Testeur Certifié

Syllabus Niveau Fondation

Version 2010 FR

International Software Testing Qualifications Board

Comité Français des Tests Logiciels

4) Historique des modifications

Version	Date	Remarques
CFTL 2010FR	1-Sept-2010	Testeur Certifié Niveau Fondation Mise à jour – voir Annexe E – Changements intégrés dans le Syllabus 2010
ISTQB 2010	30-Mars-2010	Certified Tester Foundation Level Syllabus Maintenance Release – voir Annexe E – Changements intégrés dans le Syllabus 2010
ISTQB 2007	01-Mai-2007	Certified Tester Foundation Level Syllabus Maintenance Release – voir Appendix E – Release Notes Syllabus 2007
CFTL 2005FR	01-Juillet-2005	Testeur Certifié Niveau Fondation
ISTQB 2005	01-Juillet-2005	Certified Tester Foundation Level Syllabus
ASQF V2.2	Juillet-2003	ASQF Syllabus Foundation Level Version 2.2 “Lehrplan Grundlagen des Software-testens“
ISEB V2.0	25-Février-1999	ISEB Software Testing Foundation Syllabus V2.0 25 February 1999

Traduction française: Bernard HOMES, Eric RIOU du COSQUER, Alain RIBAULT, Bertrand CORNANGUER, Cyrille RIVIERE, Olivier DENOO

Table des Matières

Remerciements

International Software Testing Qualifications Board Working Party Foundation Level (Edition 2010): Thomas Müller (responsable), Rahul Verma, Martin Klonk et Armin Beer. Le groupe de travail remercie l'équipe de revue (Rex Black, Mette Bruhn-Pederson, Debra Friedenberg, Klaus Olsen, Tuula Pääkkönen, Meile Posthuma, Hans Schaefer, Stephanie Ulrich, Pete Williams, Erik van Veendendaal) et tous les Comités Nationaux pour leurs suggestions pour la version actuelle du syllabus.

International Software Testing Qualifications Board Working Party Foundation Level (Edition 2007): Thomas Müller (responsable), Dorothy Graham, Debra Friedenberg, et Erik van Veendendaal ainsi que l'équipe de revue (Hans Schaefer, Stephanie Ulrich, Meile Posthuma, Anders Pettersson, et Wonil Kwon) et tous les Comités Nationaux pour leurs suggestions.

International Software Testing Qualifications Board Working Party Foundation Level (Edition 2005): Thomas Müller (responsable), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson , Erik van Veendendaal et l'équipe de revue ainsi que tous les Comités Nationaux pour leurs suggestions.

Introduction à ce syllabus

Objectif de ce document

Ce syllabus forme la base du niveau Fondation de la qualification Internationale en tests de logiciels. L'International Software Testing Qualifications Board (ISTQB) et le Comité Français des Tests Logiciels (ci-après appelé CFTL) fournissent ce syllabus aux comités nationaux d'examens pour accréditer les fournisseurs de formations et pour en dériver des questions d'examens dans leur langue nationale. Les fournisseurs de formations produiront leurs cours et détermineront les méthodes de formation appropriées pour l'accréditation, et le syllabus aidera les candidats à se préparer aux examens.
Des informations sur l'historique et le contexte du syllabus peuvent être trouvées en Annexe A.

Le Niveau Fondation pour les Testeurs de Logiciels Certifiés

La qualification de niveau fondation vise toutes les personnes impliquées dans les tests de logiciels. Ceci inclut les personnes ayant des rôles de testeurs, analystes de tests, ingénieurs de tests, consultants en tests, gestionnaires de tests, testeurs en phase d'acceptation utilisateur et développeurs de logiciels. Cette qualification de niveau Fondation est aussi appropriée pour toute personne souhaitant une compréhension de base des tests de logiciels, tels que les gestionnaires de projets, responsables qualité, responsables de développements logiciels, analystes métier et consultants en management. Les possesseurs du Certificat Fondation seront capables de continuer afin d'atteindre un niveau supérieur de certification en tests de logiciels.

Objectifs de connaissance /niveaux de connaissance

Les niveaux de connaissance sont fournis pour chaque section de ce syllabus et classés de la façon suivante

- K1: se souvenir, reconnaître, mémoriser;
- K2: comprendre, expliquer, donner des raisons, comparer, classifier, résumer ;
- K3: appliquer, utiliser;
- K4: analyser

De plus amples détails et des exemples d'objectifs de connaissance sont donnés en Annexe B.

Tous les termes listés sous la rubrique "termes" après les titres de chapitre seront retenus (K1), même s'ils ne sont pas explicitement mentionnés dans les objectifs de connaissance.

L'examen

L'examen pour l'obtention du Certificat Fondation sera basé sur ce syllabus. Les réponses aux questions d'examen peuvent requérir l'utilisation d'informations contenues dans plus d'une section de ce syllabus. Toutes les sections de ce syllabus peuvent donner lieu à des questions d'examen.

Le format de l'examen est un questionnaire à choix multiples.

Les examens peuvent faire partie d'une formation accréditée ou être passés indépendamment (p.ex. dans un centre d'examen ou lors d'un examen public). La participation à une formation accréditée n'est pas un pré-requis au passage de l'examen.

Accréditation

Les fournisseurs de formations dont le contenu du cours suit ce syllabus peuvent être accrédités par un comité national reconnu par l'ISTQB et le CFTL. Les directives d'accréditation doivent être obtenues auprès de l'organisme ou du comité effectuant l'accréditation. Un cours accrédité est reconnu comme se conformant à ce syllabus, et peut inclure un examen ISTQB – CFTL comme partie du cours.

Pour de plus amples informations à destination des fournisseurs de formation, voir l'Annexe D.

Niveau de détail

Le niveau de détail dans ce syllabus permet un enseignement et des examens compatibles internationalement. Pour atteindre cet objectif, le syllabus contient :

- Des objectifs généraux d'instruction, décrivant les intentions du niveau fondation
- Une liste des informations à enseigner, incluant une description, et des références à des sources additionnelles si besoin.
- Des objectifs de connaissance pour chaque domaine de connaissance, décrivant les résultats cognitifs d'enseignements et la mentalité à acquérir.
- Une liste de termes que les étudiants doivent se rappeler et comprendre.
- Une description des concepts clé à enseigner, incluant des sources comme des normes ou de la littérature reconnue.

Le contenu du syllabus n'est pas une description de l'ensemble du domaine de connaissance en tests de logiciels; il reflète le niveau de détail devant être couvert par les cours et formations du niveau fondation.

Organisation du syllabus

Ce syllabus comprend six chapitres majeurs. Le titre principal de chaque chapitre montre l'objectif d'apprentissage couvert par le chapitre, et spécifie la durée minimale pour traiter ce chapitre. Par exemple:

2. Tester pendant le Cycle de Vie (K2)	**115 minutes**

Ce titre indique que le Chapitre 2 a un objectif de connaissance K1 (supposé quand un niveau supérieur est spécifié) et K2 (mais pas K3), et qu'une durée de 115 minutes est suggérée pour couvrir le matériel de ce chapitre.
Chaque chapitre comporte plusieurs sections. Chaque section possède aussi un objectif de connaissance et la durée requise en minutes. Les sous-sections qui n'ont pas de durée précisée sont inclues dans la durée totale de la section.

1. Fondamentaux des Tests (K2) — 155 minutes

Objectifs de connaissance pour Fondamentaux des Tests

Les objectifs identifient ce que vous serez en mesure de faire en fin de chaque module.

1.1 Pourquoi les tests sont-ils nécessaires ? (K2)

LO-1.1.1 Décrire, avec des exemples, la manière par laquelle un défaut dans un logiciel peut causer des dommages à des personnes, à l'environnement ou à la société. (K2)
LO-1.1.2 Faire la différence entre la cause initiale du défaut et ses effets. (K2)
LO-1.1.3 Donner des raisons pour lesquelles les tests sont nécessaires en donnant des exemples. (K2)
LO-1.1.4 Décrire pourquoi les tests font partie de l'assurance qualité et donner des exemples sur comment les tests contribuent à une qualité accrue. (K2)
LO-1.1.5 Expliquer et comparer les termes faute, défaut, défaillance et les termes associés erreur et bug. (K2)

1.2 Que sont les tests ? (K2)

LO-1.2.1 Rappeler les objectifs habituels des tests. (K1)
LO-1.2.2 Fournir des exemples pour les objectifs des tests lors des différentes phases du cycle de vie logiciel (K2)
LO-1.2.3 Faire la différence entre tester et déboguer (K2)

1.3 Les 7 Principes généraux des tests (K2)

LO-1.3.1 Expliquer les 7 principes généraux des tests (K2)

1.4 Processus de test fondamental (K1)

LO-1.4.1 Rappeler les 5 activités de test fondamentales et les tâches associées, de la planification aux activités de clôture (K1)

1.5 La psychologie des tests (K2)

LO-1.5.1 Rappeler les facteurs psychologiques ayant une influence sur le succès des tests (K1)
LO-1.5.2 Comparer la mentalité d'un testeur avec celle d'un développeur (K2)

1.1 *Pourquoi les tests sont-ils nécessaires? (K2)* *20 minutes*

Termes
Bug, défaut, erreur, défaillance, défaut, méprise, qualité, risques, logiciel, test.

1.1.1 Contexte des systèmes logiciels (K1)

Les systèmes logiciels deviennent une part intégrante de notre existence, des applications commerciales (p.ex. bancaires) aux produits de grande consommation (p.ex. automobiles). La plupart d'entre nous ont eu l'expérience d'un logiciel qui n'a pas fonctionné comme attendu. Des logiciels ne fonctionnant pas correctement peuvent générer de nombreux problèmes, depuis des pertes financières, de temps ou de réputation, et pouvant même aller jusqu'à causer des blessures ou la mort.

1.1.2 Origine des défauts logiciels (K2)

Un être humain peut faire une erreur (méprise), qui produit un défaut (bug) dans le code, dans un logiciel ou un système, ou dans un document. Si un défaut dans du code est exécuté, le système n'effectuera pas ce qu'il aurait dû faire (ou fera ce qu'il n'aurait pas dû faire), générant une défaillance. Des défauts dans les logiciels, systèmes ou documents peuvent générer des défaillances, mais tous les défauts ne le font pas.

Les défauts apparaissent parce que les humains peuvent se tromper et à cause des échéanciers serrés, de la complexité du code et des infrastructures, des modifications de technologies et/ou de multiples interactions entre les systèmes.

Les défaillances peuvent être aussi causées par des conditions d'environnement : radiations, magnétisme, champs électroniques et pollution peuvent causer des défauts dans les microprogrammes ou influencer l'exécution des logiciels en modifiant les conditions matérielles.

1.1.3 Rôle des tests dans le développement, la maintenance et l'opération des logiciels (K2)

Des tests rigoureux des systèmes et de la documentation peuvent aider à réduire les risques d'occurrence de problèmes dans l'environnement opérationnel et contribuent à la qualité des systèmes logiciels, si les défauts découverts sont corrigés avant la livraison du système pour un usage opérationnel.

Les tests de logiciels peuvent aussi être nécessaires pour respecter des exigences légales ou contractuelles, ou atteindre des normes industrielles spécifiques.

1.1.4 Tests et qualité (K2)

Avec l'aide des tests, il est possible de mesurer la qualité des logiciels en termes de défauts trouvés, pour des caractéristiques et exigences tant fonctionnelles que non-fonctionnelles (p.ex. fiabilité, utilisabilité, rentabilité, maintenabilité et portabilité). Pour plus d'informations sur les tests non-fonctionnels voir Chapitre 2; pour plus d'informations sur les caractéristiques logicielles voir 'Software Engineering – Software Product Quality' (ISO 9126).

Les tests peuvent augmenter le niveau de confiance en la qualité d'un logiciel s'ils trouvent peu ou pas de défauts. Un test conçu correctement et qui est exécuté sans erreur réduit le niveau de risque général du système. Quand les tests trouvent des défauts, la qualité du système logiciel s'accroît quand ces défauts sont corrigés.

Des leçons devraient être apprises à partir des projets précédents. En comprenant les causes premières des défauts trouvés dans d'autres projets, les processus peuvent être améliorés, ce qui ensuite peut prévenir l'apparition de ces défauts et, en conséquence, améliorer la qualité des systèmes futurs. C'est un aspect de l'assurance qualité.

Les tests devraient être intégrés comme une activité de l'assurance qualité (p.ex. au côté des standards de développement, de la formation et de l'analyse des défauts).

1.1.5 Combien de test est suffisant? (K2)

Décider de combien de test est suffisant devrait prendre en compte le niveau de risque, incluant les risques techniques, les risques liés à la sureté et au projet, ainsi que les contraintes du projet telles que le temps et le budget. Les risques seront développés au chapitre 5.

Les tests doivent fournir suffisamment d'informations pour que les responsables puissent prendre des décisions informées concernant la mise en production du logiciel ou du système en cours de test, pour l'étape de développement suivante ou la livraison aux clients.

1.2 *Que sont les tests ? (K2)* — *30 minutes*

Termes
débogage, exigences, revues, cas de test, test, objectifs des tests, test,

Contexte
Une perception habituelle des tests est qu'ils consistent uniquement en l'exécution de tests, en fait exécuter le logiciel. C'est une partie des tests, mais pas l'ensemble des activités de test.

Des activités de test existent avant et après l'exécution des tests. Ces activités incluent la planification et le contrôle, la sélection des conditions de test, la conception et l'exécution des cas de tests, la vérification des résultats, l'évaluation des critères de sortie, l'information sur le processus de test et sur le système en cours de test ; elles incluent aussi la réalisation et la finalisation des activités de clôture définitive à la fin d'une phase de test. Les tests incluent aussi la revue de documents (incluant le code source) et les analyses statiques.

Les tests dynamiques et les tests statiques peuvent être utilisés comme des moyens pour atteindre des objectifs similaires, et fourniront des informations permettant l'amélioration du système à tester et des processus de développement et de test.

Les objectifs de tests peuvent varier :
- Trouver des défauts
- Acquérir de la confiance sur le niveau de qualité
- Fournir de l'information utile aux prises de décision
- Prévenir des défauts

Le processus de réflexion et les activités visant à concevoir des tests tôt dans le cycle de vie (vérifier les bases de tests via la conception des tests) peuvent aider à prévenir l'introduction de défauts dans le code. Les revues de documents (p.ex. exigences), l'identification et la résolution de problèmes peuvent aussi aider à prévenir l'apparition de défauts dans le code.

Les points de vue différents des tests prennent en compte des objectifs différents. Par exemple, dans les tests de développement (p.ex. tests des composants, d'intégration ou système), l'objectif principal peut être de générer le plus de défaillances possible de façon à identifier et à corriger les défauts dans le logiciel. Dans les tests d'acceptation, l'objectif principal peut être de confirmer que le système fonctionne comme attendu, pour s'assurer qu'il atteint les exigences. Dans certains cas l'objectif principal des tests peut être d'évaluer la qualité d'un logiciel (sans chercher à trouver des anomalies), de façon à fournir aux responsables de l'information sur les risques de distribuer un système à un moment précis. Les tests de maintenance incluent souvent des tests pour s'assurer que de nouvelles anomalies n'ont pas été introduites pendant le développement des évolutions. Pendant les tests d'opération, les objectifs principaux peuvent être d'évaluer les caractéristiques du système telles la disponibilité ou la fiabilité.

Tester et déboguer sont différents. Les tests dynamiques peuvent montrer des défaillances causées par des défauts. Déboguer est l'activité de développement qui trouve, analyse et supprime les causes de la défaillance. Les tests de confirmation ultérieurs effectués par un testeur assurent que la correction résout effectivement la défaillance. La responsabilité de chaque activité est différente : les testeurs testent, les développeurs déboguent.
Le processus de test et ses activités sont expliqués en Section 1.4.

1.3 *Les 7 principes généraux des tests (K2)* — *35 minutes*

Termes
Tests exhaustifs.

Principes
Un nombre de principes de test ont été suggérés au cours des 40 dernières années et offrent des indications communes à tous les tests.

Principe 1 – Les tests montrent la présence de défauts
Les tests peuvent prouver la présence de défauts, mais ne peuvent pas en prouver l'absence. Les tests réduisent la probabilité que des défauts restent cachés dans le logiciel mais, même si aucun défaut n'est découvert, ce n'est pas une preuve d'exactitude.

Principe 2 – Les tests exhaustifs sont impossibles
Tout tester (toutes les combinaisons d'entrées et de pré-conditions) n'est pas faisable sauf pour des cas triviaux. Plutôt que des tests exhaustifs, nous utilisons l'analyse des risques et des priorités pour focaliser les efforts de tests.

Principe 3 – Tester tôt
Pour trouver des défauts tôt, les activités de tests devraient commencer aussi tôt que possible dans le cycle de développement du logiciel ou du système, et devraient être focalisées vers des objectifs définis.

Principe 4 – Regroupement des défauts
L'effort de test devrait être fixé proportionnelement à la densité des défauts prévus et constatés dans les différents modules. Un petit nombre de modules contiennent généralement la majorité des défauts détectés lors des tests pré-livraison, ou affichent le plus de défaillances en opération.

Principe 5 – Paradoxe du pesticide
Si les mêmes tests sont répétés de nombreuses fois, il arrivera que le même ensemble de cas de tests ne trouvera plus de nouveaux défauts. Pour prévenir ce "paradoxe du pesticide", les cas de tests doivent être régulièrement revus et révisés, et de nouveaux tests, différents, doivent être écrits pour couvrir d'autres chemins dans le logiciel ou le système de façon à permettre la découverte de nouveaux défauts.

Principe 6 – Les tests dépendent du contexte
Les tests sont effectués différemment dans des contextes différents. Par exemple, les logiciels de sécurité critique seront testés différemment d'un site de commerce électronique.

Principe 7 – L'illusion de l'absence d'erreurs
Trouver et corriger des défauts n'aide pas si le système conçu est inutilisable et ne comble pas les besoins et les attentes des utilisateurs.

1.4 *Processus de test fondamental (K1)* *35 minutes*

Termes
Tests de confirmation, incident, test de régression, base de tests , conditions de tests, couverture de test, données de tests, exécution des tests, critères de sortie, registre de test, plan de test, stratégie de test,procédure de test, politique de test, suite de test, rapport de synthèse de tests, testware.

Contexte
La partie la plus visible des tests est l'exécution des tests. Mais pour être efficaces et rentables, les plans de tests devraient aussi inclure du temps pour planifier les tests, concevoir les cas de tests, préparer les exécutions et évaluer l'état.

Le processus de test fondamental comprend les activités principales suivantes:

- Planifier et contrôler
- Analyser et concevoir
- Implémenter et exécuter
- Evaluer les critères de sortie et informer
- Activités de clôture des tests

Bien que logiquement séquentielles, les activités du processus peuvent se chevaucher partiellement ou être concurrentes. Une adaptation de ces activités principales au contexte du système ou du projet est en général requise.

1.4.1 Planification et contrôle des tests (K1)

La planification des tests consiste à définir les objectifs du test et à spécifier les activités de test à mettre en oeuvre pour atteindre les objectifs et la mission.

