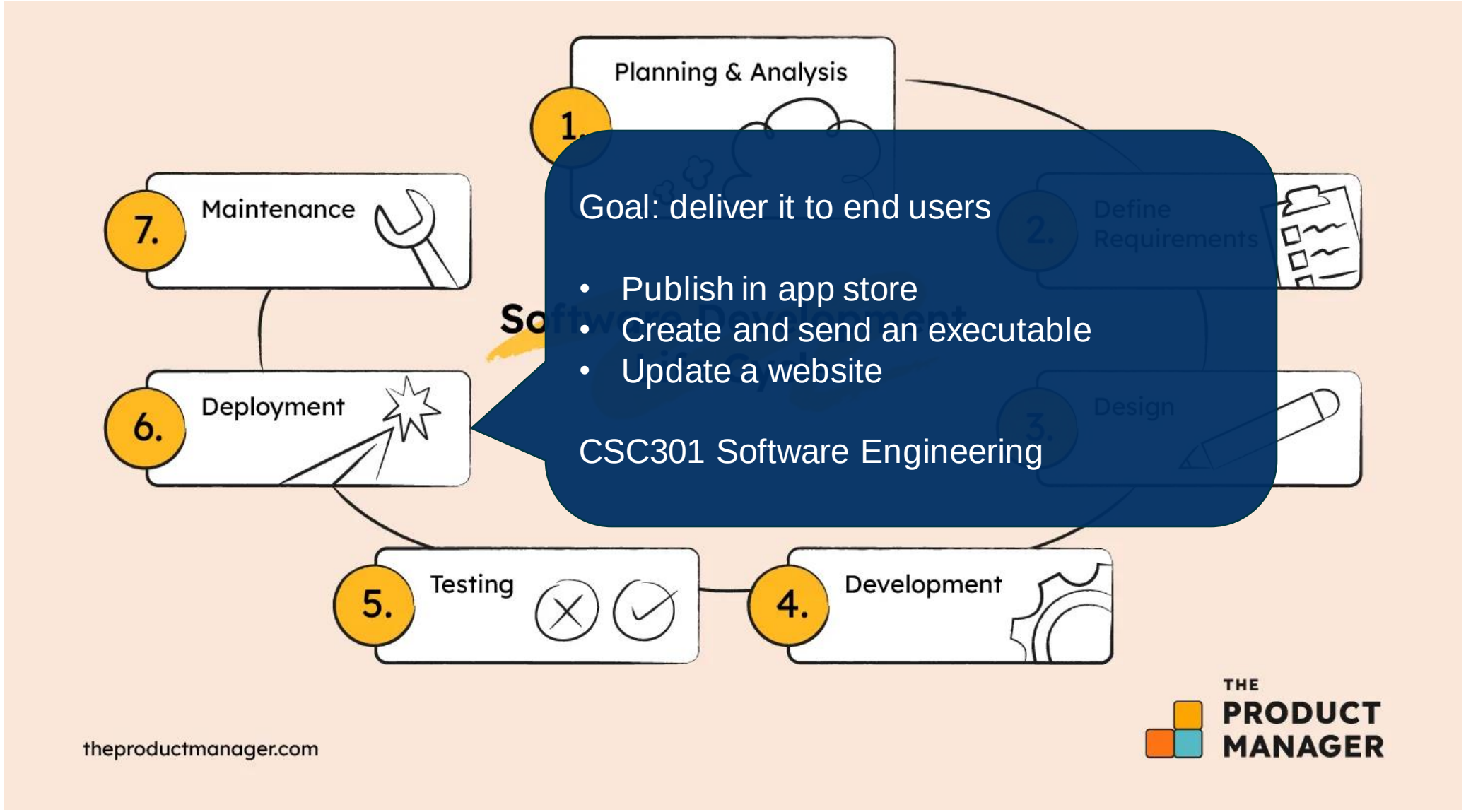
CSC318 Design of Interactive Computational Media

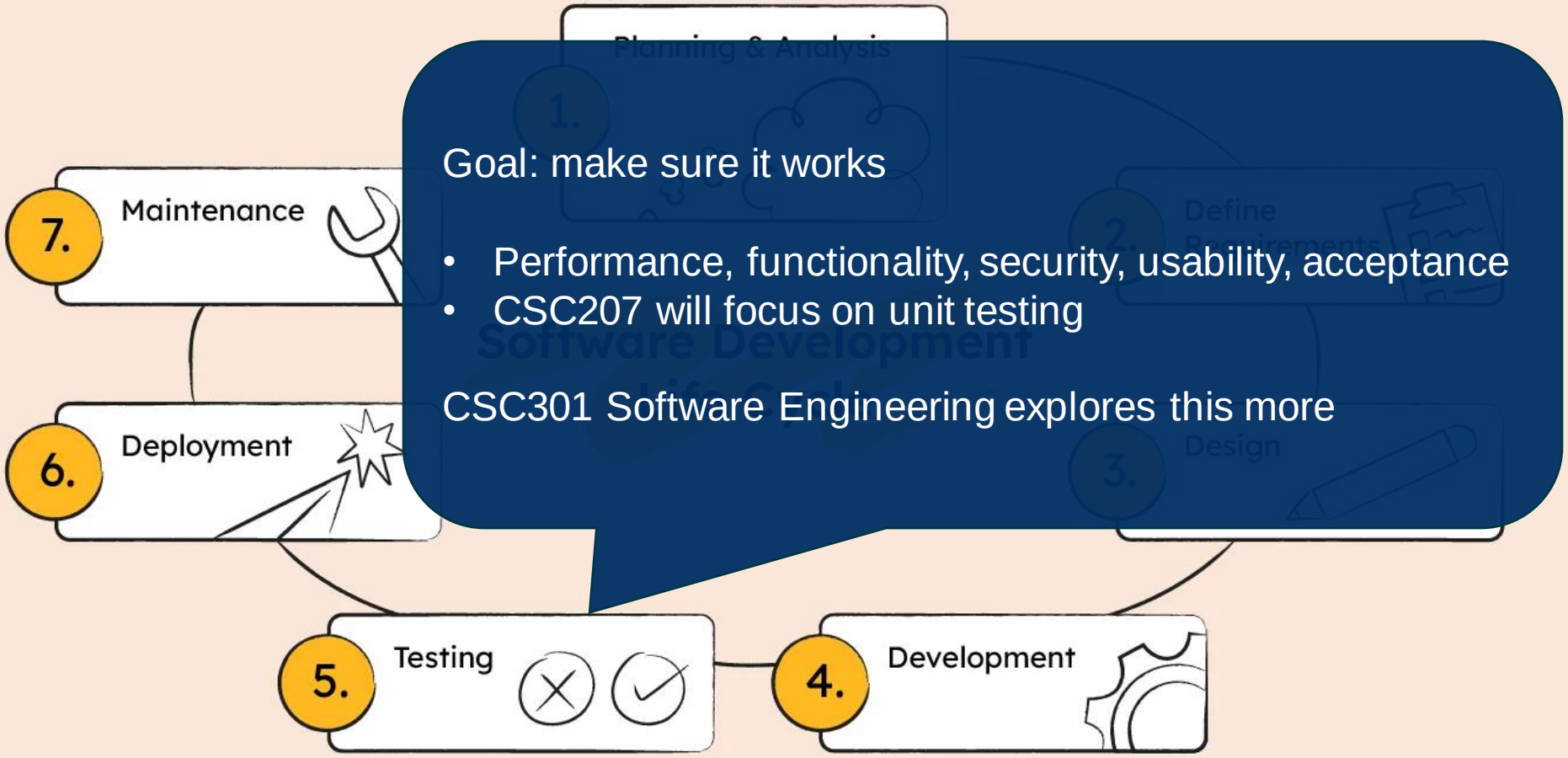


Goal: develop the actual product

- Design and grow a program that looks and behaves like the prototype, and manages real data
- Apply fancy techniques you're learn in CSC207 to make it
  - Maintainable (modular, good programming style and documentation)
  - Testable
- This is often the biggest part of the work
- This is the primary focus of CSC207



