Testeur Certifié

Syllabus Niveau Fondation

Version 2005FR

International Software Testing Qualifications Board

Comité Français des Tests Logiciels

Historique des modifications

Version	Date	Remarques
ISEB V2.0	25-Feb-1999	ISEB Software Testing Foundation Syllabus V2.0 25 February 1999
ASQF V2.2	Juillet-2003	ASQF Syllabus Foundation Level Version 2.2 “Lehrplan „Grundlagen des Softwaretestens“
ISTQB 2005	01-Juillet-2005	Certified Tester Foundation Level
CFTL 2005FR	01-Juillet-2005	Testeur Certifié Niveau Fondation

Table des Matières

Remerciements

Ce document a été produit par le groupe de travail niveau Fondation de l'International Software Testing Qualifications Board: Thomas Müller (chair), Rex Black, Sigrid Eldh, Dorothy Graham, Klaus Olsen, Maaret Pyhäjärvi, Geoff Thompson et Erik van Veendendal. Le noyau de l'équipe souhaite remercier les équipes de revue et tous les comités nationaux pour leurs suggestions apportées aux présent syllabus.

Des remerciements particuliers vont à (Autriche) Anastasios Kyriakopoulos, (Danemark) Klaus Olsen, Christine Rosenbeck-Larsen, (Allemagne) Matthias Daigl, Uwe Hehn, Tilo Linz, Horst Pohlmann, Ina Schieferdecker, Sabine Uhde, Stephanie Ulrich, (Inde) Vipul Kocher, (Israël) Shmuel Knishinsky, Ester Zabar, (Suède) Anders Claesson, Mattias Nordin, Ingvar Nordström, Stefan Ohlsson, Kennet Osbjer, Ingela Skytte, Klaus Zeuge, (Suisse) Armin Born, Sandra Harries, Silvio Moser, Reto Müller, Joerg Pietzsch, (UK) Aran Ebbett, Isabel Evans, Julie Gardiner, Andrew Goslin, Brian Hambling, James Lyndsay, Helen Moore, Peter Morgan, Trevor Newton, Angelina Samaroo, Shane Saunders, Mike Smith, Richard Taylor, Neil Thompson, Pete Williams, (USA) Dale Perry.

Traduction française : Bernard Homès (France), Comité Français des Tests Logiciels, www.cftl.net

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 (ci-après nommé ISTQB) et le Comité Français des Tests Logiciels (ci-après noté 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 background 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 dans des rôles de testeurs, analystes de tests, ingénieurs de tests, consultants en tests, gestionnaires de tests, testeurs en phase d'acceptation et développeurs de logiciels. Cette qualification au 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, directeurs des TI 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 formation /niveau de connaissance

Les niveaux de connaissance sont fournis pour chaque section de ce syllabus :

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

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 choix multiple.

Les examens peuvent faire partie d'une formation accréditée ou être passés indépendamment (p.ex. dans un centre d'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 d'apprentissage 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 logiciel (K2)	135 minutes

indique que le Chapitre 2 a un objectif de connaissance K1 (supposé quand un niveau supérieur spécifié) et K2 (mais pas K3), et qu'une durée de 135 minutes au moins 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)
- 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)
- Distinguer entre la cause initiale du défaut et ses effets. (K2)
- Donner des raisons pour lesquelles les tests sont nécessaires en donnant des exemples. (K2)
- Décrire pourquoi les tests font partie de l'assurance qualité et donner des exemples sur comment les tests contribuent à une qualité accrue. (K2)
- Définir les termes faute, défaut, défaillance et les termes associés erreur et bug. (K1)

1.2 Que sont les tests ? (K2)
- Rappeler les objectifs habituels des tests. (K1)
- Décrire les objectifs des tests dans le développement logiciel, la maintenance et les opérations comme moyen de trouver des défauts, fournir la confiance et l'information, et prévenir les défauts. (K2)

1.3 Principes généraux des tests (K2)
- Expliquer les principes fondamentaux des tests. (K2)

1.4 Processus fondamentaux des tests (K1)
- Rappeler les activités fondamentales des tests de la planification aux activités de clôture des tests et les tâches principales de chaque activité de test. (K1)

1.5 Psychologie des tests (K2)
- Rappeler que le succès des tests est influencé par des facteurs psychologiques (K1):
  - Objectifs clairs
  - Equilibre entre des tests personnels et des tests indépendants ;
  - Reconnaissance des communications et feed-back courtois concernant les défauts.
- Comparer la mentalité d'un testeur et 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, software, 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 (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 et maintenabilité). 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.

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 aspects techniques, de produit commercial et de risques projet, ainsi que les contraintes telles le temps et le budget. (Les risques sont développés dans le 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 tests, pour l'étape de développement suivante ou la livraison aux clients.

