Software Platform

Lecture 2

Emanuele Di Buccio

01/03/2024

Master Degree in Computer Engineering, A.A. 2023/2024

In the previous lecture . . .

Previous lecture topics

- Definitions of **Software Platform**
- Software Engineering
- Software Processes, Paradigms and Architecture
- Patterns
- A scenario for supporting the adoption of Microservices

Previous lecture topics

https://app.wooclap.com/HHRZHD

Software platform

- “an operating environment upon which applications can execute and which provides reusable capabilities” [Bottcher, 2018]
- other systems the software interfaces with and make up the environment in which the software will execute [Sommerville, 2016]
- “a collection of services that help companies get their software running in front of their customers” [Salatino, 2023]
- “foundation of self-service APIs, tools, services, knowledge and support which are arranged as a compelling internal product” [Bottcher, 2018]

Monolith Architecture Advantages

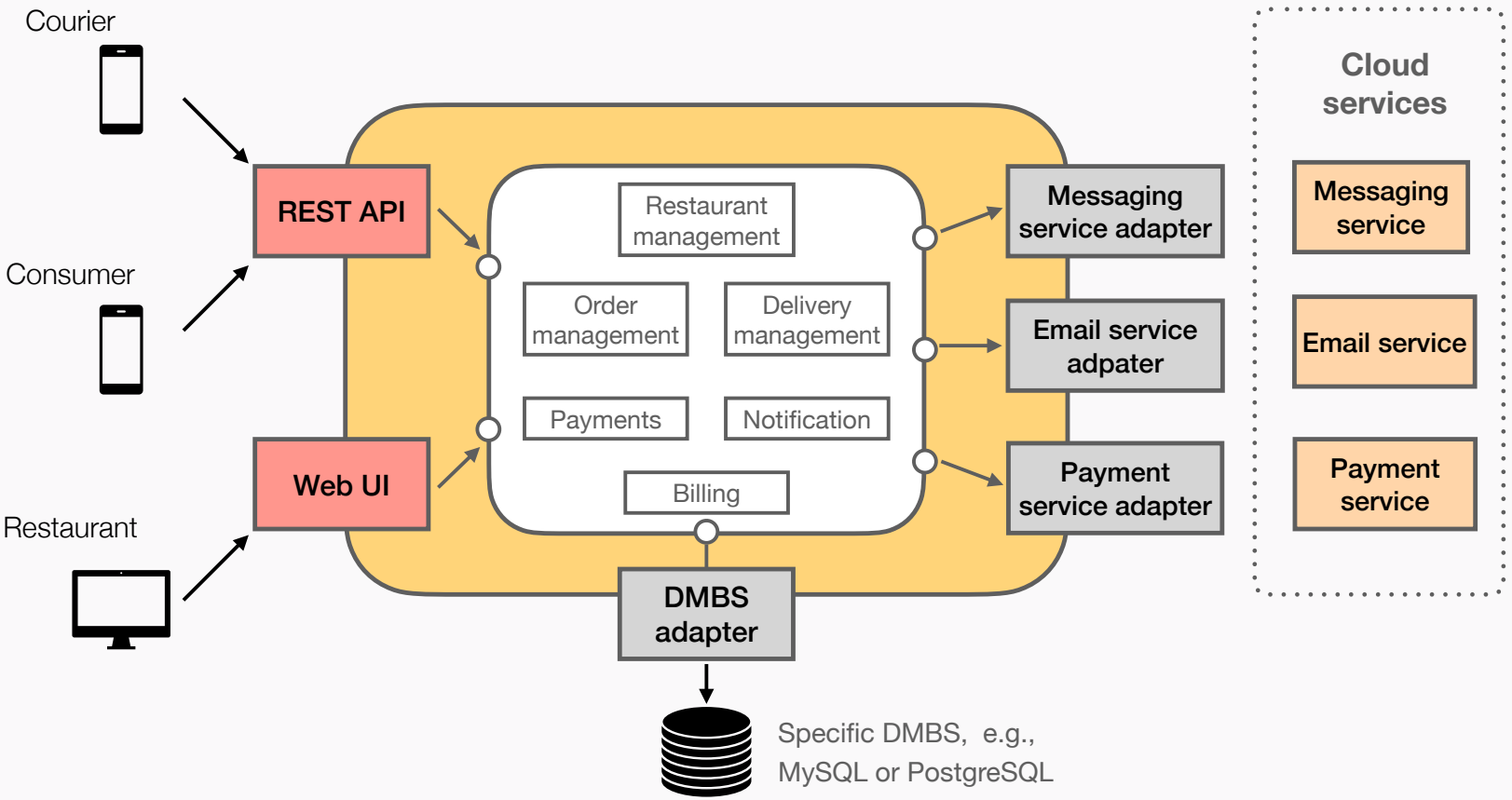
Should I stay always away from **monolithic** architectures? **No!**

Some **advantages**:

- “Simple” to develop
- Easy to make radical changes to the application
- Straightforward to test
- Straightforward to deploy
- Easy to scale

More on Course Topics

Why Microservices?



Based on Figure 1.1 from [Richardson, 2019], accessible here

What are microservices? (1)

Possible definition (by Adrian Cockcroft here)

- “Loosely coupled service-oriented architecture with bounded context”

Let us “unpack” it:

- Loosely coupled
- service-oriented architecture
- bounded context

What are microservices? (2)

- Loosely coupled
  - no need to update every service at the same time
- service-oriented architecture
  - “a specific type of distributed system framework that maintains agents that act as “software services,” performing well-defined operations” [Joseph et al., 2004]
  - note: we will discuss SOA vs Microservices
- bounded context
  - no need to know too much about the surrounding services
  - single function that does one thing and can be modified quite easily
  - understand easily downstream and upstream dependencies

