Culture Informatique

Que font les Google, Facebook, Amazon, Apple et autres?

Introduction

- *Les entreprises « traditionnelles » voient encore parfois l'informatique comme un moyen : un moyen pour baisser les coûts, un moyen pour produire plus, un moyen pour vendre différemment, mais toujours un moyen, un outil à la marge.*

- *Pour les entreprises du Web, les technologies de l'information sont au cœur même du produit, elles « sont » le produit. C'est la compréhension de cette « simple » différence, qui induit inévitablement des écarts voire des fossés dans les mentalités et les pratiques.*

Obsession de la mesure

Obsession de la mesure

- **Ce qui ne se mesure pas, ne se conduit pas, sans mesure, tout n'est qu'appréciation.**

- **L'informatique est l'outil de production de ces entreprises.** Leurs coûts sont donc directement corrèles à l'utilisation optimale des machines et du logiciel. Et toute amélioration du nombre d'utilisateurs simultanés ou de l'utilisation des processeurs a un ROI rapide.

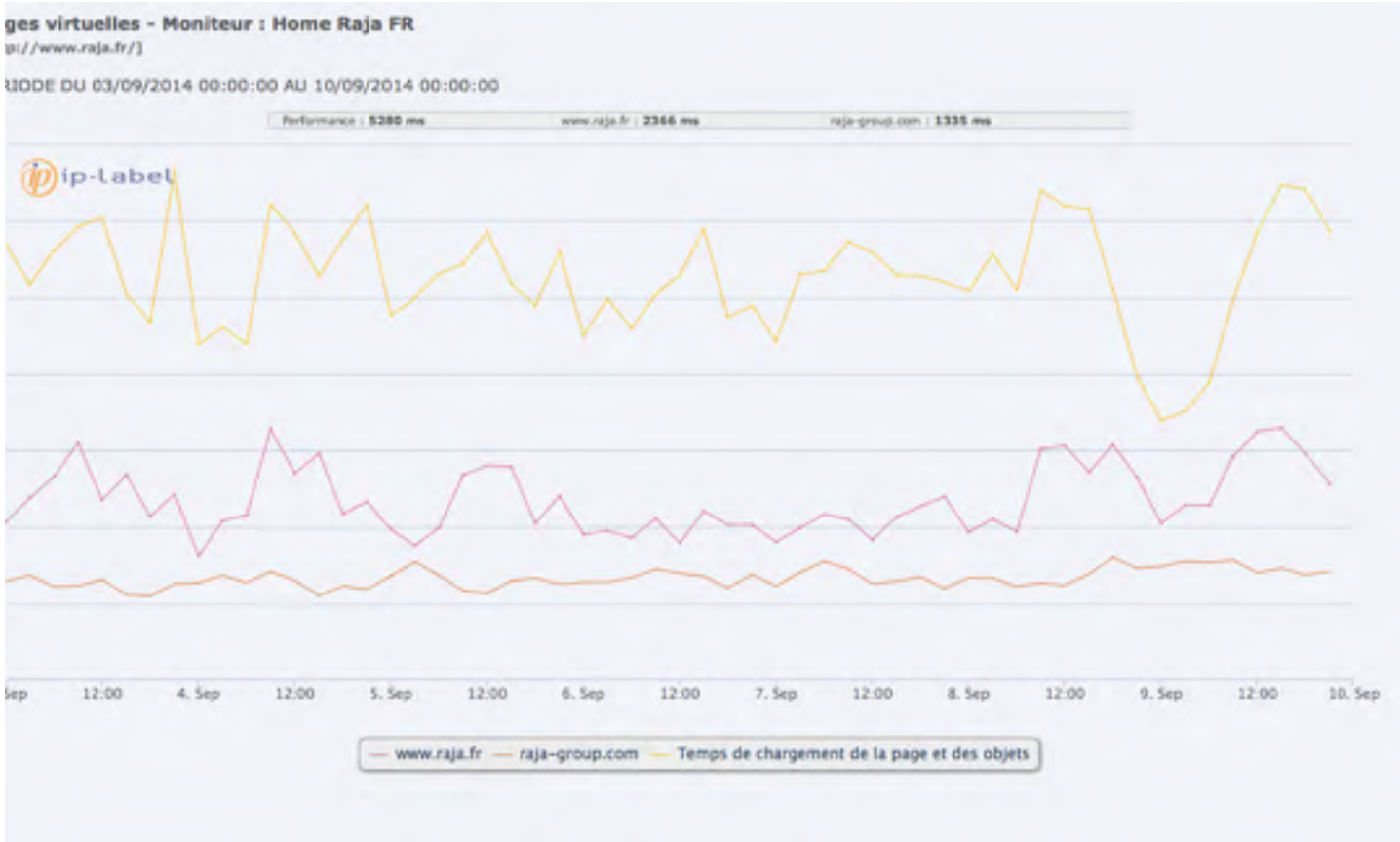
- **Les revenus sont directement corrélés à l'efficacite du service informatique rendu.** Par conséquent, l'amélioration du taux de conversion a un ROI rapide.

- **Ils ont des ordinateurs partout !** Or, ce sont de très bons instruments de mesure. Autant essayer de s'en servir !

