ARTRAGIS

Zeste de savoir

Mais pourquoi ça bug ?

12 août 2019

Table des matières

FIGURE 0. – Un bug a fait le buzz

Bonjour à tous ! Faut que je vous parle car là, on en a gros !

FIGURE 0. – Sire ! ça bug !

J'ai envie de vous le dire : ça ne marche pas !

Alors comment ce bug si embêtant a-t-il pu se glisser dans le logiciel ? Et par quelle magie les développeurs réussiront-ils à le corriger ?

Cet article parlera un peu de gestion de projet, un peu de technique, un peu d'outils pour faciliter la gestion et la correction de *bug*. L'article a aussi pour but de recueillir vos expériences en commentaires pour aider toutes les personnes qui se posent les deux questions ci-dessus.

1. Nous avons programmé

Notre logiciel a besoin d'une nouvelle fonctionnalité. Elle est primordiale cette fonctionnalité. Par exemple, tenez :

Comment les membres de Zeste de Savoir feront-ils pour savoir qu'un nouvel article a été publié ? Que quelqu'un a besoin d'aide dans le forum «Développement Web» ?…

1.1. V comme…

Alors on s'est mis à imaginer un système de **notifications centralisées** et parce qu'on était jeunes et insouciants, on s'est dit qu'il fallait qu'on prépare ça comme il faut pour qu'au moment de développer, ça soit dur comme du béton cette affaire.

Dans la vie de tous les jours, lorsqu'on se lance dans un projet, il est préférable de s'organiser un minimum. En informatique, les développeurs suivent ce même conseil.

Pour imaginer notre «*hub*» de notification, nous avons donc demandé aux utilisateurs ce dont ils avaient besoin.
Puis, on a **spécifié**[1] tout cela. Puis on a **codé**. Pour s'assurer que notre code n'était pas mauvais on a **testé unitairement**. Puis on a **qualifié**[2] tout ça. Enfin, on vous l'a présenté.

Dans le petit monde des gestionnaires de projets, on appelle ça le **Cycle en V** .

1. Cela signifie qu'on a écrit noir sur blanc *ce que le logiciel doit faire*. Plus tard la *conception* permet de définir *comment* le logiciel le fera.
2. On parle aussi de faire la *recette* du produit. C'est-à-dire qu'on regarde point par point s'il correspond aux besoins en se plaçant dans la peau d'un utilisateur et non plus d'un développeur.