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

Termes
Code, déboguage, développement (de logiciels), exigences, revues, base de test, cas de tests, test, objectifs des tests.

Background
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, des activités telles que la planification et le contrôle, la sélection des conditions de test, la conception des cas de tests et la vérification des résultats, l'évaluation des critères de complétude, l'information sur les processus de test et sur le système en cours de tests, ainsi que la clôture définitive (p.ex. à la fin d'une phase de tests). 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 et 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 dans un niveau de qualité et fournir de l'information;
- Prévenir des défauts.

Le processus de réflexion pour concevoir des tests tôt dans le cycle de vie (vérifier les bases de tests via la conception des tests) peut aider à prévenir l'introduction de défauts dans le code. Les revues de documents (p.ex. exigences) peuvent aussi 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 possibles 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 peuvent montrer que des défaillances sont causées par des défauts. Le débogage est une activité de développement qui identifie les causes d'un défaut, répare le code et vérifie que le défaut a été correctement corrigé. Les tests de confirmation ultérieurs effectués par un testeur assurent que la correction résout effectivement à la défaillance. La responsabilité de chaque activité est très différente : les testeurs testent, les développeurs déboguent.

Les processus des tests et ses activités sont expliqués en Section 1.4.

1.3 *Principes généraux des tests (K2)* — *35 minutes*

Termes
Tests exhaustifs.

Principes
Un nombre de principes de tests 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 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 les risques et les priorités pour focaliser les efforts de tests.

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

Principe 4 – Regroupement des défauts
Un petit nombre de modules contient 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 tests fondamentaux (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 tests, plan de tests, stratégie de test, rapport de synthèse de tests, testware.

Background
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.

1.4.1 Planification et contrôle des tests (K1)
La planification des tests est l'activité qui vérifie les mission des tests, définissant les objectifs de tests et spécifiant les activités de test de façon à 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, ceux-ci devraient être suivis pendant toute la durée du projet. La planification des tests prend en compte le feed-back des activités de contrôle et de suivi.

La planification des tests se compose des tâches majeures suivantes:

- Déterminer la portée et les risques, et identifier les objectifs des tests.
- Déterminer les approches de tests (techniques, objets de tests, couverture, identification et interfaces avec les équipes concernées par les tests, outils de tests).
- Déterminer les ressources de test requises (p.ex. personnel, environnement de tests, PCs).
- Implémenter la politique de test et/ou la stratégie de tests.
- Planifier les tâches d'analyse et de conception des tests.
- Planifier l'implémentation, l'exécution et l'évaluation des tests.
- Déterminer les critères de sortie des tests.

Le contrôle des tests se compose des tâches majeures suivantes:

- Mesurer et analyser les résultats;
- Mesurer et documenter l'avancement, la couverture des tests et les critères de sortie;
- Initier des actions correctives;
- Décider.

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 tests et des conceptions de tests tangibles.

L'analyse et la conception des tests se composent des tâches majeures suivantes:

- Réviser les bases de test (tels que les exigences, l'architecture, la conception et les interfaces).
- Identifier les conditions de tests ou les exigences de tests et les données de tests requises basées sur l'analyse des articles de tests, les spécifications, le comportement et la structure.
- Concevoir les tests.
- Evaluer la testabilité des exigences et du système.
- Concevoir l'environnement de test (banc de tests) et identifier les infrastructures et outils nécessaires.

1.4.3 Implémentation et exécution des tests (K1)

L'implémentation des tests et l'exécution des tests sont les activités où les conditions de tests sont transformées en cas de tests et en testware, et où l'environnement de tests est mis en place.

L'implémentation et l'exécution des tests se composent des tâches majeures suivantes:

- Développer et prioriser les cas de tests, créer les données de tests, écrire les procédures de tests et, en option, préparer les harnais de tests et écrire les scripts de tests automatisés.
- Créer des suites de tests à partir des cas de tests pour une exécution rentable des tests.
- Vérifier que les environnements de tests ont été mis en place correctement.
- Exécuter les cas de tests 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 tests, outils de tests 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 tests 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).

Kommentar [BHS1]: Vérifier la traduction aussi dans le glossaire des termes.

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 responsables.

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 testware, les faits et les valeurs. Par exemple, quand un système logiciel est mis en production, un projet de tests 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 création de demande d'évolution pour ceux restant ouverts, et documentation de l'acceptation du système.
- Finaliser et archiver les testware, environnement de tests et infrastructure de tests pour une utilisation future.
- Fournir les testware à l'organisation en charge de la maintenance.
- Analyser les leçons apprises pour les versions et projets futurs, et l'amélioration de la maturité des tests.

