SOFTWARE ARCHITECTURE PART 1

Code It Up Online Vol. 9

FOR LIVE QUESTIONS

Check The Link In The Description

Resources

https://github.com/ivaylokenov/Software-Architecture-Series

LIVE STREAM TROUBLESHOOTING

- Sometimes issues happen during a live stream…
  - And sometimes disasters happen, this is how memes are born…
- If my Internet goes down and the stream stops:
  - Wait for 5 minutes, I have another one on a different network
- If YouTube is showing "stream ended":
  - I will post a new link in the comments section below this video
- If something else happens unexpectedly:
  - Well, I will add a solution to this slide during my next event… ☹
- If a major showstopper is happening – no electricity, for example
  - I will create a new event and we will schedule a new live stream…
- In any case – write to wewritesoftware@gmail.com

THE PRESENTER

- **Ivaylo Kenov – Quality Code Advocate**
  - Various job titles at the same time:
    - Organizer & Speaker @ Code It Up
    - CTO @ SoftUni
    - Full Stack Technical Trainer @ Everywhere
    - Code General @ https://docs.mytestedasp.net/
    - Meme Copy Machine @ Daily Programming Fun
    - {*Insert Job Title Here*}
  - Contacts
    - https://github.com/ivaylokenov
    - https://facebook.com/ivaylo.kenov
    - https://linkedin.com/in/kenov
    - https://www.instagram.com/ivaylokenov/
  - YouTube & Blog
    - https://www.youtube.com/c/CodeItUpWithIvo
    - https://codeitup.today/

SPONSORS

- This lecture is free thanks to our sponsors
  - Which will interrupt the lecture here and there
  - Because it takes quite a lot of personal time to prepare the materials
- You will help the initiative a lot if you visit their web sites
  - And consider their propositions to you
- I personally select various premium jobs to present them during the talk
- These are the current ones:
  - INDEAVR - https://indeavr.com
  - Americaneagle.com - https://www.americaneagle.com
  - SmartIT - https://smartit.bg

ABOUT CODE IT UP

CODE IT UP

- The Code It Up initiative:
  - Aims to provide detailed knowledge on advanced software development topics
  - Aimed at people with at least 1 year of programming experience (mostly C#)
  - Sort of acquired by SoftUni last year
- Code It Up Online
  - Free live-streamed online events (2+ hours long)
  - Led by me – mainly .NET, architecture, and infrastructure
  - We have 7 more lectures before the initiative ends
- Code It Up Workshop
  - Paid events containing theory & practical exercises for the attendees
  - No more paid lecture planned for now

THANKFUL IF YOU SHARE A STORY

- You can be extremely helpful to the initiative
- Just share a story on Facebook or Instagram during the lecture
- Make sure you tag me so that I can reshare your post - **@ivaylokenov**
- Bonus – add the **#codeitup** hashtag
- Thank you! You rock!

ABOUT THE SERIES OF EVENTS

ABOUT THIS SERIES OF CODE IT UP EVENTS

- Theoretical lectures on software architectures
  - Widely-used design patterns and concepts in production
  - Depending on your level, you may be familiar with some of the topics
  - A real-life project example
- A practical guidebook for architecting various solutions
  - 3 real-life projects on more than 70 pages
  - Legacy systems, vast data load, lots of concurrent users, working with critical data, and more
  - Please report to wewritesoftware@gmail.com, if you find any "bugs" in the book!
- And I have a lot more to add in the future!
  - You receive free updates of the book!
- MOST IMPORTANTLY – HUGE THANK YOU! <3

THE CONTENT OF THE SERIES – A FREE COURSE

- Why Software Architecture
- What Is Software Architecture?
- Unified Modeling Language
- Designing Solution Architectures
- Common Technology Stacks
- Architecture Design Patterns
- Choosing The Right Patterns
- Architecture Quality Attributes
- System-Wide Considerations
- Deployment Considerations
- Monolithic Architecture
- Domain-Driven Design
- Microservices
- Event Sourcing
- The Architecture Document
- The Architect And The Team
- What Makes A Great Architect
- Designing A Real-Life Solution

IN THIS PART

- Why Software Architecture
- What Is Software Architecture?
- Unified Modeling Language
- Designing Solution Architectures

- Don't Forget The Optional But Practical Guide-Book
  - 3 Real-World Scenarios
  - 70+ Pages
  - Free Updates
  - Get It From The Event's Page or write to wewritesoftware@gmail.com
    - https://www.eventbrite.com/e/software-architecture-fundamentals-essentials-code-it-up-online-vol-9-registration-222550182587

ABOUT THIS TOPIC

- HUGE DISCLAIMER! THIS TOPIC IS LIKE THE LITTLE PRINCE!
- **There are a lot of things to know about software architectures**
  - **You may or may not recognize some of the patterns**
  - **And we most probably will not mention all of them**
  - **If you are a beginner, just try to absorb what you can**
- The technology world is moving very fast
  - Some of the examples shown here may be considered anti-patterns in the future
  - But the overall concept and process stays the same
- As all my other topics – this one is super intense too!
  - Even though the lectures will be a bit shorter
  - So, give yourself time and if you get the book – finish it!
  - And don't worry! The knowledge provided here will save you weeks of reading!

INDEAVR – THE EVENT'S DIAMOND SPONSOR

- They provide technology services focused on the Digital, Data, Cloud and Advanced Software Engineering expertise.
- They are always in search for creative and passionate people with the combination of a sharp strategic mind, emotional maturity, entrepreneurial instincts, and the ability to deliver results.
- https://www.indeavr.com/en/technology/application-services
- https://www.indeavr.com/en/careers

INDEAVR
Serving the high achievers

WHY SOFTWARE ARCHITECTURE

WHY THIS TOPIC

- A little back-story, first…
- 5-6 years ago, I was on interview for a tech lead or a software architect
  - Depending on my skills and knowledge
- And after the interview, they told me "you are great, but you need a little bit more to become an architect"
  - I was both disappointed and motivated
- So, I started researching a lot about software architectures and patterns
  - I purchased lots of courses, read lots of articles and books
  - Started analyzing how the big guys are doing it – StackOverflow or Netflix, for example
- These lectures are the culmination of the story above! Enjoy!

