Technology

There is a better way

CULTURE DEVOPS VOL. 01

LES INGRÉDIENTS SECRETS D'UNE ORGANISATION DEVOPS ÉQUITABLE ET DURABLE

Notre Manifeste DevOps

1. Nous savons qu'il existe des pratiques et des technologies qui donnent du sens à notre quotidien.
2. Nous avons du plaisir à travailler ensemble pour atteindre un objectif qui soit commun à l'entreprise, à l'équipe et au collaborateur.
3. Nous contribuons avec fierté à la performance des organisations et à leur capacité à apporter de la valeur plus rapidement.
4. Nous sommes convaincus qu'un meilleur usage des infrastructures est toujours possible.
5. Nous sommes en permanence à la recherche de nouvelles pratiques et technologies pour anticiper la croissance et les enjeux de demain.
6. Tel l'Artisan, nous avons le goût du travail de qualité et de la transmission des savoir-faire.
7. Tel le Magicien, nous masquons la complexité de nos systèmes pour offrir de la simplicité à la demande.
8. Nous libérons le temps, esclave des tâches répétitives ou sans valeur ajoutée.
9. Nous embrassons le changement, nous transformons l'IT pour atteindre l'excellence.
10. Nous sommes DevOps.

INTRODUCTION À LA TRILOGIE

Les trois histoires de DevOps : Organisation, Qualité et Technologie

Si vous êtes resté caché dans une caverne ces huit dernières années, vous êtes peut-être passé à côté de la mouvance DevOps. Aujourd'hui, c'est un fait : le mot est sur toutes les lèvres de notre milieu IT. Mais le terme est souvent galvaudé, chacun y allant de sa définition, et il devient urgent de converger sur un concept partagé pour éviter que DevOps ne demeure qu'un *buzzword* de plus. Ainsi, il est de notre devoir d'apporter une pierre à l'édifice de cette convergence en vous livrant, en toute humilité, notre vision[1] de DevOps : celle qui nous passionne et qui berce notre quotidien.

Nous définissons DevOps comme un ensemble de pratiques qui visent à réduire le [2] 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[3] d'architecture. Pour nous, c'est aussi un métier, une passion. Et c'est la nature même de cette passion que nous désirons vous partager ici.

Nous avons pour habitude de regrouper ces pratiques DevOps en quatre piliers :

1. **Culture, méthodes et organisation :** on y retrouve les ingrédients secrets d'une organisation IT équitable et durable.
2. ***Infrastructure as Code* et les pratiques de qualité qui l'accompagnent :** ce pilier détaille les techniques d'automatisation du déploiement des applications et des infrastructures (réseaux, stockages, serveurs). Il vise à appliquer au monde des opérations informatiques, des outils et des pratiques issus du monde de l'ingénierie logicielle (gestion de versions du code source, tests automatisés, répétabilité des opérations, etc.)
3. **Patterns d'architecture *Cloud Ready Apps*[4]** : on regroupe ici les modèles d'architectures techniques qui permettent de tirer parti du *Cloud Computing* pour offrir davantage de résilience, de sécurité, d'exploitabilité et de scalabilité[5] aux applications.

4. **Processus d'intégration et de déploiement continus :** dans ce pilier, on industrialise la production logicielle et son processus de validation de la qualité. En se reposant sur l'usine d'intégration et de déploiement continus (CI/CD[6]), on automatise toutes les étapes de la chaîne de production : tests fonctionnels, tests de performance, tests de sécurité, tests de recette, déploiement de l'application sur un environnement, etc.

Quatre piliers donc, avec le choix éditorial de les traiter en trois volumes. C'est bien à vous que cette trilogie s'adresse, vous qui êtes informaticien, développeur, administrateur système, architecte, responsable des études ou de la production, CTO, CIO, CDO, CEO, CxO, DevOps amateur ou confirmé, ou juste curieux. Vous y trouverez notre vision, certes optimiste, de cette discipline sous trois angles complémentaires et indissociables :

- la culture et les modèles d'organisation équitables,
- la passion pour les technologies ouvertes,
- l'amour du travail de qualité et le *software craftsmanship*[7] appliqué à DevOps.

L'ouvrage que vous tenez entre les mains constitue le premier volet. Il présentera les courants de pensée qui donnent naissance aux traits culturels et aux modèles d'organisation propices à DevOps. Ce premier volume a pour objectif de balayer les fondements d'une organisation DevOps saine et équitable. Il donne des pistes pour transformer en profondeur son organisation dans ce sens. On y abordera également les origines dont s'inspire le mouvement, les objectifs de la méthode, la constitution, le fonctionnement et le passage à l'échelle des équipes DevOps, les principaux écueils à éviter, la méthodologie de transformation et les résultats que l'on peut espérer.

Asseyez-vous confortablement et commencez la dégustation par le chapitre qui vous plaira.

① Notre vision est celle d'OCTO Technology et de sa Tribu OPS qui œuvre depuis plus de cinq ans à la transformation DevOps de belles et grandes entreprises de France et de Navarre.

② *Time to Market* est le temps moyen écoulé entre l'émergence d'une idée et sa commercialisation. Dans cet ouvrage, on utilisera cette notion dans un sens plus large de *Time to Value* : le temps écoulé entre l'idée et sa mise à disposition pour l'utilisateur (dans notre contexte, il s'agira du temps de mise à disposition d'une fonctionnalité logicielle).

③ Un pattern est une solution répétable et réutilisable pour un problème identifié.

④ Voir notre publication sur ces architectures : https://www.octo.com/fr/publications/23-cloud-ready-applications

⑤ "La scalabilité (*scalability*, en anglais) désigne la capacité d'un logiciel à s'adapter à la montée en charge, en particulier sa capacité à maintenir ses fonctionnalités et ses performances en cas de forte demande" - source : Wikipedia.

⑥ CI/CD : *Continuous Integration / Continuous Delivery.*

⑦ Voir le manifeste *Software Craftsmanship* : http://manifesto.softwarecraftsmanship.org/

“La vocation, c’est le bonheur d’avoir pour métier sa passion.”

Stendhal

Au menu

Introduction

Pourquoi parler d'organisation dans DevOps ?

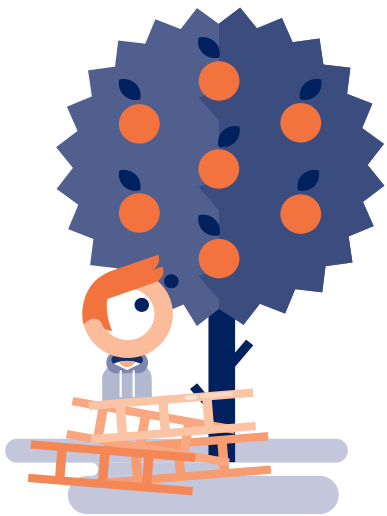
Nous avons la mission de partager avec vous l'essence, la substance et la philosophie qui font du mouvement DevOps quelque chose de Grand. Plus grand que le *buzzword*, plus grand que les intérêts propres des industries qui en font un business, plus grand que les tendances éphémères qui bercent notre métier depuis ses débuts.

"Le succès d'une démarche DevOps repose d'abord sur la culture, l'organisation et la méthodologie."

Cette mission est délicate et ambitieuse. Nous lisons encore trop de papiers et entendons trop de discours qui n'abordent DevOps qu'à travers le petit bout de la lorgnette. **Trop nombreux sont ceux qui limitent DevOps au sujet de l'automatisation.** Ce sont les mêmes acteurs qui, dans nos entreprises, prennent le sujet de la transformation DevOps par le seul spectre de l'outillage, et qui occultent *de facto* les autres piliers de la discipline.

Ceux-là pourront adopter les meilleurs outils d'*Infrastructure as Code*, de *Cloud Computing*, de conteneurisation ou de déploiement continu ; s'ils ne transforment pas en profondeur **leurs manières de travailler et leur organisation**, ils ne récolteront pas les fruits qu'ils peuvent espérer de DevOps.

Depuis 5 ans, nous accompagnons des moyennes et grandes entreprises françaises dans leurs transformations vers DevOps. Ces expériences nous ont aidés à nous forger la conviction suivante : le succès d'une démarche DevOps repose d'abord sur **la culture, l'organisation et la méthodologie.**



La volonté de mettre un terme au "mur de la confusion[8]", qui sépare les études (DEV) de la production (OPS), apparaît assez tôt dans la genèse du mouvement DevOps. C'est la prise de conscience qu'il existe un conflit, alors latent, entre DEV et OPS. Celui-ci impacte la **productivité de l'entreprise** et nuit sérieusement au climat de collaboration, voire au bien-être des collaborateurs.

La **culture DEV** est davantage tournée vers **le changement et l'innovation**. Elle se base sur l'envie de livrer rapidement de nouvelles fonctionnalités et s'arme pour cela des derniers frameworks et outils de productivité du marché. *A contrario*, la **culture OPS est orientée service** (notamment SLA[9]). Elle est garante de la stabilité des systèmes en production. Souvent considérés comme un centre de coûts, les OPS doivent vivre avec des budgets en baisse tout en faisant plus. Pour cela, ils cherchent à rationaliser les ressources et les technologies supportées. **Le "mur de la confusion" naît de ces conflits d'objectifs opposant innovation à rationalisation, changement à stabilité et culture DEV à culture OPS.**

Si le constat du "mur de la confusion" est largement partagé, il n'est que trop inégalement traité par ceux qui s'intéressent à DevOps. La transformation de l'organisation, la mise en œuvre de méthodologies et l'introduction de nouveaux traits culturels sont, selon nous, les seuls outils permettant de briser ce mur.

En tant que pratiquants de DevOps, nous partageons intimement la croyance que **le plaisir à travailler ensemble y est pour beaucoup.** Nous trouvons du sens dans le "faire ensemble", dans le partage, dans les relations humaines de qualité et par dessus tout, dans le plaisir que nous en retirons quotidiennement. Ce regain de sens, cette dynamique collaborative, cet entrain qui nous met en mouvement sont à votre portée, mais pour cela, prenez soin des **volets Organisation et Culture** lors de votre transformation DevOps.

Le mur de la confusion matérialise généralement une incompréhension profonde entre DEV et OPS. Elle peut se concrétiser par un éloignement physique des équipes (parfois un mur les sépare, mais bien souvent une rue voire des kilomètres) mais aussi dans des différences de langage, de terminologie, de culture, de rituels ou de rythmes de fonctionnement.

⑧ L'expression "mur de la confusion" a été rendue populaire en 2010 par Andrew Shafer et Lee Thompson dans l'article de blog fondateur intitulé *What is DevOps*? http://dev2ops.org/2010/02/what-is-devops/

⑨"Le *service-level agreement* (SLA) ou "accord de niveau de service" est un document qui définit la qualité de service, prestation prescrite entre un fournisseur de service et un client. Autrement dit, il s'agit de clauses basées sur un contrat définissant les objectifs précis attendus et le niveau de service que souhaite obtenir un client de la part du prestataire et fixe les responsabilités." - source : Wikipedia

Des points de vue différents...

DEV (Les développeurs)

RAPIDITÉ

- Livrer le plus **rapidement de nouvelles fonctionnalités**.
- Chercher à **innover**.
- Culture du **changement**.

BIZ (Le métier)

RAPIDITÉ QUALITÉ STABILITÉ COÛT

- Fournir le plus **rapidement** possible des services de **qualité** au client final.
- Obtenir une **qualité** de service sans faille.
- Contrôler **les budgets IT**.

OPS (La production)

STABILITÉ QUALITÉ COÛT

- Garantir la **stabilité**.
- Contrôler la **qualité** des changements.
- Chercher à **rationaliser**.
- Culture du **service** (SLA).

L'organisation comme fondation de DevOps

DevOps sans le volet organisation n'est pas DevOps. Le corollaire n'est pourtant pas vrai : l'organisation sans DevOps se porte très bien, merci.

Le mouvement DevOps n'a pour ainsi dire aucune antériorité sur ces sujets organisationnels. Nombreux courants de pensée se sont penchés, bien avant DevOps, sur des questions comme la productivité des organisations ou le bien-être au travail, par exemple.

"DevOps sans le volet organisation n'est pas DevOps."

Le mouvement DevOps n'a certainement pas l'ambition de révolutionner les disciplines traitant de l'organisation. Mais il s'inspire de modèles pragmatiques qui tirent leurs origines dans des courants divers.

La psychosociologie, la théorie des organisations, la dynamique de groupe

Ces sciences humaines et sociales apportent un éclairage anthropologique sur les comportements d'une organisation lorsque cette dernière est mise sous contrainte. Elles nous apprennent à **décrypter les enjeux du pouvoir**, les motivations, les relations de communication et **les rapports sociaux** qui régissent le quotidien dans nos équipes informatiques.

Lean, *Kanban*, "Juste à temps"

C'est dans les usines de Toyota que sont nées ces méthodes visant à améliorer la performance des flux de production et à réduire le *Time to Market*. La philosophie *Lean* s'inscrit également dans une vision long terme privilégiant le **développement des personnes** plutôt que les objectifs financiers à court terme. DevOps transpose bon nombre de ces principes au monde de la production logicielle.

Les méthodes et pratiques Agiles

Elles-mêmes très inspirées du courant *Lean*, elles sont fondatrices pour DevOps. Ainsi, les valeurs de l'agilité telles que décrites dans l'*Agile Manifesto*[10], sont parfaitement transposables à DevOps :

- Les individus et leurs interactions plus que les processus et les outils.
- Du logiciel qui fonctionne plus qu'une documentation exhaustive.
- La collaboration avec les clients plus que la négociation contractuelle.
- L'adaptation au changement plus que le suivi d'un plan.

⑩ *Agile Manifesto* : http://agilemanifesto.org/iso/fr/manifesto.html. ⑪ *Scaling Agile Framework* : http://www.scaledagileframework.com. ⑫ *The Unproject Culture* : https://files.app.net/2zwhkjS3J.pdf. ⑬ Henrik Kniberg, coach Agile et Lean chez Spotify® et LEGO®. ⑭ *Site Reliability Engineering: How Google Runs Production Systems* : http://shop.oreilly.com/product/0636920041528.do

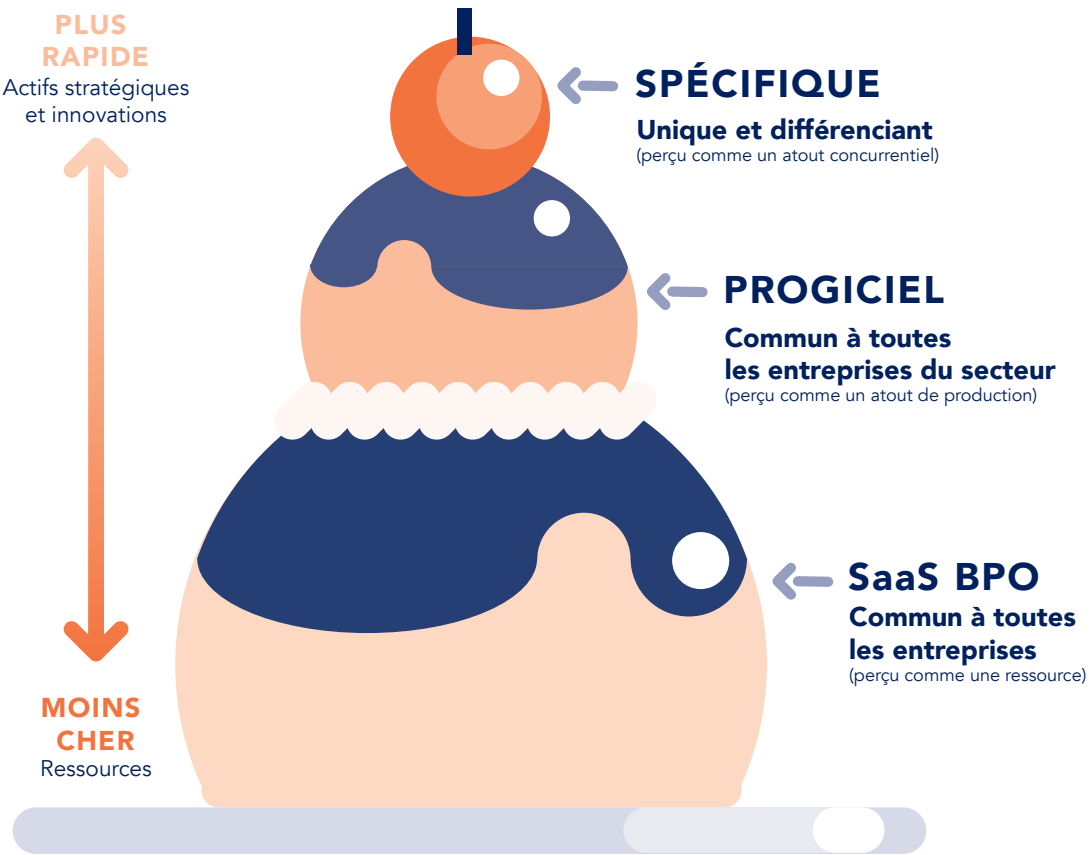
Toutefois, ces méthodes et pratiques se focalisent uniquement sur les phases de design et de *build* logiciel et n'abordent pas les phases de déploiement et d'exploitation (sur lesquelles DevOps trouve sa légitimité).

Les méthodes de passage à l'échelle de l'agilité (*Scaling Agile*)

Les méthodes Agiles s'intéressent au fonctionnement intérieur de l'équipe alors que des frameworks comme SAFe (*Scaling Agile Framework*[11]) décrivent une organisation permettant un déploiement des pratiques Agiles à l'ensemble de l'entreprise. DevOps s'inspire de ces modèles pour son passage à l'échelle. Un célèbre cas d'école intitulé *The Unproject Culture*[12] réalisé par Henrik Kniberg[13], décrit le cheminement réalisé par Spotify pour se transformer en ce sens.

