AXOPEN

La qualité de code

Vision AXOPEN

2024

La qualité de code en 2024

Table des matières

AXOPEN
235 cours Lafayette – 69006 LYON
+33 (0)4 82 53 26 44 - contact@axopen.com
www.axopen.com

SAS au capital de 127 200 euros - SIREN 494 526 908 - RCS LYON
TVA Intra : FR74494526908 - APE: 6201Z

1. Édito - La qualité de code : pourquoi faire ?

A l'heure où l'IA, avec un soutien marketing considérable, nous promet du code écrit dans les règles de l'art par des robots infaillibles, formés sur l'ensemble du code produit par l'humanité, plusieurs questions se posent :

- Pourquoi devrions-nous encore nous soucier de la qualité du code de nos applications ?
- Cette prétendue infaillibilité des machines compensera-t-elle enfin les bugs incessants qui émergent dans nos systèmes ?
- Allons-nous pouvoir nous passer des développeurs, bien qu'ils soient de plus en plus nombreux, mais apparemment toujours perfectibles ?

C'est un peu le rêve du moment : un système d'information écrit automatiquement par des machines, se rapprochant de plus en plus des enjeux métiers tout en réduisant les coûts. Le Saint Graal, dirait-on !

Ce rêve technologique tant convoité est-il réellement sur le point de se réaliser ? Et si oui, est-ce pour bientôt ?

2. Le problème du code

Dans le monde numérique d'aujourd'hui, le code source est le pilier fondamental de la plupart des technologies que nous utilisons. Du site web le plus simple à l'intelligence artificielle la plus complexe, tout repose sur des lignes de code écrites par des humains… enfin, pour l'instant ! Avant de rentrer dans le vif du sujet, arrêtons-nous un instant pour définir de manière précise ce qui se cache derrière le terme code », « code source » ou « code informatique ».

En termes simples, **le code informatique est un ensemble d'instructions que l'on donne à un ordinateur pour qu'il exécute des tâches spécifiques. Ces instructions, bien que semblant simples en surface, cachent une complexité et une subtilité immenses.**

La principale difficulté avec le code réside dans sa perfection intrinsèque requise. Un seul caractère mal placé ou une seule erreur de logique peuvent entraîner des bugs, des failles de sécurité ou des dysfonctionnements. Ces problèmes peuvent avoir des conséquences allant de légères nuisances à des catastrophes majeures, surtout dans des domaines critiques comme la finance, la santé ou la sécurité nationale.

La montée en puissance de l'abstraction dans nos pratiques de code augmente à la fois la productivité (une ligne de code produit des effets plus grands en 2024 que pendant les années 90) mais aussi, en des proportions similaires, les bugs !

On peut donc dire la chose suivante : plus il existe de code, plus il existe de bugs. C'est un peu trivial, mais ça reflète réellement une vérité.

Les clients font souvent la remarque suivante : comment se fait-il qu'avec autant d'années d'expérience, il existe toujours autant de bugs dans vos applicatifs ? Un élément de réponse à ce stade : l'évolution de l'informatique, jusqu'à présent, s'est concentré sur l'amélioration de la productivité du développeur (moins de lignes de code pour plus d'effets) et non pas sur la sécurité, au sens bugfree du code.

De plus, **le code n'est pas statique.** Il évolue constamment, nécessitant des mises à jour et des améliorations pour s'adapter à de nouvelles exigences ou intégrer de nouvelles fonctionnalités. Cette dynamique perpétuelle du développement logiciel fait que **maintenir la qualité et la fiabilité du code est un défi constant pour les développeurs.**

En outre, à mesure que la technologie progresse, le code devient de plus en plus complexe. Les systèmes que nous développons aujourd'hui sont bien plus sophistiqués que ceux d'il y a dix ou vingt ans, impliquant une plus grande probabilité d'erreurs et un besoin accru de compétences spécialisées pour gérer cette complexité.

On le comprend assez bien, et on le constate tous les jours, **le code est partout et est à l'origine du problème (presque inhérent de l'informatique).**

Alors, comment appréhender ce problème ? Voici quelques éléments de réponse !