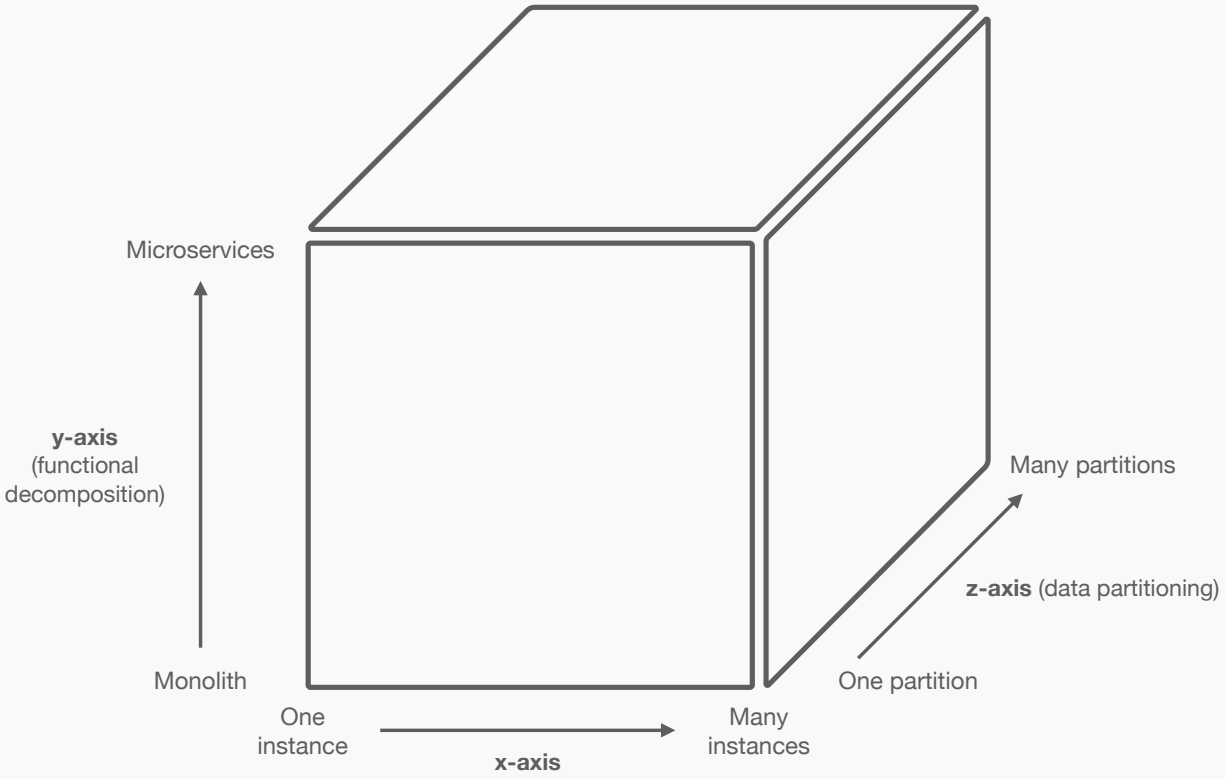
What are microservices? (3)

From [Tanenbaum and Steen, 2023]:

- Essential property that each microservice runs as a separate (network) process
- Does *micro* require service to be small? How small?
  $\Rightarrow$ no agreement on the size

What are microservices? (4)

- We will elaborate more on this definition as in [Richardson, 2019]
- the Scale Cube [Abbott and Fisher, 2009]



What are microservices? (5)

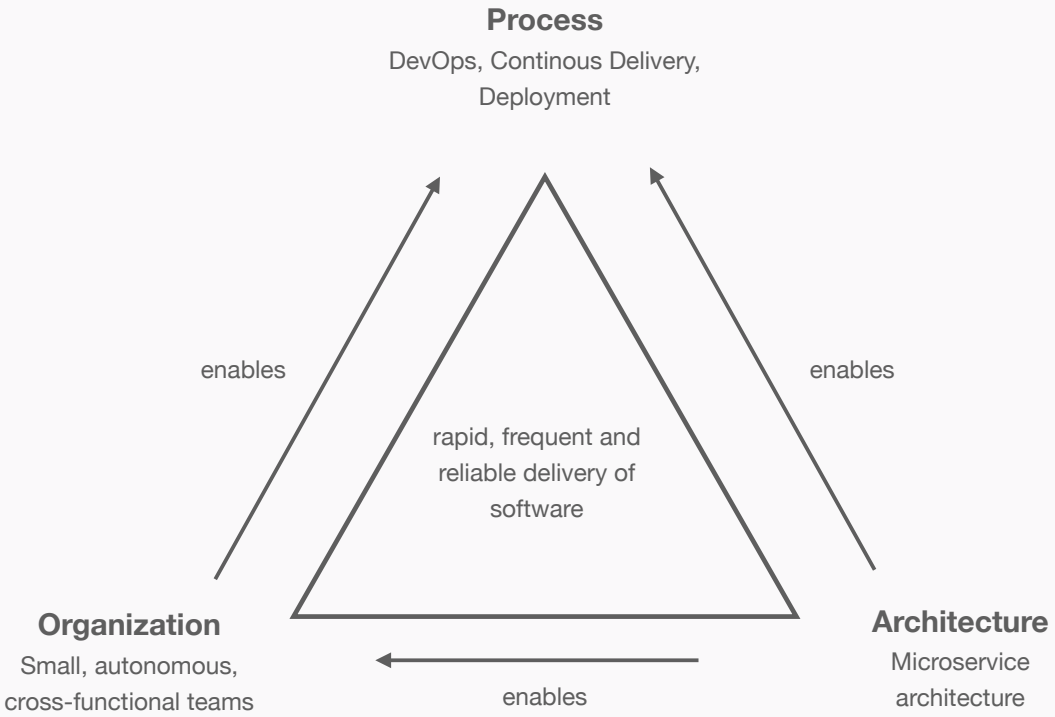
Microservices

- functional decomposition (y-axis), where the application is split in a set of **services**
- services as **unit of modularity**
- each service has **its own datastore**

Process and organization (1)

Is it just a matter of architecture? No! In addition to the architecture

- organization
- development and delivery



Java for Complex Software Platforms

- Main programming language of the course: Java
- A framework that can help in developing software platforms: Spring
- What is Spring?
  - open-source application framework
  - original form introduced in [Johnson, 2003]; history here
  - provides several modules for a range of services
  - “offers a container [. . .] that creates and manages application components. These components [. . .] are wired together inside [the container] to make a complete application, much like bricks, mortar, timber, nails, plumbing, and wiring are bound together to make a house.” [Walls, 2022]

Cloud Computing (1)

From IBM article "What is Cloud Computing?"