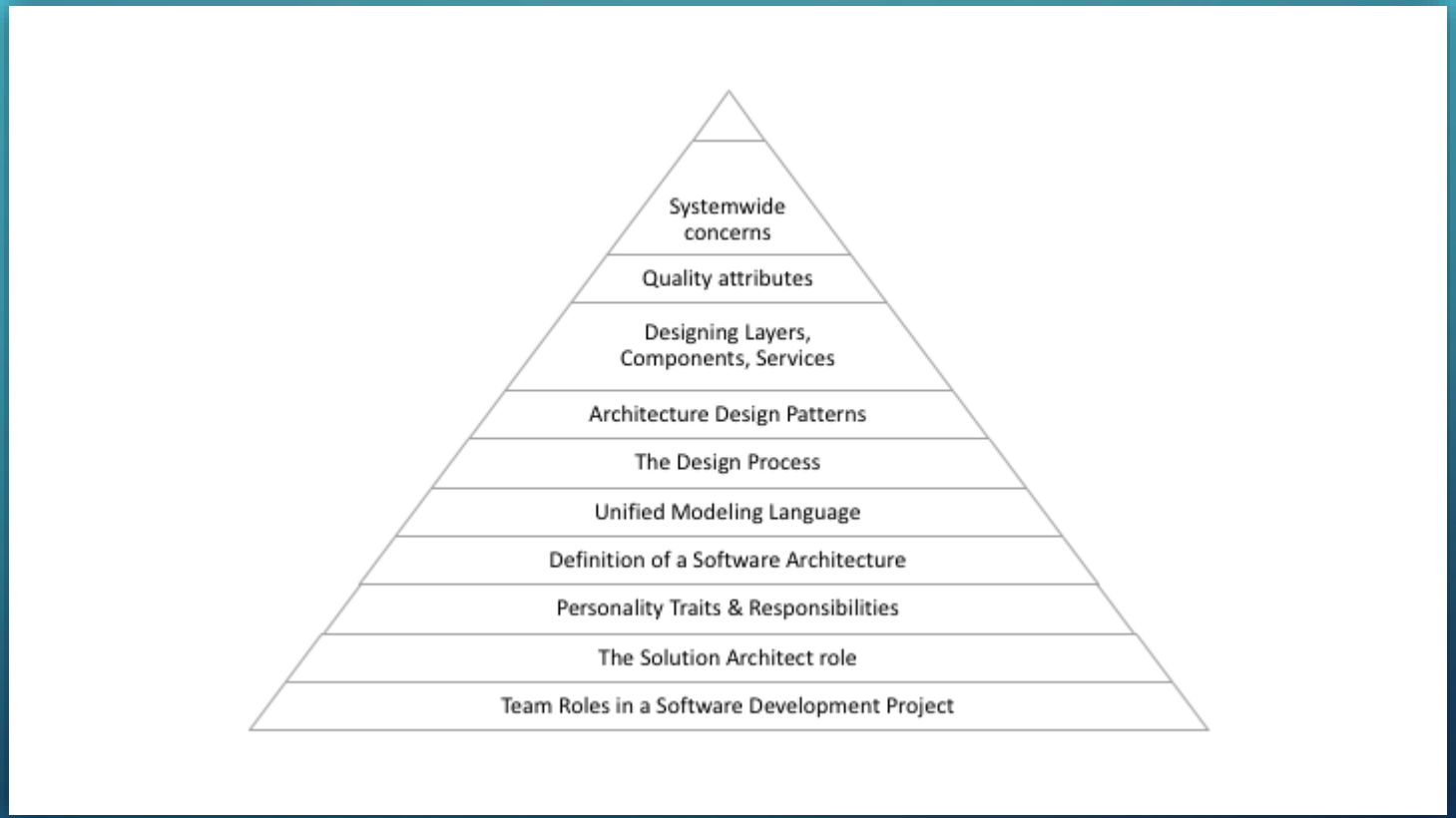
WHY SOLUTION ARCHITECTS

- Why do we need solution architects?
  - If we have a perfect tech lead?
  - And very skilled developers?
- Consider constructing a building…
  - Imagine if there is no design and architecting plans
  - Just the workers and their team leader coming every day to work
- It is the same with software development!
  - We need an architect to design and lead our high-level solution!

REASONS TO BECOME ARCHITECTS

- It is interesting!
  - You will have way more variety of work to do!
  - You will solve different challenges!
- Career path and visibility!
  - You will meet powerful people – CEOs and CTOs!
  - You learn a lot from such individuals!
- Money!
  - Software architects may earn double the salary of senior developers!
  - And if they are skilled enough, they can beat some of the executives!

THE KNOWLEDGE PATH



WHAT IS SOFTWARE ARCHITECTURE?

SOFTWARE ARCHITECTURE

- Structured solution to address all common software attributes:
  - Meets all the requirements
  - Performance optimized
  - Security in check
  - Manageability for developers
  - Scalability for business growth
  - Accessibility for easier user experience
- Major requirements:
  - User requirements – the way end-users interact with the system
  - Business requirements – cheaper, faster, better than competitors
  - IT system requirements – infrastructure requirements

WHAT COMPOSES AN ARCHITECTURE?

