Quel est le vrai coût d'une ligne de code ?

Quel développeur n'a jamais été agacé face à cette question récurrente ? Et systématiquement derrière ce type d'interrogation se cache un responsable marketing ou un client. Ces derniers attendent un montant sonnant et trébuchant, qu'ils pourront comprendre. Or, nous le savons, il n'est pas possible et pertinent de faire ce calcul. Cet article a pour vocation de vous donner des arguments pour expliquer à vos différents interlocuteurs, comment évaluer le coût du code dans sa globalité, c'est-à-dire en prenant en compte les aspects humains et les coûts cachés.

Aurélie GUILLAUME
Responsable du développement
Web chez Creads
@slig36
www.creads.fr

Partons donc de l'élément dont on nous demande le coût, à savoir "une ligne de code". Comme vous l'avez probablement constaté, il existe plusieurs manières d'écrire un test ou une fonction dans chaque langage.

Dans l'exemple ci-dessous, codé en PHP, les internautes doivent donner leur âge pour être autorisés à consulter tout ou partie des contenus du site. Voici 4 manières différentes d'écrire le code pour solliciter cette requête :

Exemple 1: 6 lignes de code

```php
$age = 42;
if ($age >= 18) {
    $majeur = true;
} else {
    $majeur = false;
}
```

Exemple 2: 5 lignes de code

```php
$age = 42;
$majeur = false;
if ($age >= 18) {
    $majeur = true;
}
```

Exemple 3 : 2 lignes de code

```php
$age = 42;
$majeur = ($age >= 18) ? true : false;
```

Exemple 4: 10 lignes de code

```php
$age = 42;

function isMajeur($age)
{
    $majeur = false;
    if ($age >= 18) {
        $majeur = true;
    }

    return $majeur;
}

$majeur = isMajeur($age);
```

Avec ces 4 exemples, on pourrait en déduire que le coût de production de l'exemple 4 est le moins important tandis que l'exemple le plus cher serait l'exemple 3. Néanmoins l'exemple 3 peut manquer de lisibilité si la condition de test est plus compliquée. En outre, sa syntaxe n'est pas employée par tous les développeurs.

Imaginons également qu'au sein de votre application vous ayez besoin de faire ce test à 15 endroits différents, l'intérêt d'une fonction générique utilisable depuis plusieurs endroits du site vous fera gagner de précieuses minutes et lignes de code. On le voit ici, on ne peut pas affirmer que moins il y a de lignes de code plus leur coût sera élevé. Le temps de conception est à prendre en compte.

En effet, reprenons notre besoin qui est de savoir si une personne est majeure ou non. L'emploi du code de l'exemple 1 ou 2 de manière récurrente mènerait inéluctablement à la nécessité de factoriser ce code dans une fonction pour en éviter la duplication. Or, cela prend plus de temps de factoriser du code déjà produit et testé que d'avoir pris un peu plus de temps en conception pour avoir anticipé ce besoin. On constate donc que le temps de la conception ne se traduit pas dans le coût chiffré d'une ligne de code.

Et l'humain dans tout cela ?

L'anticipation du besoin devrait être traitée en amont de la production d'une ligne de code. En effet, c'est à ces moments là, qu'interviennent des acteurs (architecte logiciel, data scientist, ux designer, chef de projet, Scrumaster...) qui n'écrivent pas à proprement parler la ligne de code dont on souhaite connaître le coût. Ces acteurs produisent également de la valeur. Ils interviennent sur un périmètre donné. Ils ne produisent pas de code mais sont bien souvent essentiels à la ligne de code qui sera produite en définitive.

Après la réalisation de cette ligne de code unitaire, il y a également d'autres coûts humains cachés qui peuvent être ceux des testeurs, d'autres développeurs qui reliront la ligne produite ou un admin sys / devOps qui sera en charge de mettre en ligne cette ligne de code. Tous ces acteurs ont un coût qui ne peut être calculé par le simple ratio : nombre de lignes total du projet / nombre de développeurs sur le projet x coûts de ces développeurs.

Quick and dirty ?

Imaginons maintenant que l'on vous demande de développer un site facturé 10 000 € au client et estimé 30 jours-homme. Vous n'avez pas de spécifications, pas de chef de projet ou de référent fonctionnel, et que le projet s'avère être un projet urgent pour lequel vous ne disposez plus que de 20 jours-homme. La tentation du quick & dirty (production rapide du code mais sale / mal fait) deviendra très forte surtout si l'on vous dit qu'il ne faudra pas maintenir le site après l'avoir mis en ligne.