En complément de ces courants fondateurs, DevOps trouve des sources d'inspiration dans les modèles organisationnels de startups dont certaines sont devenues des Géants du Web. On peut citer l'exemple de Google avec la création du métier de "*Site Reliability Engineer*[14]" (SRE). Il s'agit d'une discipline visant à créer des systèmes ultra-évolutifs et hautement disponibles en transposant des pratiques issues de l'ingénierie logicielle au monde des opérations. Autrement formulé par ses pionniers, SRE c'est : "ce qui se passe quand un ingénieur logiciel Google est en charge de ce qui s'appelait autrefois l'exploitation".

De manière générale, nous préconisons le déploiement de DevOps en commençant par des applications pas trop grosses, mais importantes et différenciantes.



Disclaimer : on vous aura prévenu...

Les pratiques DevOps s'appliquent aux petites et moyennes structures comme les startups, mais pas exclusivement. De grands groupes ont déjà franchi le cap et commencent à adopter – avec succès – tout ou une partie des pratiques que nous présentons dans les différents volumes de ce livre.

À ce stade, nous vous mettons en garde contre une application trop dogmatique ou systématique qui pourrait être faite de ces pratiques. L'implémentation, voire la pertinence de ces dernières dépendent fortement **des enjeux et du contexte** dans lesquels vous les appliquez. En effet, DevOps et ses pratiques ne sont pas **la solution unique à appliquer lorsqu'on est à la recherche d'améliorations**. Prenons l'exemple d'une grande entreprise disposant d'un progiciel RH pour gérer la carrière de ses salariés (GPEC) et d'une autre application développée spécifiquement pour offrir de nouveaux services différenciants à ses clients. Le premier système d'information dispose de faibles enjeux de *Time to Market* en comparaison du second. On peut ainsi aisément décider de n'appliquer en priorité une organisation DevOps qu'au cas de l'application différenciante.

L'organisation au service des objectifs de l'en+reprise et du collaborateur

Les modèles d'organisation ne sont que des moyens. Ils servent parfois des objectifs de réduction de coûts, d'autres fois de croissance ou encore contribuent à la résilience de la structure. Ces modèles d'organisation fusionnent ou découpent, grossissent ou maigrissent, rapprochent ou éloignent les groupes d'individus qui forment l'entreprise. Certains de ces modèles sont équitables car ils sont à la fois au service des objectifs de l'entreprise et du collaborateur. DevOps, par exemple, propose des modèles qui permettent de redonner au collaborateur de l'autonomie, de la responsabilité et de la clarté dans la finalité son travail.

BIZ, DEV et OPS
partagent les mêmes objectifs

BIZ, DEV et OPS partagent les mêmes objectifs

Il n'est pas rare de voir plusieurs mois se passer entre le moment où le métier pense une innovation et son arrivée effective sur le marché. DevOps apporte une réponse intéressante aux entreprises ayant pour enjeu d'**accélérer la mise à disposition de produits logiciels stratégiques sur le marché**. Pour arriver à cet objectif, la méthode propose de repenser à la fois :

- Les processus d'ingénierie logicielle et d'infrastructure pour améliorer leur productivité et ainsi réduire le *Time to Market* (TTM) et le *Time to Repair*[15] (TTR). Livrer plus vite du logiciel en production ne doit pas se faire au détriment de la qualité au risque d'accroître dangereusement le nombre de bugs et d'incidents livrés en production. C'est pourquoi nous dédions un volume complet de notre trilogie au sujet des pratiques de qualité dans DevOps.

- L'organisation pour la rendre plus réactive, adaptable et autonome. C'est alors que se brise le "mur de la confusion", réunifiant au sein d'une même équipe tous les acteurs concourant au succès du produit logiciel.

Le travail devient porteur de sens, l'équipe a désormais un cap, les objectifs sont clairs, mesurables et partagés : il faut fournir rapidement aux clients un produit logiciel de qualité, qui soit testé, stable en production, maintenable et évolutif. L'entreprise n'en est que plus compétitive et les membres de l'équipe y trouvent également leur compte.

⑮ *Time to repair* : il s'agit du temps moyen pour réparer le composant défaillant d'un système.

Tous les contextes sont-ils favorables
au DevOps ?

Tous les contextes sont-ils favorables au DevOps ?

Pragmatisme. C'est le maître-mot lorsqu'il faudra décider du périmètre de sa transformation DevOps. Certaines technologies d'une autre époque (mainframe, z/OS, AIX, par exemple) ou encore certaines applications non stratégiques (comme la gestion de la cantine) peuvent assez naturellement être considérées comme de mauvaises candidates à DevOps. Pour autant, la frontière entre les applications "*DevOps-Ready*" et celles qui ne le sont pas n'est pas si marquée que cela.

DevOps n'est pas une norme. Ce n'est pas non plus une méthodologie ou un processus. Il ne s'agit clairement pas d'une suite d'outils. DevOps, c'est à la fois un courant de pensée, un agglomérat d'opinions, de convictions et d'expérience acquises. C'est aussi et surtout, un moment de notre histoire où ont convergé des assemblages de pratiques variées – nouvelles et anciennes – servant la cause d'une même idée : réinventer notre manière de concevoir et opérer un produit logiciel de qualité. Dans DevOps, il est question d'équité, de souplesse, mais aussi de pragmatisme (cette forme d'intelligence qui sait s'adapter aux dures réalités du terrain).

Alors, nous pourrions formuler ce premier constat : **tous les produits logiciels peuvent tirer parti d'au moins une pratique DevOps. Pour autant, il n'est pas souhaitable que tous les produits logiciels bénéficient de toutes les pratiques DevOps**.

Retour sur investissement de DevOps (ROI)

Retour sur investissement de DevOps (ROI)

La plupart des grandes entreprises sortent à peine d'une époque considérant le système d'information comme une entité support des métiers, à faible valeur ajoutée. Autrement dit, une époque où l'informatique était uniquement perçue comme un centre de coûts. Le modèle d'organisation alors appliqué dans ce contexte visait des objectifs de rationalisation et d'efficacité opérationnelle. Concrètement, la mutualisation des ressources et des expertises techniques dans une entité centrale d'ingénierie d'infrastructure et de production permettait d'optimiser l'usage de ces ressources et d'espérer une réduction des coûts d'investissement et d'exploitation.

Ce modèle rencontre des limites à l'heure de la "digitalisation", où une prise de conscience collective accorde un **nouveau potentiel de valeur dans la technologie**. Le produit logiciel redevient stratégique pour l'entreprise qui le conçoit comme un différenciant de marché. L'entreprise cherche alors à se doter des moyens lui permettant de **produire du logiciel plus rapidement et de meilleure qualité**. Les pratiques d'organisation de l'Agilité et de DevOps sont ainsi utilisées comme des moyens au service de cette quête de productivité.

Pour autant DevOps peut-il servir des objectifs de réduction de coûts ?

Nos expériences de transformations DevOps nous permettent de faire les constats – parfois paradoxaux – suivants :

- Contre-intuitivement, DevOps (et l'automatisation que cela engendre) ne réduit pas la masse salariale, mais tend plutôt à l'augmenter en dédiant à chaque équipe les experts dont elle a besoin. Ce point est détaillé plus tard dans le chapitre.

- La transformation vers un modèle DevOps a un coût d'entrée non-négligeable (acquisition de compétences, investissement dans des plateformes cloud et outils, changements culturels et organisationnels).

- La multiplication des environnements de test peut engendrer une augmentation des coûts d'acquisition de licences logicielles et/ou de ressources d'infrastructure.

• La productivité logicielle augmente : le temps de fabrication d'une fonctionnalité est réduit, l'équipe DevOps produit davantage de fonctionnalités dans un temps identique.

• L'investissement dans la qualité logicielle, l'augmentation de la couverture de test, le *software craftsmanship* n'augmentent pas les coûts sur les moyen et long termes (voir le schéma qui suit illustrant l'évolution du coût de la non-qualité).

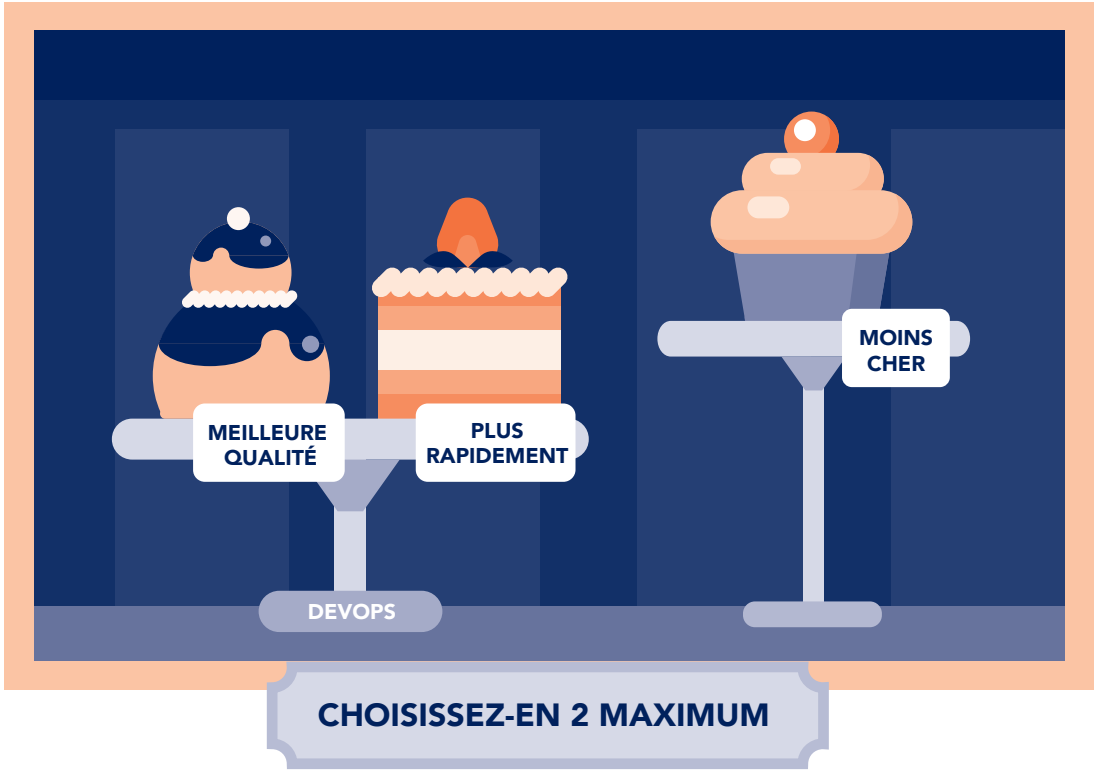
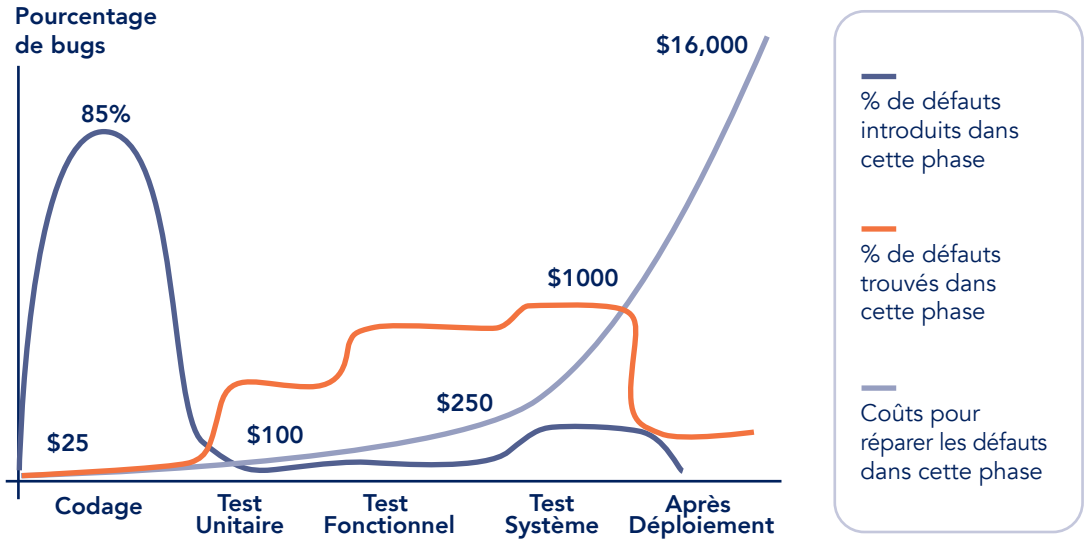


Illustration du coût de la non-qualité justifiant l'investissement en tests
(Applied software measurement, Capers Jones, 1996)



En synthèse, lors d'une transformation vers DevOps, l'investissement initial et les budgets de fonctionnement peuvent augmenter au profit d'**une productivité logicielle accrue**. Nous avons la conviction que le calcul d'un ROI plus détaillé est illusoire tant **il est impossible de calculer le gain d'un TTM accru** et la **valeur d'usage d'une fonctionnalité**. Dans un objectif de comparaison de coûts, cela impose également de connaître sa propre structure de coûts, ce qui n'est pas toujours acquis ou évident à faire.

Insistons sur ce point : le modèle d'organisation DevOps n'est pas un moyen approprié pour réduire les budgets de fonctionnement IT sur le court terme.

Aucun modèle d'organisation n'est parfait. Toutefois, certains modèles sont utiles pour servir un objectif tactique de l'organisation à un instant et un contexte particuliers. Dans notre cas, **le modèle d'organisation DevOps se focalise sur le TTM et la qualité**.

Alignement des objectifs :
impacts sur l'épanouissement personnel

Alignement des objectifs : impacts sur l'épanouissement personnel

Avez-vous déjà fait la rencontre d'un "startuper" ?

Peut-être avez-vous croisé son regard pétillant lorsque ce dernier faisait l'éloge de son produit, de son architecture ou de son équipe. Peut-être avez-vous eu le sentiment qu'une flamme imperceptible animait ce professionnel passionné et fier.

Peut-être l'avez vous déjà ressentie, vous aussi ? Il n'est pas rare de surprendre cette forme d'énergie positive dans les organisations produit de taille réduite, comme les startups ou les équipes DevOps.

Quel est le mystère de ces équipes où l'on voit naître et grandir une dynamique donnant l'impression d'une osmose parfaite entre l'individu et l'équipe ? Cette cohésion, cette complicité, voire cet esprit de corps viennent-ils d'une recette que nous pourrions répliquer ailleurs ?

Sans avoir la prétention de répondre à cette question, nous vous proposons (dans un tableau page suivante) quelques pistes de réflexion non-exhaustives, issues du fruit de nos expériences.

Penchons-nous sur ce dernier point : le *Why*[16] de l'équipe.

Il s'agit ici de faire émerger collectivement un manifeste qui synthétise la mission de l'équipe. Parfois appelé *Massive Transformation Purpose*, ce manifeste formalise les ambitions, l'audace et les aspirations collégiales de l'équipe. Ce document qui est fondateur pour **le groupe devient porteur de sens, source d'inspiration et renforce le sentiment d'appartenance.**

Voici par exemple le Why que partagent les Octos :

"Nous croyons que l'informatique transforme nos sociétés. Nous savons que les réalisations marquantes sont le fruit du partage des savoirs et du plaisir à travailler ensemble. Nous recherchons en permanence de meilleures façons de faire."

[16] Concept développé par Simon Sinek, *Start with Why* : https://startwithwhy.com/

Voici six ingrédients qui nous semblent contribuer à l'épanouissement de notre équipe :

LES SIX INGRÉDIENTS DE L'ÉPANOUISSEMENT D'ÉQUIPE	ILLUSTRATION
La boucle de feedback rapide entre le moment de l'action de son travail et le constat de l'impact de sa contribution sur l'objectif commun.	*"Je vois rapidement le fruit de mon travail".*
La satisfaction de contribuer à son niveau au succès du produit, de l'équipe et de l'entreprise.	*"Je vois concrètement l'impact de mon travail sur le produit, le chiffre d'affaires, etc.".*
Le temps accordé aux relations humaines de qualité : les espaces de rencontre, les rituels extra-professionnels, la connaissance des membres de l'équipe entre eux.	*"Mes collègues ne sont pas des numéros, j'ai pris le temps de les connaître".*
La culture managériale 3.0 : la transparence, le droit à l'erreur et la liberté d'entreprendre, le partage des savoirs.	*"Je sais que je suis épaulé, même si je me trompe. Je n'ai peur ni d'essayer, ni de rater, ni de le raconter".*
La culture d'équipe : les codes sociaux de l'équipe, les rituels, le vocabulaire partagé.	*"Je comprends mes camarades, même ceux qui ne font pas exactement le même métier que moi, je sais pourquoi leur travail est important".*
La formalisation de la raison d'être de l'équipe (Le *Why*) qui donne le sens du travail commun.	"Je sais à quoi je participe et pourquoi je le fais".

DevOps : vers l'autonomie et la responsabilisation des équipes

L'autonomie et la responsabilisation des équipes sont les secrets d'une transformation organisationnelle réussie vers DevOps.

Le Larousse définit l'autonomie comme la "capacité de quelqu'un à être autonome, à ne pas être dépendant d'autrui ; caractère de quelque chose qui fonctionne ou évolue indépendamment d'autre chose". Le concept d'autonomie repose en réalité sur 3 notions :

- L'auto-gouvernance : le droit et la légitimité d'agir selon ses règles.
- L'autosuffisance : la capacité d'agir avec ses ressources propres.
- L'indépendance : la liberté du moment et du périmètre de l'action.

Ces 3 notions réunies forment pour l'équipe qui les détient, une véritable **liberté d'action**.

Toutefois, l'autonomisation de l'équipe ne suffit pas à l'atteinte des objectifs de DevOps. L'autonomie sans en assumer les responsabilités est vouée à l'échec, tout comme la responsabilité seule, sans autonomie. **La liberté d'action** doit s'accompagner du **devoir de répondre de ses actes** afin d'éviter les dérives (comme les actes individualistes ou pouvant desservir l'intérêt de l'entreprise ou de l'équipe).

On parlera par la suite d' **autonomie-responsable** pour désigner la convergence de la prise de décision, de la liberté d'action et de la responsabilité du résultat au sein d'une même entité.

