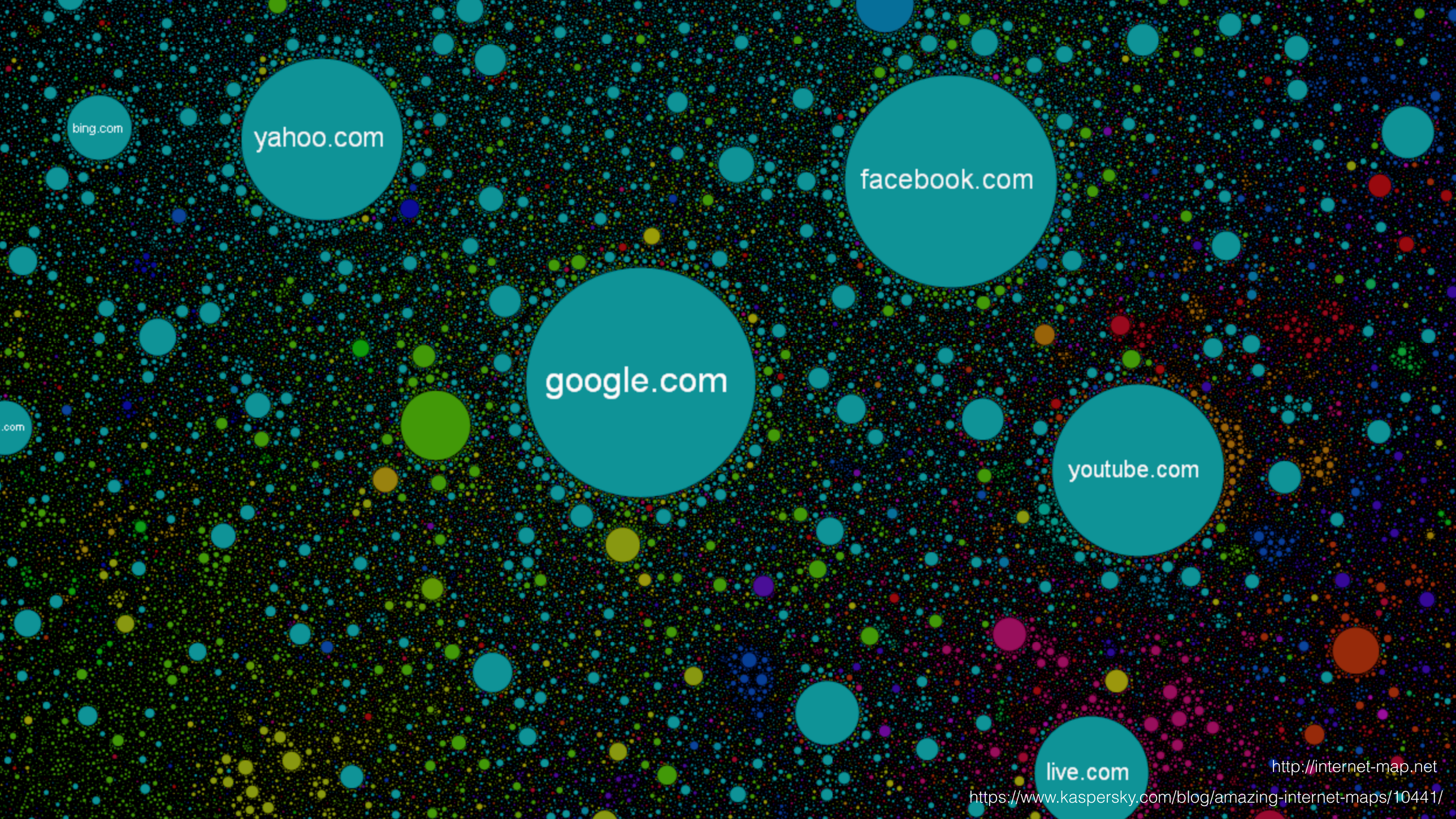
Internet Applications Design and Implementation (Lecture 1 - Introduction and Logistics)

MIEI - Integrated Master in Computer Science and Informatics
Specialization block

João Costa Seco (joao.seco@fct.unl.pt)

(with previous participations of Jácome Cunha (jacome@fct.unl.pt) and João Leitão (jc.leitao@fct.unl.pt))

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY


bing.com
yahoo.com
facebook.com
google.com
youtube.com
live.com
http://internet-map.net
https://www.kaspersky.com/blog/amazing-internet-maps/10441/

Internet Applications Design and Implementation

Internet application | ˈɪntənɛt aplɪˈkeɪʃ(ə)n |, noun

1. Any application that uses the internet to consume and/or provide services and data.

Internet Applications Design and Implementation

Web application | wɛb aplɪ'keɪʃ(ə)n |, noun

1. An internet application that runs on a browser and obtains HTML and data pages from a web server.

Mobile application | 'məʊbʌɪl aplɪ'keɪʃ(ə)n |, noun

2. An application installed and running in a mobile device, usually dependent on web services as data sources.

Web service | wɛb 'səːvɪs |, noun

3. A computational procedure available on the Internet to provide services and data to other apps and services. Usually associated to binding technologies like SOAP or REST.

Internet Applications Design and Implementation

Cloud application | klaʊd aplɪˈkeɪʃ(ə)n |, noun

1. An internet application deployed on an independent hosting service, providing computation and additional resources and features like replication and storage.

Service-based architecture | ˈsəːvɪs beɪs ˈɑːkɪtɛktʃə |, noun

2. A conceptual structure and logical organisation of web services, usually implemented using orchestration languages and tools.

Data-Centric Applications | Dataˈdeɪtə-ˈsɛntrɪk ˌæplɪˈkeɪʃ(ə)nz |, noun

3. Applications that are developed primarily based on resource-based rules, making their data available to others through well defined interfaces

Internet Applications Design and Implementation

An Internet Application is a **global concept** instantiated into many different **local components** running in many **different devices**/ systems and **communication over heterogeneous channels**

Internet Applications Design and Implementation

The challenge is to **design**, **specify**, and **implement**

modular,
loosely-coupled,
large-scale applications,

so that the software development process is more efficient and the longevity of the applications is longer.

Internet Applications Design and Implementation

Faster development cycles,

functionally correct applications,

applications that can easily be used by other systems,

heterogeneous development and execution environments,

applications are easily maintainable (corrections and extensions)

secure systems

reliable and available systems

performant applications

Internet Applications Design and Implementation

Many of these goals are specific of Internet Applications and should be attained using specific methods, tools and techniques.

