

Integrarea sistemelor informatice



Suport curs nr. 2

Programator >> Arhitect

Dezvoltare software în practică

2024-2025

C2 – Dezvoltare software în practică

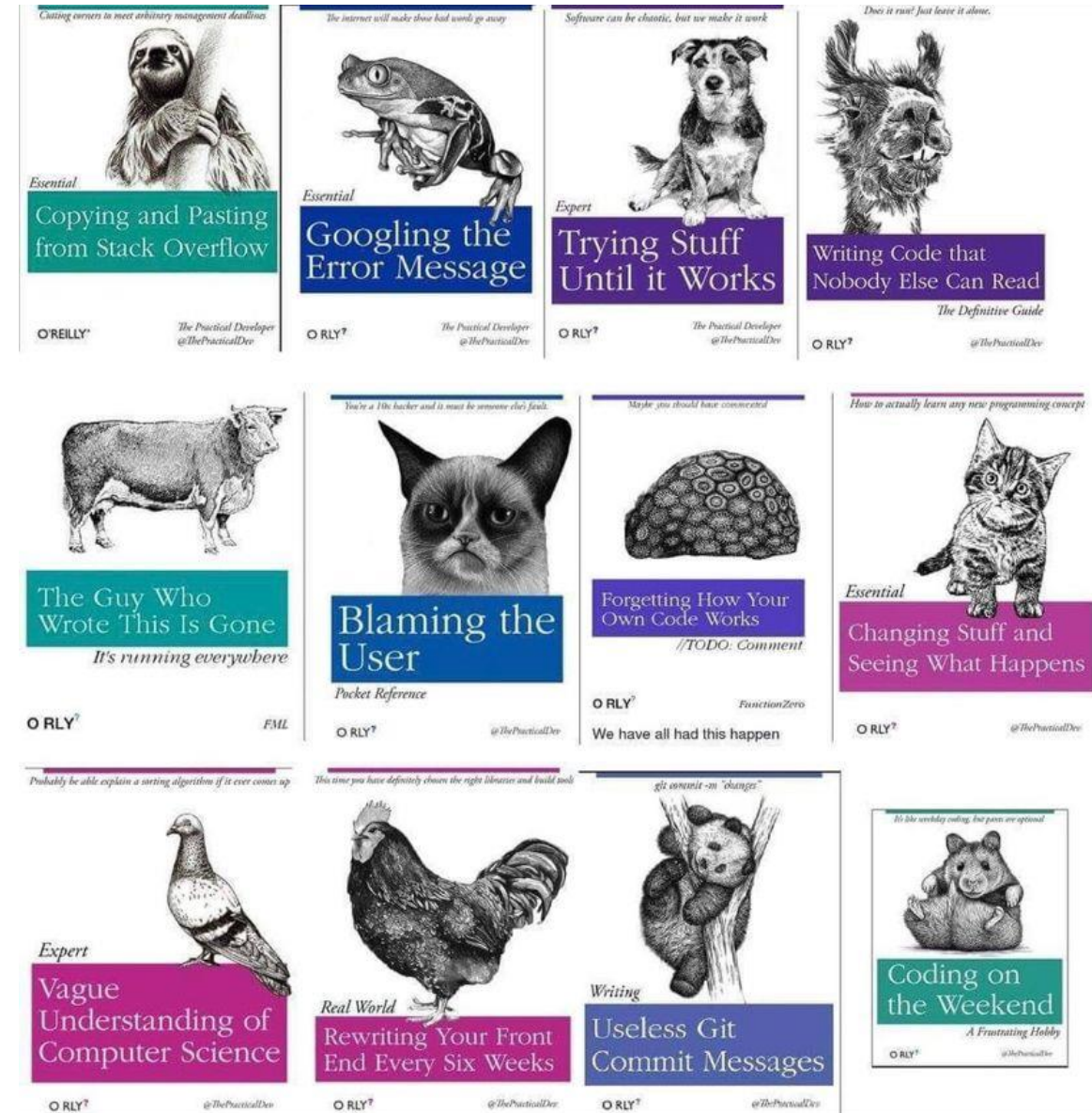
Obiective

- Analiza problemelor practice în dezvoltarea de software
- Exemplificarea arhitecturilor software în practică
- Clasificarea șabloanelor de proiectare software (design patterns)
- Înțelegerea domeniilor conexe în dezvoltarea aplicațiilor software
 - Sisteme de versionare
 - DevOps, SRE, Platform Engineering
 - Deployment

Aventurile unui programator în practică

Top developer's responses when their programs don't work...

10. "It must be a hardware problem."
 9. "I haven't touched that module in weeks!"
 8. "Somebody must have changed my code."
 7. "Did you check for a virus on your system?"
 6. "You must have the wrong version."
 5. "That's weird..."
 4. "There must be something wrong with your data"
 3. "It's never done that before."
 2. "It worked yesterday."
- And the #1 excuse...
1. "It works on my machine"

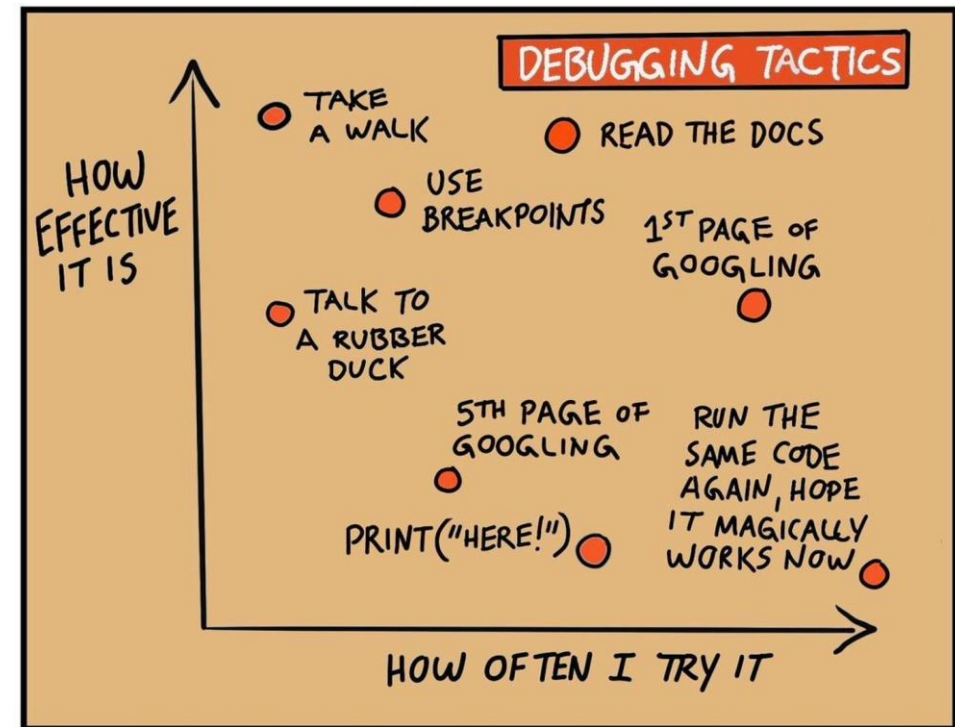


Aventurile unui programator în practică

Metode de debugging

- 📄 Print-uri – simplu, punctual, dar iterativ (necesită potențial mai multe încercări)
- ● Breakpoint-uri – exact, dar lent și dificil de folosit în aplicații complexe, asincrone, sau multithreaded
- 🦆 Rubber ducking – explicarea codului unei rațe de cauciuc va revela adesea problema reală în termeni simpli
- 🚶 Plimbare în parc – problemele devin mai simple atunci când sunt puse în perspectivă și soluțiile vin de la sine

Debugging Tactics

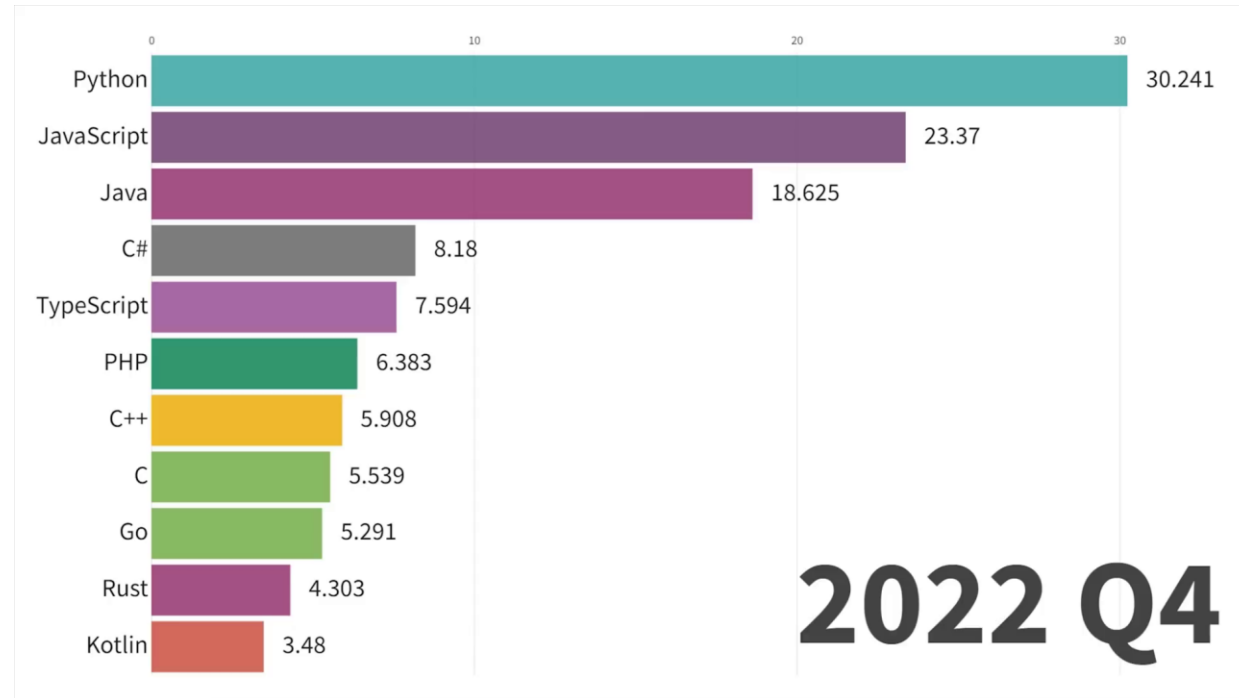
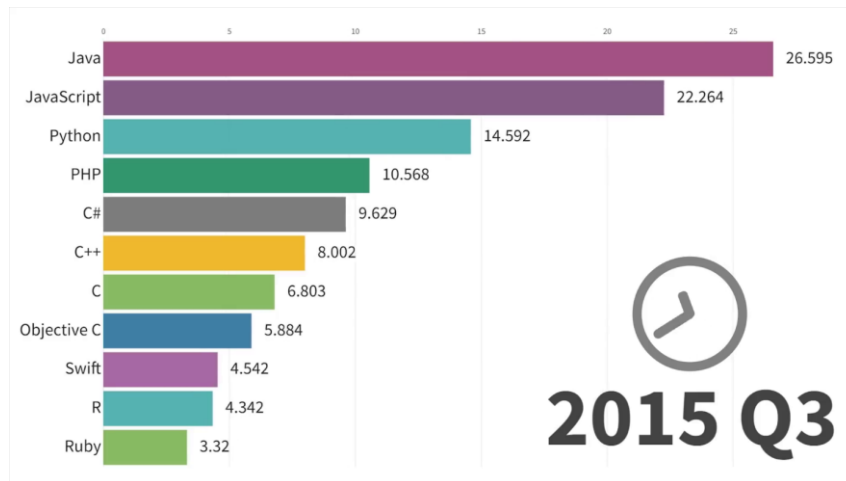


Original: Forrest Brazeal

Limbaje de programare

Cum alegem limbajul de programare?

