Software Architecture in Practice

Designing the foundations of the application

William Chan
Lead Platform Engineer, 605.tv, Capital One, FreeWheel (Comcast)

Objectives

- Learn what software architecture is
- Learn how to introduce software architecture to projects to keep the team aligned
- Learn what are the important elements of software architecture
- Review of design patterns

What is Software Architecture?

Architecture

Plan for constructing a building

Software Architecture

Plan for implementing software applications

Software Architecture Principles

- Language that helps team communicate better with the same terminology
- Structure for the application to align the team to implement features on the application

Software Architecture Concepts

1. Separation of concerns - the right components to divide the abstractions
2. Principle of least knowledge - the less knowledge an object have another object decreases coupling
3. DRY - don't repeat yourself, if something is repeated; caveat: be sure to make sure its also within the same domain

General Software Abstractions & Relations

- Encapsulation - helps decrease the amount of coupling between each component by information hiding
- Polymorphism - helps increase the cohesion of communication and relationship between objects
- Dependencies - dependent libraries and packages

Types of Coupling

- Afferent coupling - number of responsibilities; incoming
- Efferent coupling - number of dependencies; outgoing

Code Quality Measurements

- Instability - ratio of efferent coupling to afferent and efferent coupling
- Cyclomatic complexity - number of branches that a method may have, measures the complexity of methods

Cyclomatic Complexity

$E - N + 2 * P$

where
- E is the number of edges
- N is the number of nodes
- P is the number of nodes with exit paths

Principles & Patterns

Design Patterns

1. Creational patterns - patterns that determine how objects are created
2. Structural patterns - patterns that define relationships between objects
3. Behavioral patterns - patterns that define the communication between objects

Creational Design Patterns

- Abstract factory - request an object from a factory object
- Factory method - choosing an object implementation, creating the object and returning it
- Builder - builds complex objects that may have different representations
- Dependency injection - request an object from an injector
- Lazy initialization - create an object only when its needed
- Object pool - avoid expensive object creation and release of resources by recycling objects that are not in use
- Prototype - cloning an object to create a new instance
- Singleton - restrict instantiation from creating more than one object

Structural Design Patterns

- Adapter - adapts one interface to another
- Bridge - separating an interface from its implementation
- Composite - structure of different objects
- Decorator - adds additional functionality to its interface
- Facade - masks complex structural code eg. using many objects and executes different methods
- Flyweight - caching and storing objects with the same "value"
- Private class data - restrict accessor and mutator methods to access
- Proxy - connects two other components and may add more functionality

Behavioral Design Patterns

- Chain of responsibility - passes a request through a chain of objects
- Command - encapsulate information to trigger an event or action
- Interpreter - evaluation of grammatical representation
- Iterator - a way to access each element of a collection
- Mediator - coordinates between different objects
- Memento - restores an object to its previous state
- Null object - acts as the default value of a class
- Observer - a way of notifying other objects of an object's state change
- State - when an internal state changes, the object's behavior changes
- Strategy - determines the algorithm that should be used for the implementation

Behavioral Design Patterns Cont'd

- Template method - skeleton of an algorithm as an abstract class that allow subclasses to determine the behavior
- Visitor - separates the algorithm from the object it operates

SOLID Principles

1. Single responsibility principle - a class should only have one reason to change
2. Open-closed principle - classes should be open for extension but closed for modification
3. Liskov substution principle - objects should be replaceable by instances of their subtypes without altering the program
4. Interface segregation principle - no client should be forced to depend on methods, it doesn't use
5. Dependency inversion principle - modules should be dependent on abstractions and not details

Class Responsibilities

Sample responsibilities that a class can have

- Persistence - the process of saving data
- Validation - the process of validation data
- Error handling - handling errors gracefully or surfacing them
- Logging - runtime context of the application
- Class instantiation - creational design patterns
- Formatting - displaying relevant data
- Parsing - extracting information from data
- Mapping - translating data from one definition to another

