Maintenance applicative

Thierry Vert

Maintenance applicative : programme

Descriptif détaillé

Objectif

L'objectif de cette ressource est de renforcer les techniques de développement pour assurer la maintenance des applications.

Savoirs de référence étudiés

- Rétroconception Projet 1 Projet 2
- Techniques avancées par les tests Projet 1
- Remaniement de code Projet 1 Projet 2
- Les différents savoirs de référence pourront être approfondis

Restructuration de code — Projet 1 Projet 2

Analyse et Adaptation de système d'information (SI) — Projet 1

Déboguage — Projet 1 Projet 2

Concrètement :

- 1 projet en langage interprété Python
- 1 projet en langage compilé C

Maintenance applicative : organisation

3 CM, 7 TDs

5 séances TD (python) sur projet 1
2 séances TD (langage c) sur projet 2
Les codes seront évalués à la fin de chaque projet

Programmation système : plan du module

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

Programmation système : plan du cours

- Les types de maintenance
- Les acteurs de la maintenance
- Gestion des demandes clients
- Cycle de vie
- Outils de la maintenance applicative
- Les corrections et les tests

Programmation système : maintenance définition

Un service informatique connaît 2 activités principales :

- Le développement logiciel : cadre d'un projet à durée limitée, objectif répondre aux demandes clients
- La maintenance logicielle : activité continue et non ponctuelle

La maintenance est inévitable : après le développement le logiciel et son déploiement chez les clients, un logiciel est utilisé et vieillit.

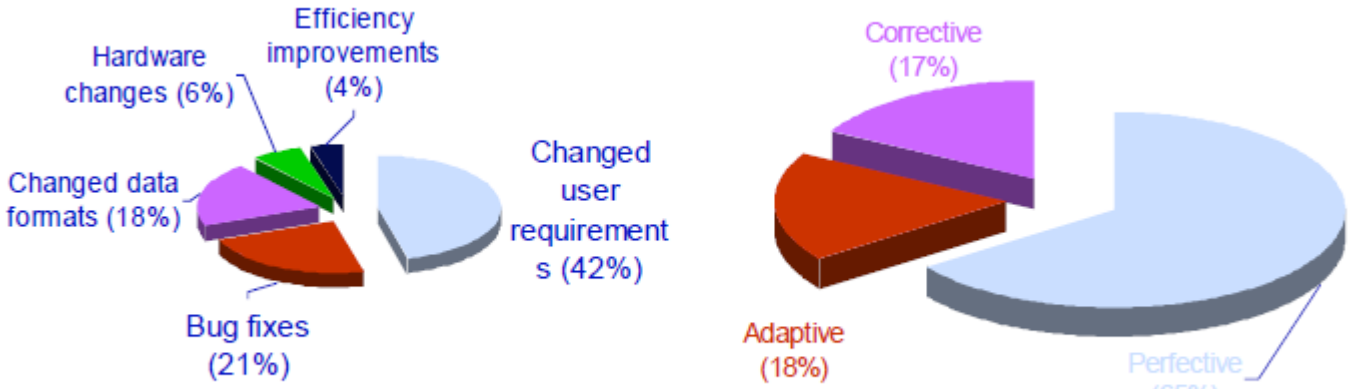
Pendant sa durée d'utilisation, des problèmes de fonctionnement (bug) ou des fonctionnalités manquantes (améliorations) peuvent apparaitre

Programmation système : maintenance différents types

- Maintenance corrective : correction des défauts, des dysfonctionnements (bug)
- Maintenance adaptative : adapter les applications à un changement d'environnement technique ou de système
- Maintenance perfective: prise en compte de nouvelles fonctionnalités (amélioration)
- Maintenance préventive : modification du logiciel pour diminuer la probabilité de défaillance

Quel que soit le type de maintenance, il est essentiel de maintenir à jour les connaissances sur le logiciel notamment grâce à une mise à jour des documents techniques.