Le contrôle des tests est une activité continue de comparaison de l'avancement actuel par rapport au plan, et d'information sur l'état, y compris les déviations par rapport au plan. Cela implique de prendre les actions nécessaires pour atteindre la mission et les objectifs du projet. Afin de contrôler les tests, les activités de test devraient être contrôlées tout au long du projet. La planification des tests prend en compte le feed-back des activités de contrôle et de suivi.

Les tâches de planification et contrôle sont définies au chapître 5 de ce syllabus.

1.4.2 Analyse et Conception des tests (K1)
L'analyse et la conception des tests représentent les activités où les objectifs de test généraux sont transformés en des conditions de test et des conceptions de test tangibles.

L'analyse et la conception des tests se composent des tâches majeures suivantes:
- Réviser les bases du test (telles que les exigences, le niveau d'intégrité logiciel[1] (niveau de risque), les rapports d'analyse de risque, l'architecture, la conception et les interfaces)
- Evaluer la testabilité des exigences et du système
- Identifier et prioriser les conditions de test sur la base de l'analyse des articles de test, la spécification, le comportement et la structure du logiciel
- Concevoir et prioriser les tests de haut niveau

[1] Le degré de conformité auquel le logiciel satisfait ou doit satisfaire par rapport à un ensemble de caractéristiques sélectionnées par les parties-prenantes ou basées sur les systèmes logiciels, et devant être définies pour refléter l'importance du logiciel pour les personnes concernées (p.ex: complexité logicielle, niveau de sureté, niveau de sécurité, performance souhaitée, robustesse ou coût).

- Identifier les données de test nécessaires pour les conditions de test et les cas de test
- Concevoir l'initialisation de l'environnement de test et identifier les infrastructures et outils requis
- Créer une traçabilité bi-directionnelle entre les bases de test et les cas de test

1.4.3 Implémentation et exécution des tests (K1)

L'implémentation et l'exécution des tests est l'activité où les procédures ou scripts de test sont spécifiés en arrangeant les cas de test selon un ordre précis et en ajoutant toute information utile pour l'exécution des tests; l'environnement est initialisé et les tests sont lancés.

L'implémentation et l'exécution des tests se composent des tâches majeures suivantes:
- Finaliser, développer et prioriser les cas de test (yc l'identification des données de test)
- Développer et prioriser les procédures de test, créer les données de test et, éventuellement, préparer les harnais de test et écrire les scripts de tests automatiques
- Créer des suites de tests à partir des procédures de test pour une exécution rentable des tests
- Vérifier que les environnements de tests ont été mis en place correctement
- Vérifier et mettre à jour la traçabilité bi-directionnelle entre les bases de test et les cas de test
- Exécuter les procédures de test soit manuellement soit en utilisant des outils d'exécution de tests, en suivant la séquence planifiée
- Consigner les résultats de l'exécution des tests et enregistrer les identités et versions des logiciels en test, outils de test et testware
- Comparer les résultats actuels et les résultats attendus.
- Signaler les divergences comme des incidents et les analyser de façon à établir leur cause (p.ex. défaut dans le code, dans les données de test, dans la documentation de test, ou méprise dans la manière d'exécuter le test)
- Répéter les activités de test en réponse aux actions prises pour chaque divergence. Par exemple, réexécution d'un test qui était préalablement défaillant de façon à valider une correction (test de confirmation), exécution d'un test corrigé et/ou exécution de tests de façon à s'assurer que des défauts n'ont pas été introduits dans des secteurs non modifiés du logiciel ou que le défaut corrigé n'a pas découvert d'autres défauts (test de régression)

1.4.4 Evaluer les critères de sortie et informer (K1)

Evaluer les critères de sortie est l'activité où l'exécution des tests est évaluée en fonction des objectifs définis. Ceci devrait être fait pour chacun des niveaux de test.

Evaluer les critères de sortie contient les tâches majeures suivantes:
- Vérifier les registres de tests en fonction des critères de sortie spécifiés dans la planification des tests
- Evaluer si des tests supplémentaires sont requis ou si les critères de sortie doivent être changés
- Ecrire un rapport de synthèse des tests pour les parties prenantes

1.4.5 Activités de clôture des tests (K1)

Les activités de clôture des tests rassemblent les données des activités de tests terminées de façon à consolider l'expérience, les testwares, les faits et les valeurs. Ces activités sont menées aux jalons d'un projet, par exemple, quand un système logiciel est mis en production, un projet de test est terminé (ou annulé), un jalon est atteint, ou une version de maintenance est terminée.

Les activités de clôture des tests incluent les tâches majeures suivantes:
- Vérifier quels livrables prévus ont été livrés
- Clôturer les rapports d'incidents ou créer des demandes d'évolution pour ceux restant ouverts
- Documenter l'acceptation du système
- Finaliser et archiver les testwares, environnements de test et infrastructures de test pour une réutilisation future
- Fournir les testwares à l'organisation en charge de la maintenance

- Analyser les leçons apprises pour identifier les changements nécessaires pour les versions et projets futurs
- Utiliser l'information collectée pour améliorer la maturité des tests

1.5 *La psychologie des tests (K2)* *25 minutes*

Termes
Estimation d'erreur, test indépendant.

Contexte

La tournure d'esprit à utiliser pendant les tests et les revues est différente de celle utilisée lors de l'analyse ou du développement. Avec la mentalité appropriée, des développeurs sont capables de tester leur propre code, mais affecter cette responsabilité à des testeurs permet typiquement de focaliser l'effort et de fournir des bénéfices additionnels tels qu'une perspective indépendante par des ressources de test entraînées et professionnelles. Les tests indépendants peuvent être effectués à n'importe quel niveau des tests

Un certain degré d'indépendance (évitant le parti-pris de l'auteur) est souvent plus efficace pour détecter des défauts et des défaillances. L'indépendance n'est pas cependant un remplacement de la familiarité, et les développeurs peuvent efficacement trouver beaucoup d'erreurs dans leur propre code. Plusieurs niveaux d'indépendance peuvent être définis, comme les niveaux suivants présentés du plus faible au plus élevé :

- Tests conçus par la (les) personne(s) qui a (ont) écrit le logiciel à tester (niveau faible d'indépendance).
- Tests conçus par une (des) autre(s) personne(s) (p.ex. de l'équipe de développement).
- Tests conçus par une (des) personne(s) d'un groupe différent au sein de la même organisation (p.ex. équipe de test indépendante) ou par des spécialistes de test (p.ex. spécialistes en tests de performance ou utilisabilité)
- Tests conçus par une (des) personne(s) d'une organisation ou société différente (p.ex. sous-traitance ou certification par un organisme externe)

Les personnes et les projets sont dirigés par des objectifs. Les personnes ont tendance à aligner leurs plans en fonction des objectifs mis en place par le management et les autres responsables, par exemple, pour trouver des défauts ou confirmer qu'un logiciel fonctionne. De ce fait, il est important de spécifier clairement les objectifs des tests.

L'identification de défaillances pendant les tests peut être perçue comme une critique contre le produit et contre son(ses) auteur(s). Les tests sont, de ce fait, souvent vus comme une activité destructrice, même si c'est très constructif dans la gestion des risques du produit. Rechercher des défaillances dans un système requiert de la curiosité, du pessimisme professionnel, un oeil critique, une attention au détail, une bonne communication avec ses pairs en développement, et de l'expérience sur laquelle baser l'estimation d'erreurs.

Si des erreurs, défauts ou défaillances sont communiqués de manière constructive, l'animosité entre les testeurs et les analystes, concepteurs et développeurs peut être évitée. Ceci s'applique autant aux revues qu'aux tests.

Les testeurs et le responsable des tests ont besoin de bonnes compétences relationnelles pour communiquer des informations factuelles sur les défauts, les progrès et les risques, de manière constructive. Pour l'auteur du logiciel ou du document, une information sur les défauts peut permettre d'améliorer leur savoir-faire. Les défauts trouvés et corrigés pendant les tests permettront de gagner du temps et de l'argent plus tard, et de réduire les risques.

Des problèmes de communication peuvent survenir, particulièrement si les testeurs sont vus uniquement comme messagers porteurs de mauvaises nouvelles concernant des défauts.

Cependant, il existe plusieurs manières d'améliorer la communication et les relations entre les testeurs et leurs interlocuteurs :

- Commencer par une collaboration plutôt que par des conflits – rappeler à chacun l'objectif commun de systèmes de meilleure qualité
- Communiquer les découvertes sur le produit de façon neutre et factuelle sans critiquer la personne responsable, par exemple, écrire des rapports d'incidents (ou des résultats de revues) objectifs et factuels
- Essayer de comprendre ce que ressent une autre personne et pourquoi elle réagit comme elle le fait
- Confirmer que l'autre personne a compris ce que l'on a dit et vice versa

1.6 *Code d'éthique (K2)* — *10 minutes*

L'implication dans le test logiciel permet aux individus d'avoir accès à des informations confidentielles et privilégiées. Un code d'éthique est nécessaire, notamment pour assurer que les informations ne soient pas utilisées dans des cas non appropriés. En référence au code d'éthique d'ACM et de l'IEEE pour les ingénieurs, l'ISTQB® définit le code d'éthique suivant :

PUBLIC – les testeurs de logiciels certifiés doivent agir en fonction de l'intérêt public
CLIENT ET EMPLOYEUR – les testeurs de logiciels certifiés doivent agir pour l'intérêt de leur client et de leur employeur tout en respectant l'intérêt public
PRODUIT – les testeurs de logiciels certifiés doivent assurer que les fournitures qu'ils produisent (concernant les produits et les systèmes qu'ils testent) répondent le plus possible aux standards professionnels
JUGEMENT – les testeurs de logiciels certifiés doivent conserver leur intégrité et leur indépendance dans leur jugement professionnel
GESTION – les chefs de projet de test de logiciels certifiés et les responsables doivent respecter et promouvoir une approche morale dans la gestion de projets de test de logiciels
PROFESSION – les testeurs de logiciels certifiés doivent mettre en avant l'intégrité et la réputation du métier en cohérence avec l'intérêt public
COLLEGUES – les testeurs de logiciels certifiés doivent être loyaux, aider leurs collègues, et promouvoir le partenariat avec les développeurs de logiciels
PERSONNELLEMENT – les testeurs de logiciels certifiés doivent participer en permanence à de la formation pour leur métier et doivent promouvoir une approche morale concernant sa pratique.

References

1.1.5 Black, 2001, Kaner, 2002
1.2 Beizer, 1990, Black, 2001, Myers, 1979
1.3 Beizer, 1990, Hetzel, 1988, Myers, 1979
1.4 Hetzel, 1988
1.4.5 Black, 2001, Craig, 2002
1.5 Black, 2001, Hetzel, 1988

2. Tester Pendant le Cycle de Vie Logiciel (K2) — 115 minutes

Objectifs de connaissance pour Tester Pendant le Cycle de Vie Logiciel

Les objectifs identifient ce que vous serez capable de faire après avoir suivi ce module.

2.1 Les modèles de développement logiciel (K2)

LO-2.1.1 Comprendre les relations entre le développement, les activités de test et les livrables dans le cycle de vie de développement, et donner des exemples basés sur des caractéristiques et contextes de produits et de projets (K2).
LO-2.1.2 Reconnaître que les modèles de développement logiciel doivent être adaptés au contexte du projet et aux caractéristiques du produit (K1).
LO-2.1.3 Rappeler que les bonnes pratiques de test sont applicables dans tous les modèles de cycle de vie (K1)

2.2 Niveaux de tests (K2)

LO-2.2.1 Comparer les différents niveaux de tests: objectifs principaux, types d'objets habituels à tester, types de tests habituels (p.ex. fonctionnels ou structurels) et livrables associés, personnes en charge des tests, types de défauts et de défaillances à identifier. (K2)

2.3 Types de tests: les cibles de tests (K2)

LO-2.3.1 Comparer les quatre types de tests (fonctionnels, non-fonctionnels, structurels et liés aux changements) avec des exemples. (K2)
LO-2.3.2 Reconnaître que les tests fonctionnels et structurels peuvent apparaître à n'importe quel niveau. (K1)
LO-2.3.3 Identifier et décrire des types de tests non-fonctionnels basés sur des exigences non-fonctionnelles. (K2)
LO-2.3.4 Identifier et décrire des types de tests basés sur l'analyse de la structure ou de l'architecture d'un système logiciel. (K2)
LO-2.3.5 Décrire l'utilité des tests de confirmation et des tests de régression. (K2)

2.4 Tests de maintenance (K2)

LO-2.4.1 Comparer les tests de maintenance (tester un système existant) et les tests d'une nouvelle application en ce qui concerne les types de tests, les déclencheurs des tests et la quantité de tests. (K2)
LO-2.4.2 Reconnaitre les indicateurs pour les tests de maintenance (modification, migration et abandon). (K1)
LO-2.4.3 Décrire le rôle des tests de régression et de l'analyse d'impact en maintenance. (K2)

2.1 *Modèles de Développement Logiciel (K2)* — *20 minutes*

Termes
Logiciel commercial sur étagère (en anglais : Commercial Off The Shelf ou COTS), modèle de développement incrémental niveaux de tests, validation, vérification, modèle en V.

Contexte
Les tests n'existent pas de façon isolée; les activités de test sont liées aux activités de développement logiciel. Les différents modèles de cycle de développement nécessitent des approches de tests différentes.

2.1.1 Modèle en V (K2)
Bien que des variantes du modèle en V existent, un modèle en V standard utilise quatre niveaux de tests, correspondant aux quatre niveaux de développement.

Les quatre niveaux utilisés dans le syllabus sont:

- Tests de composants (unitaires);
- Tests d'intégration;
- Tests système;
- Tests d'acceptation.

En pratique, un modèle en V peut avoir des niveaux de développement et de tests différents, en moins ou en plus, en fonction des projets et des produits logiciels. Par exemple, il peut y avoir des tests d'intégration des composants après les tests de composants, et des tests d'intégration de systèmes après des tests systèmes.

Les livrables logiciels (tels les scénarios d'utilisation ou les cas d'emploi, les spécifications d'exigences, les documents de conception et le code) produits pendant le développement sont souvent les bases des tests d'un ou plusieurs niveaux de tests. Des références pour des livrables génériques sont disponibles dans le modèle CMMI (Capability Maturity Model Integration) ou dans l'IEEE/IEC 12207 (Processus de cycle de vie logiciel 'Software life cycle processes'). La vérification et la validation (ainsi que la conception au plus tôt des tests) peuvent être effectuées pendant le développement des livrables logiciels.

2.1.2 Modèle de développement itératif (K2)
Le mode de développement itératif est une succession d'activités exécutées comme une série de petits développements: exigences, conception, construction et tests d'un système. Exemples : prototypage, développement rapide d'applications (RAD), Rational Unified Process (RUP) et les modèles de développement agiles. Le système logiciel résultant (l'incrément) d'une itération peut être testé à plusieurs niveaux de tests à chaque itération. Un incrément, ajouté à ceux développés préalablement, forme un système partiel en croissance, qui devrait également être testé. Les tests de régression sont de plus en plus importants sur toutes les itérations après la première. La vérification et la validation peuvent être effectuées sur chaque incrément.

2.1.3 Tester au sein d'un modèle de cycle de vie (K2)
Quel que soit le modèle de cycle de vie, plusieurs bonnes pratiques de tests s'appliquent:

- A chaque activité de développement, correspond une activité de test.
- Chaque niveau de test a des objectifs de tests spécifiques pour ce niveau.
- L'analyse et la conception des tests pour un niveau de test devraient commencer pendant l'activité correspondante de développement.

- Les testeurs doivent être impliqués dans la revue des documents aussi tôt que des brouillons sont disponibles dans le cycle de développement.

Les niveaux de tests peuvent être combinés ou réorganisés selon la nature du projet ou de l'architecture du système. Par exemple, pour l'intégration d'un logiciel sur étagère (COTS) dans un système, l'acheteur peut effectuer des tests d'intégration au niveau système (Ex. intégration dans l'infrastructure et les autres systèmes, ou déploiement du système) ainsi que des tests d'acceptation (fonctionnels et/ou non-fonctionnels, et tests d'acceptation utilisateurs et/ou opérationnels).

2.2 *Niveaux de tests (K2)* — *40 minutes*

Termes
Alpha tests, Beta tests, tests de composant (aussi connus comme tests unitaires, de modules ou de programmes), tests d'acceptation contractuelle, pilotes, tests sur le terrain, exigences fonctionnelles, intégration, tests d'intégration, exigences non-fonctionnelles, tests de robustesse, bouchons, tests système, environnement de test, niveau de tests, développement dirigé par les tests, tests d'acceptation utilisateurs.

Contexte
Pour chaque niveau de tests, les aspects suivants peuvent être identifiés: les objectifs génériques, le(s) livrable(s) référencé(s) pour dériver des cas de tests (c'est à dire la base de test), les objets de tests (càd. ce qui est testé), les défauts et les défaillances typiques à trouver, les exigences en harnais de tests et en support d'outils, ainsi que les approches et responsabilités spécifiques au niveau.

2.2.1 Tests de composants (K2)

Bases de tests:
- Exigences des composants
- Conception détaillée
- Code

Objets habituels de test:
- Composants
- Programmes
- Conversions de données / utilitaires ou programmes de migration

Modules de bases de données

Les tests de composants (également connus sous les noms de tests unitaires, de modules ou de programmes) cherchent des défauts dans, et vérifient le fonctionnement des, logiciels (p.ex. modules, programmes, objets, classes, etc.) qui sont testables séparément. Cela peut se faire de façon isolée par rapport au reste du système, en fonction du contexte du cycle de développement du logiciel et du système. Des bouchons, pilotes et simulateurs peuvent être utilisés.

Les tests de composants peuvent inclure des tests de fonctionnalités et des tests de caractéristiques non-fonctionnelles, telles que le comportement des ressources (p.ex. fuites mémoire) ou des tests de robustesse, ainsi que des tests structurels (p.ex. couverture des branches). Les cas de test sont dérivés des livrables tels que les spécifications des composants (spécifications détaillées), la conception du logiciel ou le modèle de données.

La plupart du temps, les tests de composants se font avec l'accès au code du logiciel testé et sur un environnement de développement comme un framework de tests unitaire ou avec les outils de débogage. En pratique ils impliquent généralement le programmeur ayant écrit le code. Habituellement, les défauts sont corrigés dès qu'ils sont trouvés, sans formellement enregistrer des incidents.

Une approche des tests de composants est de préparer et automatiser des cas de tests avant le codage. Cela s'appelle l'approche « Tester d'abord » (en anglais : « test first »)ou « Développement piloté par les tests. Cette approche, fortement itérative est basée sur des cycles de développement de cas de tests, puis construction et intégration de petits bouts de code, puis exécution des tests de composants et correction des défauts trouvés jusqu'à leur réussite.

2.2.2 Tests d'intégration (K2)

Bases de Tests:
- Conception du logiciel et du système
- Architecture
- Workflows
- Cas d'utilisation

Objets habituels de test:
- Implémentation de bases de données des sous-systèmes
- Infrastructure
- Interfaces

Configuration Système
- Données de configuration

Les tests d'intégration testent les interfaces entre les composants, les interactions entre différentes parties d'un système comme par exemple le système d'exploitation, le système de fichiers, le matériel ou les interfaces entre les systèmes.