"Donner l'autonomie sans la responsabilité ou la responsabilité sans l'autonomie, c'est comme offrir un hachoir à un bébé."

Pourquoi l'autonomie-responsable contribue-t-elle aux objectifs de DevOps

Pourquoi l'autonomie-responsable contribue-t-elle aux objectifs de DevOps

La véritable ambition de DevOps c'est de vouloir **un produit logiciel qui soit à la fois de bonne qualité et disponible rapidement pour ses utilisateurs.**

Alors, comment les modèles d'organisations DevOps peuvent-ils nous aider à délivrer plus rapidement du logiciel en production ? Principalement en cherchant à **supprimer les temps d'attente inutiles** et en **minimisant l'effort de synchronisation** entre équipes.

Le manque d'autonomie des équipes dans une organisation informatique traditionnelle est à l'origine d'un gaspillage de temps et d'énergie considérable. Au cours du cycle de vie d'un logiciel, on constate fréquemment, au cœur même de l'équipe projet, des phases où les personnes au cœur de l'action sont dans l'impasse pour réaliser leur travail. Par exemple, les développeurs peuvent être dans l'attente du provisionnement d'un serveur par un OPS ou encore de la validation du schéma de sa base de données par un expert DBA partagé avec d'autres équipes.

Ces temps d'attente sont, dans la plupart des cas, induits par le blocage d'une ressource mutualisée, la dépendance à la réalisation d'une action par une équipe tierce ou les délais de prise de décision. Sur ce dernier point, on peut citer l'exemple des comités d'architecture ou de sécurité qui se positionnent comme des instances centrales d'approbation sur le chemin critique de chaque nouveau projet. Ce type de comité est souvent perçu par les équipes comme un frein au *Time to Market*.

Ces "pertes de temps", couplées aux interruptions fréquentes nécessaires à la synchronisation entre équipes, sont la source de **nombreuses frustrations pour les personnes au cœur de l'action** (comme la perte de concentration due au changement fréquent de contexte, ou *context switching*). Ces irritants affectent directement la qualité de l'expérience collaborateur et *in fine* contribuent au **climat de tension** qui règne souvent entre les équipes DEV et OPS.

L'organisation DevOps, en cherchant à rendre davantage autonomes les équipes, améliore à la fois sa capacité à livrer rapidement en production et le bien-être de l'équipe.

Un seul corps et un seul esprit
pour le produit

Un seul corps et un seul esprit pour le produit

"You build it, you run it"

Concrètement dans DevOps, l'autonomie-responsable pourrait être résumée par la célèbre formule de Werner Vogels, VP et CTO d'Amazon : "*You build it, you run it*". En prolongeant une dynamique de rapprochement déjà entamée par la mouvance Agile, DevOps termine l'histoire en créant l'union parfaite réunissant au sein d'une même équipe le Métier (Biz), les Études (Dev) et la Production (Ops). L'équipe BizDevOps devient autonome et responsable sur l'ensemble du cycle de vie du logiciel : du *Design*, du *Build* et du *Run*.

"You build it, you run it"

Qu'entend-on par autonomie ? Derrière cette simple réponse : *design* + *build* + *run*, citons quelques exemples de décisions qui sont de la responsabilité entière et unique de l'équipe produit :

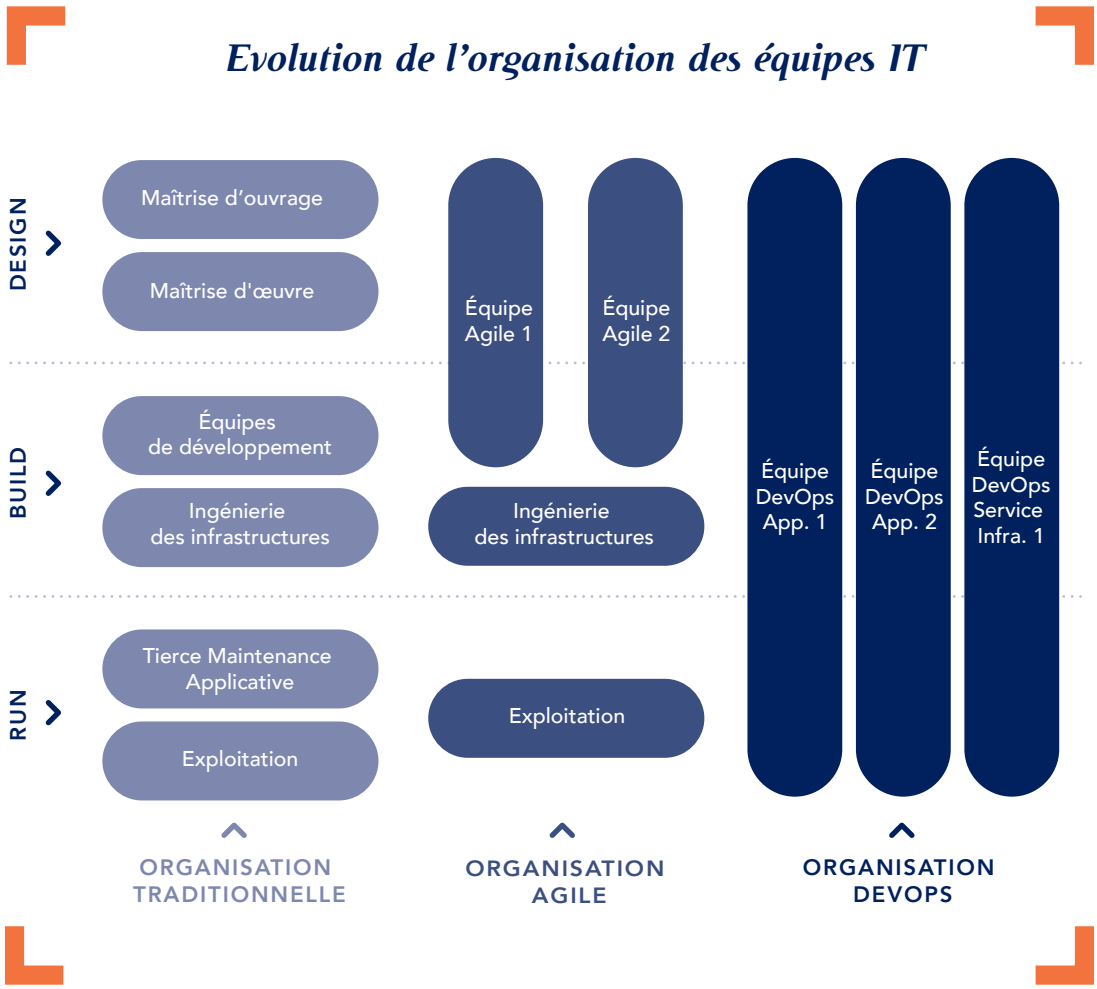
1. **La priorisation des fonctionnalités du produit**, mais également de ce qui est non-fonctionnel (qualité, performance, sécurité, résilience, etc.). Classique pour une équipe agile qui embarque un *Product Owner*[17] porteur de la vision, mais cela reste bien entendu vrai dans un contexte DevOps.

2. **La décision de Go/NoGo** en production. L'équipe identifie si elle souhaite partir en production, quand, et comment elle souhaite le faire. Elle assume collectivement ce choix et surtout ses conséquences, même s'il y a un risque d'être réveillé la nuit par un incident en production.

3. **Les choix technologiques** en début ou en cours de vie du projet (les chapitres suivants traitent notamment des risques associés à une trop forte gouvernance technologique centralisée).

4. **La liberté dans son auto-organisation**, ses rituels, ses membres constituants.

[17] Le *Product Owner* est un des rôles clés d'une équipe travaillant en mode agile. Il est le représentant des clients et des utilisateurs. Il est autonome et responsable en ce qui concerne les décisions liées au design du produit logiciel.



De l'organisation projet à l'organisation produit

DevOps, contrairement à ce que laisse penser son nom, n'a donc pas seulement vocation à fluidifier la coopération entre les équipes de développement et d'exploitation, mais bien de faire passer l'organisation d'un **modèle orienté projet** et pools de compétences à un **modèle orienté produit**.

Dans les organisations informatiques traditionnelles, on distingue 2 types d'équipes :

• **Le pool de compétences** est une équipe hiérarchique organisée en silos par compétences ou expertises. Les membres sont interchangeables car ils ont des compétences similaires. Les membres de l'équipe sont autonomes, ils n'ont pas besoin de se synchroniser entre eux pour mener à bien leur travail. L'objectif de ce mode d'organisation est de **rationaliser les compétences et d'optimiser le taux d'occupation des "ressources"**. Le pilotage de ce type de pool de compétences se fait généralement au travers d'outils de ticketing pour tracer leur activité.

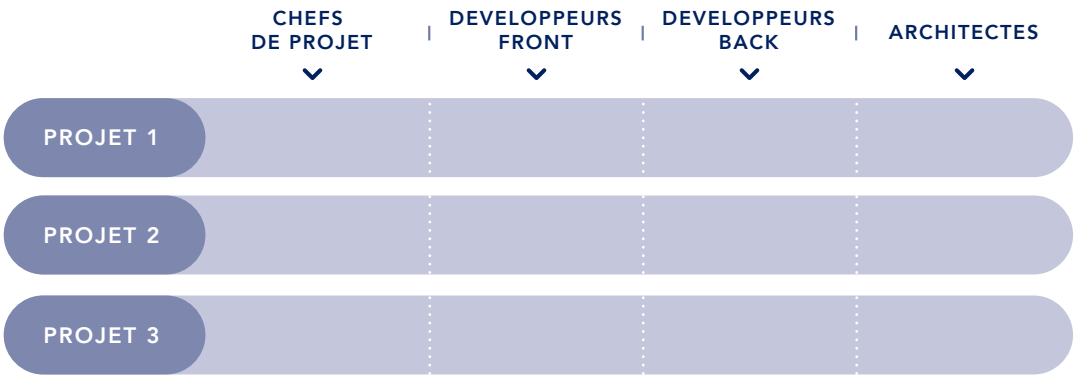
• **L'équipe projet** est une équipe recomposée à durée de vie temporaire (en général jusqu'à la livraison du projet en production, rarement au-delà), souvent non-hiérarchique. Les membres de l'équipe sont complémentaires en compétences. Le mode projet requiert un effort important de synchronisation entre ses membres. Les membres peuvent être partagés entre plusieurs projets au détriment de l'autonomie (si besoin d'une même personne dans plusieurs projets au même moment). Ce mode d'organisation permet la **construction d'une réalisation complexe et multi-compétences sans impact structurel sur l'équilibre de l'organisation**. Ce mode projet est compatible et complémentaire avec les équipes organisées en pool de compétences. Ce mode d'organisation est piloté par le délai et le budget.

Le passage en mode produit, recommandé dans une organisation DevOps, induit l'arrivée de deux nouveaux types d'équipes :

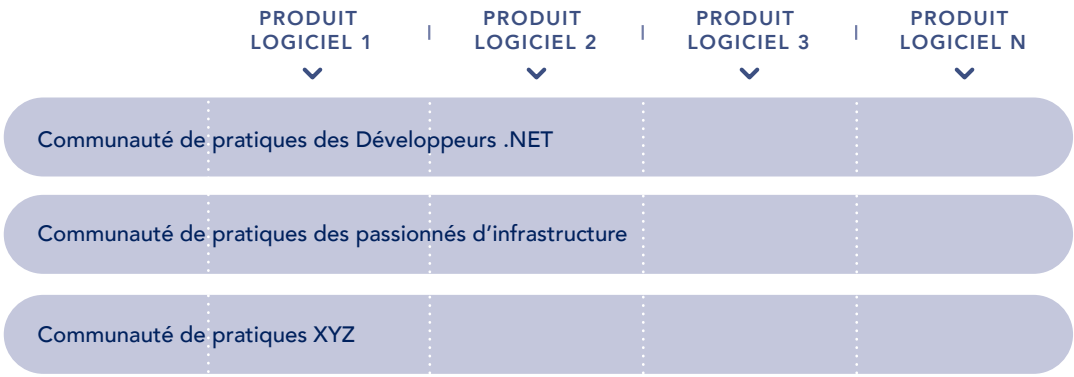
• **L'équipe produit** est proche, dans sa composition, de l'équipe projet. Elle a une durée de vie longue (la durée de vie du produit logiciel). Elle peut être hiérarchique ou non. Les membres de l'équipe sont complémentaires en compétences. L'organisation en produits logiciels requiert un effort important de synchronisation entre ses membres. Les membres sont dédiés à une équipe produit à la fois. Ce mode d'organisation **favorise l'autonomie et la responsabilité des équipes au profit d'une productivité accrue sur la réalisation et l'exploitation d'un produit**. Ce gain de productivité a un coût : le panel de compétences nécessaires pour atteindre l'autonomie de l'équipe, induit le renforcement de l'équipe ou la recherche de profils multi-compétents souvent rares sur le marché. Ce mode d'organisation est piloté par la **capacité à faire** (de l'équipe) et la **valeur délivrée** pour l'utilisateur final.

• **La communauté de pratiques** est un ensemble de personnes qui partagent une expertise technique ou méthodologique. Elle se réunit régulièrement pour échanger sur ses pratiques, faire des revues de code ou de la veille technologique, par exemple. Elle vise également à **réduire l'entropie technologique** en partageant des retours d'expérience et assure ainsi une forme de cohérence entre les produits. Une communauté de pratiques est animée par un leader qui prend en charge l'organisation (rituels réguliers, salle, sujets à aborder, etc.)

Organisation par équipes projets & pools de compétences



Organisation par équipes feature ou component & communautés de pratiques



Feature Team et Component Team

Pour beaucoup d'entreprises, passer à une organisation DevOps consiste à transformer les équipes projets Agile en équipes produits autonomes sur l'ensemble du cycle de vie du produit logiciel. Dans notre vision, DevOps s'inspire des méthodologies de passage de l'agilité à l'échelle (comme SAFe : *Scaled Agile Framework*) pour proposer 2 types d'équipes produits :

- ***Les Feature Teams***, ou équipes orientées fonctionnalités.
- ***Les Component Teams***, ou équipes orientées composants.

Le mantra de Conway sur le lien fort entre une organisation et son architecture illustre parfaitement la résurgence de ces concepts d'architecture pour distinguer les 2 types d'équipes.

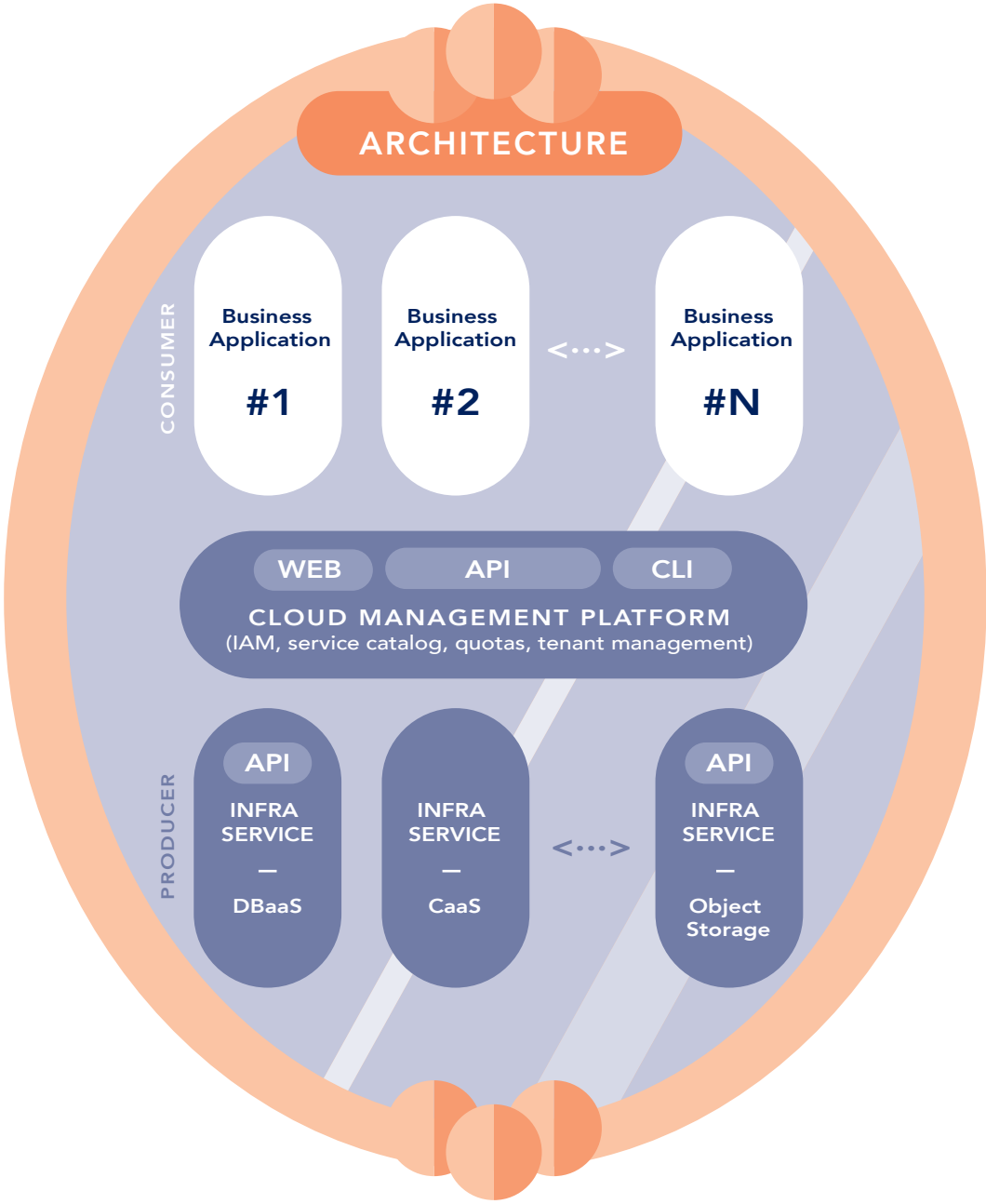
On retrouve dans SAFe le fait que l'architecture d'un système se décline en **composants** et en **fonctionnalités**. Une fonctionnalité implémente un ensemble de comportements du système qui répondent directement à des besoins de l'utilisateur. Un composant est un regroupement logique de fonctions courantes et réutilisables nécessaires à la mise en œuvre de fonctionnalités.

