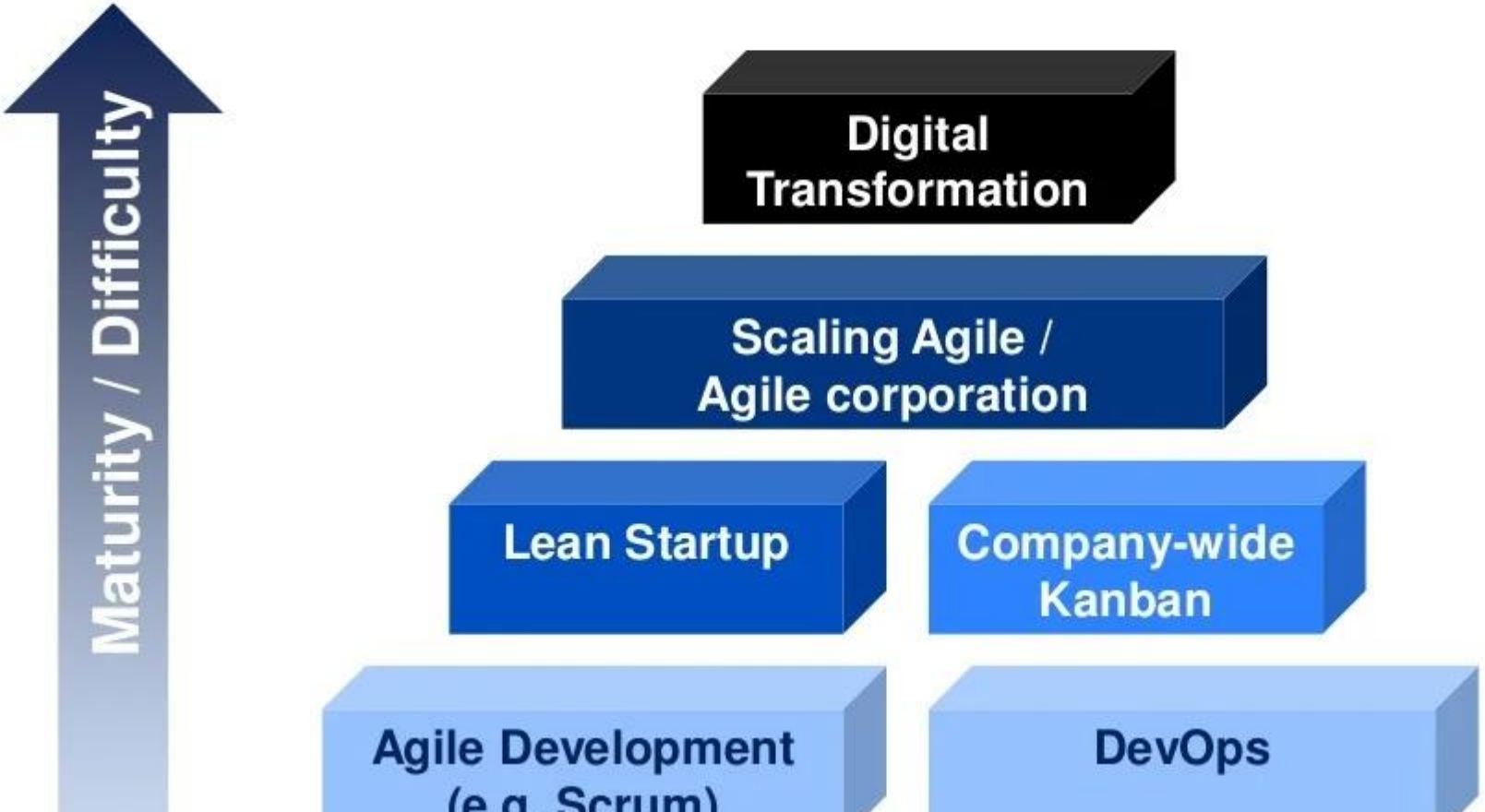
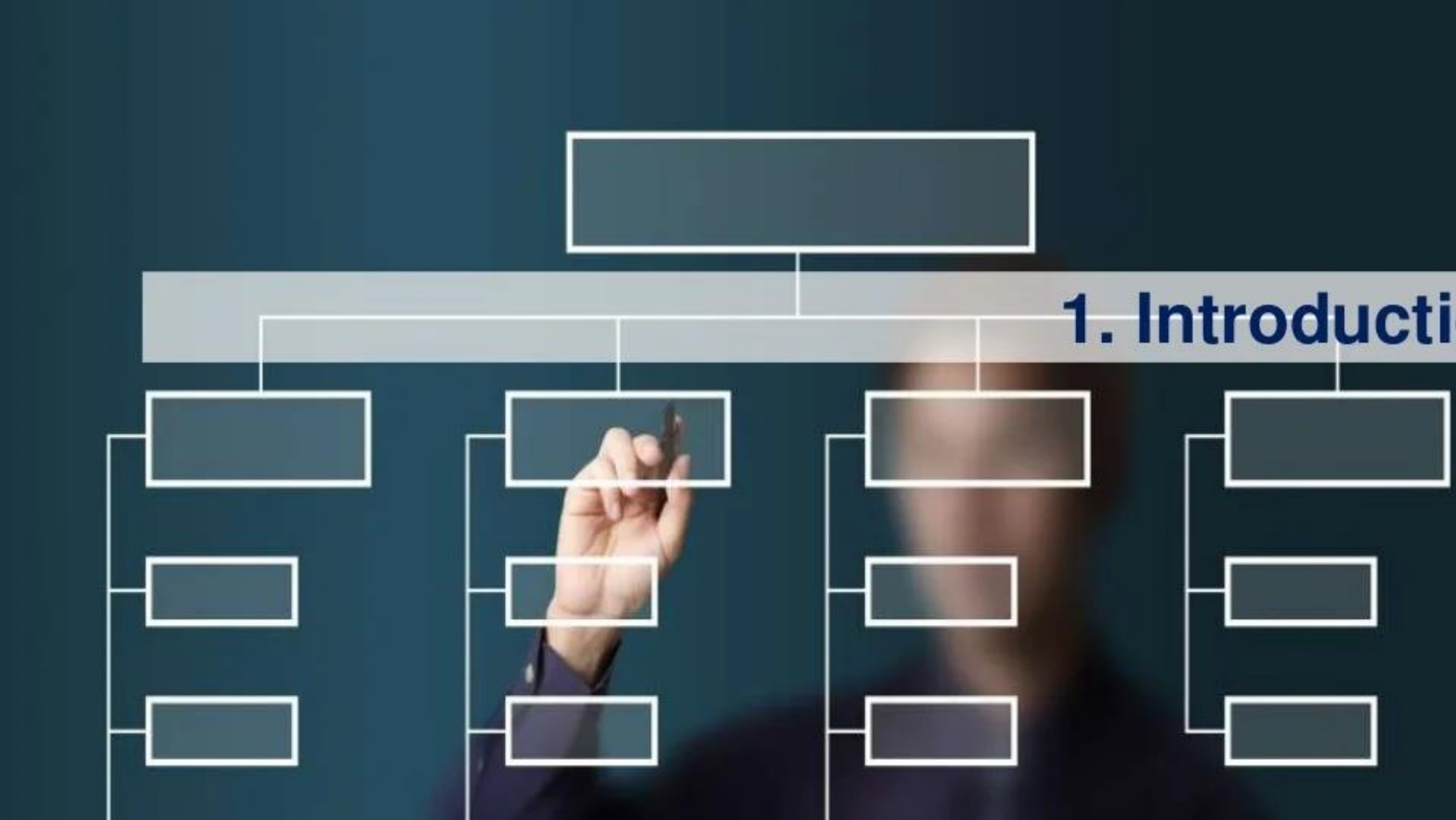
DevOps explained