2.1. La notion de clean code

La notion de « code propre » est loin d'être une notion qui coule de source ! Voici la définition qui parait la plus juste à nos yeux (écrite par les créateurs du logiciel Sonar) :

> *"The foundation of the Clean Code taxonomy is code that is clean and code that has the following properties: consistent, intentional, adaptable, and responsible."*

Sur cette base, on part donc du postulat que le code doit avoir différentes propriétés : consistent, intentional, adaptable et responsible. Concrètement, à quoi correspondent-elles ?

2.1.1. Consistent

La cohérence du code fait référence à son formalisme. **L'objectif principal de cette propriété est de retrouver une unique façon de coder et de présenter le code.** Ce formalisme peut bien évidemment dépendre des spécificités des projets de développement et avoir des variations, cependant, il est grandement recommandé de se baser sur un référentiel connu. Il en existe généralement un par langage, et concrètement, cela passe principalement par l'utilisation de **linters** sur vos projets.

La cohérence du code permet aux différents développeurs de lire et de comprendre plus rapidement les différents morceaux de code. **Cela permet notamment de faciliter l'intégration de nouveaux développeurs sur vos projets et d'améliorer leur capacité à coder rapidement, une fois le formalisme acquis.**

2.1.2. Intentional

Dans le cadre de notre définition, un code "intentionnel" signifie un code lisible et le plus compréhensible possible. À travers cette propriété, il est important de prendre conscience d'une chose : **un code écrit par un développeur, a de grandes chances d'être un jour lu et utilisé par d'autres développeurs**.

Aussi, pour faciliter ce travail collectif, on peut passer par plusieurs actions comme par exemple : travailler à un bon nommage des variables et des fonctions, utiliser des qualifiers (public, private et protected), spécifier si les attributs sont final ou non, ou encore, découper les différentes sections du code en fonctions.

Pour monter encore un cran au-dessus dans la compréhension et la lisibilité de son code, on peut utiliser des commentaires. On distingue trois grandes utilisations des commentaires :

- **La description du fonctionnement du code, le « quoi »** : cela permet d'expliquer comment ce dernier fonctionne. C'est dans cette catégorie que se retrouve le célèbre "Get Name" sur le Getter de l'attribut name.

- **L'explication de la manière dont on peut utiliser le code, le « comment » :** cela permet de laisser un petit message pour les développeurs avec des indications sur la façon dont le code doit être utilisé dans le projet.
- **La justification de l'existence d'un morceau de code, le « pourquoi »** : cela permet de démontrer la réflexion qui a été faite lors de la rédaction du code et ainsi, de justifier les choix de développement à travers un commentaire. Moins utilisé, ce dernier type de commentaire permet pourtant d'améliorer nettement la compréhension du code lors de sa lecture et plus largement, le contexte de sa création.

2.1.3. Adaptable

Lorsque l'on parle de code « adaptable », on parle d'un code qui permet de répondre à différentes problématiques. **L'objectif n'est plus seulement de penser son code pour sa problématique, mais de le penser plus largement, pour qu'il puisse facilement être réutilisé par les autres développeurs.**

Cela implique que l'on devienne plus responsable de son code ! Lorsque l'on écrit une méthode, il ne faut pas prendre les choses pour acquises ; la méthode doit être auto-suffisante, et ne pas juste fonctionner dans certains cas. Il est important de faire quelques vérifications au début de la méthode pour s'assurer de la cohérence des paramètres avant d'effectuer son action. Lorsque l'on parle de responsabilité de son code, les tests sont inévitables ! Comme on devient plus responsable de sa méthode, il est important de bien la tester pour éviter toute régression associée à cette méthode.

2.1.4. Responsible

Cette dernière notion du Clean Code a pour objectif de responsabiliser le développeur sur son code, ainsi que son utilisation. Derrière cette notion de « code responsable », on retrouve principalement des obligations « éthiques » en lien avec les données comme la gestion de leur stockage (principalement pour les données sensibles : données personnelles et mots de passe). On retrouve également un aspect qui est rarement au centre des discussions des développeurs : le respect des licences des différents codes que l'on peut retrouver sur Internet. Il est important d'utiliser uniquement des codes libres de droits dans son projet et de respecter les conditions des différents types de licences.

