Enjeux économiques et techniques des métiers du test

Dr-Ing Pierre-Emmanuel Gros
Responsable Pôle Innovation Neuresys

Pierre-emmanuel.gros@neuresys.fr

NEURÉSYS

Enjeux économiques et techniques des métiers du test

- Historique et définitions
- Opportunités du test aujourd'hui
- Eléments clés d'une validation fonctionnelle
- Coûts et rentabilité du projet de test
- Les approches du test aujourd'hui
- Outils et Infrastructure de test Agile
- Les solutions de gestion des tests
- Synthèse

Enjeux économiques et techniques des métiers du test

- Historique et définitions
  - Définitions. Rôle dans le processus de développement logiciel. Historique des approches.
  - Tests dans le cycle de vie du logiciel : modèle BOEHM.
  - Rôles sur le projet (MOA, MOE, équipe de recette, utilisateurs). Différents processus.
  - L'Independent V&V. Revues, inspections...
  - Familles de tests : unitaires, fonctionnels, cohérence en base, intégration, charge.
  - Tests de non-régression.
  - Les livrables (cahier des charges, spécifications).
- Opportunités du test aujourd'hui
- Eléments clés d'une validation fonctionnelle
- Coûts et rentabilité du projet de test
- Les approches du test aujourd'hui
- Outils et Infrastructure de test Agile
- Les solutions de gestion des tests
- Synthèse

Historique et définitions

Définitions. Rôle dans le processus de développement logiciel. Historique des approches.

- Définitions des notions de projets/anomalies/tests
- Montrer les différents rôles dans le processus de développement logiciel
- Donner quelques types d'approches

Des définitions multiples

- La définition de ce qu'est un projet et une anomalie est la problématique de la qualité logicielle
- Chaque personne/entreprise possède sa définition de la qualité logicielle, des acteurs de cette qualité logicielle et cette disparité provient de l'historique et de la manière de faire des entreprises

Qu’est-ce qu’un projet?

Si un logiciel est une réponse technique à un besoin.

Alors un logiciel implique plusieurs projets:

- Le chantier de réalisation ou « release » / « build »
- Le chantier de maintenance ou « run », avec des sous éléments
  - Un chantier de création de patch
  - Un chantier de maintenance technique/fonctionnel
  - Un chantier d’évolution

Qu’est-ce qu’un projet?

Les projets se découpe en

- Avant-projet
  - Etude d’opportunité: Est-ce que le projet est opportun?
  - Etude d’impact: Est-ce que ce dernier engendre un cout?
  - Choix de solutions: Est-ce que le projet est faisable et si oui comment?
- Création
  - Conception : Comment le projet doit être fait?
  - Réalisation : Instanciation du projet
- Recette
  - Technique: Est-ce que techniquement le projet fonctionne?
  - Fonctionnel: Est-ce que fonctionnellement le projet fonctionne?
  - Non fonctionnel: Est-ce que ce dernier est utilisable?
- Après projet
  - Bilan: Est-ce que le projet répond au besoin?
  - Clôture du projet : Fermeture du projet

Qu'est-ce qu'un test?

- Si la validation est la vérification du respect d'un logiciel au regard du besoin.
- Alors cette validation est exécutée pendant une phase de recette.
- Cette validation peut se faire via un outil appelé « test ».
  - Ce n'est pas le seul outil. Un logiciel peut ne pas être « validé » parfaitement (ex IA) ou valider par preuve (ex ligne automatique 14 en France)

Qu'est-ce qu'un test?

Notion intuitive définit par la norme IEEE 829-1998

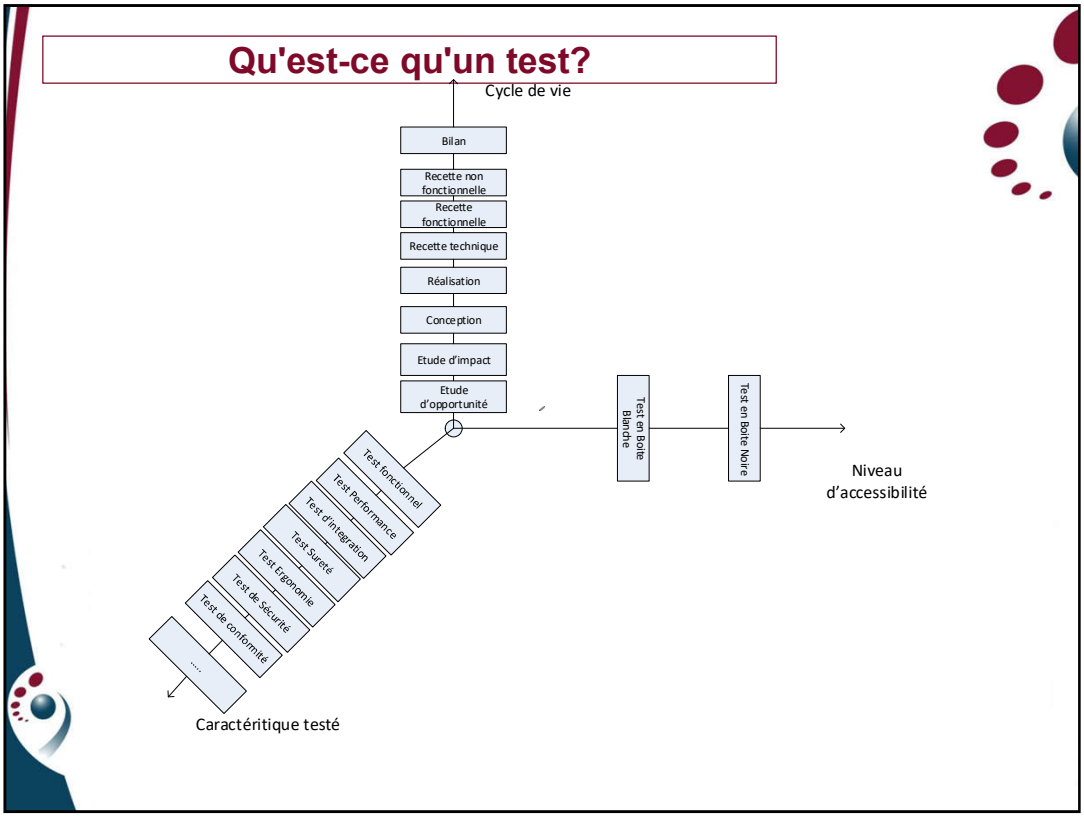
« Le test est **l'exécution ou l'évaluation** d'un système ou d'un composant, par des moyens automatiques ou manuels, pour vérifier **qu'il répond à ses spécifications** ou identifier les différences entre les **résultats attendus et les résultats obtenus**. »

Ceci implique :

- Validation dynamique d'un programme
- Comparaison entre les résultats attendus et les résultats obtenu

Si le résultat attendu est identique au résultat obtenue le test est dit positif, sinon on parle d'anomalie

Qu'est-ce qu'un test?



Qu'est-ce qu'un test?

- Tester peut révéler la présence d'erreurs mais jamais leur absence
  - Vérification partielle: le test ne peut pas montrer la conformité du système

- Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts»
  - Comment : Savoir ce que devrait faire le logiciel
  - Qui : par des personnes pouvant manipuler le logiciel et comprendre le besoin.
  - Quand : en fonction de la CQFD (calcul des couts opportuns de faire un test).
  - Pourquoi : détection des anomalies de fabrication

Qu'est-ce qu'un test?

- Ces définitions sont limitées car le test permet aussi:
  - De mener un projet de Agilité
  - De partager l'information entre les développeurs (technique) et les testeurs (plus proches du fonctionnel)
  - Représente *in fine* les spécifications réelles du logiciel.
- Une bonne définition d'un test est :
  - Un test est une exécution sur un jeu de donnée précis avec une méthodologie
  - Un test a un but qui permet de vérifier ou de valider une fonctionnalité
  - La qualité logicielle est le marqueur de la bonne/mauvaise avancé d'un projet.
- Enfin, la qualité logicielle est un arbitrage budgétaire impliquant soit un surcout (test inutile) soit un gain de cout (aide aux développeurs, « early bug avoidance », aide à la documentation, validation de l'usage), mais c'est surtout une méthode de bonne gestion de projet.

Qu’est-ce qu’un bug/anomalie?

- Il s’agit du différentiel entre un résultat attendu et un résultat obtenu.
- Les anomalies peuvent être
  - Bloquantes pour l’utilisation d’un logiciel
  - Acceptables car n’empêchant pas une utilisation normal
- Leurs effets peuvent être
  - Triviaux: le logiciel est bloqué, l’usage du logiciel s’arrête
  - Moins Triviaux : le logiciel est bloqué, mais des contournements existent
  - Non triviaux: le logiciel a un comportement sporadique, les résultats sont cohérents mais faux

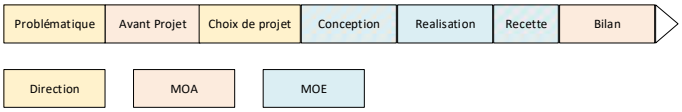
Qu’est-ce qu’un bug/anomalie?

- Une anomalie est caractérisée par plusieurs couts:
  - Cout pour détecter l’anomalie
  - Cout pour résoudre l’anomalie
  - Valeur négative de la valeur d’usage du logiciel
  - Cout d’image pour la société
- Et permet d’avoir de la valeur
  - Justification d’un chantier de maintenance
  - Anomalie en moins (syndrome de l’arbre « bug » qui cache la forêt de « bug »).
  - Non-respect d’une norme contraignante
  - Approche nouvelle/fonctionnalité étendue d’un logiciel

Différents Rôles

- Les différents projets impliquent la question du ratio valeur/cout. C'est cette question qui est posé par la direction.
- La MOA (Maitrise d'Ouvrage) a pour rôle de conduire l'avant-projet, les études d'impacts/opportunité, co-créer le logiciel avec la MOE et conduire les tests fonctionnels et de validation.
- La MOE (Maitrise d'Œuvre) a pour rôle de Co-créer le logiciel avec la MOA, de conduire les tests techniques et de faire la mise en production.

Différents Rôles



- MOA Direction
  - Commanditaire
  - Le Client
  - Le comité directeur (moyen et gros projet)
  - Le chef de projet
- MOE
  - L'équipe projet
  - Les experts
  - Le planificateur
  - L'organisateur
  - Le contrôleur
  - L'innovateur
  - L'investigateur

Différentes approches historiques du test

- Pas de test : Peu de cout sur la qualité logicielle, cout sur la maintenance
- 1980: Boris Beizer décrit le test en boite noire, définition des cas de tests et de la couverture de code
- 1990: Utilisation des bugtrackers, version code control, et définition claire des tests de régressions
- Fin 1990: Introduction du TDD, Customer driven test, Performance testing, Exploratory testing
- Début 2000: Intégration continue, méthode Agile
- Fin 2000: Acceptance test driven développement, Agile Testing

Différentes approches historiques du test

- Ces approches ont pour but de valoriser la qualité logicielle en la mettant au cœur du développement.
- Il s'agit pour la plupart du temps de la partie technique de la qualité logicielle.
- Les user story des méthodes Agiles permettent d'utiliser la qualité logicielle pour marquer les avancements d'un projet dans sa complétude (technique, fonctionnel et usage).

Historique et définitions

Tests dans le cycle de vie du logiciel : modèle BOEHM

- La question est de savoir quand est ce qu'on peut commencer les tests.
- Une réponse est le modèle de Boehm

Tests dans le cycle de vie du logiciel

- L'évaluation de la qualité d'une tache (Etude de besoins, Etude d'impacts, développement …) ne peut commencer qu'après la réalisation de cette tâche.
- La durée d'une tache est essentiel afin :
  - De ne pas réserver de ressources trop tôt.
  - Planifier la réservation des ressources critiques
- Une idée possible est d'utiliser l'outil statistique

Modèle de Boehm

- Conçu en 1981 par Barry Boehm, COCOMO est une méthode basée sur les résultats de 63 projets de développements informatiques (allant de 2 000 à 100 000 lignes de code).
- COCOMO (acronyme de l'anglais COnstructive COst MOdel) est un modèle permettant de définir une estimation de l'effort à fournir dans un développement logiciel et la durée que ce dernier prendra en fonction des ressources allouées.
- **Le résultat de ce modèle n'est qu'une estimation**, il n'est en rien infaillible et parfaitement exact.

Modèle de Boehm

3 sous modèles:

- Le *modèle de base* effectue un simple calcul de l'effort et de la durée en fonction du nombre d'instructions que l'application doit contenir et la complexité de cette dernière. Une ventilation est également possible, permettant de déterminer le temps de développement et l'effort nécessaire pour chaque partie du cycle de développement.
- Le *modèle intermédiaire* reprend l'effort et la durée du modèle de base, en appliquant cette fois-ci des coefficients prenant en compte des facteurs de coût (compétence de l'équipe, complexité de l'environnement technique, etc.).
- Le *modèle détaillé* reprend les données du modèle intermédiaire en affinant notamment les facteurs de coût en fonction de chaque phase du cycle de développement. Ce modèle n'est véritablement nécessaire que pour de très gros projets.

Modèle de Boehm

3 types d'applications modélisées:

- *S* (en anglais *organic*) : Ce sont des applications simples, n'ayant que peu de cas particuliers et de contraintes. Elles sont parfaitement déterministes.
- *P* (en anglais *semidetached*) : Ce sont des applications intermédiaires, plus complexes que les applications de type S, elles restent tout de même déterministes, bien que le nombre de cas particuliers et de tests doivent être plus important que pour les applications de type S
- *E* (en anglais *embedded*) : Ce sont des applications très complexes, que ce soit au niveau de leurs contraintes (comme un système temps réel) ou au niveau des données saisies (comme certaines interfaces graphiques où l'on ne peut envisager toutes les possibilités de saisies qu'un utilisateur pourrait effectuer). Elles ne sont pas déterministes.

Modèle de Boehm

- Complexité:
  - S : Effort=2.4*KLS^1,05, Tdev=2,5*Effort^0,38
  - P : Effort=3*KLS^1,12, Tdev=2,5*Effort^0,35
  - E : Effort=3.6*KLS^1,2, Tdev=2,5*Effort^0,32
- Le plus complexe est de déterminer le nombre de KLS. À première vue, on peut se dire que c'est une chose impossible ou avec une très grande marge d'erreur. Cependant, pour être valable le modèle COCOMO ne doit être utilisé que lorsque la phase de conception est déjà bien avancée, de manière à avoir une idée assez précise du travail à réaliser. De plus, l'expérience de la personne utilisant le modèle est déterminante, car il sera ainsi en mesure de s'approcher au plus près du bon nombre de KLS

Modèle de Boehm

- Coefficient du modèle « simple », la recette occupe 19 à 25% de l'activité.

		Taille				
Compléxité	Phase	2 KL	8KL	32KL	128KL	512KL
S	Expression des besoins	6	6	6	6	
	Conception	16	16	16	16	
	Realisation	68	65	62	59	
	Conception détaillée	26	25	24	23	
	Programmation	42	40	38	36	
	Recette	16	19	22	25	
P	Expression des besoins	7	7	7	7	7
	Conception	17	17	17	17	17
	Realisation	64	61	58	55	52
	Conception détaillée	27	26	25	24	23
	Programmation	37	35	33	31	29
	Recette	19	22	25	28	31
E	Expression des besoins	8	8	8	8	8
	Conception	18	18	18	18	18
	Realisation	60	57	54	51	48
	Conception détaillée	28	27	26	25	24
	Programmation	32	30	28	26	24
	Recette	22	25	28	31	34

Modèle de Boehm

- Ces modèles statistiques doivent être pris avec beaucoup de précaution car :
  - Ne prennent pas en compte la culture d'entreprise
  - Nécessite une conception forte avant de pouvoir être appliquer
- Néanmoins, il montre que l'effort de test est fonction du nombre de module intégré (facteur exponentiel) et de la compétence des équipes (facteur linéaire)
- Exemple de modèle http://groups.engin.umd.umich.edu/CIS/course.des/cis525/js/f00/gamel/cocomo.html

Historique et définitions

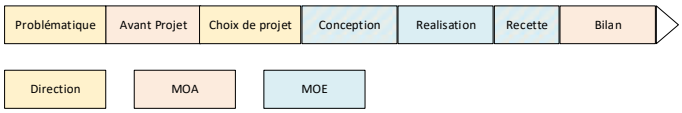
Rôles sur le projet (MOA, MOE, équipe de recette, utilisateurs). Différents processus.

- Qu'est-ce que la MOA, MOE, équipe de recette
- Comment ces équipes interviennent entre elles.

Différents Rôles

- Les différents projets impliquent la question du ratio valeur/cout. C'est cette question qui est posé par la direction.
- La MOA (Maitrise d'Ouvrage) a pour rôle de conduire l'avant-projet, les études d'impacts/opportunité, co-créer le logiciel avec la MOE et conduire les tests fonctionnels et de validation.
- La MOE (Maitrise d'Œuvre) a pour rôle de co-créer le logiciel avec la MOA, de conduire les tests techniques et de faire la mise en production.

Différents Rôles



- MOA Direction
  - Commanditaire
  - Le Client
  - Le comité directeur (moyen et gros projet)
  - Le chef de projet
- MOE
  - L'équipe projet
  - Les experts
  - Le planificateur
  - L'organisateur
  - Le contrôleur
  - L'innovateur
  - L'investigateur

Rôles sur le projet.

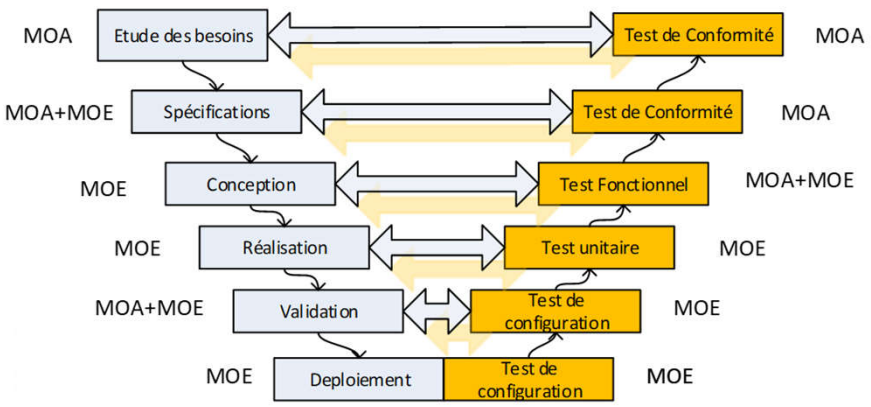
Parmi les acteurs, il faut distinguer ceux qui recherchent les défauts pour les corriger

- Le chef de projet (MOA+MOE)
  - Planification des tâches et assurance qualité (système qualité) – Organisation de l'équipe – Mise en œuvre de la stratégie et des méthodes
- L'architecte du projet (au sens large = expression de besoin, spécification fonctionnelle, conception implique la MOA et/ou le client) (MOE)
  - Architecture testable
- Le responsable de la qualification indépendante et son équipe (Assurance Qualité – Inspections et revues – Recette) (MOA+MOE)
- Le support et/ou la maintenance de 1er niveau (MOA)

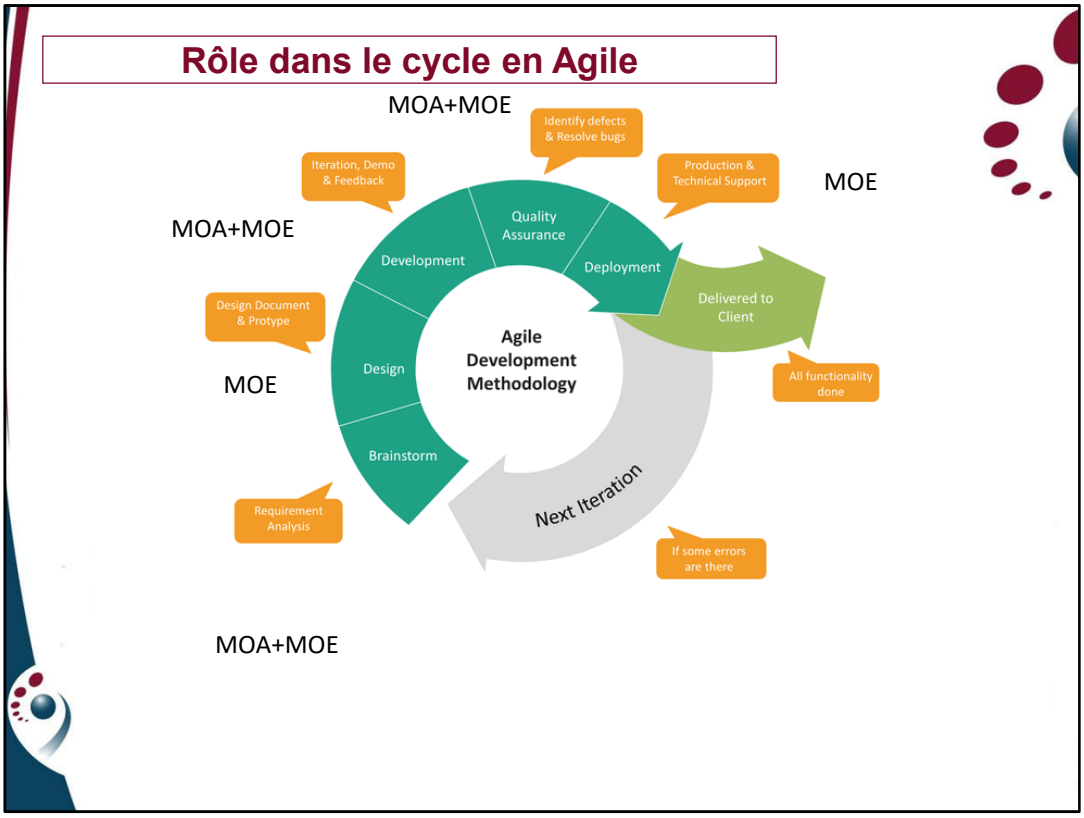
Et ceux qui ceux qui introduisent les défauts:

- Les programmeurs (MOE)
  - Conception détaillée et programmation – Composants logiciel {Données Algorithmes +Contrôles} intégrables (i.e. documentés et testés)
- Le responsable de l'intégration et son équipe (MOE)

Rôle dans le cycle en V

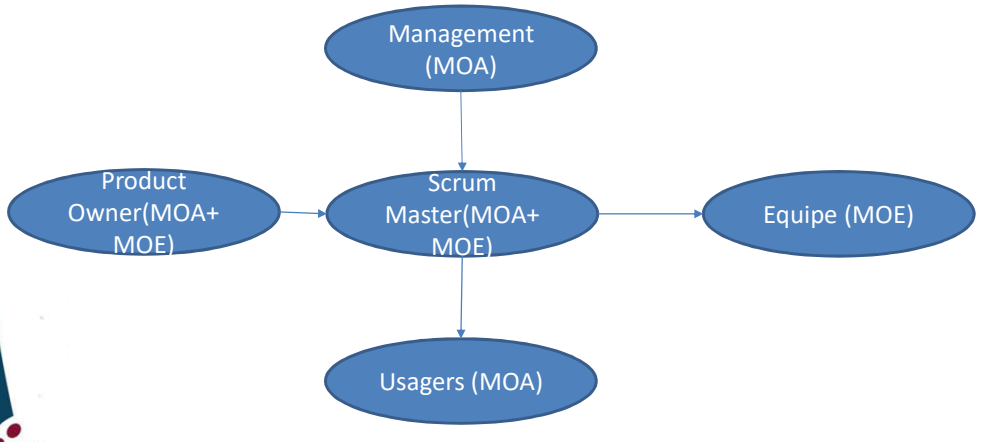


Rôle dans le cycle en Agile



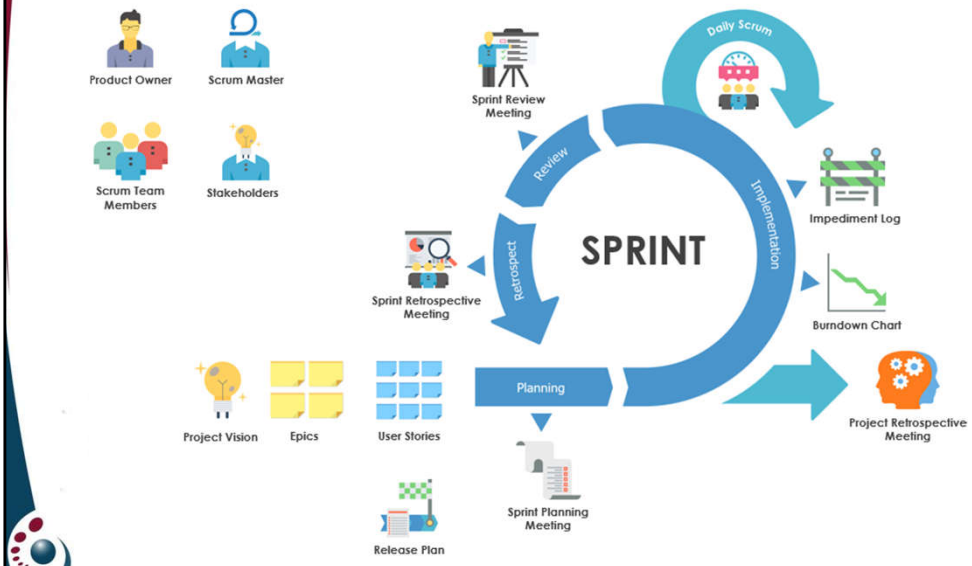
Rôle dans les méthodes ScrumXP

Les interactions correspondent aux processus de livraisons centrées sur le scrum master.