Ce determină popularitatea unui limbaj de programare?

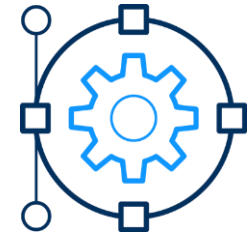


Sursa: NEW! Most Popular Programming Languages 1965 – 2022, YouTube

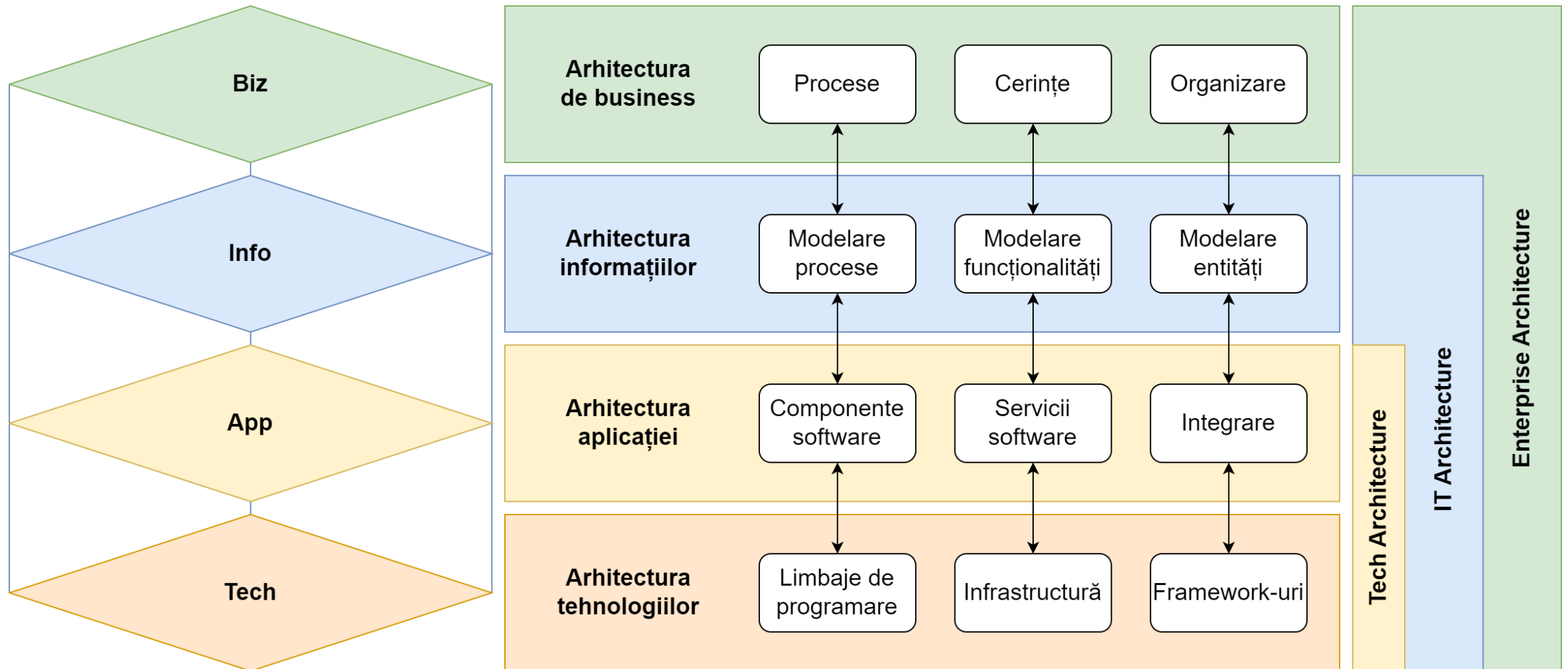
Procesul de dezvoltare a aplicațiilor software

Etape și principii de bază (din perspectivă tehnică)

- Înțelegerea problemei și a cerințelor: funcționale (business), non-funcționale (performanță)
- Alegerea unei arhitecturi (structură, organizare, implementare)
- Alegerea unui framework (modelare UML, framework software)
- Aplicarea principiilor arhitecturale (design patterns)
- Documentarea soluției (diagrame, comentarii, prezentări, wiki)
- Actualizarea periodică a soluției (îmbunătățire continuă)



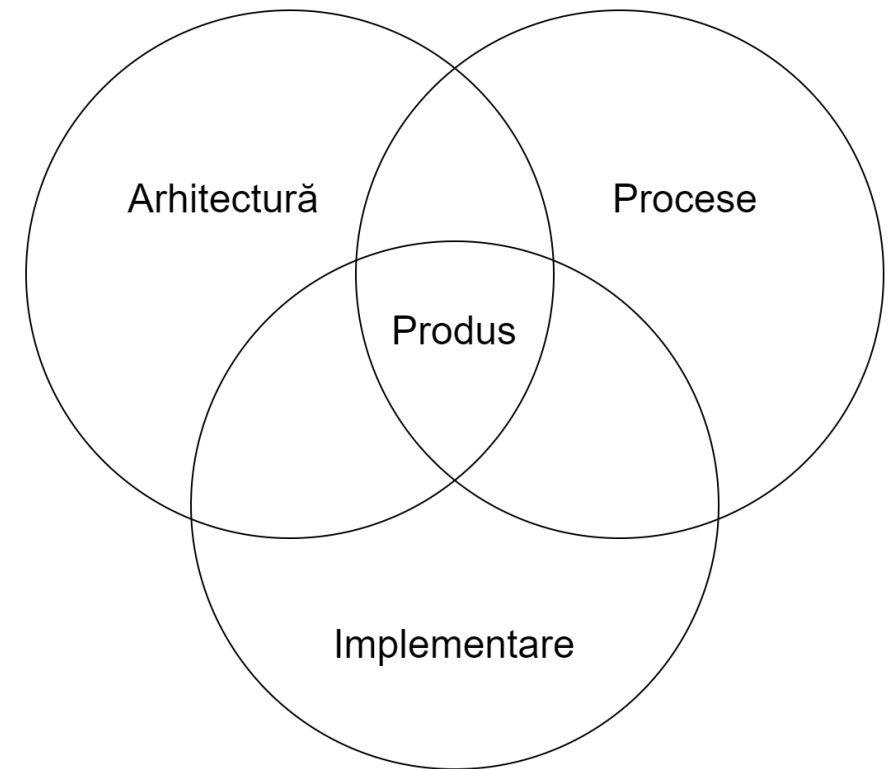
Procesul de dezvoltare a aplicațiilor software



Arhitectură vs implementare

Arhitectura sistemelor software este esențială pentru definirea funcționalităților de implementat

- Definirea componentelor sistemului și a interacțiunilor dintre ele
- Alegerea unor soluții standard pentru funcționalitățile uzuale
- Dezvoltarea unor sisteme mai robuste și mentenabile

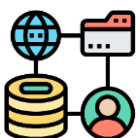


Evoluția arhitecturilor software (1980 – prezent)



Dezvoltare

Waterfall → Agile → DevOps



Arhitectură

monolit → N-Tier → microservicii



Distribuție/Deployment

servere fizice → mașini virtuale → containere virtuale



Infrastructură

servere locale → servere hostate → infrastructură cloud



Sursa: ByteByteGo, 2023

What is Cloud Native?

blog.bytebytego.com

	1980 - 1990	2000	2010 - Cloud
Development Process	 Waterfall	 Agile	 DevOps
Application Architecture	 Monolithic	 N-Tier	 Microservices
Deployment & Packaging	 Physical server	 Virtual server	 Container
Application Infrastructure	 Data center	 Hosted	 Cloud

Reference: <https://www.oracle.com/cloud/cloud-native/what-is-cloud-native/>

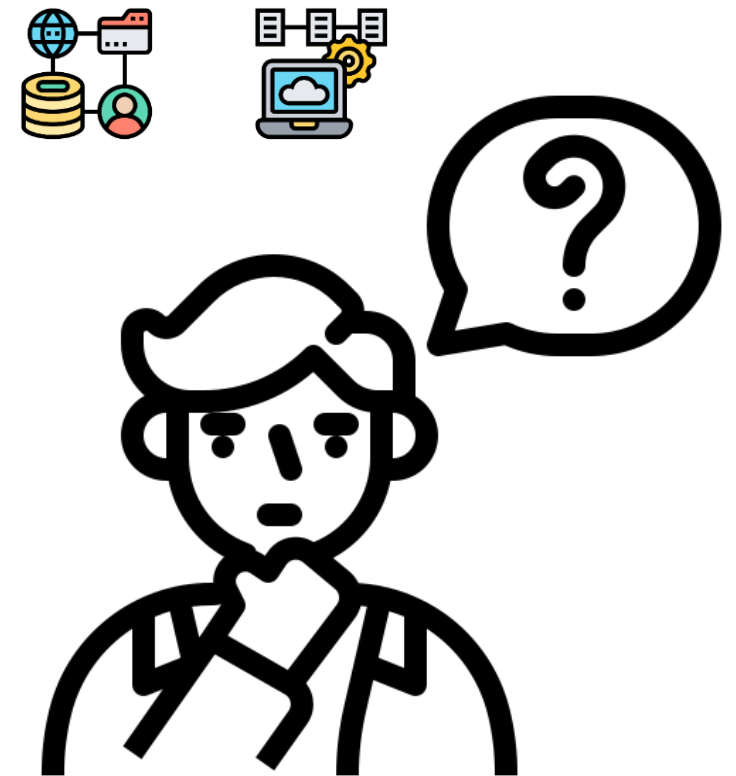
Arhitectură vs implementare în practică

Totuși cât de complex trebuie să gândim **arhitectura** în practică?

... și cât de complex trebuie să gândim **implementarea** în practică?

