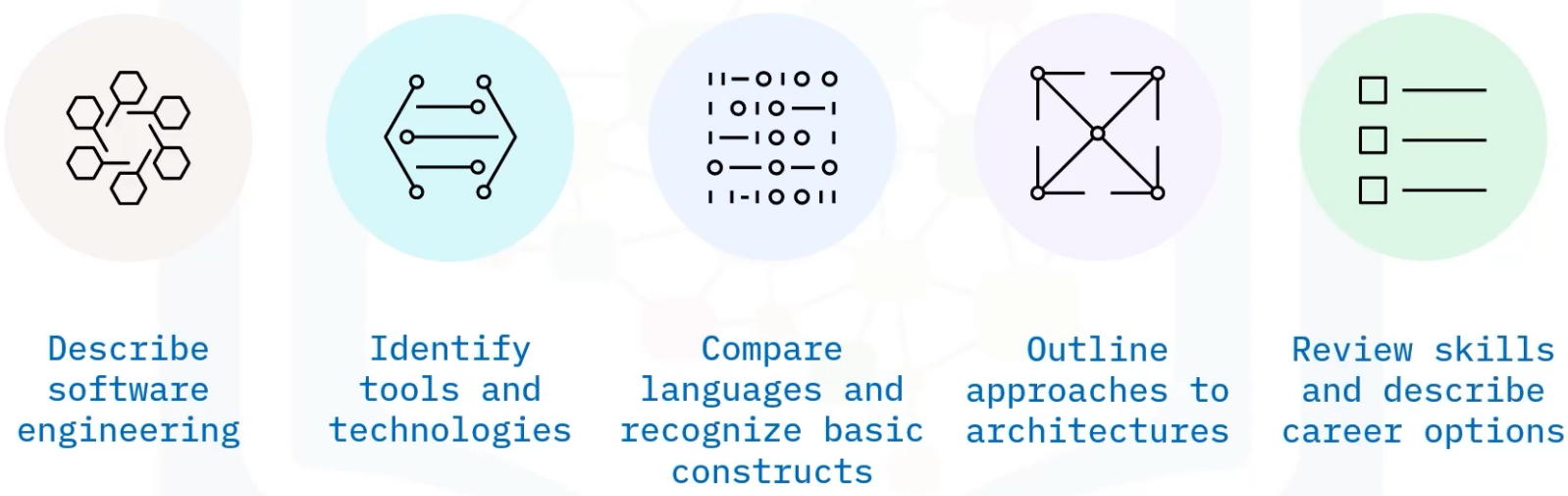
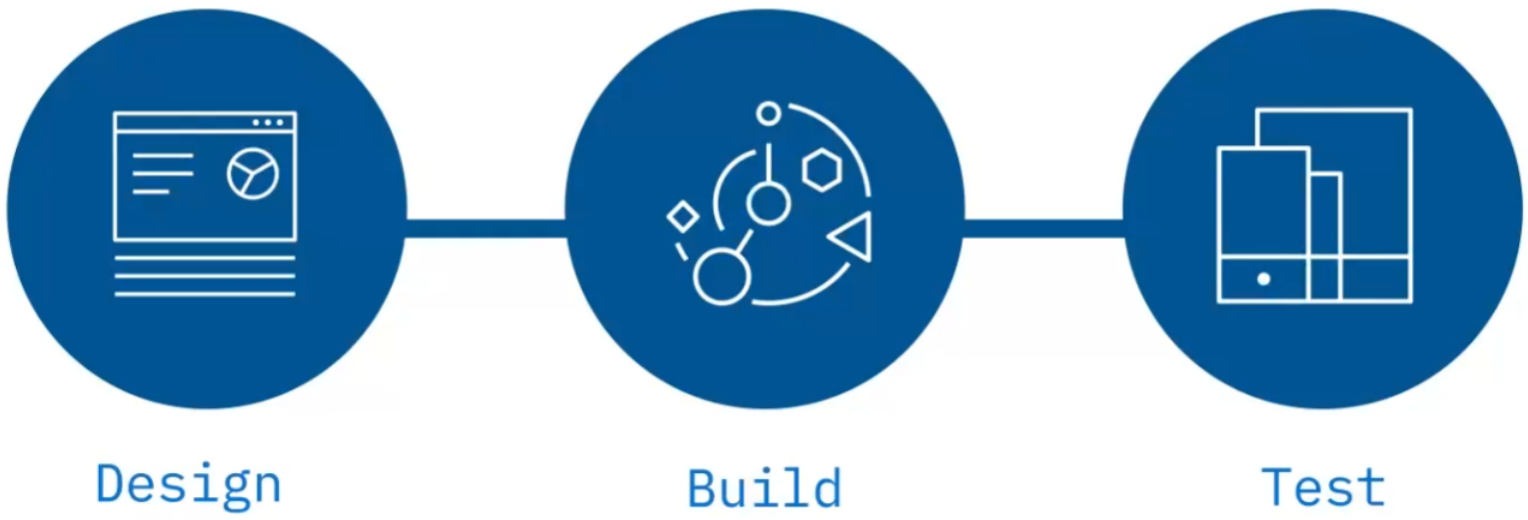
Software Engineering Basics for Everyone

AHUJA, Rav; HUDNUTT, Bethany; JOYNER Lin. Software Engineering Basics for Everyone. edX; IBM Skills Network. 2023



Module 1 - SDLC - The Software Development Lifecycle

200368.01_v4-en.txt



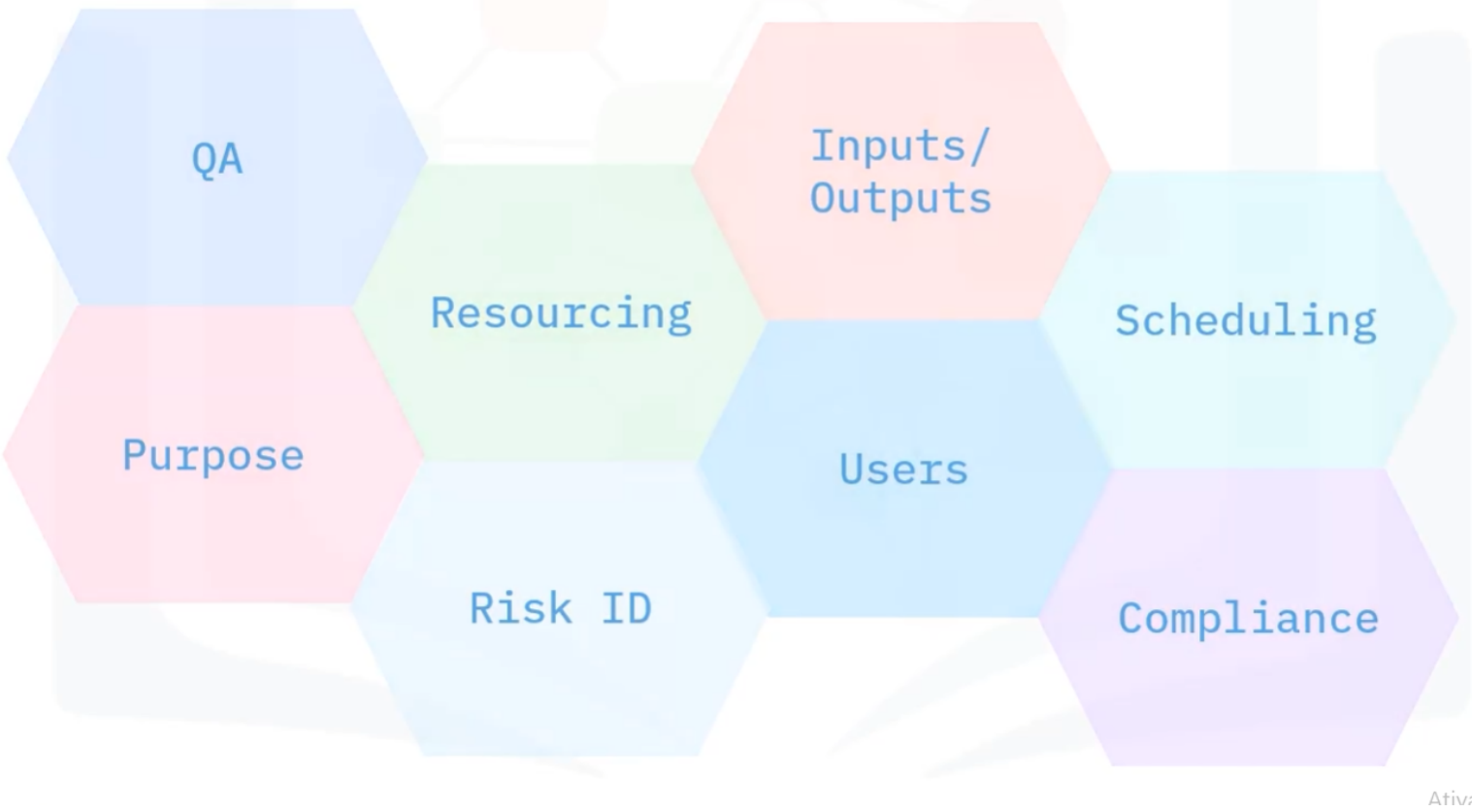
Introduction to the SDLC

200368.02_v2-en.txt

Phases of the SDLC

200368.03_v5-en.txt

Phase 1: Planning



Phase 4: Testing

Code is tested to ensure stability, security, and that it meets requirements from the SRS

Bugs reported, tracked, fixed, and retested

Unit Integration System Acceptance

- The SDLC can be divided into six phases:
  1. Planning: requirement gathering, create SRS
  2. Design: develop architecture, create design doc
  3. Development: coding
  4. Testing: identify and fix bugs
  5. Deployment: code released to production environment
  6. Maintenance: stakeholder feedback, issues / bugs identified, code enhancements suggested

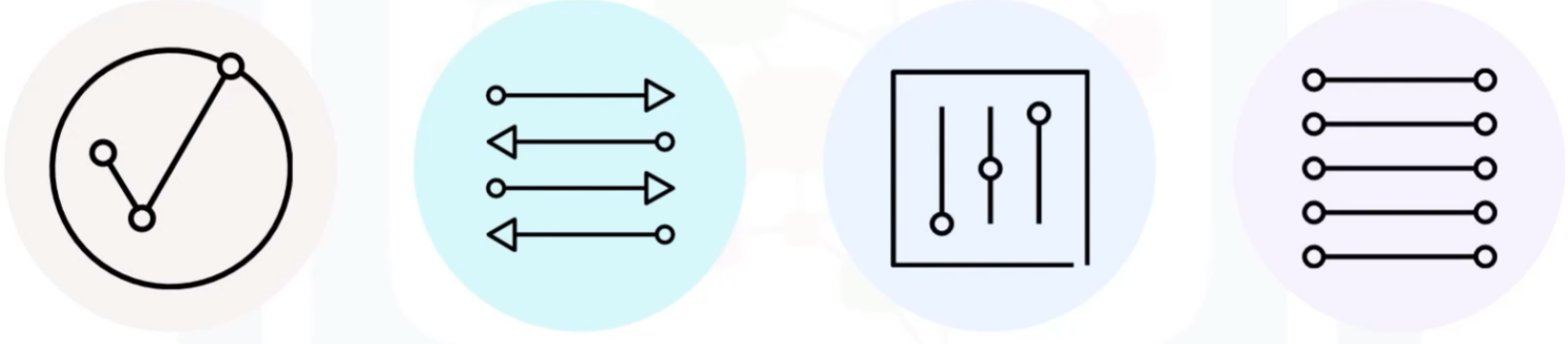
https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-CS0131EN-SkillsNetwork/labs/Module_1/SDLC_phases/index.html

Building Quality Software

200368.04_v1-en.txt

Requirement gathering

Software requirements can be classified into four broad categories:



Functional

External & user interface

System features

Nonfunctional

Design

- Transforming requirements into code

Coding for quality

Quality code must fulfill the intended requirements of the software without defects

- Clean and consistent
- Easy to read and maintain
- Well documented
- Efficient

Releases

Alpha	Beta	General Availability
• Select stakeholders • May contain errors • Preview of functioning version • Design changes may occur	• All stakeholders • User testing • Meets requirements	• Stable • All users

Requirements

200368.05_v4-en.txt

Steps to gathering requirements

- Identifying stakeholders
- Establishing goals and objectives
- Eliciting requirements from the stakeholders
- Documenting the requirements
- Analyzing and confirming the requirements
- Prioritizing

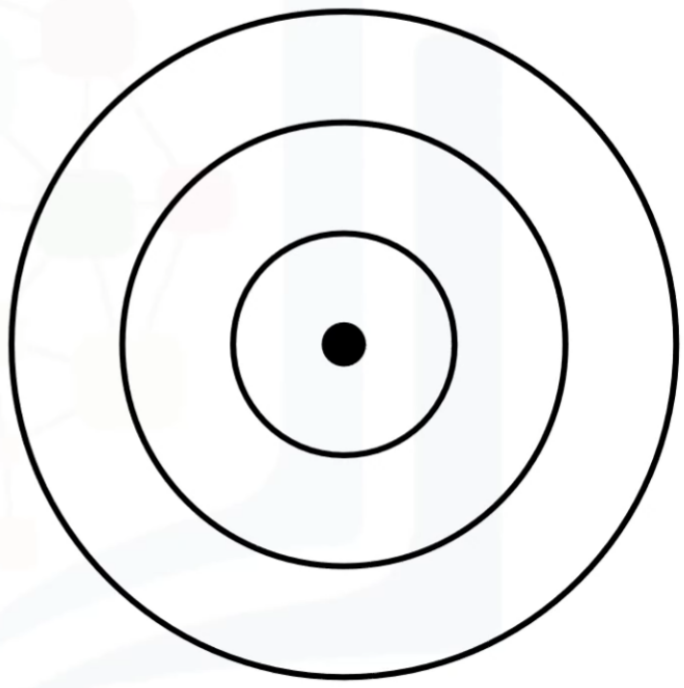
Identifying stakeholders

Key personnel:

- Decision-makers
- End-users
- System administrators
- Engineering
- Marketing
- Sales
- Customer support

Establishing goals and objectives

- **Goals:** broad, long-term achievable outcomes
- **Objectives:** actionable, measurable actions that achieve the goal



Eliciting, documenting, confirming

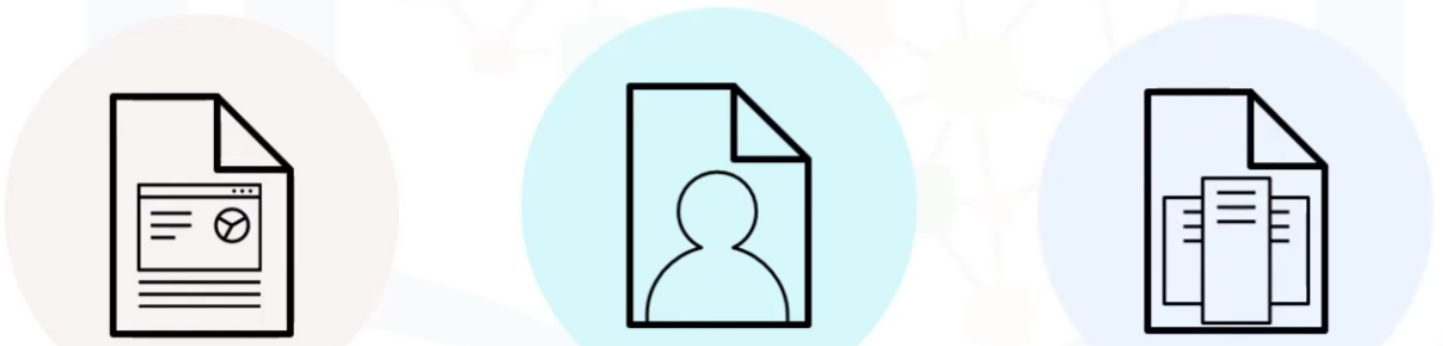
- **Elicit**
  - Surveys
  - Questionnaires
  - Interviews
- **Document**
  - Align with goals and objectives
  - Easily understood
- **Confirm**
  - Consistency
  - Clarity
  - Completeness

Prioritizing

- Must-have
- Highly desired
- Nice to have

Requirements documentation

- Software requirements specification (SRS)
- User requirements specification (URS)
- System requirements specification (SysRS)



Software requirements specification (SRS)

- Captures functionalities the software should perform
- Establishes benchmarks / service-levels for performance
- Purpose and scope
- Constraints, assumptions, dependencies
- Requirements
  - Functional
  - External interface
  - System features
  - Non-functional

SRS: Requirements

- **Functional:** functions of the software
- **External:** users and interactions with other hardware or software
- **System features:** functions of the system
- **Non-functional:** performance, safety, security, quality

- **User stories:**
  - Who is the user?
  - What is the function that needs to be performed?
  - Why does the user want this functionality?

