

TECH/TRENDS

#1

MARS
2013

PERCEVOIR LE FUTUR



FOCUS
SUR

BIG DATA

Big Data, une définition communément admise / Ce que nous voyons réellement derrière / L'obsession de la mesure
Les outils innovants / Le champion : Hadoop / Les challengers complémentaires / Les champs d'application

Également dans ce numéro...

Les Applications Web d'entreprise en 2013

Cloud Computing

Programmation fonctionnelle

Les stratégies d'entrée
sur le marché mobile

2013 la phase de maturité
de l'agilité

Contents

TechTrends : percevoir le futur.	5
Big Data	7
Big Data, une définition communément admise	7
Les innovations technologiques	8
Ce que nous voyons réellement derrière	8
L'obsession de la mesure	10
Le champion : Hadoop	10
Les challengers	12
Les champs d'application	13
Architecture Orientée Web	17
Les Applications Web d'entreprise en 2013	17
Playframework 2.0	21
Les frameworks JavaScript	22
Caching	23
Les services Web, langage universel	25
Infrastructure d'entreprise	29
Cloud Computing	29
Conteneurs légers	32
Messaging	33
Bases de données non relationnelles	34
Langages de la JVM et leurs amis	41
Programmation fonctionnelle	41
Java 8	44
Les cousins de JavaScript	45

L'écosystème Groovy	46
Domain Driven Design en Entreprise	48
Mobiles en entreprise	53
Les stratégies d'entrée sur le marché	53
Développer avec les outils Cloud	61
L'état du marché des OS mobiles	64
Agilité et Craftsmanship	71
2013 la phase de maturité de l'agilité	71
DevOps	74
Software Craftsmanship	79
Coaching agile	80
Test Driven Development et Refactoring	82
Agile Games	83
Expérience utilisateur	85
Xebia	91

TechTrends : percevoir le futur.

Le monde des nouvelles technologies évolue de plus en plus vite et apporte chaque année son lot d'usages nouveaux et d'innovations, tous plus prometteurs les uns que les autres.

Cependant, cette foison de nouveaux concepts technologiques et méthodologiques s'accompagne d'un dilemme. Il s'agira de choisir entre expérimenter, adopter (ou non) en différenciant le mainstream de l'effet de mode.

L'une des valeurs fondatrices chez Xebia est « Sharing Knowledge ».

C'est donc avec grand plaisir que nous vous présentons la première version de notre TechTrends, synthèse du savoir-faire de nos 80 consultants français qui, en 2012, ont passé plus de 880 heures en R&D dans le cadre de nos XKE (Xebia Knowledge Exchange), organisé 50 TechEvents avec les éditeurs et acteurs de notre communauté et expérimenté les technologies dont nous vous parlons dans ce document.

Organisé en quatre thèmes (Architectures Orientées Web, Infrastructure d'entreprises, Langages de la JVM and Friends, Agilité and Craftsmanship), ce document vous propose également deux études plus poussées dans des domaines qui révolutionnent et continueront de révolutionner l'IT à l'avenir : Big Data et les applications mobiles d'entreprise.

Le TechTrends est un document à destination de toutes celles et ceux qui doivent opérer des choix structurants pour leur IT et souhaitent bénéficier du retour d'expérience terrain de ceux qui ont déjà expérimenté sur des projets les technologies d'aujourd'hui et de demain.

Nous espérons qu'il vous guidera dans les décisions importantes que vous aurez à prendre.

Nous vous en souhaitons bonne lecture.

Luc Legardeur

Président de Xebia



Big Data

par Pablo Lopez

Personne ne sera surpris, l'un des grands enjeux des années à venir est l'exploitation de la formidable quantité de données créées chaque jour par l'usage massif des nouvelles technologies et en particulier du Web.

La production de données a considérablement augmenté du fait de l'adoption de plus en plus large de technologies omniprésentes : terminaux mobiles, RFID, appareils media wireless, capteurs sans fils... Les perspectives d'exploitation de ces données (géolocalisation, comportements sociaux...) offrent de nouvelles opportunités aux entreprises. On constate d'ailleurs que les sociétés précurseures dans le domaine Big Data sont d'énormes centres de profits : Google, Facebook, LinkedIn, Amazon, pour ne citer que quelques succès. Ces entreprises se sont rapidement affranchies des contraintes des systèmes de traitement traditionnels de la donnée et proposent aujourd'hui, en open-source, des logiciels capables d'adresser efficacement et à moindre coûts ces nouvelles problématiques.

Vos données valent de l'or, il est temps de les exploiter.

Big Data, une définition communément admise

Pour qualifier ce que sont ces fameuses "Big Data", on peut utiliser la règle des trois V :

- **Volume.** C'est la dimension la plus évidente. Les volumes manipulés dépassent de très loin ce que l'informatique de gestion "classique" est en mesure de stocker et de gérer.
- **Vitesse.** La création de données est mondiale, continue et la capacité à analyser ce flux au plus près du temps réel est un différentiateur vital pour les entreprises. En effet, qui peut aujourd'hui se permettre d'attendre 48 heures avant la création de tableaux de bord métier quand la concurrence est capable d'analyser le marché et ses tendances en moins de 10 minutes ?

- **Variété.** Les sources multiples (réseaux sociaux, téléphones mobiles, API ouvertes, objets communicants ...) viennent s'ajouter à des données d'entreprise dont le format varie déjà au gré des différentes briques du SI. Certains ont tenté d'insérer ces données dans de classiques systèmes relationnels à grands coups de formats pivots, de plan de migration et d'ETL. Big Data prend le contrepied de ces projets (qui ont souvent abouti à de retentissants échecs) en faisant de la multiplicité des formats la "norme".

Ces trois V changent la donne sur toute la chaîne de traitement de la donnée : capture, stockage, analyse et visualisation ne peuvent plus être réalisés avec des outils traditionnels.

Les innovations technologiques

Les systèmes Big Data se démarquent des systèmes traditionnels de gestion de la donnée sur les trois axes suivants :

- Le traitement est amené au cœur de la donnée, pas l'inverse. En partant du principe que les données sont extrêmement volumineuses alors que les binaires de traitement sont bien plus légers, Big Data inverse le paradigme classique, réduisant de fait le volume des échanges réseau et diminuant considérablement les temps de traitement. De plus, un effort particulier est porté sur la lecture en parallèle des données.
- La parallélisation massive des processus est gérée simplement. C'est au système et non au programmeur, de gérer la distribution et la synchronisation de micro-tâches résultant du découpage d'un traitement complexe.
- La défaillance est la norme. Le système autorise et gère de manière transparente les échecs, quelle que soit leur nature.

Ce que nous voyons réellement derrière

Au-delà de la définition scolaire de Big Data, nous voyons derrière ce terme des enjeux bien plus pragmatiques.

Un système à coûts maîtrisés

Aujourd’hui, de nombreux éditeurs et constructeurs se sont engouffrés dans la brèche et proposent des solutions matérielles et logicielles clés en main. Ces solutions se révèlent souvent très chères et, exprimons le tout de suite, ne correspondent pas à la vision que nous avons du Big Data. Au regard des volumétries et de la nécessaire pérénité des systèmes dans le temps, devenir “otage” d’une solution matérielle est exclu : il n’est pas raisonnable d’envisager l’achat d’une machine de plusieurs centaines de milliers d’euros pour gérer de potentiels gros volumes de données à venir sur les cinq prochaines années ; et de devoir se reposer la question rapidement en cas de succès du projet et d’explosion des volumétries stockées et traitées. Pour les mêmes raisons, le paiement d’une licence, dont le coût est souvent indexé sur la quantité de données manipulées, peut rapidement faire s’envoler le budget d’un projet d’entreprise. C’est pourquoi, de notre point de vue, les systèmes Big Data doivent allier deux impératifs :

- Les composants du système doivent être libres d’utilisation, idéalement open-source.
- Le système doit être installé sur des serveurs “classiques”, d’entrée ou de milieu de gamme.

Ces deux avantages permettent d’obtenir une scalabilité à coûts maîtrisés, par ajout de noeuds peu onéreux.

Un système à coûts prédictibles

L’une des clés du succès d’un projet Big Data est la prédictibilité de ses coûts. Il est donc nécessaire que la scalabilité du système soit linéaire, aussi bien au niveau du stockage qu’au niveau de la puissance nécessaire pour traiter les données.

Un système adaptable

Comme on l’a vu, de nouvelles sources de données peuvent être amenées à intégrer le système à tout moment et avec elles peuvent arriver de nouveaux usages. Le système doit donc être en mesure de s’adapter facilement, d’intégrer rapidement de nouveaux outils répondant à de nouveaux besoins. Là encore, l’open-source semble être une réponse idéale : l’extensibilité du système, développée par les ressources de l’entreprise aussi bien que par la communauté open-source, est une clé de réussite.

Un système hautement disponible, qui tolère facilement la panne

Le système Big Data peut rapidement devenir la clé de voûte de l'activité, comme c'est le cas pour certains pionniers du Web. Il est donc vital que ce système soit hautement disponible. De par la multiplication des éléments hardware, la panne matérielle devient un non-événement. La tolérance à la panne doit faire partie intégrante du système.

Un système automatisé

Comme évoqué, le système est appelé à croître et ce, à un rythme peu prédictible. L'ajout de noeuds doit donc être une opération simple, voire triviale pour les exploitants. L'automatisation du provisionnement, de l'installation et du déploiement est un plus indéniable. De nombreuses solutions existent, quel que soit le niveau de maturité de l'automatisation dans l'organisation.

L'obsession de la mesure

Un système Big Data est un système hautement distribué qui tolère la panne de manière transparente. Il est donc nécessaire de s'outiller pour :

- **Déetecter** les dysfonctionnements et intervenir rapidement.
- **Mesurer** l'efficacité du système et se projeter dans un processus d'optimisation continue.
- **Surveiller** la croissance des données et des traitements, afin d'anticiper les investissements, ou de faire des arbitrages éclairés.

L'un des points essentiels est que la plateforme Big Data peut s'appuyer sur de multiples composants. Ils doivent tous être monitorés et idéalement doivent l'être grâce à un outil unifié, intégré au SI de l'entreprise.

Le champion : Hadoop

Bien qu'il existe une multitude d'outils "labelisés" Big Data, s'étendant des grilles de calcul distribuées aux bases de données NoSQL, un outil se dégage, répondant à l'ensemble des critères énoncés ci-dessus : Hadoop.

Le framework, open-source, développé en s'inspirant des publications de Google, est composé de deux principaux pans :

- Un stockage de données sous la forme d'un système de fichiers distribués.
- Un puissant framework de traitement des données, reposant sur l'algorithme Map / Reduce, permettant de traiter en parallèle de gros volumes.

L'écosystème qui gravite autour d'Hadoop offre de nombreuses fonctionnalités clés :

- La "tuyauterie" nécessaire entre les applications métier et le stockage distribué.
- La possibilité de coder les traitements métier dans la quasi totalité des langages de programmation du marché.
- Des langages de haut niveau, pour permettre la manipulation des données par des non-techniciens.
- Des connecteurs avec de nombreux outils commerciaux, aussi bien dans le monde du BI que dans celui de la restitution et de la manipulation des données.

De plus, Hadoop possède de nombreux avantages intrinsèques qui en font, selon nous, l'outil Big Data par excellence aujourd'hui :

- Le framework est 100% open-source, maintenu par une communauté dynamique.
- Il s'exécute sur des serveurs d'entrée de gamme, dont la seule caractéristique notable est de posséder de nombreux disques durs.
- Le stockage est réalisé sur un système de fichiers, certes distribué, mais dont l'administration se rapproche de celle d'un système de fichiers classique, ce qui facilite son adoption par les équipes d'exploitation.
- La scalabilité est totalement linéaire, aussi bien au niveau du stockage qu'au niveau du traitement de l'information.
- Le framework d'exécution Map / Reduce abstrait l'intégralité de la programmation distribuée sur un réseau de machines.

- Le système est hautement disponible sur tous les noeuds, l'information est redondée et la défaillance n'impacte pas le système.
- Le monitoring est intégré nativement, sous un format pouvant être manipulé par les outils standards de l'entreprise.

Toutes les grandes entreprises Web ont adopté cette technologie, parfois massivement. A titre d'exemple, Yahoo! possède plus de 20 clusters Hadoop en production, dont le plus grand comporte plus de 4000 serveurs. Il n'est pas rare d'entendre parler de volumétrie manipulée de plusieurs peta-octets.

"Hadoop possède de nombreux avantages intrinsèques qui en font, selon nous, l'outil Big Data par excellence aujourd'hui"

Néanmoins, Hadoop n'est pas une "silver bullet". Certains choix architecturaux, qui font sa force, l'obligent à faire une croix sur des fonctionnalités intéressantes. Il est principalement orienté batch et répond mal aux besoins de temps réel. C'est un axe de développement important et les nouveautés à ce sujet arrivent sur le marché. De plus, même si il est très puissant, le modèle de requête Map / Reduce est complexe pour des requêtes triviales. Ceci laisse la place à de solides challengers.

Les challengers

Pour des besoins plus légers, ou plus spécifiques, nous préférons nous orienter vers les bases NoSQL. Ces dernières sont décrites plus en détails dans le pilier Infrastructure d'entreprise de cet ouvrage :

- Une base de données orientée document, **MongoDb**.
- Une base de données orientée colonne, **Cassandra**.
- Une base de données orientée graphe, **Neo4j**.

En se basant sur la définition Big Data stricto sensus, ces bases ne possèdent pas les trois V caractéristiques, mais s'en approchent fortement. C'est pour cette raison qu'elles apparaissent ici même si les puristes peuvent tiquer.

Une piste intéressante à suivre, dans le cadre de grosses volumétries de données à analyser rapidement, suivant des axes simples, est l'utilisation d'un index mémoire massivement parallélisé. Dans cette catégorie, le produit open-source ElasticSearch a notre

préférence. Il comporte toutes les qualités nécessaires à un bon système Big Data : il se déploie sur des serveurs d'entrée de gamme, possède une scalabilité totalement linéaire et une interface de requête rapide, ouverte à tous les systèmes externes via une interface REST.

Dernier challenger important, à ne pas négliger, le Cloud. En effet, la très grande majorité des acteurs du Cloud proposent aujourd'hui du "*Big Data as a Service*". Il est possible de traiter de très grands volumes de données, en utilisant une puissance de calcul théoriquement illimitée, tarifée "au robinet". La plupart des opérateurs proposent nativement Hadoop et donc des jobs Map / Reduce développés en interne peuvent tirer parti d'une puissance difficilement disponible au sein de l'entreprise. Le principal problème qui se pose est que, dans le cadre d'un SI classique, il faudra exporter les données vers le Cloud et donc se poser les questions de la taille de la bande passante vers l'opérateur Cloud, de la confidentialité des données etc.

Les champs d'application

Quels leviers de croissance représente Big Data pour mon activité ? Sur quels champs d'application puis-je lancer mon projet Big Data ? La réponse la plus brutale à cette question est : partout où il y a de la donnée à analyser.

Plus précisément : partout où les systèmes informatiques dans leur forme actuelle échouent à transformer la donnée en information. En effet, il est inutile d'envisager positionner un système Big Data en remplacement d'un système classique, qu'il soit pur SGBD ou BI, si ce dernier répond au besoin : l'effort de mise en place et d'appropriation de ces nouvelles technologies sera un frein à l'adoption. Il existe néanmoins quelques grands cas d'utilisation, facilement identifiables.

Les grands classiques

Les *scénarii* d'utilisation les plus fréquents se retrouvent aujourd'hui dans les domaines suivants :

- **La détection de la fraude**

En analysant la totalité des actions d'un utilisateur sur un système informatique, il est possible de dégager des motifs, des comportements extraordinaires qui signalent une fraude. Il est très facile, par exemple, de détecter une transaction anormale (pays de provenance, enchainement dans le temps) et de rapidement vérifier sa véracité (par téléphone par exemple) et donc de bloquer la fraude. Ce type de service représente un gain pour le consommateur final, mais aussi pour l'opérateur bancaire.

- **Les moteurs de recommandations**

Là encore, en analysant le comportement de la totalité de l'audience (produits consultés, produits achetés, commentaires), il est possible d'améliorer la pertinence d'un moteur de recommandation, voire même de créer un logiciel intelligent, qui, au fil des comportements, saura perfectionner ses recommandations.

- **Le reporting légal**

Les autorités financières de tous les pays renforcent de manière drastique leurs règles de contrôles, suite aux différents scandales financiers récents. Il est donc nécessaire de s'équiper pour répondre rapidement et quotidiennement, à la génération de rapports légaux demandant d'analyser et de manipuler, suivant de nombreux axes, des milliers de lignes de positions financières.

- **La sécurité**

D'une manière ou d'une autre, toutes les actions d'un utilisateur sur un SI sont tracées, tous les flux internes de l'entreprise peuvent être capturés. Analyser et combiner ces traces permet de rapidement mettre en oeuvre une politique de sécurité avancée au sein même de l'entreprise.

Les outsiders

D'autres cas d'utilisation, moins classiques, peuvent être envisagés, mais demandent plus d'investissement métier lors de leur mise en place :

- **Le calcul de risque**

Toutes les entreprises exposées au risque possèdent déjà des routines d'analyse. Néanmoins, il n'est pas rare d'entendre des analystes pester contre les limitations physiques des systèmes actuels, qui ne permettent de manipuler que trop peu d'axes ou trop lentement.

- **L'analyse BI temps réel**

Il s'agit dans ce cas de réduire les délais de traitement de systèmes déjà en place. Par exemple, de passer de traitements à J+2 à des résultats à H+2, ce qui permet, dans le cadre d'une campagne marketing de vite intervenir pour corriger le tir.

- **Le renforcement de la connaissance client**

Analyser toutes les traces qu'un utilisateur laisse sur votre système et les corrélérer avec celles qu'il laisse sur les réseaux sociaux ouvre la porte à un marketing personnalisé plus réactif et plus pertinent et permet de conquérir de nouveaux marchés.

Conclusion

Au delà du buzz, Big Data est aujourd’hui incontournable. Présents à un stade avancé dans de nombreuses entreprises florissantes du Web, les systèmes Big Data sont déjà à l’étude, voire en production, dans un nombre croissant d’entreprises françaises, de la PME à la multinationale et ce dans tous les domaines : banque, assurance, tourisme, e-commerce, industrie... Il est, de notre point de vue, crucial de réfléchir dès aujourd’hui au trésor de guerre que représente la multitude de données inexploitées au sein votre entreprise et urgent de transformer ces données dormantes en information, vecteur de croissance pour votre business.

“Au delà du buzz, Big Data est aujourd’hui incontournable”

Comment se lancer, dès aujourd’hui, dans un projet Big Data ? L’appropriation des technologies est un processus souvent réalisé par des informaticiens purs. Il est donc plus facile de se lancer sur un use case très orienté technique. Selon nous, un bon point d’entrée est l’analyse, via des jobs Map / Reduce Hadoop, de logs d’accès sur des frontaux Web (apache ou IIS). Ce use case, a priori simple, vous permettra de prendre en main tous les aspects clés du projet :

- Installation, paramétrage et monitoring d’Hadoop.
- Intégration entre les frontaux Web et le cluster.
- Appropriation de l’API de jobs Map/Reduce via des développements simples (top des pages les plus vues, répartition physique des utilisateurs via leur IP...).
- Possibilité d’ouvrir la plateforme par la suite à des non techniciens, via du requêtage haut niveau, comme Pig.
- Généralisation progressive à l’ensemble du SI et enrichissement des analyses, après ajout d’informations métier dans les logs.



REST

MVC

JAVASCRIPT

CACHING

Architecture Orientée Web

La nécessaire ouverture vers le Web des SI des entreprises, qui cherchent à être toujours plus compétitives sur leur marché (au moyen d'une augmentation de leur popularité ou de la vente de services en ligne), a amené à une remise en question des styles d'architecture existants.