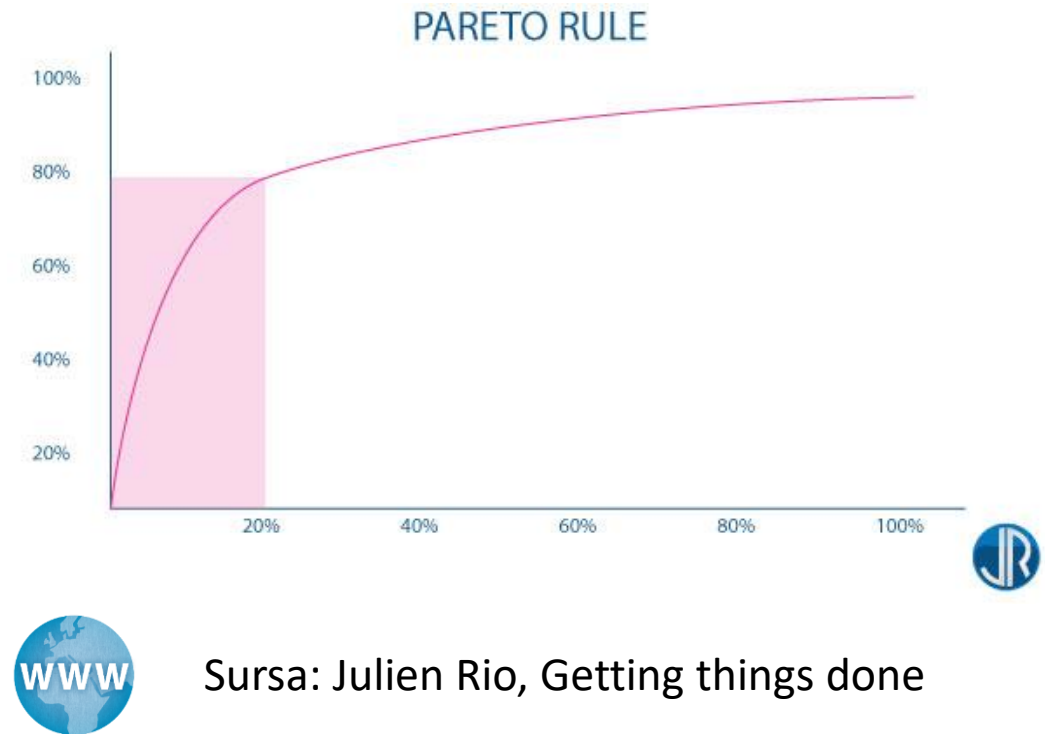
Unde ar trebui să punem mai mult accent: pe partea de **design/proiectare** ... sau pe **implementarea** efectivă a aplicației?

- “Insanely great” – Steve Jobs prezintă Apple Macintosh în 1984
- “Done is better than perfect” – Mark Zuckerberg enunță principiile de bază ale conducerii Facebook



Principii în dezvoltarea proiectelor software

- Principiul Pareto (80/20)
 - 20% timp = 80% rezultate
- Legea lui Parkinson
 - Timpul necesar pentru finalizarea unui proiect crește pe măsură ce termenul este mai îndepărtat
- Legea lui Hofstadter
 - Întotdeauna durează mai mult decât te aștepti, chiar dacă termenul include și legea lui Hofstadter
- Cunoașteți și o altă lege universală?

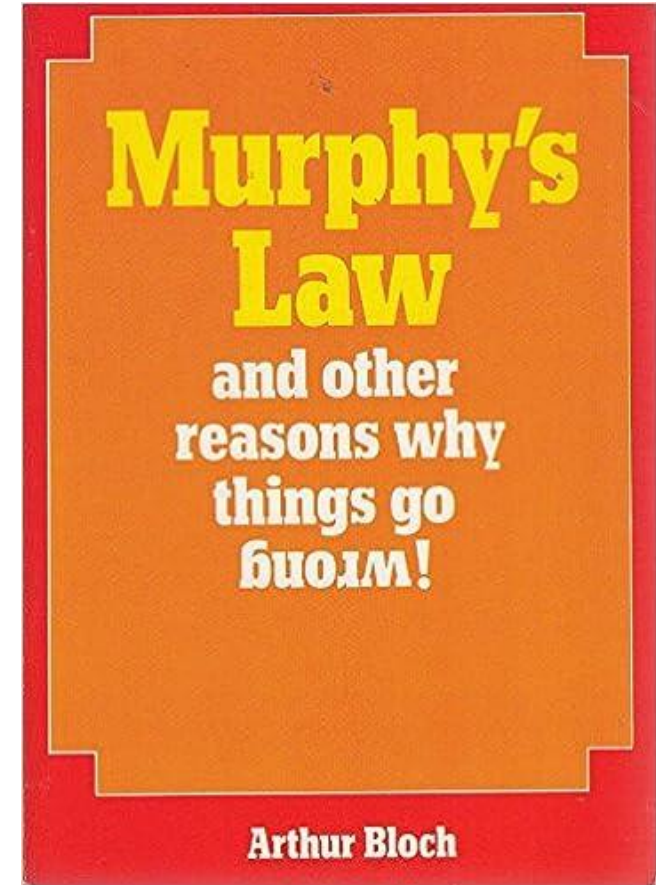


Legea lui Murphy în practică

Legea de bază a lui Murphy: Dacă ceva poate merge rău, atunci sigur va merge rău

Câteva extrase pe domeniul IT:

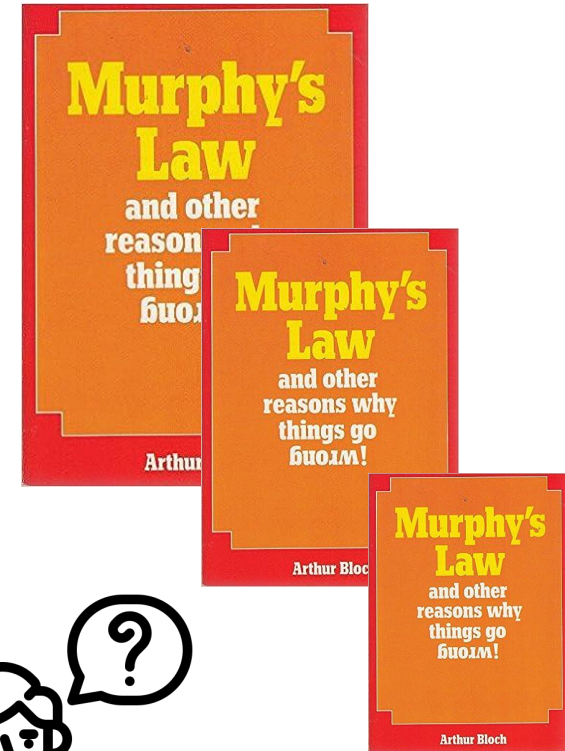
- Cu cât o modificare apare mai inofensivă, cu atât influența sa se va extinde ulterior și cu atât mai multe proiecte vor trebui refăcute
- Numai după ce un program a fost rulat în producție cel puțin 6 luni, se va descoperi în el o eroare fundamentală
- Orice sistem care depinde de fiabilitatea oamenilor este nefiabil



Dezvoltare software vs legea lui Murphy

Totuși, ce putem face pentru a minimiza efectele legii lui Murphy?

- Mai mult ca sigur, multe dintre problemele cu care urmează să ne confruntăm pe parcursul dezvoltării software **au mai fost rezolvate** în alte proiecte
- Cum identificăm **soluțiile aplicabile** în funcție de necesitățile proiectului nostru?



Dezvoltare software – Design Patterns

Design pattern-urile reprezintă **soluții șablon** pentru probleme comune în dezvoltarea de software

- Definesc un limbaj comun, foarte util pentru comunicarea în cadrul echipei de dezvoltare
- Diferă prin complexitate, nivel de detalii și aplicabilitate: intenție, motivație, structură, secvențe de cod



Sursa: Refactoring Guru, Design Patterns



Dezvoltare software – Design Patterns

Clasificare

Creăţionale – creare de obiecte

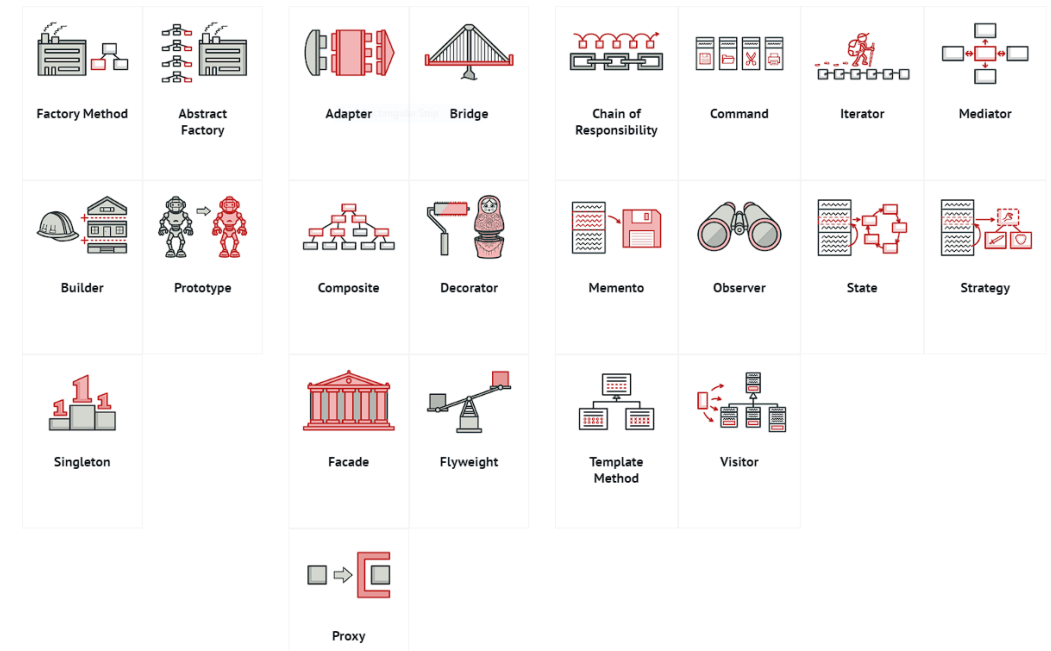
Abstract factory, Builder, Prototype, Singleton

Structurale – structuri de obiecte/clase

Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy

Comportamentale – interacţiuni între obiecte

Chain of responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy, Template, Visitor

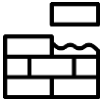


Dezvoltare software – Design Patterns

Creăţionale



Abstract Factory – Creează familii de obiecte



Builder – Construieşte obiecte complexe în mod incremental



Prototype – Creează copii ale obiectelor predefinite



Singleton – o clasă cu o singură instanţă



Factory Method

Provides an interface for creating objects in a superclass, but allows subclasses to alter the type of objects that will be created.



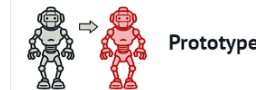
Abstract Factory

Lets you produce families of related objects without specifying their concrete classes.



Builder

Lets you construct complex objects step by step. The pattern allows you to produce different types and representations of an object using the same construction code.



Prototype

Lets you copy existing objects without making your code dependent on their classes.



Singleton

Lets you ensure that a class has only one instance, while providing a global access point to this instance.