1.5 *La psychologie des tests (K2)* — *35 minutes*

Termes
Test indépendant.

Background
La tournure d'esprit utilisée 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:

- 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).
- 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 sa recherche 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 leur 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.

Références

1.1.5 Black, 2001, Kaner, 2002
1.2 Beizer, 1990, Black, 2001, Myers, 1979
1.3 Beizer, 1990, Hetzel, 1998, Myers, 1979
1.4 Hetzel, 1998
1.4.5 Black, 2001, Craig, 2002
1.5 Black, 2001, Hetzel, 1998

2. Tester pendant le cycle de vie logiciel (K2) — 135 minutes

Objectifs de connaissance pour tester pendant le cycle de vie logiciel
Les objectifs identifient ce que vous serez capable de faire suite à l'achèvement de chaque module.

2.1 Les modèles de développement logiciels (K2)

- Comprendre les relations entre le développement, les activités de tests 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).
- Reconnaître le fait que les modèles de développement logiciel doivent être adaptés aux contextes du projet et aux caractéristiques du produit (K1).
- Rappeler les raisons pour les différents niveaux de tests, et les caractéristiques de bons tests dans chacun des modèles de cycle de vie. (K1)

2.2 Niveaux de tests (K2)

- Comparer les différents niveaux de tests: objectifs principaux, types d'objet à tester typiques, types de tests typiques (p.ex. fonctionnel ou structurel) 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)

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

2.4 Tests de maintenance (K2)

- 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)
- Identifier les raisons pour les tests de maintenance (modification, migration et abandon). (K1)
- Décrire le rôle des tests de régression et de l'analyse d'impact en maintenance. (K2)

2.1 *Modèles de développement logiciels (K2)* — *20 minutes*

Termes
Commercial off the shelf (COTS), modèle de développement incrémental niveaux de tests, validation, vérification, modèle en V.

Background
Les tests n'existent pas de façon isolée; les activités de test sont liées aux activités de développements 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 en moins, en plus ou différents par rapport aux niveaux de tests, dépendant du projet et du produit logiciel. 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 système.

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 dans 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 avancée des tests) peut être effectuée pendant le développement des livrables logiciels.

2.1.2 Modèle de développement itératif (K2)

Le mode de développement itératif est le processus d'établissement d'exigences, de conception, de construction et de tests d'un système, exécuté comme une série de petits développements. Exemples : prototypage, développement rapide d'applications (RAD), Rational Unified Process (RUP) et les modèles de développement agiles. L'incrément produit par une itération peut être testé à plusieurs niveaux intégrés dans son développement. Un incrément, ajouté à ceux développés préalablement, forme un système partiel en croissance, qui devrait aussi 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 caractéristiques déterminent de bons tests:

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

2.2 *Niveaux de tests (K2)* — *60 minutes*

Termes
Tests alpha, 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 d'acceptation opérationnels, regulation acceptance testing, tests basés sur les exigences, tests de robustesse, bouchon, tests système, test-driven development, environnement de test, tests d'acceptation utilisateur.

Background
Pour chaque niveau de test, 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 défaillances typiques à trouver, les exigences en harnais de tests et en support d'outils, ainsi que les approches et responsabilités spécifiques.

2.2.1 Tests de composants (K2)
Les tests de composants 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 isolément par rapport au reste du système, dépendant 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 caractéristiques fonctionnelles et 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 les spécifications des composants, la conception du logiciel ou le modèle de données.

Typiquement les tests de composants ont lieu avec accès au code en cours de tests et avec le support de l'environnement de développement, tel que la structure de tests unitaire ou les outils de déboguage , et, en pratique, implique généralement le programmeur ayant écrit le code. Les défauts sont typiquement corrigés dès qu'ils sont trouvés, sans enregistrer des incidents formels.

Une approche des tests de composants est de préparer et automatiser des cas de tests avant le codage. Ceci est appelé l'approche test first ou le développement dirigé par les tests. Cette approche est hautement itérative et est basée sur des cycles de développement de cas de tests, puis la construction et l'intégration de petits bouts de code et l'exécution des tests de composants jusqu'à leur réussite.

2.2.2 Tests d'intégration (K2)
Les tests d'intégration testent les interfaces entre les composants, les interactions entre différentes parties d'un système, par exemple le système d'opération, 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 effectué après les tests de composants;
- Tests d'intégration système testant l'intégration entre les différents système 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, donc des modifications peuvent être déstabilisantes. Les processus commerciaux implémentés comme workflow 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 et de les associer à un composant ou système spécifique, ce qui peut entraîner un accroissement des risques.