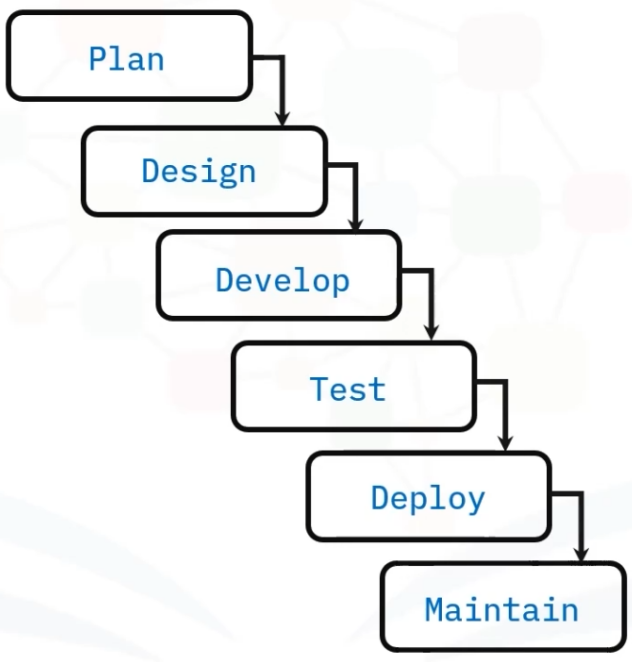
System Requirement Specification (SysRS)

- Outlines requirements of the system
- Broader than an SRS
- Contains:
  - System capabilities
  - Interfaces and user characteristics
  - Policy
  - Regulation
  - Personnel
  - Performance
  - Security
  - System acceptance criteria
  - Hardware expectations

Software Development Methodologies

M1L2-200368.06_v2-en.txt

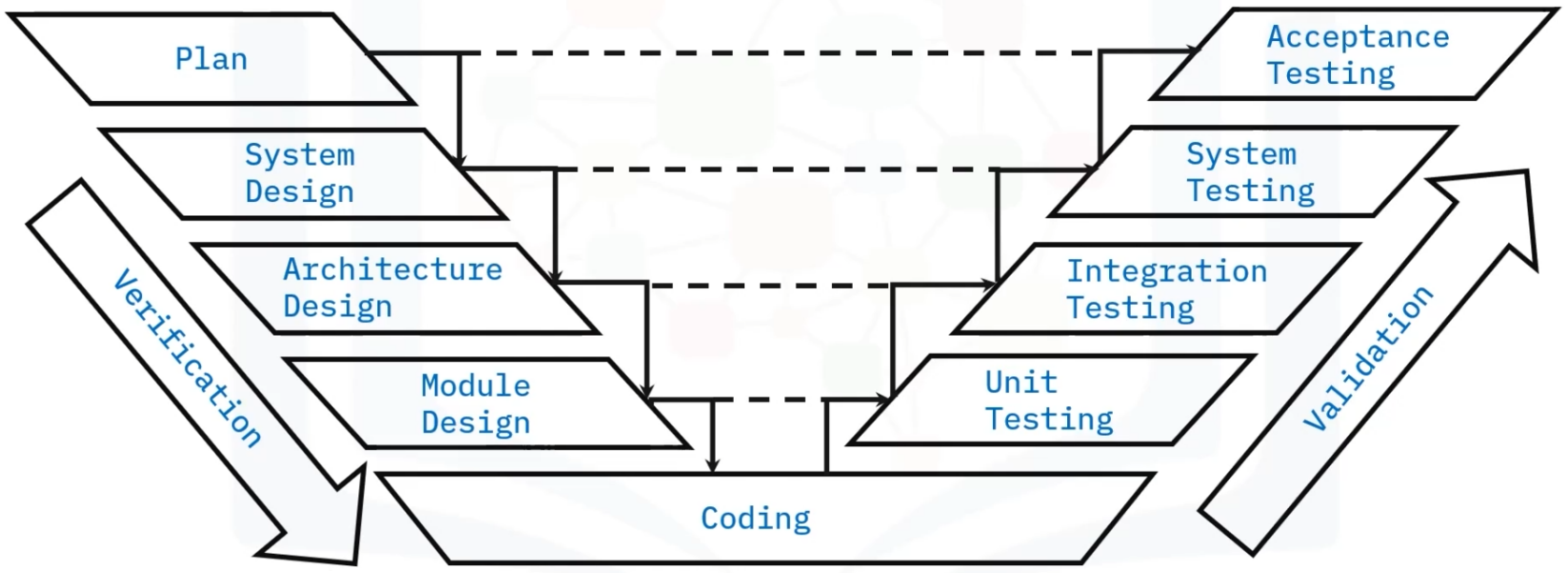
Waterfall method



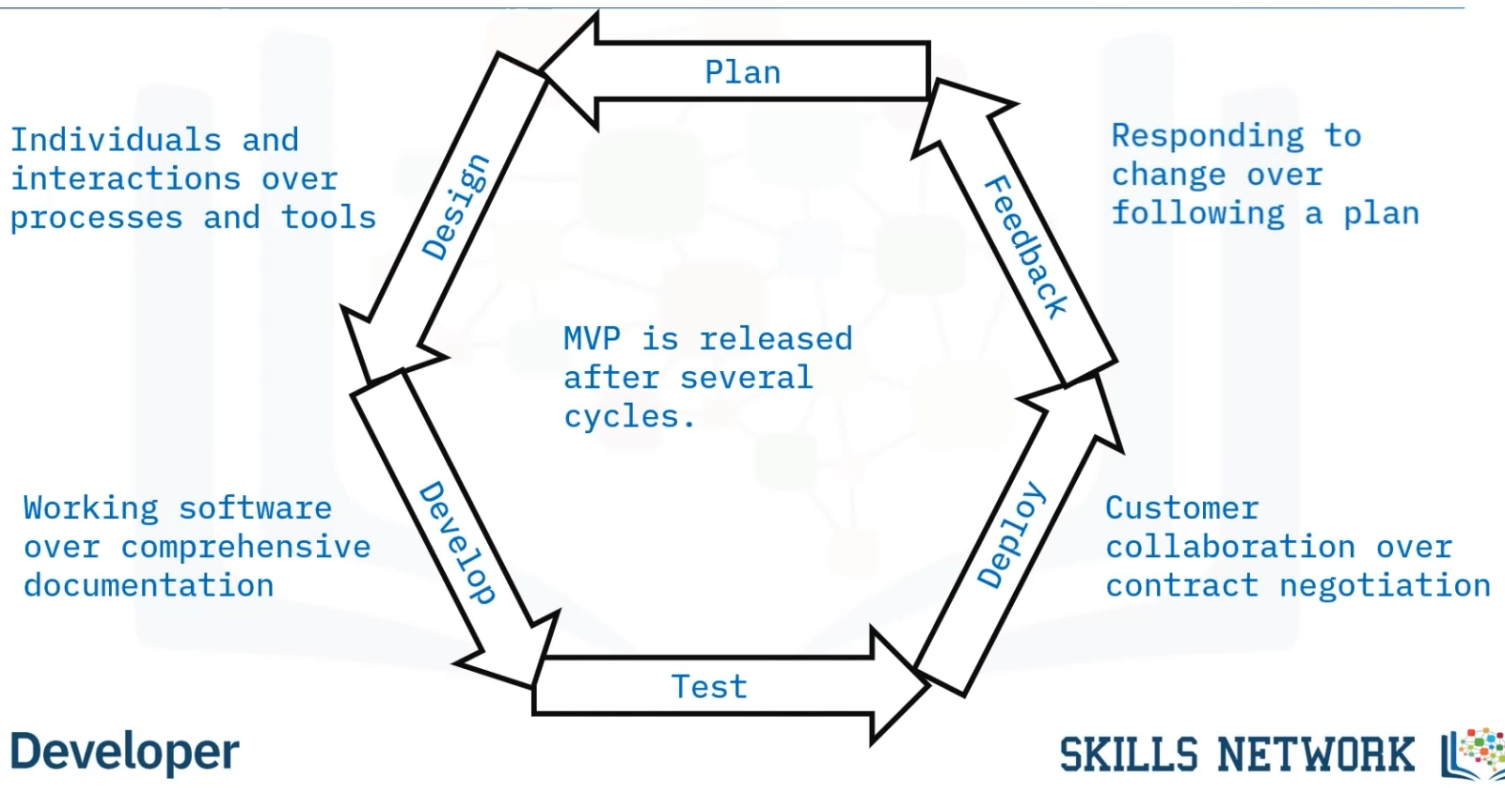
IBM Developer

SKILLS NETWORK

V-shape model



Agile



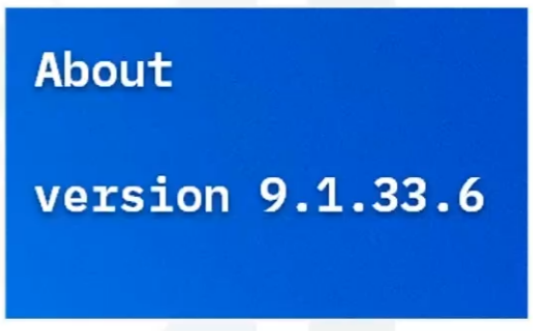
Software Versions

ITC.200303.29_v3-en.txt

What do version numbers mean?

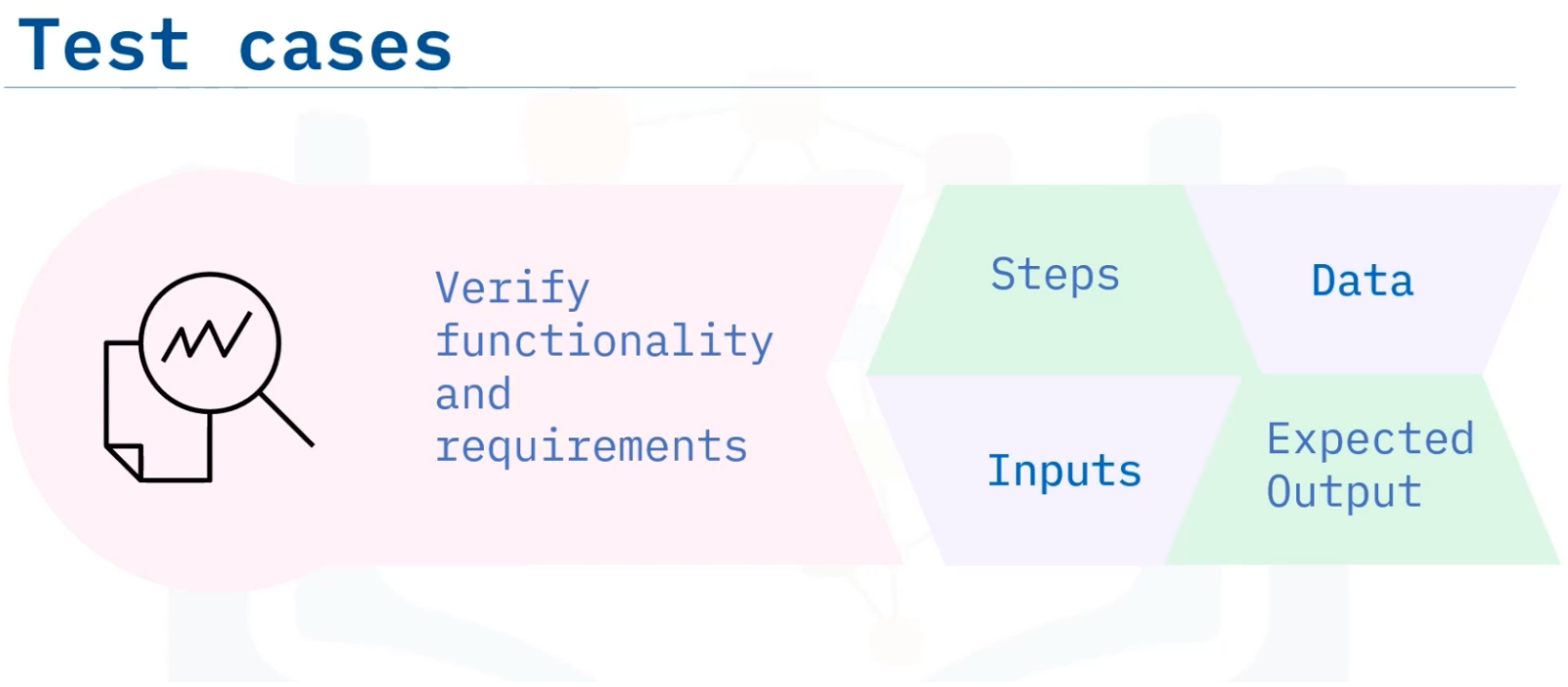
Some version numbers follow the semantic numbering system and have 4 parts separated by a period

- The first number indicates major changes to the software, such as a new release
- The second number indicates that minor changes were made to a piece of software
- The third number in the version number indicates patches or minor bug fixes
- The fourth number indicates build numbers, build dates, and less significant changes