Le nommage est également un point important, que ce soit dans l'UI ou dans les variables du code. Les noms utilisés ne doivent en aucun cas discriminer ou manquer de respect à quiconque.

Pour résumer, le Clean Code a pour objectif de rendre le code plus lisible et collaboratif. **L'objectif du code n'est plus uniquement de répondre à une problématique métier, mais également d'être le plus générique possible et compréhensible pour d'autres développeurs.** Si on extrapole, on pourrait même dire que l'objectif est de rendre « l'expérience développeur » la plus agréable possible.

3. Comment approcher la notion de qualité de code ?

3.1. Quality gate

En développement informatique, un **« quality gate » (ou portail de qualité) est un point de contrôle dans le processus de développement logiciel qui permet de s'assurer que le code répond à un ensemble prédéfini de critères de qualité.**

Les quality gates évaluent le code sur la base de critères spécifiques, tels que la couverture de test, les conventions de codage, la complexité du code, les vulnérabilités de sécurité, et d'autres métriques de qualité du code.

Ils fournissent un retour d'information continu aux développeurs sur la qualité de leur code, ce qui favorise l'apprentissage et l'amélioration des pratiques de codage.

Il faut considérer le code source comme un patrimoine immobilier qui a une histoire. Il ne peut être parfait, et dès qu'il est écrit, il est nécessaire de le corriger, de le maintenir et de l'améliorer.

Souvent, quand on arrive sur un nouveau contexte, on s'aperçoit que la qualité de code et mauvaise et que, même si des outils sont mis en place pour y remédier, la qualité de code ne s'améliore pas. A l'inverse, on constate parfois des contextes dans lesquels la qualité de code est bonne, mais rien n'est produit par les équipes de développements. **Il faut savoir jouer avec la qualité de code, pour continuer à être en mesure de délivrer des fonctionnalités tout en assurant une qualité de code raisonnable.** Pour trouver la bonne stratégie à adopter, il convient de travailler avec son équipe Génie Logiciel, ou les responsables de la qualité pour trouver le juste milieu et définir une quality gate acceptable.

Petite astuce : si vous démarrez avec une mauvaise qualité de code, au lieu d'essayer de tout corriger d'un coup (ce qui épuisera à coup sûr vos équipes), prenez comme principe de passer uniquement le nouveau code produit à la quality gate. Ainsi, vous pouvez forcer les développeurs à passer la quality gate à chaque nouvelle MR. Au début, les effets sont assez faibles, mais très rapidement, la qualité de code globale va s'améliorer. Et bonus, avec cette stratégie, vous embarquerez bien plus facilement vos équipes de développeurs !

3.2. Comment définir une bonne quality gate ?

Pour définir une bonne quality gate, il faut d'abord réfléchir à identifier quels sont les problèmes de votre code base. Dire « j'ai des bugs » et en conclure que la qualité du code est mauvaise, ce n'est pas tout à fait exact et cela ne permet pas d'agir correctement. Pourquoi ? Parce qu'il n'y a pas nécessairement un lien évident entre un bug et la qualité de code. La plupart des bugs des applications sont des bugs dit « métier » qui correspondent plus à des cas non pensés par les développeurs ou les équipes de recette.

Certaines applications souffrent de régression, ou d'autres encore, de problèmes qui ne se produisent qu'en production, de problèmes de performances, ou de problèmes que l'on qualifie faussement d'aléatoire. D'autres applications ont plutôt des difficultés à livrer de nouvelles fonctionnalités, à limiter les risques de sécurité, etc. La liste des problèmes est presque aussi longue que le nombre d'applications ! Il convient donc de réfléchir et d'identifier le problème pour y apporter une saine solution. Le tableau si dessous montre qu'il faut bien catégoriser ses problèmes avant d'y apporter des solutions.