Dezvoltare software – Design Patterns

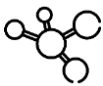
Structurale (1)



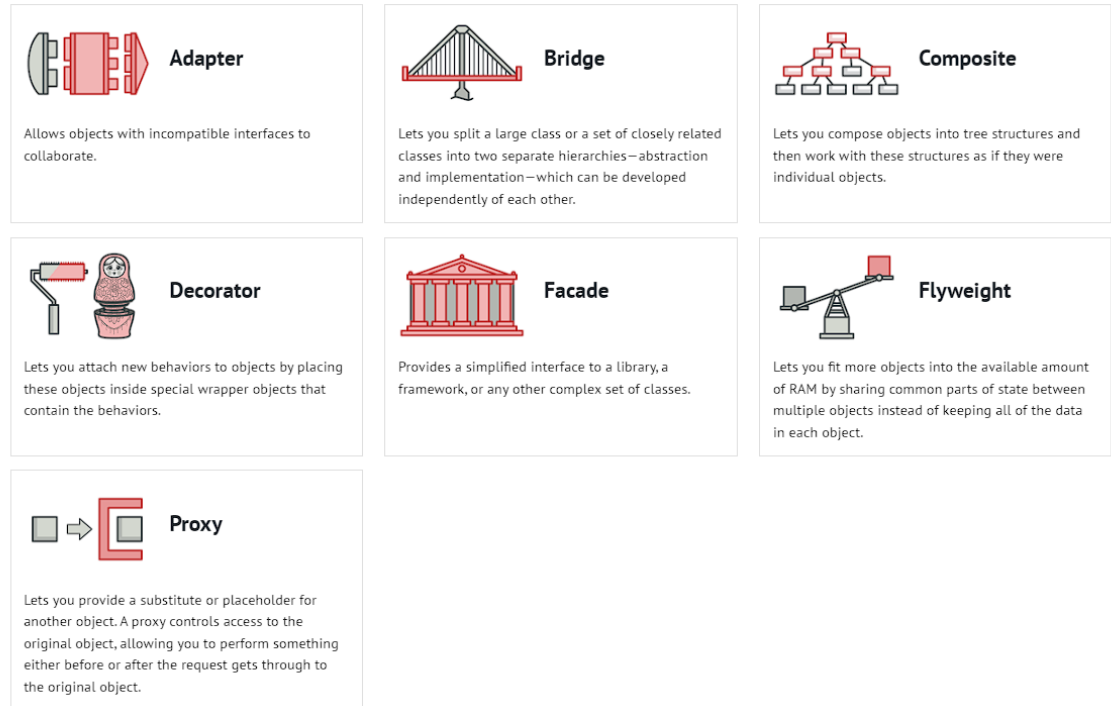
Adapter – Permite interacțiunea dintre obiecte cu interfețe incompatibile



Bridge – Conectează implementarea obiectelor cu funcționalitatea lor



Composite – Compune structuri (arborescente) de obiecte



Dezvoltare software – Design Patterns

Structurale (2)



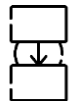
Decorator – Adaugă funcționalități obiectelor fără a le schimba implementarea



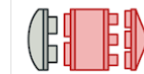
Facade – Furnizează o interfață singulară, simplificată pentru un set complex de clase (de ex. o bibliotecă)



Flyweight – Partajează obiecte reutilizabile în mod eficient



Proxy – Furnizează un substitut al altui obiect, prin care este posibilă interacțiunea controlată cu acesta



Adapter

Allows objects with incompatible interfaces to collaborate.



Bridge

Lets you split a large class or a set of closely related classes into two separate hierarchies—abstraction and implementation—which can be developed independently of each other.



Composite

Lets you compose objects into tree structures and then work with these structures as if they were individual objects.



Decorator

Lets you attach new behaviors to objects by placing these objects inside special wrapper objects that contain the behaviors.



Facade

Provides a simplified interface to a library, a framework, or any other complex set of classes.



Flyweight

Lets you fit more objects into the available amount of RAM by sharing common parts of state between multiple objects instead of keeping all of the data in each object.



Proxy

Lets you provide a substitute or placeholder for another object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.

Dezvoltare software – Design Patterns

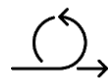
Comportamentale (1)



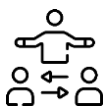
Chain of Responsibility – Pasează o cerere printr-un set de obiecte până când este preluată



Command – Transformă o cerere într-un obiect



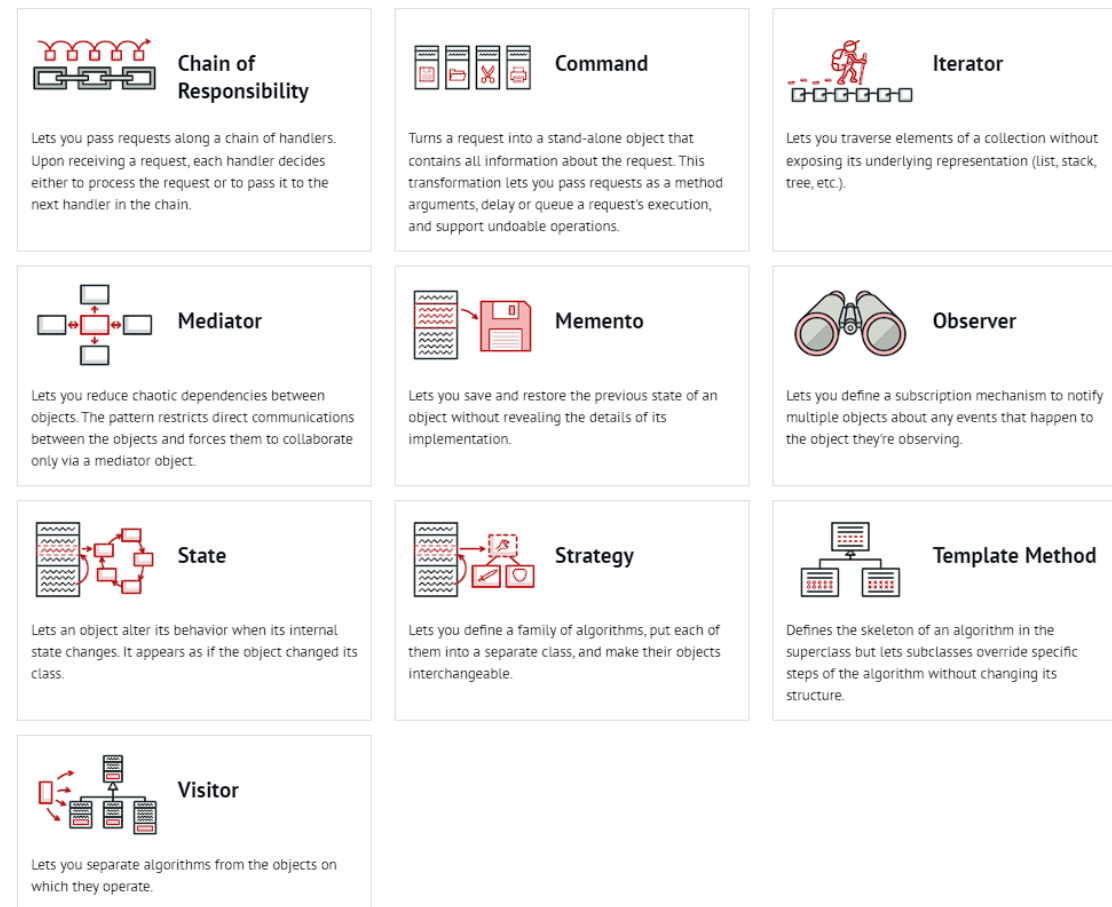
Iterator – Accesează elementele unei colecții în mod secvențial



Mediator – Simplifică interacțiunile și dependențele dintre obiecte diferite



Memento – Salvează și reconstituie starea unui obiect




Dezvoltare software – Design Patterns

Comportamentale (2)

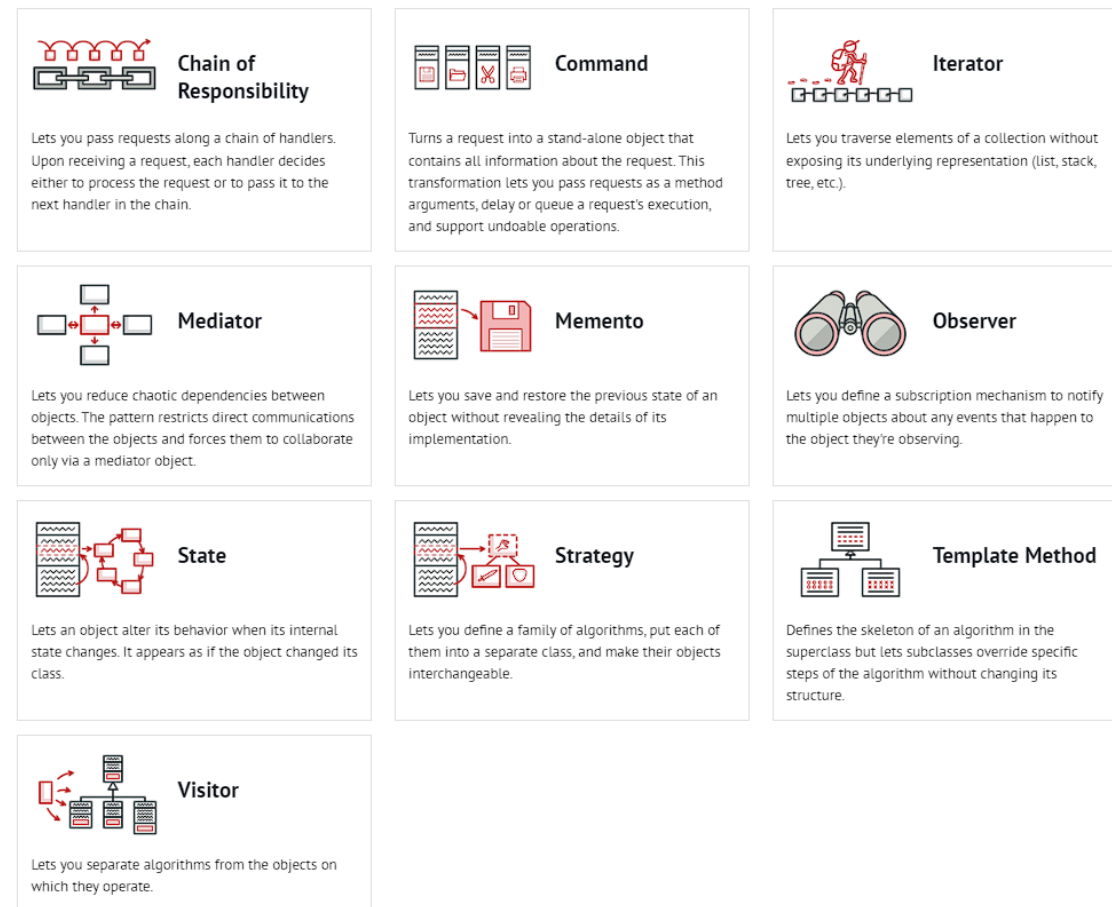
 **Observer** – Notifică obiectele subscrise la evenimente ce apar în alte obiecte

 **State** – Permite modificarea comportamentului unui obiect atunci când se modifică starea internă

 **Strategy** – Definește familii de algoritmi interschimbabili

 **Template** – Definește scheletul unui algoritm într-o superclasă

 **Visitor** – Adaugă operații noi unei clase fără a o modifica



Dezvoltare software – Design Patterns ex.



