SE577: Software Architecture

Week 1: Course Overview

Dr. Brian Mitchell
CCI

Drexel University

About the Instructor

- Dr. Brian Mitchell
  - Professor in Computer Science
- Background
  - B.S., M.S., and Ph.D. in Computer Science
  - M.E. in Computer and Telecommunication Engineering
  - Involved with CS Department Teaching, Research and Industry collaboration since 1997
  - Ph.D. work in Software Architecture Recovery

- Contact Information
  - Email: bsm23@Drexel.edu
  - Office Hours: by appointment, standing office hours posted on Blackboard
  - Webpage: https://www.cs.drexel.edu/~bmitchell/

Teaching Assistant Information

- See Blackboard

- Office Hours: TBD

About this course

- Objectives
  - Teach students about the knowledge, duty, and skills needed by software architects
  - Teach students the essential skills of software architecture modeling and analysis
  - Teach students architecture patterns and styles
  - Teach students how to specify and analyze quality attributes
  - Teach students how to analyze architecture embodied in open source projects.
  - Teach students advanced topics in architectural design.

About this course

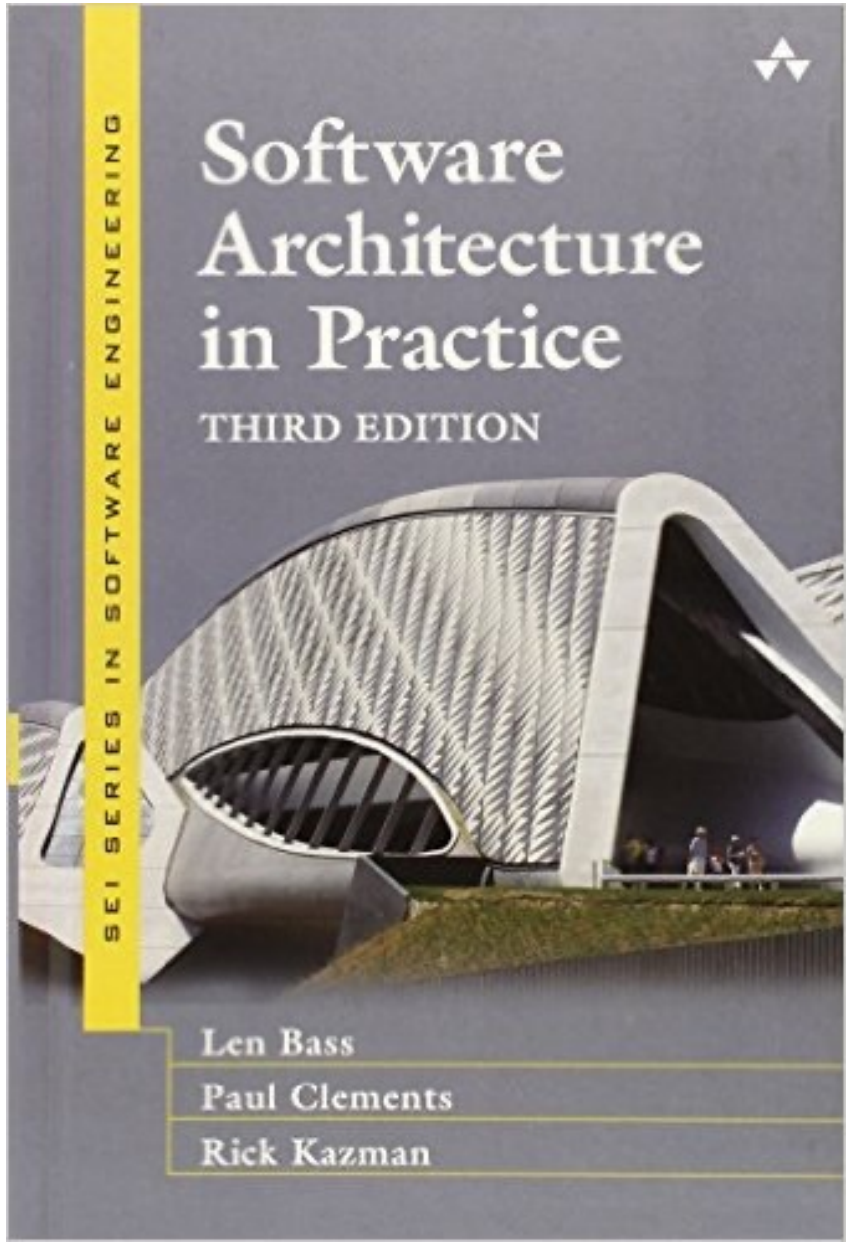
- Learning outcomes: students completing this course should be able to:
  - Apply architectural principles and skills to create software systems
  - Model software architecture and analyze various trade-offs.
  - Apply architectural patterns to solve real problems
  - Identify architectural styles in open source projects
  - Analyze software requirements and choose proper architecture styles.
  - Reason about architecture quality attributes.