Ce sont les pionniers du Web, tels Amazon, Apple, Google ou Facebook, qui ont ouvert la voie en abandonnant le modèle traditionnel dit « Entreprise » au profit d'une construction basée entièrement sur le Web. En bouleversant l'ordre établi et en remettant en cause notre héritage informatique, ces entreprises ont bâti leur SI avec des technologies innovantes et leur succès n'a cessé de croître. Grâce à des architectures plus légères et de qualité, leurs coûts ont été réduits et leur performance s'est accrue car elles ont pu se concentrer sur leur cœur de métier et améliorer leur time to market.

Les architectures orientées Web (plus communément appelées WOA pour Web Oriented Architecture) choisissent de réutiliser tous les standards du Web ayant fait leurs preuves (HTTP, URL, etc.) plutôt que de réinventer des protocoles qui risquent de coûter cher en cas de défaillance. La gestion de la défaillance même est chamboulée. La contractualisation systématique, véritable bouclier illusoire des entreprises, laisse place à une souplesse et une tolérance qui font de la panne un non-événement. Les principes des WOA permettent de créer des architectures puissantes répondant aux exigences de l'IT pour les sites à fort trafic mais aussi de bâtir des services fonctionnels qui vont répondre aux problématiques business et aux stratégies multi-canal des entreprises. L'objectif de ce pilier est de mettre en lumière les bonnes pratiques à prendre au Web pour l'entreprise (applications, frameworks, caching ou encore services Web).

Cette nouvelle vision tend à tirer le meilleur parti du Web. Le monde informatique assiste à une véritable mutation dans la construction des SI des entreprises, avec des techniques plus simples, plus adaptables, plus pérennes, plus sécurisées et plus économiques, en adéquation avec les contraintes des entreprises d'aujourd'hui.

Les Applications Web d'entreprise en 2013

par Philippe Antoine

Les Applications Web de 2013 n'ont plus grand chose à voir avec celles qui ont pu être développées ces dernières années.

L'arrivée des navigateurs récents sur smartphones et tablettes a poussé les technologies Web un cran plus loin :

- Grâce aux nouvelles librairies et aux nouveaux frameworks, il est possible de développer des applications Web qui rivalisent avec les applications natives.
- L'innovation est de retour et, cette fois-ci, les applications d'entreprise peuvent en profiter.
- L'angle d'attaque des applications d'entreprise bascule : au lieu de suivre les technologies propriétaires, il est aujourd'hui possible de développer des projets ambitieux.

Nous vous proposons d'illustrer ces changements du point de vue d'un porteur de projet, puis d'un point de vue technique.

Web ou natif ?

Historiquement, les applications Web ont toujours été développées pour les ordinateurs de bureau. Il convient dorénavant de développer des applications compatibles pour les téléphones et autres tablettes. Faut-il développer en utilisant des technologies propriétaires, compléter par des applications mobiles natives et/ou se lancer dans le développement d'une application Web ?

Les applications lourdes, basées sur des technologies propriétaires, ont ouvert la voie à des innovations, en termes d'ergonomie et d'expérience utilisateur. La comparaison des applications utilisées tous les jours sur nos smartphones à celles qui ont pu être développées, il n'y a que quelques années dans le cadre de projets d'entreprise, ne fait que mettre en avant un changement profond.

Pour proposer ce nouveau type d'applications aux utilisateurs, développer une application mobile native peut être une solution. Il est néanmoins possible de regarder du côté de ces nouveaux frameworks Web basés sur JavaScript. L'approche est un peu différente. Une application Web compatible mobile pourra être consultée à la fois sur un ordinateur de bureau et sur un smartphone ou une tablette avec un navigateur récent. L'application native, quant à elle, ne pourra cibler qu'un seul type de système d'exploitation. L'idéal étant, bien sûr, de développer à la fois une version de son application Web HTML5 "tout terrain" et de la compléter par des applications natives en fonction du type d'utilisateur ciblé.

“En 2013, les applications Web d’entreprises peuvent proposer à l’utilisateur une expérience aboutie allant du desktop au mobile en passant par la tablette”

Plusieurs possibilités peuvent être envisagées pour une application Web. Un de nos clients a récemment été confronté à ce choix technologique pour le lancement d'un nouveau produit. Une des premières préoccupations a été de choisir un langage déjà maîtrisé par leurs développeurs internes. Des premiers "Proof Of Concepts" avaient été réalisés en Wicket ou Flex. Voyant que ces technologies n'étaient plus forcément d'actualité, il a été décidé de mettre en concurrence deux autres technologies: GWT et HTML5. Les développements GWT réalisés en Java offrent un énorme avantage : les bibliothèques de composants disponibles permettent rapidement aux développeurs de créer les premières fonctionnalités. Malgré ce gain de temps initial, deux points noirs viennent obscurcir le tableau :

- Google a déjà complètement abandonné des technologies par le passé. Il faut donc rester méfiant face à leur sponsoring actuel.
- De plus, pour des développements avancés en GWT, il est nécessaire de contourner le cadre proposé par leur système de composants et finalement retomber dans un développement spécifique.

Le choix s'est finalement porté sur le développement d'une application Web HTML5. Outre les points mentionnés ci-dessus, HTML5 présente les avantages suivants :

- Les développements reposent sur des standards : une application développée aujourd'hui ne risque pas de devenir obsolète suite à la décision d'un grand nom.
- Les développeurs portent un vif intérêt pour ces technologies : le passage à ces dernières est facilité.
- L'application développée est compatible avec les tablettes du marché utilisant un navigateur récent.

Après ce tour d'horizon du point de vue du porteur de projet, intéressons nous maintenant aux impacts techniques.

Ce qui change pour les développeurs

Le développeur d'application Web en 2013 se retrouve dans un écosystème qui a considérablement évolué.

Il peut tout d'abord s'appuyer sur un arsenal de nouveaux frameworks (cf. le chapitre sur ces frameworks) et de librairies qu'il ira piocher sur [github.com](#). Ces outils apportent leur lot de nouveautés :

- **Les applications Web monopage**
- **Le “Responsive Design”**

Avec l'arrivée du mobile et la multiplication des plateformes, le concept de “Responsive Design” s'est imposé. Il s'agit de proposer une interface utilisateur adaptée au terminal, tablette ou téléphone, en largeur (moins souvent en hauteur). Ce principe s'appuie sur une nouvelle API HTML5 : les media queries. Il s'agit de développer des feuilles de styles différentes en fonction de résolutions cibles, par exemple, une résolution mobile, tablette ou écran 16/9. Il est possible d'utiliser le même système dans IE8, mais cette fois en utilisant une librairie JavaScript adaptée.

- **Compatibilité multi-navigateurs**

La compatibilité de ces nouvelles applications avec Internet Explorer 8 peut cependant être un sujet d'inquiétude. Après avoir réalisé plusieurs projets qui ciblent ce navigateur, les fameux “bugs” d'Internet Explorer ne sont pas ce qu'il y a de plus handicapant : la plupart sont documentés sur Internet et il est facile de les contourner. C'est surtout concernant les performances qu'il faut fournir un effort dans les développements. Il existe des techniques pour charger en asynchrone des parties de l'application ou encore maîtriser la consommation de mémoire. En améliorant les performances pour l'utilisateur, par ricochet, la fluidité sur les appareils mobiles (limités en mémoire et en capacité de calcul) est elle aussi augmentée.

En bref

Grâce à toutes ces techniques et nouveautés, des applications ambitieuses, pouvant rivaliser avec les applications natives et toucher un plus grand nombre d'utilisateurs, quel que soit le terminal, sont aujourd'hui à portée de main.

Développer des Applications Web n'est plus réservé à un marché de niche : Xebia Studio en a réalisées plusieurs ces derniers mois. La tendance pour 2013 est à l'explosion de ce type de projets.

Les points clés :

- Des projets Web ambitieux, même en entreprise.

- Des applications “tout terrain” avec le responsive design et la compatibilité multi-navigateurs.
- Des interfaces utilisateur nouvelle génération et réactives.

Playframework 2.0

par Jean Helou

Bien plus qu'un simple framework, Playframework offre toutes les possibilités d'une plateforme logicielle. Sa conception est fortement inspirée par les plateformes RAD (Rapid Application Development) telles que Ruby on Rails ou Grails. Ses concepteurs ont repris à leur compte les principes qui ont fait le succès de ces dernières : productivité, cohérence, scalabilité.

En termes de productivité, la plateforme permet un démarrage très rapide au travers d'un kit de démarrage téléchargeable incluant les principaux frameworks utilisés dans une application Web. Ce kit de base peut facilement être étendu avec des dépendances supplémentaires. Il inclut également un serveur de développement configuré pour prendre en compte toutes les modifications “à chaud”. Les éventuelles erreurs, ainsi que les résultats d'exécution des tests, sont remontés directement dans le navigateur. De cette façon, les développeurs n'ont pas besoin de sortir de la boucle de feedback Editeur - Navigateur. Sans redémarrage du serveur, ni redéploiement des ressources, tout est fait pour que le développeur attende aussi peu que possible. Enfin, les applications peuvent être développées dans deux langages interopérables : Java et Scala. Scala, en particulier, apporte des bénéfices de productivité grâce à une syntaxe très concise et un modèle de programmation hybride entre programmation orientée objet et programmation fonctionnelle. Comme Java, il est fortement typé et effectue encore plus de vérifications à la compilation que ce dernier, permettant de détecter certaines catégories de bugs encore plus tôt.

“Productivité, cohérence et scalabilité”

Les applications sont toutes produites à partir d'un même “moule”, un ensemble de conventions suivies par la plupart des applications. Le bénéfice de ces conventions est double. D'une part, les applications sont très cohérentes, ainsi les développeurs peuvent passer de l'une à l'autre sans être bloqués par des différences techniques. Les différences fonctionnelles persistent mais la barrière est amoindrie. D'autre part, les conventions permettent de limiter au maximum la configuration initiale de l'application.

Avec la plateforme, il est possible de créer en quelques secondes une application et lancer un serveur capable de répondre aux requêtes extérieures.

Les applications développées sur Playframework sont nécessairement influencées par le modèle de la plateforme. Les choix de conception de ce dernier ont été mûrement réfléchis pour pousser au développement d'applications sans état (stateless) côté serveur, respectant parfaitement le protocole HTTP. Ceci permet de tirer pleinement partie des couches intermédiaires entre le navigateur et le serveur : les caches, les load balancers, etc. Le modèle d'application stateless permet également de dupliquer très facilement les instances de l'application. Couplé avec le Cloud, la capacité de réponse de l'application peut être optimisée. Afin d'aller encore plus loin, la plateforme offre également un modèle asynchrone, dont le développement est rendu plus facile par l'absence d'état à partager. Ainsi une application peut utiliser au maximum les capacités de chaque serveur.

Les frameworks JavaScript

par Benoît Lemoine

Grâce à l'amélioration des possibilités techniques des navigateurs, les applications Web ont, ces dernières années, connu une évolution rapide vers toujours plus de fluidité de l'expérience utilisateur. À tel point que beaucoup d'applications récentes sont aujourd'hui bâties autour du concept d'application monopage (Single Page Interface), se présentant comme une page HTML unique, composée d'éléments dynamiques. La navigation et la récupération des données sont alors entièrement gérées côté client.

“Les favoris du moment, Backbone, Ember et Angular, répondent aux enjeux du Web par des approches très différentes”

Ce type d'architecture applicative comporte certains obstacles techniques, tels que le routage, la gestion des actions utilisateurs ou encore le templating. Pour répondre à ces enjeux relativement standards, de nombreux frameworks ont émergé ; les favoris du moment étant Backbone, Ember et Angular.

Les trois tentent de répondre aux mêmes problématiques, mais le font avec une approche très différente :

- **Backbone**¹ choisit une approche MVC volontairement minimaliste, laissant ainsi au développeur un maximum de liberté dans ses choix techniques et architecturaux. Par défaut, Backbone ne fournit ainsi pas de moteur de template ou de data-binding.
- **Ember**² suit une approche MVC beaucoup plus dirigiste, en imposant de développer en suivant ses conventions. S'appuyant sur Handlebars pour le templating et fournissant du data-binding, Ember présente malheureusement une courbe d'apprentissage assez raide.
- **Angular**³, framework soutenu par Google, choisit une approche MV-VM et propose lui aussi du data-binding ainsi qu'un système de template basé sur le langage HTML. Mais tout comme Ember, il est nécessaire de se plier aux conventions du framework pour en tirer parti.

Si ces derniers mois Angular semble tenir le devant de la scène (le sponsoring de Google n'y étant pas pour rien), le marché du framework JavaScript est encore jeune, de nombreux concurrents moins connus (Knockout ou Batman, par exemple) existent et il est encore difficile de dire aujourd'hui quels seront les challengers incontournables, et lesquels tomberont dans l'oubli.

Caching

par Séven Le Mesle

Avec l'augmentation exponentielle du nombre d'internautes, liée à l'adoption en masse des terminaux connectés de type tablettes et smartphones, les sites enregistrent un nombre de visiteurs record. La tendance n'est donc plus à la réservation des ressources pour un utilisateur. Pour économiser de la bande passante et des serveurs, l'utilisation de cache est devenue incontournable.

Cache applicatif

L'augmentation du trafic met sous pression les bases de données et force à mettre en place des caches applicatifs. Le désormais célèbre Memcache offre une solution très

¹ <http://backbonejs.org/>

² <http://emberjs.com/>

³ <http://angularjs.org/>

rapide et fiable. Mais de nouveaux acteurs arrivent sur le marché, tels Redis et MongoDB. Ces bases de données NoSQL (de type clé-valeur pour Redis et de type document pour MongoDB), ont été propulsées comme caches applicatifs notamment grâce à leurs performances. De plus en plus d'applications s'appuient sur ces solutions et la méthode se rôde. Ces dernières tendent à supplanter l'utilisation des DataGrid et autres caches distribués très à la mode il y a quelques années.

Proxy-cache

En amont des serveurs d'applications classiques, Varnish fournit une solution stable et éprouvée de proxy-cache. Varnish s'occupera de cacher toutes les ressources statiques ou dynamiques servies par l'application backend, de façon à ce qu'elles ne soient servies qu'une seule fois par le backend. L'utilisation de Varnish permet en outre d'assurer un fonctionnement en failover. Il continuera, en effet, à servir des ressources expirées si les serveurs applicatifs ne sont plus disponibles. En pratique, un site à fort trafic peut utiliser un délai d'expiration court d'une minute garantissant ainsi que la page mise en cache sera rafraîchie à intervalle régulier tout en ne laissant passer qu'une requête par minute vers le serveur applicatif.

"Les bases de données NoSQL type Document (MongoDB) ou Clé-Valeur (Redis) sont propulsées comme caches applicatifs"

Sur le marché des proxy-cache, un nouvel acteur est toutefois en train de s'imposer : Nginx. Nginx est le serveur Web déjà utilisé par certains grands noms du Web comme Facebook, GitHub et Dropbox, pour ne citer qu'eux. Bien que Nginx soit un serveur Web prévu pour résoudre le problème des 10 000 connexions concurrentes, il devient également incontournable en matière de gestion de cache, notamment grâce à sa capacité à soutenir de fortes charges en consommant peu de processeur et de mémoire.

Content Delivery Network

Au bout de la chaîne, en amont des serveurs Web frontaux, la bonne pratique consiste à utiliser un CDN pour accélérer le temps de chargement ressenti par l'internaute et ainsi économiser les ressources de production. Les CDN sont des réseaux de proxy-cache positionnés au plus près de l'internaute (sur le réseau du FAI par exemple).

Dans ce domaine, Akamaï est le leader historique et incontesté. Néanmoins, avec la démocratisation d'internet et l'augmentation exponentielle du nombre de connexions, de nouveaux acteurs et de nouvelles tendances émergent. Les CDN hébergeant des frameworks JavaScript et CSS OpenSource comme Googleapis.com ou cdnjs.com sont

désormais couramment utilisés et recommandés. Ils permettent d'économiser la bande passante et tirent avantage du cache des navigateurs. Parmi les concurrents d'Akamaï, nous retiendrons donc Amazon CloudFront qui facture à la carte tout comme CloudFare. Finissons ce tour d'horizon des CDN par un petit cocorico en citant la solution OVH CDN, actuellement en bêta test gratuite, qui satisfait déjà de nombreuses entreprises.

En bref

- Généraliser l'utilisation de CDN en veillant à ne pas utiliser d'extension spécifique à votre fournisseur.
- Mettre en place un proxy-cache comme Varnish, intégré de façon transparente au serveur applicatif, à défaut Nginx est un bon candidat.
- Utiliser Memcached comme middleware de cache applicatif, dans le cas du besoin d'un cache persistant, se tourner vers MongoDB.

Les services Web, langage universel

par Yves Amsellem

Nombreux sont les moyens que trouvent les applications pour s'entendre. Les services Web se sont distingués dans ce domaine, notamment depuis l'essor d'internet et de la mobilité. Deux technologies aux approches assez différentes s'y distinguent — toutes deux basées sur HTTP, le socle des communications du Web — SOAP, Simple Object Access Protocol et REST, REpresentational State Transfer.

Afin d'exposer des informations, un système indique les données échangées, les opérations proposées et les erreurs potentielles. Pour cela, SOAP propose une sur-couche à HTTP. Pour accéder à un service SOAP, un client doit connaître son contrat, un fichier WSDL, à partir duquel il génère des objets dotés d'opérations. Chacune de ces opérations représente un appel distant que SOAP effectue pour le client. REST essaye à l'opposé d'être le plus fidèle possible à HTTP. Pour accéder à un service REST, un client doit connaître son URL et peut le manipuler avec les opérations HTTP standards, GET — accéder, POST — créer, PUT — modifier et DELETE — supprimer.

SOAP encode les échanges en XML là où REST est ouvert, utilisé avec JSON, il est l'allié idéal des applications JavaScript dont c'est le format de représentation natif. Les erreurs SOAP sont personnalisées là où REST retient une fois encore le standard HTTP, 200 — ok, 301 — redirection, 401 — non autorisé, 404 — inexistant.

"Son respect du standard HTTP permet à REST de profiter de nombreux avantages des infrastructures réseau"

Côté sécurité, SOAP propose avec son protocole WS-Security des facilités de sécurisation des données confidentielles. La couche transport de HTTP — une couche plus basse que la couche applicative utilisée par SOAP — offre également ces possibilités avec SSL. Sécuriser le canal SOAP ou REST avec SSL est préférable.

SOAP est connu pour sa gestion des transactions : effectuer un virement bancaire avec la garantie que le crédit d'un compte ne soit effectué que si le débit associé l'est également. En d'autres termes, appeler plusieurs opérations, créditer et débiter, et annuler l'ensemble si une échoue. REST propose un angle d'attaque différent : regrouper cette suite d'opérations dans un seul service, par exemple transfert. Un protocole stateless, sans état, où chaque appel est indépendant, a le net avantage de ne pas nécessiter de session côté serveur offrant ainsi une meilleure scalabilité (à noter, SOAP peut être utilisé sans transaction).

L'évolution des services Web implique souvent le recours au versionning : la cohabitation de plusieurs versions des mêmes services. SOAP et REST offrent des solutions à cette problématique, avec un léger avantage pour REST, si les services existants ne sont pas impactés, aucune opération n'est nécessaire.

Il n'est plus de pionnier du Web — Google, Amazon, Twitter — sans service REST. Effectuer un appel HTTP est significativement plus simple qu'effectuer un appel SOAP, et peut même être fait directement dans un navigateur Web. Sa légèreté ouvre REST à davantage de clients, et son respect du standard HTTP lui permet de profiter de nombreux avantages des infrastructures réseau (cache, sécurité, concurrence), ce qui explique son succès sur les applications Web et mobiles.