Si nous avions réalisé ce développement en 30 jours comme prévu initialement la somme de nos lignes de code n'aurait pas été un tiers supérieur au nombre de lignes de code produites sur les 20 jours. Cependant, dans 6 mois, la version 30 jours aura un coût de maintenance plus faible. Et pourtant dans notre exemple, le client paiera toujours 10 000 € son projet et ne connaîtra jamais cette différence qualitative de coût de conception qui restera dans les coulisses de l'agence Web ou la SSII. Il se demandera juste dans 8 mois pourquoi les coûts de maintenance (TMA) sont aussi chers...

Cost of Legacy code ?

La factorisation des lignes de code, au-delà de la conception en amont du projet, est un sujet important chez les développeurs. En effet, nous avons chacun notre manière de faire et donc notre manière de factoriser. Donc forcément lorsque l'on reprend le code d'un autre, a fortiori si c'est du "legacy code" (code non évolutif historique d'un projet), nous avons très envie de le faire évoluer, ou carrément de le réécrire entièrement. Cela ne doit pas être perçu par le client comme un frein ou une perte de temps et de productivité immédiate du projet, mais comme un investissement qualitatif à long terme. Cette réécriture va forcément faire varier le coût de notre ligne de code unitaire qui valait N € avant notre intervention, et vaudra Y € après notre passage.

La reprise de legacy code fait accroître le coût de celle-ci. En effet, plusieurs facteurs vont entrer en compte. Depuis combien de temps entretient-on ce code ? Qui en est à l'origine et qui est intervenu dessus ? Dans quel langage a-t-il été écrit ?

La notion de l'intervenant sur une ligne de code est importante et va fortement faire varier un coût d'intervention. En effet, si le développeur

intervenant est un développeur expérimenté, qui connaît l'historique du projet et de la techno, il sera en mesure d'intervenir de manière chirurgicale. Un jeune padawan (le nouveau stagiaire), passera son temps à essayer de comprendre le sens du code, et ne pourra pas intervenir aussi efficacement. Il devra être aidé par un senior assistant ou un autre développeur ce qui va aussi gonfler le coût du code.

De la même manière si un projet est passé entre de nombreuses mains sans avoir été un minimum géré par un Responsable Technique ou des gardes fous type "code Review" (relecture du code produit par d'autres développeurs de l'équipe), garants des grandes règles et principes de code du projet, il sera plus coûteux de le maintenir au fil du temps.

Processus qualité, un investissement utile ?

La notion de qualité entrevue au travers des revues de codes (Code Review) peut amener bien plus de coûts supplémentaires cachés qu'il faut être en mesure d'expliquer à un manager ou un client. En effet, la mise en place d'un process de contrôle qualité fonctionnel ou technique (test fonctionnel &/ou unitaire) participera au bon développement du projet mais reste un investissement initial à faire. Cela va forcément produire un plus grand nombre de lignes de code associées au projet mais facilitera sa maintenance. Il en va de même pour l'organisation et l'automatisation du processus de déploiement qui ne rentre pas directement dans le coût d'une ligne de code mais qui est à prendre en compte dans le coût global du projet.

La dette technique enfin est une notion qui n'est jamais budgétée par les entreprises. La dette technique comprend le code qui a été produit, mais qui n'est pas évolutif, ou les raccourcis de développement qui ont pu être utilisés dans l'urgence. Les entreprises tendent à ne pas prendre en compte les défauts de conception, de failles de sécurité ou d'obsolescence de la techno utilisée. Une étude réalisée par l'éditeur Cast a conclu qu'en moyenne le coût de la dette technique revenait à 3.61 $ par ligne de code et que le langage qui serait le plus touché serait le Java. Cela démontre encore une fois que plus la problématique de la qualité de code est prise au sérieux par le client et l'équipe en amont, plus ce coût diminuera et permettra des économies sur le long terme.

Les failles de sécurité ont aussi un coût !

La sécurité et le respect des lois sont également des enjeux à garder en tête pour une grande majorité de clients. Pour vous aider à les comprendre, nous allons vous mettre dans leur situation. Imaginons que vous êtes l'un des décideurs de Volkswagen et que vous demandez à l'un de vos développeurs de rajouter quelques lignes de code pour flooder le système de contrôle anti-pollution aux États-Unis. Le coût de cette modification en termes strict de lignes de code sera très faible. Mais le contre coup du non-respect de la législation américaine est énorme en termes de procès devant les tribunaux et d'image de marque.

Autre exemple, vous êtes maintenant à la tête d'un gros site e-commerce français et certains de vos clients un peu bidouilleurs arrivent à payer un panier de 800 €, 1 € symbolique... A ce rythme-là vous allez mettre la clé sous la porte rapidement et tout cela car un développeur n'a pas correctement sécurisé le système de paiement en ligne. Cela vous paraît possible car vous êtes développeur, mais si vous prenez des exemples réels pour votre interlocuteur, il s'intéressera beaucoup plus attentivement au temps supplémentaire pour développer son produit ou à l'audit sécurité que vous lui conseillez à la fin du développement. Et tout ceci n'aura pas changé le coût de la ligne de code en soit !