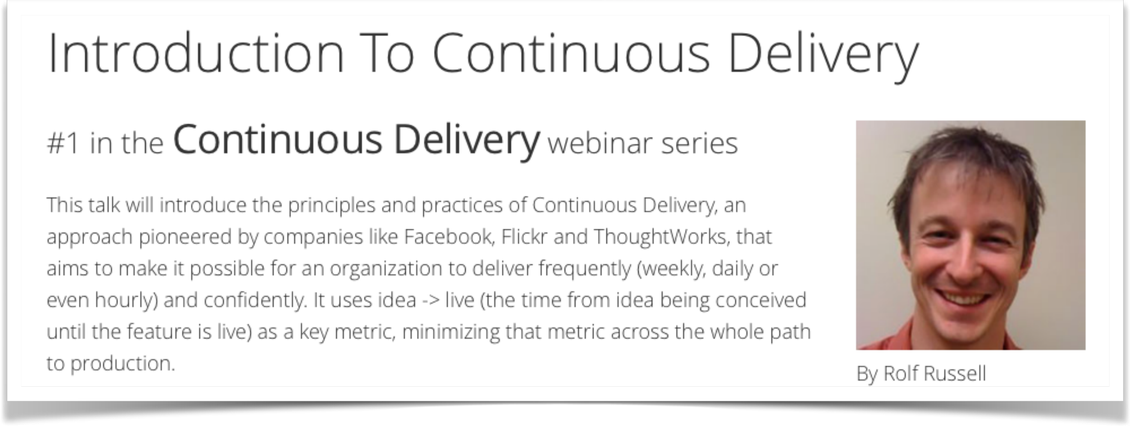
What about Internet Applications?

Features of Internet Applications

- Everything is (inter)connected and developing software for interconnection requires special skills and methods
- Internet Apps can be:
  - Standalone / Desktop with native interfaces (RPC) or http connections
  - Web, via http, accessible through browsers
  - Mobile native or PWAs (connected via http of native, w/ offline capabilities)
  - Compound and orchestrated services
  - Mash-up interfaces
    (e.g. google maps + rentals, friends, …http://mashable.com/2009/10/08/top-mashups/)

Skills for Internet Applications

- Special skills because
  - software evolves fast, we need to develop extensible and maintainable software.
  - data and code have different wills, we need to develop software in sync with data.
  - software is “open” and connected (the world changes), we need to develop software that is loosely coupled, robust and follows standard API conventions.
  - Internet Apps may grow to have a huge user base (milions of requests)…
  - It also needs to be scalable, secure, replicated, reliable, concurrent, safe…