**NOS
RECOMMANDATIONS**

Les architectures orientées Web

Inspirez-vous des pionniers du Web ! Les techniques utilisées pour gérer les sites Web à fort trafic dirigent une innovation importante.

Les systèmes de cache, les architectures faiblement couplées, tolérantes aux pannes sont des applications directes de l'évolution des site Web d'aujourd'hui. Pensez-y !

JavaScript est incontournable ainsi que les frameworks clients pour le navigateur. L'ordre est à l'allégement du backend pour une gestion des applications côté client. Il serait intéressant d'expérimenter une application Web monopage pour un nouveau projet.

CLOUD

MONGODB

CASSANDRA

HADOOP

CLOUDBEES

HEROKU

Infrastructure d'entreprise

L'infrastructure de l'entreprise est en profonde mutation, amplifiée par les phénomènes Cloud et NoSQL.

Le Cloud est aujourd'hui un enjeu majeur. Un service hébergé sur le Cloud permettra à une entreprise de diminuer ses coûts, tout en profitant d'une plus grande fiabilité et d'une meilleure disponibilité de son service. Par exemple, il suffit de quelques secondes pour disposer d'une dizaine de serveurs avec l'offre d'Amazon Web Services. Le revers de la médaille peut néanmoins se révéler bloquant dans certains contextes : les serveurs Cloud sont aujourd'hui pour la plupart localisés aux USA, donc soumis au PatriotAct. Si les contraintes de confidentialité d'une entreprise sont fortes, comment outsourcer dans le Cloud ? Cloudwatt, localisée en France, va proposer d'ici octobre une solution pouvant répondre à ces problématiques.

Hier encore, un projet informatique se devait d'entreposer tout ou partie de ses données dans une base de données relationnelle, assurant les beaux jours des fournisseurs de ces dernières. Aujourd'hui, de nombreuses alternatives existent, offrant un stockage personnalisé en termes de rapidité ou de consistance. Le thème NoSQL abordé dans ce pilier a pour ambition de vous aider à prendre des décisions concernant le stockage de vos données pour votre ou vos projet(s). MongoDB, Neo4j mais aussi Cassandra passeront ainsi sous nos radars. Nous vous recommandons de croiser ce thème avec le chapitre sur la programmation fonctionnelle abordé dans le pilier JVM et amis.

Cloud Computing

par Cyrille Le Clerc

Le Cloud Computing transforme l'infrastructure et les middlewares en commodités utilisables sur un mode self-service par les équipes projet ; plus révolutionnaire encore, le Cloud suggère de louer ces plateformes applicatives chez des fournisseurs comme Amazon plutôt que de les gérer avec des équipes et des data-centers traditionnels. La compétitivité économique du Cloud s'accompagne d'importants gains en réactivité : l'infrastructure devient "agile". Ces bouleversements ne vont pas sans impact sur les organisations : les métiers des "OPS" - les équipes infrastructures et exploitations - vont être réinventés.

Voici un éclairage sur deux grands débats actuels du Cloud Computing :

- Infrastructure as a Service (IaaS) vs. Platform as a Service (PaaS).
- Public Cloud vs. Private Cloud.

En préambule, nous vous proposons de définir le IaaS comme un Cloud qui fournit "réseau, linux/windows et stockage" et le PaaS comme un Cloud qui fournit, en plus du IaaS, les middlewares et les services techniques utilisés par les applications.

Le débat "IaaS vs. PaaS" est clos sur les Public Clouds ; tous les acteurs ont franchi le stade du IaaS depuis des années et rivalisent sur le catalogue des middlewares et des services techniques. Il n'y a pas de sens sur un Public Cloud à se limiter à un usage IaaS et à installer soi-même les load-balancers ou les bases de données alors que ces services sont proposés sur étagère par la couche PaaS. Pire, installer soi-même sur un Cloud IaaS un service "prêt pour la production" comme une base de données est réservé à des équipes expérimentées : les bonnes pratiques sur le Cloud sont différentes de celles des infrastructures traditionnelles et nécessitent un apprentissage.

Parmi les Public Clouds, Amazon bénéficie d'un écosystème incomparable de fournisseurs de services disponibles sur leur infrastructure : Amazon est le point de passage obligé pour tous les services Cloud. Il se crée un effet de club et la supériorité d'Amazon sur ses concurrents n'est plus seulement liée à son catalogue de services déjà unique, mais renforcée par la profusion de fournisseurs tiers qui gravitent sur la plateforme.

Public Cloud vs. Private Cloud

Le Public Cloud, et particulièrement le Cloud Amazon AWS, est l'endroit où le Cloud Computing prend forme.

"Le Public Cloud est l'endroit où le Cloud Computing prend forme"

Le Private Cloud fait rêver : obtenir chez soi le même bouleversement qu'Amazon AWS sans vraiment changer, sans remettre en cause des investissements aussi coûteux que les data-centers et surtout sans remettre en cause l'organisation des équipes et le rôle de chacun.

Hélas, il n'y a pas de solution miracle et le rêve du Private Cloud à la Amazon AWS n'existe pas encore.

Tout d'abord, les offres de Private Cloud actuelles sont très en retard sur le Public Cloud. Elles se cantonnent à des simples IaaS quand Amazon AWS offre des dizaines de services au-dessus de sa couche IaaS sans compter la richesse de l'écosystème des fournisseurs de services sur le Cloud Amazon. Ensuite, les solutions de Private Cloud sont très coûteuses avec souvent, le renouvellement complet de l'infrastructure par des "armoires appliances" qui embarquent stockage, réseau et serveurs (VBlock pour EMC-Cisco-VMWare, FlexPod pour NetApp-Cisco-VMWare, etc). Enfin, ces solutions sont compliquées et fragiles, le Cloud computing manque de maturité pour que les éditeurs parviennent à figer des versions stables qui ne nécessitent pas d'être administrées au quotidien par leurs équipes R&D. À ces difficultés s'ajoute un défi culturel : il est beaucoup plus facile d'embrasser une rupture dans son mode de fonctionnement quand on change de cadre.

"Le rêve du Private Cloud à la Amazon AWS n'existe pas encore"

Les risques juridiques et de sécurité des données sont parfois invoqués pour bannir les Public Clouds au profit de Private Clouds. S'il ne faut pas négliger ces problématiques, il serait dommage de dramatiser. Il existe des solutions pour les data-centers hors d'Europe des multinationales ou encore pour le transit par le Canada des emails des cadres dirigeants friands de BlackBerry ; il existe des solutions pour utiliser des Public Clouds sur le sol américain voire directement sous juridiction européenne en Irlande.

En bref

Nous vous recommandons d'engager les équipes projets et les équipes infrastructure/exploitation en empruntant deux chemins simultanément.

Côté projet, nous vous recommandons d'engager une équipe agile, si possible sensible à DevOps, sur un PaaS "bout-en-bout" comme [CloudBees](#)⁴ qui gère tout le cycle de vie d'une application depuis le développement jusqu'à la production.

Côté infrastructure/exploitation, le choix de l'équipe est primordial car les risques et réticences sont plus élevés. Nous vous recommandons d'engager une équipe adepte de gestion de configuration de serveurs (e.g. Chef, Puppet) et d'infrastructure-as-code sur le Cloud Amazon AWS.

⁴ <http://cloudbees.com>

Conteneurs légers

par Aurélien Maury

La mort des éléphants

Historiquement, parmi les grandes sociétés produisant et hébergeant des applications Web Java, les serveurs d'applications "sérieux" étaient plutôt du côté de IBM avec Web-sphere Application Server ou Oracle avec Weblogic. Mais ces dernières années, les architectes consommant du IaaS ont dû faire face à un problème : les solutions à base de serveurs d'applications aussi complètes sont très coûteuses et rigides. Elles sont coûteuses par leur mode de facturation au processeur et rigides car le démarrage et le paramétrage sont souvent longs et l'automatisation complexe.

La relève

C'est grâce à ce constat que des solutions plus simples, comme les conteneurs légers ont été privilégiées. Tomcat et Jetty, les deux serveurs d'application légers phares, se sont donc diffusés dans le Cloud. Leur simplicité de mise en oeuvre offre des possibilités d'automatisation forte pour un investissement de développement relativement faible. Et c'est ce qui les a rendus incontournables dans le Cloud. Ils ont acquis leurs lettres de noblesse et sont maintenant une pierre angulaire des PaaS Java tels que [Amazon Beanstalk⁵](#), [Run@CloudBees⁶](#), [Heroku⁷](#) et [JElastic⁸](#).

Plus rapide, plus facile, plus attrayant

Parallèlement à cette implantation des légers, voilà que des ultra légers arrivent dans le paysage pour repousser un peu plus loin les limites de Tomcat et Jetty : le modèle Servlet. Des serveurs comme NodeJS, ou Vert.x et Play Framework, basés sur Netty, proposent l'outillage pour réaliser des applications Web avec entrées-sorties non-bloquantes ([NIO⁹](#)), permettant de traiter un plus grand nombre de requêtes par serveur. L'installation et la configuration des ces serveurs ultra légers sont très simples, par conséquent, ils ont été intégrés rapidement aux catalogues des PaaS. Jetty a aussi

⁵ <http://aws.amazon.com/fr/elasticbeanstalk/>

⁶ <http://cloudbees.com>

⁷ <http://heroku.com>

⁸ <http://jelastic.com/>

⁹ http://en.wikipedia.org/wiki/Asynchronous_I/O

pris le train du NIO en marche, tandis que Tomcat rattrape son retard en la matière avec sa version 7.

Pour suivre cette tendance, il faut supprimer les gros serveurs monolithiques au profit de conteneurs légers. Cela peut entraîner des modifications d'architecture logicielle et du temps, mais, à terme, les gains en terme de coût et sur la maîtrise de votre infrastructure sont conséquents et annulent la dépendance à un éditeur.

Messaging

par Guillaume Arnaud

Le marché du MOM (Message-Oriented Middleware) a longtemps été dominé par les éditeurs établis comme Websphere MQ ou Tibco et de fait, l'écosystème Java utilisait exclusivement l'API JMS. Récemment, ce standard a été bousculé avec l'arrivée de nouveaux protocoles (AMQP, STOMP, MQTT...), l'amélioration des éditeurs ouverts (RabbitMQ, ActiveMQ, HornetQ, Qpid...) et l'évolution des applications notamment vers le web et la mobilité.

Après une longue maturation, le protocole AMQP est enfin sorti en version 1.0. Mais sa forte complexité semble être un frein pour l'adoption par de nouveaux éditeurs. Si RabbitMQ et Qpid ont réussi à s'imposer, c'est sans doute plus lié à leur stabilité et leurs fonctionnalités qu'à l'utilisation du protocole (remarquons d'ailleurs qu'ils n'implémentent pas encore la version 1.0). De plus, JMS 2.0 est en cours d'élaboration, plus de dix ans après la dernière version 1.1, et déposera sérieusement l'API.

Des protocoles plus simples s'imposent afin de répondre à des besoins précis. En effet, avec les applications Web et mobiles, les contraintes pour le messaging sont nouvelles :

- Légèreté des messages.
- Tolérance aux nombreuses ruptures de connexion.
- Interopérabilité.
- Etc.

Pour autant, les nouveaux protocoles ne cherchent plus nécessairement à être complets en termes de richesse fonctionnelle (routage, acquittement ou transactions par exemple). STOMP (Simple Text Orientated Messaging Protocol) est un bon exemple d'un protocole simple et qui devient de plus en plus populaire. La spécification orientée texte en fait un bon candidat pour le messaging sur le web. Avec la montée en force de HTML5,

STOMP est venu assez naturellement ajouter un protocole de messaging au-dessus des WebSockets. De même MQTT, un protocole déjà ancien, connaît un regain d'intérêt avec la multiplication des solutions mobiles et des domaines comme *internet of things*. Ce protocole a la particularité de tenir compte des modes dégradés et des déconnexions fréquentes, et propose différents niveaux de qualité de service. Grâce à leur relative simplicité, ces deux protocoles sont maintenant disponibles dans la plupart des solutions éditeurs.

Nous voyons donc une fragmentation des solutions à notre disposition qui bouscule nos habitudes. Par exemple, des solutions de cache comme Redis ou Hazelcast offrent des fonctionnalités de messaging. À un plus bas niveau, ZeroMQ propose leur implémentation de sockets pour faire du messaging sans serveur et connaît un certain succès dans des applications massivement distribuées. Les grands du Web ouvrent également leurs applications internes, comme Kafka développé par LinkedIn qui prend le contrepied les MOMs classiques en stockant les messages sur une longue durée et en étant massivement distribué.

Le choix pour aller vers telle ou telle solution dépendra alors en grande partie des contraintes du projet (intégrité des messages, haute disponibilité, nombre de files, volume de messages...).

Bases de données non relationnelles

Autrefois considérées comme avant-gardistes, les bases de données non relationnelles (communément appelées "NoSQL") prennent progressivement leur place dans nos systèmes d'information. En échange d'un assouplissement des contraintes du modèle relationnel classique, elles promettent le support d'une plus forte volumétrie tout en assurant la performance et la haute disponibilité.

MongoDB

par Olivier Michallat

Très simple de prise en main et attrayante pour les développeurs, MongoDB est l'une des bases NoSQL les plus populaires du moment.

Un modèle orienté document

Dans MongoDB, les données ne sont pas stockées en lignes, mais dans des structures hiérarchiques appelées documents. Chaque entité métier peut être contenue dans un

seul document, sans jointure. Ce modèle s'intègre particulièrement bien avec les langages orientés objet.

L'autre différence avec SQL est l'absence de schéma : les documents n'ont aucune structure prédéfinie et peuvent même être différents les uns des autres.

Bien que ce modèle soit plus naturel et flexible, il demande aussi quelques points d'attention :

- L'absence de jointures impose une dénormalisation plus aggressive au moment de l'insertion des données, ce qu'il faut accepter comme un compromis en faveur des performances de lecture.
- Le schéma de données ne disparaît pas : bien qu'il ne soit plus imposé par la base elle-même, il sera implicitement porté par le code applicatif. Les phases initiales de développement s'en trouvent accélérées ; mais, à terme, les migrations de données sont toujours nécessaires : soit en masse avant le déploiement d'une nouvelle version, soit au fil de l'eau dans le code applicatif, qui doit alors pouvoir gérer plusieurs versions de documents (ce qui est loin d'être trivial).

Un langage de requêtage puissant

Contrairement à d'autres produits de la famille NoSQL, MongoDB conserve la possibilité d'écrire des requêtes dynamiques. Sans atteindre la versatilité de SQL, il met à disposition du développeur un langage riche, disposant d'un grand nombre d'opérateurs ; la principale restriction est l'absence de jointures entre plusieurs documents.

Des mécanismes pour le partitionnement

Dès le départ, MongoDB a été pensée pour le fonctionnement en cluster. Quand un serveur unique ne suffit plus, deux stratégies (complémentaires) peuvent être employées :

- La **réPLICATION** consiste à dupliquer les mêmes données sur plusieurs machines. Elle améliore les performances en lecture, la disponibilité et la tolérance aux pannes.
- Le **sharding** consiste à partitionner les données entre plusieurs noeuds. Il améliore les performances en écriture et augmente le volume de données global du cluster.

Cassandra

par Guillaume Balaine

Autre DataStore NoSQL, Cassandra est "orienté colonne". Créé par Facebook en 2008, puis donné à la fondation Apache et inspiré du design BigTable de Google, Cassandra permet de stocker les données de manière distribuée et dénormalisée.

"Les bases NoSQL promettent le support d'une plus forte volumétrie tout en assurant la performance et la haute disponibilité"

Contrairement aux bases de données NoSQL avec des cas d'utilisation simple à définir comme document ou clé/valeur (des index), les bases orientées colonnes sont définies par leur paradigme. Cela consiste à définir les questions métier à poser aux données avant de concevoir les modèles et schémas.

Cassandra propose des performances linéaires et peut donc être utilisé sans risque dès le démarrage d'un projet. Par ailleurs, même si ce type d'architecture se démarque vraiment pour de fortes volumétries, Cassandra est plus rapide que la plupart des caches distribués du marché. De plus, le fait que Cassandra soit un système distribué complet sans "maître" rend son architecture extrêmement résiliente, tolérante au partitionnement (même au travers de plusieurs DataCenters) et minimise les risques.

Quant à la perte supposée de consistance définie par le théorème CAP (Consistency, Availability et Partition Tolerance), elle peut être réduite presque à néant suivant la configuration BASE de Cassandra, ce qui donne aux entreprises énormément de flexibilité pour un coût modique. Bien que le projet soit jeune (2008), des entreprises du Web qui traitent parmi les plus gros pics d'utilisation au monde comme Twitter, Netflix ou Reddit ont fait le choix de Cassandra.

Le seul réel inconvénient étant la complexité et le manque d'expertise sur le sujet en France à l'aube de 2013. Cassandra, tout comme Neo4J se base sur un paradigme de programmation différent des paradigmes usuels. Les développeurs et concepteurs doivent apprendre à penser autrement et à gérer cette nouvelle liberté avec les données. De même, l'abandon de la consistance immédiate des données nécessite des compétences supplémentaires pour les opérationnels dans la gestion des systèmes distribués et une collaboration accrue avec les développeurs ce qui peut être bénéfique pour le produit (lire le chapitre DevOps du pilier Agile).

Neo4j

par Mathieu Breton

Neo4j, une des bases NoSQL orientées graphe, offre une souplesse et une approche de la persistance très intéressante quand il s'agit d'exploiter des données fortement reliées entre elles, comme par exemple pour gérer des réseaux sociaux ou des listes de contrôle d'accès. Viadeo utilise cette base pour stocker ses utilisateurs et leurs relations. Poussée par de gros investisseurs et dirigée en partie par le créateur de Spring, Rod Johnson, cette base de données un peu particulière est à surveiller dans les prochaines années.

En bref

Simple à mettre en œuvre, portée par une entreprise solide et par une communauté très active, MongoDB est une valeur sûre. Pour les cas d'utilisation qui ne nécessitent pas d'agrégations complexes et peuvent supporter un assouplissement des propriétés ACID, c'est une alternative très intéressante à une base relationnelle.

Les bases orientées graphe restent des outils de niche, qu'il faut garder en tête pour des cas d'utilisations particuliers, tels que le social ou la sécurité.

Les bases orientées colonne, plus compliquées à mettre en œuvre et à concevoir, sont à réservier aux très fortes volumétries (lire le focus sur Big Data).



NOS RECOMMANDATIONS

Parmi les révolutions de l'infrastructure d'entreprise aujourd'hui, nous conseillons :

D'externaliser dans le Cloud votre projet le plus innovant, si possible mené par des équipes agiles et sensibilisées à DevOps, afin d'avoir un projet en production au plus vite à moindre coût.

De faire monter en compétence sur le Cloud une équipe infrastructure/exploitation, car les projets, poussés par les développeurs, arrivent !

D'initier l'abandon des serveurs monolithiques historiques au profit d'architectures modulaires et de conteneurs légers tels que Tomcat.

D'implémenter des architectures fortement découplées, à l'aide d'infrastructures de messaging basées sur ActiveMq ou RabbitMq.

De sortir du carcan des bases de données relationnelles et d'expérimenter des bases plus adaptées au contraintes modernes :

MongoDB, orientée document, avec une grande facilité de requêtage, et ainsi l'utiliser dans le Cloud, as a service !