Learning Material

- Reference Book 1:

  "*Software Architecture in Practice*"

  4th Edition, by Len Bass, Paul Clements, Rick Kazman

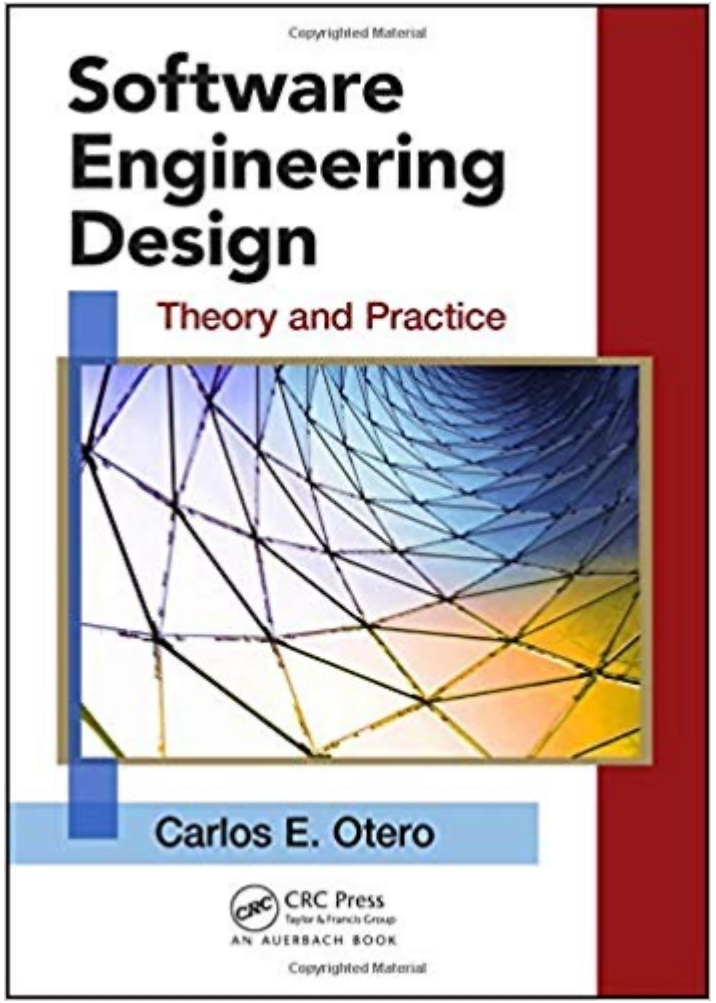


Learning Material

- Reference Book 2:

  “*Software Engineering Design: Theory and Practice*”

  First Edition, by Carlos Otero

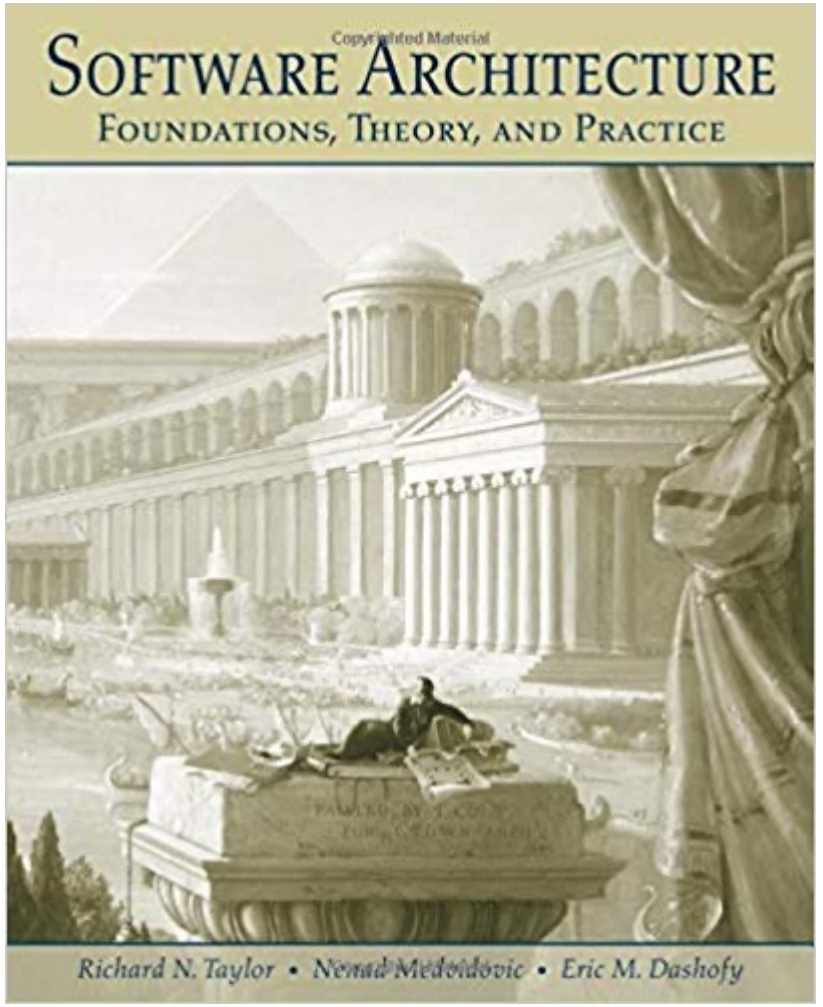


Learning Material

- Reference Book 3:

  "*Software Architecture: Foundations, Theory, and Practice*"