Plusieurs niveaux de tests d'intégration peuvent être effectués sur des objets de taille variable. Par exemple:

- Tests d'intégration des composants testant les interactions entre les composants logiciels et effectués après les tests de composants;
- Tests d'intégration système testant l'intégration entre les différents systèmes ou entre logiciel et matériel et pouvant être effectués après les tests système. Dans ce cas, l'organisation en charge du développement peut ne contrôler qu'un côté de l'interface. Cela peut être considéré comme un risque métier. Les processus commerciaux mis en œuvre comme les workflows peuvent impliquer plusieurs systèmes. Les aspects inter-plateformes peuvent être significatifs.

Plus la portée de l'intégration est vaste, plus il devient difficile d'isoler les défaillances liées à un composant ou un système particulier. Cela peut conduire à un accroissement des risques et du temps nécessaire pour résoudre les problèmes.

Des stratégies d'intégration systématique peuvent être basées sur l'architecture des systèmes (telles que top-down ou bottom-up), les tâches fonctionnelles, les séquences d'exécution de transactions, ou d'autres aspects du système ou du composant. Afin d'isoler facilement les fautes, et détecter les défauts au plus tôt, l'intégration devrait normalement être incrémentale plutôt qu'être effectuée en une fois ("big bang").

Les tests de caractéristiques non-fonctionnelles particulières (p.ex. performances) peuvent être inclus dans les tests d'intégration.

A chaque étape de l'intégration, les testeurs se concentrent uniquement sur l'intégration elle même. Par exemple, s'ils sont en train d'intégrer le module A avec le module B, les testeurs s'appliquent à tester la communication entre les modules et non pas leurs fonctionnalités individuelles. Les approches fonctionnelles et structurelles peuvent toutes deux être utilisées.

Idéalement, les testeurs devraient comprendre l'architecture et influencer le planning d'intégration. Si les tests d'intégration sont prévus avant que les composants ou les systèmes ne soient fabriqués, les tests pourront être conçus dans le bon ordre pour être efficaces et efficients.

2.2.3 Tests système (K2)

Bases de Test:
- Spécifications d'exigences Système et logiciel

- Cas d'utilisation
- Spécifications fonctionnelles
- Rapports d'analyse des risques

Objets de tests habituels:
- Manuels système, utilisateur et opérationnels
- Configuration système

Données de configuration

Les tests systèmes traitent le comportement d'un système/produit complet. Le périmètre des tests doit être clairement défini dans le plan de test maitre ou le plan de test du niveau.

Pour les tests système, l'environnement de test devrait correspondre à la cible finale ou à un environnement de production de façon à minimiser les risques de défaillances dues à l'environnement.

Les tests système peuvent inclure des tests basés sur les risques et/ou sur les spécifications et les exigences, les processus commerciaux, les cas d'utilisation, ou autres descriptions de haut niveau du comportement du système, les modèles de comportement, les interactions avec le système d'exploitation et les ressources système.

Les tests système devraient examiner à la fois les exigences fonctionnelles et les non-fonctionnelles du système ainsi que la qualité des données. Les testeurs doivent également s'adapter aux exigences peu ou pas documentées. Les tests fonctionnels au niveau système commencent par l'utilisation des techniques de spécification de tests les plus appropriées (boîte noire). Par exemple, une table de décision peut être crée pour les combinaisons d'effets décrits dans les processus commerciaux. Les techniques basées sur les structures (boîte blanche) peuvent ensuite être utilisées pour évaluer la minutie des tests par rapport à un élément comme la structure de menu ou la navigation dans la page web. (voir Chapitre 4.)

Une équipe de tests indépendante exécute souvent des tests système.

2.2.4 Tests d'acceptation (K2)

Bases de test:
- Exigences utilisateur
- Exigences du système
- Cas d'utilisation
- Processus métier
- Rapports d'analyse des risques

Objets habituels de test:
- Processus métier sur l'intégralité du système
- Processus opérationnels de maintenance
- Procédures utilisateur
- Formulaires
- Rapports

Données de configuration

Les tests d'acceptation relèvent souvent de la responsabilité des clients ou utilisateurs d'un système; d'autres responsables (parties prenantes) peuvent aussi être impliqués.

Les objectifs des tests d'acceptation sont de prendre confiance dans le système, dans des parties du système ou dans des caractéristiques non-fonctionnelles du système. La recherche d'anomalies n'est pas l'objectif principal des tests d'acceptation. Les tests d'acceptation peuvent évaluer si le système est prêt à être déployé et utilisé, bien que ce ne soit pas nécessairement le dernier niveau

de tests. Par exemple, une intégration système à grande échelle peut arriver après les tests d'acceptation du système.

Les tests d'acceptation peuvent se faire à plusieurs niveaux de tests, par exemple:

- Des tests d'acceptation peuvent avoir lieu sur un composant sur étagère quand il est installé ou intégré.
- Les tests d'acceptation de l'utilisabilité d'un composant peuvent être effectués pendant les tests unitaires.
- Les tests d'acceptation d'une évolution fonctionnelle peuvent être exécutés avant les tests système.

Les formes habituelles des tests d'acceptation incluent :

Tests d'acceptation utilisateur
Ces tests vérifient l'aptitude et l'utilisabilité du système par des utilisateurs.

Tests (d'acceptation) opérationnelle
L'acceptation du système par les administrateurs système, dont:

- Tests des backups et restaurations;
- Reprise après sinistre;
- Gestion des utilisateurs;
- Tâches de maintenance;
- Chargements de données et tâches de migration
- Vérification périodique des vulnérabilités de sécurité.

Tests d'acceptation contractuelle et réglementaire
Les tests d'acceptation contractuelle sont exécutés sur base des critères d'acceptation contractuels pour la production de logiciels développés sur mesure. Les critères d'acceptation devraient être définis lors de la rédaction du contrat.
Les tests d'acceptation réglementaires sont exécutés par rapport aux règlements et législations qui doivent être respectés, telles que les obligations légales, gouvernementales ou de sécurité.

Tests alpha et beta (ou sur le terrain)
Les développeurs de logiciels pour le public, ou de COTS (logiciel commercial sur étagère), souhaitent souvent recueillir l'avis des clients potentiels ou existants sur leur marché, avant de mettre en vente.
Les Alpha tests sont exécutés sur le site de l'organisation effectuant le développement mais pas par les équipes de développement. Les Béta tests ou tests sur le terrain sont exécutés par des personnes sur leurs sites propres.
Les organisations peuvent aussi utiliser d'autres termes, tels que « tests d'acceptation usine » ou « tests d'acceptation sur site » pour des systèmes qui sont testés avant d'être transférés sur le site client.

2.3 *Types de tests (K2)* — *40 minutes*

Termes
tests boîte noire, couverture du code, tests fonctionnels, tests d'interopérabilité, tests de charge, tests de maintenabilité, tests de performances, tests de portabilité, tests de fiabilité, tests de sécurité, tests de stress, tests structurels, tests d'utilisabilité, tests boîte blanche.

Contexte
Un groupe d'activités de tests peut être axé sur la vérification du système logiciel (ou d'une partie du système) pour une raison ou une cible particulière.

Un type de tests est focalisé sur un objectif de tests particulier, qui peut être

- le test d'une fonction devant être effectuée par le logiciel;
- une caractéristique non-fonctionnelle, telle que la fiabilité ou l'utilisabilité,
- la structure ou l'architecture du logiciel ou du système;
- lié aux changements, p.ex. confirmer que des anomalies ont été corrigées (tests de confirmation) et pour rechercher la présence de modifications non voulues (tests de régression).

Un modèle du logiciel peut être développé et/ou utilisé dans des tests structurels et fonctionnels (p.ex un diagramme de contrôle de flux ou une structure de menu), dans des tests non fonctionnels (modèles de menaces de sécurité), dans les tests fonctionnels (p.ex un diagramme de flux de processus, un diagramme de transitions d'état ou des spécifications en langage naturel).

2.3.1 Tests des fonctions (tests fonctionnels) (K2)
Les fonctionnalités qu'un système, sous-système ou composant doit effectuer peuvent être décrites dans des livrables tels que des spécifications d'exigences, les cas d'utilisation, ou les spécifications fonctionnelles, ou encore non documentées. Les fonctions sont ce que « fait » le système.

Les tests fonctionnels sont basés sur ces fonctions et caractéristiques (décrites dans des documents ou comprises par les testeurs), et leur interopérabilité avec d'autres systèmes. Ils peuvent être exécutés à tous les niveaux de tests (p.ex. les tests d'un composant peuvent être basés sur les spécifications du composant).

Des techniques basées sur les spécifications peuvent être utilisées pour dériver des conditions de tests et des cas de tests à partir des fonctionnalités du logiciel ou du système (voir Chapitre 4.) Les tests fonctionnels concernent le comportement extérieur du logiciel (tests boîte noire).

Un type de test fonctionnel, le test de sécurité, examine les fonctions (p.ex. pare-feu) liées à la détection de menaces, comme des virus, provenant de tiers malveillants. Un autre type de test fonctionnel, le test d'interopérabilité, évalue la capacité du logiciel à interagir avec un ou plusieurs composants ou systèmes spécifiés.

2.3.2 Tests des caractéristiques non fonctionnelles des produits logiciels (tests non-fonctionnels) (K2)
Les tests non-fonctionnels incluent, mais pas uniquement, les tests de performances, tests de charge, tests de stress, tests d'utilisabilité, tests de maintenabilité, tests de fiabilité et les tests de portabilité. Ces tests évaluent "comment" le système fonctionne.

Les tests non-fonctionnels peuvent être effectués à tous les niveaux de tests. Le terme de tests non-fonctionnels décrit les tests requis pour mesurer les caractéristiques des systèmes et logiciels qui peuvent être quantifiées sur une échelle variable, comme les temps de réponse pour les tests de performances. Ces tests peuvent être référencés dans un modèle qualité tel que celui défini par

l'ISO9126 'Ingénierie Logicielle – Qualité des Produits Logiciels' ('Software Engineering – Software Product Quality'). Les tests non fonctionnels concernent l'aspect extérieur du logiciel et la plupart du temps utilisent les techniques de conception de tests boîte noire.

2.3.3 Tests de la structure / architecture logicielle (tests structurels) (K2)

Les tests structurels (boîte blanche) peuvent être effectués à tous les niveaux de tests. Les techniques structurelles sont utilisées de façon optimale après les techniques basées sur les spécifications, pour aider à mesurer l'ampleur des tests via l'évaluation de la couverture d'un type de structure.

La couverture indique à quel point une structure a été testée par une suite de tests. Elle est exprimée en pourcentage d'éléments couverts. Si la couverture n'est pas de 100%, alors de nouveaux tests peuvent être conçus pour tester les éléments manquants et ainsi augmenter la couverture. Les techniques de couverture sont traitées dans le chapitre 4.

A tous les niveaux de tests, mais spécialement dans les tests de composants et les tests d'intégration de composants, des outils peuvent être utilisés pour mesurer la couverture du code des éléments, telle que les instructions ou les décisions. Les tests structurels peuvent être basés sur l'architecture du système comme par exemple la hiérarchie d'appels.

Les approches de tests structurels peuvent être aussi appliquées au niveau système, à l'intégration système ou aux niveaux de tests d'acceptation (p.ex. pour les processus métier ou les structures de menus).

2.3.4 Tests liés au changement (tests de confirmation et de régression) (K2)

Quand un défaut est détecté et corrigé, le logiciel devrait être re-testé pour confirmer que le défaut original a été correctement ôté. Ceci est appelé test de confirmation. Le débogage (correction de défaut) est une activité de développement, pas une activité de tests.

Ré-exécuter des tests sur un programme déjà testé s'appelle « test de régression ». Cela se fait après que des modifications du programme aient eu lieu, pour identifier tout nouveau défaut dû à ce(s) changement(s). Ces défauts peuvent se trouver soit dans le logiciel en cours de tests, soit dans d'autres composants logiciels liés ou non. Les tests de régression sont exécutés quand le logiciel, ou son environnement, est modifié. Le périmètre des tests de régression est basé sur le risque de ne pas trouver d'anomalie dans un logiciel fonctionnant auparavant.

Les tests devraient être répétables s'ils sont utilisés pour des tests de confirmation ou pour les tests de régression.

Les tests de régression peuvent être exécutés à tous les niveaux de tests, et s'appliquent aux tests fonctionnels, non-fonctionnels et structurels. Les suites de tests de régression sont exécutées plusieurs fois et évoluent généralement lentement. Donc les tests de régression sont de bons candidats à l'automatisation.

2.4 *Tests de maintenance (K2)* — *15 minutes*

Termes
Analyse d'impact, tests de maintenance

Contexte
Une fois déployé, un système logiciel est souvent en service pendant des années, voire des dizaines d'années. Pendant ce temps, le système, son paramétrage et son environnement sont fréquemment corrigés, modifiés ou étendus. La planification au plus tôt des livraisons est cruciale pour le succès des tests de maintenance. Une distinction doit être faite entre les livraisons planifiées et les mises à jour à chaud (hot fixes). Les tests de maintenance sont effectués sur un système opérationnel existant et sont déclenchés par des modifications, migrations ou suppression de logiciels ou de systèmes.

Les modifications incluent les changements dûs aux évolutions planifiées (p.ex. livraisons de nouvelles versions), aux modifications correctives et d'urgence, ainsi qu'aux changements d'environnements tels que les montées en version planifiées des systèmes d'exploitation, des bases de données ou des COTS. Elles incluent également les patchs de correction des vulnérabilités de sécurité potentielles ou découvertes d'un système d'exploitation.

Les tests de maintenance lors de migrations (p.ex. d'une plate-forme à une autre) devraient inclure les tests opérationnels du nouvel environnement tout comme les tests des modifications du logiciel. Les tests de migration (tests de conversion) sont également nécessaires lorsque les données d'une autre application seront migrées dans le système à maintenir.

Les tests de maintenance pour la suppression (mise au rebut) d'un système incluent le test de la migration des données ou leur archivage si de longues périodes de stockage de données sont nécessaires.

En plus des tests des éléments changés, les tests de maintenance incluent des tests de régression approfondis sur les parties du système qui n'ont pas été modifiées. Le périmètre des tests de maintenance est fonction des risques liés aux modifications, à la taille du système existant et à la taille des modifications effectuées. Selon le changement, les tests de maintenance peuvent être effectués à chacun ou à tous les niveaux de tests et pour certains ou tous les types de tests.

Déterminer comment un système existant est affecté par les changements est appelé analyse d'impact, et est utilisé pour décider de la quantité de tests de régression devant être exécutés. L'analyse d'impacts peut être utilisée pour déterminer les suites de tests de régression.

Les tests de maintenance peuvent être difficiles si les spécifications sont périmées ou manquantes, ou si les testeurs ayant la connaissance fonctionnelle ne sont pas disponibles.

Références
2.1.3 CMMI, Craig, 2002, Hetzel, 1998, IEEE 12207
2.2 Hetzel, 1998
2.2.4 Copeland, 2004, Myers, 1979
2.3.1 Beizer, 1990, Black, 2001, Copeland, 2004
2.3.2 Black, 2001, ISO 9126
2.3.3 Beizer, 1990, Copeland, 2004, Hetzel, 1998
2.3.4 Hetzel, 1998, IEEE 829
2.4 Black, 2001, Craig, 2002, Hetzel, 1998, IEEE 829

3. Techniques Statiques (K2) — 60 minutes

Objectifs de connaissance pour Techniques Statiques

Les objectifs identifient ce que vous serez capable de faire à la suite de chaque module.

3.1 Techniques statiques et processus de test (K2)

LO-3.1.1 Reconnaître les livrables qui peuvent être examinés par les différentes techniques statiques. (K1)
LO-3.1.2 Décrire l'importance et la valeur de l'utilisation de techniques statiques dans l'évaluation de livrables logiciels. (K2)
LO-3.1.3 Expliquer les différences entre les techniques statiques et dynamiques, en considérant les objectifs, les types de défauts à identifier et le rôle de ces techniques dans le cycle de vie du logiciel. (K2)

3.2 Processus de revue (K2)

LO-3.2.1 Rappeler les activités, rôles et responsabilités d'une revue formelle typique. (K1)
LO-3.2.2 Expliquer les différences entre les différents types de revues: revue informelle, revue technique, relecture technique et inspection. (K2)
LO-3.2.3 Expliquer les facteurs liés à l'exécution de revues couronnées de succès. (K2)

3.3 Analyse statique outillée (K2)

LO-3.3.1 Rappeler les défauts et erreurs typiques identifiées par les analyses statiques et les comparer à ceux détectés par les revues et tests dynamiques. (K1)
LO-3.3.2 Décrire, en utilisant des exemples, les avantages typiques des analyses statiques. (K2)
LO-3.3.3 Lister les défauts typiques dans le code et la conception qui peuvent être identifiés par des outils d'analyse statique. (K1)

3.1 *Techniques statiques et processus de test (K2)* — *15 minutes*

Termes
Test dynamique, test statique.

Contexte
Contrairement au test dynamique, qui exige l'exécution du logiciel, les techniques de tests statiques reposent sur l'examen manuel (revues) ou l'analyse(analyse statique) du code ou de la documentation du projet sans exécution du code.

Les revues sont une manière de tester des produits logiciels (y compris du code) et peuvent être exécutées bien avant l'exécution de tests dynamiques. Les défauts détectés pendant les revues effectuées tôt dans le cycle de vie sont souvent bien moins coûteux à ôter que les défauts détectés lors de l'exécution des tests (p.ex. défauts trouvés dans les exigences).

Une revue pourrait être effectuée entièrement manuellement, mais il existe aussi des outils de support. L'activité manuelle principale est d'examiner le produit et d'en faire des commentaires. Tout produit logiciel peut être revu, y compris les exigences, les spécifications, les spécifications de conception, le code, les plans de test, les spécifications de test, les cas de test, les scripts de test, les guides utilisateur ou pages web.

Les bénéfices des revues incluent une détection et une correction anticipées des défauts , des améliorations de productivité dans le développement, des durées de développement réduites, des durées et des coûts de tests réduits, des réductions de coûts tout au long de l'existence du logiciel, moins de défauts et une meilleure communication. Les revues peuvent détecter des omissions, par exemple, dans des exigences, dont la détection pendant les tests dynamiques est peu probable.

Les revues, les analyses statiques et les tests dynamiques ont le même objectif – identifier des défauts. Ils sont complémentaires : les différentes techniques peuvent trouver différents types de défauts efficacement et rapidement. Contrairement aux tests dynamiques, les techniques statiques trouvent les causes des défauts plutôt que les défaillances elles-mêmes.

Les défauts typiques plus faciles à trouver lors de revues que pendant les tests dynamiques sont : déviations par rapport aux standards, défauts d'exigences, défauts de conception, maintenabilité insuffisante et spécifications incorrectes d'interfaces.

3.2 *Processus de revue (K2)* — *25 minutes*

Termes
Critère d'entrée, revue formelle, revue informelle, inspection, métriques, modérateur, revue de pairs, réviseur, scribe, revue technique, relecture technique.