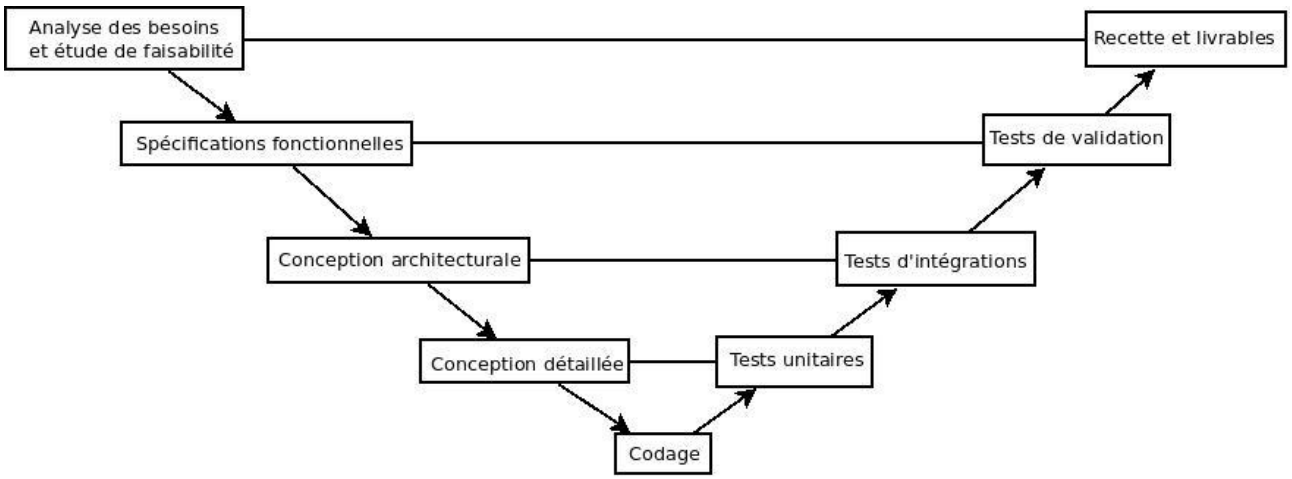

FIGURE 1. – Vous voyez, ça ressemble à un V[^source]

Comme vous pouvez le constater, on a bien une suite de tâches qui part du besoin utilisateur puis se rapproche de plus en plus de la technique. Une fois le développement fait, on s'éloigne de la technique pour voir ce qu'il se passe du point de vue de l'utilisateur.

Aujourd'hui, cette méthode n'est plus systématiquement utilisée – surtout en informatique – car elle est considérée comme trop peu *souple*.

Si l'équipe de ZDS avait choisi le cycle en V au départ (et les ZEP), c'était pour permettre à chacun de s'exprimer. Mais trop peu de membres non techniques s'exprimaient et trop de débats s'avéraient sans fin entre les développeurs, ce qui a mené à l'abandon de cette pratique pour la suite.

2. Ça a buggé

2.1. Mais du coup pourquoi ça a buggé ?

Nous avons donc développé notre logiciel. Au fur et à mesure, nous l'avons testé *unitairement*.

Tester un logiciel *unitairement* ça veut dire qu'on a pris toutes les briques qui composent le logiciel et qu'on les a testées une a une. C'est un moyen de savoir quelle brique a des défauts.

Car il faut savoir utiliser le bon vocabulaire :

— Lorsqu'on parle du bout de code qui génère un problème, on dit que c'est un défaut. Un défaut est inclus dans le produit. Parfois il est sans conséquence. Parfois, il en a.
— Les conséquences, quand elles arrivent sont appelées *défaillances* ou *bug*. Une défaillance peut être due à plusieurs défauts.

3. Image par ineumann sur http ://ineumann.developpez.com/tutoriels/alm/agile_scrum/

Quand un développeur *débug* son logiciel, il essaie de **provoquer une défaillance** pour ensuite **corriger le défaut**[4].

Tenter de provoquer une défaillance, c'est ce que fera la qualification[5].

Souvent le test consistera à regarder ce qu'il se passe quand :

— il n'y a aucune notification ;
— il y a une notification ;
— il y a plusieurs notifications (3 est un nombre sympa pour tester) ;
— il y a beaucoup de notifications (c'est-à-dire plus que la limite qu'on s'est imposé pour vous les afficher).

Pourtant, il est impossible de tout tester. Parfois parce que le logiciel est vraiment trop complexe pour penser à tout, mais le plus souvent parce qu'entre ce que le développeur pense que l'utilisateur va faire et ce que l'utilisateur fait, il y a une différence.

4. En fait, tous les tests fonctionnent comme ça. Mais ici le but est d'agir comme le ferait un utilisateur. Les tests unitaires tentent de provoquer une défaillance entre ce que le développeur essaie de faire dans sa *brique* et ce qu'il fait vraiment.

5. et je remercie pierre et firm1 qui savent trouver les défaillances du site durant les périodes de bêtatest. Soyez attentif aux messages de la zone dev si vous voulez faire comme eux.

FIGURE 2. – Un exemple très connu de cette différence

2.2. Mais ça bug quand même

Eh oui, il semblerait que tu aies trouvé un cas auquel nous n'avions pas pensé. De ce fait, le défaut est toujours dans le code et ni les tests unitaires, ni la qualification, ni les autres tests (je *tease* la troisième partie là) n'ont mis à jour de défaillance.

