

Software Architecture and High-Level Design

Foundations of Software Engineering

FSE v2020.1, Block 1 Module 4

Alexey Artemov, Fall 2020

1

FSE v2020.1, Block 1 Module 4

Lecture Outline

§1. Why software architecture [15 min]

- 1.1. Partitioning, knowledge, and abstraction
- 1.2. Example architecture of a logging system
- 1.3. What is architecture?
- 1.4. When/why is architecture important?

2

FSE v2020.1, Block 1 Module 4

Lecture Outline

§2. Some principles of software architecture [10 min]

- 2.1. Prioritizing quality attributes
- 2.2. Architecture drivers, tradeoffs, and design decisions

§3. Software architecture notation [15 min]

- 3.1. Canonical model of software design
- 3.2. Component and connector

3

FSE v2020.1, Block 1 Module 4

The Goal:
Think Before You Code

4

FSE v2020.1, Block 1 Module 4

§1. Why software architecture?

5

FSE v2020.1, Block 1 Module 4

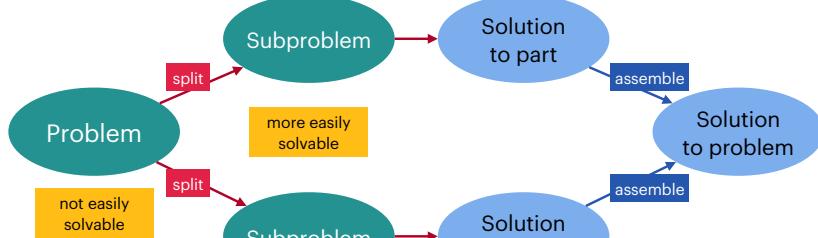
Partitioning, knowledge, abstraction

6

FSE v2020.1, Block 1 Module 4

§1. Why software architecture?

1.1. Partitioning, knowledge, and abstraction



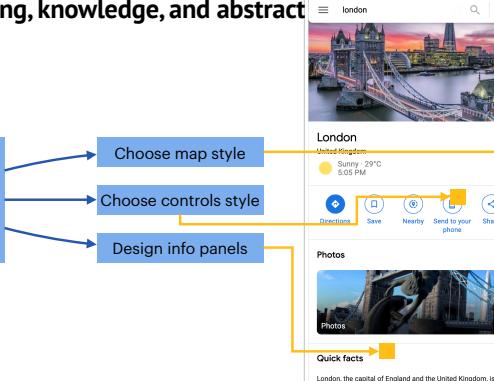
7

FSE v2020.1, Block 1 Module 4

§1. Why software architecture?

1.1. Partitioning, knowledge, and abstraction

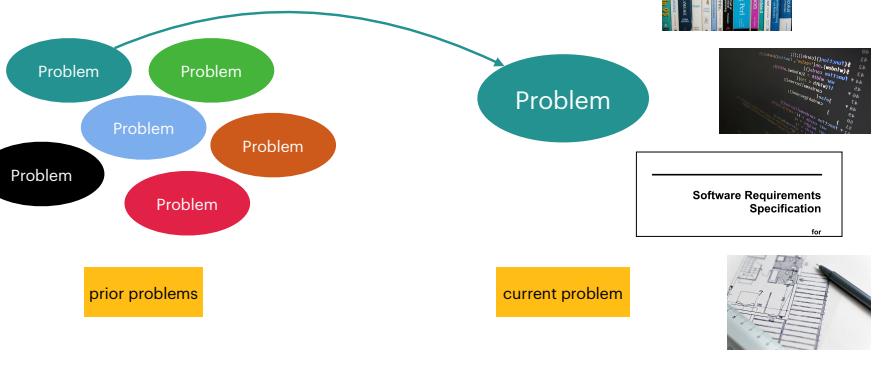
Design
Google
Maps



8

§1. Why software architecture?