Des stratégies systématiques d'intégration peuvent être basées sur l'architecture des systèmes (tel que top-down ou bottom-up), les tâches fonctionnelles, les séquences d'exécution de transactions, ou d'autres aspects du système ou composant. Afin de réduire les risques de découverte tardive d'anomalies, l'intégration devrait normalement être incrémentale plutôt qu'être effectuée en une fois ("big bang").

Les tests de caractéristiques spécifiques non-fonctionnelles (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. Par exemple, si le module A est intégré avec le module B, les testeurs sont intéressés à tester la communication entre les modules, non pas les fonctionnalités de l'un ou l'autre des modules. 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 la conception des composants ou systèmes, ils peuvent être conçus dans l'ordre requis pour des tests efficaces et rentables.

2.2.3 Tests système (K2)

Les tests systèmes traitent le comportement des systèmes/produits complets tels que définis dans la portée du projet ou du programme de développement.

Dans les tests système, l'environnement de test devrait, autant que possible, correspondre à la cible finale ou à un environnement de production de façon à minimiser les risques de défaillances spécifiques à l'environnement et non trouvés pendant les tests.

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

Les tests système devraient examiner à la fois les exigences fonctionnelles et les non-fonctionnelles du système. Les exigences peuvent exister sous forme de textes et/ou de modèles. Les testeurs ont aussi besoin de traiter avec des exigences non-documentées ou incomplètes. Les tests systèmes d'exigences fonctionnelles commencent par l'utilisation des techniques les plus appropriées de spécification des tests (boîte noire) pour les aspects à tester. 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 l'élément structurel, tel que 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)

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

Les objectifs des tests d'acceptation sont de s'assurer du système, de parties du système ou de caractéristiques non-fonctionnelles spécifiques du système. La recherche d'anomalies n'est pas l'objectif principal des tests d'acceptation. Les tests d'acceptation peuvent évaluer le niveau de préparation du système préalablement à son déploiement et à son utilisation, 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 d'un système.

Les tests d'acceptation peuvent survenir comme plus qu'un seul niveau de tests, par exemple:

- Un produit logiciel COTS peut subir des tests d'acceptation quand il est installé ou intégré.
- Les tests d'acceptation de l'utilisabilité d'un composant peuvent être effectués pendant les tests de ce composant.
- Les tests d'acceptation d'une amélioration fonctionnelle nouvelle peuvent être exécutés avant les tests système.

Les formes typiques de tests d'acceptation incluent :

Tests d'acceptation utilisateur
Typiquement 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, inclut:

- Tests des backups et restaurations;
- Reprise après sinistre;
- Gestion des utilisateurs;
- Tâches de maintenance;
- 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 marché, ou COTS, souhaitent souvent obtenir du feed-back des clients potentiels ou existants sur leur marché, avant de mettre le produit logiciel sur le marché. Les tests Alpha sont exécutés sur le site de l'organisation effectuant le développement. Les tests béta ou tests sur le terrain sont exécutés par des personnes sur leurs sites propres. Les deux sont exécutés par des clients potentiels et non par des développeurs du produit.

Les organisations peuvent aussi utiliser d'autres termes, telsque test d'acceptation usine (factory acceptance testing) et tests d'acceptation sur site pour des systèmes qui sont testés avant et après être déplacés sur le site client.

2.3 *Types de tests: les cibles de tests (K2)* — *40 minutes*

Termes
Automatisation, tests boîte noire, couverture du code, tests de confirmation, tests fonctionnels, tests d'interopérabilité, tests de charge, tests de maintenabilité, tests de performances, tests de portabilité, tests de régression, tests de fiabilité, tests de sécurité, tests basés sur les spécifications, tests de stress, tests structurels, suite de tests, tests d'utilisabilité, tests boîte blanche.

Background
Un groupe d'activités de tests peut être axé sur la vérification du système logiciel (ou d'une partie du système) pour des raisons spécifiques ou cibles de tests.

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 le fiabilité ou l'utilisabilité, la structure ou l'architecture du logiciel ou du système; ou lié aux changements, p.ex. confirmer que des anomalies ont été corrigées (tests de confirmation) et pour rechercher la présence de modifications non-intentionnelles (tests de régression).

Un modèle du logiciel peut être développé et/ou utilisé dans des tests structurels et fonctionnels. Par exemple, dans les tests fonctionnels un modèle du processus de flux, un modèle de transition d'états ou des spécifications en langage naturel; et pour les tests structurels un modèle du flux de contrôle ou de la structure des menus.

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 exigences de spécifications, cas d'utilisation, ou spécifications fonctionnelles, ou peuvent être 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 peuvent être exécutés à tous les niveaux de tests (p.ex. les tests de composants peuvent être basés sur les spécifications d'un composant).