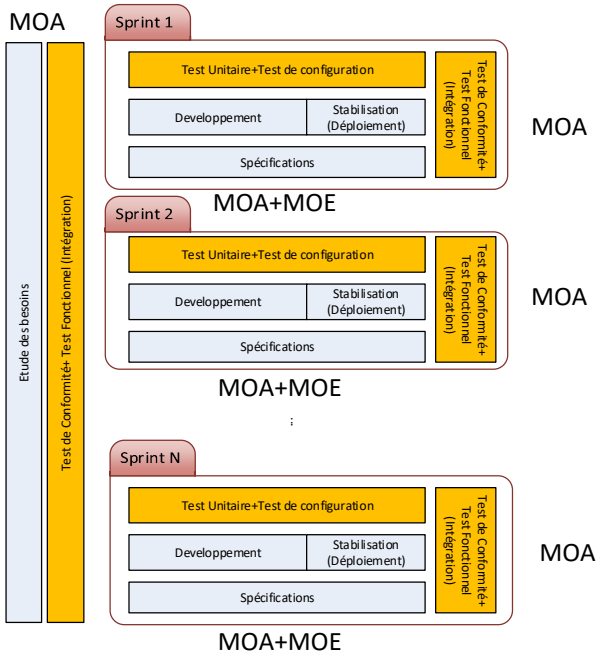
ScrumXP

The Agile – Scrum Framework



Rôle dans le mode Agile Editeur

NB: Dans le modèle la MOA est experte!!



Historique et définitions

L'Independent V&V. Revues, inspections...

- Qu’est que le V&V
- Que sont une revue, une inspection et une validation?

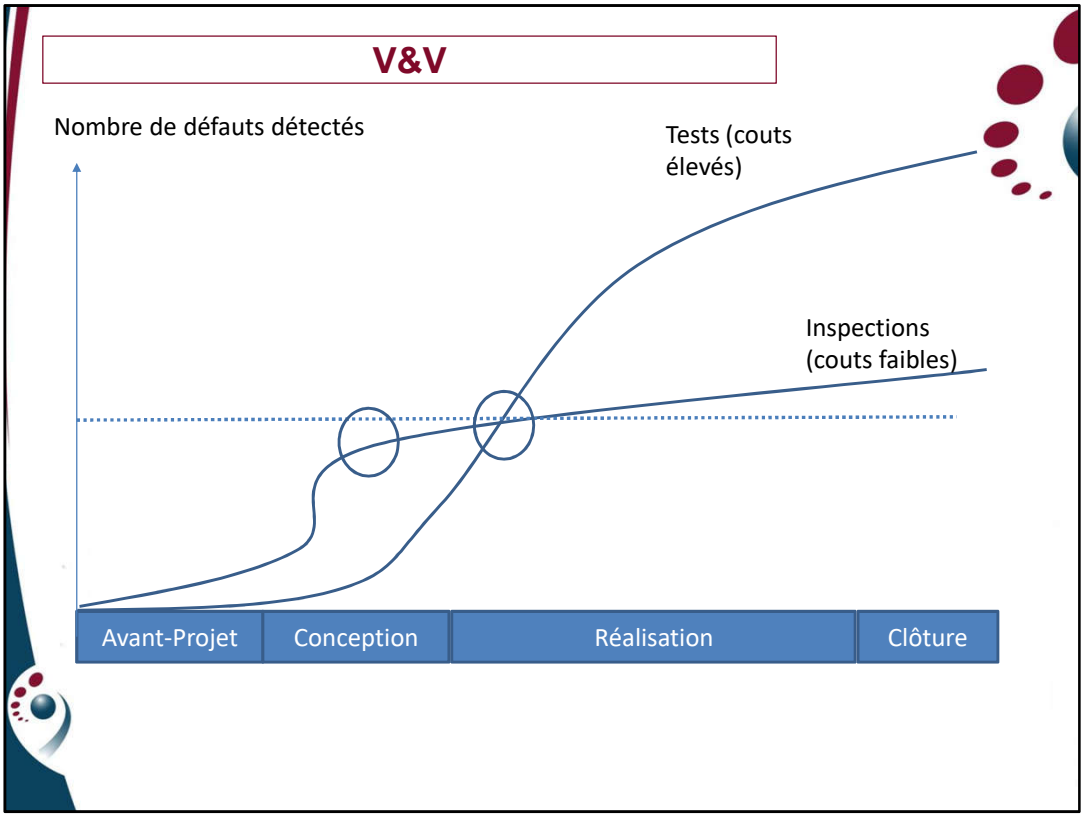
V&V

- Vérification : confirmation par l'examen et la fourniture de preuves objectives que des exigences spécifiées ont été remplies [ISO 9000].

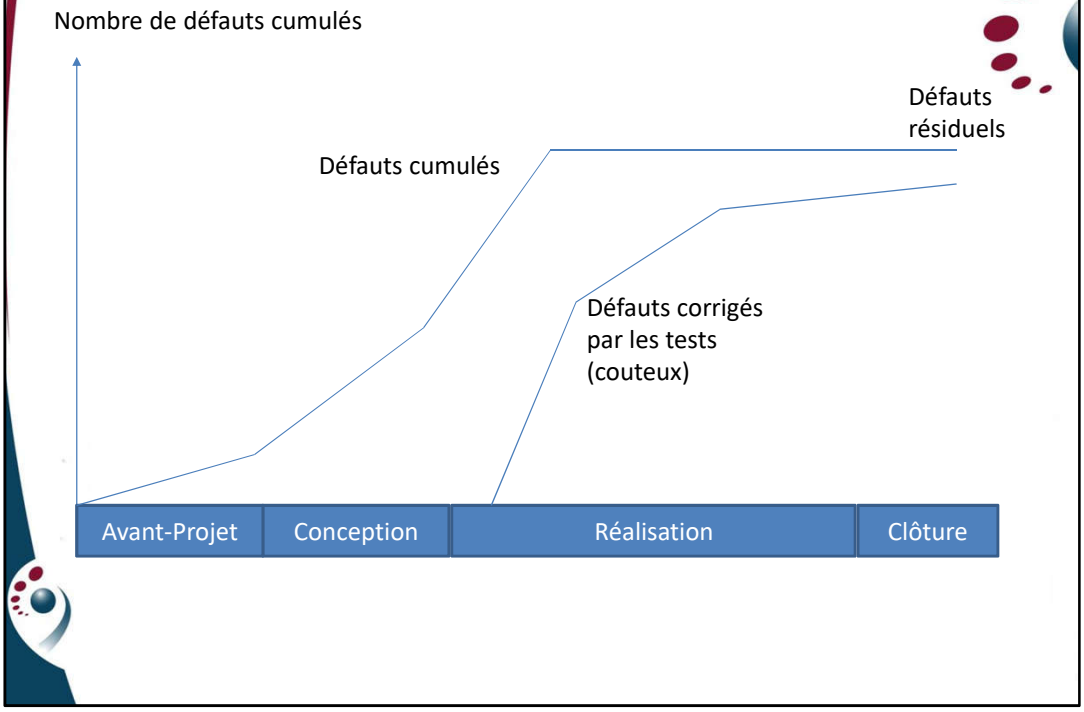
- Validation : confirmation par l'examen et la fourniture de preuves objectives que les exigences, pour un usage ou une application voulue, ont été remplies [ISO 9000]

V&V

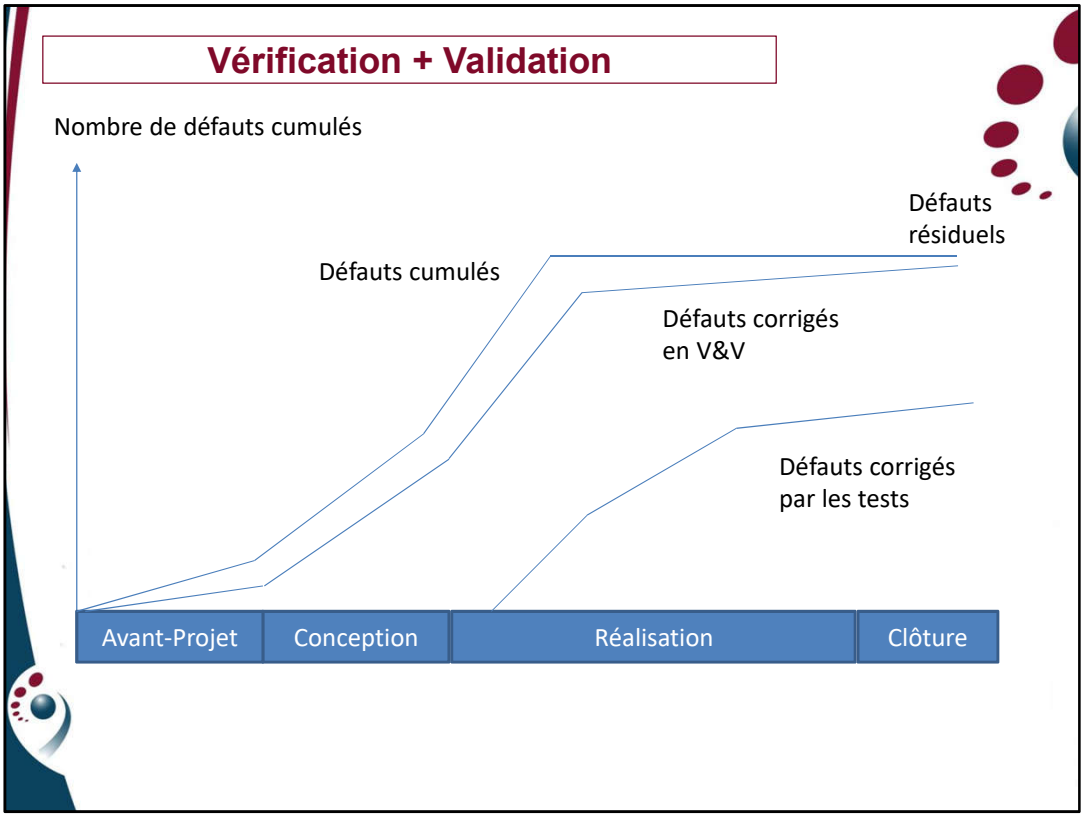
- La vérification se fait par « inspection » et la validation par des tests.
- En pratique, l'étude de besoins et la conception est vérifier par relecture (peut chère).
- La réalisation est constaté par relecture, et valider par des tests.


V&V
Nombre de défauts détectés
Tests (couts élevés)
Inspections (couts faibles)
Avant-Projet
Conception
Réalisation
Clôture

Si Validation uniquement



Vérification + Validation



V&V

2 profils de qualité en V&V:

- Le profil exponentiel qui montre que plus on corrige de bug, plus le nombre de bug explose
- Le profil linéaire (défaut résiduel trouvé en faible nombre)
  - Soit la puissance des tests est faible. Le niveau de qualité est médiocre.
  - Soit les tests sont suffisants. Le niveau de qualité est correct

V&V

- Principe N°1: Tester exhaustivement un logiciel est généralement impossible
- Principe N°2: Tester correctement un logiciel est une tâche difficile qui requiert intelligence et créativité
- Principe N°3: L'essence de l'activité de test est la prévention Elle s'applique à toutes les phases du cycle Il est futile de concevoir ce que l'on ne saura pas tester
- Principe N°4: Le volume et la nature des tests à effectuer doit s'apprécier en terme de risques que l'emploi du logiciel fait courir à l'organisation cible
- Principe N°5:La planification sérieuse de la V&V est indispensable à la maîtrise du projet. Chaque tâche du projet a sa propre V&V afin d'éviter l'effet d'avalanche lors de l'intégration
- Principe N°6:L'évaluation honnête de la qualité exige la présence d'un tiers de confiance (AQ logiciel) indépendant du développement

Historique et définitions

Familles de tests : unitaires, fonctionnels, cohérence en base, intégration, charge.

- Quels sont les grandes familles de tests?
- Comment se placent les tests en fonction de la phase du projet

Familles de tests

- les tests sont de natures différentes en fonction du fait qu'ils sont réalisés avant le projet, pendant le projet ou après le projet.
- Elles servent à valider les différentes parties d'un projet et peuvent faire intervenir des organisations différentes

Famille de tests : Avant-Projet

Le but de ces tests est de valider que l'étude de besoins et l'étude d'impact correspondent à une réalité.

Parmi ces tests on trouve:

- Calcul des besoins d'une solution : Etude de l'état de l'art, calcul des couts du SI actuelle
- Création d'un/plusieurs POC : valide une faisabilité technique et permet de limiter l'impact d'une solution
- Etude marketing/sociologique pour valider la potentialité des changements dans une société

Famille de tests: Pendant le projet

- Relecture des spécifications: Travailler en binôme sur cette relecture. Faites remonter les observations et en particulier sur la description des données, sur la description des traitements et sur les éléments graphiques.
- Tests unitaires: Il s'agit de test de cas d'usage simple qui doivent prendre en compte les parties de logiciel non/mal conçue ainsi qu'un échantillonnage possible des données. Les tests d'intégration sont regroupé dans les tests unitaires

Famille de tests: Pendant le projet

Les tests techniques sont faits pendant le projet et on retrouve les tests:

- Nominaux
- Valeurs clefs
- Aux limites
- Hors domaine (non passants)
- De robustesse
- De configuration matérielle
- De performance

Famille de tests: Pendant le projet

- Les tests d'interface permettent de valider la bonne circulation des données (entre deux systèmes ou entre l'utilisateur et le logiciel)
- Test de validation technico-fonctionnel (nominaux, valeurs clefs et aux limites) permettent de valider que le logiciel répond aux besoins métiers.
- Test de validation fonctionnel permettent de mettre sur le marché que le logiciel.

- Les tests faits pendant le projet peuvent être écrit :
  - En boite noire : le code source du logiciel n'est pas utilisé pour faire le test
  - En boite blanche: le code source est utilisé pour décrire le test

Famille de tests: Après le projet

- Les tests de performance peuvent commencer après une certaines volumétrie.
- Les tests d'usage: quel partie du logiciel est réellement utilisé
- Les tests de changements
- Valorisation des tests pour la maintenance et les tests de non régression

Historique et définitions

Tests de non-régression.

- Qu'est-ce qu'un test de non régression?
- Quel est le périmètre d'un test de non régression?
- Pourquoi ces tests sont si importants ?

Tests de non régression

- Une régression est quelque chose qui fonctionnait dans le passé et qui ne fonctionne plus.
- La problématique des régressions apparaissent soit dans un chantier de maintenance (nouvelle fonctionnalité, patch …) ou dans un chantier de run dans le cadre des méthodes Agile
- Une **régression** est récurrente lors de corrections de bogues ou n'importe quel changement établi dans un logiciel : ajout de nouvelles fonctionnalités, modification de fonctionnalités existantes ou modification d'un composant externe au logiciel lui-même : nouvelle version du système, de l'interface graphique, d'un compilateur ou d'une bibliothèque tierce qui interviennent dans son fonctionnement. Elle menace souvent le bon fonctionnement du logiciel ou de certaines de ses fonctionnalités majeures ou mineures, ce qu'il est important d'éviter ; on effectue à cette fin des **tests de régression**.

Test de non régression

- Les tests de non régression sont dans l'idéal l'adjonction de nouveaux tests (de la nouvelle fonctionnalité) à l'ensemble des tests existants (d'où la nécessité de capitaliser).
- Deux cas, soit il est possible de faire faire les tests par de robots (tests automatiques), soit il faut faire une analyse d'impact des modifications:
  - L'analyse de l'application (code…), des impacts de chaque modification faite et la comparaison de chaque version permettent d'identifier tous les changements et les risques associés. La difficulté est d'obtenir une vision de ces risques qui soit exploitable pour des tests fonctionnels : au-delà du fichier modifié il faut évaluer son impact sur les fonctionnalités.
  - Pour améliorer cette analyse, une solution consiste à prendre l' « empreinte » de chaque test sur l'application (ce qui est exécuté dans l'application lors d'un test). Cette prise d'empreinte fait le lien entre code, modules fonctionnels et scénarios de tests. Une fois ce lien établi, il est possible de savoir exactement ce que couvre un test en particulier.

Test de non régression

- Pour les tests en boite blanche (chantier run), il est possible :
  - De s'appuyer sur le différentiel entre les sources des deux versions.
  - Sur les méthodes les plus complexes
  - Sur les dépendances entre les classes

- Pour les tests en boite noir (chantier maintenance), on essaye de tester:
  - Les nouvelles fonctionnalités
  - Les fonctionnalités les plus utilisé par les utilisateurs

Historique et définitions

Les livrables (cahier des charges, spécifications).

- Lister les différents livrables.
- Attente des livrables

Les livrables

Les livrables par ordres d'importance sont:

- Un PV de recette vierges
- Un manuel d'installation et d'exploitation
- Spécifications fonctionnel du logiciel
- Le cahier des charges qui décrit brièvement ce que doit faire le logiciel.
- Matrices de risques produits et projet

PV de recette

Logo

Procès-verbal de recette

Client	Nom du client	**Projet**	Nom du projet
Créé le	Jj/mm/aaaa	**MAJ**	Jj/mm/aaaa
Auteur	Nom Auteur	**Version**	1
Destinataires	Chef de projet		
Fichier	pv_recette.doc		

Livrables

Nom du livrable, date de livraison, éventuellement nom du fichier et/ou version
…

L'entreprise XXX reconnaît par la présente (cocher les cases correspondantes)[1] :

☐ avoir reçu la livraison du (des) livrable(s) mentionné(s) ci-dessus[2] ;

☐ que ce(s) livrable(s) est (sont) conforme(s) à ses attentes[3] ;

☐ que ce(s) livrable(s) est (sont) conforme(s) à ses attentes sous réserves (voir ci-dessous)[4] ;

☐ que ce(s) livrable(s) n'est (ne sont) pas conforme(s) à ses attentes (voir ci-dessous).

Réserves

Préciser les réserves

Fait en deux exemplaires à ..... le .....

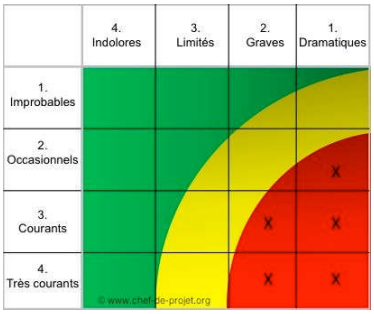
Pour le fournisseur XXX	Pour l'entreprise XXX
Prénom Nom	Prénom Nom
Fonction	Fonction
Signature	Signature

Matrice des risques

- Bien qu'il ne participe pas nécessairement à l'identification d'un projet, lors de la prise en charge, l'ingénieur devrait analyser les divers aspects dont il aura la responsabilité, afin de clarifier les objectifs du client ainsi que ceux de son entreprise, et s'assurer de sa compréhension des enjeux.
- En, effet, ces derniers peuvent être négligés dans la fébrilité des activités de démarrage du projet et les obligations professionnelles quotidiennes
- Les risques critiques doivent être couvert par des tests

Matrice des risques

- **La typologie des risques** permet de classer le risque pour mieux guider l'analyse. Un risque peut être catalogué dans cinq grandes classes:
  - Organisationnel
  - Technique
  - Financier
  - Humain
  - Juridique



- Cette classification du risque apporte une vision globale qui facilite la mise en place de solution. Il est à noter que le caractère humain est le risque le plus complexe, et que l'insatisfaction du client est le risque le plus incontrôlable.

Enjeux économiques et techniques des métiers du test

- Historique et définitions
- Opportunités du test aujourd'hui
  - Pourquoi les tests sont-ils nécessaires ?
  - Constats de la situation du test logiciel.
  - Les avancées et les difficultés des projets de tests.
  - Les axes d'amélioration, les facteurs de succès.
- Eléments clés d'une validation fonctionnelle
- Coûts et rentabilité du projet de test
- Les approches du test aujourd'hui
- Outils et Infrastructure de test Agile
- Les solutions de gestion des tests
- Synthèse

Opportunités du test aujourd'hui

Pourquoi les tests sont-ils nécessaires ?

- Quelle est le cout d’un bug?
- Pourquoi le test est nécessaire pour éviter des surcouts utilisateurs?

Pourquoi les tests sont-ils nécessaires ?

Ariane V vol 501, 1996 : problème de conversion de nombres

Coût : 370 millions de dollars

Therac-25, 1985-87 : problème de synchronisation

Coût : plusieurs vies

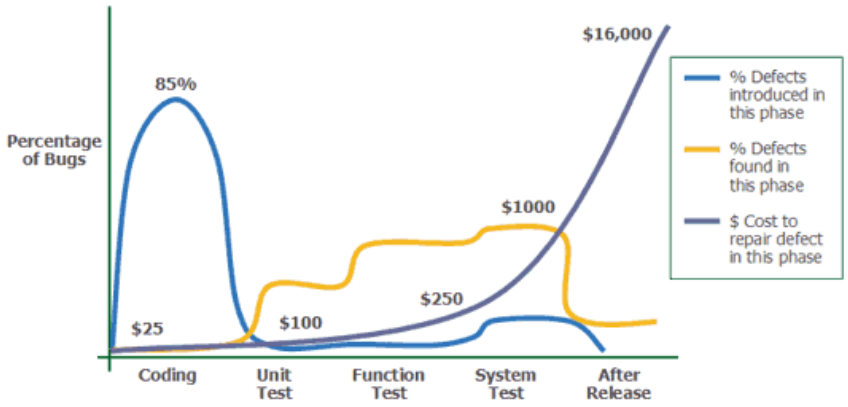
Panne d'électricité aux États-Unis et au Canada, 2003

Impact sur 55 millions d'habitants. Coût : 6 milliards de dollars

2020 :

- Casino a un bug sur l'affichage de prix
- Caisse général de securité social en France
- Dépôt pétrolier de Brest
- Débit double du Crédit mutuelle

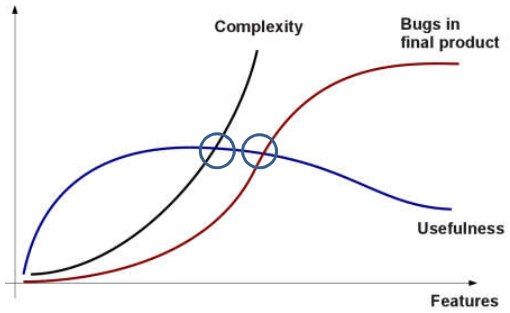
Coût d'un bug



source: Applied Software Measurement, Capers Jones

Ce schéma ne prend pas le coût du retard de mise sur le marché

Cout d'un bug en fonction de la complexité



- Plus un produit a de fonctionnalité:
  - Plus il y a des bugs
  - Plus c'est complexe
  - Moins les fonctionnalités sont utilisés
- Source : https://blog.pdark.de/tag/agile-development/

Cout d'un bug (étude NIST)

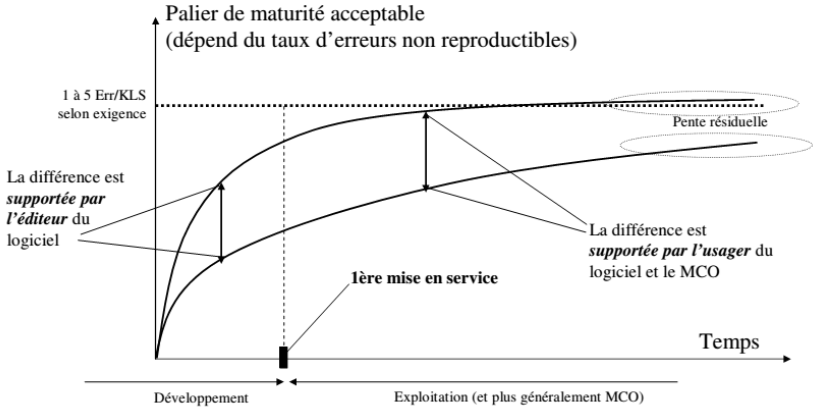
Cout de réparation		Temps de détection				
		Besoin	Architecture	Construction	Pré-Production	Mise sur le marché
Moment de création du bug	Besoin	X 1	X 3	X 5-10	X 10	X10 – X 100
	Architecture		X 1	X 10	X 15	X 25 – X 100
	Construction			X 1	X 10	X10 – X25

Le cout des bugs

- Le cout des bugs invite à pouvoir vérifier et valider un produit.
- Une question est pourquoi vérifier un produit :
  - Pour l'utilisateur : coût économique, humain, environnemental
  - Pour le fournisseur : coût de la correction des bugs, coût image
- Une autre question est pourquoi NE PAS vérifier un produit :
  - Pour l'utilisateur : moins d'attente, délaie d'utilisation plus rapide, tolérance aux erreurs
  - Pour le fournisseur : moins couteux, permet un management cost box d'un projet
- Pour info 16000€ équivaut à 16 j/h en couts environné. Est-ce qu'un bug va être gênant pour les utilisateurs? Et est-ce que le test de la fonctionnalité consomme plus de 16j?