GROOVY

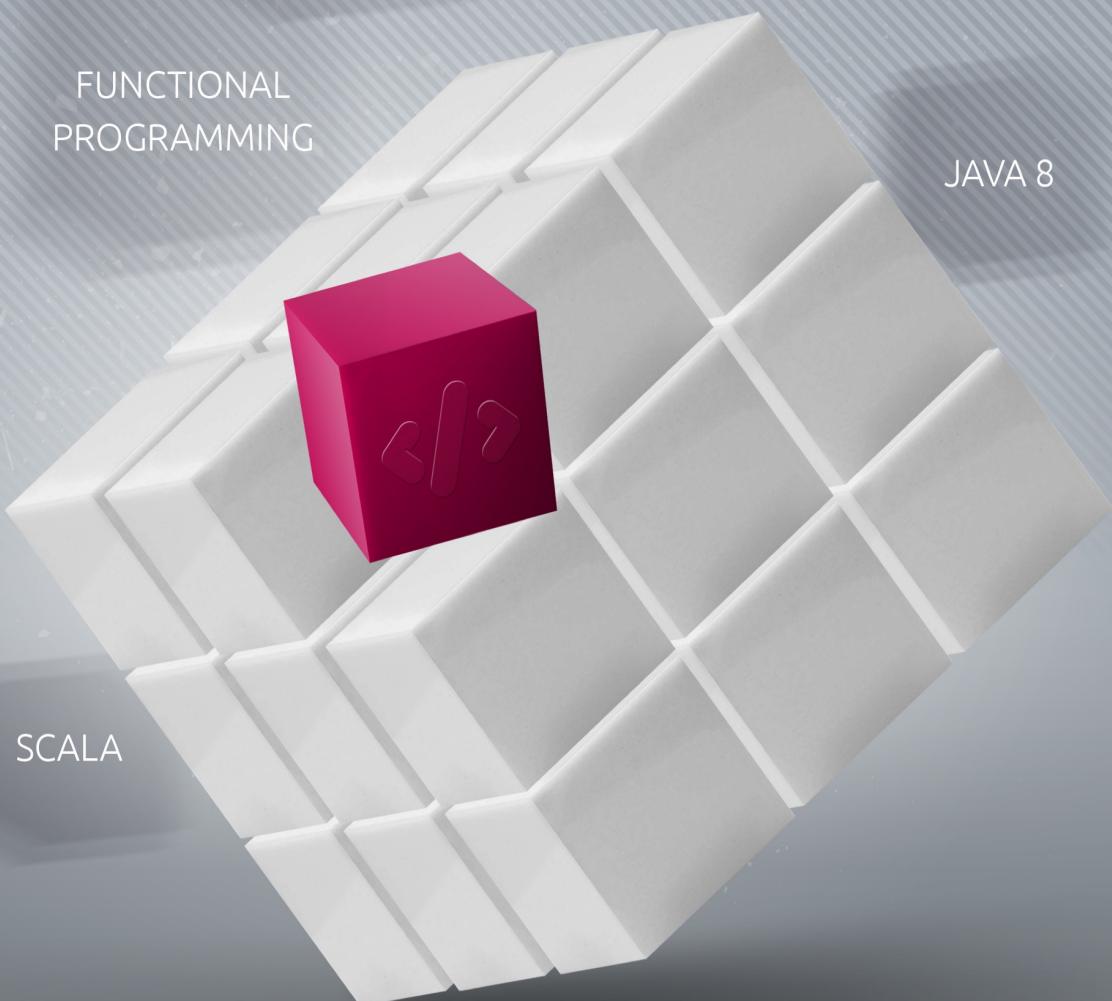
JAVASCRIPT

COFFEESCRIPT

FUNCTIONAL
PROGRAMMING

JAVA 8

SCALA



Langages de la JVM et leurs amis

La JVM abrite un écosystème bien plus large que le seul langage Java. Ces dernières années ont montré qu'elle constitue une zone d'innovation vibrante. Il existe, de nos jours, de nombreux langages qui s'exécutent sur la JVM. Oracle l'a bien compris en améliorant la JVM en ce sens, notamment avec l'API Java Scripting apparue dans Java 6.

Force est de constater qu'un langage unique constitue la plupart des applications actuelles. En 2013 néanmoins, de nouvelles technologies telles que JavaScript se démocratisent dans le monde du Web. Nous utilisons également d'autres formes de langage de description, notamment XML, langage permettant d'embarquer de la configuration dans le monde Java (et ailleurs).

Basés aujourd'hui sur de multiples processeurs, les ordinateurs nous permettraient de profiter de beaucoup de puissance en écrivant du code multi-thread de qualité. Or, il existe des langages, ou plutôt des paradigmes, simplifiant l'écriture de code multi-thread : notamment le paradigme fonctionnel.

Tout laisse à penser que l'avenir de la programmation ne sera plus une monoculture liée à un unique langage. Au contraire, nous nous tournons de plus en plus vers ce que certains appellent déjà la programmation polyglotte.

Dans ce chapitre, nous vous donnons un aperçu des langages dynamiques et fonctionnels en vogue, ainsi que la dernière mouture de Java et de sa JVM.

Programmation fonctionnelle

par Jean Helou

Parmi les styles de programmation répandus dans le monde du développement, le paradigme impératif est de loin le plus utilisé. Il est le descendant direct de la machine de Turing, utilisé dans la plupart des langages de programmation et au centre de l'assembleur et du C. Le paradigme orienté objet a gagné du terrain depuis l'introduction de Java et du C++ qui le combinent avec l'impératif afin d'augmenter la productivité des langages. Le paradigme fonctionnel est, quant à lui, l'héritier direct du lambda-calcul, produit du travail d'Alonzo Church sur la calculabilité. Longtemps confiné au monde académique et à quelques marchés de niche, il connaît aujourd'hui une résurgence face au modèle impératif. Nous voyons même le paradigme fonctionnel prendre petit à petit la place du paradigme impératif dans l'association avec la programmation orientée objet.

Une différence fondamentale a favorisé l'essor du paradigme impératif face à son "ennemi" fonctionnel : Les langages machine et les CPUs suivent tous le modèle impératif, avec une liste restreinte d'instructions et la capacité de changer l'état de la mémoire. Par conséquent, l'implémentation des langages impératifs est plus directe et, à l'origine, plus efficace. Les premières implémentations de langages fonctionnels nécessitant des manipulations très complexes de la mémoire se sont souvent avérées moins performantes que leurs équivalentes impératifs. Avec le temps et l'explosion de l'informatique, les ingénieurs habitués au modèle impératif se sont retrouvés bien plus nombreux que ceux habitués à d'autres formalismes. Certaines sociétés ont créé des CPUs optimisés pour la programmation fonctionnelle mais la masse critique favorisant le paradigme impératif avait déjà été atteinte.

Aux vues de l'investissement financier en faveur du paradigme impératif, les composants développés selon ce modèle ont été orientés pour une réutilisation pragmatique et en plus grand nombre. Chaque nouvelle génération propose un niveau d'abstraction plus important que le précédent, à tel point que la plupart des applications développées aujourd'hui ne sont qu'un assemblage de composants existants, enrichis de quelques règles métiers. Bien qu'il existe des marchés de niche dans lesquels les langages fonctionnels règnent en maîtres (tels que le monde des télécoms dominé par Erlang ou encore les logiciels calculant des preuves de programmes utilisés dans l'armée ou dans l'aérospatiale), tout langage du paradigme fonctionnel qui voudrait percer au grand jour devra nécessairement offrir une compatibilité avec les composants développés selon le modèle impératif.

"Comme il existait du code assembleur au milieu de programmes en C, il existe aujourd'hui du code Java au milieu d'un programme Scala"

De nos jours, plusieurs facteurs viennent susciter un regain d'intérêt envers le paradigme fonctionnel. Pour commencer, une série de percées dans la recherche académique a permis aux langages fonctionnels d'atteindre des performances équivalentes à celles des langages impératifs, dans la plupart des cas grâce à la résolution formelle de certains problèmes de compilation qui permettent de traduire les expressions fonctionnelles, en instructions impératives destinées à la machine. Les progrès en compilation ont effacé progressivement les avantages qu'avaient les langages impératifs de par leur proximité avec l'architecture du CPU. Le paradigme impératif encourage fortement un style de programmation basé sur les effets de bords et la mutabilité, tout en ne proposant qu'une faible expressivité. Ces mécanismes rendent très complexes, voire impossibles, des catégories entières d'optimisations. De par la difficulté intrinsèque de ces mécanismes, le paradigme impératif semble de moins en moins capable de faire face à la complexité sans cesse croissante des applications. À l'inverse, le paradigme fonctionnel, basé sur la transparence référentielle et l'immutabilité

facilite la mise en place de ces optimisations (mémorisation, optimisation des appels récursifs, etc.) en autorisant le compilateur à modifier le programme proposé par le développeur. Ces modifications suivent des règles mathématiques dont la validité a été formellement prouvée et permettent là encore des optimisations inaccessibles aux programmes impératifs. Toute nouvelle optimisation découverte au niveau de ces transformations peut être appliquée sans changement du programme initial, offrant des perspectives de gains par simple recompilations et surtout de réutilisabilité des optimisations. Enfin, le paradigme fonctionnel, en considérant les traitements comme des valeurs à part entière, offre un mécanisme de composition ultime qui favorise la réutilisation, l'expressivité des langages et donc la productivité du développeur.

Ensuite, viennent les progrès en terme d'interopérabilité. Le niveau d'abstraction atteint par les langages impératifs modernes (beaucoup fonctionnent sur des machines virtuelles, qui elles-mêmes sont optimisées pour la machine physique comme la JVM ou la CLR, la VM .NET) fait qu'il est devenu plus simple de créer des langages fonctionnels interopérables avec des composants impératifs. À tel point que les langages impératifs se dotent eux-même de composantes fonctionnelles ! Ainsi, Java, héritier du C, voit arriver les notions de *lambdas* dans sa version 8 — notions que C# intègre depuis déjà longtemps. Sur la JVM, tous les langages de dernière génération (Scala, Clojure, Groovy, Kotlin, Ceylon, etc.) intègrent plus ou moins la programmation fonctionnelle tout en proposant une intéropérabilité avec les composants Java.

La fin du règne de la loi de Moore vient également changer le *status quo*. Les fabricants de puces ayant atteint les limites physiques des semi-conducteurs, l'augmentation de la puissance de calcul passe désormais par la parallélisation et la multiplication du nombres de coeurs dans les CPU. Le style de programmation mutable, encouragé en impératif, nécessite de complexes mécanismes de synchronisation lorsque plusieurs unités de calcul travaillent sur les mêmes données. Les idiomes fonctionnels, encourageant l'immutabilité, nécessitent beaucoup moins de synchronisation, bien qu'ils entraînent souvent des copies de données en mémoire. Ce problème est largement compensé par la diminution du coût de la mémoire vive et les gros progrès réalisés en matière de gestion de mémoire automatisée avec notamment les "Garbage collectors".

Enfin, il y a un facteur beaucoup plus subjectif, assez difficile à mesurer. Nombre de développeurs qui se sont formés à des langages favorisant le paradigme fonctionnel disent qu'ils sont plus productifs qu'avec ceux favorisant l'impératif. Bien qu'il soit difficile de le vérifier, force est de constater que dans plusieurs domaines les abstractions proposées par le modèle fonctionnel sont très intéressantes, en particulier concernant le parcours et la manipulation de collections d'informations.

L'adoption réelle du paradigme fonctionnel dans le milieu pragmatique des entreprises n'est pas encore très visible. Selon certains c'est parce qu'il est considéré comme un avantage compétitif. Cette hypothèse pourrait être confirmée par l'augmentation des offres et solutions émergentes autour de ces langages, à commencer par des domaines

très exigeants en termes de performance comme la finance de marché, les sites internets à gros volume, etc.

Les applications nécessitant de tirer l'absolue quintessence des performances des machines pourront être écrites dans un langage supportant le modèle fonctionnel pour le gain de productivité. Il faudra cependant sans doute encore revenir à du code impératif soigneusement construit pour supprimer les goulots d'étranglement au stade de l'optimisation. Comme il existait, à l'époque, du code assembleur au milieu de programmes en C, il existe aujourd'hui du code Java au milieu d'un programme Scala, dans les zones où le compilateur n'arrive pas encore à optimiser aussi bien qu'un humain.

Java 8

par François Sarradin

Java version 8 ([JSR337¹⁰](#)) devrait sortir au cours du quatrième trimestre 2013. Cette version s'articule autour de deux principaux axes :

1. Améliorer les performances offertes par le langage grâce à une meilleure exploitation des architectures multi-coeurs.
2. Aider le développeur à être plus productif en augmentant l'expressivité du langage.

Dans ce cadre, différentes évolutions sont à prévoir. Deux d'entre elles sont notables : l'amélioration de l'API date ([JSR310](#)) ou l'intégration de Nashorn, le nouveau moteur JavaScript. Mais ce qui va certainement profondément bouleverser le monde Java, c'est l'intégration du projet Lambda ([JSR335¹¹](#)).

Le **projet Lambda** apporte à Java des caractéristiques de la programmation fonctionnelle afin d'atteindre ce double objectif : performance et productivité. Cela passe par une modification du langage et une extension de l'API existante. Côté langage, le développeur voit apparaître les *lambda expressions*, proposant une notation abrégée pour représenter des traitements unitaires qu'il est possible de stocker dans des variables ou dans les paramètres des méthodes. Le développeur voit aussi apparaître les *méthodes d'extension virtuelles*, qui permettent de définir un comportement par défaut au niveau des interfaces. Côté API, il sera désormais possible de réaliser du traitement par lot sur les collections et de paralléliser ces traitements en toute transparence, sans passer par la gestion bas niveau des threads en Java.

¹⁰ <http://jcp.org/en/jsr/detail?id=337>

¹¹ <http://jcp.org/en/jsr/detail?id=335>

Java 8 et donc le projet Lambda sont toujours en cours de développement, mais il est possible de télécharger les versions en cours. De plus, la [mailing list](#)¹² est très active et attire les développeurs à travers le monde, ces derniers ont la possibilité d'exprimer au fur et à mesure leur ressenti sur les nouveautés apportées par le projet.

Les cousins de JavaScript

par Mathieu Breton

JavaScript est désormais partout, langage le plus utilisé sur GitHub, JavaScript permet de concevoir des applications très rapidement. Cependant, il lui est reproché de ne pas être adapté pour les applications d'entreprises. La principale raison est son typage faible. Cette souplesse le rend complexe à maintenir et pas assez prédictif, induisant de ce fait des outils (IDE en particulier) peu évolués. Plusieurs langages tentent de combler ces lacunes.

Les symbiotiques pures

Les langages symbiotiques visent à être exécutés sur une plateforme destinée initialement à un autre langage, comme dans le monde Java Jython ou JRuby sur la Java Virtual Machine. Néanmoins, avec l'émergence des problématiques de grosses applications d'entreprise, ces langages se retrouvent également sur les moteurs JavaScript. Ils apportent tous un typage fort, mais optionnel, à la compilation de la programmation orientée objet et génèrent un JavaScript propre et lisible.

“Ils apportent tous un typage fort à la compilation et génèrent un JavaScript propre et lisible.”

Le plus connu d'entre eux, [CoffeeScript](#)¹³, offre un apport syntaxique important, notamment en supprimant tout ce qui peut être pesant en JavaScript. Cela peut d'ailleurs être un frein pour certains développeurs, trouvant le langage trop dépouillé de structure grammaticale visible et donc, difficile à lire. Ceci n'a pas empêché DropBox de réécrire toute son application dans en CoffeeScript.

Dans la même lignée, Microsoft a sorti [TypeScript](#)¹⁴ il y a quelques mois, qui reste plus proche syntaxiquement parlant de JavaScript et ajoute aussi des concepts permettant

¹² <http://mail.openjdk.java.net/mailman/listinfo/lambda-dev>

¹³ <http://coffeescript.org/>

¹⁴ <http://www.typescriptlang.org/>

une meilleure granularité des applications. Il s'intègre également parfaitement dans la suite Visual Studio.

Un avenir sans JavaScript ?

Les langages symbiotiques offrent déjà une réponse viable à court et moyen terme mais Google ne l'entend pas de cette oreille et pense qu'il est nécessaire de créer un nouveau standard pour les applications Web. **Dart**¹⁵, lancé par Google il y a peu, réunit les avantages des langages symbiotiques avec, en plus, une API riche et du multithreading grâce à sa machine virtuelle. Ses performances sont déjà supérieures de cinquante pourcent à celles de JavaScript, son IDE est évolué et son système de dépendance permet d'inclure facilement d'autres librairies.

Dart n'a cependant qu'un peu plus d'un an et n'est donc pas encore mature. Son avenir va dépendre de son intégration dans les autres navigateurs du marché, qui l'ont pour l'instant refusé. Ceci dit, Dart offre aussi la possibilité d'être compilé en JavaScript.

L'écosystème Groovy

par Aurélien Maury

Groovy¹⁶ est un langage basé sur Java, inspiré de Python, Ruby et Smalltalk. La syntaxe est très allégée par rapport à Java et parfaitement interopérable (possibilité de mixer Java et Groovy). Sa version 1.0 existe depuis 2007. Groovy a été fortement sous estimé durant ses premières années, cantonné dans ses aspects scripting. La communauté Groovy est discrète mais fidèle, ce qui a amené au développement d'un écosystème très riche, aujourd'hui robuste et éprouvé.

“Tous ces projets conservent l'esprit de simplicité d'utilisation et de concision de Groovy”

Une devise de la communauté Groovy pourrait être “Getting things done”, arriver à un résultat fonctionnel et performant, en un minimum de temps et de lignes de code.

¹⁵ <http://www.dartlang.org/>

¹⁶ <http://groovy.codehaus.org/>

Aujourd’hui, Groovy sert de base à de multiples projets Open Source, notamment :

- **Grails**¹⁷

Un framework de développement rapide basé sur les technologies SpringSource. De plus en plus populaire, il permet d’écrire des applications Web tournant sur une JVM en un temps record. Sa structure basée sur un système de plugins le rend facilement extensible. Plus de 800 plugins existent à l’heure actuelle.

- **Griffon**¹⁸

Inspiré par Grails, Griffon reprend ses concepts et les applique aux projets d’application desktop. Ce framework permet de rédiger des applications Swing ou SWT en très peu de lignes de code.

- **Gradle**¹⁹

Développé par la société Gradleware, c’est un outil de build visant à résoudre les problèmes posés à la fois par la rigidité du dominant Maven, et par le manque de cadre du vieillissant Ant. Capable d’importer des build Ant comme Maven, il facilite les migrations.

Il existe également une kyrielle de librairies basées sur Groovy : du client HTTP ([HTTP-Builder](#)²⁰) à la parallélisation de traitements ([GPars](#)²¹), en passant par les frameworks de test ([Spock](#)²²). Tous ces projets conservent l’esprit de simplicité d’utilisation et de concision de Groovy.

Groovy monte en puissance, en visibilité, et s’inscrit dans une forte démarche d’amélioration continue. Pour rejoindre le mouvement, prenez Grails pour votre prochain projet Web, quelle que soit sa taille, ou Griffon pour un projet desktop. Pensez à Gradle si vous devez migrer un système de build vieillissant, apporter un cadre à un système de build un peu anarchique ou au contraire donner un peu de liberté à un projet restreint par l’architecture Maven. Enfin, tout besoin de scripting interagissant avec une JVM devrait être fait couvert par une application en Groovy.

¹⁷ <http://www.grails.org/>

¹⁸ <http://griffon.codehaus.org/>

¹⁹ <http://www.gradle.org/>

²⁰ <http://groovy.codehaus.org/modules/http-builder/>

²¹ <http://gpars.codehaus.org/>

²² <http://code.google.com/p/spock/>

Domain Driven Design en Entreprise

par Frédéric Dubois

Comment créer des applications qui adressent des besoins métier complexes et critiques pour l'entreprise ? Comment combler le fossé qui sépare les connaissances des équipes de développement du besoin métier ?