Programmation système : maintenance répartition des types de maintenance



[Lientz&Swanson 1980]

Programmation système : maintenance référentiel ITIL

Le processus de maintenance applicative est coordonné par un responsable dans un service de support logiciel. Le responsable identifie les développeurs dont une partie des tâches concerne la maintenance.

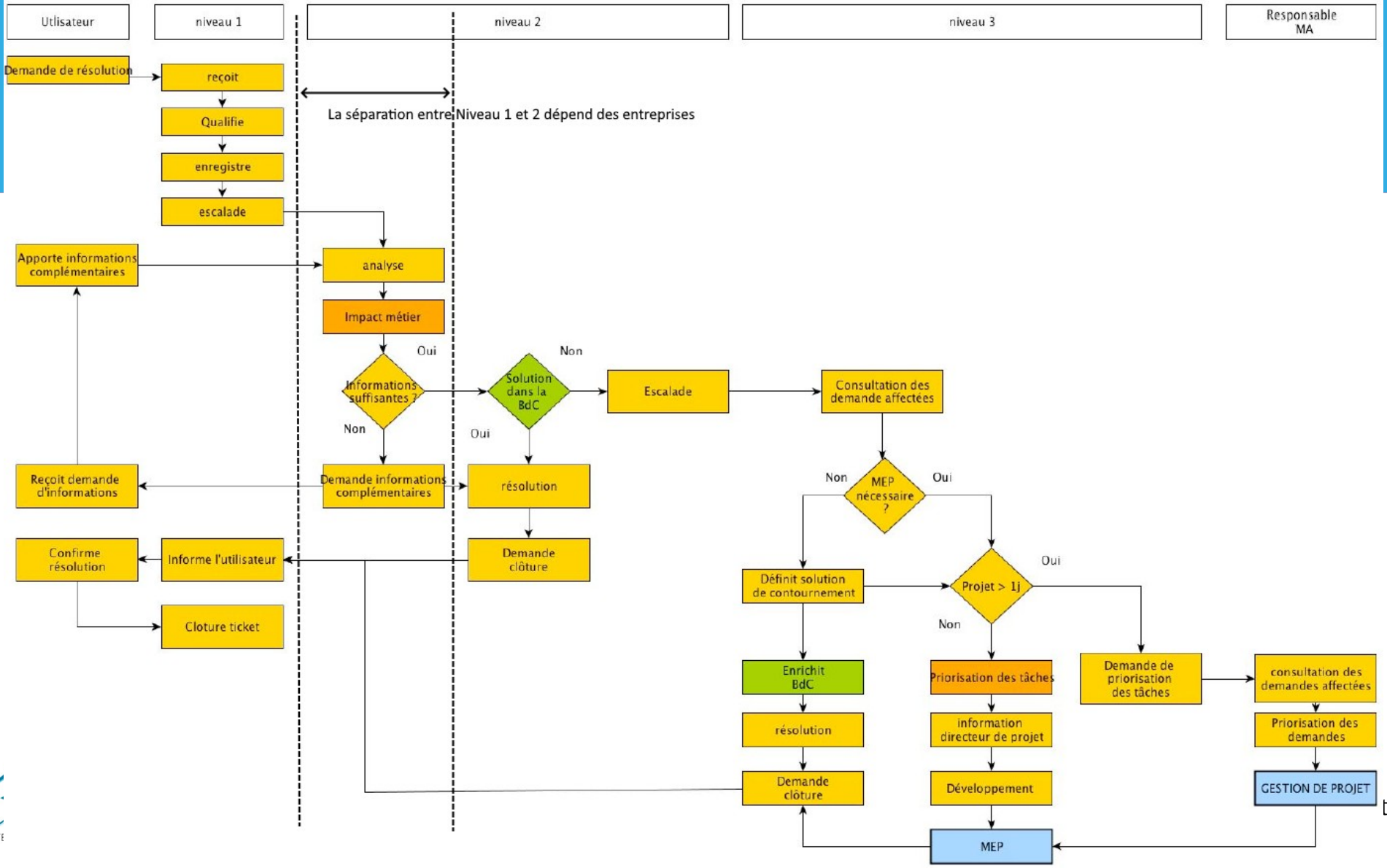
Le référentiel numérique ITIL (Information Technology Infrastructure Library) organise la maintenance en plusieurs niveaux :

- Le niveau 1 est chargé du contact client et la résolution si le problème est connu
- Le niveau 2 est chargé de la reproduction et résolution des incidents que le niveau 1 n'arrive pas à résoudre.
- Le niveau 3 est composé d'experts qui interviennent sur les problèmes techniques difficiles non résolus aux niveaux précédents.