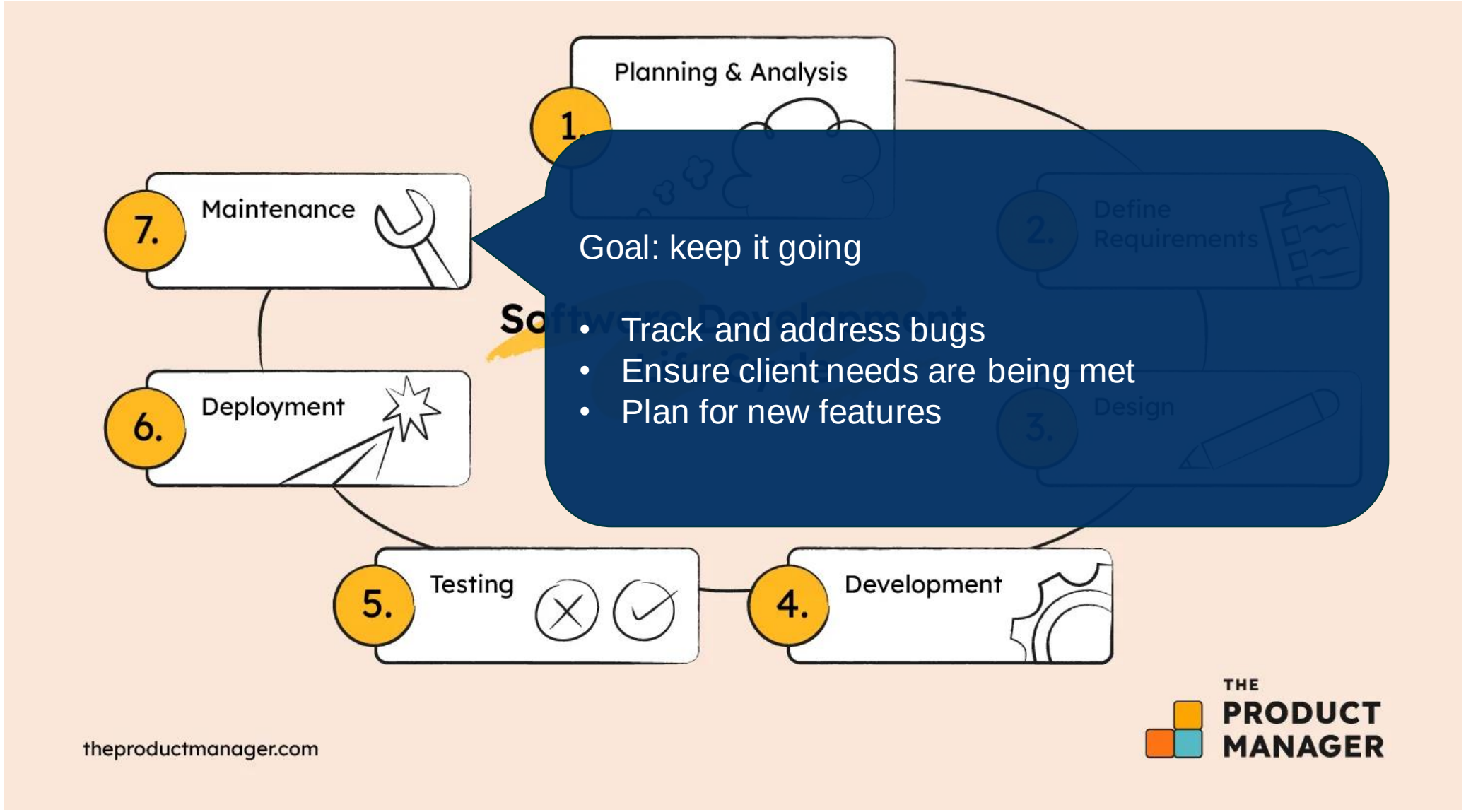

Planning & Analysis
1.
2.
Define Requirements
3.
Design
4.
Development
5.
Testing
6.
Deployment
7.
Maintenance
Goal: deliver it to end users
• Publish in app store
• Create and send an executable
• Update a website
CSC301 Software Engineering
theproductmanager.com
THE PRODUCT MANAGER



Goal: make sure it works

- Performance, functionality, security, usability, acceptance
- CSC207 will focus on unit testing

CSC301 Software Engineering explores this more


Planning & Analysis
1.
2.
Define Requirements
3.
Design
4.
Development
5.
Testing
6.
Deployment
7.
Maintenance
Goal: keep it going
• Track and address bugs
• Ensure client needs are being met
• Plan for new features
theproductmanager.com
THE PRODUCT MANAGER

LEARNING OUTCOMES OF THIS LECTURE

- Understand that (good) industry software is organized into layers
  - User interface and persistence, interface adapters, use cases, core classes
- Explain why each layer has a “public interface”
  - the set of classes and methods that it exposes to the world
  - often called an application programming interface (API)
- Explain which parts of a program need to change when moving to a new platform, and how to structure your program to enable this
- Get a feel for what this course will be like

CSC108/148/110/111 STUFF YOU KNOW

- value and type; expressions
- naming a value using an assignment statement (assigning a value to a variable)
- control flow: sequence of statements, if, while, for, function call, return statement, call stack, recursion
- ADTs and data structures: string, list, dictionary, linked list, stack, queue, tree
- classes and the objects they describe; composition; inheritance (OOP)
- some variables and methods are private (Python: use a leading `_underscore`)
- computational complexity (big-Oh)
- unit testing, debugging
- function and class design recipes — processes by which to write code

USE CASES FOR A PROGRAM

Imagine you were asked to write a program that allows users to

- **register a new user account** (with a username and password)
- **log in to a user account**
- **log out of a user account**
- They're planning on having a few different kinds of accounts, but we'll start with just one for now.

Bold words are *use cases*: what will the user want to do?

1. What data needs to be represented?
2. What data structure might you use while the program is running?
3. What should happen if the user quits and restarts the program?

