WAVESTONE

DEVOPS

THE SECRET OF AGILE START-UPS—NOW WITHIN THE REACH OF LARGE ORGANIZATIONS

WAVESTONE

www.wavestone.com

Wavestone is a consulting firm, created from the merger of Solucom and Kurt Salmon's European Business (excluding retails and consumer goods outside of France). The firm is counted amongst the lead players in European independent consulting.

Wavestone's mission is to enlighten and guide their clients in their most critical decisions, drawing on functional, sectoral and technological expertise.

DEVOPS: TOWARD END-TO-END AGILE

In an age of digital disruption, companies are seeking all possible ways to increase Agility and improve the time-to-market of their products.

And when it comes to getting there, there's a key word: Agile!

But you can't legislate for Agile. It involves a transformation on every level, from ways of working, through organizational design and decision-making processes, to the very architecture of information systems.

While companies are starting to employ Agile methods in the design and development of products, they still come up against an unsurmountable wall that separates "Dev" and "Ops."

This wall was created gradually through the construction of information systems that became increasingly heavy, monolithic, and strongly interwoven with each other. To this have been added organizational silos that keep development and production teams apart.

To change, and move toward DevOps at large scale, we must take our inspiration from the practices of the GAFA (Google, Apple, Facebook, and Amazon) companies. These web giants have been lucky enough to be able to build their ISs in green-field mode, thinking modularity and Agile from the start.

What they teach us is that DevOps is, on the one hand, a subtle blend of modularity and major autonomy, and, on the other, strict rules and operating frameworks in order to ensure that the entire system fits together.

This publication presents the principles of DevOps implementation, and our recommendations for how best to embark on this profound transformation, something that all ISDs must pursue.

I hope you enjoy reading it!

LAURENT BELLEFIN
Associate Director

AUTHORS

Maximilien Moulin

Maximilien is a manager in Wavestone's IT & Data Architecture Practice, who specializes in Cloud and Agility projects. A graduate of INSA Lyon engineering school, he began his career working on internal IS architecture at Orange. After a period as an entrepreneur, he joined Wavestone where he has spent four years advising the company's key accounts on their Cloud and Agility strategies.

maximilien.moulin@wavestone.com

Hasmik Manouchian

Hasmik is a manager in Wavestone's IT & Data Architecture Practice. She specializes in technical architecture, agility, and governance. She joined Wavestone 6 years ago, after graduating from IAE Lyon 3 engineering school. She assists the company's clients on their transformation projects.

hasmik.manouchian@wavestone.com

Pascal Bour

Pascal Bour is a manager who specializes in technical architecture and the industrialization of production. A graduate of ENSEIRB, he has been supporting Wavestone's clients for some eight years in the design phases of strategic projects, as well as helping them secure and optimize their operations.

pascal.bour@wavestone.com

This publication was written in collaboration with Matthieu Barret, Franck Lenormand et Riyad Yakine.

SUMMARY

DEVOPS

AGILE FROM END-TO-END

The rapid and reliable provision of relevant IT services has become an essential element of business competitiveness, whatever the sector of activity.

To increase responsiveness, without sacrificing reliability and quality, large organizations must transform their practices, in depth, throughout the IT production cycle: from design to deployment.

This transformation is already well under way through the democratization of Agile methodologies, such as Scrum, which have accelerated and increased the number of application deliveries. However, encumbered by the long delivery times for IT infrastructure, these methods are still not able to fulfill their potential. To truly improve time-to-value, that is, the time between the business expressing a need and the launch of an appropriate service, the entire IT value chain, from the business functions to operations, must be transformed.

The DevOps approach extends Agile practices from development into production, to take full advantage of this acceleration in application deliveries.

IMPROVING TIME-TO-VALUE

BY BRINGING THE OPS AND DEVS CLOSER TOGETHER

DevOps is an approach that aligns stakeholders in the IT value chain (business functions, development, and operations, but also security, architecture, compliance, and other cross-functional departments) toward a common business objective, with the aim of increasing the responsiveness of the company within its market.

This approach is embodied in a set of good practices, methodologies, and tools, which serves two purposes:

- Aligning objectives between Dev and Ops
- Automating the entire production cycle

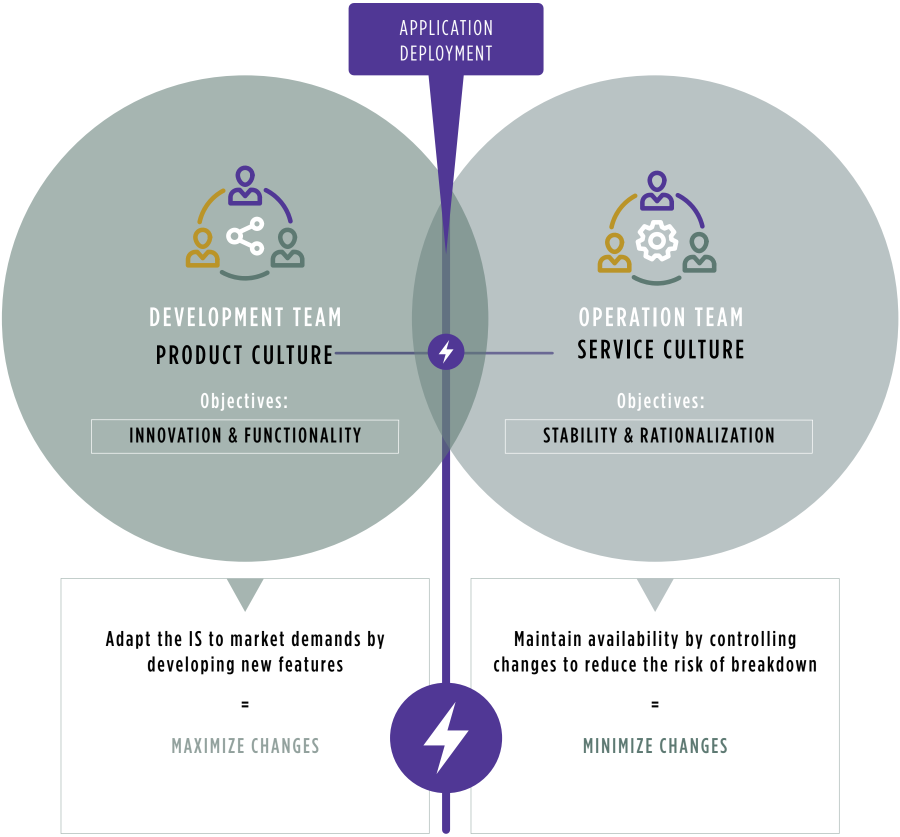
ALIGNING OBJECTIVES BETWEEN DEV AND OPS

The term DevOps was born out of a contraction of the terms «Development» and «Operations» (the teams involved in infrastructure engineering and the operation of IS production). Historically, the teams involved in the construction and operation of the IS were one, but the complexity of the architectures and the significant growth of ISs have led to a clear separation between those who develop the IS (Devs) and those who ensure its operation and proper functioning (Ops).

DEVOPSWASH

There is, as yet, no single, conventional view on what the term DevOps means. Some, such as Forrester, define it as the creation of an automated software delivery pipeline; others, such as Gartner, stress the adoption of Lean and agile practices to rapidly deliver IT services. This lack of precise definition, and the branding of many tools under the DevOps banner, clearly suggest that we are experiencing a period of «DevOpswash,» following on from previous waves of «greenwash» and «cloudwash" seen in recent years. There's a need, then, to remain cautious and not be caught out by the labeling game.

The wall of confusion



APPLICATION DEPLOYMENT

DEVELOPMENT TEAM

PRODUCT CULTURE

Objectives:

INNOVATION & FUNCTIONALITY

OPERATION TEAM

SERVICE CULTURE

Objectives:

STABILITY & RATIONALIZATION

Adapt the IS to market demands by developing new features

=

MAXIMIZE CHANGES

Maintain availability by controlling changes to reduce the risk of breakdown

=

MINIMIZE CHANGES

Today, collaboration between Development and Operations teams is often painful. Their respective aims, which cannot be a priori reconciled - innovation, development and reactivity for Dev, and the stability and reliability of the IS for Ops - and their strong mutual dependencies, explain the complexity of their relationship. This state of affairs can lead to a lack of performance, and also mutual «annoyance.»

Because there is too little involvement of Operations teams, Agile methodologies, as they are applied in most large companies, have not been able to address this problem. On the contrary, the acceleration of the application delivery frequency and use of a test and learn philosophy have sometimes widened the gap even more. Operations teams must deploy an increasing number of new features in production, often with an increase in error rates. At the same time, reliability requirements remain just as demanding.

This not only results in the creation of bottlenecks in the transition to production, but also to ever-greater «friction» between the Development and Operations teams.

DevOps practices and tools enable the inclusion of all players in the IT value chain into a single, integrated, and continuous process based on Agile principles (iterations, strong business collaboration, test and learn, etc.) and the development of a genuine culture of collaboration between Dev and Ops.

AUTOMATING THE ENTIRE PRODUCTION CYCLE

DevOps also relies on the implementation of automation tools throughout the entire IT production cycle: automation of infrastructure provisioning, application building, testing, and application deployment.

The primary aim of such automation is to absorb the increase in the volume and frequency of tests resulting from iterative operation. It is, therefore, critical to guaranteeing the quality of the application code at each iteration without creating a bottleneck downstream of application construction.

EXAMPLE OF KPIs

SOME EXAMPLES OF INDICATORS:		AND THE EXPECTED GAINS*:
Number of application deliveries	▶	200x more deliveries
Time to putting into production	▶	2 555x faster
Number of incidents	▶	3x fewer incidents
Resolution Time (MTTR)	▶	24x more rapid resolution
Number of rollbacks	▶	0 rollback

* Puppet and DORA, 2016 State of DevOps Report

Because it reduces human errors and facilitates measurability at each stage, automation in the DevOps process is also a lever to continuously improve, and render reliable, the entire manufacturing and deployment cycle.

Here, high-performance automation of the entire cycle is the main lever to avoid the bottleneck between development and going into production. This is the key to getting Operations teams out of their firefighting role (continuous management of problems) which they tend to be pushed into, following the widespread adoption of Agile practices.

