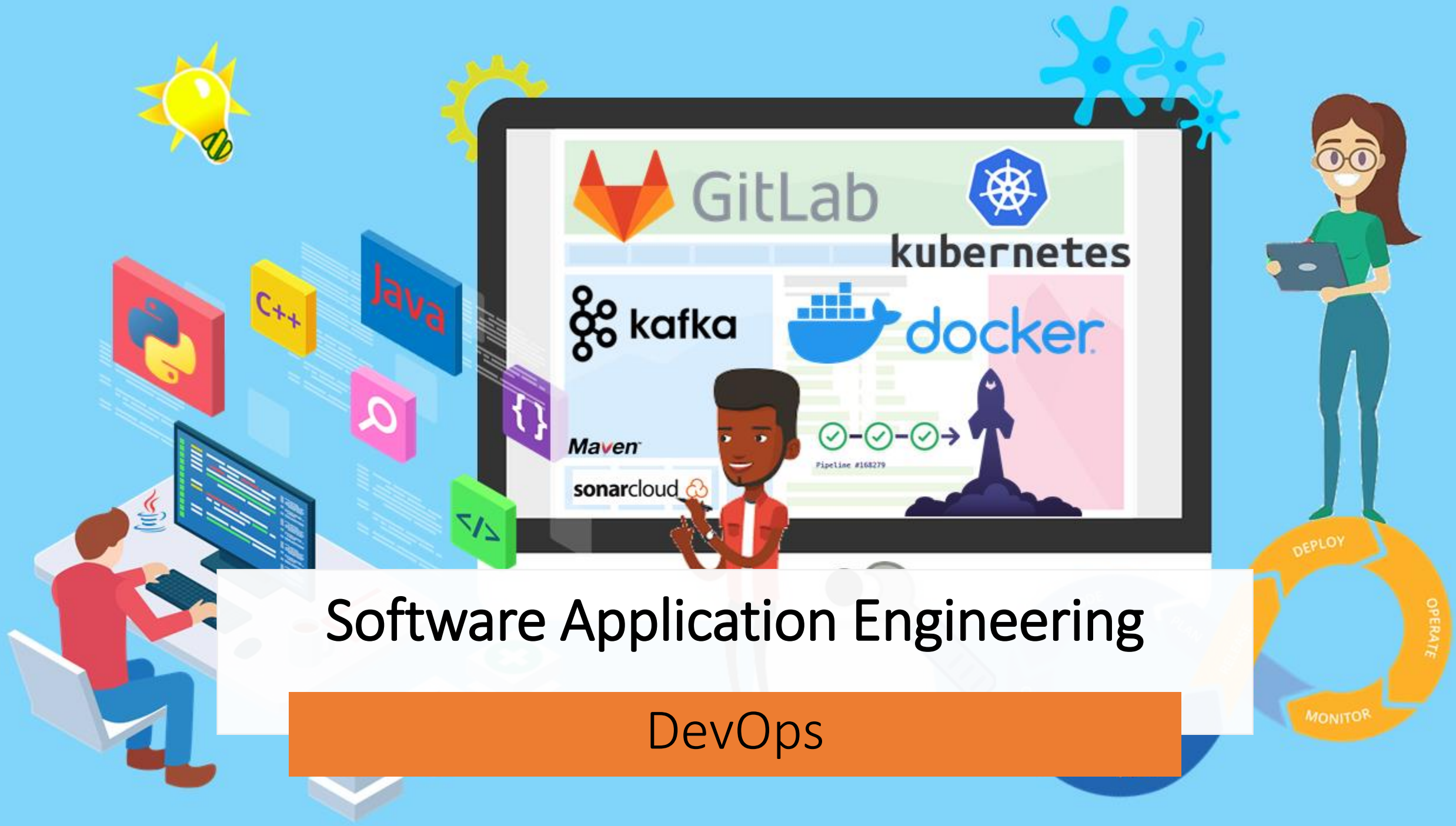
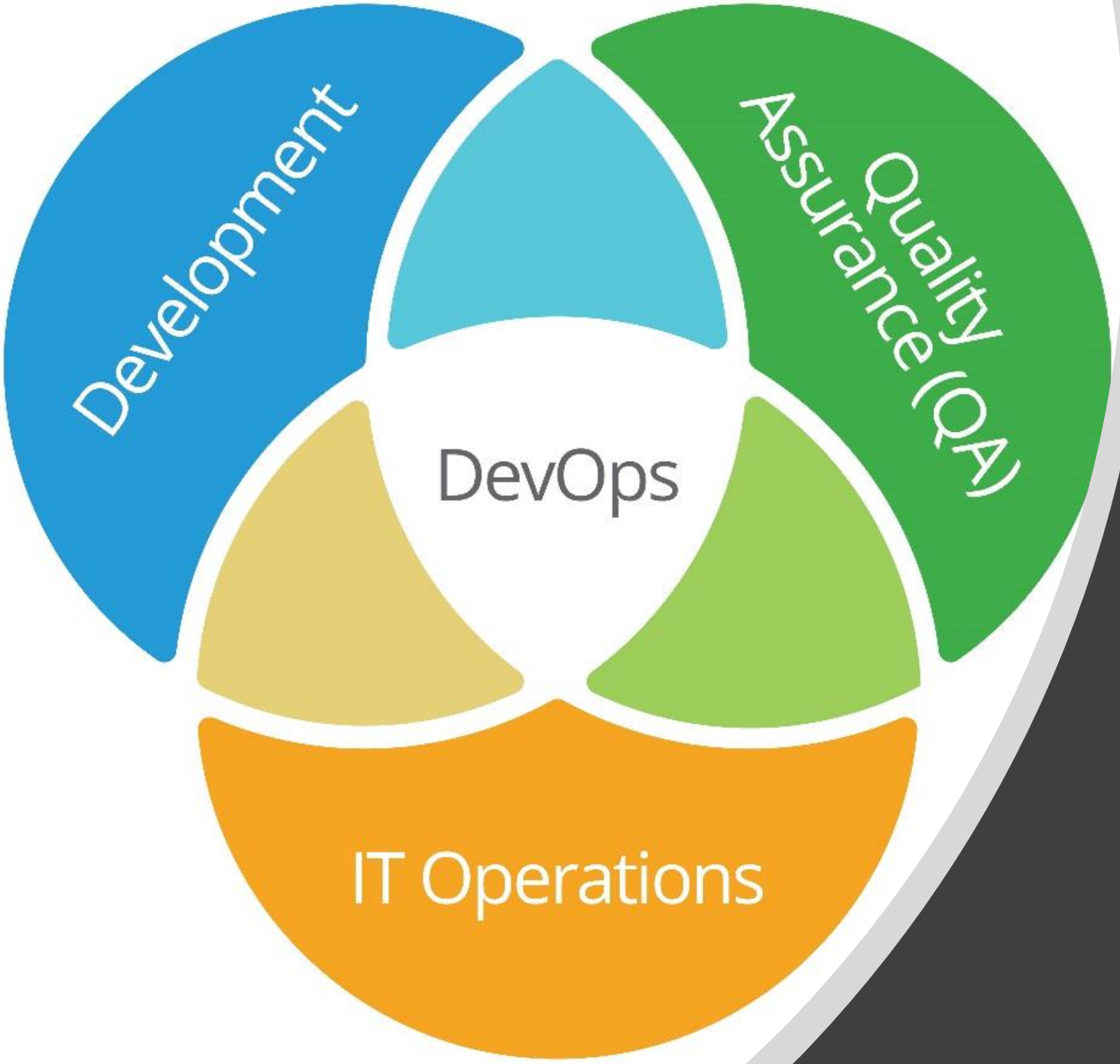