© Jerome Kehrli @ niceideas.ch
https://www.niceideas.ch/roller2/badtrash/entry/devops-explained

DevOps / Lean Startup – why are these important

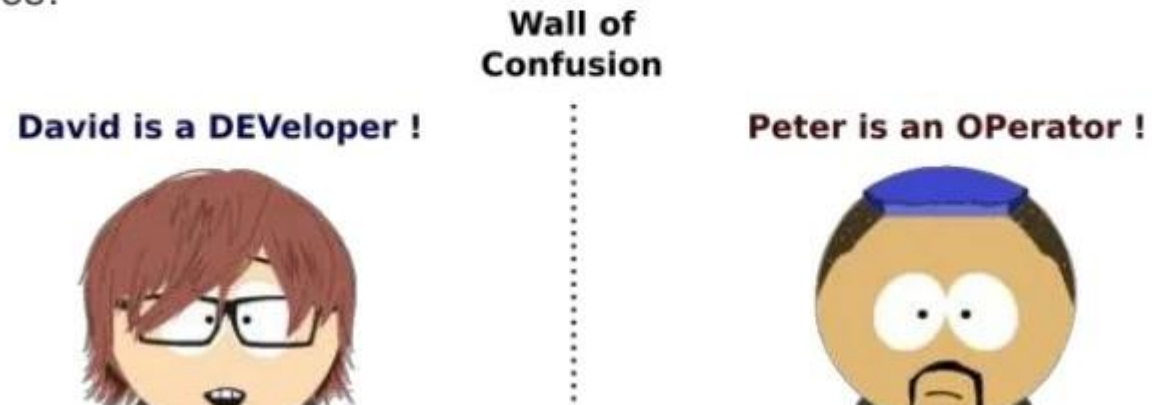




1. Introducti

- **DevOps is a methodology**
  - A set of principles and practices
  - help both developers and operators reach their goals
    - while maximizing value delivery to the customers or the users
    - as well as the quality of these deliverables.

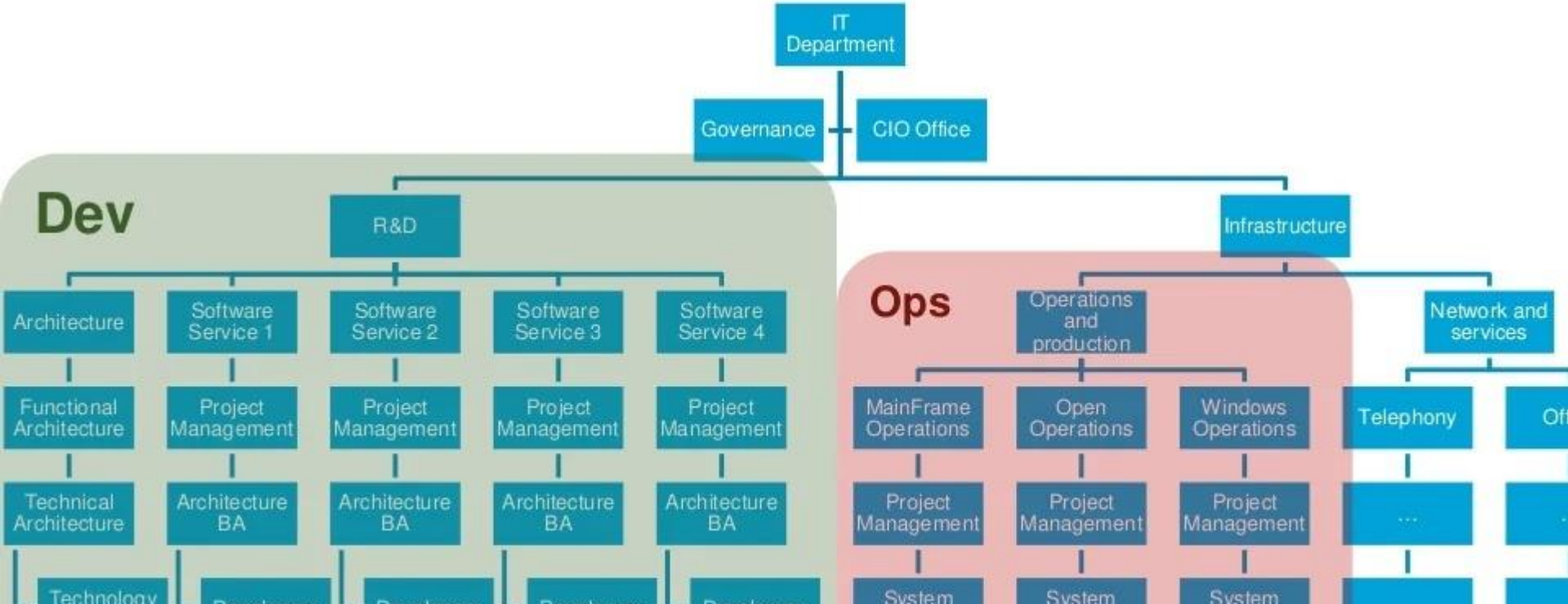
- The problem comes from the fact that developers and operators - while both required by corporations with large IT departments - have very different objectives.



Improve TTM – Time To Market

- shorten lead time (from business idea to production)
- Improve time required for technical and technological evolutions
- … while of course keeping a **very high** level of quality, stability, availability, operability, performance, etc.
- **Problem:**