Contexte
Les différents types de revues varient de « informel », caractérisé par l'absence d'instructions écrites pour les relecteurs, à « systématique », caractérisé par la participation de l'équipe, des résultats documentés de la revue, et des procédures documentées pour mener la revue. La formalité d'un processus de revue est liée à des facteurs tels que la maturité du processus de développement, toute disposition légale ou exigence réglementaire ou la nécessité de traces d'audit.

La manière dont une revue est exécutée dépend des objectifs convenus pour la revue (p.ex. trouver des défauts, augmenter la compréhension, former les testeurs et les nouveaux membres d'une équipe ou organiser la discussion et décider par consensus).

3.2.1 Phases d'une revue formelle (K1)
Une revue formelle typique comprend les phases principales suivantes:

1. Planification :
   - Définir les critères de revues
   - Choisir le personnel
   - Allouer les rôles
2. Définition des critères d'entrée et de sortie pour des types de revues plus formels (p.ex., inspections)
   - Sélectionner la partie des documents à revoir
3. Lancement:
   - Distribuer les documents
   - Expliquer les objectifs, le processus et les documents aux participants
4. Vérification des critères d'entrée (pour des types de revues plus formels)
5. Préparation individuelle
   - Préparer la réunion de revue en revoyant le(s) document(s)
6. Ecriture des défauts potentiels, questions et commentaires
7. Examen/évaluation/enregistrement des résultats (réunion de revue):
   - Discuter ou enregistrer, avec des résultats ou minutes documentés (pour des types de revues plus formels)
   - Noter les défauts, faire des recommandations concernant le traitement des défauts, prendre des décisions à propos des défauts
8. Examen/évaluation et enregistrement pendant toutes les réunions physiques ou enregistrement de toutes les communications électroniques
9. Re travail
10. Correction des défauts détectés (réalisé généralement par l'auteur)
    - Enregistrer le statut modifié des défauts (dans les revues formelles)
11. Suivi:
    - Vérifier que les défauts ont bien été traités
    - Récolter les métriques
12. Contrôle sur la base des critères de sorties (pour des types de revues plus formels)

3.2.2 Rôles et responsabilités (K1)
Une revue formelle typique inclura les rôles ci-dessous :

- Manager : décide l'exécution des revues, alloue le temps dans la planification du projet et détermine si les objectifs de revue ont été atteints.
- Modérateur: la personne qui dirige la revue du document ou de l'ensemble des documents, incluant la planification de la revue, l'exécution de la revue, et le suivi post-réunion. Si besoin, le modérateur peut servir d'intermédiaire entre les différents points de vue et est souvent la personne sur qui repose le succès d'une revue.
- Auteur : l'auteur ou la personne à qui incombe la responsabilité principale du ou des document(s) à revoir.
- Réviseurs : les individus avec une culture technique ou métier spécifique (aussi appelés vérificateurs ou inspecteurs) qui, après la préparation nécessaire, identifient et décrivent les constatations (p.ex. défauts) dans le produit en cours de revue. Les réviseurs devraient être choisis pour représenter les perspectives et rôles différents dans le processus de revue et prennent part à toute réunion de revue.
- Scribe (ou greffier): documente tous les aspects, problèmes et points ouverts identifiés pendant la réunion.

En examinant des documents selon différentes perspectives, et en utilisant des check-lists, les revues peuvent devenir plus efficaces et rentables, par exemple, une check-list basée sur la perspective d'un utilisateur, d'un mainteneur, d'un testeur ou d'un opérateur, ou une check-list reprenant des problèmes d'exigences typiques.

3.2.3 Types de revues (K2)

Un produit logiciel unique ou les éléments associés peuvent être le sujet de plusieurs revues. Si plus d'un type de revue est utilisé, l'ordre peut varier. Par exemple, une revue informelle peut être effectuée avant une revue technique, ou une inspection peut être effectuée sur une spécification d'exigences, avant une relecture technique avec des clients. Les caractéristiques principales, options et objectifs des types de revues habituelles sont:

Revue informelle

- Pas de processus formel;
- Peut inclure la programmation par paires ou une revue de conception et de code par un responsable technique;
- Les résultats peuvent être documentés;
- Peut varier en utilité selon les réviseurs;
- Objectif principal : manière bon marché d'obtenir des résultats.

Relecture technique

- Réunion dirigée par l'auteur;
- Peut prendre la forme de scénarios, répétitions à blanc, participation de groupes de pairs;
- Sessions sans limite de durée;
  - Optionnellement une réunion de préparation de revue par les réviseurs
  - Optionnellement préparation d'un rapport de revue incluant une liste de constatations
- Optionnellement un scribe (qui n'est pas l'auteur) ;
- Varie en pratique de quasiment informel à très formel;
- Objectifs principaux: apprendre, gagner en compréhension, trouver des défauts.

Revue technique

- Documentée, processus de détection de défauts défini incluant des pairs et des experts techniques avec optionnellement la participation de l'encadrement;
- Peut être effectuée comme une revue de pairs sans participation de l'encadrement;
- Idéalement dirigée par un modérateur formé (pas l'auteur);
- Réunion de préparation par les réviseurs;
- Peut optionnellement utiliser des check-lists ;

- Préparation d'un rapport de revue incluant la liste de constatations, le verdict indiquant si le produit logiciel répond à ses exigences et, si approprié, des recommandations relatives aux constatations ;
- Peut varier en pratique de quasiment informelle à très formelle;
- Objectifs principaux: discuter, décider, évaluer des alternatives, trouver des défauts, résoudre des problèmes techniques et vérifier la conformité aux spécifications, plans, réglementations et standards.

I**nspection**

- Dirigée par un modérateur formé (pas l'auteur);
- Généralement menée comme un examen par les pairs;
- Rôles définis;
- Inclut des métriques ;
- Processus formel basé sur des règles et des check-lists ;
- Critères d'entrée et de sortie spécifiés pour l'acceptation du produit logiciel ;
- Réunion de préparation;
- Rapport d'inspection incluant la liste de constatations;
- Processus formel de suivi;
  - Optionnellement, processus d'amélioration des composants
- Optionnellement un lecteur;
- Objectif principal: trouver des défauts.

Les relectures techniques, les revues techniques et les inspections peuvent être réalisées au sein d'un groupe de pairs, càd. des collègues ayant le même niveau dans l'organisation. Ce type de revue est appelé "revue de pairs" ".

3.2.4 Facteurs de succès des revues (K2)

Les facteurs de succès des revues incluent:

- Chaque revue a des objectifs prédéfinis et clairs.
- Les personnes impliquées sont adéquates pour les objectifs de la revue.
- Les testeurs sont des réviseurs de valeur qui contribuent à la revue et ainsi prennent connaissance du produit afin de pouvoir préparer les tests plus tôt.
- Les défauts trouvés sont bien acceptés, et exprimés objectivement.
- Les aspects personnels et psychologiques sont traités (p.ex. en faisant de cela une expérience positive pour l'auteur)).
- La revue est menée dans une atmosphère de confiance ; les résultats ne sont pas utilisés pour évaluer les participants ;
- Les techniques de revue adaptées aux objectifs, adaptées aux types et au niveau de livrable logiciel, et adaptées aux types et niveau des réviseurs, sont appliquées.
- Des check-lists ou des rôles sont utilisés lorsque cela est approprié, afin d'augmenter l'efficacité de détection des défauts.
- Des formations sont données sur les techniques de revue, en particulier celles concernant les techniques plus formelles telles que les inspections.
- L'encadrement supporte un bon processus de revue (p.ex. en incorporant du temps pour les activités de revue dans les plannings des projets).
- L'accent est mis sur l'apprentissage et l'amélioration du processus.

3.3 *Analyse statique avec des outils (K2)* — *20 minutes*

Termes
Compilateur, complexité, flux de contrôle, flux de données, analyse statique.

Contexte
L'objectif de l'analyse statique est de trouver des défauts dans le code source et les modèles logiciels. L'analyse statique est effectuée sans vraiment exécuter le code examiné par l'outil ; les tests dynamiques exécutent le code logiciel. L'analyse statique peut détecter des défauts qui sont difficiles à trouver par les tests dynamiques. Comme pour les revues, l'analyse statique trouve des défauts plutôt que des défaillances. Les outils d'analyse statique analysent le code du programme (p.ex. les flux de contrôle et les flux de données), ainsi que les sorties générées telles que HTML et/ou XML.

La valeur des tests statiques est :
- La détection très tôt de défauts avant l'exécution des tests.
- Une information très tôt sur certains aspects suspects du code ou de la conception, par le calcul de métriques, par exemple une mesure de complexité élevée.
- L'identification de défauts difficilement détectables par des tests dynamiques.
- La détection de dépendances et d'inconsistances dans les modèles logiciels tels que des liens dans les modèles logiciels
- L'amélioration de la maintenabilité du code et de la conception.
- La prévention des défauts, si les leçons sont prises en compte lors du développement.

Les défauts typiques découverts par des outils d'analyse statique incluent :
- Référencement d'une variable avec une valeur indéfinie;
- Interface inconsistante entre modules et composants;
- Variables qui ne sont jamais utilisées ou déclarées de façon incorrecte;
- Code non accessible (code mort);
- Logique absente et erronée (potentiellement des boucles infinies) ;
- Constructions trop compliquées ;
- Violation des standards de programmation;
- Vulnérabilités de sécurité;
- Violation de syntaxe dans le code et les modèles logiciels.

Les outils d'analyse statique sont typiquement utilisés par des développeurs (vérification par rapport à des règles prédéfinies ou des standards de programmation) avant et pendant les tests de composants et les tests d'intégration, ou pendant la mise à jour du code dans les outils de gestion de configuration, et par les concepteurs pendant la modélisation logicielle. Les outils d'analyse statique peuvent produire un nombre important de messages d'avertissement qui doivent être gérés convenablement afin de permettre une utilisation optimale de l'outil.

Les compilateurs peuvent offrir un certain support aux analyses statiques, notamment en incluant le calcul de métriques.

Références
3.2 IEEE 1028
3.2.2 Gilb, 1993, van Veenendaal, 2004
3.2.4 Gilb, 1993, IEEE 1028
3.3 Van Veenendaal, 2004

4. Techniques de Conception de Tests (K3) — 255 minutes

Objectifs de connaissance pour Techniques de Conception de Tests

Les objectifs identifient les capacités que vous aurez à la fin de chaque module.

4.1 Le processus de développement de test (K3)

LO-4.1.1 Différencier la spécification de conception de test de la spécification des cas de test et des spécifications de procédures de test. (K2)
LO-4.1.2 Comparer les termes : condition de test, cas de test et procédure de test. (K2)
LO-4.1.3 Evaluer la qualité des cas de test en terme de traçabilité vers les exigences et les résultats attendus (K2)
LO-4.1.4 Traduire les cas de test en des spécifications de procédures de tests correctement structurées avec le niveau de détail adéquat par rapport au niveau de connaissance des testeurs. (K3)

4.2 Catégories de techniques de conception de tests (K2)

LO-4.2.1 Rappeler les raisons pour lesquelles les approches basées sur les spécifications (boîte-noire) et celles basées sur les structures (boîte blanche) sont utiles, et lister les techniques habituelles pour chacune d'entre elles. (K1)
LO-4.2.2 Expliquer les caractéristiques, les points communs et différences entre les tests basés sur les spécifications, les tests basés sur les structures et les tests basés sur l'expérience. (K2)

4.3 Techniques basées sur les spécifications ou boîte noire (K3)

LO-4.3.1 Ecrire les cas de test pour un modèle logiciel donné en utilisant les partitions d'équivalence, l'analyse des valeurs limites, les tables de décision et les diagrammes/tables de transition d'états (K3)
LO-4.3.2 Expliquer les objectifs principaux de chacune des quatre techniques, quel niveau et type de test peut utiliser la technique, et comment la couverture peut être mesurée. (K2)
LO-4.3.3 Expliquer les concepts des tests basés sur les cas d'utilisation et ses avantages. (K2)

4.4 Techniques basées sur la structure ou boîte blanche (K3)

LO-4.4.1 Décrire le concept et l'importance de la couverture du code. (K2)
LO-4.4.2 Expliquer les concepts de couverture des instructions et des décisions, et donner les raisons pour lesquelles ces concepts peuvent aussi être utilisés pour d'autres niveaux de tests que les tests de composants (p.ex. sur les procédures métier au niveau système). (K2)
LO-4.4.3 Ecrire les cas de test à partir des données du flux de contrôle en utilisant les techniques de conception de tests de couverture des instructions et couverture des décisions. (K3)
LO-4.4.4 Evaluer la complétude des couvertures d'instructions et des décisions en respectant les critères de sortie définis. (K4)

4.5 Techniques basées sur l'expérience (K2)

LO-4.5.1 Rappeler les raisons justifiant l'écriture des cas de test basés sur l'intuition, l'expérience et la connaissance des défauts communs. (K1)
LO-4.5.2 Comparer les techniques basées sur l'expérience avec les techniques basées sur les spécifications. (K2)

4.6 Sélectionner les techniques de test (K2)

LO-4.6.1 Classer les techniques de conception de tests en fonction de leur adaptation à un contexte donné, pour la base de tests, les caractéristiques respectives aux modèles et au logiciel. (K2)

4.1 *Le processus de développement de test (K3)* — *15 minutes*

Termes
Spécification de cas de test, conception de tests, ordonnancement de l'exécution de tests, spécification des procédures de test, script de test, traçabilité.

Contexte
Le processus de développement de test décrit dans cette section peut être effectué de diverses manières, de « très informelles » avec peu ou pas de documentation, à « très formelles » (comme décrit dans cette section). Le niveau de formalisme dépend du contexte des tests incluant la maturité des tests et des processus de développement, des contraintes de temps, des exigences de sureté de fonctionnement ou réglementaires et des personnes impliquées.

Pendant l'analyse de conception des tests, la documentation de la base des tests est analysée de façon à déterminer ce qui est à tester, c'est à dire à identifier les conditions de test. Une condition de test est définie comme un article ou événement qui peut être vérifié par un ou plusieurs cas de test (p.ex. une fonction, transaction, caractéristique qualité ou élément structurel).

Etablir la traçabilité des conditions de tests vers les spécifications et exigences permet à la fois l'analyse d'impact, quand les exigences changent, et une couverture des exigences à définir pour un ensemble de tests. Pendant l'analyse de conception des tests, les approches détaillées de tests sont implémentées pour sélectionner des techniques de tests basées sur, entre autres considérations, les risques identifiés (voir Chapitre 5 pour plus d'informations sur les analyses de risques).

Pendant la conception des tests, les cas de test et données de test sont créés et spécifiés. Un cas de test consiste en un ensemble de valeurs d'entrée, de pré-conditions d'exécution, de résultats attendus et de post-conditions d'exécution, développé pour couvrir certains objectifs de tests et conditions de tests. Le Standard pour la Documentation de Tests Logiciel ('Standard for Software Test Documentation' IEEE 829-1998) décrit le contenu des spécifications de conception de tests (contenant les conditions de test) et des spécifications des cas de test.

Les résultats attendus devraient être produits comme faisant partie des spécifications de cas de test et inclure les sorties, modifications de données et d'états, et toute autre conséquence du test. Si des résultats attendus n'ont pas été définis alors un résultat plausible mais erroné peut être interprété comme un résultat correct. Les résultats attendus devraient idéalement être définis avant l'exécution des tests.

Pendant l'implémentation des tests, les cas de test sont développés, implémentés, priorisés et organisés dans la spécification de procédure de test (IEEE STD 829-1998). La procédure de tests spécifie la séquence des actions pour l'exécution d'un test. Si les tests sont exécutés avec un outil d'exécution des tests, la séquence d'actions est spécifiée dans un script de tests (qui est une procédure de test automatisée).

Les diverses procédures de tests et scripts de tests automatisés sont ensuite consolidées en un planning d'exécution de tests qui définit l'ordre dans lequel les diverses procédures de test, et potentiellement les scripts de tests automatisés, sont exécutés. Les plannings d'exécution des tests prendront en compte les facteurs tels que les tests de régression, de priorisation, et de dépendances technique et logique.

4.2 *Catégories de techniques de conception de tests (K2)* — *15 minutes*

Termes
Technique de conception boîte noire, technique de conception basée sur l'expérience, technique de conception de test, technique boîte blanche.

Contexte
L'objectif de la technique de conception des tests est d'identifier les conditions de tests, les cas de test et les données de tests.

Il est fréquent de faire la distinction entre les techniques de tests boîte blanche et les techniques de test boîte noire. Les techniques de conception boîte noire (aussi appelées techniques basées sur les spécifications) sont une façon de dériver et de sélectionner les conditions de tests, les cas de test ou les données de test en se basant sur une analyse de la documentation de la base des tests. Ceci inclut les tests fonctionnels et non fonctionnels. Le test boîte noire, par définition, n'utilise aucune information concernant la structure interne d'un composant ou système à tester. Les techniques de conception boîte blanche (aussi dites techniques structurelles ou basées sur les structures) sont basées sur une analyse de la structure du composant ou du système. Les tests boîte noire et boîte blanche peuvent aussi s'inspirer de l'expérience des développeurs, des testeurs et des utilisateurs pour déterminer ce qui doit être testé.

Quelques techniques se retrouvent dans une seule catégorie, d'autres comprennent des éléments de plus d'une catégorie.

Ce syllabus référence comme techniques boîte noire les approches basées sur les spécifications, et référence comme techniques boîte blanche les approches basées sur les structures. De plus, les techniques de conception basées sur l'expérience sont couvertes.

Caractéristiques habituelles des techniques de conception basées sur les spécifications:

- Les modèles, soit formels soit informels, sont utilisés pour la spécification des problèmes à résoudre, des logiciels ou des composants.
- Depuis ces modèles les cas de test sont dérivés de façon systématique.

Caractéristiques habituelles des techniques de conception basées sur la structure:

- L'information sur la manière dont le logiciel est construit est utilisée pour dériver les cas de test (par exemple, le code et les informations de conception détaillée).
- Le niveau de couverture du logiciel peut être mesuré à partir de cas de test existants, et des cas de test complémentaires peuvent être dérivés de façon systématique pour augmenter la couverture.

Caractéristiques habituelles des techniques de conception basées sur l'expérience:

- La connaissance et l'expérience des personnes sont utilisées pour dériver les cas de test.
- La connaissance des testeurs, développeurs, utilisateurs et autres parties prenantes concernant le logiciel, son utilisation et son environnement sont autant de sources d'information;
- La connaissance des défauts possibles et de leur distribution est une autre source d'information.

4.3 *Techniques basées sur les spécifications ou techniques boîte noire (K3)* — *120 minutes*

Termes
Analyse des valeurs limites, tests par tables de décisions, partitions d'équivalence, tests de transition d'états, tests de cas d'utilisation.

4.3.1 Partitions d'équivalence (K3)

Pour les partitions d'équivalence, les entrées d'un logiciel ou système sont divisées en groupes qui doivent montrer un comportement similaire, de ce fait elles auront un traitement identique. Les partitions (ou classes) d'équivalence peuvent être trouvées pour des données valides, c'est-à-dire des données qui devraient être acceptées et des données invalides, c'est-à-dire des valeurs qui devraient être rejetées. Les partitions peuvent aussi être identifiées pour les sorties, les valeurs internes, les valeurs liées au temps (p.ex. avant ou après un événement) et pour les paramètres d'interface (p.ex. composants intégrés en cours de test pendant les tests d'intégration). Des tests peuvent être conçus pour couvrir toutes les partitions valides et invalides. Les partitions d'équivalence sont applicables à tous les niveaux de tests.