Software Application Engineering

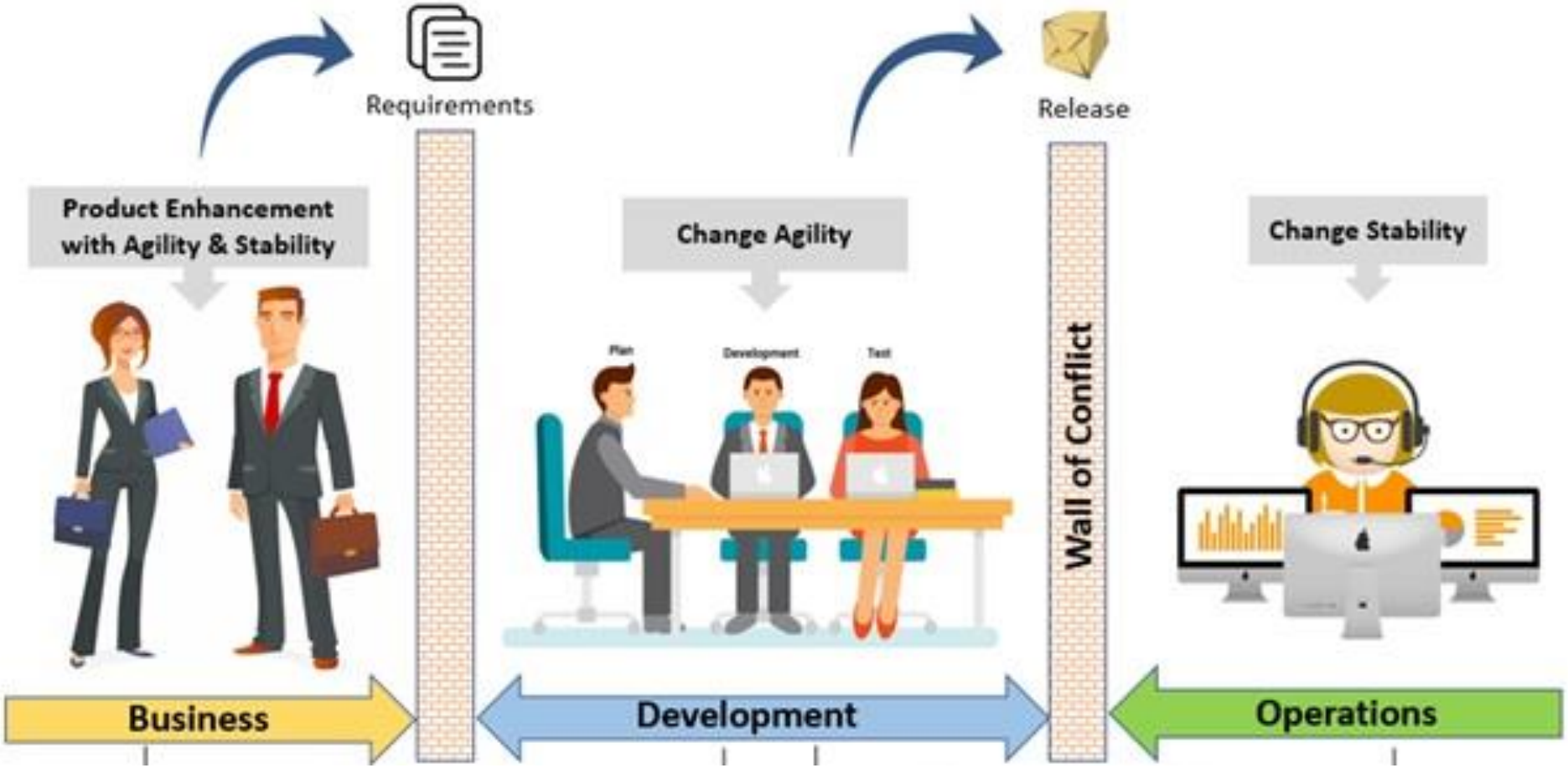
DevOps



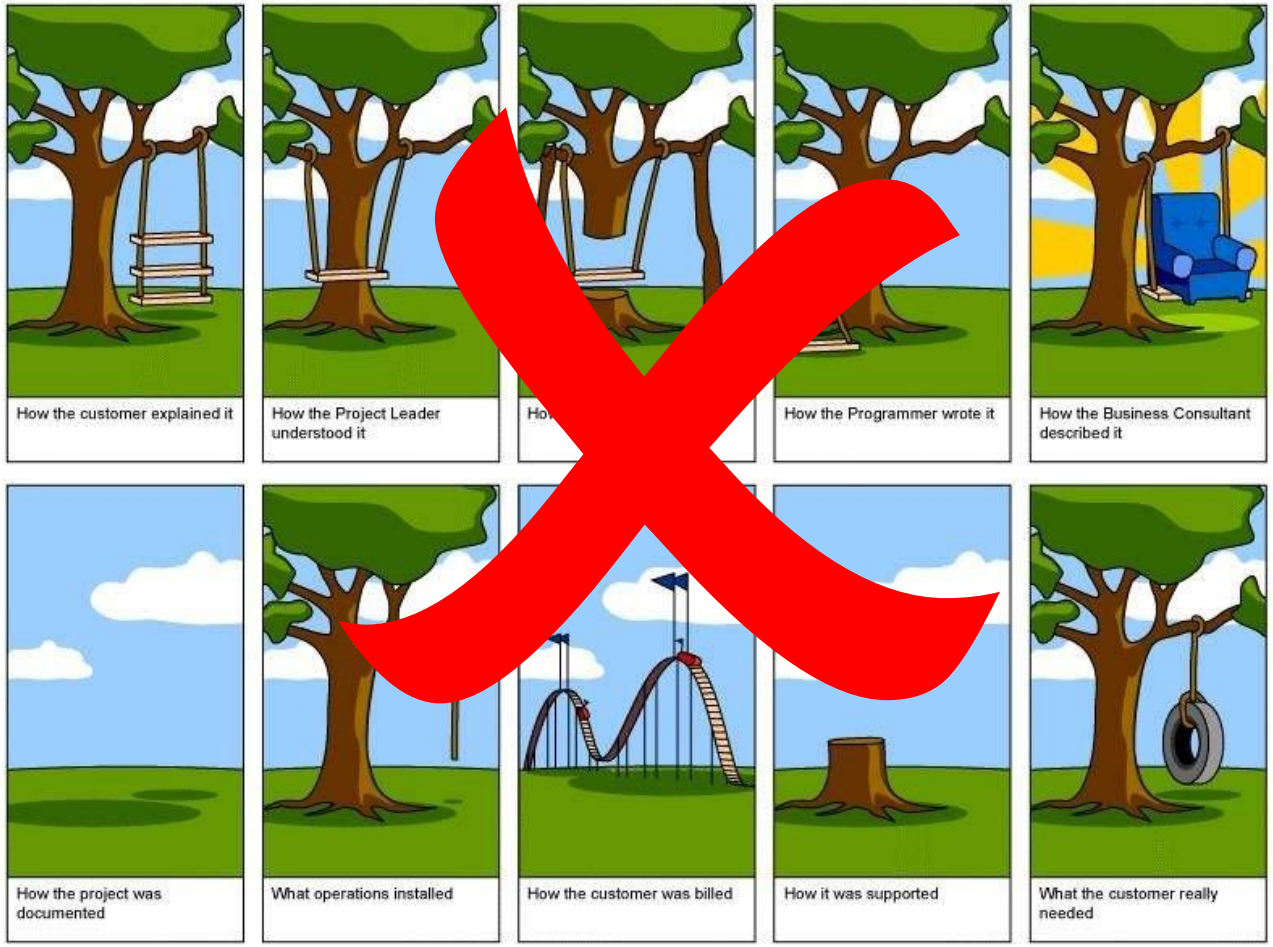
devops

Credits: cours DevOps MS-ASI de CSExed d'O. Truong

Software development walls of conflict



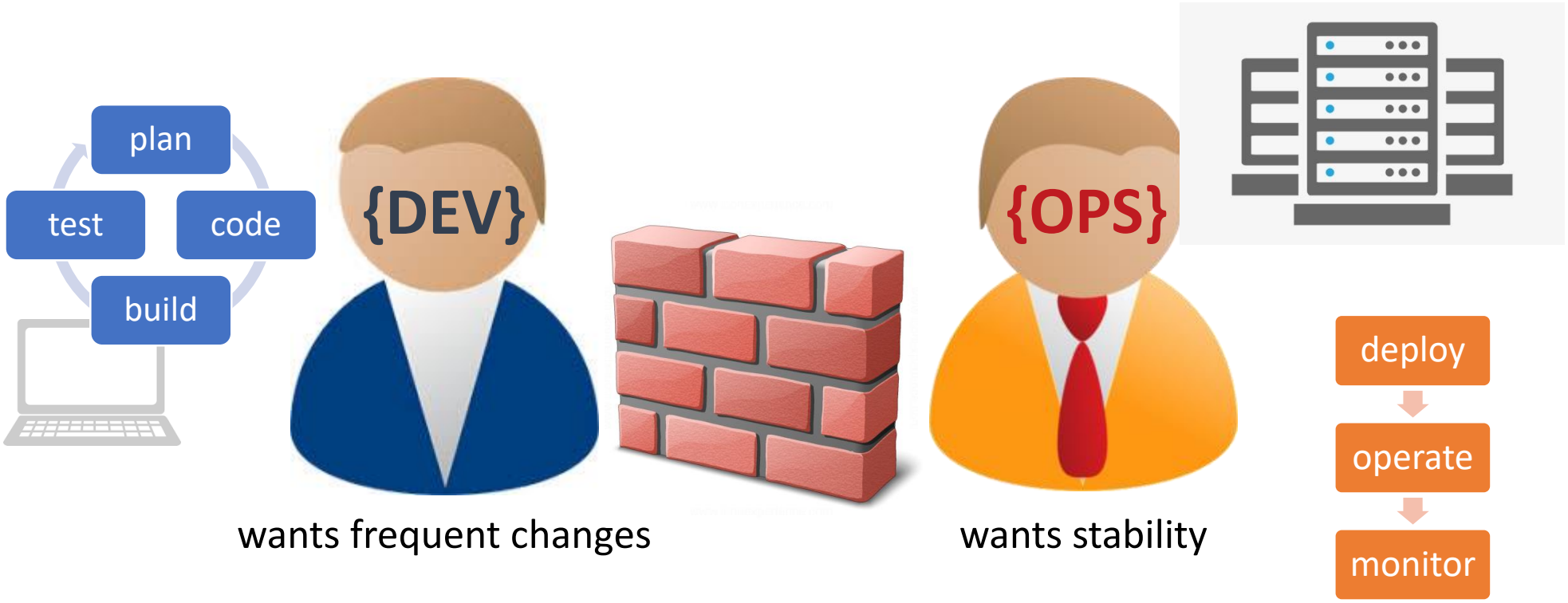
The wall of “changing requirements”



shut down by agile methodology

https://dc-consultants.net/transformation-agile-5-bonnes-raisons-faire-appel-coaching-scrum-kanban

Second wall, between Dev and Ops



wants frequent changes

software

wants stability

hardware

same objective: please the client!

Consequences

- **Long time** between an idea of a functionality and its effective deployment in production
- **Push to production is risky** because a lot of things are changed at once and this operation is not “business as usual” (“big bang”)

DevOps

https://opensenselabs.com/blog/articles/automation-devops

DevOps: definitions

Ensemble de pratiques qui visent à réduire le Time to Market et à améliorer la qualité des produits logiciels, en réinventant la coopération entre DEV et OPS.

DevOps, c'est un modèle d'organisation, une culture, un assemblage de processus, d'outils et de patterns d'architecture.

Octo

DevOps represents a change in IT culture, focusing on rapid IT service delivery through the adoption of agile, lean practices in the context of a system-oriented approach.

DevOps emphasizes people (and culture), and it seeks to improve collaboration between operations and development teams. DevOps implementations utilize technology — especially automation tools that can leverage an increasingly programmable and dynamic infrastructure from a life cycle perspective.

Gartner

DevOps is the union of people, process, and technology to continually provide value to customers.

Microsoft

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity.

Amazon

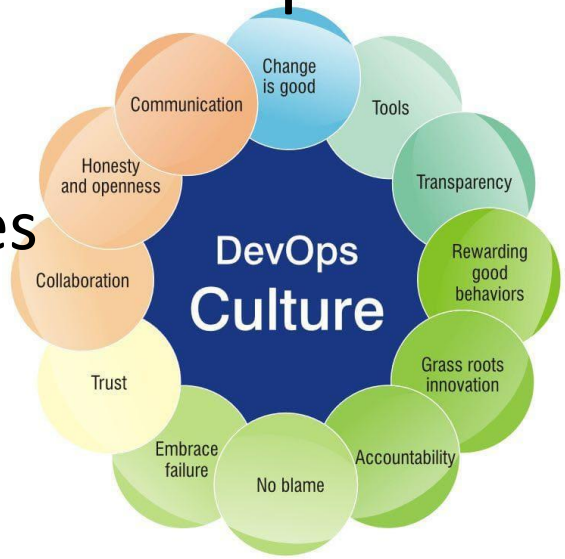
DevOps is an organizational and cultural movement that aims to increase software delivery velocity, improve service reliability, and build shared ownership among software stakeholders.