Le Domain Driven Design (DDD) est une approche du développement logiciel qui prend le parti de centrer la conception des applications sur la modélisation et, en particulier, la modélisation du cœur de métier de l'entreprise. Les aspects techniques passent au second plan. En effet, une erreur introduite dans le cœur du modèle métier aura des répercussions extrêmement fortes et induira des corrections complexes et coûteuses. De même, un modèle bien pensé, robuste, maintenable, évolutif apportera à l'entreprise un élément fort de différentiation, de compétitivité. Qu'est-ce qui fait la différence de mon entreprise ? Le DDD prend le parti de dire que c'est avant tout sur son métier qu'il faut porter l'attention nécessaire pour capitaliser et se différencier.

Les experts du métier et les équipes de développement doivent, par conséquent, travailler en collaboration très étroite et par là même, parler le même langage, se comprendre. Parmi les concepts principaux du DDD, il y a l'élaboration d'un langage commun. Chaque notion est définie par un vocabulaire précis et partagé. Qui n'a pas déjà vu, dans une application, la même notion nommée de plusieurs façons différentes ? Comment peut-on implémenter un logiciel sans comprendre ce qu'il réalise ? C'est pourtant le cas de beaucoup de projets avec des conséquences désastreuses pour la qualité du produit.

"Le DDD prend le parti de dire que c'est avant tout sur son métier qu'il faut apporter l'attention nécessaire pour capitaliser et se différencier"

Outre ce langage commun, le DDD propose un ensemble de principes de modélisation et d'architecture qui permettent de développer un logiciel de qualité, maintenable et fiable. Parmi eux, le plus simple, mais aussi le plus représentatif est le ValueObject. L'idée est d'encapsuler les notions qui ont un sens à part entière. Pourquoi représenter un prix par un entier ou un flottant (comme dans 95% des projets) ? Si vous changez de représentation, il faut revoir la totalité de l'application. Si vous ne gérez qu'une seule devise, vous passez à l'international ? Il vous faut rajouter la devise dans toute l'application. En revanche, si vous représentez un prix par une classe Prix, vous limitez l'impact. De plus, toutes les opérations et les subtilités qui les accompagnent (arrondis, gestion des erreurs, TVA), seront réalisées dans cette même classe évitant ainsi duplication, divergence dans les implémentations, etc.

Le DDD est une approche qui renverse l'ordre établi, dans lequel les équipes de développement commencent par la technique en définissant l'architecture, les frameworks, la base de donnée avant de développer. En recentrant le développement sur le besoin métier, le DDD engendre des applications plus fiables et plus maintenables. Cela nécessite une forte collaboration métier/développement, mais est, de notre point de vue, indispensable sur les projets critiques où le fonctionnel est complexe.



NOS RECOMMANDATIONS



D'importantes évolutions du langage Java, voire même de nouveaux langages, arrivent et devraient modifier les façons de développer.

Nous vous recommandons particulièrement de suivre :

La programmation fonctionnelle, qui permet aux développeurs d'expliciter plus facilement leur code source. Si Java 8 et ses lambdas ne sont pas encore là, vous pouvez d'ores et déjà vous y essayer à travers Scala, porté en entreprise par des frameworks comme Play! ou Akka.

L'émergence de la programmation polyglotte permet d'abaisser le dogmatisme sur un langage en particulier, et ouvre des perspectives importantes en terme d'interopérabilité des outils. Si vos équipes maîtrisent et manipulent au quotidien plusieurs langages, alors n'hésitez pas à sauter le pas et à créer une vraie application multi-langages.

Attention cependant, qui dit plusieurs langages dit une maintenance plus ardue.

La JVM héberge déjà des langages 'alternatifs' robustes. Que ce soit pour du scripting sur la JVM ou pour un projet Web, n'hésitez pas à employer Groovy et son eco-système.

IOS

TITANIUM

APPSTORE

WINDOWS8

SANCHA

PHONEGAP



Mobiles en entreprise

Les révolutions technologiques se sont succédées à un rythme effréné ces cinq dernières années pour nous proposer ce que nous connaissons aujourd’hui : écrans tactiles capacitifs, diagonales de plus de 4 pouces, GPS, réseaux data 3 et maintenant 4G, technologies NFC.

Difficile de croire qu'il y a à peine 5 ans, les usages autour de la mobilité tels que nous les connaissons aujourd’hui n'existaient pas encore. Pour beaucoup, les téléphones et tablettes sont devenus des composants essentiels du quotidien. Ils permettent, bien sûr, de téléphoner, mais également de mesurer les performances sportives, de réserver un billet de train, d'écouter de la musique, ou bien encore de consulter ses mails et de naviguer sur internet.

Cette révolution mobile permet chaque jour d'imaginer de nouveaux usages qui se déportent d'ailleurs de plus en plus du Web vers le mobile pour proposer une expérience utilisateur plus aboutie que celle obtenue derrière un écran d'ordinateur.

Pour de nombreuses entreprises, il est devenu vital d'être présent sur le créneau de la mobilité pour ne pas voir leurs parts de marché s'effriter et leur leadership remis en cause par de nouveaux acteurs.

Cependant, ce créneau est difficile à attaquer, les questions à se poser avant de se lancer étant nombreuses. Contrairement au Web, les plateformes mobiles sont multiples, il faut donc non seulement proposer un produit ergonomique adapté à différentes tailles d'écrans, mais également développer ce produit pour différentes plateformes. Bien sûr, plusieurs possibilités nous sont offertes, mais quels sont les avantages et les inconvénients de chacunes ? Comment faire les bons choix ? Limiter ce choix à des questions budgétaires serait un peu simpliste, il faut prendre en compte la problématique de maintenabilité / évolutivité, mais également la capacité à couvrir de nouvelles plateformes et la réutilisabilité de la base de code.

Les stratégies d'entrée sur le marché

par *Alexis Kinsella*

Bien que beaucoup d'entreprises se soient déjà lancées dans l'aventure de la mobilité, nombreuses sont celles qui n'ont pas encore franchi le pas et cherchent la stratégie adéquate.

Il y a 5 ans, la réponse était simple puisqu'il suffisait d'être présent sur l'AppStore d'Apple avec une application développée pour un seul type d'appareil. Les choses ont quelque peu évolué depuis, puisqu'il est possible d'exister sur pas moins de quatre plateformes majeures : iOS, Android, Windows Phone, BlackBerry, mais aussi sur d'autres plateformes moins représentées. Pour chacune de ces plateformes, il faut gérer plusieurs formats d'écran, mais également une fragmentation logicielle et matérielle.

Pour traiter ce problème et réduire la complexité de la question, une stratégie possible est de choisir une approche ciblée.

L'approche ciblée

Elle consiste à se focaliser, dans un premier temps, sur une plateforme unique, mettant ainsi de côté les problématiques liées à la multiplicité. Le problème de la fragmentation logicielle et matérielle peut être traité de la même façon en choisissant de se focaliser uniquement sur un matériel et en ne supportant qu'une version récente de l'OS.

Cette approche permet de développer très rapidement un produit optimisé pour une plateforme sans subir les contraintes de maintenabilité et de fragmentation. Il n'est pas non plus nécessaire d'avoir à disposition des équipes multi-compétences. Cette approche impacte directement le coût du produit et donc le risque associé. Il faudra cependant se poser la question du portage du produit sur les autres plateformes s'il rencontre un franc succès. En pratique et sauf exception, le code ne sera pas réutilisable sur une autre plateforme. Le produit devra être repensé pour fonctionner de façon optimale sur de nouvelles tailles d'écrans et certaines parties de codes devront être réécrites pour supporter des versions plus anciennes d'OS.

Cette approche est un bon choix lorsque :

- Le produit est créé avec un objectif exploratoire de la mobilité.
- L'objectif est de proposer au plus vite un produit sur le marché, en attaquant, par exemple, la plateforme cible la plus répandue dans le contexte du produit.
- Le produit est spécialisé pour une plateforme.
- Le budget est limité et le besoin de lancer un premier produit est essentiel.

Cette première approche est souvent choisie car elle permet la création d'un produit pour une seule et unique plateforme ce qui simplifie évidemment beaucoup de choses et permet d'avoir un feedback très rapidement. De plus, le risque et les coûts de cette approche sont faibles, permettant de convaincre les décisionnaires à entrer sur le marché.

L'approche globale

L'approche globale consiste à développer une solution en vue de la distribuer sur un maximum de plateformes. Ce choix est à privilégier lorsqu'une stratégie mobile forte est déjà établie, avec une ambition de conquête d'audience rapide.

La mobilité est un écosystème en évolution permanente, aussi bien en termes de plateformes qu'en termes de solutions de développement. Sur ce dernier point, il existe déjà des solutions solides qui permettent généralement de faire un choix pertinent en fonction des besoins de chaque projet.

Le natif

Le premier choix est le natif. Il consiste à développer une application native pour chacune des plateformes ciblées.

Ce choix est intéressant à plus d'un titre. Il permet de bénéficier au mieux des spécificités de chacune des plateformes et donc de proposer aux utilisateurs une expérience optimale sur chacune d'entre elles.

Contrairement aux solutions multi-plateformes que nous verrons ultérieurement, le temps passé à l'interopérabilité est plus faible, puisqu'il est nécessaire de traiter uniquement les problèmes de fragmentation propre à une plateforme, ce qui est souvent déjà bien suffisant.

Lorsqu'une nouvelle plateforme vient à prendre des parts suffisantes pour en proposer le support, il suffit de démarrer le projet sans autre contrainte que la mise à disposition des outils de développement de la plateforme, comme ce fut le cas avec la sortie des Windows Phone cette année. Comme nous pourrons le voir par la suite, le choix de certaines solutions qui ajoutent une surcouche de développement, implique parfois une longue attente avant l'ajout d'un support attendu, support qui reste d'ailleurs parfois hypothétique.

Bien entendu, il existe également des inconvénients à choisir un développement full natif. Tout d'abord, parce que cela a un coût. Il faut disposer soit d'équipes multi-compétentes, soit de plusieurs équipes spécialisées. Cette contrainte risque d'exclure ceux qui ont les budgets les plus limités. Il ne faudra pas non plus négliger que le code d'une plateforme dépréciée est du code perdu. Côté maintenabilité, cela fait autant de code à maintenir dans le temps, il est donc difficile de gérer une redesccente de la capacité des équipes, puisqu'il faut maintenir des bases de codes pour chacune des plateformes ciblées.

Ce dernier point est à nuancer, car rien n'impose de supporter l'ensemble des plateformes du marché. Il est d'ailleurs tout à fait possible de ne supporter que les 2 ou 3

plateformes les plus utilisées. Cela permet en général, de passer en douceur d'une approche ciblée à une approche plus globale lorsqu'un produit rencontre le succès: on démarre le support pour une seconde puis une troisième plateforme.

Le Web

Le Web est omniprésent, aussi bien au niveau grand public qu'au sein des DSI, les équipes projets en ont d'ailleurs habituellement une bonne maîtrise. L'option du Web semble évidemment être une piste alléchante. On parle beaucoup du Web as a Platform. Qu'en est-il dans le domaine de la mobilité ? Est-ce bien adapté ?

En observant de plus près les contraintes des plateformes mobiles, quelques éléments nous permettent de nous faire une idée objective. Tout d'abord, le Web se passe dans le browser. Une chance pour nous, les browsers mobiles sont plutôt respectueux des normes HTML et le support des normes récentes est bon (HTML5 / CSS3). Là où le bâ blesse, c'est au niveau des performances des browsers mobiles. Malgré un support des normes efficace, les performances côté JavaScript et côté rendu sont loin d'être au rendez-vous: ce qui est performant sur un desktop n'est pas assuré de l'être sur un browser mobile. Par ailleurs, une composante majeure de la mobilité est la gestion de la déconnection, de la latence et des débits. La problématique des débits est en passe d'être traitée par les performances des nouveaux réseaux opérateurs. Le débit n'est, en fait, pas un problème par rapport aux problématiques engendrées par la latence ou la perte de connection. Qui n'a jamais pesté contre une page qui met plusieurs minutes à s'afficher ou bien qui finit par n'être chargée que partiellement ?

"Le Web reste le seul standard multiplateformes"

Les browses n'apportent que des solutions partielles à ces problématiques. L'API local-Storage peut apporter une solution efficace pour le stockage des données offline, mais l'API de cache des ressources offline n'est pas convainquante ou du moins suffisante pour gérer proprement un mode offline. De même, il est difficile de pré-cacher des données puisque les API de type Webworker sont très mal supportées par les browsers mobiles. Il en va de même pour les WebSockets qui offrent un support efficace pour l'échange de données, mais qui restent mal supportées par les différents navigateurs mobiles. Une API permet de gérer des événements notifiant de l'état online ou offline de l'appareil mobile. Cependant, là aussi, le support est assez médiocre et ne permet pas de se reposer dessus de manière fiable, même avec les dernières versions des OS mobiles. Enfin, dernier problème mais pas des moindres, lorsque le navigateur est envoyé en fond de tâche, l'application Web est mise en pause, sans possibilité de continuer l'exécution. Difficile dans ce contexte de télécharger des données en tâche de fond ou bien de réveiller l'application Web via des notifications extérieures.

Malgré quelques APIs Web complémentaires orientées mobilité telles que la *location API*, la *motion API* ou bien encore l'*orientation API*, les APIs normalisées orientées mobilité sont encore peu nombreuses. Il suffit cependant de se rendre sur la [page des APIs²³](#) de la fondation Mozilla pour se rendre compte que les APIs Web potentielles sont nombreuses. Malheureusement, la plupart ne sont pas normalisées et ne seront pas disponibles dans nos browsers mobiles avant un certain temps.

Le Web mobile reste particulièrement utile lorsqu'il joue un rôle d'extension des sites Web classiques visant à fournir une navigation adaptée à nos combinés mobiles. Il est bien sûr possible de mimer les interfaces natives, mais attention à ne pas tomber dans le piège de l'[Uncanny valley²⁴](#) qui aura tendance à renvoyer une impression désagréable à vos utilisateurs. La gestion des listes dans un browser en est un très bon exemple. De nombreuses applications Web tentent de reproduire l'[effet momemtum²⁵](#) des interfaces natives sans parvenir à un résultat convenable.

L'hybridation Web

Rassurez-vous, tout n'est pas perdu. Si vous souhaitez exploiter les compétences Web de vos équipes, il est toujours possible de passer par la solution des applications hybrides. Le développement d'une application hybride consiste à mélanger différentes technologies. En l'occurrence, dans le domaine de la mobilité, il s'agit de développer une application qui mélange les technologies natives avec les technologies Web.

L'idée est de maximiser l'usage des technologies Web, voire même de gommer tout développement natif en utilisant des frameworks qui vont se charger de fournir l'accès à des fonctionnalités natives au travers d'un bridge JavaScript. L'objectif est de s'appuyer sur des fonctions disponibles uniquement au travers de développement d'applications natives. Pour ce faire, il faut fournir des API manquantes à votre browser.

Afin de réellement gommer les différences avec les applications natives, il faut être en mesure de proposer leur distribution au travers des stores classiques, tels que Google Play ou bien l'App Store. Pour répondre à ce besoin, les frameworks hybrides proposent des coquilles vides d'applications natives, qui vont se charger de lancer votre application en chargeant les pages HTML au travers d'une WebView. Cela présente plusieurs avantages, notamment la suppression de la barre de navigation qui a tendance à gaspiller un espace précieux alors que vous êtes dans un navigateur et non dans une application native.

²³ <https://wiki.mozilla.org/WebAPI>

²⁴ http://en.wikipedia.org/wiki/Uncanny_valley

²⁵ <http://lab.cubiq.org/iscroll/examples/simple/>

PhoneGap Le gap est comblé entre les capacités classiques des applications Web et les capacités des applications natives. Le framework de référence en la matière s'appelle [PhoneGap²⁶](#). Son nom est évocateur de son objectif.

PhoneGap permet, entre autres, de lever la Sandbox de sécurité des WebViews et ainsi d'effectuer des requêtes cross-domain sans se soucier des restrictions habituelles.

Un inconvénient majeur reste le besoin de disposer de l'ensemble des outils de build des différentes plateformes ciblées pour construire les applications. Si nous souhaitons à la fois développer une application pour iOS et Windows Phone, il faudra disposer d'une station de travail avec ces différents environnements ou, au mieux, jongler avec différentes VM.

Les outils tels que PhoneGap se basent sur des technologies Web au travers de l'usage de WebViews. Il faut cependant savoir que les WebViews proposées au sein des OS mobiles ne sont pas toujours optimisées tel qu'on pourrait l'attendre. Ainsi, les WebViews Android ne sont pas basées sur le moteur Web Chrome pour mobile, hormis sur les versions les plus récentes de l'OS et elles ne bénéficient pas de toutes les optimisations du moteur Web proposées par Chrome. De même sur iOS, le moteur JavaScript Nitro permettant d'exécuter du code JavaScript jusqu'à trois fois plus rapidement n'est pas disponible. Il repose sur la capacité du moteur JavaScript à utiliser un compilateur JIT (Just In Time), ce qui nécessite de marquer les pages de mémoire RAM comme exécutables et n'est pas autorisé pour une application tiers sur iOS.

Une des richesses d'un outil tel que PhoneGap réside dans sa capacité à tirer parti de plugins permettant d'ajouter des capacités au framework de base. Ainsi, il est possible, grâce à une liste conséquente de plugins disponibles sur la grande majorité des plateformes, d'utiliser l'appareil photo pour lire des codes barres, de s'intégrer avec Google Analytics ou bien encore de gérer les notifications push.

Les alternatives Des alternatives à PhoneGap existent. Sencha Touch propose un framework fullstack permettant de développer des applications mobiles avec les technologies Web. Sencha Touch est un framework orienté performance, dont les applications sont essentiellement développées via JavaScript. L'outil propose un packaging natif pour iOS, Android et certains combinés BlackBerry.

Trigger.io, moins connu, propose également une solution intéressante basée sur les mêmes principes que PhoneGap. La solution, principalement sur le Cloud, supporte aussi bien iOS, Android que Windows Phone. Le produit met en avant sa meilleure intégration avec le système, ainsi que la capacité de mettre à jour le contenu des applications sans avoir à resoumettre les applications. Cela permet d'éviter d'anticiper des temps de validation qui peuvent atteindre plusieurs jours comme cela est le cas

²⁶ <http://phonegap.com/>

sur l'App Store. Le service propose, en complément, un système de plugins permettant d'étendre le framework de base pour y ajouter les fonctionnalités manquantes. Il met également à disposition un outil permettant de débugger facilement les applications développées avec l'outil.

L'hybridation exotique

Si les précédentes solutions ne conviennent pas à certains besoins, il existe des solutions un peu plus exotiques.

Ces dernières sont généralement basées sur des runtimes intégrés aux applications ou bien sur des compilateurs qui se chargent de construire des applicatifs natifs à partir d'applications développées avec des langages autres que ceux des plateformes cibles.

Titanium Le produit le plus connu de cette famille est Titanium. Celui-ci existe depuis plusieurs années et propose de créer votre application en se basant sur des technologies Web avec un SDK JavaScript et créer, à partir de ce code, une application native.

La formule peut se révéler attirante, mais qu'en est-il de la pérennité de ce type de produit ? L'utilisation d'un tel produit induit de nouveaux risques à accepter :

- La disparition potentielle du produit. Chaque révolution informatique apporte son lot d'outils de ce type, mais leur pérennité n'est jamais garantie. Le lock-in technologique peut coûter très cher en cas de disparition ou d'obsolescence.
- La couverture des plateformes est limitée par le support proposé. Titanium d'AppCelerator propose, par exemple, un support d'iOS et d'Android, mais qu'en est-il pour une nouvelle plateforme, telle que Windows Phone, qui risque fortement de gagner une part non négligeable d'utilisateurs sur 2013 ?

