R4.02 – Qualité de développement
Notes de cours 2

Semestre 4
2022/2023

1 Types de tests

1.1 Tests et phases de développement

La principale manière de classifier les tests se concentre sur les différentes phases de développement qui interviennent dans un projet. Le cycle en V est un modèle d'organisation de projet qui permet une lecture particulièrement claire des différents types de tests qui peuvent exister dans un projet. Ce modèle, illustré dans la Figure 1, organise les projets en différentes phases de définition (en partant des besoins clients, et en allant de manière de plus en plus détaillé vers la conception technique), suivie d'une phase de réalisation, et enfin différentes phases de tests. À chaque phase de conception est associée une phase de test ou de validation, pour contrôler que les choix faits en amont ont été respectés par la réalisation.

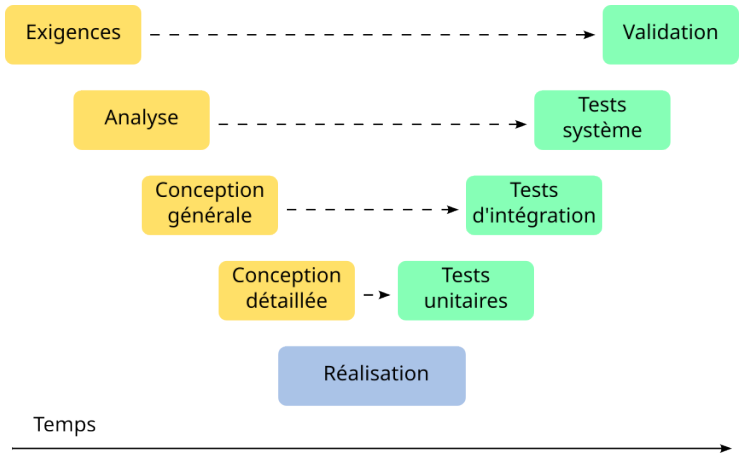

FIGURE 1 – Les étapes du cycle en V

Les différentes phases du cycle en V sont les suivantes :

— les **exigences** sont l'expression des besoins métier. La **validation** (aussi appelée tests d'acceptation) servira à contrôler que ces besoins sont satisfaits par le système livré.
— l'**analyse** vise à établir la spécification fonctionnelle du système. Les **tests système** vérifieront que le système correspond à cette spécification.
— la **conception générale** (ou conception architecturale) établit l'architecture du système (division en composants, et leurs relations). Du côté des tests, cela correspond aux **tests d'intégration** qui servent à vérifier que l'assemblage des différents composants fonctionne comme prévu.
— la **conception détaillée** définit chaque composant individuellement. Les **tests unitaires** permettront de vérifier que ces composants sont conformes à cette conception.
— la **réalisation** consiste à produire le système, c'est-à-dire principalement à coder.

On voit ainsi que différent types de tests interviennent à différentes étapes du projet, et contrôlent des propriétés différentes du système. Toutefois ces tests partagent aussi beaucoup de points communs, et grande partie des idées qui régissent la conception ou la mise en œuvre des tests sont applicables indifféremment à différents types de test.

On distingue toutefois la validation, qui n'est pas à proprement parler une phase de test. La validation sert à vérifier que le système répond aux besoins métier, c'est donc une activité qui demande une connaissance du domaine d'application, plus que du système. Par exemple la validation d'un logiciel de traitement du son devra être effectuée par des experts du domaine tels que des ingénieurs du son.

1.2 Boîte noire, boîte blanche

Une autre classification possible des tests, orthogonale à la première, est la distinction boîte noire/boîte blanche. Les tests en boîte noire sont conçus uniquement en se basant sur une description externe du système, telle que sa spécification ou sa documentation. Le système est "opaque", c'est-à-dire que le testeur n'a pas connaissance de ses détails internes. Un test en boîte blanche est au contraire conçu en s'appuyant sur ces détails internes, en plus de la spécification. Par exemple, si le testeur a accès au code source, il peut concevoir les tests de façon à ce qu'ils exécutent toutes les lignes de code.

Les tests en boîte blanche demandent une connaissance du système et sont donc plus complexes et coûteux à concevoir que les tests en boîte noire, toutefois l'information supplémentaire disponible permet de concevoir des tests qui sont plus susceptible de révéler les défauts. D'un autre côté, la méthode boîte noire garantit une meilleur découplage entre les tests et le système, et évite par exemple que les tests ne se concentrent que sur les cas prévus par l'implémenteur.

1.3 Tests fonctionnels et non-fonctionnels

Jusqu'à présent, nous surtout évoqué les tests fonctionnels, c'est-à-dire ceux qui vérifient les fonctionnalités du système à tester. Les tests non-fonctionnels sont liés aux exigences non-fonctionnelles qui décrivent la façon dont le système remplit son rôle. Pour le dire autrement, les tests fonctionnels essayent de répondre la question "qu'est-ce que le système fait ?", tandis que les tests non-fonctionnels s'intéressent à "comment le système le fait-il ?".

Il existe beaucoup de catégorie d'exigences non-fonctionnelles, et il est impossible de toutes les lister. Toutefois certains exemples sont : la performance, l'accessibilité, la sécurité, la fiabilité, la protection des données, l'efficacité, la résistance aux pannes, l'interopérabilité. Les exigences non-fonctionnelles à tester varient grandement suivant le domaine d'application : une application bancaire n'a pas du tout les même contraintes qu'un système embarqué. Il est important de cerner ces aspects lors de la phase d'élaboration des exigences, afin de pouvoir mettre en place les tests non-fonctionnels adaptés par la suite.