Google

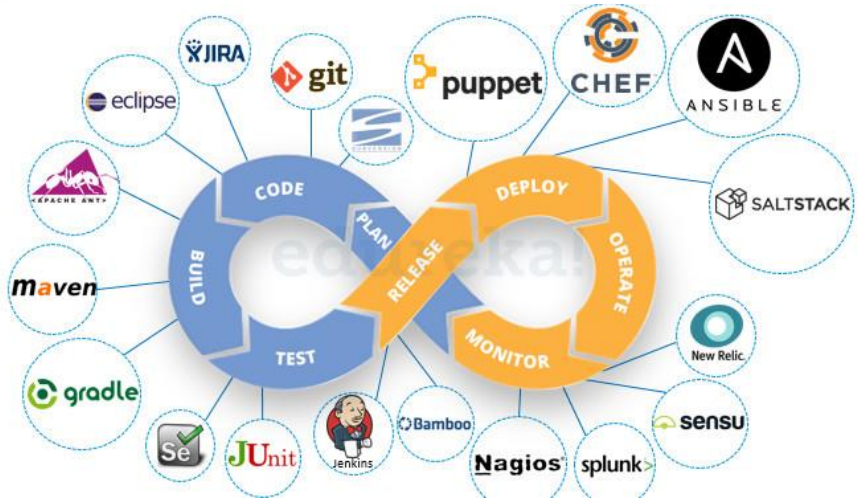
- DevOps is the outcome of applying the most trusted principles from the domain of physical manufacturing and leadership to the IT value stream.
- DevOps relies on bodies of knowledge from
  - Lean,
  - Theory of Constraints,
  - the Toyota Production System,
  - resilience engineering,
  - learning organizations,
  - safety culture,
  - high-trust management cultures,
  - organizational change management,
  - …

Definitions emphasize 3 aspects

- DevOps is a culture, it involves human beings



- DevOps relies on tools and practices



- DevOps focuses on accelerating the delivery of value

https://itsvit.com/blog/devops-culture-huge-step-mankind/ https://eduinpro.com/blog/top-devops-tools-in-the-digital-market/

https://www.tradecarsspain.com/news/2014/05/28/average-used-car-prices-rising-murcia

3 Principles underpinning DevOps

- *A development has no value until it reaches production.*
  - aim at the performance of the entire system, as opposed to the performance of a specific silo of work or department
  - always seek to increase flow (reduce time between initial idea and push to production of the new functionality)
- *To measure is to know, if you can't measure it you can't improve it.*
  - monitor
  - provide feedback as soon as possible, as usable as possible
- Create a culture that fosters experimenting and learning from failure
  - *Do painful things more frequently so you can make it less painful…*

Benefits

- 46x more frequent code deployments
- 440x faster lead time from commit to deploy
- 170x faster mean time to recover from downtime
- 5x lower change failure rates

https://www.27global.com/in-praise-of-process-how-continuous-delivery-accelerates-devops/

- Better work experience
  - See how your work fits in the big picture
  - Tedious tasks are automated
  - Learn and improve your skills continuously
  - No “big bang” day stress

  → less burn-out, less turn-over

https://www.idgconnect.com/analysis-review/1505101/happy-developers-security-conscious

Infrastructure on Demand

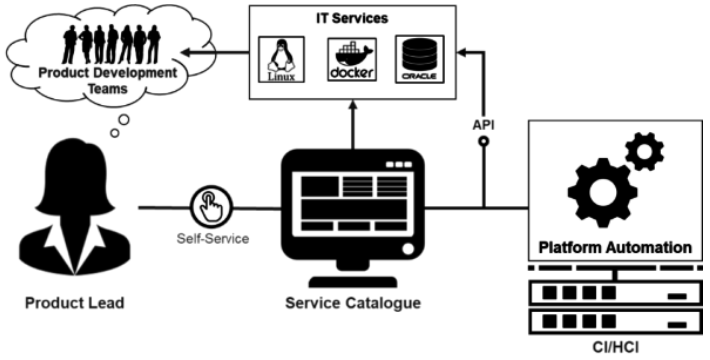
- Provide environments for developing and testing
- Identical to the production environments
- In self-service

https://in.pinterest.com/pin/476959416788550864/

Infrastructure as Code (IaC)

- Formal declaration of the infrastructure (and its configuration!), manual operations (ssh, kubectl…) are forbidden
- Often cloud-based (remember to shut down!)

```
{
   "Resources":{
      "WebServer":{
         "Type":"AWS::EC2::Instance",
         "Properties":{
            "ImageId":"ami-6869aa05",
            "InstanceType": "t2.micro"
            "KeyName": "mykey"
         }
      }
   }
}
```



https://infocus.delltechnologies.com/joe_pearlman/what-is-infrastructure-as-code-iac-and-why-it-is-the-future-of-cloud-automation/

https://searchitoperations.techtarget.com/definition/Infrastructure-as-Code-IAC

https://www.hashroot.com/infrastructure-as-code

IaC Tools

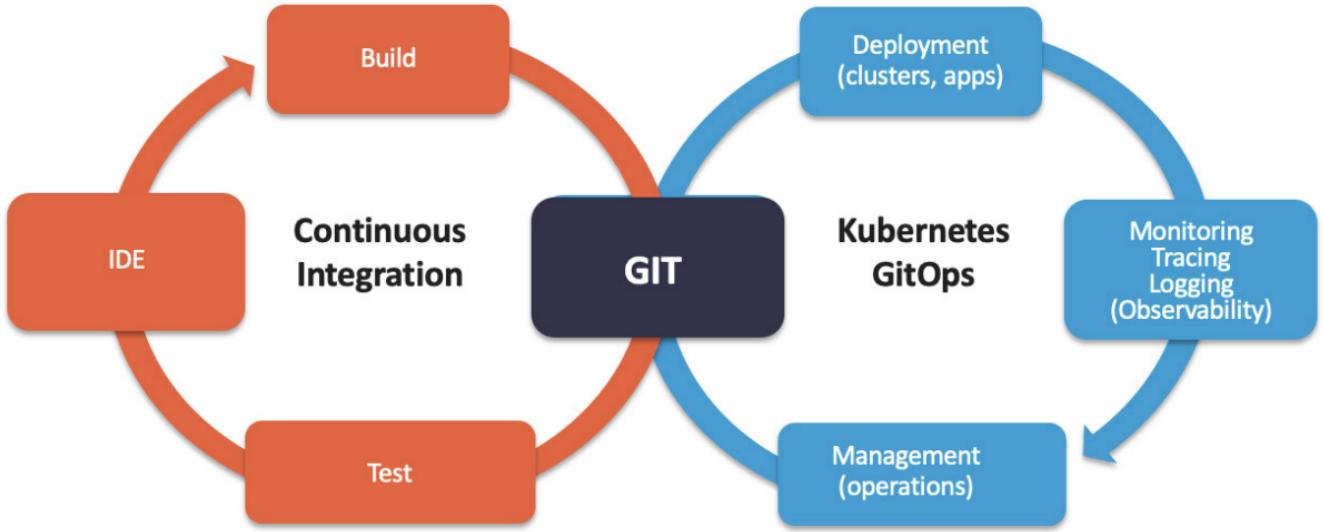
- cloud agnostic
  - \+ multi-cloud, migration
  - ex: Terraform, Ansible, Chef/Puppet, Kubernetes, …
- from cloud providers
  - \- specific → lock-in
  - \+ but additional services (versioning, rollback, resources export…)
  - ex:
    - Amazon Web Service: Cloud Formation
    - Microsoft Azure: Resource Manager
    - Google Cloud: Deployment Manager

IaC is a shared resource

- IaC is versioned in the same Git repository as the application code and the build and deployment scripts

  → “GitOps”

  - Git as the single source of truth of a system
  - Git as the single place where we’re operate (create, change and destroy) all environments (dev, test, prod…)



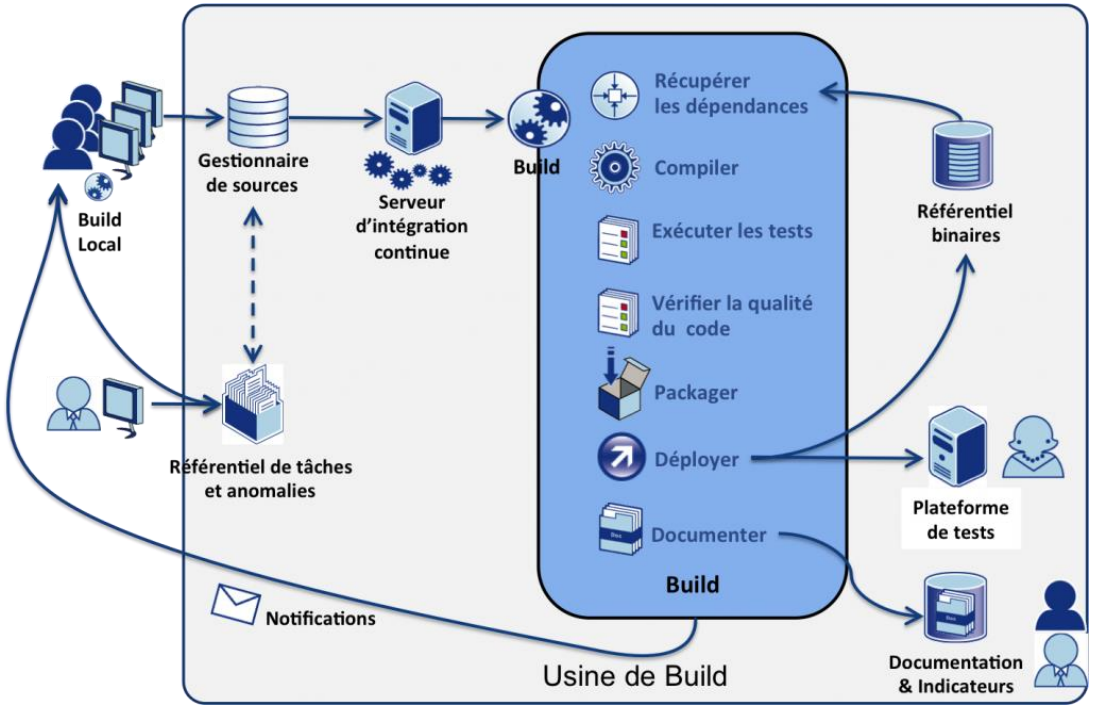
https://cloudbestpractices.net/weaveworks-gitops/