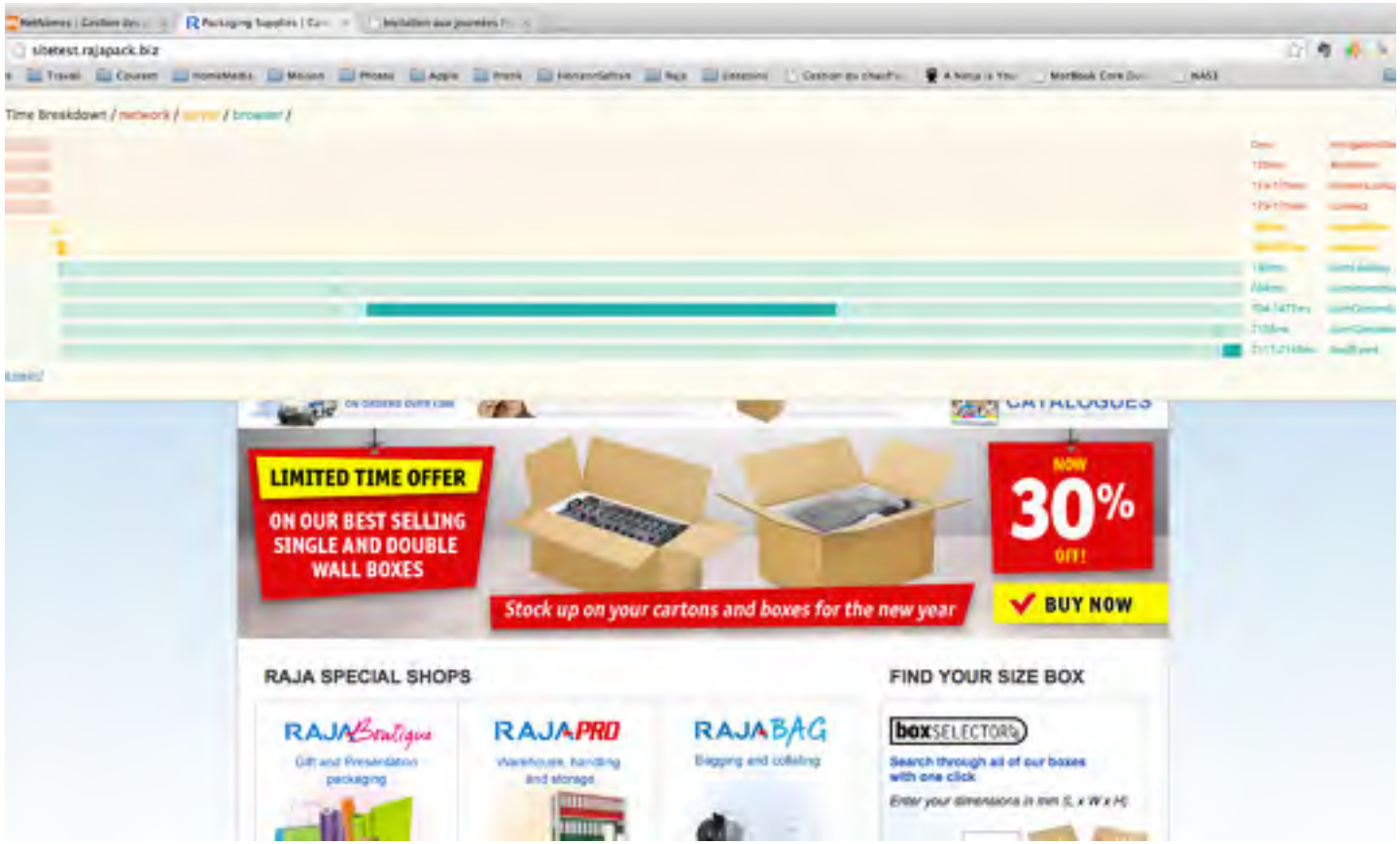
Ils mesurent tout

- Les progressions des utilisateurs
  - Certaines entreprises voyaient leur CA progresser en pourcentage chaque jour.
- La température des processeurs
- Les tests A/B
- Les MTBF
- Les requêtes
- Le temps des requêtes.
- En 2009 ils inventent Le RUM real user monitoring
- Les SLAs
  - Disponibilité des systemes
  - Les pénalités
- Application performance Monitoring
- Business activity monitoring.