	Type de problème	Symptômes	Solutions
Problème de régression			
Problème de vitesse de livraison (célérité)			
Problème de performance			
Problème d'instabilité			
Problème ne se produisant qu'en production			
Problème de sécurité			

3.3. Sonar et autres outils

Quand on parle de qualité de code, on a tout de suite tendance à essayer de s'outiller avec des logiciels permettant d'automatiser le contrôle. **C'est évidemment une question de gain de temps, mais ces outils ne permettent pas de garantir la meilleure qualité pour votre projet.** Ils peuvent être une aide qui accompagne un mouvement plus vaste de correction des bugs, de respect des guidelines des langages et frameworks et plus simplement, pour avoir un code de qualité et optimisé pour le besoin métier qu'il doit remplir.

Parmi ces outils, on retrouve d'abord **SonarQube qui est un outil d'analyse statique de code, conçu pour soutenir le maintien de la qualité du code.**

Ses avantages sont multiples : il excelle dans la détection des bugs et des vulnérabilités, ce qui contribue à renforcer la robustesse des applications. Son intégration fluide dans les pipelines CI/CD en fait un choix privilégié pour les processus de développement modernes, offrant ainsi une approche cohérente de l'assurance qualité tout au long du cycle de vie du logiciel. De plus, **SonarQube fournit des métriques précieuses sur la qualité du code, permettant aux équipes de suivre leur progression et de prendre des décisions éclairées pour améliorer leurs pratiques de développement.** Cependant, malgré ses nombreux avantages, cet outil présente quelques inconvénients à prendre en compte. Notamment, il ne couvre pas les tests de pénétration, ce qui limite sa capacité à identifier les vulnérabilités de sécurité plus profondes. De plus, certaines limitations dans l'analyse peuvent parfois restreindre sa pertinence dans des contextes spécifiques, nécessitant ainsi une évaluation minutieuse de son utilisation en fonction des besoins spécifiques du projet.

Ensuite, nous pouvons retrouver **un autre type de logiciel, permettant de vérifier l'intégrité du livrable face aux failles de sécurités.** Nous pouvons citer 2 outils : Dependency Track, qui répertorie les CVE liés aux librairies d'un projet et Snyk, qui permet de faire la même chose que Dependency Track en ajoutant des vérifications supplémentaires sur les fichiers de description d'Infrastructure as a Code et sur les failles de sécurité des conteneurs et leur registry.

A ce sujet, **notre conviction est qu'il faut utiliser ces outils comme une aide à la revue de code au quotidien** car l'utilisation de tels outils en "One-Shot" est quelque chose d'assez anxiogène au vu du nombre de retours qu'il est facile d'imaginer à traiter. De plus, il ne faut pas oublier que **ces outils ne permettent pas d'avoir le recul métier nécessaire à la bonne compréhension du code** et parfois, il est compliqué de corriger certaines remontées car ces outils ne prennent pas en compte les contraintes métiers du projet en développement.

4. L'IA au service de la qualité de code

L'intelligence artificielle (IA) est LE sujet tendance du moment, il est bien vu de l'intégrer dans divers aspects du processus de développement. Nous allons voir ici comment l'utiliser intelligemment à travers différents cas d'utilisation qui pourront vous permettre d'améliorer la qualité du code de vos projets.

Lorsque l'on parle d'IA, nous parlons en réalité principalement d'IA générative. Cette dernière est excellente pour rédiger, que ce soit pour écrire une recette de cuisine ou du code. Si on devait résumer, cette IA est principalement basée sur des statistiques pour générer un texte qui fait le plus de sens par rapport à la demande. C'est d'ailleurs pour cette raison que l'IA génère les mots les uns après les autres, elle n'a aucune idée de la fin de la phrase lorsqu'elle la commence.

L'objectif de cette explication est de mettre en évidence qu'il peut être difficile de croire aveuglément tous les codes que l'IA peut générer. Dans vos processus de rédaction de code, ce sera une aide très précieuse, mais il vous sera donc difficile d'automatiser des processus de vérification de code grâce à de l'IA. L'IA apportera une réponse un peu banale si vous lui demandez « Est-ce que la qualité de mon code est bonne ? » alors qu'elle sera bien plus compétente pour répondre à la question « Est-ce que mon code est bien découpé en fonctions ? ». Si vous souhaitez automatiser des revues de code grâce à l'IA, la demande doit être très précise.