Pourquoi tester?

- Néanmoins, après la mise en production, chaque bug résiduel va couter entre 10 et 300 plus cher.
- Cette charge devient une charge utilisateur



Pourquoi tester?

La question n'est donc pas pourquoi tester? Mais comment tester?

- Comment bâtir une bonne stratégie de test?
- Comment bien fabriquer des tests?

Opportunités du test aujourd'hui

Constats de la situation du test logiciel.

- Les avancées et les difficultés des projets de tests.
- Les axes d'amélioration, les facteurs de succès.

Constats de la situation du test

Technique du test est issue d'une réponse à la crise du logiciel apparue dans les années 70 (prise de conscience que le coût du logiciel dépassait le coût du matériel)

- Répondre à la croissance de la taille et de la complexité des systèmes
  - Besoins et fonctionnalités augmentent, évoluent
  - Technologies en perpétuelle évolution
  - Diversification des architectures
  - Faire face aux délais de plus en plus courts,
- Gérer des équipes de plus en plus grosses, avec des compétences multiples

Constats de la situation du test

Préoccupations:

- L'industrialisation de la production du logiciel :
  - Organisation des procédés de production (cycle de vie, méthodes, notations, outils), organisation des équipes de développement, établissement de plan qualité rigoureux, etc.
  - Incluse dans les principes de fonctionnement de l'entreprise
- Rigueur et formalisation, Séparation des problèmes, Modularité, Abstraction, Anticipation des changements, Généricité, Construction incrémentale
  - Règle du CQFD (Coût Qualité Fonctionnalités Délai)
- Le système qui est fabriqué répond aux besoins des utilisateurs (correction fonctionnelle).
  - La qualité correspond au contrat de service initial.
  - Les coûts restent dans les limites prévues au départ.
  - Les délais restent dans les limites prévues au départ

Constats de la situation du test

Demandes Utilisateurs:

- Correction (validité) : aptitude à répondre aux besoins et à remplir les fonctions définies dans le cahier des charges
- Robustesse (fiabilité) : aptitude à fonctionner dans des conditions non prévues au cahier des charges, éventuellement anormales
- Extensibilité : facilité avec laquelle de nouvelles fonctionnalités peuvent être ajoutées à un logiciel
- Compatibilité : facilité avec laquelle un logiciel peut être combiné avec d'autres
- Efficacité : utilisation optimale des ressources matérielles (processeur, mémoires, réseau, …)
- Convivialité : facilité d'apprentissage et d'utilisation, facilité de préparation des données, facilité de correction des erreurs d 'utilisation, facilité d'interprétation des résultats

- Demandes Techniques
  - Intégrité (sécurité) : aptitude d'un logiciel à protéger son code contre des accès non autorisés.
  - Réutilisabilité : Aptitude d'un logiciel à être réutilisé, en tout ou en partie, pour d'autres applications
  - Vérifiabilité : aptitude d'un logiciel à être testé (optimisation de la préparation et de la vérification des jeux d'essai)
  - Portabilité : aptitude d'un logiciel à être transféré dans des environnements logiciels et matériels différents
  - Lisibilité,
  - Modularité.

Constats de la situation du test

- Chaos Standish Group:
  - 19 % des projets arrivé en qualité correcte à l'heure, 19% ne sorte pas
  - 61% des projets réussie coutent moins de 750k$
  - Uniquement 2% des projets de plus de 10M$ réussissent

MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

The Modern Resolution (OnTime, OnBudget, with a satisfactory result) of all software projects from FY2011-2015 within the new CHAOS database. Please note that for the rest of this report CHAOS Resolution will refer to the Modern Resolution definition not the Traditional Resolution definition.

Constats de la situation du test

- 2% des projets d'envergure sont des succès.

CHAOS RESOLUTION BY PROJECT SIZE

	SUCCESSFUL	CHALLENGED	FAILED
Grand	2%	7%	17%
Large	6%	17%	24%
Medium	9%	26%	31%
Moderate	21%	32%	17%
Small	62%	16%	11%
TOTAL	100%	100%	100%

The resolution of all software projects by size from FY2011–2015 within the new CHAOS database.

Constats de la situation du test

Avec l'envol des méthodes de développement agile ces dernières années, il a été possible de comparer les résultats des projets entre projets classiques et agiles. Ces approches agiles performent mieux que les cycles en cascade quelque soit la taille du projet, et échouent plus rarement, comme le montre le tableau ci-après.

CHAOS RESOLUTION BY AGILE VERSUS WATERFALL

SIZE	METHOD	SUCCESSFUL	CHALLENGED	FAILED
All Size Projects	Agile	39%	52%	9%
	Waterfall	11%	60%	29%
Large Size Projects	Agile	18%	59%	23%
	Waterfall	3%	55%	42%
Medium Size Projects	Agile	27%	62%	11%
	Waterfall	7%	68%	25%
Small Size Projects	Agile	58%	38%	4%
	Waterfall	44%	45%	11%

The resolution of all software projects from FY2011–2015 within the new CHAOS database, segmented by the agile process and waterfall method. The total number of software projects is over 10,000.

Constat de la situation du test

CHAOS FACTORS OF SUCCESS

FACTORS OF SUCCESS	POINTS	INVESTMENT
Executive Sponsorship	15	15%
Emotional Maturity	15	15%
User Involvement	15	15%
Optimization	15	15%
Skilled Resources	10	10%
Standard Architecture	8	8%
Agile Process	7	7%
Modest Execution	6	6%
Project Management Expertise	5	5%
Clear Business Objectives	4	4%

Constat de la situation du test

- **Executive Support** (Sponsorship) implique qu'un dirigeant ou groupe de dirigeants accepte de porter financièrement et émotionnellement le projet. Le ou les dirigeants encourage et accompagne l'accomplissement du projet.
- **Emotional Maturity** (Maturité émotionnelle) est l'ensemble des comportements permettant de travailler en équipe. Dans tout groupe, organisation ou entreprise, ce niveau de maturité émotionnelle est à la fois la somme de toutes les compétences et le lien le plus faible.
- **User Involvement** (Engagement de l'utilisateur) a lieu quand les utilisateurs sont engagés dans les processus de décisions et de collectes d'informations du projet. Cela passe également par du feedback, des revues de demandes, de la recherche classique, du prototypage, et autres outils de construction de consensus.
- **Optimization** est une manière structurée d'améliorer l'efficacité métier et d'optimiser un ensemble de petits projets ou de demandes importantes. L'optimisation commence avec la gestion du périmètre d'après les valeurs métier relatives.
- **Skilled Staff** (Personnel compétent) sont les personnes comprenant à la fois le métier et la technologie. La compétence donne une forte capacité dans la réalisation des demandes du projet et la livraison du projet ou produit.

Constat de la situation du test

- **SAME** est le "Standard Architectural Management Environment". Le Standish Group définit le SAME comme un groupe cohérent et intégré de pratiques, services et de produits permettant de développer, implémenter et opérer une application.
- **Agile Proficiency** (Connaissance) signifie que l'équipe et le product owner maîtrisent l'agilité. La connaissance agile est ce qui fait la différence entre de bons résultats avec l'agile et de mauvais.
- **Modest Execution** (Réalisation frugale) implique d'avoir un processus avec peu d'encours automatisés et fluides. La réalisation frugale passe aussi par une gestion de projet parcimonieuse et avec seulement quelques fonctionnalités.
- **Project Management Expertise** est l'usage de connaissances, compétences et techniques aux activités de projet pour atteindre ou dépasser les attentes des parties prenantes et produire de la valeur pour l'organisation.
- **Clear Business Objectives** (Objectifs métier clairs) est la compréhension des attentes par toutes les parties prenantes et participants dans le projet. Des objectifs métier clairs peuvent aussi signifier un alignement sur les enjeux et la stratégie de l'entreprise.

Constat de la situation du test

- Loi des 2 faces: Les utilisateurs sont les meilleurs amis… Et les pires ennemies
- Loi des singes: Il est important d'avoir un chef
- Loi de la route: La route doit être clair et dégagé
- Loi des 5 signes de morts (Ambition, Arrogance, Ignorance, Fraude, Abstinence))
- Loi du monstre sans queue : Prendre en compte que 50% des fonctionnalités de servent à rien
- Loi de l'éléphant: Un logiciel doit rester simple
- Loi du beau parleur: Comprendre l'importance du management de projet dans l'entreprise
- Loi de la chaise vide: Eviter la perte de ressources
- Loi du Panda: L'inaction est la pire des approches
- Loi des fous: éviter d'utiliser des outils sans maitrise

Enjeux économiques et techniques des métiers du test

- Historique et définitions
- Opportunités du test aujourd'hui
- Eléments clés d'une validation fonctionnelle
  - Eléments Stratégiques
  - Les fondements. Le processus de recette.
  - Le plan de recette.
  - Organisation et suivi. Gestion des jeux de données.
  - L'enregistrement des résultats et des anomalies.
  - Le processus de packaging et de livraison.
  - Les clauses de recette.
- Coûts et rentabilité du projet de test
- Les approches du test aujourd'hui
- Outils et Infrastructure de test Agile
- Les solutions de gestion des tests
- Synthèse

Eléments clés d'une validation fonctionnelle

Eléments Stratégiques

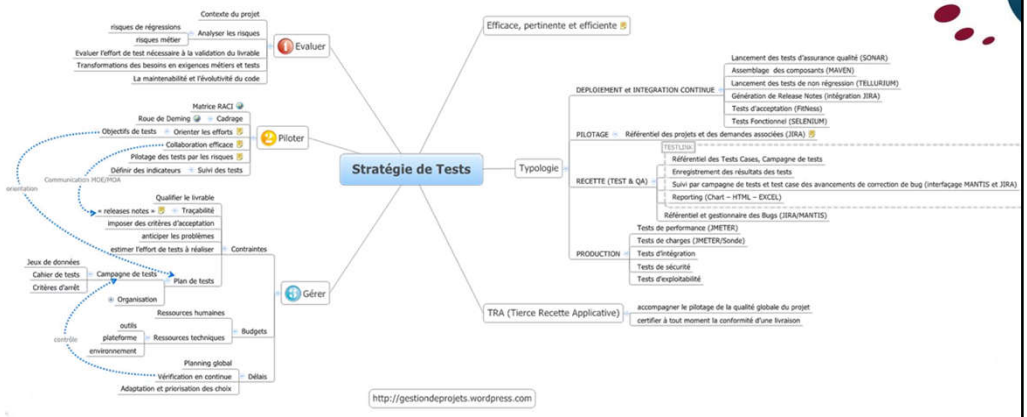
- Comprendre comment positionner une validation fonctionnelle.
- Comprendre et anticiper les risques/impacts d’un projet sur une validation fonctionnelle

Eléments Stratégiques

« Qui connaît son ennemi comme il se connaît, en cent combats ne sera point défait. Qui se connaît mais ne connaît pas l'ennemi sera victorieux une fois sur deux. Qui ne connaît ni son ennemi ni lui-même est toujours en danger. »

Sun Tzu

Stratégie de test



Stratégie de test

- **Évaluation**; le contexte du projet détermine l'orientation stratégique de vos tests (on ne teste pas avec les mêmes objectifs et priorités, une application e-commerce ou un site institutionnel), il est donc primordiale d'amasser certaines informations du projet. Cette étape débute donc par l'analyse des informations significatives du projet. Cette étude doit contrôler la faisabilité d'une transformation des exigences en test, à partir de la qualité des informations à disposition (anticiper sur les futurs livrables).
- Une partie importante de cette analyse consiste à identifier et évaluer les risques. L'autre sujet de cette phase consiste à évaluer l'effort de test nécessaire à la validation du livrable.

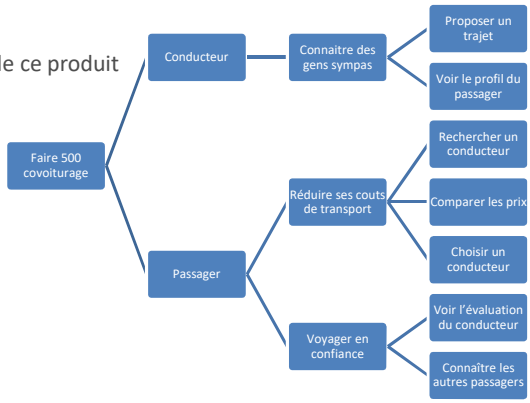
Le bon produit

- Les méthodes Agiles, WaterFaill donnent un cadre pour créer un produit, mais ne dise pas comment arriver au bon produit.
- L'idée est d'arriver à se poser les questions suviante:
  - Quelle est notre vision du produit?
  - Qui sont les acteurs ?
  - Quels seront les impacts?
  - Quelles features proposer pour obtenir cette impact?
- Ce n'est pas en Agile un problème du Product Owner mais de toute l'équipe, et en particulier du responsable QA.
- D'après le Chaos Standish Group, une mauvaise évaluation du produit est une première cause d'échec.

Impact Mapping

L'impact Mapping est une carte heuristique issue des mind mapping qui contient. C'est cette carte « Vision du produit » que nous souhaitons obtenir

- en son centre les objectifs du produit.
- Puis les acteurs du produit
- Les impacts du produit
- Et enfin les fonctionnalités de ce produit



Les parties prenantes

- Les parties prenantes vont servir a faire émerger les impacts du produits.
- Nous avons plusieurs parties prenantes:
  - Les utilisateur primaires
  - Les utilisateurs secondaires Administrateur, Modérateur…
  - Les parties prennantes tierces qui ne sont pas utilisateurs : hierarchies, sponsor …

Les parties prenantes

- On essaye de faire émerger des « personnas » des parties prenantes.
- Ces profils fictifs sont le regroupement:
  - D'un nom, age profession.
  - Son utilisation des terminaux (Desktop, tablette, mobile ..)
  - Son appétence aux changements, aux nouvelles technologies
  - Ses attentes et objectif du produit.
  - Une phrase courte pour définir se définir par rapport au produit

Pour ces partie prenantes, trouver les impacts

- Dans la vision non-agiles, ce document est le besoin utilisateur.
- Ici on parle d'analyse d'impacts:
  - Quels sont les changement dans le comportement des parties prenantes va induire.
  - Ce n'est pas trouver la solution.
- Une solution parmis d'autres est de jouer au retro futur:
  - Un jeu d'acteur décrivant les changements positifs que le produit a eu dans leurs pratiques

La release

- La release a pour objectif de définir les fonctionnalités qui vont résoudre le maximum d'impact.
- Ces fonctionnalité contribue à un impact et se décompose en User Stories.
- Avec une approche Lean Startup:
  - On essaye de dépenser le moins d'énergie possible pour résoudre les impacts les plus fort.
  - Le but est créer un produit minimal mais viable.
- La question est de savoir quels sont les fonctionnalités essentielles pour construire ce produit minimal.
  - Une solution parmis d'autre est la matrice de Kano

Modèle de Kano

Le modèle de Kano est théorisé par Noriaki Kano en 1984 à partir du constat que la satisfaction et l'insatisfaction résultant chez un observateur de la perception d'un produit ou d'un service ne sont pas des concepts de valeur symétrique. D'après lui en effet, l'existence d'une caractéristique d'un produit/service peut satisfaire un utilisateur, sans que son absence ne provoque une sensation d'insatisfaction.

Modèle de Kano

C'est une théorie intéressante et féconde pour l'évaluation, la conception et le développement d'une offre de produit/service:

- Elle se fonde sur la perception du client réel ou potentiel
- Elle prend en compte les attentes explicites mais peut faire émerger les attentes latentes (non exprimées) par ce client
- Elle permet de renouveler le cas échéant la vision initiale faite au début du cycle de vie et qui est sujette avec le temps à banalisation.

Modèle de Kano

- Les attributs d'un produit/service sont les supports des attentes des clients. Ils peuvent être classés en cinq catégories :
- obligatoires, indispensables (Must-Be)
- attractifs (attirantes)
- proportionnelles ou linéaires (One-Dimensional)
- indifférents
- à double tranchant (Reverse)

Modèle de Kano

Obligatoires, indispensables - (Must be) Ces attributs souvent implicites du fait qu'ils sont considérés comme essentiels : Leur présence et leur bon fonctionnement sont perçus comme « normaux » sinon obligatoires. S'ils remplissent leur rôle, tout va bien, mais la situation est juste normale. À l'inverse, leur absence ou leur dysfonctionnement provoquent immanquablement une insatisfaction - ressentie comme légitime- qui peut tourner à la frustration. Exemples : une porte qui ne ferme pas, une bouteille d'eau qui fuit.

Modèle de Kano

Attractifs - (attirants) Ces attributs rencontrent la satisfaction du client quand ils sont présents et fonctionnent effectivement. Ces attributs sont réputés ne pas faire « normalement » partie du produit. Leur perception constitue donc une « heureuse surprise », mais leur absence n'entraîne pas de mécontentement ou d'avis négatif sur le produit. Ces attributs augmentent la satisfaction de manière implicite : ils correspondent à des besoins latents ou émergents (pas vraiment exprimés, sinon inconscients). Exemples : un testeur de charge sur une pile, un thermomètre sur une brique de lait.

Modèle de Kano

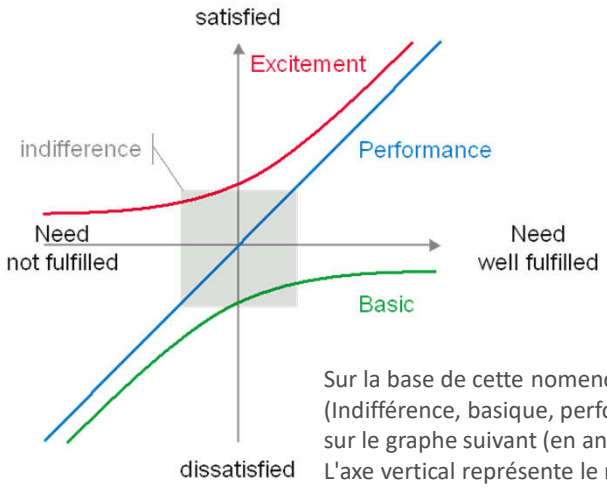
Proportionnels ou linéaires - (One-Dimensional) Ces attributs accroissent la satisfaction à proportion de la façon dont on les satisfait. La satisfaction ou l'insatisfaction résulte de l'adéquation de la performance constatée par rapport à la performance annoncée ou suggérée : Satisfaction quand ils sont présents et fonctionnent, insatisfaction lorsque ces attributs ne sont pas présents ou ne correspondent pas à l'attente créée. Exemples : une pile censée durer 1h de plus que les autres, ou un paquet de biscuit avec 10 % de poids en plus au même prix. Si l'argument est faux (la pile ne dure pas plus longtemps ou que ce n'est pas 10 %, mais 3 % en plus uniquement), le non-respect d'une attente créée provoque directement l'insatisfaction du client/consommateur. La frustration est proportionnelle à l'intensité de la promesse censée être faite par la présence de ces attributs.

Modèle de Kano

Indifférents - (Zone d'indifférence) De tels attributs ne provoquent pas du fait de leur présence/absence de mécontentement ou d'insatisfaction significatifs.

A double tranchant - (Reverse) Ces attributs - du fait de la diversité des opinions et des usages - sont interprétés par les uns comme un avantage (satisfaction) et considérés par d'autres comme un inconvénient (insatisfaction). Exemple : certains préfèrent utiliser un téléphone portable muni seulement de fonctions de base, tandis que d'autres ne jurent que par le « dernier cri » doté de toutes les options.

Modèle de Kano



Sur la base de cette nomenclature, quatre zones-fonctions (Indifférence, basique, performance, excitant) sont repérées sur le graphe suivant (en anglais) qui comporte deux axes :
L'axe vertical représente le niveau de satisfaction : En bas, le moins satisfait (dissatisfied), En haut, le plus satisfait (satisfied)
L'axe horizontal représente la prise en compte des attentes : à gauche faible couverture (Need not fullfilled), à droite forte couverture (Need well fullfilled).

Modèle de Kano: Mise en Oeuvre

- Lister les fonctions du produit ou du service que vous souhaitez évaluer.
- Préparer dans un tableur une maquette pour recenser les avis de chaque étudiant.
- Simplifier la vie des clients en préparant des listes déroulantes pour les réponses.
- Demander aux clients de renseigner la maquette. Ils doivent « noter » chaque fonction selon si celle-ci est présente ou absente.
- Recueillir les données

Formulaire

	J'aime	Je l'espère	Je suis neutre	Je peux vivre avec	Je ne le supporte pas
Si ca fonctionne					
Comment le vivrer vous si la function est dans le projet					
Si la function existe est qu'elle peux plus ?					
Si ca ne fonctionne pas					
Comment le vivrer vous si la function n'est pas dans le projet					
Si la function existe de facon basique					

Modèle de Kano: Mise en Oeuvre

- On applique la matrice de Kano :

Question fonctionnelle \ Question dysfonctionnelle	1	2	3	4	5
1	D	A	A	A	P
2	C	I	I	I	O
3	C	I	I	I	O
4	C	I	I	I	O
5	C	C	C	C	D

A – Fonction attractive

P – Fonction proportionnel

O – Fonction obligatoire

I – Fonction indifférente

C – Fonction contraire (hostile)

D – Erreur de compréhension ou de codage

Modèle de Kano

- Importance de la Variable temps : un attribut peut plus ou moins vite passer d'une Zone-fonction à l'autre. Ainsi, dans le high-tech, un attribut passe de la zone-fonction excitante à performante et même basique en quelques mois et peut-être moins.
- Le questionnement du client à propos de la valeur accordée à tel ou tel attribut doit être d'une qualité impeccable : l'échelle de subjectivité d'un individu peut être importante et l'agrégation de préférences individuelles en une fonction de préférence collective demeure un exercice délicat.

Stratégie de test

- Pilotage; la communication est le pivot d'une stratégie de tests. L'élaboration d'une matrice RACI permet de contrôler cette communication en établissant des règles.

- Piloter une stratégie de tests consiste à orienter les efforts de test, sur les bons composants et fonctionnalités, au moment opportun.
  - Pour gagner en efficacité, réaliser un pilotage des tests par les risques (RBT – Risk-Based Testing, qui constate qu'une couverture à 100% des exigences fonctionnelles et non-fonctionnelles relève de la théorie, et recommande de mieux cibler ses efforts).

- Ces objectifs de tests régentent le plan de tests.