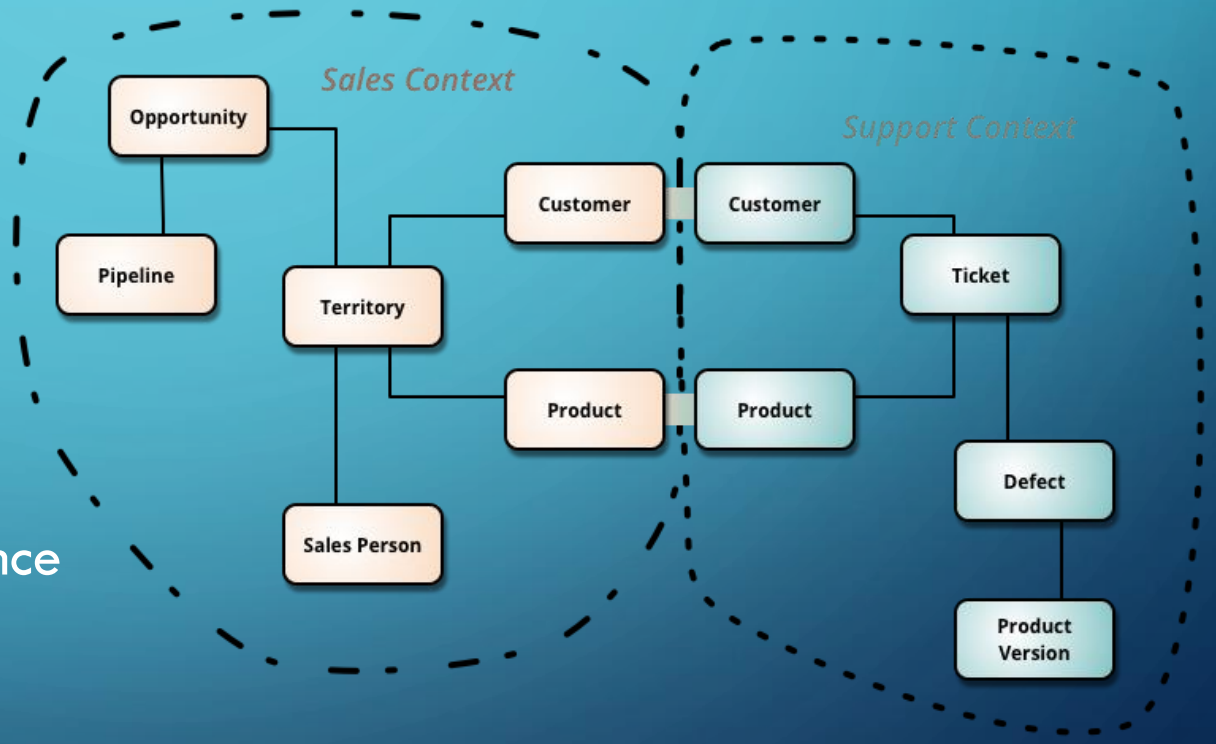
- The structural elements and interfaces composing the system
  - Objects – the low-level building blocks of the system
- How these elements behave in collaboration
  - How communication is done in the architecture
- The composition of elements into larger subsystems
  - How the elements are mapped to trees or graphs – the high-level data structure
- The architectural style that guides this composition

WHAT COMPOSES AN ARCHITECTURE?

- Additionally, our design should cover:
  - Functionality
  - Usability
  - Resilience
  - Performance
  - Economic and technology constraints
  - Trade-offs and aesthetic concerns
- The goal is to:
  - Document high-level structure
  - Do not go into implementation details
  - Minimize complexity
  - Address all requirements
  - Be compatible with all use cases and scenarios

OUR ARCHITECTURE NEEDS

- Separation of Concerns
- Encapsulation
- Dependency Inversion
- Explicit Components
- Single Responsibility
- Don't Repeat Yourself
- Persistence & Infrastructure Ignorance
- Presentation Ignorance
- Bounded Contexts
- Testability



ARCHITECTURE ABSTRACTION

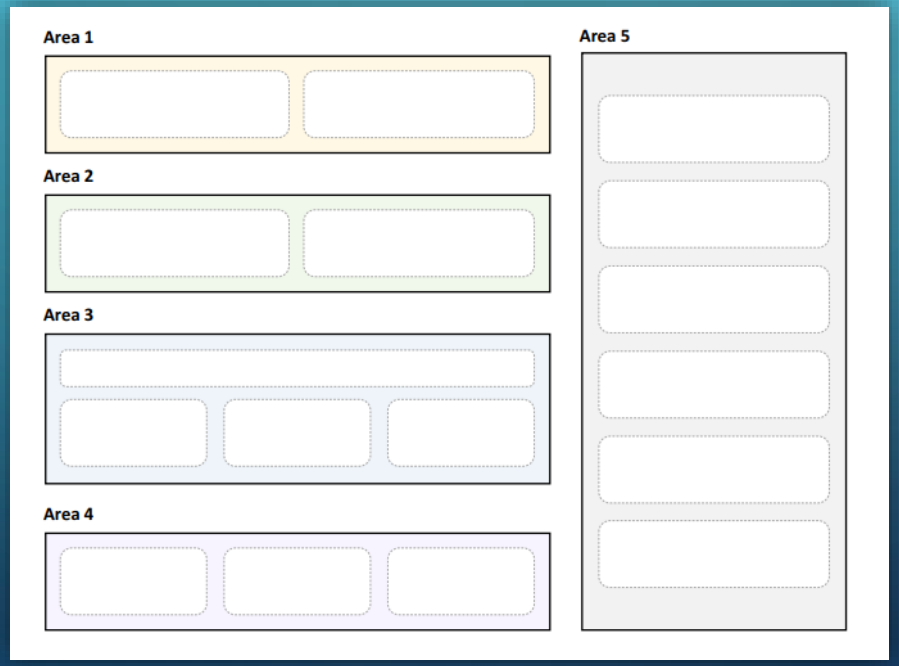
- Layers:
  - System, Sub-systems, Layers, Components, Classes, Data and Methods
- Bad Architecture:
  - Complex, Incoherent, Brittle, Untestable, Unmaintainable
- Good Architecture:
  - Simple, Understandable, Flexible, Testable, Maintainable

SOFTWARE ARCHITECTURE DESIGN TIPS

- Build to change instead of building to last
  - You are going to change your solution multiple times
  - Create around modularity and flexibility
- Use models, but only to analyze and reduce risk
  - The models should be more abstract, otherwise you are taking the role of a developer
- Use visualizations to communicate and collaborate
  - Your architecture is going to be a large diagram
- Identify and research critical points of failure
  - Put in work in research in everything advanced
  - Security, scalability, resiliency

AREAS OF SOFTWARE ARCHITECTURES

- The goal of a software architect is to minimize complexity
- This can be accomplished by separating the design into different areas of concern
- Also known as modules (or components)



