Maintenance applicative

Maintenance applicative : plan du module

- Gestion de la maintenance applicative
- **Maintenabilité**
- Changement d’outils

Maintenance applicative : plan du cours

- Maintenabilité
- Legacy et dette technique
- Clean code et Refactoring
- Documentation

Maintenance applicative : maintenabilité definition

- La maintenabilité d’un code est sa capacité d’un code à être modifié.
- Les modifications sont apportées dans le cadre de la maintenance corrective ou préventive.
- Il arrive très souvent que ce critère de qualité soit en concurrence avec d’autres facteurs lors d’un nouveau développement ou d’un projet de maintenance : durée, coût et compétences des développeurs.

Maintenance applicative : le code legacy definition

Un code legacy est un code existant souvent ancien. Sa connotation est négative car il est souvent difficile à maintenir pour plusieurs raisons :

- Évolution des technologies : le code utilise d'anciens framework, librairies ou formats de langage
- Faible qualité de codage : duplication, mauvais découpage fonctionnel ou classes trop grandes, peu ou pas de tests unitaires

Il arrive rarement qu'un code legacy soit de bonne qualité, ce qui se traduit par une application ou une librairie utilisée pendant longtemps (plusieurs années voire une dizaine d'années).

Maintenance applicative : le code legacy
la dette technique

- La dette technique est le coût associé à la modification d'un code existant.
- Sur un code legacy : plus le code à faire évoluer est de mauvaise qualité, plus la dette technique est importante.
- Sur un code en développement : on dit qu'on crée de la dette technique (un coût élevé de la future maintenance) lorsqu'on prend des mauvaises décisions de codage (mauvais choix d'architecture, peu ou pas de tests, peu ou pas de documentation technique) volontairement ou pas, le plus souvent à cause de délai trop court ou de budget limité ou de manque de compétences.

Maintenance applicative : le code legacy comment le maintenir ?

3 difficultés apparaissent :

- Comprendre : il faut documenter, schématiser techniquement un code à maintenir au fur et à mesure qu'on cherche à le comprendre. Les statistiques montrent que 50 % du temps de maintenance correspond à la compréhension du code legacy.
- Modifier : il faut utiliser les framework et librairies existantes autant que possible plutôt que produire du code maison pour réinventer des fonctionnalités. La maintenance peut introduire l'utilisation d'une librairie externe, testée, sécurisée à la place d'une partie de code legacy.
- Tester : l'approche TDD est parfaite pour la maintenance. Dans tous les cas, il faut enrichir les tests existants autour des modifications du code legacy.

Maintenance applicative : le code legacy l'inévitable maintenance

“Avoir un code Legacy n'est pas un échec, ça prouve que l'application a eu du succès dans le temps, malgré la dette technique. De toute façon n'importe quel code, aussi beau soit-il écrit aujourd'hui sera le Legacy de demain. Il est quand même important de revenir régulièrement pour rafraîchir le code et enlever progressivement de la dette technique.” explique Gabriel Pillet, CTO chez Web^ID.

Lorsque la dette technique est trop importante, un projet de maintenance caractérisé par une durée courte (<3 mois) et des ressources réduites n'est pas suffisant.

Un projet de réécriture de code peut être décidé sur du code legacy. Ce projet sera plus long (> 6mois) et implique plusieurs développeurs.

Maintenance applicative : les modifications clean code

Le clean code est un style de codage apparu dans la méthode Agile Software Craftsmanship.

Il est aujourd'hui une bonne pratique qui préconise :

- l'usage de noms significatifs
- l'éviction de commentaires inutiles
- l'optimisation des fonctions (une fonctionnalité par fonction)
- Une meilleure organisation du code

Il se pratique lors de tout développement ou maintenance de code.

Maintenance applicative : les modifications refactoring definition

- Le refactoring est à la fois une méthodologie de développement dans le cadre de projets Agile ou un projet dédié.
- Refactorer un code consiste à modifier la structure interne d’un code sans changer son comportement externe.
- Refactorer suppose un jeu de tests unitaires et d’intégration important pour détecter tout changement externe.

Maintenance applicative : les modifications refactoring techniques

Les techniques sont nombreuses mais ont en commun d'être courtes afin de vérifier le comportement externe du code par des tests à chaque étape.