On parle d'escalade pour désigner le passage d'un incident d'un niveau au supérieur.

Programmation système : maintenance procédure de gestion des cas

- Dans les entreprises, une demande de résolution de problème (incident) ou d'amélioration est appelé **un cas ou un ticket**.
- Dans le référentiel ITIL, la gestion des ticket (workflow), la modification du code, la gestion de la base de connaissances (BdC) et la mise en production (MEP) sont distincts. A chaque étape, un responsable et un indicateur de performance sont associés au ticket.
- Le traitement d'un ticket est constitué de plusieurs étapes : contact par le client, la demande client est gérée au niveau 1, 2 ou 3 selon son caractère connu ou nouveau (BdC), son état de reproduction et sa résolution (correction, MEP). Une réponse est apportée au client. Le ticket est finalement fermé et la BdC mise à jour.
- Un ticket possède donc un état qui varie dans le temps.


Utlisateur
niveau 1
niveau 2
niveau 3
Responsable MA
Demande de résolution
reçoit
Qualifie
enregistre
escalade
La séparation entre Niveau 1 et 2 dépend des entreprises
Apporte informations complémentaires
analyse
Impact métier
Oui
Informations suffisantes ?
Non
Solution dans la BdC
Non
Oui
Escalade
Consultation des demande affectées
Reçoit demande d'informations
Demande informations complémentaires
résolution
Non
MEP nécessaire ?
Oui
Confirme résolution
Informe l'utilisateur
Demande clôture
Définit solution de contournement
Projet > 1j
Oui
Non
Cloture ticket
Enrichit BdC
Priorisation des tâches
Demande de priorisation des tâches
consultation des demandes affectées
résolution
information directeur de projet
Priorisation des demandes
Demande clôture
Développement
GESTION DE PROJET
MEP

Programmation système : maintenance client et support niveau 1

Les utilisateurs ont un seul point de contact, le support de niveau 1, pour demander l'ouverture et la résolution d'un ticket.

Cette unique point de contact permet de :

- Simplifier et harmoniser la communication avec le client
- Avoir une vue d'ensemble des problèmes d'un client

Le support niveau 1 enregistre et catégorise la demande du client. Il demande des renseignements complémentaires si nécessaire et apporte une solution technique rapide si celle-ci figure dans la BdC. S'il n'en est pas capable, il escalade le cas.

Le support de niveau 1 est le seul autorisé à fermer une demande client. On parle de clore un ticket.

Programmation système : maintenance support niveaux 2 et 3

- Le support niveau 2 est en charge de la reproduction et de la résolution de problème non mentionnés dans la BdC.
- Si le niveau 2 arrive à résoudre le problème, il met à jour la BdC, communique la solution au niveau 1 et demande la fermeture du ticket.
- Si le niveau 2 n'y arrive pas, il escalade le cas au niveau 3.
- Le niveau 3 fait appel à des experts. Leur rôle est de reproduire, identifier et corriger le nouveau problème (amélioration) remonté(e). Ensuite, il met à jour la BdC, communique la solution au niveau 2 et demande la fermeture du ticket.
- Les supports niveau 1, 2 et 3 sont nommés **L1, L2, L3** en anglais