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

Un type de tests fonctionnels, les tests de sécurité, examinent les fonctions (p.ex. pare-feu) liées à la détection de menaces, comme des virus, provenant de tiers malveillants.

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

Les tests non-fonctionnels peuvent être effectués à tous les niveaux de test. Le terme de tests non-fonctionnels décrit les tests requis pour mesurer les caractéristiques des systèmes et logiciels qui peuvent être quantifiés sur une échelle variable, telles que les temps de réponse pour les tests de performances. Ces tests peuvent référencés par un modèle qualité tel que celui défini dans l'ISO9126 'Ingénierie Logicielle – Qualité des Produits Logiciels' ('Software Engineering – Software Product Quality').

2.3.3 Test 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 exercée par une suite de tests, exprimée en pourcentage des éléments couverts. Si la couverture n'est pas de 100%, alors d'autres tests peuvent être conçus pour tester les éléments qui ont été manqués 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, 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 tel que la hiérarchie d'appels.

Les approches structurelles de tests peuvent être aussi appliquées aux systèmes, à l'intégration système ou aux niveaux de tests d'acceptation système (p.ex. pour les modèles de gestion 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 retesté pour confirmer que le défaut original a été correctement ôté. Ceci est appelé test de confirmation. Le déboguage (correction de défaut) est une activité de développement, pas une activité de tests.

Le test de régression est la répétition des tests sur un programme déjà testé, après des modifications, pour mettre à jour tout défaut introduit ou découvert en résultat de 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. L'étendue des tests de régression est basée sur le risque de ne pas trouver d'anomalie dans un logiciel fonctionnant auparavant.

Les tests devraient être répétables s'ils sont amenés à pouvoir être utilisés pour des tests de confirmation et assister 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 régression sont exécutées de nombreuses fois et généralement évoluent 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, migration, modifications, suppression.

Background
Une fois déployé, un système logiciel est souvent en service pendant des années, voire des dizaines d'années. Pendant ce temps, le système et son environnement sont fréquemment corrigés, modifiés ou étendus. 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 modifications évolutives (p.ex. basées sur les versions), correctives et d'urgence, et les changements d'environnement, tels que des modernisations planifiées de système opératoire ou des évolutions de bases de données, ou des corrections (patch) de vulnérabilités affichées ou découvertes sur le système opératoire.

Les tests de maintenance lors de migrations (p.ex. d'une plate-forme à une autre) devraient inclure des tests opérationnels du nouvel environnement, ainsi que du logiciel modifié.

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

En plus des tests de ce qui a changé, les tests de maintenance incluent des tests de régression approfondis sur les parties du système qui n'ont pas changé. L'étendue 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 tous les types de tests.

La détermination de comment un système existant est affecté par des modifications est appelé analyse d'impact, et est utilisé pour décider de la quantité de tests de régression devant être fait.

Les tests de maintenance peuvent être difficiles si les spécifications sont périmées ou manquantes.

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 Revues et processus de test (K2)
- Reconnaître les livrables qui peuvent être examinés par les différentes techniques statiques. (K1)
- Décrire l'importance et la valeur de l'utilisation de techniques statiques dans l'évaluation de livrables logiciels. (K2)
- Expliquer les différences entre les techniques statiques et dynamiques. (K2)

3.2 Processus de revue (K2)
- Rappeler les phases, rôles et responsabilités d'une revue formelle typique. (K1)
- Expliquer les différences entre les différents types de revues: revue informelle, revue technique, walkthrough et inspection. (K2)
- Expliquer les facteurs liés à une exécution de revues couronnées de succès. (K2)

3.3 Analyse statique outillée (K2)
- Décrire les objectifs d'une analyse statique et les comparer à l'analyse dynamique. (K2)
- Rappeler les défauts et erreurs typiques identifiées par les analyses statiques et les comparer aux revues et tests dynamiques. (K1)
- Lister les avantages typiques des analyses statiques. (K1)
- Lister les défauts typiques dans le code et la conception qui peuvent être identifiés par des outils d'analyse statique. (K1)

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

Termes
Tests dynamiques, revue, analyse statique.

Background
Les techniques de tests statiques n'exécutent pas le logiciel qui est testé ; elles sont soit manuelles (revues) soit automatisées (analyse statique).

Les revues sont une manière de tester des produits logiciel (y compris du code) et peuvent être exécutées bien avant l'exécution de tests dynamiques. Les défauts détectés pendant les revues 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 tests, les spécifications de tests, cas de tests, scripts de tests, guides d'utilisateurs ou pages web.

Les bénéfices des revues incluent une détection anticipée des défauts et leur correction, des améliorations de productivité dans le développement, des durées de développement réduites, des durées et des coûts de tests réduits, des réduction 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 dynamique est peu probable.