> *Cloud computing is* **on-demand access**, *via the internet,* **to computing resources** – *applications, servers (physical servers and virtual servers), data storage, development tools, networking capabilities, and more* – **hosted at a remote data center managed by a cloud services provider** *(or CSP). The CSP makes these resources available for a monthly subscription fee or bills them according to usage.*

Cloud Computing (2)

What is cloud computing? [Tanenbaum and Steen, 2023]

- organizations running data centers opening up their resources to customers $\rightarrow$ **utility computing**
  - customer could upload tasks to a data center and be charged on a per-resource basis
  - basis for cloud computing
- **cloud computing**
  - characterized by easily usable and accessible pool or *virtualized* resources
  - *dynamic configuration* of which and how resources are used
  - need to scale? acquire more resources!
  - generally cloud computing based on a *pay-per-use* model

Cloud Computing (3)

Cloud organized in four layers:

- **Hardware**
  - processors, routers, power and cooling systems
- **Infrastructure**
  - virtual storage and computing resources
- **Platform**
  - vendor-specific API
  - includes calls to upload and execute a program in the vendor's cloud
  - higher-level abstractions, e.g., Amazon S3 for storage
- **Application**
  - *examples*: applications found in office suites

Cloud Computing (4)

Services offered through various interfaces:

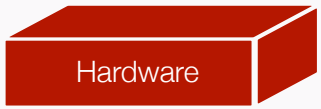
Type of service	Resource managed at the layer		Examples
Software as a Service (SaaS)		Web services, multimedia, business app	Google docs, Gmail, Flickr
Platform as a Service (PaaS)		Software framework (Java), Storage (databases)	MS Azure, Google App engine
Infrastructure as a Service (IaaS)		Computation (VM), storage (block, file)	Amazon S3, Amazon EC2
		CPU, memory, disk, bandwidth	Datacenters

Image adapted from [Zhang et al., 2010] and [Tanenbaum and Steen, 2023]

Cloud Computing (5)

Serverless and Function-as-as-Service (FaaS)

- “In serverless, the cloud provider manages the provisioning of resources for a service in a transparent, auto-scaling manner, without the developer in the loop” [Shahrad et al., 2019]
- “[FaaS] is a way to scale the execution of simple, standalone, developer-written functions, where state is not kept across function invocations” [Shahrad et al., 2019]

Cloud Computing (6)

Possible benefits (from “What is Cloud Computing?”)

- Lower IT costs
- Improve agility and time-to-value
- Scale more easily and cost-effectively

Software Engineering

Software Processes and Paradigms

Software processes

- “A software process is a set of related activities that leads to the production of a software system” [Sommerville, 2016]
- “A *process* is a collection of activities, actions, and tasks that are performed when some work product is to be created” [Pressman, 2009]
- Real software processes are interleaved sequences of diverse activities:
  - technical
  - collaborative
  - managerial

Process Activities

Fundamental activities is a software process [Sommerville, 2016]

1. Software specification
   → definition of software functionality and constraints
2. Software development
   → software to meet the specification is designed and programmed
3. Software validation
   → ensure that the software is what the customer requires
4. Software evolution
   → changes software to reflect changing customer and market requirements

Process Activities - Software specification (1)

Software specification

- process of
  - understanding and defining what services are required from the system
  - identifying the constraints on the system's operation and development
- mistakes made at this stage inevitably lead to later problems in subsequent stages

Process Activities - Software specification (2)

- May be preceded by feasibility or marketing study to assess
  - whether or not there is a need or a market for the software
  - whether or not it is technically and financially realistic to develop the software required
- Result of the activity: an **agreed requirements document** that specifies a system satisfying stakeholder requirements
- What is a stakeholder?
  - every person or group that will be affected by the system

Process Activities - Software specification (3)

- Requirements:
  - descriptions of the services that a system should provide
  - constraints on its operation
- Requirement Engineering:
  - finding out
  - analyzing
  - documenting
  - checking

  services and constraints

Process Activities - Software specification (4)

Possible classification of requirements

- functional requirements
- non-functional requirements

Process Activities - Software specification (5)

Functional requirements

- statements of services the system should provide
- how the system should react to particular inputs
- how the system should behave in particular situations

Process Activities - Software specification (6)

Non-functional requirements

- constraints on the services or functions offered by the system
- timing constraints
- constraints on the development process
- constraints imposed by standards

Process Activities - Software specification (7)

Main activities in software specification:

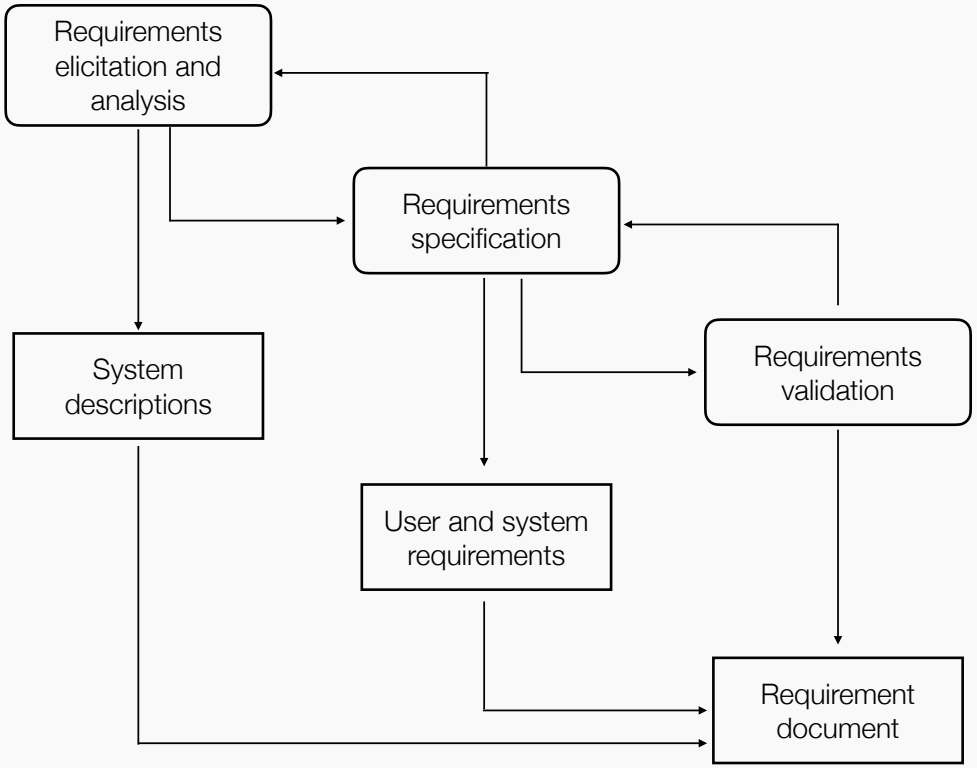


Figure from [Sommerville, 2016]

