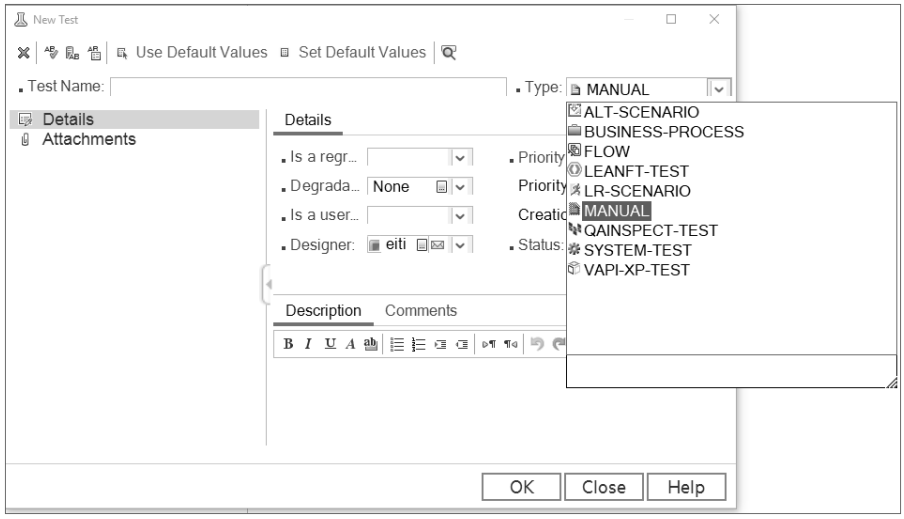
Chapitre 5
Organiser les tests

1. Les catégories de tests selon QC

1.1 Les types prédéfinis

Lorsque nous créons un test manuellement dans QC, l'interface nous propose dix types de tests différents (source : Guide Utilisateur ALM Quality Center 11.50, pages 401-402) :

- ALT-SCENARIO (non documenté - ne semble plus utilisé).
- BUSINESS-PROCESS et FLOW sont deux types utilisés pour les Business Process Test.
- LEANFT-TEST est un type utilisé pour l'automatisation sous Lean FT.
- LR-SCENARIO est un type de test pour LoadRunner, l'outil de test de performance.
- QAINSPECT-TEST est un type de test pour l'outil QAInspect, l'outil de test de sécurité.
- SYSTEM-TEST est un type de test système pour capturer le bureau et relancer la machine.
- VAPI-XP-TEST est un type de test créé sous Visual API-XP.
- Et surtout, MANUAL, le test manuel par défaut.



Remarque

Vous notez immédiatement que les types de tests intéressent surtout des outils spécifiques. Vous n'aurez probablement jamais à les utiliser. Seul le type MANUAL concerne vraiment le présent ouvrage.

1.2 Les niveaux de test : le rôle des cycles

Une caractéristique importante d'un test est son niveau. Mais que signifie cette notion concrètement ? Est-ce une caractéristique intrinsèque du test ? Malgré les apparences, nous devons répondre à cette question par la négative. Car un même test peut être exécuté à différents moments d'un projet pour des objets différents.

Par exemple, le test d'une fonctionnalité d'une application pourra être effectué sans mode opératoire précis par un développeur (ce faisant il réalise un test unitaire), le même test rédigé et exécuté par un testeur en fera un test de validation technico-fonctionnel, et enfin la même vérification réalisée par une MOA pourra correspondre à une Vérification d'Aptitude au Bon Fonctionnement – donc un acte contractuel.

Structurer les tests en niveaux est ici équivalent à raisonner en cycles au sens Quality Center. Il y a donc :

– **Les tests unitaires**, réalisés par les développeurs, pouvant ou non être formalisés dans un outil de test (dans notre cas, nous ne retenons pas cette pratique pour QC, mais rien n'empêche un CP MOE d'utiliser un autre outil ou simplement une feuille Excel).

- **Les tests technico-fonctionnels**, réalisés par notre équipe de test dédiée, dans le cas présent sous QC.
- **Les tests de Vérification d'Aptitude au Bon Fonctionnement (VABF)**, réalisés par une équipe MOA à des fins contractuelles principalement ou tout simplement d'homologation vers la Production.
- **Les tests Métiers**, visant à la bonne acceptation d'un applicatif par les futurs utilisateurs.
- **Les tests de préproduction**, principalement la Procédure Technique d'Installation.
- **Les tests de production**, s'il en est, pour conduire les vérifications qui n'auraient pu être réalisées dans les niveaux précédents.