1.1. Partitioning, knowledge, and abstraction

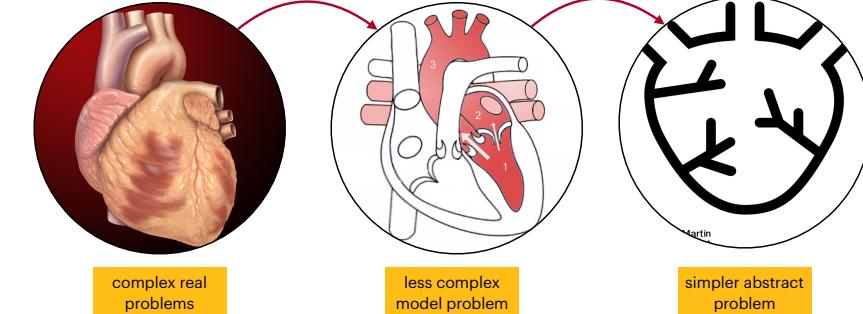


9

FSE v2020.1, Block 1 Module 4

§1. Why software architecture?

1.1. Partitioning, knowledge, and abstraction



10

FSE v2020.1, Block 1 Module 4

§1. Why software architecture?

1.1. Partitioning, knowledge, and abstraction

- Partitioning: divide and conquer
- Knowledge: experience transfer
- Abstractions: shrink problems

11

FSE v2020.1, Block 1 Module 4

Example architectures:
Searching for events
in log files

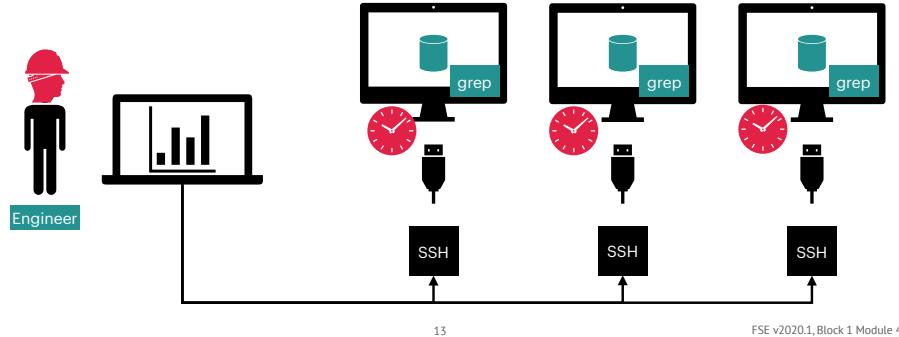
12

FSE v2020.1, Block 1 Module 4

§1. Why software architecture?

1.2. Example architecture of a logging system

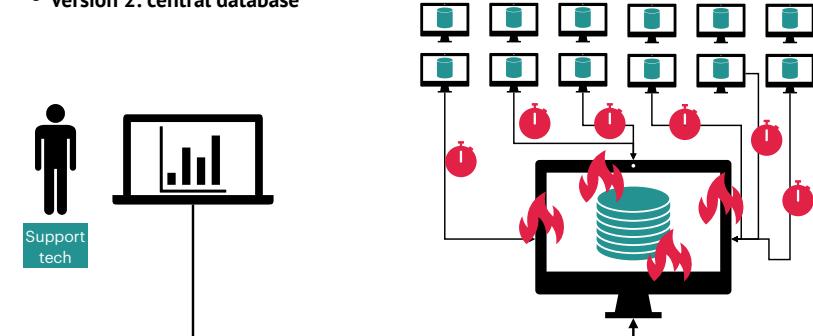
- Version 1: local log files



§1. Why software architecture?

1.2. Example architecture of a logging system

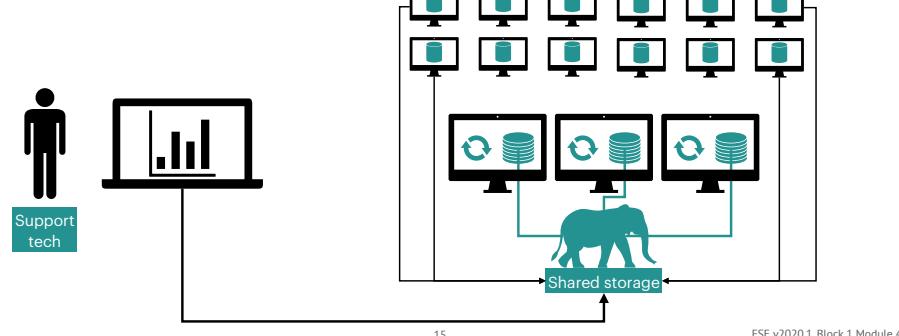
- Version 2: central database



§1. Why software architecture?