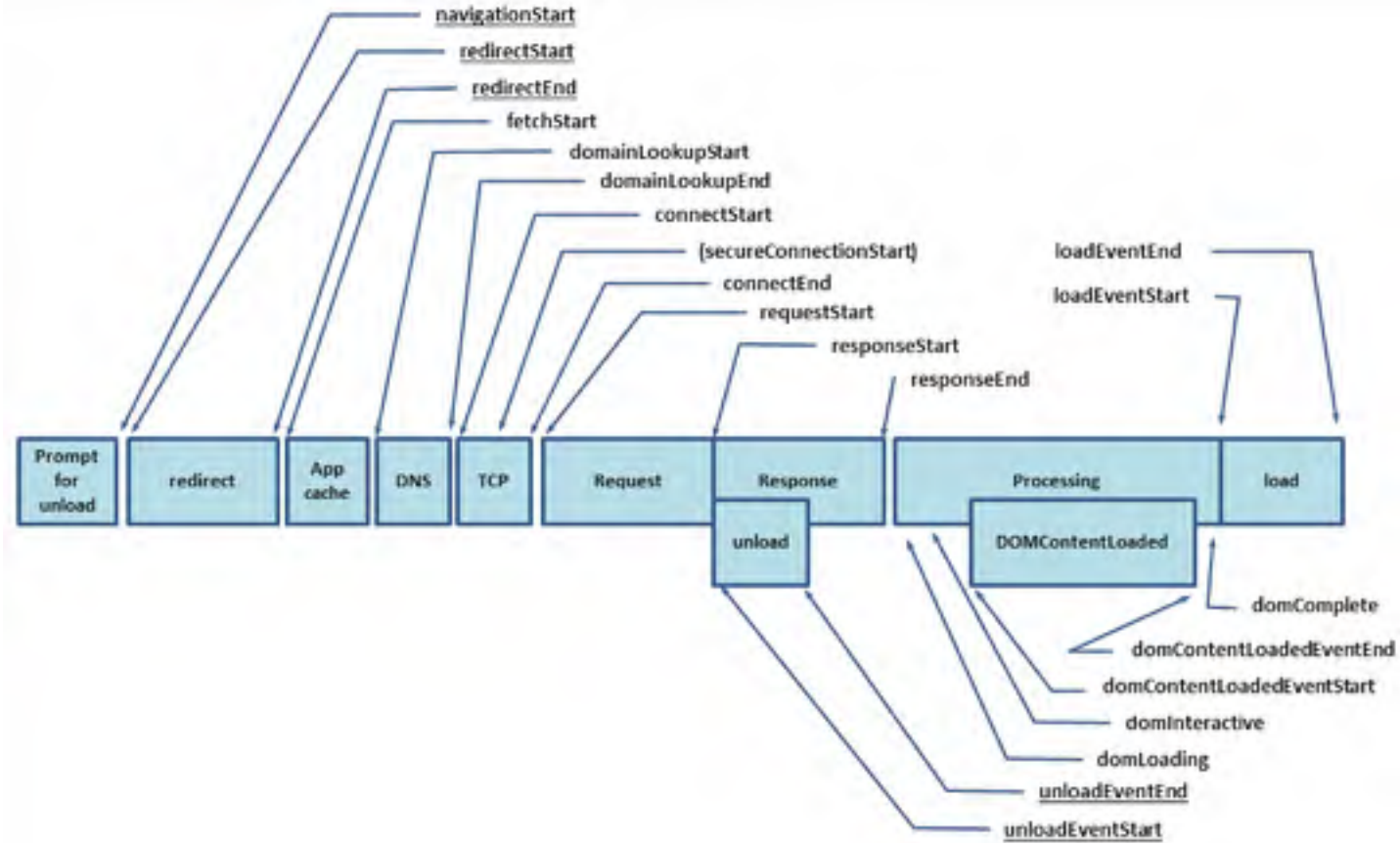
Mesure d'un site internet

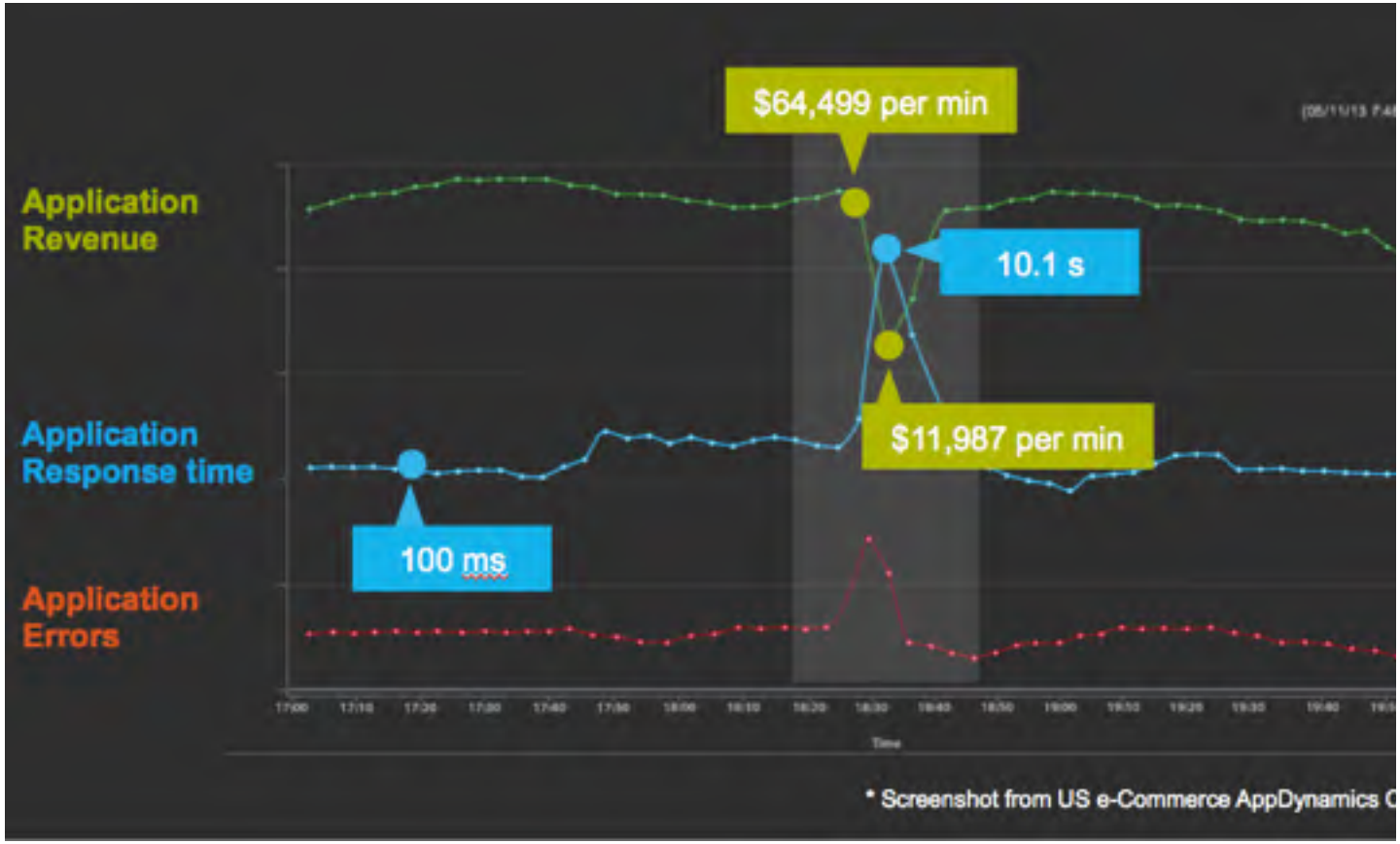


RUM (real user monitoring)



RUM (real user monitoring)




$64,499 per min
Application Revenue
10.1 s
Application Response time
$11,987 per min
100 ms
Application Errors
Time
* Screenshot from US e-Commerce AppDynamics

Ironie

- Une concrétisation extrême de cette tendance, qui frise la caricature, est l'initiative Oxygen de Google : une équipe interne de statisticiens a disséqué le matériel RH disponible dans l'entreprise (revues annuelles des collaborateurs, enquêtes de satisfaction, nominations pour les prix de managers) pour en extraire les pratiques managériales les plus efficaces. Et a énoncé à cette suite les 8 règles du bon manager. À leur lecture, n'importe quel manager expérimenté pourrait considérer que Google a réinventé l'eau chaude du management. Mais la différence, c'est qu'ils l'ont statistiquement prouvé par des métriques !