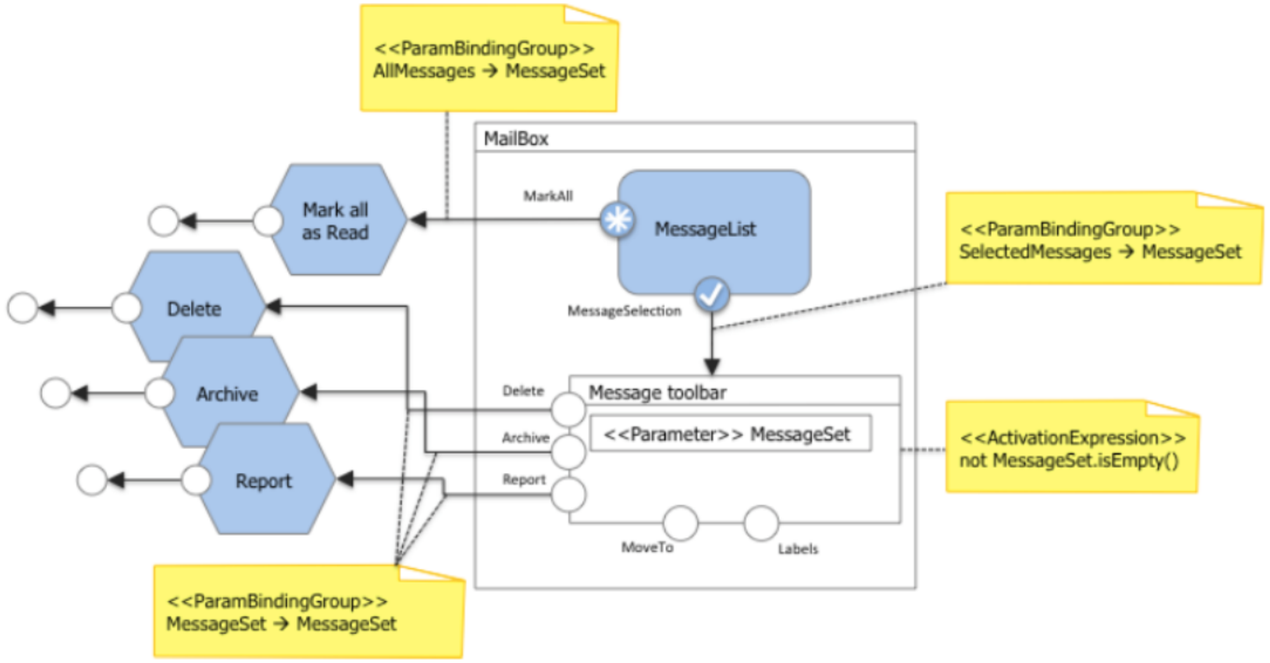


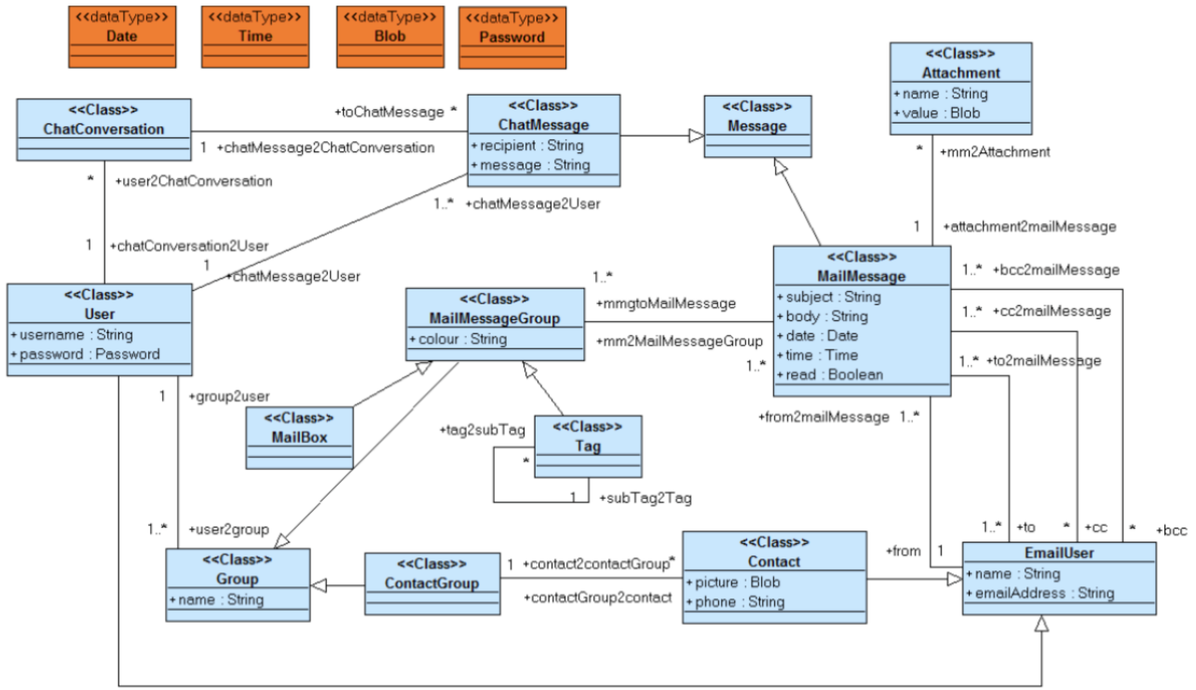
“Tools” for Internet Applications

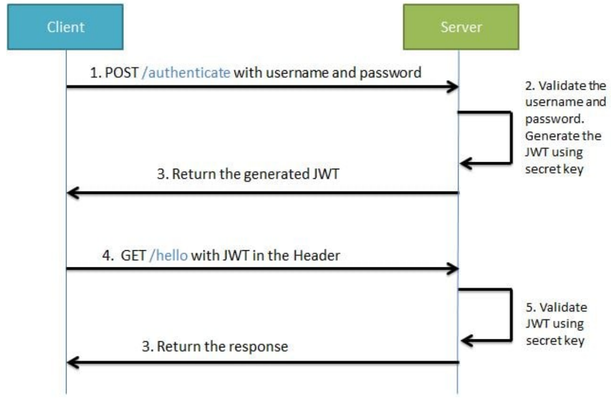
- Software Specification (UML, OpenAPI, IFML)
- Software Architecture (Web MVC, SOA, Micro Services)
- Software Patterns (Observer, Chain of Respons., Facade, Builder, …)
- Programming Languages (Statically Typed, Dynamic, Concurrent, …)
- Software Frameworks (Web, Component, Reactive, Data-Abstraction,…)
- Development Methods (Agile, TDD, BDD, Continuous Integration, C. Delivery)
- Deployment Tools (Cloud Managers, Containers,)

Software Specification for Internet Applications

- Data schema specification
- Behaviour specification
- Interface-flow specification
- Security specification

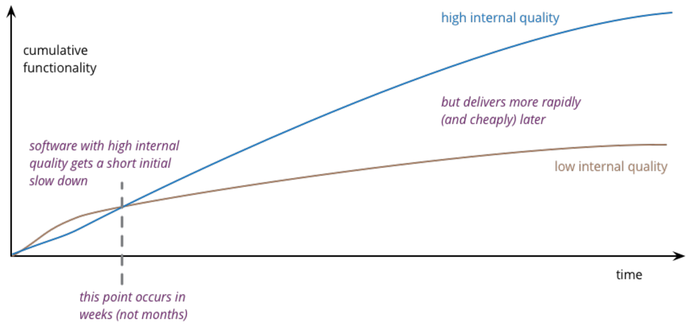




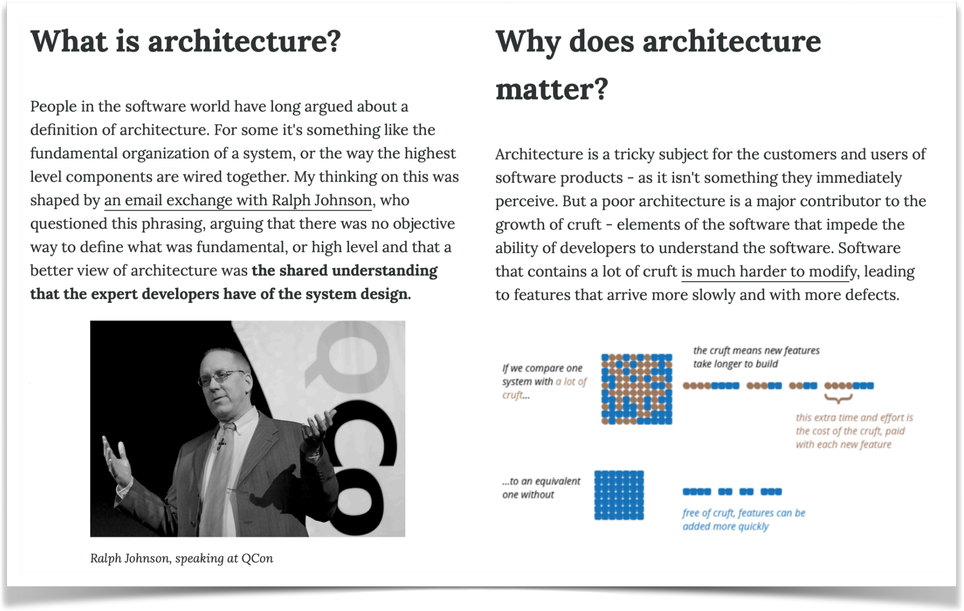


Software Architecture for Internet Applications

- Architecture refers to the internal design of a software system
- It describes the way the highest level components are wired together.
- A good architecture allows a system to be expanded with new capabilities
- A good architecture pays off in quality



in martinfowler.com/architecture/



What is architecture?