- La conduite de tests réclame un suivi efficace, qu'il faut établir à partir d'indicateurs pertinents. La communication autour de ces rapports doit se mettre à niveau en fonction des interlocuteurs. Cette activité de contrôle doit être opérationnelle tout au long de votre campagne de tests.

Stratégie de test Agile

- Modèle en spirale:
  - 1986, Barry Boehm publie le modèle en spirale (développement incrémental) .
- Modèle RAD:
  - 1991, James Martin (en) (RAD), s'appuyant sur une vision de l'évolution continue des technologies informatiques, propose une méthode de développement rapide d'application
  - 1994, Jean-Pierre Vickoff en France, notamment avec le Processus RAD2
- Modèle Agile:
  - En 2001, dix-sept figures éminentes du développement logiciel se réunissent pour débattre d'un thème unificateur de leurs méthodes respectives. Les plus connus d'entre eux sont Ward Cunningham(WikiWikiWeb), Kent Beck(Junit), Ken Schwaber (Scrum).
  - Ces 17 experts extraient alors de leurs usages respectifs les critères communs et les principes qui, selon eux, conduisent aux meilleures concepts de direction de projets et de développement de logiciels.

Stratégie de test Agile

Les méthodes agiles prônent 4 valeurs fondamentales (Agile Manifesto) :

- Individus et interactions plutôt que processus et outils
- Fonctionnalités opérationnelles plutôt que documentation exhaustive
- Collaboration avec le client plutôt que contractualisation des relations
- Acceptation du changement plutôt que conformité aux plans

Stratégie de test Agile

Douze principes généraux en découle:

1. Satisfaire le client en priorité
2. Accueillir favorablement les demandes de changement
3. Livrer le plus souvent possible des versions opérationnelles de l'application
4. Assurer une coopération permanente entre le client et l'équipe projet
5. Construire des projets autour d'individus motivés
6. Privilégier la conversation en face à face
7. Mesurer l'avancement du projet en termes de fonctionnalités de l'application
8. Faire avancer le projet à un rythme soutenable et constant
9. Porter une attention continue à l'excellence technique et à la conception
10. Faire simple
11. Responsabiliser les équipes
12. Ajuster à intervalles réguliers son comportement et ses processus pour être plus efficace

Scrum

- Scrum est une pratique Agile rajoutant:
  - La capacité de faire des démo du produit
  - La cohésion de l'équipe par les locaux
  - Faire une réunion de planification hebdomadaire (Daily Scrum)
  - Faire une réunion de planification de l'itération
  - Les itérations sont bornées dans le temps.
  - Faire la rétrospective du travail effectué
  - Faire un recueil des fonctionnalités a faire par itération/produit

Nouveaux Rôles

- Product Owner: Représentant des métiers
  - Planifie la release
  - Etudie le fonctionnel. Aide à rédiger les test plans
  - Détient et partage la vision du produit
  - Protège l'équipe pendant l'itération
  - Accepte ou refuse le travail
  - Communique à l'extérieur
  - Fait partie de l'équipe.
  - Prend la responsabilité du projet

Nouveaux Rôles

- Coach Agile/Scrum master:
  - Facilite : Vérifie l'application de la méthode, la cohésion du groupe
  - Apprend et Partage: Il doit se tenir au courant des bonnes pratiques Agile, enseigner à l'équipe les bonnes pratiques
  - Protection : Il doit gérer les conflits dans l'équipe et la proteger de l'exterieur
  - Reflexivité: Il renvoie à l'équipe se qui fonctionne ou pas. Reporting de la situation de l'équipe
  - Coach : Il doit aider a fixer les problèmes rapidement (indépendamment du qui/quoi). Communique avec l'équipe et la fait communiquer
  - Victoire: Il fait partager la victoire du projet

Nouveaux Rôles

- Testeurs, avant dévolue à faire les tests fonctionnels et d'intégration
- Il devient Coach Agile, Expert Qualité
  - Il a pour rôle de connaitre suffisamenet le métier pour pouvoir évaluer la qualité
  - Aide les developpeurs a mettre en place des tests unitaires pertinent
  - Il mets en place les tests d'acceptation pour le fonctionnel
  - Il aide a mettre en place l'intégration continue
  - Il participe directement à l'évaluation des charges
  - Il décrète la fin d'un developpement.
- A proprement parler le rôle de testeur disparait de la méthode scrum

Nouveaux termes

- Features : fonctionnalité du produit
- User Story : Sous fonctionnalité du produit exprimé en terme métiers
- Epic Story : Ensemble de User Story
- Iteration : Cycle de developpement contenant des User Story
- Release : ensemble d'itération.
- Sprint : Issue de Scrum, est une sous itération.
- User Story Backlog : ensemble de User Story à développer
- Sprint/Iteration User Story Backlog: ensemble de User Story pour le sprint/itération

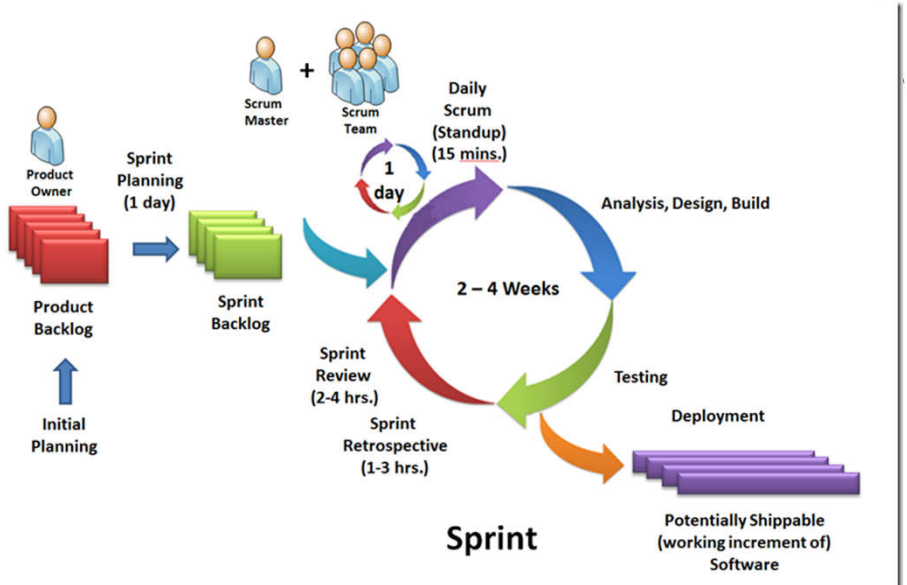
Méthodologie: Vulgairement

- Le produit est séparé en fonctionnalité.
- Chaque fonctionnalité est séparé en tache utilisateur : User Story.
- Les User Story sont regroupé en Epics (ensemble cohérent de User Story).
- Le produit est construit par itération. A chaque itération, ce dernier est fonctionnel.
- Une itération est composé dans Scrum de 1 ou plusieurs Sprint:
  - Temps fixé à l'avance pour coder un ensemble de User Story
  - Généralement de 2 à 4 semaines.
  - La notion d'itération disparait dans Scrum derrière la notion de Sprint.

Méthodologie Vulgairement

- Avant chaque itération, on rajoute ou en enlève des user story de l'itération backlog en fonction du retour métier.
- On choisit pour les Sprint suivants des User Story a exécuter. Ces User Story sont choisit dans l'équipe le matin.
- Ceux-ci sont executer pendant le sprint, le sprint est reviewer et on passe au sprint suivante.
- Chaque matin, une review du Sprint est fait.
  - Et on passe au Sprint suivant
  - Et on passe à l'itération suivante

Sprint/Iteration

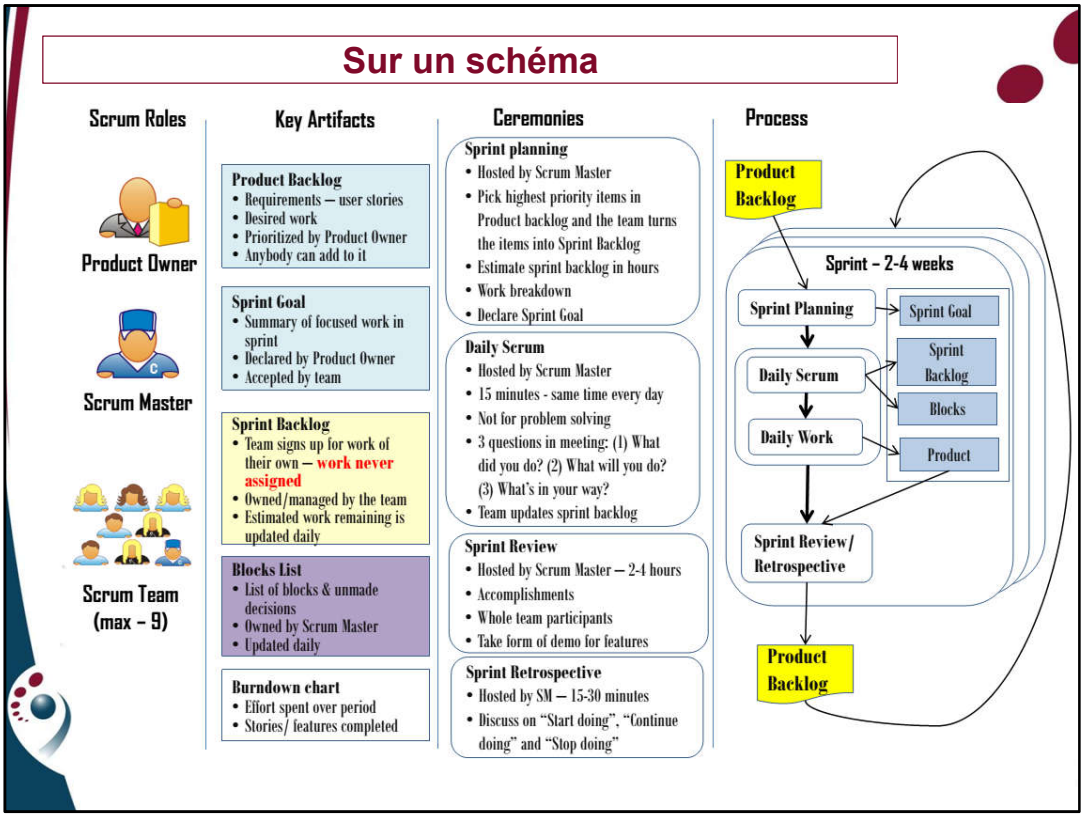


Méthodologie: Les cérémonies

Les cérémonies sont les passages obligés de la méthodologie Agile/Scrum

- La planification du Sprint:
  - Sélection des User Story, Estimation du reste à faire, Définition du but du sprint
- Daily Scrum Meeting:
  - L'idée est de s'accorder 15 mn par jour pour dire où on en est et quels sont les difficulté
- Sprint Review et Prospective:
  - En fin de sprint on regarde où on en est du produit. Qu'est ce qui c'est bien/mal passé et est ce qu'un autre sprint est nécéssaire.

Sur un schéma



Agile Testing vs Cycle en V

- Dans le cycle en V, les tests sont une phase du projet relégué :
  - En fin de projet avec le risque d'un manque de moyen
  - Après les phases de production, et donc avec un risque de non correction.
- Les tests sont souvent non pas issue du besoin métier et exprimer à partir des spécifications.

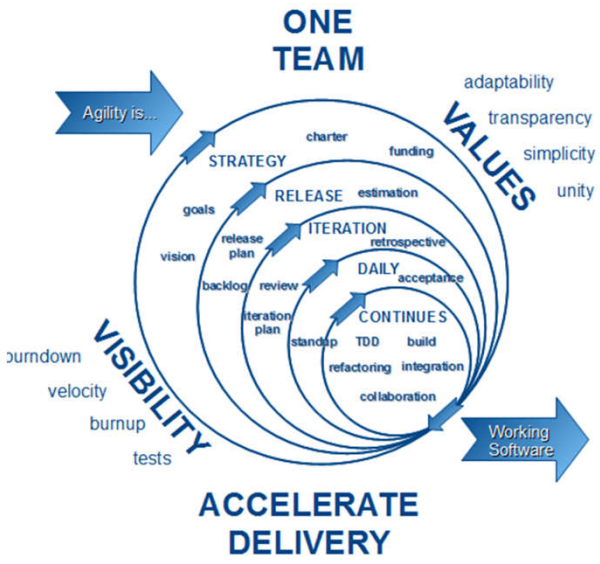
Agile Testing vs Cycle en V

Agile Testing	Test en V
Le test n'est pas une phase distincte et s'execute simultanément avec le développement	Les tests sont une phase distincte placé en fin de projet
Les testeurs travaillent de concert avec les développeurs	Les activités de test et de production sont séparé.
Les testeurs participent avec les métiers à l'écriture des spécifications. C'est cette discussion qui donne les cas de tests métiers	Les tests ne sont pas issue d'une discussion entre les métiers et les testeurs
Ces tests métiers sont executés après chaque itérations.	Les tests métiers ne sont fait qu'en fin de projet.
Une itération n'est finit que si les tests sont executés.	Les tests qui échoue en fin de projet sont souvent basculé en test de regression.
Une partie des tests sont mis en intégration continue. Plusieurs type de tests (métier, unitaire…) sont executé de concert.	La phase de test est sous découpé en test unitaire, fonctionnel … en mode séquencé.
L'activité de test est valorisé	Cela peut être une activité coupé pour raisons budgétaires ou temporelle.

Cycle de vie Agile

- Le cycle de vie Agile consiste à définir:
  - Une vision pour le projet. A quoi sert le produit, pour qui avec quels fonctionnalités.
  - Découper le projet en release fonctionnel. Chaque version du projet répond aux besoins des utilisateurs de façon plus ou moins complete.
  - Une release est séparé en itération dont le but est de fournir un produit fonctionnel.
  - Chaque itération couvrent les fonctionnalités en executant de sous fonctionnalité du produit (User Story).

Cycle de vie Agile



Principes du Test Agile

- L'activité de test est la métrique de l'avancé d'un projet. L'intégration continue et les tests valident la bonne avancé du projet et permet de revenir rapidemment vers le metier avec un produit fonctionnel.
- Le test n'est pas une phase mais doit faire partie du développement. Un developpement n'est terminé que si les tests sont écrit et executé avec succès.
- Toutes l'équipe doit faire des tests (Le testeur étant vu comme un spécialiste). Les developpeurs et les métiers participent à l'activité de test

Principes du Test Agile

- Une itération se finit si et seulement si tous les bugs remonté lors des tests sont corrigés.
- L'activité de test doit pouvoir être exécuté en mode manuel et en mode automatique.
- Les tests utilisateurs (ou d'acceptance) sont écrit en mode Test Driven Developpement. Autrement dit, on écrit d'abord les tests puis le code.

Principe du test Agile

Type de tests:

- Tests unitaires: Tests automatiques exécutés chaque nuit sur le code et à chaque ajout
- Tests d'integration: Tests d'interaction entre les différents modules
- Tests fonctionnels: Tests exécutés lors de la fin d'une fonctionnalité
- Tests système: Vérifie que les contraintes annoncés sur le système sont vérifié
- Test de régression: Tests pour vérifier le comportement du système après un fix
- Test de compatibilité: Tests d'interaction entre le produit et le système.

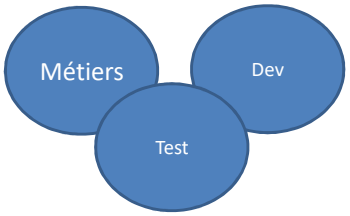
Planing du test Agile

Pendant une itération, l'activité de test agile comprend:

- La participation au planing de l'itération
- L'évaluation de la charge des différentes tâches
- L'écriture des Test Cases en utilisant les informations métiers.
- Testing Unitaire, Integration, Fonctionnel.
- Test metiers
- Reporting
- Suivie des problèmes.

En Pratique : un changement organisationel

Dans les cycles non agile, les tests sont fait par des testeurs associés à la MOE.
La qualité logiciel est une étape « technique »



Dans les cycles agile, les tests sont fait par la totalité des personnes impliqués dans le projet.
Le testeur est vue comme un expert en qualité logiciel.

En Pratique : un changement organisationel

- Le test "Agile" est une pratique de test logiciel qui suit les principes du développement de logiciels agiles.
- Cela implique tous les membres de l'équipe de projet. Les testeurs étant les membres de l'équipe ayant une expertise spécifique.
- Le test n'est pas une phase distincte et est entrelacé avec toutes les phases de développement telles que les exigences, la conception et le codage et la production.
- Les tests se déroulent simultanément au cours du cycle de vie de développement.

  En outre, les testeurs participant à l'ensemble du cycle de vie du développement en collaboration avec les membres de l'équipe multifonctionnelle, la contribution des testeurs à la construction du logiciel selon les besoins du client, permet une meilleure conception, un meilleur code et surtout un produit adapté.
- Agile Testing couvre tous les niveaux de tests et tous les types de tests.

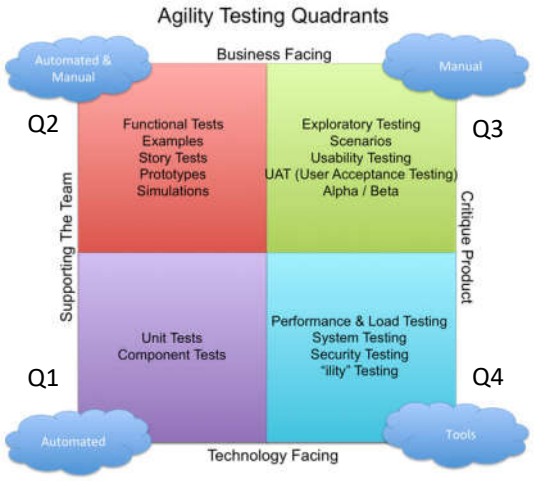
En Pratique: un changement d'exigence

- Il est attendue de l'activité de test et des testeurs:
  - Qu'ils soient impliqués dans toutes les phases du projet.
  - Qu'ils puissent avoir accès aux métiers (la connaissance) et qu'il fournisse cette connaissance aux developpeurs
  - Qu'ils puissent avoir une vision globale du projet ainsi que sur les taches
  - Qu'ils mettent en place les outils afin de voir rapidemment les problèmes.
  - Qu'ils mettent en place la testabiltié des codes
  - Qu'ils restent indépendant et des métiers et des developpeurs

En Pratique: un changement d'exigence

- Le test en Agilité bouge aussi les approches sur la qualité logicielle:
  - La qualité logiciel est le problème de tous et non plus seulement des testeurs
  - Le code doit être testable, et le produit doit être tester par l'équipe complète
  - Il n'y a plus de QA team a proprement parlé. L'activité QA étant plongé dans la totalité de l'équipe.
  - Chaque membre de l'équipe à la même valeur
- L'activité de QA est scindé en 4 parties d'importance égale:
  - L'activité technique de QA
  - L'activité métier de la QA
  - L'activité de la QA de support au groupe
  - L'activité de QA de critique positive du produit

AgileTesting Quadrants



Q1

- Le premier quadrant a pour but de supporter l'équipe et d'être le support technique du produit.
- Ce quarter représente l'héritage du TDD dans la méthode Agile.
- Dans les méthodes Agile, l'activité de test en Q1 est définit comme étant une méthode de design et non de qualité.
  - La qualité est définit comme étant l'équilibre entre celle définit par les developpeurs et les contraintes de temps. Le différentiel est la dette technique.
  - Le code est donc par construction (design) testable.

Q2

- Ce quadrant désigne les tests devant montrer aux utilisateurs la qualité « externe » du produit ainsi que les fonctionnalités.
- On retrouve les tests fonctionels, les tests sur examples, les prototype
- Ces tests sont issue de petit morceau de fonctionnel (dites User Story Testing) écrit avant le codage.
- Ces tests sont semi-automatisé. La tendance est d'automatisé complètement ces derniers.

Q3

- Le troisième quadrant est celui des tests afin de critiquer positivement le produit.
- Elle represente l'appetance des utilisateurs pour le produit.
- Tests d'exploration, Test d'utilisabilté …
- Ces tests ne sont pas automatisé dans la mesure où ils représentent l'interraction des utilisateurs pour le produit.

Q4

- Les tests du Q4 sont les tests non fonctionnels de l'application.
- On retrouve les tests de performances, de sécurité, de sureté, d'installation …
- Ces tâches sont :
  - Difficile à automatiser complètement.
  - Requiert de l'expertise dans les différents domaines (test de sécurité par exemple).

Les quadrants

- Le but de ces quadrants est de regrouper les tests par famille mais :
  - Il n'y a pas de prédominance entre eux. C'est l'adjonction de tous les quadrants qui fait la qualité.
  - Il n'y a pas de responsabilité dans ces quadrants. La qualité logiciel est l'activité de tous et les testeurs sont « simplement » des experts qualité.
- Ils reposent en partie sur l'automatisation (en partie Q1 et Q2) car:
  - Les tests automatiques peuvent tourner à volonté et libèrent donc pour des activité de QA de plus hauts niveaux.
  - Ils servent de méthode de design « externe » pour le développement du produit.
  - Evitent les régressions.

Qu’est-ce qu’un testeur Agile

- Pendant la phase de planification du produit, un testeur doit:
  - Comprendre correctement le contexte métier.
  - Définir les tests de hauts niveaux (Q2,Q3,Q4) pour la fin de l’itération.
  - Aider à la mise en place des tests en Q1 (formation, revue de code)
  - Aider évaluer chaque fonctionnalités, leurs impacts et leurs ordonnancement.

Qu'est-ce qu'un testeur Agile?

- Pendant la phase de planification de l'itération:
  - Réfléchir à comment tester les User Story
  - Regarder comment les User Story s'inscrivent dans les features, et vérifier qu'elles sont bien couverte par les tests de haut niveau.
  - Aider les développeurs à avoir une architecture testable, et si ce cela ne semble pas possible résoudre la problématique.
  - Commencer a faire les test plans Agile et les relire avec les métiers et les developpeurs

Qu’est ce qu’un testeur Agile?

- Pendant la planification du contenue (spécifications des fonctions de l’itération):
  - Il doit s’assurer de la plus value de la fonctionnalité sur le produit
  - Identifier comment la tester de façon unitaire (quitte a découper la fonctionnalité en sous fonctionnalité).
  - Identifier les impacts de la fonctionnalité sur le produit et sur les futurs itérations.

Carte de Test Agile RACI-VS

- Matrice de definition de taches:
  - Le ou les R (le A peut aussi jouer le rôle de R) réalisent l'action. Il doit y avoir au moins un R pour chaque action. Le A s'organise comme il le souhaite pour sous-traiter au(x) R, mais c'est son problème : si les R ne remplissent pas leurs objectifs (ou n'existent pas), c'est le A qui assume.
  - Le A est comme son nom l'indique celui qui doit rendre des comptes sur l'avancement de l'action. Il y a toujours un A (et un seul) pour chaque action. « Avoir le A » signifie être totalement responsable d'une action.
  - Les C sont les entités (personnes, groupes) qui doivent être consultées.
  - Les I sont les entités qui doivent être informées.
  - V pour validateur (généralement les validations qualités)
  - S pour signataire (la validation de l'autorité ou du A de l'activité suivante)

Stratégie de test

- Gestion; les facteurs contraintes/budgets/Délais déterminent la consistance de la stratégie de tests. Il est fondamentale de lister les contraintes, et de les suivre dans un planning. Les ressources techniques et humaines sont jaugées en fonction du budget. La composante « délais » fixe le planning, qui doit être adapté en continue en fonction des contraintes et du pilotage.

Eléments clés d'une validation fonctionnelle

Les fondements. Le processus de recette.