  First Edition, by R. N. Taylor, N. Medvidovic, E. M. Dashofy

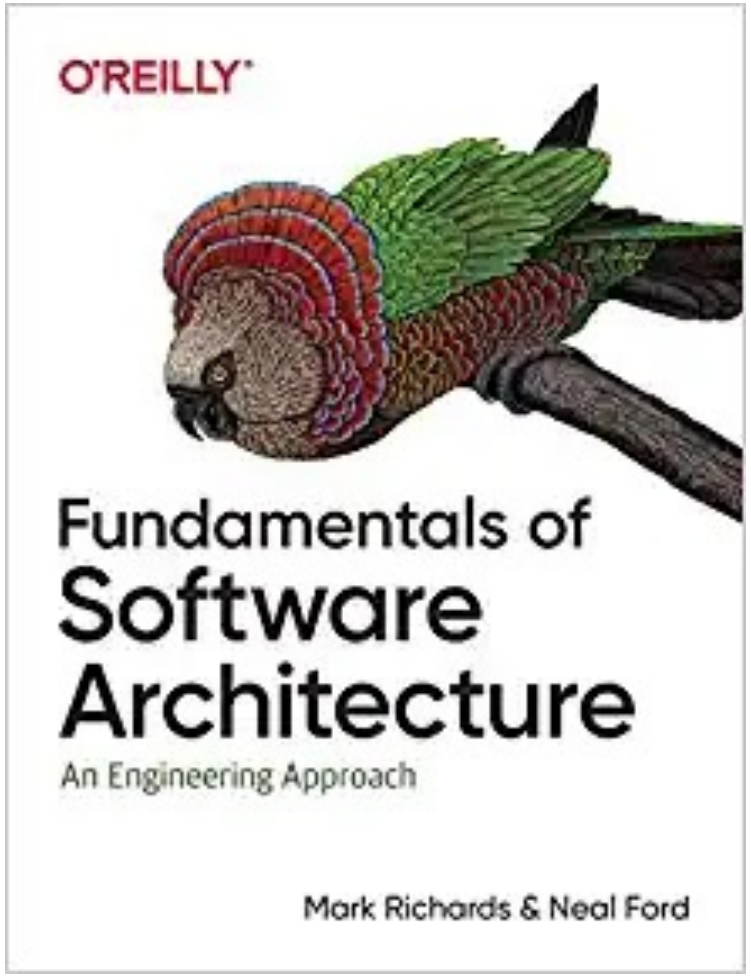


Learning Material

- Reference Book 4:

  “Fundamentals of Software Architecture”

  by Mark Richards, Neil Ford

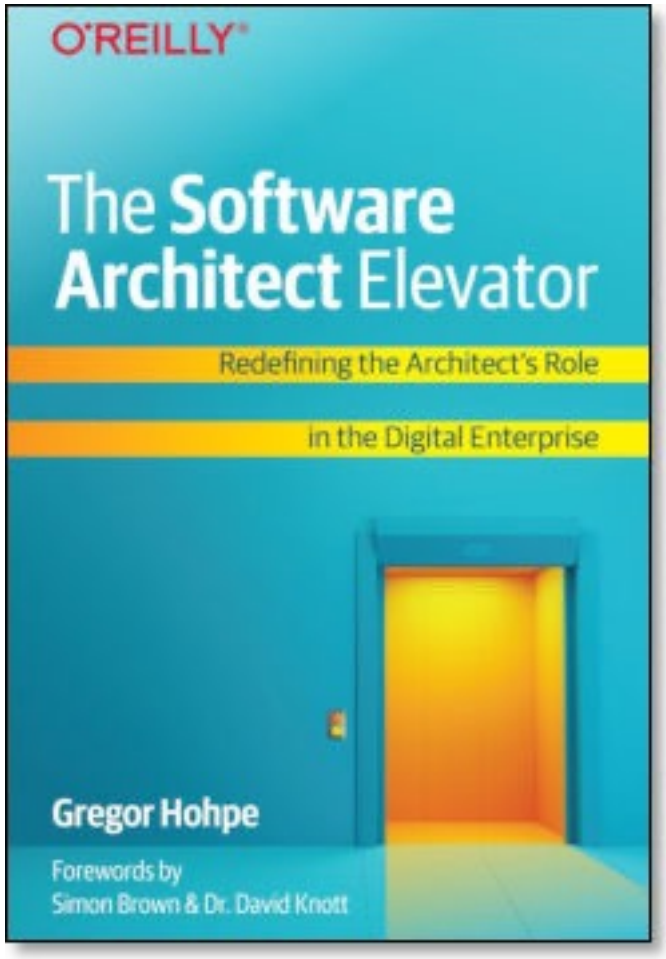


Learning Material

- Reference Book 5:

  "The Software Architect Elevator"

  by Gregor Hohpe



45 Minute Podcast Summary

https://podcasts.google.com/feed/aHR0cHM6Ly90aG91Z2h0d29ya3MubGlc3luLmNvbS9yc3M/episode/YzYzNjA2ZjYtYzVkZC00YTRmLTg2NDItNTU3YzExOWIzZWRm?hl=en&ved=2ahUKEwisvZnZ8u72AhULZd8KHXNQDpwQjrkEegQIAhAF&ep=6