A typical IT organizatio



Again – different objective

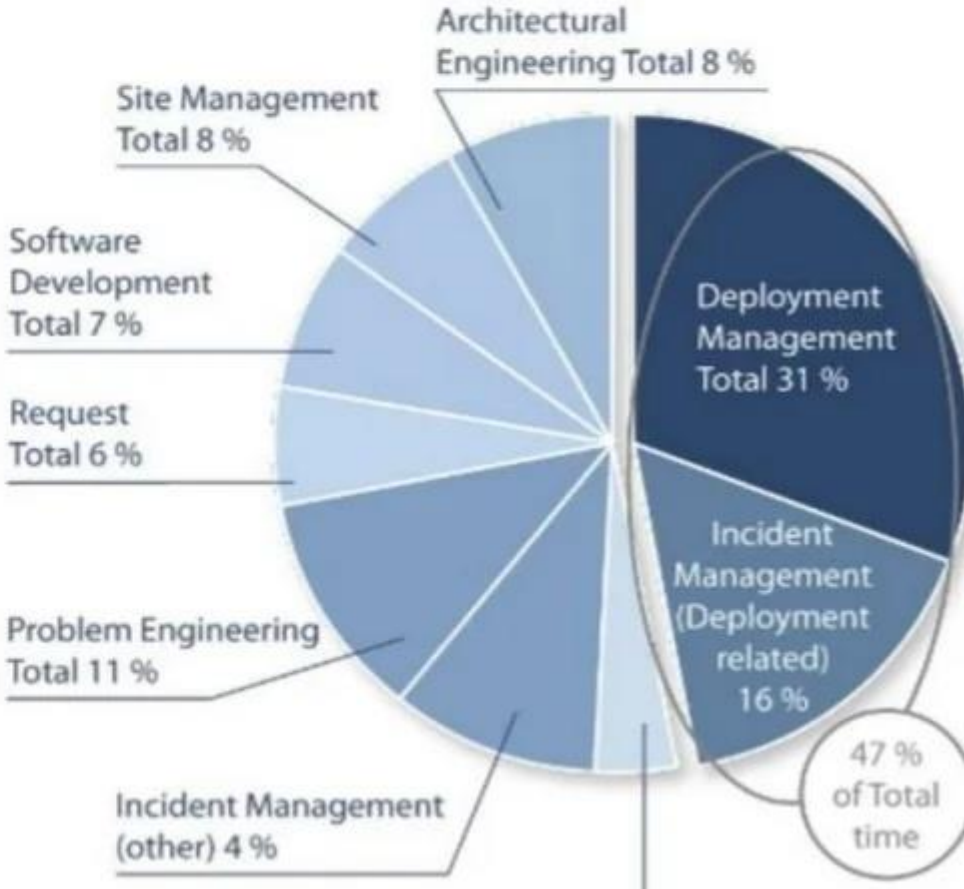
Wall of Confusion

David is a DEVeloper !

Peter is an OPerator !

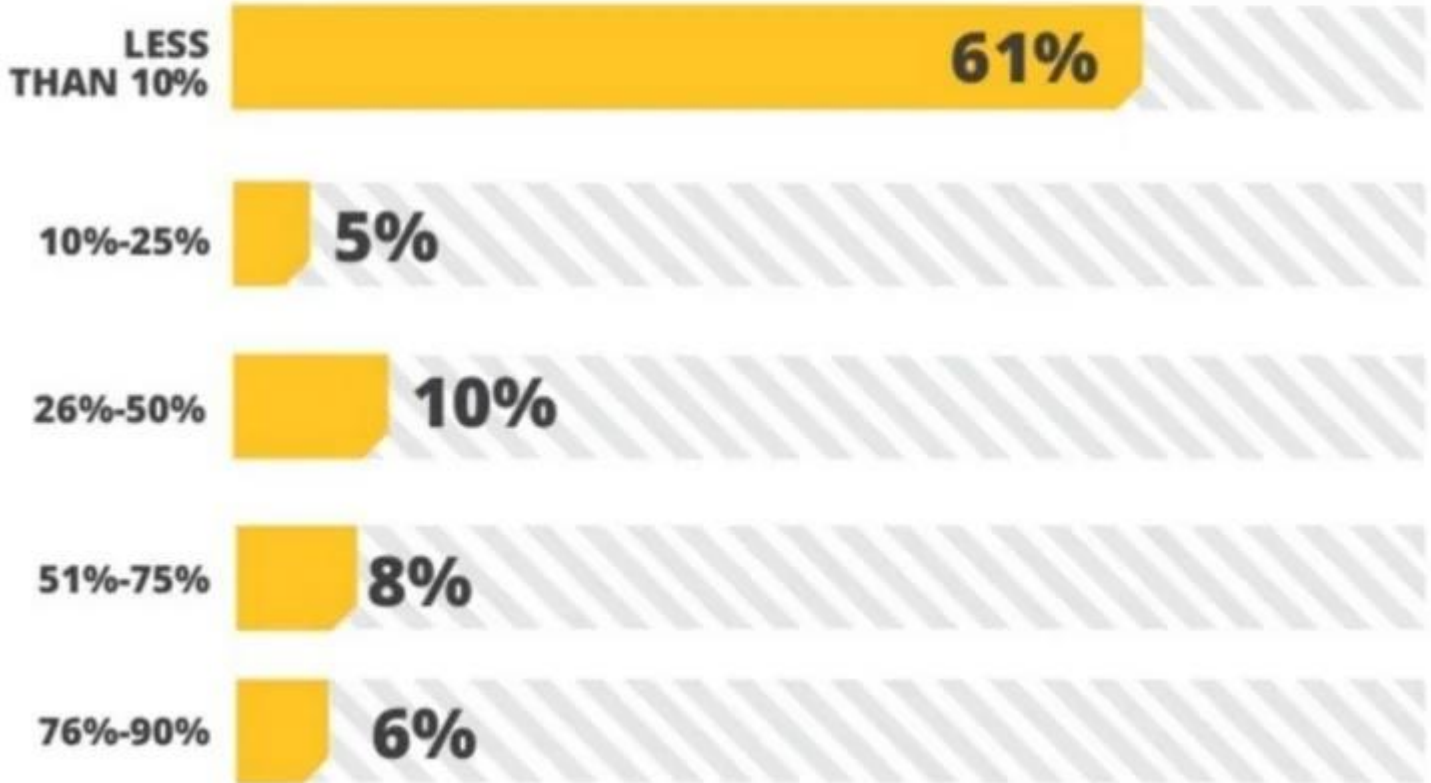
Ops frustration

- Almost 50% (47) of total time of Production Teams is dedicated to deployment
  - Doing the deployment
  - Fixing problems related to deployments
- An interesting KPI to follow …
- Two critical needs for evoluting these processes:
  - Automate the deployments to



Probability of succeeding an installation / deployment expressed as a function of the amount of manual shell commands