[creaționale]

Singleton

- Asigură că o clasă are o singură instanță și furnizează un mod global de acces la acea instanță
- **Exemple de utilizări**
 - clasă care gestionează conexiunea la o bază de date
 - serviciu de printare (en. print spooler)

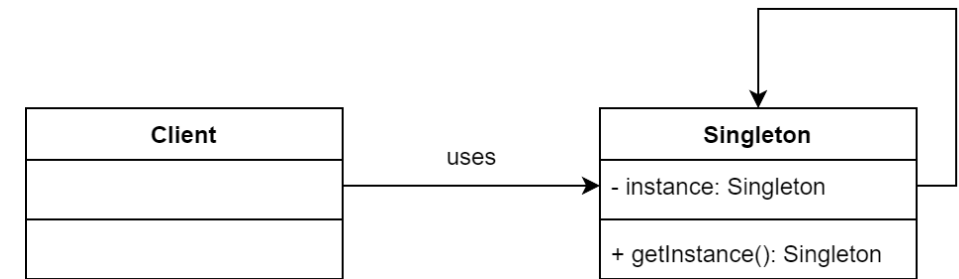


diagrama de clase

Dezvoltare software – Design Patterns ex.



[structurale]

Facade

- Furnizează o **interfață simplificată** pentru o structură complexă de clase
- **Exemple de utilizări**
 - clasă care gestionează validarea unei cereri într-un sistem compus
 - interfața de control a unui sistem Smart Home

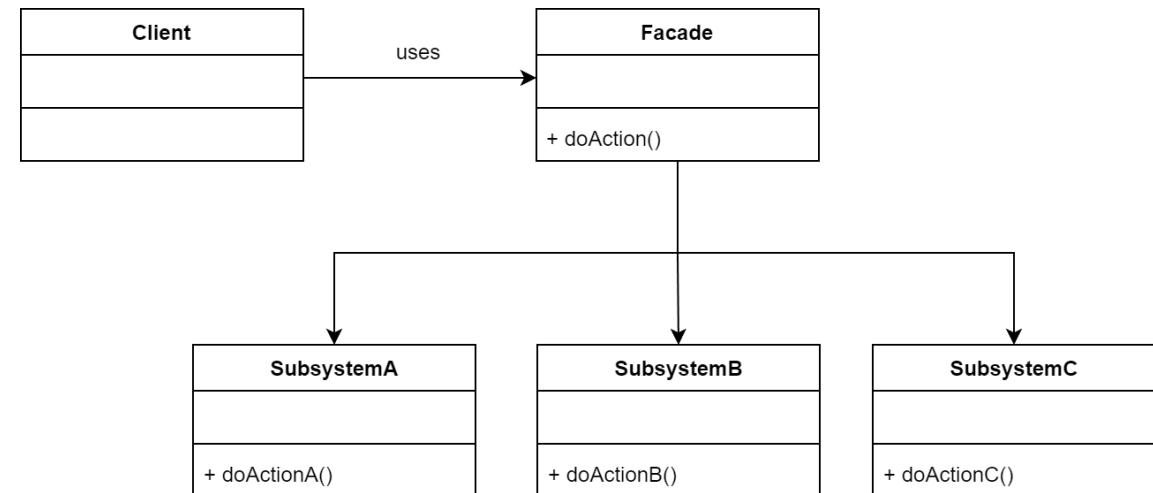


diagrama de clase

Dezvoltare software – Design Patterns ex.



[comportamentale]

Observer

- Asigură transmiterea evenimentelor către alte obiecte, în mod publish/subscribe
- **Exemple de utilizări**
 - clasă care gestionează transmiterea de notificări la apariția unor evenimente
 - abonarea la notificări despre vreme

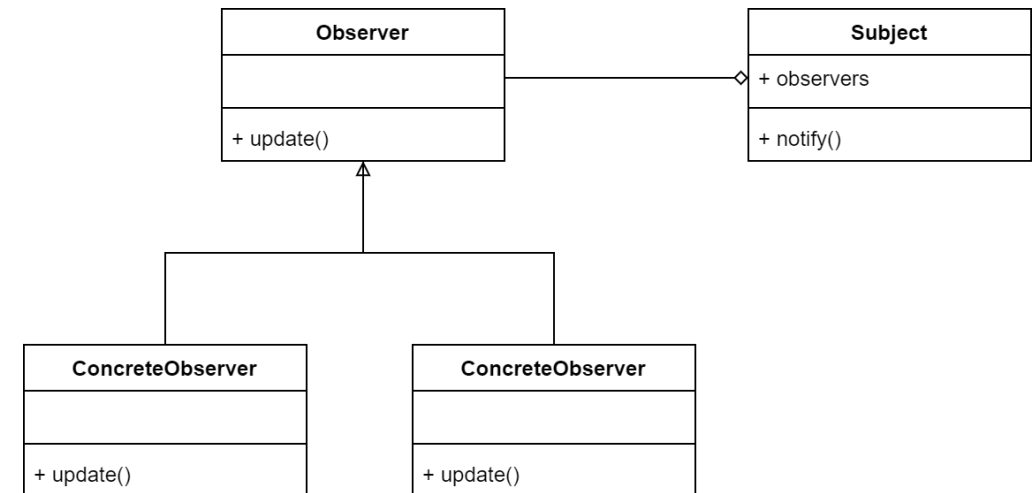
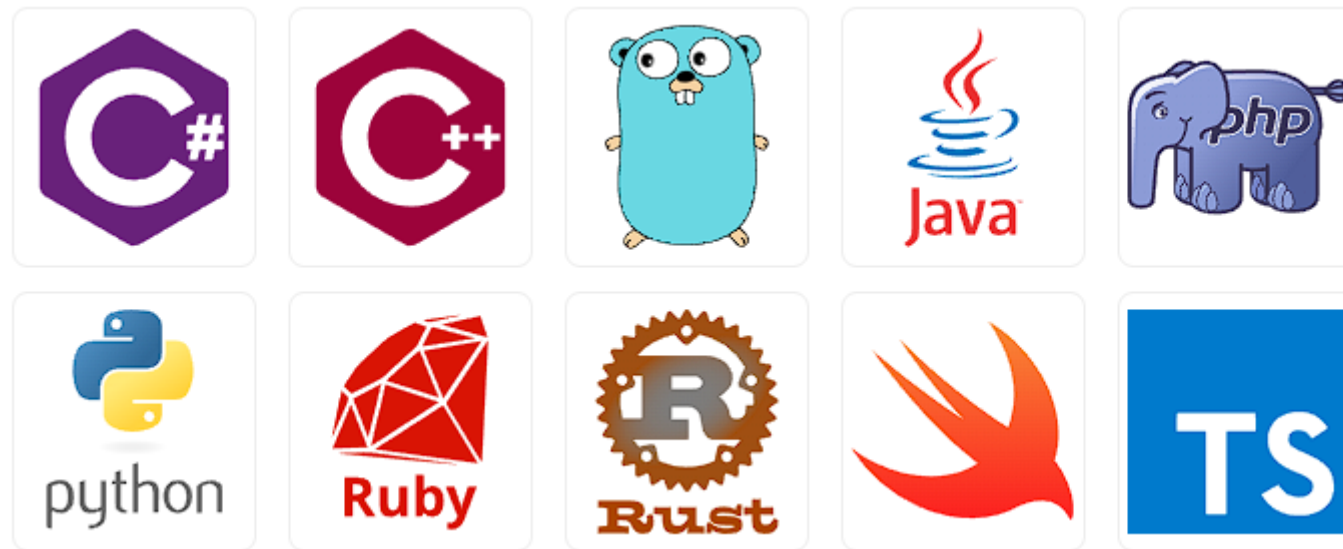


diagrama de clase

Dezvoltare software – Design Patterns ex.

Exemple practice

[Code Examples of Design Patterns](#)



Integrarea sistemelor – continuare

Pe lângă procesul de dezvoltare a aplicațiilor, există și alte aspecte/activități ce țin de integrarea sistemelor software

Exemple

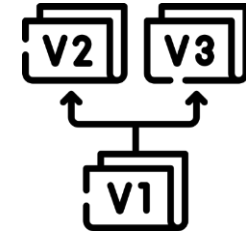
DEV.



PROD.

- Sisteme de versionare
- Distribuție/Deployment
- DevOps → SRE* → Platform Engineering

*Site Reliability Engineering



Integrarea sistemelor – Sisteme de versionare

Git – Unde se află codul aplicației?

LOCAL



DISTRIB.

Working directory

Sistemul de fișiere local

Staging area

Locație temporară pentru fișierele pregătite pentru versionare

Local repository

Conține codul versionat pe sistemul local

Remote repository

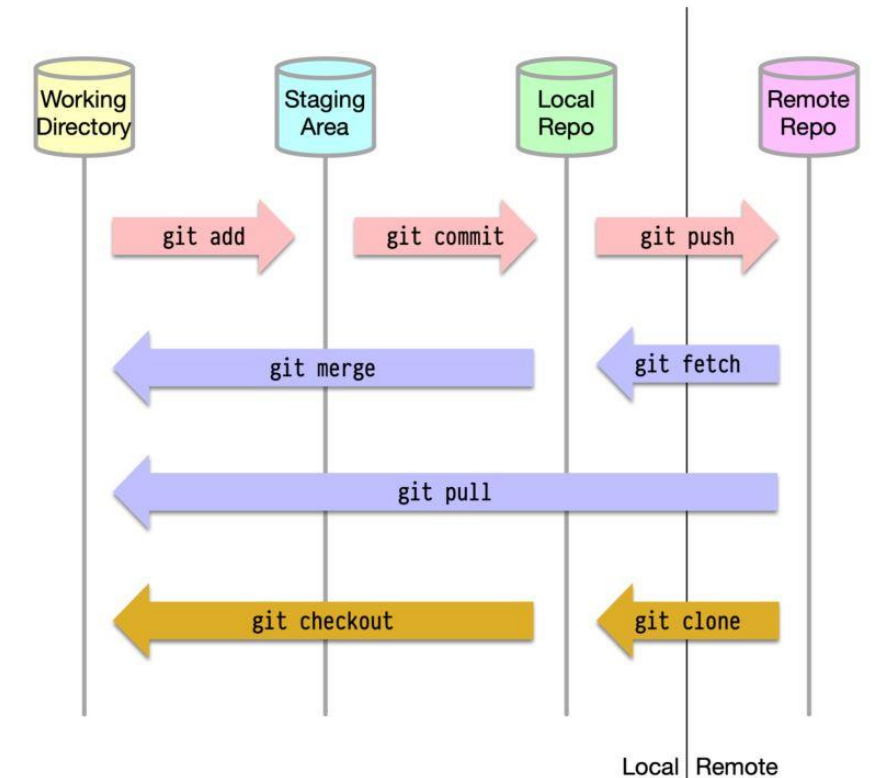
Conține codul versionat pe un server



Sursa: ByteByteGo, 2023

How Git Commands work

ByteByteGo.com



Integrarea sistemelor – Sisteme de versionare

- Ce este un **repo**? (repository)
- Cum poate fi el structurat?
- Monorepo vs Microrepo
 - Monorepo – un singur repo pentru întreaga bază de cod (dependențe comune)
 - Microrepo – repo-uri separate pentru aplicații / proiecte diferite (resurse separate)



Sursa: ByteByteGo, 2023

Monorepo vs Microrepo: which is the best?