USE CASE: USER REGISTERS NEW ACCOUNT

USE CASE: USER REGISTERS NEW ACCOUNT

- The user chooses a username
- The user chooses a password and enters it twice (to help them remember)
- If the username already exists, the system alerts the user
- If the two passwords don't match, the system alerts the user
- If the username doesn’t exist in the system and the passwords match, then the system creates the user but does not log them in

USE CASE: USER LOGS IN

USE CASE: USER LOGS IN

- The user enters a username and password
- If the username exists in the system and the passwords match, then the system shows that the user is logged in
- If there is no such username, the system alerts the user
- If the password doesn't match the one in the system, the system alerts the user

USE CASE: USER LOGS OUT

- The system logs the user out and informs the user

USE CASE: WHEN LOGGED OUT, CHOOSE USE CASE

- The user chooses between the user *registers a new account* use case and the *user logs in* use case

BURNING QUESTIONS

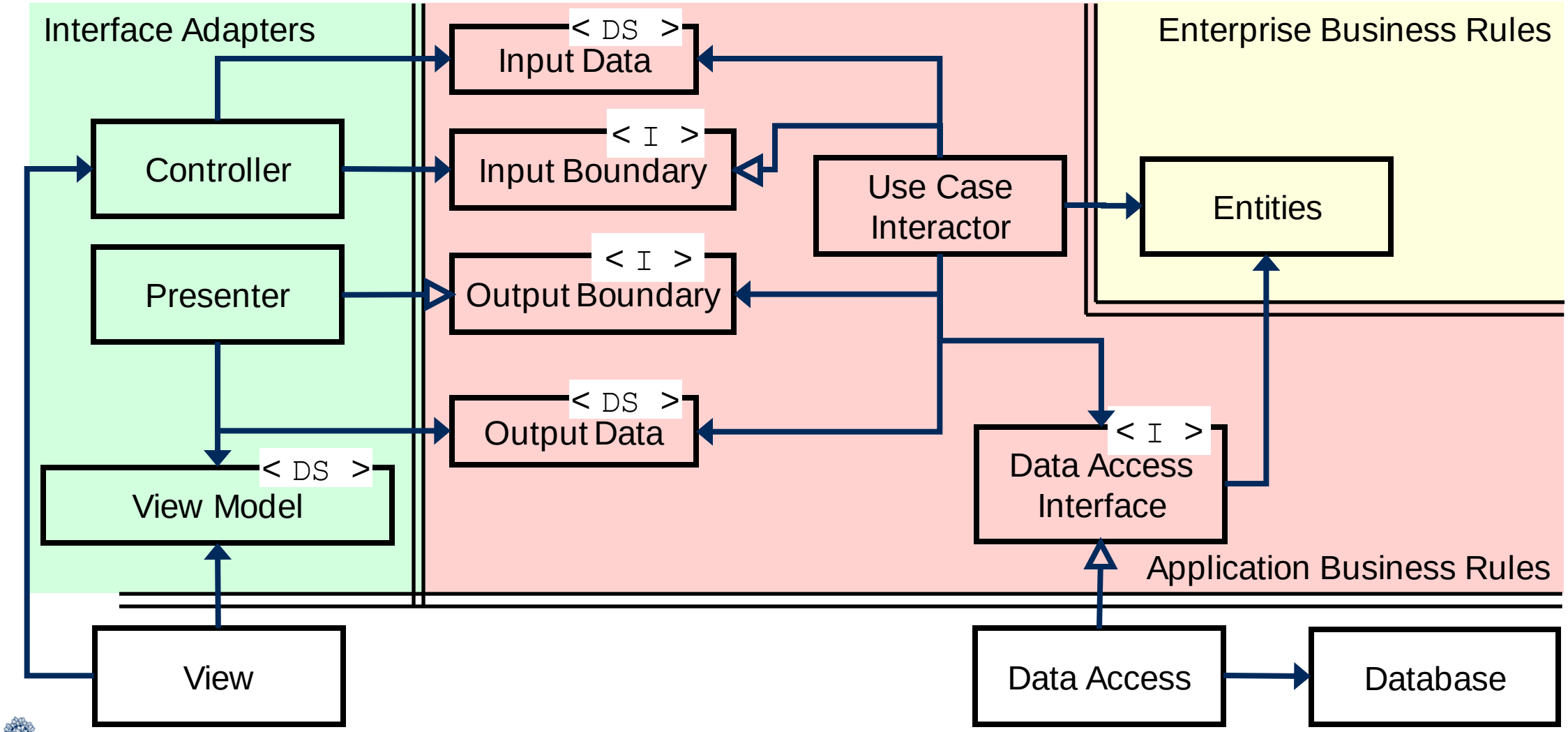
- What is the user interface?
  - A webpage?
  - A Java application on your computer?
  - A Python command-line program?
  - A mobile app?
- How to do data persistence?
  - A text file?
  - A database?
  - Google Drive/OneDrive/etc.?

- How can you design your program so that it's easy to move to a new UI?
- How can you design your program so that it's easy to save data to a different kind of storage?
- How can you design your program so that **as much code as possible** stays the same when you do these things?