Les techniques de tests non-fonctionnels varient selon les propriétés à tester, et peuvent être assez différentes des techniques de test fonctionnels.

1.4 Tests de non-régression

Les tests de non-régression (parfois appelés tests de régression) ont pour but de vérifier que les fonctionnalités existantes d'un logiciel ne sont pas dégradées par des modifications dans le logiciel. De tels tests sont utiles dans le cadre d'un modèle de projet itératif, qui alternent plusieurs cycles de conception/réalisation. Ils peuvent aussi servir après la livraison, pour vérifier que des modifications ultérieures (patches de sécurité par exemple) n'ont pas d'effets négatifs sur la fonctionnalité du système. De fait, il est désormais de plus en plus courant pour un logiciel de continuer à évoluer après sa livraison initiale. Dans ce contexte, les tests de non-régression ont une importance croissante.

Lorsqu'un test de non-régression échoue, il est important de questionner le résultat. Il est possible qu'un défaut logiciel ait été introduit dans le système et qu'il doive être réparé, mais une seconde possibilité est que la spécification du système ait évolué. Dans ce cas il faudra modifier le test pour le mettre à jour.

Tous les types de tests mentionnés précédemment peuvent être amenés à être utilisés comme tests unitaires. Toutefois comme les tests de non-régression sont exécutés fréquemment et concernent tout le système, il est important de choisir un sous-ensemble de tests qui (i) puisse être exécuté en un temps raisonnable et (ii) ait la meilleure capacité à révéler des défauts. Dans la suite du cours, nous verrons des techniques pour accélérer les tests (notamment les *mocks* qui permettent d'éviter d'exécuter du code lent, tel que des accès à une base de donnée), et pour mesurer leur potentiel (les critères de couverture).

Encore plus que pour la phase initiale de tests, l'automatisation est cruciale pour les tests de non-régression. Ceux-ci sont généralement intégrés dans une procédure d'intégration continue qui construit et teste le système à intervalles réguliers (par exemple des *nightly builds*).

2 Le développement piloté par les tests

Le développement piloté par les tests (souvent abrégé en TDD, pour *test-driven development*) est une méthode de développement logiciel qui consiste à développer le logiciel de manière itérative, par étapes courtes.

1. Lors de chaque itération, on commence par spécifier la fonctionnalité à implémenter, par exemple sous forme de *user story*.
2. On transforme ensuite cette spécification en une suite de tests. À ce stade, les tests doivent échouer, puisque les fonctionnalités n'ont pas encore été implémentées. Seules les signatures des méthodes publiques auront été définies, et leur code se limitera au minimum requis pour permettre la compilation (par exemple *return null* si la méthode doit retourner un objet).
3. La phase d'implémentation vient ensuite : on choisit un test qui échoue et on écrit le code nécessaire pour le faire passer. Il s'agit pour l'instant d'écrire le code le plus simple qui fasse passer le test, des solutions inefficaces ou inélégantes sont donc acceptables.
4. Une fois que tous les tests passent, on peut refactorer le code, c'est à dire l'améliorer, tout en vérifiant que les tests continuent de passer.

Cette méthodologie est résumée par la phrase *red/green/refactor*, où *red* indique que le test doit d'abord échouer, puis passer suite à la première implémentation (*green*), avant de pouvoir améliorer le code (*refactor*).

Le TDD demande d'écrire les tests avant le code, ce qui peut s'avérer déroutant au premier abord. Cette méthode a pourtant plusieurs avantages :

— Les tests forcent les développeurs à se concentrer sur les fonctionnalités du code en priorité, et fournissent une mesure du travail accompli.
— L'écriture des tests n'est pas influencée par l'implémentation. Lorsque les tests sont écrits après le code, surtout si la même personne se charge des deux aspects, il y a un risque que les tests se limitent à vérifier que "le code écrit est le code écrit".
— Les tests permettent de mettre à l'épreuve rapidement les choix de conception. Par exemple, si il s'avère qu'une méthode est difficile à tester car elle prend en charge beaucoup de fonctionnalités, c'est un indicateur d'un problème dans l'architecture de l'application.

Toutefois le TDD a aussi ses limites. L'un d'elles est que la spécification décrit ce que le logiciel doit faire, et par conséquent les tests qui en découlent se concentrent sur les cas d'exécution normaux (*happy path*). De tels tests sont utiles pour vérifier que les fonctionnalités attendues sont là, mais peu adaptés à débusquer des erreurs qui surviennent en dehors de l'utilisation normale du logiciel (valeurs d'entrées inattendus, comportements exceptionnels, etc.).

3 Ressources supplémentaires

— Un article de blog (en français) qui compare le cycle en V à la méthode Agile, et le rôle du testeur dans ces deux approches :
https://www.atecna.fr/articles/testeur-agile-testeur-cyclev/

— Un article de blog (en français) sur l'importance des exigences non-fonctionnelles :
https://latavernedutesteur.fr/2020/01/13/limportance-du-non-fonctionnel-par-lexemple/

— La chaîne YouTube "Continuous Delivery" (en anglais) propose de nombreuses vidéos sur les pratiques de développement logiciel, notamment le développement piloté par les tests, par exemple cette introduction :
https://youtu.be/llaUBH5oayw