Team → Integration → Difficulties

> ***Team programming*** *isn't a divide and conquer problem. It is a divide, conquer, and* ***integrate*** *problem. The integration step is unpredictable, but can easily take more time than the original programming.* ***The longer you wait to integrate, the more it costs*** *and the more unpredictable the cost becomes.*
>
> Kent Beck, Extreme Programming Explained

Continuous Integration

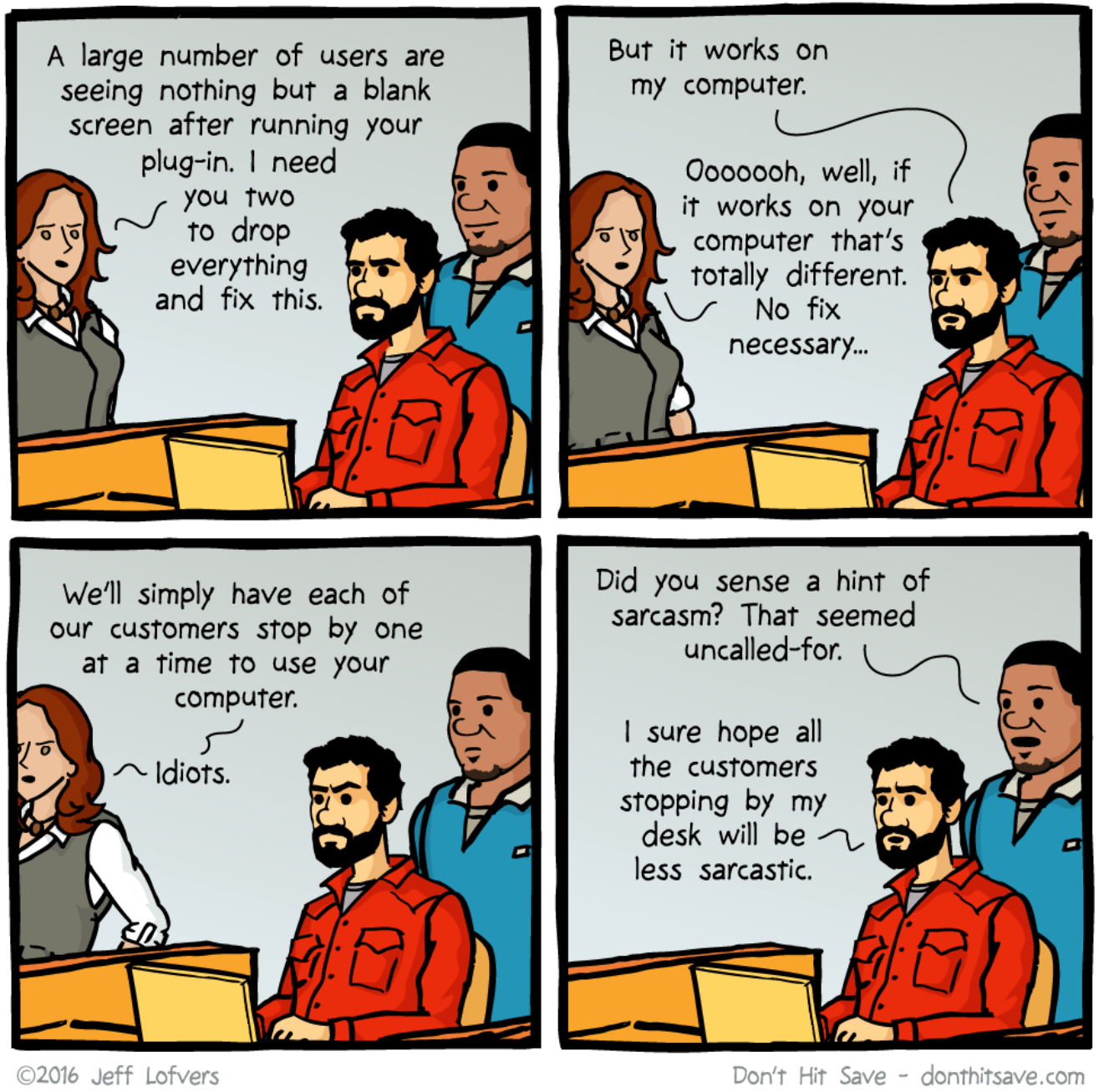
- Dev practice
- Dev integrate code as frequently as possible on a shared server
- Each change triggers a build and check of the project, with a feedback to the dev
- Usually: YAML file describes build requirements (java? Python?) + Docker image + workflow/steps to follow



https://blog.octo.com/vers-une-usine-de-developpement-2-0/

Benefits

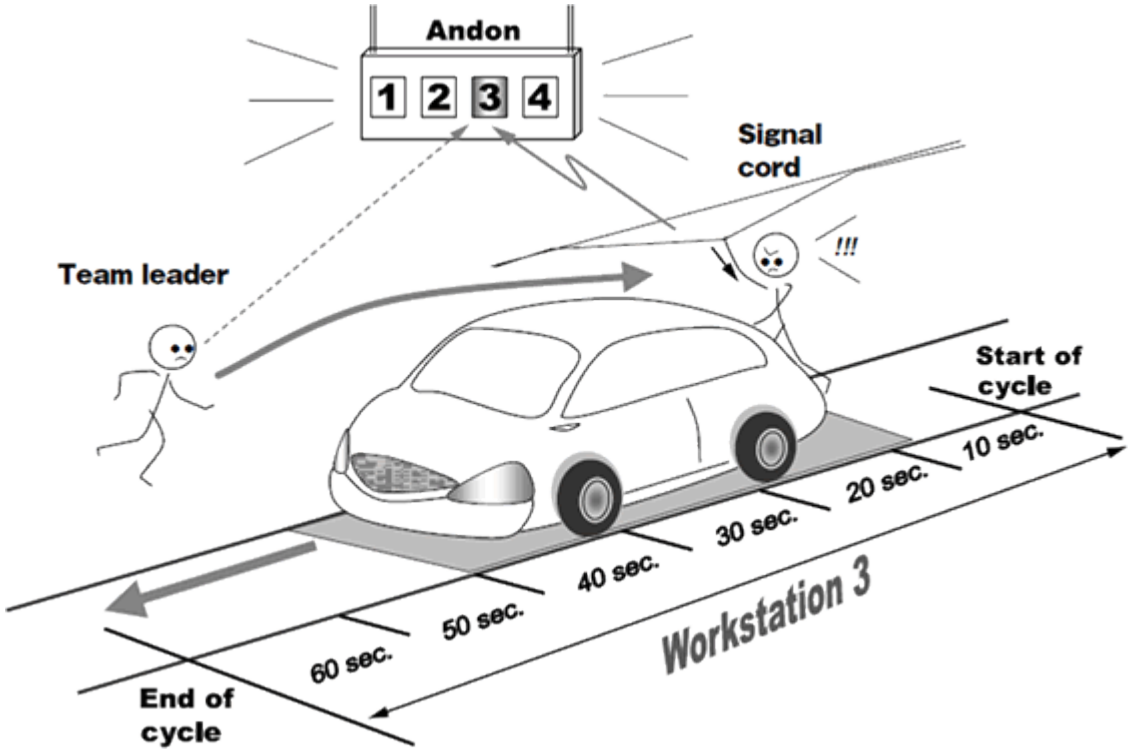
- Make sure all necessary code is present
- Limit the “it works on my machine” syndrome



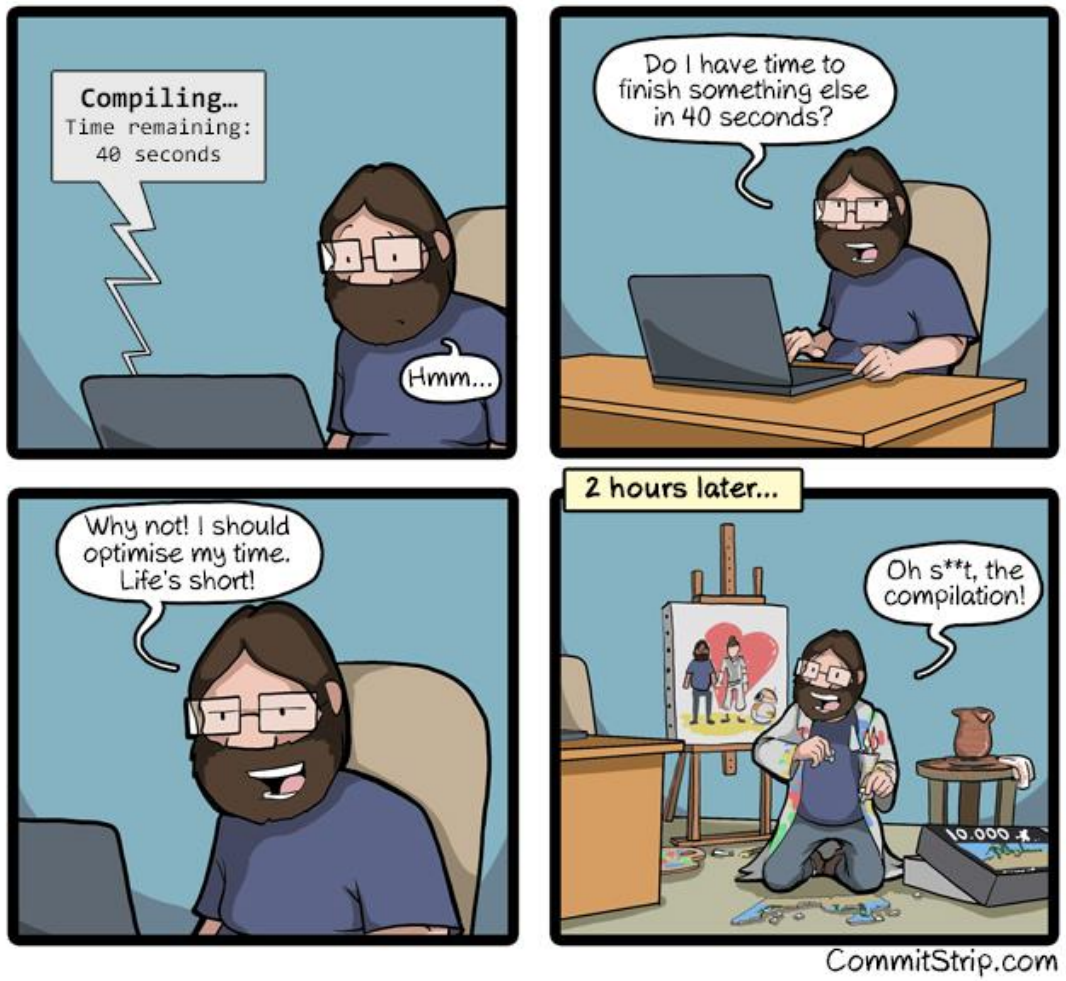
https://www.kevinwanke.com/why-you-should-never-use-the-phrase-but-it-works-on-my-machine/