Process Activities - Software specification (8)

Main activities in software specification:

1. **Requirement elicitation and analysis**
   - observation of existing systems
   - discussions with potential users
   - task analysis
   - development of one or more system models and prototypes
2. **Requirement specification**
   - user requirements (high-level description for customer/end-user)
   - system requirements (a more detailed description)
3. **Requirement validation**
   - checking requirements for realism, consistency, and completeness
   - fix error in the requirement document, if any

Process Activities - Software design and implementation (1)

- **Goal**: developing an executable system for delivery to the customer
- may include two separate activities:
  - design
  - implementation
- **software design**: description of
  - structure of the software
  - data models and structures used by the system
  - interfaces between system components
  - (sometimes) algorithms used

Process Activities - Software design and implementation (2)

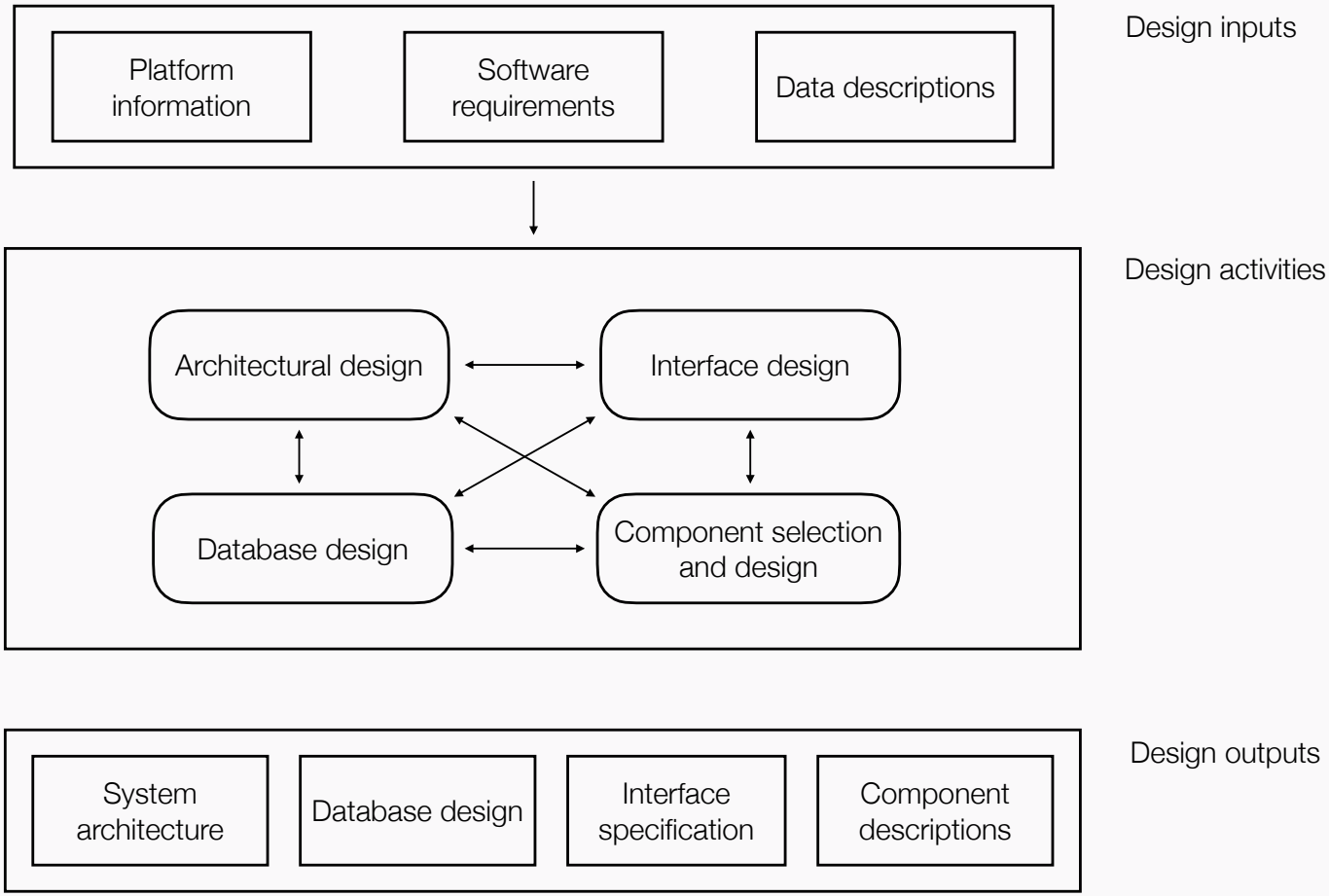


Figure from [Sommerville, 2016]

Process Activities - Software design and implementation (3)

- Architectural design
  - overall structure of the system
  - principal components (subsystems/modules)
  - components relationships and distribution
- Database design
  - system data structures and their representation in a database
- Interface design
  - interfaces (non-ambiguous) between system components
  - (then) components can be separately designed and developed
- Component selection and design
  - search for reusable components (list of changes, if necessary)
  - design new software components

Process Activities - Software design and implementation (4)

Design outputs:

- detailed design documents (for critical systems)
- design diagrams (for a model-driven approach)
- outputs represented in the code (agile methods)

Process activities: Software validation (1)

Software Verification and Validation (V & V)

- **goal**: show that a system both conforms to its specification and meets the expectations of the system customer
- principal validation technique: program testing with simulated data
- may involve checking processes, such as **inspections** and **reviews**