KEY PRINCIPLES OF SOFTWARE ARCHITECTURES

- Separation of concerns
  - Each object and module should be in its own concern and context
- Single responsibility principle
  - Elements in our design must have a single purpose
- Principle of least knowledge
  - Components do not know about the internals of other components
  - Done through interfaces
- Don’t repeat yourself
  - Don't have multiple components with the same purpose
- Minimize upfront design
  - Try to design the minimum architecture so that developers can work
  - You will polish the next sections later

GENERAL GUIDELINES

- Use consistent patterns in each layer
  - If you decide to use MVC pattern in the presentation layer – stick to it
  - If you use CQRS in the application layer – stick to it
- Do not duplicate functionality
  - Do not have a caching component in all layers
- Prefer composition over inheritance
  - It is complicated to create huge hierarchies
  - Work with the Lead Developer here
- Establish a code convention
  - Define the code style and preferably automate it
  - Work with the Lead Developer here

LAYER GUIDELINES

- Separate areas of concern
  - Each layer should have one role only
- Define communication between layers
  - How does the presentation layer communicate with the business layer?
- Use abstraction to loosely couple layers
  - Use interfaces for every communication
  - Objects should not be tight to a particular platform or technology
- Don’t mix different types of components in a layer
- Use a consistent data format within a layer

COMPONENT GUIDELINES

- No component should rely on the internals of another
  - Components should be black boxes
- Do not mix roles in a single components
  - Web controllers should not have business logic
  - Business layer should not have HTTP concerns
- Define clear contracts for components
  - All public methods and properties should be scoped in advance
- Abstract system wide components away from other layers
  - As soon you have a component which is accessible in multiple layers
  - Put it in a system-wide area of concern

BEFORE WE CONTINUE…

HUGE THANKS FOR YOUR SUPPORT & TRUST!

- Everyone who got a paid ticket – 156 people in total! Thank you!
- Top supporter – **Nikolay Stoychev – 80 BGN**! Thank you, you rock! <3
- Thanks to – **Valentin, Nikolay, Borislava, Georgi, Пламен, Stefan, Dimitar, Plamen, Ivelin, Ивайло, Ivan, Teodor, Rosen, Георги, Ivan, Kristian, Deyan, Nikolay, Tsvetelin, Valentin, Kiril, Volen, Andrey, Diana, Дончо, Petar, Mincho, Elitsa, Цветан , Stanimir, Lyubomir , Veselin, Dobromir, Kristiyan, Nikolai, Борислав, Vladimir, Stefan, Valeri, Rostislav, Raya, Ivan, АЛЕКСАНДЪР, Kristian, Petko, Rosen, Todor, Diana, Pavel, Val, Радослав, Petar, Alexander, Mariyana, Ани, Emiliyan, Иван, Ivaylo, Petar, Vasil, Ivo, Irina, Stanislav, Vanya, Ivan, Julia, Kamen, Ivan, Bobi, Hristina, Zlatko, Maria, Ilcho, Vladimir, Svilen, Dinyo, Daniel, Pavel, Veselin, Petar, Hristo, Pavel, Plamen, Ivaylo, Калин, Sonya, Ivan, Krassimir, Zhelyazko, Chika, Pirin, Hristo, Dimitar , Ivaylo, Georgi , Albena, Svetoslav, Vladimir , Zlatin, Mira, Venelin, Dimitar, Iliya, Vanya, Kiril, Dobromir, Grigoris, Miroslav, Boris, Marin, Мирослава, Mariyan, Andrey, Galya, Teodor, Momchil, Simon, Plamen, Iva, Martin, Kosta, Hristo, Evgeniya, Nikolai, Ivo, Viktor, Tsvetan, Miroslav, Hyusein, Mira, Tanya, Zhivko, Dimitar, Ivaylo, Kalina, Emil, Ivaylo, Dimitar, Zhivko, Alexander, Denis, Denitsa, Boyan, Hristo, Donika, Georgi, Lyuboslav, Ilia, Stefan, Nikola, Valentin, Borislav, Teodor, Svetoslav, Veselin**
- And everyone who supported the initiative during the years!

THESE EVENTS ARE NOT EXACTLY FREE

- I prefer to call them "Pay what you want"
  - Depending on how much you value the provided knowledge
- It takes me a considerable amount of free time to prepare these lectures
  - And I want them to be perfect and complete!
  - I put my soul in them!
- For this reason, I will be extremely thankful, if you decide to support me and my projects!
  - It is never expected, but always appreciated!
- The easiest way is via
  - PayPal: **http://paypal.me/ivaylokenov**
  - Revolut: **@ivaylokenov**

UNIFIED MODELLING LANGUAGE

WHAT IS UML?

- From Wikipedia:
  - "A general-purpose, developmental, modeling language that is intended to provide a standard way to visualize the design of a system"
- Main attributes:
  - Visual – it is easy to see the representation of the architecture
  - Abstract – it stays away from implementation details
  - Descriptive – shows the complete representation
  - Standard – UML is the world standard
  - Supports Code Generation – specific sections can be converted to code
  - Supports Reverse Engineering – create UML from code
- You should know it in case the client wants it
  - But can easily avoid it

UML DESIGN ELEMENTS

- Models
  - Container for the design
  - Are you looking at the entire system? A subsystem? A feature?
- Views
  - A way to look at the system
  - Is it an external view? Or the internal structure?
- Diagrams
  - Specific drawings that illustrate the architecture