Requirements for CI

- Automated build
- Automated tests
- Visible results (is the build broken?)
- Process must be fast

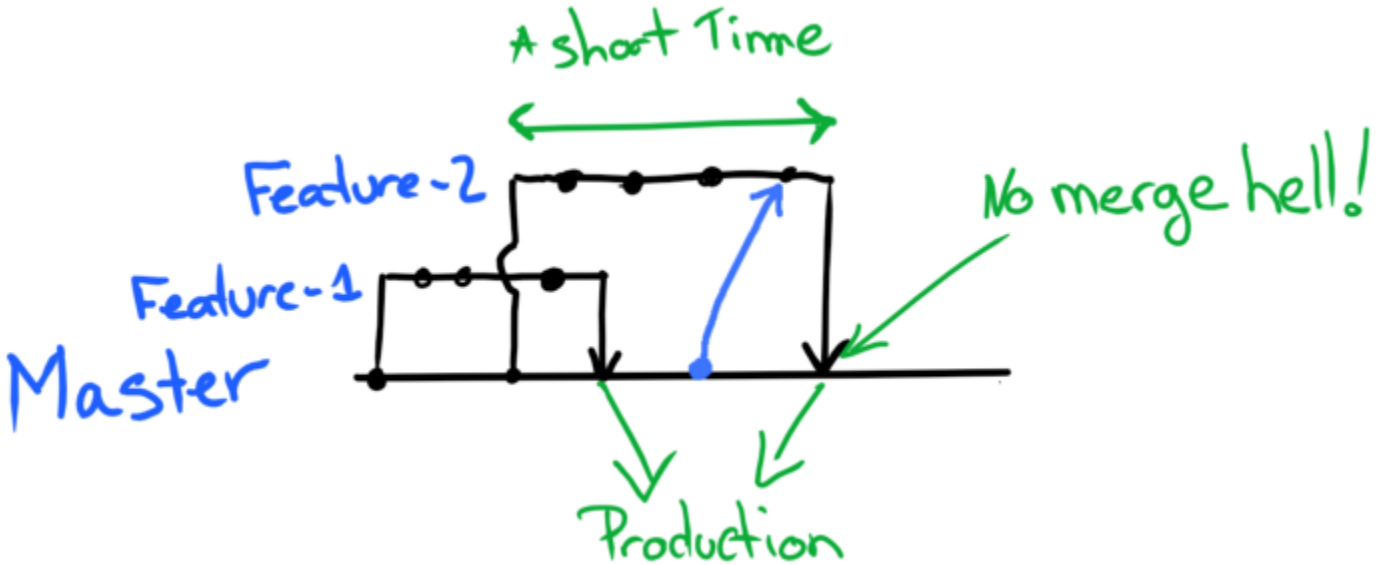


https://www.lean.org/shook/displayobject.cfm?o=1321



Continuous integration → integrate very frequently!

- Feature branches are merged to the master within hours, 1 day max (“trunk based development”)



https://medium.com/safetyculture-engineering/trunks-are-not-just-for-trees-from-git-flow-to-trunk-based-development-949d580697ef

Some popular CI Tools

- On premises

Jenkins

- Cloud

Bitbucket Pipelines

https://linux-notes.org/ustanovka-gitlab-runner-a-v-unix-linux/

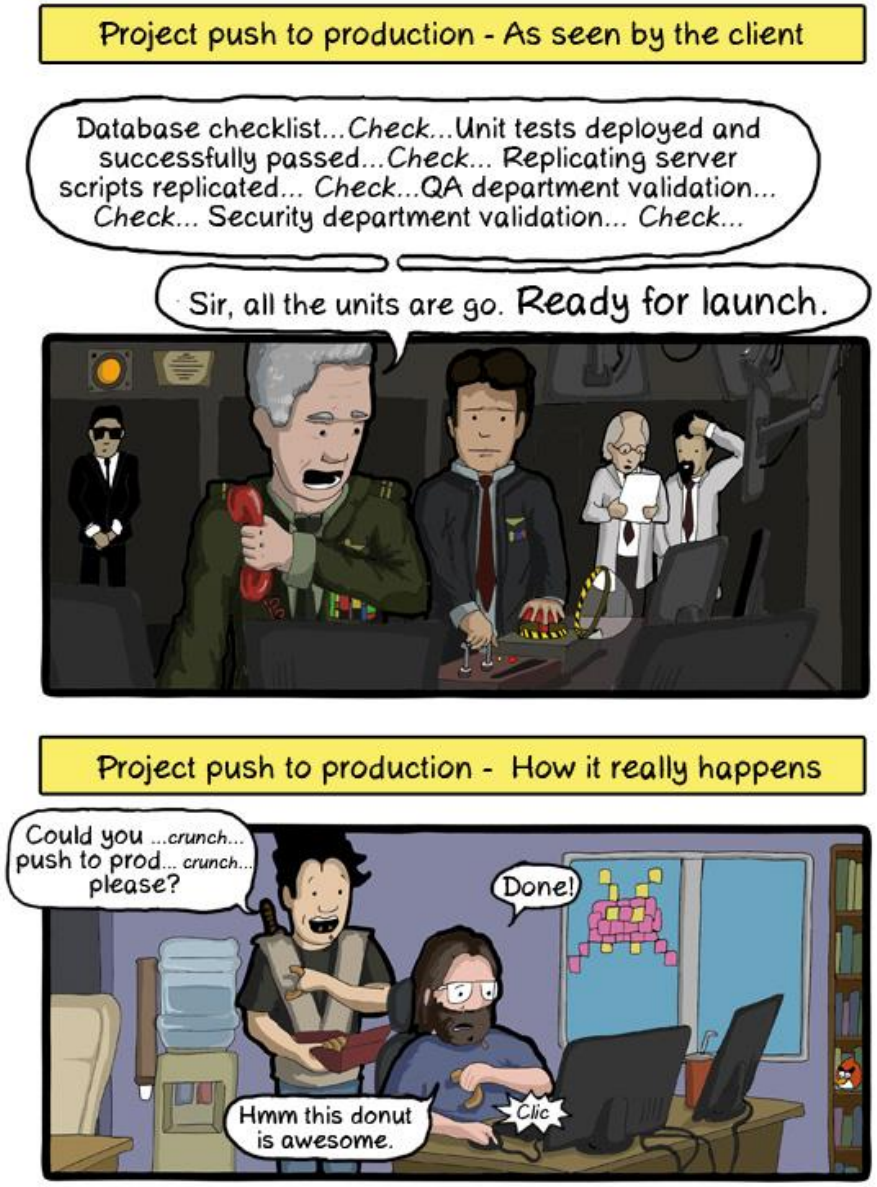
Continuous Delivery

- Development process making sure the software can be deploy in a given environment (production for instance) at any time

→

  - ability to deploy software is more important than new functionalities
  - deployment is automated (one-click, self-service)

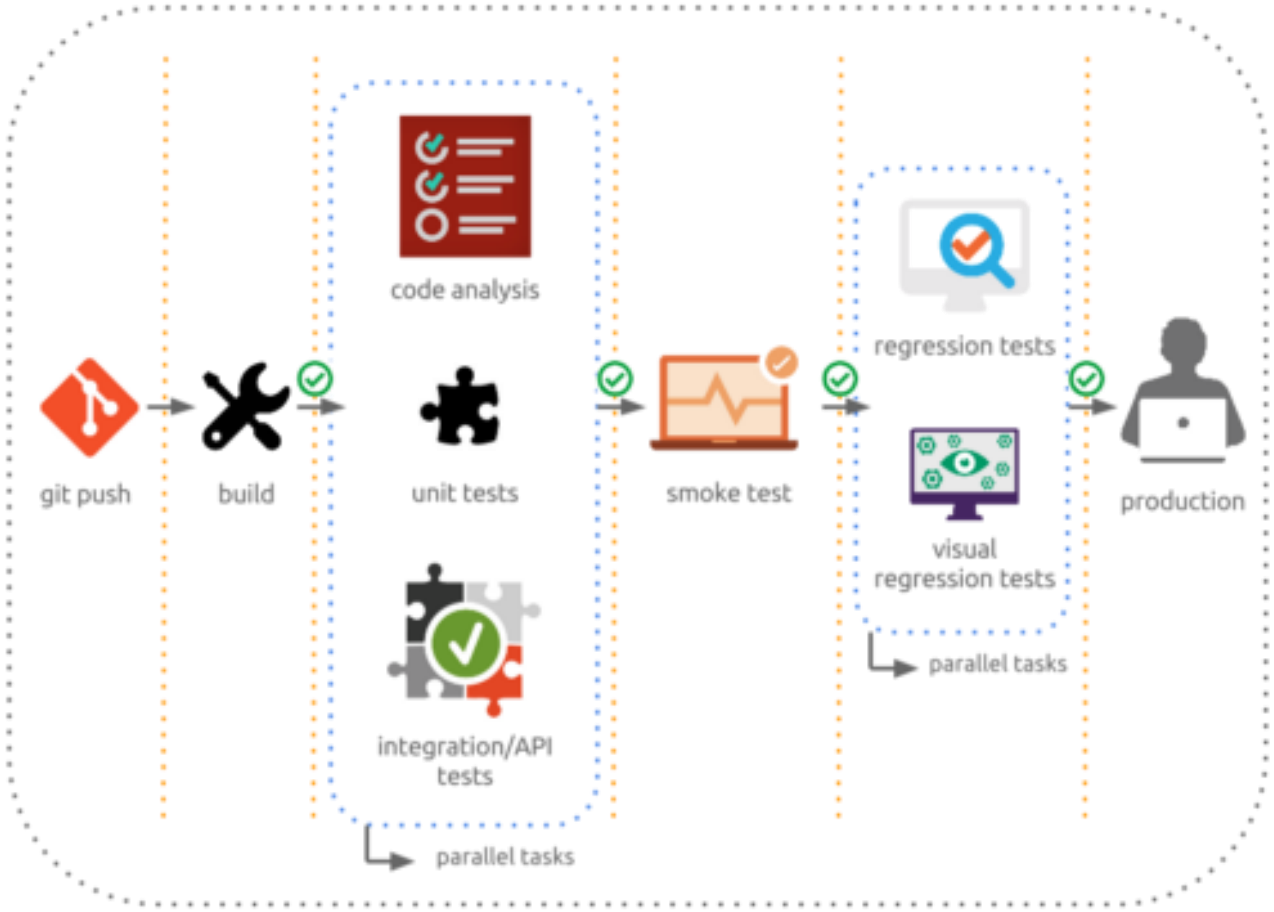
- Pushing to prod is a non-event



https://www.commitstrip.com/en/2012/07/20/mise-en-prod/?