À cette première catégorisation, nous ajoutons un sous-niveau pour les tests technico-fonctionnels réalisés par une équipe de test :

- Les tests d'environnement (dont les installations)
- Les tests des applications :
  - Les tests d'intégration des données entrantes
  - Les tests d'IHM
  - Les tests d'export des données sortantes
  - Les tests fonctionnels
  - Les tests techniques des traitements automatiques
- Les tests d'interfaces :
  - Les tests techniques des échanges entre les bases de données
  - Les tests de processus Métier
- Les tests de performance

Ce sous-type est lui aussi décliné selon l'ordonnancement chronologique des tests, celui-ci étant établi selon une priorisation spécifique. Ainsi les tests d'installation sont-ils toujours nécessairement les premiers. Ils sont suivis par les tests des produits, eux-mêmes hiérarchisés par interdépendance : il est en effet difficile de tester des IHM ou des fonctionnalités avec une base de données vide. Ainsi, les tests d'alimentation des bases de données sont-ils préférentiellement effectués avant tous les autres.

De la même manière, les exports ne sont testables qu'à partir d'une base alimentée pour laquelle on s'est assuré par les tests des IHM que les données ne sont pas altérées.

S'en suivent les tests fonctionnels puis les traitements automatiques réalisables, car les sous-niveaux précédents garantissent un meilleur contexte de validation.

Chaque application étant testée individuellement, il devient alors possible de vérifier d'une part les échanges entre elles, et d'autre part, que celles-ci s'insèrent correctement dans un processus Métier où elles interviennent : les tests d'interface suivent donc par nécessité.

Nous terminons par les tests de performance réalisés sur un système globalement stable s'il a passé les précédents sous-niveaux.

Remarque

Il est entendu que cette classification est théoriquement applicable à chacun des niveaux de test proposés en début de section. Dans la pratique, il est évidemment souhaitable que toute la couverture ait été vérifiée bien en amont de la Production, donc de manière privilégiée lors des tests technico-fonctionnels qui suivent immédiatement les tests unitaires.

1.3 Typologies que nous traiterons

QC n'a donc pas une typologie de test au sens du processus d'ingénierie logicielle, mais une batterie de tests techniques dépendant d'un éventail d'outils trop spécifiques.

Notre besoin de typologie de test n'est pas exprimé par les champs standards de QC. Il est :

– d'exprimer si un test est obsolète ;
– de pouvoir définir un test comme appartenant à un périmètre de non-régression priorisé ;
– de pouvoir cataloguer un test comme ayant un mode opératoire compréhensible par un utilisateur final ;
– d'indiquer si un test fait l'objet d'une dégradation applicative introduisant une limite de validité de son résultat (et l'expression d'une nouvelle priorité alors).

1.3.1 Les tests obsolètes

Le test dans QC est un objet qui dispose notamment d'un champ **Plan Status** qui permet de gérer l'état de rédaction d'un test.

Il ne dispose pas cependant d'une valeur permettant d'en gérer l'obsolescence par défaut à l'installation de l'outil.

Nous l'avons vu dans le chapitre précédent, une exigence produit dans QC peut représenter un élément de l'application qui en est retiré à un moment ou un autre de la vie du logiciel. Si l'exigence peut devenir obsolète, logiquement les tests qui lui sont associés aussi.

Dès lors, nous nous devons de traiter l'obsolescence des tests : pour cela nous proposons dans les sections suivantes la customisation et le workflow qui le permettent.

1.3.2 Les tests de non-régression priorisés

De la même manière, QC ne propose pas de gérer les tests de non-régression. Nous l'avons aussi évoqué dans le chapitre précédent lors de la détermination d'une stratégie de test. Mais qu'est-ce qu'un test de non-régression ? Et qu'appelle-t-on sa priorisation ?

Il ne s'agit pas de la vérification d'une évolution, mais d'une garantie : si toute modification d'une partie d'un système peut être vérifiée, la partie non modifiée de ce système peut alors être altérée éventuellement. Car par définition, un système n'est pas cloisonné, mais en interaction.

Au-delà de cela, la modification d'un élément peut aussi accidentellement entraîner la modification d'un autre élément dès lors que du code peut être factorisé entre les deux. D'un point de vue du testeur, ce particularisme peut (doit ?) être ignoré. Il convient donc de définir un périmètre supplémentaire de tests dans un projet pour prévenir une possible régression.

Notre démarche prévoit donc dans le dossier de stratégie un chapitre consacré à ce périmètre.

Toutefois, cette garantie n'est jamais que l'expression d'une sécurité et elle a un coût. Quel budget peut-on alors mettre ?

Raisonnons par l'absurde et supposons que nous soyons contraints d'exécuter la totalité des tests du référentiel à chaque fois : la charge devient exponentielle et n'est donc pas viable financièrement.

Il nous faut alors réduire les tests de non-régression à un sous-périmètre acceptable d'un point de vue coût. Cependant, nous notons alors que nous avons de petites évolutions dont la charge de test est faible : devrait-on alors jouer tous les tests identifiés de non-régression ? Là aussi la viabilité financière ne sera peut-être pas au rendez-vous.