UML MODEL TYPES

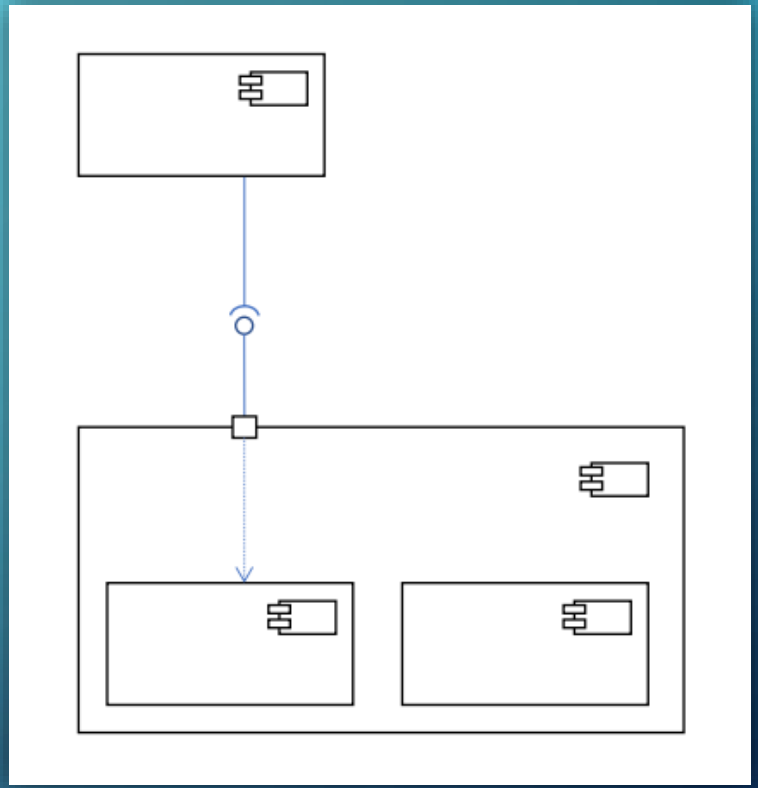
- Business System Model
  - Non-technical model
  - External View
    - Looking at the system as a black box
  - Internal View
    - A lot more details
- IT System Model
  - Specifically looks at technology
  - Static View (Structure View)
    - How the different elements fit together
  - Dynamic View (Behavioral View)
    - How these elements call each other

UML DIAGRAM TYPES

- Component Diagram
- Class Diagram
- Sequence Diagram
- State Diagram
- Activity Diagram
- Layer Diagram
- Use Case Diagram
- There are others…

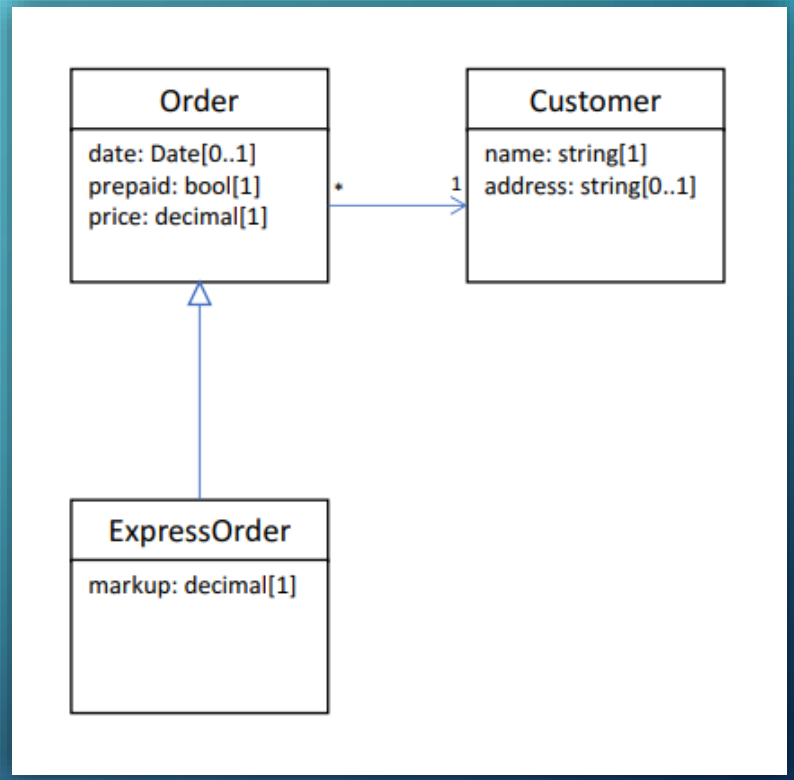
THE COMPONENT DIAGRAM

- Shows components
  - Modular building blocks
- Shows implemented and required interfaces
- Components can be nested
- If you decide to describe your architecture with a component diagram, you will have a very descriptive image



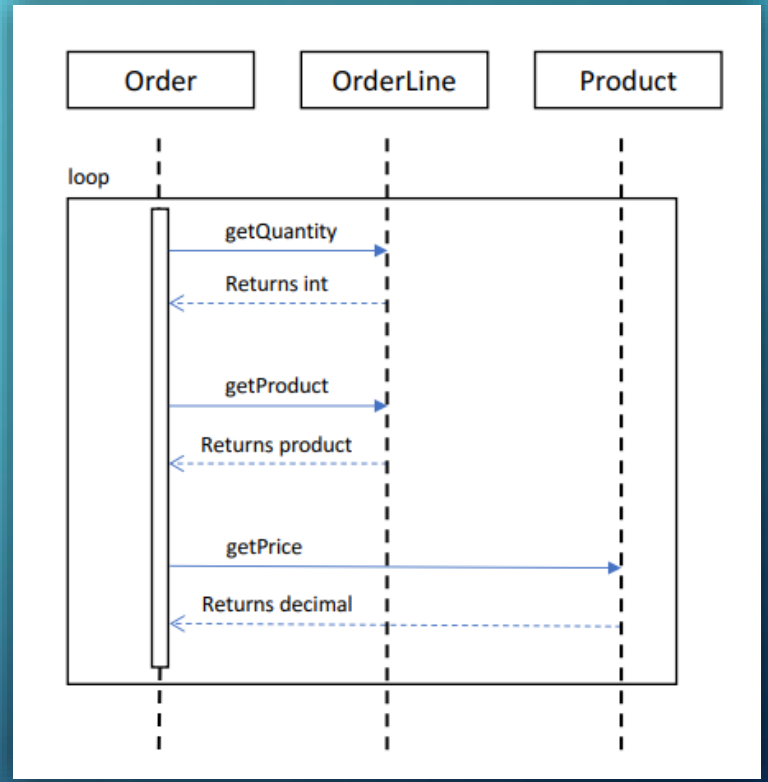
THE CLASS DIAGRAM

- Shows classes
- Shows methods and fields
- Shows associations, generalizations, and cardinality
- Quite detailed in terms of implementation
- The Lead Developer should do this