- Sélectionner les meilleurs ressources humaines en vue d'inspirer le respect mutuel et maintenir la hauteur des attentes réciproques;
- Privilégier la motivation intrinsèque puisqu'elle incite à travailler plus, à faire preuve de créativité et à rencontrer les normes de la communauté de travail;
- Miser sur les petites équipes de manière à réduire les coûts de coordination, faciliter la communication et accélérer les apprentissages ;
- Se coordonner grâce à l'usage systématique des technologies limitant ainsi les opérations liées au management et favorisant la connaissance, le partage et l'échange;
- À la manière du «couteau suisse», rendre chacun des outils indépendants afin d'accélérer l'innovation;
- Appliquer la règle du 20%, en permettant aux ingénieurs d'utiliser 20% de leur temps pour des travaux personnels sur les heures de travail;
- Offrir à la clientèle des prix justes basés sur un système d'enchères permanentes;
- Prioriser la satisfaction des utilisateurs en toutes circonstances;
- Multiplier les systèmes de mesure de satisfaction afin de réagir directement et faire évoluer les produits au plus près des pratiques des utilisateurs;
- Partager et décentraliser les données numériques pour alimenter les échanges entre ingénieurs et favoriser une culture d'entreprise fondée sur la rigueur;
- Automatiser la relation commerciale afin de réduire les coûts et étendre le bassin de la clientèle;
- Exploiter l'apport écologique des communautés qui contribuent à la réputation de l'entreprise Google, lui évitent de commettre des erreurs, l'informent de ce qui se passe dans son environnement et mettent à sa disposition compétences et ressources gratuites.

Conclusion

- Pour mesurer il est nécessaire d'avoir des étalons, donc des tests:
  - Automatisation des tests
  - Création automatiques des jeux d'essais
  - Homothétie
- Cycles courts.
  - Grace à la mesure on peut déployer sans tester!
    - S'il y a déviance des chiffres c'est qu'il y a un problème. (cdiscount, La redoute…)
- Logiciels et prestataires de mesure.

Build Versus Buy

Build Versus Buy

- ***Un écart marquant entre la stratégie des géants du Web et celle des DSI dans lesquelles nous intervenons porte sur le sujet du Build vs Buy.***

- ***Ce dilemme est vieux comme le monde de l'informatique : vaut-il mieux investir dans la fabrication d'un logiciel taillé au mieux pour ses besoins ou bien s'appuyer sur un progiciel et des outils pré-packagés qui embarquent la capitalisation et la R&D d'un éditeur (ou d'une communauté) qui a le temps de creuser les sujets technologiques et métier ?***

- La plupart des grandes DSI françaises ont tranché et ont inscrit la progicialisation maximale dans leurs principes directeurs, partant souvent du principe que l'informatique n'est pas une activité cœur de métier pour elles et qu'il vaut mieux la laisser à des acteurs spécialisés.

- Les acteurs du Web tendent à faire exactement l'inverse, avec une certaine logique puisque, précisément, **l'informatique est leur cœur de métier et, par conséquent, trop stratégique pour être laissée à des tiers.**

La juste adéquation des solutions

- L'un des défauts intrinsèques du progiciel réside dans le fait qu'il constitue le PPCM des besoins habituellement constatés chez les clients de l'éditeur. Vos besoins particuliers ne sont par conséquent qu'un sous-ensemble limité de la couverture du progiciel. Ainsi, **prendre le parti du progiciel, c'est accepter par définition d'avoir une solution trop lourde, trop complexe et non optimisée pour répondre au besoin que l'on a** et que l'on paye donc en exécution et en complexité ce que l'on gagne en n'ayant pas à investir sur le design et la fabrication d'une application complète.

Les contraintes d'un progiciel

- Ceci est particulièrement frappant dans les modèles de données des progiciels. Une grande partie de la complexité du modèle est induite par le fait que le progiciel doit être configurable pour s'adapter à différents contextes (Modèle Conceptuel de Données très normalisé, tables d'extensions, faible expressivité du modèle car il s'agit d'un méta-modèle...). Mais les abstractions et « l'hyper-généricité » que cela implique dans le design du progiciel ont un coût sur les performances à l'éxécution.

L'alternative

- D'autre part, les géants du Web ont des contraintes en termes de volumétrie, de débit de transactions et de nombre d'utilisateurs simultanés qui font exploser les approches traditionnelles d'architecture et qui requièrent par conséquent des optimisations extrêmement fines en fonction des patterns d'accès constatés. Telle transaction très sollicitée en lecture ne sera pas optimisée de la même façon que telle autre dont l'enjeu sera plutôt le temps de réponse en écriture.

- Bref, **pour arriver à de tels résultats, il faut pouvoir soulever le capot et mettre les mains dans le moteur**, ce qui n'est précisément pas possible avec du progiciel (pour lequel la garantie saute si on ouvre le boitier).

- Ainsi la performance étant l'une des obsessions des géants du Web, *l'overhead* et les faibles possibilités d'optimisation induits par la progicialisation ne sont tout simplement pas acceptables pour eux.

L'alternative

- Le volet coût est particulièrement critique quand on passe à l'échelle. Quand on multiplie les processeurs et les serveurs, la facture grimpe très vite et pas toujours linéairement, et les coûts deviennent dès lors très visibles. Qu'il s'agisse ici de progiciel métier ou de brique d'infrastructure.
- C'est précisément l'un des arguments qui a conduit LinkedIn à remplacer progressivement leur BDD Oracle par une solution maison, Voldemort!

Le choix

Business Process Outsourcing

Contribution au logiciel libre

Logiciel libre ?

- Un logiciel libre est un logiciel qui est distribué selon une licence libre. Précisément, ce sont les licences libres qui définissent les logiciels comme tels.

- Plus concrètement et de manière un peu simplifiée, cela se matérialise par le fait qu'un logiciel libre est un logiciel qui peut être utilisé, modifié et redistribué sans restriction par la personne à qui il a été distribué. Un tel logiciel est ainsi susceptible d'être soumis à étude, critique et correction. Cette caractéristique confère aux logiciels libres une certaine fiabilité et réactivité.