Pour la qualité du code, l'IA est une grande aide ! Cette dernière peut très facilement vous expliquer comment rédiger votre code dans les grandes lignes en vous sauvant des nombreuses heures de lecture de documentation. Cependant, **il faudra toujours bien relire le code ainsi que d'y apporter des modifications pour qu'il soit parfaitement intégré dans votre projet.**

L'IA est également très douée lors de la rédaction des tests, cela permet de rédiger bien plus rapidement des tests unitaires ou d'intégration. Vous n'avez plus de bonne raison de ne pas écrire de tests sur vos projets.

Un point bien souvent oublié lorsque l'on parle d'IA et de qualité de code est l'apprentissage. L'IA ayant accès à de très larges connaissances, elle sera facilement capable de répondre à vos questions en lien avec la qualité de code. Elle sera toujours une très bonne source d'information pour parfaire vos connaissances sur différents sujets. Attention tout de même, **il est toujours important de vérifier ce que l'IA vous dira.** Dans ce cas d'usage, **on ne peut que grandement conseiller l'utilisation de Bing (Copilot) qui, à côté de vos explications, vous fournit les différents liens de sites utilisés pour rédiger sa réponse**. Cela vous permet d'avoir directement accès aux différents sites qui peuvent répondre à votre question et ainsi vous faire un avis par vous-même.

Nous souhaitons conclure cette dernière partie sur l'IA avec un petit avertissement : **lors de l'utilisation de l'IA, il est bien important d'avoir en tête ses limitations.** Il nous a été couramment donné de voir l'IA générer des codes qui contiennent des failles de sécurité. Il est important d'avoir une bonne connaissance du code et des bonnes pratiques. Cela va vous permettre de remettre facilement en question le code généré par l'IA ainsi que de l'aiguiller au mieux lors de sa rédaction pour obtenir le meilleur code possible. Le meilleur conseil que l'on peut vous donner lorsque l'on parle de génération de code par l'IA est de ne jamais demander à une IA de générer un code dont vous n'avez aucune idée du résultat final attendu. **Si vous n'êtes pas capable de challenger la proposition de l'IA en termes de qualité de code, le risque d'intégrer le code généré est bien trop grand.**

5. L'audit de code - L'arme ultime

L'audit de code consiste à examiner le code afin d'identifier les problèmes potentiels et les améliorations possibles.

Étonnamment, en 2024, cette tâche doit encore être effectuée par un humain. Bien que l'intelligence artificielle commence à jouer un rôle dans cette activité, l'audit humain reste essentiel pour améliorer le code source. Cependant, l'audit de code est un exercice délicat. Premièrement, **cela peut sembler surprenant mais, il n'existe pas de manière parfaite d'écrire un logiciel.**

Pour illustrer ce problème, considérons le paradoxe suivant : en tentant d'améliorer la lisibilité du code, on peut en diminuer la performance. Pourquoi ? Car un code source performant doit être écrit à un niveau bas, utilisant des instructions primitives. Plus on emploie ces instructions, moins le code est lisible et maintenable. Ce n'est qu'un exemple parmi tant d'autres qui démontre qu'**il est impossible d'obtenir un code parfait**. L'enjeu de l'audit de code est donc de naviguer dans ces eaux troubles pour répondre au mieux aux besoins du projet. Pour cela, il faut d'abord discuter des priorités : performance, maintenabilité, rapidité de livraison des fonctionnalités, ou prévention des régressions ?

Ce n'est qu'après cette analyse qu'on peut évaluer un code source. Et pour l'instant, seuls les humains en sont capables, car il n'est plus possible d'appliquer des règles génériques. De notre côté, on pense que cette situation ne changera pas de sitôt, et que, même si l'IA peut assister dans l'amélioration de la qualité du code, elle ne pourra pas remplacer l'expertise humaine.

Ce que sait faire un humain, et qui reste hors de portée d'une machine, est la capacité à gérer des contextes d'applications complexes, avec de nombreux modules interagissant entre eux. La capacité d'abstraction d'un développeur expérimenté est toujours supérieure.

Enfin, **une compétence primordiale de l'audit de code est de remettre en question la fonctionnalité même du code.** Il n'est pas rare qu'en cherchant à comprendre et à améliorer un code, on conclue que celui-ci est inutile et pourrait être conçu différemment.