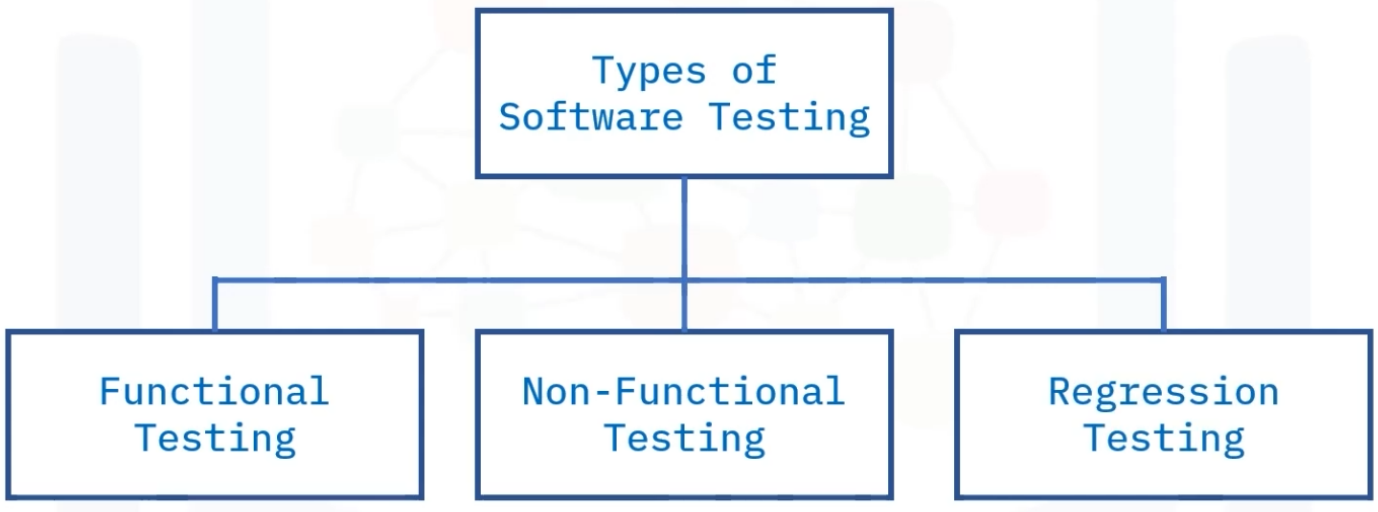


Software Testing

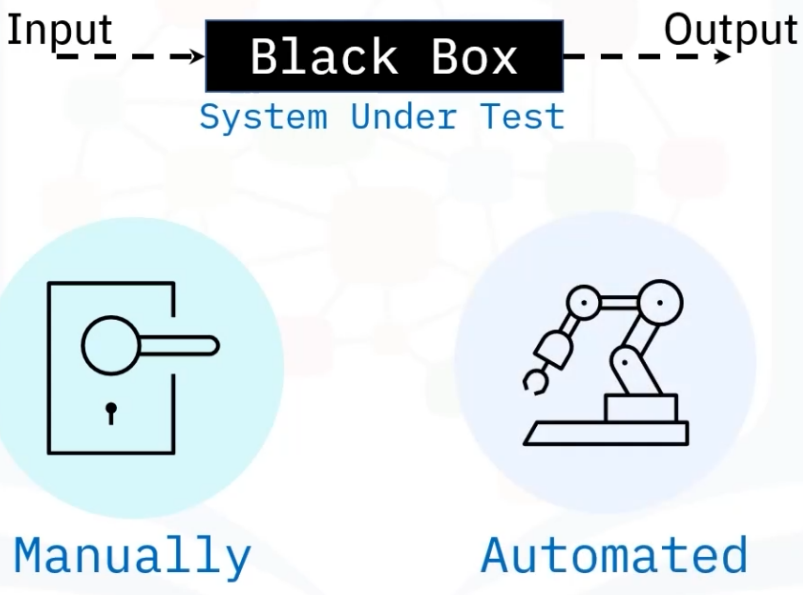
200368.07_v2-en.txt



Three types of testing



Functional testing

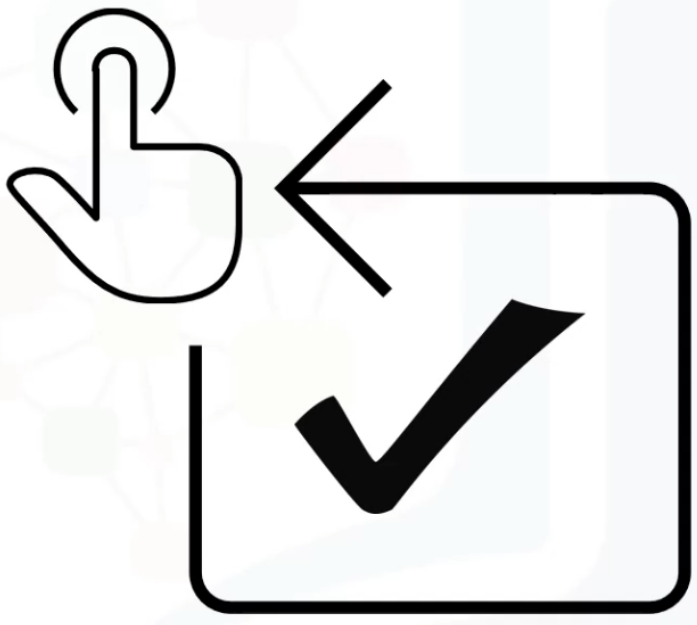


Non-functional testing attributes

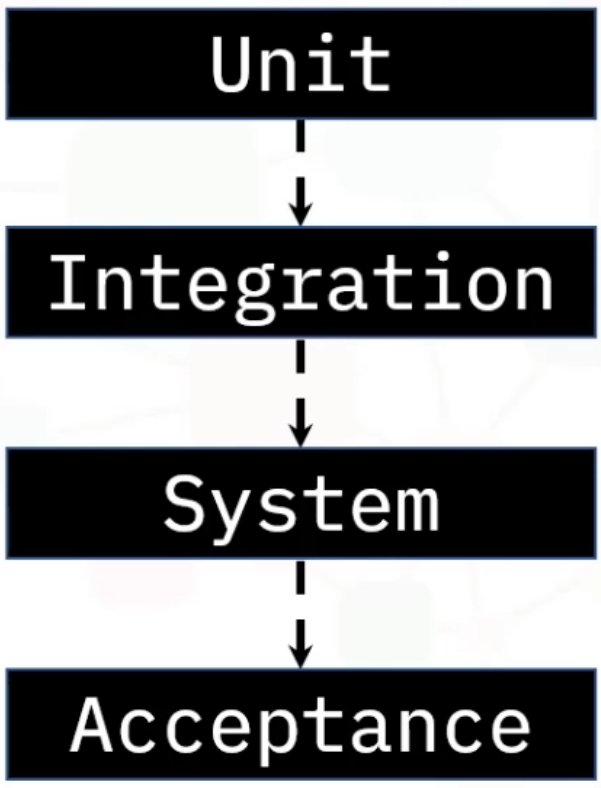
- Performance
- Security
- Scalability
- Availability

Regression testing

- Confirms changes don’t break the application
- Occurs after fixes such as a change in requirements or when defects are fixed



Testing levels



- There are three categories of testing: functional, non-functional, and regression
- Unit testing verifies small, independent chunks of code
- Integration testing looks for errors when two or more small chunks of code are combined
- System testing validates the system as a fully completed product
- Acceptance testing verifies implementation of user requirements and business processes

Software Documentation

200424.01_v1-en.txt

- Documentation can be in written, video, or graphical format
- Process documentation describes how to complete a task
- Product documentation relates to how a product functions
- Types of product documentation include requirements, design, technical, QA, and user
- SOPs are written instructions detailing an organization-specific procedure

Roles in Software Engineering Projects

200424.09_v3-en.txt

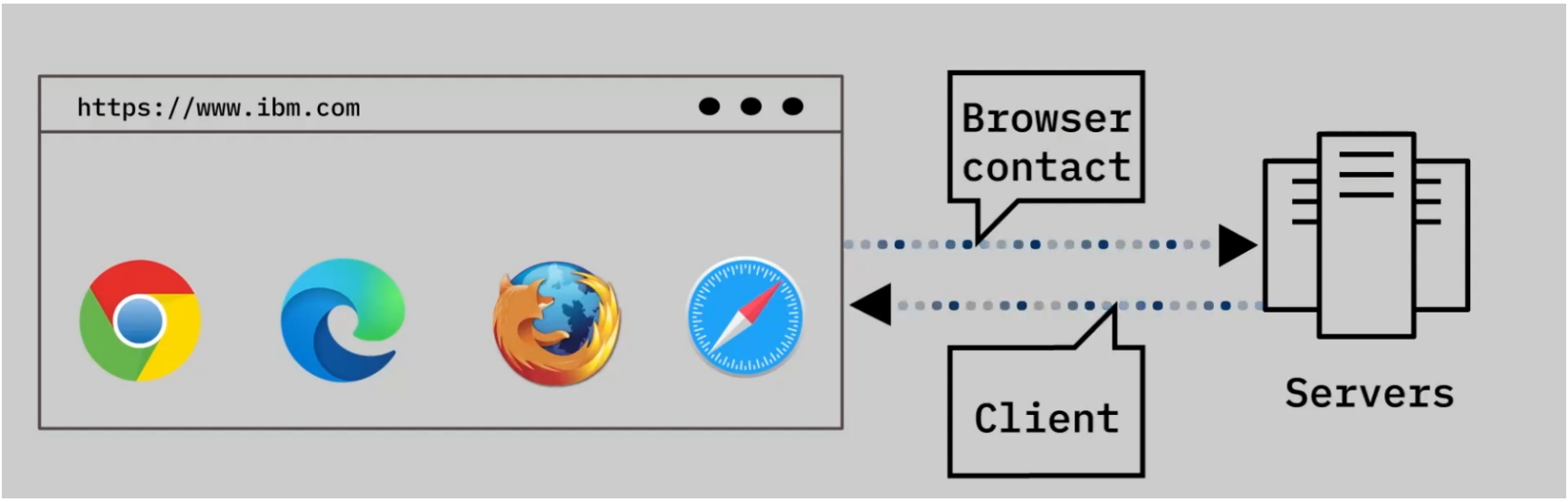
Software development project roles

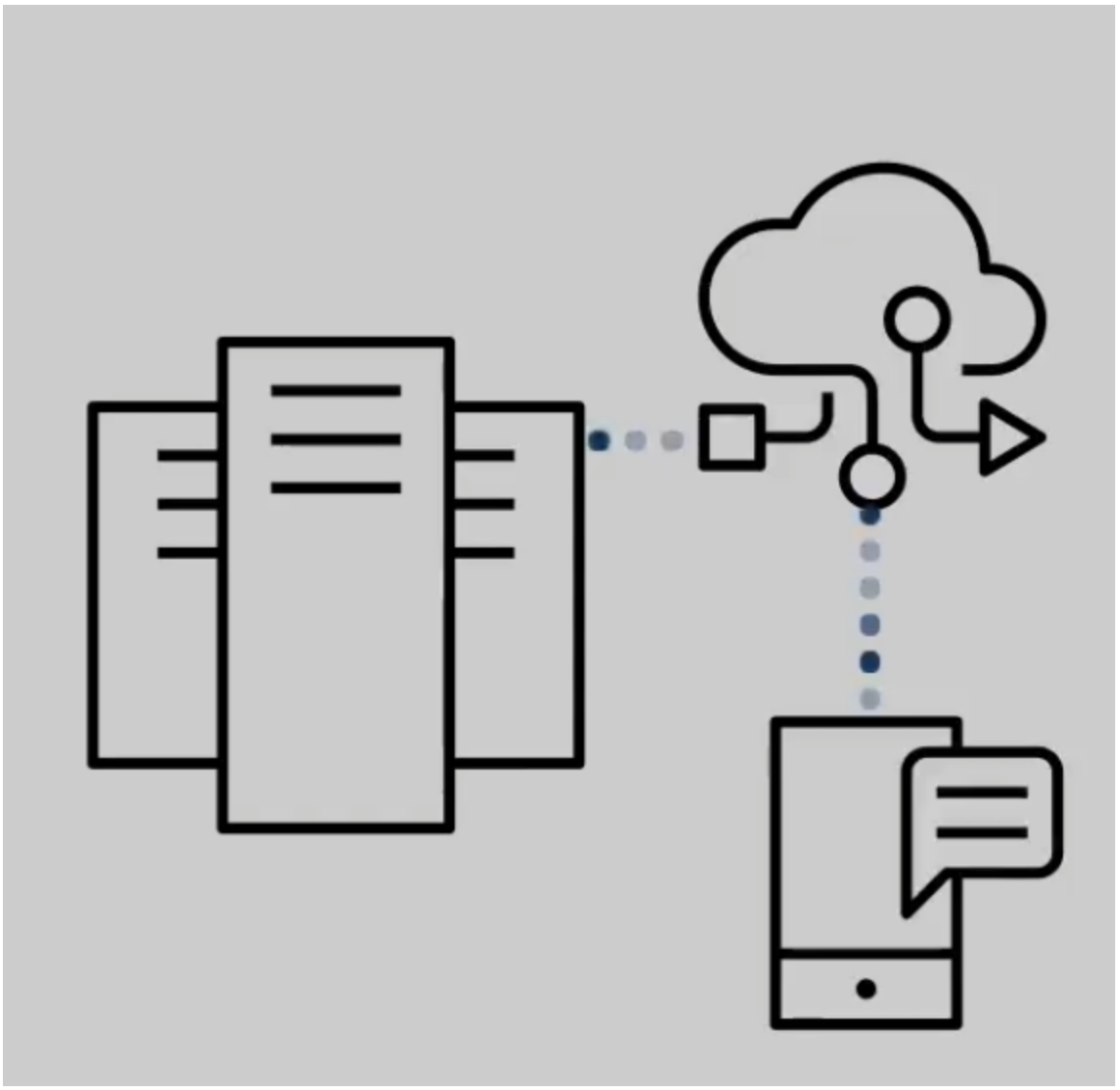
1. Project manager / Scrum master
2. Stakeholder
3. System / Software architect
4. UX Designer
5. Developer
6. Tester / QA engineer
7. Site reliability / Ops engineer
8. Product manager / Product owner
9. Technical writer / Information developer

Module 2 - Introduction to Software Development

Overview of Web and Cloud Development

CAD-1.1.2_v2---Overview-of-Web-and-Cloud-Dev-en.txt





Integrated Development Environment or IDE.

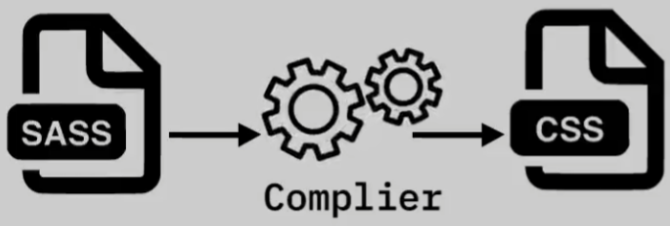
Learning Front-End Development

CAD_1.1.3_v2_-Learning-Front-End-Dev-en.txt

Hypertext Markup Language (HTML)

Cascading Style Sheets (CSS)

A front-end development language is Syntactically Awesome Style Sheets (SASS)

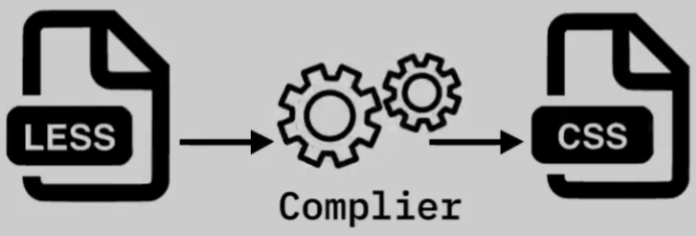


An extension of CSS that is compatible with all versions of CSS.

SASS enables you to use things like variables, nested rules, inline imports to keep things organized.

SASS allows you to create style sheets faster and more easily.

Learner Style Sheets (LESS)



LESS enhances CSS, adding more styles and functions.

It is backwards compatible with CSS.

Less.js is a JavaScript tool that converts the LESS styles to CSS styles.

The Importance of Back-End Development

CAD_1.1-en.txt

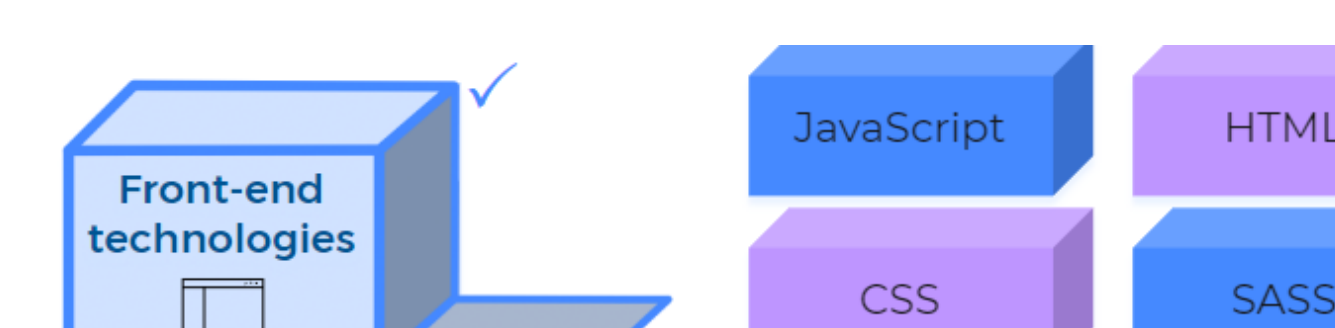
- Back-end developers focus on the functionality that keeps websites, cloud apps and mobile apps up and running
- Back-end development covers a wide range of technologies:
  - Managing user accounts
  - Authentication and authorization
  - Secure storage and handling of sensitive
- Back-end developers work with databases, retrieving, processing and storing data

Teamwork and Squads

200368.10_v3-en.txt

Squads

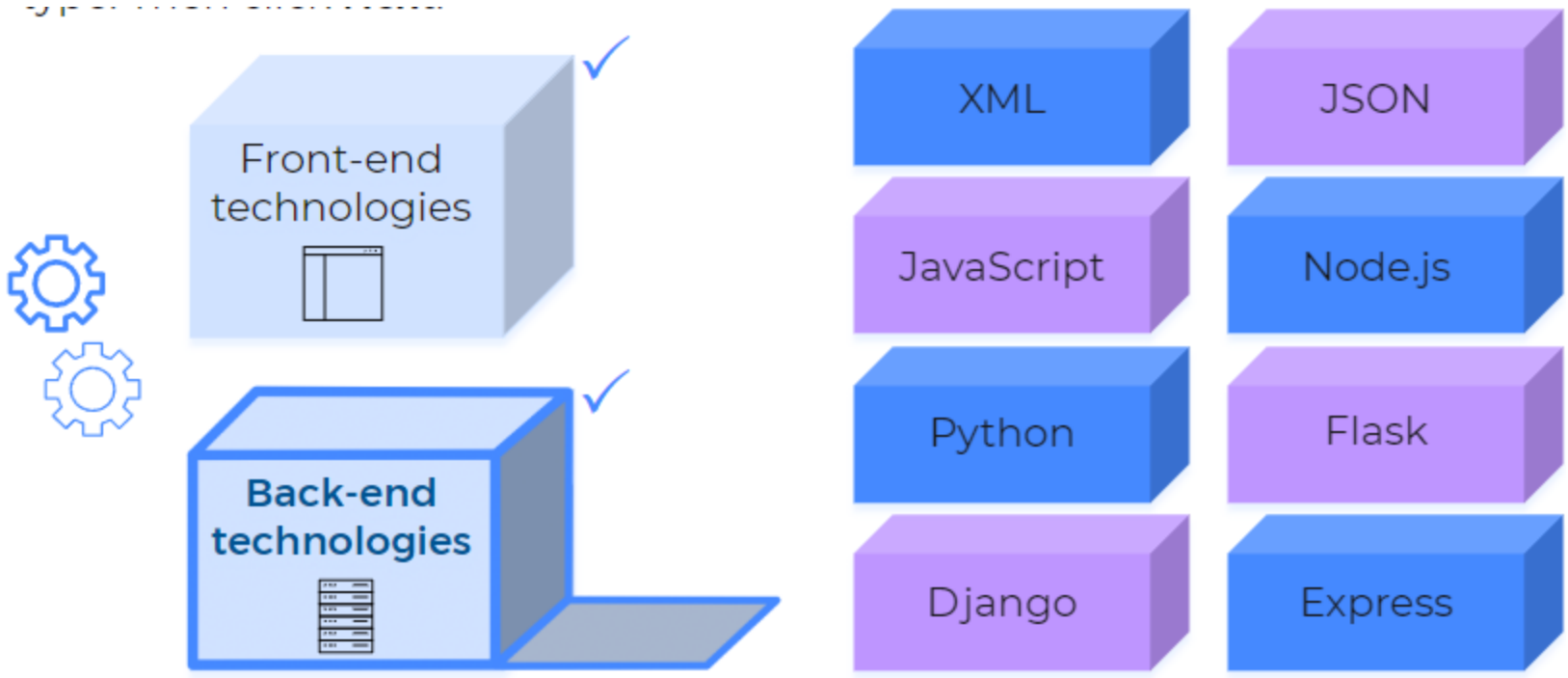
- Used in agile development
- Small team of up to 10 developers:
  - Squad leader
  - Product developers
  - UX developers or designers
- May practice pair programming