- Mozilla Firefox, Mozilla Thunderbird, OpenOffice.org et VLC sont des exemples de logiciels libres célèbres. Si vous avez déjà utilisé un de ces logiciels, vous avez donc déjà utilisé un logiciel libre !

Qu'est-ce qui différencie un logiciel propriétaire d'un logiciel libre ?

- La grande majorité des logiciels vendus dans le commerce sont des logiciels propriétaires, qui sont distribués en version «exécutable», alors que les logiciels libres sont fournis avec leur « code source ». Source ? Exécutable ? Un petit détour par une analogie musicale permet d'éclairer ces termes. On peut considérer le code source d'un logiciel comme la partition de celui-ci, et le code exécutable comme sa version enregistrée. Une partition peut être jouée sur un piano, une flûte ou par l'orchestre philharmonique de Berlin. En revanche un enregistrement pressé sur disque ne permet pas de modifier la musique, de changer d'instrument ou de moduler l'interprétation.

- Le passage de l'un à l'autre s'opère par traduction du code source (lu et écrit par l'homme) en code exécutable (que seul l'ordinateur comprend). Les logiciels libres sont distribués sous ces deux formes, tandis que Microsoft ou Adobe ne vendent que le code « exécutable » et cachent le reste.

Qui crée des logiciels libres ?

- Toute personne (informaticiens, graphistes, musiciens, traducteurs, relecteurs, testeurs, etc.) désireuse de bénéficier de créations collectives et/ou souhaitant maximiser la diffusion de ses œuvres (logiciels, textes, images, vidéos, musiques).
- A la tête de chaque projet de logiciel libre il y a une structure plus ou moins formelle qui est composée de simples particuliers et/ou d'entreprises.
- Par exemple, la FSF, dirigée par Richard Stallman, produit et/ou organise le développement de logiciels libres. Ainsi le projet GNU (dont le logo est, bien sûr, un gnou) de la FSF a joué un rôle déterminant dans la création de Linux (dont le logo est un manchot).

Mais pourquoi les géants du Web comme Facebook, Google et autre Twitter contribuent-ils autant à l'Open-Source ?

- L'avance technologique est un atout important dans la conquête du Web. Que ce soit pour se démarquer de la concurrence en lançant de nouveaux services (pensez à la sortie de Gmail et de son large espace de stockage à l'époque de l'hégémonie Hotmail), ou plus pragmatiquement pour faire face aux contraintes qui leur sont propres comme le défi de croissance lié à la croissance de leurs bases utilisateurs, les géants du Web ont su à plusieurs reprises inventer de nouvelles technologies.

- Alors que l'on pourrait penser que cette maîtrise technologique, et cet actif que représente le code, devraient tous deux être secrètement conservés, voilà qu'un pattern largement répandu nous interpelle : les géants du Web ne sont pas seulement de grands consommateurs de technologies open-source, ils en sont aussi les principaux contributeurs.

- Le pattern « contribution au logiciel libre » consiste ainsi à rendre public un outil logiciel (librairie, framework...) construit et utilisé en interne. Le code est mis à disposition sur un serveur public, comme GitHub, et une licence libre de type Apache, par exemple, autorise son utilisation et son adaptation par d'autres sociétés. Ce code devient par la même occasion potentiellement ouvert aux contributions des développeurs du monde entier. Notons que ce passage à l'Open-Source s'accompagne traditionnellement d'une large communication en ligne et lors de conférences dédiées aux développeurs.

Chez qui ça fonctionne ?

- Les exemples sont nombreux. Parmi les plus représentatifs, on pense à Facebook et à sa base de donnée Cassandra, construite pour gérer des quantités massives de données réparties sur plusieurs serveurs. Il est intéressant de noter que dans les utilisateurs actuels de Cassandra on peut compter d'autres géants du Web comme Twitter ou Digg. Alors que dans le même temps, Facebook a abandonné Cassandra au profit d'une autre solution de stockage également *Open-Source* – HBase – initiée quant à elle par la société Powerset. A l'image de la mouvance NoSQL, les nouvelles fondations du Web d'aujourd'hui sont massivement issues des technologies des géants du Web.

Pourquoi ça fonctionne ?

- Ouverture et gratuité ne s'opposent pas forcément à guerre économique et rentabilité. D'une certaine façon cette ouverture pourrait même apparaître comme un moyen d'annihiler l'émergence de la concurrence sur certains pans technologiques. Faire un don à l'Open-Source permet de redessiner les contours d'un secteur technologique, tout en s'assurant de rester influent sur la meilleure solution disponible. A ce jour, Google est toujours le principal sponsor de la fondation Mozilla et de son projet phare Firefox, à plus de 80 %... Un moyen de diversifier la concurrence face à Microsoft ? Mais revenons sur l'analyse de nos trois bénéfices.

Améliorer l'image de marque

- En ouvrant à la communauté une technologie de pointe, les géants du Web se forgent une image de leaders, de pionniers. Ils communiquent implicitement sur l'esprit d'innovation qui règne dans leurs murs, sur la recherche permanente. Ils font passer le message qu'ils sont en mesure de résoudre de grands problèmes ; ils sont puissants technologiquement. Livrer un framework open-source qui trouve le succès, c'est montrer que l'on a su résoudre avant tout le monde, ou mieux que les autres, un problème partagé. Et que d'une certaine façon ce problème est maintenant derrière eux. Ils l'ont mis en boîte et continuent de bâtir. Ils gardent une longueur d'avance.

- Ouvrir un framework à l'Open-Source, est en soi une action de communication forte, un renforcement de l'image de marque. Un moyen de faire passer à l'utilisateur le message implicite et primaire : « Nous sommes les meilleurs, sois rassuré ».

- Et puis, comme pour se prémunir d'apparaître comme les nouveaux Big Brothers, on ne peut s'empêcher de lire entre les lignes de cette démarche « Nous sommes ouverts – gentils – n'aie pas peur ».