Xamarin Il existe, bien entendu, d'autres outils concurrents proposant des solutions originales. Parmi eux, nous pouvons citer la solution de Xamarin qui se propose de développer des applications Android ou iOS natives avec pour seul langage C#. Cet outil est tout particulièrement intéressant si vous disposez d'équipes spécialisées dans les technologies Microsoft et DotNet. Malheureusement, les API de support d'iOS et d'Android ne semblent pas être les mêmes, il n'est donc pas possible de maintenir une base de code unique. L'absence de support de la plateforme Windows Phone reste étonnant pour un outil de développement C#, mais s'explique par l'absence d'API unifiée entre les 3 OS majeurs : pourquoi proposer une API doublon à l'API officielle alors qu'il serait possible de tirer parti d'une API unifiée ?

RubyMotion Les équipes ayant des compétences Ruby seront, quant à elles, intéressées par l'outil RubyMotion qui propose de développer des applications iOS avec le langage Ruby. Le produit est basé sur le runtime MacRuby qui a été adapté pour être intégrable au binaire des applications iOS. Malgré quelques restrictions techniques liées aux contraintes imposées par la plateforme iOS, le produit RubyMotion permet de tirer parti des aspects dynamiques du langage Ruby et de simplifier le développement d'applications pour iOS qui se révèle parfois complexe.

Cette solution n'est pas multi-plateformes, mais permettra aux développeurs Ruby, nombreux dans le monde Web, de tirer parti de leurs compétences afin de proposer des applications pour une des plateformes mobiles majeures.

Par ailleurs, la plateforme semble jouir d'un support très enthousiaste de la communauté Rubyiste, connue pour être très prolifique en contributions Open-Source autour de Ruby On Rails.

Conclusion Bien-sûr, ces formats hybrides sont plus exotiques que la simple hybridation Web. Cependant, les solutions proposées sont matures et correspondent à des marchés de niches bien identifiés.

Comment faire les bons choix ?

Difficile de choisir la bonne stratégie pour démarrer un nouveau projet mobile. Cependant, des éléments tangibles tels que la couverture des différentes plateformes, le souhait d'être simplement présent dans le domaine mobile avant de se lancer ou bien proposer une expérience utilisateur optimale sont autant d'éléments à prendre en compte.

Les compétences des équipes font également partie des aspects à ne pas négliger avant de démarrer un projet mobile. Sont-elles orientées Web ou non ? Les technologies du Web ont aujourd'hui une importance toute particulière, elles représentent à ce jour le seul standard interopérable entre les différentes plateformes. Il y a donc tout intérêt à maximiser leurs usages, là où l'expérience utilisateur n'en pâtrira pas.

Le choix de maximiser l'usage des technologies Web permet non seulement de capitaliser au fur et à mesure des développements, mais également de maintenir une application basée sur des technologies moins spécialisées et de réduire la quantité de travail à produire pour supporter de nouvelles plateformes.

L'usage des WebViews PhoneGap au sein d'applications réellement natives est un bon moyen d'y parvenir, même s'il est possible d'aller plus loin dans l'intégration des technologies Web et du natif.

Développer avec les outils Cloud

par Alexis Kinsella

Le Cloud devient incontournable dans l'informatique moderne et encore plus dans le contexte de la mobilité. Les outils Cloud y sont légions et permettent de développer plus rapidement les applications mobiles.

Les applications mobiles sont faites pour être connectées. Le plus souvent, les combinés mobiles ne portent pas les données mais les consomment ou en produisent.

Un mobile repose presque toujours sur une partie client, le plus souvent mise en avant, ainsi qu'une partie serveur dont on parle un peu moins.

Les technologies Backends

Quelques technologies se sont imposées comme des standards essentiels à maîtriser. Ainsi pour communiquer entre le client et le serveur, les transports à privilégier sont le HTTP ou bien les WebSockets.

De même, pour transporter les données, il est nécessaire d'utiliser des formats facilement exploitables par les applications mobiles. Dans ce domaine JSON règne en maître et la tendance lourde est au design d'APIs RESTful car facilement intégrables que ce soit au travers des technologies Web ou natives.

JSON n'est cependant pas le seul format à pouvoir être utilisé pour échanger des données. D'autres formats peuvent être exploités pour des besoins d'optimisation de bande passante ou de performance lors des phases de lecture/écriture. Dans de tels cas, des formats binaires tels que Protobuf seront à privilégier. Ce dernier optimise au maximum la bande passante en limitant la redondance. Protobuf a également l'avantage de reposer sur un format à base de descripteurs contrairement au format JSON, ce qui permet de connaître facilement les structures des messages échangés et facilite la création du code d'intégration.

La mise à disposition de données sur internet implique, sauf exception, de protéger ces données au travers de protocoles d'identification et d'autorisation. Les protocoles qui s'imposent aujourd'hui dans le monde Web sont OpenID et OAuth. Les principaux acteurs du Web utilisent ces standards pour l'accès à leurs services. La normalisation de l'accès aux données permet, d'une part, de s'appuyer sur des librairies communes facilitant les développements et d'autre part de ne pas obliger l'utilisateur à systématiquement créer un compte pour un nouveau service en lui donnant la possibilité d'utiliser ses différentes identités (Twitter, Facebook, Google) pour se connecter. Ainsi, le nombre de comptes qu'il a à gérer est significativement réduit.

Backend as a Service

Créer une application est déjà coûteux, mais développer les backends associés peut induire des défis bien plus complexes de type infrastructure, scalabilité, sauvegarde des données et autres joyeusetés. Si nous ajoutons à ce temps de création des applications mobiles, celui de création de backends solides, correctement architecturés, avec une infrastructure qui tient le coup, nous pouvons rapidement arriver à des délais et coûts de développement rédhibitoires.

Les solutions Cloud et type IaaS comme Amazon EC2 adressent déjà une partie de la problématique en permettant de s'affranchir des complexités de mise en oeuvre d'une infrastructure. Cependant, des solutions de type PaaS telles qu'Heroku ou bien encore CloudFoundry vont plus loin. Elles proposent de gérer, non seulement les problématiques d'infrastructure et de scalabilité, mais également celles de gestion des déploiements de middlewares tels que les bases de données relationnelles ou NoSQL. Heroku propose, par ailleurs, de choisir, comme les briques d'un Lego, les SaaS avec lesquels l'application doit s'interfacer et permet de gérer la facturation de ces services. Une grande partie du problème est alors réglée. Il existe bien entendu d'autres services comme AppFog ou encore DotCloud proposant des supports similaires à ceux proposés par Heroku, mais ce dernier reste la référence en la matière.

Malgré tous ces services permettant de s'affranchir de la gestion de l'infrastructure, de la scalabilité, des middlewares et de backup des données, l'élaboration des backends reste très coûteuse en énergie.

Partie stratégique, le backend est encore habituellement développé avec les méthodologies classiques, puis déployé sur un serveur d'application ou bien sur un service Cloud. De nombreuses librairies permettent de faciliter et de structurer les développements. Cependant, ces derniers peuvent être perçus comme fastidieux et parfois comme une véritable perte de temps quand l'ambition est de produire au plus vite. C'est pourquoi, depuis quelques temps, des services qui proposent de simplifier cette problématique sont apparus, en fournissant des solutions de backends quasiment prêtes à l'emploi. On appelle cela les technologies BaaS pour Backend as a Service. Le fer de lance de ces nouveaux outils est parse.com²⁷.

“Les outils Cloud se proposent de gérer votre cœur applicatif et vos données à votre place”

Ces outils fonctionnent, bien entendu, sur le Cloud et en reprennent tous les avantages mais en y ajoutant un petit plus qui fait toute la différence : ils se proposent de gérer votre cœur applicatif et vos données à votre place. Le principe est simple, il repose sur

²⁷ <http://parse.com>

le stockage des données applicatives et de leur exposition automatique via un accès RESTful gérant les aspects d'identification et d'autorisation pour vous. Les données sont généralement stockées dans des bases NoSQL de type Document offrant une très grande souplesse sur la façon dont le format des données peut être amené à évoluer. Les outils tels que parse.com proposent des SDK ciblant les principales plateformes, qu'elles soient de type mobile, desktop ou Web. Elles proposent donc une solution globale avec une intégration profonde côté client et, par conséquent, une délégation forte de la complexité d'intégration avec les backends.

Ces services proposent généralement une intégration poussée avec différents services en ligne, qu'ils soient sociaux via des intégrations de type Twitter ou Facebook, ou bien techniques pour s'interfacer avec les services push des plateformes mobiles ou encore l'envoi d'emails.

Parmis les services qui fleurissent sur le Web, il est possible de citer: parse.com²⁸, [stackmob](http://stackmob.com)²⁹, kinvey.com³⁰, ou bien encore [deployd](http://deployd.com)³¹.

Les limitations de ce type de solutions résident dans la gamme de services proposés qui peut devenir obsolète dès lors que de nouveaux besoins émergent. Il devient alors parfois difficile de trouver des solutions. Ce problème met en avant un locking technologique très fort entre les développements de l'application client qui délègue une grande partie de complexité au SDK, ainsi qu'une dépendance totale au produit côté backend. Revenir vers une solution plus classique une fois le produit lancé présente des risques. Ce locking rend le produit dépendant des aléas de la solution Cloud en matière de maîtrise des coûts ou bien encore d'obsolescence potentielle de celle-ci. Mais rien de nouveau ici au regard des problématiques de locking technologique de certaines solutions déjà connues telles que l'AppEngine de Google ou bien encore le CRM salesforce.

Ces solutions sont donc très alléchantes car elles font gagner beaucoup de temps et permettent de faire des économies non négligeables, même s'il faut faire attention aux risques cachés. Les candidats idéaux pour ce type de solutions sont les applications avec des durées de vie limitées pour lesquelles il est nécessaire de stocker des données, ou bien encore de les exposer. Les applications événementielles ou les jeux sont de bons exemples.

²⁸ <http://parse.com>

²⁹ <http://stackmob.com>

³⁰ <http://kinvey.com>

³¹ <http://deployd.com>

Usines logicielles en ligne

Les usines logicielles en ligne représentent une autre facette intéressante des outils Cloud dans un contexte mobile. Les outils [Trigger.io³²](#) ou encore [PhoneGap Build³³](#) en sont de très bon exemples. Ils permettent de s'affranchir de problématiques d'industrialisation telles que la génération de livrables pour les différentes plateformes mobiles. Les équipes de développement peuvent se focaliser davantage sur la création de l'application sans avoir à se soucier des problématiques de mise en place d'une usine logicielle.

Le service PhoneGap build permet, par exemple, de déclencher la construction de l'ensemble des livrables ciblés via un simple push de la dernière version du code source de l'application vers un repository Git. Le service trigger.io, quant à lui, ajoute la possibilité de mettre à jour directement l'application sans avoir à la relivrer vers le parc d'applications installées via une technologie de push de modifications.

Conclusion

Les outils Cloud font aujourd'hui partie intégrante de la trousse à outils des équipes projets. Il serait dommage de passer à côté, mais restons pragmatiques, il ne s'agit pas non plus d'une course à l'armement.

L'état du marché des OS mobiles

par Alexis Kinsella

L'année 2012 fût l'année de la confirmation de la suprématie de Google avec son système Android, mais aussi celle de la décadence de BlackBerry et de la renaissance de Microsoft avec la sortie de son système Windows Phone 8. Que peut-on attendre de l'année 2013 ? Une chose est sûre, le marché du mobile est un marché en perpétuel mouvement et le status quo est loin d'être l'option sur laquelle parier.

Bien sûr, le système Android gardera pour lui une part presque indécente du gâteau et Apple s'accrochera tant bien que mal à sa seconde place, mais nos deux systèmes leaders risquent de se faire bousculer par leurs challengers qui ont affûté leurs armes en 2012.

³² <https://trigger.io/>

³³ <https://build.phonegap.com/>

Windows Phone

Windows Mobile n'est plus qu'un mauvais souvenir et Microsoft avec Windows Phone, en collaboration avec Nokia et HTC a vraiment des arguments à faire valoir, d'une part parce que Microsoft n'a plus à rougir de son système et d'autre part car le matériel est à un niveau suffisant pour concurrencer les deux leaders. Le système a fait un énorme pas en avant en seulement quelques mois pour passer d'un Windows Phone 7 rapidement né à un système plus mature et surtout innovant et différentiateur grâce à un look & feel unique. Le geek aura un peu de mal à être convaincu et ira plutôt vers un système plus ouvert de type Android, mais ce produit répond aux exigences du grand public ainsi qu'aux attentes des professionnels.

BlackBerry

Concernant BlackBerry, l'affaire est plus complexe. La société joue son dernier atout avec son nouveau système BlackBerry 10 et ses nouveaux combinés attendus pour le premier trimestre de l'année 2013 dont le BlackBerry Z10 disponible depuis le 17 février en France. Un lancement raté pourrait bien signifier, à terme, la fin de BlackBerry. Dans le domaine de la mobilité, le marché ne connaît aucune pitié et une société à la pointe un jour peut être totalement dépassée le lendemain. Le cas de Palm est éloquent: la société a connu de nombreuses déconvenues pendant des années avant qu'HP ne lui donne le coup de grâce. On peut également citer l'exemple de Windows Mobile devenu obsolète en un temps record avec l'arrivée sur le marché d'iOS et d'Android. Il n'y a pas de place pour la médiocrité sur un créneau aussi concurrentiel. C'est pourquoi l'année 2013 sera décisive pour BlackBerry et sa survie à plus long terme.

La génération de combinés BlackBerry 10 bénéficie d'une base logicielle complètement retravaillée reposant maintenant sur QNX, BlackBerry ayant racheté ce système pour en équiper ses téléphones. QNX est intéressant à plus d'un titre, tant en termes d'optimisation qu'en termes de sécurité et fiabilité. Il équipe de nombreux matériels critiques dans des domaines variés tels que le médical, l'aérien ou bien encore le nucléaire.

La tablette PlayBook a été la première tablette à être équipée du nouveau système BlackBerry, mais le système arrivé trop tôt et sans aucun polish a renvoyé des impressions très contrastées aussi bien auprès du grand public qu'auprès des professionnels. Cela s'est d'ailleurs fait sentir sur les ventes du PlayBook qui n'ont pas décollé.

"C'est pourquoi l'année 2013 sera décisive pour BlackBerry"

On peut espérer mieux pour le démarrage de la gamme smartphone équipée du système BlackBerry 10 et voir un système avec un vernis digne de ce nom. En attendant,

BlackBerry essaye d'aller là où les attentes des clients n'ont pas encore été comblées par la concurrence, c'est à dire sur le marché des professionnels, marché dans lequel les équipements BlackBerry se distinguent encore.

BlackBerry souhaite, par exemple, proposer un réel support au BYOD (Bring Your Own Device) en proposant de gérer nativement sur leurs téléphones deux partitions distinctes, une pour l'usage privé et une seconde pour l'usage professionnel. Toutes deux seraient cryptées de façon indépendante. La partition professionnelle isolée de la partition privée pourrait être activée à la demande par les entreprises et gérée sans interférences avec les données personnelles.

L'idée est de proposer aux entreprises une gestion intelligente de leur parc d'appareils mobiles avec une vraie stratégie BYOD mature. Il faut toutefois prendre en compte que le BYOD repose également sur une adoption de ce genre de combinés à titre personnel, ce qui est loin d'être évident au vu de la concurrence acharnée que livrent les concurrents directs de BlackBerry. Si les nouveaux terminaux ne remportent pas le succès escompté auprès du grand public, BlackBerry se retrouverait dans une situation plus classique dans laquelle ce sont les entreprises qui fournissent le matériel à leurs salariés. Ceci dépouille la notion de BYOD d'une partie de son intérêt, les entreprises devant toujours financer une grande partie de leur parc mobile.

Sur quel système parier ?

S'il s'agit de développer une application mobile, la donne ne changera pas en 2013: il sera toujours essentiel d'être présent sur les plateformes iOS et Android. D'une part, parce qu'iOS est la plateforme qui génère encore aujourd'hui le plus de traffic mobile et de transformation d'achat, d'autre part, parce que le parc de terminaux Android est le plus vaste. En toute logique, en développant pour ces deux plateformes, vous serez amenés à toucher un très large public.

Le "petit nouveau" à ne pas négliger est Microsoft avec son système Windows Phone qui a réussi à faire de son produit un concurrent tout à fait sérieux et crédible, capable de gagner des parts du marché mobile en 2013.

Parier sur BlackBerry est plus risqué pour la partie grand public, mais cela peut aussi être un vecteur de rentrée d'argent intéressant, puisque la concurrence est plus faible et les prix pratiqués sur le Market BlackBerry plus élevés du fait de la cible professionnelle. Les combinés BlackBerry disposant d'un bridge pour faire tourner les applications Android, il peut être intéressant de cibler à la fois la plateforme Android et BlackBerry avec un développement unique.

Le Web est, quant à lui, une valeur sûre. Plus que jamais les alternatives basées sur les développements Web mobiles sont solides et vont gagner mois après mois en crédibilité, ainsi qu'en visibilité. Faire le choix des applications Web est un pari sur l'avenir,

un avenir proche mais qui ne laisse pas de place à la médiocrité. Le développement d'applications Web de qualité implique, encore aujourd'hui, de relever de nombreux défis techniques en termes de qualité perçue, mais s'avère être un choix pertinent dans une optique de pérennité, de maintenabilité et de réutilisabilité. Le Web est le seul standard viable multiplateformes.

Les nouveaux usages

De nouveaux usages mobiles et connectés apparaissent régulièrement. L'année 2013 sera, selon toute vraisemblance, une année propice à l'émergence de nouveaux produits et usages: le marché des smartphones et des tablettes est en train de devenir mature. Les grands constructeurs préparent le terrain pour lancer de nouveaux produits. Apple semble être dans les starting blocks car ils sont attaqués de toute part sur leurs produits phares. La société de Cupertino a besoin de renforcer sa gamme avec de nouveaux produits innovants. L'année 2013 sera, bien sûr, l'occasion de présenter de nouveaux produits autour de la notion de TV connectées. Apple pourrait être amené à enfin sortir une TV connectée pour concurrencer celle de Google.

Une chose est sûre, Apple a fait de la place dans son agenda en groupant les sorties de ses tablettes et téléphones au mois de septembre et ainsi il est possible de mettre en valeur de nouveaux produits au cours du premier semestre. Cela peut vouloir dire deux choses: Apple compte accélérer la cadence de sortie de ses nouveaux produits dans ce contexte très concurrentiel pour tirer son épingle du jeu, ou alors souhaite proposer des produits totalement nouveaux.

“Qu'on se le dise, l'heure est au matériel connecté”

On parle déjà d'un nouvel iPad Mini pour mars 2013, voire même d'un tout nouvel iPhone Mini, mais ce qui est plus probable est la sortie d'un produit totalement innovant tel qu'une montre connectée. L'avant dernière génération d'iPod Nano de forme carré, matériel Apple le plus petit disposant d'un écran tactile, a remporté un franc succès et a vu apparaître de nombreux accessoires permettant de le porter comme une montre. Curieusement, alors que le produit était largement adopté, celui-ci a disparu des étalages en septembre dernier en faveur d'un iPod Nano avec une form factor rectangulaire incompatible avec le port au poignet.

Apple pourrait donc sortir très prochainement une iWatch et venir concurrencer un secteur de niche en plein essor. Sony et la société italienne i'm Watch ont déjà tenté l'expérience avec un système basé sur Android, mais c'est dans le monde du sport que ce type de montre est le plus répandu. Elles sont, en général, accompagnées d'un GPS et permettent de mesurer les performances sportives via différents capteurs.

Ces montres telles que l'iWatch ou bien la i'm Watch ont pour ambition de proposer une expérience connectée plus aboutie que les montres sportives et se veulent un relais complet avec un téléphone. Les usages ne sont pas encore vraiment identifiés, mais Apple a déjà démontré sa capacité à créer un marché de toutes pièces avec celui des tablettes, alors parions sur son aptitude à reitérer l'expérience en créant un marché autour des montres connectées.