Le moyen le plus pragmatique est donc de définir une priorité pour subdiviser les tests de non-régression en plusieurs périmètres successifs et additifs pour gérer cet aspect.

La norme MoSCoW proposée par l'International *Software Testing Quality Board* (ISTQB) permet cela. La priorisation des tests en MUST, SHOULD, COULD et WOULD déjà évoquée précédemment, autorise la construction de quatre périmètres progressifs de tests de non-régression. Le but principal de cette hiérarchisation est de créer une échelle budgétaire dans les tests.

Dès lors, nous pouvons considérer que les tests du référentiel estampillés "Test de Non Régression" et de priorité "MUST" constituent le ticket d'entrée d'un projet de test, le minimum syndical pour sécuriser la modification d'un système.

Mais comment gérer le cas où ce budget est encore trop élevé ?

Bien entendu, cette situation est problématique : elle dépend du rapport entre la charge de test des évolutions et celle de la non-régression, mises en perspective de la criticité de ces dernières.

Un changement peu critique, mais coûteux en test est un cas limite demandant une approche plus précise. Dès lors, le périmètre des TNR sera réduit à sa plus simple expression.

Remarque

Le chapitre 3.4.8 de notre modèle de dossier de stratégie proposé a pour rôle la définition d'un périmètre de TNR par défaut sur la base de cette définition de notre ticket d'entrée. Le CP Test adapte alors dans ce chapitre une proposition qui est faite à la MOA, qu'elle validera ou non. Ce chapitre est probablement le plus important du Dossier de Stratégie puisqu'il établit une base de négociation, un compromis risque/charge.

1.3.3 Nature d'un mode opératoire : testcase et usercase

Le mode opératoire d'un test est la description de son déroulé, conceptualisé traditionnellement dans tous les outils de test par une matrice de quatre colonnes :

- Le numéro de ligne, ou pas, ou *step* en anglais.
- L'action de test qui peut être :
  - Une interaction avec l'application testée,
  - Une action de test avec un assistant de test, un outil externe permettant d'effectuer une vérification,
  - Une observation, type d'acte souvent oublié alors qu'indispensable.
- Un résultat attendu ;
- Un résultat de test valant :
  - Vide si le step n'a pas été exécuté,
  - OK, si le step a été exécuté et ne remonte aucun écart entre résultat attendu et résultat observé,
  - KO, si le step a été exécuté et remonte un écart,
  - N/A, pour "non applicable" si le step n'a pas pu être exécuté, car le contexte rend le test non adapté pour son exécution.

QC n'échappe pas à la règle et propose un onglet facultatif **Design Steps** pour remplir ce rôle.

Remarque

Notons qu'en l'absence de mode opératoire un tel test est nommé "sondage". Il est alors exécuté sur la seule base de l'exécutant qui, à la seule évocation du nom du test, comprend ce qu'il doit faire.

Cependant, nous pouvons dégager deux familles de modes opératoires : ceux qui sont à portée d'un utilisateur final et ceux qui sont destinés à être déroulés par des experts en test, car ils contiennent un facteur de technicité spécifique.

Le terme générique en anglais pour désigner le mode opératoire d'un test est "testcase", notion d'ailleurs non exprimée dans QC qui emploie le terme générique "test". Le mot "usecase" correspond alors à un testcase considéré d'un point de vue utilisateur. Nous pourrions le traduire par "cas d'utilisation". Nous précisons ici ces définitions pour deux raisons : d'abord parce que les confusions entre les deux termes sont fréquentes, ensuite parce que QC ne les distingue pas.

Notre customisation propose de pallier cela.

Remarque

L'intérêt pour une équipe Test de distinguer les angles de vue est faible à moins de considérer les tests comme des éléments exportables ou partageables avec le Métier, soit à des fins d'utilisation, soit plus simplement à des fins de validation.

1.3.4 Dégradation applicative et tests

Lors du test d'un système, ses conditions idéales d'exécution devraient être à l'image de la production pour être valide.

Cependant, cette capacité de l'environnement de test n'est parfois pas remplie. Le testeur se retrouve alors confronté à trois possibilités :

- La fonction à tester n'est absolument pas disponible.
- La fonction à tester est disponible, mais s'appuie sur des données fictives : il n'y a pas réellement de base de données derrière le traitement. Cette situation fait alors appel à des bouchons, c'est-à-dire des données statiques permanentes se substituant aux données réelles que l'on trouverait dans un environnement de production.
- La fonction à tester est disponible, permet l'accès aux données en base, mais un traitement automatique (un batch) est absent de l'environnement : un outil spécifique, nommé un simulateur, permet de modifier manuellement les données pour "faire croire" à l'application que le traitement a été déroulé.