Attirer – et conserver – les meilleurs

- Voila un aspect essentiel pouvant être induit par une démarche Open- Source. Car « afficher son code » c'est afficher une partie de son ADN, de son mode de pensée, de sa façon de résoudre les problèmes – Montre-moi ton code, et je te dirai qui tu es. Un moyen naturel de communiquer vers l'extérieur sur ce qu'il se passe précisément à l'intérieur de son entreprise : le niveau de ses développeurs, ses standards de qualité, les sujets qui préoccupent le quotidien des équipes… Un bon moyen d'attirer des développeurs « compatibles » qui se seraient intéressés d'eux-mêmes aux projets soutenus par l'entreprise.

- Grâce à ces contributions, il devient alors facile de détecter les développeurs les plus impliqués, les plus compétents, les plus motivés. Et de les embaucher alors avec la garantie de leur capacité à s'intégrer dans leur écosystème. D'une certaine façon, l'Open-Source est un peu comme une immense période d'essai ouverte à tous.

Attirer – et conserver – les meilleurs

- Attirer les meilleurs geeks est une chose, les garder en est une autre. Sur ce second point, l'Open-Source peut s'avérer être un formidable moyen d'offrir aux meilleurs développeurs de votre société une vitrine sur le monde extérieur, une tribune.
- A partir de maintenant, ils pourront briller autant à l'intérieur qu'à l'extérieur de l'entreprise. Favoriser l'Open-Source c'est permettre aux développeurs de prendre soin de leur CV. C'est comprendre le besoin de Personal Branding[3] de vos équipes, tout en les conservant au sein de l'entreprise. Tout développeur recherche un environnement qui valorise les développeurs, un environnement qui propose une carrière aux profils techniques. Parole de développeur.

Améliorer la qualité

- « Penser Open-Source » permet déjà en soi de faire un bond en avant niveau qualité: ouvrir un morceau de code – un framework – à la communauté, nécessite tout d'abord d'en définir les contours, de le nommer, de décrire ce framework et d'en définir le but. À elle seule, cette démarche est un considérable pas en avant dans la qualité de vos logiciels. Car elle induit inévitablement la modularisation du code, sa structuration. Elle facilite également la réutilisation de code au sein de l'entreprise. Elle définit des responsabilités dans le code, et peut-être même au sein des équipes.

- Inutile de préciser qu'un développeur qui sait que son code sera relu (a fortiori relu par des développeurs de la terre entière), s'y reprendra à deux fois avant de « commiter » cette méthode non testée, ou ce bout de code fait à la va-vite. Au-delà de cet effet responsabilisant, récolter du feedback de la part de développeurs externes à l'entreprise sera sûrement salvateur.

Pizza Teams
Feature teams

Quelle est la bonne taille d'équipe pour fabriquer un produit logiciel remarquable ?

- La sociologie des organisations s'est penchée sur le sujet de la taille des équipes depuis plusieurs années déjà. Si la réponse n'est pas uniforme à effectuer, le niveau moyen et la diversite de l'équipe, un consensus émerge sur des tailles de 5 à 12. En decà de 5, l'équipe devient fragile aux événements extérieurs et manque de créativité. Au-delà de apparaitre des comportements de parasitisme et des luttes de pouvoir, et la performance de l'équipe décroît très rapidement avec le nombre de membres.

- Cela s'applique évidemment aussi en informatique. Le cabinet Quantitative Software Management, qui s'est fait une spécialité de la conservation et de l'analyse de métriques sur les projets informatiques, a ainsi publié quelques statistiques intéressantes. Sur un échantillon de 491 projets, le cabinet QSM a mesuré une baisse de productivité et une augmentation de la variabilité lorsque la taille des équipes augmente, avec un décrochage assez net à partir de 7 personnes. De façon corrélée, la durée moyenne des projets augmente et l'effort de développement explose surtout au-delà de 15

- « La taille d'une équipe ne dépasse pas le nombre de personnes que l'on peut nourrir avec **2 pizzas** (modèle américain, tout de même), soit de l'ordre de 8 personnes. »
  - Werner Vogels (CTO et VP Amazon)

- Pour illustrer l'importance que les géants du Web accordent à la dynamique d'équipe, Google a recruté Evan Wittenberg en tant que responsable Global Leadership Development ; cet ancien universitaire s'est fait connaître (entre autre) par des travaux sur les tailles d'équipe.

- Même hygiène chez Yahoo! qui limite la taille de ses équipes produit la première année à 5 à 10 personnes.

- Viadeo, de son côté, pratique les pizza teams de taille française, avec des équipes composées de cinq à six personnes.

- Dans le domaine des startups, Instagram, Dropbox, Evernote... sont connues pour avoir maintenu le plus longtemps possible une équipe de développement très réduite.

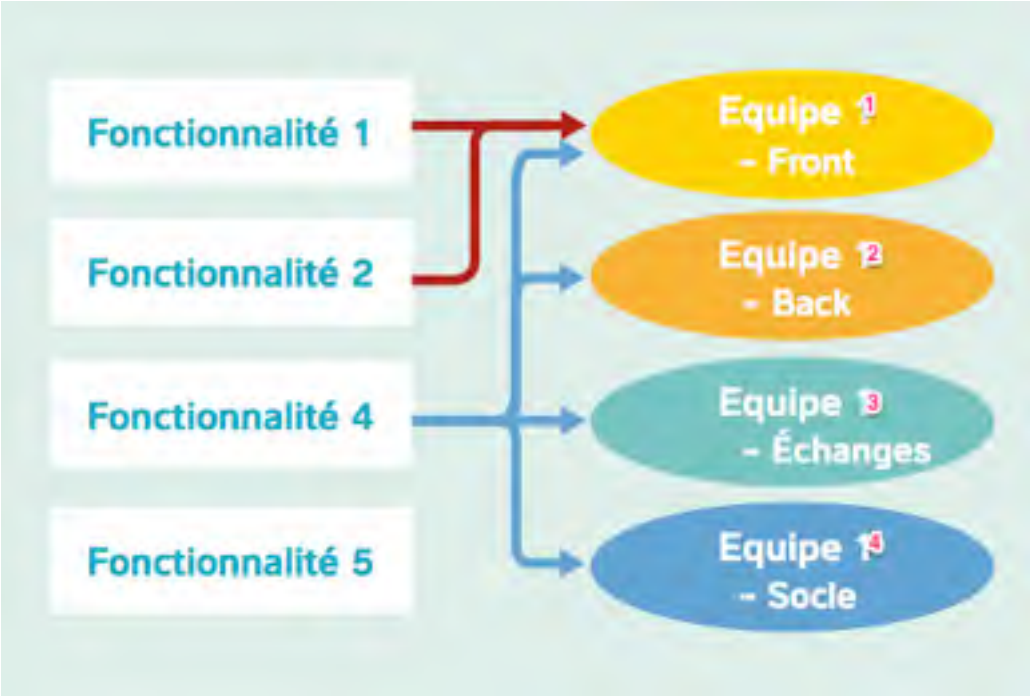
OK, mais comment faire sur un projet important!