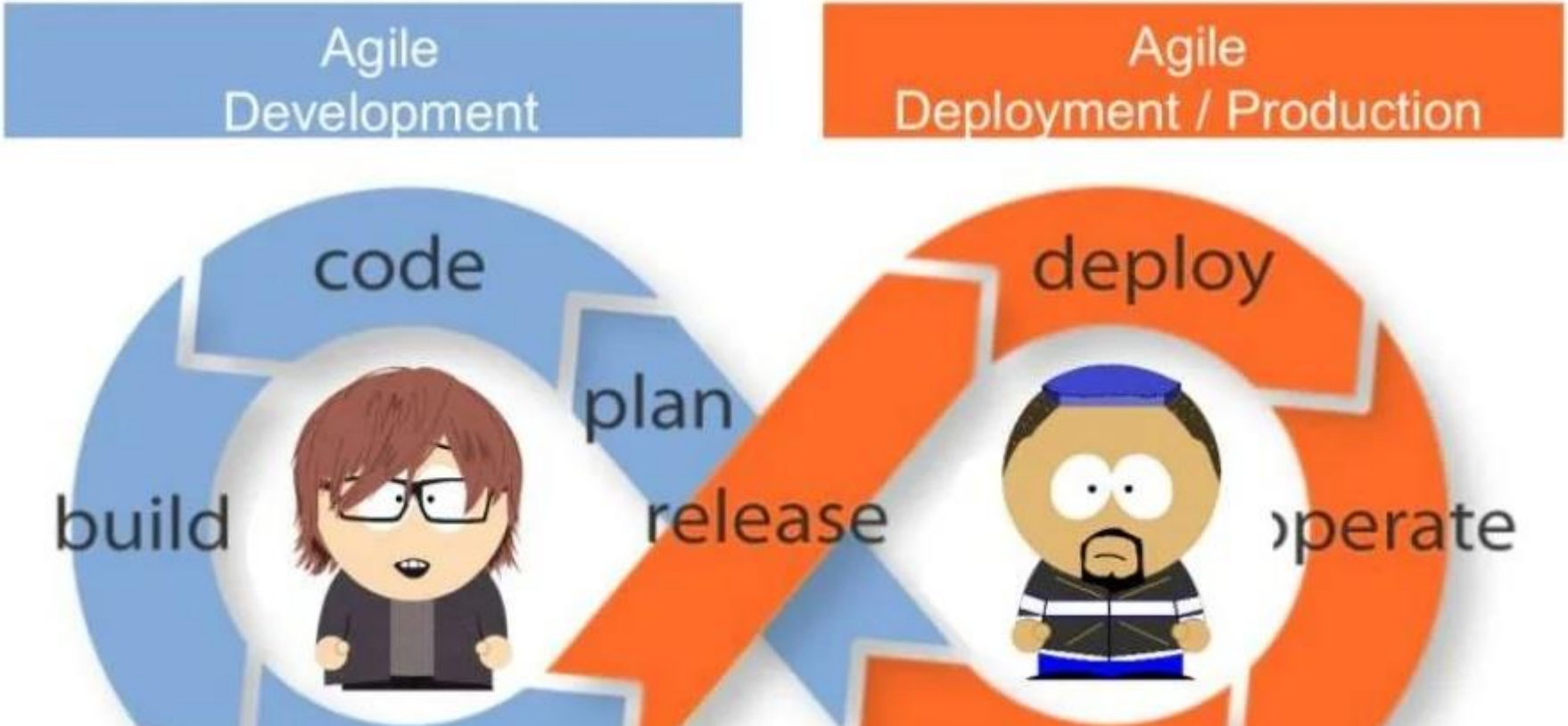
> To what extent is infrastructure automated in banking institutions ?



- Production release is difficult and tedious
- Operators are frustrated and developers do really nothing to help them
- But … on the other hand … web giants are successfully do several production rollout a day

Surprisingly,
for once,
there actually is a
(not so) magic silver bullet !

Extend Agility to Productio



- DevOps
  - consists mostly in extending agile development practices
    - further streamlining the movement of software change thru the build, validate, deploy and delivery stages
    - empowering cross-functional teams with full ownership of software applications - from design thru production support.
  - encourages **communication**, **collaboration**, **integration** and **automation** among software developers and IT operators in order to improve both the speed and quality of delivering software.
  - encourages empowering teams with the autonomy to build, validate, deliver and support their own applications.

- DevOps teams focus on standardizing development environments and automating delivery processes to improve delivery predictability, efficiency, security and maintainability.

1. Infrastructure as Code

2. Continuous Delivery

2. Infrastructure as co

- Because human make mistakes
- Because human brain is terribly bad at repetitive tasks
- Because human is slow compared to a bash script
- … and because we are humans

Infrastructure

=

code

Infrastructure as Co

Application Deployment

- Deploy application code … and rollback
- Configure resources (RDBMS, etc.)
- Start applications
- Join clusters

System Configuration

- JVM, app servers …
- Middlewares …
- Service configuration (logs, ports, user / groups, etc.
- Registration to supervision

Machine Installation

onfiguration Management Database
s is versionned under SVN, git, …)

Application / Service Deployment / Orchestration

- Capistrano
- Shell scripts (Python / Ba etc.)

System Installation / Configuration

- Chef
- Puppet
- CFEngine
- Red Hat Satelite
- Vagrant

- Consider **all of this is code** !
  - Stored and versioned in an SCM: git, SVN or whatever makes you happy
  - Like software code, one can test it !
    - And make sure it doesn't fail when one needs it at 2:00 am !
  - Can scale to any number of servers in parallel

- A *DevOps toolchain* consisting of multiple tools. Such tools fit into one or more of these categories, which is reflective of the software development and delivery process:
  - **Code** : Code development and review, version control tools, code merging
  - **Build** : Continuous integration tools, build status
  - **Test** : Test and results determine performance
  - **Package** : Artifact repository, application pre-deployment staging
  - **Release** : Change management, release approvals, release automation
  - **Configure** : Infrastructure configuration and management, Infrastructure as Code tools
  - **Monitor** : Applications performance monitoring, end user experience

- **Versioning, Continuous Integration and Automated testing of infrastructure components**
  - The ability to **version** the infrastructure - or rather the infrastructure building scripts or

Benefits (1/

- **Repeatability and Reliability** :
  - building the production machine is now simply running that script or that puppet command. With proper usage of docker containers or vagrant virtual machines, a production machine with the Operating System layer and, of course, all the software properly installed and configured can be set up by typing one single command - **One Single Command**. And of course this building script or mechanism is continuously integrated upon changes or when being developed, continuously and automatically tested, etc.
    Finally we can benefit on the operation side from the same practices we use with success on the software development side, thanks to XP or Agile.
- **Productivity** :
  - one click deployment, one click provisioning, one click new environment creation, etc. Again, the whole production environment is set-up using one single command or one click. Now of course that command can well run for hours, but during that time the operator can focus on more interesting things, instead of waiting for a single individual command to complete before typing the next one, and that sometimes for several

Benefits (2/

- **Guarantee that infrastructure is homogeneous** :
  - completely eliminating the possibility for an operator to build an environment or install a software slightly differently every time is the only way to guarantee that the infrastructure is perfectly homogeneous and reproducible. Even further, with version control of scripts or puppet configuration files, one can rebuild the production environment precisely as it was last week, last month, or for that particular release of the software.
- **Make sure standards are respected** :
  - infrastructure standards are not even required anymore. The standard is the code.
- **Allow developer to do lots of tasks themselves** :
  - if developers become themselves suddenly able to re-create the production environment on their own infrastructure by one single click, they become able to do a lot of production related tasks by themselves as well, such as understanding production failures, providing proper configuration, implementing deployment scripts, etc.

3. Continuous Delive

If it hurts, do it more ofte

- **The more often you deploy, the more you master the deployment process and the better you automate it**. If you have to do something 3 times a day, you **will** make it bullet proof and reliable soon enough, when you will be fed up of fixing the same issues over and over again.