DevOps emphasizes the automation of the entire manufacturing and deployment cycle. As a result, with the appropriate tooling, everything becomes measurable over the entire cycle. Measurement is the key to enabling stakeholder buy-in to the process, and to continuous improvement. That is how «DevOpswashing” can be defeated, but also a way of having the right indicators to demonstrate the gains over the whole cycle.

LE DEVOPS IS NOT :

- A FINISHED PRODUCT or a generic approach. There is no general theory that can be applied to all businesses; what's required is to tailor DevOps good practices to the specific business context.
- A FUSION of the remits of Dev and Ops. Moreover, DevOps does not necessarily trigger a change in organizational design, nor does it require the two teams to be grouped within the same structure.
- A "QUICK AND DIRTY" approach ignoring the processes or inducing a loss of control. In general, production remains in the domain of Ops.

FOUR PILLARS TO BUILD DEVOPS

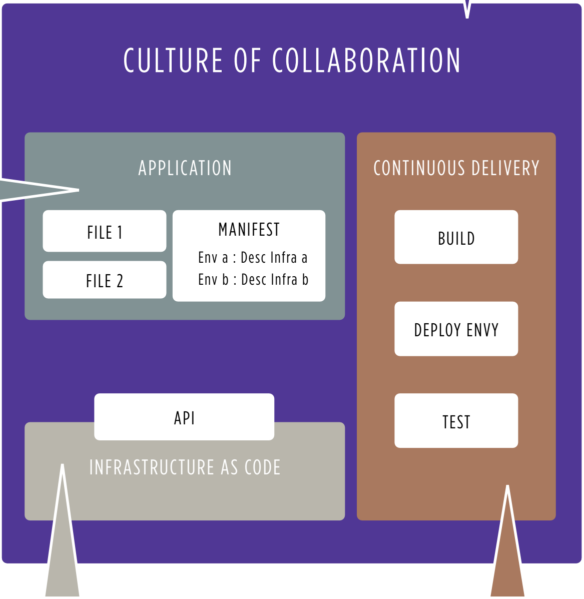
To understand what DevOps is, we need to understand what it is based on:

- A spot of automation
- And a lot of collaboration!

WHILE COLLABORATION HAS TO BE ALL-ENCOMPASSING, AUTOMATION CAN BEDIVIDED INTO THREE COMPONENTS:

COLLABORATION, which is the link between all these pillars, focusing on the alignment between Dev and Ops.

THE APPLICATION itself should be modular and integrate information about the infrastructure to be deployed, as well as the operational elements and automated tests to be performed.

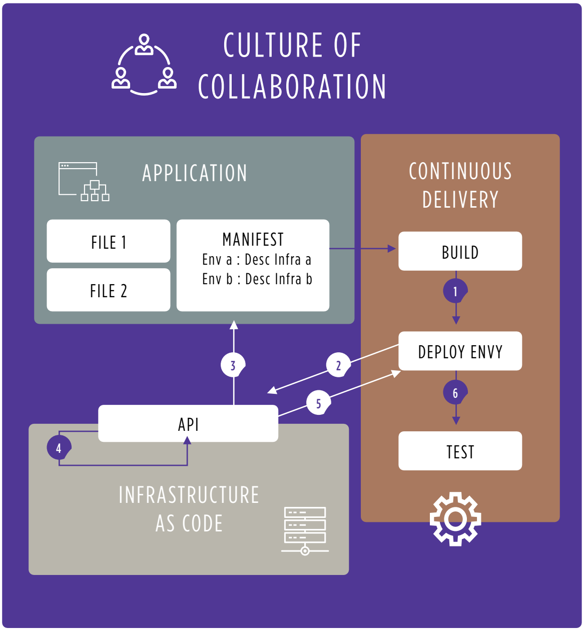


INFRASTRUCTURE AS CODE, which must deliver automated and standardized-interface runtime environments for the application, as a function of the needs expressed within its source code.

CONTINUOUS DELIVERY, which has to offer a pipeline that allows the testing of the application in an automated way, and its deployment on the right environment at the right time.

So, to deliver a DevOps transformation successfully, it is essential to pursue four parallel threads—the four pillars of DevOps:

- **Applications:** Prepare for the transformation of applications by making them modular and possible to automate
- **Continuous Delivery:** automate the delivery chain from the development to going into production
- **Infrastructure as Code:** lean toward a consumable service infrastructure, and also, one that can be integrated with application delivery
- **Collaboration:** change the culture and practices to move toward an organization without silos, and driven by Agile methodologies



1 The application once built (compiled) must be deployed in an environment (such as test, integration, production, etc.).

2 The Continuous Delivery pipeline uses the Infrastructure API to request construction of the environment.

3 The infrastructure uses the application's configuration file (Manifest) to create an environment tailored to the application.

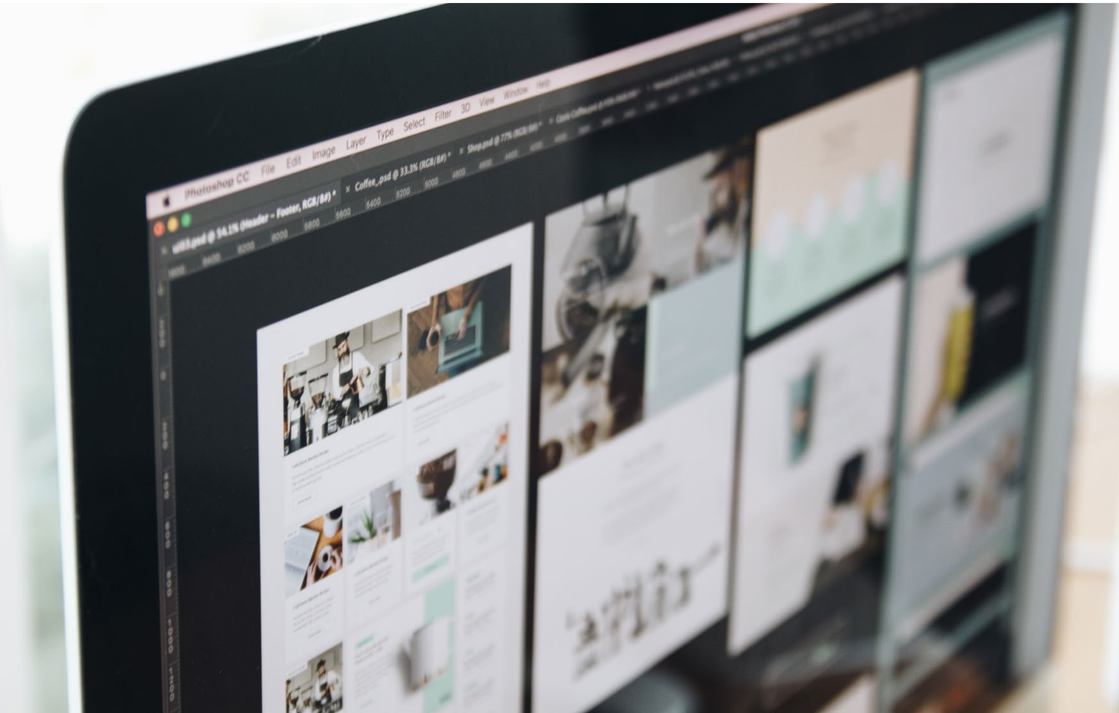
4 The infrastructure then orchestrates the effective creation of the environment.

5 Once the required environment is created, the application is deployed on it.

6 The application can then be tested according to the tests present in the application code

PILLAR I

MODULAR, LOOSELY-COUPLED APPLICATIONS



AN ARCHITECTURE THAT IS NECESSARILY MORE MODULAR

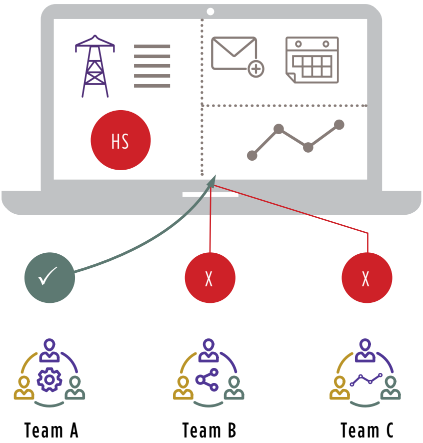
Being more Agile involves delivering new application versions, usually by sharing the work over several teams (often small, and multidisciplinary, “feature teams”). **This requires the decoupling of their work to achieve greater efficiency, and, therefore, the use of more modular application architectures.**

While service-oriented architectures have been held back by centralized and rigid governance structures, the focus here is on decentralized approaches that give developers autonomy in their publication and consumption activities.

An application built in this way is an assembly of modules (up to several dozen), which are all independent code elements, each under the responsibility of a «feature team”: a small, autonomous team that carries out the think, build, and run activities for the module.

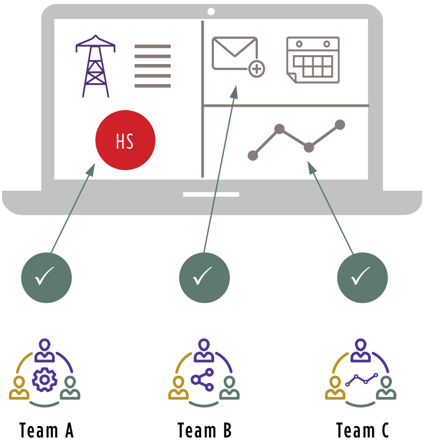
The exchanges between modules are standardized, and rely on interfaces (APIs), which allow them to be decoupled. These APIs will be able to use management solutions (API management) to allow developers to work autonomously on their creation, publication, and consumption. This will, however, require **service-mapping solutions to avoid a loss of control over the IS.**

MONOLITHIC ARCHITECTURE



With monolithic architecture, each delivery requires the entire monolith to be put into production.

MODULAR ARCHITECTURE



With modular architecture, several teams can work on different modules at the same time.

Beyond these broad threads, this approach is also an important facilitator in developing an Agile approach because it helps improve the quality of code, facilitate teamwork, speed up the execution of tests, and so on.