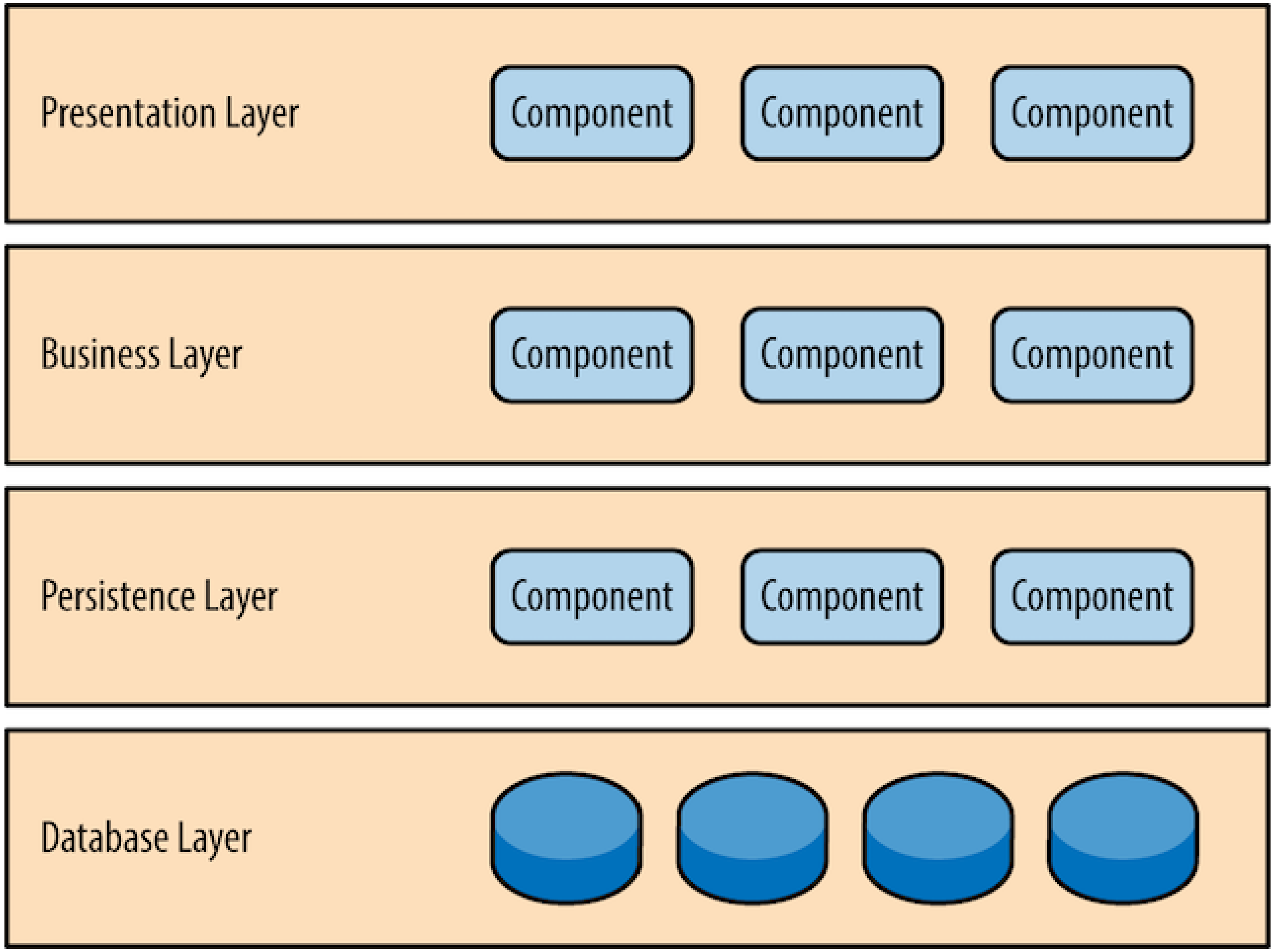
Architectures Classifications

Layered Architecture

Components are organized into horizontal layers

An example of a layered architecture is the MVC architecture

Layered Architecture Diagram Example



Source: https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch01.html

Model-View-Controller (MVC)