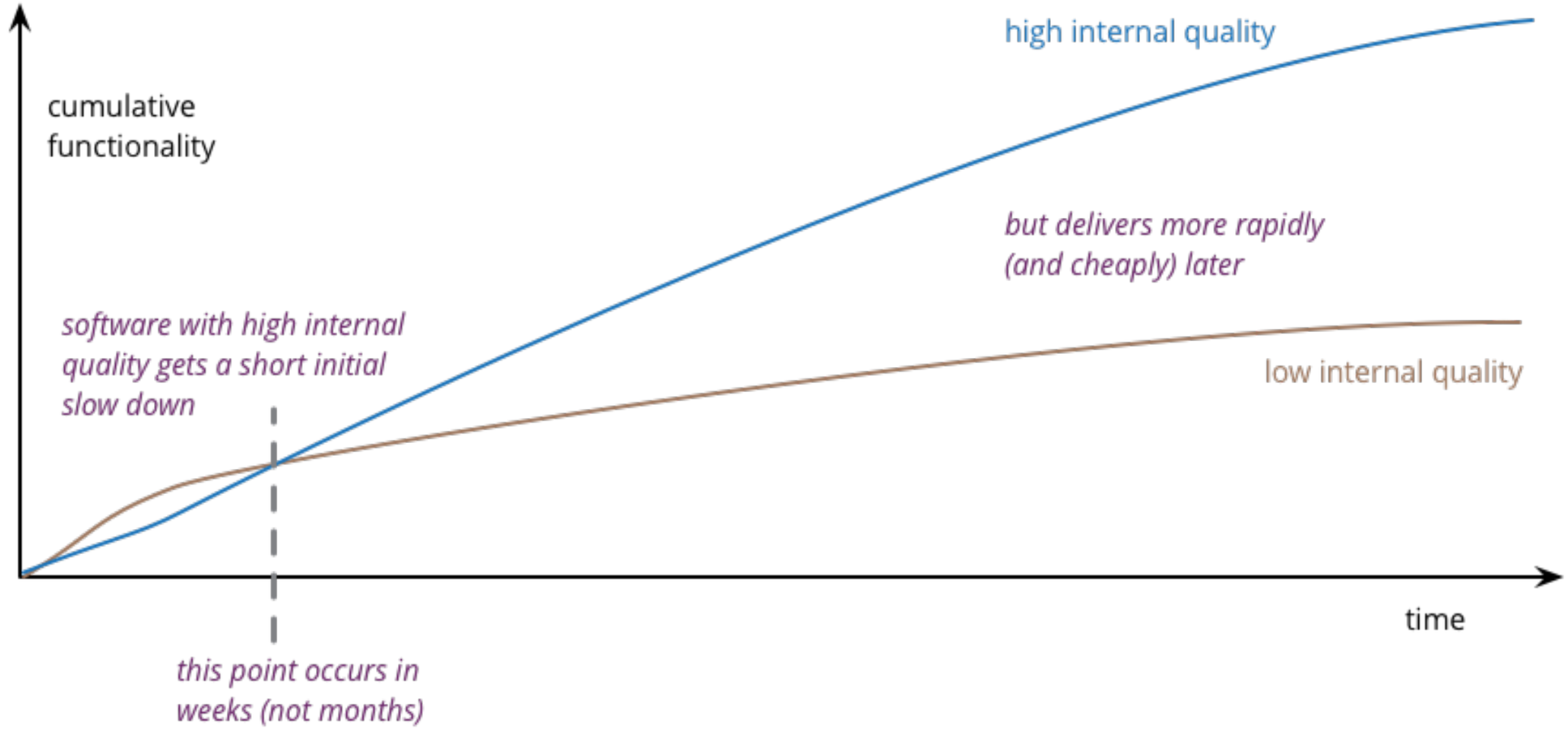
People in the software world have long argued about a definition of architecture. For some it's something like the fundamental organization of a system, or the way the highest level components are wired together. My thinking on this was shaped by an email exchange with Ralph Johnson, who questioned this phrasing, arguing that there was no objective way to define what was fundamental, or high level and that a better view of architecture was **the shared understanding that the expert developers have of the system design.**

Ralph Johnson, speaking at QCon

Why does architecture matter?

Architecture is a tricky subject for the customers and users of software products - as it isn't something they immediately perceive. But a poor architecture is a major contributor to the growth of cruft - elements of the software that impede the ability of developers to understand the software. Software that contains a lot of cruft is much harder to modify, leading to features that arrive more slowly and with more defects.

Software Architecture for Internet Applications



in martinfowler.com/architecture/

Software Architecture for Internet Applications

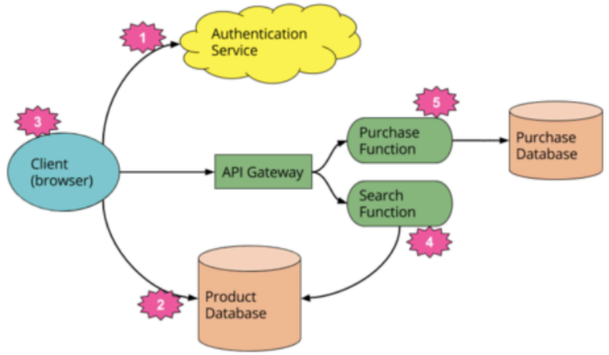
Application Boundary

One of the undecided problems of software development is deciding what the boundaries of a piece of software is. (Is a browser part of an operating system or not?) Many proponents of Service Oriented Architecture believe that applications are going away - thus future enterprise software development will be about

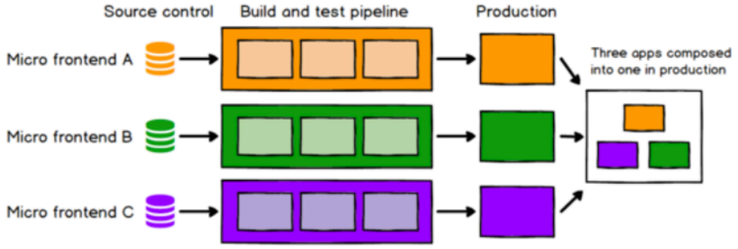
Microservices Guide

The microservice architectural pattern is an approach to developing a single application as a

Serverless Architectures



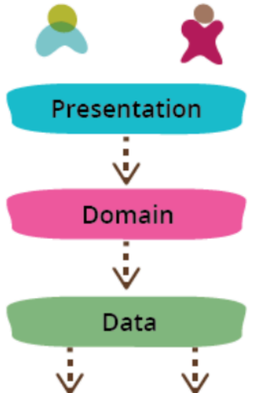
Micro Frontends



GUI Architectures

In the mid 2000s I was pursuing a couple writing projects that could have turned into books, but haven't yet made it. One was on the architecture of user interfaces. As part of this work, I drafted a description of how GUI architectures evolved, comparing the default approach of Forms and Controls with the the Model-View-Controller (MVC) pattern. MVC is one of the most ill-understood patterns in the software world