Process activities: Software validation (2)

Testing process stages

1. **Component testing**
   - components: simple entities (e.g., functions or object classes) or coherent groupings of entities
   - components tested independently by developers
2. **System testing**
   - components are integrated to create a complete system
   - may reveal errors from unanticipated interactions between components and component interface problems
   - (sometimes) multistage process: components integrated to form subsystems, individually tested and then integrated
3. **Customer testing**
   - final stage before the system is accepted for operational use
   - tested by the customer with real data

Process activities: Software validation (3)

- Normally, component testing is part of the development process
  - programmers make up their own test data
  - programmers incrementally test the code as it is developed
- Why programmers?
  - knows the component
  - (therefore) is the best person to generate test cases

Process activities: Software evolution (1)

- Flexibility of software (when compared to hardware)
- Historically, a split between the process of software development and the process of software evolution
- distinction increasingly irrelevant

$\Rightarrow$ software engineering as an evolutionary process

More on Software processes (1)

Description of a process must include:

1. **people involved in the process**
   - specification of the roles, and therefore of the responsibilities
   - *examples*: project manager, configuration manager, and programmer

2. **products or deliverables**
   - outcomes are not only software
   - *example*: architectural design $\Rightarrow$ software architecture model

3. **conditions** affecting the sequence of activities:
   - *pre-conditions*, e.g., approval of the requirements by the consumer before architecture design
   - *post-conditions*, e.g., update, if necessary, of the models describing the architecture

More on Software processes (2)

In professional software development **planning** is an inherent part of all processes.

Possible categorization:

- **Plan-driven** processes
  - all of the process activities are planned in advance
  - progress **measured** against the plan
- **Agile** processes
  - **incremental** and continual planning
  - easier to change the process to reflect changing requirements

Note: can be a compromise between the two

Process Paradigms

- General process models
- High-level, abstract descriptions of software processes
- Process frameworks that may be extended/adapted to create more specific software engineering processes
- Also called **Software Development Life Cycle models**
- Examples:
  - Waterfall model
  - Incremental development
  - Integration and configuration

Waterfall Model (1)

- Software development process as a number of stages
- Why *waterfall*? Because of the cascade from one phase to another
- Example of plan-driven
- Stages:
    1. requirements analysis and definition
    2. system and software design
    3. implementation and unit testing
    4. integration and system testing
    5. operation and maintenance

Waterfall Model (2)

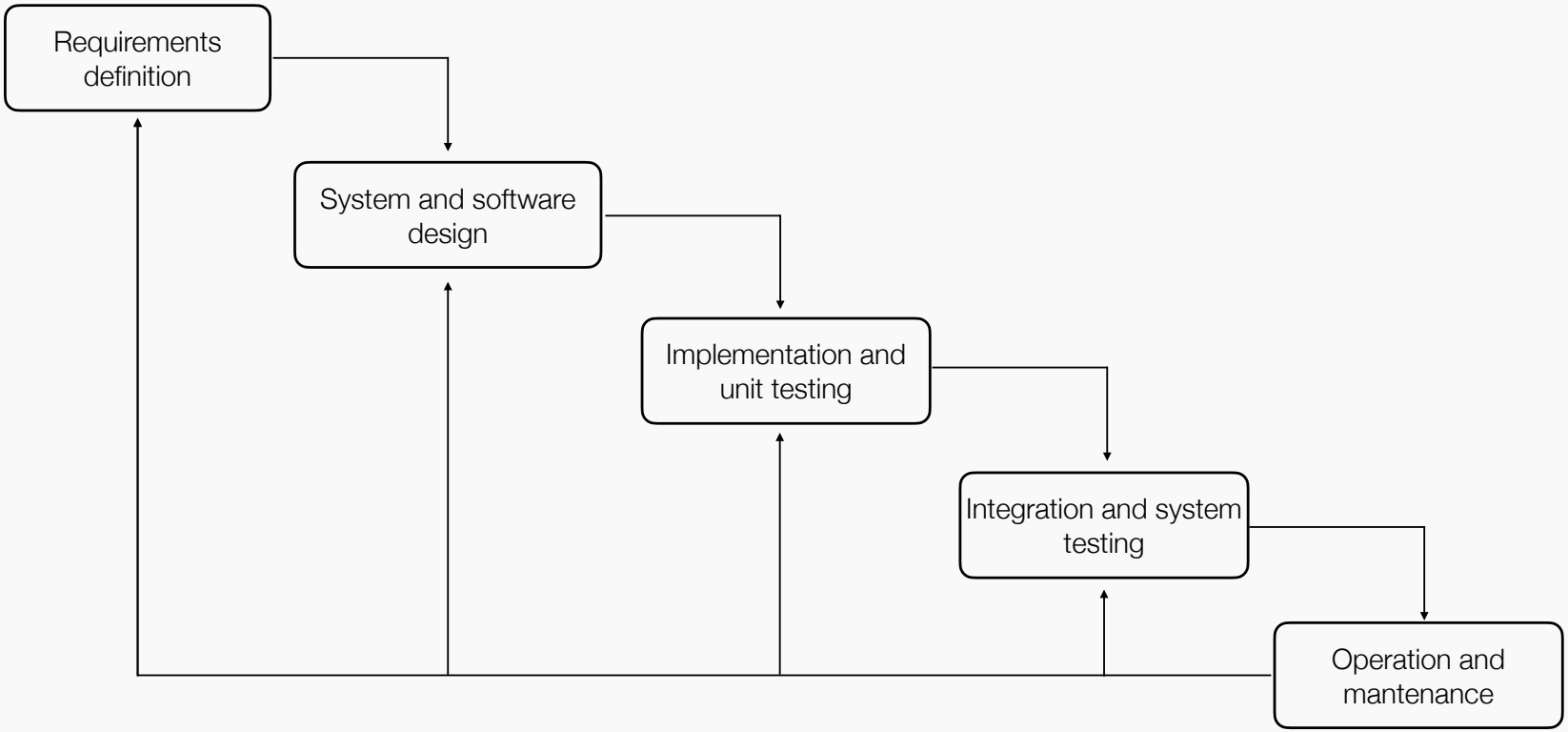


Figure from [Sommerville, 2016]

Waterfall Model (3)

- result of each phase (in principle) is one or more documents that are approved
- following phase should not start until the previous phase has finished
- how to handle new information emerged in a process stage?
  - modification of documents in previous stage
  - changes approval by the customer

Waterfall Model (4)

When the Waterfall model is beneficial?