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

Les défauts typiques plus facile à trouver lors de revues plutôt que pendant les tests dynamiques sont : déviation des standards, défauts d'exigences, défauts de conception, maintenabilité insuffisante et spécifications incorrectes d'interfaces.

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

Termes
Critères d'entrée, critères de sortie, revue formelle, revue informelle, inspection, lancement, métriques, modérateur/ inspection leader, revue de pairs, réviseur, réunion de revue, processus de revue, scribe, revue technique, walkthrough.

Background
Les revues varient de très informelles à très formelles (c'est à dire très structurées et réglementées). Le formalisme d'un processus de revue est lié à des facteurs tels la maturité du processus de développement, les exigences légales ou réglementaires ou le besoin de preuves formelles.

La manière dont une revue est exécutée dépend des objectifs convenus pour la revue (p.ex. trouver des défauts, augmenter la compréhension, ou discussion et décision par consensus).

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

- Planification: choisir le personnel, affecter les rôles, définir les critères d'entrée et de sortie pour des types de revues plus formelles (p.ex. inspection); et sélectionner les parties du document à regarder.
- Lancement: distribuer les documents; expliquer les objectifs, processus et documents aux participants; vérifier les critères d'entrée (pour des types de revues plus formelles).
- Préparation individuelle: travail effectué par chacun des participants individuellement avant la réunion de revue, noter les défauts potentiels, les questions et commentaires.
- Réunion de revue: discussion ou consignation, avec des résultats documentés ou des minutes (pour les types de revues plus formelles). Les participants à la réunion peuvent simplement noter les défauts, faire des recommandations pour la gestion des défauts, ou prendre des décision au sujet des défauts.
- Correction: réparer les défauts trouvés, typiquement effectuée par l'auteur.
- Suivi: vérifier que les défauts ont été pris en compte, récolter les métriques et vérifier les critères de sortie (pour les types de revues plus formelles).

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 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 l document(s) à revoir.
- Réviseurs: les individus avec une culture technique ou commerciale 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 de différentes perspectives, et en utilisant des check-lists, les revues peuvent devenir plus efficaces et rentables, par exemple, une check-list basée sur le 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 document unique peut 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 un walkthrough avec des clients. Les caractéristiques principales, option et objectifs des types de revues habituelles sont:

Revue informelle
Caractéristiques clé:
- Pas de processus formel;
- Peut inclure la programmation par paires ou une revue de conception et de code par un responsable technique;
- Peut optionnellement être documentée;
- Peut varier en utilité selon le réviseur;
- Objectif principal: manière bon marché d'obtenir des résultats.

Walkthrough
Caractéristiques clé:
- Réunion dirigée par l'auteur;
- Scénarios, répétition à blanc, groupe de pairs;
- Sessions sans limite de durée;
- Optionnellement une réunion de préparation de revue par les réviseurs, un rapport de revue, une liste de constatations et un scribe (qui n'est pas l'auteur) ;
- Varie en pratique d'informel à très formel;
- Objectifs principaux: apprendre, gagner en compréhension, trouver des défauts.

Revue technique
Caractéristiques clé:
- Documentée, processus de détection de défauts défini incluant des pairs et des experts techniques;
- 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;
- Peut optionnellement utiliser des check-lists, rapports de revue, liste de constatations et participation de l'encadrement;
- Peut varier en pratique d'informelle à très formelle;
- Objectifs principaux: discuter, décider, évaluer des alternatives, trouver des défauts, résoudre des problèmes techniques et vérifier la conformité à des standards et des spécifications.

I**nspection**
Caractéristiques clé:
- Dirigée par un modérateur formé (pas l'auteur);
- Généralement examen par les pairs;
- Rôles définis;
- Inclut des métriques ;
- Processus formel basé sur des règles et des check-lists avec des critères d'entrée et de sortie ;
- Réunion de préparation;
- Rapport d'inspection, liste de constatations;
- Processus formel de suivi;
- Optionnellement, processus d'amélioration et lecteur;
- Objectif principal: trouver des défauts.

3.2.4 Facteurs de succès des revues (K2)

Les facteurs de succès des revues incluent:

- Chaque revue a un objectif prédéfini et clair.
- Les personnes impliquées sont adéquates pour les objectifs de la revue.
- 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)).
- Les techniques de revue adaptées aux types et au niveau de livrable logiciel, et aux types et niveau de réviseurs, sont appliquées.
- Des check-lists ou rôles sont utilisées si approprié, afin d'augmenter l'efficacité de détection des défauts.
- Des formations sont données sur les techniques de revue, spécialement concernant les techniques plus formelles telles les inspections.
- L'encadrement supporte un bon processus de revue (p.ex. en incorporant du temps pour les activités de revue dans les plannings 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.