Après ce long préambule, nous pouvons détailler **les points importants à examiner lors d'un audit :**

- **Identifier les bugs évidents** (en s'appuyant sur des listes de bugs classiques selon les langages et frameworks).
- **Maximiser l'utilisation des frameworks** (vérifier si des tâches sont effectuées manuellement alors qu'elles sont disponibles de manière standard dans le framework - cela nécessite que l'auditeur effectue une veille technologique).
- Porter une **attention particulière aux problèmes de performance classiques** (avec un focus sur les requêtes à la base de données).
- Assurer la **lisibilité du code** (il ne s'agit pas seulement de vérifier la présence de commentaires, mais de s'assurer que le code est compréhensible).
- **Examiner le processus de développement** (il est important de regarder comment le git flow est utilisé, la gestion des environnements, le DevOps, etc.).
- Profiter de l'audit pour **former les nouveaux arrivants** (tout projet respectable devrait intégrer cette pratique pour transmettre ses valeurs).
- **Valider la modélisation** (ce qui est difficile car on tend à adopter la perspective du développeur initial. Il faut prendre du recul et remettre en question à la fois le code et la base de données).
- Examiner impérativement la **sécurité du code** (même si elle est gérée par le framework, il est crucial de se baser sur les vulnérabilités communes comme celles du OWASP top 10).
- **Examiner les dépendances** pour s'assurer qu'elles soient à jour et éviter l'ajout de librairies suspectes.

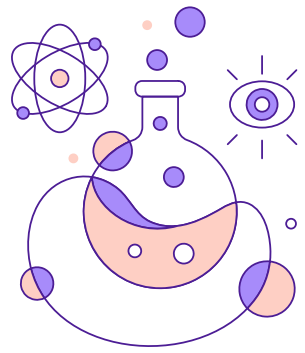
6. La sensibilisation des développeurs - La grande oubliée ?

Lorsque nous abordons la question de la qualité du code, **le développeur se trouve au centre de la discussion et devrait être la principale priorité en termes d'investissement** si l'on souhaite améliorer la qualité globale de nos livrables.

La formation joue un rôle crucial à cet égard, agissant sur plusieurs fronts. Tout d'abord, sur le plan humain, une entreprise prospère a besoin d'une équipe motivée et qualifiée pour accomplir les tâches qui lui sont confiées. Ainsi, la formation est essentielle pour valoriser les compétences de vos développeurs. Ensuite, du point de vue du code à proprement parler, un développeur qui acquiert, comprend et applique des concepts visant à améliorer la qualité du code le fera de manière durable, contribuant ainsi à la réussite continue des projets. Enfin, en participant à la revue croisée des "merge requests" ou "pull requests" de son projet, il favorisera également le développement des autres membres de l'équipe.

7.Le DevOps - Voiture balai pour assurer une qualité optimale ?

Le rôle du DevOps dans l'assurance de la qualité du code est essentiel et se manifeste à travers plusieurs aspects clés.



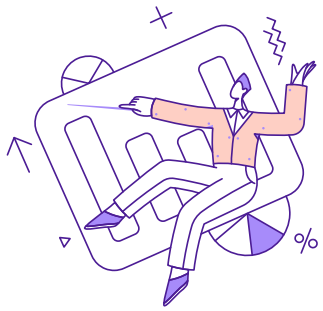
Tout d'abord, en favorisant l'intégration continue (CI), le DevOps encourage une pratique où les développeurs fusionnent leurs modifications de code dans une branche principale plusieurs fois par jour. **Cette approche permet une détection rapide des erreurs de code, facilitant ainsi leur correction immédiate**. De même, le déploiement continu (CD) est facilité par le DevOps, automatisant les tests et le déploiement des modifications dans l'environnement de production après leur intégration, réduisant ainsi les risques d'erreurs de déploiement et assurant une livraison plus rapide du produit.

Parallèlement, **le DevOps encourage la collaboration entre les équipes de développement et d'exploitation,** favorisant une communication plus fluide, une résolution rapide des problèmes et une amélioration globale de la qualité du code. L'automatisation des tests est également un pilier du DevOps, permettant une détection précoce des erreurs et des problèmes de qualité du code, libérant ainsi du temps pour les développeurs afin qu'ils puissent se concentrer sur la résolution des problèmes plutôt que sur la recherche des erreurs.