Plusieurs techniques peuvent être utilisées à la suite :

- Modification de fonctions par composition
- Modification d'objets par mouvement d'attributs ou méthodes
- Réorganisation des données
- Simplification des conditions de tests
- Simplification des chaînes d'appel
- Factorisation ou généralisation

Maintenance applicative : les modifications réécriture de code

- Une réécriture de code est un projet dédié plus long que le refactoring qui est composé de cycle (de sprints) de quelques jours soit un total de quelques semaines.
- Réécrire un code support un délai plus long (en mois) et une équipe de développeurs dédiée. Par rapport au refactoring, une étape d'analyse du code et d'identification de tous les morceaux à refaire est nécessaire.

Maintenance applicative : documentation différentes solutions

La documentation d'un code est réalisée à 3 niveaux :

- Les commentaires : ce sont des informations explicatives de quelques lignes de code : symbole # ou """ pour commenter une ou plusieurs lignes
- Les docstrings : ce sont des informations explicatives d'une fonction ou d'une classe
- La documentation technique (générée à partir du code) : ce sont des informations sur tout le code. L'outil de génération Sphinx est utilisé.

Maintenance applicative : documentation docstrings (1)

Un bloc docstring est encadré par """. Il est intégré à une variable __doc__ caractérisant un module, une classe ou une fonction.

Les docstrings restent courts et explicites.

Pour les montrer, on peut par exemple dans l'interpréteur python taper :

- Import backlog ; print(backlog)
- Import backlog ; help(bakclog)

```
class Backlog(common.Common)
 |  Backlog class dedicated to read/write tickets into a ram
 |
 |  Method resolution order:
 |      Backlog
 |      common.Common
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      Backlog __init__ : backlogs and ticket creation
 |
 |  __repr__(self)
 |      Backlog __repr__ : debug purpose
 |
 |  close_backlogs(self)
 |      Backlog close_backlogs : unused for backlogs in ram
 |
```

Maintenance applicative : documentation docstrings (2)

Pour les montrer, on peut par exemple à l'exécution du code python appeler :

```
b = Backlog() # Backlog objet
print(b.__doc__) # Class docstrings
print(b.process.__doc__)
# Process method docstrings
help(b) # Full class docstrings
```

La différence entre print et help concerne le contenu précis ou général (tous les docstrings)

```
Class purpose : read/write tickets into a ram

        Backlog process : interface with tms

Help on Backlog in module __main__ object:

class Backlog(common.Common)
 |  Class purpose : read/write tickets into a
 |
 |  Method resolution order:
 |      Backlog
 |      common.Common
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |      backlogs and ticket creation
 |
```


Thierry Vert

Maintenance applicative : documentation sphinx généralités

- Sphinx est un outil de génération de documentation technique. Il va plus loin que les informations docstrings grâce à l'extraction d'informations dans le code et à la génération automatique d'une documentation au format pdf ou html par exemple.
- Installation du module Sphinx sous l'interpréteur Python : pip intsall sphinx
- Le module s'appuie sur un format de texte appelé le reStructuredText. Il permet à l'aide de marqueurs spécifiques d'enrichir le format texte traditionnel avec une numérotation imbriquée, une insertion d'image, de liens...

Maintenance applicative : documentation le format sphinx

Plusieurs formats existent et vous pourrez choisir parmi exu. L'exemple ci-dessous utilise le format Google.

Dans un docstring, on précise par :

- Args les arguments (type entre parenthèses) et descriptif après :
- Returns les valeurs retours (type entre parenthèses) et descriptif après :

```
def search_case_id(self,id):
    """
    Backlog search_case_id : backlog search for a ticket knowing id
    Looks for a ticket into backlog filtering on ticket id

    Args:
        id (str): ticket id

    Returns:
        ticket (dict): found ticket or {}
    """
    if self.current_backlog != []:
        for elt in self.current_backlog:
```

Maintenance applicative : documentation sphinx les commandes

4 commandes dans la console Python :

- Pip install sphinx : installation de l'outil
- Sphinx-quickstart : configuration d'un projet sphinx et stockage des préférences mentionnées dans conf.py
- sphinx-apidoc : auto-génération de fichiers d'informations et copie dans le fichier source
- Make html (pdf) : génération de la documentation au format html (pdf)

Maintenance applicative : conclusion

- Ecrire un code maintenable n’est pas facile. La documentation technique et les tests sont très importants pour ne pas alourdir la dette technique et favoriser la maintenabilité du code.
- Maintenir un code legacy est également difficile. Sa compréhension ets une étape indispensable et souvent longue qui nécessite l’écriture ou l’amélioration d’une documentation technique et des tests unitaires.
- Dans les deux cas, des étapes courtes de clean code et refactoring sont très utilisées afin de maintenir un code et d’améliorer sa dette technique. Un projet de réécriture est un dernier outil plus rarement utilisé.