THE CASE OF MICRO-SERVICES

Micro-service architecture, a paradigm that has been developed from the practices of B2C players with very broad user bases, offers, in particular, a response to the need for specific functions in a system to be highly scalable. When following a rationale of dismantling functional boundaries and replacing them with autonomous finely-meshed services, which nevertheless continue to interact with each other, architecture, infrastructure, and operational practices must evolve considerably in order to manage a panoply of micro-services which may be instantiated in an ad-hoc fashion.

In a modular approach, the components of the application must, therefore, be very loosely linked—at all levels:

/ **In terms of frameworks and libraries**, which can be used as tools, but must not define the structure of the application layer.

/ **Between application modules and all things external, in order to simplify the carrying out of tests.** The application and its modules must be unitary testable without requiring a user interface, database, or other applications. This places a great importance on the unit tests within the code of each module of the application, as well as on the creation of plugs to allow unitary testing of a service.

/ **In terms of the user interface**. This must be replaceable without any impact on the application layer and its functions.

/ **In terms of data persistence**. To be scalable, this type of architecture requires stateless services. The persistence of data and sessions must be managed in a way that ensures an increase or reduction in load, while ensuring the consistency of the data.

/ **In terms of any other function that is external** to the application and comes from other applications or services.

RAPID AND WELL-MANAGED RELEASE

Designing applications in DevOps mode requires more robust tests to be performed (in particular through automating them) and taking more risks in the production stages. This needs development methods to evolve, and the introduction of new approaches such as Test Driven Development, Feature Flipping, or Blue-Green Deployment, in order to put certain functionalities into production in a partial or targeted way.

In the same way, this requires the application to be conceived so that it is directly operable, without waiting until it is put into production to address its placing under supervision or backup, the design of new automation scripts, performance problems, scalability or other operational issues.

UNIT TESTING: THE TEST-DRIVEN-DEVELOPMENT APPROACH

WHEN A DEVELOPER CODES A FUNCTION, THEY GENERALLY DO THE FOLLOWING THREE THINGS:

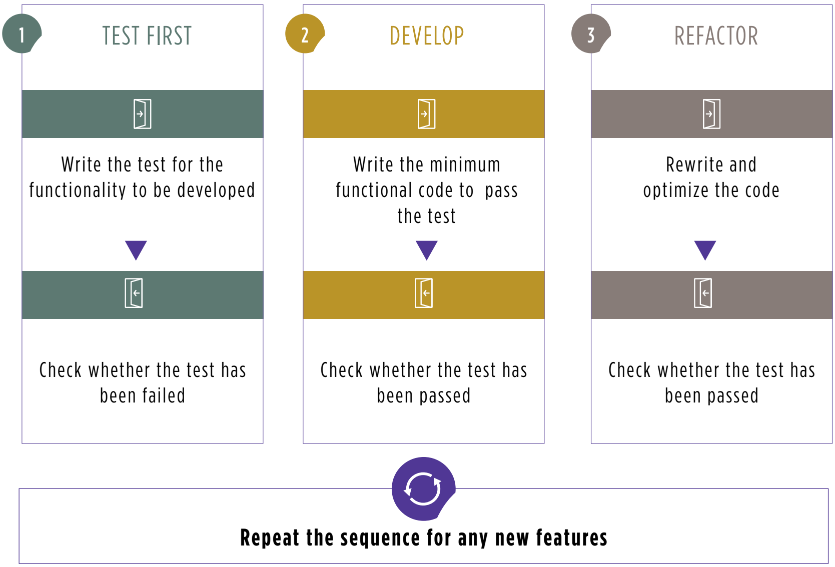
1. Write the function’s source code;
2. Write the unit-test source code for the function;
3. Run the unit test for the function in order to verify that it passes.

WITH A TEST-DRIVEN DEVELOPMENT APPROACH, DEVELOPERS DO THE OPPOSITE. HERE THEY:

1. Write the source code for the unit test for the function to be developed;
2. Write the function’s source code which allows the test to be passed;
3. Rework the code to improve it, while making sure that the test continues to be passed.

This method ensures that each function is associated with one or more unit tests, and it thus facilitates the non-regression tests while, at the same time, detailing the specifications, because the test describes the expected behavior.

TEST-DRIVEN DEVELOPMENT



1 TEST FIRST

Write the test for the functionality to be developed

Check whether the test has been failed

2 DEVELOP

Write the minimum functional code to pass the test

Check whether the test has been passed

3 REFACTOR

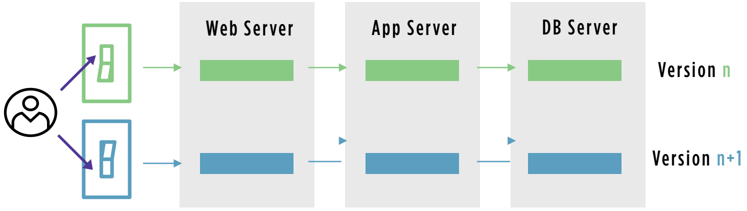
Rewrite and optimize the code

Check whether the test has been passed

Repeat the sequence for any new features

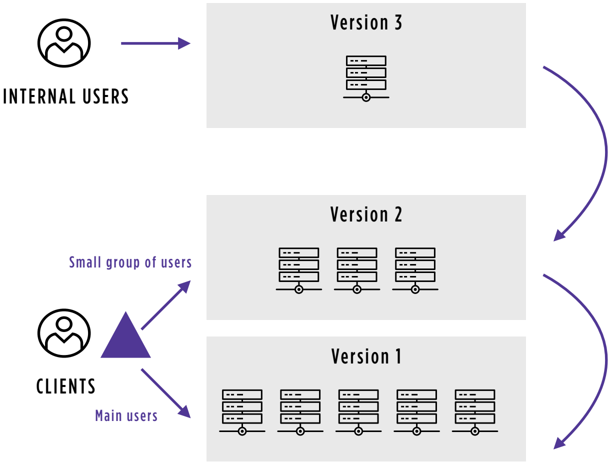
EXAMPLES OF DEPLOYMENT METHODS

BLUE/GREEN DEPLOYMENT



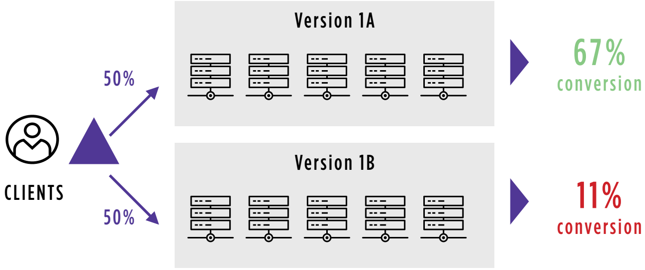
Deployment of an n+1 version on an environment parallel to the production version, n, and facilitated flip-flop from n to n+1, and from n+1 to n (rollback).

CANARY RELEASE



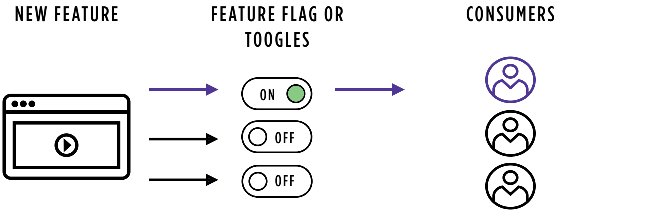
Deployment of several versions in parallel, some versions (or functionalities) are only open to certain populations (alpha then beta testers) before being rolled out (or not).

A/B TESTING



Deployment of two variants of the same version in parallel in order to compare the results and determine which one to retain.

FEATURE FLIPPING



Deployment of features that can be activated via an application interface. This enables certain functionalities not to be activated if the tests do not take place, without slowing down release. Can be combined with the Canary Release.

Functional tests (or recipes) should be mostly automated too, in order to check that there are no functional regressions during each iteration, and verify that newly created, or modified, functions are behaving correctly.

This is not trivial and involves being able to test function calls and human-machine interactions, in order to retrieve and analyze the elements (data or graphics) returned, and to maintain and develop all the data sets required to perform the tests.

Enabling this automation for an existing application could mean a significant workload, even higher than that for unit-test implementation, and could lead to an increase in delays and workloads to be managed.

TECHNICAL DEBT IS NOT A PROBLEM, PROVIDED IT IS MANAGED

More rapid construction of an application necessarily involves compromises. As a result, technical debt (in terms of code, infrastructure, management tools, etc.) is quickly and easily incurred, with a need to «pay it off» throughout the life of the project.

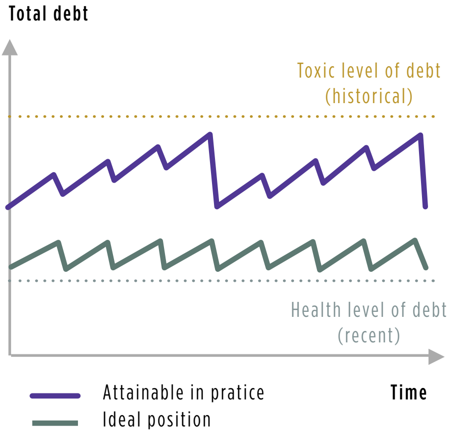
In a DevOps approach, it is important to monitor this technical debt using a KPI, and, as and when required - when the agreed limits are approached - to invest the time to pay it off. Conversely, there is no benefit in trying to get to a position of zero debt; aside from the risk of over engineering, doing this puts a real brake on Agility.

THE ANONYMIZATION OF DATA

In certain sectors (in particular, banking and insurance) the need for data to remain anonymous is becoming increasingly important. In contexts like this, the implementation of DevOps should ensure that the automation put in place is capable of anonymizing the live data used in functional testing.

Currently, this involves a considerable effort because, as yet, there are no tools on the market that are sufficiently mature.

However, studies show that a DevOps approach reduces the time spent on unplanned work (bugs), or code recovery, by up to 22%.