Pair Programming

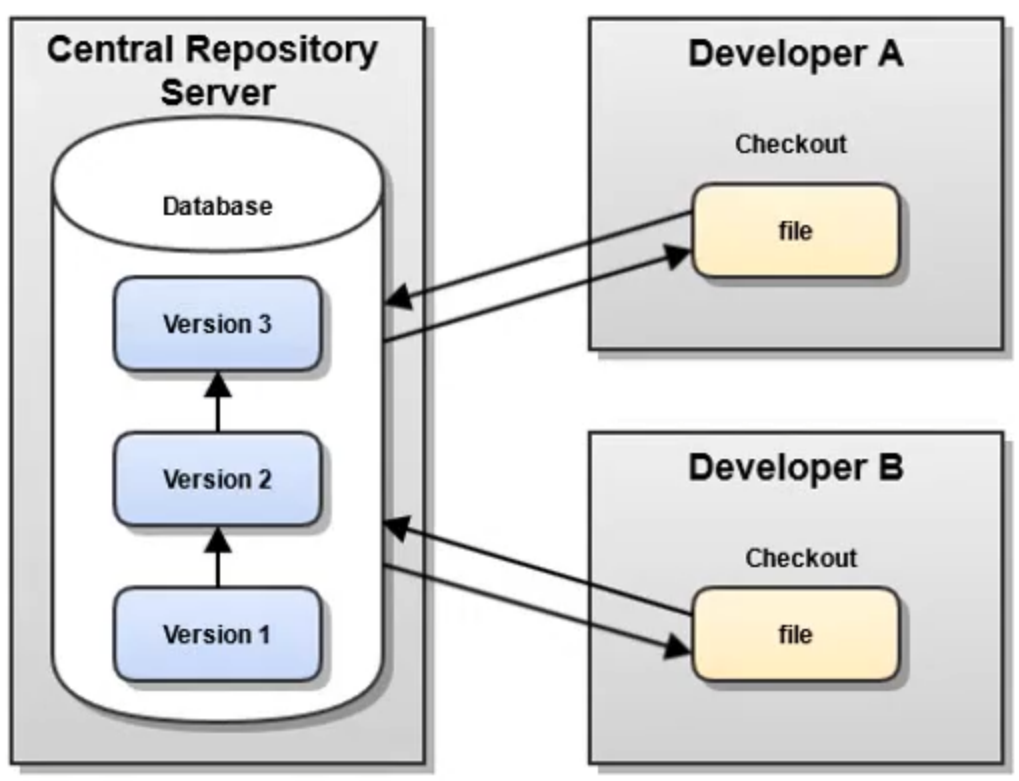
200424.06_v1-en.txt

Categorizing Front- and Back-End Technologies



Introducing Application Development Tools

CAD_1.1-en.txt



[O que o Nilo pedia em relação aos documentos]

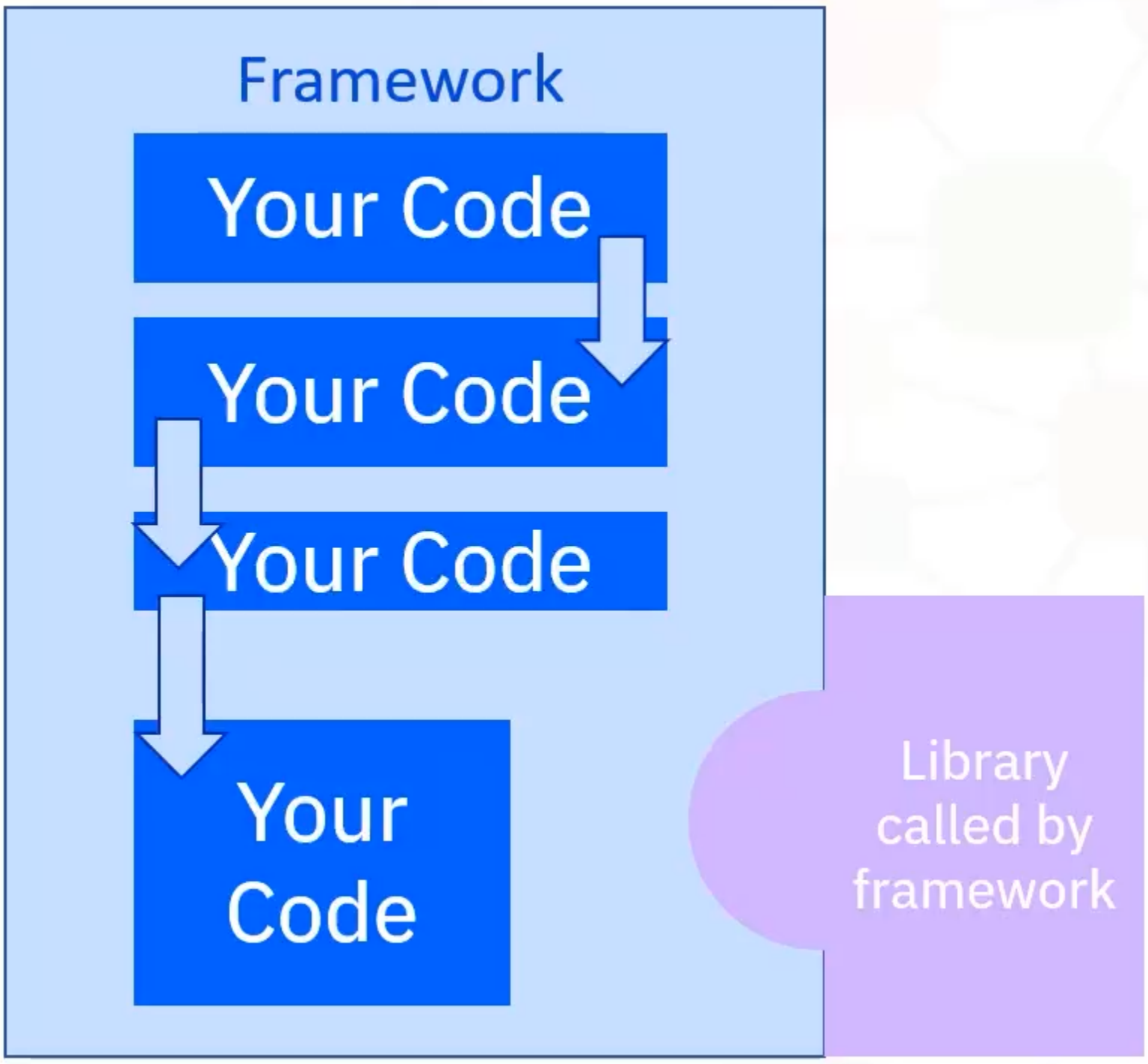
Libraries

Email Validator

Apache Commons™
http://commons.apache.org/

Frameworks

- Provide a standard way to build and deploy applications
- Act as a skeleton you extend by adding your own code
- Dictate the architecture of the app
- Call your code
- Allow you less control than libraries



More Application Development Tools

CAD_1.1-en.txt

Examples of Build Tools

webpack

JavaScript module bundler

BABEL

JavaScript compiler

Package Managers by Platform

Linux

- Debian Package Management System
- Red Hat Package Manager

Android

- Package Manager

Windows

- Chocolatey

macOS macOS

- Homebrew
- MacPorts

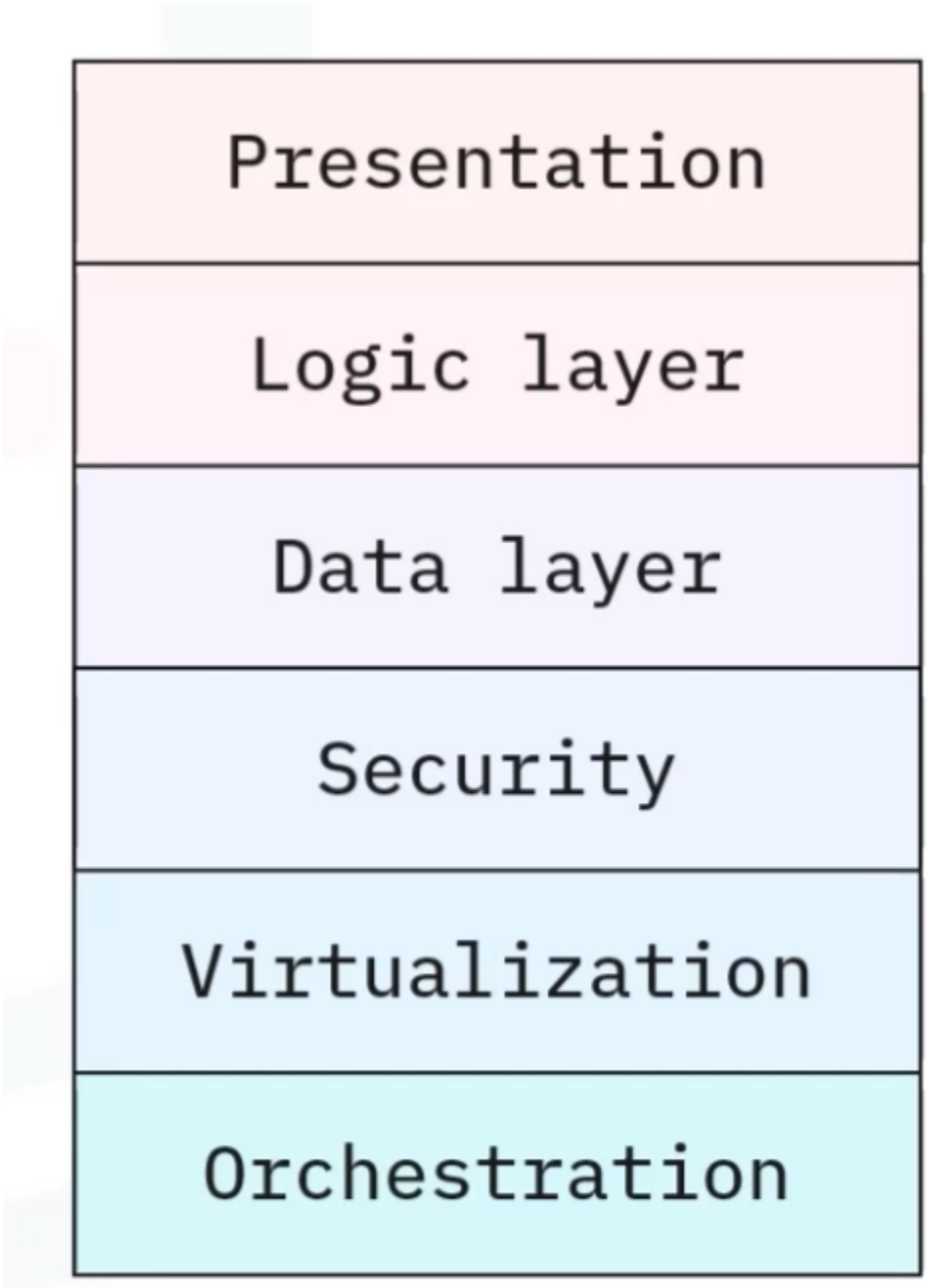
Cloud Application Package Managers

- Libraries and utility code are managed with the cloud application package managers

Node.js / Javascript	Java	Ruby	Python
• npm	• Gradle • Maven	• RubyGems	• Pip • Conda

Introduction to Software Stacks

200424.02_v1-en.txt

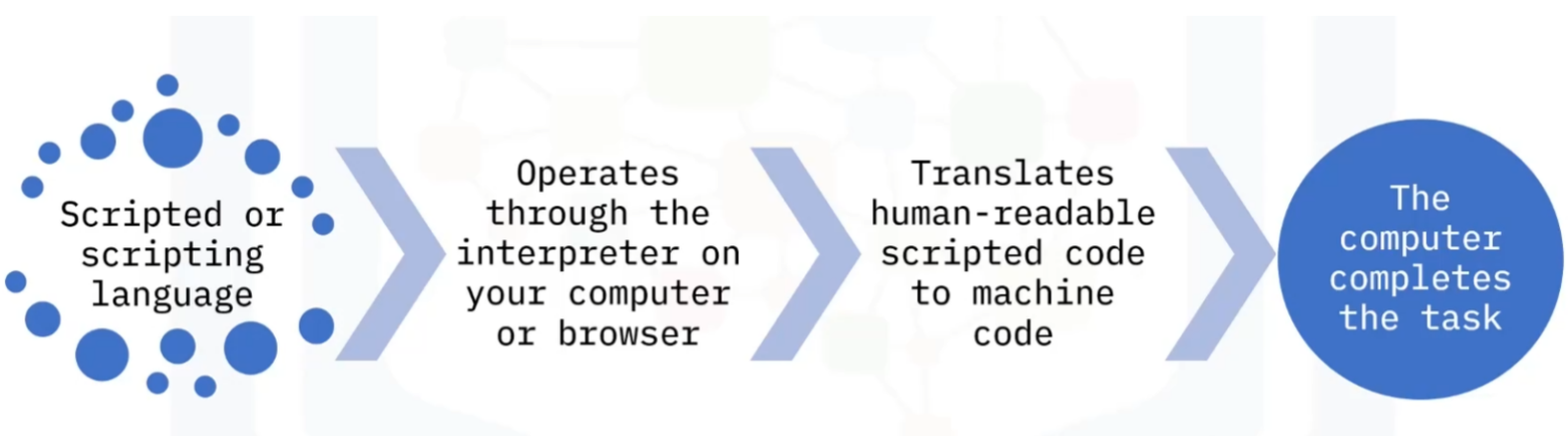


- Software stacks are a combination of technologies for creating applications and solutions
- Software stacks can range from simple three-layered systems to many layers
- There are numerous types of software stacks for different developers and environments
- The MEAN, MERN, and MEVN stacks use JavaScript throughout and are suited to large-scaled apps using relational data
- The LAMP stack on Linux works well for non-relational data

Module 3 - Basics of Programming

Interpreted and Compiled Programming Languages

ITC.200303.14_v1-en.txt



Interpreted programming examples

```
<p>Before the script...</p>

<script>
  alert( 'Hello, world!' );
</script>

<p>...After the script.</p>

</body>

</html>
```

JavaScript
simpler scripting language that runs through web browser interpreter

```
print("Hello, World!")
```