CD Pipeline

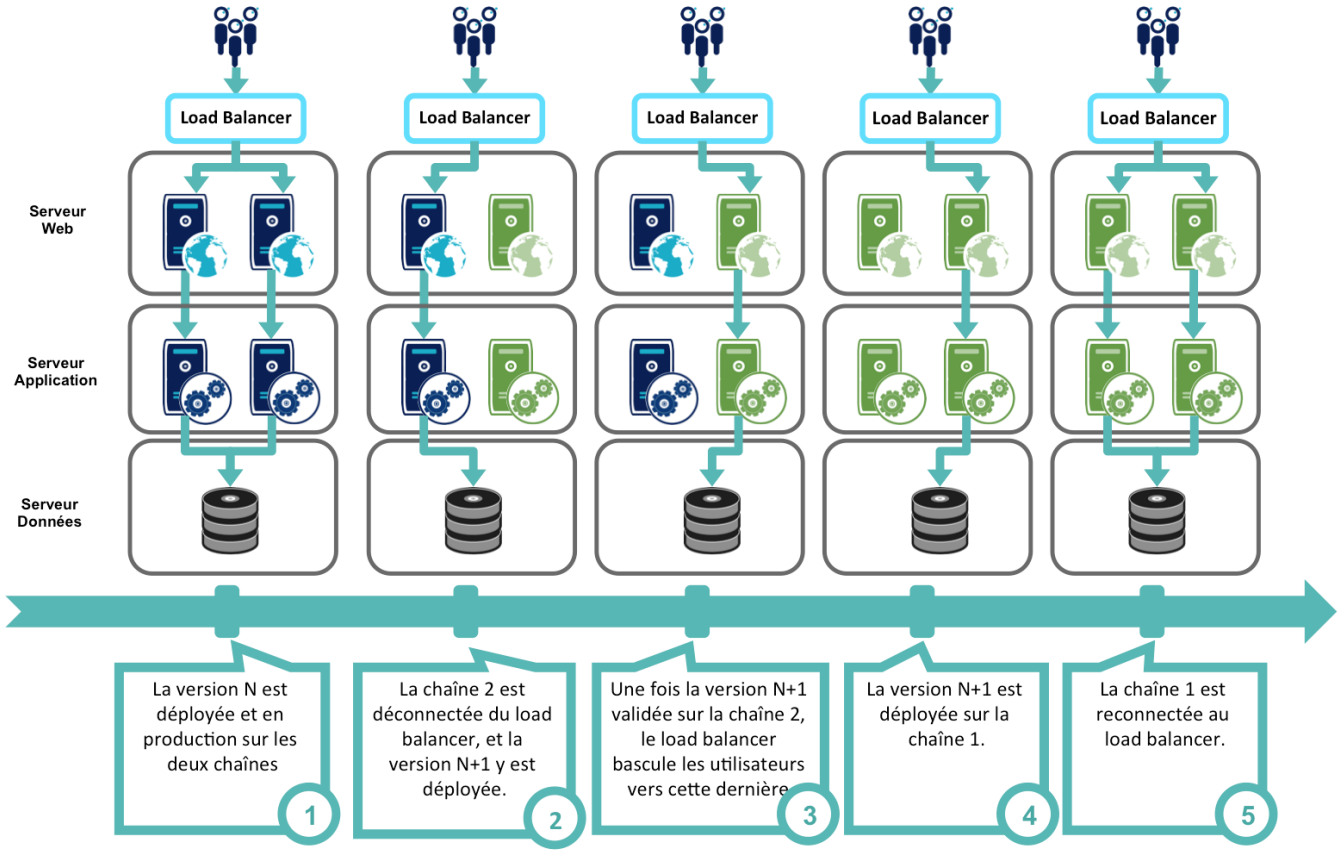
- Benefits:
  - Avoid doomed tasks (don’t move downstream in case of errors upstream)
  - Parallelize tasks
  - [Visualize tasks towards production]
- Requirements:
  - Similar builds for the various environments
  - Environments should be as similar as possible (12-factor #10)



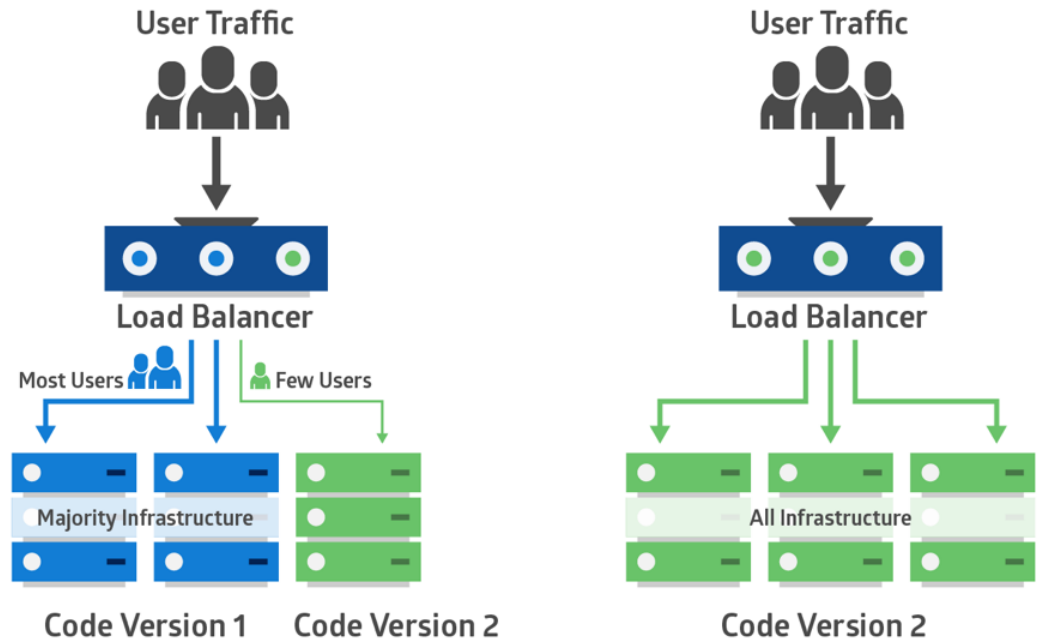
https://medium.com/the-making-of-whereby/what-is-a-deployment-pipeline-and-how-it-helps-software-development-teams-6cb29917ceea

Seamless deployment

- Blue/green deployment



- Canary release



https://www.ncm.org.uk/learning/learning-resources/image-bank

https://thenewstack.io/win-win-deployment-strategies-for-modern-apps/
https://www.google.com/search?q=blue+green+deployment&client=ubuntu&hs=q7t&channel=fs&sxsrf=ALeKk02kPYx93EP5OOEooc18K6mQV3mcKQ:1598906677249&source=lnms&tbm=isch&sa=X&ved=2ahUKEwiD2JrEp8brAhVQ4YUKHVY5D3AQ_AUoAXoECBIQAw&biw=1920&bih=1086#imgrc=iQyPowVKxZrHfM

CD: Delivery vs Deployment?