- **The more often you deploy, the smallest will be the changesets you deploy** and hence the smallest will be the risk of something going wrong, or the chances of losing control over the changesets

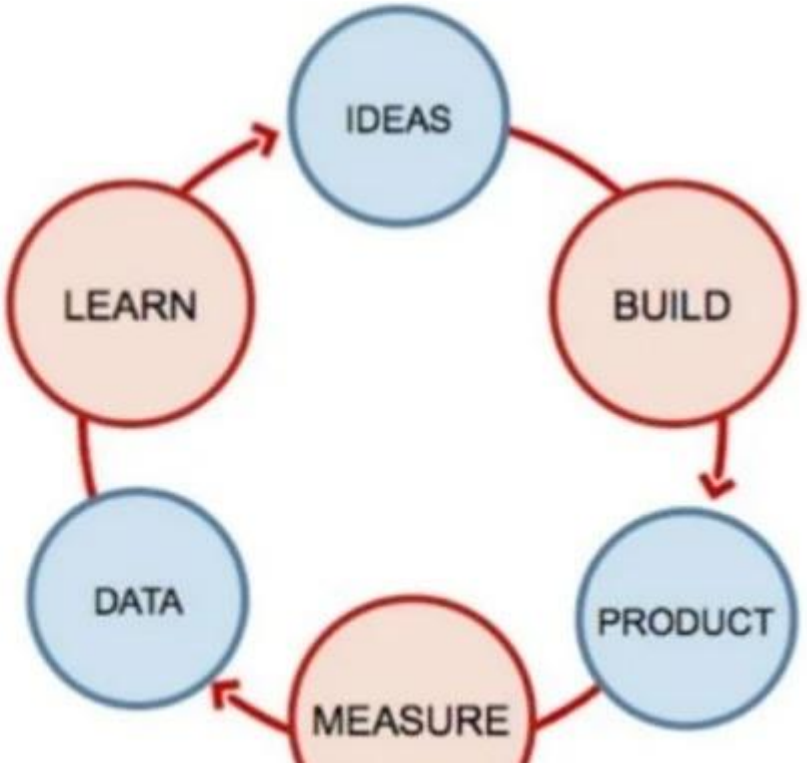
- **The more often you deploy, the best will be your TTR (Time to Repair / Resolution)** and hence the sooner will be the feedback you will get from your business users regarding that feature

3 key practice

- Continuous delivery refers to 3 key practices:
  - Learn from the fields
  - Automation
  - Deploy more often

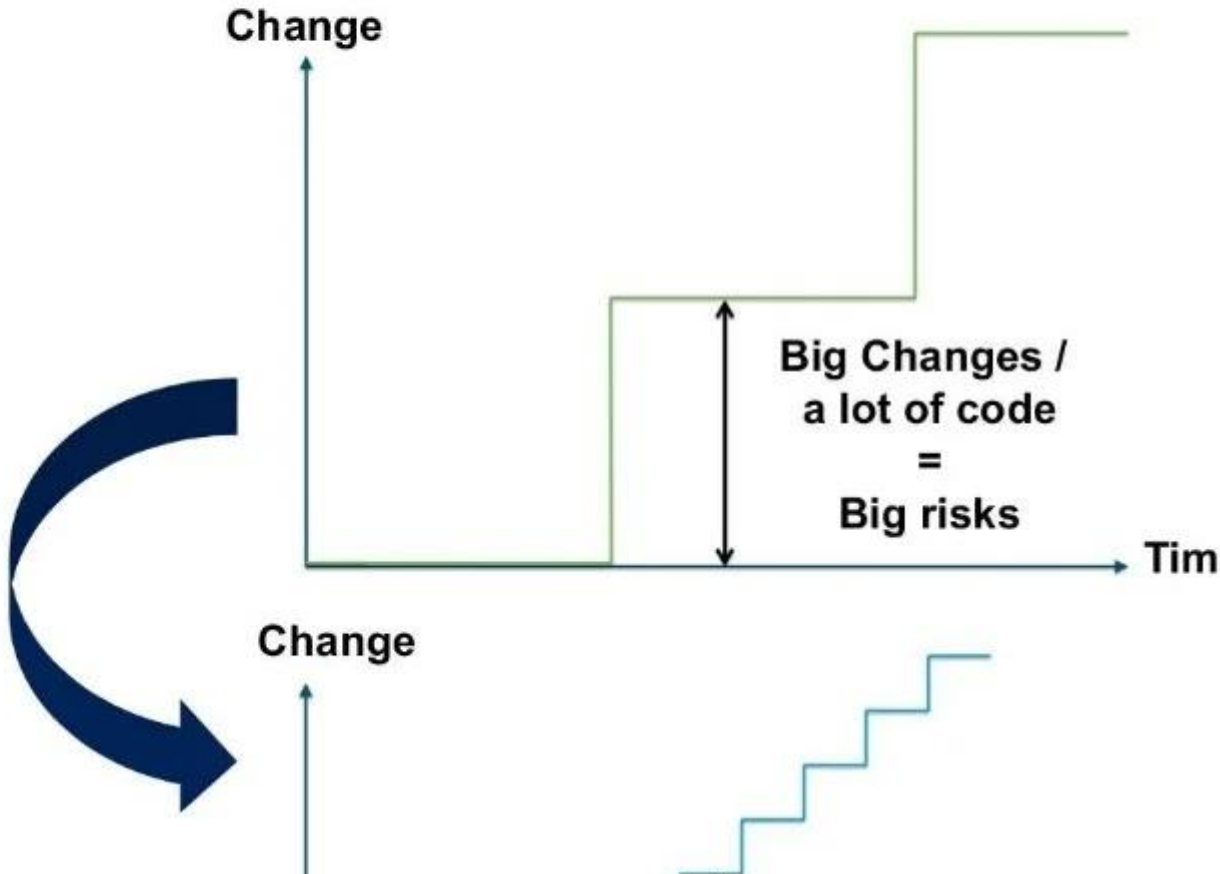
- There is no truth in the development team
- **Measure Obsession** (lean startup)
- Don't think, know!
  - And the only way to know is to measure, measure everything

- Speed is key :
  - Learn fast !
  - Code fast !
  - Measure fast !



Deploy More Ofte

- *"If it hurts, do it more often"*
  - Firstly most of these tasks become much more difficult as the amount of work to be done increases, but when broken up into smaller chunks they compose easily.
  - The second reason is *Feedback*. Much of agile thinking is about setting up feedback loops so that we can learn



Continuous Delivery requiremen

- **Continuous integration** of both the software components development as well as the platform provisioning and setup.
- **TDD** - Test Driven Development. This is questionable ... But in the end let's face it: TDD is really the single and only way to have an acceptable coverage of the code and branches with unit tests (and unit tests makes is so much easier to fix issues than integration or functional tests).
- **Code reviews** ! At least code reviews ... pair programming would be better of course.
- **Continuous auditing software** - such as Sonar.
- **Functional testing automation** on production-level environment
- Strong **non-functional testing automation** (performance, availability, etc.)
- **Automated packaging and deployment**, independent of target environment

Zero Downtime Deploymen

"Zero Downtime Deployment (ZDD) consists in deploying a new version of a system without any interruption of service."