Assignments, Assessments, and Evaluations

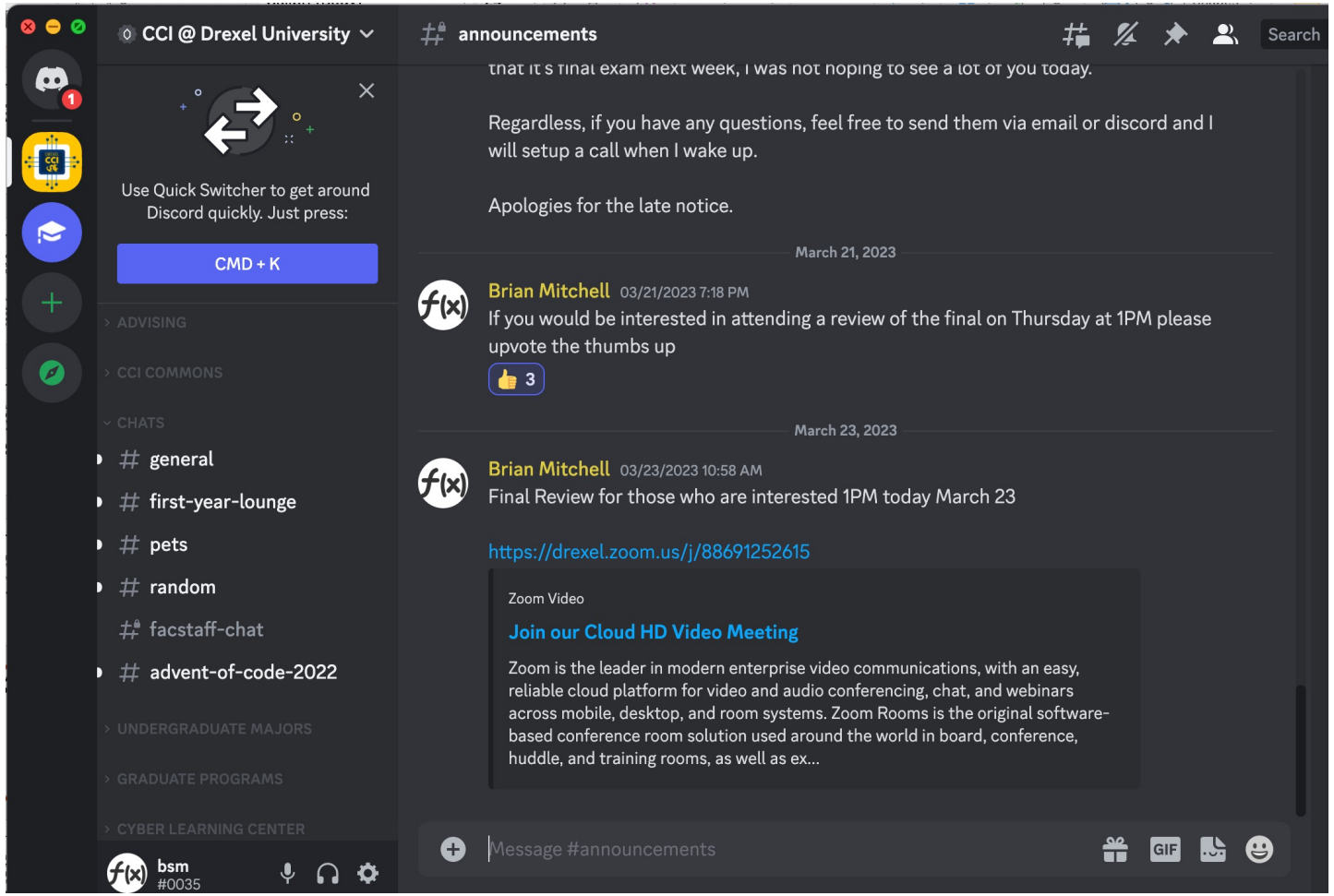
- Individual assignment with multiple deliverables:
  - Design, analyze, and implement software architecture
  - Apply architecture patterns to achieve extensibility
  - Reason about various architecture quality attributes

- Reading assignments, 2 foundational and 2 modern software architecture papers

Assignments, Assessments, and Evaluations

- Grading Component:
  - Course Reading summaries and other assignments 40%
  - Course Project 50%
  - Participation: 10%

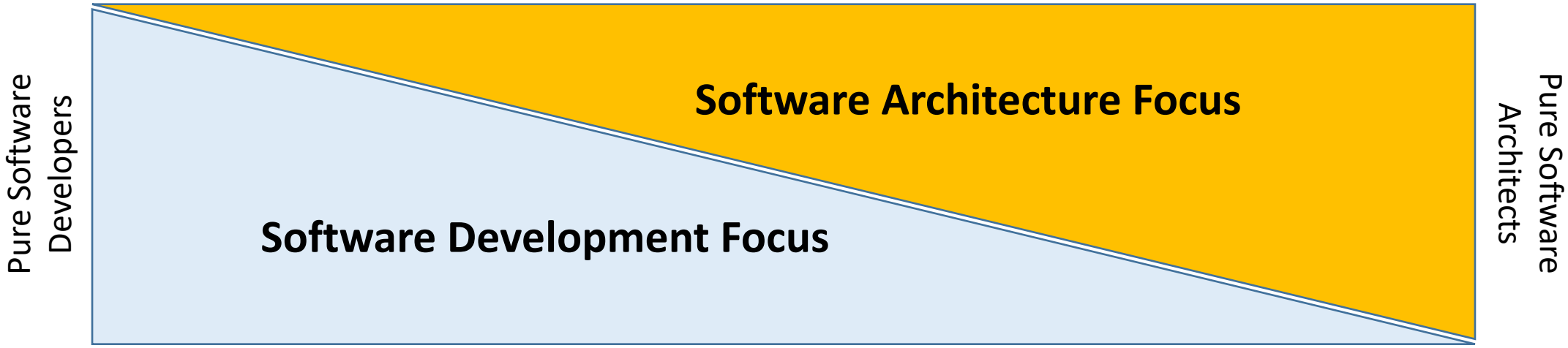
Discord will be our PRIMARY mode of communication