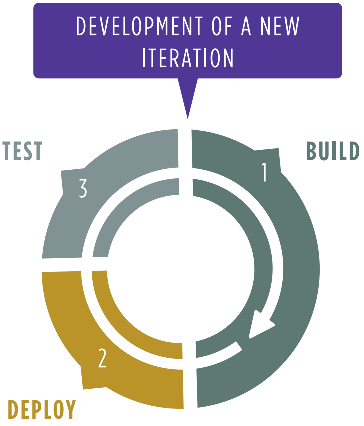


* Puppet & DORA 2016 State of DevOps Report

PILLAR II

CONTINUOUS DELIVERY FOR AUTOMAGIC DELIVERIES

Continuous Delivery (CD) is an automated software construction chain. It is a set of processes, tools, and techniques to manage application deliveries, from the production of code, through build, deployment, testing, and packaging, to the delivery of functionality... The aim? To increase the frequency and speed of deliveries in a reliable way—both rapidly and continuously.



Continuous Delivery is normally considered to be based on two automation chains:

- / **The Continuous Integration (CI) chain, targeted at development** and integrating the processes of build, measurement of technical debt (code quality), unit tests, and user acceptance;
- / **The Continuous Deployment chain,** which extends the CI chain by automating the provision of infrastructure environments and application deliveries. This extension is supported by Ops using methodologies and tools that are almost identical to those of Dev. Ops thus manages the consumption of infrastructure elements, configuration management, and application deployment.

Therefore, the entire chain is automated until going into production. Before release, Ops must verify that a number of prerequisites have been met, such as the conformity of the application with the rest of the IS, that there are sufficient resources available within the relevant teams to respond to incidents, the availability of infrastructure, the timeliness of deploying a new version, etc.

Once this is confirmed, all Ops then has to do is to trigger the deployment of the application (i.e. "push-button" deployment).

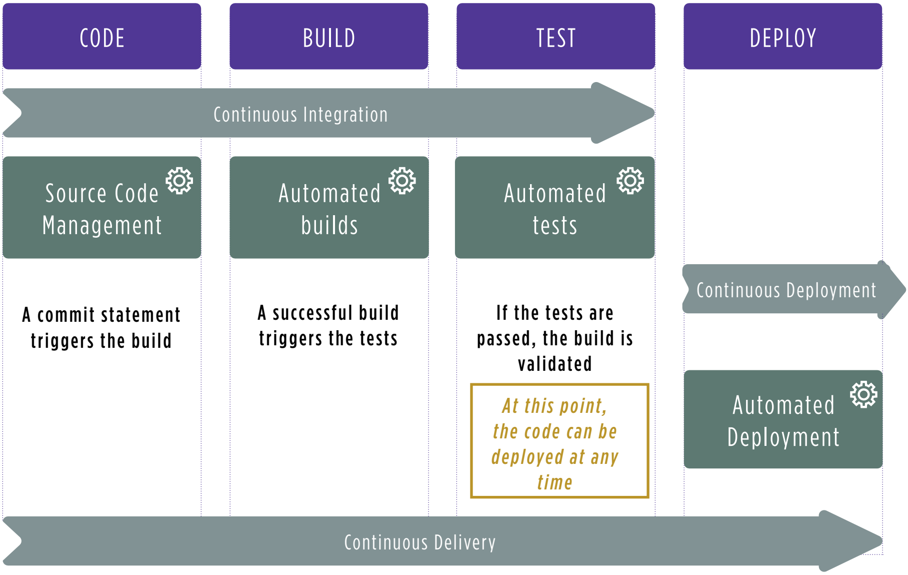
Conversely, deployment on other environments (such as non-production, or pre-production) can be completely automated.

Once a certain level of maturity has been achieved, it is entirely possible to envisage automatic, end-to-end deployment (c.f. diagram opposite).

CHOOSE TOOLS ACCORDING TO THE CIRCUMSTANCES, NOT THE LABEL

Today, the tools for the Continuous Delivery chain are quite heterogeneous: given the burgeoning market for DevOps tools, each function has its tool, which can be chosen from a myriad of solutions.

The first reason for this is that there is no clear, common definition of DevOps, which allows many vendors to apply the DevOps label to any tool related to software design, a software factory, configuration management, or any other form of orchestration («DevOpswash»).



CODE

BUILD

TEST

DEPLOY

Continuous Integration

Source Code Management

Automated builds

Automated tests

Continuous Deployment

A commit statement triggers the build

A successful build triggers the tests

If the tests are passed, the build is validated

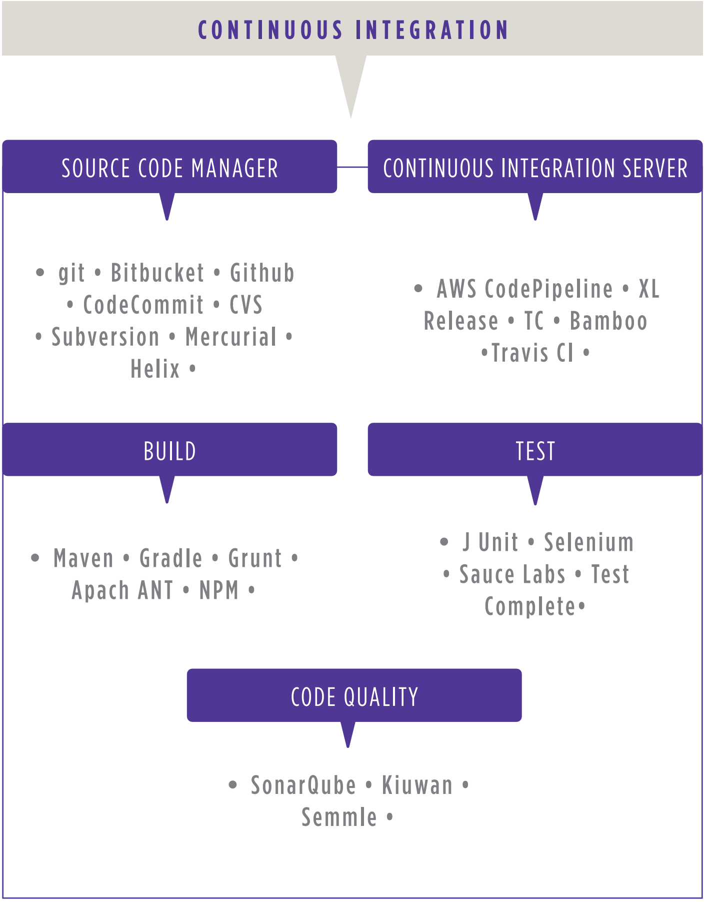
At this point, the code can be deployed at any time

Automated Deployment

Continuous Delivery

Moreover, considerable effort is going into extending the scope of many tools beyond their traditional technico-functional domains, which makes the actual contribution of the various tools more complicated to discern, while still not producing true, end-to-end software suites.

To find a way through this complexity, we recommend classifying tools as shown in the following matrix:



CONTINUOUS INTEGRATION

SOURCE CODE MANAGER

• git • Bitbucket • Github • CodeCommit • CVS • Subversion • Mercurial • Helix •

CONTINUOUS INTEGRATION SERVER

• AWS CodePipeline • XL Release • TC • Bamboo •Travis CI •

BUILD

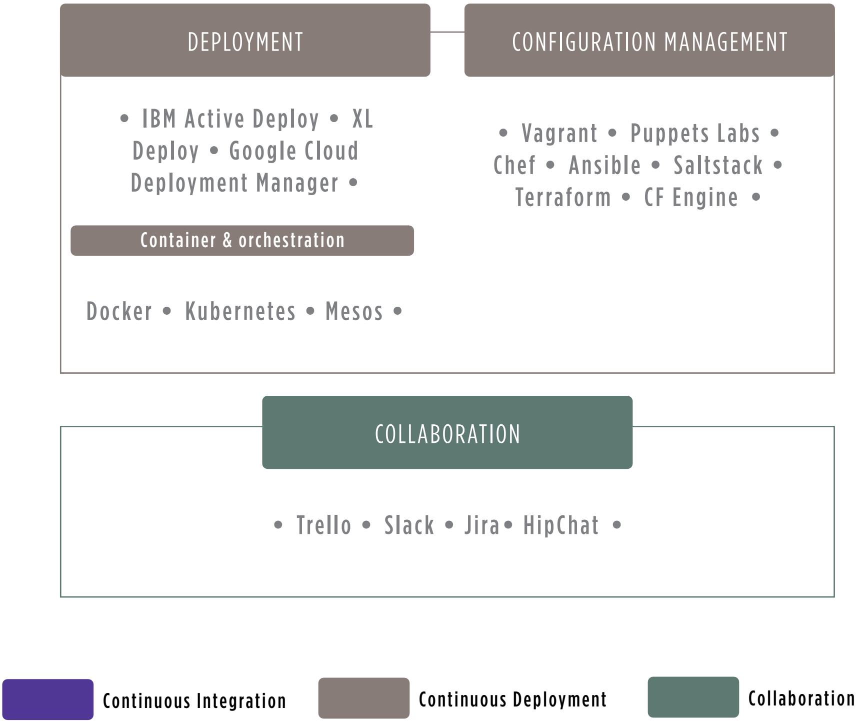
• Maven • Gradle • Grunt • Apach ANT • NPM •

TEST

• J Unit • Selenium • Sauce Labs • Test Complete•

CODE QUALITY

• SonarQube • Kiuwan • Semmle •



DEPLOYMENT

• IBM Active Deploy • XL Deploy • Google Cloud Deployment Manager •

Container & orchestration

Docker • Kubernetes • Mesos •

CONFIGURATION MANAGEMENT

• Vagrant • Puppets Labs • Chef • Ansible • Saltstack • Terraform • CF Engine •

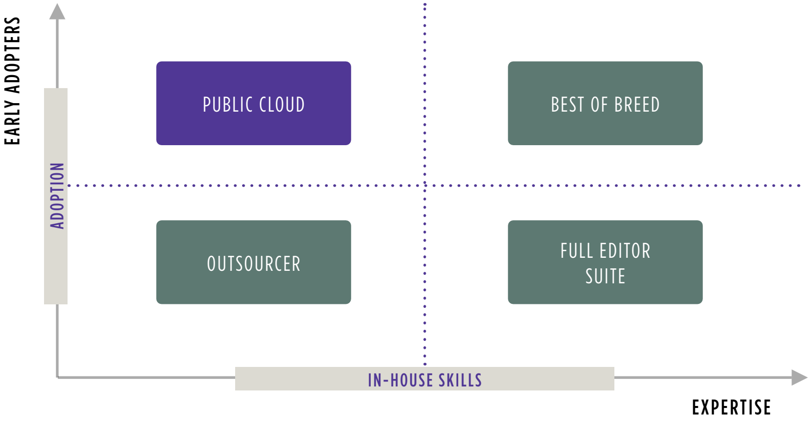
COLLABORATION

• Trello • Slack • Jira• HipChat •

🟪 Continuous Integration 🟫 Continuous Deployment 🟩 Collaboration

The present lack of consensus on the implementation of DevOps currently prevents the software industry from offering dedicated solutions. However, these solutions will emerge in the coming years, which will require adherence to their particular modes of operation (and, therefore, the associated practices).