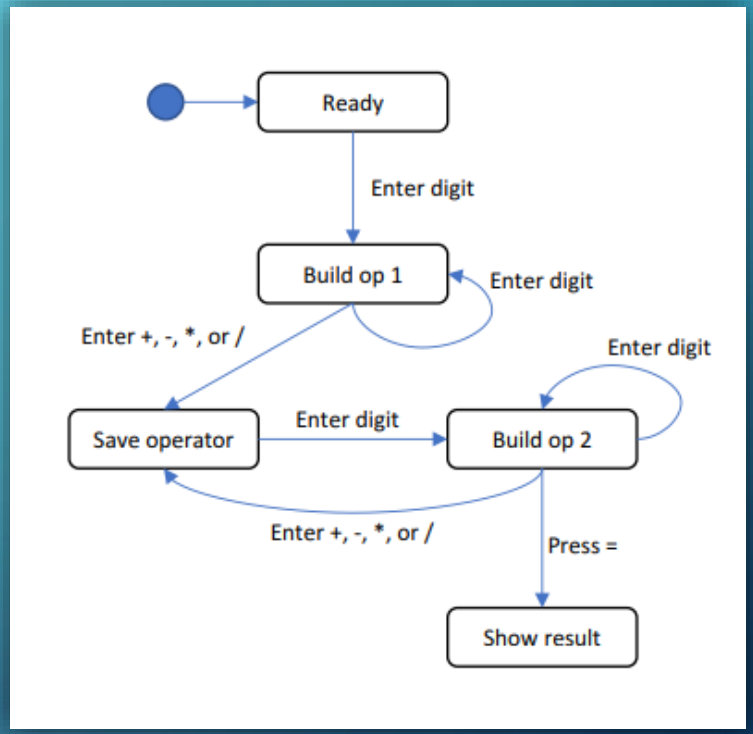
THE SEQUENCE DIAGRAM

- Shows call sequence
- Shows calling class, called method, and return data type
- Can depict loops



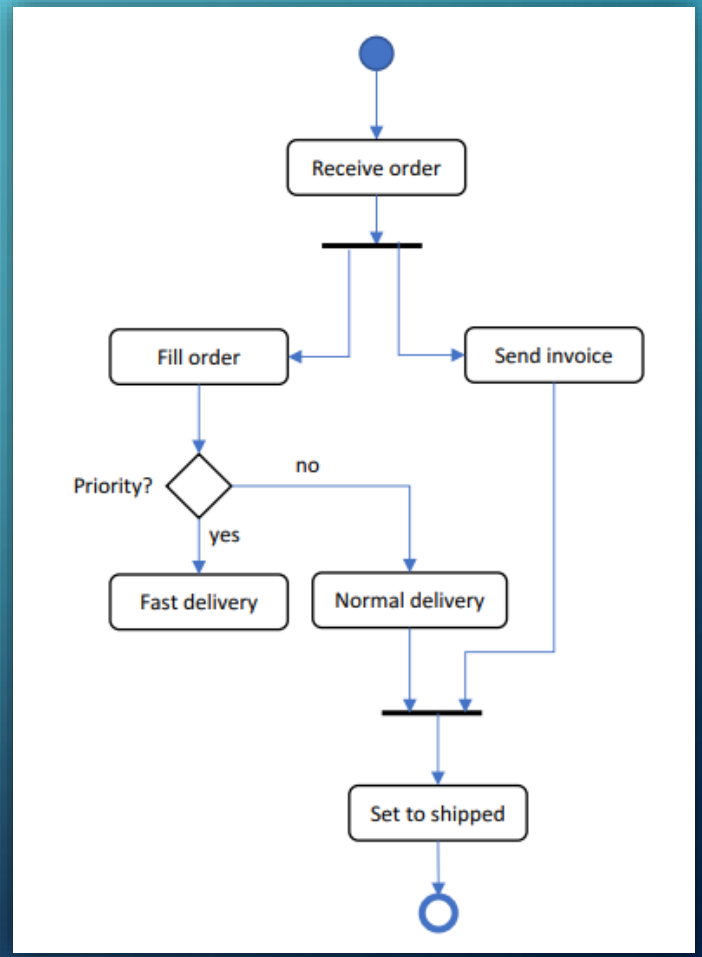
THE STATE DIAGRAM

- Shows states or activities
- Shows allowed transitions
- Can be nested
- Can depict internal activities



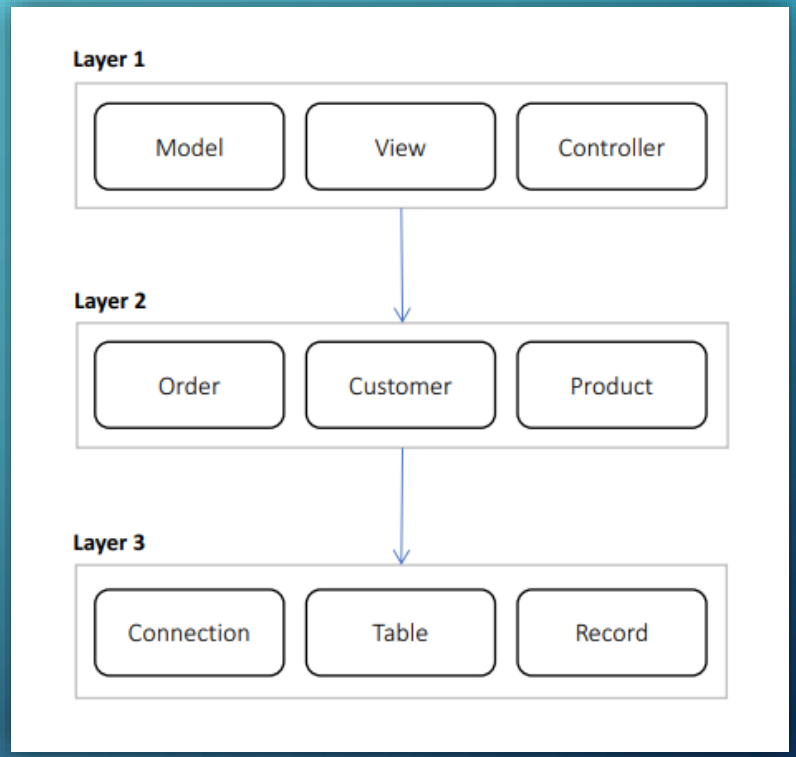
THE ACTIVITY DIAGRAM

- Shows process or workflow
  - Like the famous flow charts
- Can be nested
- Can show concurrent actions
- Can have swim lanes