Please make sure you have it on your computer/phone

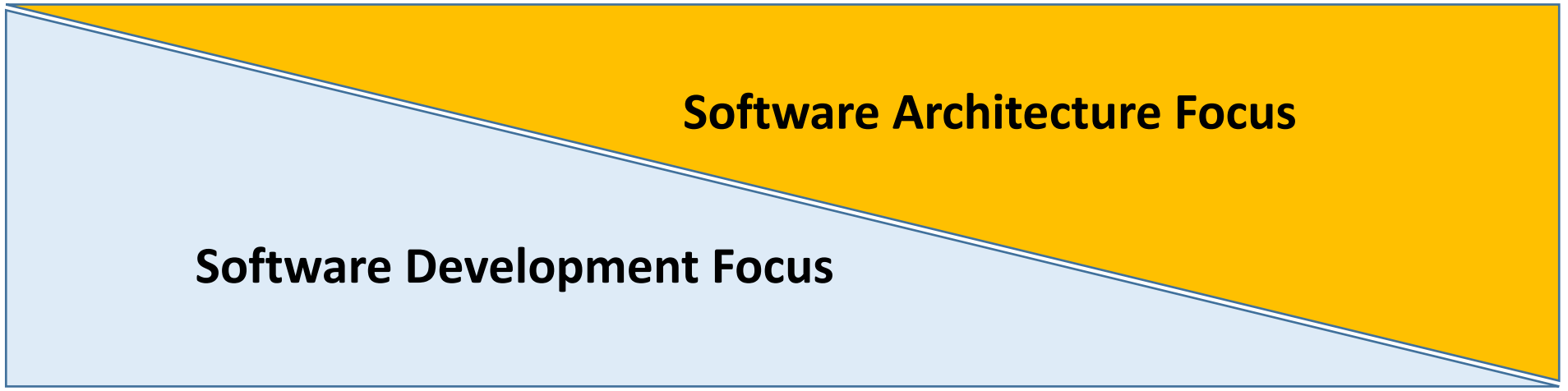
This course will focus on the technical and non-technical aspects of being a software architect

The titling and labeling in industry of software engineering roles is problematic…



Most modern software engineers will have a blend of software development and software engineering skills
Picking the right blend matters however

This course will focus on the technical and non-technical aspects of being a software architect



"All I Care About is the Tech" Zone

I focus on the technical architecture decisions and tradeoffs

"I am the master of nothing" zone – "Wikipedia Architecture"

I focus on the strategic product architecture decisions and tradeoffs

"Let's deploy this latest architecture I read about" zone

A word about the technologies we will be using

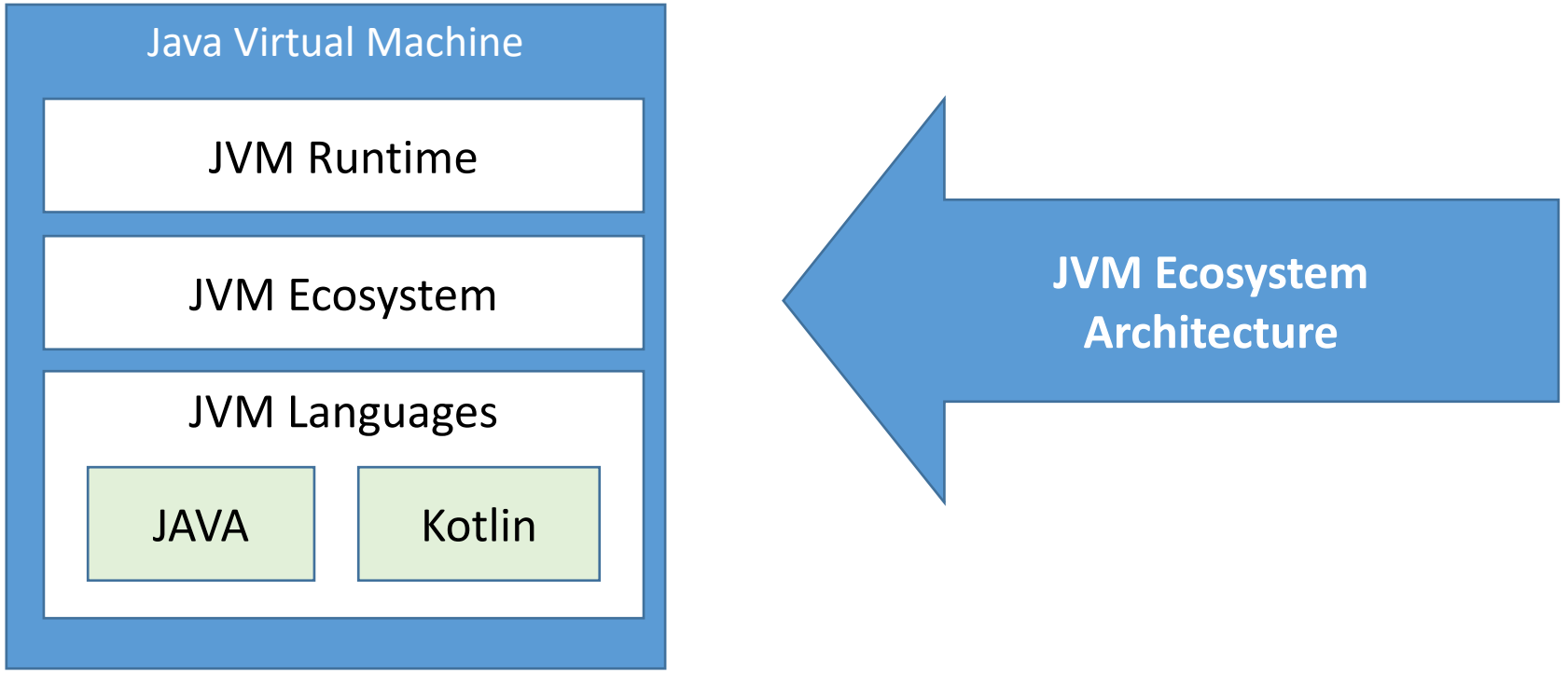
- We will be using modern technologies to demonstrate modern software architecture toolchains and implementation patterns
- I will allow students to program in Java, but I will not be using Java in this class
- My views about why its time to move away from Java for modern software engineering

LETS USE THIS AS AN EXAMPLE DEMONSTRATING WHAT AN ARCHITECT DOES AND WHAT YOU SHOULD BE ABLE TO DO AFTER TAKING THIS CLASS

Our first software architecture example

Architecture Guidance: For most new modern development the overall JVM ecosystem should be avoided, if the criteria for acceptable use of the JVM ecosystem are met, then:

Existing Java code bases can continue to be developed and extended in Java – significant new development or enhancements to these systems should consider moving over to Kotlin.