ByteByteGo.com

	Monorepo	Microrepo
 Company		
 Collaboration	 Services work under the same repository.	 Service owners work under separate repositories.
 Dependency	 Services share the same dependency.	 Services choose their own dependency.
 Scalability	 Services share the same standard.	 Services set their own standard.
 Tooling		

Integrarea sistemelor – Sisteme de versionare

- Exemple
 - Google, Meta – folosesc monorepo deși au o bază de cod imensă (dar **politici comune** pentru aplicațiile partajate)
 - Amazon, Netflix – folosesc microrepo pentru arhitectura preponderent bazată pe **microservicii** (gestionare diferită a resurselor)



Sursa: ByteByteGo, 2023

Monorepo vs Microrepo: which is the best?

ByteByteGo.com






























	Monorepo	Microrepo
 Company		
 Collaboration	 Services work under the same repository.	 Service owners work under separate repositories.
 Dependency	 Service share the same dependency.	 Service choose their own dependency.
 Scalability	 Services share the same standard.	 Services set their own standard.
 Tooling		

Integrarea sistemelor – Distribuție

Strategii de distribuție/deployment după etapa proiectului/dimensiunea companiei:

- **S. prototipare** – distribuție manuală
- **M. dezvoltare** – distribuție programată, proceduri de testare, monitorizare
- **L. scalare** – integrare instrumente comerciale (ex. Atlassian), automatizare, QA
- **XXL. proiecte mature** – testare controlată în producție, monitorizare avansată, automatizare, suport

Company Size vs. Tools They Use To Ship to Production blog.bytebytego.com

	S	M	L	XXL
Engineer	1-10	10-100	100 - 1000	1000 - 10,000+
Develop	 Visual Studio Code  GitHub Free or low-cost Truck based development	 Visual Studio Code  IntelliJ IDEA  GitHub Enterprise Free and commercial Truck/Feature based development	 Confluence  Jira  Visual Studio Code  IntelliJ IDEA  GitHub Enterprise  Jenkins Largely Commercial Feature based development	 Confluence  Jira  Visual Studio Code  IntelliJ IDEA  git  Workflow Commercial or customized tooling Truck based development
Testing	 Manual Testing	 Quality Assurance	 Quality Assurance + Automated Testing	 Automated Testing + Quality Assurance
Deploy	 Manual deployment	 Schedule-based deployment	 Schedule + staged canary rollout	 Schedule + Staged canary rollout + experiment
Operations	 Customer report	 Monitoring/Alert + Tiered Customer Support + Customer report	 Monitoring/Alert + Tiered Customer Support + Customer report	 SLA/SLO + Monitoring/Alerting + Tiered Customer Support

Integrarea sistemelor – Distribuție

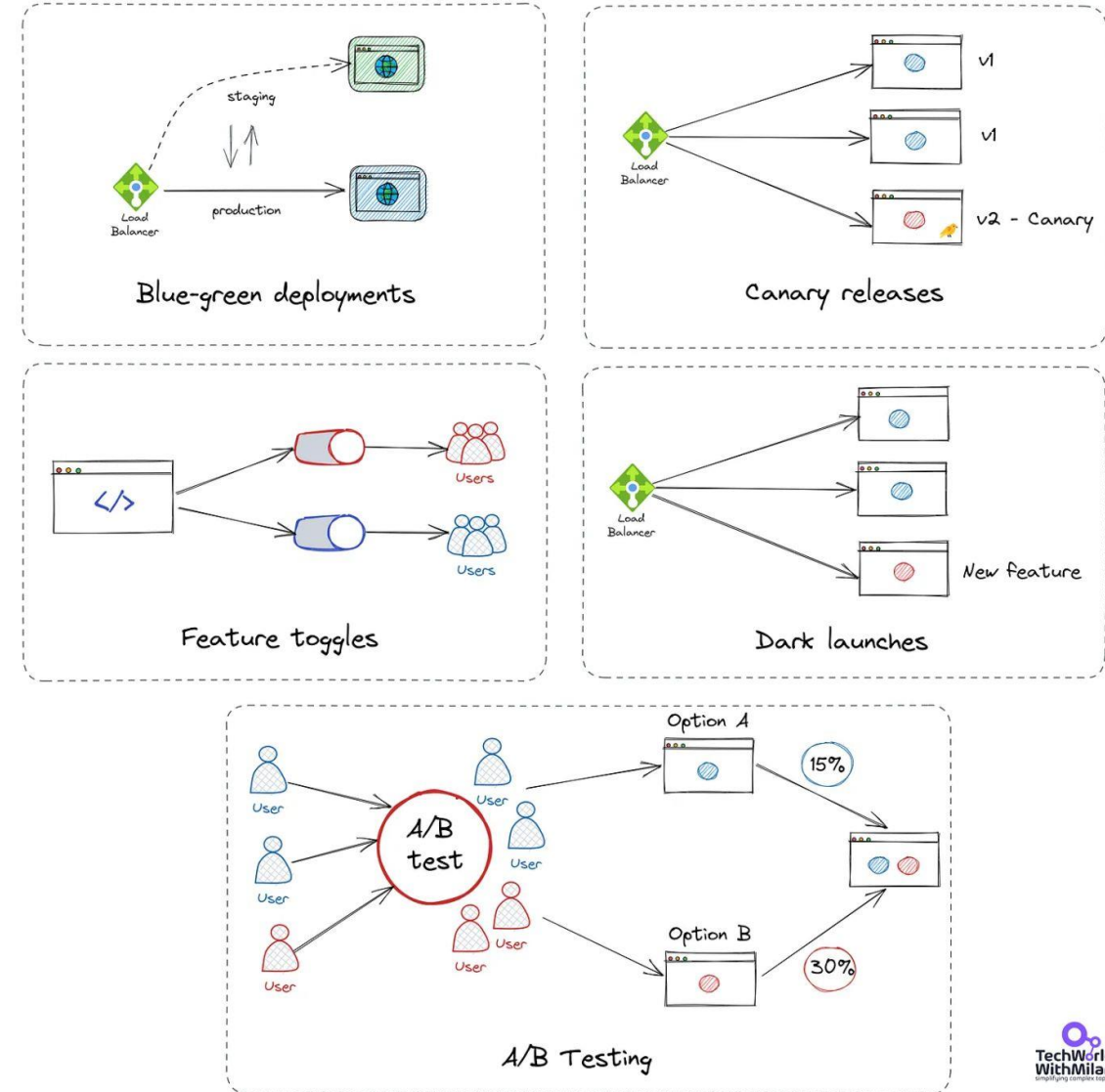
Strategii

Blue/Green deployment – rulare simultană/concurentă pe două instanțe și activarea uneia dintre ele

Feature toggles – activare/dezactivare funcționalități în producție

A/B Testing – evaluare comparativă

Canary releases / Dark launches – testare versiune nouă / funcționalități noi cu un grup select



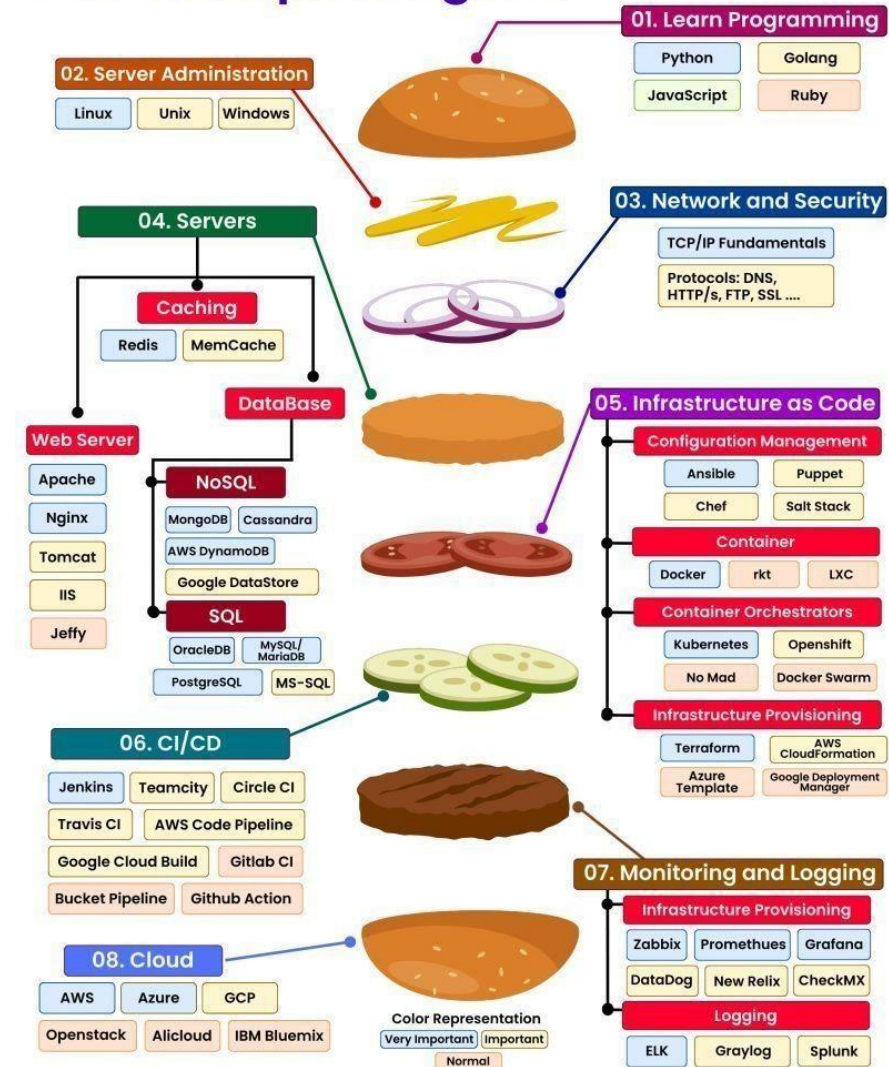
Integrarea sistemelor – DevOps

DevOps este un domeniu care combină **dezvoltarea software (Dev)** și **operațiunile IT (Ops)** pentru a eficientiza procesul de dezvoltare și implementare a software-ului

Aspecte cheie:

- Gestionare resurse
- Integrare continuă (continuous integration /CI)
- Distribuție/Deployment (continuous deployment /CD)

The DevOps Burger !!