1.2. Example architecture of a logging system

- Version 3: indexing cluster



§1. Why software architecture?

1.2. Example architecture of a logging system

Version 1: local log files

Same functionality

Version 2: central DB

Different architecture

Version 3: indexing cluster

Different quality attributes

§1. Why software architecture?

1.2. Example architecture of a logging system

Same functionality	Different architecture	Different quality attributes
Ad hoc queries?	modifiability seconds	scalability <i>not so scalable</i> latency online
Version 1: local log files	seconds	<i>not so scalable</i> near online
Version 2: central DB	<i>schedule & run</i>	very scalable a bit stale

17

FSE v2020.1, Block 1 Module 4

What is software architecture?

§1. Why software architecture?

1.3. What is software architecture?

- “Software architecture level of design is... organization of a system as a composition of components; global control structures; the protocols for communication, synchronization, and data access; the assignment of functionality to design elements; the composition of design elements; physical distribution; scaling and performance; dimensions of evolution; and selection among design alternatives” – Shaw & Garlan, 1996
- “The software architecture of a computing system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both” – Clemens, 2010.
- “The fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution” – IEEE 1471

components relations properties

19

FSE v2020.1, Block 1 Module 4

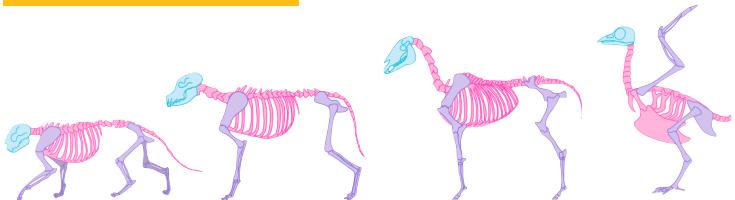
Why and When is architecture important?

FSE v2020.1, Block 1 Module 4

§1. Why software architecture?

1.4. Why and When is architecture important?

Architecture as a skeleton



21

FSE v2020.1, Block 1 Module 4

§1. Why software architecture?

1.4. Why and When is architecture important?

Architecture influences quality



Image Credit: TechPrevue

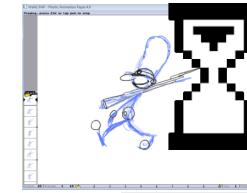


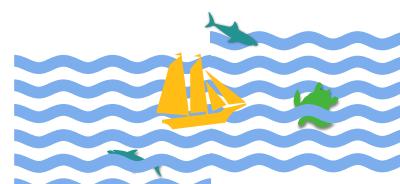
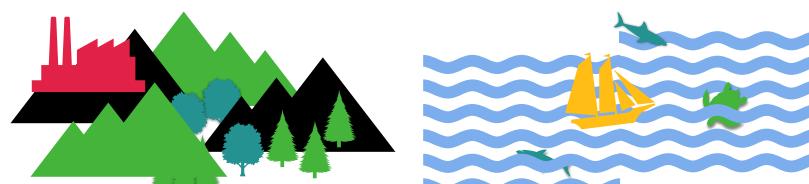
Image Credit: istofffreeware.com

22

§1. Why software architecture?

1.4. Why and When is architecture important?

Architecture \perp functionality

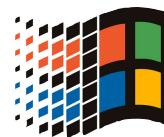


FSE v2020.1, Block 1 Module 4

§1. Why software architecture?

1.4. Why and When is architecture important?

Architecture constrains systems



legacy



vendors



resources

FSE v2020.1, Block 1 Module 4

§1. Why software architecture?