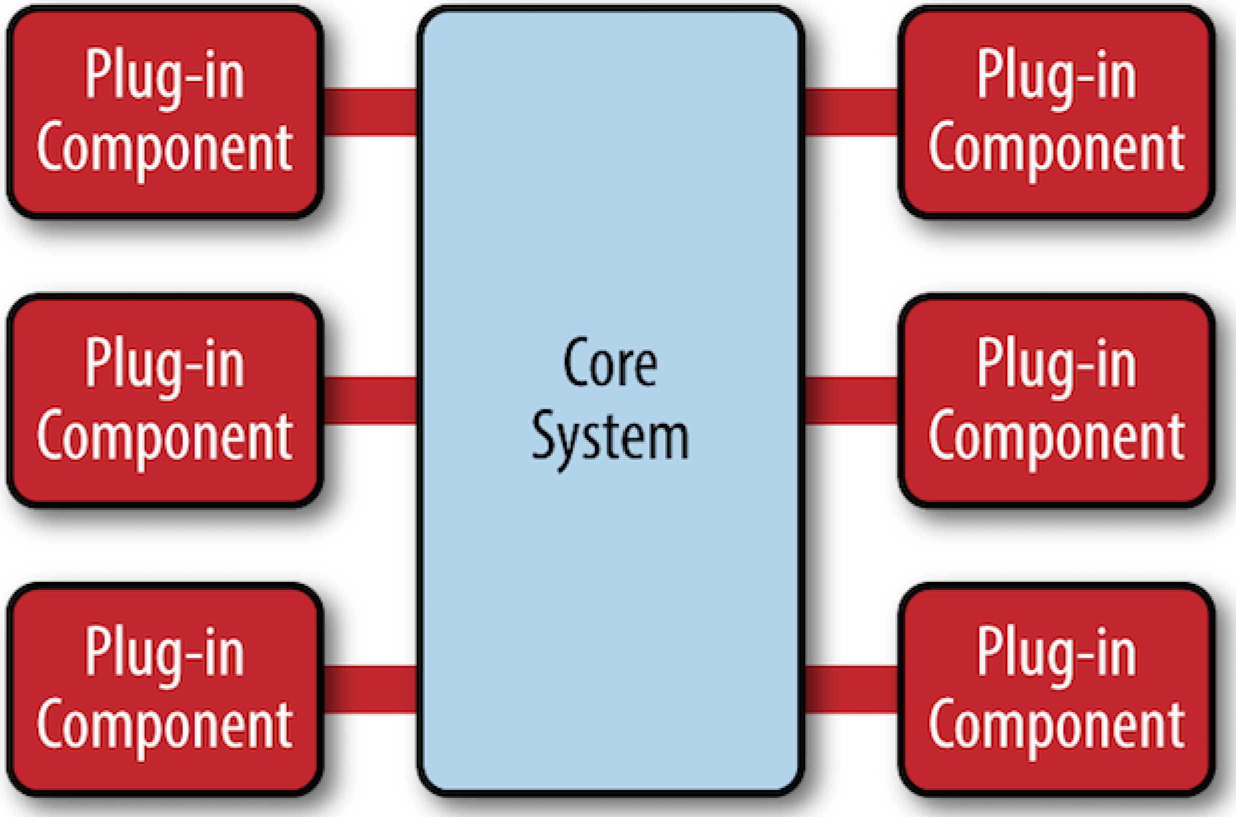
A way to build applications with a user interface

- Model - represents how data is persisted
- View - the presentational layer that the end user sees
- Controller - the mediator between the model and the view

Plugin Architecture

Components are organized as plugins where you can use different components in an extensible and reusable way