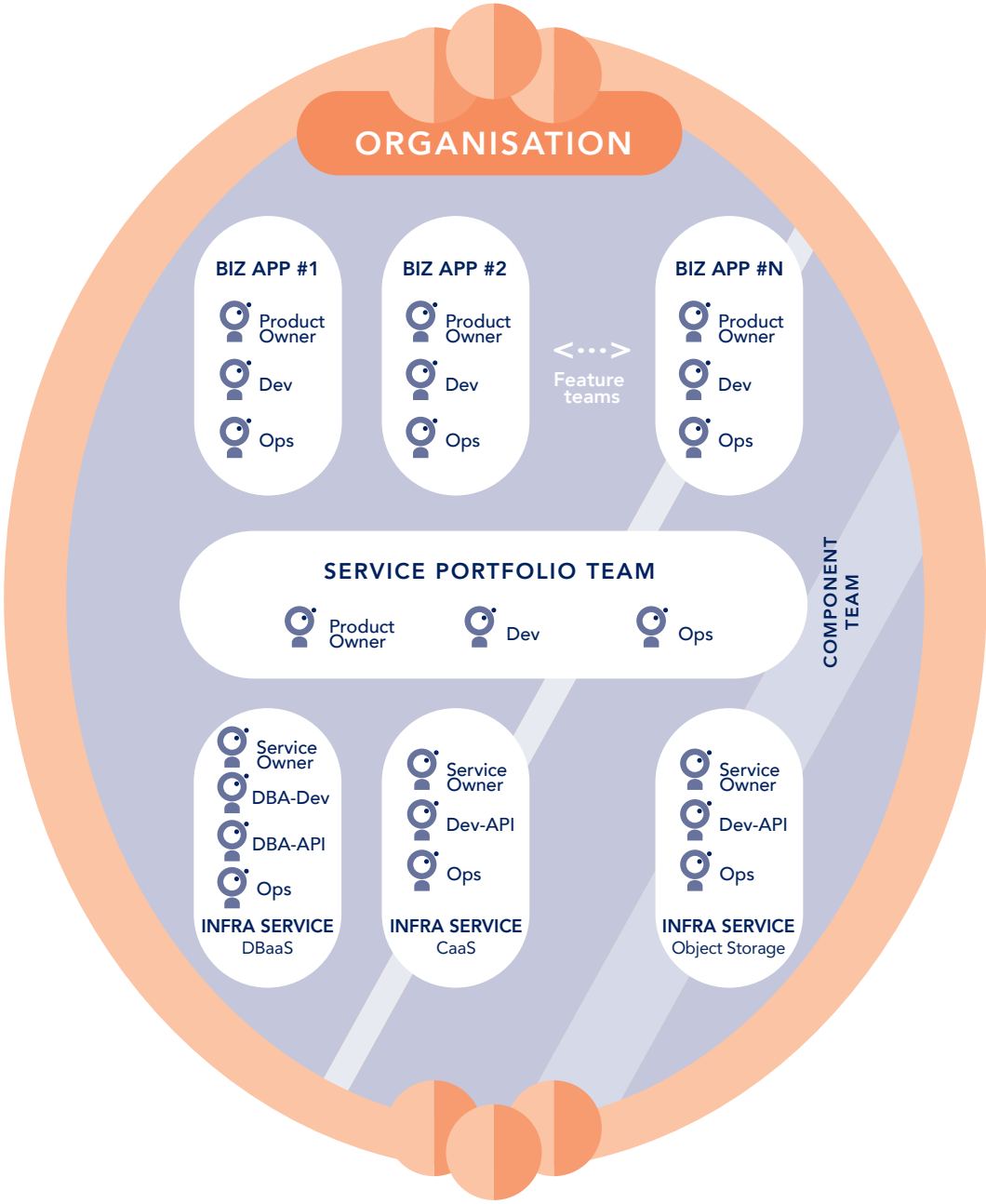
Ainsi une *Component Team* fournit **un produit logiciel de type "socle"** qui est utilisé comme fondation par les *Feature Teams*. Dans ce contexte, un composant peut être, par exemple, un framework logiciel orienté microservices ou un service partagé de queuing de messages applicatifs.

Les *Feature Teams* quant à elles, travaillent sur **des fonctionnalités à valeur cliente** (au sens Agile). Cela peut prendre la forme d'un service métier destiné aux clients de l'entreprise ou de services à destination des utilisateurs internes à la société. Par exemple, une *Feature Team* A peut travailler sur un produit de régie publicitaire sur un site de e-commerce et une autre *Feature Team* B sur un service de Machine Virtuelle à la demande (IaaS) à destination de collaborateurs de l'entreprise.

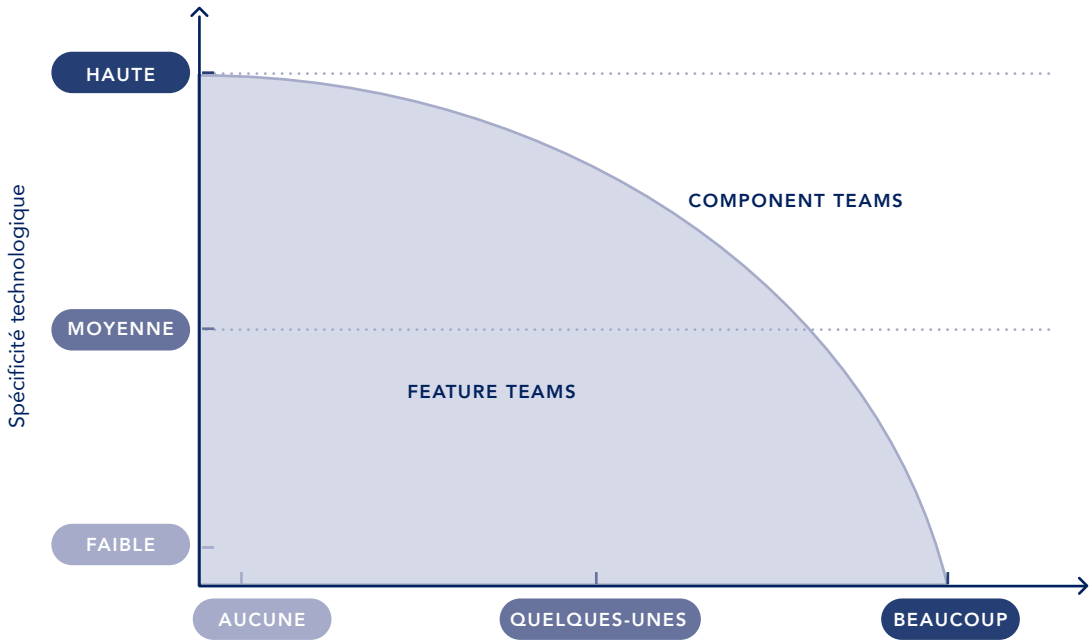
> *"Organizations which design systems are constrained to produce systems which are copies of the communication structures of these organizations."*
>
> Conway's Law[18]

[18] Les organisations produisent des systèmes qui calquent les structures de communication de ces organisations.


ARCHITECTURE
CONSUMER
Business Application #1
Business Application #2
<···>
Business Application #N
WEB
API
CLI
CLOUD MANAGEMENT PLATFORM
(IAM, service catalog, quotas, tenant management)
PRODUCER
API
INFRA SERVICE
—
DBaaS
INFRA SERVICE
—
CaaS
<···>
API
INFRA SERVICE
—
Object Storage


ORGANISATION
BIZ APP #1
Product Owner
Dev
Ops
BIZ APP #2
Product Owner
Dev
Ops
<...>
Feature teams
BIZ APP #N
Product Owner
Dev
Ops
SERVICE PORTFOLIO TEAM
Product Owner
Dev
Ops
COMPONENT TEAM
Service Owner
DBA-Dev
DBA-API
Ops
INFRA SERVICE
DBaaS
Service Owner
Dev-API
Ops
INFRA SERVICE
CaaS
Service Owner
Dev-API
Ops
INFRA SERVICE
Object Storage

Dean Leffingwell (auteur du framework SAFe) présente ici cette différence entre Feature et Component teams :



Si les *Component Teams* sont bien nécessaires, il est néanmoins souhaitable d'en limiter la quantité. Comme évoqué précédemment, les composants produits par ces équipes servent de bases aux fonctionnalités développées par les *Feature Teams*. Bien que cela ait un réel bénéfice en matière de réutilisabilité, ceci engendre un couplage technique plus ou moins fort entre le composant et la fonctionnalité. Cette dépendance, en synchronisant les cycles de vie des sous-systèmes, dégrade l'autonomie de l'équipe. *In fine*, il convient de limiter la prolifération des *Component Teams* en conservant un ratio théorique idéal de 80 % de *Feature Teams* et 20 % de *Component Teams*.

Le choix de créer une équipe orientée composant devra se restreindre aux **sous-systèmes hautement spécialisés technologiquement et à forte réutilisabilité avérée**. Pour autant, les frameworks d'architecture d'entreprise ou les méthodes d'urbanisation du SI ne semblent pas avoir réussi à limiter la prolifération – souvent injustifiée – de composants fortement couplés et faiblement cohérents d'un point de vue fonctionnel.

En synthèse, DevOps propose de passer d'un modèle d'organisation traditionnel en projet, à un modèle orienté produit où chaque équipe est en autonomie-responsable pour développer un composant réutilisé ou une fonctionnalité logicielle.

Le passage à l'échelle de l'équipe produit

Le passage à l'échelle de l'équipe produit

L'équipe DevOps idéale est donc une équipe produit. Nous l'avons vu, pour fonctionner, ce type d'équipe a un fort besoin de **collaboration impliquant des synchronisations fréquentes entre les membres de l'équipe**. Plus le nombre de membres de l'équipe est important, plus cet effort de synchronisation devient chronophage. Les travaux du psychologue J. Richard Hackman sur la dynamique de groupe ont montré que le nombre de canaux de communication interpersonnels croît de manière exponentielle avec la taille de l'équipe, rendant la collaboration de plus en plus inefficace.

$$\Delta = N(N-1)/2$$

Ainsi, selon la formule suivante, il n'existe que 15 canaux de communication dans une équipe de 6 de personnes, puis 66 dans une équipe de 12 et jusqu'à 1 225 dans une équipe de 50 !
$\Delta = N(N-1)/2$

Voilà pourquoi la collaboration ne passe pas à l'échelle au sein d'une même équipe. Mais bonne nouvelle, ce n'est pas pour autant que l'organisation DevOps doit se cantonner au monde des startups et des petites entreprises. Pour accompagner la croissance, il convient d'ajouter autant de *Feature Teams* et de *Component Teams* que nécessaire en s'assurant de leur autonomie les unes par rapport aux autres. Procéder au découpage (voire au redécoupage en cours de vie des produits) représente par conséquent une activité stratégique à réaliser en continu.

Two-pizza team : la taille idéale de l'équipe DevOps

Pour Jeff Bezos, le PDG d'Amazon, davantage de communication n'est pas toujours la bonne solution aux problèmes de communication dans les entreprises. En proposant de **réduire la taille standard des équipes** au nombre de personnes pouvant être nourries par deux pizzas (de taille américaine, précisons), Jeff Bezos réaffirme un fait que les psycho-sociologues connaissent bien : la collaboration est optimale dans des équipes de taille réduite, comprenant entre 5 et 12 personnes.

Mais construire une équipe autonome de taille réduite est plus complexe qu'il n'y paraît...

Le premier défi consiste à **réunir toutes les compétences dont a besoin l'équipe pour être autonome** et cela avec un nombre de personnes très réduit. Cela revient bien souvent à chercher des moutons à cinq pattes : les fameux profils pluridisciplinaires. Ainsi, les développeurs seront aussi leader technique, testeur, architecte logiciel full-stack et les profils DevOps mixeront les casquettes d'architecte technique, de développeur d'*infrastructure as code*, d'exploitant, d'ingénieur sécurité, système, stockage et réseaux… Ajoutez à cela une pénurie de ces deux profils sur la marché de l'emploi et vous comprendrez la difficulté pour construire ce type d'équipe.

La seconde difficulté commune de ces équipes réduites est **la résilience des compétences**. Le déséquilibre entre la grande quantité de compétences et le faible nombre de membres dans l'équipe peut induire un silotage des savoirs et savoir-faire entre les personnes. Prenons l'exemple du profil Ops. Dans une équipe de six personnes, il est courant que ce profil ne soit représenté qu'une seule fois. Qu'advient-il si cette personne part en congés ou tombe malade ? Cela met-il le produit et l'équipe en péril ?

La stratégie usuelle pour atténuer ce risque consiste à **"désiloter" les rôles**. Il s'agit ici de **créer des zones de recouvrement des savoir-faire sur les profils "rares"** avec d'autres membres de l'équipe. Par exemple, le développeur leader technique aura également des compétences pour maintenir l'usine d'intégration et de déploiement continus mise en place par l'Ops. Une autre stratégie, complémentaire de cette première, a pour objectif d'améliorer la résilience des savoirs grâce à des **pratiques de binômage** (telles que le *pair-programming* ou les *peer-reviews* par exemple).

Les communautés de pratiques

Faire passer DevOps à l'échelle revient à **multiplier les équipes produits** autant que nécessaire (*Feature* ou *Component teams*). Toutefois, cela a pour impact de réduire la surface d'échange et de communication entre les personnes partageant un intérêt technique ou méthodologique, et n'appartenant pas à la même équipe produit. Pour palier ce silotage, il est recommandé de **créer et d'animer des communautés de pratiques** transverses aux équipes produits permettant :

- Le partage des bonnes pratiques du métier.
- Le travail sur des sujets transverses (choix outillage, POC, etc.).
- La réalisation de veille technologique ou méthodologique organisée.

Les communautés de pratiques se réunissent habituellement lors d'un rituel hebdomadaire ou bihebdomadaire d'une demi-journée à date fixe. En dehors de ce rituel, la communauté vit grâce à des espaces de collaboration (comme Slack, par exemple). On peut dédier classiquement 5 à 10 % du temps productif de l'équipe aux activités de la communauté

de pratiques. Il est recommandé que la communauté désigne un leader en charge d'animer la vie du groupe (sans nécessairement de lien hiérarchique).

Au final, **les communautés de pratiques participent à la maîtrise de l'entropie technologique de l'entreprise**. Si vous savez qu'il existe une communauté dynamique avec des compétences et des retours d'expérience concrets sur une technologie, il y a fort à parier que cela oriente vos choix techniques au moment de démarrer un nouveau projet.

Pratiques méthodologiques & traits culturels propices à DevOps

Certaines organisations ont un terreau plus fertile que d'autres lorsqu'il s'agit d'y semer les graines du DevOps. Le modèle machiavélique de J.P. Kotter sur la conduite du changement (et ses fameuses huit étapes[19]) semble ignorer l'importance à accorder aux spécificités culturelles lors d'une transformation d'organisation. Prendre en compte et agir sur la culture – ou plutôt les cultures – de l'entreprise est fondamental lors du déploiement de DevOps.

⑲ Les huit étapes du changements de J.P. Kotter : créer un sentiment d'urgence. Former une coalition. Développer une vision. Communiquer la vision. Inciter à l'action. Démontrer des résultats à court terme. Bâtir sur les premiers résultats pour accélérer le changement. Ancrer les nouvelles pratiques dans la culture d'entreprise.

Traits culturels facilitateurs

Traits culturels facilitateurs

Certaines pratiques managériales peuvent faciliter l'adoption de DevOps. Dans ce chapitre, nous illustrerons ces éléments de culture par le biais de trois paraboles :

Le savoir, c'est le pouvoir[20]

"Au quatre coins du monde quatre chercheurs travaillent séparément au même objectif : celui de créer un ordinateur quantique. Chacun réalise des avancées scientifiques remarquables et développe un savoir qui lui est propre. Au hasard d'un colloque scientifique, les quatre chercheurs se rencontrent. L'un des chercheurs, attiré par l'appât du gain, décide de garder le secret sur ses découvertes et passe son chemin afin de se remettre au travail le plus tôt possible. Les trois autres discutent toute la journée et toute la nuit. C'est alors qu'en assemblant – tel un puzzle – leurs savoirs respectifs, ils résolvent ensemble les mystères quantiques qui permettront la création du fameux ordinateur. La morale de cette histoire, c'est que les savoirs ne se divisent pas, ils ne s'additionnent pas, mais ils se multiplient lorsqu'on les partage."

Le premier trait culturel que nous aborderons est la **culture du partage**. La transmission des savoirs entre le senior et le junior ou entre le DEV et l'OPS, l'open-data, la transparence, le management visuel ou encore la contribution au logiciel libre sont autant d'exemples incarnant la culture du partage. *A contrario*, il existe des pratiques culturelles allant à son encontre, comme la culture du secret, la paranoïa sécuritaire, les comportements individualistes ou politiques engendrant la rétention, la division ou le silotage de l'information.

L'inertie du conditionnement culturel

"Une vingtaine de chimpanzés sont isolés dans une pièce où est accrochée au plafond une banane, et seule une échelle permet d'y accéder. La pièce est également dotée d'un système qui permet de faire couler de l'eau glacée dans la chambre dès qu'un singe tente d'escalader l'échelle. Rapidement, les chimpanzés apprennent qu'ils ne doivent pas escalader l'échelle. Le système d'aspersion

[20] On attribue cette formule au philosophe anglais Francis Bacon.

d'eau glacée est ensuite rendu inactif, mais les chimpanzés conservent l'expérience acquise et ne tentent pas d'approcher de l'échelle. Un des singes est remplacé par un nouveau. Lorsque ce dernier tente d'attraper la banane en gravissant l'échelle, les autres singes l'agressent violemment et le repoussent. Lorsqu'un second chimpanzé est remplacé, lui aussi se fait agresser en tentant d'escalader l'échelle, y compris par le premier singe remplaçant. L'expérience est poursuivie jusqu'à ce que la totalité des premiers chimpanzés qui avaient effectivement eu à subir les douches froides soient tous remplacés. Pourtant, les singes ne tentent plus d'escalader l'échelle pour atteindre la banane. Et si l'un d'entre eux s'y essaye néanmoins, il est puni par les autres, sans savoir pourquoi cela est interdit et en n'ayant jamais subi de douche glacée."