Summary

- Components and their separation
- Partitioning to create sub-systems to be created separately
- Buy vs. build decisions
- Mistakes are hard to recover from
- Team organization

§2. Some principles of software architecture

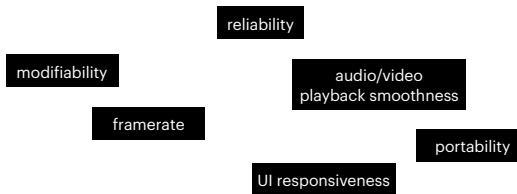


Prioritizing quality attributes

§2. Some principles of software architecture

2.1. Prioritizing quality attributes

- **Identify** quality attributes
 - Customer survey
 - Requirements engineering
- **Prioritize** quality attributes



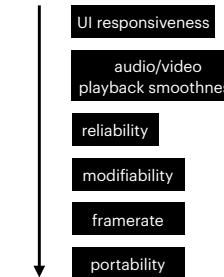
29

FSE v2020.1, Block 1 Module 4

§2. Some principles of software architecture

2.1. Prioritizing quality attributes

- **Identify** quality attributes
 - Customer survey
 - Requirements engineering
- **Prioritize** quality attributes



30

FSE v2020.1, Block 1 Module 4

§2. Some principles of software architecture

2.1. Prioritizing quality attributes

- Quality attribute scenarios
 - Source
 - Stimulus
 - Environment
 - Artifact
 - Response
 - Response measure

Source	Yinzer Member
Stimulus	Request web page from Yinzer server
Environment	Normal operations
Artifact	Entire system
Response	Server replies with web page
Response Measure	Web page sent out by Yinzer system within 1 second
Full QA scenario	A Yinzer Member clicks on a link in his Web Browser; it sends a request to the Yinzer system, which sends out a web page as a reply within 1 second

Figure 9.7: An example of a full quality attribute scenario for the Yinzer System. You may omit some sections, but you should strive to write falsifiable scenarios.

31

FSE v2020.1, Block 1 Module 4

Skoltech
Skolkovo Institute of Science and Technology

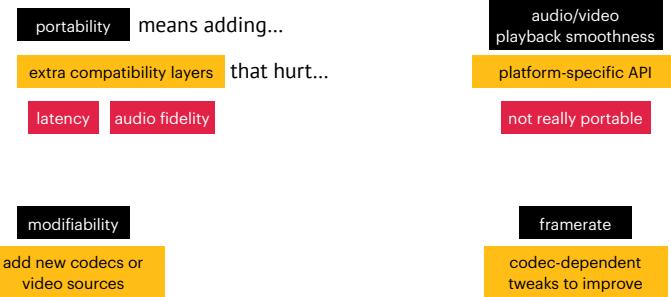
Tradeoffs, design decisions,
and architecture drivers

32

FSE v2020.1, Block 1 Module 4

§2. Some principles of software architecture

2.2. Tradeoffs, design decisions, and architecture drivers



33

FSE v2020.1, Block 1 Module 4

§2. Some principles of software architecture

2.2. Tradeoffs, design decisions, and architecture drivers

- Prioritization of quality attribute scenarios

• Stakeholders: what is **important**

• Developers: how **difficult to achieve**



- Some priorities are pretty clear (easy / deferrable)



- Architecture drivers: important and hard to achieve



34

FSE v2020.1, Block 1 Module 4

§2. Some principles of software architecture

2.2. Tradeoffs, design decisions, and architecture drivers

- Architecture driver #1:
 - "When a user gives a command, such as pressing pause on the remote control, the system should comply with the command within 50ms."
- Architecture driver #2:
 - "The reference H.264/MPEG-4 AVC video from local disk should play smoothly on the reference hardware."

35

FSE v2020.1, Block 1 Module 4

§2. Some principles of software architecture

2.2. Tradeoffs, design decisions, and architecture drivers

- Design decision:
 - "To promote reliability, each top-level component will run in its own process to isolate faults, like services in an operating system."
- Design decision:
 - "The Media Rendering/Playback component communicates using shared memory with the Media Buffer component to minimize latency, considering the high rate of data movement."

