SpaceFox

Zeste de savoir

Mort aux commentaires inutiles ! Écrivez des commentaires pertinents!

17 décembre 2022

Table des matières

Introduction

1. Vos utilisateurs savent lire le code

Tout le principe de ce billet repose sur ce principe:

Les utilisateurs de votre code, savent lire le code.

La conséquence, c'est que vos commentaires ne doivent pas paraphraser le code, mais apporter quelque chose en plus.

2. Mort aux commentaires inutiles !

Les commentaires redondants

Vous avez peut-être déjà vu ce genre de commentaires:

```
1     /**
2      * Get the name
3      * @return the name
4      */
5     public String getName() {
6         return name;
7     }
```

Pire, peut-être que vous avez dû **écrire** (ou auto-générer) ce genre de commentaire pour satisfaire un collègue ou un outil d'analyse pinailleur, qui ne vous lâchera pas tant qu'il n'y aura pas *un commentaire sur chaque méthode publique.*

Et pourtant, sur ce commentaire est strictement inutile. Les lignes 1 à 4 de l'exemple ne sont que du bruit, qui n'apportent absolument rien par rapport à la méthode elle-même (la question de savoir si cette méthode est elle-même utile est traitée dans un autre billet).

Donc, si vous écrivez ce genre de commentaire, non seulement vous perdez du temps, mais en plus vous risquez des commentaires menteurs en cas de copier/coller ou de modification d'une méthode.

Les palliatifs aux mauvais nommages

Un autre type de commentaire inutile et hélas trop courant, c'est les commentaires qui pallient un mauvais nommage (de classe, de méthode, de fonction, de variable…).

Si vous devez écrire un pavé pour qu'on ait une idée de ce que fait l'élément que vous commentez, c'est sans doute qu'il est mal nommé. Mieux vaut un nom clair et un peu long qu'un nom court et obscur.

Et oui, trouver un nom pertinent, c'est souvent difficile.

Expliquer le « comment ? »

Il y a toute une catégorie de commentaires qui expliquent comment fonctionne le code, et qui paraphrasent complètement celui-ci.

Ça va du simple commentaire de débutant du type:

```
1 // If row count > max row
2 if (rowCount > maxRow) {
```

(ce genre de commentaire est aussi à ranger dans la première catégorie, mais est rarement demandé par les outils d'analyse statique)

… aux versions plus vicieuses qui réécrivent toute l'implémentation dans le commentaire d'en-tête de la classe ou de la méthode. Et très souvent ces commentaires sont mal mis à jour, et sont faux après la première modification de l'implémentation. Rappelez-vous: les gens qui vont utiliser votre code savent le lire.

Et si vous avez besoin de détailler ce qui est fait, c'est rarement le *«comment»*, mais plutôt le *«quoi»*, cf ci-dessous. Une alternative, c'est que vous êtes en train de faire un document d'architecture dans les commentaires de votre code, ce qui n'est pas beaucoup mieux.

Réponses à quelques arguments classiques en faveur des commentaires systématiques

C'est utile dans la doc autogénérée. *(Javadoc ou équivalent)*
Non. Cette documentation contient *déjà* les noms des méthodes, de leurs paramètres, les types des paramètres et de retour quand c'est pertinent. Répéter ça dans dans la documentation n'apporte rien, sauf de la frustration pour la personne qui voudra lire des

détails et découvrira qu'il n'y en a pas.
C'est utile pour la navigation dans l'IDE
Même réponse que ci-dessus.
L'outil d'analyse impose ces commentaires. *(Variante: il impose un certain nombre de lignes de commentaires)*
Un outil, ça se paramètre; les faux positifs, ça s'ignore. Vous avez des *merges request* pour signaler les commentaires utiles manquants. Si l'outil bloque sur ce genre de cas, c'est un problème d'organisation, pas d'outil.
C'est pour inciter à mettre des commentaires utiles.
Alors mettez sans attendre des commentaires utiles… et admettez que parfois, il n'y a rien de plus à dire que ce qu'indique déjà la signature de la méthode.
Ces commentaires paraphrasent le code mais permettent de suivre les grandes étapes d'un processus long.
C'est très clairement le signe que cette méthode doit être découpée en sous-méthodes bien nommées.

3. Qu'est-ce qu'un commentaire utile ?

Il y a pourtant plein de commentaires qui sont utiles à ajouter à votre code sans tomber dans la redondance qui se répète; voyons-en quelques-uns ici:

Expliquer le fonctionnel, le « quoi ? »

C'est très pratique d'avoir un commentaire qui explique l'utilité *fonctionnelle* d'un élément, sa raison d'être en-dehors de toute considération d'implémentation. Un commentaire qui explique le «quoi» et pas le «comment» (il y a le code pour ça).

Non seulement ça permet d'avoir des informations qui, par nature, ne sont pas déjà présentes dans le code; mais en plus en cas de bizarrerie, ça permet d'avoir une bonne idée de si l'étrangeté est due à une erreur d'implémentation ou à une règle fonctionnelle inattendue.

Ces commentaires devraient être systématiques sur les éléments de haut niveau (classes dans les langages objets, méthodes principales…)

Définir les cas aux limites

Le commentaire précise les cas aux limites (valeurs autorisées ou non pour les paramètres) et les différents types d'erreurs qui peuvent être rencontrés. C'est d'autant plus utile que les limites si les erreurs ne sont pas évidents à la lecture du code.

Ce type de commentaire est surtout utile pour les API publiques, en particulier de bibliothèques, pour lesquelles le code peut être difficile d'accès.

Expliquer les subtilités d'implémentation

Tout code non-trivial, tout algorithme un peu tordu ou qui sort de l'ordinaire devrait être commenté en précisant comment il fonctionne (si ça n'est pas clair à la lecture du code) mais surtout **pourquoi** ce fonctionnement inhabituel a été choisi.

Ça simplifiera la maintenance, et ça évitera des régressions ou du temps perdu en essayant de corriger un «problème» qui n'en est pas un. Et parfois même, rédiger ce commentaire vous permettra de vous rendre compte qu'en fait il y avait plus simple.

4. L'exception du code invisible

Il peut arriver que vous écriviez du code qui sera invisible, l'exemple typique est celui d'une bibliothèque, en particulier d'une bibliothèque qui sera utilisée dans un contexte où les sources ne sont pas disponibles (code privateur, environnement logiciel qui ne permet pas facilement d'accéder aux sources des bibliothèques utilisées…)

Dans ce cas, en effet, les utilisateurs ne «savent pas» lire le code – parce qu'ils ne *peuvent pas* le lire. Et donc il devient utile de répéter dans les commentaires des choses évidentes quand on a le code sous les yeux.

Conclusion

J'espère maintenant que vous n'exigerez plus de commentaires «par principe» dans votre code, et que vous dépenserez du temps et de l'énergie à écrire des commentaires *utiles*, plutôt qu'à paraphraser le code.

Mais oui, c'est plus difficile à faire. Par exemple, il n'y a pas d'outil de génération de commentaires pour ça…