Le théorème du Singe[21] illustre l'impact des décisions du passé sur les comportements du futur. En entreprise, certaines règles ou processus mis en place à juste titre sont à l'origine de situations absurdes, quelques années plus tard, alors que le contexte a changé. **La remise en cause de cet "ordre établi"** ancré dans la culture d'entreprise, peut engendrer d'importantes réticences. On observe ainsi dans certaines organisations un effet de **vieillissement de la culture d'entreprise** par le développement d'**anticorps au changement**. Le poids des règles du passé peut étouffer à feu doux la capacité d'adaptation de l'organisation. Cela explique en partie l'émergence des paradigmes de DevOps chez les Géants du Web, jeunes entreprises sans passif culturel.

Errare humanum est, sed perseverare diabolicum est [22]

"Un entrepreneur de travaux publics avait trois architectes. Il demanda à chacun de construire le pont le plus incroyable jamais connu.

Le premier, réalisa une étude ambitieuse et décida de valider la viabilité de son concept grâce à une maquette. L'assemblage des poutrelles d'acier était trop lourd et fit s'écrouler la maquette. Malgré cela, l'architecte décida de construire le pont, qui – sans surprise – subit la même fin que la maquette. Culpabilisant de cet échec, ce dernier démissionna.

Le second architecte ayant eu écho du sort de son collègue ne prit aucun risque et réalisa un pont en pierre commun, dont il maîtrisait parfaitement la conception. Mais l'entrepreneur, voyant le manque d'audace de l'ouvrage blâma l'architecte.

Le troisième, attaché à la consigne de son chef, décida de construire un incroyable pont en cristal. La fragilité du matériau et l'ambition folle du projet ralentirent l'avancée de l'étude. A chaque nouvelle maquette, le pont s'écroulait. Sans se décourager, l'architecte documentait

---

㉑ Théorème du Singe, source Wikipédia : https://fr.wikipedia.org/wiki/Th%C3%A9or%C3%A8me_du_singe. ㉒ "Errare humanum est, sed perseverare diabolicum est" est une locution latine qui signifie "L'erreur est humaine, l'entêtement [dans son erreur] est diabolique". ㉓ *The Unexpected Benefit of Celebrating Failure* : https://www.ted.com/talks/astro_teller_the_unexpected_benefit_of_celebrating_failure. ㉔ Voir l'ouvrage *Management 3.0* par Jurgen Appelo : https://management30.com/product/management30/. ㉕ Post Mortem public lors d'un incident sur GitLab.com : https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/.

minutieusement les raisons de son échec et présentait chaque jour ses apprentissages à l'entrepreneur. Ce dernier, malgré les retards, l'encouragea, jusqu'au jour où la maquette résista aux tests. L'architecte construisit le fabuleux pont de cristal et fût récompensé par l'entrepreneur. Le prestigieux prix Pritzker d'architecture lui fut même décerné."

Il y a deux principales leçons à tirer de cette histoire :

1. Si l'erreur est humaine, **l'entêtement dans son erreur est nuisible.** Les méthodes d'essai-erreur sont couramment employées dans la résolution de problèmes complexes (dans la recherche fondamentale, les mathématiques et également dans le développement logiciel). **L'échec est donc une opportunité d'apprendre** (empirisme apprenant). Il convient dans ce contexte de ne pas sanctionner l'erreur, mais au contraire de la célébrer (comme le fait Google[23] par exemple).

2. **Sans droit à l'erreur, il n'y a pas de prise de risque**. La peur de l'échec peut être naturellement paralysante. Inciter la prise d'initiatives sans culture du "droit à l'erreur" ne fonctionne pas.

Mieux vaut demander pardon que la permission. **La concrétisation du droit à l'erreur** peut se faire à trois niveaux :

"Toute personne qui n'a jamais commis d'erreurs n'a jamais tenté d'innover."

Albert Einstein

- À titre individuel : en se déculpabilisant lors de l'échec et en cherchant à apprendre de ses erreurs.
- Au niveau de l'équipe : en promouvant les pratiques managériales 3.0[24], comme la culture du *feedback* efficace, l'*egoless management* ou la bienveillance collective.
- À l'extérieur de l'organisation : en communiquant publiquement sur ses échecs et les apprentissages qui en sont issus (exemple de *post mortem* public de Gitlab[25]).

Selon leur taille, leur ancienneté, ou leur secteur d'activité, les organisations développent **un ADN culturel** qui leur est propre. En résumé, on peut citer 5 traits culturels qui sont des catalyseurs à l'adoption de nouvelles pratiques comme DevOps :

- La transmission des savoirs.
- La transparence sur les données.
- La capacité à remettre en question des processus ou des règles établies.
- Le droit à l'erreur d'apprentissage.
- La liberté et l'incitation à entreprendre.

Rythmes & rituels

Rythmes & rituels

Si toutes les organisations n'ont pas une culture propice à DevOps, est-il quand même possible de faire changer les choses ?

Transformer la culture d'une organisation n'est pas trivial. Décentralisée par nature, **la culture est lourdement enracinée dans l'organisation**. Chaque collaborateur est tel un coffres-forts défendant précieusement une sous-partie de cette culture. Il est impossible de modifier simultanément le contenu de tous les coffre-forts. Le *hacking* de la culture doit se faire sur un modèle de contagion virale, en faisant **changer les comportements localement** pour qu'ils se transmettent de proche en proche. Dans cette partie, nous nous intéresserons à l'impact des rituels, rythmes et cadences pour injecter le virus du changement dans la culture.

Faites danser l'hippopotame

La culture est tel un hippopotame que l'on voudrait faire bouger. Donnez-lui l'ordre de se déplacer, cela n'aura aucun effet. Tentez de le pousser, c'est une cause perdue car l'animal est trop lourd. En revanche, donnez-lui à entendre le rythme endiablé d'un air de percussions brésiliennes et vous verrez le pachyderme se mettre à danser. **Transformer la culture est une affaire de rythme.**

En sociologie, le **rythme social**[26] **est un élément fondateur de la vie d'un groupe**. Le rythme est comme le battement de cœur qui donne vie au groupe. Il synchronise pour chaque individu, le mouvement sinusoïdal où s'alternent les phases où le groupe est réuni et celles où il est dispersé. Cette synchronisation pendulaire se fait grâce à la mise en place de rituels.

Ces rituels sont des espaces réguliers, périodiques, normés où se retrouvent les individus d'un groupe pour y négocier les règles sociales qui le régissent. **Redéfinir le rythme par le biais de nouveaux rituels** est un moyen efficace pour mettre en mouvement la culture d'une organisation.

Ceux qui ont cherché à adopter la culture Agile savent que la méthode consiste notamment à redéfinir les rituels de l'équipe. Le découpage du temps de travail en itérations ou *sprints*, la mise en place de *stand-up meetings*, de rétrospectives, de *backlog-grooming* ou encore de démonstrations[27] sont autant de rituels fortement normés, au service d'un nouveau rythme.

[26] Voir le courant de pensée du sociologue Emile Durkheim : http://rhuthmos.eu/spip.php?article511
[27] Se reporter aux définitions des rituels *Scrum* : https://fr.wikipedia.org/wiki/Scrum_(Boite_à_outils)

De même que pour l'adoption de la culture Agile, **la transformation vers DevOps change le rythme**, redéfinit les rituels de l'équipe. Pour cela DevOps s'appuie sur un mouvement de convergence des rituels des deux mondes DEV et OPS :

- Le prolongement des rituels Agiles à l'ensemble de l'équipe produit DevOps (BIZ+DEV ➔ BIZ+DEV+OPS) : itération, *stand-up*, rétrospective, démonstration, etc.

- L'appropriation de rituels OPS par l'équipe produit DevOps : *post mortem*, astreintes tournantes, gestion des changements et des problèmes, etc.

L'adoption d'une partie de ces rituels par l'équipe DevOps permet d'**optimiser la qualité de la collaboration en réduisant la quantité de documentation**, en allant à l'essentiel (*timeboxing*), en renforçant les liens interpersonnels et en créant des **repères dans le travail** pour qu'il soit plus prévisible, moins stressant. Ces rituels sont également de puissants médias pour **promouvoir de nouvelles conventions culturelles** comme l'amélioration continue (rétrospective), le droit à l'erreur (*post mortem*), la transmission des savoirs et la transparence (*stand-ups*, dojos, *meetups*), par exemple. Imposer des rituels aux équipes n'est pas souhaitable pour autant, il faut que les collaborateurs soient convaincus de leur utilité, qu'ils se les approprient et les adaptent en continu (grâce à la rétrospective). Ainsi, d'une équipe à l'autre, les rituels et leurs formes pourront varier et cette diversité globale s'imposera au profit de l'efficacité locale.

"Imposer des rituels aux équipes n'est pas souhaitable, il faut que les collaborateurs soient convaincus de leur utilité."

Management par les pairs
et compagnonnage

Management par les pairs et compagnonnage

Pour garantir la résilience des compétences et des savoirs dans l'équipe, DevOps s'appuie sur des pratiques de compagnonnage et de management par les pairs. Elle met ainsi au centre du quotidien des équipes **la transmission des savoirs** pour se faire grandir mutuellement et faire grandir la qualité du produit.

Selon la phase de maturité DevOps de l'organisation, on peut promouvoir les pratiques suivantes :

- **Lorsque les équipes DEV et OPS sont (encore) séparées**[28] : les pratiques de "Vis ma vie" ou de "*Shadowing*". Elles consistent à partager pendant quelques jours le quotidien d'un OPS lorsqu'on appartient aux DEV et vice-versa. Ces pratiques visent quatre principaux objectifs : la création de relations interpersonnelles inter-équipes, la compréhension empathique des objectifs et contraintes de l'autre, le partage d'un vocabulaire commun et la transmission de compétences.
- **Lorsque DEV et OPS sont réunis dans la même équipe** : les pratiques de binômage comme les "*peer-reviews*" ou encore le "*pair-working*" mélangeant DEV et OPS. Elles consistent à travailler en binôme sur une même unité de travail, soit de manière synchrone (deux personnes travaillant sur un même ordinateur) soit de manière séquentielle (l'un revoit après-coup le travail de l'autre et propose des améliorations).

C'est dans l'optique de faire grandir les autres, en leur transmettant le fruit de son expérience, que s'inscrivent ces pratiques DevOps de compagnonnage et de management par les pairs.

[28] Cette étape de maturité a lieu avant que la transformation organisationnelle DevOps n'ait été opérée. Cela reste dans tous les cas une mauvaise pratique de séparer DEV et OPS. Nous en reparlerons plus loin.

Les anti-patterns organisationnels les plus courants

Adopter une organisation et une culture DevOps est un processus complexe qui requiert ***beaucoup de pragmatisme et de patience****. Toutefois, il existe des pièges récurrents qu'il est possible d'éviter. Dans les paragraphes qui suivent, nous vous présenterons un petit recueil des mauvaises pratiques les plus couramment constatées lors de la mise en œuvre d'organisations DevOps et nous vous montrerons comment les éviter.*

Dépendance de compétences clés
en dehors de l'équipe

Dépendance de compétences clés en dehors de l'équipe

Ceci est une histoire vraie. Le directeur des systèmes d'information d'un grand groupe industriel décide d'engager son organisation vers DevOps. À cette fin, il identifie 3 projets pilotes comme "cobayes". Chaque équipe est constituée d'un Product Owner, de 3 à 6 développeurs et d'un profil OPS. Ce dernier profil étant difficilement mobilisable dans l'organisation à cause des nombreux autres sujets jugés plus prioritaires, il est partagé entre les trois projets pilotes. L'Ops passant d'équipe en équipe, ne développe pas de sentiment d'appartenance, il ne participe pas aux rituels et fait du mieux qu'il peut pour concilier les objectifs des différents projets. Malgré cela, il devient rapidement un goulet d'étranglement, saturé par les besoins des équipes. Il est un point de passage obligatoire notamment pour les déploiements en production. Un stock de fonctionnalités s'accumule aux portes de la production, sans qu'elles puissent y être déployées – en autonomie – par les projets pilotes. La promesse d'amélioration du Time to Market attendue par le DSI n'est pas à la hauteur des espérances.

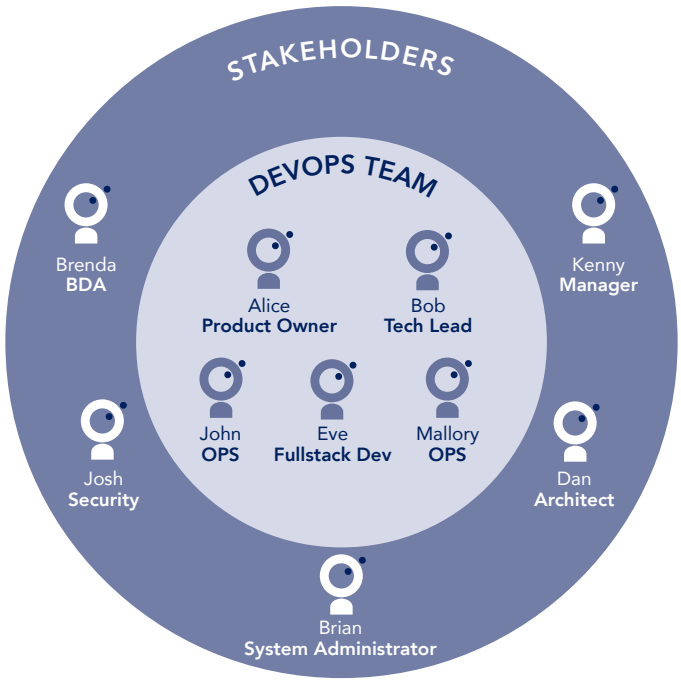
L'autonomie d'une équipe est fragile. Elle peut aisément être abîmée par une dépendance vis-à-vis d'une expertise clé extérieure à l'équipe. Ici, le profil OPS est partagé entre trois équipes différentes. Compte-tenu des besoins propres à chaque équipe, il aurait été judicieux de dédier cette expertise à chaque projet pilote.

Pour éviter cet écueil, il peut être intéressant d'**adopter une vision systémique et organique** de l'équipe DevOps. Elle est un organisme vivant. Pour évoluer et s'adapter à son environnement, elle a besoin d'ingérer des ressources extérieures (comme des compétences, par exemple). Ses besoins vitaux changent dans le temps, au gré des challenges techniques ou des nouveaux besoins des utilisateurs. Pour survivre, l'équipe DevOps doit passer d'une conception minérale, statique, à un modèle organique et dynamique. En d'autres termes, la composition des membres de l'équipe doit évoluer dans le temps, en fonction des besoins. Pour cela, l'équipe a à sa disposition deux outils simples mais puissants : l'**organigramme d'équipe et le sociogramme d'itération**.

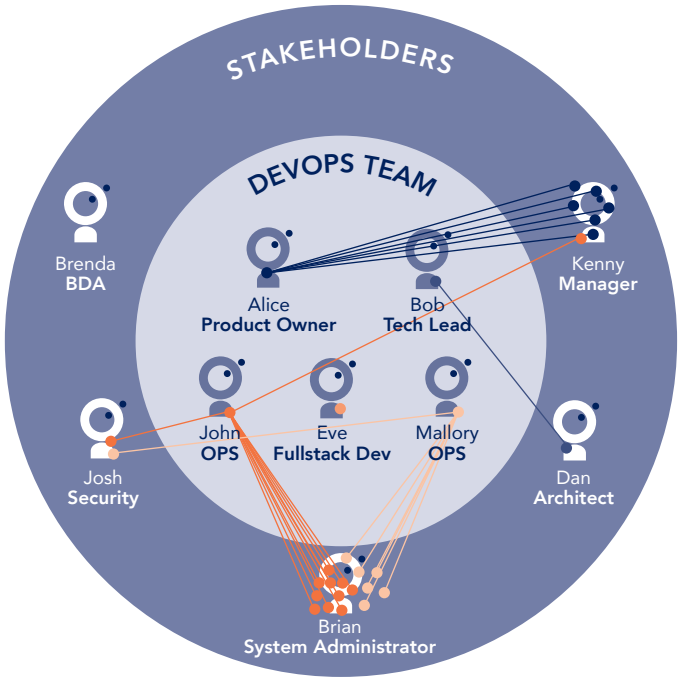
L'organigramme d'équipe est un instantané modélisant une vue des membres à l'intérieur de l'équipe et les parties-prenantes, extérieures. Rappelons-le, faire partie de l'équipe induit un engagement proche du temps plein pour les activités de l'équipe et une participation assidue aux rituels. L'organigramme est largement partagé et souvent affiché sur les murs des bureaux de l'équipe.

L'organigramme d'équipe (ci-dessous) est complété par un sociogramme d'itération (page droite). Ce sociogramme modélise la quantité d'interactions sociales entre les membres de l'équipe et les parties-prenantes.