36

FSE v2020.1, Block 1 Module 4

§2. Some principles of software architecture

Summary

- Quality attributes (QAs): describe observable properties of a system.
- Quality attribute scenarios: describing QA requirements
- Architecture drivers: QA scenarios that need increased attention during design
- Tradeoffs: balancing quality attributes to achieve a desired result

37

FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

38

FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

3.1. Canonical model of software design

- **Domain model:**
 - enduring truth about the domain
 - something “just true”
 - “billing cycles are 30 days”
- **Design model:**
 - the system to be built
 - partial set of design commitments
 - “deallocate resources explicitly”
- **Code model**
 - full set of design commitments
 - “the customer address is stored in a varchar(80) field”

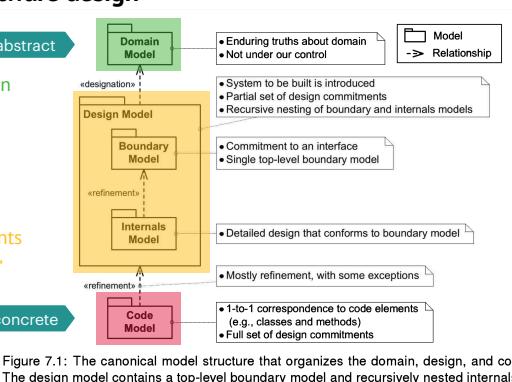


Figure 7.1: The canonical model structure that organizes the domain, design, and code models. The design model contains a top-level boundary model and recursively nested internals models.

39

Credit: George Fairbanks, Just Enough Software Architecture FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

3.1. Canonical model of software design



UML Use Case



UML Class



UML Sequence

FSE v2020.1, Block 1 Module 4

FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

3.1. Canonical model of software design: Domain model

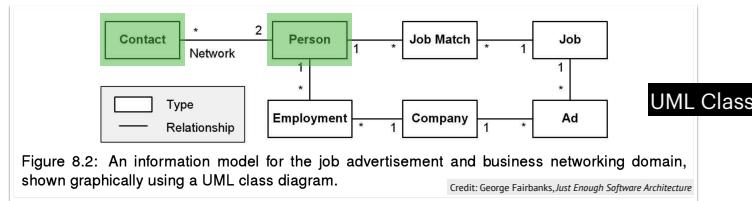


Figure 8.2: An information model for the job advertisement and business networking domain, shown graphically using a UML class diagram.
Credit: George Fairbanks, Just Enough Software Architecture

- **Company:** an employer that offers *Jobs* to *People*
- **Employment:** a relationship indicating that the *Person* is or was employed at a *Job* at the *Company*

41

FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

3.1. Canonical model of software design: Design model

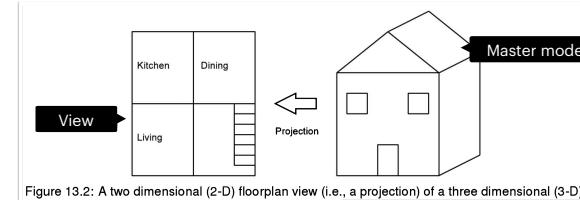


Figure 13.2: A two dimensional (2-D) floorplan view (i.e., a projection) of a three dimensional (3-D) house model.
Credit: George Fairbanks, Just Enough Software Architecture

- **View:** a projection of the model that reveals important detail
- Examine a variety of views on the job advertisement “Yinzer system”:
 - Use case, system context, components, ports/connectors, modules, component assembly, deployment

42

FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

3.1. Design model: Use case

- **Use case:**
 - Compact view of functionality
 - Actors
 - Use-cases
- **Functionality scenario:**
 - X invites a contact Y / System sends Y email
 - Y clicks a link in the email and accepts invitation to be a contact
- Make some commitments, but keep other design options open

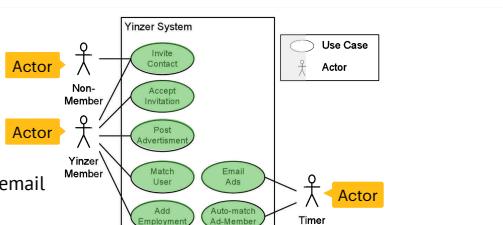


Figure 9.2: A use case diagram for the Yinzer system. It is effective at presenting a broad overview of the Yinzer system's functionality.
Credit: George Fairbanks, Just Enough Software Architecture

43

FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

3.1. Design model: System context

- **System context:**
 - Model channels of communication with external systems
 - Connectors/ports
 - Components