Les partitions d'équivalence peuvent être utilisées pour atteindre des objectifs de couverture d'entrées et de sorties. Elles peuvent être appliquées aux entrées humaines, aux entrées via l'interface du système, ou aux paramètres d'interface des tests d'intégration.

4.3.2 **Analyse des valeurs limites** (K3)

Le comportement au bord de chaque partition d'équivalence risque plus d'être incorrect qu'à l'intérieur, donc les limites sont des zones plus propices pour découvrir des défauts. Les valeurs maximum et minimum d'une partition sont ses valeurs limites. Une valeur limite pour une partition valide est une valeur limite valide; la limite d'une partition invalide est une valeur limite invalide. Des tests peuvent être conçus pour couvrir les valeurs limites valides et invalides. Quand on conçoit des cas de test, une valeur de chaque limite est sélectionnée pour un test.

L'analyse des valeurs limites peut être appliquée à tous les niveaux de tests. C'est relativement aisé à appliquer et la capacité de détection de défauts est élevée. Des spécifications détaillées sont utiles pour déterminer les limites intéressantes.

Cette technique est souvent considérée comme une extension des partitions d'équivalences ou autre technique de conception boîte noire. Elle peut être utilisée pour les classes d'équivalence aussi bien sur les données d'entrées humaines ou écran, par exemple, que sur des limites de temps (par exemple, le time out, les exigences de rapidité de transactions) ou sur des limites de tableaux (par exemple, une taille de tableau de 256*256).

4.3.3 Tests par tables de décisions (K3)

Les tables de décisions sont une bonne façon de capturer des exigences système contenant des conditions logiques, et pour documenter la conception système interne. Elles peuvent être utilisées pour enregistrer les règles de gestion complexes que doit implémenter un système. Lors de la création des tables de décisions, la spécification est analysée, et les actions et conditions du système sont identifiées. Les conditions d'entrée et les actions sont souvent décrites de façon à ce qu'elles peuvent être soit vraies soit fausses (Booléen). Les tables de décision contiennent les conditions de déclenchement, souvent des combinaisons de vrais et faux pour toutes les conditions d'entrée, et sur les actions résultantes pour chaque combinaison de conditions. Chaque colonne de la table correspond à une règle de gestion qui définit une combinaison unique de conditions qui résultent en l'exécution de l'action associée à cette règle. La couverture standard généralement utilisée avec les tables de décisions est d'avoir au moins un test par colonne, ce qui implique typiquement de couvrir toutes les combinaisons de conditions de déclenchement.

La force des tests par les tables de décisions est la création de combinaisons de conditions qui autrement n'auraient pas été traitées pendant les tests. Cela peut être appliqué à toutes les situations quand l'action du logiciel dépend de plusieurs décisions logiques.

4.3.4 Test de transition d'états (K3)

Un système peut montrer plusieurs réponses différentes en fonction des conditions actuelles ou passées (son état). Dans ce cas, cet aspect du système peut être montré par un diagramme d'états et de transitions. Cela permet au testeur de visualiser le logiciel en termes d'états, de transitions entre les états, de données d'entrées ou des événements qui déclenchent des changements d'état (transitions) et des actions qui peuvent résulter de ces transitions. Les états du système ou de l'objet sous test sont séparées, identifiables et en nombre fini.

Une table des états montre la relation entre les états et les entrées, et peut mettre en lumière des transitions possibles invalides.

Des tests peuvent être conçus pour couvrir une séquence typique d'états, pour couvrir chaque état, pour exécuter toutes les transitions, pour exécuter une séquence spécifique de transitions ou tester des transitions invalides.

Les tests de transitions d'états sont fréquemment utilisés dans l'industrie du logiciel embarqué et généralement dans l'automatisation technique. Cependant, la technique est aussi utilisable pour modéliser les objets métier possédant des états spécifiques ou pour tester les dialogues d'interfaces écran (p.ex. pour les applications Internet ou les scénarios métier).

4.3.5 Tests de cas d'utilisation (K2)

Les tests peuvent être spécifiés à partir de cas d'utilisation. Un cas d'utilisation décrit l'interaction entre des acteurs, incluant utilisateurs et système, qui produit un résultat ayant une valeur pour l'utilisateur du système ou le client. Les cas d'utilisation peuvent être décrits à un niveau abstrait (cas d'utilisation métier, indépendant de la technologie, niveau processus métier). Chaque cas d'utilisation a des pré-conditions, qui doivent être atteintes pour qu'un cas d'utilisation soit exécuté avec succès. Chaque cas d'utilisation se termine par des post-conditions, qui sont les résultats observables et l'état final du système après la fin de l'exécution du cas d'utilisation. Un cas d'utilisation a généralement un scénario dominant (c'est à dire le plus plausible), et parfois des branches alternatives.

Les cas d'utilisation décrivent les "flux du processus" dans un système, basé sur une utilisation probable, donc les cas de test dérivés des cas d'utilisation sont extrêmement utiles pour découvrir les défauts dans le flux de traitement pendant l'utilisation réelle du système. Les cas d'utilisation, souvent appelés scénarios, sont très utiles pour concevoir des tests d'acceptation avec la participation du client/utilisateur. Ils permettent aussi de découvrir des défauts d'intégration causés par l'interaction et les interférences entre divers composants, ce que ne découvrent pas les tests individuels de composants. La conception de cas de test à partir des cas d'utilisation peut être combinée avec d'autres techniques de tests basées sur les spécifications.

4.4 *Technique de conception basée sur la structure ou technique de conception boîte blanche (K3)* — *60 minutes*

Termes
Couverture de code, couverture des décisions, couverture des instructions, tests structurels.

Contexte
Les tests basés sur la structure ou tests boîte blanche suivent la structure identifiée du logiciel ou du système, comme décrit dans les exemples suivants:

- Niveau composant: la structure d'un composant logiciel c'est à dire instructions, décisions, branches ou même des chemins distincts
- Niveau intégration: la structure peut être un arbre (ou graphe) d'appel (un diagramme où des modules appellent d'autres modules).
- Niveau système: la structure peut être une structure de menus, des processus métier ou la structure d'une page web.

Dans cette section, trois techniques structurelles liées à la couverture du code, portant sur les instructions et les décisions, sont discutées. Pour les tests des décisions, un diagramme de contrôle de flux peut être utilisé pour visualiser les alternatives de chaque décision.

4.4.1 Test des instructions et couverture (K3)
Dans les tests de composants, la couverture des instructions est l'évaluation du pourcentage d'instructions exécutables qui ont été exercées par une suite de cas de test. Le test des instructions dérive des cas de test pour exécuter des instructions spécifiques, généralement pour accroître la couverture des instructions.

La couverture des instructions est déterminée par le nombre d'instructions exécutables couvertes par des cas de test (conçus ou exécutés) divisé par le nombre de toutes les instructions exécutables dans le code testé.

4.4.2 Test des décisions et couverture (K3)
La couverture des décisions, liées aux tests de branches, est l'évaluation des pourcentages de résultats de décisions (p.ex. les options Vrai et Faux d'une instruction IF) qui ont été traitées par une suite de cas de test. Les cas de test provenant des tests de décisions sont amenés à exécuter des résultats de décisions spécifiques. Les branches forment des points de décision dans le code.

La couverture des décisions est déterminée par le nombre de tous les résultats des décisions couvertes par des cas de test (conçus ou exécutés) divisé par le nombre de tous les résultats possibles des décisions dans le code sous test.

Les tests de décisions sont une forme de contrôle de flux car ils génèrent un flux spécifique de contrôle entre des points de décisions. La couverture des décisions est supérieure à la couverture des instructions: une couverture de 100% des décisions garantit une couverture à 100% des instructions, mais l'inverse n'est pas vrai.

4.4.3 Autres techniques basées sur les structures (K1)
Il existe des niveaux de couverture structurelle plus forts que les couvertures de décisions, par exemple les couvertures de conditions et les couvertures de conditions multiples.

Le concept de couverture peut aussi être appliqué aux autres niveaux de tests. Par exemple, au niveau intégration, le pourcentage des modules, composants ou classes qui ont été exercées par

une suite de cas de test peut être exprimé comme une couverture de modules, composants ou classes.

L'aide via des outils est utile pour le test structurel de code.

4.5 *Techniques basées sur l'expérience (K2)* — *30 minutes*

Termes
Tests exploratoires, (faute) attaque.

Contexte
Le test basé sur l'expérience a lieu lorsque les tests sont conçus à partir des compétences des testeurs, de leur intuition et de leur expérience avec des applications et technologies similaires. Quand ils sont utilisés pour améliorer les techniques systématiques, les tests intuitifs peuvent être utiles pour identifier des tests spéciaux, difficilement atteints par des approches formelles. Cependant, cette technique peut donner des degrés d'efficacité extrêmement différents, selon l'expérience des testeurs.

Une technique basée sur l'expérience communément utilisée est l'estimation d'erreur. Généralement, les testeurs anticipent les défauts basés sur l'expérience. Une approche structurée pour la technique d'estimation d'erreurs est d'énumérer une liste des défauts possibles et de concevoir des tests pour mettre en évidence ces défauts. Cette approche systématique est appelée « attaque par faute » . Ces listes de défauts et de défaillances peuvent être construites sur la base de l'expérience, des données disponibles sur les défauts et anomalies, et de la connaissance commune sur les causes de défaillances.

Les tests exploratoires comprennent la conception et l'exécution des tests, l'écriture des résultats de tests et l'apprentissage (de l'application), basés sur une charte de tests contenant les objectifs de tests, et effectués dans des périodes de temps délimitées. C'est l'approche la plus utile pour augmenter ou compléter d'autres méthodes de tests plus formelles lorsque les spécifications sont rares ou non adéquates, et que le test est soumis à de sévères contraintes de temps. Cela peut servir comme vérification de processus de tests, pour s'assurer que les défauts les plus sérieux sont trouvés.

4.6 *Sélectionner les techniques de tests (K2)* — *15 minutes*

Termes
Pas de terme spécifique.

Contexte
Le choix des techniques de tests à utiliser dépend de différents facteurs incluant le type de système, les standards réglementaires, les exigences client ou contractuelles, le niveau de risque, le type de risque, les objectifs de test, la documentation disponible, la connaissance des testeurs, le temps disponible et le budget, le cycle de vie de développement utilisé, les modèles de cas d'utilisation et l'expérience sur les défauts découverts précédemment.

Quelques techniques sont plus applicables que d'autres à certaines situations et niveaux de tests, d'autres sont applicables à tous les niveaux de tests.

Lors de la création de cas de test, les testeurs utilisent généralement une combinaison de techniques de tests incluant les techniques orientées processus, règles et données pour assurer une couverture adéquate de l'objet testé.

Références
4.1 Craig, 2002, Hetzel, 1998, IEEE 829
4.2 Beizer, 1990, Copeland, 2004
4.3.1 Copeland, 2004, Myers, 1979
4.3.2 Copeland, 2004, Myers, 1979
4.3.3 Beizer, 1990, Copeland, 2004
4.3.4 Beizer, 1990, Copeland, 2004
4.3.5 Copeland, 2004
4.4.3 Beizer, 1990, Copeland, 2004
4.5 Kaner, 2002
4.6 Beizer, 1990, Copeland, 2004

5. Gestion des Tests (K3) — 170 minutes

Objectifs de connaissance pour la gestion des tests
Les objectifs identifient les capacités que vous aurez à la fin de chaque module

5.1 Organisation des tests (K2)

LO-5.1.1 Reconnaître l'importance du test indépendant. (K1)

LO-5.1.2 Énumérer les avantages et inconvénients du test indépendant pour une organisation. (K2)
LO-5.1.3 Reconnaître les différents membres d'une équipe à envisager lors de la formation d'une équipe de test. (K1)
LO-5.1.4 Rappeler les tâches typiques d'un responsable de test et d'un testeur. (K1)

5.2 Estimation et planification du test (K3)

LO-5.2.1 Reconnaître les différents niveaux et objectifs de la planification du test. (K1)
LO-5.2.2 Résumer le but et le contenu des documents plan de test, conception de tests, spécification de tests et procédure de test suivant le guide « Standard for Software Test Documentation » (IEEE 829). (K2)
LO-5.2.3 Faire la différence entre des approches de test conceptuellement différentes, telles que des approches analytiques; basées sur des modèles; méthodiques à des fins de compatibilité avec un processus ou un standard; dynamique/heuristique; consultative et visant à empêcher la régression. (K2)
LO-5.2.4 Faire la différence entre la planification des tests pour un système et l'ordonnancement de l'exécution des tests. (K2)
LO-5.2.5 Ecrire un calendrier d'exécution des tests pour une série de cas de test donnés en tenant compte des priorités ainsi que des dépendances logiques et techniques (K3)
LO-5.2.6 Lister les activités de préparation et d'exécution des tests qui doivent être prises en compte lors de la planification des tests (K1)
LO-5.2.7 Rappeler les facteurs typiques qui influencent l'effort de test (K1)
LO-5.2.8 Faire la différence entre deux approches d'estimation conceptuellement différentes : l'approche basée sur des mesures et l'approche basée sur l'expertise. (K2)
LO-5.2.9 Reconnaître et justifier les critères de début et de fin de test appropriés à des niveaux de test spécifiques et des groupes de cas de test (par exemple, test d'intégration, test de recette ou cas de test pour un test d'utilisabilité). (K2)

5.3 Suivre et contrôler le déroulement du test (K2)

LO-5.3.1 Rappeler les mesures habituelles utilisées pour suivre la préparation et l'exécution du test (K1)
LO-5.3.2 Comprendre et interpréter les mesures de test pour la documentation et le contrôle du test (par exemple, les défauts trouvés et corrigés, les tests réussis et défaillants) en fonction du contexte (K2)
LO-5.3.3 Résumer le but et le contenu du rapport de synthèse établi selon le guide « Standard for Software Test Documentation » (IEEE Std 829-1998) (K2)

5.4 Gestion de configuration (K2)

LO-5.4.1 Résumer la manière dont la gestion de configuration assiste le test. (K2)

5.5 Test et risques (K2)

LO-5.5.1 Décrire un risque comme un problème probable qui peut compromettre l'atteinte des objectifs de projet d'un ou de plusieurs acteurs (K2)
LO-5.5.2 Se rappeler que le niveau de risque est déterminé par sa probabilité (d'occurrence) et son impact (dommages en résultant) (K1)
LO-5.5.3 Distinguer entre les risques liés au projet et ceux liés au produit (K2)

LO-5.5.4 Reconnaître les risques typiques du produit et du projet (K1)
LO-5.5.5 Décrire, avec l'appui d'exemples, comment utiliser l'analyse et la gestion de risques pour la planification du test (K2)

5.6 Gestion d'incidents (K3)

LO-5.6.1 Reconnaître le contenu du rapport d'incident établi selon le guide « Standard for Software Test Documentation » (IEEE Std 829-1998) (K1)
LO-5.6.2 Rédiger un rapport d'incident couvrant l'observation d'une défaillance pendant le test. (K3)

5.1 *Organisation des tests (K2)* *30 minutes*

Termes
Testeur, responsable du test, gestionnaire du test.

5.1.1 Organisation du test et indépendance **(K2)**

L'efficacité de la découverte d'anomalies par le test et les revues peut être améliorée par l'emploi de testeurs indépendants. Les options pour l'indépendance sont les suivantes :

- Pas de testeurs indépendants, développeurs testant leur propre code
- Testeurs indépendants incorporés à l'équipe de développement.
- Équipe ou groupe de test indépendant au sein de l'organisation, qui réfère au gestionnaire du projet ou aux responsables décisionnaires.
- Testeurs indépendants de l'organisation, de la communauté des utilisateurs et de l'Informatique.
- Spécialistes de test indépendants pour des objectifs spécifiques de test tels que des tests d'utilisabilité, de sécurité informatique ou de certification (qui certifient un produit logiciel par rapport à des normes ou réglementations).
- Testeurs indépendants externes à l'organisation

Pour des projets de grande taille, complexes ou présentant un niveau de sécurité critique, il est habituellement préférable d'avoir plusieurs niveaux de tests, dont certains ou tous sont traités par des testeurs indépendants. L'équipe de développement peut prendre part aux tests, plus spécialement aux plus bas niveaux, mais son manque d'objectivité limite son efficacité. Les testeurs indépendants peuvent avoir l'autorité pour exiger et définir des processus et règles, mais les testeurs ne devraient accepter ce rôle touchant aux processus qu'en présence d'un mandat clair du management.

Les avantages de l'indépendance comprennent les suivants :

- Des testeurs indépendants voient des défauts différents et d'une autre nature et sont impartiaux.
- Un testeur indépendant peut vérifier les hypothèses faites pendant la spécification et l'implémentation du système.

Les inconvénients comprennent les suivants :

- Déconnexion vis-à-vis de l'équipe de développement (en cas de totale indépendance).
- Les développeurs perdent le sens de la responsabilité pour la qualité.
- Des testeurs indépendants peuvent constituer un goulet d'étranglement comme dernier point de vérification et être accusés des retards.

Les tâches de test peuvent être exécutées par des personnes jouant un rôle spécifique du test, mais aussi par quelqu'un exerçant un autre rôle, comme le responsable de projet, le gestionnaire de la qualité, un développeur, un expert métier ou domaine, infrastructure ou exploitation de l'informatique.

5.1.2 Tâches du responsable des tests et des testeurs **(K1)**

Deux fonctions sont abordées dans ce syllabus, celles du responsable du test et du testeur. Les activités et tâches accomplies par des personnes dans ces deux rôles dépendent du contexte du produit et du projet, ainsi que des personnes jouant ces rôles et de l'organisation.

Parfois, le responsable de test est appelé gestionnaire de test ou coordinateur de test. Ce rôle peut être rempli par un chef de projet, un responsable de développement, un responsable de la qualité

ou un responsable d'un groupe de test. Typiquement, le responsable de test planifie, surveille et contrôle les activités et tâches de test définies dans la section 1.4.

Les tâches habituelles du responsable de test peuvent comprendre les suivantes :

- Coordonner la stratégie et le plan du test avec le chef de projet et d'autres acteurs.
- Établir ou réviser une stratégie de test pour le projet ainsi qu'une politique de test pour l'organisation.
- Apporter le point de vue du test aux autres activités du projet, comme la planification de l'intégration.
- Planifier les tests – en considérant le contexte et en comprenant les objectifs et les risques – y compris la sélection des approches de test, l'estimation du temps, de l'effort et des coûts du test, l'acquisition des ressources, la définition des niveaux de test, les cycles, l'approche et les objectifs ainsi que la planification de la gestion d'incidents;
- Démarrer la spécification, la préparation, l'implémentation et l'exécution des tests ainsi que surveiller et contrôler l'exécution.
- Adapter le planning en fonction des résultats et de l'avancement du test (parfois documenté dans un rapport d'état d'avancement) et entreprendre les actions nécessaires pour résoudre les problèmes
- Mettre en place une gestion de configuration adéquate du logiciel de test à des fins de traçabilité.
- Introduire des mesures appropriées pour mesurer l'avancement du test et évaluer la qualité du test et du produit
- Décider ce qui devrait être automatisé, à quel degré et comment.
- Sélectionner les outils pour aider le test et organiser la formation des testeurs à l'usage des outils.
- Décider de la mise en œuvre de l'environnement de test.
- Établir des rapports de synthèse de test à partir des informations recueillies pendant le test.