- Objectif d'une recette
- Qu'est-ce que le processus de recette

Définitions

- MTTF (temps moyen de fonctionnement).
  - Si l'on suit n systèmes depuis leur mise en service neuf, et que t(i) est la durée écoulée au moment où survient la première panne, alors le MTTF est la moyenne des t(i)
- MTTR: Temps de réparation moyen d'un disfonctionnement.
  - A priori ce temps est non calculable
  - Définis par contrat/usage

Objectifs

- Un double objectif :
  - Choix N°1 : Augmenter le MTTF
    - Réduire au maximum le taux d'erreurs résiduelles
    - Reconfigurer dynamiquement le système sur des états cohérents malgré le non-déterminisme (c'est une compensation les effets de certaines défaillances connues)
  - Choix N°2 : Diminuer le MTTR
    - Se donner les moyens d'observation des états déterministe du système (élimination systématique des erreurs reproductibles)
    - Réserver des ressources en quantité suffisante pour les tests « en ligne »

En respectant les contraintes de Coût-Délai du projet

- Comment répartir et organiser l'effort de test – Comment choisir

Caractéristique de l'activité de test

- Spécifier et développer les « bons » tests – élaborer les résultats attendus
- Classer, organiser et gérer les tests (gestion de configuration)
- Exécuter les tests dans un environnement ad hoc – Localiser les défauts
- Analyser les résultats obtenus et diagnostiquer les erreurs commises
- Rédaction des rapports d'anomalies RA et envoi des RA aux acteurs impliqués

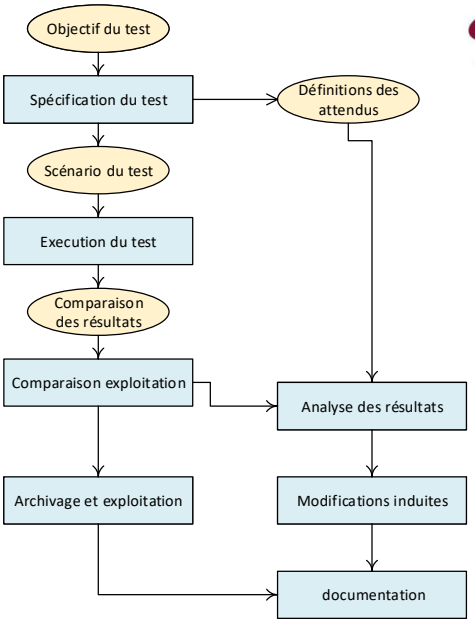
Définition

- Processus de test ou cas de test : spécification de l'ensemble des éléments qui permettent de vérifier une exigence donnée. Le cas de test est caractérisé par : un objectif, des préconditions, des données de tests (jeux d'essai), une procédure détaillée des actions à réaliser pour faire le test et vérifier un ou plusieurs fonctionnalités/exigences, un résultat attendu, des post-conditions et éventuellement des manipulations après tests. La partie procédure du cas de test doit décrire dans le détail comment le testeur va concrètement exécuter les tests. Il faut être précis dans la description de cette partie.

Le Processus de test

Le processus de test consiste à définir :

- un test
- Les attendus « apriori »
- A exécuter le test
- Comparer les résultats à l'attendu



Définition: Campagne de test

- Campagne de test : correspond à l'exécution des tests par rapport à une version de l'application en cours de développement/maintenance en fonction d'un socle technique.
- Il s'agit d'une collection de test

Définition plan de recette

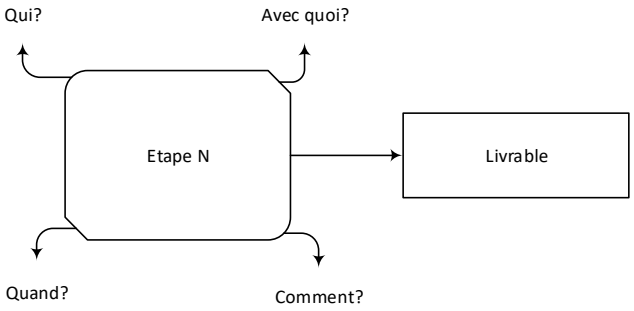
- Un plan de recette est un assemblage de
  - Une collection de campagne de tests
  - Les objectifs, le planning, les ressources,…
  - Des fonctionnalités du logiciels
  - Des analyses de criticités et de priorités
  - …

Le processus de recette

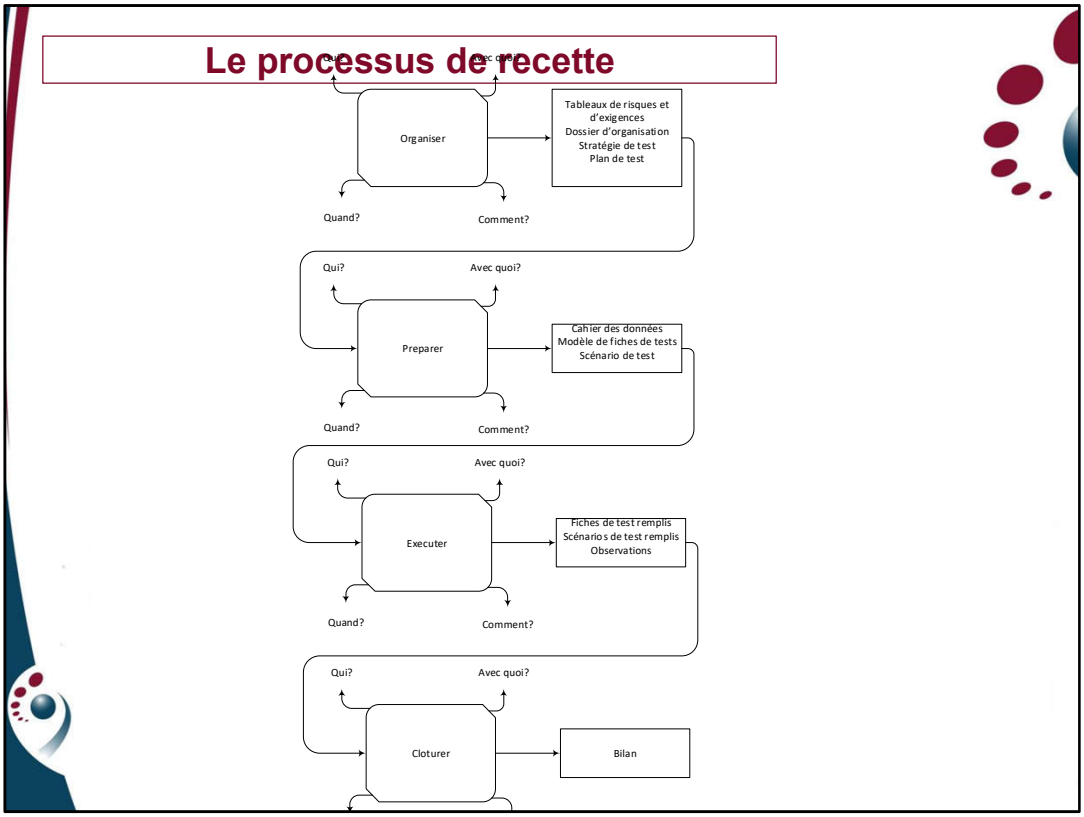
Le processus de recette consiste donc à :

- Comprendre le contexte de la recette
- Organiser les campagnes de test
- Préparer les campagnes de tests
- Exécuter les campagnes tests
- Clôturer les campagnes tests

Utilisation du diagramme de Crosby Turtle Qui, Quand, Avec quoi et Comment?



Le processus de recette



Eléments clés d'une validation fonctionnelle

Le plan de recette.

- Comprendre l'importance du contexte
- Définition des plans de recettes en fonction des contextes

Plan de recette

Pour choisir la bonne stratégie de plan de recette, il est nécessaire de s'équiper si possible :

- D'une liste des exigences avec si possible les criticités
- Un diagramme de ressources
- Du contexte du projet

Tableaux des exigences

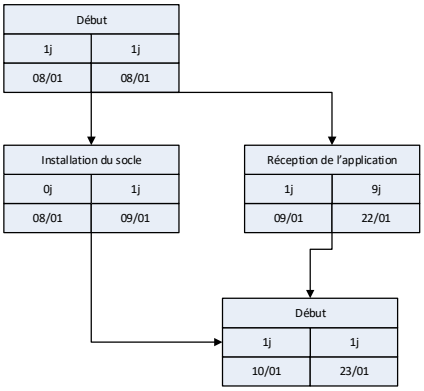
Il s'agit de la liste si possible exhaustive :

- De ce que doit faire un logiciel ?
- Sur quelle plateforme technique ?

Ce tableaux doit être si possible ordonnée en fonction de la nécessité de l'exigence (Must, Could, Should, Would, Ignore)

Diagramme de PERT

Il s'agit de pouvoir donner des jalons approximatifs pouvant modéliser l'usage de ressources critiques. (Eq au diagramme de Gantt)



Plan de recette

Les contextes principaux sont :

- Recette urgente/pompier. Il s'agit d'une recette à faire dans un contexte soit d'inorganisation complète soit d'urgence technique. La plupart du temps il s'agit d'un chantier de run
- Recette patch: Il s'agit d'un correctif d'urgence pour un chantier de maintenance
- Recette technico-fonctionnel : Recette classique d'une équipe MOA/MOE
- Recette fonctionnel : Il s'agit d'une recette MOA de validation de produit

Recette Pompier

- Il s'agit d'un projet dont la situation est préoccupante
- Les recettes n'ont pas été anticipées et les premiers tests montrent un taux d'anomalies anormalement élevé.
- La plupart des anomalies sont bloquantes et empêche la recette

Il s'agit la plupart du temps d'un projet en mode run qu'il faut gérer en Agilité.

Recette Patch

- Il s'agit d'une recette demandé par la MOE suite à un incident de production. Il s'agit donc d'une urgence en maintenance sur un produit « stable »
- Attention: la recette d'un patch doit se synchroniser avec une potentielle recette en maintenance de l'application. De ce fait, le patch seras appliqué soit en production (pas de maintenance) soit en maintenance (et donc un délai de livraison du patch).

Recette technico-fonctionnel

- Il s'agit d'une recette « technique » qui doit être fait avant la recette de validation que le chantier soit en mode run ou maintenance.
- Cette recette est souvent cumulée avec la recette fonctionnelle, et souvent externalisé.
- Parfois cette phase est confondue avec la recette de validation ce qui implique généralement des problèmes.

Recette Validation

- Il s'agit d'une recette non technique faites par le métiers/MOA afin de pouvoir valider le bon fonctionnement de l'applicatif.
- Il s'agit de valider l'aspect métier (règle de gestion par exemple) du logiciel.

Eléments clés d'une validation fonctionnelle

Organisation et suivi. Gestion des jeux de données.

- Définir les différentes organisations de recette en fonction des contextes.

Organisation d'une recette pompier

- La recette est en mode urgence et arrive trop tard. Il n'y a pas de temps pour les diagrammes de Crosby et de PERT ainsi que pour définir les cas de tests.
- La préparation des tests se fait cotée « utilisateur » en utilisant l'interaction pour définir:
  - Le « menu » de l'application
  - Chaque sous-menu de l'application
  - Définir les différents profils utilisateurs
  - Définir les configurations matérielles.
- C'est l'intuition et l'expertise qui peuvent rattraper la situation

Préparation d'une recette pompier

- Utilisation d'Excel en mode partagé est un bon choix.
- Faire le partage directement entre les testeurs et la MOE.
- Souder les équipes
- Créer un tableau double entrée avec en colonne les fonctionnalités utilisateurs triviales et en ordonnée les différentes pages/écran

Préparation de recette pompier

Exigence / Page Web	La page s'affiche-t-elle ?	Peut-on valider le formulaire ?	Tous les liens sont-ils navigables ?	Y-a-t'il une anomalie fonctionnelle triviale ?	Y-a-t'il une anomalie technique triviale ?	Le temps d'affichage de la page est-il acceptable ?
Page d'accueil						
Authentification						
Menu principal						
Menu secondaire						
Plan du site						
[xxx]						
[xxx]						
[xxx]						
[xxx]						
[xxx]						
[xxx]						
[xxx]						
[xxx]						

Organiser d'une recette patch

- Soit il existe un référentiel de test et un tableau des criticités.
  - Dans ce cas il est possible de caractérisé la « criticité » du patch et d'appliquer en fonction de cette dernière les tests fonctionnalités « Must », « Must+Could » ou Must+Could+Should.
- Soit il n'existe pas de référentiel de test, et dans ce cas il faut faire une analyse du code du patch afin de « deviner » les tests de non régression à exécuter.

Organiser une recette technique-fonctionnel

- Deux types de recettes différentes en fonction :
  - D'un chantier de run
  - D'un chantier de maintenance.
- Il s'agit d'une recette mêlant des tests techniques et des tests fonctionnels.
  - Nécessite des compétences techniques : test de robustesse/compatibilité/installation…
  - Et des compétences fonctionnelles pour les tests de cas d'usage, test valeurs clefs…
- Il est nécessaire d'avoir une équipe aguerrie (interne ou externe) pour les mettre en œuvre.

Organiser une recette technique-fonctionnel

- En mode run, il est nécessaire d'intervenir assez tôt dans le projet afin de pouvoir avoir un bon aperçue des exigences, criticités et des risques.
- On mettra en place une matrice des risques produit et des exigences produits (fonctionnelle et technique).
- On met en place les fiches de test, et les scénarios fonctionnels ainsi que les jeux d'essai.

Organiser une recette technique-fonctionnel

- Répertorier les exigences en fonctions de fréquences d'utilisation, des utilisateurs, de l'impact et de la criticité:

Nom	Type d'utilisateur	Fréquence d'utilisation	Impact
BO	Back Office - Webmestre	1-Annuelle	10
Clients	Back Office - Responsable Clientèle	6-Heure	10
Voir les commandes en attente	Back Office - Responsable Clientèle	6-Heure	10
Traiter une commande client	Back Office - Responsable Clientèle	6-Heure	10
Stocks	Back Office - Responsable des stocks	4-Hebdomadaire	8

Organiser une recette technique-fonctionnel

- Chaque exigence est couverte par un ou plusieurs tests ou scénario fonctionnel, il s'agit d'une campagne:

Référence de l'exigence	Cas de test	Priorité du test*
\BO\Clients\Voir les commandes en attente	Liste Cde C vide	1-Must
\BO\Clients\Voir les commandes en attente	Liste Cde C non vide	1-Must

Organiser une recette technique-fonctionnel

- Fiches de tests: Ce sont des tests « technique ».
  - Une partie du test consiste à commencer un cas d'usage, c'est le prérequis.
  - Ensuite une liste de points à vérifier est positionner par plateforme
  - Réutilisable en mode test de non régression

Observez sur l'écran/la page **[xxx]**	Elle / Il contient les éléments graphiques suivants : - Un logo - Un menu principal composé des items * * - Un titre - Etc...
Observez le coin supérieur gauche de la fenêtre	Il comporte une icône d'accès au menu système des fenêtres sous Windows
Observez le coin supérieur droit de la fenêtre	Il comporte les icones de réduction, maximisation et fermeture de la fenêtre.
Passez la souris sur les bordures et coins de la fenêtre	La souris change d'aspect et il est possible de redimensionner la fenêtre.
Observez la barre des tâches de Windows	Elle comporte une icône indiquant que l'application est lancée. Son titre est **[XXXXX]**
Observez le bord droit de la fenêtre.	Il comporte un ascenseur vertical.

Organiser une recette technique-fonctionnel

- Scénario fonctionnel: ce sont des cas d'usage permettant de valider que l'application possède des fonctionnalités.
  - Ces tests sont utilisés pour l'homologation du logiciel et pour les homologations précédentes.
  - Sont souvent des prérequis pour les tests techniques

Lancez l'application **[xxx]**. **OU** Lancez le navigateur et sélectionnez le favori permettant d'accéder à l'application Web **[xxx]**	L'écran demandant votre authentification apparaît.
Saisissez le login et le mot de passe nécessaire à ce test comme défini dans le cahier des jeux d'essai.	Vous êtes authentifié et l'écran/la page d'accueil est affiché(e).
Allez sur l'écran/la page **[xxx]**	L'écran/la page s'affiche normalement.

Organiser une recette technique-fonctionnel

- En mode maintenance, le projet consiste à faire des évolutions fonctionnelles du logiciel.
- Il est donc logique de créer des tests fonctionnels pour les ajouts aux logiciels.
- La priorité est donc donner aux nouveaux scénarios fonctionnels ainsi qu'aux tests de non régression.

Organiser une recette technique-fonctionnel

- Jointure entre les campagnes de tests, les ressources, les configurations matérielles:

Référence du cas de test	Type de test	Charge d'exécution	Priorité du test	Configuration matérielle	Testeur
\BO\Clients\Voir les commandes en attente-Liste Cde C vide	Fiche de test	0,2	1-Must	BO-PC-W7-IE9-FR-1024	Testeur 2 - no DE
\BO\Clients\Voir les commandes en attente-Liste Cde C non vide	Fiche de test	0,2	1-Must	BO-PC-W7-IE9-FR-1024	Testeur 2 - no DE
\BO\Clients\Voir les commandes en attente-Zoom d'une commande C	Fiche de test	0,2	1-Must	BO-PC-W7-IE9-FR-1024	Testeur 2 - no DE

Organiser une recette de validation

- La validation d'un logiciel est une succession de test fonctionnel issue des cas d'usage utilisateur.
- Ces cas doivent avoir été définis dans l'expression de besoin et exécuter avec les métiers.
- Cette validation donne lieu à un bon de validation permettant la mise en production.

Eléments clés d'une validation fonctionnelle

L'enregistrement des résultats et des anomalies.

- Importance des résultats des tests (Valorisation)
- Remonter des anomalies

L'enregistrement des résultats et des anomalies.

- L'enregistrement des résultats et des anomalies est importantes car elles justifient en grande partie l'activité de tests.
- L'enregistrement des anomalies et des résultats se fait en général soit par le biais soit de fiches Word, soit via un outil dédié.
- La remonté de résultats et d'anomalie est une étape essentiel afin :
  - De pouvoir corriger au plus vite une anomalie
  - De pouvoir anticiper la période de maintenance (cas d'un futur bug proche d'une anomalie corrigée)

L'enregistrement des résultats et des anomalies.

- Pour une recette pompier, le mieux est d'utiliser un fichier Excel partagé entre les développeurs et les testeurs
  - Il n'y a pas de temps pour capitaliser les résultats

- En recette normal, il est essentiel de faire remonter:
  - Le numéro du test
  - La configuration matérielle testé
  - Le nom de la personne qui exécute le test

En cas d'anomalie

- En sus, si le test échoue:
  - Il faut indiquer précisément l'erreur
  - Une série d'information des étapes de reproduction du bug
  - Une qualification du bug (bloquant, gênant…)
  - Les logs de l'application, dump mémoire ….

Eléments clés d'une validation fonctionnelle

Le processus de packaging et de livraison.

- Comment tester le packaging et la livraison
- Les clauses de recette.

Le processus de packaging et de livraison

- Un package est un ensemble contenant :
  - Le logiciel proprement dit, exécutable et validé.
  - La documentation d'installation et d'utilisation du logiciel.
- La livraison est un ensemble :
  - Du package logiciel.
  - Du bon de validation client montrant le bon fonctionnement de l'application.
  - De la signature du client de la bonne réception du package.
  - D'une équipe prête à recevoir le package.

Le processus de packaging et de livraison.

- Le processus de packaging est signe la quasi fin du projet.
- Il s'agit de donner forme au logiciel de façon pouvoir faire une installation.
- Le cahier des charges est utilisé à ce moment afin :
  - D'isoler les configurations matérielles devant être supporté
  - Les droits utilisateurs nécessaires
- La documentation d'installation doit être suffisamment explicite au regard des capacités des utilisateurs.
  - Elle doit contenir les prérequis techniques avant l'installation
  - La démarche d'installation
  - Les tests de bon fonctionnements

Le processus de packaging et de livraison.

- Les tests de bon fonctionnement sont des tests fonctionnels simples couvrant les cas d'usages « classiques ».
- Dans le meilleur des mondes, ces tests sont automatisés
- L'idée est de prendre autant de machines que de configuration voulue et de créer les comptes utilisateurs d'installation.
  - L'usage des machines virtuelles est la bienvenue.
- Ensuite, si possible le processus d'installation est « scripté »

Le processus de packaging et de livraison.

- Avec les machines et les scripts d'installation, le logiciel est installé.
- Les tests de validation du paquet sont exécuté

- Les problèmes proviennent souvent:
  - Du manque de disponibilités des ressources matérielles. (Ex serveur Tomcat 8 avec base de données Teradata sur HP Unix).
  - Du nettoyage des ressources matérielles
  - De la difficulté à exécuter les scripts
  - De la distance entre l'environnement de test (machines virtuelles par exemple) avec l'environnement client (machine physique)

Le processus de packaging et de livraison

- La livraison est la fin du projet en tant que tel.
- Il s'agit ensuite de vérifier les clauses contractuelles de fin de contrat par rapport au logiciel:
  - S'assurer de la fin de l'ensemble des travaux, incluant les tâches en sous-traitance
  - Validation du client comme quoi il a reçu le produit/système et les autres livrables
  - S'assurer que la documentation est à jour et que les rapports de clôture ont été réalisés (si requis)
  - Régler les dernières transactions financières (facturation)
  - Relocalisation du personnel, des équipements, des matériaux
  - Consolider la documentation à conserve

Enjeux économiques et techniques des métiers du test

- Historique et définitions
- Opportunités du test aujourd'hui
- Eléments clés d'une validation fonctionnelle
- Coûts et rentabilité du projet de test
  - L'équipe de validation, le besoin en ressources. La professionnalisation du métier de testeur.
  - Estimation du coût des tests par famille de test. Coûts et gains.
  - Coûts et gains de la non-régression. Estimation du coût de la maintenance des tests.
  - TRA pour la maintenance des tests. Les modèles offshores. Rentabilité du test.
- Les approches du test aujourd'hui
- Outils et Infrastructure de test Agile
- Les solutions de gestion des tests
- Synthèse

Coûts et rentabilité du projet de test

L'équipe de validation, le besoin en ressources. La professionnalisation du métier de testeur.

- Besoin en ressources humaines
- Besoin en matériel
- Certification et professionnalisation.

L'équipe de validation, le besoin en ressources

- L'équipe de validation se construit donc avec:
  - Un chef de projet de test ayant pour charge la création d'un plan de recette et l'organisation des tests.
  - En charge avec le chef du projet de planifier les matrices de risques produits et des risques projet.
  - Un testeur ayant la capacité de vérifier les fonctionnalités.
  - Un testeur pouvant valider les fonctionnalités.
- La charge de recette doit occuper entre 20% et 30% des ressources humaines d'un projet.

L'équipe de validation, le besoin en ressources

- Le besoin en ressources est aussi un besoin matériel couvrant la prise en charge du logiciel (OS, navigateur, base de données …).
- Pour les applications multilingues, il est essentiel d'avoir des testeurs ayant une compétence dans la langue testé.
- De façon règlementaire, il est essentiel de s'équiper d'outil pour les problèmes d'accessibilité.

La professionnalisation du métier de testeur

- Le test ou la qualité logicielle est un vrai métier quasi reconnu et sous découpé bien souvent en :
  - Testeur : la personne en charge de créer les tests et de les valider.
  - Dev Testeur: Des développeurs ayant pour but de créer des outils de tests automatiques.
  - Chef de projet Test: La personne en charge de mettre en place les plans de tests et la stratégie de recette.
- Une certification possible est ISTQB

ISTQB

- L'**ISTQB** (en anglais « *International Software Testing Qualifications Board* ») est le *Comité international de qualification du test logiciel*. Cette organisation propose une certification à vie reconnue dans le monde entier.
- L'ISTQB propose trois niveaux de certification :
  - Le niveau fondamental
  - Le niveau avancé : contient trois modules avec chacun un examen et une certification distincte « manager de test », « analyste de test » et analyste technique de test
  - Le niveau expert : contient quatre modules avec chacun un examen et une certification distincte « l'amélioration du processus de test », « le management du test », « automatisation de test », « test de sécurité »

Coûts et rentabilité du projet de test

Estimation du coût des tests par famille de test. Coûts et gains.

- Quel est le coût des tests?
- Modèle de COCOMO?
- Coûts et gains par famille de test

Définitions

- MTTF: Durée moyenne de bon fonctionnement d'un logiciel
- MTTR: Durée moyenne d'indisponibilité du logiciel.