- ZDD consists in deploying an application in such a way that one introduces a new version of an application to production without making the user see that the application went down in the meantime. From the user's and the company's point of view it's the best possible scenario of deployment since new features can be introduced and bugs can be eliminated without any outage.

- I'll mention 4 techniques:
  - Feature Flipping
  - Dark launch
  - Blue/Green Deployments
  - Canari release

Feature flippi

- Feature flipping allows to enable / disable features while the software is running. It's really straightforward to understand and put in place: simply use a configuration properly to entirely disable a feature from production and only activate it when its completely polished and working well.

- For instance to disable or activate a feature globally for a whole application:

```
if Feature.isEnabled('new_awesome_feature')
    # Do something new, cool and awesome
else
    # Do old, same as always stuff
end
```

- Or if one wants to do it on a per-user basis:

Dark Launc

Use production to simulate load !

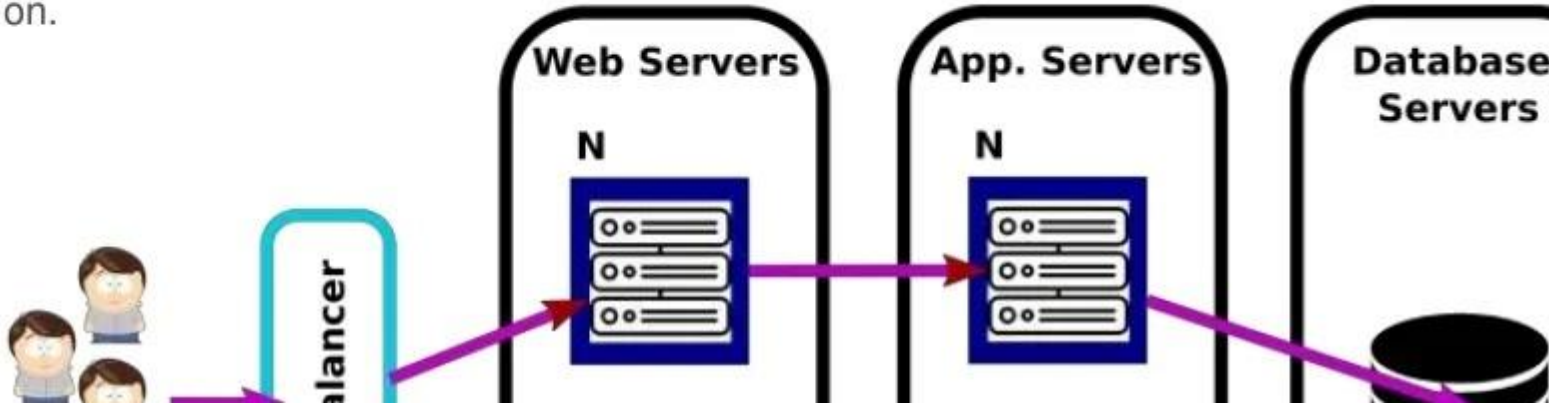
- Its difficult to simulate load of a software used by hundreds of millions of people in a testing environment.
  without realistic load tests, it's impossible to know if infrastructure will stand up to the pressure.

- Instead of simulating load, why not just deploy the feature to see what happens without disrupting usability?
- Facebook calls this a dark launch of the feature.

  - Let's say you want to turn a static search field used by 500 million people into an autocomplete field so your users don't have to wait as long for the search results. You built a web service for it and want to simulate all those people typing words at once and generating multiple requests to the web service.
  - The dark launch strategy is where you would augment the existing form with a hidden background process t sends the entered search keyword to the new autocomplete service multiple times.
  - If the web service explodes unexpectedly then no harm is done; the server errors would just be ignored on t web page. But if it does explode then, great, you can tune and refine the service until it holds up.