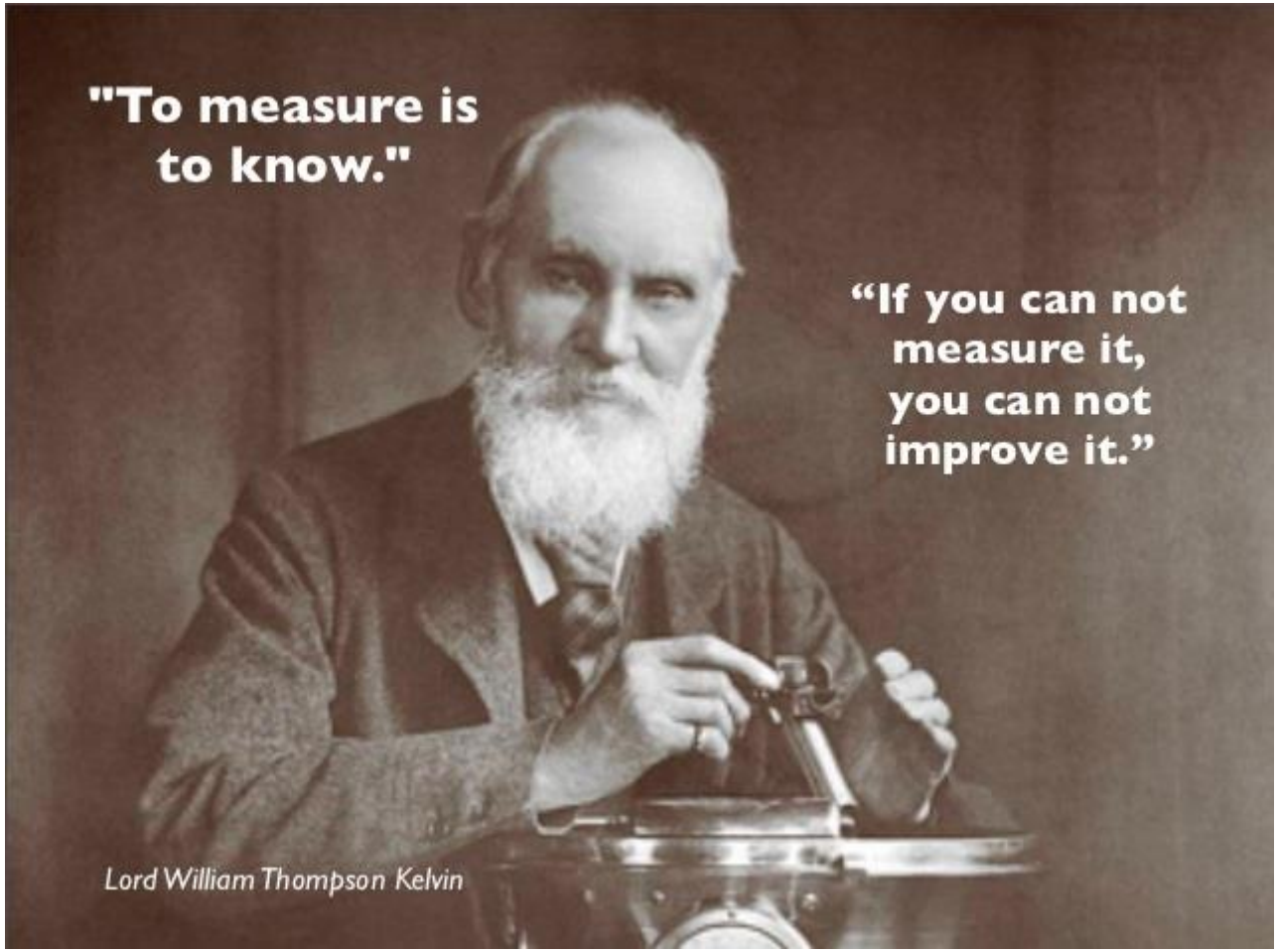
Continuous delivery

- To be able to deploy, but deciding not to do it
- (business model: selling new licence)

Continuous deployment

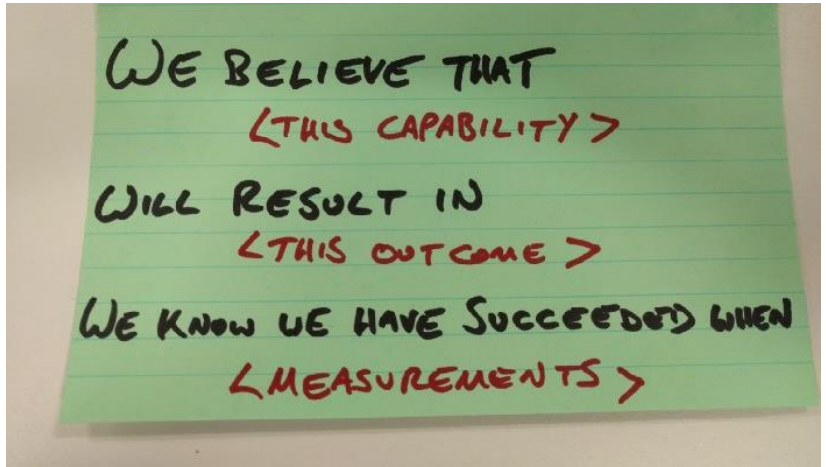
- Requires continuous delivery
- Each change is actually deployed in an environment (usually production)
- (business model: subscription)

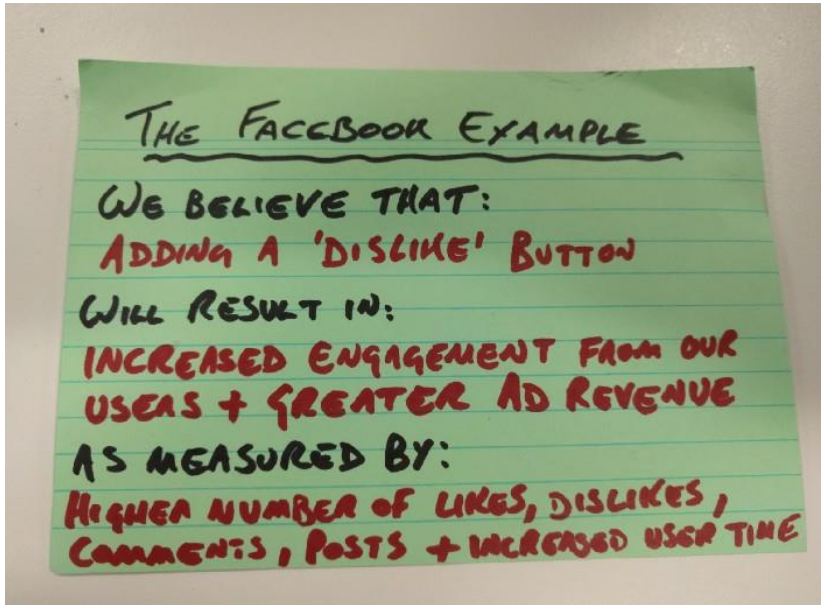
Monitoring


"To measure is to know."
“If you can not measure it, you can not improve it.”
Lord William Thompson Kelvin

Step 1: define measurements

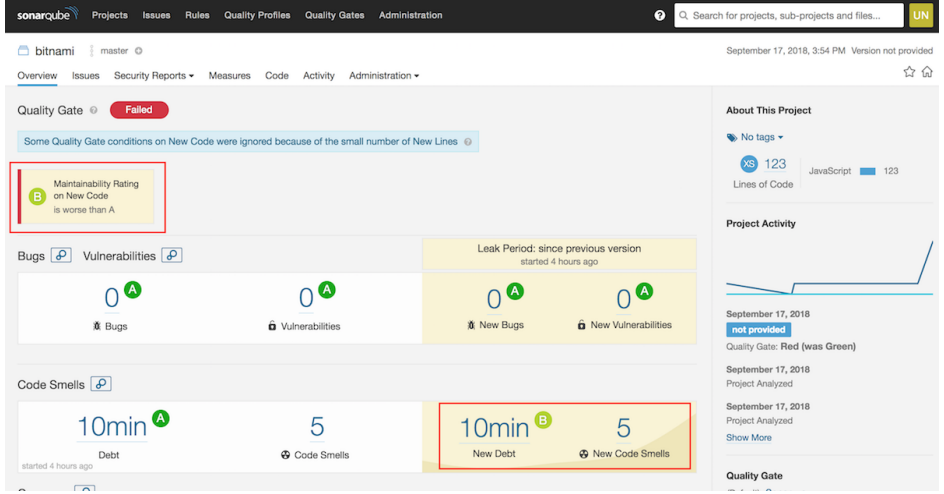
- What to measure?
  - memory consumption
  - network traffic
  - kubernetes cluster pod restarts
  - response times
  - number of messages in forum
  - mails to client support service
  - …
- After each problem, think about which metrics would have help anticipate or detect it
- “Hypothesis Driven Development”





https://medium.com/cambridge-agile-exchange/learning-is-fun-experiments-with-hypothesis-driven-development-e3b723951 6a5

Step 2: collect measurements



- Static code analysis tools

  Ex: SonarKube/SonarCloud

- a central system where to push logs + a visualization solution

  Ex: DataDog, Grafana, Splunk…

- definition of dev “Done” includes defining and collecting metrics

→ simple tools to encourage this practice: logstash, fluentd…

https://docs.bitnami.com/oci/how-to/analyze-projects-sonarqube/

Step 3: exploiting the logs, alerting

- Proactive monitoring:
  - Statistical tools: detect potential issue
    - Setup alert thresholds rather than requiring multiple graphs continuous monitoring