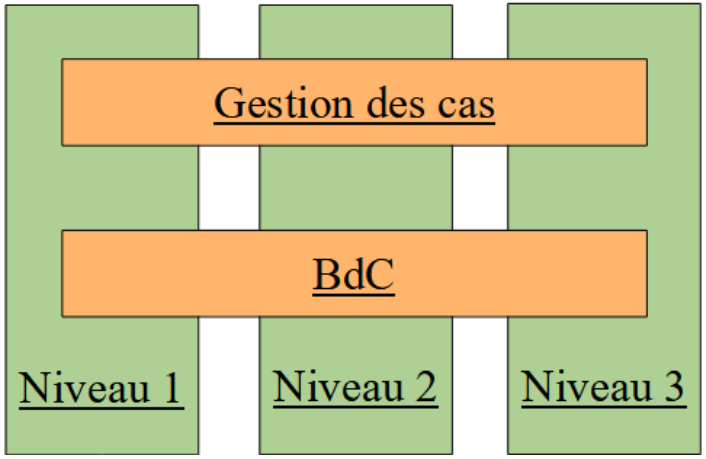
Programmation système : maintenance le support et les outils

D’après les actions de la maintenance gérées par les équipes support niveau 1, 2 et 3, il est nécessaire de partager :

- une base de connaissances (BdC)
- un outil de gestion des ticket aussi appelé bug tracker en anglais

Les outils utilisés sont propriétaires ou standards selon le choix des entreprises. Ils s’appuient sur une ou plusieurs bases de données partagées.

L’ensemble des tickets ouverts, présents dans l’outil est appelé un **backlog**.



Programmation système : maintenance caractéristiques des cas

Un ticket contient plusieurs informations qui seront ces attributs dans une DB :

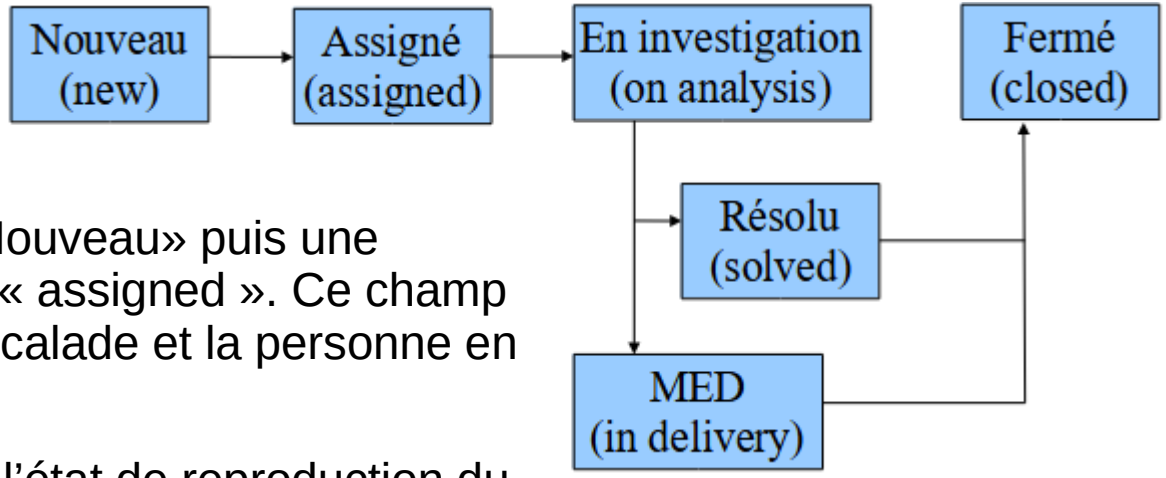
- Client : nom du client
- Type : problème / amélioration aussi appelés PR (Problem Report) / IR (Improvement Request)
- Description : détails sur le problème remonté ou la demande d'évolution
- Date d'ouverture : timestamp déterminant le délai depuis ouverture
- Etat : étape dans le workflow
- Assigné à : responsable du traitement correspondant à l'état

Pour avoir une clé primaire, il faut ajouter un numéro d'identifiant permettant de rendre unique un cas.