En outre, **le DevOps permet un feedback continu tout au long du cycle de vie du développement logiciel,** ce qui permet aux équipes de comprendre les problèmes potentiels plus tôt et de prendre des mesures correctives en conséquence. Dans l'ensemble, le DevOps a un impact significatif sur l'assurance qualité du code, le cycle de vie du développement logiciel et la collaboration entre les équipes, accélérant le processus de développement, améliorant la qualité du produit et augmentant la satisfaction des clients.

8. Le FinOps - La maîtrise du budget

Le FinOps occupe une place primordiale dans la gestion des ressources attribuées à un projet, notamment dans un environnement Cloud, et son importance ne peut être sous-estimée ! Tout d'abord, en matière de gestion des coûts, **le FinOps fournit une visibilité et un contrôle précieux sur les dépenses liées à l'utilisation des ressources Cloud,** permettant ainsi aux développeurs de comprendre l'impact financier de leurs décisions et de prendre des mesures pour optimiser leur utilisation.



De même, le FinOps permet une allocation efficace des ressources en fonction des besoins spécifiques du projet, évitant ainsi les pièges du sur-provisionnement ou du sous-provisionnement, qui peuvent entraîner des erreurs d'infrastructure coûteuses. En offrant une transparence sur l'utilisation des ressources et les coûts associés, **le FinOps permet aux équipes de mieux comprendre où et comment les ressources sont utilisées, favorisant ainsi des décisions éclairées pour optimiser leur utilisation.**

La collaboration entre les équipes financières et techniques est également encouragée par le FinOps, ce qui favorise une meilleure compréhension des coûts et une gestion plus efficace des ressources. De plus, l'automatisation des processus de gestion des ressources et des coûts, promue par le FinOps, permet de réduire les erreurs manuelles et d'améliorer globalement l'efficacité des opérations.

En résumé, le FinOps joue un rôle crucial dans le contrôle des ressources allouées par un projet, limitant les erreurs d'infrastructure et aidant les développeurs à appréhender l'impact financier de leurs décisions. Son importance est particulièrement flagrante dans un environnement Cloud, où les développeurs ont de plus en plus de responsabilités dans la création et la gestion des ressources.

9. Performance et éco-conception, même combat ?

De plus en plus de travaux existent sur l'éco-conception des applications. Pour certains, c'est du bullshit ou du greenwashing, mais de notre côté, nous sommes convaincus qu'il est possible de faire de grands gains en suivant ces approches.

Pour avoir pris le temps de le faire depuis un moment, la notion d'éco-conception peut, de prime abord, braquer les développeurs. Aussi, **il nous parait plus pertinent de prendre l'axe « performance » et « qualité » pour faire passer le message, à savoir, la nécessité de travailler à réduire l'empreinte de nos applications.**

Pourquoi s'en occuper ? Cette question naïve n'est pas nécessairement la plus simple à traiter ! On peut clairement se demander quel est l'intérêt de réduire l'usage des serveurs alors que les louer ne coûte pas bien cher, ou en tout cas beaucoup moins cher que l'heure d'experts techniques !

On peut apporter quelques éléments et remarques :

- **Le coût des hébergements explose** et commence à peser un poids certain dans les comptes des entreprises.
- **Les entreprises sont de plus en plus contraintes par l'établissement d'une comptabilité carbone**
- **Les applications éco-conçues sont souvent les plus optimisées et les mieux codées.** Il n'y a pas d'antinomie entre la notion d'éco-conception et de performances
- **L'évolution de la puissance des CPUs va moins vite que la demande de calcul…** On ne peut plus être sauvé par la loi de Moore.

Honnêtement, si vous n'êtes pas convaincus jusqu'à présent, ce n'est pas quelques arguments en plus qui vont changer la donne, alors **passons au concret : comment s'y prendre si on veut faire de l'éco-conception sans greenwashing ?**

La clé pour nous qu'il faut avoir en tête**, c'est que ce n'est pas l'optimisation du code qui est primordiale, mais bien la conception.** Si votre conception n'est pas optimale, le code ne pourra pas palier à ce problème ! Donc, on vous conseille vraiment de mettre le paquet sur l'analyse de la conception et de l'architecture des applications.