Organigramme d'équipe



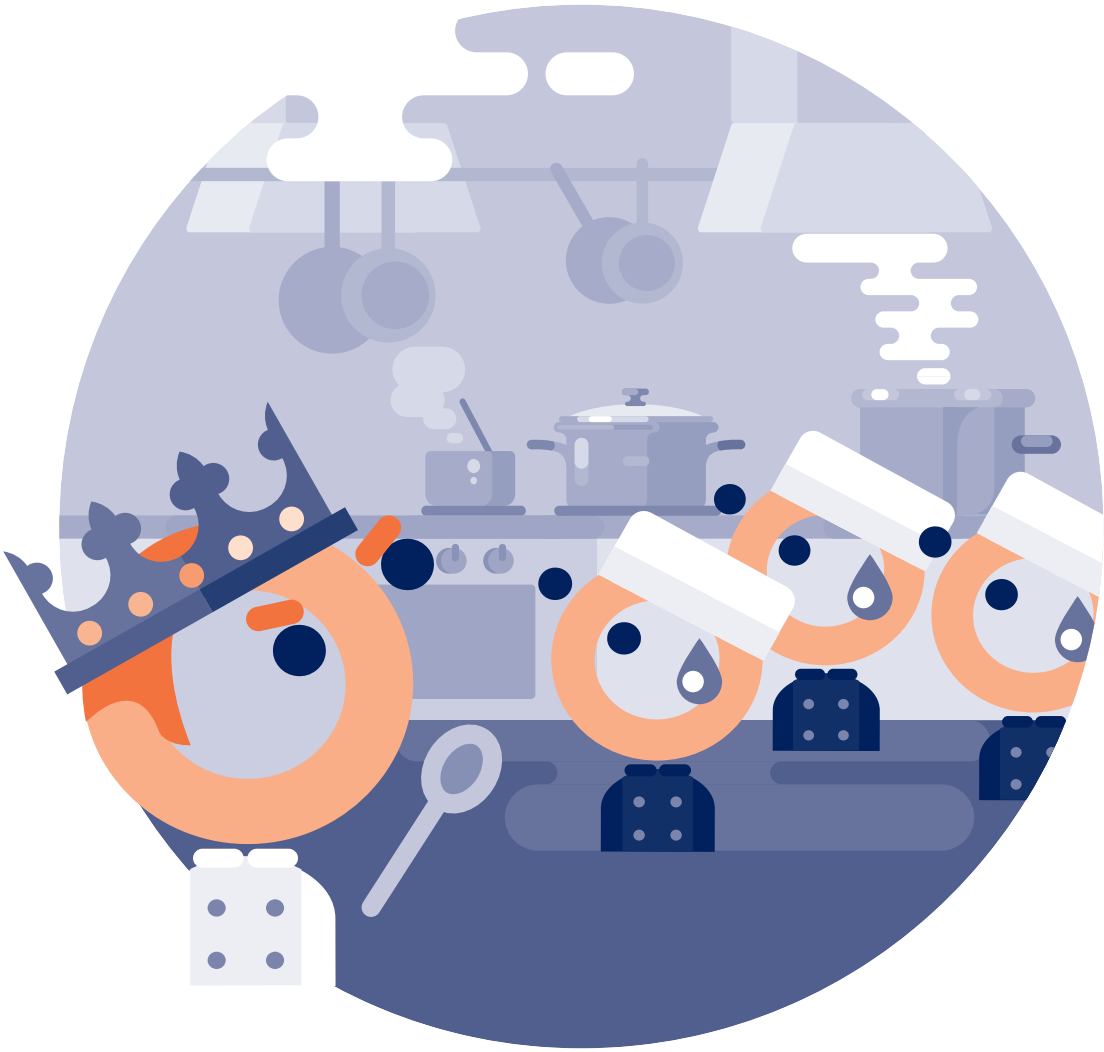
Sociogramme



À chaque fois qu'un membre de l'équipe a une interaction avec une personne extérieure à l'équipe (demande, requête, question, échange), ce dernier ajoute une gommette sur le rôle correspondant dans le sociogramme. À la fin de l'itération, lors de la rétrospective, on identifie les rôles ayant le plus de gommettes et, collectivement, on se pose les questions suivantes :

- Y a-t-il des dépendances fortes avec ces rôles extérieurs à l'équipe ?
- Au vu des prochaines *user stories* du *backlog*, ces dépendances sont-elles durables ? Pour combien d'itérations encore ?
- Est-il souhaitable de faire évoluer l'organisation de l'équipe pour s'approprier cette compétence, ou revoir son mode d'interaction avec ces rôles ?

En ritualisant ainsi l'analyse de ses dépendances vis-à-vis de compétences externes, l'équipe se met en **posture proactive** pour protéger son autonomie, se protéger des interactions parasites et anticiper l'évolution des compétences clés nécessaires à sa survie.



Les ennemis de l'autonomie :
les trois anti-patterns

Les ennemis de l'autonomie : les trois anti-patterns

La dépendance à une expertise extérieure à l'équipe n'est malheureusement pas le seul écueil que l'on rencontre lorsque l'on s'intéresse à l'autonomie des équipes produits. On peut citer *a minima* trois autres pièges que l'on rencontre fréquemment à ce sujet.

Le premier piège

Le premier piège porte sur la **mutualisation de ressources clés** entre plusieurs équipes DevOps. Par exemple, il est courant que pour des raisons de rationalisation de coûts (de licences ou d'infrastructure), certaines entreprises imposent aux équipes de mettre en commun des ressources, de cohabiter sur un serveur, de partager un environnement de pré-production, un outil d'intégration continue comme Jenkins ou encore un cluster de base de données. Ces types de ressources n'ont généralement pas été conçues dans une optique *multi-tenant* – c'est-à-dire en capacité d'accueillir plusieurs applications, produits, projets ou organisations sur une même instance. Ainsi, il est presque impossible de **garantir des quotas d'utilisation sur les ressources partagées** (processeur, mémoire, I/O disque, par exemple), de ségréguer les espaces logiques par équipe, par projet, par environnement et d'en assurer l'étanchéité et la gestion appropriée des droits d'accès.

"La mutualisation de ressources a la fâcheuse tendance à lier les cycles de vies des composants qui reposent dessus. "

Par ailleurs, la mutualisation de ressources a la fâcheuse tendance à lier les cycles de vies des composants qui reposent dessus. Prenons l'exemple de deux équipes DevOps qui partagent un serveur d'applications en cluster. Suite à une vulnérabilité de sécurité, l'éditeur publie un patch qui doit être appliqué rapidement sur le cluster. La première équipe, qui dispose d'un harnais de tests de non-régression automatisés, valide immédiatement sur l'environnement de test que ce patch n'impacte pas le bon fonctionnement de son application.

La seconde équipe, ne disposant pas de tests automatisés, mettra plusieurs semaines avant d'arriver à la même conclusion. Alors qu'il aurait été possible et souhaitable de protéger l'application de la première équipe de cette vulnérabilité en installant le patch sur la production aussitôt les tests passés, il aura fallu attendre plusieurs semaines, que la seconde équipe ait également validé la non-régression. *In fine*, en raison de la mutualisation du cluster la seconde équipe a imposé à la première un rythme qui n'est pas le sien, diminuant par la même occasion son autonomie.

Pour éviter d'abîmer l'autonomie des équipes par cette mutualisation de ressources, il est important de s'assurer que ces dernières supportent parfaitement une **approche *multi-tenant*** qui recrée une forme **d'étanchéité et d'autonomie** (comme le sont la plupart des plateformes cloud du marché).

"Une approche one-size-fits-all trouve rapidement ses limites lorsque l'on cherche à faire des produits sur mesure, de qualité et ayant un Time to Market réduit. "

Le second piège

Le second piège commun porte sur une **gouvernance technologique ou méthodologique trop assertive**, voire dogmatique. L'édiction de standards d'outillage, de normes méthodologiques, de développement ou de qualité est bien trop souvent imposée par des équipes transverses aux équipes DevOps et ceci au détriment de leur autonomie locale à faire des choix éclairés et contextualisés aux spécificités de leurs produits. Dans l'absolu, la présence de standard n'est pas une mauvaise chose si elle n'est pas utilisée comme un moyen de pallier un défaut de compétence interne à l'équipe ou d'imposer une solution unique comme une réponse universelle à tous les besoins. Cette approche *one-size-fits-all* trouve rapidement ses limites lorsque l'on cherche à faire des produits sur mesure, de qualité et ayant un *Time to Market* véloce.

Prenons l'exemple d'une DSI disposant d'un catalogue de *middlewares* "sur étagère". Parmi ces standards, on retrouve deux choix possibles en matière de base de données : Oracle et PostgreSQL. Si pour répondre à ses contraintes d'architecture, une équipe DevOps souhaite se reposer sur une base de données de type NoSQL, il est alors légitime que cette dernière puisse faire un choix en dehors du standard, comme une base de données Cassandra, par exemple. En gardant la possibilité de ne pas choisir le standard, l'équipe préserve son autonomie régalienne. Toutefois, il existe une contrepartie : en prenant cette décision l'équipe doit assumer de nouvelles responsabilités. Elle doit s'assurer de disposer – à l'intérieur de l'équipe – de la compétence nécessaire pour déployer, opérer et maintenir dans le temps la solution choisie. Par la même occasion, elle se prive de possibles économies d'échelle qui peuvent exister lorsque l'on se repose sur les

standards (prix de licence négocié, expertise DBA à proximité, réutilisation de scripts, etc.).

Rendre aux équipes la liberté de s'organiser, de choisir leurs outils ou leurs architectures est nécessaire mais peut parfois poser des problèmes à l'échelle. La prolifération technologique, la multiplication des langages de développement, l'hétérogénéité des pratiques méthodologiques peuvent limiter la mobilité des compétences inter-équipe, augmenter les coûts (licences, contrats de support, etc.) voire complexifier la collaboration entre les équipes. Une des solutions possibles pour garder la maîtrise de son système d'information consiste à **rendre attrayants ces standards**. Cela peut par exemple prendre la forme d'une plateforme PaaS offrant des services managés à la demande (comme une base de données MySQL managée sur *AWS Relational Database Service*). En utilisant ce service standard, l'équipe transfère une partie de ses responsabilités d'exploitation à la plateforme et se dégage du temps pour produire davantage de valeur pour ses clients. Le standard devient alors attractif pour l'équipe. Il est perçu comme **un catalyseur qui facilite le travail et augmente la productivité**.

Cette liberté retrouvée par l'équipe met fin à de nombreuses frustrations et incompréhensions ressenties par les individus et liées à des choix souvent imposés par une gouvernance centrale qui est parfois dogmatique et lointaine des réalités opérationnelles du terrain. L'épanouissement individuel et le plaisir à travailler n'en sont que boostés.

Malheureusement, dans certains grands groupes, il est encore fréquent que les choix technologiques soient imposés par le management pour des raisons politiques, en dehors de toute considération pragmatique sur la technologie.

Le troisième piège

Le troisième piège est le plus sournois, car il est presque impossible de s'en dépêtrer lorsqu'on est tombé dedans. Il s'agit du **couplage fort pouvant se créer entre les architectures de deux produits** créant ainsi des interdépendances. Prenons l'exemple d'une société audiovisuelle qui édite des sites web d'actualités. Pour les besoins d'un nouveau site, la société décide de s'organiser en deux équipes produits : l'une développant une application web de back-office permettant aux journalistes de saisir le contenu éditorial et l'autre développant le front-office qui expose le contenu sur Internet. Pour s'échanger ce contenu éditorial, les deux applications partagent la même base de données.

Entre ces deux applications, on constate alors :

- **Un couplage d'architecture fort** – de niveau 6[29] – sur le composant partagé (l'instance de base de données) et sur la structure du modèle de données.

[29] Voir les 7 niveaux de couplage de Pressman : https://fr.wikipedia.org/wiki/Couplage_(informatique).

• **Une cohérence fonctionnelle faible** (le fait de devoir faire évoluer conjointement, de manière synchrone, les deux systèmes pour implémenter une même fonctionnalité utilisateur : comme par exemple l'ajout d'images dans un article par un journaliste).

En d'autres termes, ces deux applications seront interdépendantes lorsqu'il s'agira d'implémenter une fonctionnalité à cheval entre les systèmes ou de faire évoluer le format de données ou le composant de communication. Le rythme de sortie d'une fonctionnalité sera dicté par le plus lent des deux à réaliser sa part du travail. Les *roadmaps* produits devenant ainsi liées, elles imposeront un effort important de synchronisation inter-équipes et finalement une perte réelle d'autonomie.

Le découpage de son architecture en **composants faiblement couplés**, maîtres de leur données et hautement cohérents d'un point de vue fonctionnel, est une pratique structurante à prendre en compte dès la conception de son système. Cela permet d'éviter par ce biais d'endommager l'autonomie des équipes[30] pour prioriser ses efforts sur ce qui apporte réellement de la valeur à ses utilisateurs.

[30] Corollaire à la loi de Conway : créons des architectures qui servent les objectifs d'autonomie-responsable de nos équipes.

Isolation : la distance géographique
comme frein à la collaboration

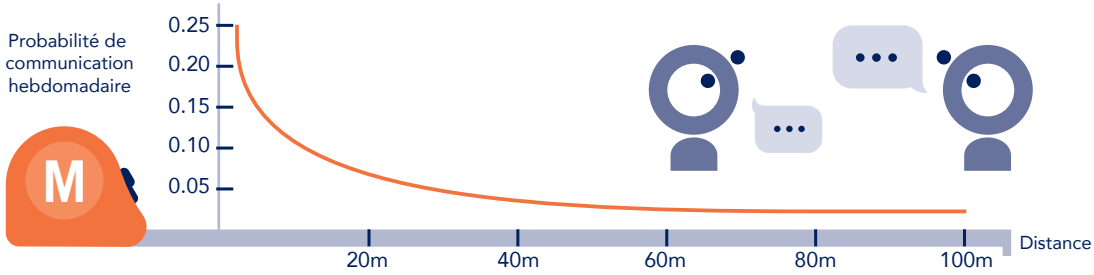
Isolation : la distance géographique comme frein à la collaboration

Dans les organisations traditionnelles, on constate fréquemment que le "mur de la confusion" qui oppose culturellement les DEV aux OPS peut incarner une réalité physique. Une porte, une cloison, un étage, un bâtiment, une ville, voire un pays séparent parfois les équipes. Il est difficile de discerner la causalité : **la distance est-elle l'origine ou la conséquence de cette collaboration douloureuse ?**

Pour Thomas Allen[31], qui a étudié la probabilité de communication au sein d'une équipe en fonction de la distance, le verdict est sans appel : ce qui est loin des yeux est loin du cœur. Au-delà de neuf mètres, il y a moins d'une chance sur dix pour que deux collaborateurs communiquent au cours de la semaine.

Cette forme d'évidence appuie l'idée d'une équipe produit qui soit co-localisée, dans un même espace, à portée de voix. Malheureusement, on constate encore fréquemment que cette réalité n'est pas prise en compte – comme un facteur essentiel de la cohésion – lors de la création d'une équipe DevOps.

Allen a également démontré[32] que la multiplication des outils collaboratifs (email, chats, etc.) ne résout pas le problème de la distance dans la probabilité de communication. Le cas de certaines communautés *open-source* qui sont extrêmement distribuées et productives reste une exception à la règle tant le niveau de maturité requis est considérable.



[31] Thomas Allen, professeur au MIT et auteur de l'ouvrage de référence *Managing the Flow of Technology* : https://blog.octo.com/ce-que-la-science-nous-dit-de-la-colocalisation/. [32] Dans son ouvrage *The Organization and Architecture of Innovation* : https://www.amazon.com/Organization-Architecture-Innovation-Managing-Technology/dp/0750682361.

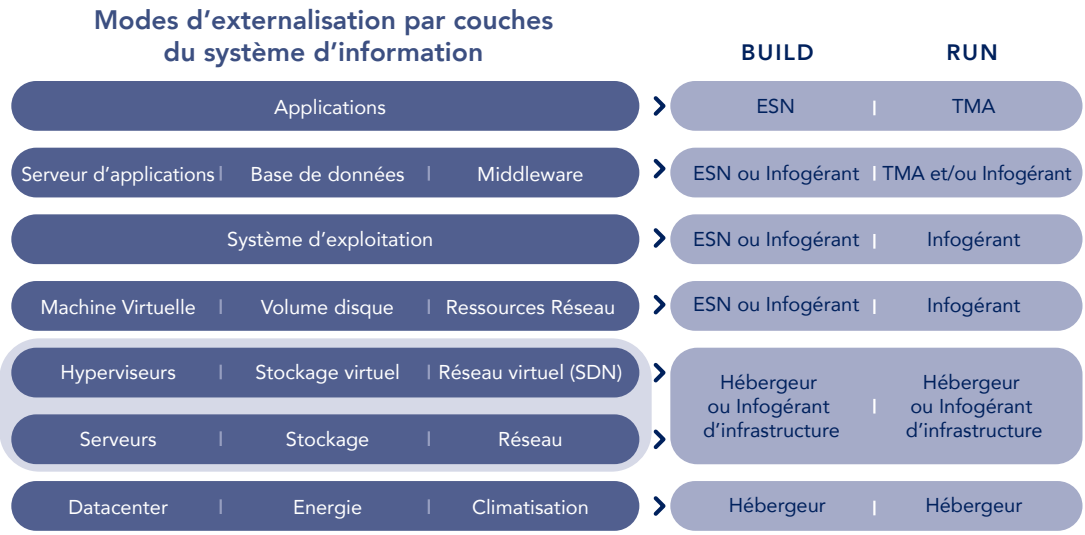
Les limites de l'externalisation

Les limites de l'externalisation

La plupart des entreprises ont recours à **l'externalisation IT** sous l'une de ses nombreuses formes. Que ce soit pour réduire les budgets de fonctionnement, faire face à un pic d'activité, avoir un accès ponctuel à une expertise rare ou encore pallier des savoir-faire absents, il est courant de se reposer sur un prestataire.

Bien que l'externalisation ne soit pas un problème en elle-même, elle peut – dans certaines configurations – **complexifier l'adoption d'une organisation DevOps**. Regardons le schéma ci-dessous qui détaille les différents modes d'externalisation traditionnels selon les couches du système d'information :

Les modes traditionnels d'externalisation



On constate ainsi qu'il existe un grand nombre d'acteurs possibles. Une même couche du SI peut être gérée par des contrats de prestations de natures diverses avec des niveaux de responsabilités variables sur le "*build*" et le "*run*".