Lets examine this from an architects viewpoint – We will learn later that this can be articulated as an architecture decision record (ADR)

First Architecture Example: Lets analyze the Java

Java first appeared in 1995

Objective: Improve on scale, simplicity, portability over things like C/C++

Obsessive focus on Backwards Compatibility

Implementing “best of class” support for object orientation

The Java Virtual Machine is one of the most impressive pieces of software ever developed

Provides an isolated sandbox for running applications

Provides a portable runtime for multiple architectures

Over the years has achieved near native performance

ON THE SURFACE PRETTY AWESOME – WHAT DO YOU THINK?

First Architecture Example: Lets analyze the Java - II

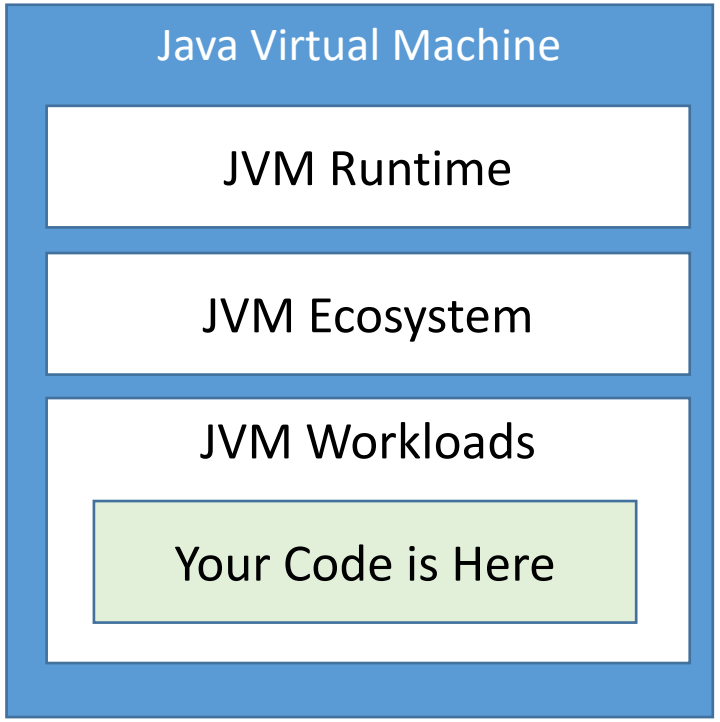
We will be learning that architecture is largely about tradeoffs

Feature	Implication
Backwards Compatibility	GREAT: Code written today or 20 years ago largely compiles as is GREAT: JVM enhancements over the years enables Java to run at speeds comparable with any modern programming language ISSUE: Early Java language decisions based on the state of software engineering and object orientation in the 1990s have been shown to be suboptimal – but we are still stuck with them. **Implications – Type erasure at compile time for generics, improper lambdas/closure support, awkward functional programing extensions, inconsistent views of object vs native types**
Language Syntax	Great 30 years ago when compared to other languages at the time But lacking many of the things that have proven to be useful in modern programming languages: awkward type inference, immutability not a first-class concern, extremely verbose
Developer familiarity	When Java rolled out, standardizing around a language made a lot of sense. Although Java has been "free" since the get go, a large ecosystem of expensive commercial tools were centered around Java-based systems – the infamous Java Application Server – if you are investing millions on a Java based commercial runtime, you need Java developers/engineers
Portability	In the early days you wanted to develop on one OS but deploy to another OS – Java had excellent support for this, but this is no longer a "big deal"

Arguing Recommendation 1:
Major refactoring of Java apps should move to Kotlin

- Kotlin is significantly less verbose than java
  - Less code = less bugs
  - Less code = more productivity
- Kotlin embraces modern language features
  - Better support for distributed systems, which are most systems these days
- Kotlin is interoperable and easy to learn
  - Simple to start integrating kotlin into existing projects as well as starting new projects with kotlin
- Kotlin is popular
  - Kotlin's killer use case was Android development, but has now expanded as a popular choice with traditional Java developers

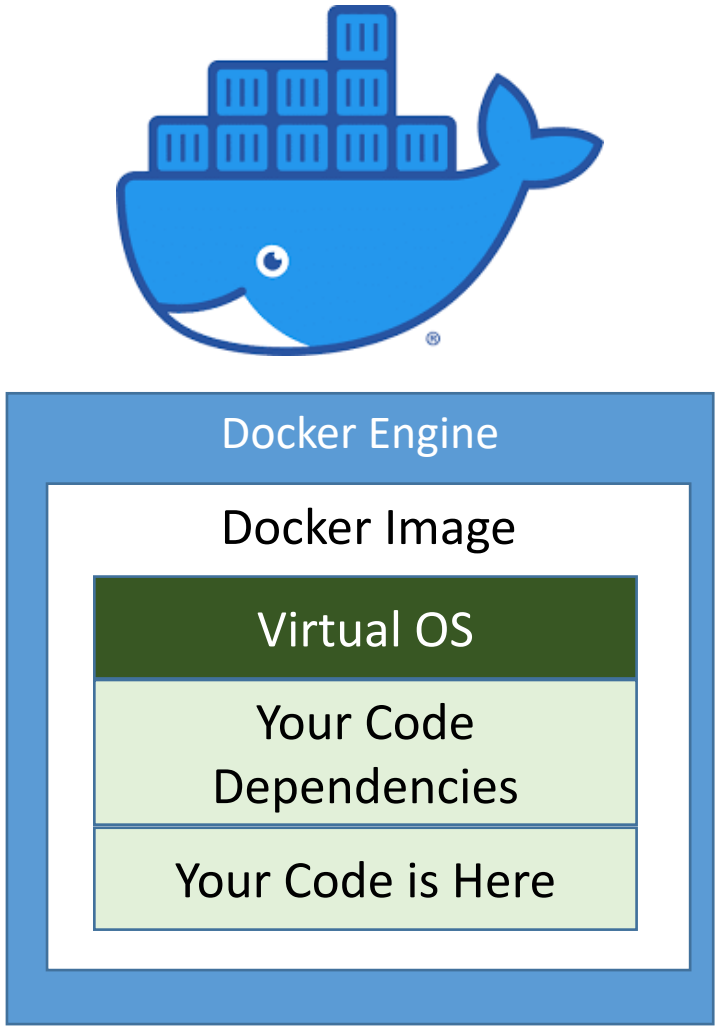
Arguing Recommendation 2
Avoid the JVM for largely Net New efforts, Part 1