Et l'hébergement ?

Il en va de même pour les coûts cachés liés à l'infrastructure. Reprenons l'exemple de notre site e-commerce Français. Leur service de communication a décroché un passage TV dans l'émission Capital. Appel en panique du DG à 21h52 le dimanche soir : *"Le site ne fonctionne plus, je vais perdre des milliers de commandes à cause de vous !!!"*. En fait non, c'est plutôt à cause de lui. En effet, avant une augmentation prévue de l'audience, il est judicieux de faire faire des tests de montée en charge pour déceler des problèmes, et les résoudre grâce à la mise en place d'outils spécifiques (ajout de serveurs, systèmes de cache, CDN). Au delà des lignes de code supplémentaires nécessaires à la mise en place de ces outils, le coût en temps-homme pour la configuration de ces derniers n'est pas négligeable. Ces nouvelles lignes de code auront un coût. Cependant, ce coût supplémentaire sera moindre face au manque à gagner d'un site en panne suite à un pic d'audience mal anticipé.

.Net vs PHP, Objective C vs Java ?

Le choix de la techno de développement du projet est également un facteur influent sur le coût du projet. En effet, si le choix se porte vers du .Net vs du PHP, le coût du projet sera supérieur. En effet, le développeur .Net est plus rare que le développeur PHP donc plus cher, CQFD. De plus, un développement dans un langage propriétaire induira aussi une augmentation des coûts annexes (qui ne sont pas directement liés au développeur produisant du code) tel que l'hébergement (ici IISS), le coût d'utilisation de licence tierce (SQL server, Oracle...).

De plus, l'écriture en soit d'un test ou d'une fonction ne produira probablement pas le même nombre de lignes que l'on soit en python, en C++, ou en PHP.

En 2016, le choix de la techno n'est plus exclusivement de la responsabilité du développeur. En effet, dans le cadre d'une demande d'une application mobile, il est possible que le client ait besoin d'une application développée en langage natif, donc spécifique pour le type de mobile. Ou bien on pourra lui proposer le développement d'une application "universelle" grâce au framework d'encapsulation type "Phonegap", Cordova ou Ionic. Ce choix orientera forcément le coût unitaire de la ligne de code, car le développement natif est plus onéreux que le développement "cross platform'. De plus, si l'application demande un développement de fonctionnalités spécifiques sur les 2 platform leaders , le coût total de l'application sera au minimum 2 fois supérieur à celui d'un développement "hybride".

CMS, from scratch

Dans le questionnement des outils et technos à utiliser pour développer un projet, vous pouvez également être amené à vous poser la question : *"Dois-je utiliser un développement "From scratch", un framework ou un CMS (pour du Web) ?"*

En termes de nombre de lignes de code pur, le framework aura un nombre de fichiers et donc de lignes de code assez important pour faire tourner son coeur (routeur, moteur graphique, base de données...). Il en est de même pour le CMS qui aura des fichiers pour son fonctionnement mais qui embarquera probablement un back office d'édition du contenu plus ou moins complexe, ce qui va produire aussi beaucoup de lignes de code supplémentaires qui viendront noyer le véritable coût du développement et de la valeur ajoutée du développeur sur le projet.

Car oui, dans l'absolu produire un projet qui a 350 000 lignes de code dont 98% ont été produites indirectement, que ce soit par un Framework ou un CMS cela ne reflétera pas le coût réel du projet.

Prenons le site creads.fr, à titre d'exemple, nous utilisons le Framework Symfony2 pour nos développements. Cela fait 2,5 ans que nous travaillons sur le site. Le ratio de notre dev vs la base de Symfony est totalement différent et reflète plus la valeur ajoutée de nos développements. En effet, si l'on prend un projet Symfony 2.3 fraîchement installé, il contiendra 306 851 lignes de code pur PHP. Par contre, sur notre

projet, le nombre de lignes de code PHP comprend 482 643 lignes produites par nos soins. Faut-il pour autant déprécier la valeur des lignes du Framework utilisé ? Bien sûr que non. Le code du Framework, nous permet de gagner un temps précieux pour nous concentrer sur l'essentiel : le code métier du client, celui qui aura une forte valeur ajoutée mais également celui qu'il paye au final.

Si vous souhaitez connaître le nombre de lignes de code de vos projets, je vous invite à installer Cloc. Cloc est un utilitaire open source qui va permettre de connaître le nombre de lignes de code de vos projets par langage, tout en ignorant les fichiers types .git et les commentaires.