Les tâches habituelles des testeurs peuvent comprendre les suivantes :

- Passer en revue les plans du test et y contribuer.
- Analyser, passer en revue et évaluer, quant à leur testabilité, les exigences utilisateurs, les spécifications et les modèles.
- Créer des spécifications de test.
- Mettre en place l'environnement de test (souvent en coordination avec l'administration système et la gestion de réseau).
- Préparer et obtenir les données de test.
- Implémenter des tests à tous les niveaux, exécuter et consigner les tests, évaluer les résultats et documenter les écarts vis-à-vis des résultats attendus.
- Employer les outils d'administration ou de gestion des tests et les outils de surveillance des tests en fonction du besoin.
- Automatiser les tests (éventuellement avec l'aide d'un développeur ou d'un expert en automatisation de test).
- Mesurer les performances des composants et systèmes (si pertinent).
- Passer en revue les tests développés par d'autres.

Les personnes travaillant sur l'analyse, la conception des tests, sur des types de tests spécifiques ou l'automatisation des tests peuvent être des spécialistes de ces rôles. Selon le niveau de test et les risques du projet ainsi que ceux du produit, des personnes différentes peuvent jouer le rôle de testeur, en gardant un certain niveau d'indépendance. Typiquement, les testeurs au niveau du test de composants et d'intégration seront des développeurs, ceux du test d'acceptation seront des experts métier et des utilisateurs et ceux pour l'acceptation opérationnelle seront des opérateurs.

5.2 *Estimation et planification des tests (K3)* — *40 minutes*

Termes
Approche de test, stratégie de test

5.2.1 Planification des tests **(K2)**

Cette section couvre l'objectif de la planification du test dans des projets de développement et d'implémentation ainsi que pour des activités de maintenance. La planification peut être documentée dans un plan de projet ou un plan de test maître ainsi que dans des plans de test séparés pour les niveaux de test, tels que le test système ou le test d'acceptation. Le schéma des documents de planification de test est défini dans le guide « Standard for Software Test Documentation » (IEEE Std 829-1998).

La planification est influencée par la politique de test de l'organisation, la portée du test, les objectifs, les risques, les contraintes, la criticité, la testabilité et la disponibilité des ressources. Au fur et à mesure de l'avancement du projet et de la planification des tests, davantage d'informations seront disponibles et davantage de détails pourront être inclus dans le plan.

La planification du test est une activité continue et est effectuée tout au long des processus et activités du cycle de développement. Le retour issu des activités de test est employé pour constater l'évolution des risques et modifier alors la planification.

5.2.2 Activités de planification des tests (K3)

Les activités de la planification des tests pour un système ou pour une partie de celui-ci peuvent être:

- Définir le périmètre du test, les risques et identifier les objectifs du test
- Définir l'approche générale du test (stratégie de test), y compris la définition des niveaux de test ainsi que celle des critères d'entrée et de sortie.
- Intégrer et coordonner des activités de test dans les activités du cycle de développement : acquisition, fourniture, développement, exploitation et maintenance.
- Prendre des décisions quant à ce qui doit être testé, quels rôles vont exercer quelles activités, quand et comment ces activités doivent être exercées, comment évaluer les résultats des tests et quand arrêter les tests (critères de sortie).
- Planifier les activités d'analyse et de conception des tests
- Planifier les activités de conception, d'exécution et d'évaluation des tests
- Assigner les ressources aux différentes tâches définies.
- Définir le volume, le niveau de détail, la structure et les modèles pour la documentation du test.
- Sélectionner des mesures pour suivre et contrôler la préparation et l'exécution des tests, l'élimination des défauts et la résolution des problèmes relatifs aux risques.
- Déterminer le niveau de détail pour les procédures de test de façon à fournir suffisamment de détails pour permettre une préparation et une exécution reproductibles des tests.

5.2.3 Critères d'entrée (K2)

Les critères d'entrée définissent quand démarrent les tests (par exemple quand débute un niveau de test, quand un jeu de test est prêt à être exécuté)

Typiquement, les critères d'entrée peuvent couvrir:
- Disponibilité et préparation de l'environnement de test
- Préparation des outils de tests dans l'environnement de test
- Disponibilité de code testable
- Disponibilité des jeux de données

5.2.4 Critères de sortie (K2)

L'objectif des critères de sortie est de définir quand arrêter le test, par exemple, à la fin d'un niveau de test ou lorsqu'une série de tests a atteint un objectif donné.

Typiquement, les critères de sortie peuvent comprendre les suivants :

- Des mesures d'exhaustivité, comme la couverture de code, de fonctionnalités ou de risques.
- L'estimation de la densité des anomalies ou des mesures de fiabilité.
- Le coût.
- Les risques résiduels, comme les anomalies non corrigées ou le manque de couverture du test dans certaines parties.
- Un calendrier, par exemple, basé sur la date de mise sur le marché.

5.2.5 Estimation des tests (K2)

Deux approches pour l'estimation de l'effort de test sont couvertes par le présent syllabus :

- Estimation de l'effort de test basée sur des mesures issues de projets antérieurs ou similaires ou basée sur des valeurs typiques.
- L'approche experte : estimation des tâches par le détenteur de ces tâches ou par des experts.

Une fois que l'effort de test a été estimé, il est possible d'identifier les ressources nécessaires et d'établir un échéancier.

L'effort de test peut dépendre d'un certain nombre de facteurs, dont les suivants :

- Les caractéristiques du produit : la qualité des exigences et autres informations utilisées pour les modèles de test (c'est-à-dire la base de test), la taille du produit, la complexité du domaine, les exigences de fiabilité et de sécurité ainsi que celles de la documentation.
- Les caractéristiques du processus de développement : la stabilité de l'organisation, les outils employés, le processus de test, le savoir-faire des personnes impliquées et les contraintes de temps.
- Les résultats du tests : le nombre de défauts et le volume des reprises exigées.

5.2.6 Stratégie de test, Approche de test (K2)

L'approche de test est la mise en œuvre d'une stratégie de test pour un projet spécifique. Elle est définie et affinée dans les plans et scénarii de test. Elle comprend typiquement les décisions basées sur le projet (de test), ses buts ainsi qu'une analyse des risques. Elle constitue le point de départ pour la planification du processus de test, pour sélectionner les types et techniques de test qui seront appliqués ainsi que pour définir les critères d'entrée et de sortie.

L'approche sélectionnée dépend du contexte et peut prendre en considération les risques, la sécurité et les dangers, les ressources disponibles et leur niveau d'expertise, la technologie et la nature du système (par exemple spécifique ou générique (COTS)), les objectifs, les normes et règlements en vigueur.

Les approches typiques peuvent comprendre:

- Une approche analytique, comme le test basé sur les risques où le test est focalisé sur les parties à plus haut risque.
- Les approches basées sur les modèles, comme le test stochastique qui utilise des informations statistiques sur les taux d'erreurs (comme les modèles de croissance de fiabilité) ou sur l'utilisation (comme les profils opérationnels).
- Les approches méthodiques, comme le test basé sur les erreurs (y compris l'estimation d'erreurs et l'attaque par faute), basées sur des listes de vérification et sur des caractéristiques de la qualité.
- Des approches basées sur la conformité aux processus et normes, tels que ceux spécifiés par des normes industrielles spécifiques ou les diverses méthodes agiles.

- Les approches dynamiques et heuristiques, comme le test exploratoire où le test est plutôt réactif aux événements que planifié, et où l'exécution et l'évaluation sont des tâches parallèles.
- Les approches consultatives, comme celles où la couverture de test est principalement définie par les avis et le guidage d'experts métier ou en technologie étrangers à l'équipe de test.
- Les approches basées régression, comme celles qui comportent le réemploi de matériel de test existant, une automatisation extensive du test de régression fonctionnel et des suites de tests standard.

Différentes approches peuvent être combinées, par exemple, une approche dynamique basée sur les risques.

5.3 *Suivi et contrôle du déroulement des tests* (K2)	*20 minutes*

Termes
Densité de défauts, taux de défaillance, contrôle des tests, suivi des tests, rapport de test

5.3.1 Suivi de l'avancement des tests **(K1)**

L'objectif du suivi du test est de fournir un retour et une visibilité sur les activités de test. Les informations à suivre peuvent être recueillies manuellement ou automatiquement et peuvent être utilisées pour évaluer les critères de sortie, comme la couverture. Des mesures peuvent aussi être utilisées pour évaluer l'avancement par rapport au calendrier et au budget planifiés. Les mesures de test habituelles sont:

- Le pourcentage du travail consacré à la préparation des cas de test (ou pourcentage des cas de test planifiés et préparés).
- Pourcentage du travail consacré à la préparation de l'environnement de test.
- L'exécution de cas de test (par exemple, nombre de cas de test exécutés ou non et nombre de cas de test réussis ou échoués).
- Les informations sur les défauts (par exemple, densité des défauts, défauts trouvés et corrigés, taux des défaillances et résultats du re-test).
- Couverture par le test des exigences, des risques ou du code.
- Confiance subjective des testeurs dans le produit.
- Dates des jalons du test.
- Coût du test, y compris le coût de l'avantage de trouver le prochain défaut comparé à celui de l'exécution du test suivant.

5.3.2 Reporting des tests (K2)

Le reporting des tests consiste à résumer les informations relatives à l'entreprise du test ; il comprend les activités suivantes :

- Ce qui s'est passé pendant une phase de test, comme les dates où les critères de sortie ont été atteints.
- Les informations et mesures analysées pour étayer les recommandations et décisions pour de futures actions, comme une évaluation des défauts restants, les avantages économiques de tests prolongés, les risques non couverts et le niveau de confiance dans le logiciel testé.

Le descriptif d'un rapport de synthèse de test se trouve dans le guide « Standard for Software Test Documentation » (IEEE Std 829-1998).

Des mesures devraient être recueillies pendant et à la fin d'un niveau de test dans le but d'évaluer :

- L'adéquation des objectifs du test avec ce niveau de test.
- L'adéquation des approches du test empruntées.
- L'efficacité du test par rapport à ses objectifs.

5.3.3 Contrôle des tests (K2)

Le contrôle du test décrit les actions d'orientation et de correction entreprises comme résultat des informations et mesures recueillies et consignées. Ces actions peuvent couvrir toute activité de test et influencer toute autre activité ou tâche du cycle de vie logiciel.
Exemples d'actions de contrôle des tests :

- Prendre des décisions sur la base des informations recueillies lors du suivi des tests

- Une nouvelle affectation de priorités aux tests en cas de mise en évidence d'un risque identifié (par exemple, retard de livraison du logiciel).
- Une modification du calendrier de test en raison de la disponibilité d'un environnement de test.
- Définition d'un critère d'entrée exigeant que des corrections soient testées par le développeur avant de les accepter dans une version.

5.4 *Gestion de configuration* (K2) — *10 minutes*

Termes
Gestion de configuration, contrôle de versions

Contexte
L'objectif de la gestion de configuration est d'établir et de maintenir l'intégrité des produits (composants, données et documentation) du logiciel ou du système durant le cycle de vie du projet et du produit.

Pour le test, la gestion de configuration peut permettre d'assurer que :

- Tous les éléments faisant partie du testware sont identifiés, sous contrôle de versions, que les changements sont identifiés et re-traçables, reliés les uns aux autres et aux éléments de développement (objets de test), de sorte que la traçabilité peut être maintenue pendant tout le processus du test.
- Tous les documents identifiés et les éléments du logiciel sont référencés de manière non ambiguë dans la documentation de test.

La gestion de configuration aide le testeur à identifier de manière unique (et à reproduire) l'élément testé, les documents de test, les tests et le harnais de test.

Pendant la planification du test, les procédures et l'infrastructure (outils) de la gestion de configuration devraient être choisis, documentés et implémentés.

5.5 *Test et risques* (K2) *30 minutes*

Termes
Risques liés au produit, risques liés au projet, risques, test basé sur les risques

Contexte
Un risque peut être défini comme la probabilité qu'un événement, un danger, une menace ou une situation arrive, et que les conséquences indésirables qui en découlent constituent un problème potentiel. Le niveau de risque sera déterminé par la probabilité qu'un événement adverse arrive et par l'impact de ce dernier (les dommages résultant de cet événement).

5.5.1 Risques liés au projet (K2)
Les risques liés au projet sont les risques menaçant la capacité de ce dernier à atteindre ses objectifs, tels que :

- Facteurs organisationnels :
  - Manque de savoir-faire et de personnel ;
  - Problèmes de personnel;
  - Problèmes politiques, tels que
    - problèmes dus au fait que des testeurs ne communiquent pas leurs besoins et les résultats du test ;
    - incapacité à suivre les informations recueillies pendant le test et les revues (par exemple, ne pas améliorer les pratiques de développement et de test).
  - Attitude ou attentes inappropriées vis-à-vis du test (par exemple, ne pas apprécier la valeur de la découverte de défauts durant le test).
- Problèmes techniques :
  - Problèmes pour définir des exigences correctes ;
  - La mesure selon laquelle les exigences seront satisfaites en fonction de contraintes existantes ;
  - Environnement de test indisponible
  - Conversion des données tardive ; planning de migration et de développement ; conversion des données de test / outils de migration
  - Qualité de la conception, du code et des tests
- Problèmes d'acquisition :
  - Défaillance d'une tierce partie ;
  - Problèmes contractuels.

Pour analyser, gérer et diminuer ces risques, le gestionnaire de test suivra les principes bien établis de la gestion de projet. Le guide « Standard for Software Test Documentation » (IEEE Std 829-1998) pour le plan de test exige que les risques et contingences soient documentés..

5.5.2 Risques liés au produit (K2)
Les défaillances potentielles (événements futurs indésirables ou dangers) des parties du logiciel ou du système sont définies comme des risques liés au produit, puisqu'elles représentent un risque pour la qualité du produit, comme :

- Livraison d'un logiciel défectueux.
- Possibilité qu'un logiciel ou système entraîne des dommages à des personnes ou à des entreprises.
- Caractéristiques logicielles de moindre qualité (par exemple, fonctionnalité, sécurité, fiabilité, utilisabilté et performances).
- Intégrité et qualité des données de faible qualité (par exemple en raison de problèmes liés à leur migration, à leur conversion, à leur transport, à un non respect des standards)
- Logiciel n'offrant pas les fonctionnalités voulues.

Les risques sont employés pour décider quand commencer à tester et où tester davantage ; le test est utilisé pour réduire le risque qu'un événement indésirable ne survienne ou pour réduire l'impact de ce dernier.

Les risques relatifs au produit sont un type particulier de risque pour le succès d'un projet. Le test, comme activité de contrôle des risques fournit un retour quant aux risques restants par la mesure de l'efficacité de l'élimination des défauts critiques et des plans de contingence.

Une approche des tests basée sur les risques fournit des possibilités proactives de réduire le niveau des risques relatifs au produit, en partant des premières étapes d'un projet. Elle comprend l'identification des risques liés au produit et de leur emploi comme guide pour la planification du test, la spécification, la préparation et l'exécution des tests. Dans une approche basée sur les risques, les risques identifiés peuvent être utilisés pour :

- déterminer les techniques de test à employer,
- déterminer le volume du test à effectuer.
- définir les priorités à affecter aux tests dans le but de trouver les défauts critiques le plus tôt possible.
- déterminer si des activités ne faisant pas partie du test pourraient aider à réduire les risques (par exemple, une formation fournie aux développeurs inexpérimentés).

Le test basé sur les risques repose sur les connaissances et la compréhension collectives des acteurs d'un projet pour déterminer les risques et les niveaux de test nécessaires pour couvrir ces derniers.

Pour s'assurer que la probabilité de défaillance d'un produit est minimisée, les activités de gestion des risques fournissent une approche disciplinée pour :

- estimer (et re-estimer régulièrement) ce qui peut faillir (les risques),
- déterminer quels sont les risques importants à couvrir,
- entreprendre des actions pour couvrir ces risques.

De plus, le test peut aider à identifier de nouveaux risques, à déterminer quels risques doivent être minimisés et à réduire l'incertitude relative aux risques.

5.6 *Gestion des incidents* (K3) — *40 minutes*

Termes
Consignation d'incidents, gestion des incidents, rapport d'incidents

Contexte
Comme l'un des objectifs du test est de découvrir des défauts, les différences entre les résultats attendus et les résultats effectifs doivent être consignées en tant qu'incidents. Un incident doit être analysé et peut par la suite devenir un défaut. Des actions adéquates doivent être définies afin de traiter les incidents et les défauts. Les incidents et les défauts doivent être suivis depuis leur découverte et classification jusqu'à leur correction et la confirmation de leur résolution. Pour pouvoir gérer tous les incidents jusqu'à la fin d'un projet, une organisation devrait établir un processus de gestion des incidents et des règles pour leur classification.

Les incidents peuvent survenir pendant le développement, les revues, le test ou l'utilisation d'un produit logiciel. Ils peuvent survenir en raison de problèmes dans le code, dans le système opérationnel ou dans tout type de documentation, notamment les documents de développement, les documents de test ou les informations destinées à l'utilisateur tels que le manuel d'utilisation ou le manuel d'installation.

Les rapports d'incidents peuvent avoir les objectifs suivants :
- Fournir aux développeurs et aux autres parties un retour sur le problème concerné pour en permettre l'identification, la localisation et la correction nécessaires.
- Fournir aux responsables du test le moyen de suivre la qualité d'un système sous test et l'avancement du test.
- Fournir des idées pour l'amélioration du processus de test.

Les détails du rapport d'incident peuvent inclure:
- Date de l'incident, organisation faisant part de l'incident, auteur
- Résultats attendus et effectifs.
- Identification ou élément de configuration du logiciel ou système.
- Processus du cycle de vie du logiciel ou du système au cours duquel l'incident a été observé
- Description de l'anomalie pour permettre son élimination, y compris les traces, extraits de la base de données ou captures d'écrans
- Degré de l'impact sur les intérêts des détenteurs d'enjeux.
- Sévérité de l'impact sur le système.
- Urgence ou priorité de la correction.
- Etat de l'incident (par exemple, ouvert, soumis, doublon, en attente de correction, corrigé en attente de test de confirmation, clôturé).
- Conclusions et recommandations.
- Problèmes globaux, comme d'autres parties pouvant être impactées par une modification résultant de l'incident.
- Historique des modifications, telles que la séquence des actions entreprises par des membres de l'équipe du projet afin de localiser l'incident, de corriger et d'en confirmer la correction.
- Références, incluant l'identité du cas de test ou la spécification qui a permis d'identifier le problème

La structure d'un rapport d'incident est couverte par le guide « Standard for Software Test Documentation » (IEEE Std 829-1998).