Background
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. Comme pour les revues, l'analyse statique trouve des défauts plutôt que des défaillances. Les outils d'analyse statique analysent le code programme (p.ex. flux de contrôle et flux de données), ainsi que les sorties générées tel que HTML et/ou XML.

La valeur des tests statiques est dans:

- La détection rapide d'erreurs avant l'exécution des tests.
- Une information avancée sur certains aspects louches du code ou de la conception, par le calcul de métriques, par exempleune 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.
- 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;
- Code non accessible (code mort);
- Violation des standards de programmation;
- Vulnérabilités de sécurité;
- Violation de syntaxe dans le code ou le modèle logiciel.

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, et par les concepteurs pendant la modélisation logicielle. Les outils d'analyse statique peuvent produire un nombre important de messages d'avertissement, lesquels doivent être gérés convenablement afin de permettre une utilisation optimale de l'outil.

Les compilateurs peuvent offrir un certain support aux analyse statiques, y inclus 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 Identifier les conditions de test et concevoir les cas de tests (K3)

- Différencier la spécification de conception d'un testde la spécification des cas de tests et des spécifications de procédures de test. (K1)
- Comparer les termes condition de test, cas de tests et procédure de tests. (K2)
- Ecrire les cas de tests: (K3)
  - Montrant une traçabilité vers les exigences;
  - Contenant un résultat attendu.
- Traduire les cas de tests en des spécifications de procédures de tests correctement structurées avec le niveau de détail adéquat par rapport au niveau de connaissance des testeurs. (K3)
- Ecrire un planning d'exécution des tests pour un sous-ensemble donné de cas de tests, en considérant les priorités, et les dépendances techniques et logiques. (K3)

4.2 Catégories de techniques de conception de tests (K2)

- 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)
- Expliquer les caractéristiques 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)

- Ecrire les cas de tests pour un modèle logiciel donné en utilisant les techniques de conception de tests suivantes: (K3)
  - Partitions d'équivalence;
  - Analyse des valeurs limites;
  - Tables de décision;
  - Diagrammes de transition d'états.
- Comprendre 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)
- Comprendre les concept des tests basés sur les cas d'utilisation et ses avantages. (K2)

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

- Décrire le concept et l'importance de la couverture du code. (K2)
- Expliquer le concept de couverture des instructions, et comprendre que ces concepts peuvent aussi être utilisés pour d'autres niveaux de tests que les tests de composants (p.ex. sur les procédures commerciales au niveau système). (K2)
- Ecrire les cas de test à partir des données du flux de contrôle en suivant les techniques de conception de tests suivantes: (K3)
  - Couverture des instructions;
  - Couvertures des décisions.
- Evaluer la complétude des couvertures d'instructions et de décisions. (K3)

4.5 Techniques basées sur l'expérience (K2)

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

4.6 Sélectionner les techniques de test (K2)

- Lister les facteurs qui influencent la sélection des techniques de conception de test pour un type de problème particulier, tel que le type de système, les risques, les exigences client, modèles pour la modélisation des cas d'utilisation, modèles d'exigences ou connaissance des testeurs. (K2)

4.1 *Identifier les conditions de test et concevoir des cas de tests (K3)* — *15 minutes*

Termes
Cas de tests, spécification de cas de tests, conditions de tests, données de tests, spécification des procédures de test, script de test, traçabilité.

Background
Le processus d'identification des conditions de tests et de conception des tests consiste en un certain nombre d'étapes:

- Concevoir les tests en identifiant les conditions de tests.
- Spécifier les cas de tests.
- Spécifier les procédures de tests.

Le processus 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 formalité dépend du contexte des tests incluant l'organisation, la maturité des tests et des processus de développement, des contraintes de temps et des personnes concernées.

Pendant la conception des tests, la documentation de base des tests est analysée de façon à déterminer ce qui est à tester, c'est à dire 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 tests (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 la conception des tests, les approches détaillées de tests sont implémentées basées sur, entre autres considérations, les risques identifiés (voir Chapitre 5 pour plus d'informations sur les analyses de risques).

Pendant les spécifications des cas de tests, les cas de test et données de test sont développés et décrits en détail en utilisant les techniques de conception des tests. Un cas de tests 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 certaines conditions de tests. Le Standard pour la Documentation de Tests Logiciel ('Standard for Software Test Documentation' IEEE 829) décrit le contenu des spécifications de conception de tests et des spécifications des cas de tests.

Les résultats attendus devraient être produits comme faisant partie des spécifications de cas de tests 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.