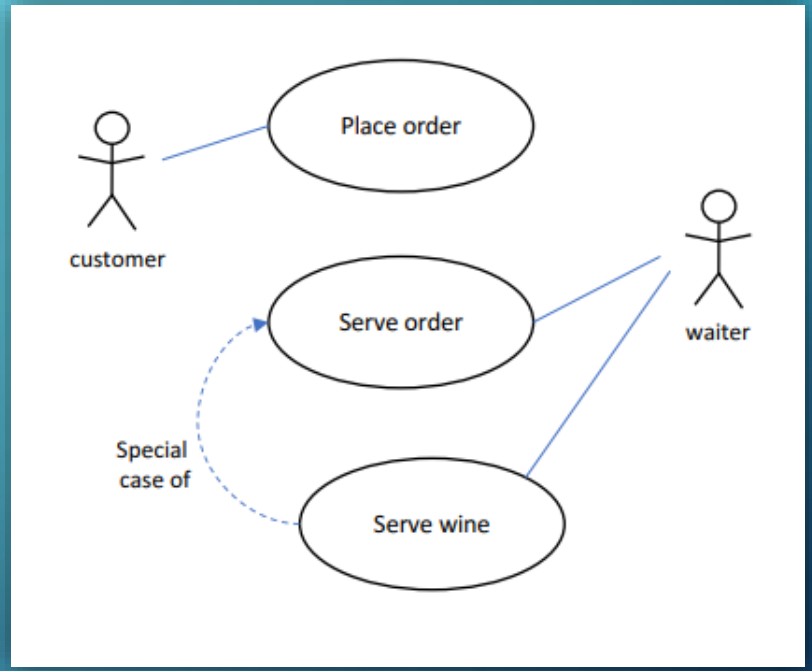
THE LAYER DIAGRAM

- Non-standard, invented by MS
- Shows areas of concern
- Shows references between areas
- Can be validated



THE USE CASE DIAGRAM

- Shows actors
- Shows use cases
- Binds actors to use cases
- Can depict generalizations



DESIGNING SOLUTION ARCHITECTURES

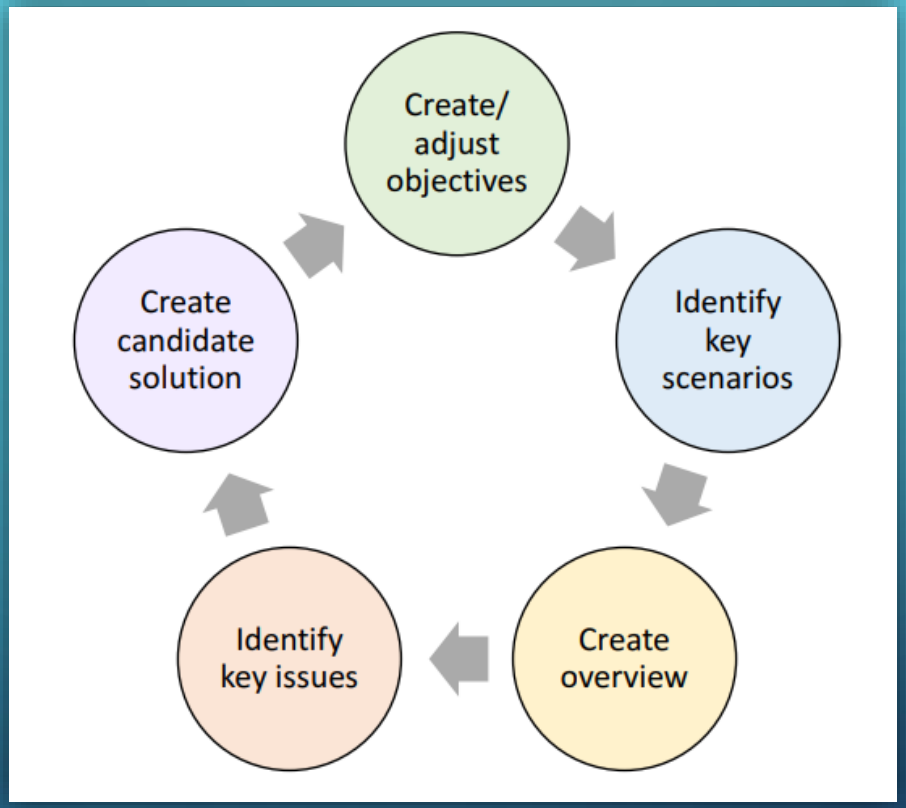
UML DIAGRAMS IN ARCHITECTURES

Solution Architecture Element	UML Diagram
Functional Requirement	Use Case Diagram
Structural Elements, Composition	Class Diagram Component Diagram
Structural Elements, Collaboration	Sequence Diagram Activity Diagram State Diagram
Areas Of Concern	Layer Diagram

UML DESIGN STRATEGIES

- UML as Sketch
  - Intended for brainstorming and general loose guidelines
- UML as Blueprint
  - Very detailed, you can write code based on the diagrams
  - Forward Engineering - use diagram to generate code
  - Reverse Engineering - build diagram from existing code
- UML as Validation
  - Validate implementation against diagram

THE PROCESS FOR DESIGNING ARCHITECTURES



CREATE OR ADJUST OBJECTIVES

CREATE OR ADJUST OBJECTIVES

- Identify scope of architecture
  - What are the high-level objectives and requirements?
  - New technologies? Server? Client? Proof of concept?
- Estimate time to spend
  - Do you have months? Do you have weeks?
- Identify audience
  - CEO? Developers? Functional Analyst?
- Identify technical, usage and deployment constraints
  - What are the available technologies?
  - How many people use the system simultaneously?
  - How many servers do you have? Cloud? On premise?
  - These requirements are super important!