Overall JVM Architecture

- JVM Benefits
  - Provides isolated environment for executing your code
  - Provides runtime management for your executing code
  - Provides multi-threaded, and multi-process abstractions on top of OS
  - Portable across Operating systems
- JVM Limitations
  - Cold start times because the JVM adaptively optimizes
  - Only supports JVM-based languages – Java, Kotlin, etc

Arguing Recommendation 2
Avoid the JVM for largely Net New efforts, Part 2

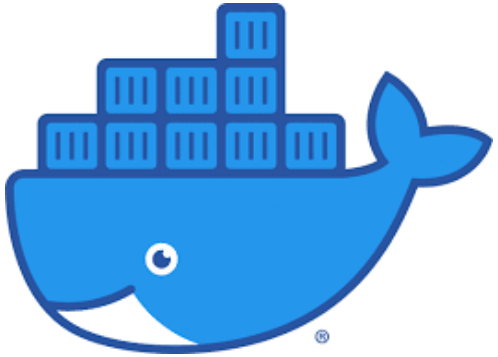


Overall Docker Architecture

- Docker Benefits
  - Provides isolated environment for executing your code
  - Provides runtime management for your executing code
  - Provides multi-threaded, and multi-process abstractions on top of OS
  - Portable across Operating systems
  - Language, framework agnostic
  - Can be composed with other containers
  - Light footprint – e.g, Alpine Linux base image is 5Mb.
  - Has all of the benefits of the JVM and then some

Unrelated – some are speculating that server-side webassembly will be the "next thing" to take over from docker

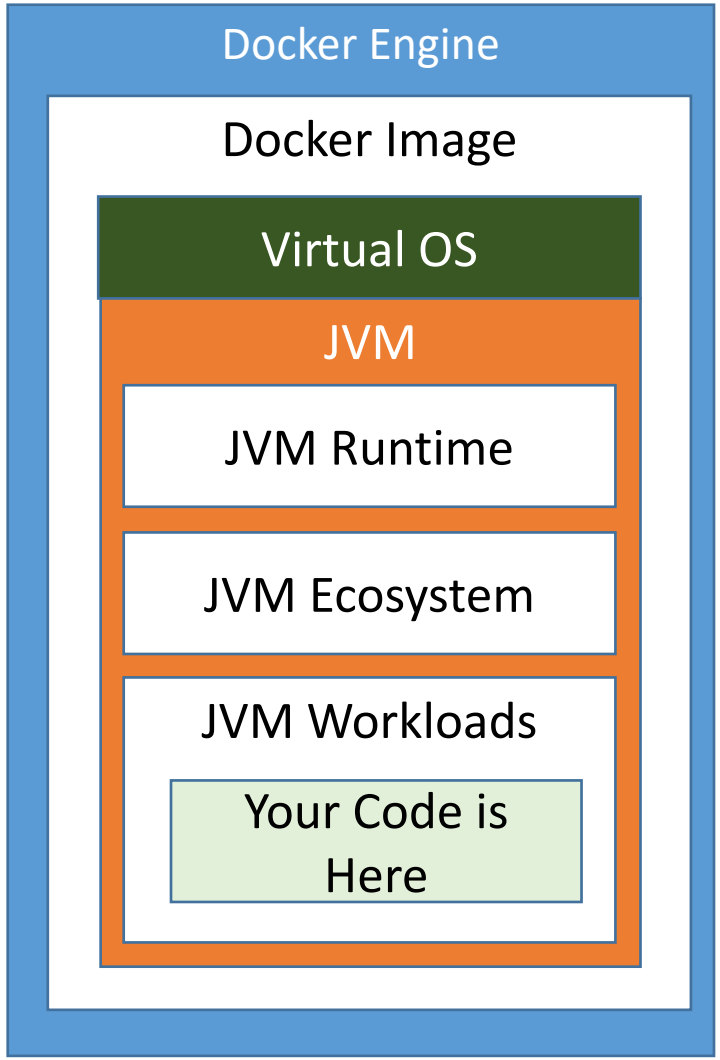
Arguing Recommendation 2
Avoid the JVM for largely Net New efforts, Part 3