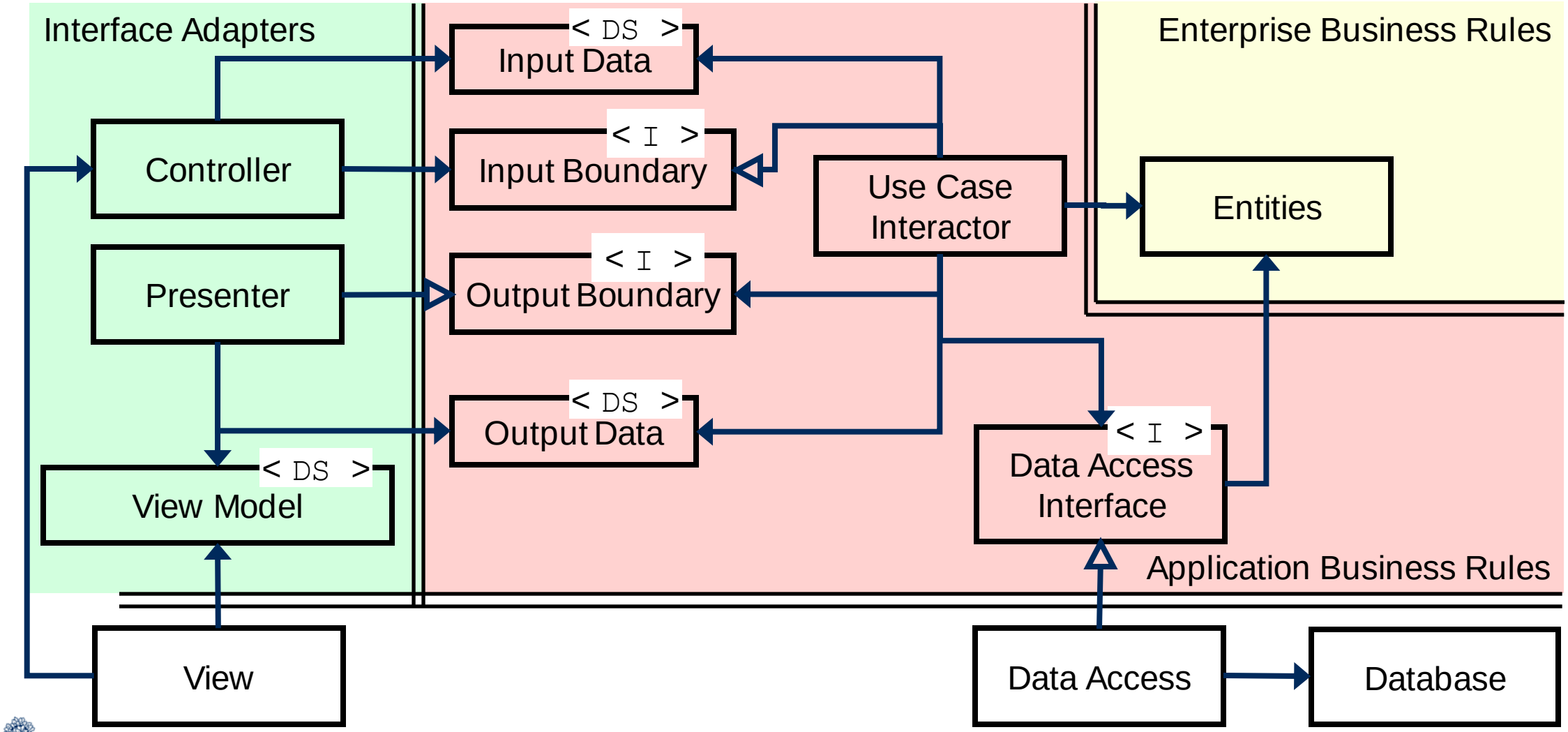
DESIGN CONUNDRUMS

- How can we design the use cases so that they **do not directly depend** on the UI and persistence choices?
  - Then we can test all the use cases thoroughly!
- What are the use case APIs?
  - What is the interface to each use case?
  - What public methods do we want to provide to call the use cases from the UI?
- What persistence methods will we need in any storage?
    - Saving
    - Finding a user by username
    - Etc.

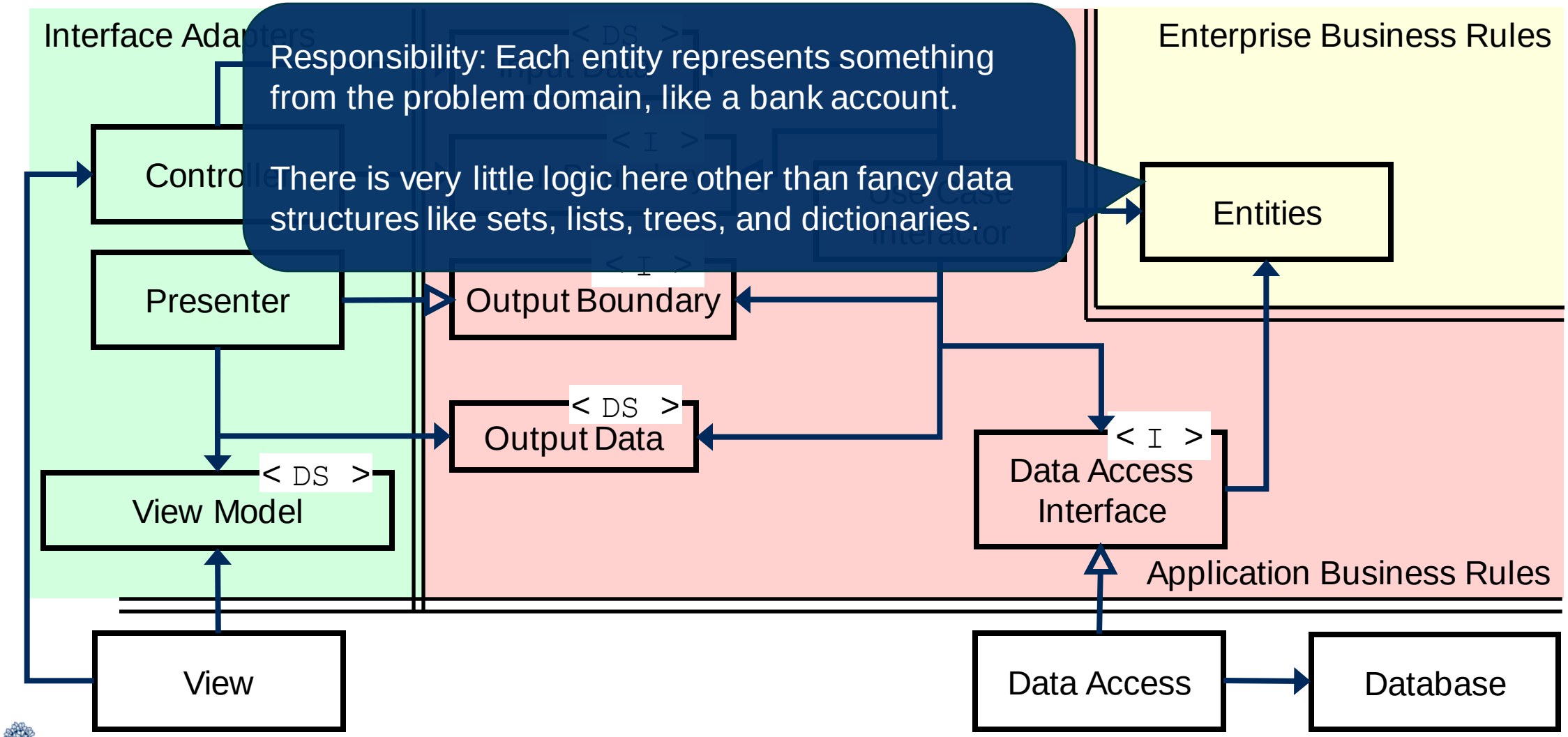
THE PARTS OF AN ENGINE TO MAKE A FEATURE WORK