- L'idée est :
  - D'augmenter au maximum la MTTF et de réduire le MTTR.
  - D'avoir un cout de test « bornée »

Estimation du coût des tests

- Parmi tous les tests possibles, identifier ceux qui sont véritablement pertinents pour le système à tester
  - Objectif et critère de test explicite
- Pour un coût donné, trouver le maximum de défauts (rentabilité, rendement)
  - NB : Coût = effort humain + outillages + plates-formes
- Capitaliser ce qui est répétitif et automatiser si le coût de l'automatisation est compatible avec l'économie du système
  - Coût de l'automatisation << Coût humain (projet et/ou coût complet)

Estimation du coût des tests

- Pour le maître d'ouvrage qui représente le client, il faut toujours raisonner en coût complet
  - Le MOA doit intégrer dans son analyse économique l'exploitation, le support et la maintenance, en plus du développement–l'investissement test doit être géré comme un livrable du projet
- Pour le maître d'œuvre et a fortiori le chef de projet, la tendance sera de raisonner en coût projet sur une version du système
  - La vision du MOE est bornée à celle de son projet – Son action s'arrête dès que la recette est prononcée
- Quand faut-il arrêter les tests
  - Mesure et/ou estimation de la maturité du point de vue de l'usager

Perception de l'usager

- Défaillances et pannes perturbent plus ou moins fortement le fonctionnement de l'organisation cible
- Il faut soigneusement distinguer :
  - Le coût de la réparation du point de vue du fournisseur, tel que vu par le chef de projet
  - Le coût de l'interruption de service tel que perçu par l'usager
    - Un cas extrême : ARIANE 501
  - Le rapport de ces deux coûts peut être de plusieurs ordres de grandeur
- Il est prudent de considérer le début de l'exploitation comme la fin du projet

Estimation de l'effort de test

- Le nombre de défauts introduits est une fonction croissante de la taille du référentiel de programmation
  - Documents de conception et volume de code réellement écrit par les programmeurs
- L'effort de test est une fonction croissante de la taille du référentiel de programmation ET du degré d'organisation de ce référentiel

Effort de test

- Formule de Boehm pour l'effort de test

$$Effort = \underbrace{k}_{\text{Terme linéaire}} \times (KLS)^{\underbrace{1+\alpha}_{\text{Terme NON linéaire}}}$$

Facteurs d'influences

Facteurs de coût

Dépend de la puissance expressive et du « grain » sémantique du langage
+ ***Expérience des acteurs***

Facteurs d'échelle

Dépend de la complexité de l'application et de la maturité du processus de développement
 α est le ***facteur d'intégration***

Estimation de l'effort de test (COCOMO)

- Coûts, Qualités, Fonctionnalités, Délais

C

Q

F

D

$$Effort = k \times (KLS)^{1+\alpha}$$

$$D_{\text{en année}} = (0{,}5 \pm 0{,}04) \times (E_{\text{en HA}})^{0{,}3 \pm \varepsilon}$$

Rendement de l'effort VVT exprimé en termes de :

- **Taux de défauts résiduels**
- **Disponibilité de l'application ou du système**

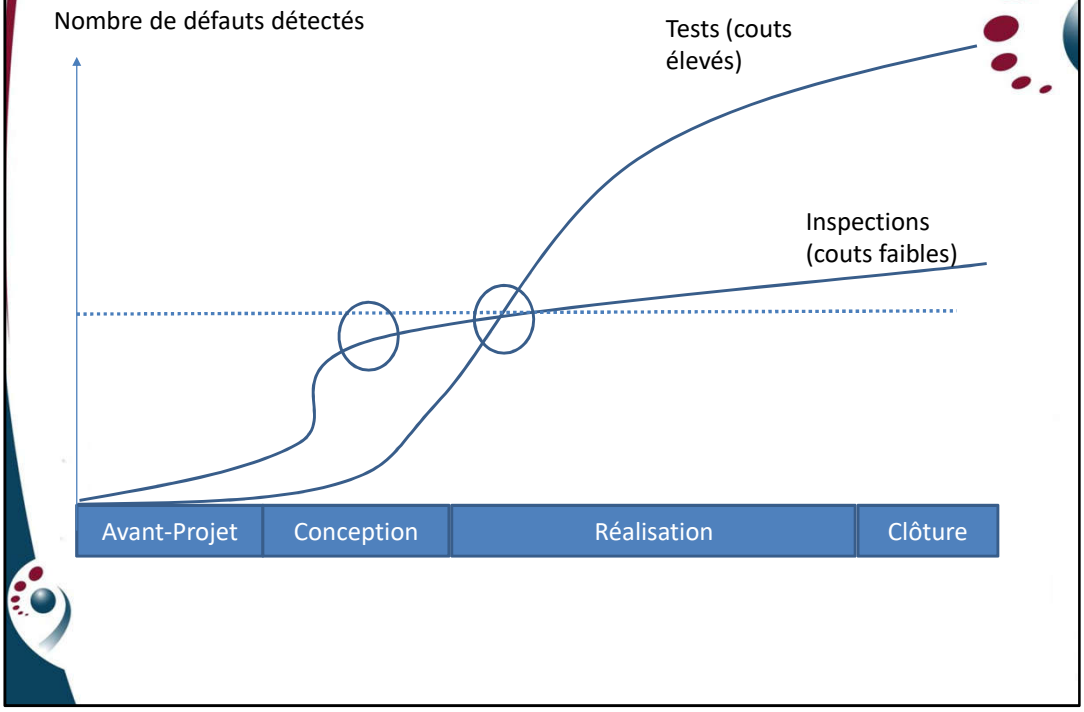
Moyen de réduire l'effort de test

- Architecture testable
  - Créer les conditions d'observation des états du système que l'on saura interpréter et reproduire pour améliorer le diagnostic
- Évaluation des caractéristiques non fonctionnelles (i.e. réduire le facteur d'incertitude)
  - Exemples : Déterminer les performances et la fiabilité véritablement souhaitables ; Valider l'ergonomie avec les usagers REELS ; etc.
- Équilibrage entre :
  - Techniques AQ : revues, inspections, audits,
  - Tests Boîte Noire et tests Boîte Blanche
  - Tests de robustesse et tests d'innocuité/sûreté

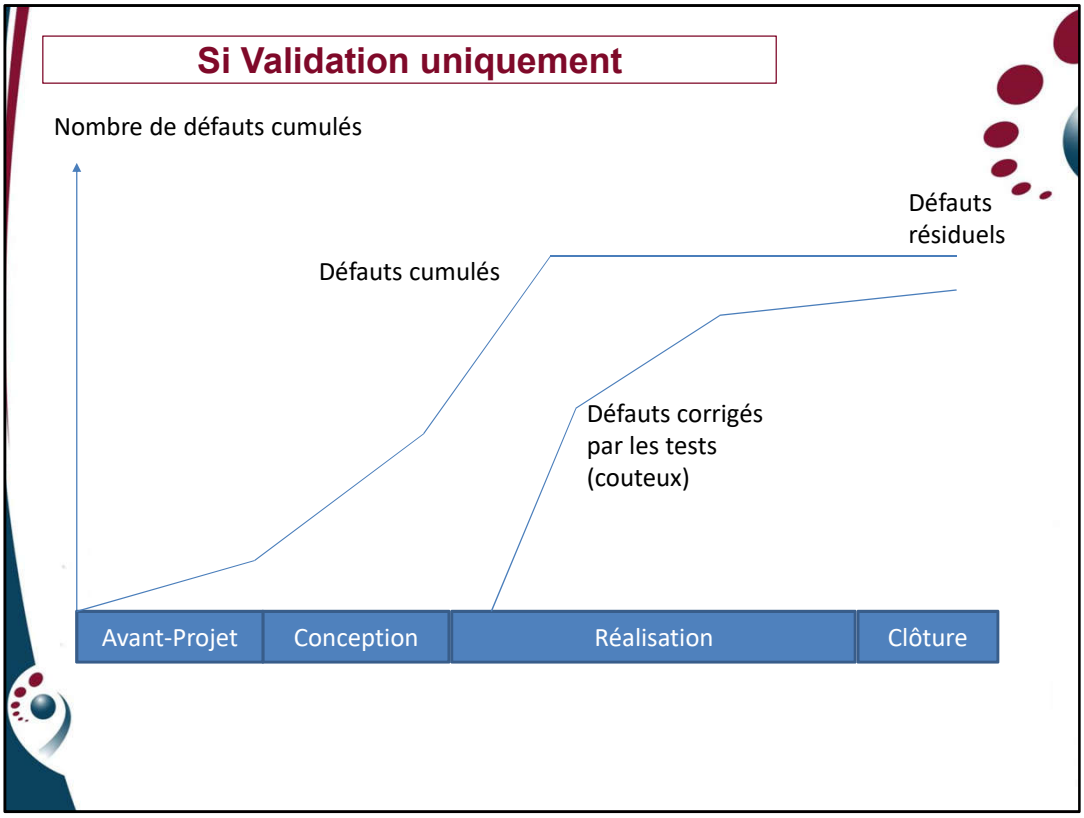
Moyen de réduire l'effort de test

- Le cout des tests est donc d'abord une fonction de l'intégration de modules entre eux.
- Une solution pour réduire cet effort est de faire:
  - Les tests structurels en boites blanches (fait par les développeurs)
  - Les tests fonctionnels en boites « blanches » pour valider les couvertures de code.
  - Faire les tests d'installation en boite noire.

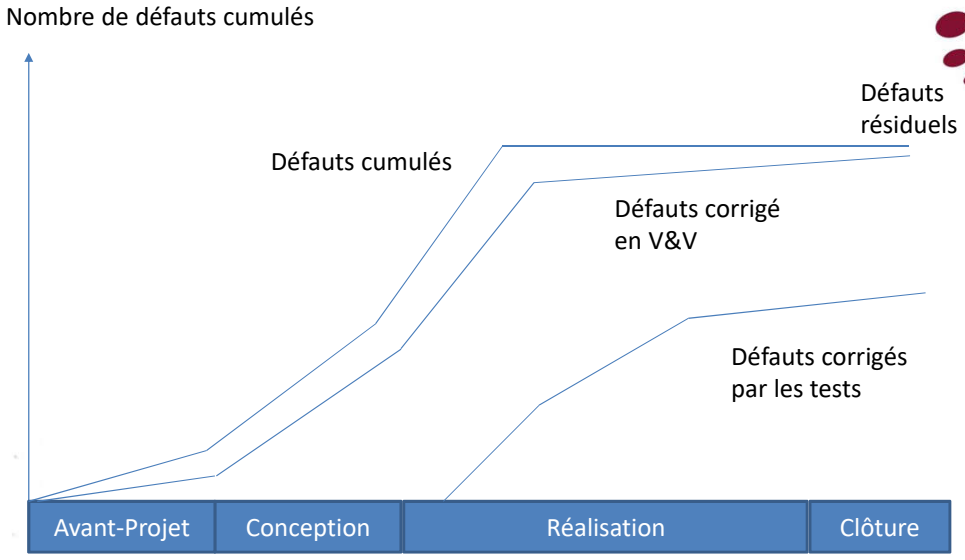
Moyen de réduire l'effort de test le V&V



Si Validation uniquement



Moyen de réduire l'effort de test le V&V



Early bug avoidance

- Il est donc recommander de faire de la détection de bug le plus rapidement possible.
- La solution est sensibiliser chaque équipe à la qualité et de commencer l'intégration le plus tôt.
  - Faire de la revue de code à 2
  - Créer les interfaces d'intégrations au plus tôt

Coûts et rentabilité du projet de test

Coûts et gains de la non-régression. Estimation du coût de la maintenance des tests.

- Problématique de la régression?
- Calcul du cout complet de la régression.

Coûts et gains de la non-régression.

- Chaque mise à jour de l'application nécessiterait de refaire l'intégralité des tests de non régression. Le “time to market” et le “juste coût “rendent cette option impossible. L'entreprise prend alors des risques pouvant aller jusqu'à des régressions sur les fonctions critiques comme les ventes ou les prises de commandes.
- Problématique du cout des tests de non régression et donc de la maintenance de ces tests

Coûts et gains de la non-régression.

- Néanmoins le cout de la régression en terme technique se calcule ainsi:
- Fréquence de la défaillance : liée au taux d'erreurs résiduelles, effet de parc (variété/nombre des configurations installées)
- Coût de la réparation: dépend de l'architecture du système (par exemple : avec ou sans dispositif de journalisation, avec ou sans gestion de configuration, etc.)
- Coût de la correction : très dépendant de l'automatisation des tests (cf. caractéristique de maintenabilité du logiciel)
- Coût de l'installation : dépend de l'architecture du système (par exemple : avec ou sans édition de liens dynamique, avec ou sans moniteur de machines virtuelles, etc.) + effet de parc
- Coût de l'interruption de service : la non disponibilité du système peut induire des pertes qui doivent être comptabilisées (dommages et intérêts)

Coûts et gains de la non-régression.

- Le cout d'usage de la régression :
  - Possibilité d'impact catastrophique (Aviation, Ariane 5 …)
  - Perte d'image de la société
  - Perte d'activité
  - ..

Coûts et gains de la non-régression.

- Le coût de traitement d'une erreur dépend fortement du temps de latence (Introduction/Découverte) :
- Plus le temps de latence est long, plus le coût de la correction est élevé
- Toute erreur non détectée peut occasionner d'autres erreurs (amplification)
- Avec l'augmentation de complexité, seule une stratégie préventive est gagnante

Coûts et gains de la non-régression.

- Le cout est globalement égal à la somme des nouveaux bugs plus:
  - La somme des anciens bugs non trouvé
  - Fois un facteur d'amplification (dette techniques, syndrome du bug cachant la forêt)
- L'efficacité de la détection dépend de la documentation, des standards, de l'organisation qualité et de l'expérience de l'équipe de recette

Coûts et rentabilité du projet de test

TRA pour la maintenance des tests. Les modèles offshore.

- Qu'est-ce que la TRA/TMA?
- Caractéristiques, Inconvénients et Avantages

TRA pour la maintenance des tests.

- TMA (Tierce Maintenance Applicative):Une prise en charge totale de la maintenance de vos applications web ou métier en prévoyant des engagements sur les moyens, les coûts, la qualité et les délais.
- TRA (Tierce Recette Applicative):équipe spécialisée dans les activités de tests fonctionnels et techniques pour assurer une qualité de des applications et en assurant une maîtrise des coûts et des délais de livraison.

TRA

- Il s'agit pour l'essentiel d'un mode d'organisation mutualisé du travail de test et de recette, où différentes entreprises clientes s'adressent à une même équipe de la SSII, ce qui permet de réduire les coûts liés à une telle structure.
- Une même entreprise n'a pas nécessairement besoin d'une équipe de testeurs permanents et dédiés. Un même testeur peut parfaitement travailler pour 2 projets de 2 clients différents, mais pas au même moment. Ceci permet d'équilibrer la charge de chaque testeur, d'écrêter les pics de charge et de combler les creux d'activités, tout en réduisant les coûts via l'optimisation de "l'inactivité".

Caractéristiques

- Transfert de la responsabilité: La société de TRA assure la qualité du logiciel ainsi que la maintenance. Cependant, l'utilisateur demeure responsable de l'usage des logiciels et des résultats obtenus.
- Maîtriser et réduire les coûts de maintenance (meilleure maitrise des budgets)
- Améliorer l'organisation du personnel informatique grâce aux gains de flexibilité
- Assurer un maintien des conditions opérationnelles, ce qui passe par l'apport de corrections puis par une adaptation aux évolutions
- Garantir une bonne utilisation des outils grâce aux conseils et aux supports de professionnels, ce qui passe notamment par la formation
- Faciliter la gestion des ressources et/ou des infrastructures
- Optimiser les ressources informatiques internes
- Améliorer le niveau de service
- Recentrer les équipes internes sur d'autres projets
- Maîtriser les technologies
- Transférer la responsabilité technique et managériale d'un logiciel
- La catalyse d'une maintenance industrielle déployée (nationale, locale...) nécessitant des compétences accrues en termes de SI que l'entreprise ne maîtrise pas.

Avantages de la TRA

- Une expertise technique beaucoup plus pointue et une compétence pluridisciplinaire
- Une organisation qui combine le support et le développement des ressources, les ressources du sous-traitant et celles de l'entreprise cliente étant mutualisées (optimisation)
- La TMA est un gage de réactivité
- Une infrastructure d'hébergement adaptée à la structure et à l'environnement de l'entreprise cliente
- Des procédures de sécurité et de confidentialité des données fournies, ces procédures étant fondées sur des dispositions légales notamment l'A.N.S.S.I.(Agence Nationale de la Sécurité des Systèmes d'Information) et la C.N.I.L. (Commission Nationale de l'Informatique et des Libertés) auxquelles s'ajoute la rédaction de clauses de confidentialité dans les contrats
- Un recentrage sur le cœur de métier pour l'entreprise cliente
- Une augmentation de la qualité du service car l'entreprise fournisseur est un spécialiste
- Éviction des problèmes de début de vie des mises à jour des socles de recettes

Inconvénients de la TRA

- La dépendance aux sous-traitants. En effet, ces derniers veulent conserver leur savoir-faire en limitant volontairement le conseil et la transmission d'informations à l'entreprise sous-traitée afin de préserver un chiffre d'affaires futur
- Le non suivi des demandes de maintenance dans les différents services entraîne un alourdissement des budgets du fait des augmentations des coûts de maintenance pour mettre à jour les logiciels qui n'ont pas fait l'objet d'une mise à jour régulière
- La perte de la maîtrise du SI causée par une sous-traitance trop importante. En effet, en faisant appel à celle-ci, l'entreprise cliente risque de perdre ses compétences internes
- Clause de confidentialité rompue : perte ou revente de données confidentielles. Possibilité de dispersion des données sur le SI qui peuvent être interceptées ou piratées
- Lenteur des traitements des mises à jour demandées par les utilisateurs et donc des recettes. En effet, il est préférable de centraliser les demandes de mises de jour pour éviter une explosion du coût. La demande de mise à jour devra donc « remonter » vers la direction du SI de l'entreprise qui examinera la demande. Si elle est acceptée, elle sera alors communiquée à la SSII. Cette procédure prend plus de temps que si l'utilisateur pouvait directement demander à un membre de la fonction informatique de lui installer la mise à jour
- Possible perte d'efficience. Les mises à jour mineures ne doivent être faites qu'une fois par an. Ce qui suppose que le consommateur ne pourra utiliser ces nouvelles mises à jour avant un certain temps. Il faudra donc mettre en place un système de formation du personnel adapté
- Baisse de la qualité du service due à la fois à l'insuffisance d'informations transmises par la société cliente et l'éventuelle inadaptabilité des méthodes de travail de la SSII à cette même entreprise. Par conséquent, la performance attendue peut ne pas être constatée
- Risque de piratage informatique.

Damage control de la TRA

- Pour éviter la baisse de la qualité de service il faut prendre soin de la qualité des informations transmises, détailler les contrats en y intégrant des pénalités en cas de non-respect de certaines clauses ou conventions et prévoir une dépense suffisante pour rémunérer la qualité de service exigée (environ 10 % du budget selon l'évolution environnementale).
- Pour lutter contre la dépendance vis-à-vis de la société de tierce maintenance, il faut garder une certaine maîtrise des applications informatiques et restreindre le champ d'action de la société en mettant en place un suivi contractuel permanent. Cela va de pair avec une rédaction et une structure de contrat très rigoureuse.
- Afin d'éviter l'alourdissement des budgets il est nécessaire que les utilisateurs n'aient aucun lien avec la société de TRA car cela démultiplierait le coût de recette par le nombre d'utilisateurs. En effet il vaut mieux globaliser la demande en passant, notamment, par la hiérarchie, ce qui entraînera un coût unique pour un service unique. De plus, il est préférable de regrouper les évolutions fonctionnelles mineures une fois par an ce qui minimisera le coût de l'adaptation. Il paraît aussi nécessaire de prévoir les évolutions majeures qui auront préalablement été intégrées dans des plans annuels de développement. Ainsi la réactivité n'est nécessaire quant à la maintenance que lorsque les évolutions sont majeures et non prévues dans les plans annuels.
- Pour lutter contre la perte de la maîtrise des applications, il est conseillé de garder un chef de projet interne qui sera l'intermédiaire entre l'entreprise cliente et la société de TRA . Ainsi, la société de TRA n'aura aucun pouvoir sur la maîtrise d'ouvrage et cela réduira également la dépendance dont elle pourrait bénéficier. De plus, l'entreprise cliente doit contrôler les actions de la TRA afin de s'assurer que celle-ci fait bien ce qu'elle doit faire. L'objectif est d'assurer la transparence des actions faites par cette dernière.
- Assurer la sécurité de la transmission des informations : il faut éviter une fuite d'informations lors de la transmission entre la société de TRA et la société cliente. Il devra donc être mis en place un système adapté qui garantira la sécurité des transferts.

Modèle OffShore

- La TRA en modèle OffShore est une TRA pratiqué dans un pays dont le cout de main d'œuvre est jugé faible.
- Nécessite en sus des contraintes de la TRA:
  - D'avoir du personnel qualifié sur place
  - De pratique du management interculturelle
  - Etre sûr de la compétence linguistique des interlocuteurs.
  - S'équiper d'un cabinet d'avocat permettant de contractualiser les choses.
  - Faire un reporting permanent
  - Gestion des problèmes d'image de la société cliente

Enjeux économiques et techniques des métiers du test

- Historique et définitions
- Opportunités du test aujourd'hui
- Eléments clés d'une validation fonctionnelle
- Coûts et rentabilité du projet de test
- Les approches du test aujourd'hui
  - Criticité et niveau de confiance. La testabilité d'une application. La notion de couverture des tests.
  - L'approche par les risques.
  - Les apports des approches Agiles (Scrum, XP...).
  - Le Test-Driven Development (TDD).
  - La maturité des processus (TMMI, Test Process Improvement, ISO/SPICE).
- Outils et Infrastructure de test Agile
- Les solutions de gestion des tests
- Synthèse

Les approches du test aujourd'hui

Criticité et niveau de confiance

- Définition de la criticité d’une application
- Quel sont les niveaux de criticités.?

Criticité et niveau de confiance

- Les documents contractuels, tels le contrat et les avenants à celui-ci, les cahiers des clauses techniques particulières (CCTP), cahiers d'appels d'offre, etc. ;
- Les spécifications de l'application, les spécifications générales et détaillées, l'architecture des composants, l'organisation de la base de données ou des systèmes de fichiers ;
- Les documentations des utilisateurs, de maintenance, d'installation, etc. ;
- Les risques identifiés ou associés à l'utilisation ou au développement de l'application ;
- Les standards applicables, qu'ils soient spécifiques à l'entreprise, applicables à la profession ou spécifiés contractuellement

Criticité et niveau de confiance

Les niveaux d'intégrité, définis dans le standard IEEE 829:2008 *7* sont :

- Niveau 4 : catastrophique : le logiciel doit s'exécuter correctement ou des conséquences graves (perte de vie humaine, perte du système, dommages environnementaux, pertes économiques ou sociales) auront lieu. Il n'y a pas de possibilité de mitigation ;
- Niveau 3 : critique : le logiciel doit s'exécuter correctement ou l'objet (mission) du logiciel ou du système ne sera pas réalisé, causant des conséquences sérieuses (blessures permanentes, dégradations majeures du système, dommages environnementaux, impact social ou économique). Une limitation partielle ou complète de ces impacts est possible ;
- Niveau 2 : marginal : le logiciel doit s'exécuter correctement ou une fonction attendue ne sera pas réalisée, causant des conséquences mineures. Une limitation complète des impacts est possible ;
- Niveau 1 : négligeable : le logiciel doit s'exécuter correctement ou une fonction attendue ne sera pas réalisée, causant des conséquences négligeables. Une mitigation n'est pas nécessaire.

Les approches du test aujourd'hui

Testabilité d’une application

- Définition de la testabilité d’une application.
- Pourquoi la testabilité d’une application est essentielle?