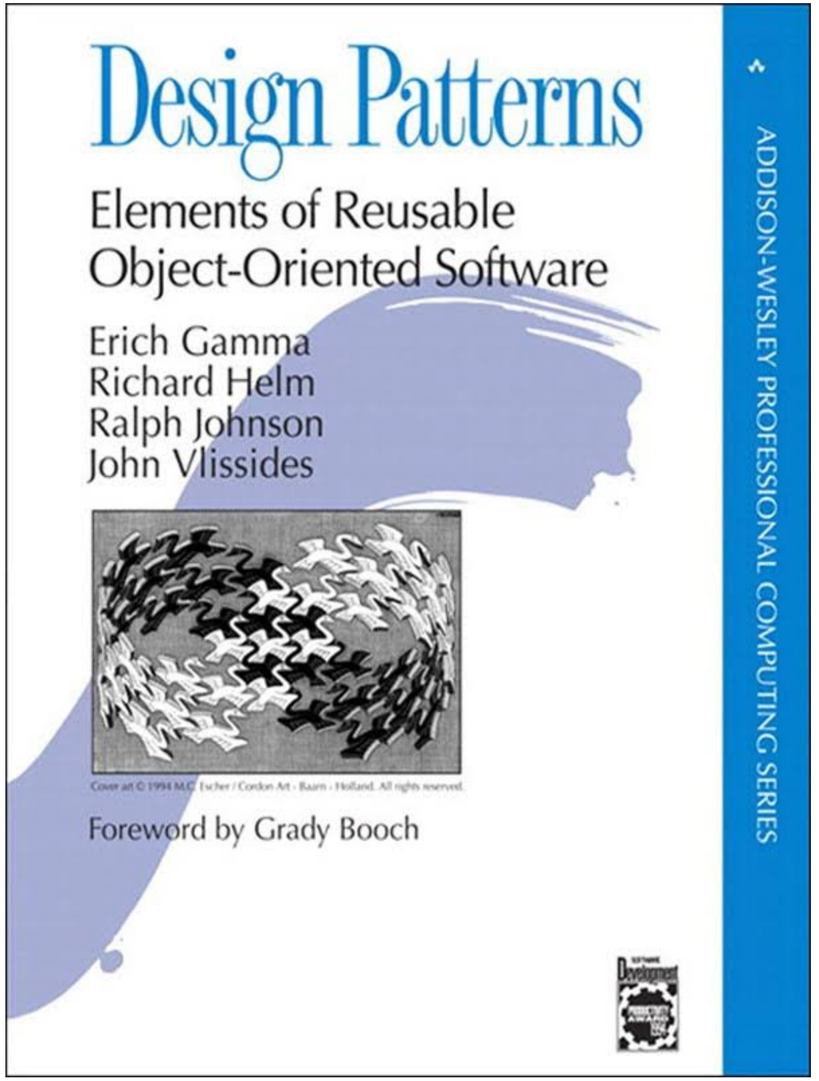
Presentation Domain Data Layering



in martinfowler.com

(Design) Patterns for Internet Applications

- Observer (Web Controllers, Reactive web)
- Chain of Responsibility (Security filters on requests)
- Facade (Service objects, REST interfaces, ORMs)
- Builders (inner to outer representations of data)
- …
- Frameworks implement design and architectural patterns and styles
  - Layered Architecture
  - Model View Controller
  - Inversion of control
  - REST interfaces



Software Frameworks for Internet Applications

- provide **re-usable code** (as libraries)
- provides **controlled and managed resources**
  - e.g. transactions, log, security
- introduce **abstraction layers**
  - to hide complexity of concrete execution scenarios (hardware/software configuration)
  - Sometimes by scaffolding code (does not evolve well)
- support **test driven development** and **code evolution**
- They work by explicit **configuration** or by **conventions**
- **Improve the overall quality! (correction, maintainability, extensibility)**

Spring Initializr
Bootstrap your application

Github Twitter Help

Project: **Maven Project** | Gradle Project

Language: Java | **Kotlin** | Groovy

Spring Boot: 2.2.0 M5 | 2.2.0 (SNAPSHOT) | 2.1.9 (SNAPSHOT) | **2.1.8**

Dependencies

Start projects with

Developer Tools

Spring Boot DevTools
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Lombok
Java annotation library which helps to reduce boilerplate code.

Spring Configuration Processor
Generate metadata for developers to offer contextual help and “code completion” when working with custom configuration keys (ex.application.properties/.yml files).

Web

Spring Web
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default

Spring Reactive Web
Build reactive web applications with Spring WebFlux and Netty.

Rest Repositories
Exposing Spring Data repositories over REST via Spring Data REST.

Software Frameworks for Internet Applications

- provide **re-usable code** (as libraries)
- provides **controlled and managed resources**
  - e.g. transactions, log, security

```
UserTransaction utx = entityManager.getTransaction();

try {
    utx.begin();
    businessLogic();
    utx.commit();
} catch(Exception ex) {
    utx.rollback();
    throw ex;
}
```

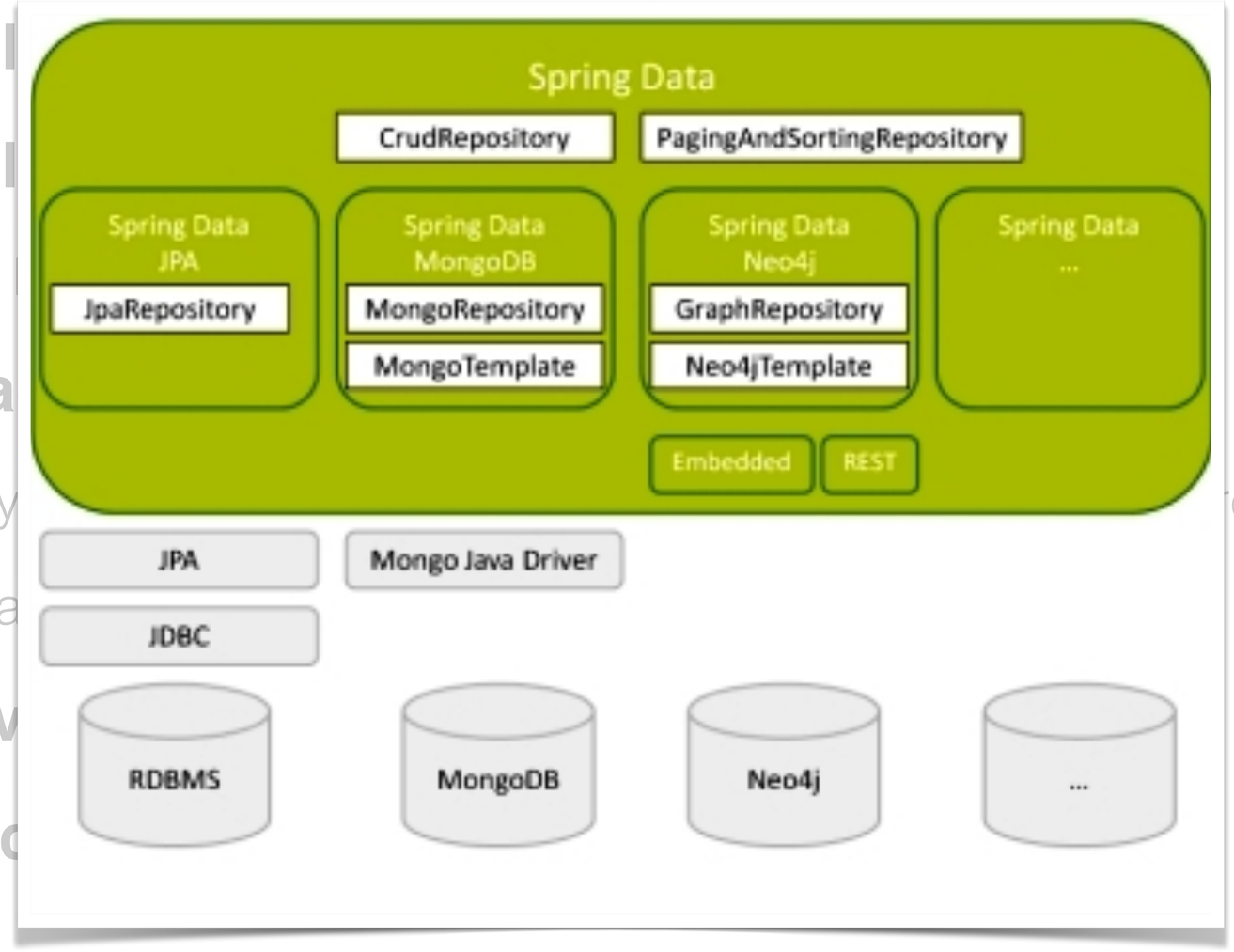
```
@Transactional
public void businessLogic() {
... use entity manager inside a transaction ...
}
```