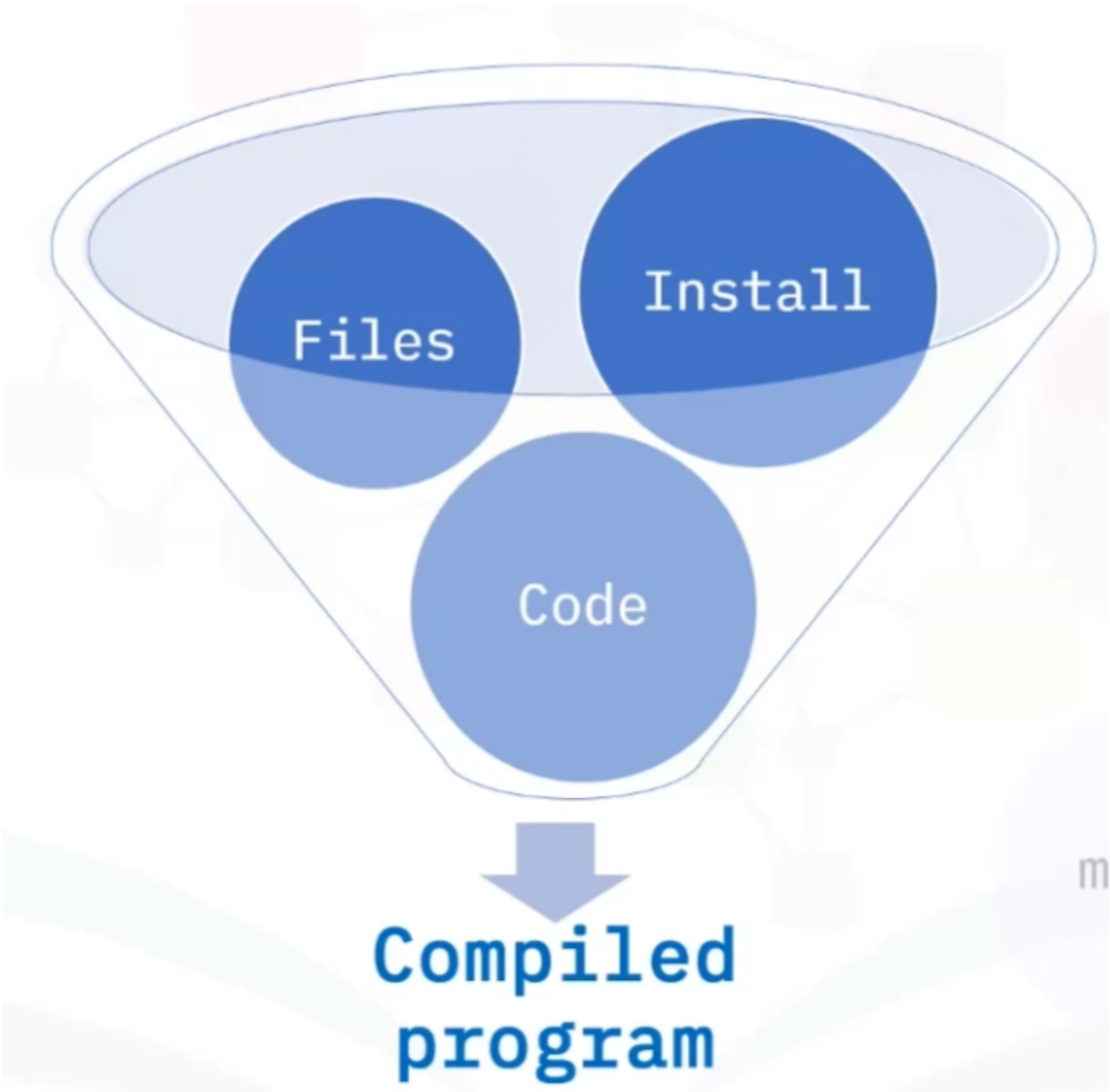
Python
popular because it is easy to learn and use for developers

```
isBusy = false
d = script.Parent.Parent.door

function useDoor(hitPart)
    if isBusy then
        return
    end
    isBusy = true
    if d.Transparency <= 0 then
        -- it's closed. open it.
        while d.Transparency < 1 do
            d.Transparency = d.Transparen
            wait(0.1)
        end
```

Lua
general purpose game scripting language, easy to learn and use

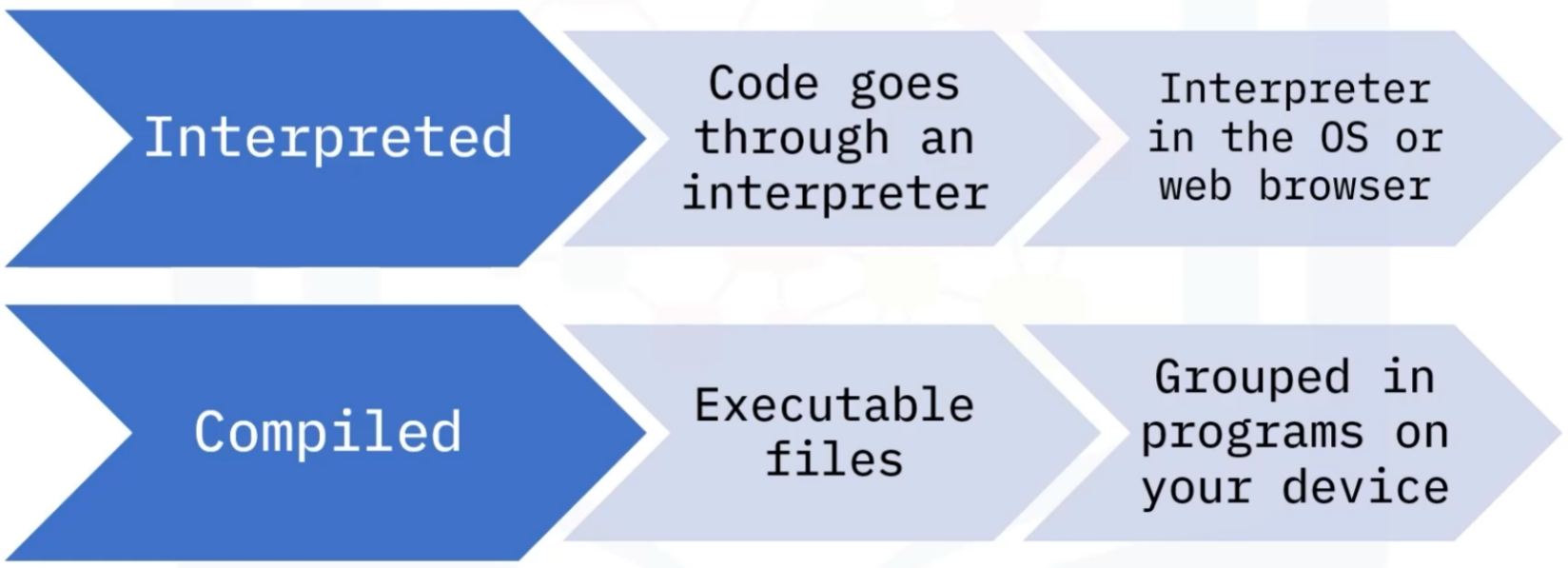
```
<html>
 <body>
  <h1> Hello, world! </h1>
 </body>
</html>
```

HTML
markup language used for formatting web pages



Comparing Compiled and Interpreted Programming Languages

ITC.200303.15_v2-en.txt



Interpreted versus compiled

Interpreted			Compiled		
Available across multiple platforms or in the cloud	Easier to learn and use	Better for websites and smaller, repeated processes	Same operating system coded in the same language	More difficult to learn and use	Better for larger tasks, like an OS

- Interpreted programming languages create source code that runs through an interpreter in your device’s operating system or on your web browser
- Compiled programming languages create executable files that are grouped in programs on your device
- Compiled programming languages like C and Java are used to write larger programs like operating systems and other executable files
- Interpreted programming languages like Python and HTML are used to write code that can complete repetitive tasks within a web browser or a computer

Query and Assembly Programming Languages

ITC.200303.16_v3-en.txt

SQL Statement	Task
SELECT	Extract data from a database. All request queries start with a SELECT statement.
WHERE	Filter the query criteria to return results that match a set condition (used in combination with SELECT).
ORDER BY	Define the sort order of returned results.
UPDATE	Modify existing data in a database.
INSERT INTO	Add new data records to a database.
ALTER TABLE	Add or remove columns in a database.
CREATE	Create a new object in a database. Object options are DATABASE, TABLE, INDEX, and VIEW.
DROP	Delete an object from a database. Object options are the same as above.
SUM	Return total sum of a column of numeric data.
COUNT	Return number of rows that match a condition.
AVG	Return average value of a column of numeric data.

Query statement examples

- SELECT * FROM suppliers;
- SELECT name FROM suppliers WHERE name = 'Mike';
- CREATE DATABASE products;
- DROP TABLE suppliers;
- ALTER TABLE suppliers;
- DROP COLUMN firstname;
- SELECT AVG(purchases);
- FROM suppliers;

- Query languages, structured programming languages, and object-oriented programming languages are categorized as high-level programming languages
- Assembly languages are categorized as low-level programming languages
- A query language is predominantly used to request data from a database or to manipulate data in a database
- The most prevalent query language for database queries and database management is Structured Query Language (SQL)
- Select queries request data from a database, whereas action queries manipulate data in a database

- Assembly languages use a simple set of symbols to represent the 0s and 1s of machine code
- Assembly languages are closely tied to the processor architecture from hardware manufacturers
- Assembly languages are translated using an assembler instead of a compiler or interpreter
- Assembly language instructions have a one-to-one association with their machine code counterpart

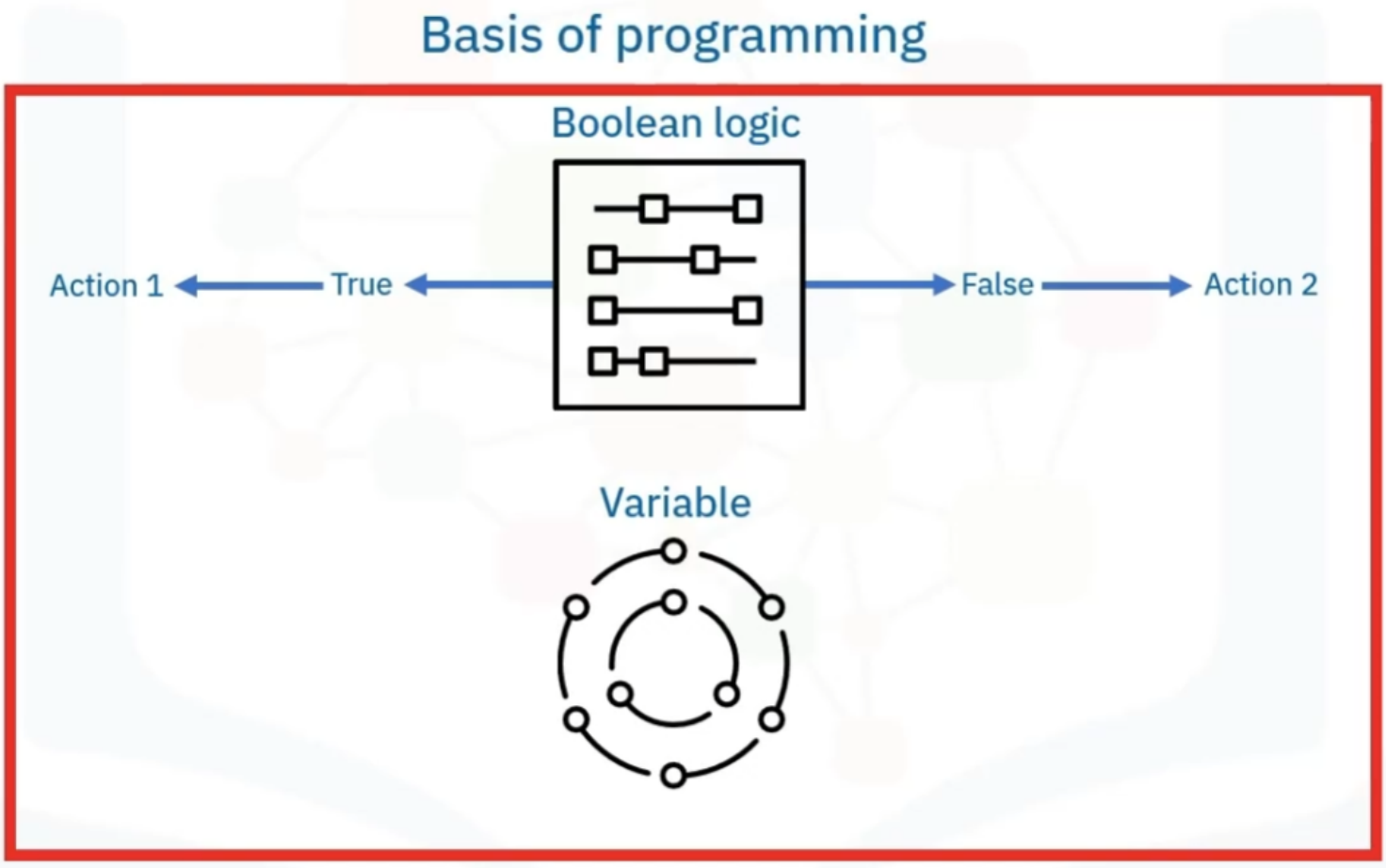
Understanding Code Organization Methods

ITC.200303.18_v2-en.txt

- Organizing and planning software before development is crucial to good code development
- Organized code is easier to read, maintain, and configure
- Flowcharts and pseudocode are two methods of planning and organizing code before development
- Flowcharts display a process, represented using different shapes and arrows
- Pseudocode explains exactly what each line does, making it easier for programmers to understand