The type of tools to choose will depend on a company's specific circumstances: if it is an early adopter, it will prefer a best-of-breed model or a solution, based on a public cloud with its associated tooling; if it is more of a "follower", it will be able to choose an integrated suite from a supplier in the market or an outsourced, turnkey solution.



Lastly, some tools naturally have to be centralized while others can be instantiated on a team-by-team basis, with each one maintaining control of the frequency of updates for the tool in question, or managing some specific aspects of configuration.

Generally, tools that are centralized and common to all are:

/ Source/artifact management tools (libraries, configuration files, etc.);

/ Test coverage management tools;

/ Tools for publishing analyses of results.

While the tools that lend themselves to being instantiated by individual teams may be:

/ Applications builders;

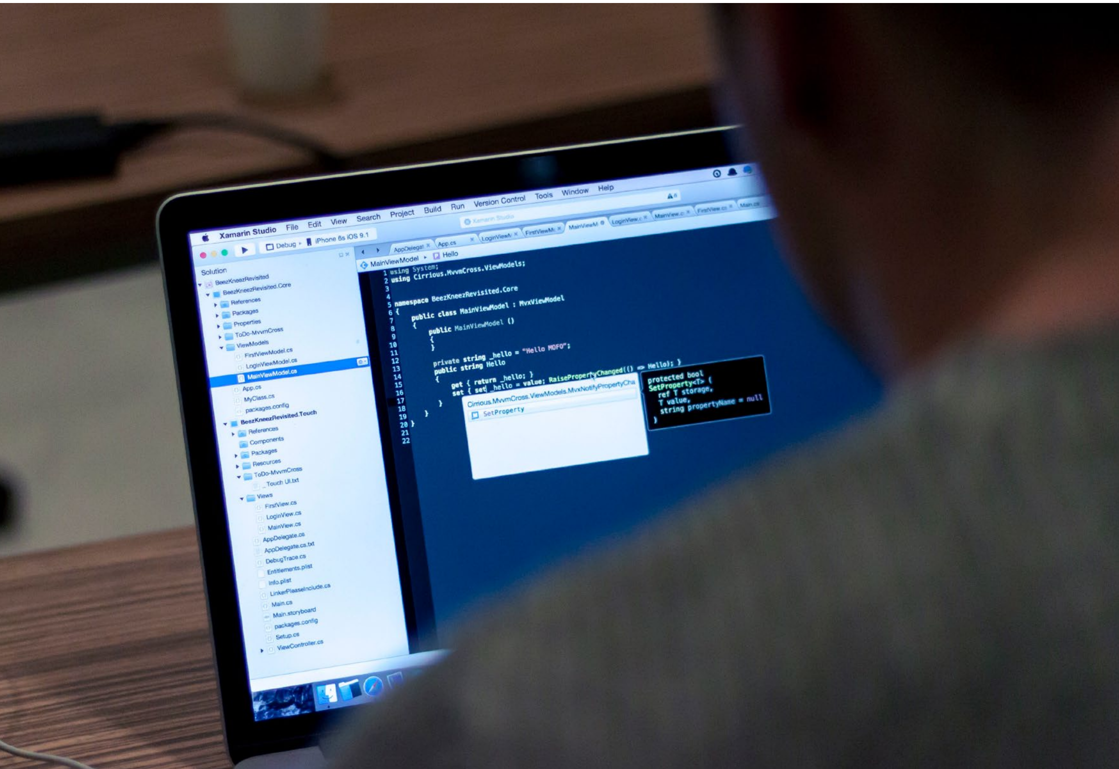
/ Tools for unit testing, code coverage, and compliance with coding standards;

/ Deployment tools (configuration management and application deployment).

PILLAR III

THE INFRASTRUCTURE AS CODE, AN INFRASTRUCTURE DRIVEN BY THE APPLICATION CODE

Infrastructure as Code (IaC) is the ultimate stage of Agility and infrastructure commoditization.



```
using System;
using Cirrious.MvvmCross.ViewModels;


namespace BeezKneezRevisited.Core
{
    public class MainViewModel : MvxViewModel
    {
        public MainViewModel ()
        {
        }

        private string _hello = "Hello MOFO";
        public string Hello
        {
            get { return _hello; }
            set { set| _hello = value; RaisePropertyChanged(() => Hello); }
        }
    }
}
```

For several years now, the market has been making considerable progress in moving toward the automation and consumption of infrastructure: we now commonly talk about "as a service", «on demand", "catalogs of services", "the Cloud," "API," and so on. Infrastructure as Code takes this concept further still.

The purpose of IaC is to enable the automatic creation, and configuration, of a complete execution environment (both infrastructure elements and middleware), through the application code. Developers manipulate the infrastructure in the application code itself, in the same way as functionality.

IaC therefore allows the infrastructure to be defined in a new way: it becomes simply a form of computing power, scalable without difficulty, and manipulable by the Devs through normal application code languages.

BEGIN MAKING INFRASTRUCTURE AGILE—BY AUTOMATING IT

Of course, infrastructure automation remains a prerequisite for IaC: in order to manipulate infrastructure through application code, it must be exposed through programmatic interfaces (APIs).

This automation can be done in three steps, which correspond to three levels of maturity.

Implementing a state-of-the-art IAC is not a prerequisite for DevOps: you can begin with an initial set of projects by automating only part of the infrastructure and moving toward end-to-end Infrastructure as Code through successive iterations.

1 **The first step - providing the envelope of the Virtual Machine - is often the simplest.** From a DevOps perspective, however, final barriers to complete automation (often the configuration of the network) must be removed, and the environment provisionable in a single click.

2 **The second step - deploying middleware - becomes immediately more complex.** One way to simplify it is to create Virtual-Machine models with pre-installed middleware; but this method soon reaches its limits (as a result of difficulties in managing the life cycle of the model, adding new middleware, etc.). It is therefore preferable to be able to directly provision the required machines and middleware in an automated fashion, perhaps even using defined application topologies (the deployment of a set of elements according to a previously defined scheme).

3 The third step is often the most complex, as it requires **interaction with other elements of the infrastructure** (firewalls, load balancers, exchange gateways, etc.) in order to provide the application **with a complete execution environment.**

These three steps can be carried out sequentially, or at the same time, depending on the extent to which the infrastructure has been automated.

There is also a need to move toward automation of infrastructure services: backup, technical and application supervision, scheduling, and security. The aim here is to ensure that resilience, security, and operational services are directly managed in the application by the developers themselves; it is, therefore, important that developers can subscribe directly to these services through an API.

PAAS - AN ACCELERATOR

PaaS (Platform as a Service) tools can be powerful accelerators, if you are able to meet the challenge of integrating them with the IS, and, in particular, the exploitation chain.

We are wary of private PaaS solutions (developed in-house or solutions from software editors, deployed on-premises) which are often not very mature, and which simply do not include solutions to the issues associated with infrastructure services (backup, supervision, network, and security).

SIMPLIFY THE INFRASTRUCTURE TO GIVE DEVELOPERS AUTONOMY

Mere automation of this process is not sufficient to qualify it as Infrastructure as Code; the deployment must be accessible via an API and be able to define, in the application's source code, the infrastructure required to make the application function:

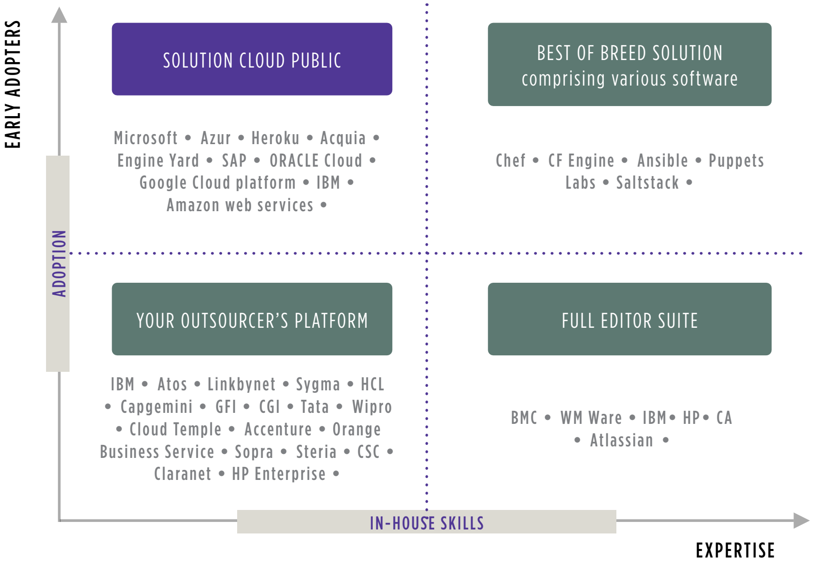
- The number of servers and associated middleware;
- Application services to be deployed (memory cache, message queue, etc.);

This must make it possible to simplify the infrastructure in order to offer developers straightforward computing power only.

USE THE PUBLIC CLOUD WHEN A PROJECT STARTS WITH A BLANK SHEET OF PAPER

Tools are, above all, a matter of context. There are numerous tools: from ones designed for a particular function only, to the complete suites offered by large publishers, and Pure Player cloud solutions; solutions exist for all contexts.