Qu'on se le dise, l'heure est au matériel connecté. S'il ne s'agit pas de montre, ce sera un autre appareil. Samsung l'a prouvé avec son appareil photo Galaxy Camera. La prochaine étape pourrait bien être la démocratisation de la domotique auprès du grand public avec un nombre toujours croissant d'outils domestiques interconnectés.



NOS RECOMMANDATIONS

Entreprise Mobile

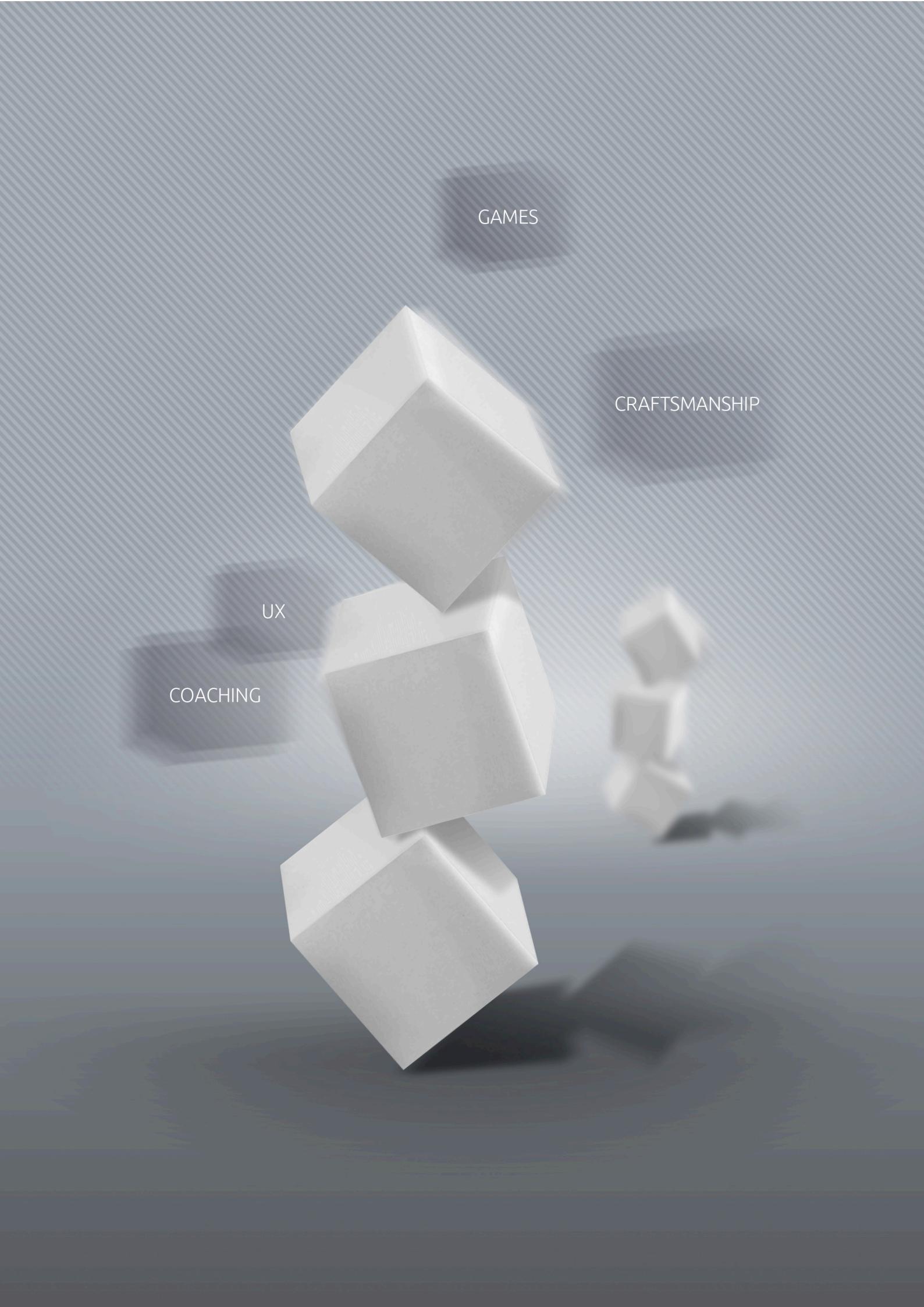
Ciblez en toute confiance les deux plateformes majeures que sont iOS et Android pour une entrée sur le marché. Concernant les autres plateformes, le choix est plus complexe et nécessite une analyse précise des publics ciblés par votre produit.

Le Web reste le seul standard multiplateforme. Capitalisez sur vos équipes Web !

2013 est l'année du tout connecté, il ne faut pas hésiter à se lancer si vous souhaitez prendre une place sur le créneau.

Les problèmes de qualité et de dette technique ne sont pas réservés au monde du développement serveur, ils touchent également les développements Web et mobiles. Vous en occuper dès aujourd'hui vous permettra d'avoir des bases saines !

N'oubliez pas de prendre en considération l'aide que peuvent vous apporter des solutions SaaS. Elles permettent de rester focalisé sur le produit initial tout en déléguant certaines complexités, telles que la gestion d'infrastructure, à des services spécialisés.



GAMES

CRAFTSMANSHIP

UX

COACHING

Agilité et Craftsmanship

L'agilité a pris de l'ampleur ces dernières années. Nées à la fin des années 80, les méthodes agiles permettent aux entreprises d'être plus réactives et plus pertinentes pour leurs utilisateurs tout en délivrant des logiciels de qualité. Certaines entreprises l'ont compris très tôt et savent aujourd'hui en tirer parti pour réussir. LinkedIn, par exemple, a plus de 150 millions de membres à travers le monde, fidélisés par une application en perpétuelle évolution toujours plus agréable à manipuler.

Inspirées par ces success stories, beaucoup d'entreprises cherchent à expérimenter l'agilité. La transformation d'une organisation depuis un fonctionnement en cascade n'est néanmoins pas trivial. L'avenir de l'agilité est à présent fortement corrélé à la capacité des entreprises à changer. Une fois le cap franchi, le logiciel n'est plus un coût pour l'entreprise mais une véritable valeur métier permettant d'accroître sa compétitivité.

2013 la phase de maturité de l'agilité

par Gilles Mantel

L'agilité a atteint en 2012 une vague d'adoption sans précédent. Nous sommes probablement proches du pic de la courbe d'adoption de l'innovation, zone qui définit la transition entre la majorité précoce et la majorité tardive. En 2012, la plupart des grandes DSI françaises avait déjà expérimenté au moins un projet en agile. Les retours d'expériences sont donc de plus en plus nombreux et, avec l'adoption en masse, vient également la multiplication des échecs : les principes d'origine se diluent et le modèle subit parfois des distorsions importantes.

Le constat est malgré tout assez clair, l'agilité est un modèle qui tend à se généraliser. Le Gartner conseillait d'ailleurs dès le début 2012 d'abandonner complètement le cycle en cascade (<http://www.gartner.com/id=1837016>). Nous voyons également fleurir les initiatives dans des domaines qui s'écartent fortement du moule initial des pratiques agiles : déploiement d'ERP, projets de Business Intelligence, mise en place de Progiciel, gestion des infrastructures, systèmes embarqués. Appliquer l'agilité dans ces domaines nécessite une bonne maîtrise des concepts de base et des valeurs sous-jacentes. La réussite de ce type de projets en agile repose sur la capacité à prendre du recul sur la méthode pour adapter les pratiques sans dévoyer les principes ni tomber dans l'application aveugle. C'est un signe de maturité indéniable. Pour autant les organisations réellement matures sur l'agilité sont encore trop peu nombreuses. La pérennité du modèle ou sa mise à l'échelle se confronte à plusieurs difficultés que nous allons parcourir.

La séparation MOA/MOE ancrée dans nos organisations IT

Le modèle agile gomme les frontières et redistribue les cartes. La notion de MOA cristallise les griefs de nombreux agilistes mais elle n'est pas la seule à freiner l'agilité d'une organisation. La notion de MOE est également bousculée. L'agilité demande aux équipes de développement de s'intéresser à l'usage pour exercer leur devoir de conseil auprès des donneurs d'ordre et trouver le meilleur compromis entre richesse fonctionnelle et coût de développement. Il ne s'agit plus d'être de simples exécutants mais de former une cellule intrapreneuriale qui fonctionne en binôme avec le métier. Ce changement n'est pas anodin car ce n'est pas seulement une question de modification des rôles, ce sont souvent les fiches de poste qui sont directement impactées par ce changement. Autant les petites structures peuvent évoluer vite car leurs salariés sont souvent multi-compétences, autant les grandes structures ont du mal à incorporer ce glissement dans les responsabilités et demandent des réponses très claires qui ne sont pas précisées dans les méthodes agiles les plus populaires. C'est la raison pour laquelle nous voyons fleurir des offres d'emploi du type "Chef de Projet/Scrum Master" ou "Product Owner/AMOA", elles sont caractéristiques de cette dichotomie entre organisation existante inchangée et volonté de faire de l'agile.

Le mode projet systématique au détriment de la notion de produit logiciel

Le mode projet emmène avec lui une organisation et un système d'investissement et de contrôle des coûts qui se marient mal avec une approche agile, beaucoup plus proche d'un modèle de financement itératif centré sur le produit. La différence est fondamentale : le projet vise à livrer des fonctionnalités, le produit vise à satisfaire les utilisateurs. Un projet ne fait que la moitié du chemin, son succès se définit par la livraison ou non du périmètre fonctionnel imaginé, sans mesurer les résultats en termes d'usage du produit par les utilisateurs, de réponse au besoin.

"Ne vous y trompez pas, Scrum est une méthode de développement de produit et non une méthode de gestion de projet"

L'agilité demande d'embrasser les deux aspects à la fois, la livraison de fonctionnel n'est pas en soi un critère de succès, l'utilisateur est impliqué très tôt pour mesurer l'utilité des fonctionnalités et éventuellement faire évoluer radicalement le périmètre. La méthode agile la plus populaire est Scrum mais ne vous y trompez pas, Scrum est une méthode de développement de produit et non une méthode de gestion de projet.

Le terme "Projet Scrum" est presque une aporie et pourtant il est couramment utilisé dans les organisations qui pratiquent Scrum. L'agilité et son approche produit remettent en cause la structure et le suivi des coûts d'un projet.

L'acculturation nécessaire du management opérationnel

L'un des freins les plus fréquents à une généralisation de l'agilité est la collision entre les méthodes de management classiques et les implications d'un mode de développement agile. L'agilité sous-entend par exemple d'embrasser l'incertitude plutôt que de vouloir à tout prix tout clarifier de manière prématurée. La gestion de l'incertitude signifie de maîtriser des techniques de management basées sur les options : explorer les options disponibles, évaluer le prix à payer pour garder les options ouvertes, évaluer les dates limites de prise de décision, éliminer les options non viables. A contrario, un manager aujourd'hui est évalué sur sa capacité à prendre des décisions rapidement et à avoir une feuille de route très claire, l'écosystème n'est alors que moyennement favorable à l'expérimentation et aux modes de fonctionnement empiriques induit par l'agile.

D'autres aspects sont également orthogonaux entre l'approche manageriale communément répandue et les besoins de l'agilité : accepter les échecs plutôt que de les éviter à tout prix, responsabiliser les individus plutôt que de contrôler et suivre leurs tâches, mettre en place une dynamique d'amélioration continue plutôt que centraliser toutes les décisions, etc. Le rôle du manager agile est de créer l'environnement qui permettra de pérenniser l'agilité. Au lieu de cela les réflexes reviennent vite : création systématique de procédures pour éviter les erreurs, course à la productivité, phases préliminaires interminables avant de se lancer dans les développements agiles, pression sur les estimations en jour-homme.

Les principales tendances agiles pour 2013

Croissance forte :

- Multiplication des approches basées sur le flux (Kanban) en complément des approches itératives (Scrum, XP) pour organiser le processus de fabrication et livraison de logiciel.
- Utilisation de Serious Games pour améliorer la collaboration et l'efficacité du travail en équipe.
- Contractualisation agile.

- Utilisation de l'agilité sur tout type de produits et projets.

Croissance modérée :

- Product Owner et Scrum Master comme des métiers à part entière.
- Développement d'une organisation centrée sur les produits et diminution du mode projet.
- Emergence de nouveaux modèles d'investissement au sein des DSI : startup internes (LeanStartup), financement incrémental.

DevOps

Comme le mouvement Agile a rapproché donneurs d'ordre et équipes de développement autour d'une vision commune orientée « produit », le mouvement DevOps rapproche les équipes de développement (DEV), de recette (QA) et d'exploitation (OPS) autour d'une vision commune orientée « service applicatif » afin de mieux concilier réactivité et qualité de service.

“Les DEV cherchent le changement alors que les OPS veulent la stabilité”

DevOps aborde le paradoxe entre des équipes projets qui cherchent à livrer toujours plus fréquemment des nouvelles fonctionnalités d'une part et des équipes d'exploitation qui cherchent à fiabiliser les systèmes tout en maîtrisant leur coût d'autre part. Schématiquement, les DEV cherchent le changement alors que les OPS veulent la stabilité.

On peut décrire DevOps selon trois axes :

- **Aligner les OPS sur les enjeux métiers** comme l'agilité a déjà aligné le développement sur le métier.
- **Aligner les DEV sur les réalités de l'exploitation** pour rendre possible la mise en production, la disponibilité et la fiabilité des fonctionnalités métier.

- **La transformation du métier d'OPS** pour gérer des topologies chaque jour plus grosses et plus complexes avec l'adoption du Cloud Computing, de concept comme "Software Defined Data Center" ou d'outils comme Chef ou Puppet. Les nouveaux OPS sont des programmeurs ! Ils débattent à la machine à café de tests unitaires, de choix de langage de programmation (Ruby vs. DSL), de choix d'IDE, de Git vs. Subversion, ...

Notre recommandation

Nous recommandons de préférer une mise en place concrète de DevOps qui s'appuiera sur un projet commun aux équipes DEV et OPS plutôt que de tomber dans l'écueil de réunions et séminaires de "team building DevOps" aux résultats souvent décevants.

Si votre organisation permet un projet ambitieux, nous vous recommandons la mise en place d'un processus de **Continuous Delivery** qui, dans la mouvance Lean, vise à déployer les fonctionnalités en production au plus vite et à maximiser les feedbacks. Nous consacrons dans ces Tech Trends un chapitre à Continuous Delivery.

Si vous préférez des "quick wins", nous vous recommandons des projets de briques de services d'infrastructure mise en accès self-service pour les équipes projet dans l'esprit des approches Cloud :

- Service centralisé de logs techniques et applicatives ;
- Service de monitoring d'indicateurs techniques et de KPI métiers ;
- Automatisation des déploiements et des installations des applications.

Une fois ces projets communs réalisés, les volets organisationnels de DevOps comme l'intégration d'OPS dans les projets pourront être abordés.

Focus Continuous Delivery ou Livraison Continue

Le **Continuous Delivery** est un nouveau mode d'organisation agile dans lequel l'équipe projet maintient en permanence le logiciel prêt à être relâché, prêt à être déployé en production. Ce principe diffère du mode agile classique dans lequel le logiciel n'est prêt à être relâché que périodiquement, à la fin de certains sprints. Les bénéfices sont multiples :

- Fiabiliser les processus de création des releases et de déploiement ;

- Donner au Product Owner le pouvoir de mettre en production quand il le souhaite ;
- Mieux intégrer les équipes de recette (la QA) et surtout d'exploitation (les OPS) au quotidien du projet ;
- Mieux sensibiliser l'équipe projet sur la finalité de son travail : déployer en production de nouvelles fonctionnalités.

La mise en oeuvre du *Continuous Delivery* comporte des étapes essentielles :

- L'automatisation du déploiement des applications.
- La mise en place d'un certain nombre de best practices comme : faire transiter un même package applicatif (indépendant de l'environnement) sur les différents environnements de Dev, QA et Production et sur lequel on va appliquer une configuration différente en fonction de l'environnement cible ; produire toujours des packages applicatifs complets (ne pas faire de packages différentiels). Aujourd'hui il est aisé de produire des packages complets (cela réduit les risques d'erreurs humaines ou d'incompréhension) et des outils comme Deployit se chargent eux-mêmes de calculer les deltas entre deux versions d'un même package pour n'installer que ce qui a été modifié ; externaliser de la configuration des packages et le stockage des paramètres de configuration dans un référentiel partagé au sein de l'organisation (concepts de dictionnaires dans Deployit par exemple).
- L'uniformisation des techniques d'installation sur les environnements de Dev, QA et Production : les mêmes procédures de déploiement et les mêmes outils sont utilisés partout et donc testés en permanence.
- L'autonomie des équipes de QA pour déployer sur les environnements de validation : des GUI et des API leurs permettent de déployer de nouvelles versions sans l'intervention d'autres équipes.
- La mise en place de GUI et d'API accessibles en self-service sur les bases de données pour permettre aux équipes de QA de recharger des jeux de données à la demande ;
- L'automatisation d'une partie des User Acceptance Tests et des tests de performances.

- L'introduction dans les développements de Feature Toggles pour permettre la mise en production de versions dont certaines fonctionnalités inachevées sont maintenues cachées par configuration.
- Dans les organisations très rodées, l'automatisation de la création d'environnements complets avec applications et bases de données.

Des outils pour y parvenir

L'outillage est très important pour mettre en place le Continuous Delivery. Voici des suggestions de solutions qui ont fait leurs preuves:

- Intégration continue : Jenkins ;
- Automatisation de l'installation des applications : Deployit ;
- Autonomisation de la création d'environnements complets : Puppet, Chef ;
- Automatisation des User Acceptance Tests : Selenium, Fitnesse ;
- Automatisation de tests de performances : JMeter, Gatling ;
- Feature toggle : JMX, fichiers properties.

Un changement de mentalités

Au delà des outils, le Continuous Delivery demande un changement des rôles et responsabilités entre les équipes de DEV, QA et OPS.

Roder les déploiements à travers des outils, procédures et référentiels partagés entre équipes Les équipes de développement doivent comprendre que, via leurs usines logicielles (outils de buis et d'intégration continue) elles alimentent le reste des équipes en packages applicatifs qui vont pouvoir traverser un pipeline d'environnements (Dev, QA, Pré-prod, Prod, Formation...). Très tôt dans le process, elles vont pouvoir commencer à tester et roder des procédures de déploiement sur leurs environnements de tests. Elles peuvent commencer à partager un référentiel de procédures de déploiement et de paramètres de configuration avec les autres équipes.

On va également permettre aux équipes de QA d'être beaucoup plus autonomes dans l'instantiation de leurs environnements et l'installation de leurs applications en mode self service.

Le passage en production ne doit finalement devenir qu'une énième installation de l'application même si elle porte un risque plus grand et des contraintes plus fortes.

Passer de briques d'infrastructure à des services de plateforme utilisables en self-service Le plus grand changement concerne les équipes d'infrastructure (les OPS). L'automatisation des déploiements nécessite de repenser le traditionnel catalogue de services d'infrastructure en un outil de plateforme utilisable en self-service par les équipes projet. C'est un effort de productisation de l'infrastructure.

"Repenser le traditionnel catalogue de services d'infrastructure en un service de plateforme utilisable en self-service par les équipes projet"

Par exemple, un service de plateforme java (e.g. Tomcat As a Service) doit exposer à ses utilisateurs des API et GUI qui masquent l'intégration de serveurs d'applications java, d'un load balancer, d'un DNS, sans compter les briques techniques de concentration de logs, de monitoring et de traçabilité. Les plateformes Cloud comme Amazon AWS et CloudBees sont des exemples d'approches Platform as a Service.