- Embedded systems
  - inflexibility of hardware does not allow decisions on the software's functionality to be delayed
- Critical systems
  - specification and design documents must be complete for extensive safety and security analysis
- Large software systems developed by several partner companies
  - easier to follow a common model for (hardware and) software
  - complete specifications to allow for the independent development of different subsystems

Incremental development (1)

Main idea

- developing an initial implementation
- getting feedback from users and others
- evolving the software through several versions

$\Rightarrow$ Specification, development, and validation are **interleaved**

Incremental development (2)

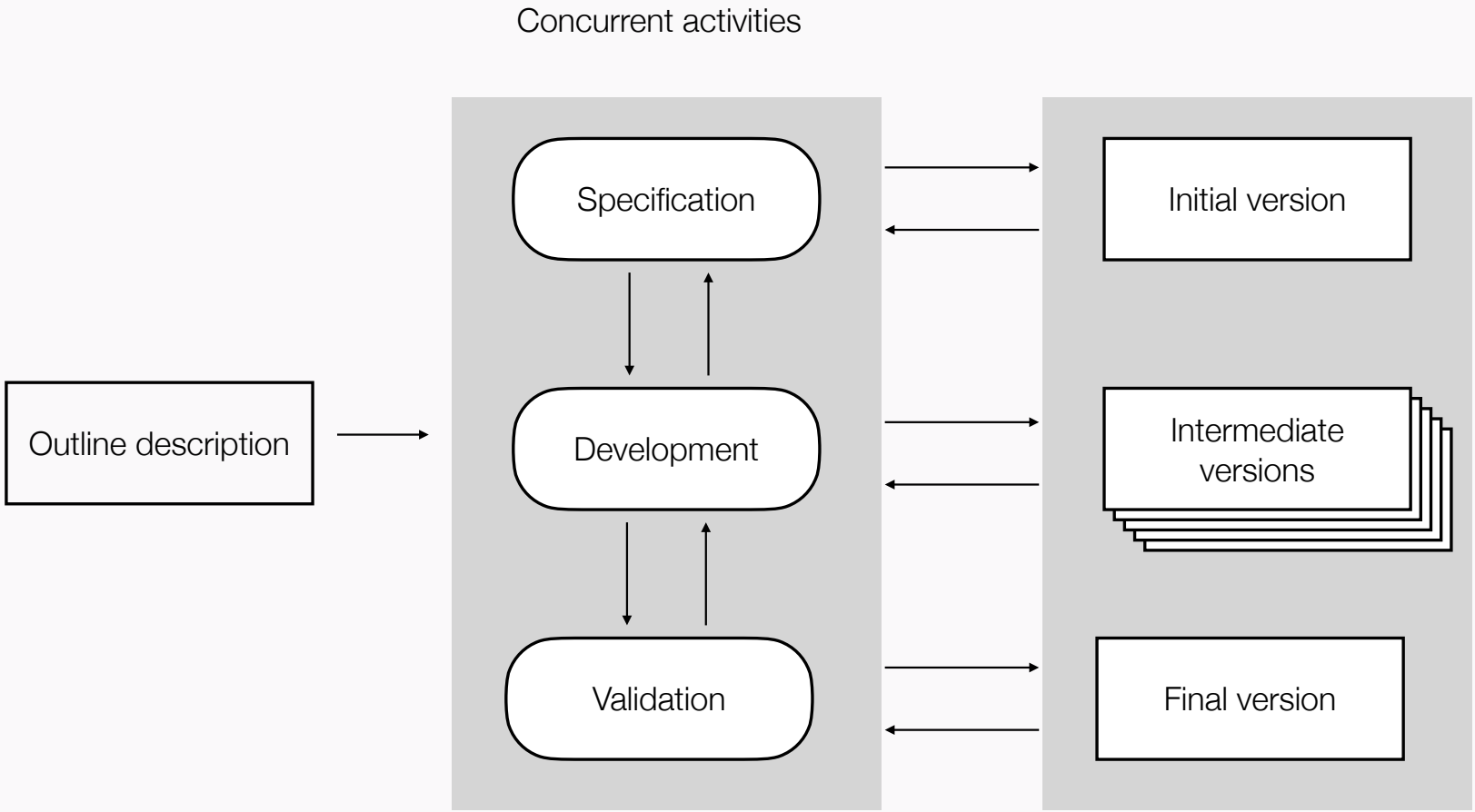


Figure from [Sommerville, 2016]

Incremental development (3)

Incremental developments can be:

- **plan-driven**: system increments identified in advance
- **agile**: early increments identified, but the development of later increments depends on progress and customer priorities