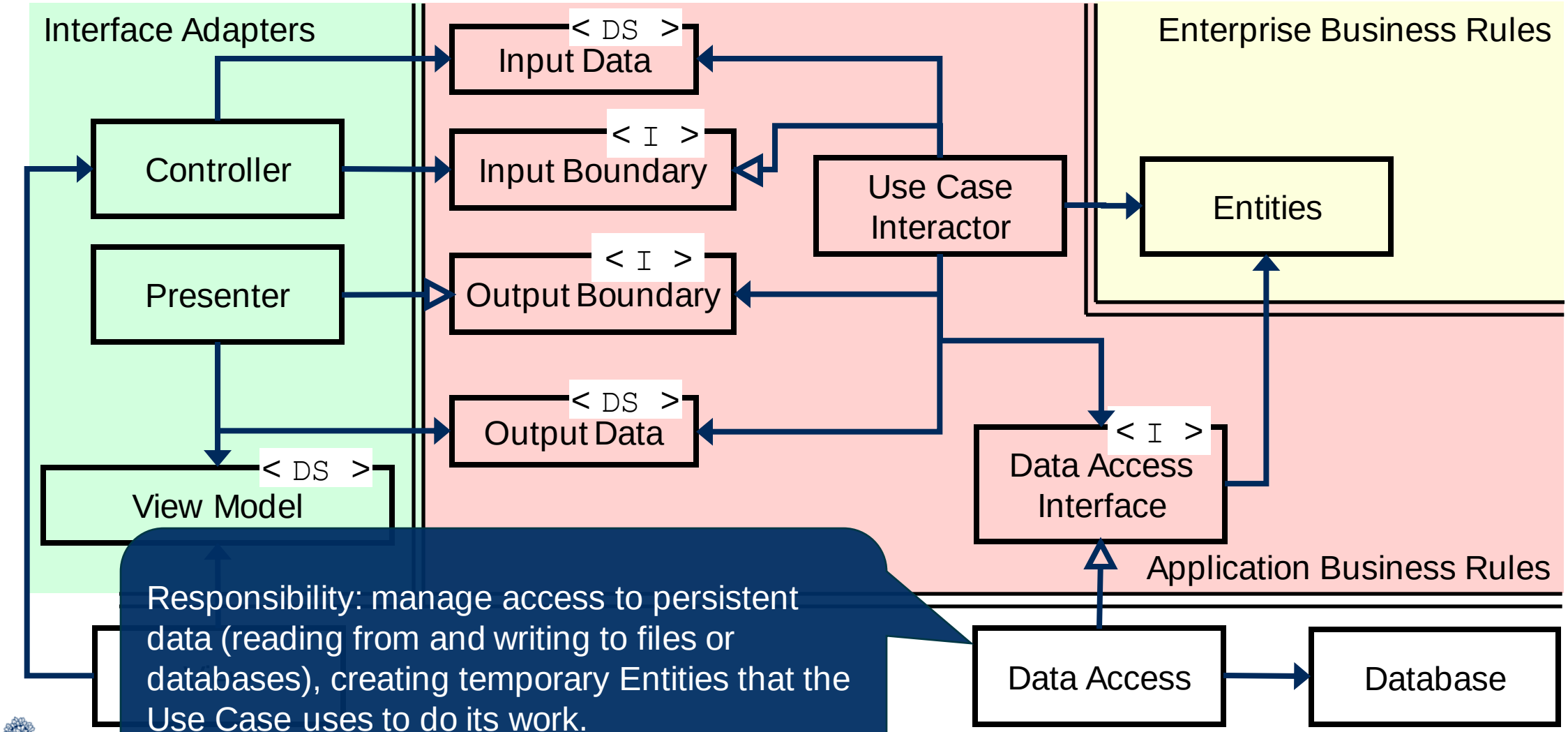
PROJECT: EACH OF YOU WILL BUILD THIS ENGINE TO IMPLEMENT A FEATURE



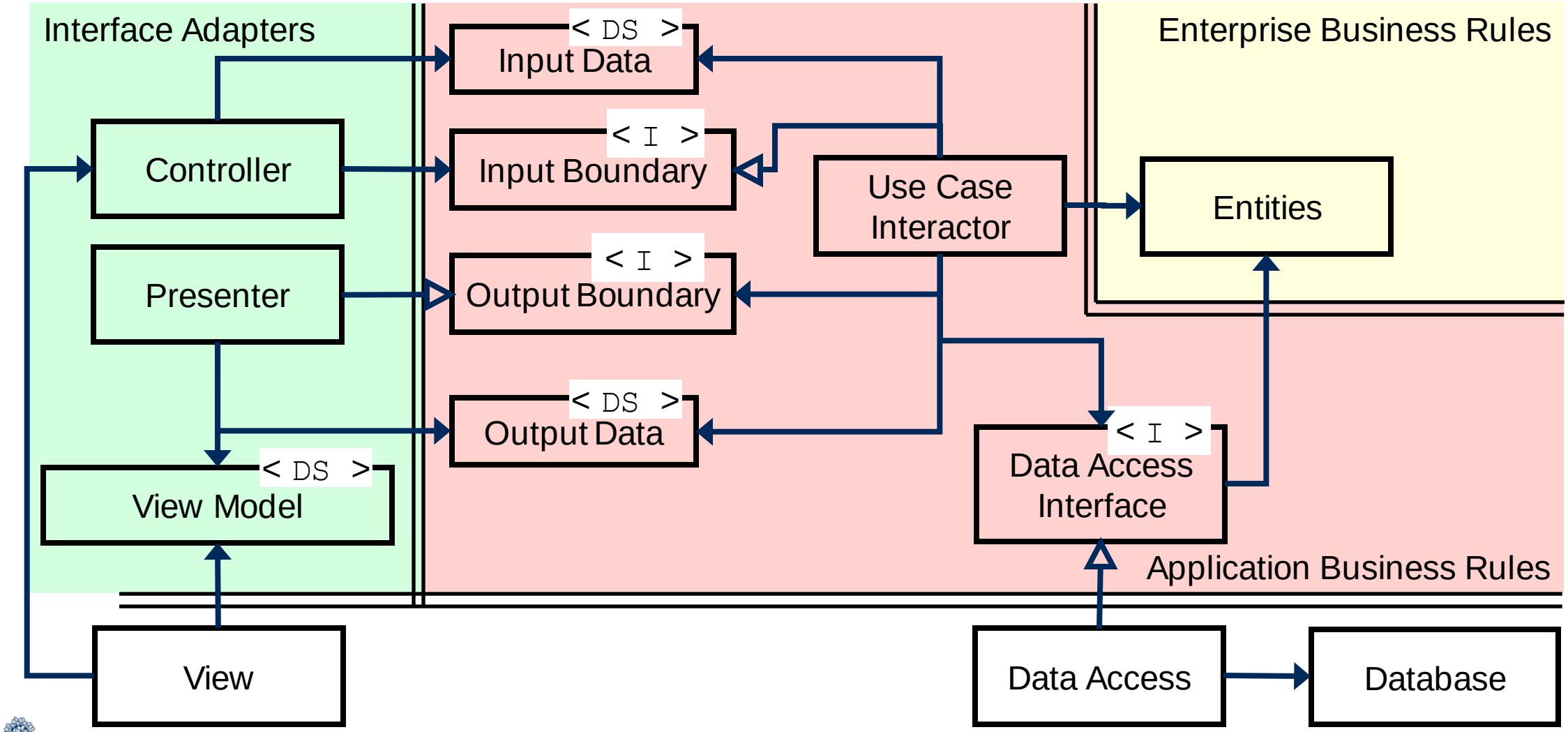
PROGRAM DATA: ENTITIES



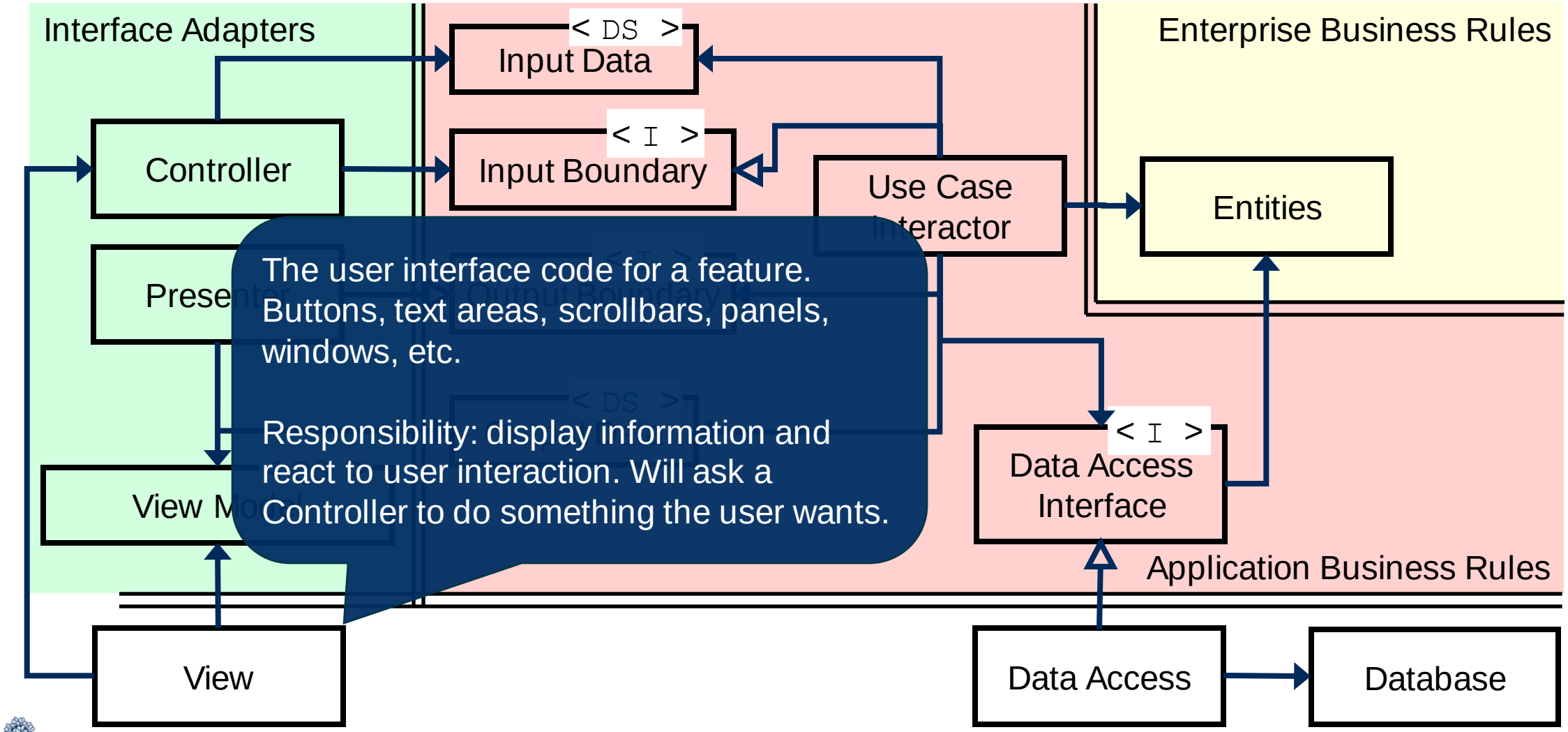
PROGRAM DATA: PERSISTENT DATA



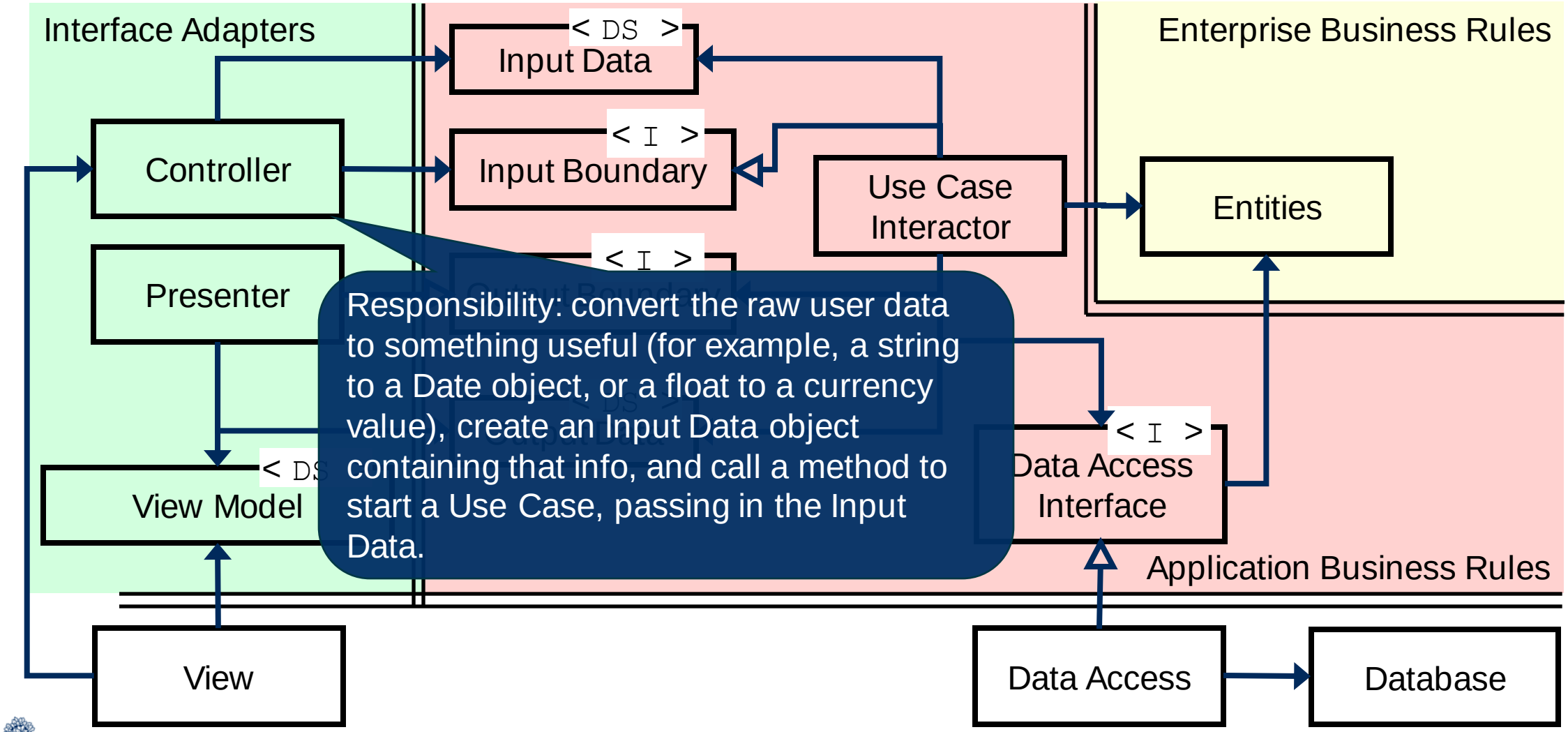
HOW THE ENGINE WORKS



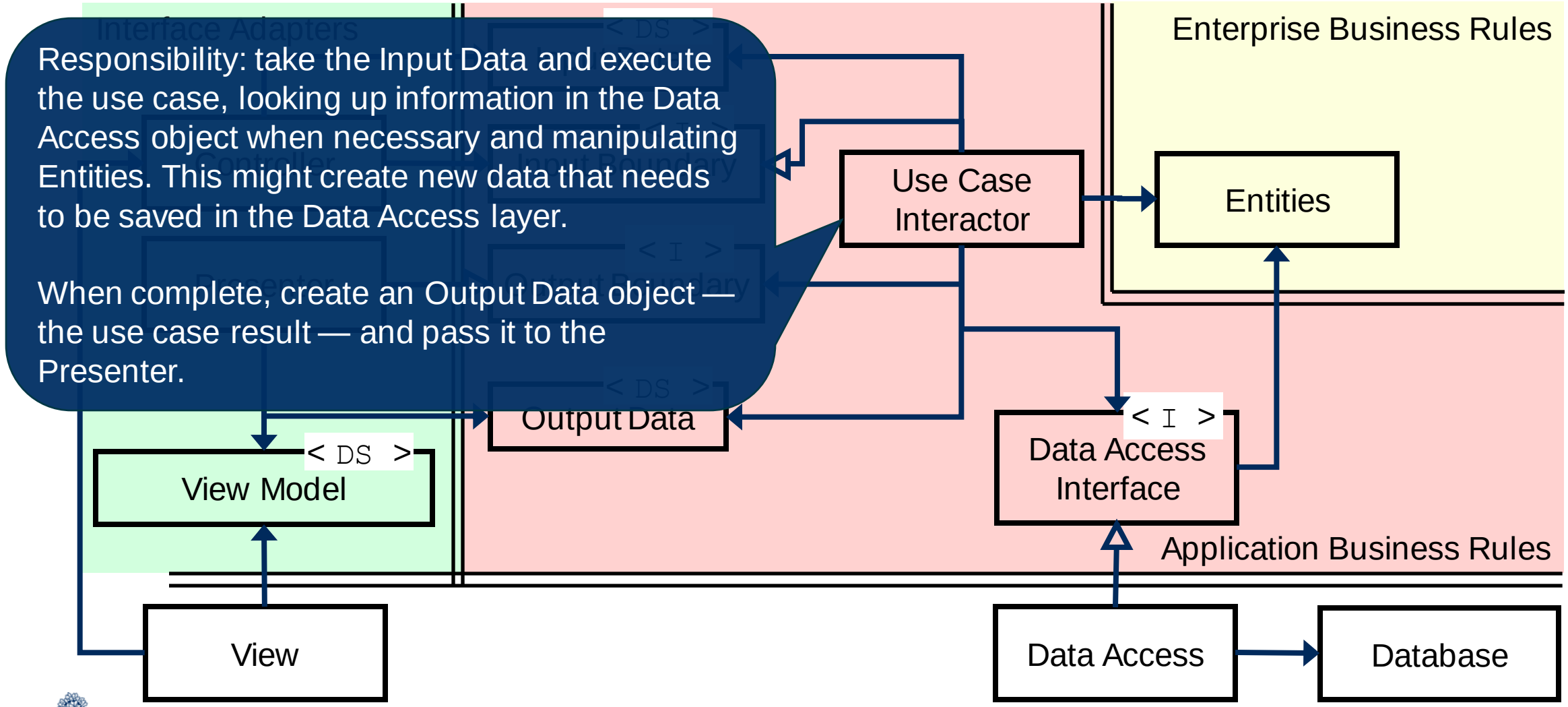
HOW THE ENGINE WORKS: THE VIEW



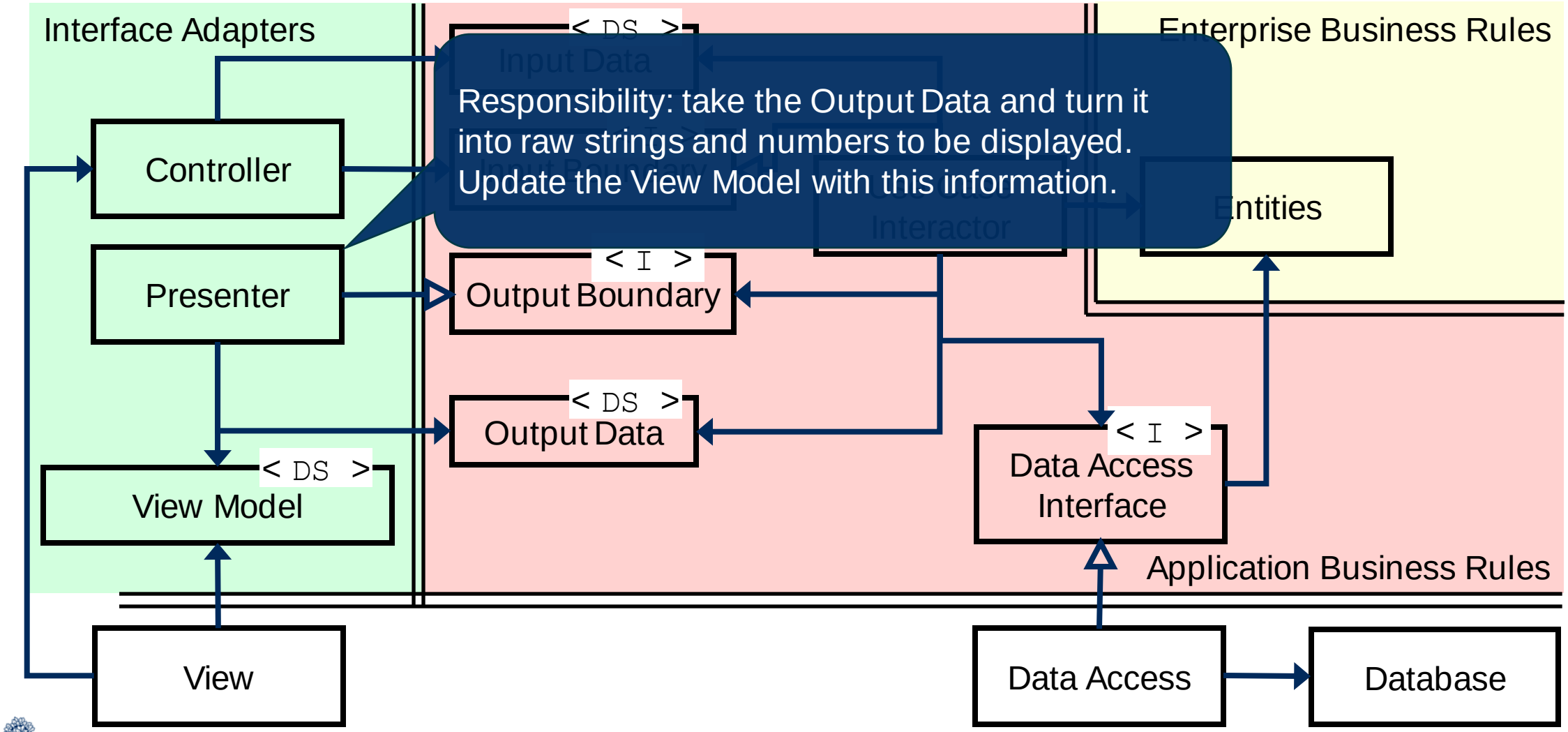
HOW THE ENGINE WORKS: THE CONTROLLER