- Il existe deux grandes options :
  - Le découpage par couche « technologique ».
  - Le découpage par « pan fonctionnel ».

Le découpage par couche « technologique ».

- le découpage par couche technologique consiste à dédier chaque équipe sur un type de technologie : typiquement, la couche présentation, la couche métier, les socles transverses, la base de données...

Le découpage par « pan fonctionnel », le problème



- L'équipe 1 est surcharge de travail avec les fonctionnalités 1 et 2
- Mais les 4 équipes ne peuvent travailler ensemble sur la fonctionnalité 4 car on attend tous l'équipe 1
- Moralité on a 3 équipes au chômage ou qui ne font que du travail de TMA.

La « Feature Teams »

- Les feature teams permettent de corriger ces défauts : chaque équipe travaillant sur un sous ensemble fonctionnel cohérent -et ce indépendamment des technologies - elle est capable à tout moment de livrer de la valeur final en en dépendant faiblement des autres équipes. Cela implique que dans une même équipe se retrouvent toutes les compétences nécessaires à la production de fonctionnalités, et cela peut vouloir dire (entre autre) un architecte, un ergonome, un développeur Web, un développeur Java, un expert base de données, et, même un exploitant… car quand on pousse la logique à l'extrême, on tombe sur le « you build it, you run it » de DevOps

Comment assure-t-on la cohérence technologique de ce qui est produit ?

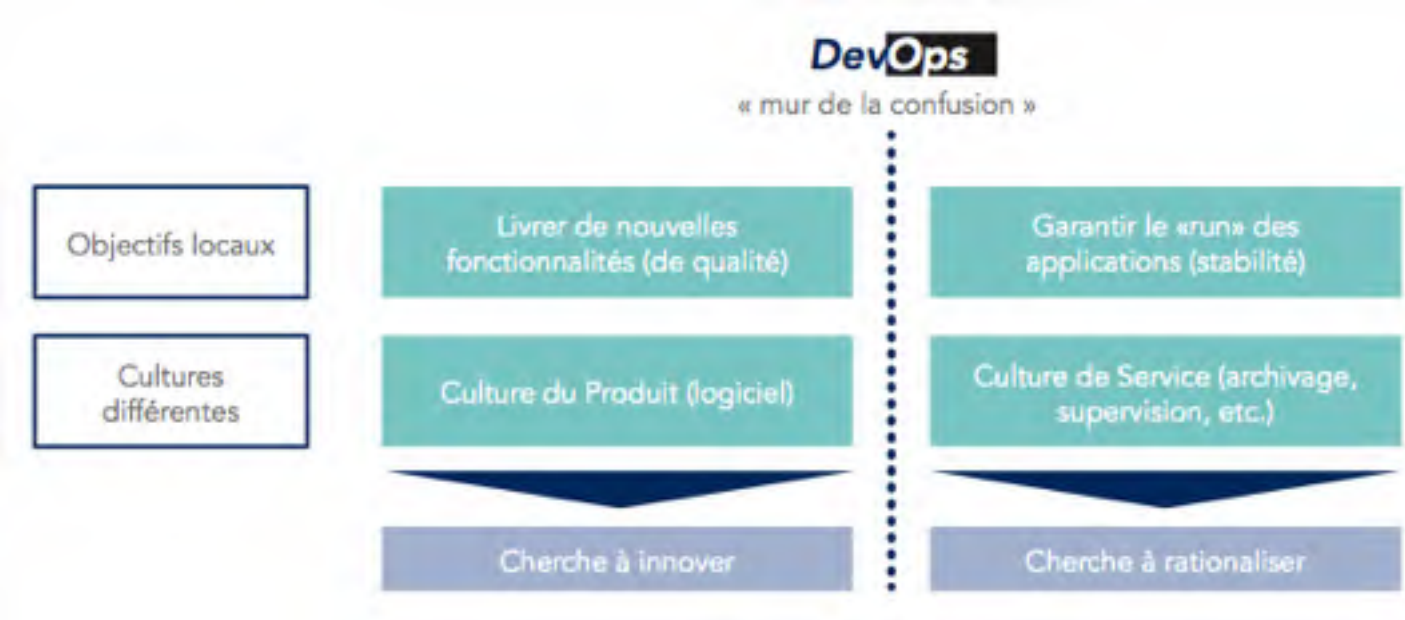
- Ce point est adressé par le principe des communautés de pratiques. Les pairs de chaque type d'expertise se réunissent à intervalles réguliers pour échanger sur leurs pratiques et s'accorder sur des stratégies technologiques par rapport à la fabrication du produit en cours.
- Les feature teams présentent également un intérêt induit non négligeable : les équipes progressent très vite sur le métier et cela favorise l'implication des développeurs dans la vision produit et dans la qualité du résultat final.

DevOps

Dev Ops ?

- Le mouvement « DevOps » nous invite à repenser la frontière classique de nos organisations qui séparent d'un coté les études, i.e. ceux qui écrivent le code des applications (les « devs ») et de l'autre coté la production, i.e. ceux qui déploient et exploitent ces applications (les « OPS »).
- D'un coté les agilistes qui ont levé la contrainte coté développement, et sont maintenant capables de livrer beaucoup plus souvent du logiciel valorisé par le client… de l'autre, des experts ou des managers de la «prod » des géants du Web (Amazon, Facebook, LinkedIn…) partageant leurs retours d'expérience et la façon qu'ils ont d'aborder la frontière « Dev » et « Ops ».