Thus, an organization with strong, internal technical skills will use a different strategy to a company that typically relies on external resources. Similarly, a company willing to take risks to differentiate itself (an «early adopter») will not use the same approach as a company that values stability and seeks mature solutions to safeguard the proper functioning of its business activities.

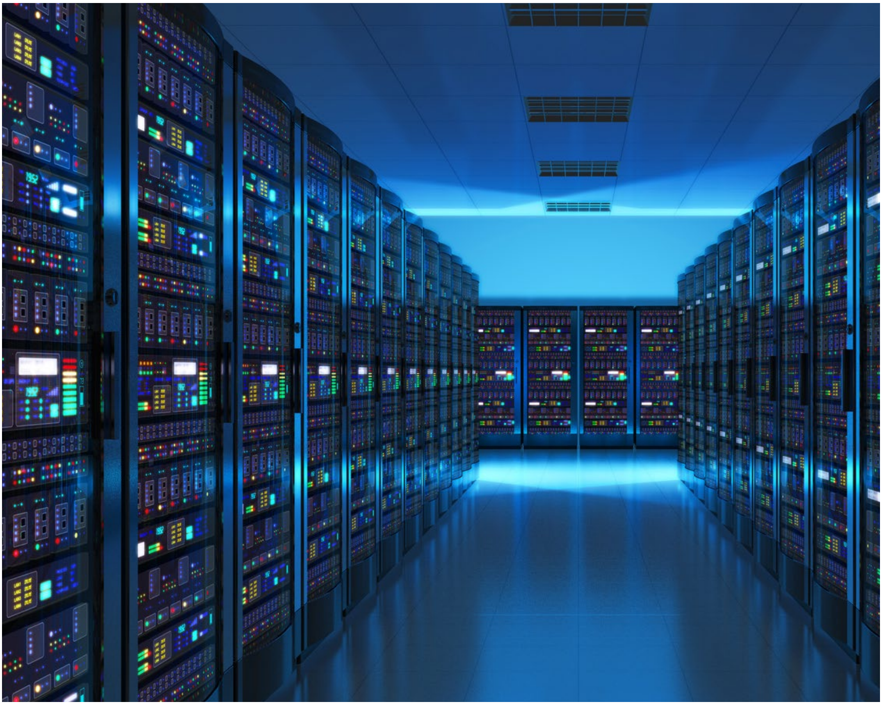

EARLY ADOPTERS
ADOPTION
SOLUTION CLOUD PUBLIC
Microsoft • Azur • Heroku • Acquia • Engine Yard • SAP • ORACLE Cloud • Google Cloud platform • IBM • Amazon web services •
BEST OF BREED SOLUTION comprising various software
Chef • CF Engine • Ansible • Puppets Labs • Saltstack •
YOUR OUTSOURCER'S PLATFORM
IBM • Atos • Linkbynet • Sygma • HCL • Capgemini • GFI • CGI • Tata • Wipro • Cloud Temple • Accenture • Orange Business Service • Sopra • Steria • CSC • Claranet • HP Enterprise •
FULL EDITOR SUITE
BMC • WM Ware • IBM• HP• CA • Atlassian •
IN-HOUSE SKILLS
EXPERTISE

Whatever the circumstances, a Public Cloud approach is something to consider because the players associated with this type of Cloud are currently ahead of the market. Amazon, Microsoft, and Google have achieved a level of maturity that cannot be bettered by companies' internal teams, because this area is not a core activity for them. Therefore, if you are using a Private Cloud strategy, it makes sense to know why, and for which workloads, this strategy is the best choice.

Lastly, when designing an IaC, it is essential to standardize the infrastructure and, therefore, make choices:

/ Choose where to invest the effort to automate and mass produce the components that make sense.

/ Accept that workloads which do not fit with this standard will not be included. This also means that any new application must be designed to follow these standards.



PILLAR IV

BREAKING DOWN SILOS, AND DEVELOPING AGILE AND COLLABORATIVE WORK PROCESSES

The DevOps culture draws on Agile and Lean methods (empowerment, trust, respect, and transparent communication) as well as a business-centric approach: a catalyst for greater cooperation between the business functions, Development and Production.

This cultural change will not happen overnight, and it is best to spread it over two broad stages:

1 ON A PROJECT

- **Assemble a team that includes people with all necessary skills** (Devs and Ops, but also members from Architecture, Security, Compliance, etc.)
- **Refocus the Ops function on building new, automated services**
- **Make Dev accountable for ensuring the operability its products**

2 ENTERPRISE-WIDE

- **Broaden this way of working to most projects**
- **Propagate the DevOps culture widely by sharing issues and success stories**
- **Plan the organizational structure of the ISD and how the target operating models will evolve**

Achieving this requires a common, mutually-agreed language, and shared indicators and tools:

- Share a vision for «products» rather than «projects»: the teams can deliver products that will be used by end customers rather than simply being contributions to a lambda project.
- Develop the indicators so that developers are not assessed only on the quality of their code, or the frequency of their releases, but also on having a good understanding of the requirements of production (and vice versa for Ops).
- Leverage collaborative tools (such as Trello or Jira Agile) to facilitate and strengthen the link between Dev and Ops

“IT IS ALL ABOUT PEOPLE» #HUMANFIRST

A DevOps transformation is, above all, about people—human beings—and collaboration.

Therefore, the first obstacles to be removed will be those related to issues of collaboration: people who need to be convinced that things can be done better by doing them differently, unhelpful processes to be dismantled, objectives (including supplier objectives) to be reviewed, tools to be changed, etc. All these things can be achieved only by drawing on reliable, motivated, and highly-competent people with a solid capacity to adapt (Google talks about «Highly Skilled Engineers» in its «Site Reliability Engineering» (SRE) methodology).

Moreover, as in any change management project, there will be a requirement to communicate the successes and failures on the objectives, and the impacts on the organization and its business functions.

ORGANIZATIONS ARE ALSO DRAWING INSPIRATION FROM THE INTERNET GIANTS IN THEIR QUEST FOR AGILITY

Others are now emulating the practices pioneered by Google (SRE), Spotify, Amazon, and Facebook. These practices are moving things further along the road toward full Agility. They have led to the emergence of somewhat complex principles and methods aimed at developing company-wide Agility.

SPOTIFY

/ Spotify is pursuing a methodology whose main pillar is the autonomy given to its teams (known as feature teams). In order to maintain the integrity of its IS, Spotify has set up communities (known as chapters, tribes, and guilds) which bring together those with specialist knowledge of a particular topic to form a type of matrix-based hierarchy.

/ Recognized as being a state-of-the-art structure for Agile organizations, it nevertheless involves a methodology that is not well documented, and one that requires a high level of maturity in terms of Agility and team management.

SAFe
Scale Agile Framework

/ SAFe is a framework developed in 2011 by a team of multidisciplinary experts on Agile organizations. It was designed for traditional organizations (rather than internet Pure Players) to address complex programs involving numerous teams.

/ It is aimed at companies who have already mastered the Agile Scrum method.

SCRUM OF SCRUMS

/ The Scrum of Scrums is a technique that enables several Scrum teams to work together in a very straightforward way, without the need for a complicated framework.

/ While this technique does not help in developing Agility at enterprise level, it can be a step toward developing it at large-project scale.

	TRADITIONAL ORGANIZATION	AGILE ORGANIZATION
STRUCTURE	ORGANIZATIONAL AND FUNCTIONAL SILOS ● PART-TIME INVOLVEMENT IN PROJECTS ● COMPLEX, MULTILEVEL HIERARCHY	MULTIDISCIPLINARY TEAMS AND UNITS ● FULL-TIME INVOLVEMENT IN A TEAM AND ON TASKS ● FEWER HIERARCHICAL LEVELS
PROCESS	CENTRALIZATION OF MANAGEMENT DECISIONS ● MANAGEMENT PROCESS FOR MAJOR PROJECTS ● BIG BANG V-CYCLE AND CHANGE MANAGEMENT	CONFIDENCE AND DELEGATION ● CONTINUOUS CHANGE AND DELIVERY ● DEVELOPMENT OF MINIMUM VALUABLE PRODUCTS AND RAPID CLIENT FEEDBACK
PEOPLE	NUMEROUS INDIVIDUAL KPIS ● RISK AVERSION AND CONTROL CULTURE	MORE COLLECTIVE KPIS ● FLEXIBLE PROCESSES ADAPTED TO THE NEEDS OF THE BUSINESS

MOBILIZE YOUR CHAMPIONS
TO BEGIN THE TRANSITION TO DEVOP

IaC, Continuous Delivery (CD), Culture... Where should you start? How can you initiate a DevOps approach?

The transition from traditional methods to DevOps represents a real break in the organization of work. Its deployment is a progressive and delicate undertaking that requires a degree of human and technical investment not to be underestimated.

The early stages of the transformation are key to assembling the relevant players, setting out the rationale for future investments, and creating a dynamic of sustainable transformation. The path taken can, and must, make it possible to realize returns on investment from the very first stages of transformation.

A project-by-project route makes it possible to stage the effort over the long-term, and to quickly obtain gains that are both visible and measurable.

> “The path taken can, and must, make it possible to realize returns on investment from the very first stages of transformation.”

A DevOps deployment strategy can be broken down into three areas:

- The choice of initial scope
- The deployment of a platform[1]
- Skills development

CHOOSE A SHOWCASE PROJECT FOR THE TRANSFORMATION, WHICH IS BOTH AMBITIOUS AND SAFE-TO-FAIL

Projects must be chosen to gain commitment from the business functions from the start, something that can then be maintained for each subsequent project.

To achieve this, it is essential to select a scope such that it focuses on the key needs of the business functions, and for which the returns from a DevOps approach will be tangible, significant, and simple to implement.

In DevOps, an iterative approach, coupled with strong team involvement - from the business functions to production - makes it possible to quickly adapt to change by removing operational constraints. DevOps therefore fully guarantees a response adapted to the specific need, even if things are clarified - or change - as the project progresses. In any case, it is these projects or products, whose needs frequently change, that will allow a company to

[1] Platform means: the deployment of collaborative tools, Continuous Delivery platforms, and automation (Infrastructure as Code).

demonstrate that DevOps is a better and faster approach than conventional methods, including those that typically require the close involvement of Ops.