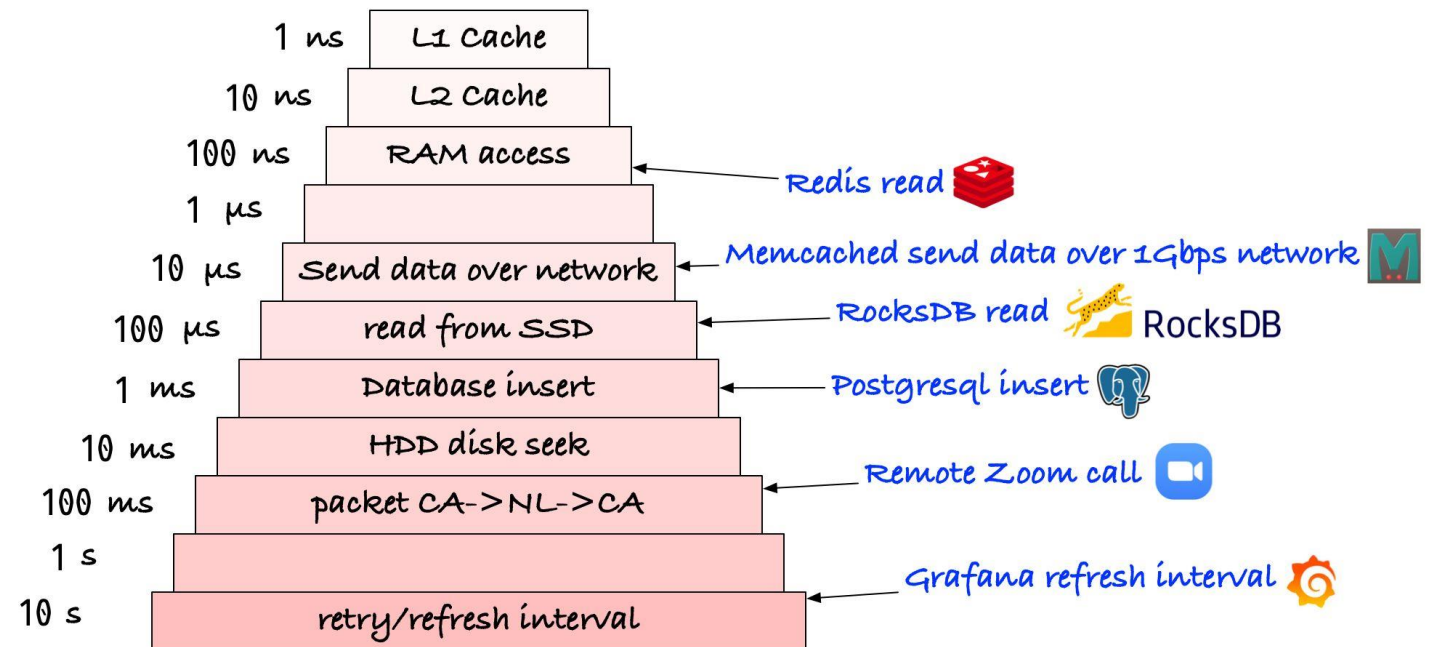
Integrarea sistemelor – DevOps

Problemă: dezvoltarea aplicațiilor de mare capacitate

- Scalabilitate
- Necesită optimizări
- Nu doar la nivel de cod

Latency Numbers You Should Know

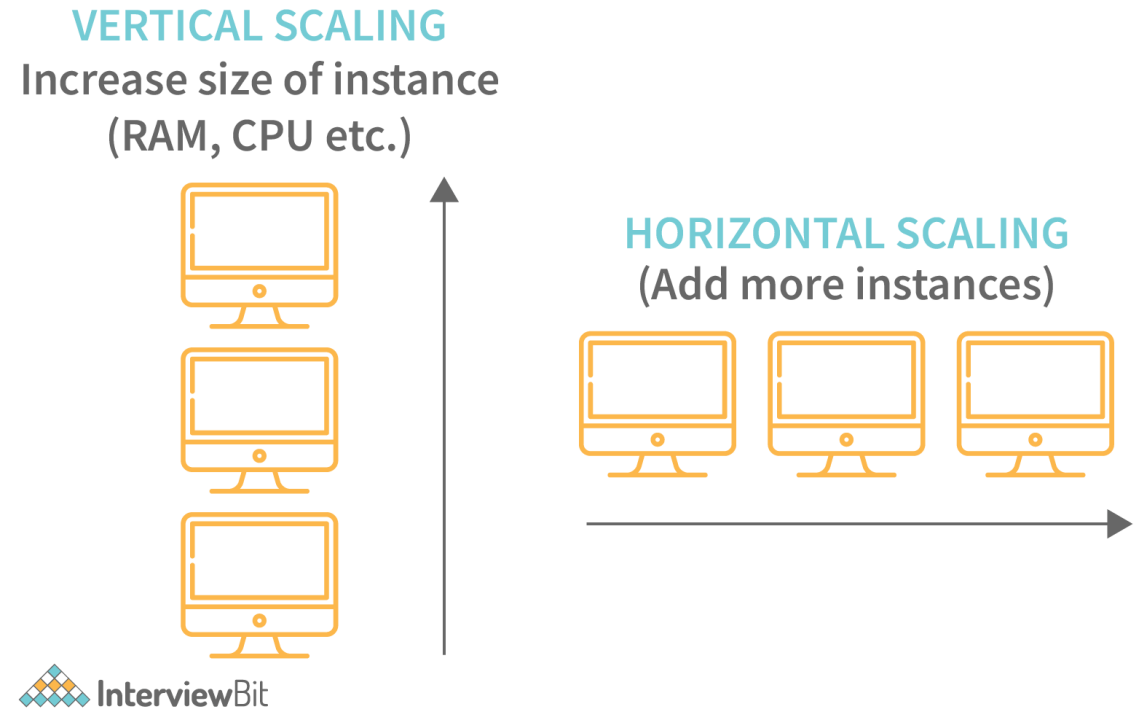
ByteByteGo.com



Integrarea sistemelor – DevOps

Problemă: dezvoltarea aplicațiilor de mare capacitate – scalabilitate

- Scalare verticală
- Scalare orizontală



Integrarea sistemelor – DevOps++

DevOps

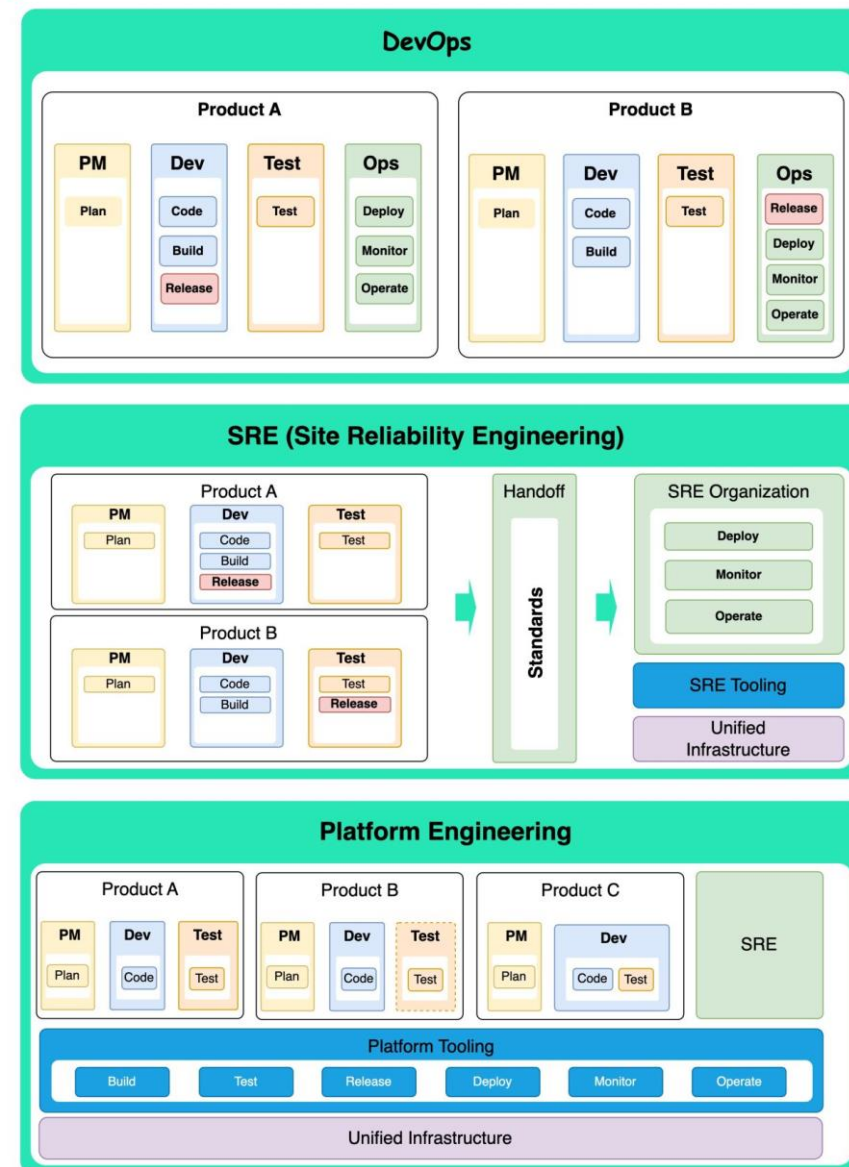
Concept introdus pentru a sincroniza mai bine activitatea de dezvoltare software cu partea operațională

+ SRE (Site Reliability Engineering)

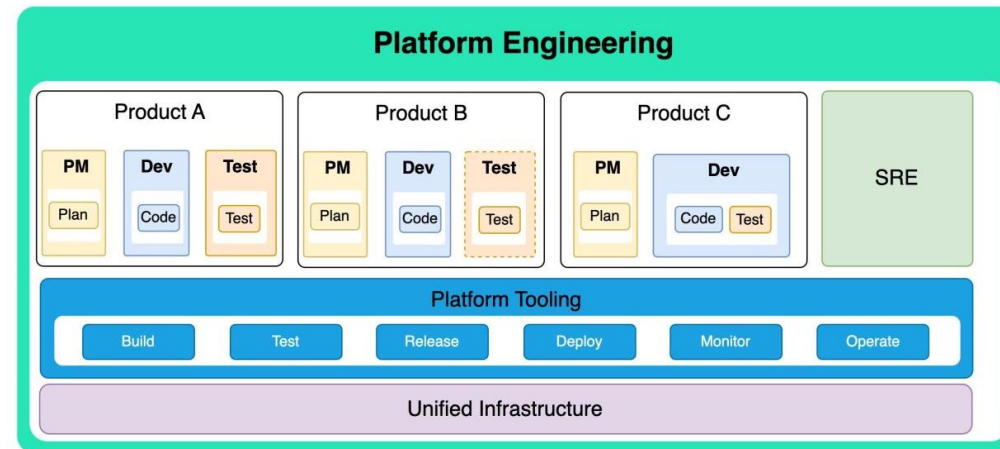
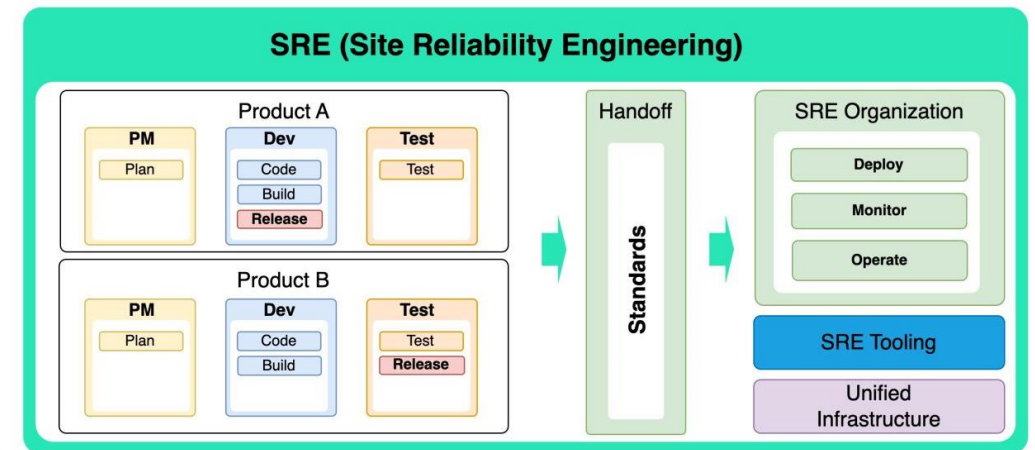
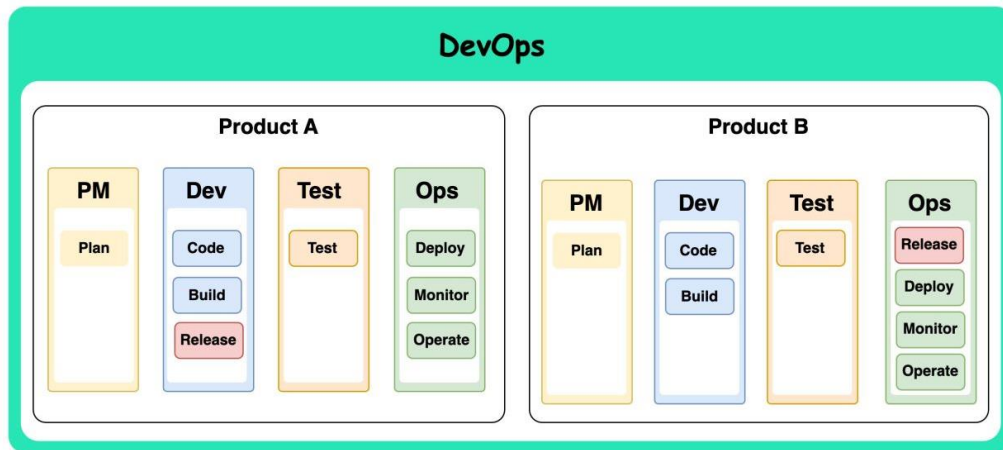
Domeniu dezvoltat de Google pentru a adresa provocările operaționale în sisteme complexe de mare capacitate