Branching and Looping Programming Logic

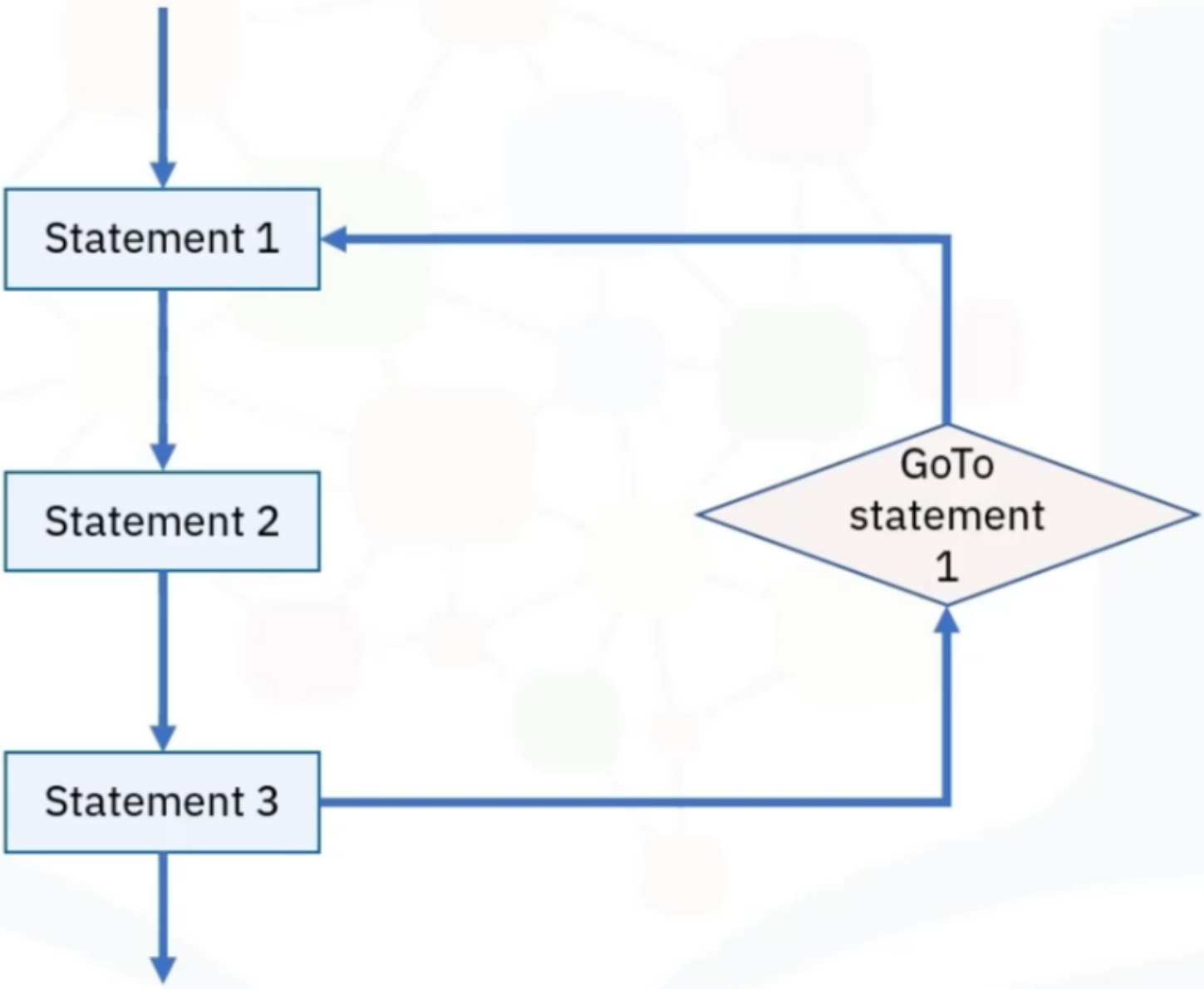
ITC.200303.17_v3-en.txt

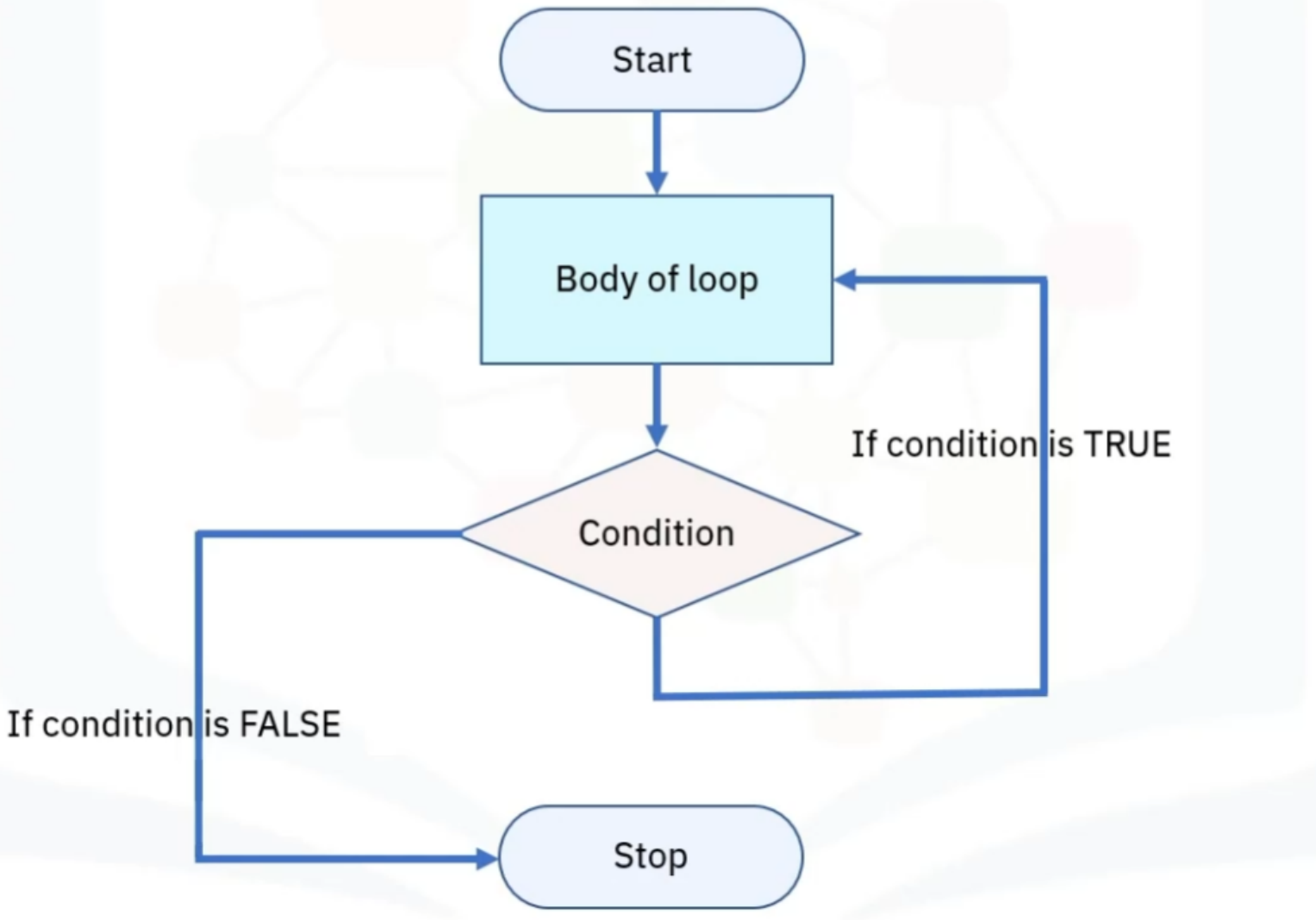


Basis of programming

```
Condition is true:

int number = 10;

if (number > 0) {
    //code
}

// code after if
```

```
Condition is false:

int number = 10;

if (number < 0) {
    //code
}

// code after if
```


Statement 1
Statement 2
Statement 3
GoTo statement 1


Start
Body of loop
If condition is TRUE
Condition
If condition is FALSE
Stop

Looping programming logic

While loop: Condition is evaluated before processing, if true, then loop is executed

```
while (condition) {
                statements;
}
```

For loop: Initial value performed once, condition tests and compares, if false is returned, loop is stopped

```
for (initial value; condition; incrementation or decrementation )
{
  statements;
}
```

Do-while loop: Condition always executed after the body of a loop

```
do {
  statements
} while (expression);
```

Introduction to Programming Concepts Part 1

ITC.200303.13_v1-en.txt

Introduction to Programming Concepts Part 2

ITC.200303.30_v3-en.txt

Module 4 - Software Architecture, Design and Patterns

Introduction to Software Architecture

200424.03_v1-en.txt

Early design decisions

- How components interact
- Operating environment
- Design principles
- Costly to change once implemented
- Addresses non-functional aspects

Architectural diagrams

Displays:

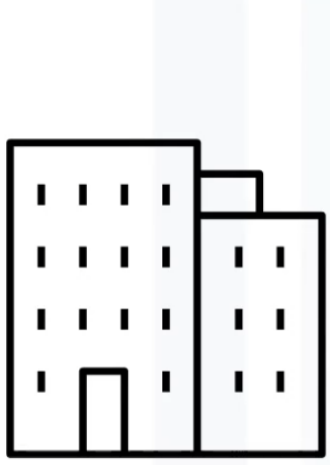
Components | Interactions | Constraints | Confines | Architectural patterns

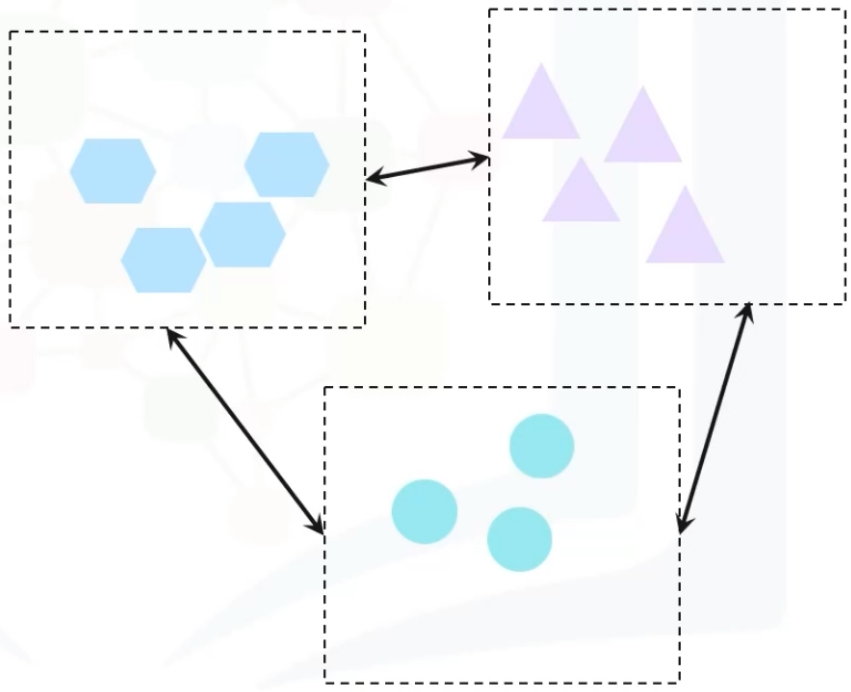
- Software architecture functions as a blueprint and represents the underlying organization of the application
- Software architecture is important because it serves as a basis for communication among team members
- It represents the earliest design decisions, is hard to change once development starts, and accommodates changing requirements during development
- Architectural design influences technology stack choices and the production environment
- Artifacts resulting from the design include the SDD, an architecture diagram, and UML diagrams