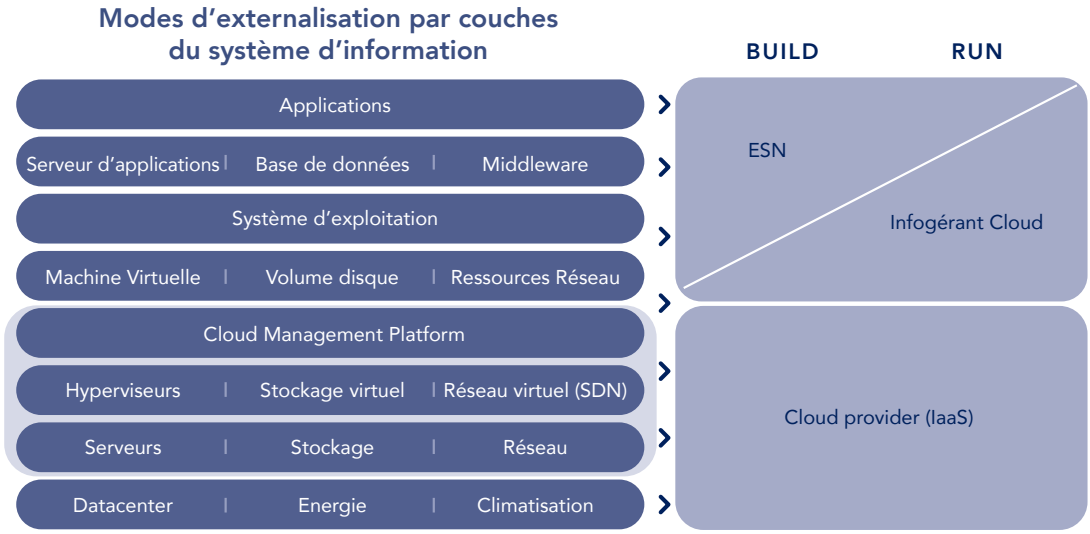
Les écueils les plus couramment constatés sont :

- Le manque de clarté sur les frontières de responsabilités avec le prestataire.
- La crise de confiance due à la distance et à une relation davantage contractuelle que collaborative.
- La multiplication des acteurs de la chaîne de valeur et la dissolution des responsabilités : notamment sur le partage du "*build*" et du "*run*".
- Le problème de souveraineté engendré par la perte des savoir-faire opérationnels sur les moyen et long termes.
- La perte de souplesse sur l'évolution de l'organisation, des méthodes et des processus (contrats de longue durée).

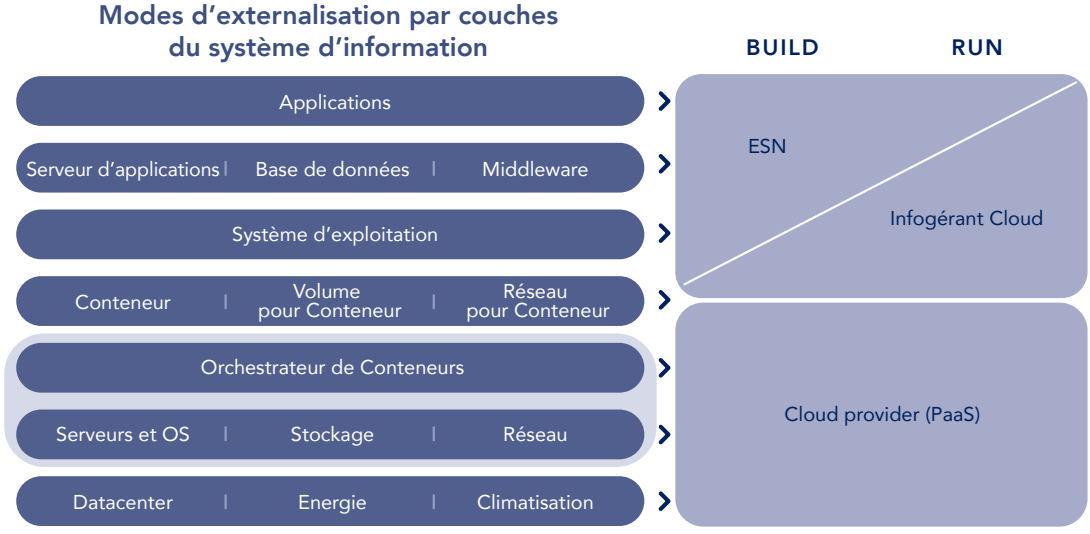
Les prestations de développements agiles au forfait ou en centres de services, l'infogérance traditionnelle et la TMA[33], sont des modèles peu compatibles avec une organisation en équipes DevOps en autonomie-responsable. Toutefois, avec **l'émergence des plateformes de Cloud** et l'évolution des offres des infogérants spécialistes du Cloud et de certaines ESN[34], on voit se dessiner un nouveau paysage de l'externalisation, davantage compatible avec DevOps. Ces offres s'appliquent que l'on soit dans une **stratégie Cloud IaaS** (historique, basée sur des capacités basiques en réseau, calcul ou stockage), ou PaaS (type services managés ou conteneurs).

[33] TMA : Tierce Maintenance Applicative. [34] ESN : Entreprise de Services du Numérique, ou anciennement Société de Services en Ingénierie Informatique (SSII), est une société de services experte dans le domaine des nouvelles technologies et de l'informatique - source : Wikipedia.

Les modes d'externalisation actuels orientés Cloud IaaS



Les modes d'externalisation actuels orientés Cloud PaaS

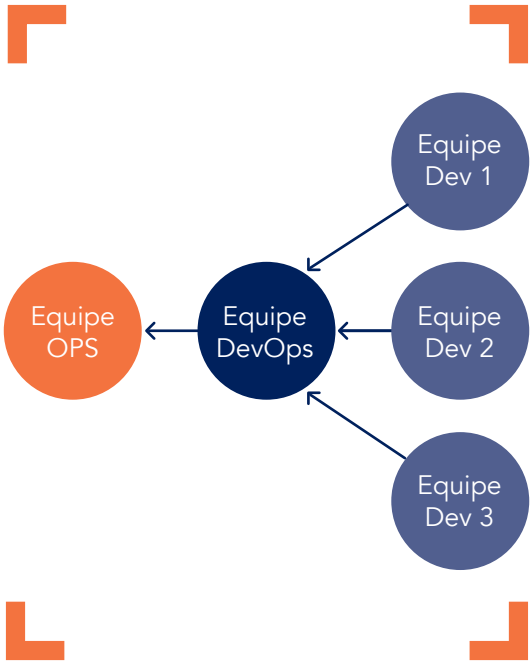


L’équipe DevOps comme proxy entre DEV et OPS

L'équipe DevOps comme proxy entre DEV et OPS

Un *anti-pattern* classique consiste à créer une équipe DevOps à mi-chemin entre les DEV et les OPS. Cette équipe est en charge de **l'approvisionnement des environnements, de l'intégration et des tests de non-régression** pour l'ensemble des applications. Il peut s'agir d'un simple renommage des équipes historiques QA[35] ou Intégration en équipe DevOps.

Bien que cette solution puisse paraître séduisante au premier abord, car peu impactante sur l'organisation, il n'est pas recommandé de s'en inspirer. En effet, ce modèle n'est pas idéal[36] en matière d'autonomie-responsable des équipes. **Ajouter une équipe de médiation augmente en réalité la difficulté de la situation** : au lieu d'un problème de frontière (entre DEV & OPS), nous en avons deux (entre DEV & DevOps et entre DevOps & OPS). Changer en profondeur l'organisation n'est pas toujours une chose facile : **conflits de pouvoir et impacts RH** peuvent parfois en résulter. Pourtant, c'est bien la seule voie possible pour une transformation durable vers DevOps.



35 QA : *Quality Assurance*. 36 Voir les anti-patterns détaillés dans les paragraphes précédents.

Implémentation rigide d’ITIL

Implémentation rigide d'ITIL

Il est indéniable que la méthode ITIL[37] a contribué à structurer les processus d'exploitation et de support pour de nombreuses entreprises. Beaucoup des concepts assemblés dans ce recueil de pratiques ont apporté clarification et rigueur dans des contextes où le *run* n'était pas toujours au niveau de maîtrise et de qualité requis.

À son origine, ITIL se veut comme un recensement de bonnes pratiques qui sont connues comme ayant donné des résultats (positifs) dans certains contextes pour certaines organisations. L'interprétation qui en est faite donne régulièrement lieu à des implémentations trop rigides et bureaucratiques, ce qui a eu pour effet d'enliser les équipes et brider leur capacité à livrer rapidement des fonctionnalités ou des correctifs en production. La version 3 d'ITIL a pris un virage en intégrant **une notion d'amélioration continue des processus**, mais la mise en application de cette dernière thématique reste très en retrait et très théorique.

Prenons l'exemple du *Change Advisory Board* (CAB) préconisé par ITIL. Il s'agit d'un comité consultatif ayant pour objectif de valider l'éligibilité à la production d'un changement ou d'une évolution du SI. En général, ce comité se réunit de manière hebdomadaire ou bimensuelle et s'assure que les changements candidats à la production sont suffisamment testés, documentés, conformes aux diverses règles d'urbanisation du SI et qu'ils ne vont pas se télescoper entre eux.

Lorsqu'il s'agit de réduire leur *Time to Market*, **beaucoup d'entreprises se heurtent aux limites de ce processus**. Il faut parfois attendre plusieurs jours voire plusieurs semaines pour autoriser la promotion d'une fonctionnalité en production. Un stock de fonctionnalités inutilisées s'accumule alors. Dans la vision *Lean*, le stock constitue de la valeur (*business)* qui n'arrive pas en production et donc qui ne rapporte pas (encore) d'argent.

Une des implémentations possibles des promesses du CAB dans un modèle DevOps reposerait sur "l'usine d'intégration et de

[37] ITIL (*Information Technology Infrastructure Library* pour "Bibliothèque pour l'infrastructure des technologies de l'information") est un ensemble d'ouvrages recensant les bonnes pratiques du management du système d'information - source : Wikipedia.

déploiement continus" ou CI/CD. Cette suite d'outils permet d'automatiser toute la chaîne de tests de non-régression intra et inter applications et va jusqu'à déployer l'application en production lorsque le niveau de qualité minimum est atteint. **La maîtrise du changement est garantie** (si la couverture de tests automatiques est suffisante, mais nous aurons l'occasion d'en reparler dans la suite de cette trilogie) et il n'aura fallu que quelques minutes entre le moment du *commit* du code par le développeur et sa mise en production effective. Victoire !

Comme l'adage le suggère, faisons attention de "ne pas jeter le bébé avec l'eau du bain". **Tout n'est pas à jeter dans ITIL**. Il faut juste repenser la manière dont ces processus sont mis en œuvre dans nos entreprises pour encore plus d'agilité et de qualité. ITIL reste pour les adeptes de DevOps **un excellent recueil de toutes les thématiques à adresser pour préparer son application ou son pan de SI à être *production-ready***. La démarche proposée ici est de travailler à supprimer les opérations humaines qui interviennent comme des garde-fous lors de ces processus et de les remplacer par des opérations automatisées qui produisent les mêmes résultats avec un niveau de confiance identique ou supérieur.

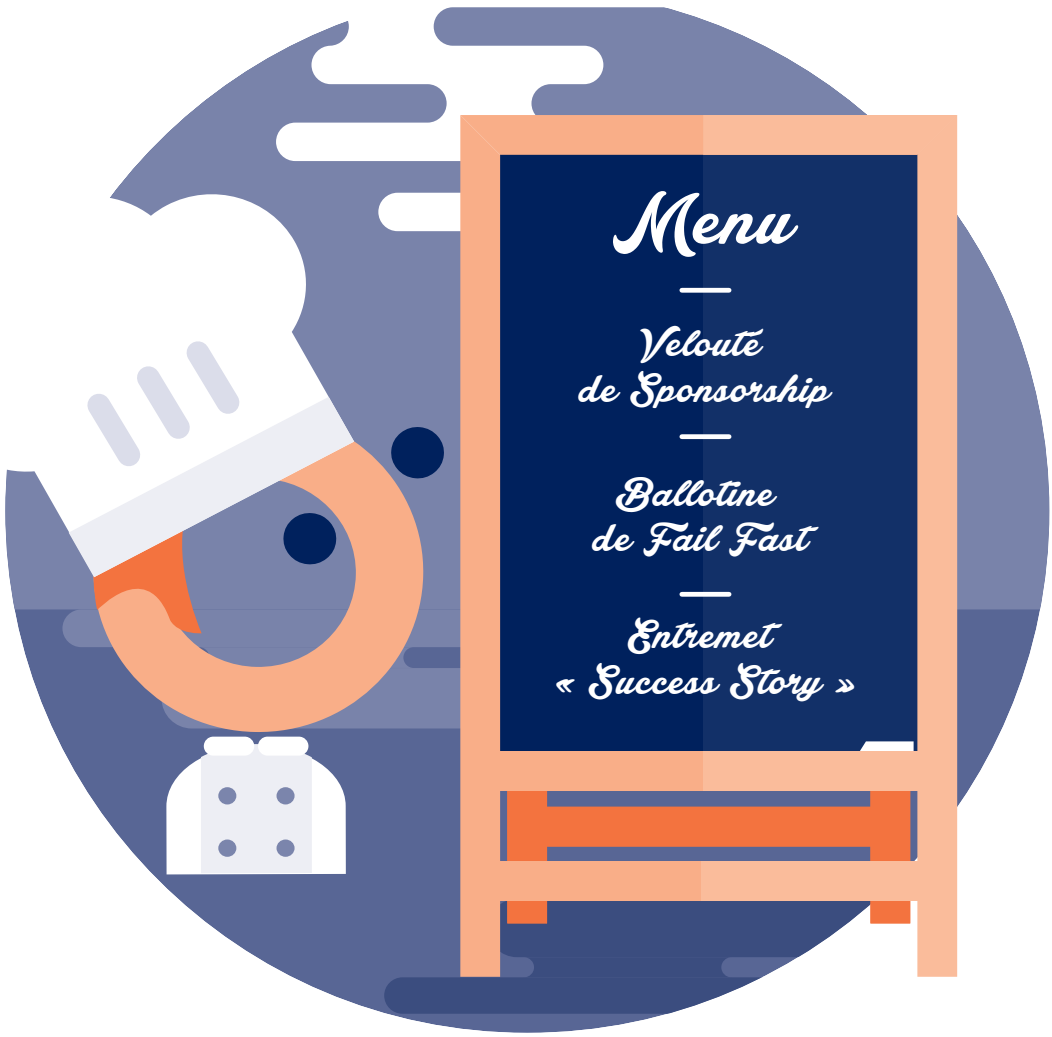
Doggy bag

On pourrait résumer les gènes d'une organisation DevOps mature comme un ***écosystème d'équipes de taille réduite, colocalisées et en autonomie-responsable*** *sur l'ensemble du cycle de vie du produit logiciel : du "build" au "run". Le découpage de ces équipes est calqué sur celui de l'architecture du SI : en fonctionnalités et en composants faiblement couplés et fortement cohérents d'un point de vue fonctionnel.*

On retrouve au sein de chacune de ces équipes une ***culture forte****, impliquant partage, amour de la qualité, transparence, liberté d'entreprendre et droit à l'erreur. Les interactions sociales sont cadencées au rythme des* ***rituels issus des pratiques de l'agile*** *et du monde de la production.*

Les objectifs du collaborateur, de l'équipe et de l'organisation sont alignés : ***l'équipe et le produit ne font plus qu'un****. L'appropriation individuelle du travail collectif se fait grâce à* ***des pratiques de binômage et de management par les pairs****.*

Atteindre ce niveau de maturité est complexe lorsque l'on ne part pas d'une feuille blanche comme ont pu le faire les Géants du Web. Alors est-il possible de transformer une entreprise traditionnelle vers ce modèle ? Et par où faut-il commencer ?



Par où commencer sa transformation ?

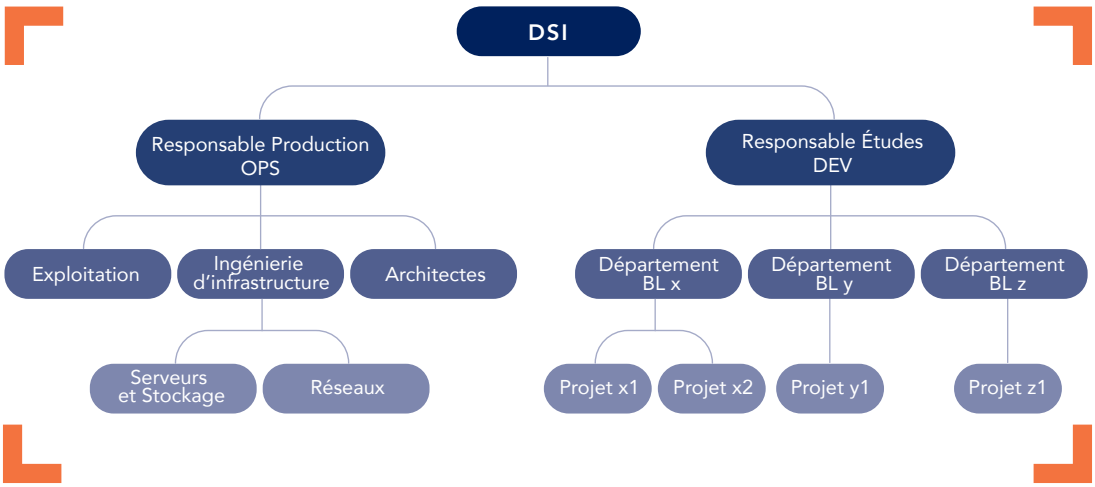
Par où commencer sa transformation ?

Il faut une forme d'humilité (ou de folie) pour s'atteler à une telle transformation. Le chemin est périlleux et semé d'embûches pour un résultat qui sera – à coup sûr – imparfait. Vous devrez **déraciner les vieilles habitudes** pour y planter la fragile **graine de la culture DevOps**. Vous aurez à l'arroser chaque jour, sans perdre patience. Vous lutterez sans répit contre les anticorps de la résistance au changement, qui vous assureront qu'il est urgent de ne rien faire.

Mais, même s'il vous incombera de relever des dizaines de challenges technologiques et humains, soyez-en certain : le jeu en vaut la chandelle !