IDENTIFY KEY SCENARIOS

IDENTIFY KEY SCENARIOS

- Key scenarios:
  - Significant unknown/risk – promo codes on scale
  - Significant use case – payments integration
  - Intersection of quality/function – clashing of these two attributes
  - Tradeoff between attributes – consider the trade offs
- Significant use cases:
  - Business-critical – part of the core business domain
  - High impact – very important for the end-users
- Create Use Case Diagrams

TWO TYPES OF REQUIREMENTS

- Functional
  - Business flows
  - What the user should do
  - User interfaces
  - And many more…
- Non-functional
  - Performance
  - Load
  - Data volume
  - Concurrent users
  - SLA
  - These are the most common ones

PERFORMANCE REQUIREMENTS

- "What is the required performance of the system?"
  - "SUPER FAST!"
- Always talk in numbers!
  - For example, if you have an end user – each task should complete in less than a second!
- Think about latency
  - How much time does it take to perform a single task?
  - How much time will it take to store a user in the database?
- Think about throughput
  - How many tasks can be performed for a given time unit?
  - How many users can be saved in the database in a minute?

PERFORMANCE NUMBERS EXAMPLE

- Let's say we have a task
  - Storing a user in database
- And its latency is 1 second
  - This is quite slow, but it is just a demonstrational number
- What is the throughput for 1 minute then?
  - In a well-designed system – more than 1000 users
  - In a badly designed system – around 60 users
- Both attributes are important!

LOAD REQUIREMENTS

- Load is quantity of work without crashing
  - In a Web API – how many concurrent requests could the server handle without crashing
- Difference with throughput:
  - Throughput – 100 requests/second
  - Load – 500 requests without crashing
- Users can tolerate a bit slow system
  - But they hate crashing ones!
- Best practice is to always plan for the most extreme cases!
  - For example – Black Friday in an e-commerce site!

DATA VOLUME REQUIREMENTS

- How much data the system will accumulate over time
- This requirement dictates:
  - The database type
  - Designing queries in the database
  - Storage planning
- Two aspects:
  - Data required on "day one"
    - For example – initially we need 1 GB of data
  - Data growth
    - For example – our database grows annually with 2 TB of data

CONCURRENT USERS REQUIREMENTS

- How many users will use the system simultaneously
- It is different than the load requirements
  - Concurrent users have "dead times"
  - They read the page, watch something on it, but do not make requests
  - Load requirements consider the actual requests
- The rule of thumb is to take the load requirements
  - And multiple them by 10!
  - But this depends on the system type

SERVICE LEVEL AGREEMENT REQUIREMENTS

- What is the required uptime for the system in percentage
  - For example, 99.99% uptime
  - This is translated to ~1 hour of downtime in a year
  - It would be quite impressive!
- It is the solution's architect to manage the clients' expectations
  - If the expected uptime is 99.999%
  - Just tell them that we will need at least five different data centers
    - On independent continents
    - With variety of power supplies
    - And multiple Internet providers
  - This answer usually brings them down to Earth ☺

WHO DEFINES THESE REQUIREMENTS?

- The Functional Analysts and the CEO usually provide a well documented business and functional requirements of the system
- But who defines the non-functional ones?
  - It is the Solution's Architect job to frame them
    - By starting discussions
    - And asking the right questions
    - And keeping the expectations realistic and meaningful
    - There is no need to fight for every millisecond
    - The users will not notice that
- Never design a system without the non-functional requirements!

FINAL WORDS BEFORE Q & A

SUMMARY

- Why Software Architecture
- What Is Software Architecture?
- Unified Modeling Language
- Designing Solution Architectures

- Don't Forget The Optional But Practical Guide-Book
  - 3 Real-World Scenarios
  - 70+ Pages
  - Free Updates
  - Get It From The Event's Page or write to wewritesoftware@gmail.com
    - https://www.eventbrite.com/e/software-architecture-fundamentals-essentials-code-it-up-online-vol-9-registration-222550182587

IN THE NEXT PART

- Common Technology Stacks
- Architecture Design Patterns
- Choosing The Right Patterns
- Register here:
  - https://www.eventbrite.com/e/software-architecture-technology-patterns-code-it-up-online-vol-10-registration-244365432587
- Don't Forget The Optional But Practical Guide-Book
  - 3 Real-World Scenarios
  - 70+ Pages
  - Free Updates
  - Get It From The Event's Page or write to wewritesoftware@gmail.com
    - https://www.eventbrite.com/e/software-architecture-fundamentals-essentials-code-it-up-online-vol-9-registration-222550182587

OTHER GOODIES

- You can check the Code It Up blog and subscribe:
  - https://codeitup.today
- You can watch some of the free videos:
  - https://www.youtube.com/CodeItUpwithIvo
  - Clean code & The art of testing
  - Docker, CI/CD, Redis, Elasticsearch
  - And many more…
- The source code in all free lessons is available on Patreon:
  - https://www.patreon.com/ivaylokenov
- Unrelated to the IT sector but check out my art T-Shirts:
  - https://way-ve.com/
  - Use CODE10 during checkout for 10% discount

PAST CODE IT UP EVENTS

- You can get the recordings of the past C# events from the event page:
  - C# Async-Await In Detail
  - The C# ORM Battle
  - Docker – From ABC To XYZ
  - Identity Server Demystified
  - Eventual Consistency Done Right
  - Let's Get Functional With C#
  - C# API Scenarios – REST, GraphQL & gRPC
- Past workshops are also available:
  - C# Multithreading
  - Domain-Driven Design With ASP.NET Core
  - Kubernetes For Web Developers
- If interested, you can write to wewritesoftware@gmail.com

ANY QUESTIONS?

- You can support me and my projects:
  - Via PayPal: http://paypal.me/ivaylokenov
  - Via Revolut: **@ivaylokenov**
  - On Patreon: https://www.patreon.com/ivaylokenov
  - On Open Collective: https://opencollective.com/mytestedaspnet
  - Via Buy Me A Coffee: http://buymeacoff.ee/ivaylokenov
  - Crypto: http://bit.ly/ik-sponsors
- Never expected, always appreciated!
- Make sure you check my sponsors!
  - INDEAVR - https://indeavr.com
  - Americaneagle.com - https://www.americaneagle.com

THANK YOU!

RESOURCES:

HTTPS://GITHUB.COM/IVAYLOKENOV/SOFTWARE-ARCHITECTURE-SERIES