Software Design and Modeling

200368.12_v3-en.txt

- Structural elements:modules & sub-module

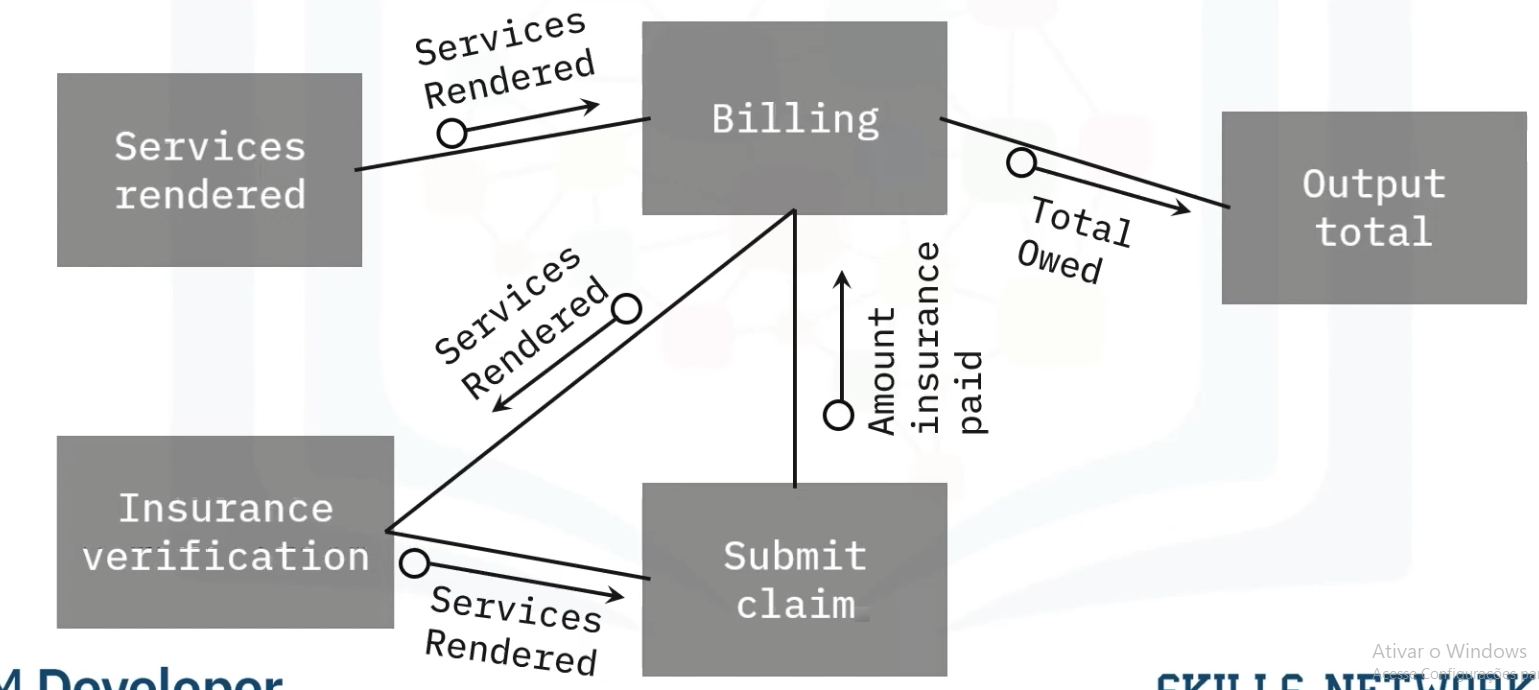




- Cohesive
- Loosely coupled

Structure diagram example

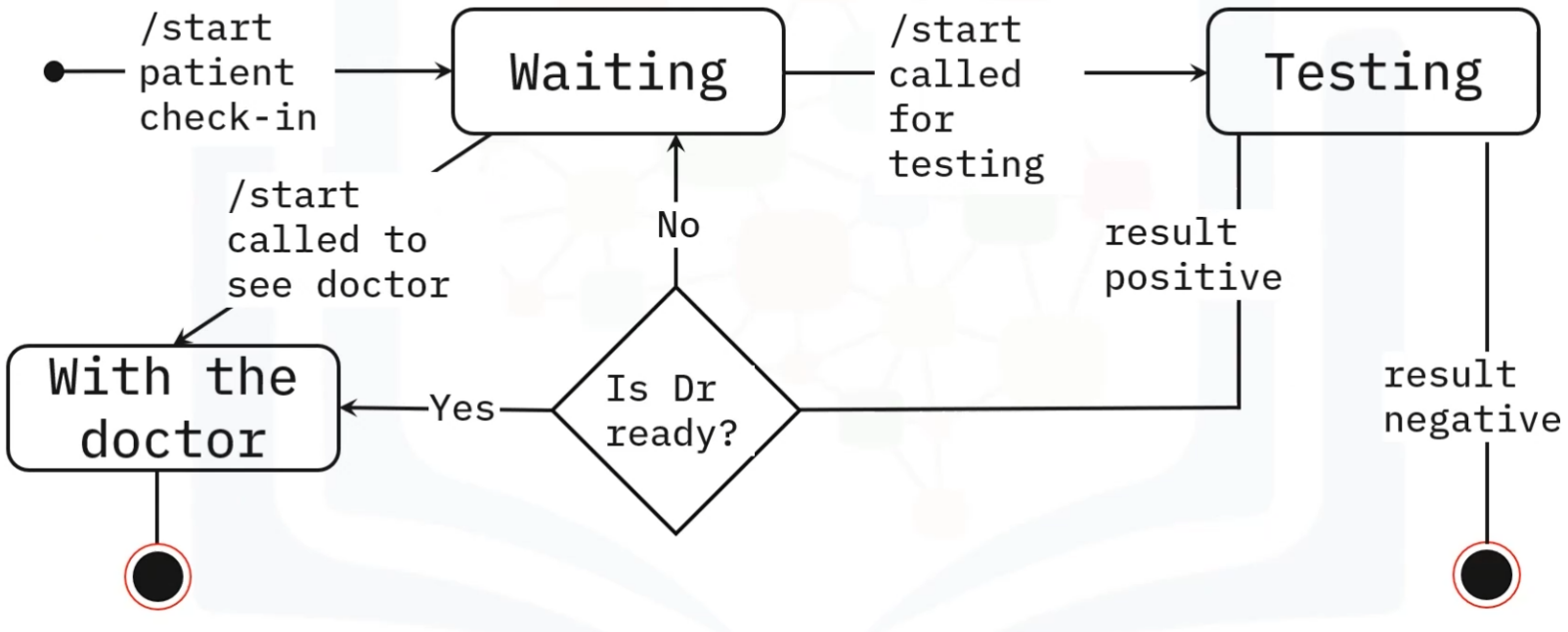
Billing system



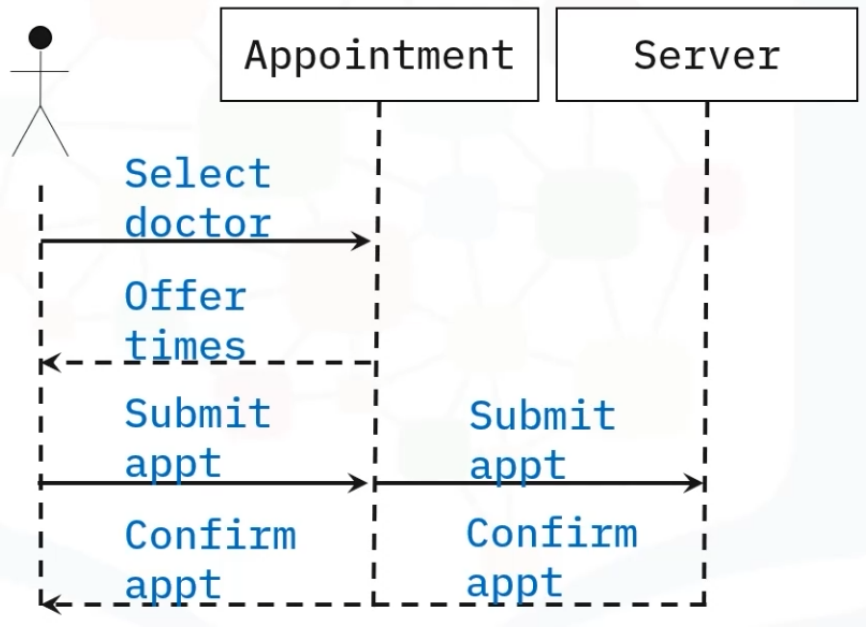
BM Developer

SKILLS NETWORK

State transition diagram example



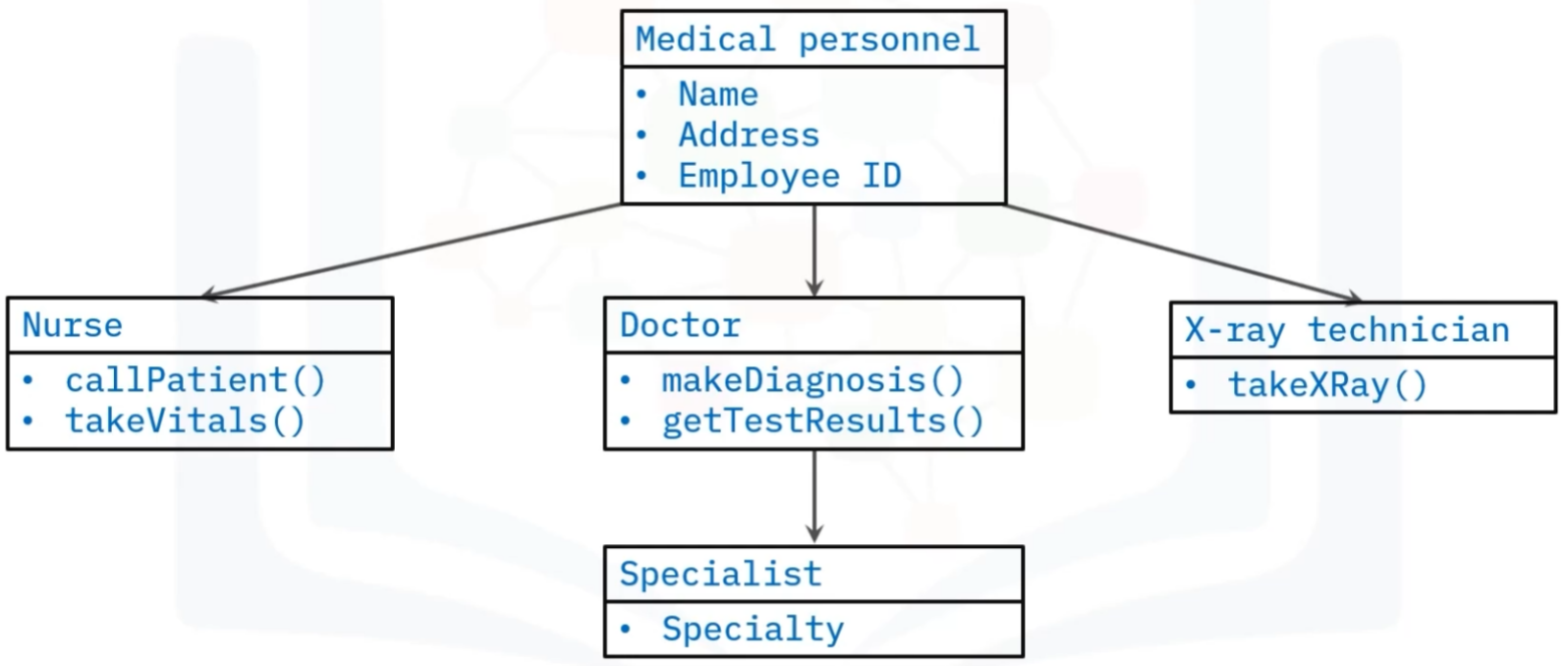
Interaction diagram

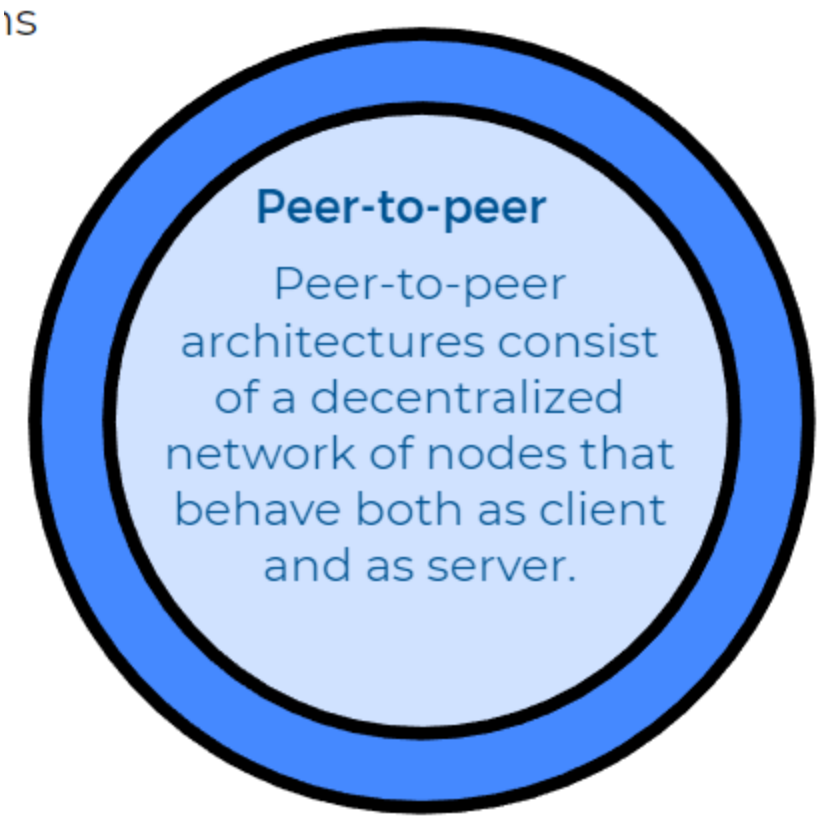


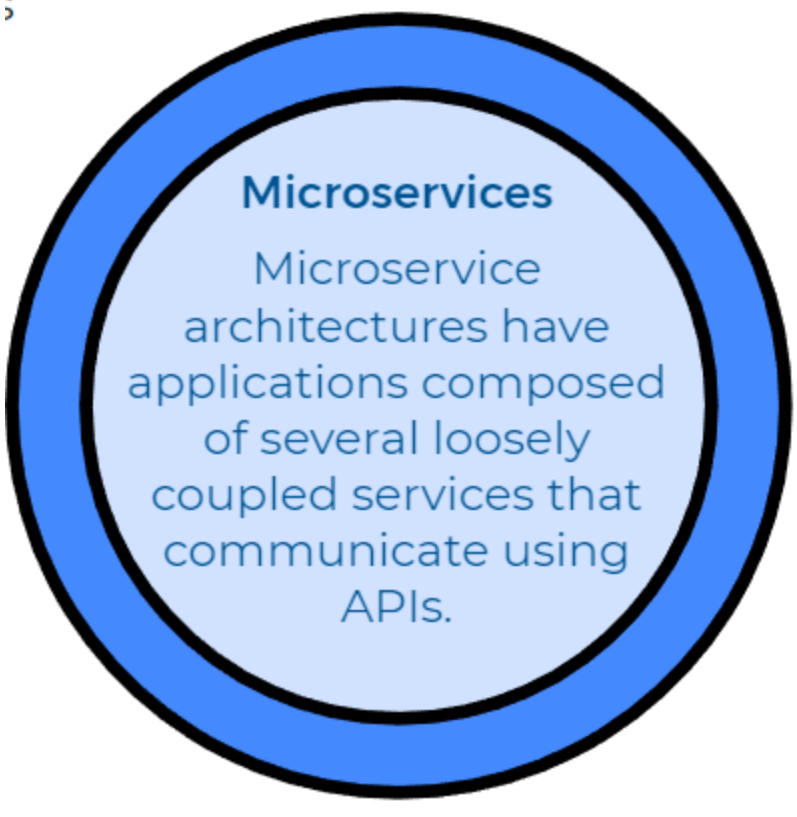
Object-Oriented Analysis and Design

200368.24_v4-en.txt

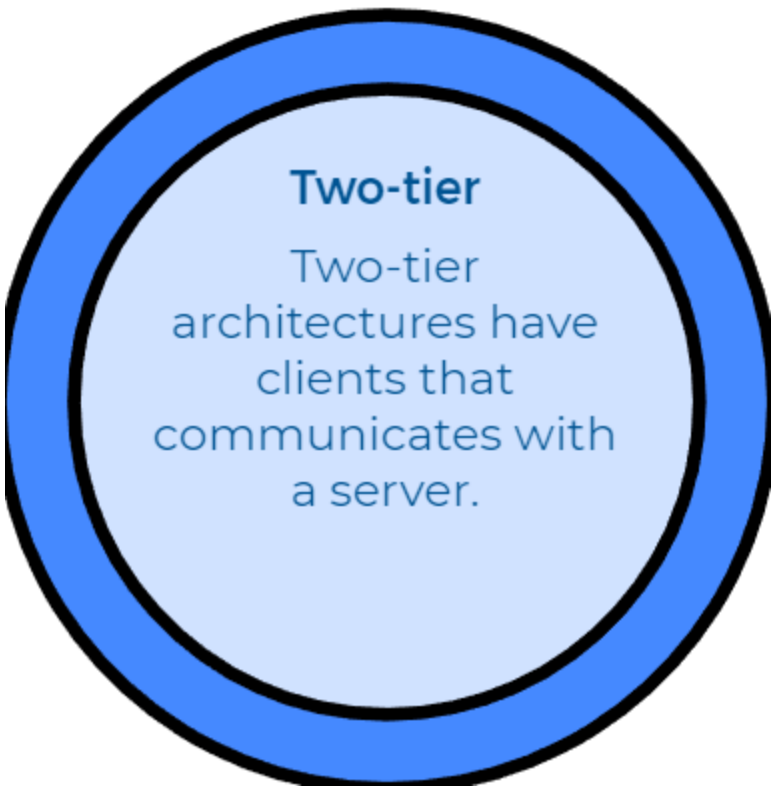
Class diagram

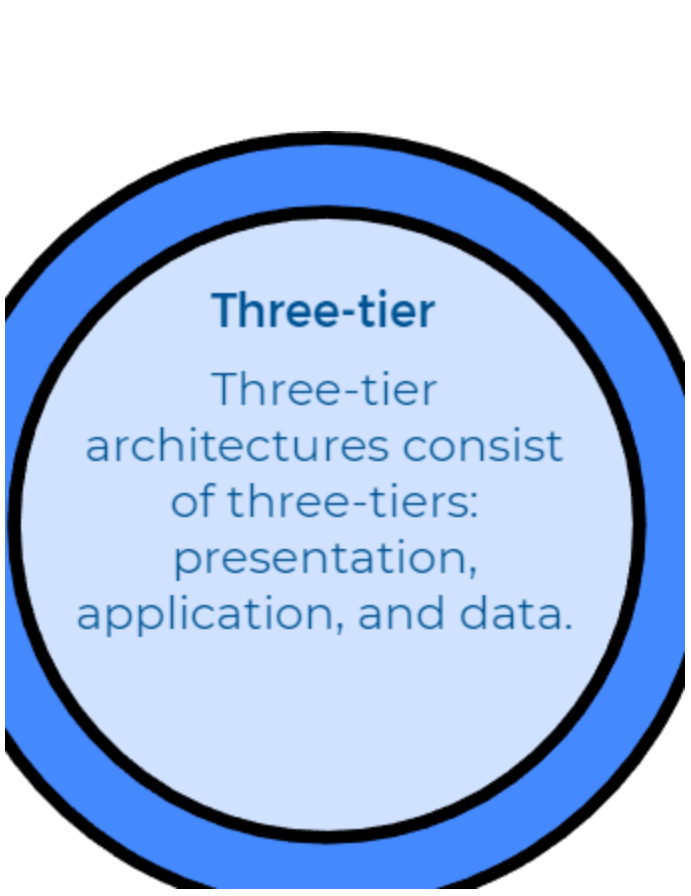


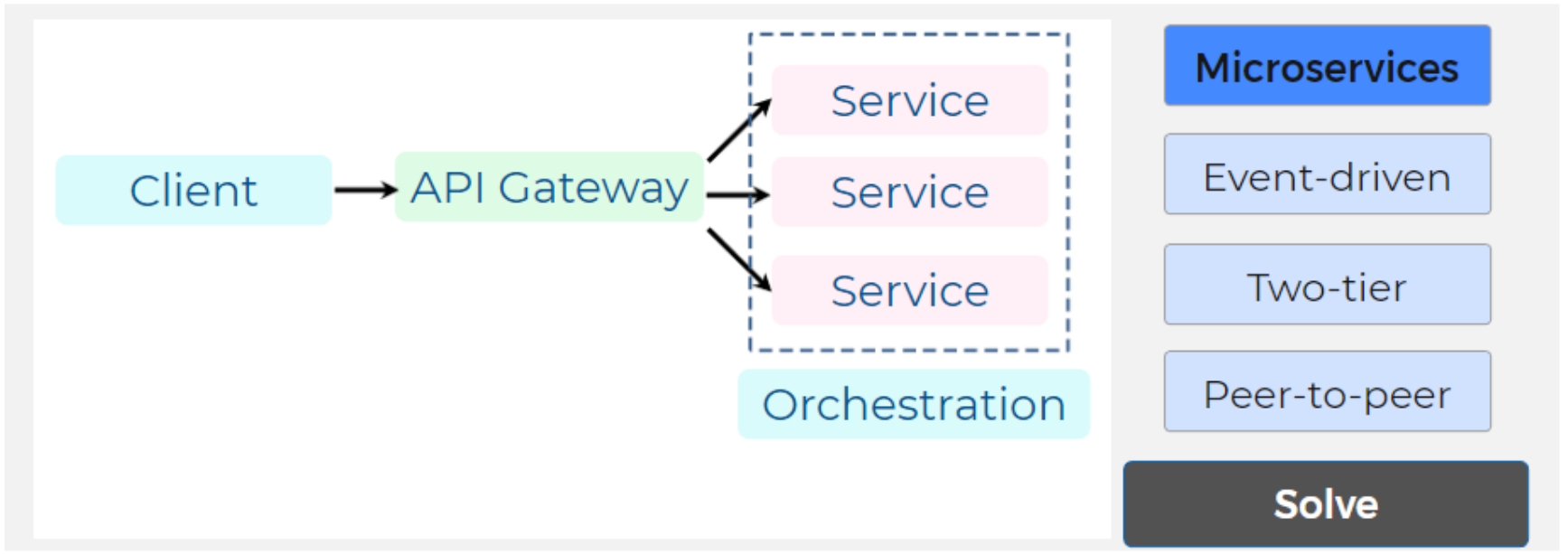

Peer-to-peer
Peer-to-peer architectures consist of a decentralized network of nodes that behave both as client and as server.

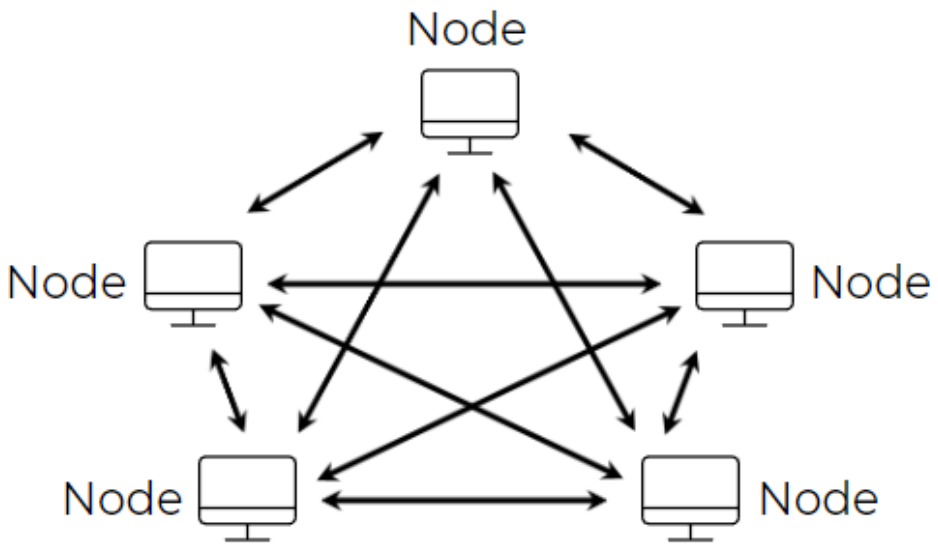

Microservices
Microservice architectures have applications composed of several loosely coupled services that communicate using APIs.


Event-driven
Event-driven architectures have consumers that send requests to producers.


Two-tier
Two-tier architectures have clients that communicates with a server.


Three-tier
Three-tier architectures consist of three-tiers: presentation, application, and data.