References
5.1.1 Black, 2001, Hetzel, 1988
5.1.2 Black, 2001, Hetzel, 1988
5.2.5 Black, 2001, Craig, 2002, IEEE Std 829-1998, Kaner 2002
5.3.3 Black, 2001, Craig, 2002, Hetzel, 1988, IEEE Std 829-1998

5.4 Craig, 2002
5.5.2 Black, 2001 , IEEE Std 829-1998
5.6 Black, 2001, IEEE Std 829-1998

6. Outils de Support aux Tests (K2) — 80 minutes

Objectifs de connaissance pour les outils de support aux tests
Les objectifs identifient ce que vous aurez à la fin de chacun des modules.

6.1 Les types d'outils de tests (K2)

LO-6.1.1 Classer les différents types d'outils de test selon leur sujet et les activités du processus de tests et le cycle de vie logiciel. (K2)
LO-6.1.3 Reconnaître les outils qui peuvent aider les développeurs dans leurs tests. (K2)

6.2 Utilisation pertinente des outils : Avantages et risques potentiels (K2)

LO-6.2.1 Résumer les bénéfices et risques potentiels d'automatisation et d'utilisation d'outils pour les tests. (K2)
LO-6.2.2 Reconnaître les considérations spécifiques pour les outils d'exécution des tests, l'analyse statique, et les outils de gestion de tests. (K1)

6.3 Introduire un outil dans une organisation (K1)

LO-6.3.1 Exposer les principes majeurs liés à l'introduction d'un outil dans une organisation. (K1)
LO-6.3.2 Exposer les objectifs d'une preuve de concept pour l'évaluation d'un outil et une phase de pilotage pour l'implémentation d'un outil. (K1)
LO-6.3.3 Reconnaître que des facteurs autres que l'acquisition d'un outil sont nécessaires pour un bon support des tests par les outils. (K1)

6.1 *Types d'outils de test (K2)* — *45 minutes*

Termes
Outils de gestion de configuration, outils de mesure de couverture, outils de débogage, outils d'analyse dynamique, outils de gestion d'incidents, outils de tests de charge, outils de modélisation, outils de monitoring, outils de tests de performances, effet de sonde, outils de gestion d'exigences, outils de support de revues, outils de sécurité, outils d'analyse statique, outils de test de stress, comparateur de tests, outils de préparation de données de tests, outils de conception de tests, harnais de tests, outils d'exécution des tests, outils de gestion des tests, outils de tests unitaires et structurels.

6.1.1 Comprendre la signification et le but des outils de support aux tests (K2)

Des outils de test peuvent être utilisés pour une ou plusieurs activités que supporte le test. Ceux-ci incluent :

1. Outils qui sont directement utilisés dans le test tel que les outils d'exécution de test, outils de génération de données de test et outils de comparaison de résultats.
2. Outils qui aident en contrôlant le processus de test comme ceux employés pour gérer les tests, des résultats de test, des données, des conditions, des incidents, des défauts, etc., et pour reporter et contrôler l'exécution des tests
3. Outils qui sont utilisés dans la reconnaissance, ou, en termes simples : exploration (par exemple, outils qui contrôle l'activité d'un fichier pour une application)
4. N'importe quel outil qui facilite le test (un tableur est également un outil de test dans cette signification)

L'outil de support au test peut avoir un ou plusieurs des buts suivants selon le contexte :

- Améliorer l'efficacité des activités de test en automatisant des tâches répétitives ou en supportant des activités de test manuelles comme la planification des tests, la conception de tests, le compte-rendu et le contrôle des tests
- Automatiser les activités qui exigent les ressources importantes une fois faites manuellement (par exemple, tests statiques)
- Automatiser les activités qui ne peuvent pas être exécutées manuellement (par exemple, test de performance exhaustif des applications client-serveur)
- Augmenter la fiabilité du test (par exemple, en automatisant la comparaison de beaucoup de données ou en simulant le comportement)

Le terme "framework de test" est également fréquemment utilisé dans l'industrie, avec au moins trois significations :

- Bibliothèques réutilisables et extensibles de test qui peuvent être employées pour construire des outils de test (appelées aussi harnais de tests)
- Un type de conception d'automatisation des tests (par exemple, pilotés par les données, piloté par mots-clés)
- Processus globaux d'exécution de test

Dans ce syllabus, le terme "framework de test" est utilisé dans ses deux premières significations comme décrit dans la section 1.1.6.

6.1.2 Classification des outils de test (K2)

Il y a un certain nombre d'outils qui supportent les différents aspects du test. Des outils peuvent être classés selon plusieurs critères tels que le but, commercial/ libre / logiciel libre / shareware, technologie utilisée et ainsi de suite. Dans le présent syllabus, ces outils sont classés selon les activités du test qu'ils assistent.

Certains outils supportent clairement une seule activité; d'autres supportent plus d'une activité, mais sont classés dans la rubrique relative à l'activité à laquelle ils sont plus étroitement associés. Des outils d'un fournisseur unique, particulièrement ceux qui ont été conçus pour fonctionner ensemble, peuvent être empaquetés dans un module.

Quelques types d'outil de test peuvent être intrusifs, ce qui signifie qu'ils peuvent affecter le résultat obtenu par le test. . Par exemple, une mesure de temps peut varier à cause des instructions supplémentaires exécutées par l'outil., ou vous pouvez obtenir une mesure de couverture de code différente. La conséquence des outils intrusifs s'appelle l'effet de sonde.

Quelques outils offrent une assistance davantage tournée vers les développeurs (càd. pendant les tests de composants et d'intégration des composants). Ces outils sont notés avec un "(D)" dans les classifications ci-dessous.

6.1.3 Outils d'aide à la gestion du test et des tests (K1)

Les outils de gestion s'appliquent à toutes les activités de tests pendant toute la durée du cycle de vie du logiciel.

Outils de gestion des tests
Ces outils fournissent des interfaces pour exécuter des tests, dépister des défauts et gérer les exigences, avec une aide pour l'analyse quantitative et le rapport des objets de tests. Ils supportent également la traçabilité des objets de tests vers les spécifications des exigences. Ils peuvent posséder des fonctionnalités de gestion de version indépendantes ou s'interfacer avec un outil de gestion de version externe.

Outils de gestion des exigences
Ces outils enregistrent les énoncés des exigences, enregistrent les attributs des exigences (priorité y compris), fournissent des identifiants uniques et supportent la traçabilité des exigences vers les tests individuels. Ces outils peuvent également aider à identifier des exigences contradictoires ou manquantes.

Outils de gestion d'incidents (outils de suivi de défauts)
Ces outils enregistrent et gèrent les rapports d'incidents, c.-à-d. les défauts, les défaillances, les demandes de modification ou les problèmes et anomalies rencontrés. Ils aident à gérer le cycle de vie des incidents, éventuellement avec le support de l'analyse statistique.

Outils de gestion de configuration
Bien que pas strictement des outils de test, ceux-ci sont nécessaires pour le stockage et la gestion de version du testware et du logiciel associé particulièrement lorsque sont configurés plusieurs environnements matériel/logiciel en termes de versions du système d'exploitation, compilateurs, navigateurs web, etc.

6.1.4 Outils d'aide aux tests statiques (K1)

Les outils de test statique fournissent une manière rentable de trouver plus de défauts à une étape amont dans le processus de développement.

Outils de revue
Ces outils aident aux revues des processus, des check-lists, des directives de revue et sont utilisés pour enregistrer et communiquer les commentaires des revues, des rapports sur des défauts et l'effort. Ils peuvent être d'un aide supplémentaire en fournissant l'assistance pour des revues en ligne pour de équipes de taille importante ou géographiquement dispersées.

Outils d'analyse statique (D)

Ces outils aident les développeurs et les testeurs à trouver des défauts avant le test dynamique en fournissant une aide pour introduire des normes de codage (codage sécurisé y compris), l'analyse des structures et des dépendances. Ils peuvent également aider dans la planification ou l'analyse de risque en fournissant des métriques pour le code (par exemple, complexité).

Outils de modélisation (D)
Ces outils sont employés pour valider les modèles de logiciel (par exemple, modèle de données physiques (MDP) pour une base de données relationnelle), en énumérant des incohérences et en trouvant des défauts. Ces outils peuvent souvent aider en produisant quelques cas de test basés sur le modèle.

6.1.5 Outils d'aide à la spécification des tests (K1)

Outils de conception de tests
Ces outils sont utilisés pour générer des entrées de tests ou des tests exécutables et/ou des oracles de tests à partir des exigences, des interfaces utilisateur graphiques, des modèles de conception (état, données ou objet) ou à partir du code.

Outils de préparations de données de tests
Les outils de préparation des données agissent sur des bases de données, fichiers ou transferts de données afin d'élaborer des données de tests utilisables lors de l'exécution des tests, ceci en assurant la sécurité des données grâce à leur anonymisation.

6.1.6 Outils d'aide à l'exécution et à l'enregistrement des tests (K1)

Outils d'exécution des tests
Ces outils permettent à des tests d'être exécutés automatiquement, ou semi-automatiquement, utilisant les entrées enregistrées et les résultats prévus, par l'utilisation d'un langage de script et fournissent habituellement un registre de test pour chaque exécution de test. Ils peuvent également être employés pour enregistrer des tests, et supportent habituellement un langage de script ou une configuration via une interface graphique utilisateur pour le paramétrage des données et toute autre personnalisation dans les tests.

Harnais de tests/Outils framework de tests unitaires (D)
Un harnais ou framework de tests unitaires facilite le test des composants ou des parties d'un système en simulant l'environnement dans lequel cet objet de tests s'exécutera, par la fourniture d'objets factices comme des bouchons ou des pilotes.

Comparateurs de tests
Les comparateurs de tests déterminent des différences entre les fichiers, bases de données ou résultats de tests. Les outils d'exécution des tests incluent en général des comparateurs dynamiques, mais les comparaisons post-exécution peuvent être faites par un outil de comparaison distinct. Un comparateur de tests peut utiliser un oracle de tests, en particulier s'il est automatisé.

Outils de mesure de couverture (D)
Ces outils, par des moyens intrusifs ou non intrusifs, mesurent le pourcentage de certains types de structures de code qui ont été exercés (par exemple, des déclarations, des branches ou des décisions, et appels de module ou fonction) par un ensemble de tests.

Outils de test de sécurité
Ces outils sont utilisés pour évaluer les caractéristiques de sécurité du logiciel. Ceci inclut l'évaluation de la capacité du logiciel à protéger la confidentialité, l'intégrité, l'authentification, l'autorisation, la disponibilité, et la non-répudiation des données. Les outils de sécurité sont en grande partie concentrés sur une technologie, une plate-forme, et un but particulier.

6.1.7 Outils de support de performance et de surveillance (K1)

Outils d'analyse dynamique (D)

Les outils d'analyse dynamique détectent des défauts qui ne se manifestent que lors de l'exécution du logiciel, telles que les dépendances temporelles ou les fuites de mémoire. Ils sont typiquement utilisés dans des tests de composants et d'intégration de composants, et dans les tests de middleware.

Outils de test de performance/test de charge/test de stress
Les outils de test de performance surveillent et rapportent sur la façon dont se comporte un système selon une grande variété de conditions d'usage simulées en termes de nombre d'utilisateurs simulanés, de leur modèle de montée en charge, de fréquence et de pourcentage relatif de transactions. La simulation de la charge est réalisée au moyen de la création d'utilisateurs virtuels effectuant un ensemble choisi de transactions diffusées à travers divers machines de test identifiés comme injecteurs de charge.

Outils de surveillance
Les outils de surveillance analysent continuellement, vérifient et rendent compte de l'utilisation de ressources systèmes spécifiques, et donnent des alertes sur de possibles problèmes de service.

6.1.8 Outils de support pour des besoins de tests spécifiques (K1)

Evaluation de la qualité des données
Les données sont au centre de certains projets tels que des projets de conversion/migration des données et des applications comme le stockage de données. Leurs caractéristiques peuvent varier en termes de criticité et volume. Dans de tels contextes, des outils d'évaluation de la qualité des données doivent être utilisés pour revoir et vérifier les règles de conversion et de migration des données, pour s'assurer que les données traitées sont correctes, complètes et se conforment à une norme prédéfinie spécifique au contexte.

D'autres outils de test existent pour le test d'utilisabilité.

6.2 *Utilisation efficace des outils : Bénéfices potentiels et Risques (K2)* — *20 minutes*

Termes
Tests pilotés par les données, tests pilotés par mots clé, langage de script

6.2.1 Bénéfices potentiels et risques liés aux outils de test (pour tous les outils) (K2)

Le simple achat, ou la location d'un outil ne garantit par le succès. Chaque type d'outil nécessite des efforts supplémentaires pour atteindre des bénéfices réels et durables. S'il existe des opportunités potentielles de bénéfices à utiliser des outils dans le test, il existe aussi des risques.

Bénéfices potentiels à l'utilisation d'outils :

- Réduction du travail répétitif (p.ex. exécution de tests de régression, réintroduction des mêmes données de tests, et vérification du respect de standards de codage).
- Répétabilité et cohérence accrues (p.ex. tests exécutés par un outil suivant un ordre et une fréquence précis et tests déduits des exigences).
- Evaluation objective (p.ex. mesure statiques, couverture).
- Facilité d'accès aux informations concernant les tests ou leur exécution (p.ex. statistiques et graphiques sur l'avancement des tests, le taux d'incidents et les performances).

Risques liés à l'utilisation d'outils :

- Attentes irréalistes placées dans l'outil (dont la facilité d'utilisation et la fonctionnalité).
- Sous-estimation du temps, du coût et de l'effort pour l'introduction initiale d'un outil (dont la formation et l'expertise externe).
- Sous-estimation du temps et de l'effort nécessaires pour obtenir de l'outil des bénéfices significatifs et continus de l'outil (incluant le besoin de modification du processus de tests et l'amélioration continue dans la manière d'utiliser l'outil).
- Sous-estimation de l'effort requis pour maintenir les acquis générés par l'outil.
- Confiance excessive dans l'outil (comme substitut à la conception des tests ou alors que des tests manuels seraient plus appropriés).
- Négligence du contrôle de version des éléments de test dans l'outil
- Négligence des problèmes de relation et interopérabilité entre les outils critiques, tels que des outils de gestion d'exigences, des outils de contrôle de version, des outils de gestion d'incidents, des outils de suivi de défauts et des outils de différents éditeurs
- Risque de faillite de l'éditeur d'outil, de retirer l'outil, ou de vendre l'outil à un autre éditeur
- Faible réactivité du vendeur pour le support, les mises à jour, et corrections des défauts
- Risque de suspension d'un logiciel ou projet open-source ou libre
- Imprévu, comme l'incapacité de support d'une nouvelle plate-forme

6.2.2 Considérations spéciales pour quelques types d'outil (K1)

Outils d'exécution des tests
Les outils d'exécution de tests exécutent des objets de tests utilisant des scripts de test automatisés. Ce type d'outil nécessite souvent un effort important pour obtenir des bénéfices significatifs.

Saisir des tests en enregistrant les actions d'un testeur manuel semble séduisant, mais cette approche ne peut pas être appliquée lorsque le nombre de tests automatisés est important. Un script capturé est une représentation linéaire avec des données et actions spécifiques faisant partie de chaque script. Ce type de script peut être instable lors de l'occurrence d'événements inattendus.

Une approche guidée par les données isole les entrées de tests (les données), habituellement au moyen d'un tableur et utilise un script plus générique qui peut lire les données de test et effectuer le

même test avec des données différentes. Les testeurs qui ne sont pas familiarisés avec le langage de scripts peuvent alors entrer des données de tests pour ces scripts prédéfinis.

Il y a d'autres techniques utilisées avec des techniques pilotées par les données, où au lieu des combinaisons codées en dur de données placées dans un tableur, des données sont produites utilisant des algorithmes basés sur des paramètres configurables lors de l'exécution et fournis à l'application. Par exemple, un outil peut utiliser un algorithme, qui produit un identifiant de l'utilisateur de façon aléatoire, et pour la répétabilité dans le modèle, une injection est utilisée pour en contrôler le caractère aléatoire.

Dans une approche par mots-clés, le tableur contient des mots-clés décrivant les actions à effectuer (aussi appelés mots d'action ou action words), et des données de tests. Les testeurs (même s'ils ne sont pas familiers avec le langage de script) peuvent ensuite définir des tests en utilisant les mots-clés, qui peuvent être adaptés en fonction de l'application à tester.

Une expertise technique quant au langage de script est requise pour toutes les approches (soit par les testeurs, soit par des spécialistes en automatisation des tests).

Quelle que soit la technique de script utilisée, le résultat attendu pour chaque test est archivé pour une comparaison ultérieure.

Outils d'analyse statique
Les outils d'analyse statique appliqués au code source peuvent imposer des standards de codage, mais s'ils sont appliqués à du code existant, peuvent engendrer de nombreux messages. Les messages d'avertissement n'empêchent pas le code d'être traduit en programme exécutable, mais doivent être pris en compte afin de faciliter la maintenance du code dans le futur. Une implémentation graduelle avec des filtres initiaux pour exclure certains messages constitue une approche efficace.

Outils de gestion des tests
Les outils de gestion des tests doivent s'interfacer avec d'autres outils ou des tableurs de façon à produire les informations dans le format le mieux adapté aux besoins présents de l'organisation.

6.3 *Introduire un outil dans une organisation (K1)* — *15 minutes*

Termes
Aucun terme spécifique.

Contexte
Les principes principaux liés à l'introduction d'un outil dans une organisation incluent:

- Evaluation de la maturité de l'organisation, de ses forces et de ses faiblesses et identification des possibilités d'amélioration du processus de test par le support d'outils.
- Evaluation au regard d'exigences claires et de critères objectifs.
- Une preuve de concept, à l'aide d'un outil de test pendant la phase d'évaluation pour vérifier qu'il fonctionne efficacement avec le logiciel en cours de test et dans l'infrastructure courante ou pour identifier des modifications requises de cette infrastructure pour utiliser l'outil efficacement
- Evaluation du vendeur (aspects de formation, de support et de commerce y compris) ou des fournisseurs de service de support en cas d'outils non commerciaux
- Identification des exigences internes pour le soutien et la tutelle dans l'utilisation de l'outil
- Evaluation de besoin de formation par rapport aux compétences en automatisation de tests de l'équipe de test courante
- Evaluation du rapport coût/bénéfice basé sur un cas métier concret

Introduire l'outil choisi dans une organisation commence par un projet pilote, qui a les objectifs suivants :

- Apprendre l'outil plus en profondeur.
- Voir comment l'outil s'adapte à des processus et pratiques existants, et comment ces derniers devraient évoluer.
- Décider d'une manière standard d'utiliser, de gérer, de stocker et de maintenir l'outil et le testware (p.ex. décider d'une convention de nommage pour les fichiers et les tests, créer des bibliothèques et définir la modularité des suites de tests).
- Evaluer si les bénéfices escomptés seront atteints pour un coût raisonnable.