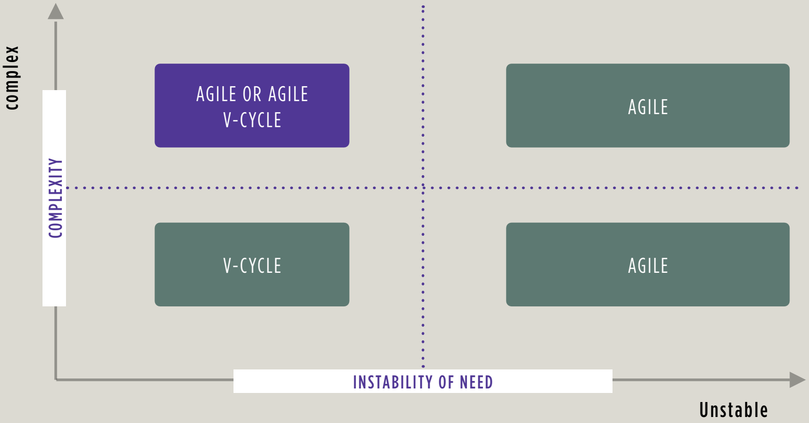
However, the chosen scope must be secure («safe-to-fail»). As the chance of failure is non-negligible, it is important to have an alternative plan in place that allows the acceptance of such risks.

ALL PROJECTS CAN BENEFIT FROM AGILE METHODS, BUT THE PRIZE IS APPLYING IT TO PROJECTS WITH RAPIDLY CHANGING NEEDS

Projects where the need is unstable (i.e. the end result is not predictable) are natural candidates for the Agile and DevOps methodologies because they benefit directly from their iterative approaches.

Simple and stable projects can be safely handled by conventional methods. In time, these too could also be addressed effectively by Agile methodologies.

Lastly, complex projects with stable needs can use either approach, provided the needs can be properly understood. If not, Agile methodologies are preferable.



START WITH EASILY AUTOMATED APPLICATIONS

The choice of initial projects must also consider the applications involved.

The amount of effort required to integrate an application into the continuous delivery chain is rarely negligible. On the Dev side, there is a need to transform the usual integration processes, and sometimes to change elements of the tools. On the Ops side, all delivery process automation scripts must be reviewed, in order that they can be managed using the same tools as those employed by Dev (versioning, build, testing, etc.). **Avoiding breakdowns in the delivery chain is at the heart of DevOps principles.**

To minimize initial investments, it is important to select applications for which the integration effort, in terms of maintaining a continuous delivery chain, is minimal.

Typically, software packages that require a quasi-manual intervention for a license key to be acquired, or the use of a graphical interface for installation, should be set aside in the early stages of the transformation.

Similarly, some applications, by nature, lend themselves very badly to the automation of tests.

Among the applications identified as suitable, it makes sense to target those whose delivery process has already been identified as a barrier in terms of time-to-market and quality of service issues, as well as those with relatively short development cycles. The greater the obstacle associated with such applications, the more obvious the benefits of a redesign. This can also serve as a rationale for making the investments required to automate application delivery.

PRIORITIZE THE CONSTRUCTION OF THE IAC AND CD PLATFORM AS A FUNCTION OF YOUR PAIN POINTS

Deployment of the platform will require significant investment over the long-term, though, paradoxically, this will not be very visible within the business. However, it is this platform that will sustain the success of the approach.

Worse still, if it is badly or half-heartedly implemented, the result can be to make Ops's pain points worse and endanger the success of the projects selected. This, in turn, risks the business abandoning the DevOps approach and returning to traditional methods.

The first projects will be those **to build the initial platform for Infrastructure as Code and Continuous Delivery**. The scope of this platform will be extended over time, project by project.

Each DevOps project should be seen as an opportunity **to draw on the budget to develop the building blocks that constitute the platform** - and each improvement to the platform as a way to reduce the effort required in subsequent projects. Prioritization is the key to maintaining this virtuous circle.

A good practice is to **set an automation goal for each project** which will be **processed** within a sprint, in the same way as application functionality. In some cases, if the project is sufficiently long and complex, it may be possible to see a return on investment even before it ends.

As for the automation of application deliveries, a particularly effective practice is to make use of projects to transform the building blocks already identified as pain points by the teams.

The deployment of collaborative tools, whether for project management or application code, must be treated as a priority. These tools are key to improving interactions between teams and aligning them toward common goals.

Lastly, some organizations, faced with the complexity of managing a very open and heterogeneous IS (multiplying, for example, the number of IaC platforms) will need to pursue automation until an orchestration layer can be put in place for their different IaCs (both internal and external, if they rely on external infrastructures) and an orchestration platform for delivery chains.

A Meta-Orchestrator or Business Process Management (BPM) may also be required to manage deployment processes on a set of applications, by ensuring consistency across the business. The deployment of a Meta-Orchestrator is the highest level of automation of deliveries - it automates the releases, taking into account all dependencies between different applications.

THE PUBLIC CLOUD
AN ACCELERATOR FOR DEVOPS TRANSFORMATION

Creating a new platform from an existing one can be long and costly.

The Public Cloud can therefore be a powerful, and low-cost, accelerator to launch a DevOps experiment using a suite that is already integrated.

All major Public Clouds (Microsoft Azure, Amazon Web Services, and the Google App Engine) offer their own suites. These are composed of IaCs and tools to build a Continuous-Delivery database, including a number of free tools (in particular for testing and code quality).

We strongly recommend using these initially in order to benefit more quickly from concrete results and enable a more ambitious transformation that does not prejudge what exists already or the processes already installed.

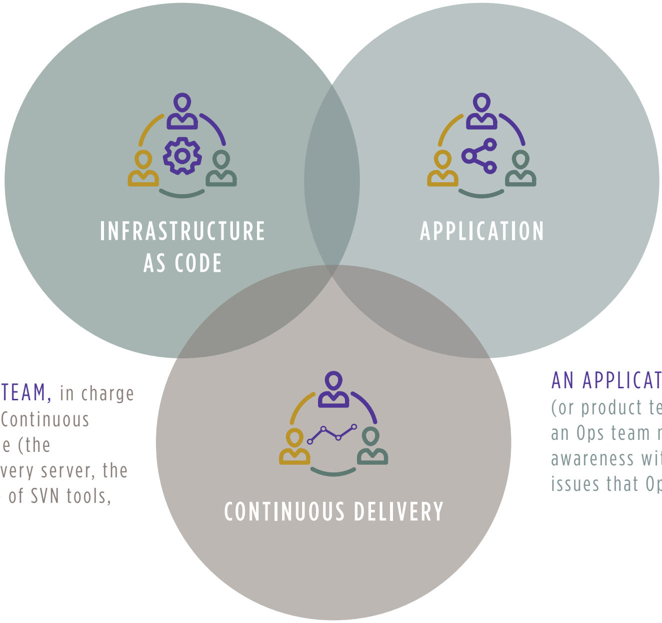
START WITH THREE «FEATURE TEAMS» OF CHAMPIONS TO BUILD THE PLATFORM AND FIRST APPLICATION IN DEVOPS MODE

In order to ensure the success of the first project and effective propagation of the associated practices, three teams must be brought together - with their members selected from those who can be considered champions in their respective fields:

In parallel with the first project, the IaC and CD feature teams will put in place the first elements of the platform needed to launch the projects in DevOps mode. The scale of these will be determined as a function of the speed at which these first elements are to be put in place and, naturally, the means that will be used to do this.

At the beginning, these teams should be based in the same place, or at least linked using effective collaborative tools. Similarly, they should undergo common training in order to forge strong links from the beginning.

AN IAC FEATURE TEAM, in charge of building the Infrastructure as Code platform (orchestration, API, and deployment templates).



A CD FEATURE TEAM, in charge of building the Continuous Delivery pipeline (the Continuous Delivery server, the putting in place of SVN tools, testing, etc.)

AN APPLICATION FEATURE TEAM (or product team), which includes an Ops team member, to build awareness within Dev about the issues that Ops faces.

THE ORGANIZATION WILL BE AGAINST YOU: YOU NEED TO PROVE THEM WRONG!

Transformation toward an Agile model will necessarily experience resistance to change. **It is vital that the success of the first project is communicated widely.**

It is also essential to set up, right from the initial project, quantified indicators that will help make the success of DevOps a reality.

The tools used in the process allow detailed analysis of the code quality, deployment times, numbers of application deliveries, numbers of bugs, etc. These aspects also serve as points of measurement, making it possible to demonstrate the attractiveness of the approach and the return on investment it brings.

It is a question of highlighting these gains and promoting them, in order to provide the rationale for the investments already made - and those required in the future.

THE MISTAKES TO AVOID

The first significant transformations to have been achieved in this area have already revealed some of the pitfalls to avoid:

- THE SYSTEM WILL BE AGAINST YOU: you must therefore stress the associated ROI.
- THE NEED TO DEMONSTRATE THAT YOU ARE RIGHT: to do this, you must measure the benefits of the transformation.
- YOU WILL EXPERIENCE FAILUREs: it is essential to accept failures and errors by taking an iterative, «test and learn» approach.
- “ONE SIZE DOES NOT FIT ALL:» each DevOps approach must be adapted to the circumstances and challenges of the company where it is being implemented.
- RESPECT FOR THE MONOLITH: it is essential to integrate the new approach with historic/legacy systems, and to respect existing processes, as appropriate. Don’t try to rebuild everything from scratch.