- it's repetitive and error prone
- any error can have a very high impact
- errors are hard to debug and reproduce
- this decreases the readability of the code base
- What if this method calls another transactional method?

https://dzone.com/articles/how-does-spring-transactional

Software Frameworks for Internet Applications

- provide **re-usabl**
- provides **control**
  - e.g. transactions,
- introduce **abstra**
  - to hide complexity
  - Sometimes by sca
- support **test driv**
- work by explicit **c**



e configuration)

https://www.infoq.com/articles/spring-data-intro/

A spectrum of frameworks for a spectrum of domains

- Web Frameworks
- Data manipulation Frameworks
- Resource Oriented Frameworks (REST)
- Process Oriented Frameworks
- Client-side frameworks

Software Frameworks for Internet Applications

- Examples of Web Frameworks:
  - Ruby on Rails, Django and Python, Spring and Java, Cake and PHP, nodeJS, Meteor, Revel and Go
- Examples of Client Frameworks:
  - AngularJS, React, Vue, Lightweight form
- Examples of Data Frameworks:
  - LINQ, Hibernate (JPA), …
- Examples of Web Service Frameworks/Languages:
  - WS-BPEL…
- Examples of Multi-tier Languages:
  - Ocaml (w/ Ocsigen), Elm, LINKS, UrWeb, Loom

Software Frameworks for Web Applications

- Python (+Django)
  - Easy-to-learn programming language
  - Easy-to-use data structures
  - Available libraries

Software Frameworks for Web Applications

- Based on Ruby (dynamically typed language)
  - Implements the MVC architectural pattern
  - Pattern components assembled by **conventions** on folders, filenames, and language identifiers
- Very flexible programming language
  - Everything is “re-programmable”
- Rails is a versatile tool
  - Support for scaffolding, migrating code and data, and TDD
- Convention over configuration (even more)
- Big community
  - Big pool of top-of-the-line gems

Software Frameworks for Web Applications

- Java + J2EE / Spring / Play
- Most used programming language
  - Statically typed and dynamically assembled
  - Well know
  - Easy to start web development
- Huge amount of ready-to-use libraries (Beans)
- Industrial grade efficient implementations
  - Beans framework (MVC is just a module - webmvc)
  - Patterns are assembled **programatically**, or by XML **configuration** files

Software Frameworks for Web Applications

- Elixir + Phoenix is a web development framework written in Elixir which implements the server-side Model View Controller (MVC) pattern.
- Provides high developer productivity and high application performance
- Scaffolding tools like Rails and Django
- Provides LiveView
  - based on web sockets
  - efficiently handle cross border events
  - efficiently refresh web pages based on server side templates

Software Frameworks for Web Applications

- Spring + Kotlin
- Advantages:
  - It's Completely Interoperable With Java
  - It's (way) More Concise Than Java
  - Safer Code
  - It Comes With a Smarter and Safer Compiler
  - It's Easier to Maintain
  - It's Been Created to Boost Your Productivity
  - It “Spoils” You with Better Support for Functional Programming
  - It Has Null in Its Type System

Ktor

VERT.X

https://dzone.com/articles/what-are-the-biggest-advantages-of-kotlin-over-jav

https://spring.io/blog/2017/01/04/introducing-kotlin-support-in-spring-framework-5-0

Jobs for Internet Applications

A **software architect** is a **software** developer expert who makes high-level design choices and dictates technical standards, including **software** coding standards, tools, and platforms.

Software architect - Wikipedia

https://en.wikipedia.org › wiki › Software_architect

About Featured Snippets Feedback

The **2019 Roadmap** To Fullstack **Web Development**

Gift This Course Wishlist

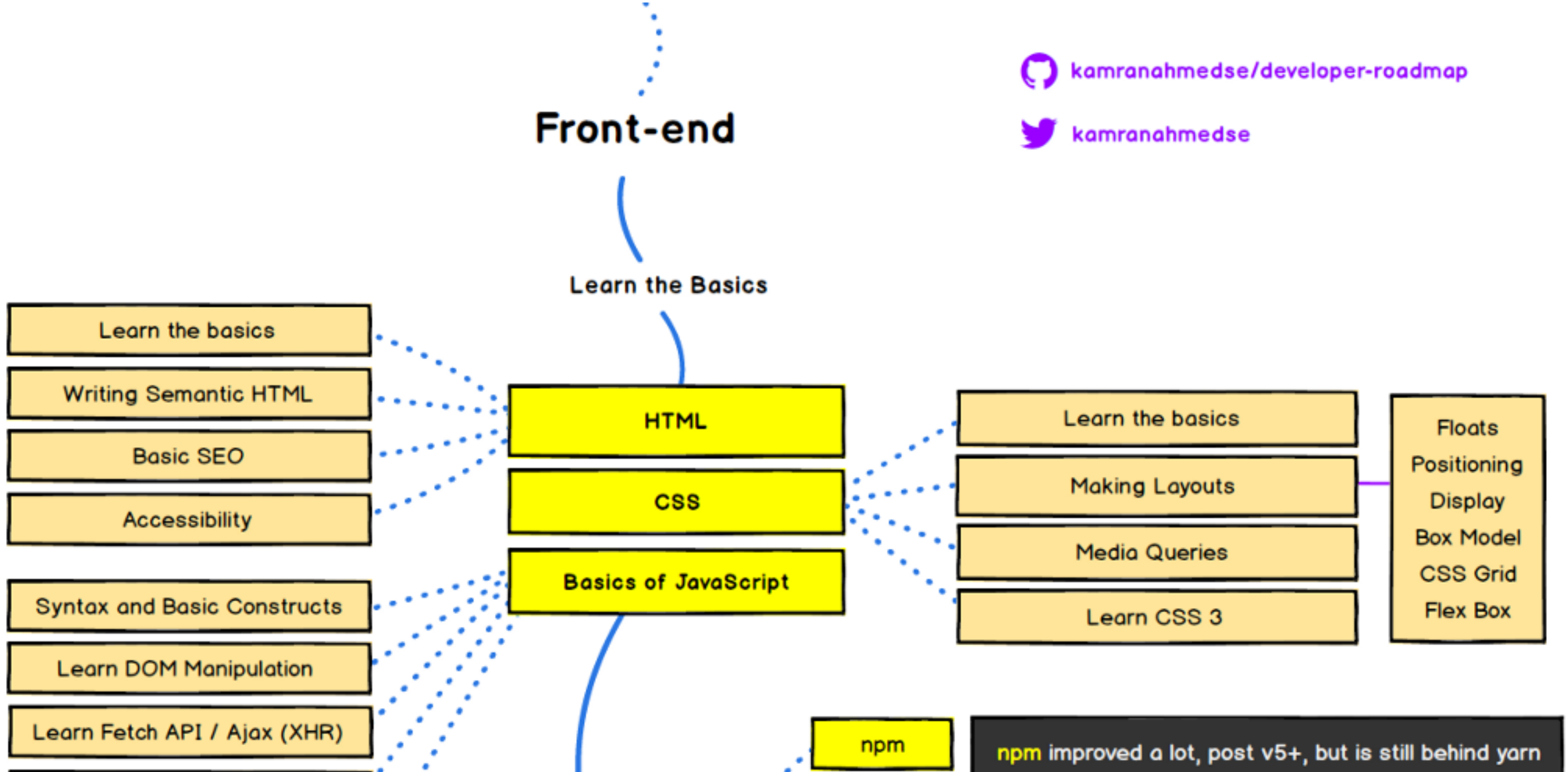
Go Full Stack with Spring Boot and React

Build Your First Full Stack Application with React and Spring Boot. Become a Full Stack Web Developer Now!