Accepter l'inachevé Un changement de mentalités plus inattendu est la culture des fonctionnalités inachevées désactivées en production. Pour qu'un logiciel soit en permanence releasable et déployable en production, il faut régulièrement qu'il intègre des fonctionnalités incomplètes ; des fonctionnalités que l'on maintient cachées tant qu'elles ne sont pas finies pour laisser au Product Owner la liberté de mettre en production s'il le souhaite.

En bref

Nous recommandons de mettre en place le Continuous Delivery avec une approche *bottom-up* en prenant la plateforme d'intégration continue comme point de départ :

- Ajout au build du déploiement automatique de l'application sur un environnement de test ;
- Mise à disposition des équipes de DEV et de QA en accès self-service de GUI et d'API de déploiement sur les environnements de validation ;
- Intégration dans le build de quelques User Acceptance Tests avec Selenium ou équivalent ;

- Introduction de Feature Toggles dans les développements pour masquer en production les fonctionnalités inachevées.

Une fois cette dynamique créée et les équipes autonomes sur leurs environnements, nous vous recommandons d'augmenter la fréquence des déploiements sur les différents environnements (jusqu'à la mise en production) afin de gagner en confiance et de donner au projet le temps de prendre du recul sur son mode de fonctionnement.

Par la suite, les chantiers d'automatisation d'un plus grand nombre de tests de bout-en-bout et l'automatisation de la création d'environnements complets en mode Platform as a Service pourront être étudiés.

Software Craftsmanship

par Jean-Laurent de Morlhon

Le **Software Craftsmanship** est un mouvement récent qui vise à rappeler que l'agilité concerne tout autant les pratiques techniques que les méthodologies. L'agilité arrivant en phase de maturité en 2013, beaucoup de projets embrassent la méthode sans se soucier un instant des pratiques techniques. Scrum et Kanban sont aujourd'hui les méthodes les plus utilisées. Elles ne précisent néanmoins absolument rien quant aux pratiques techniques à mettre en oeuvre. Le mouvement Software Craftsmanship tire la sonnette d'alarme : quantité de projets livrent de manière incrémentale une qualité décevante.

Le mouvement est d'autant plus populaire qu'il replace le développeur au centre du dispositif projet et non plus comme un simple exécutant, dactylographiant du code sans sourciller ! Le développeur doit être force de proposition, respecté et s'impliquer autant dans le métier, les tests et la gestion du projet que dans la technique. Il est au cœur du dispositif projet et capable, mieux que quiconque, d'évaluer les charges et les risques au niveau tactique.

“Le mouvement est d'autant plus populaire qu'il replace le développeur au centre du dispositif projet et non plus comme un simple exécutant”

Le *mentoring* est une des valeurs du Software Craftsmanship. Chaque développeur est incité à être le *mentor* ou le *mentee* d'autres développeurs du projet. Le rôle du *mentor* consiste à accompagner son *mentee* afin que ce dernier devienne un professionnel du développement logiciel sur le plan technique, mais aussi sur le plan personnel.

Le mouvement rappelle également toute une série de techniques de modélisation, code ou test, parfois issuent de la méthode *Extreme Programming*, mais pas uniquement : design simple, Test Driven Development, tests d'acceptance, pratiquer plusieurs langages, programmation fonctionnelle, etc. On peut notamment apprendre ces techniques lors d'entraînements spécifiques : Code Katas, Coding Dojo, Code Retreat.

Dans le cadre d'un projet informatique classique, il peut s'avérer compliqué de mettre en place ce type d'apprentissage. Certaines entreprises ont malgré tout compris ce besoin de réelles formations continues et aménagent au sein des projets, voire des départements, des sessions d'expérimentation et de partage. Usuellement sous la forme d'un Coding Dojo, ces sessions se déroulent en partie sur le temps du repas, en partie sur le début d'après-midi de manière hebdomadaire.

Coaching agile

par Nicolas Jozwiak

Le constat

Les méthodes agiles telles que Scrum et Kanban sont adoptées par un grand nombre de projets et d'organisations. Malheureusement, nous observons sur le terrain encore de nombreux problèmes dans la mise en place et la pérennité de ces méthodes :

- Manque d'adaptation des méthodes au contexte du projet. Scrum et Kanban proposent des outils à mettre en place mais certains nécessitent d'être modifiés ou même mixés afin de répondre aux particularités de contextes particuliers. Scrum-Ban est un bon exemple d'adaptation.
- Mise en place ou adoption difficile par les équipes.
- ScrumMaster dépassé et / ou n'arrivant plus à gérer son équipe.
- Équipes n'arrivant plus à faire émerger les problèmes inhérents et qui stagnent.
- Manque de compétences pour instaurer l'agilité.
- Mauvaises pratiques : les méthodes agiles ne se résument pas au Stand Up.

Beaucoup d'équipes se retrouvent dans une ou plusieurs de ces situations sans toujours en avoir conscience. Arrivées à ce stade, il est nécessaire pour elles de prendre du recul

mais cela n'est pas toujours évident, et encore moins concernant des équipes à faible expérience dans les méthodes agiles. Comment faire alors ?

L'intervention d'une personne externe à l'équipe est une solution : cette dernière est neutre et n'a aucun rapport avec les membres de l'équipe. Elle peut intervenir plus « librement ». Encore faut-il trouver une personne ayant de solides connaissances et expériences dans l'agilité et la conduite de projets.

"Attention, les méthodes agiles ne se résument pas au Stand Up"

C'est dans cette optique que le métier de Coach Agile est né. Ce dernier possède, entre autres, des qualités de facilitateur et d'agent du changement lui permettant d'intervenir sur plusieurs axes :

- Gérer les conflits au sein des équipes et des organisations.
- (Re)dynamiser les équipes : mettre en avant les obstacles et les résoudre.
- Encadrer le ScrumMaster.
- Aider le Product Owner : définition de la vision produit, rédaction User Stories, mise en pratique d'innovation games, etc.
- Instaurer une culture agile à tous les niveaux.
- Gérer le chiffrage et le budget en mode agile.

Au travers de ses expériences et de son regard extérieur, ce dernier a la légitimité pour intervenir sur les projets et organisations afin d'aider à résoudre des situations difficiles et conflictuelles.

Si nous allons plus loin, un Coach Agile s'inscrit dans la notion d'Enterprise Transition Community (ETC) de Mike Cohn : ce sont des communautés agiles identifiées au sein des entreprises qui se chargent d'aider les équipes à adhérer à la culture agile. Cela passe par la création d'environnements agiles et des aides pour abaisser les barrières.

La tendance

Nous constatons que ce métier a le vent en poupe : de plus en plus d'organisations sollicitent des Coach pour les aider à (re)mettre d'aplomb les projets et les équipes.

D'ailleurs, l'évolution est si rapide que nous voyons fleurir de plus en plus de centres agiles. Au même titre que nous avons des cellules d'architectures, nous retrouvons des structures internes spécialisées dans ce métier, se rapprochant encore plus des ETC.

Nous émettons néanmoins une mise en garde : à l'instar des ScrumMaster, encore trop de Coach Agile ne sont pas assez bien formés. Ce métier nécessite des compétences particulières qui s'acquièrent avec de l'expérience.

Test Driven Development et Refactoring

par Nicolas Demengel

L'intérêt des tests automatisés est désormais bien intégré sur la plupart des projets et on constate en général l'écriture de tests unitaires et de tests d'intégration par les développeurs, voire de tests fonctionnels en collaboration avec les acteurs du métier. Le but est de mettre en place un filet de sécurité pour tous les aspects externes ou internes du logiciel : fonctionnalités métier et performance bien sûr, mais aussi ergonomie, conception, maintenabilité...

C'est ainsi que nombre d'équipes sont à présent passées ou sont prêtes à passer à l'étape suivante : le développement piloté par les tests. Les tests - qu'ils soient de haut niveau (fonctionnels) ou de plus bas niveau (unitaires) - sont alors écrits avant de développer les aspects du système concernés.

Les avantages sont nombreux :

- Les tests sont désormais une spécification exécutable, documentant précisément les capacités et le comportement du système.
- Ils montrent avec exactitude l'avancée du développement : toute fonctionnalité est représentée par des tests ; un test qui ne passe pas traduit une fonctionnalité non terminée.
- Ils font office de guide pour les développeurs, leur permettant de se focaliser sur un besoin à la fois. Les tests définissent en effet une succession de petits objectifs précis à remplir.
- Enfin, ils forcent très tôt la réflexion sur l'utilisation réelle du système, faisant ressortir les zones d'ombre et les imprécisions, et aidant à la conception de l'interface du système.

“La règle du Boy Scout : un développeur laissera toujours la zone du système sur laquelle il travaille dans un meilleur état que celui dans lequel il l'a trouvée”

Les notions de vieillissement du code et de dette technique ont également fait leur chemin dans les esprits. Il est généralement compris que la qualité interne d'un système se dégrade naturellement au fil des nouveaux développements, alors que le système gagne en complexité. Cela est d'autant plus vrai lorsque l'on fait le choix ponctuel de délivrer certaines fonctionnalités très rapidement, au détriment d'une réflexion poussée sur leur intégration au sein de l'existant. Cette dégradation du système va le rendre plus difficile à comprendre et à modifier, et va donc augmenter le coût et le risque des développements suivants.

Il est alors essentiel pour les développeurs de s'adonner au refactoring. Il s'agit de modifier le système dans le but unique d'améliorer sa qualité, en ne changeant rien à son comportement. Pour que ces modifications ne provoquent pas de régressions, il est donc nécessaire que le système soit bien testé.

De petits refactorings réguliers seront moins coûteux et moins risqués que de gros refactorings ponctuels. On cherchera alors à refactorer le système de manière continue, en appliquant la règle du Boy Scout : un développeur laissera toujours la zone du système sur laquelle il travaille dans un meilleur état que celui dans lequel il l'a trouvée.

Malgré cela, des parties du système peuvent se dégrader insidieusement, ou on peut accuser volontairement une dette technique. Il convient alors de prioriser les refactorings nécessaires pour qu'ils soient réalisés au plus tôt : autrement, la dette ne peut que s'accroître avec le temps !

Agile Games

par Nathaniel Richand

L'utilisation du jeu en entreprise n'est pas un phénomène nouveau. Celui-ci est utilisé depuis de nombreuses années pour créer et fédérer les équipes (les fameux "team building") ou bien pour permettre l'acquisition de nouvelles compétences en créant des contextes sécurisés reproduisant la réalité. La tendance plus récente, actuellement en expansion, est l'utilisation du jeu pour de nouvelles populations et de nouveaux usages. Le jeu en entreprise a longtemps souffert de la croyance : "jouer n'est pas sérieux". Cette croyance est néanmoins bousculée petit à petit et ainsi, les équipes de développement, les équipes produits et le management peuvent innover, définir une vision et une stratégie à travers le jeu. Ce dernier facilite la prise de décision, aide à la planification et donne un cadre à la résolution de problème.

“Nous assistons aujourd’hui à un développement très rapide du jeu en entreprise”

L'utilisation du jeu en entreprise peut revêtir plusieurs formes :

- **Serious games & agile games**

Ces jeux correspondent à l'introduction de la réalité dans un jeu. Les serious games étaient, dans un premier temps, utilisé essentiellement dans des jeux vidéos immersifs (le très connu America's army faisant la promotion de l'US army) mais se sont massivement développés sur d'autres formats tels les jeux de plateau. Les agilistes ont ensuite créé de nombreux jeux comme le XP Game, Scrum lego game, GetKanban ...

- **Innovation games**

Les innovations games© sont un ensemble de jeux développés par Luke Hohmann au travers desquels les clients sont amenés à participer à différents jeux dirigés dans le but de générer du feedback sur un produit ou un service.

- **Gamification**

Elle correspond à l'utilisation des mécanismes du jeu dans la réalité. On peut citer comme exemple les badges de foursquare ou dans LinkedIn, la barre de progression représentant le dégré de complétude d'un profil LinkedIn.

Dans les trois cas l'intérêt de jouer est multiple :

- Impliquer les participants grâce à l'aspect ludique du jeu.
- Eviter la levée de bouclier qui peut s'opérer lorsque l'on reste dans un cadre rigide (grâce aux métaphores que fournit le jeu).
- Être amené à penser différemment et produire ainsi des solutions nouvelles.
- Favoriser le travail collectif.

Nous assistons aujourd'hui à un développement très rapide du jeu en entreprise, au point de voir naître des conférences qui lui sont dédiées partout dans le monde ainsi qu'en France. Les deux livres “Innovation games” et “Gamestorming” ont participé à cette diffusion en proposant des référentiels de jeux conséquents. De la petite startup

à la grosse banque d'investissement internationale, au travers de formats très courts de quelques minutes à des formats de plusieurs jours, les possibilités de jeu en entreprise sont très variées et en plein essor.

Les principaux freins aujourd'hui demeurent l'image du "jeu", peu sérieux et infantilisant. Cette dernière est encore résistante dans certains milieux. Enfin, il reste difficile de trouver les facilitateurs expérimentés capables de créer et maintenir les bonnes conditions de jeu.

Expérience utilisateur

par Yannick Grenzinger

Il est de plus en plus difficile d'ignorer l'importance de l'**expérience utilisateur** dans l'évolution du Web, des sites et des applications. Le design du produit joue un rôle important dans le succès d'une startup et des entreprises. Apple a prouvé qu'une expérience utilisateur réussie peut amener une entreprise à dominer son marché et, même si Apple est le cas emblématique, le succès par le design de nombreuses entreprises comme Dropbox ou Netflix prouve que ce n'est pas un cas isolé. Il faut aussi noter que des **news technologiques et business majeures**³⁴ de 2012 étaient aussi directement liées à des questions de design: le duel juridique entre Apple et Samsung, le rachat d'Instagram par Facebook ou encore le fiasco des cartes d'iOS 6.

De manière encore plus pragmatique, le mouvement **Lean Startup** mené par Eric Ries qui reprend en grande partie les méthodes des designers (prototypage, tests avec l'utilisateur) met en avant la mesurabilité et permet de prouver le **ROI d'une bonne expérience utilisateur**³⁵.

"Le succès par le design de nombreuses entreprises comme Dropbox ou Netflix prouve que Apple n'est pas un cas isolé"

Même dans le cadre plus classique d'une DSI, travailler sur l'UX (User eXperience) est un excellent moyen **d'éviter de gâcher des ressources**³⁶ en permettant de mieux prioriser les fonctionnalités du produit et de mieux construire vos cas d'utilisation. Bonne nouvelle supplémentaire, les principes et les méthodes de l'UX peuvent s'incorporer aussi bien aux **méthodes Agiles**³⁷ qu'aux phases amonts de méthodes plus traditionnelles.

³⁴ http://www.uie.com/articles/ux_lessons

³⁵ <http://www.uxmatters.com/mt/archives/2012/07/how-to-calculate-the-roi-of-ux-using-metrics.php>

³⁶ <http://johnnyholland.org/2012/08/an-example-how-good-ux-practices-can-keep-you-from-wasting-resources/>

³⁷ <http://www.uxmatters.com/mt/archives/2013/01/when-agile-and-user-experience-click.php>

En conclusion, il est clairement temps d'investir dans l'[UX de vos produits](#)³⁸ pour mieux comprendre vos utilisateurs, mieux comprendre le besoin et ainsi améliorer la valeur ajoutée de vos produits. Les cycles business raccourcissent et il faut à la fois l'excellence technique et la pertinence du design tout en minimisant l'effort. Penser à l'UX en amont et tout au long du projet permet de répondre à ces contraintes.

³⁸ <http://www.infoq.com/presentations/Co-Making-Great-Products>



NOS RECOMMANDATIONS

Agilité & Craftsmanship

La migration vers les méthodes agiles se démocratise.

Attention néanmoins à ne pas perdre de vue les valeurs fondamentales, par exemple, en vous faisant coacher !

Nous vous recommandons, si vous ne l'avez pas déjà fait, de commencer à utiliser l'approche Kanban pour piloter l'ensemble de la chaîne de fabrication logicielle, depuis la demande métier jusqu'à la mise en production. Cette approche permet de sortir des indicateurs de pilotage qui vont au-delà de l'équipe agile et augmentent la prédictibilité. Il s'agit ici de la réponse agile à la planification : temps de cycle, volume du travail en cours, efficience globale.

Nous vous recommandons également de mettre l'accent sur la professionalisation des métiers agiles. Cet effort ciblera en premier lieu les développeurs, avec des éléments issus du Software Craftsmanship et les Product Owners, qui ne sont pas de simples AMOA mais de véritables intrapreneurs.



L'agilité n'est pas qu'une question de rôles, c'est une évolution dans les métiers du logiciel et cela nécessite un effort d'apprentissage au long court. Il conviendra peut-être de restaurer des parcours professionnels pour développeurs qui ne soient pas une évolution vers du management mais plutôt vers de l'expertise et du tutorat. Les rôles de Scrum Master et Managers doivent également solliciter votre attention.

Pour obtenir des gains durables, vous devrez intégrer l'agilité dans l'ADN de votre organisation. Vous pouvez agir au niveau de la structure en nommant des champions agiles permanents ou en mettant en place une cellule agile. Vous pouvez également favoriser les communautés de pratiques ou les sessions de partage régulières. En revanche, une part de succès dans votre changement d'ADN sera liée à la restauration d'une vision produit en complément de l'approche projet : stabilisez vos équipes et éloignez-vous des "ressources planning" ou "staffing plan" qui sont consommateurs d'énergie et apportent peu de valeur à l'utilisateur.

Enfin, nous vous conseillons de vous intéresser au Jeux Agiles, d'une part pour doper le moral des équipes et soutenir leur apprentissage continu, d'autre part, pour rechercher plus d'efficacité dans la conduite de réunion, véritable piège de l'agilité.



Xebia

"Xebia est un cabinet agile de conseil, d'expertise technique et de réalisation exclusivement dédié aux technologies Java/J2EE. Créé en 2001, Xebia compte 300 collaborateurs intervenant en France, aux Pays Bas et en Inde. Une des clés de notre réussite consiste à rester constamment à la pointe des nouvelles technologies. Les passions actuelles des Xebians sont passées au radar dans ce Techtrends :

- BigData ;
- Software Craftsmanship ;
- Cloud ;
- Agile ;
- DevOps ;
- Web ;
- Mobility ;
- Java & friends.

La mission de Xebia consiste à occuper auprès de ses clients le rôle de bras droit et de tiers de confiance afin de les aider à concevoir, développer et gérer des infrastructures, des applications et des architectures basées sur Java en appliquant les meilleures pratiques du Software Craftsmanship. Pour ce faire, Xebia réunit en son sein des experts passionnés du développement logiciel ainsi que les coaches agiles les plus reconnus. Les quatre valeurs fondatrices, clés du succès de Xebia, illustrent sa mission :

- Partage de connaissance ;
- Aucun compromis sur la qualité ;
- Intimité client ;
- People first.

Les auteurs



Yannick
Grenzinger



Jean-Laurent
de Morthon



Philippe
Antoine



Audrey
Pedro



Nathaniel
Richard



Benoit
Lemoine



Jean
Helou



Christophe
Heubès



Séven
Lemesle



Pablo
Lopez



Alexis
Kinsella



Frédéric
Dubois



Gilles
Mantel



Luc
Legradeur



Mathieu
Breton



Cyrille
Leclerc



Guillaume
Balaine



Nicolas
Jozwiak



Yves
Amsellem



Aurélien
Maury



Olivier
Michallat



Guillaume
Arnaud



François
Sarradin



Bertrand
Dechoux

XEBIA

EST UNE ENTREPRISE AGILE QUI DÉLIVRE DES LOGICIELS

sur mesure
À SES CLIENTS

NOS VALEURS FONDATRICES,
CLÉ DE NOTRE SUCCÈS

People First

Partage de connaissance

Intimité client

Qualité sans compromis



Pour nous joindre : info@xebia.fr

Xebia