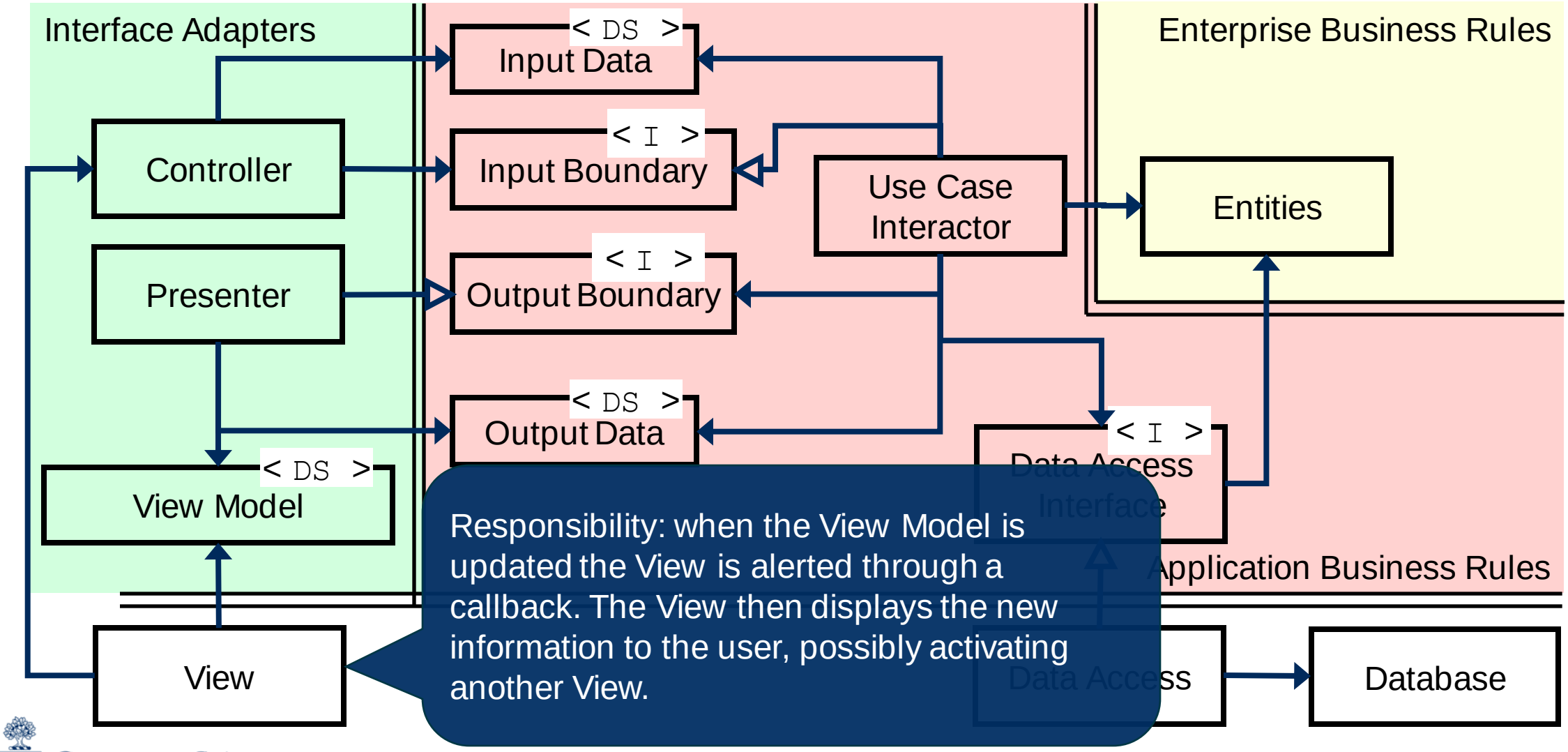
HOW THE ENGINE WORKS: THE INTERACTOR



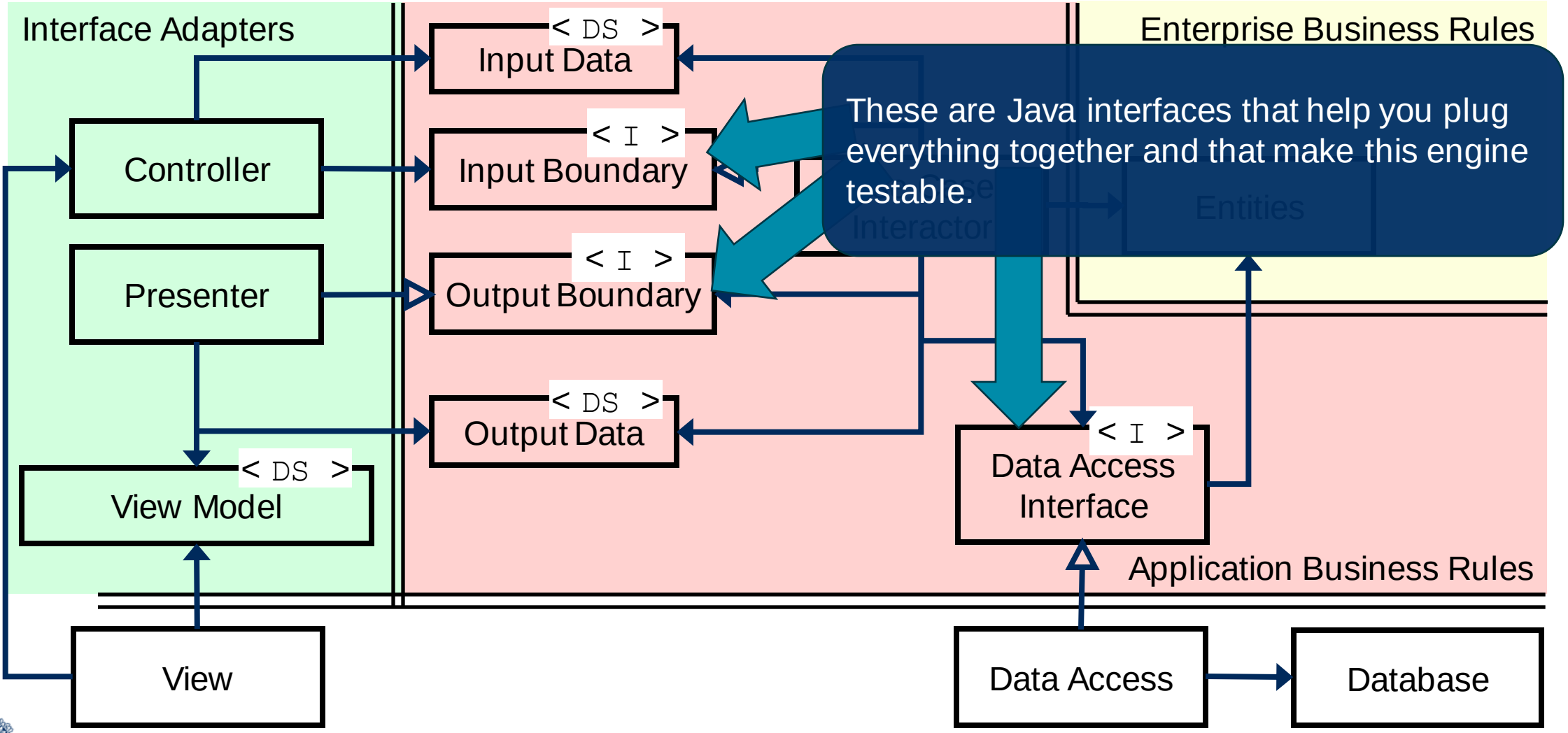
HOW THE ENGINE WORKS: THE PRESENTER



HOW THE ENGINE WORKS: THE VIEW (AGAIN)



HOW THE ENGINE WORKS: SPECIFYING THE INPUT AND OUTPUT PORTS



TERMINOLOGY (OH DEAR ME!)

- **Entity**: a basic bit of data that we're storing in our program (like a user with a username and password). Often called a "model" of the real world.
- **Factory**: an object that knows how to instantiate a class or a collection of interrelated classes.
- **Use case**: something a user wants to do with the program.
- **Interactor**: the object that responds to a user interaction, usually part of a use case (implements the input boundary)
- **Input boundary**: the public interface for calling the use case
- **Output boundary**: the public interface that the Interactor will call when the use case is complete
- **Data Access Object (DAO)**: involves persistence (a file or database). Often called a Gateway or a Repository. Reads data and creates Entities
- **Controller**: the object that the UI asks to run a use case
- **Presenter**: the object that tells the UI what to do when a use case finishes
- **Model**: a model of a concept from the problem domain. A collection of data representing a concept from the problem domain. Entities are often called models.