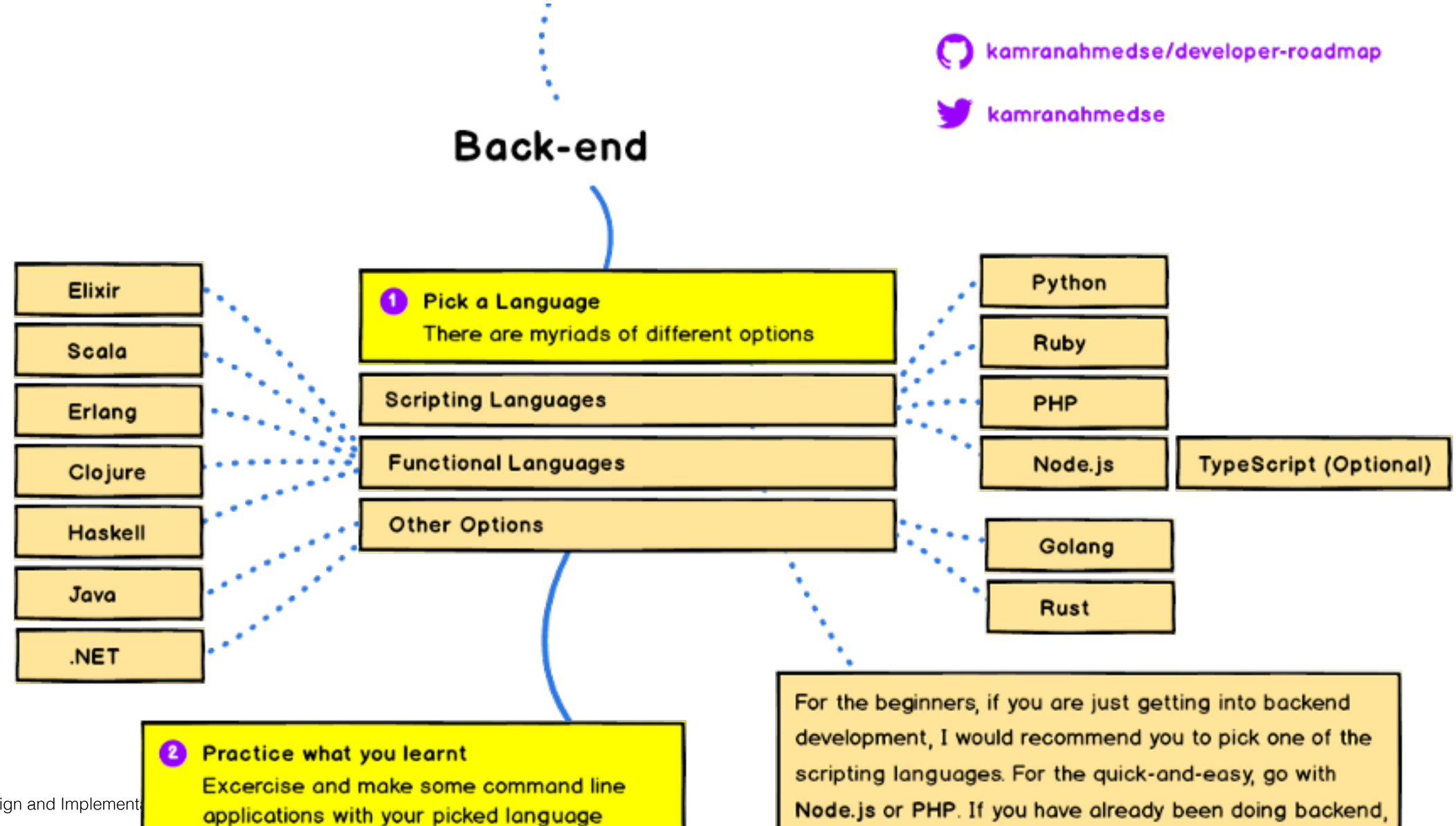
4.5 (378 ratings) 1,918 students enrolled

Created by in28Minutes Official Last updated 8/2019 English English

Programming Languages for Internet Applications



Programming Languages for Internet Applications



Development Methods for Internet Applications

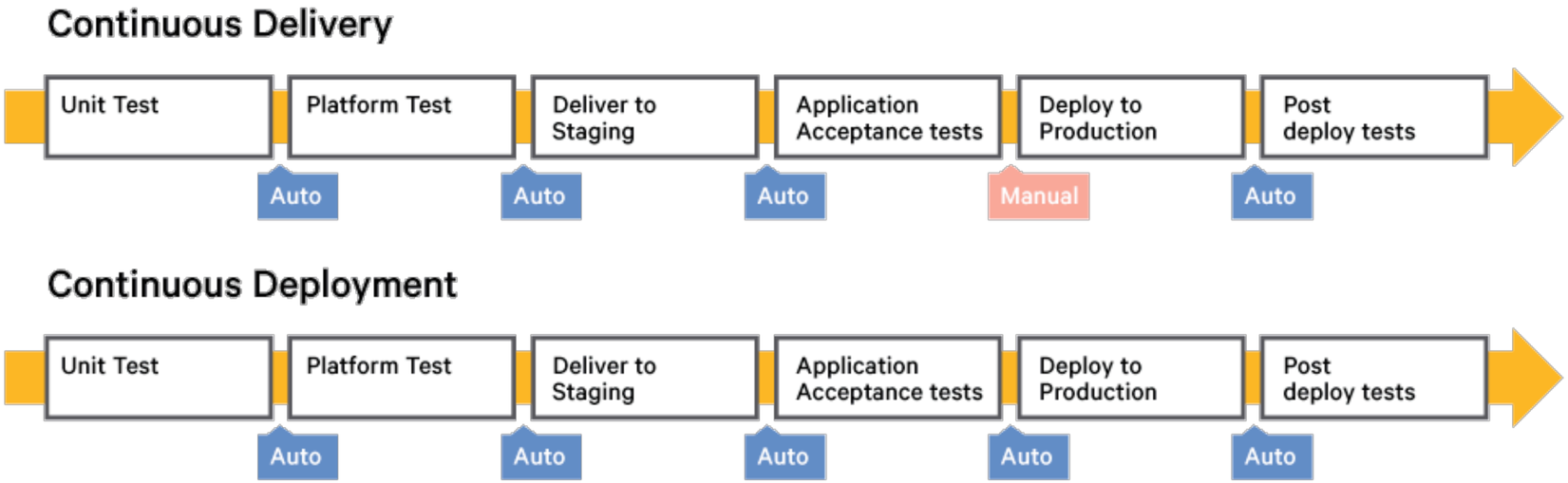
- Test Driven Development
  - Founded as part of the “extreme programming (XP)” methodology (1999)
  - Part of the Agile methodology
  - Steps:
    Add a test; Run all tests and see if the new test fails; Write the code; Run tests; Refactor code; Repeat
- Behaviour Driven Development
  - Tests are complete examples
- Generic Tests and Test Generation
  (e.g. QuickCheck)