Un schéma simple SQL serait : cas{numéro, client, type, description, date, etat, assigne_a}

Programmation système : maintenance cycle de vie d'un ticket

Une demande remontée par le client est transformé en un cas dans l'outil de gestion des ticket.



A sa création, le ticket est dans l'état «Nouveau» puis une personne se charge de sa gestion, état « assigned ». Ce champ peut être mis à jour selon le niveau d'escalade et la personne en charge au niveau 1,2 ou 3.

Il est ensuite en « investigation » : c'est l'état de reproduction du problème et d'analyse. L'état suivant est « Résolu » si le problème est dans la BdC ou si une solution est trouvée sans modification du code. Il passera en Mis en production « MED » si une modification de code a été nécessaire et prête à être délivrée. Le cas sera fermé par le niveau 1.

Programmation système : maintenance
les indicateurs de la maintenance corrective

L'objectif de la maintenance applicative est de gérer et faire diminuer le backlog. Ainsi, le temps de résolution d'un ticket est crucial. Il impacte directement l'image de marque de l'entreprise et sa capacité à faire du chiffre d'affaires.

Plusieurs indicateurs de temps (TAR : Time Answer Ratio) sont généralement utilisés dans la maintenance corrective :

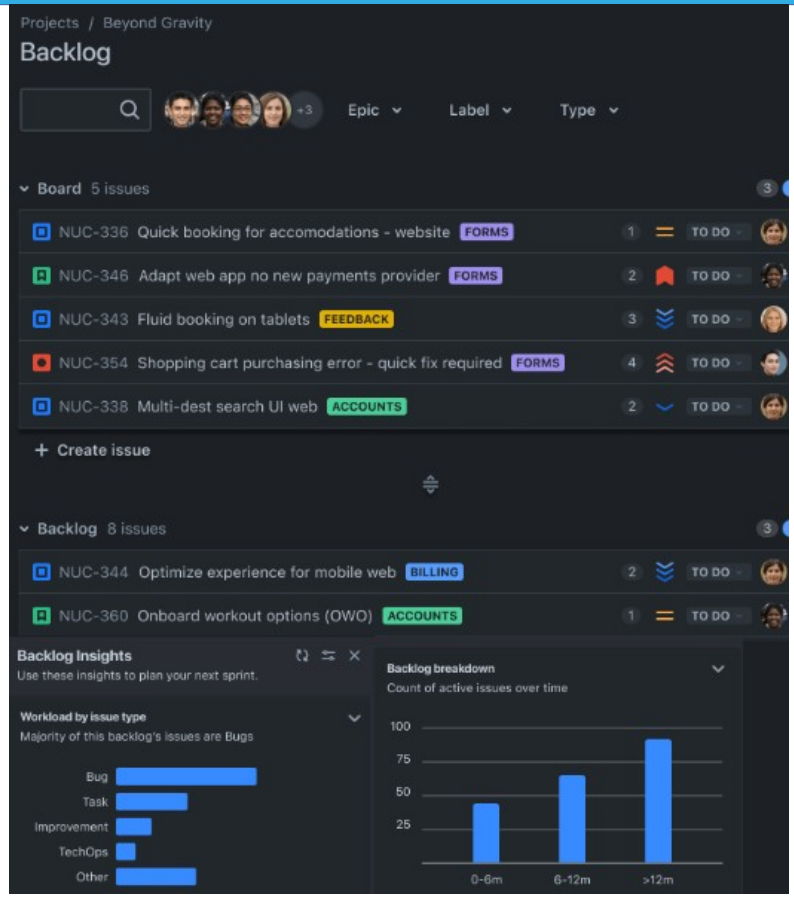
- TAR-3 : une première communication doit être faite par le L1 sous 3 j
- TAR-10 : une investigation doit être faite par le L2 sous 10 j
- TAR-20 : une résolution doit être apportée par le L3 sous 20 j

Le décompte en jours (j) est pris par rapport à la date de création du cas.

Les indicateurs de la maintenance évolutive ou adaptative ne sont pas les mêmes car le niveau d'urgence n'est celui de la maintenance corrective.