Client
API Gateway
Service
Service
Orchestration
Microservices
Event-driven
Two-tier
Peer-to-peer
Solve



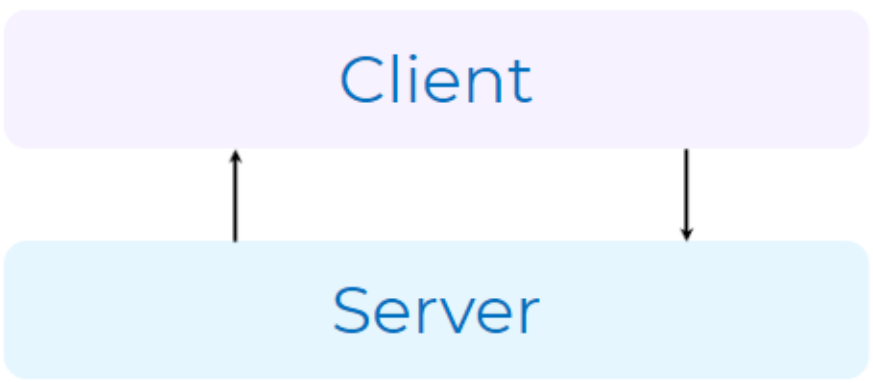
Two-tier

Microservices

Three-tier

Peer-to-peer

Solve



Microservices

Two-tier

Event-driven

Peer-to-peer

Solve

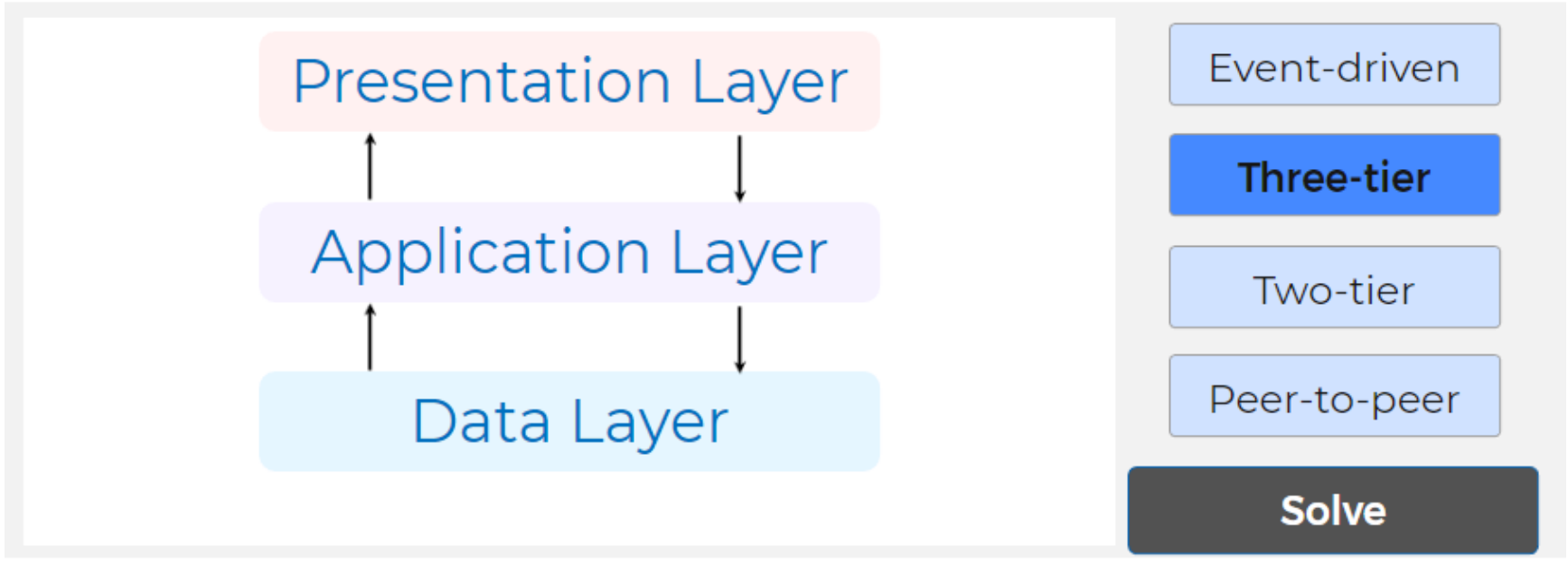
Two-tier

Event-driven

Peer-to-peer

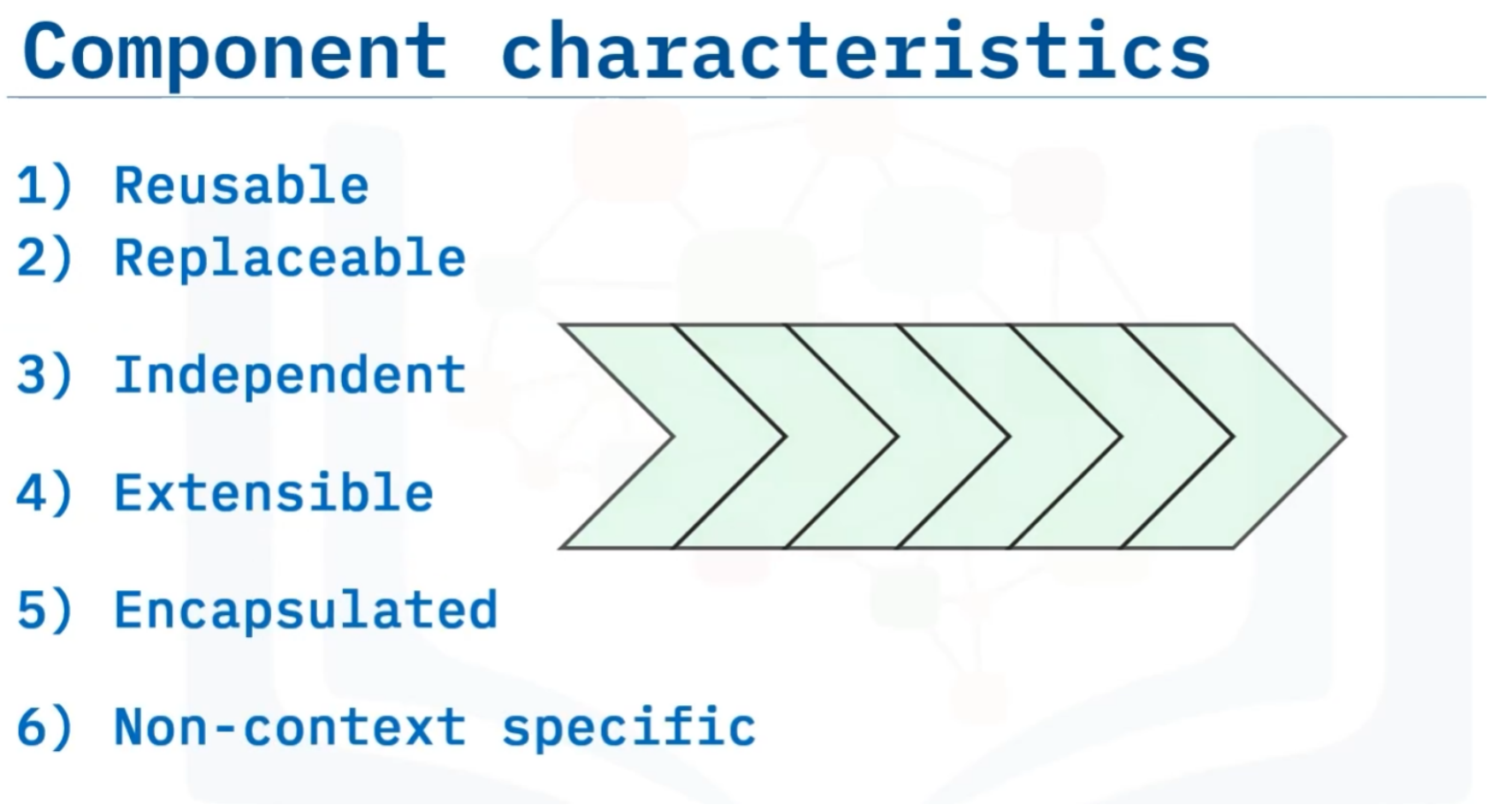
Microservices

Solve

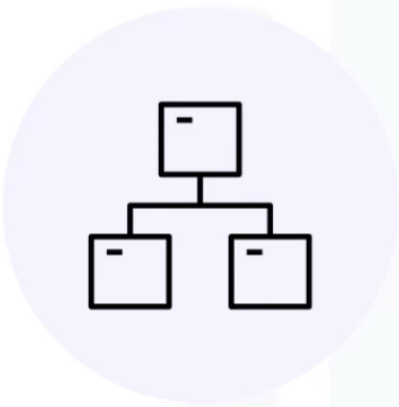


Approaches to Application Architecture

200368.13_v5-en.txt

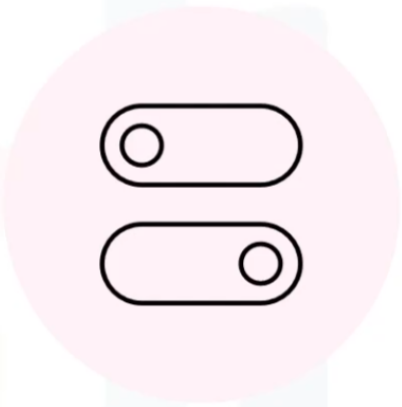


Component examples



API

Data Access Object



Controller

Services

- Designed to be deployed independently and reused by multiple systems
- Solution to a business need
- Has one unique, always running instance with whom multiple clients communicate

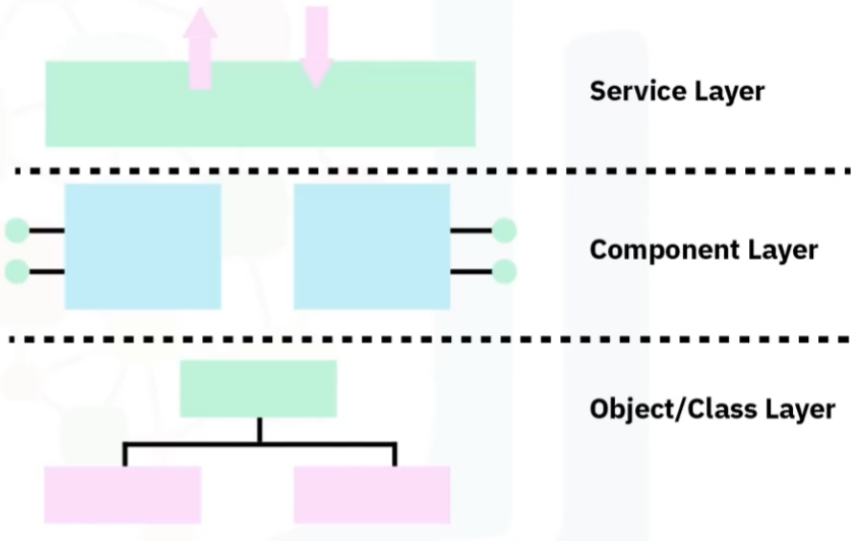


Figure 5 - Application implementation layers

Distributed system characteristics

- Shares resources

- Fault-tolerant

- Multiple activities run concurrently

- Scalable

- Runs on a variety of computers

- Programmed in a variety of languages

- Components are reusable, independent, replaceable, extensible, encapsulated, and non-context specific
- Component-based architecture is the decomposition of the system into logical independent components
- Services are made of components and components consist of objects. Services are deployed independently and can be reused by multiple systems
- SOA is a loosely coupled services that interface with each other via a communication protocol over a network
- Distributed systems:
  - multiple services on different machines
  - appear to the end-user as a single coherent system

Architectural Patterns in Software

200368.14_v5-en.txt

Examples

2-tier: Messaging apps

3-tier: Web apps

Event-driven: Ride sharing

Peer-to-peer: Cryptocurrency

Microservices: Social media

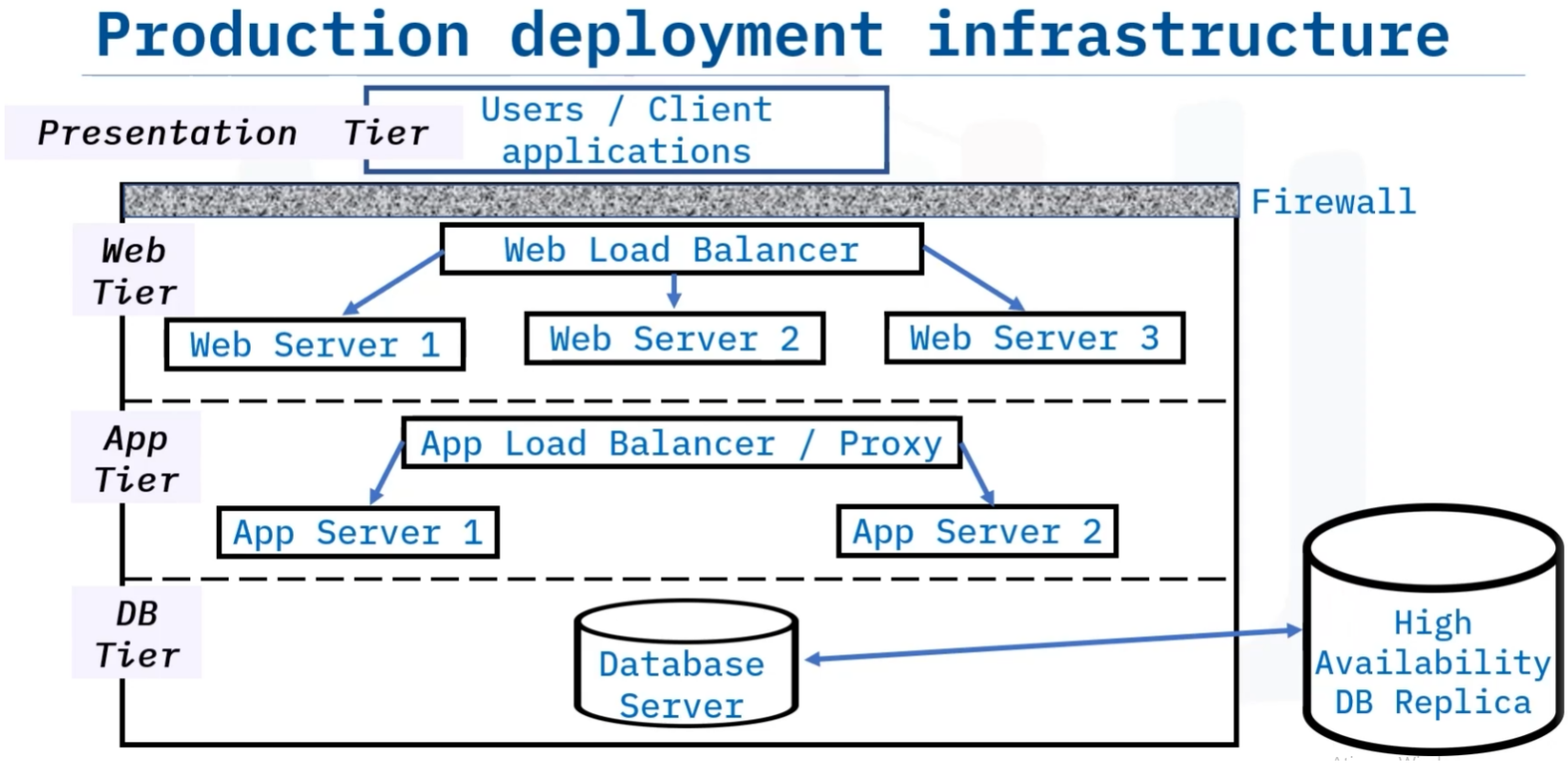
Application Deployment Environments

200368.15_v3-en.txt

- Application environments include development, test/QA, staging, and production
- Production environments must also take into account non-functional requirements like load, security, reliability, and scalability
- Deployment platforms: on-premises, public, private cloud, & hybrid cloud

Production Deployment Components

200368.25_v5 (2)-en.txt



- Common production environment components include a firewall, a load balancer, web and application servers, proxy servers, and database servers
- A firewall is a security device that monitors traffic between networks
- Load balancers distribute network traffic among servers
- A web server delivers web content
- An app server provides business logic
- A database server stores and controls the flow of data through a DBMS

Module 5 - Job Opportunities and Skillsets in Software Engineering

What Does a Software Engineer Do?

200368.16_V3-en.txt

Software engineering tasks

- Designing new software systems
- Writing and testing code
- Evaluating and testing software
- Optimizing software programs
- Maintaining and updating software systems
- Documenting code
- Presenting new systems to users and customers
- Integrating and deploying software

- Software engineers design and develop a range of software solutions:
  - Back-end engineers build the computer systems and networks
  - Front-end engineers build the user interfaces
- Software engineers undertake a range of tasks, from designing and writing new software to maintaining and updating existing software
- As your career develops, you will move from being responsible for one small section of code to many areas of one or more products

Skills Required for Software Engineering

200424.05_v2-en.txt

- A combination of hard and soft skills are essential to a software engineer
- Hard skills are measurable learned skills:
  - Programming
  - Testing
  - Troubleshooting
- Soft skills are personality and characteristics:
  - Communication
  - Problem solving

Job Outlook for Software Engineers

200424.07_v4 - Job Outlook for Software Engineers-en.txt

- The high demand for software engineers is due to:
  - Increasing needs for software
  - Increasing complexity of applications
  - Continuing growth of technology
- The outlook for software engineers is promising
- The role of a software engineer can be flexible and satisfying
- Employment options for software engineers are flexible and varied, from full-time employment through to voluntary activities

Career Paths in Software Engineering

200368.20_v3-en.txt

- A career in software engineering follows a technical or managerial path
- At each stage on that path, you'll take on more responsibility and a wider range of tasks
- Starting out as a software engineer enables you to take an array of other options at any stage in your career

Software Engineering Job Titles

200424.08_v4 - Software Engineering Job Titles-en.txt

Job titles

- Front-end engineer
- Back-end engineer
- Full-stack engineer
- DevOps engineer
- Software quality assurance engineer
- Software integration engineer
- Software security engineer
- Mobile app developer
- Games developer

Software Engineering Code of Ethics

200424.04_v2-en.txt

The 8 principles

1. Public
2. Client/Employer
3. Product
4. Judgement
5. Management
6. Profession
7. Colleagues
8. Self