

# Layered Architecture

## *Software Architecture*

Richard Thomas

February 20, 2023

Ogres are like onions.

*Ogres have layers*, onions have layers...  
You get it? We both have layers.

- *Shrek*

*In the beginning...*

There was the big ball of mud *[Foote and Yoder, 1997]*



Figure: Image from "How to Avoid Spaghetti Code" [Gulsah, 2020].

*Problem*

Any change can affect any other part of the software.

*“Solution”*

# Modularity



---

<sup>1</sup>From <https://pixabay.com/illustrations/lego-building-game-toy-drawing-3388163/>.

## *Problem*

Lack of discipline lets any module communicate with any other module.



2

---

<sup>2</sup>From <https://pixabay.com/photos/lego-to-play-to-build-module-1629073/>.

*“Solution”*

# Layered Architecture





Figure: *Traditional* 4-tier, layered architecture.



Figure: *Traditional* 4-tier, layered architecture.



Figure: *Traditional* 4-tier, layered architecture.



Figure: *Traditional* 4-tier, layered architecture.



Figure: *Traditional* 4-tier, layered architecture.

*Question*

Can you identify an example of layered architecture?

*Question*

Can you identify an example of layered architecture?

*Answer*

Pick any website.

### *Definition 1.* Layer Isolation Principle

Layers should not depend on implementation details of another layer. Layers should only communicate through well defined interfaces (*contracts*).



*Definition 2.* Neighbour Communication Principle

Components can communicate across layers only through directly neighbouring layers.

*Definition 3.* Downward Dependency Principle

Higher-level layers depend on lower layers, but lower-level layers do not depend on higher layers.

### *Definition 4.* Upward Notification Principle

Lower layers communicate with higher layers using general interfaces, callbacks and/or events. Dependencies are minimised by not relying on specific details published in a higher layer's interface.

### *Definition 5.* Sidecar Spanning Principle

A sidecar layer contains interfaces that support complex communication between layers (e.g. design patterns like the observer pattern) or external services (e.g. a logging framework).

*Good architectural design...*

Applies these principles to deliver simple, modular designs that support modifiability.



**Figure:** J2EE layered architecture (from *Requirements Analysis and System Design* [Maciaszek, 2007]).



**Figure:** PCBMER layered architecture with sidecars (adapted from *Requirements Analysis and System Design* [\[Maciaszek, 2007\]](#)).

# PCBMER Layers

**Presentation** Displays bean data, implements UI logic, and updates beans.



# PCBMER Layers

**Presentation** Displays bean data, implements UI logic, and updates beans.

**Controller** Implements application specific logic and instantiates beans.

# PCBMER Layers

**Presentation** Displays bean data, implements UI logic, and updates beans.

**Controller** Implements application specific logic and instantiates beans.

**Bean** Data transfer objects used by the Presentation layer.

# PCBMER Layers

**Presentation** Displays bean data, implements UI logic, and updates beans.

**Controller** Implements application specific logic and instantiates beans.

**Bean** Data transfer objects used by the Presentation layer.

**Mediator** Manages business transactions, enforces business rules, instantiates business objects in the Entity layer, and manages the entity memory cache.

# PCBMER Layers

**Presentation** Displays bean data, implements UI logic, and updates beans.

**Controller** Implements application specific logic and instantiates beans.

**Bean** Data transfer objects used by the Presentation layer.

**Mediator** Manages business transactions, enforces business rules, instantiates business objects in the Entity layer, and manages the entity memory cache.

**Entity** Classes representing persistent business objects.

# PCBMER Layers

**Presentation** Displays bean data, implements UI logic, and updates beans.

**Controller** Implements application specific logic and instantiates beans.

**Bean** Data transfer objects used by the Presentation layer.

**Mediator** Manages business transactions, enforces business rules, instantiates business objects in the Entity layer, and manages the entity memory cache.

**Entity** Classes representing persistent business objects.

**Resource** Manages interactions with external persistent data sources.

## References

[Foote and Yoder, 1997] Foote, B. and Yoder, J. (1997).

Big ball of mud.

*Pattern languages of program design*, 4:654–692.

[Gulsah, 2020] Gulsah (2020).

How to avoid spaghetti code.

<https://tech.zensurance.com/posts/spaghetti-code>.

note = "Accessed: 2022-02-18".

[Maciaszek, 2007] Maciaszek, L. A. (2007).

*Requirements Analysis and System Design*.

Addison-Wesley Harlow, 3rd edition.