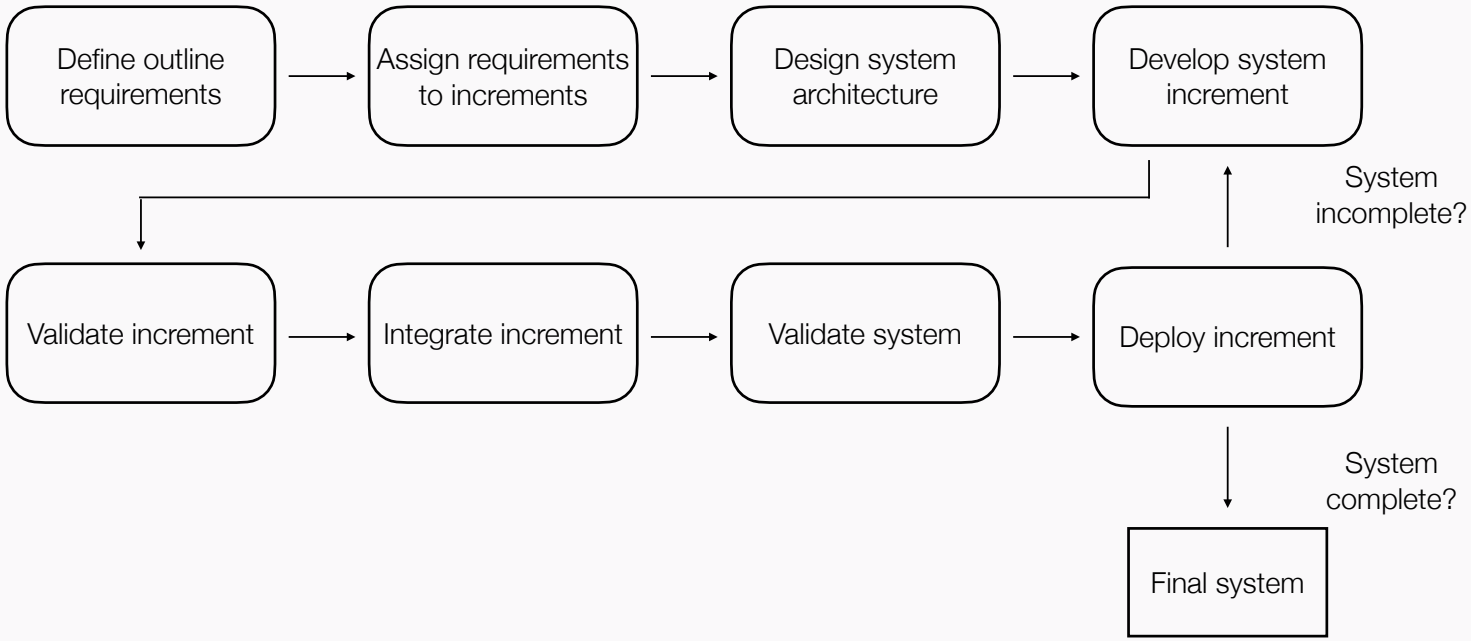
Incremental development (4)

- Advantages
  - reduced cost (VS waterfall) of implementing requirement changes
  - easier to get customer feedback on the development work (e.g., through software demos)
  - early delivery and deployment of useful software
- Disadvantages
  - process is not visible
  - system structure tends to degrade as new increments are added

Note: increment ! = release to the customer
(not necessarily incremental delivery)

Incremental Delivery (1)

- one of the methods for copying with change
- some of the developed increments are delivered to the customer and deployed for use in their working environment



Incremental Delivery (2)

Advantages

- Customers can use the early increments as prototypes and gain experience for later system increments
- Increments are part of the real system
- Customers do not have to wait until the entire system is delivered before they can gain value from it
- As the highest priority services are delivered first and later increments then integrated

  $\Rightarrow$ the most important system services receive the most testing

Incremental Delivery (3)

Problems with incremental delivery

- when the new system is intended to replace an existing system, users need all of the functionality of the old system
  - users unwilling to experiment with an incomplete new system
  - often impractical to use the old and the new systems alongside
- as requirements are not defined in detail until an increment is to be implemented, it can be hard to identify, if any,
  - set of basic facilities that are used by different parts of the system
  - are needed by all increments
- new form of contract, not based on the complete system specification

Integration and configuration (1)

- In software projects, often, some software reuse
- Reuse-oriented approaches rely on
  - reusable software components
  - integrating framework for composition
- Components commonly used:
  - stand-alone application systems configured for use in a particular environment
  - collections of objects developed as a component /package to be integrated with a component framework, e.g., Java Spring
  - Web services available for remote invocation

Integration and configuration (2)

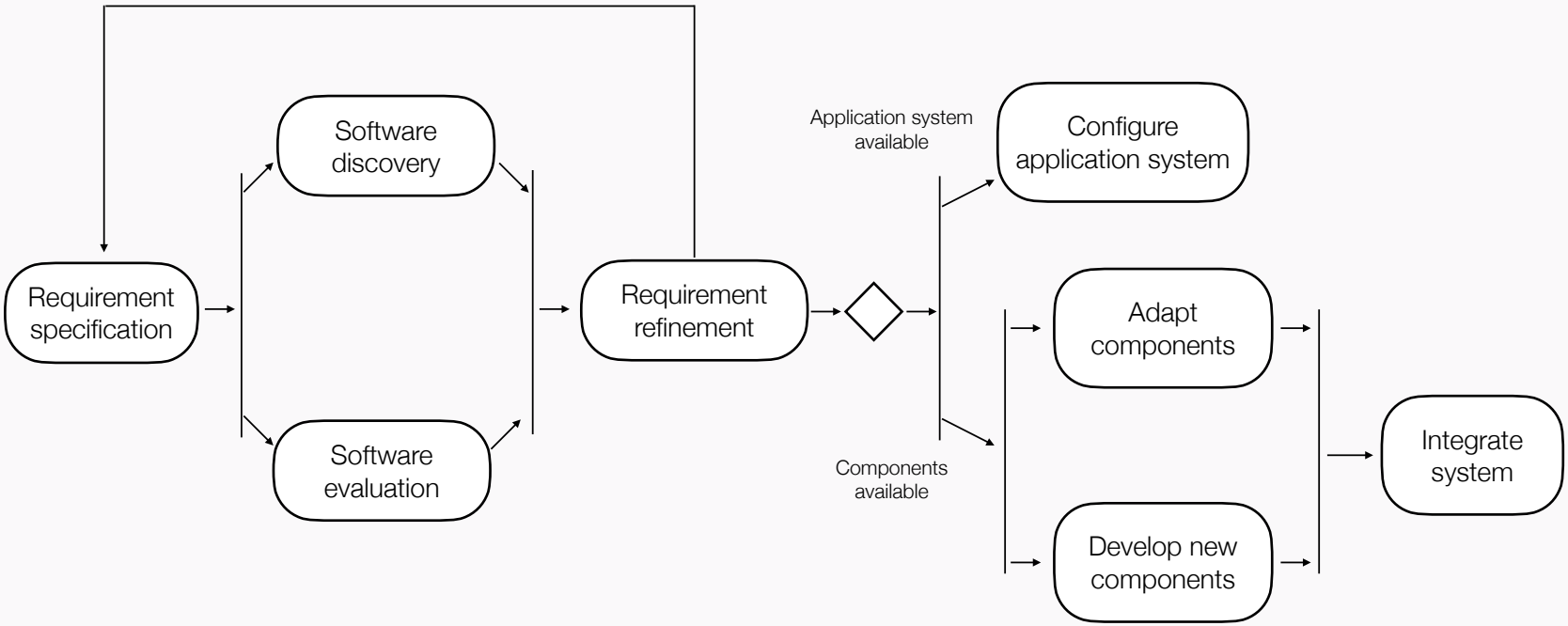


Figure from [Sommerville, 2016]