Programmation système : maintenance exemple d'outil Jira

- Jira est une solution commerciale, développée par l'entreprise Atlassian
- C'est un mélange de 2 outils : un outil de gestion de projet et un outil de gestion de ticket
- L'outil permet de gérer le cycle de vie d'un ticket ainsi que les différents indicateurs associés en affichant les informations dans des GUI.



Source : Atlassian

Programmation système : maintenance exemple Bugzilla

- Bugzilla est un logiciel libre de système de suivi de problèmes avec interface web, développé et utilisé par l'organisation Mozilla.
- Il permet le suivi de bogues ou de « demande d'amélioration » sous la forme de « tickets ».
- Logiciel de type serveur, il fonctionne en architecture trois tiers et nécessite un serveur web ainsi qu'une base de données.

Source : Linux Journal

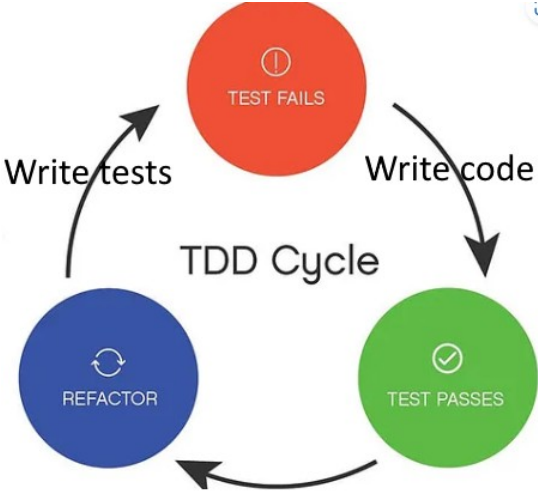
Thierry Vert

Programmation système : maintenance une modification de code et des tests

- Dans le cadre de la maintenance corrective, la modification d'un code ancien de plusieurs mois ou années est toujours délicat.
- Il est indispensable de s'assurer que la correction apportée ne « casse » rien et en même temps qu'elle corrige le cas d'utilisation problématique.
- Pour cela, il faut utiliser les tests existants selon leur type : unitaires, d'intégration et de non régression. On parle de débogage par les tests.
- Il faut également enrichir les tests avec des cas d'utilisation liés à la correction apportée. Les concepts du TDD s'appliquent bien pour une correction logicielle et les tests qui lui sont liés. En réalité le TDD n'est pas restreint aux corrections logicielles et peut être utilisé dans tout développement.

Programmation système : maintenance correction logicielle et TDD

- Le TDD est une méthode de développement logicielle pilotée par les tests (Test Driven Development). Elle consiste à penser aux cas d'utilisation en particulier les cas problématiques afin d'écrire les tests d'abord. Puis les scénarios de test sont utilisés pour écrire et implémenter le meilleur code possible.
- Cette approche privilégie des cycles de développement court, facilement gérable par les méthodes agiles. Elle assure que le logiciel est bien pensé avant d'écrire le code. Cette procédure augmente non seulement considérablement la qualité du logiciel, mais réduit aussi les frais de maintenance.
- Plus d'informations sur wikipedia.



Programmation système : maintenance
les tests unitaires en Python

- En Python unittest ou pytest peuvent être utilisés pour produire ou enrichir les tests existants.
- Ils offrent des fonctionnalités communes mais possèdent également des différences dans leur syntaxe.
- Unittest est installé dans les packages Python alors que le module pytest nécessite une installation supplémentaire.
- La découverte des tests et la syntaxe sont plus légèrement simples dans pytest plutôt que unittest.
- Leur utilisation repose sur un choix du développeur.