Les facteurs de succès du déploiement d'un outil dans une organisation incluent :

- Etendre l'outil au reste de l'organisation de façon incrémentale.
- Adapter et améliorer les processus de façon à les adapter à l'utilisation de l'outil.
- Fournir de la formation et une assistance aux nouveaux utilisateurs.
- Établir des guides d'utilisation.
- Implémenter une manière de tirer des enseignements de l'utilisation de l'outil.
- Surveiller l'utilisation de l'outil et les bénéfices recueillis
- Fournir le support pour l'équipe de test pour un outil donné
- Recueillir l'expérience acquise de toutes les équipes

Références
6.2.2 Buwalda, 2001, Fewster, 1999
6.3 Fewster, 1999

7. Références

Standards

Glossaire des tests de logiciel - 2 1 F ISTQB.pdf

ISTQB Glossary of terms used in Software Testing Version 2.1

[CMMI] Chrissis, M.B., Konrad, M. and Shrum, S. (2004) CMMI, Guidelines for Process Integration *and Product Improvement*, Addison Wesley: Reading, MA
Voir Section 2.1

[IEEE Std 829-1998] IEEE Std 829™ (1998) IEEE Standard for Software Test Documentation*,*
Voir Sections 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6

[IEEE 1028] IEEE Std 1028™ (2008) IEEE Standard for Software Reviews and Audits*,*
Voir Section 3.2
[IEEE 12207] IEEE 12207/ISO/IEC 12207-2008, Software life cycle processesVoir Section 2.1

[ISO 9126] ISO/IEC 9126-1:2001, Software Engineering – Software Product Quality*,*
Voir Section 2.3

Livres

[Beizer, 1990] Beizer, B. (1990*) Software Testing Techniques* (2nd edition), Van Nostrand Reinhold: Boston
Voir Sections 1.2, 1.3, 2.3, 4.2, 4.3, 4.4, 4.6

[Black, 2001] Black, R. (2001) *Managing the Testing Process* (2nd edition), John Wiley & Sons: New York
Voir Sections 1.1, 1.2, 1.4, 1.5, 2.3, 2.4, 5.1, 5.2, 5.3, 5.5, 5.6

[Buwalda, 2001] Buwalda, H. et al. (2001) *Integrated Test Design and Automation*, Addison Wesley: Reading, MA
Voir Section 6.2

[Copeland, 2004] Copeland, L. (2004) *A Practitioner's Guide to Software Test Design*, Artech House: Norwood, MA
Voir Sections 2.2, 2.3, 4.2, 4.3, 4.4, 4.6

[Craig, 2002] Craig, Rick D. and Jaskiel, Stefan P. (2002) *Systematic Software Testing*, Artech House: Norwood, MAVoir Sections 1.4.5, 2.1.3, 2.4, 4.1, 5.2.5, 5.3, 5.4

[Fewster, 1999] Fewster, M. and Graham, D. (1999) *Software Test Automation*, Addison Wesley: Reading, MA
Voir Sections 6.2, 6.3

[Gilb, 1993]: Gilb, Tom and Graham, Dorothy (1993) *Software Inspection*, Addison Wesley: Reading, MA
Voir Sections 3.2.2, 3.2.4

[Hetzel, 1988] Hetzel, W. (1988) *Complete Guide to Software Testing*, QED: Wellesley, MA
Voir Sections 1.3, 1.4, 1.5, 2.1, 2.2, 2.3, 2.4, 4.1, 5.1, 5.3

[Kaner, 2002] Kaner, C., Bach, J. and Pettticord, B. (2002) *Lessons Learned in Software Testing*, John Wiley & Sons: New York
Voir Sections 1.1, 4.5, 5.2

[Myers 1979] Myers, Glenford J. (1979) *The Art of Software Testing*, John Wiley & Sons: New York
Voir Sections 1.2, 1.3, 2.2, 4.3

[van Veenendaal, 2004] van Veenendaal, E. (ed.) (2004) *The Testing Practitioner* (Chapters 6, 8, 10), UTN Publishers: The Netherlands
Voir Sections 3.2, 3.3

8. Annexe A – Informations sur le syllabus

Historique de ce document

Ce document a été préparé entre 2004 et 2007 par un groupe de travail composé de membres de l'International Software Testing Qualifications Board (ISTQB). Il a été initialement révisé par une commission de révision, puis par des représentants de la communauté internationale des testeurs de logiciels. Les règles utilisées pour la conception de ce document sont décrites en Annexe C.

Ce document est le syllabus pour le certificat niveau Fondation en Tests de Logiciels, le premier niveau du schéma de qualification international approuvé par l'ISTQB (www.istqb.org).

Objectifs de la qualification Certificat Fondation

- Acquérir une reconnaissance des tests comme une spécialisation professionnelle et essentielle de l'ingénierie logicielle.
- Fournir un cadre standard pour le développement des carrières des testeurs.
- Permettre une reconnaissance des testeurs qualifiés professionnellement par les employeurs, clients et leurs pairs, et accroître le profil des testeurs.
- Promouvoir de bonnes pratiques, régulières, dans les disciplines de l'ingénierie logicielle.
- Identifier des thèmes de tests qui sont pertinents et valables pour l'industrie.
- Permettre aux fournisseurs de logiciels d'embaucher des testeurs certifiés et gagner ainsi un avantage commercial sur la compétition en annonçant leur politique de recrutement.
- Fournir une opportunité aux testeurs et aux personnes intéressées par les tests d'acquérir une qualification reconnue internationalement sur le sujet.

Objectifs de la qualification internationale (adapté de la réunion ISTQB de Novembre 2001 à Sollentuna)

- Permettre de comparer les compétences des tests de pays différents
- Permettre aux testeurs de traverser les frontières plus facilement.
- Faciliter une compréhension commune des aspects de tests dans les projets multi-nationaux et/ou internationaux
- Augmenter le nombre de testeurs qualifiés dans le monde
- Avoir plus d'impact ou de valeur en tant qu'initiative internationale qu'une approche nationale spécifique
- Développer une compréhension et une connaissance commune internationale sur les tests via un syllabus et une terminologie, et accroître le niveau de connaissance des tests de tous les participants
- Promouvoir les tests en tant que profession dans plus de pays
- Permettre aux testeurs d'acquérir une qualification reconnue dans leur langue maternelle
- Faciliter le partage de connaissances et de ressources entre les pays.
- Fournir une reconnaissance internationale des testeurs et de cette qualification par une participation de nombreux pays

Conditions d'entrée pour cette certification

Le critère d'entrée pour passer l'examen du certificat de Testeur de Logiciels Certifié CFTL niveau Fondation (ISTQB Foundation Certificate in Software Testing) est que le candidat démontre un intérêt dans les tests logiciels. Cependant il est fortement recommandé que le candidat :

- Possède une connaissance minimale soit du développement de logiciels, soit des tests de logiciels, tels qu'un minimum de six mois d'expérience en tests systèmes ou tests d'acceptation utilisateurs, ou en tant que développeur de logiciels
- Suive un cours accrédité CFTL ou au niveau des standards ISTQB (par un des comités nationaux reconnus par l'ISTQB)

Contexte et histoire du Certificat Fondation en Tests de Logiciels

La certification indépendante des testeurs de logiciels a commencé en Grande Bretagne avec le « Information Systems Examination Board (ISEB) » du British Computer Society, quand un Comité des Tests Logiciels a été mis en place en 1998 (www.bcs.org.uk/iseb). En 2002, ASQF en Allemagne commença a supporter un schéma de qualification allemand (www.asqf.de). Ce syllabus est basé sur les syllabus ISEB et ASQF; il inclut une réorganisation et une mise à jour du contenu, ainsi que du contenu supplémentaire, et une emphase est mise sur les points qui fourniront une aide pratique aux testeurs.

Un Certificat en Tests de Logiciels niveau Fondation existant (p.ex. de l'ISEB, de l'ASQF ou d'un comité national reconnu par l'ISTQB) décerné avant la publication de ce Certificat International, sera considéré comme équivalent au Certificat International. Le Certificat niveau Fondation n'expire pas et ne doit pas être renouvelé. La date d'attribution est indiquée sur le Certificat.

Dans chaque pays participant, les aspects locaux sont contrôlés par un Comité National des Tests Logiciels reconnu par l'ISTQB. Les devoirs des comités nationaux sont spécifiés par l'ISTQB, mais implémentés dans chaque pays. Les devoirs des comités nationaux incluent l'accréditation des organismes de formation et la tenue des examens.

9. Annexe B – Objectifs de connaissance/Niveaux de connaissance

Les objectifs de connaissance suivants sont définis comme s'appliquant à ce syllabus. Chaque aspect de ce syllabus sera examiné en fonction de son objectif de connaissance.

Niveau 1: Se souvenir (K1)

Le candidat reconnaîtra, se rappellera et se souviendra des termes ou des concepts.

Mots clés: Se souvenir, retrouver, reconnaître, mémoriser, savoir

Exemple
Peut reconnaître la définition de "défaillance" comme :

- "non livraison d'un service à l'utilisateur final ou tout autre personne impliquée" ou
- "Déviation constatée du composant ou système de la fourniture, service ou résultat attendu".

Niveau 2: comprendre (K2)

Le candidat peut sélectionner les raisons ou explications pour des affirmations liées au sujet traité, et peut résumer, différencier, classer et donner des exemples sur les concepts de test

Mots clés: Résumer, classer, comparer, relier, opposer, donner des exemples, interpréter, traduire, représenter, déduire, conclure, construire des modèles, classer, généraliser, faire abstraction

Exemples
Peut expliquer les raisons pour lesquelles les tests doivent être exécutés aussi tôt que possible :

- Pour trouver des défauts quand il est intéressant de les supprimer.
- Pour trouver d'abord les défauts les plus importants.

Peut expliquer les similitudes et les différences entre les tests d'intégration et les tests système :

- Similitudes : tester plus d'un composant, et tester des aspects non-fonctionnels.

Différences : les tests d'intégration se concentrent sur les interfaces et les interactions, les tests système se focalisent sur les aspects de systèmes entiers, tels le processus de bout en bout.

Niveau 3: Appliquer (K3)

Le candidat peut sélectionner l'application correcte d'un concept ou d'une technique et l'appliquer à un contexte donné.

Mots clés: Implémenter, exécuter, utiliser, suivre une procédure, appliquer une procédure

Exemple

- Peut identifier les valeurs limites pour des partitions valides et invalides
- Peut sélectionner les cas de test nécessaires à la couverture de toutes les transitions pour un diagramme d'état donné

Niveau 4: Analyser (K4)

Le candidat peut décomposer l'information relative à la procédure ou à la technique en différentes parties pour une meilleure compréhension et peut faire la différence entre des faits et les conclusions qui en sont tirées. Une application classique est de proposer, après l'analyse d'un document, d'un logiciel ou de la situation d'un projet, les actions appropriées pour résoudre un problème ou une tâche.

Mots clés: Analyser, différencier, sélectionner, structurer, cibler, attribuer, déconstruire, organiser, trouver la cohérence, intégrer, mettre en évidence, parcourir, distinguer, cibler, discriminer

Exemple

- Analyser les risques produit et proposer des activités correctives et préventives pour les atténuer
- Décrire quelles parties d'un rapport d'incident sont factuelles et quelles parties ont été déduites des résultats.

Références

(pour les niveaux cognitifs des objectifs d'apprentissage)
Anderson, L. W. and Krathwohl, D. R. (eds) (2001) *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, Allyn & Bacon

10. Annexe C – Règles appliquées au syllabus ISTQB niveau Fondation

Les règles listées ci-après ont été utilisées dans le développement et la revue de ce syllabus. (une référence “TAG” est précisée après chaque règle pour référencer celle-ci.)

Règles générales

SG1. le syllabus doit être compréhensible et absorbable par des personnes ayant de 0 à 6 mois (ou plus) d’expérience en tests. (6-MOIS)

SG2. Le syllabus doit être pratique plutôt que théorique. (PRATIQUE)

SG3. Le syllabus doit être clair et non ambigu pour son lectorat prévu. (CLAIR)

SG4. Le syllabus doit être compréhensible par des personnes de pays différents, et facilement traduisible en différentes langues. (TRADUISIBLE)

SG5. Le syllabus doit utiliser la tournure américaine de l’anglais. (ANGLAIS-AMERICAIN)

Contenu actualisé

SC1. Le syllabus doit inclure des concepts de tests récents et doit refléter les meilleures pratiques actuelles en tests de logiciels là où elles sont généralement acceptées. Le syllabus est sujet à revue tous les trois ou cinq ans. (RECENT)

SC2. Le syllabus doit minimiser les aspects liés au temps, telles que les conditions du marché, pour lui permettre d’être valide pour une période de trois à cinq ans. (DUREE-DE-VIE).

Objectifs de connaissance

LO1. Les objectifs de connaissance doivent distinguer entre des aspects à reconnaitre/mémoriser (niveau cognitive K1), les aspects que le candidat doit comprendre conceptuellement (K2), des aspects que le candidat doit être capable d’utiliser et de mettre en pratique (K3) et des aspects que le candidat doit être capable d’utiliser pour analyser un document, du logiciel, la situation d’un projet dans son contexte (K4). (NIVEAU DE CONNAISSANCE)

LO2. La description du contenu doit être consistante avec les objectifs de connaissance. (LO-CONSISTANT)

LO3. Pour illustrer les objectifs de connaissance, des exemples de questions d’examen pour chacune des sections majeures seront fournies avec le syllabus. (LO-EXAMEN)

Structure générale

ST1. La structure du syllabus doit être claire et permettre une référence croisée à partir de (et vers) chaque partie, depuis les questions d’examen et depuis d’autres documents pertinents. (CROSS-REF)

ST2. Le recouvrement entre les sections du syllabus doit être minimisé. (RECOUVREMENT)

ST3. Chaque section du syllabus doit avoir la même structure. (CONSISTANCE STRUCTURELLE)

ST4. Le syllabus doit contenir version, date d’émission et numéro de page sur chaque page. (VERSION)
ST5. Le syllabus doit inclure une information sur le temps à passer sur chaque section (pour refléter l’importance relative de chaque aspect). (TEMPS PASSE)

Références

SR1. Sources et Références seront fournis pour les concepts du syllabus pour aider les fournisseurs de formations à trouver plus d’information sur le sujet. (REFS)

SR2. Si la source n'est pas clairement identifiée et/ou claire, plus de détails doit être fourni dans le syllabus. Par exemple, les définitions sont dans le Glossaire, donc seuls les termes sont listés dans le syllabus. (NON-REF DETAIL)

Sources d'informations

Les termes utilisés dans le syllabus sont ceux définis dans le Glossaire CFTL/ISTQB des termes utilisés en tests de logiciels. Une version de ce glossaire est disponible sur les sites de l'ISTQB et du CFTL.

Une liste de livres recommandés sur les tests de logiciels est aussi fournie en parallèle à ce syllabus. La liste principale de livres est incluse dans la section référence.

11. Annexe D – Note pour les organismes de formation

Chaque titre de sujet principal de ce syllabus est associé à une durée en minutes. L'objectif de cette instruction est de donner des indications sur la proportion de temps à allouer à chaque section dans un cours accrédité, et en même temps fournir une durée minimum approximative pour enseigner chaque section. Les fournisseurs de formations peuvent passer plus de temps qu'indiqué et les candidats peuvent prendre plus de temps en lectures et en recherche. Un curriculum de cours ne doit pas spécifiquement suivre le même ordre que le syllabus.

Le syllabus contient des références à des normes établies, qui doivent être utilisées dans la préparation du matériel de formation. Chaque norme utilisée doit être dans la version mentionnée dans ce syllabus. D'autres publications, modèles ou standards non référencés dans ce syllabus peuvent aussi être utilisés et référencés, mais ne seront pas examinés.

Les domaines spécifiques de ce syllabus nécessitant des exercices pratiques sont les suivantes :

4.3 Techniques basées sur les spécifications ou techniques boîte noire

Des exercices pratiques (exercices courts) devraient couvrir les quatre techniques suivantes : classe d'équivalence, analyse des valeurs limites, tests par tables de décisions et tests de transition d'états. Les enseignements et les exercices se référant à ces techniques devraient être basés sur les références fournies pour chaque technique.

4.4 Tests bases sur les structures ou Boîte blanche

Des exercices pratiques (exercices courts) devraient être inclus afin de s'assurer ou non qu'un ensemble de tests atteint 100% de couverture des instructions et 100% de couverture des décisions et aussi pour concevoir des cas de test pour des flux de contrôle.

5.6 Gestion des incidents

Des exercices pratiques devraient inclure la rédaction et le traitement d'un rapport d'incident.

12. Annexe E – Changements intégrés dans le Syllabus 2010

1. Changements des objectifs de connaissance (LO) pour y inclure des clarifications
    a. Vocabulaire changé pour les LOs suivants (Le contenu et le niveau des LOs demeure inchangé): LO-1.2.2,LO-1.4.1, LO-2.1.1, LO-2.1.3, LO-4.6.1, LO-6.3.2
    b. K4 a été ajouté. Raison: Certaines exigences (LO-4.4.4 and LO-5.6.2) sont déjà écrites au niveau K4, et les questions du LO-4.6.1 sont plus faciles à écrire et à examiner au niveau K4.
    c. LO-1.1.5 a été réécrit et mis au niveau K2. Une comparaison des termes relatifs aux defaults peut être attendue.
    d. LO-1.2.3 Expliquer les différences entre les deux activités “déboguer” et “tester” devient un nouvel OC. Le contenu du syllabus le couvrait déjà.
    e. LO-3.1.3 Comparaison de ces techniques couverte
    f. LO-3.1.4 Supprimé car redondant en partie avec le LO-3.1.3.
    g. LO-3.2.1 Amélioration du contenu.
    h. LO-3.3.2 Mis à jour en K2 pour compatibilité avec le LO-3.1.2
    i. LO-6.1.2 Supprimé car c’est une partie du LO-6.1.3, réécrite due à une utilisation non appropriée du niveau K2
2. Usage de l’approche de tests selon la définition du glossaire. Le terme “Stratégie de tests” ne sera pas requis comme devant être appris.
3. Le chapitre 1.4 contient maintenant le concept de traçabilité entre les bases de tests et les cas de test.
4. Le chapitre 2.x contient maintenant les objets de test et les bases de test.
5. Re-tester remplace maintenant “Tests de confirmation” en liaison avec le syllabus.
6. Les aspects “Qualité des données et tests » ont été ajouté en plusieurs endroits du syllabus: Qualité des données et risques dans les chapitres 2.2, 5.5, 6.1.8
7. Les critères d’entrée du chapitre 5.2.3 ont été ajoutés comme un nouveau sous-chapitre. Raison: La consistance du contenu des Critères de sortie (-> critère d’entrée ajouté au LO-5.2.9).
8. Utilisation des termes “Stratégie de tests” et “Approche de tests” conformément à leur définition dans le glossaire.
9. Le chapitre 6.1 a été raccourci car les descriptions des outils étaient trop exhaustives pour une leçon de 45 minutes.
10. La norme IEEE Std 829:2008 est parue. Cette version du syllabus ne prend pas encore en compte cette nouvelle édition. La section 5.2 fait référence au document Plan de Test. Le contenu du “Plan de Test Maitre” est couvert par le concept qui dit que le Plan de test couvre différents niveaux de planification de tests: Les plans de test s’appliquent à tous les niveaux de tests comme au niveau projet qui lui couvrira plusieurs niveaux de tests. Ce dernier est appelé “Plan de Test Maitre” dans ce syllabus et dans le glossaire de l’ISTQB.
11. Le code d’éthique est déplacé du syllabus Avancé vers le syllabus Fondation.

13. Index