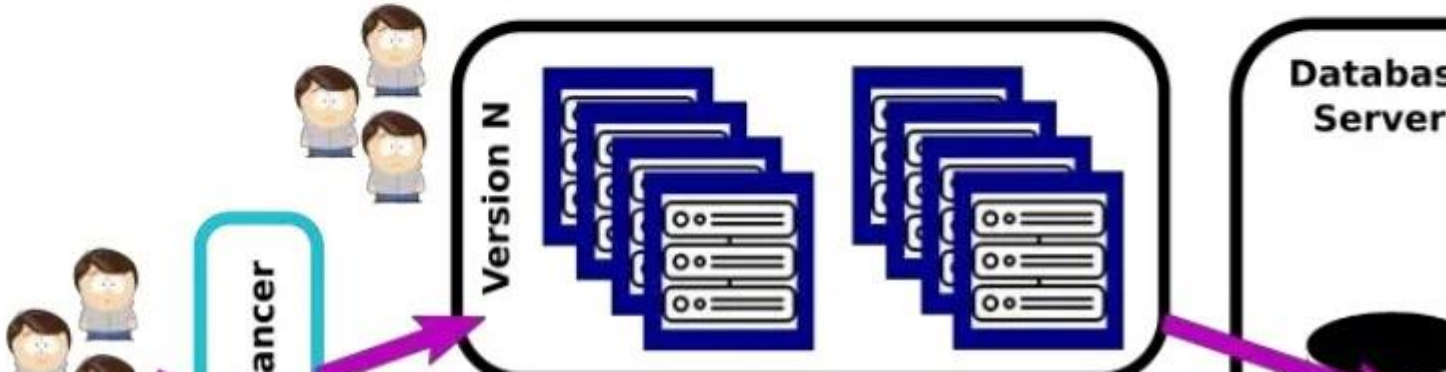
Blue/Green Deploymen

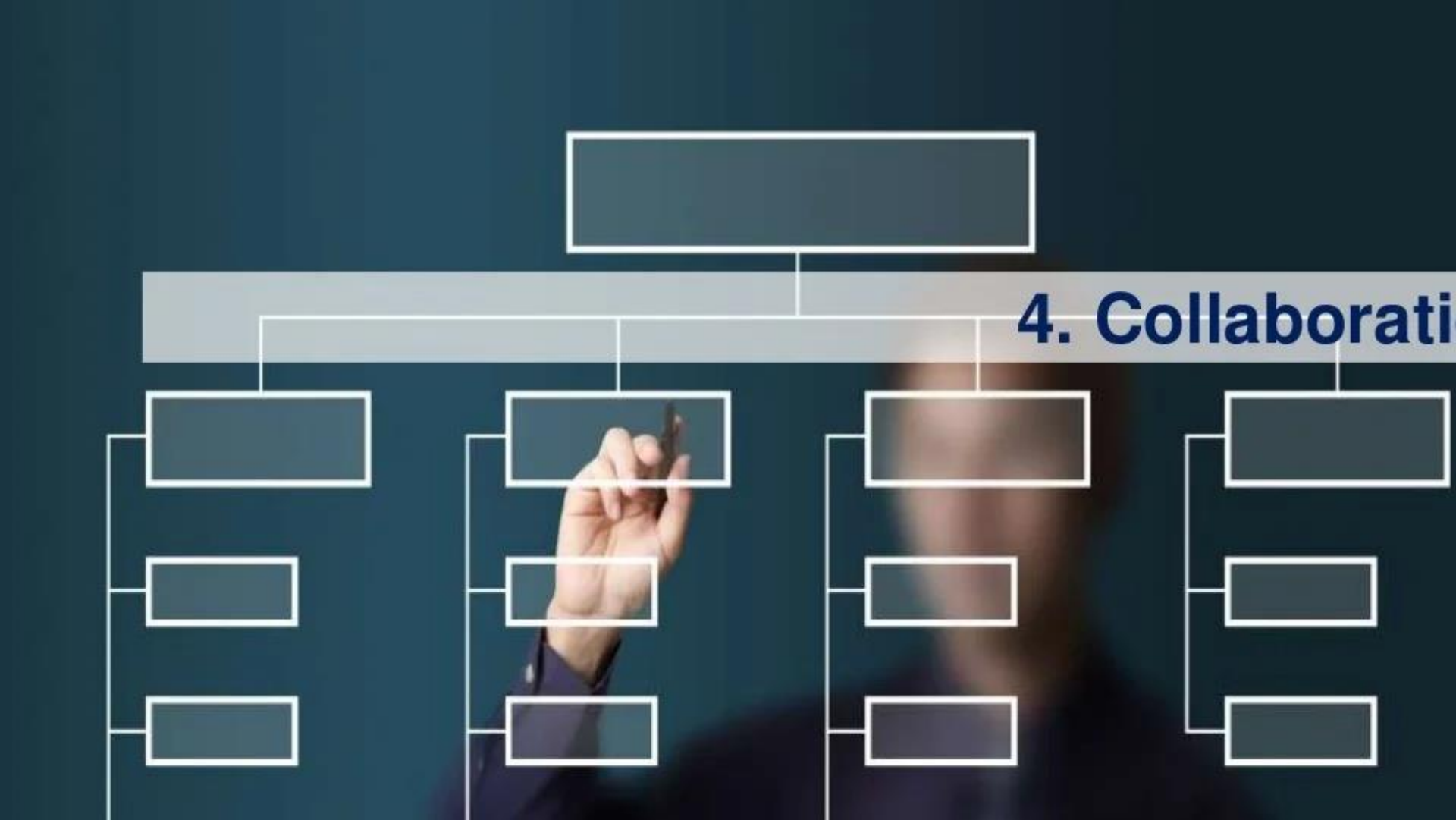
- *Blue/Green Deployments* consists in building a second complete line of production for version N + 1.
  - Both development and operation teams can peacefully build up version N + 1 on this second production line.
  - Whenever the version N + 1 is ready to be used, the configuration is changed on the load balancer and users are automatically and transparently redirected to the new version N + 1.
  - At this moment, the production line for version N is recovered and used to peacefully build version N + 2.
  - And so on.



Canari Releas

- *Canari release* is very similar in nature to *Blue/Green Deployments* but it addresses the problem to have multiple complete production lines.
  - The idea is to switch users to the new version in an incremental fashion : as more servers are migrated from the version N line to the version N + 1 line, an equivalent proportion of users are migrated as well.
  - At first, only a few servers are migrated to version N + 1 along with a small subset of the users. This also allows to test the new release without risking an impact on all users.
  - When all servers have eventually been migrated from line N to line N + 1, the release is finished and everything can start all over again for release N + 2.




4. Collaborati

WORKED FINE IN
DEV

The wall of confusio

Wall of Confusion

LOCAL OBJECTIVES : Deliver new functionalities (ideally of quality)

Product culture

LOCAL OBJECTIVES : Guarantee the “run” of applications (stability)

Service culture

A Fairy tale

- *The development team kicks things off by "throwing" a software release "over the wall" to Operations.*
- *Operations picks up the release artifacts and begins preparing for their deployment. Operations manually hacks the deployment scripts provided by the developers or, most of the time, maintains their own scripts. They also manually edit configuration files to reflect the production environmen*
- *At best they are duplicating work that was already done in previous environments, at worst they a about to introduce or uncover new bugs.*
- *The IT Operations team then embarks on what they understand to be the currently correct deployment process, which at this point is essentially being performed for the first time due to the script, configuration, process, and environment differences between Development and Operation*
- *Of course, somewhere along the way a problem occurs and the developers are called in to help troubleshoot.*
- *Operations claims that Development gave them faulty code. Developers respond by pointing out that it worked just fine in their environments, so it must be the case that Operations did something wrong.*
- *Developers are having a difficult time even diagnosing the problem because the configuration, file*

- DevOps helps to enable IT alignment by aligning development and operations roles and processes in the context of shared business objectives. Both development and operations need to understand that they are part of a unified business process. DevOps thinking ensures that individual decisions and actions strive to support and improve that unified business process, regardless of organizational structure.

- Even further, as Werner Vogel, CTO of Amazon, said in 2014 :

"You build it, you run it."