Deployment Methods for Internet Applications

- Software development is increasingly competitive
- Any mistake can be extremely expensive
- Pressure is on to deliver fast and change even faster
- Companies deploy software at an astonishing pace:
  - Amazon: “every 11.7 seconds”
  - Netflix: “thousands of times per day”
  - Facebook: “bi-weekly app updates”

Deployment Methods for Internet Applications

- Processes and Methods for software construction and software deployment (DevOps)
- Specification and development methods
- Testing tools and toolchains
- Validation and Verification techniques



Course Overview

Internet Applications Design and Implementation

We focus on the **principles and concepts on the development of Internet applications**.

The syllabus follows an approach based on the fundamentals of software development based on **web and service oriented architectural patterns**, **advanced modularity mechanisms**, **data persistency abstractions**, **good development practices**, **performance concerns**, and **validation techniques**.

Lectures run along **practical assignments** and the **development of a running project** using frameworks, languages, and programming tools for Internet Applications that ensure the safety and compliance of the solution with relation to a specification

Goals: To Know

- Essential aspects of **architectural patterns** for inversion of control and software architectures specific for Internet Applications.
- **Principles of the development** of web applications and single page web applications.
- Mechanisms of **specifying and implementing web services** and web service orchestrations.
- **Internal structure** of an Internet **browser** and its **client applications**.
- Principles of **data-centric** and **user-centric development** in the context of Internet applications.
- Main **data abstraction** mechanisms used in Internet applications.
- Major performance **pitfalls** of Internet applications and their workarounds.
- Main specification and implementation mechanisms for **security policies in Internet Applications**.

Goals: To Do

- **Use development frameworks** that implement architectural styles for Internet applications.
- **Specify and build** web and cloud **applications** to support thin, flat, and native clients.
- **Specify and build** client **applications with reactive and rich behaviour**.
- **Implement authentication mechanisms** and **specify** the core **security rules** of an Internet Application
- Specify and efficiently use abstraction data layers such as **Object Relational Mappings** in Internet applications.
- **Design and deploy** Internet Applications that are efficient and maintainable.

Logistics (Plan)

- Online lectures (final format still to be defined, may require some student polls)
  - Regular Live lecture (2h), or
  - Weekly pre-recorded YouTube videos (approximately 1h per week) + new pre-recorded tutorials for the labs.
  - +Interactive discussions — must see videos first
- 7 isolated assignments in GitHub classroom, starting today, first submission on the 8th October
- Online on-campus Labs
- One-on-one Online Hours on-campus or online (by appointment)
- Resources:
  - Youtube (lecture and demo videos)
  - Github classroom (assignments, starter code)
  - CLIP (slides and other documents)
  - Discord (direct contact, conversation, questions)

Evaluation (CLIP to be updated soon)

- Written evaluation component (60%) — 2 on-campus Tests or Exam
- Laboratory work component (40%) — not mandatory / not valid for next year
  - teams of **3** members
  - 7 isolated assignments in GitHub classroom (first assignment 8th October 2021)

    If you don't know how to operate with git, go learn TODAY!
  - Submissions in the format of pull requests in GitHub,
  - Each assignment is evaluated for a pre-published set of criteria
  - assignments may include a written report,
  - A final presentation and discussion may be required to assert the obtained results.

Lecture Plan

Week	Lecture	Tutorial/Lab	Assignments
1	Overview and Logistics	Basic HTML/CSS/Javascript Technologies	
2	Software Archirecture		HTML & Javascript basics
3	Specification of RESTful APIS	Design of Server Application in Kotlin	
4	Frameworks for web and service-based applications		RESTful API specification and implementation
5	Data Abstraction. Data Access Patterns.	JPA layer, associations and custom queries	
6	Data Abstraction in Spring (SpringData)		An architecture using Controllers/ Services and JPA
7	Security in Internet Applications	Token Based Security and Policies	
8	Client Applications		A secure server application
9	Client Frameworks and Languages		
10	Specification and Implementation of Interactive Systems - IFML	Javascript First / React Applications	
11	Model transformations		IFML specification of an application
12	State Management	Tutorial: Redux Demo	
13	Redux/Guest Lecture		Client application: redux and react
	CHRISTMAS!		
14	Guest Lecture (TBD)		A secure client application

Lab Exercises

- Tech Stack (mandatory):
  - Server-side: SpringBoot + Kotlin + Swagger + JUnit5 + MySQL
  - Client-side: TypeScript + React + Redux
- Tools stack
  - git (GitHub — mandatory),
  - IntelliJ
  - httpie/curl/postman

Bibliography

spring

TypeScript

learn play download interact

tutorial handbook samples language spec

Kotlin

Bibliography

- Marco Brambilla and Piero Fraternali. Interaction Flow Modeling Language – Model-Driven UI Engineering of Web and Mobile Apps with IFML. Morgan Kaufmann. 2014
- Martin L. Abbott and Michael T. Fischer, The Art of Scalability: Scalable Web Architecture, Processes and Organizations for the Modern Enterprise, Addison-Wesley Professional. 2009
- Martin Fowler. Patterns of Enterprise Application Architecture. Addison-Wesley. 2002
- Software Architecture in Practice (3rd ed). Len Bass, Paul Clements, and Rick Kazman. Addison-Wesley 2015.
- Craig Walls. Spring in Action (4th edition). Manning. 2015

spring.io facebook.github.io/react/ typescriptlang.org kotlinlang.org

applications
web
service
techniques
development
Internet
services
mobile
mechanisms
orchestrate
clients
modularity
patterns
data
specific
advanced
tools
single
providers
oriented
page
pages
architectural
resources
software
Clients
validation
native
abstractions
methods
compound
Components
performance
collaborative
online
reactive
result
practices
persistency
challenges
concerns
good
integration
fundamentals
Applications
Spring
REST
Model
Hibernate
MVC
TDD
Web
Beans
Aspects
JPA
Service