KEY SUCCESS FACTORS

Similarly, there are several key success factors that can be highlighted:

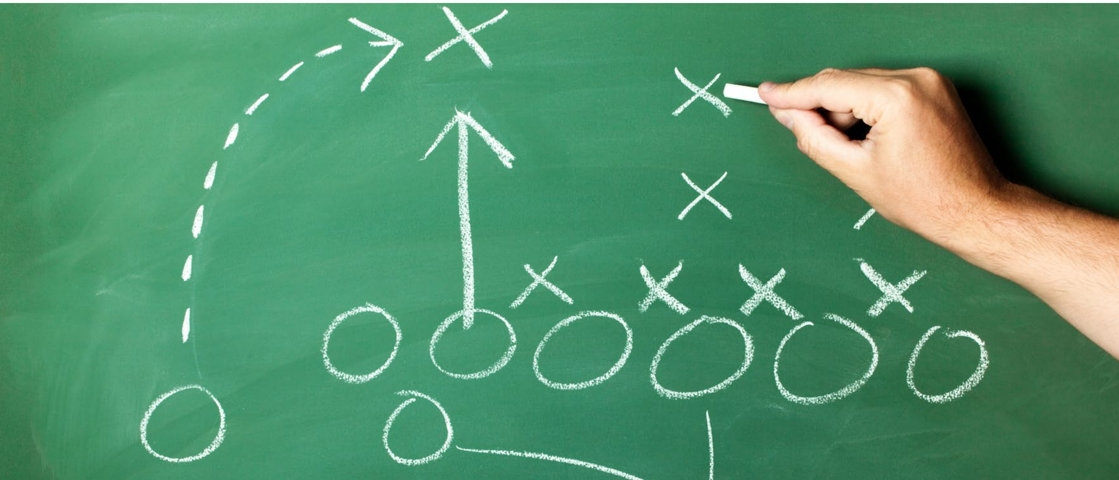
- SIMPLIFY: processes, infrastructure, etc. Everything must be standardized.
- AUTOMATE: your infrastructure, processes, and all repetitive, low added-value actions.
- RECRUIT HIGH-QUALITY ENGINEERS AND DEVELOPERS: turning them into properly qualified Ops with responsibility for the design of new services and IS building blocks.

TOWARD A NEW OPERATIONAL MODEL FOR THE ISD

Once the first projects have demonstrated the value of the approach, it will be a question of extending good practices throughout the business.

The members of the first teams must serve as coaches and ambassadors to disseminate good practice to the product teams that will progressively be formed within the ISD.

The teams that contributed to the construction of the IaC and CD platform must be the first building blocks of the competence centers that will develop the platform, according to the needs of the Product teams, and act as support to Ops.



To ensure a degree of «cross-fertilization» (the dissemination of knowledge between the teams), it is important to foster the construction of communities that share good practices between Dev and Ops, but also Architects and others.

These communities should also allow a common reference framework on Agile methodologies (project, management, community coordination, etc.) to be developed.

Lastly, this transformation must respect the legacy system and the work of those who will continue to maintain it. Minimizing the risks of this stage is, therefore, not only a question of managing the technical elements (such as interoperability, scalability, maintenance, etc.) but also the people aspects (for example, the attractiveness of positions, skills management, etc.). Later, it will also be necessary to develop it, either by refreshing it

THE PRODUCT TEAMS WILL DEVELOP TO CREATE MULTIPLE TEAMS, ACCORDING TO FUNCTIONALITY, WITHIN THE DEPARTMENTAL ISDS

Working as closely as possible with the business functions to adapt to their needs

Integrating Devs and Ops, and using IaC and CD, to deliver new functionality more rapidly and frequently

THE CREATION OF AN INTEGRATION AND SERVICES MANAGEMENT LAYER

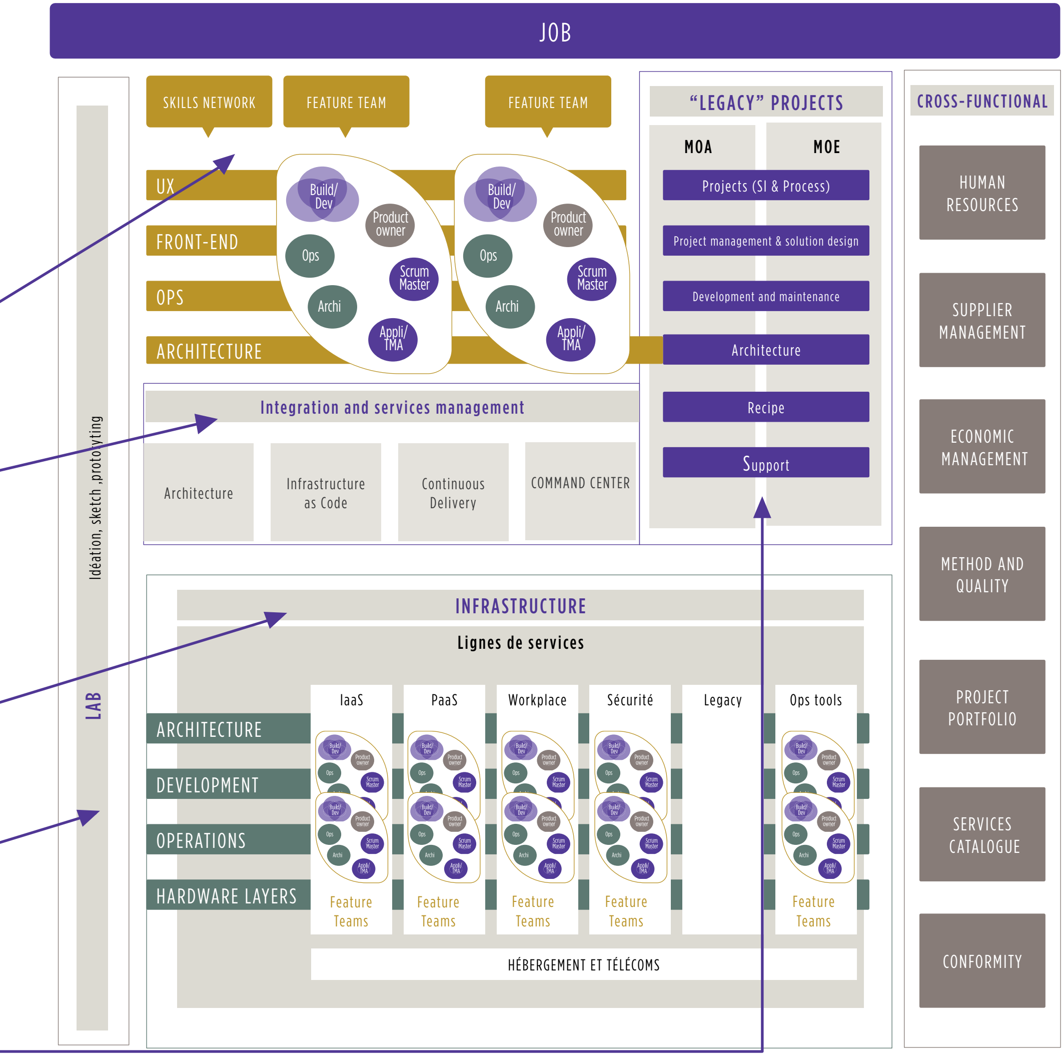
Drawing on IaC and CD competence centers made up of the teams involved in the first DevOps projects

Which also has responsibility for overall consistency (architecture, security, tools, etc.)

Simple, modular, INFRASTRUCTURE SERVICES LINES and automated operations, «secured by design»

Developing INNOVATION AND R&D CAPABILITIES to stay ahead of events and maintain close links to the business functions

BUT ALSO, sustainable integration with traditional operational models (with new technologies), or by transforming it completely (rewriting it piece-by-piece using an Agile philosophy).



SUPPORTING YOUR DEVS & OPS IN THEIR MODIFIED DISCIPLINES

The new organizational design necessarily affects the disciplines of operations and development. It is not a matter of turning Ops into Devs, or vice versa, but of helping the two disciplines evolve toward new practices.

Ops will evolve towards the **preparation of reusable and automated infrastructure elements**. **This will take place at** the same time as the Devs are upskilling **to ensure the development of the deployment scripts for each application, something that will be done using the same tools as the Devs.**

Development skills, and the knowledge of the configuration management tools, are particularly important here. Ops already has a culture of scripting. The challenge for them will be to move from old-style practices - scripts that are not reusable, rarely reviewed, and difficult to maintain - to using uniform practices and tools throughout the IT value chain.

Conversely, increasing the automation and quality of the deployments will lead to a reduction in the number of incidents and change requests that Ops have had to manage until this point.

For their part, Devs can no longer simply produce the application code. They will take account of the deployment constraints of environments, design application deployment models, and allow application materials to be varied - such that they can be deployed in any environment without additional management costs. From now on, they will be part of a complete continuous-delivery chain, integrating build processes (in contrast to their previous ways of working) and the measurement of technical debt (code quality), as well as automated unit and user acceptance tests.

This moves them toward:

/ **a better understanding of production and application-deployment models in order to «variabilize the application,» enabling it to be deployed differently depending on the environment in question;**

/ **following an approach known as «test-driven development,» to ensure a high level of quality, and non-regression to be controlled.**

/

CONCLUSION

While it is now imperative for companies to be more Agile and deploy applications more quickly, the DevOps practices that need to be put in place to do this will, nevertheless, have profound impacts on ISDs:

/ **Transformation of the way an application is designed** by making it more modular, and integrating infrastructure and operations into its code;

/ **The automation and «softwarization» of infrastructure,** and the provision of infrastructure services, via programmable interfaces contained directly in the code;

/ **The evolution of the Ops discipline toward development** and vice versa (and therefore the need to recruit differently, train people, and manage change).

/ **Different ways of working** with the business functions, through feature teams that involve all stakeholders and also modify **the ISD's organizational structure and operational model.**

You need to start now, progressively adopting these practices, using a test and learn approach, and drawing inspiration from the practices of the internet Pure Players. This requires a change of mindset: with a need to focus on results; think "product" rather than "project," and take smaller, but more frequent, steps. This is a profound change which will not happen in the space of a few months, but it will yield rapid results.

And with "NoOps" still a distant target, there is all the more need to move quickly. This concept means that the developers themselves are responsible for running applications, while Operations focuses on automation (and end-to-end supervision). This may seem like a utopian vision, but it is already a reality for giants like Amazon, who have derived real benefits by refocusing people on the added value they can offer.

WAVESTONE

www.wavestone.com