  - Anticipate load and up or down scale change needs

NB: ML approaches are gaining momentum

https://www.hattrickmedia.co.uk/website-monitoring-tools/

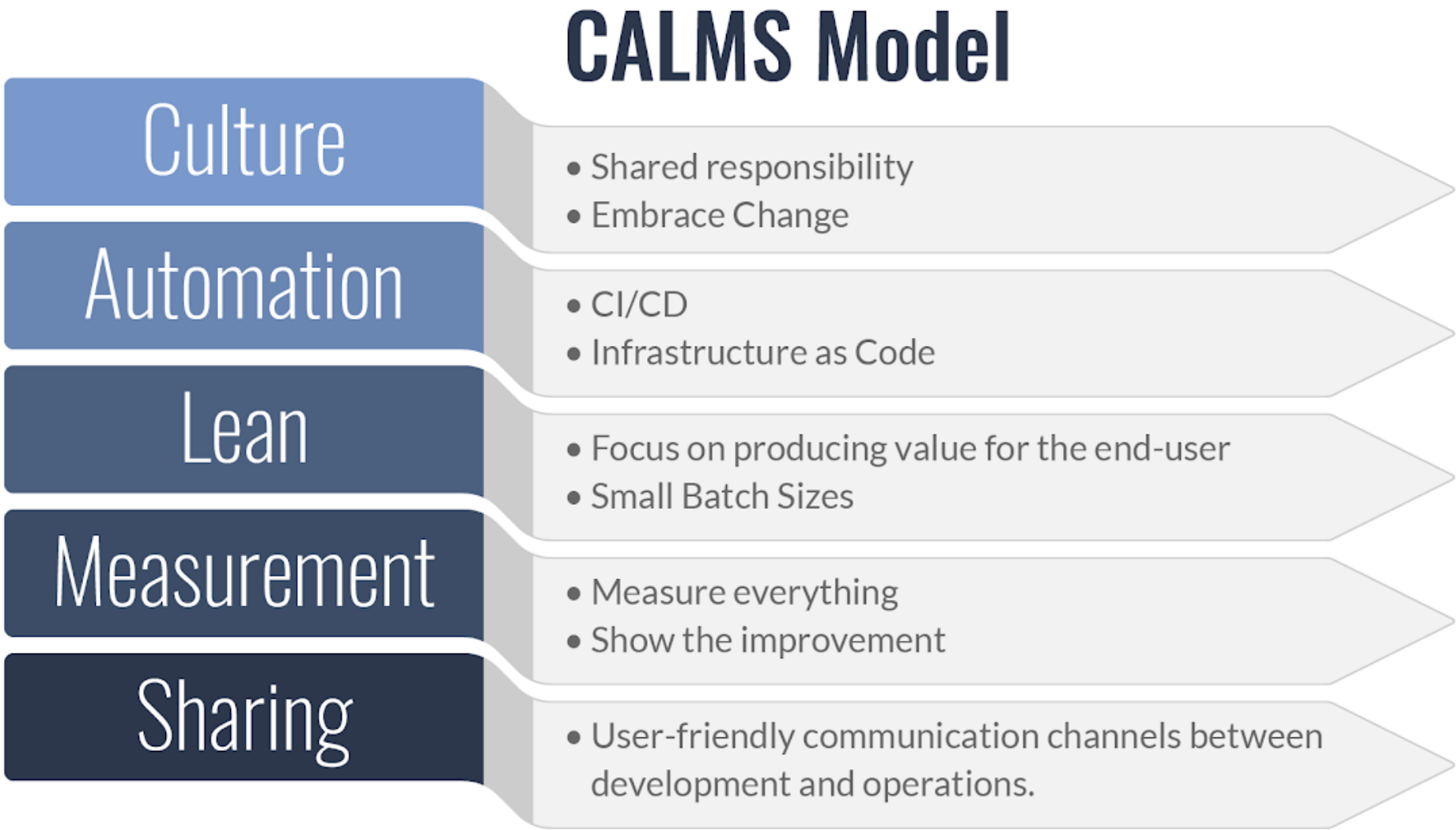
Promote a learning organization

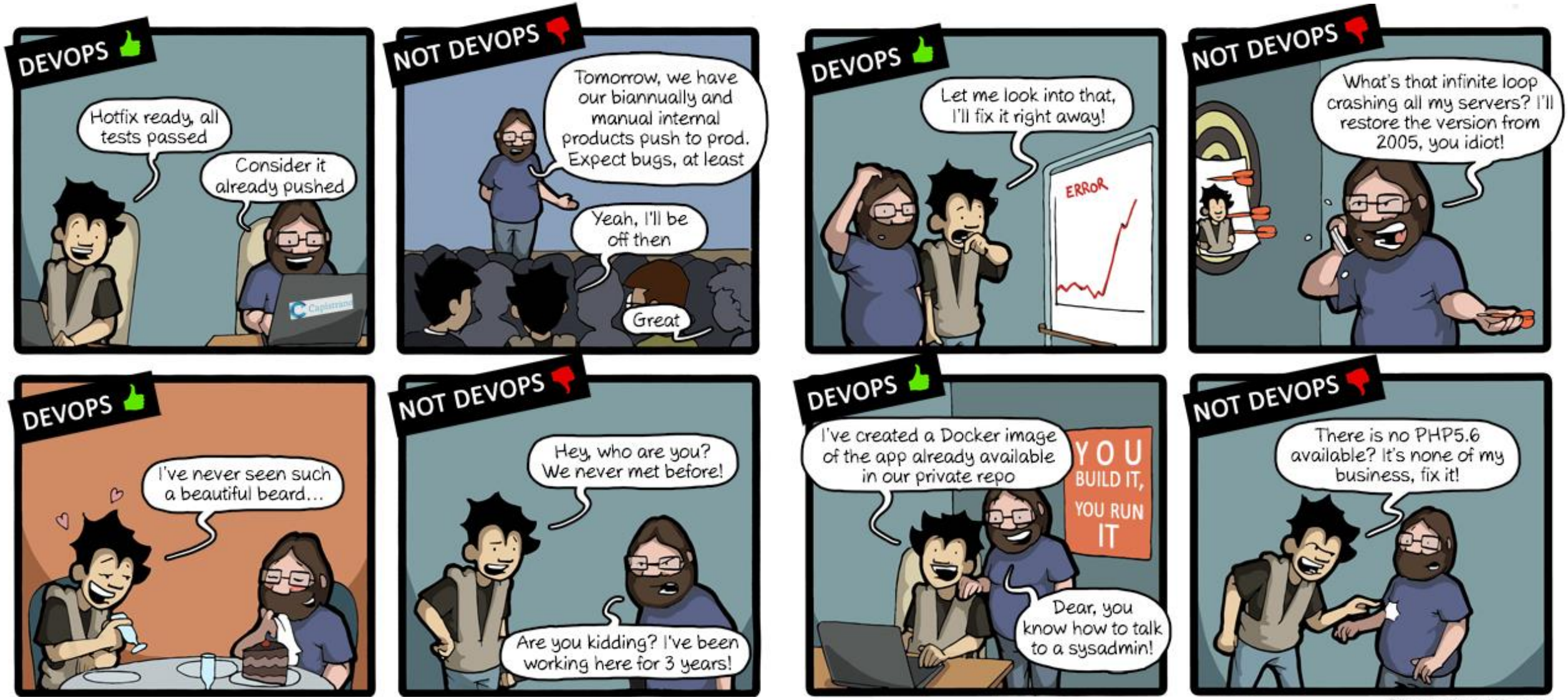
- Appropriate/constructive/"just"/blameless response to incident report (fear → dishonesty → problems keep unsolved)

- Adopt concrete responses, new guard rails etc

- Spread knowledge gained from experience (wiki, brown bag lunch…)

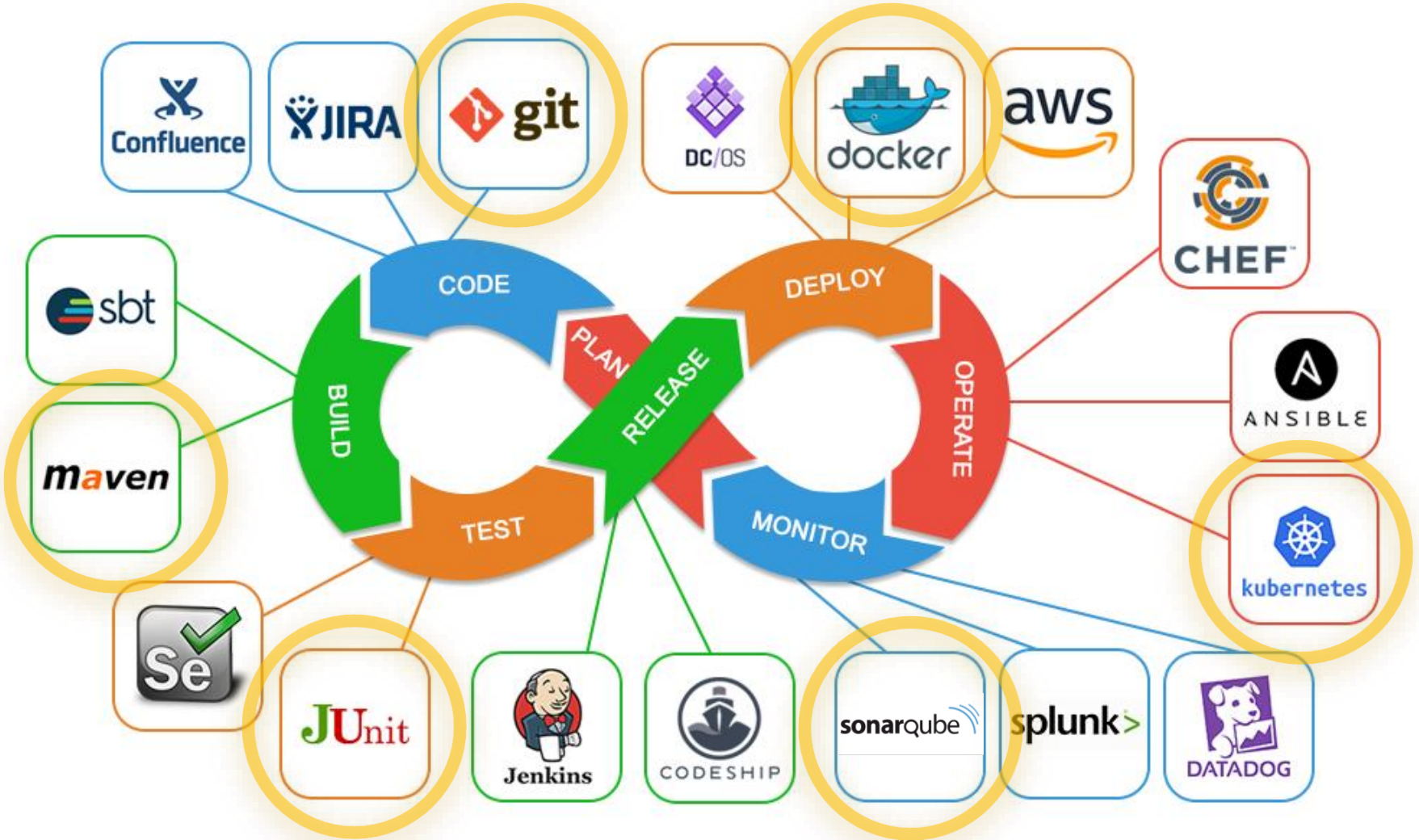
Keep CALMS and do DevOps





CommitStrip.com

DevOps tools



(from 12factor.net)

The 12-Factor App

methodology for building software-as-a-service apps that:

- Use declarative formats for setup automation, to minimize time and cost for new developers joining the project;
- Have a clean contract with the underlying operating system, offering maximum portability between execution environments;
- Are suitable for deployment on modern cloud platforms, obviating the need for servers and systems administration;
- Minimize divergence between development and production, enabling continuous deployment for maximum agility;
- And can scale up without significant changes to tooling, architecture, or development practices.

12 factors

I. Codebase
- Use version control
- One repo per application

II. Dependencies
- Explicitly declare and isolate dependencies
- Never rely on implicit existence of system-wide packages

III. Configuration (= anything that changes between environments)
- Should be stored as environmental variables, strictly separated from code: don’t check passwords (DB password, Amazon token…) into Git!

IV. Backing services (services consumed by your app, such as DB)
- Should be treated as attachable resources, neutral to “local vs. third-party”
- Located via config: service URL should be stored as an environment variable

V. Build, release, run
- Separate the stages, never change code at runtime

VI. Processes
- Should be stateless and share nothing
- Store state (with other persistent date) in a stateful backing service

VII. Port binding
- App is self-contained, server is a dependency and HTTP is exported as a service by binding to a port, don’t rely on runtime server injection

VIII. Concurrency
- Be able to scale up and scale out (decompose into micro-services and scale each type independently)

IX. Disposability
- Quick startup, resilience to failure, graceful shutdown (*servers are cattle, not pets*: treat servers as disposable commodities/resources)

X. Dev/Prod parity
- Dev environment should be as identical as possible to prod env (and to any env in between).

XI. Logs
- Treat logs as events

XII. Admin processes
- Admin tasks (DB migration…) should be run in same env as normal processes