En conclusion

Désormais, lorsque l'on vous posera la question *"Quel est le vrai coût d'une ligne de code ?"*, vous pourrez énumérer un certain nombre d'arguments à votre interlocuteur et lui expliquer en quoi sa question n'est peut-être pas pertinente. Car, en effet, le coût d'une ligne de code ne veut pas dire grand chose. L'intégralité des coûts visibles et cachés doivent être pris en compte et non juste appliquer une logique de coûts unitaires. En somme, on pourra retenir que ce n'est pas le nombre de lignes de code d'un projet qui en définira son coût final pour le client mais plutôt le temps, les moyens humains, fonctionnels et technologiques affectés au projet qui devront être pris en compte.

```
http://cloc.sourceforge.net v 1.60  T=226.57 s (64.6 files/s, 7055.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
PHP                          10835         159391         310669         789494
Ruby                          1035           9674           2203          65952
Javascript                     479           9446           9504          59571
XML                            431           1472            119          34639
LESS                            85           3393            938          25715
Pascal                         600           3468          12325          22063
CSS                            258           2274            810          21631
HTML                           144           2785            265          18112
YAML                           626           1438            397          15617
XSD                             19            317             88           4297
Bourne Shell                    70            695            510           3342
Bourne Again Shell              20            189            193           1190
C                                1            166            150            793
Perl                             1             56            126            784
SQL                              9            169              2            731
Ant                              4            111             65            434
make                             4             60             28            178
DOS Batch                        2             17              1            104
Python                           2             69            144             81
C/C++ Header                     1              7             13             11
m4                               1              2              1              5
ASP.Net                          1              0              0              1
-------------------------------------------------------------------------------
SUM:                         14628         195199         338551        1064745
-------------------------------------------------------------------------------
```

(Nombre de lignes de code de creads.fr)

SOURCES

La dette technique qui coûte le plus chère est en JAVA :
http://www.lemagit.fr/actualites/2240198129/Les-defauts-logiciels-coutent-en-moyenne-361-par-ligne-de-code

CLOC : https://github.com/AlDanial/cloc#Overview

Google 2 000 000 lignes de codes
http://www.abondance.com/actualites/20150918-15577-google-cest-2-milliards-de-ligne-de-code.html

Infographie ligne de code des grands projets mondiaux :
http://www.numerama.com//content/uploads/2013/10/lignesdecodeinfographie.png

Moi je code m'ADOC

En tant que développeurs, nous sommes évidemment amenés à produire du code, mais également de la documentation : fichiers README, spécifications techniques/fonctionnelles, guide d'installation, manuel utilisateur, wiki, ... N'oublions pas nos amis étudiants et le fameux rapport de stage. Le but de cet article est de montrer comment mieux écrire sa « doc » en choisissant les bons outils collaboratifs provenant du monde Open Source. Préférons des outils simples, ouverts, gratuits, maintenus à jour par la communauté.

Mehdi Rebiai
mrebiai@sqli.com
Architecte technique SQLI. Speaker Bdx.io 2015 « moi je code ma doc ». Plus de 10 ans d'expérience dans le développement d'applications Java/JavaEE. Curieux d'apprendre de nouveaux sujets et aimant transmettre au plus grand nombre.

Attention au grand écart entre le monde du développement et les outils de la sacro-sainte documentation ! À ma gauche, un univers industrialisé (IDE, GIT, releases), et à ma droite une suite bureautique (MS Office, OpenOffice ...) complétée d'un modeleur UML et d'outils annexes. À moins d'utiliser une GED ou des outils collaboratifs genre « GoogleDoc », la documentation est souvent stockée sur un répertoire réseau d'entreprise avec la fameuse technique du « copier-coller-suffixer »! MonFicher-v1.doc deviendra MonFicher-v2.doc et un dossier « archives » sera créé pour gérer l'historique des anciennes versions. Drôle de versioning.

La figure 1 illustre ce grand écart « Doc vs Code » pour une application, chaque branche ayant son propre cycle de vie. Et pourtant, lors d'une livraison, on se doit de réconcilier l'artéfact applicatif (WAR, ZIP) et l'artéfact documentaire (DOC, ZIP). À noter également que le code contient sa propre documentation interne (commentaire, JavaDoc). Nous y reviendrons un peu plus tard. **Fig.1.**

Et si la « doc » était du code? La figure 2 prend cette hypothèse. Et maintenant tout devient plus simple. Il faut néanmoins changer certains outils. En allant plus loin, tout code est documentation, dixit le dicton du développeur « la vérité est dans le code »! **Fig.2.**

Vive le fichier plat!

Le principe du MDA (Model Driven Architecture) a longtemps été mis en avant pour son côté rigoureux. Un modèle UML source contient la conception UML complète de l'application ainsi que sa documentation. L'application est ainsi générée à partir de ce "gros" modèle source. Cette approche,