Software Development Proces

Functional requirements
- Specification of features
- Priorization of features

Functional tests
- automated functional tests
- Integrated in build process

Business Usage
- Issues and bugs
- Evolutions

Operators are the other users of the softwar

Functional requirements
- Specification of features
- Priorization of features

Functional tests
- automated functional tests
- Integrated in build process

Business Usage
- Issues and bugs
- Evolutions

Analyze → Plan → Design → Build → Test → Deploy

Non-functional requirements
- Useful troubleshooting logs / Key metrics

Non-Functional tests
- Automated technical tests

Technical Operation
- Performances and

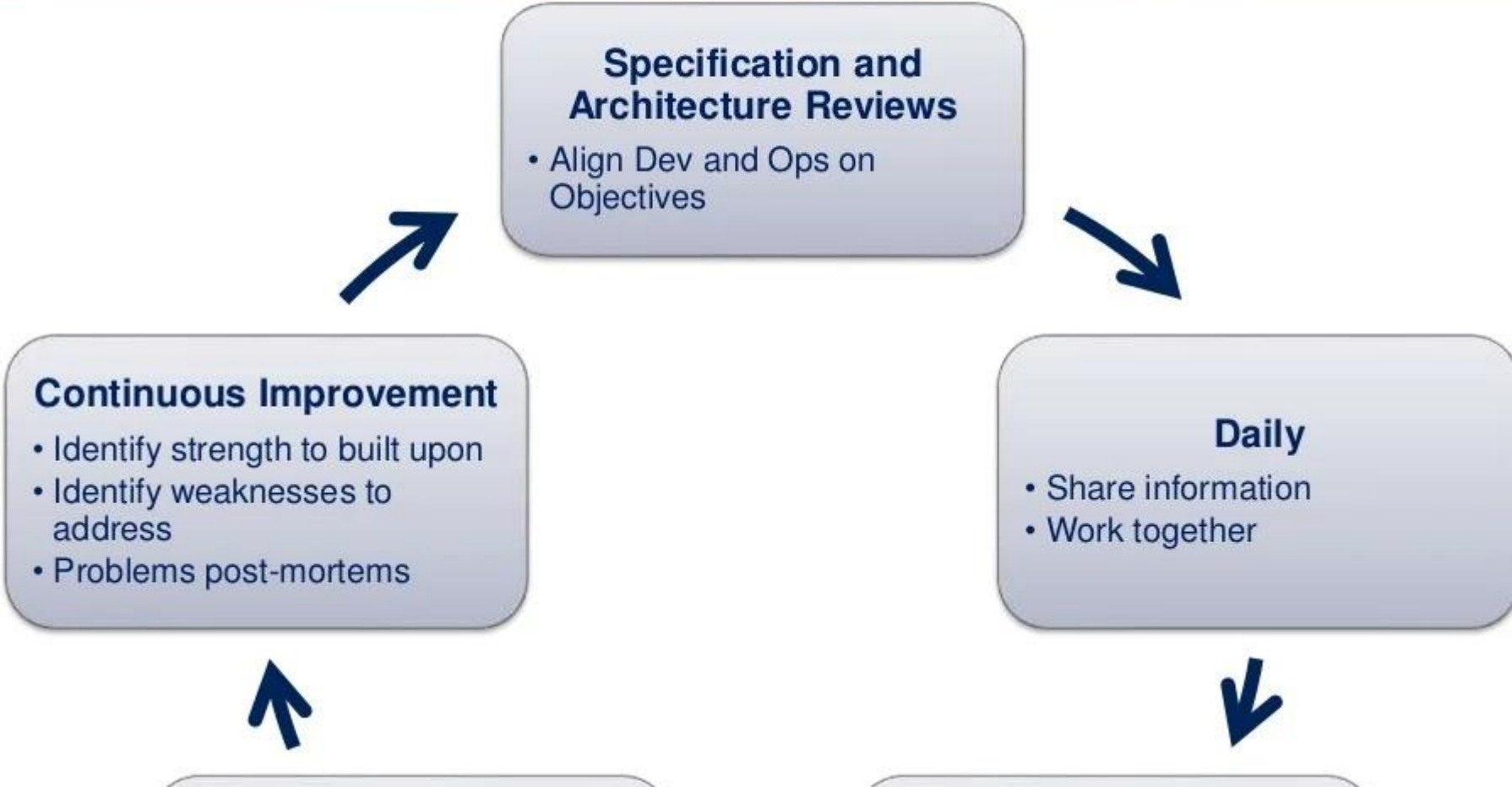
- SCM System
- Binary Repositories
- Continuous Integration

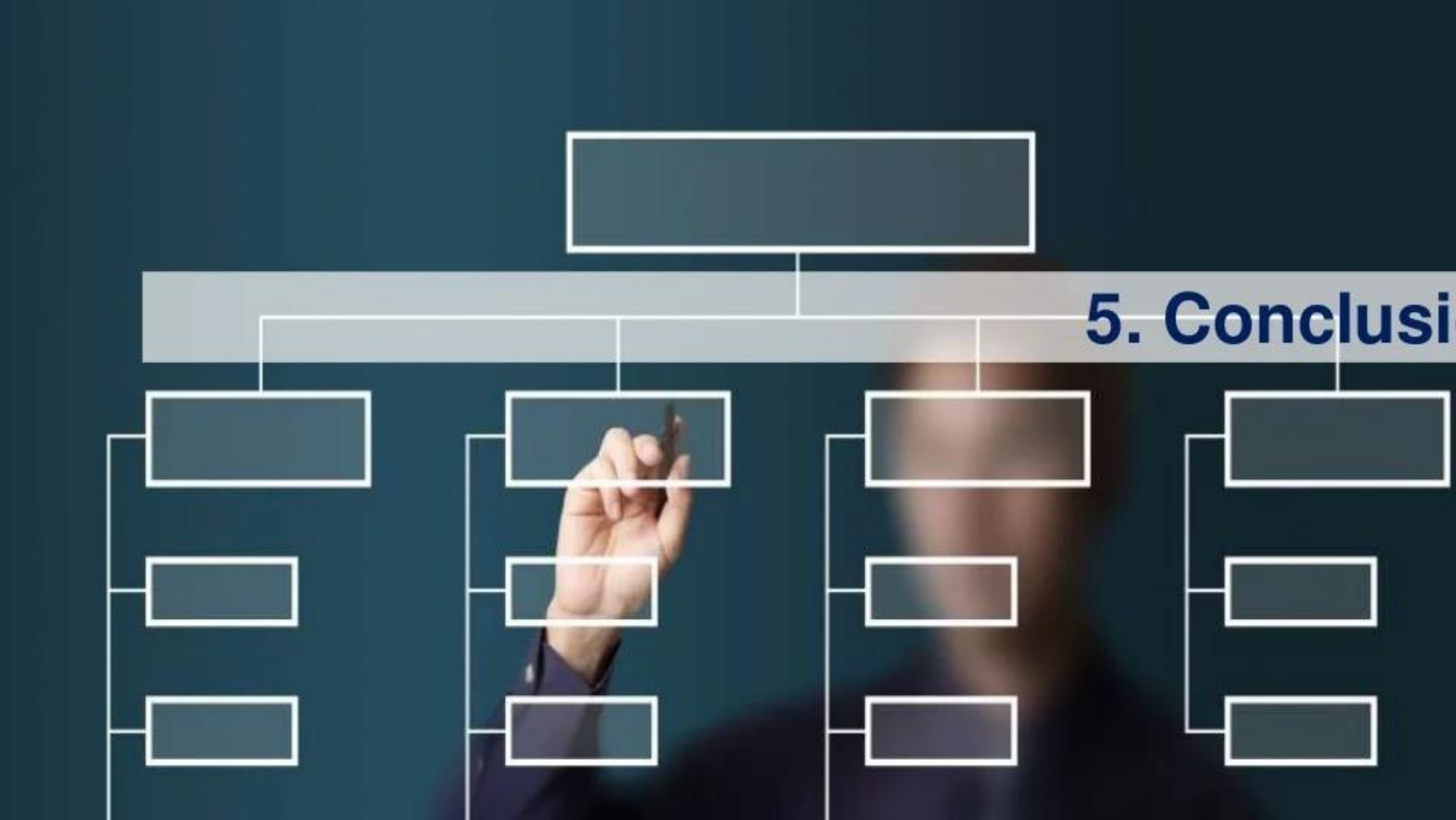
- Facilitate / Automate deploymen
- Help dev team become autonomous (in particular for deployment befor eproductions)

- Facilitates diagnostics

- Give access to logs

Aligning Development and Operation Team




5. Conclusi

1. Infrastructure as Code
- Automate provisionning
- Speed up deployments
- Make them repeatable and reliable

2. Continuous Delivery
- Improve TTM
- Lower TTR
- Zero Downtime Deployments

Thanks for listening