Plugin Architecture Diagram Example



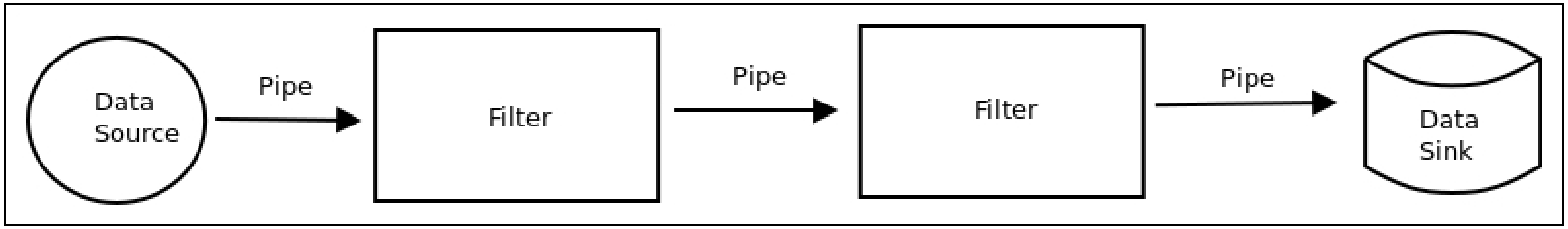
Source: https://www.safaribooksonline.com/library/view/software-architecture-patterns/9781491971437/ch03.html#idp1081312

Pipes and Filters Architecture

The main components of this architecture are pipes and filters, particularly with processing data

- Pipe - data flows from point to point
- Filters - transformations or filters that mutates the data that feeds into other pipes or sinks
- Sinks - where the data ultimately will be stored after a series of transformations

Pipes and Filters Architecture Diagram Example



Source: https://www.safaribooksonline.com/library/view/software-architecture-with/9781786468529/ch08s04.html

Mental Model for Applications

Domain Driven Design

- Helps classify components into logical categories to orchestrate certain behaviors
- Works well with modeling business applications

Domain Driven Design Objects

- Entity - objects with distinct identities and has a lifecycle
- Value Objects - immutable objects that has value but has no unique identity
- Domain Event - a record of something that happened in the software, can be used to publish to components that are interested
- Aggregate - cluster of objects that can be related and treated as a single unit
- Service - business logic operations that are stateless
- Repositories - deals with the entities and value objects lifecycle
- Factories - encapsulates creation of complex entities, aggregates and value objects

Functional Programming

Functional programming stresses the importance on mutability and therefore is natural fit for parallel and concurrent programming

Because state is often not kept in functions
- it has one less complexity to debug against, as you can always regenerate that state
- race conditions are kept to a minimal
- are great for pipeline architectures

The Process of Defining an Architecture

Understanding the Software Architecture Procedures

1. Understand - learn about the people and the process
2. Explore - explore the combination of components
3. Make - build prototypes and write documents
4. Evaluate - treat solutions as experiments and examine the risks

Software Architecture Quality Attributes

- Security - what security measures do we need?
- Privacy - what do we have to do to protect the information of our users?
- Portability - how deployable it is to our servers?
- Extensibility - how quickly can a module be added?
- Reliability - what is the expected uptime of the system?
- Performance - how quickly does our service need to respond to requests?
- Compatibility - what does our architecture have to be compatible with?

Note: *This is not a complete list of all architecture qualities, there are more.*

Trade-offs

Quality attributes are often at odds with each other so there are often trade-offs that need to be made

For example, if something is expected to be highly reliable, it may sacrifice performance because you need to store data in redundant locations

Releasing Software

Deployments

1. Configuration management to set up your servers
2. Virtual environments that servers can be deployed to
3. Containerization to isolate host dependencies from application dependencies

Logging

Logging helps debug the application when something goes wrong

Log Levels:
- Debug - granular diagnostic information for developers
- Info - information relevant to support staff to help figure out an error
- Warn - there might be a problem that warrants attention
- Error - log an error that has occurred
- Fatal - something critical happened that you have to halt your application

Monitoring

- Monitor the host such as CPU utilization, memory utilization, disk availability
- Monitor request traffic
- Monitor the amount of errors
- Monitor response times

Maintaining Software

- Technical debt - prioritize to see where technical debt is and refactor
- Making software debuggable - low coupling and logs help debug unintended behavior

Summary

- Software architecture organizes your application into logical categories to make them easier to navigate to add features, fix bugs, etc
- Software architecture is about trade-offs and choosing some quality attributes over others
- Software architecture and development is always an ongoing process. As more features are added, the structure of the architecture may change

Thank you

William Chan
Lead Platform Engineer, 605.tv, Capital One, FreeWheel (Comcast)
http://linkedin.com/in/wchan2 (http://linkedin.com/in/wchan2)