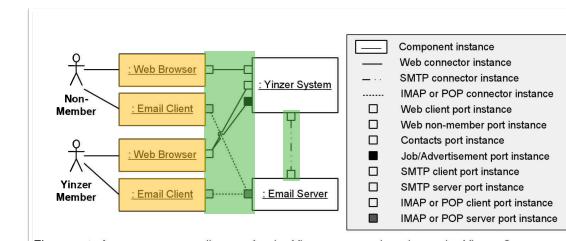


Figure 9.4: A system context diagram for the Yinzer system that shows the Yinzer System component instance and all the external systems it connects to.
Credit: George Fairbanks, Just Enough Software Architecture

44

FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

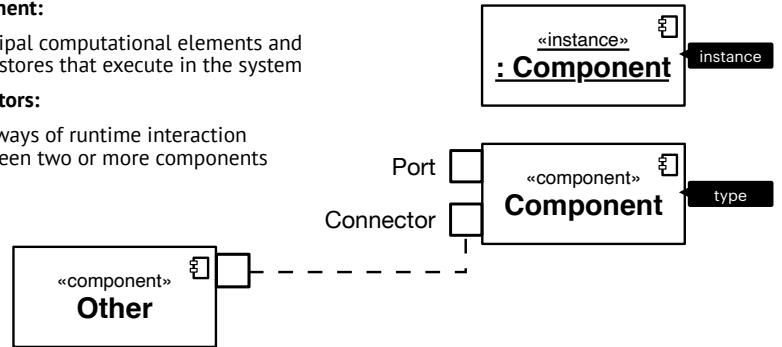
3.2. Components and connectors

- **Component:**

- Principal computational elements and data stores that execute in the system

- **Connectors:**

- Pathways of runtime interaction between two or more components



45

FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

3.2. Components and connectors

On importance of connectors

convert, transform, translate datatypes

local/remote procedure calls

adapt protocols

pipes, batch transfer

broadcast, clean, prioritize events

interrupts, shared memory

encrypt, compress, synchronize

RPC, ESB

46

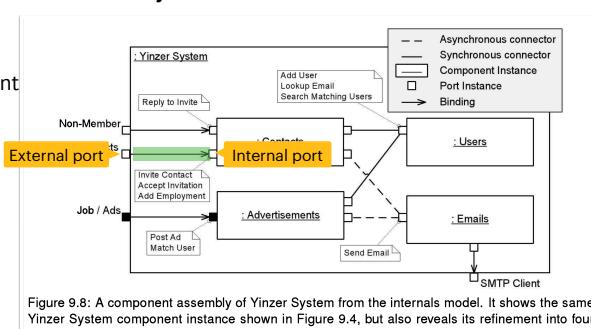
FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

3.1. Design model: Component assembly

- **Component assembly:**

- Configuration of component instances in a system
- External ports bound to internal ports



47

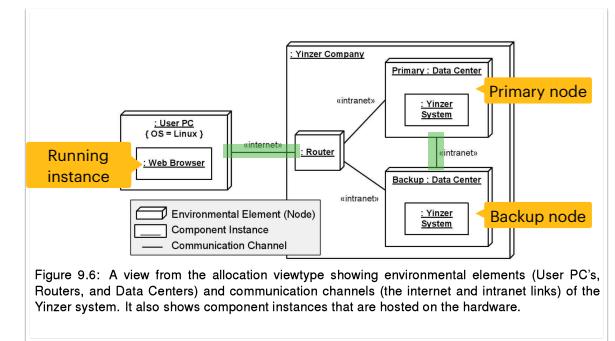
FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

3.1. Design model: Deployment

- **Deployment:**

- Environmental elements
- Communication channels
- Allocation of running component instances to hardware



48

FSE v2020.1, Block 1 Module 4

§3. Software architecture notation

Summary

- Canonical model: from abstract to concrete
 - Domain model, Design model, Code model
 - UML notation
- Domain models: information models of enduring truths about the domain
- Design model: describe the system to be built
 - Many views (use case, system context, components, ports/connectors, modules, component assembly, deployment)
 - Reasoning with components and connectors

References

1. R. Stephens (2015). *Beginning software engineering*. John Wiley & Sons.
2. G. Fairbanks (2010). *Just enough software architecture: a risk-driven approach*. Marshall & Brainerd.
3. I. Sommerville (2020) *Software engineering 9th Edition*.