Note : comme ce n'est pas l'objet de ce document, on met ici de côté volontairement les notions de sobriété, pourtant essentielles sur ce sujet. On vous recommande le référentiel du gouvernement qui est de bonne qualité.

10. Conclusion – Que faut-il retenir ?

Pas évident de faire une synthèse sur un tel sujet ! Que retenir ? On vous donne quelques points qui synthétisent notre vision :

- L'IA est un bon allié quand on souhaite générer du code
- MAIS, il ne peut pas encore s'assurer à lui tout seul de la qualité et SURTOUT, il n'est pas en mesure de challenger l'ensemble d'une application, de son architecture et de son code !
- Peu importe les outils, l'important c'est que les humains (les développeurs) qui les utilisent soient formés !
- Enfin, et c'est peut-être le plus important : s'occuper de la qualité du code de ses applications est primordiale ! Qu'importe la méthode, occupez-vous-en !

11. Histoire - La qualité de code à travers les âges

11.1. Années 1950-1960 : Les Premiers Pas

- **Début du codage :** À cette époque, la programmation était principalement réalisée en langages de bas niveau comme l'assembleur.
- **Peu de normes :** Les pratiques étaient souvent définies par les limitations matérielles et les spécificités des premiers ordinateurs.

11.2. Années 1970 : Structuration et Modularité

- **Apparition des langages de haut niveau :** Des langages comme le C ont introduit des concepts de structuration et de modularité.
- **Début de la programmation structurée :** Cette approche, favorisant la clarté et la réduction de la complexité, a commencé à façonner les normes de qualité.

11.3. Années 1980 : L'Ère des Méthodologies

- **Méthodologies de développement :** Des méthodes formelles comme le modèle en cascade ont été adoptées, mettant l'accent sur la planification et la spécification.
- **Qualité et maintenance :** La qualité du code commence à être associée à la facilité de maintenance et à la réduction des coûts sur le long terme.

11.4. Années 1990 : L'Avènement de l'Objet et de l'Internet

- **Programmation orientée objet :** Des langages comme Java et C++ ont popularisé ce paradigme, influençant les standards de qualité autour de la réutilisabilité et de l'encapsulation.

11.5. Années 2000 : Qualité et Automatisation

- **Tests automatisés :** L'importance des tests unitaires et de l'intégration continue s'est affirmée pour garantir la qualité du code.
- **Normes et outils :** Des outils de vérification de la qualité du code (comme SonarQube) et des normes (comme ISO/IEC 9126) ont été développés.

- **Développement Agile :** Les méthodologies agiles ont commencé à remettre en question les approches traditionnelles, mettant l'accent sur la flexibilité et l'adaptation continue.

11.6. Années 2010 : DevOps et Culture du Code

- **Culture DevOps :** L'intégration des opérations et du développement a renforcé l'importance de la qualité du code pour la livraison continue et la gestion des infrastructures.
- **Code propre et artisanat logiciel :** La culture du "clean code" et de l'artisanat logiciel a mis l'accent sur des pratiques de codage lisibles, maintenables et élégantes.

11.7. Années 2020 : IA et Évolution Continue

- **Assistance par IA :** L'intégration de l'intelligence artificielle dans les outils de développement aide à la détection de problèmes de qualité et à la suggestion d'améliorations.
- **Pratiques évolutives :** Les pratiques de qualité continuent d'évoluer avec les nouvelles technologies, les frameworks et les paradigmes de programmation.

12. Références

- https://itsocial.fr/enjeux-it/enjeux-production/devops-devsecops/des-mesures-pour-ecrire-du-code-de-qualite/ Par **Solange Belkhayat-Fuchs** 07/04/2022
- https://owasp.org/ La référence des failles : OWASP
- https://dependencytrack.org/ Le super conseil outil : Dependency-Track
- https://www.axopen.com/blog/2023/11/audit-code-source-erreurs/ Les 5 erreurs les plus fréquentes de l'audit de code