++ Platform Engineering

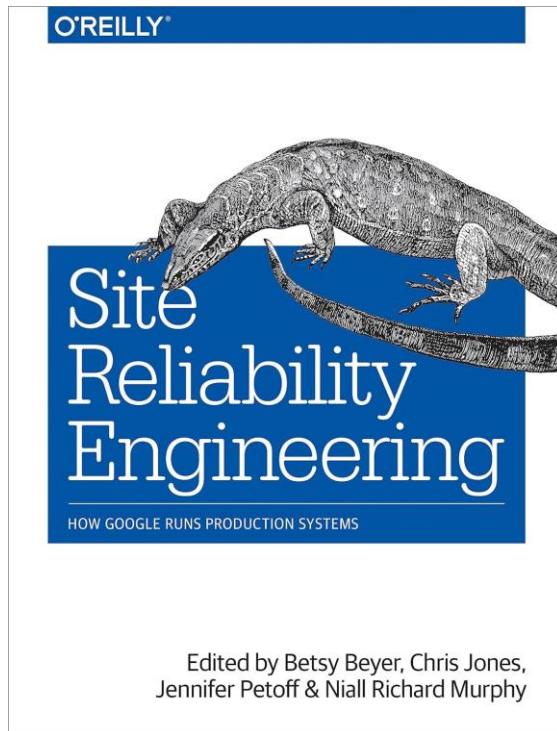
Extensie a SRE în contextul platformelor



Integrarea sistemelor – DevOps++



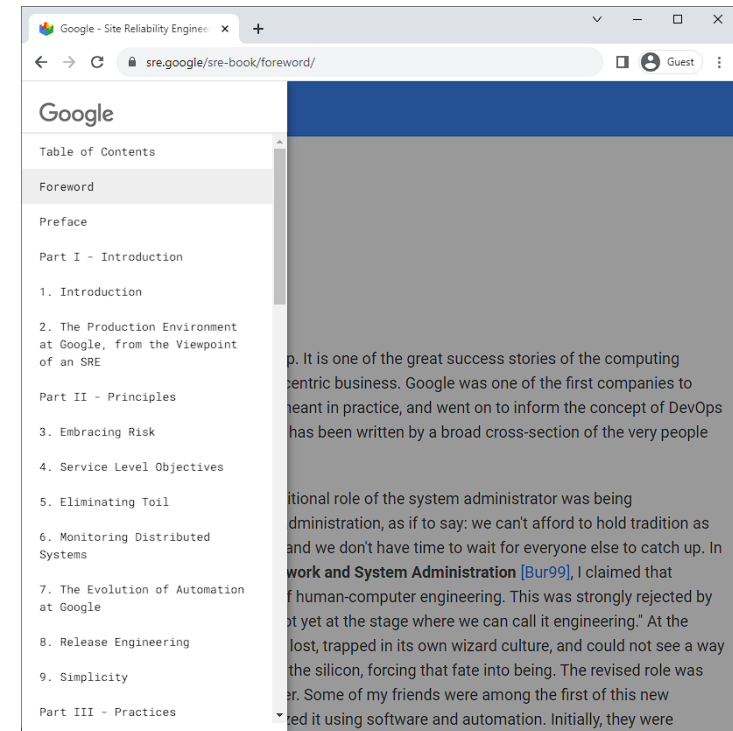
Integrarea sistemelor – DevOps++



This book is a series of essays written by members and alumni of Google's Site Reliability Engineering organization. [...]
— The Editors.



J. Petoff, Site Reliability Engineering: How Google Runs Production Systems 1st Edition, O'Reilly



Google, Site Reliability Engineering [online]
<https://sre.google/sre-book/table-of-contents/>

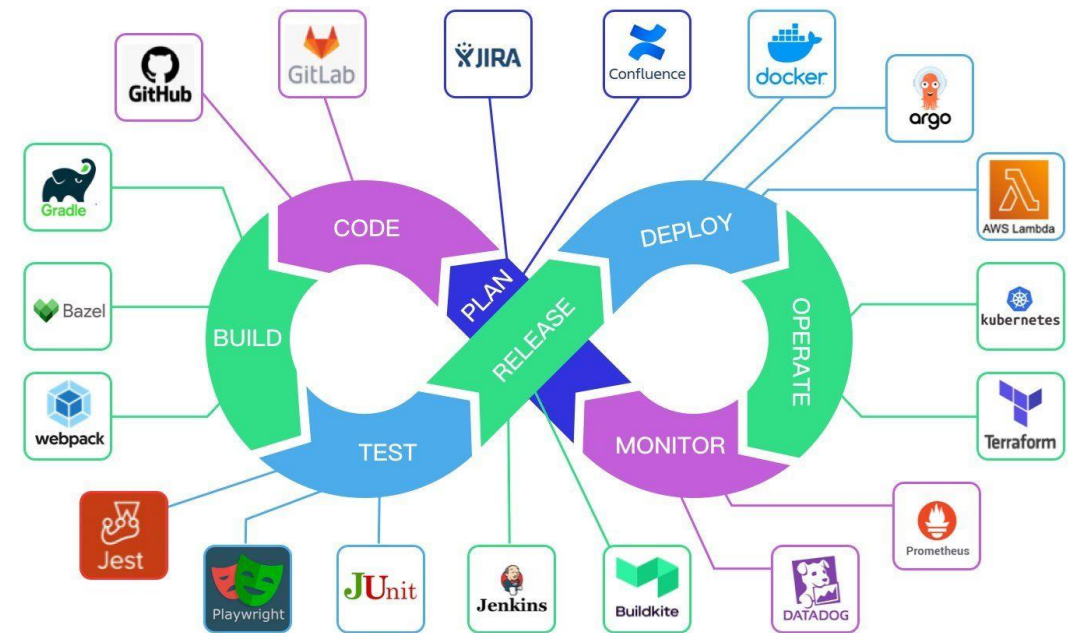
Integrarea sistemelor – CI/CD

Integrare continuă/Distribuție continuă (CI/CD)



- Ciclul de viață al produsului
- Nu este doar dezvoltare software și apoi punere în funcțiune ...
- ... Este un proces iterativ continuu pe durata existenței proiectului

CI/CD Pipeline Explained in 5 Minutes



Integrarea sistemelor – CI/CD

Integrare continuă/Distribuție continuă (CI/CD) – Etape

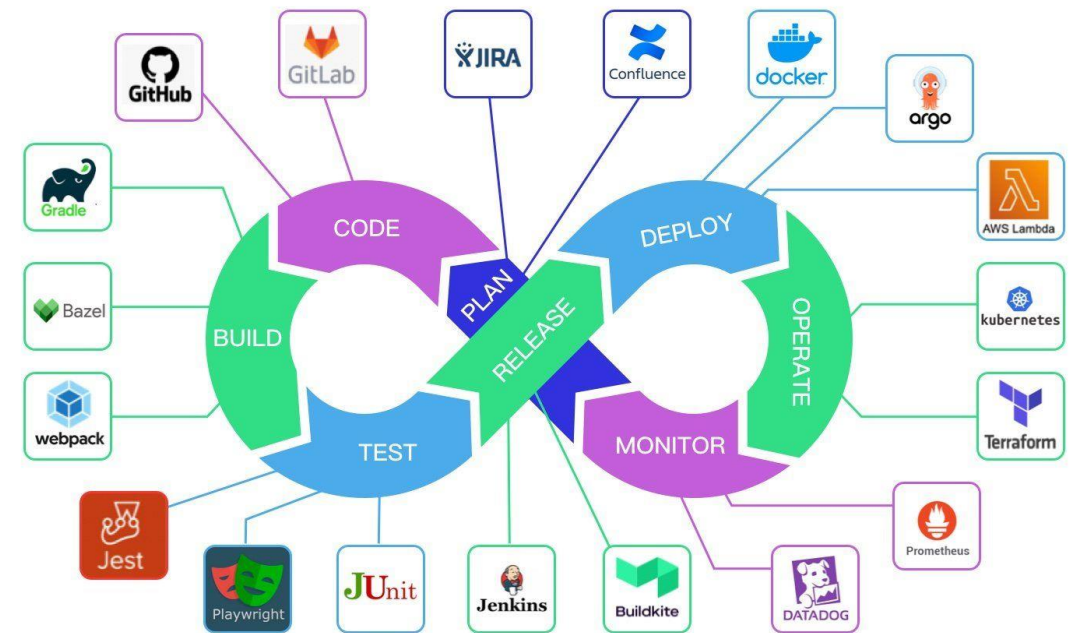
PROIECTARE



PRODUCȚIE

- **Planificare** – JIRA, Confluence
- **Programare** – GitHub, Bitbucket
- **Integrare** – Gradle, Webpack
- **Testare** – JUnit, JMeter
- **Lansare** - Jenkins
- **Distribuție** – Docker
- **Operare** – Kubernetes
- **Monitorizare** – Prometheus

CI/CD Pipeline Explained in 5 Minutes



Sursa: ByteByteGo, 2023

Întrebări?



Bibliografie

- [M. Widjaja, What is IT Architecture & Types of Architectures, IT Architecture, 2020](#)
- [D. Patel, Software Architecture – Five Common Design Principles, DEV.to, 2020](#)
- [ByteByteGo Newsletter, 2023](#)
- [Refactoring Guru, Design Patterns, 2023](#)
- [Powered by AI and the LinkedIn community, How do you approach software design and architecture to ensure quality, scalability, and maintainability?, 2023](#)
- [Powered by AI and the LinkedIn community, What are some common debugging techniques and when to use them?, 2023](#)