La testabilité

- Si la validation est la vérification du respect d'un logiciel au regard de sa spécification.
- Un certain nombre de bugs sont trouvé pendant les tests et pendant la mise en production.
- Il y a donc une notion de détectabilité des bugs par les tests, ce qui donne la notion de facilité à tester un logiciel

Testabilité

- Lors d'un dysfonctionnement d'un système complexe (char, avion, satellite…), il est essentiel pour des raisons de sécurité et d'efficacité, de pouvoir détecter et localiser la (ou les) panne(s) dans les plus brefs délais afin de compenser les fonctions de(s) l'équipement(s) défaillant(s).
- D'une manière générale, la testabilité est l'une des composantes de la sûreté de fonctionnement et contribue à la maintenabilité et à la disponibilité d'une entité. Elle joue un rôle prépondérant dans la conception d'un système complexe et que sa participation doit se faire dans la phase aval de l'étude afin de déterminer les équipements qui satisferont aux mieux aux exigences de testabilité. Ces exigences sont des exigences de détection et de localisation des pannes qui sont demandées par le client et doivent être respectées. Elles sont calculées de la manière suivante :

La testabilité

- La testabilité :
  - Est un facteur qualité du système
  - N'est pas le test
- Quand étudier la testabilité?
  - Le plus tôt possible dans la conception du logiciel
  - Afin d'anticiper des couts de post production
  - Accentuer l'effort de test sur les parties les plus (ou les moins) facile à tester

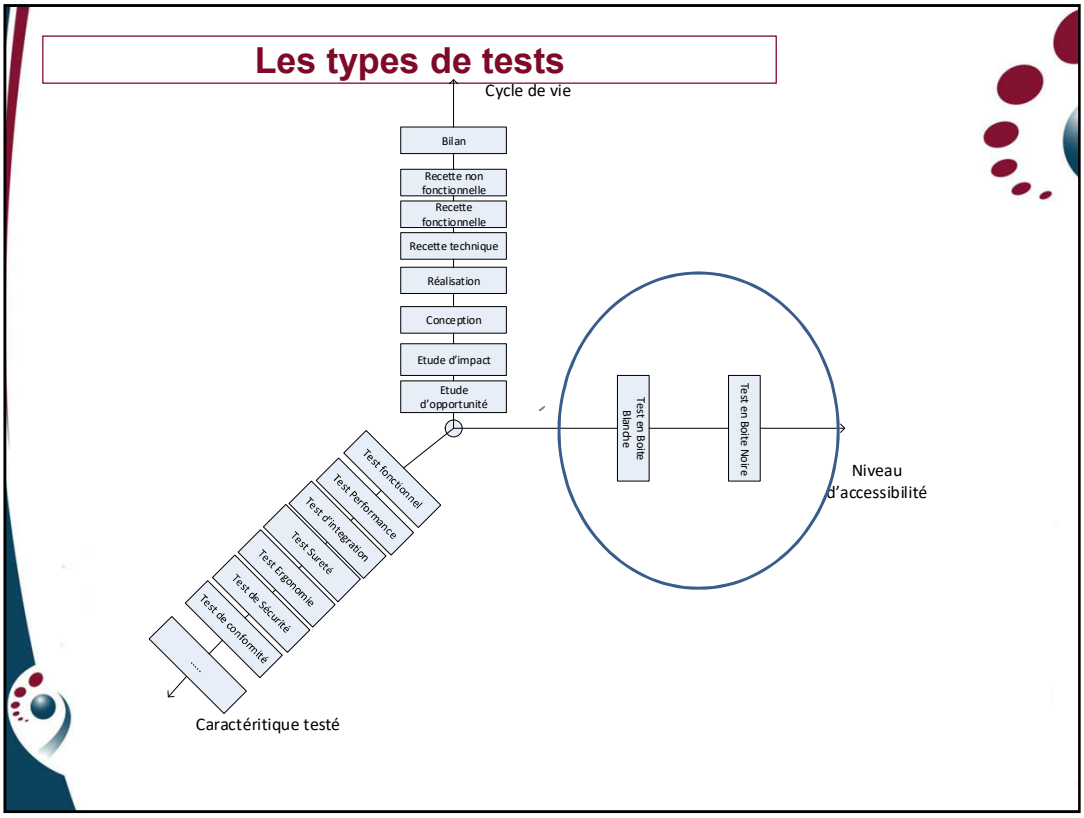
Testabilité

- LOI 1 : toute méthode de tests laisse un résidu d'erreurs contre lesquelles la méthode adoptée est inefficace.
  - Corollaire : Le potentiel de détection des défauts d'une suite de test s'épuise → Il faut constamment renouveler les tests en changeant de point de vue (objectif et stratégie de test).
- LOI 2 : l'accroissement de complexité des systèmes dépasse très facilement le niveau de complexité que l'on sait raisonnablement valider, vérifier et tester.
- LOI 3 : code et données entretiennent des relations de dualité ; le code se transforme facilement en données. Les données sont une source d'erreurs aussi importante que le code.
- LOI 4 : la topologie des défauts passe progressivement de l'état « dense » à l'état « diffus » ; l'analyse locale devient globale. Les « heisenbugs » deviennent prépondérants

Les approches du test aujourd'hui

La notion de couverture des tests.

- Qu'est-ce qu'un test structurel? Fonctionnel?
- Comment trouver les bons tests structurels?
- Comment calculer la couverture d'un test?


Les types de tests
Cycle de vie
Bilan
Recette non fonctionnelle
Recette fonctionnelle
Recette technique
Réalisation
Conception
Etude d'impact
Etude d'opportunité
Test en Boite Blanche
Test en Boite Noire
Niveau d'accessibilité
Test fonctionnel
Test Performance
Test d'integration
Test Sureté
Test Ergonomie
Test de Sécurité
Test de conformité
...
Caractéritique testé

Les types de tests

- Il y a trois grands types de tests par version de produit:
  - Les tests dit structurelles appelé souvent test en boite blanche car unitaire et technique.
  - Les tests fonctionnels exécuté en en boite noir
  - Les tests non fonctionnels. Il s'agit de test de performance, robustesse…
- Ces trois types de tests sont complémentaires et vérifie respectivement qu'un logiciel ne contient pas d'erreur, fait ce qu'il doit faire, et le fait « correctement »
- Enfin les tests dit de non régression devant montrer que ce qui fonctionnait avant…. fonctionne encore

Tests Structurels

- Les tests structurels s'appuie sur le code de l'application (d'où le surnom de test en boite blanche) pour faire ressortir des cas de tests.
- Ces cas de tests sont censé aboutir à une bonne couverture du code en terme de :
  - Nombre de chemin couvert
  - Nombre de débranchement couvert
  - Nombre d'instruction couvert.
  - Espace des données couvert
- L'hypothèse forte est que si un code est couvert à 100% alors il ne contient pas d'erreur. (A ne pas confondre avec ce code fait ce qu'il doit faire)

Tests Structurels

```
public Integer Sum(Integer x, Integer y)
{ int xx=x==null?0:x;
Int yy=y==null?0:y;
return xx+yy;
}
```

- L'espace de données est l'espace des entiers + la valeur null (Integer est objet).
- L'ensemble des chemins est égal à l'ensemble des débranchements
- Les cas de test critiques sont (null,0),(0,null),(null,null).

Tests structurels

- Si les tests structurels sont pratiqués de façon « parfaite » alors ils relèvent la totalité des erreurs structurelles.
- Cette méthodologie est très pratiquée dans le contexte de code critique (additionner avec des méthodes de preuves de programmes et de langage fortement type).
- Dans les autres contextes se sont des tests complexes à mettre en place et n'apportant pas de plus-value « fonctionnel ». Autrement dit, on test ce qui fonctionne et moins ce qui ne devrait pas fonctionner.
- La question est : « à la lecture du code, où dois-je placer des tests structurels ».
- L'utilisation de sonar est essentiel pour mettre en place des tests de structures

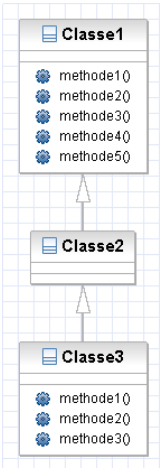
Tests structurels

- Différents indicateurs de méthodes à tester structurellement :
  - Les triviaux: Nombre total de lignes de codes, Nombre de méthodes par objet, Nombre de méthode appelé dans une méthode…
  - Les indices de spécialisation, d'instabilité
  - La complexité cyclomatique
- Ces métriques sont des « trucs » pour placer au mieux les tests structurels.
- Un bon emplacement est aussi de tester de façon importante les parties désignées comme critique par les développeurs, les méthodes ayant dépassé le temps assigné pour les écrire, les méthodes ayant été « repris » en cours d'écriture par un développeur

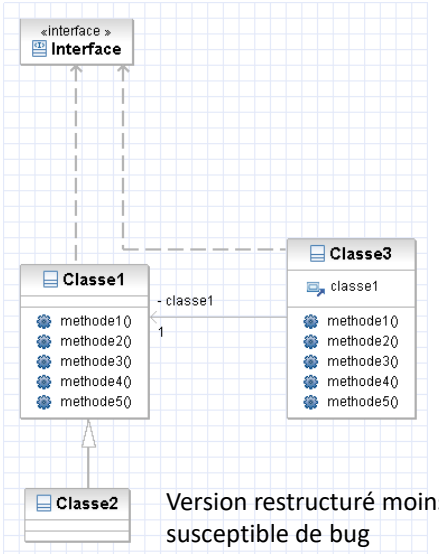
Les indices de spécialisation en OO

- L'indice de spécialisation se définit comme (NORM*DIT)/NOM.
  - NORM : Number of Overriden Methods, le nombre de méthodes redéfinies.
  - DIT : Depth in Inheritance Tree, la profondeur dans l'arbre d'héritage.
  - NOM : Number Of Methods, le nombre de méthodes de la classe étudié.
- L'indice est calculé pour toutes les classes d'un package.
- Les classes ayant un indices >1.5 sont de bonne candidates pour des tests structurels (trop de redéfinition de méthode pour une classe « trop » loin de la classe de basse)

Les indices de spécialisation en OO



L'indice de spécialisation est de 3 pour la classe 3 (3 méthode réécrite pour 3 méthode à 3 de distance)



Version restructuré moins susceptible de bug

Indice de stabilité en OO

- La stabilité d'une classe définit son côté « central » dans une application.
- Cet indice vaut Ce/(Ce+Ca) avec:
  - Ca : Couplage affèrent, le nombre de classes en dehors du paquetage qui dépendent de classes de ce paquetage.
  - Ce : Couplage efférent, le nombre de classes de ce paquetage qui dépendent de classes en dehors de ce paquetage.
- Cet indice va faire ressortir les paquetages qui dépendent plus des autres que les autres ne dépendent d'eux. Ces paquetages peuvent être dangereux, puisqu'une modification dans un des paquetages dont ils dépendent impacte potentiellement leur fonctionnement.
- Les tests structurels doivent donc d'abord s'attaquer au classe stable

Complexité cyclomatique

- La complexité cyclomatique d'une méthode est définie par le nombre de chemins linéairement indépendants qu'il est possible d'emprunter dans cette méthode.
- Plus simplement, il s'agit du nombre de points de décision de la méthode (if, case, while, ...) + 1 (le chemin principal).
- La complexité cyclomatique d'une méthode vaut au minimum 1, puisqu'il y a toujours au moins un chemin.

Complexité cyclomatique

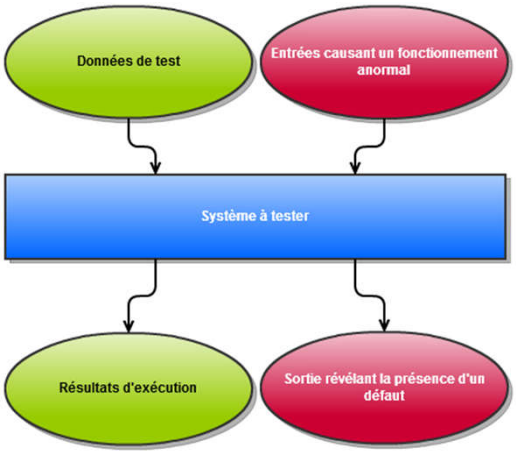
- La complexité cyclomatique d'une méthode augmente proportionnellement au nombre de points de décision. Une méthode avec une haute complexité cyclomatique est plus difficile à comprendre et à maintenir.
- Une complexité cyclomatique trop élevée (supérieure à 30) indique qu'il faut restructurer la méthode (code non testable).
- Une complexité cyclomatique inférieure à 30 peut être acceptable si la méthode est suffisamment testée.

Test fonctionnel

- Un test fonctionnel est un test de conformité par rapport à la spécification.
- Voici comment créer un test fonctionnel :
- Depuis les *spécifications* nous pouvons prédire les *résultats de test* : notion d'**Oracle**.
- Toujours depuis les *spécifications* nous pouvons définir les *données du test*.
- En passant ces *données de test* dans le *programme* à tester, nous pouvons comparer les *résultats d'exécution* avec les *résultats de test*.

Test fonctionnel

- Les tests fonctionnels nécessitent d'un part des spécifications à peu près clair et d'autre par la construction d'un Oracle (élément de comparaison)



Test fonctionnel

- L'idée d'un test fonctionnel est de vérifier qu'un logiciel fait ce qu'il doit faire.
- La notion de couverture fonctionnel existe et se pose sur le nombre de cas « clients » couvert par l'ensemble des tests.
- Ces tests sont « rapide » à mettre en place mais sont des tests « positifs »
- L'idéal est de mener ces tests en boite « blanche » afin de vérifier la couverture des tests de structures

Couverture de test

- La couverture de test est bien souvent appelée dans le contexte de test structurel par le terme de couverture de code.
- La couverture de test peut aussi être une couverture dite « fonctionnelle ».
- L'idée générale est de qualifier la « qualité » d'un test en vérifiant le pourcentage d'un logiciel testé par un test.
- L'autre idée est d'arriver à trouver la bonne granularité des tests:
  - Plus un test couvre plus il est « suffisant » pour valider un test
  - Plus un test couvre plus il est difficile de résoudre un bug en cas de problème.
  - Si on fait le choix de multiple test peut couvrant, il est facile de résoudre les bugs mais le cout des tests augmentent.

Couverture de code

- **Function coverage**: le nombre de fonction d'un logiciel couverte par un/des tests
- **Statement coverage** : le % d'instruction exécuté pendant les tests
- Branch coverage: le % de branchement testé
- Decision coverage: le % d'entrée/sortie d'un programme testé
- Condition coverage: le % d'expression booléenne évalué complètement
- Parameter coverage: le % d'espace de paramètre testé pour une fonction
- State coverage : le nombre d'état de l'application couverte par un programme (très proche de la couverture fonctionnelle)

Couverture de code

- **Path coverage**: le % de chemin d'exécution couvert par les tests
- Loop coverage : le % de boucle testé en en mode exécuté 0,1,n fois

- Ces notions se recouvrent les unes les autres (par exemple le path coverage recouvre le loop coverage).
- L'idée générale est de qu'un code bien testé couvre 80% des instructions

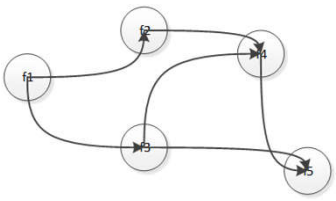
Couverture fonctionnel

- La couverture de code « classique » vérifie en boite blanche définissant le taux de « code » couvert par les tests.
- Cette couverture est indépendante des fonctionnalités d'une produits (une instruction d'une fonctionnalité inutile est testé de la même façon qu'une instruction utile).
- Nécessite un graphe de dépendance fonctionnel de l'application (quels sont la totalité des cas d'usages)

Couverture fonctionnel

- Le graph fonctionnel est utilisé et pondérer en fonction de la potentialité de transfert entre chaque fonction.
  - Après avoir rentré son login/password l'utilisateur clique sur « Login »: pondération forte
  - Après avoir rentré son login/password l'utilisateur quitte la page : pondération faible
- On calcule sur le graphique une liste d'arbres couvrant de poids maximal afin de calculer les meilleurs couvertures

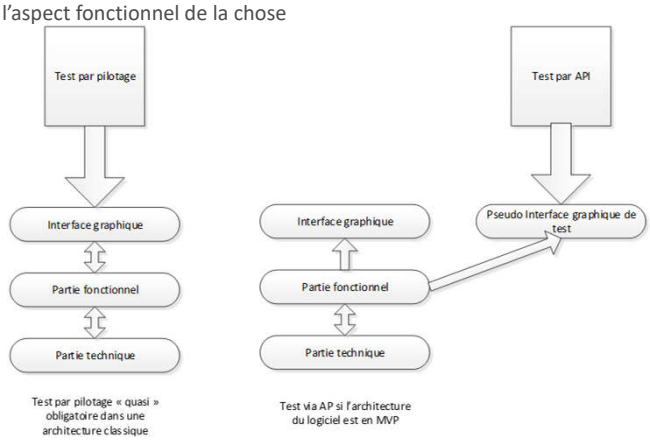
Couverture fonctionnel



- L'idée est de tester f1,f2,f3,f4,f5 de façon atomique.
- Cela n'est possible que pour f1.
- Pour tester f3, il faut d'abord pouvoir tester f1.
- Une idée est d'exécuter f1 le plus simplement possible pour tester f3.

Couverture fonctionnel

- La vraie difficulté est de mettre en place ces tests sachant que souvent l'interface graphique « gène ».
  - Soit on travaille par pilotage de l'interface graphique. Le résultat du test est souvent une comparaison d'image.
  - Soit on travaille avec des API, ce qui nécessite en pratique une programmation par MVP. Cette approche évite les bugs due à l'interface graphique et se concentre sur l'aspect fonctionnel de la chose



Couverture fonctionnel

- La couverture des cas d'usage en boite noire permet de valider « rapidement » le logiciel par des utilisateurs « standard ». Ces tests permettent après une bonne couverture de « valider » le logiciel.
- La couverture des cas d'usage en boite blanche permet de valider en sus la qualité des tests de structures par couverture de code.
- C'est le différentiel entre la couverture de code et la couverture fonctionnel qui est inquiétant

Les approches du test aujourd'hui

Approches par les risques

- Qu'est-ce que l'approche par les risques?
- Comment mettre en place cette approche?

Notion de risque

Le risque peut être analysé comme une fonction à deux composantes : la probabilité d'occurrence d'un événement indésirable et l'impact de la défaillance en termes de coûts induits. La probabilité d'occurrence d'un événement sur une application logiciel dépend en général de la fréquence d'utilisation de la fonction concernée par l'événement, plus cette fonction est utilisée plus il est probable que l'événement redouté se produise.

Approches par les risques

- L'approche par les risques est l'ordonnancement des tests en prenant en compte les risques d'une fonctionnalité.
- Le calcul du risque d'une fonctionnalité est une fonction de la complexité de cette fonctionnalité, du type d'utilisateur de la fonctionnalité ainsi que sa criticité

Criticité et niveau de confiance

Catégorie de risque dans l'aviation:

- Catastrophique (catégorie A) : condition de panne susceptible d'empêcher la poursuite en toute sécurité d'un vol et d'un atterrissage ;
- Dangereuse (catégorie B) : condition de panne susceptible de réduire les possibilités de l'aéronef ou la capacité de l'équipage à faire face à des conditions hostiles pouvant aller jusqu'à :
  - Une réduction importante des marges de sécurité ou des capacités fonctionnelles,
  - Des problèmes physiques ou un accroissement de charge de travail tels que l'équipage ne soit plus en mesure d'accomplir sa tâche de manière précise ou complète
  - Des effets négatifs sur les occupants telles que des blessures graves ou potentiellement mortelles pour un petit nombre d'entre eux ;
- Majeure (catégorie C) : condition de panne susceptible de réduire les possibilités de l'aéronef ou la capacité de l'équipage à faire face à des conditions hostiles d'une gravité se traduisant, par exemple , par une réduction significative des marges de sécurité ou des capacités fonctionnelles, un accroissement significatif de la charge de travail de l'équipage ou des conditions affectant son efficacité, ou un inconfort pour les occupants, comportant éventuellement des blessures ;
- Mineure (catégorie D) : condition de panne n'engendrant pas de réduction significative de la sécurité de l'aéronef, et susceptible d'entraîner pour l'équipage des actions se situant tout à fait dans le domaine de leurs capacités ;
- Sans effet (catégorie E) : condition de panne n'affectant pas la capacité opérationnelle de l'aéronef ou n'entraînant pas une augmentation de la charge de travail de l'équipage

Approche orienté risque

- On calcul pour chaque fonctionnalité le risque à ne pas faire.
- Puis on range les fonctionnalités par ordre de risque, puis on crée les campagnes associées.
- Chaque test est en fonction de sa couverture traduit en priorité must, could, should, would, ignore

Les approches du test aujourd'hui

Les apports des approches Agiles

- En quoi les démarches Agiles ont changé la pratique du test
- Modèle de développement Agile en mode éditeur de logiciel

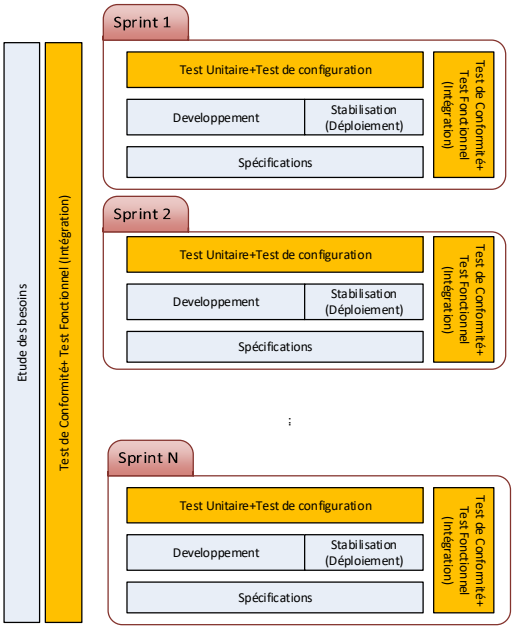
Apport de l'agilité

- Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.
- Accueillez positivement les changements de besoins, **même tard dans le projet**. Les processus agiles exploitent le changement pour donner un avantage compétitif au client.
- Livrez fréquemment **un logiciel opérationnel** avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.
- Les utilisateurs ou leurs **représentants et les développeurs doivent travailler ensemble** quotidiennement tout au long du projet.
- Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont elles ont besoin et faites-leur confiance pour atteindre les objectifs fixés.
- La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face.
- **Un logiciel opérationnel est la principale mesure d'avancement**.
- Les processus agiles encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.
- Une attention continue à l'excellence technique et à une bonne conception renforce l'agilité.
- **La simplicité – c'est-à-dire l'art de minimiser la quantité de travail inutile – est essentielle**.
- Les meilleures architectures, spécifications et conceptions émergent d'équipes auto-organisées.
- À intervalles réguliers, **l'équipe réfléchit aux moyens de devenir plus efficace**, puis règle et modifie son comportement en conséquence.

Apport de l'agilité

- L'agilité a permis de mettre ensemble le métier, les développeurs et la qualité logicielle.
- La qualité est mise en avant puisque le logiciel doit être perpétuellement opérationnel.
- Néanmoins c'est un challenge pour les équipes Qualités qui doivent souvent à ressources constante

Modèle Agile Editeur



Apport de l'agilité

En pratique l'agilité dans le domaine de la qualité passe par :

- La mise en place de banc de test automatique
- La transformation en fin de sprint des tests « standard » en test de non régression.
- L'interface graphique est utilisée pour les tests fonctionnels (en mode recette pompier).

Apport de l'agilité

- Les librairies nUnit sont mise en place pour les tests techniques/fonctionnels
- L'utilisation de Sélénium (pour les sites Web) ou de robot est mise en place pour les tests fonctionnels
  - Ceci fait que le cout de maintenance des tests fonctionnels est très important car ils doivent être quasi réécrits pour chaque itération.
- Mise en place de source control pour la non-régression
- Mise en place des gestionnaires d'anomalies
- La TRA est utilisable pour des raisons de couts, mais doit être piloter finement (la relation client – TRA n'est pas Agile)

Les approches du test aujourd'hui

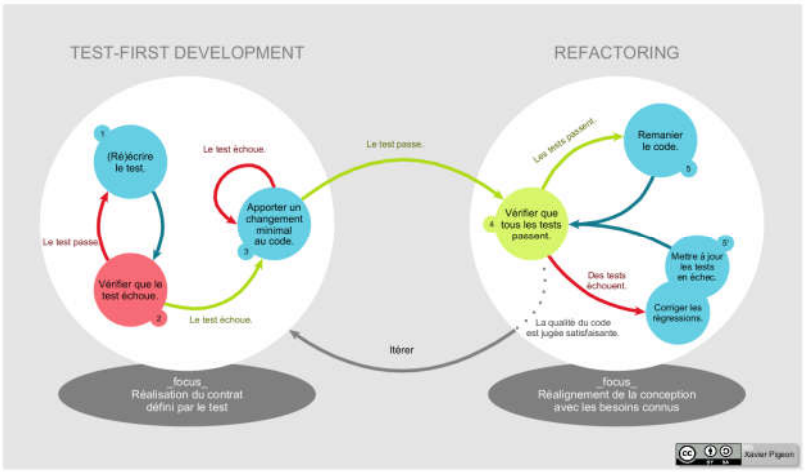
Le Test-Driven Development

- Qu'est-ce que le TDD?
- Comment mettre en place le TDD?
- Critiques

TDD

- Le test-driven development (TDD) ou en français développement piloté par les tests est une technique de développement de logiciel qui préconise d'écrire **les tests unitaires** avant d'écrire le code source d'un logiciel.
- Le cycle préconisé par TDD comporte cinq étapes :
  - Ecrire un premier test
  - Vérifier qu'il échoue (car le code qu'il teste n'existe pas), afin de vérifier que le test est valide
  - Ecrire juste le code suffisant pour passer le test
  - Vérifier que le test passe
  - Puis ré-usiner le code, c'est-à-dire l'améliorer tout en gardant les mêmes fonctionnalités.