Le mur de la confusion



Les études

- Les études recherchent plus de réactivité (sous la pression du métier et du marché notamment) : il faut aller vite, ajouter de nouvelles fonctionnalités, réorienter les directions, refactorer, upgrader les frameworks, déployer rapidement sur de nouveaux environnements pour tester… C'est la nature même du code (« software ») : malléable, adaptable.

La méthode Agile

- L'Agile est apparu en France il y a maintenant plus de dix ans avec comme principal objectif de lever la contrainte autour du processus de développement.
- Cette méthodologie introduisait les notions de cycles courts, de feedback terrain et client, la notion de Product Owner, une personne du métier responsable de la roadmap, des priorisations…
- L'Agile a également mis à mal l'organisation traditionnelle (dans les DSI françaises tout du moins) en introduisant des équipes pluri-disciplinaires (du métier aux développeurs) et challengeant du même coup les découpages organisationnels.
- Aujourd'hui, lorsque ces contraintes sont levées, les développements suivent le plus fréquemment des itérations de une à deux semaines. Les métiers voient le logiciel évoluer durant la phase de construction.

Les opérationnels

- **La production a besoin de stabilité et de standardisation.**
- Stabilité, car il est souvent difficile d'anticiper quels impact aura telle modification de code, d'architecture ou d'infrastructure . Exemple : Un disque ,local qui devient un disque réseau mais impacte les temps de réponse, ou bien un changement de code qui impacte la consommation CPU et ainsi le Capacity planning.
- Standardisation, car les processus doivent être joués intégralement avant leur mise en production.

Les attentes des Devs sur les opérationnels

- **Approvisionnement / mise en place des environnements** : dans la plupart des entreprises, l'approvisionnement d'un environnement peut demander de un à quatre mois (pourtant aujourd'hui en environnement virtualisé). C'est étonnamment long, surtout face à des challengers comme Amazon ou Google…

- **Déploiement** : cette phase est certainement celle qui cristallise le plus le problème et crée le plus d'instabilité ; les équipes agiles se limitent parfois à un déploiement par trimestre pour limiter les impacts en production et garantir la stabilité du système, d'autant plus que ces déploiements sont souvent manuels (donc longs, sources d'erreur...), bref, risqués.

- **Résolution d'incidents** et prise en compte des besoins non fonctionnels : les acteurs de la production sont les autres utilisateurs de l'application. La facilité à diagnostiquer, expliquer les problèmes, les enjeux de résilience, robustesse doivent être pris en compte.

Le Dev Ops tente la réconciliation

- **L'infrastructure as code**, il s'agit de pouvoir déployer de nouveaux environnements comme nous le ferions avec du code informatique pour des programmes. Cela implique l'automatisation de la création des environnements. Ainsi on les met en place pour un test puis on les détruit une fois celui-ci terminé

- **Le continuous delivery**, c'est faire en sorte que les cycles de déploiement soient les plus cours possibles. Changements plus petits-risques plus petits-métier plus satisfait!

- La culture de la coopération, passe par le partage des métriques, dans les deux sens, tant l'augmentation des temps de réponses que du CA, des logs et l'acceptation par les développeurs de l'intégration des contraintes opérationnelles. Enfin ce sont des approches comme la gestion des « Post-Mortem » une démarche « Lean » ou la gestion des « 5 Pourquoi »

Device Agnostic

Les géants du web ne négocie pas avec l'ergonomie!

- Déjà en 2003, le manifeste du Web 2.0 plaidait pour une « Rich User Experience », et aujourd'hui la nécessité d'offrir la meilleure interface utilisateur possible fait consensus dans le monde du Web. Elle est considérée un facteur important de gain de parts de marché.

- ***anywhere/anytime***

- On distingue en général les situations sédentaires (ex. : usage au bureau), nomades (ex. : usage en position assise dans un aéroport) ou mobiles (ex. : usage en marchant dans la rue).
- 3 types d'appareils :
  - Les PC de bureau
  - Les portables ou tablettes
  - Les mobiles

- Le pattern « Device Agnostic » il est indispensable d'offrir la meilleure ergonomie possible quelle que soit la situation d'usage et le device.
- L'une des premières sociétés à avoir développé ce pattern est Apple avec l'écosystème iTunes. En effet, Apple a rendu la musique accessible d'abord sur PC/Mac et iPod, puis par la suite sur iPhone et iPad. Apple couvre donc les trois situations d'usage. Par contre, Apple n'applique pas complètement le pattern car la musique n'est pas accessible sur Android ou Windows Phone.

- Pour mettre en œuvre cette approche, il est parfois nécessaire d'offrir autant d'interfaces qu'il existe de situations d'usages. En effet, une interface générique de type «One size fits all» ne permet pas cette ergonomie optimale sur ordinateur, tablette, smartphone, etc.

- Les géants du Web investissent donc dans le développement de nombreuses interfaces, ce qui est rendu possible par l'application du pattern «API First». Selon ce principe, l'architecture de l'application doit reposer sur une API générique, les différentes interfaces seront ensuite développées directement par l'entreprise ou indirectement par l'écosystème des développeurs et partenaires sur la base de cette API.

APIs First

- Pour tirer le meilleur parti de chacun des terminaux, il est aujourd'hui délicat de ne réaliser que des interfaces WEB, ces dernières ne gèrent pas les fonctionnalités propres des devices (push, capteur photo ou vidéo, accéléromètre, etc.). Elles donnent aussi à l'utilisateur une impression de manque de réactivité, car elles nécessitent le téléchargement initial de tout le contenu[1], là où les applications natives peuvent fonctionner sans rien ou en téléchargeant éventuellement quelques ressources XML ou JSON.

- Chez qui cela fonctionne :
  - Facebook
    - avec Messenger
    - Avec Facebook sur mobile
  - Evernote