Docker OK

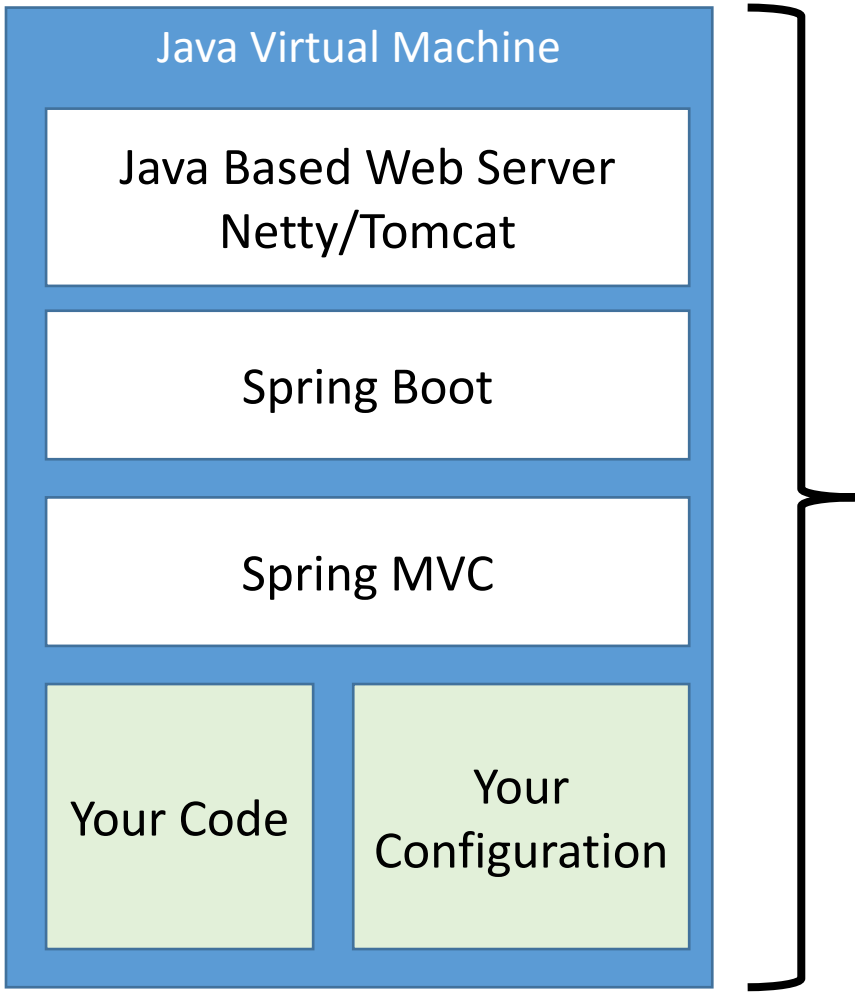
JVM OK

Thoughts on Embedding the JVM in a Docker Container?

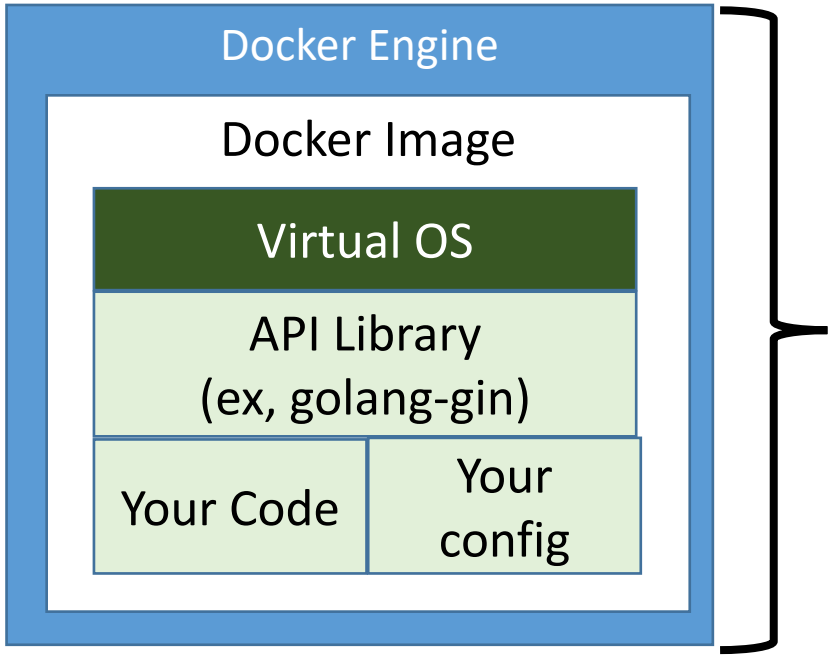


First Architecture Example: Lets analyze the Java - II

Lets look at a "Layered" Architecture that's popular these days for web services



Pretty typical memory footprint is 1G minimum, most is needed for the JVM vs your code



Pretty typical memory footprint is 30-50Mb

Not a big deal if you have relatively few services, but most organizations these days have thousands of services. Plus in the cloud you pay for memory

Architecture Rationale, Guidance Recommendations I

- Javas key strength is backwards compatibility, but its design is old and results in code-bloat that can largely be mitigated with newer JVM languages like Kotlin
- Kotlin drives higher productivity and less errors (write less boilerplate), but also embraces modern language features that lend themselves to better support of modern distributed applications
- Since Kotlin interoperates well with existing Java codebases, significant enhancements of existing Java codebases should consider Kotlin
- New JVM development should STRONGLY consider Kotlin

Architecture Rationale, Guidance Recommendations II

- For more modern workloads, containerizing in docker is a better choice than running in the JVM, especially for cloud native architectures, or public cloud deployments
- While containerizing JVM based architectures is possible, it should be avoided due to the heavy JVM footprint coupled by all of the JVM benefits being mitigated by docker
- Additionally, the memory and compute footprint of Dockerized JVM solutions is large, and these resources cost money, especially in the public cloud
- **Recommendation for new development - Leverage a modern programming framework such as go or typescript, which deploy naturally to both dockerized and serverless runtimes.**