TDD


TEST-FIRST DEVELOPMENT
REFACTORING
(Ré)écrire le test.
Le test échoue.
Le test passe.
Apporter un changement minimal au code.
Le test passe.
Vérifier que le test échoue.
Le test échoue.
Itérer
Les tests passent.
Remanier le code.
Vérifier que tous les tests passent.
Mettre à jour les tests en échec.
Des tests échouent.
Corriger les régressions.
La qualité du code est jugée satisfaisante.
focus
Réalisation du contrat défini par le test
focus
Réalignement de la conception avec les besoins connus
Xavier Pigeon

TDD

- Ces tests permettent aussi de préciser les spécifications du code, et donc son comportement ultérieur en fonction des situations auxquelles il sera exposé. Ce qui facilite la production d'un code valide en toutes circonstances. On obtient donc un code plus juste et plus fiable.

- En écrivant les tests d'abord, on utilise le programme avant même qu'il existe. On s'assure ainsi que le code produit est testable unitairement. Il est donc impératif d'avoir une vision précise de la manière dont on va utiliser le programme avant même d'envisager son implémentation. Cela évite souvent des erreurs de conception dues à une précipitation dans l'implémentation avant d'avoir défini les objectifs.

L'utilisation de TDD permet la construction conjointe du programme et d'une suite de tests de non-régression.

TDD

- Néanmoins les approches TDD sont complexes à mettre en œuvre:
  - Comment faire avec les interfaces graphiques
  - Qui écrit les tests? Les développeurs ou les testeurs?
  - Qui maintient les tests?

- Ces tests ne couvrent que rarement la partie fonctionnelle des choses.

Les approches du test aujourd'hui

La maturité des processus (TMMI, Test Process Improvement, ISO/SPICE).

- Il s'agit d'une rapide présentation de processus afin de tenter rentabiliser une activité de test

Maturité des processus

Le besoin est d'avoir une vision précise des progrès et bénéfices réalisés par rapport à une situation initiale et à une situation ciblée précisément identifiées et mesurables. C'est dans ce contexte qu'un modèle comme TMMi (Test Maturity Model integration) prend forme.

- Il est indépendant de toute société et affiche les niveaux de maturité des organisations certifiées ainsi que la liste des professionnels habilités à conduire des audits.
- Il est structuré, d'une façon similaire au modèle CMMI
- Il s'applique à tous les types de cycles de développement car, centré sur l'efficacité, il ne va pas rechercher des pratiques ou modèles de documents prédéfinis mais des pratiques efficaces dans un contexte donné.
- Il est reproductible, au sein d'une même entreprise mais aussi d'une entreprise à une autre, ce qui permet de réaliser des comparaisons de maturité fiables entre différentes organisations de test.
- Il couvre totalement le domaine du test mais aussi les domaines connexes comme la gestion des métiers du test et les relations entre MOA et MOE.

Maturité des processus

Définition de 5 niveaux de maturité:

- Au niveau 1, le test n'est qu'un processus chaotique non défini où les pratiques en test sont souvent incomplètes et hétérogènes entre les projets.
- Au niveau 2, le test est un processus géré, indépendant du développement et du débogage. De bonnes pratiques peuvent être en place mais il n'y a pas de réelle « gestion de projet de test » ni d'activités centralisées.
- Au niveau 3, le test est parfaitement intégré au cycle de développement logiciel, il commence tôt dans les projets et les pratiques de test sont homogènes entre les projets, sur la base d'un processus défini et mis en œuvre.
- Au niveau 4, le test est un processus déployé et mesurable c'est-à-dire que son efficacité est mesurée, de même que la qualité des produits ou systèmes testés.
- Au niveau 5, le test est un processus en optimisation continue et des pratiques avancées pour le contrôle de la qualité et la prévention des défauts sont en place.

Maturité des processus

- La plupart des entreprises sont de facto entre le niveau 2 et le niveau 3.
- TMMi et consort définisse des framework de processus de gestion de projet de test:
  - Ils ont l'avantage de donner un cadre d'entreprise à l'activité de qualité logicielle.
  - Néanmoins, ces processus sont souvent lourds et prennent difficilement en compte la taille des projets, la nécessité de vitesse.

Enjeux économiques et techniques des métiers du test

- Historique et définitions
- Opportunités du test aujourd'hui
- Eléments clés d'une validation fonctionnelle
- Coûts et rentabilité du projet de test
- Les approches du test aujourd'hui
- Outils et infrastructure de test Agile
  - Outils de tests
  - Intégration continue
- Les solutions de gestion des tests
- Synthèse

Outils et infrastructure de test Agile

Quels sont les principaux outils pour faire des tests

- Qu'est ce qu'un banc de test automatique
- Typologie d'outils de tests
- Liste d'outils

Qu’est ce que l’intégration continue

- La mise en place de tests peut se faire de manière plus ou moins automatique.
- Il ne s’agit pas ici de parler de méthodologie de test mais de savoir comment optimiser l’utilisation des tests pour améliorer la qualité général d’un produit.
- L’intégration continue est une méthode d’optimisation issue de la pratique de l’agilité dans le domaine du test.

Qu’est ce que l’intégration continue

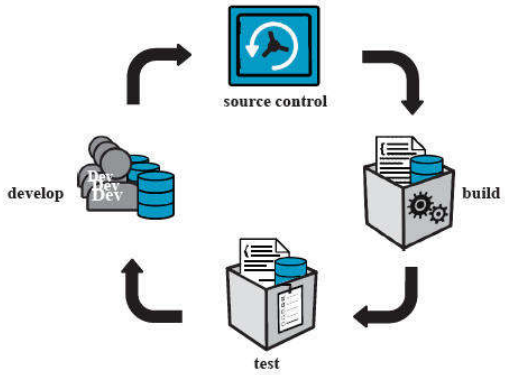
- L’idée général est que les développeurs sauvegardent tous leurs codes à un même endroit.
- Puis le code sauvegardé est compiler pour fabriquer un produit finit appelé la build. Si la construction est réussi, on envoie un statut sur le fait que la build est faites.
- Les tests sont exécutés contre la build en mode plus ou moins automatique.
- Le résultats des tests est envoyé aux membres du projet.

Qu'est ce que l'intégration continue

- Les hypothèses (fortes) que suggère cette méthode sont:
  - Le produit est séparable en petits morceaux dont le développement itératif et permet d'obtenir le produit. Le chargé de projet est une personnalité forte capable d'agencer les fonctionnalités dans un ordre précis.
  - Les développeurs sont capable de développer les différents morceaux et les intégrer rapidement.
  - Le produit est à tous moment compilable et « runnable »
  - Les équipes de tests sont capable d'écrire les tests en mode « code ».

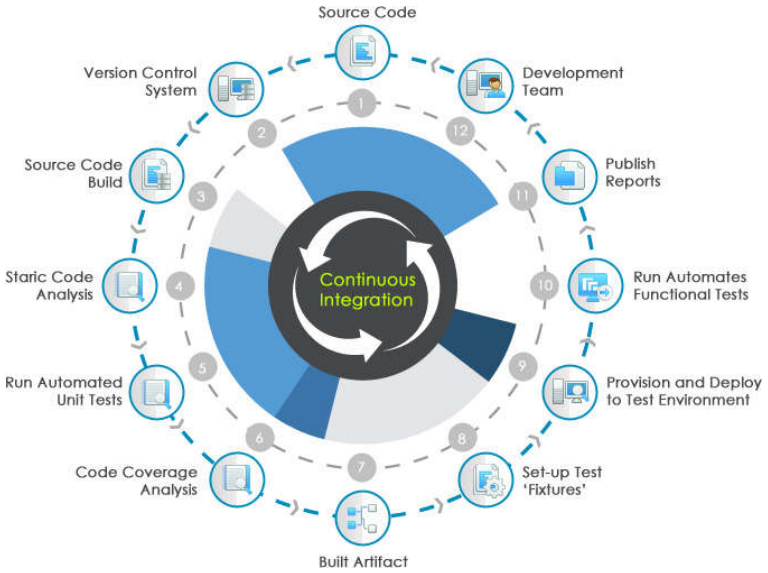
Qu’est ce que l’intégration continue

- Vulgairement, on pousse les sources, on les compile et on test en mode itératif



Qu’est ce que l’intégration continue

- En version précise avec l’intégration des stratégie de tests



Intégration continue: valeurs/couts

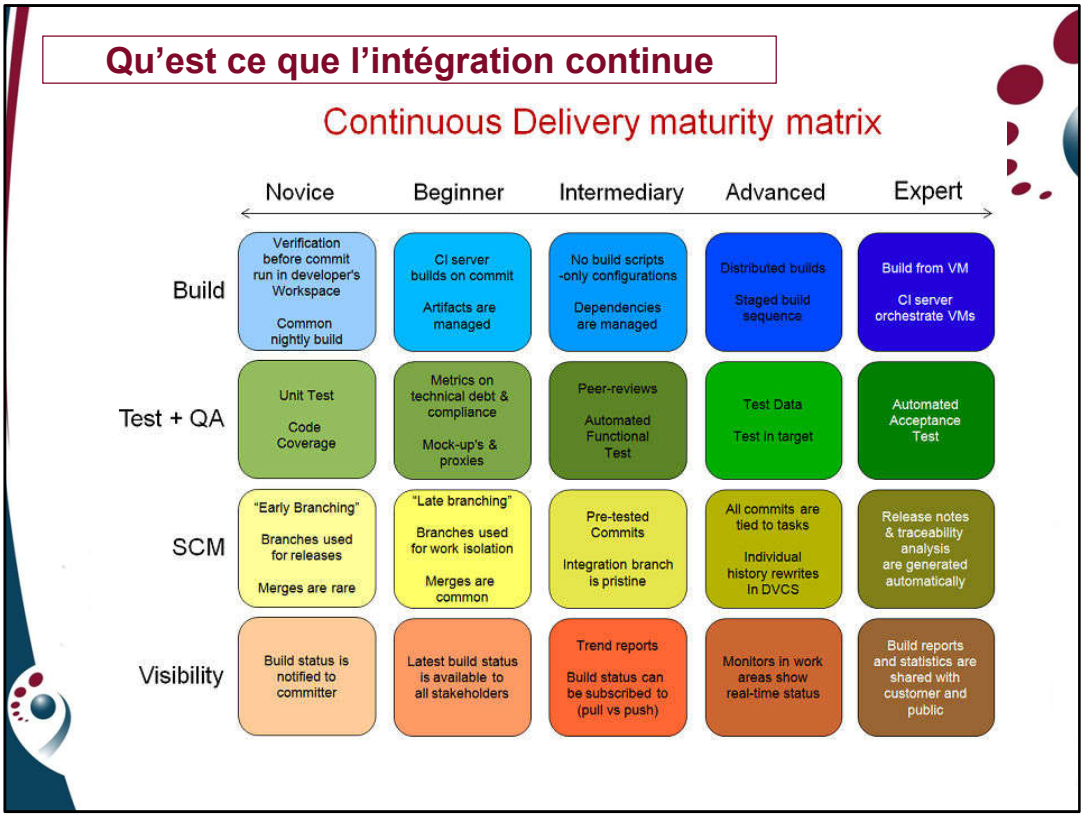
- Coûts de mise en place:
  - Outillage
  - Formation
  - Accompagnement
- Coûts récurrents
  - Excecution des tests
  - Nouveaux tests/maintenances
  - Analyse
- Réutilisation des scénarios
  - Suivie de production
  - Contrôle de plate-form de test
- Processus de test plus fluide:
  - Tests plus fréquents
  - Temps de test abaissée
  - Moins d'intervention
- Amélioration de la qualité

Qu'est ce que l'intégration continue

- L'intégration continue est donc la mise en place d'une usine capable :
  - De mettre les sources dans un repository commun
  - La construction du produit en mode +/- automatique
  - L'execution des tests en mode +/- automatique
  - L'envoie des rapports de tests.

- La mise en place de l'usine d'intégration continue est fonction de la maturité de l'équipe

Qu'est ce que l'intégration continue



Règles de l'Intégration continue

- L'intégration continue est un gain si certaines règles sont suivit (sinon c'est une catastrophe).
- 1 Commiter le code fréquemment:
  - L'idée est de faire des petits changements sur le code et de le commiter après chaque tache. Si cela n'est pas fait, il y auras trop de code commiter dans le serveur, et les build crée par l'Integration continue, ne marcherons pas.
- 2 Commiter du code testé.
  - Le but des tests Agiles est de tester souvent le code qui est fait avec les tests unitaires. Il est essentiel de faire les tests AVANT de commiter. Sinon, le produit seras rejetter par l'Integration continue pour des mauvaises raisons.

Règles de l'Intégration continue

- 3) Résoudre les builds ratés rapidement.
  - Lorsqu'une build est failed par l'intégration continue, il faut passer en priorité le fait de la réparer afin de permettre aux autres developpeurs de continue a comiter.
- 4) Ecrire les tests Automatisé:
  - L'idée de IC est de construire le produit automatique en le testant en mode automatique. Cela part du principe qu'il existe des tests automatique.
- 5) Ne pas commenter un test
  - Si la build est failed du fait d'un test non passant, la bonne solution n'est PAS de commenté le test

Règles de l'Intégration continue

- 6) Crée des build privé.
  - Le workflow normal est d'arriver le matin et de récupérer sur son poste la dernière vesion du code source du serveur de source. Faire le travail, ses tests et commiter les modifications. Il est contre productif de travailler sur des sources ancièn ne
- 7) Ne pas commiter/récupérer si la build est cassé.
  - Si la build est cassé, le code dans le serveur de source va être modifié. La priorité est de le réparer et non ni de pousser son code, ni de récupérer du code qui ne fonctionne pas.

Typologie d'outils de tests

- La mise en place de l'usine nécessite la mise en place d'outils permettant une automatisation plus ou moins poussée.
- Les principaux types d'outils recouvrent les parties suivante :
  - Outils de construction de produit
  - Outils de scripting afin de construire le produit
  - Logiciel de repository de source
  - Logiciel de gestion de données
  - Logiciel de gestion de test
  - Logiciel de test automatique
  - Logiciel de verification de source
  - Logiciel de déploiement automatique
  - Logiciel de documentation

Outils de construction de produit

- On attend de ce produit de pouvoir servir de chef d'orchestre à la construction de la build et l'execution des tests.
- Parmi les produits utilisé on trouve:
  - CruiseControl/CruiseControl.NET
  - Hudson (fork de cruise control)
  - Jenkins
  - BuildForge
- Ces outils doivent être équipé d'un outil de scripting afin de faire l'automatisation:
  - Batch file
  - Fichier de description maven
  - Fichier de build de type ant

Base de donnée utilisable en test

- La mise en place de l'intégration continue nécéssite de pouvoir manipuler la base de données en mode Agile (construction, destruction, création de schéma….).
- Petites bases de données: SQLite, MySQL, HSQLDB
- Base de données industrielles: Oracle Express, SQLServer Express, PostgreSQL.

Logiciel permettant de faire les tests

- Il est nécessaire de pouvoir coder les tests afin de les faire exécuter par l'usine de test:
  - xUNIT : Framework de création de test (DBUnit, JUnit,CPPUnit, NUnit, TestNG)
  - Test Web: HtmlUnit, JWebUnit, Selenium
  - Intégration des tests fonctionnel:Fit, FitNesse, Floyd
  - Base de Donnée: SQLUnit, utPLSQL

Logiciel permettant de vérifier le code

- Il est possible d'essayer de vérifier les codes sources de façon statique afin d'éviter les « bugs » communs:
  - CheckStyle: vérification du style du code source
  - Cobertura, EMMA: calcul la qualité de test par couverture de code.
  - FindBugs: vérification du code par rapport aux erreurs courantes.
- Metriques de la qualité des sources
  - JavaNCSS, Jdepend : calcul de la complexité des codes
  - PMD : analyzer de code orienté qualité

Logiciel permettant documenter

- La documentation du code est automatisable pour la maintenance.
  - Basiquement il est possible d'executer Javadoc ou NDoc pour documenter a minima le code.
  - Plus utile, il est possible de generer la documentation UML afin de vérifier l'evolution du code au regard des fonctionnalités. Doxygen est utilisable pour cela.

Enjeux économiques et techniques des métiers du test

- Historique et définitions
- Opportunités du test aujourd'hui
- Eléments clés d'une validation fonctionnelle
- Coûts et rentabilité du projet de test
- Les approches du test aujourd'hui
- Les solutions de gestion des tests
  - Les gestionnaires d'exigences et de traçabilité. Les gestionnaires de référentiels des tests.
  - Gestionnaires d'anomalies, principes et solutions du marché. Coût de prise en main des différents outils.
- Synthèse

Les solutions de gestion des tests

Les gestionnaires d'exigences et de traçabilité. Les gestionnaires de référentiels des tests.

- Qu’est-ce qu’on attend d’un gestionnaire d’exigences, et de tests

Les gestionnaires

- Le but d'un gestionnaires de tests est de pouvoir faire un plan de recette qui soit partagé entre les testeurs et les développeurs.
- La plupart du temps (équipes réduites) on utilise des fichiers Excel/Word pour faire les fichiers de tests ou les scénarios de tests.
- Il est possible d'utiliser des outils spécialisé comme TestLink, HP Quality Center

Les gestionnaires

- Ces logiciels plus ou moins souple permettent :
  - De créer des plan de test, des cas de test et des scénarios fonctionnelles
  - Permettent de mettre en regard des configurations matérielles ainsi que des versions du logiciel pour la pratique agile.

- L'usage de ces logiciels est plus que recommandé pour les TRA (moins pour les équipes internes)

Les solutions de gestion des tests

Gestionnaires d'anomalies, principes et solutions du marché. Coût de prise en main des différents outils.

- Comment gérer les anomalies
- Qu’est-ce qu’un bug trackers.

Gestionnaires d'anomalies

- Les gestionnaires d'anomalies permettent de faire remonter les bugs/remarques pendant le cycle de vie du logiciel (chantier de run+maintenance).
- Plusieurs solutions « à la main » sont possible (Hotline téléphonique pour la maintenance, fichier Word, mail..).
- La problématique est que ces méthodes à la main doivent répondre à des contraintes afin de pouvoir réduire le temps de résolution des problèmes.

Gestionnaire d'anomalies

A minima pour résoudre rapidement un problème, il est nécessaire:

- D'identifier le logiciel incriminé ainsi que sa version voir sous version.
- Eléments pouvoir démontrer la problématique.
- La configuration matérielle mettant en exergue le problème.
- La partie du logiciel incriminé.
- Un mode opératoire pas par pas pour reproduire le problème.

Gestionnaire d'anomalies

- Deux processus sont possible:
  - Les « utilisateurs » décrivent de façon informelle le problème. Ce dernier est saisie tel quel et envoyé aux équipes techniques. Ensuite les équipes techniques reviennent vers les « utilisateurs ». Cette démarche souple est privilégié en production mais à tendance à multiplier les rapports d'anomalies (beaucoup de rapports mal rédigé (1 par utilisateurs) pour 1 problème).
  - Les « utilisateurs » saisissent eux même les anomalies dans un logiciels de saisies d'anomalies. Le logiciel impose un formalisme et la recherche de duplication d'erreur permettant un traitement rapide des problèmes. Néanmoins, tous les « utilisateurs » ne sont pas égaux face aux logiciels.

Gestionnaire d'anomalies

- Les principaux gestionnaires d'anomalies permettent:
  - De saisir un problème avec la démarche de reproduction
  - Permettent de façon facultative de préciser la configuration matérielle et la zone du logiciel incriminé.
  - De pouvoir rajouter des fichiers de description de l'anomalie
  - De faire un reporting des problèmes avec leur statut (nouveau, en cours, …)
- Mais aussi, ces logiciels ont des capacités RH car :
  - Il est possible de préciser le temps prévue pour corriger un problème
  - D'assigner le problème aux différents équipes techniques
  - De suivre le temps réels de correction du problème
  - De faire du reporting analytics.

Gestionnaire d'anomalies

Parmi les gestionnaires d'anomalies utilisé, on as:

- BugZilla OpenSource Gratuit (Serveur HTTP+Perl)
- Mantis OpenSource Gratuit (Serveur HTTP+PHP)
- Jira OpenSource (Serveur Java)
- Redmine OpenSource Gratuit (Serveur HTTP+Ruby)

Et en fonction peuvent se connecter à un gestionnaire de test.

Enjeux économiques et techniques des métiers du test

- Historique et définitions
- Opportunités du test aujourd'hui
- Eléments clés d'une validation fonctionnelle
- Coûts et rentabilité du projet de test
- Les approches du test aujourd'hui
- Les solutions de gestion des tests
- Synthèse

Synthèse

Bilan et bonnes pratiques

- Facteur de succès/échecs
- Bonne pratiques

Synthèse

- L'activité de test est une activité complexe:
  - Cette dernière est bien évidemment nécessaire, mais la visibilité est faible
  - Soit le logiciel marche et la qualité est vue comme une perte de ressource
  - Soit le logiciel ne marche pas et la qualité est vue comme déficiente.
- La qualité logicielle est le garant de la bonne marche d'un projet.
- C'est un rôle dual de l'aspect technique du projet et de la relation avec les clients.

Synthèse

- Les clients ont souvent des attentes impossibles à tenir dans un temps trop court avec un budget limité.
  - La qualité a pour rôle de rappeler que 13% des projets en échec provienne d'une mauvaise spécification.
  - Un projet de bonne qualité permet d'avoir une maintenance de bonne qualité.
- Les développeurs sont « créatif » et s'arrangent pour que les tests réussissent.
  - Les tests sont là pour vérifier et valider le travail

Synthèse

Mise en œuvre de bonne pratiques pour combattre les mythes urbains

Mythes de l'usager

Mythe	***Réalité***
• Un énoncé général des objectifs est suffisant pour commencer. On verra les détails plus tard.	• Une définition insuffisante des besoins des utilisateurs est la cause majeure d'un logiciel de mauvaise qualité et en retard
• Les besoins du projet changent continuellement, mais ces changements peuvent être facilement incorporés parce que le logiciel est flexible	• Les coûts d'un changement pour corriger une erreur augmentent dramatiquement dans les dernières phases de la vie d'un logiciel

Synthèse

Mise en œuvre de bonne pratiques pour combattre les mythes urbains

Mythes du développeur

Mythe	***Réalité***
• Une fois que le programme est écrit, et marche, le travail du développeur est terminé	• 50%-70% de l'effort consacré à un programme se produit après sa livraison à l'usager
• Tant qu'un programme ne fonctionne pas, il n'y a aucun moyen d'en mesurer la qualité	• Les revues de logiciel peuvent être plus efficaces pour détecter les erreurs que les jeux d'essais pour certaines classes d'erreurs
• Pour le succès d'un projet, le bien livrable le plus important est un programme fonctionnel	• ....

Synthèse

Mise en œuvre de bonne pratiques pour combattre les mythes urbains

Mythes du gestionnaire

Mythe	***Réalité***
• L'entreprise possède des normes, le logiciel développé devrait être satisfaisant	• Une configuration de logiciel inclue de la documentation, des fichiers de régénération, des données d'entrée pour des tests, et les résultats des tests sur ces données
• Les ordinateurs et les outils logiciels que l'entreprise possède sont suffisants	• ....
• Si le projet prend du retard, on ajoutera des programmeurs	