Programmation système : maintenance exemple de correction logicielle (1)

- Code existant avec une fonction add

"add_module_test.py"

```python
#!/usr/bin/python
import unittest
from add_module import add

class TestAdd(unittest.TestCase):
    def test_num_add(self):
        self.assertEqual(add(3, 5), 8, "La somme de 3 et 5 est 8")
    def test_char_add(self):
        self.assertEqual(add('a', 'b'), 'ab', "a et b agrégés en ab")

if __name__ == '__main__':
    unittest.main()
```

"add_module.py"

```python
#!/usr/bin/python
def add(a, b):
    return a + b

if __name__ == '__main__':
    print("5 + 3 =",add(3, 5))
    print("a et b agrégés en",add('a','b'))
```

- Bug : le paramètre a du add doit être >0 ou nul.
- Avec l'approche TDD, cas d'utilisation à tester : add(-5,1) devrait donner un résultat positif et add(-3,5) est équivalent à add(0,5)

Programmation système : maintenance exemple de correction logicielle (2)

On modifie le fichier de test, l'exécution retourne 2 erreurs sur 4 tests :

```python
#!/usr/bin/python
import unittest
from add_module import add

class TestAdd(unittest.TestCase):
    def test_num_param_a(self):
        self.assertFalse(add(-5,1)<0,"La somme est toujours positive")
    def test_num_param_a2(self):
        self.assertEqual(add(-3, 5), 5, "La somme de -3 et 5 est 5")
    def test_num_add(self):
        self.assertEqual(add(3, 5), 8, "La somme de 3 et 5 est 8")
    def test_char_add(self):
        self.assertEqual(add('a', 'b'), 'ab', "a et b agrégés en ab")

if __name__ == '__main__':
    unittest.main()
```

```
(base) C:\Users\thier\anacondacode\TDD>python add_module_test.py
..FF
======================================================================
FAIL: test_num_param_a (__main__.TestAdd)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\thier\anacondacode\TDD\add_module_test.py", line 7, in test_num_param_a
    self.assertFalse(add(-5,1)<0,"La somme est toujours positive")
AssertionError: True is not false : La somme est toujours positive

======================================================================
FAIL: test_num_param_a2 (__main__.TestAdd)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\thier\anacondacode\TDD\add_module_test.py", line 9, in test_num_param_a
2
    self.assertEqual(add(-3, 5), 5, "La somme de -3 et 5 est 5")
AssertionError: 2 != 5 : La somme de -3 et 5 est 5

----------------------------------------------------------------------
Ran 4 tests in 0.002s
```

On modifie le fichier add et on vérifie que <u>tous</u> les tests passent mais ...

```python
#!/usr/bin/python
def add_numbers(a, b):
    if a<0:
        return b
    else:
        return a + b

if __name__ == '__main__':
    print("1 + 5 =",add(1,5))
    print("a et b agrégés en",add('a','b'))
    print("-5 + 1 est positif",add(-5,1))
    print("-3 + 5 équivaut à 0 + 5 = ",add(-3,5))
```

```
(base) C:\Users\thier\anacondacode\TDD>python add_module_test.py
E...
======================================================================
ERROR: test_char_add (__main__.TestAdd)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\thier\anacondacode\TDD\add_module_test.py", line 13, in
    self.assertEqual(add('a', 'b'), 'ab', "a et b agrégés en ab")
  File "C:\Users\thier\anacondacode\TDD\add_module.py", line 3, in add
    if a<0:
TypeError: '<' not supported between instances of 'str' and 'int'

----------------------------------------------------------------------
Ran 4 tests in 0.002s

FAILED (errors=1)
```


Thierry Vert

Programmation système : maintenance exemple de correction logicielle (3)

- La correction n'est pas assez réfléchie et on a cassé l'utilisation des char dans add.
- On modifie add_module.py et tous les tests passent !

```python
#!/usr/bin/python
def add(a, b):
    if not(isinstance(a,str)) and a<0:
        a=0;
    return a + b

if __name__ == '__main__':
    print("1 + 5 =",add(1,5))
    print("a et b agrégés en",add('a','b'))
    print("-5 + 1 est positif",add(-5,1))
    print("-3 + 5 équivaut à 0 + 5 = ",add(-3,5))
```

```
(base) C:\Users\thier\anacondacode\TDD>python add_module_test.py
....
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
```