Ayez le bon mandat d'intervention

Avant de partager avec vous les étapes clés d'une telle transformation, précisons ici qu'il est absolument indispensable de disposer du bon mandat dans l'organisation avant de se lancer.



Il est malheureusement assez courant que le sujet de la transformation DevOps soit porté par le responsable des études ou par celui de la production. Ces derniers n'ont pas autorité pour transformer l'autre. Par exemple, le responsable des études n'a pas de levier exécutif pour mobiliser des collaborateurs OPS dans ses équipes agiles. Une transformation avec un tel mandat n'aura pour conséquence que d'accentuer le fossé déjà existant entre DEV et OPS.

Le niveau de mandat minimum se situe alors auprès du Directeur du Système d'Information (DSI) qui a autorité à la fois sur les DEV et sur les OPS.

Ayez conscience de vos forces et de vos faiblesses

Nous avons largement insisté sur le risque d'une application trop littérale des pratiques organisationnelles. Une transposition pragmatique doit être réalisée, avec pour éclairage, une prise de recul sur votre contexte, vos forces et vos faiblesses. Ainsi, avant de vous lancer tête baissée dans l'action, nous vous recommandons de prendre le temps de **faire un état des lieux des accélérateurs et des freins** à une telle transformation (sur les plans organisationnels, RH, culturels, architecturaux et technologiques). Ce constat de base vous permettra par la suite de **fixer un niveau d'ambition réaliste** sur les pratiques que vous pourrez atteindre en cible.

> *"You can't directly change culture. But you can change behavior, and behavior becomes culture."*
>
> Lloyd Taylor
> VP Infrastructure, Ngmoco

Il est important par exemple, de **sonder votre maturité sur l'adoption des pratiques agiles** dans les projets. Si votre organisation est majoritairement adepte du Cycle en V, il est certainement nécessaire d'investir en priorité sur une transformation agile, comme un prérequis à DevOps. La marche à gravir est trop haute pour espérer passer directement à un mode d'organisation DevOps.

Commencez petit et échouez rapidement

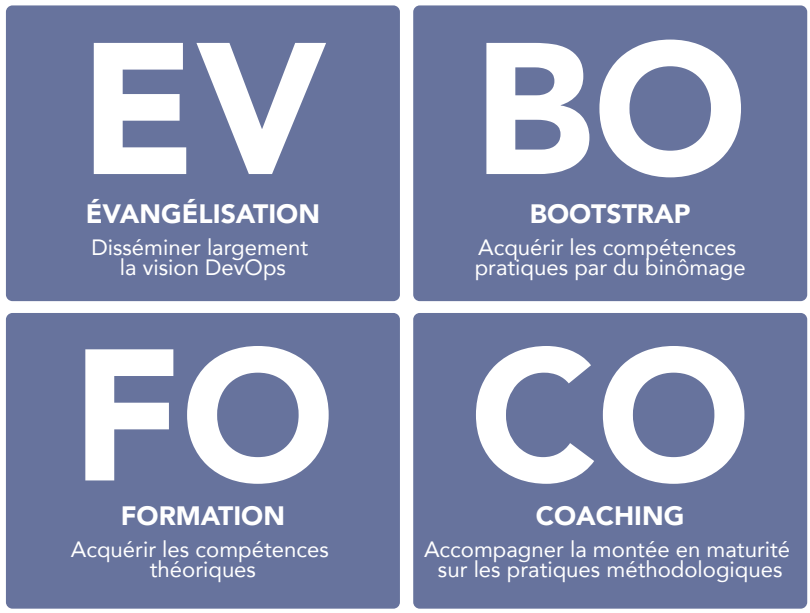
Rappelons une évidence ; une transformation de rupture ou en "*big bang*" n'est pas souhaitable tant ce genre d'approche se porte mal aux sujets d'organisation. **La culture ne se décrète pas**. Sa transformation passe par la **mise en place de nouveaux comportements** qui, de manière itérative et incrémentale, deviendront les habitudes de demain.

Dès lors, il faut être patient. Vous commencerez par une première équipe DevOps pilote qui vous permettra peut-être de vous offrir votre premier échec (ou votre premier succès). Dans un cas comme dans l'autre, vous aurez peu investi, mais vous aurez appris beaucoup.

Créez un terrain de jeu avec vos propres règles

Pour cette première équipe, mettez toutes les chances de succès de votre côté. Prenez le soin de créer un cocon douillet qui la protège du reste de l'organisation et qui assure un **niveau élevé d'autonomie-responsable**. Soyez vigilants aux différents pièges énumérés précédemment dans les anti-patterns les plus courants. Assurez-vous de disposer dans l'équipe de personnes hautement motivées pour jouer ce rôle de "cobaye". Affinez l'équilibre des **compétences de l'équipe** (DEV et OPS). Dopez ce dispositif en y injectant des **catalyseurs extérieurs** comme de l'évangélisation aux pratiques DevOps, de la formation, du *bootstrap hands-on* et du coaching méthodologique.

Protégez l'équipe de l'entropie de l'organisation. Facilitez l'**accès aux ressources** dont elle a besoin et filtrez toute forme d'ingérence pouvant perturber le focus ou l'autonomie de l'équipe. Assurez-vous que l'équipe **ritualise la collaboration et s'améliore en continu**, notamment grâce aux rétrospectives.



Passez à l'échelle viralement

Au bout d'un certain nombre d'itérations, vous aurez la chance de constater que l'équipe pilote a gagné en maturité sur les pratiques DevOps. Peut-être observerez-vous que l'équipe atteint une certaine constance dans sa **capacité à livrer de nouvelles fonctionnalités en production** dans un *Time to Market* que vous jugerez honorable. Alors, vous saurez qu'il est peut-être le moment de passer à l'échelle.

C'est sur la base de ce premier succès et du fruit de **l'expérience acquise au fil des échecs** que vous bâtirez la légitimité de votre propre modèle d'organisation DevOps. Celui qui fonctionne dans votre contexte et que nous ne pourrions généraliser ici. Maintenant, il nous faut propager ce vent nouveau au reste de l'organisation et cela se fait généralement en trois temporalités :

"In fine, pour réussir cette transformation, il vous faudra rester humble, penser grand, commencer petit, échouer vite et passer à l'échelle viralement."

1. La transformation culturelle sur le long terme

Un bon moyen d'arriver à cette fin est d'augmenter la surface de communication entre ceux qui sont porteurs du virus DevOps et ceux qui ont vocation à le devenir. La mise en place de communautés d'échange comme les *Meetups*, les *Brown Bag Lunchs*[38] (BBL), ou les immersions comme le "Vis ma vie" d'équipe DevOps, peuvent aller dans ce sens. C'est ainsi, au gré des espaces de rencontre, des recrutements, des rotations d'équipe que se diffusera – semaine après semaine – le virus de la culture DevOps.

2. La transformation opérationnelle sur le court terme

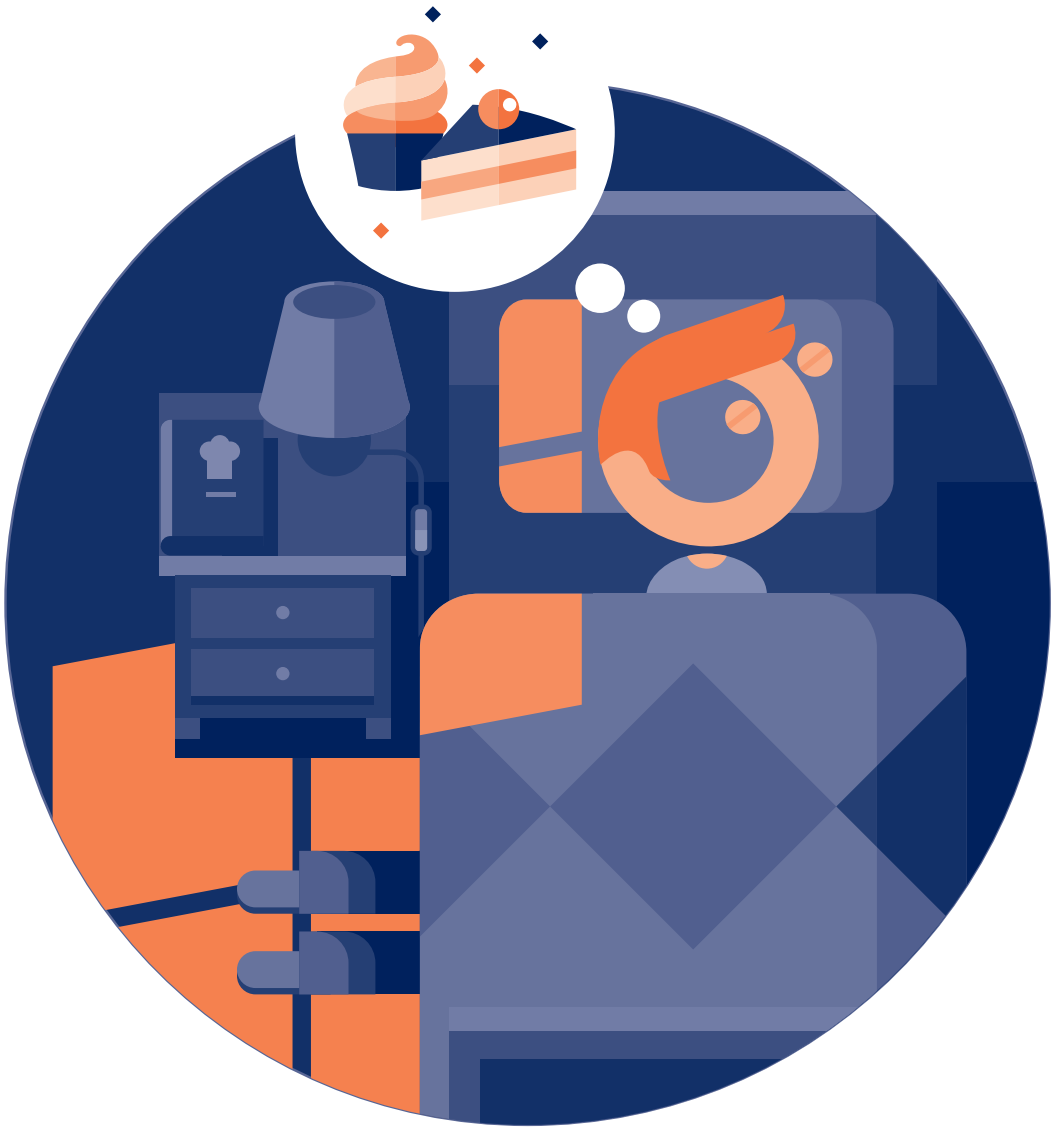
Dans cette temporalité, on lancera les chantiers permettant de faire évoluer les technologies (*cloud computing*), de refondre les processus IT et d'acquérir les compétences en vue de lancer de nouvelles équipes produits DevOps (sur le même modèle que celui de l'équipe pilote).

3. La transformation structurelle sur le moyen terme

Il s'agit ici de réformer l'organisation pour l'aplanir. On change la structure hiérarchique pour passer d'un modèle orienté silos de compétences DEV / OPS et équipes projets, à un modèle orienté équipes produits et communautés de pratiques. Pour moins de rupture dans la transformation, ce changement peut se faire "à côté" de l'organisation existante, dans une hiérarchie différente.

BBL Brown Bag Lunchs

Les BBL visent à diffuser la connaissance lors d'une session courte sur la pause de midi.
Plus en détails : https://www.infoq.com/fr/articles/bbl-fr.



DevOps :
vivre le bonheur d'avoir pour métier sa passion

DevOps : vivre le bonheur d'avoir pour métier sa passion

Nous avons la conviction profonde que malgré les difficultés d'une telle transformation, la promesse faite par DevOps de transformer votre métier en passion justifie amplement ce qu'il vous en coûtera pour y parvenir. Car oui, la routine, le désengagement, l'absence de sens, la contradiction à nos valeurs, les tensions ou l'individualisme qui règnent parfois dans les équipes IT peuvent être abandonnés au profit d'**une organisation et d'une culture équitables** pour chacun et épanouissantes pour tous. DevOps est une opportunité de prouver qu'**il n'y a pas de fatalisme dans nos entreprises** et que nous pouvons – peut-être – **vivre le bonheur d'avoir pour métier sa passion**.

Les différents patterns d'organisation DevOps, s'ils sont couplés aux traits culturels présentés dans ce livre, forment ensemble les **fondations d'un environnement de travail** qui est sain, bienveillant et sécurisant.

L'appartenance à une petite équipe dont nous connaissons personnellement chacun des membres, redonne du poids aux **relations humaines de qualité**. Notre sentiment d'appartenance est fort car notre équipe, auto-organisée, a bâti ses règles, ses rituels, son identité au profit de ce qui devient dans le temps une véritable culture d'équipe. La **responsabilité collective du succès du produit devient notre responsabilité individuelle** tant nous sommes solidaires dans l'équipe. La portée de notre action technique devient directement visible dans sa contribution aux objectifs de l'équipe, du produit et de l'entreprise. La boucle de *feedback* entre notre action et son impact en production est plus courte. Nous avons donc accès plus facilement à des **pistes d'amélioration** et à de la reconnaissance pour ce que nous réussissons. Nous retrouvons du **sens dans notre travail** et dans le "faire ensemble". Nous ne nous épuisons plus dans le *multitasking* et le *context-switching* car nous sommes focalisés sur notre action et nous ne contribuons qu'à un seul produit à la fois. Nous avons le temps de faire du travail de qualité et nous nous améliorons en permanence. Nous transmettons quotidiennement notre connaissance pour **faire grandir les autres**. **Nous sommes une équipe DevOps, et fiers de l'être**.

Le mot de la fin :

⊙ Remerciements :

Un grand merci à tous les contributeurs et relecteurs : Alban Dalle, Alexandre Raoul, Arnaud Jacob-Mathon, Arnaud Mazin, Aurélien Gabet, Christian Faure, Frédéric Petit, Joy Boswell, Ludovic Cinquin, Nelly Grellier, Nicolas Bordier, Rémi Rey, Salim Boulkour, Victor Mignot et Yohan Lascombe.

La réalisation de ce livre est le fruit d'une dynamique collective incroyable qui dure depuis une année entière et qui perdure pour donner vie aux deux prochains volumes de la trilogie. Ce projet n'aurait pas pu voir le jour sans le soutien actif d'Eric Fredj, ni sans l'inspiration et la solidarité dont font preuve chacun des membres de la Tribu OPS d'OCTO Technology : ALR, AMZ, ARJA, ASI, AUG, BBR, BGA, CHL, EDE, EFR, EPE, ETC, FRP, FXV, GLE, JGU, KSZ, LBO, MAT, MBO, MBU, MCY, MDI, MHE, NBO, PYN, RRE, SBO, SCA, TPA, TWI, VMI et YOL.

⊙ À propos de l'auteur :

Ce premier volume de la trilogie a été écrit par Edouard Devouge, consultant DevOps chez OCTO Technology.

La direction artistique et les illustrations sont l'œuvre de Caroline Bretagne.

"DevOps is a human problem"

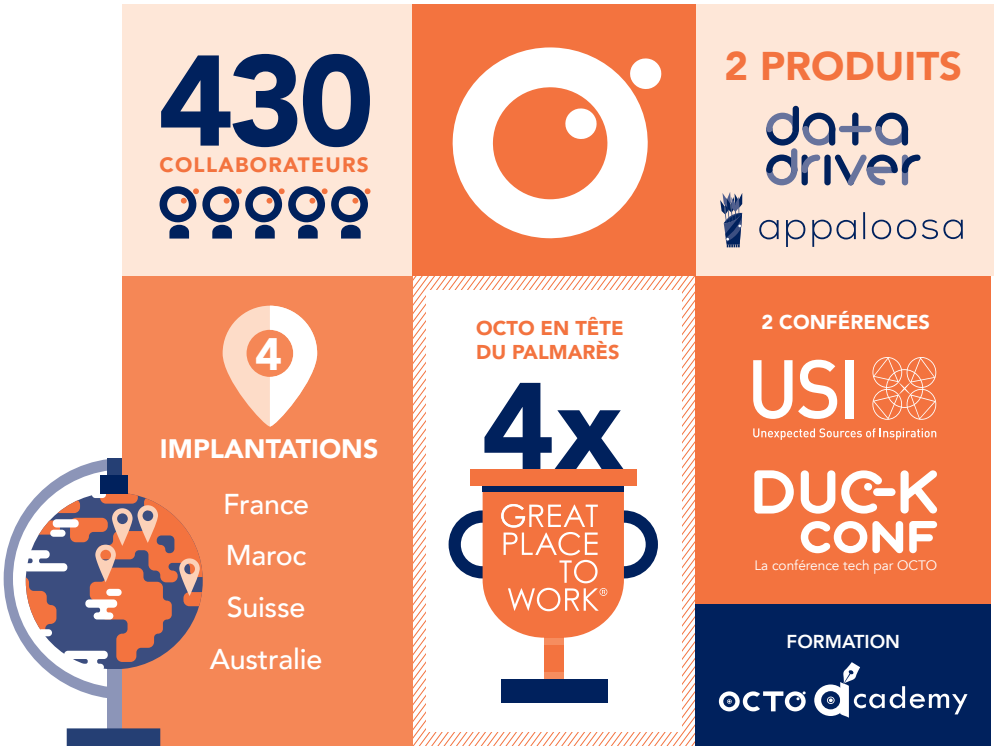
Patrick Debois

OCTO Technology

CABINET DE CONSEIL ET DE RÉALISATION IT

« Nous croyons que l'informatique transforme nos sociétés. Nous savons que les réalisations marquantes sont le fruit du partage des savoirs et du plaisir à travailler ensemble. Nous recherchons en permanence de meilleures façons de faire. THERE IS A BETTER WAY ! »

– Manifeste OCTO Technology



Dépot légal : Février 2018

Conçu, réalisé et édité par OCTO Technology.
Imprimé par IMPRO
98 Rue Alexis Pesnon, 93100 Montreuil