Pour les développeurs, ça sera ce moment qu'ils choisiront pour «ouvrir un ticket».

Cette phrase signifie qu'ils vont mettre «résoudre le bug» dans leur base de «tâche à faire». Mais ils ne vont pas le faire n'importe comment.

Ils vont devoir donner quelques informations :

1. Quelles sont les étapes pour en arriver là ?
2. Quel est le comportement observé ?
3. Quel est le comportement désiré ?

Ces trois informations sont primordiales. Si on ne connaît pas les étapes pour reproduire, on ne peut pas à notre tour provoquer la défaillance.

Si on ne sait pas quoi observer, on ne verra même pas la défaillance.

Si on ne sait pas ce qui est désiré, on ne saura pas comment retirer le défaut puisqu'a priori «tout sauf ce qu'il y a maintenant» est possible.

3. Et ce fut corrigé !

3.1. La correction passe aussi par le test

Armé de cette description, les développeurs ne vont pas tout de suite aller modifier le code pour corriger le défaut. Ils vont écrire un **test**. C'est-à-dire qu'il vont créer un programme qui va faire fonctionner le logiciel (dans notre cas créer des notifications) puis vérifier que le comportement est celui attendu.

Ce test sera ajouté à l'ensemble des tests qui sont déjà ajoutés dans le projet.

?

Mais du coup le test échoue, non ?

Eh oui, le test va échouer, il **faut** qu'il échoue. Cela permet de s'assurer qu'on reproduit la défaillance.

Une fois que le test a échoué, le développeur va probablement lancer un programme appelé **débuggeur**[6]. Grâce à lui, ils pourront parcourir le programme pas à pas tout en connaissant son état à tout moment.
Cela permettra de comprendre ce qu'il se passe pour s'assurer que le défaut sera totalement corrigé.

Élément externe (Video) —
Consultez cet élément à l'adresse `https://www.youtube.com/embed/VlaIeGhIeWM?feature=oembed`.
Un exemple de suivi pas à pas des *bugs*

---

6. Si vous avez l'habitude d'utiliser ces logiciels et que vous vous demandez comment ça fonctionne, je vous encourage à regarder cette conférence

3. *Et ce fut corrigé !*

Maintenant que le défaut est identifié et corrigé, on va lancer **tous les tests**. Cela permet deux choses :

— s'assurer qu'on a corrigé le défaut (car le nouveau test doit désormais réussir !) ;
— s'assurer qu'on n'a pas ajouté un défaut lorsqu'on a corrigé le défaut précédent.

Vous vous dites que c'est probablement long de lancer tous les tests vous-même ? **PAS DE PANIQUE** !

Il existe une classe d'outils appelée «intégration continue» (C.I. en anglais). Pour Zeste de Savoir, nous utilisons *Travis CI*, mais si vous êtes plutôt fan de framagit vous allez probablement utiliser gitlabci[7].

En tout cas ces outils permettent de :

— faire tourner les tests unitaires et de non régression (c'est-à-dire les tests qu'on a ajoutés pour s'assurer qu'on ne cassait pas ce qu'on a déjà réparé) ;
— faire tourner des outils qui analysent le code pour être certains qu'il sera lisible et bien formaté ;
— créer des **packages** – c'est-à-dire une sorte de «zip» qui va contenir le logiciel pour qu'on le déploie sur le serveur final/chez l'utilisateur.

Finalement, pour un développeur, comprendre et corriger un bug, est une activité qui tient plus de la méthode scientifique ou du diagnostic médical que de la magie.

Les tests sont alors le moyen le plus efficace pour les développeurs de s'assurer qu'ils ont corrigé le problème sans en créer d'autres.

Merci à Gabbro pour ses retours et à qwerty pour sa validation.

7. mangez-en, gitlabci permet aussi de créer des instances complètes pour tester vos modifications puis les éteints une fois que vous avez terminé les tests