Integration and configuration (3)

Process phases:

1. **Requirements specification**:
   - proposal of the initial requirements (brief description)
2. **Software discovery and evaluation**:
   - search for components and systems that provide the functionality required
3. **Requirements refinement**
   - requirements refined using information about the discovered reusable components and applications.
   - if requirement changes are not possible, back to step 2

Integration and configuration (4)

4. **Application system configuration**
   - if off-the-shelf application system available, configuration if needed

5. **Component adaptation and integration**
   - no off-the-shelf system available
   - adaptation of the components, if possible
   - development of novel components, if needed

Integration and configuration (5)

- Advantages
  - reducing the amount of software to be developed
  - reducing cost
  - (usually) faster delivery
- Disadvantages
  - sometimes requirements compromises
  - if reusable components are not under the control of the organization, some control over the system evolution is lost (e.g., newer components)

Agile

Why Agile?

- Businesses now operate in a **global**, **rapidly changing** environment
- Software has to be developed quickly to
  - take advantage of new opportunities
  - respond to competitive pressure
- Changing environment
  - impossible to derive a **complete** set of **stable** requirements
  - by the time the software is available for use, often the original reason for its procurement may have changed radically
  - some real requirements may become clear after a system has been delivered

$\Rightarrow$ **Agile methods**

Agile Methods: Common Characteristics (1)

- specification, design and implementation are **interleaved**
- system is developed in a series of **increments**
- extensive **tool support** is used to support the development process

Agile Methods: Common Characteristics (2)

- specification, design and implementation are **interleaved**
  - no detailed system specification
  - minimize documentation
  - “informal” documentation
  - when possible, generated automatically by the programming environment

Agile Methods: Common Characteristics (3)

- system is developed in a series of **increments**
  - End-users and other system stakeholders involved in specifying and evaluating each increment
  - users and stakeholders may propose changes to the software and new requirements, but implemented in later versions
  - small increments
  - new system releases every 2-3 weeks

Agile Methods: Common Characteristics (4)

- extensive **tool support** is used to support the development process
  - automated testing
  - support configuration management
  - system integration
  - automate User Interface (UI) production

Agile Manifesto

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

When use agile methods?

Particularly successful for two kinds of system development

- Product development where a software company is developing a small or medium-sized product for sale
- Custom system development within an organization
  - clear commitment from the customer to become involved in the development process
  - few external stakeholders
  - few regulations that affect the software

Agile development techniques (1)

- Underlying ideas developed around the same time by a number of different people in the 1990s
- For and historical perspective: **Clean Agile** [Martin, 2019]
- One of the most significant approaches: **Extreme Programming**

Agile development techniques (2)

Extreme Programming (XP):

- developed by pushing recognized good practice, such as iterative development, to “extreme” levels
- Several new versions of a system may be developed by different programmers, integrated, and tested in a day
- short time gap between releases of the system

Agile development techniques (3)

Extreme Programming (XP):

- requirements are expressed as scenarios (called **user stories**)
- stories implemented directly as a series of **task**s
- Programmers
  - **work in pairs**
  - **develop tests** for each task before writing the code
- All tests must be successfully executed when new code is integrated into the system

Agile development techniques (4)

Practices in XP reflect the principles of the agile manifesto

- Incremental development is supported through small, frequent releases
- Customer involvement is supported through the continuous engagement of the customer in the development team
- People, not process, are supported through pair programming, collective ownership of the code, and sustainable development
- Change embraced through regular system releases to customers, test-first development, refactoring to avoid code degeneration and continuous integration of new functionality
- Maintaining simplicity is supported by constant refactoring that improves code quality and by using simple designs that do not unnecessarily anticipate future changes to the system

Agile development techniques (5)

- XP is not only a “set of techniques”
- 5 values
  - **Communication**: everyone on a team works jointly at every stage of the project
  - **Simplicity**: Developers strive to write simple code bringing more value to a product, as it saves time and effort
  - **Feedback**: Team members deliver software frequently, get feedback, and improve it according to the new requirements
  - **Courage**: Programmers objectively evaluate their own results without making excuses and are ready to respond to changes
  - **Respect**: Every person assigned to a project contributes to a common goal

References

References (1)


Abbott, M. L. and Fisher, M. T. (2009).

The Art of Scalability: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise.

Addison-Wesley Professional, 1st edition.


Bottcher, E. (2018).

What i talk about when i talk about platforms.

https://martinfowler.com/articles/talk-about-platforms.html.


Johnson, R. (2003).

Expert One-on-one J2EE Design and Development.

Wiley & Sons, erste edition.

References (2)

Joseph, J., Ernest, M., and Fellenstein, C. (2004).

Evolution of grid computing architecture and grid adoption models.

IBM Systems journal, 43(4):624–645.

Martin, R. C. (2019).

Clean Agile.

Robert C. Martin Series. Pearson Education.

Pressman, R. (2009).

Software Engineering: A Practitioner's Approach.

McGraw-Hill, Inc., USA, 7 edition.

References (3)


Richardson, C. (2019).
Microservices Patterns: With examples in Java.
Manning.


Salatino, M. (2023).
Platform Engineering on Kubernetes.
Manning.


Shahrad, M., Balkind, J., and Wentzlaff, D. (2019).
Architectural implications of function-as-a-service computing.
pages 1063–1075. ACM.


Sommerville, I. (2016).
Software Engineering.
Pearson, 10th - global edition edition.

References (4)


Tanenbaum, A. S. and Steen, M. v. (2023).

"Distributed Systems (4th edition).

Maarten van Steen.


Walls, C. (2022).

Spring in Action, Sixth Edition.

Manning.


Zhang, Q., Cheng, L., and Boutaba, R. (2010).

Cloud computing: state-of-the-art and research challenges.

Journal of Internet Services and Applications, 1(1):7–18.

Questions?