Les cas de tests sont mis en un ordre exécutable pour obtenir la spécification des procédures de tests. Les procédures de tests (ou scripts de tests manuels) spécifient la séquence des actions pour l'exécution d'un tests. Si les tests sont exécutés avec un outil d'exécution des tests, la séquence d'action est spécifiée dans un script de tests (ce 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, et possiblement les scripts de tests automatisés, sont exécutés, quand et par qui. Les plannings d'exécution des tests prendront en compte les facteurs tels que les tests de régression, de priorisation, et de dépendance technique et logique.

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

Termes
Techniques boîte noire, techniques basées sur l'expérience, techniques basées sur les spécifications, techniques basées sur les structures, techniques boîte blanche.

Background
L'objectif des techniques de conception des tests est d'identifier les conditions de tests et les cas de tests.

C'est une distinction classique d'indiquer les techniques de tests comme boîte blanche ou boîte noire. Les techniques boîte noire (aussi appelées techniques basées sur les spécifications) sont une façon de dériver et de sélectionner les conditions de tests ou les cas de tests en se basant sur une analyse de la documentation de base des tests, que ce soit fonctionnels ou non fonctionnels, pour un composant ou système sans référence à sa structure interne. Les techniques boîte blanche (aussi dites techniques structurelles ou basées sur les structures) sont basées sur une analyse de la structure du composant ou du système.

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 ou l'expérience, et référence comme techniques boîte blanche les approches basée sur les structures.

Caractéristiques communes des techniques 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 tests sont dérivés de façon systématique.

Caractéristiques habituelles des techniques basées sur la structure:

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

Caractéristiques communes des techniques basées sur l'expérience:

- La connaissance et l'expérience des personnes sont utilisées pour dériver les cas de tests.
  - Connaissance des testeurs, développeurs, utilisateurs et autres personnes concernant le logiciel, son utilisation et son environnement;
  - Connaissance des défauts possibles et de leur distribution.

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'emploi.

4.3.1 Partitions d'équivalence (K3)

Les entrées d'un logiciel ou système sont divisées en groupes qui doivent montrer un comportement similaire, donc elles auront un traitement identique. Les partitions (ou classes) d'équivalence peuvent être trouvées pour des données valides 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. pendant les tests d'intégration). Des tests peuvent être conçus pour couvrir des partitions. Les partitions d'équivalence sont applicables à tous les niveaux de tests.

Les partitions d'équivalence représentent une technique qui peut être utilisée pour atteindre une couverture d'entrées et une couverture 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, 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 tests, une valeur de chaque limite est sélectionnée.

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.

Cette technique est souvent considérée comme une extension des partitions d'équivalences et peut être utilisée aussi bien sur les données d'entrées humaines, par exemple, que sur des limites de tables ou de temps. Les valeurs limites peuvent aussi être utilisées pour la sélection des données de tests.

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. 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 couvrir toutes les combinaisons de conditions de déclenchement.

La force des tests par les tables de décision 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 les actions du logiciel dépendent 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 testeurs de visualiser le logiciel en termes d'états, de transitions entre les états, de données d'entrées ou d'actions qui déclenchent des changements d'état (transitions) et des actions qui peuvent résulter de ces transitions. Les états du système ou de l'objet en 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 exercer 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 de gestion possédant des états spécifiques ou pour tester les dialogues d'interfaces écran (p.ex. pour les applications Internet ou les scénarios de gestion).

4.3.5 Tests de cas d'emploi (K2)

Les tests peuvent être spécifiés à partir de cas d'utilisation ou de scénarios de gestion. 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. Chaque cas d'utilisation a des pré-conditions, qui doivent être atteintes afin pour qu'un cas d'emploi soit exécuté avec succès. Chaque cas d'emploi 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'emploi. Un cas d'emploi a généralement un scénario dominant (c'est à dire le plus plausible), et parfois des branches alternatives.

Les cas d'utilisation décrivent le "flux du processus" dans un système, sur base d'une utilisation probable, donc les cas de tests dérivés des cas d'emploi sont extrêmement utiles pour découvrir les défauts dans le flux de traitement pendant l'utilisation réelle du système. Les cas d'emploi, 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.

4.4 *Techniques basées sur la structure ou Boîte blanche (K3)* — *60 minutes*

Termes
Couverture de code, couverture des décisions, couverture des instructions, tests structurels, tests basés sur la structure, tests boîte blanche.

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

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

Dans cette section, deux 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 tests. Le test des instructions dérive des cas de tests pour exécuter des instructions spécifiques, généralement pour accroître la couvertures des instructions.

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 tests. Les cas de tests provenant des tests de décisions sont amenés à exécuter des résultats de décisions spécifiques, généralement pour accroître la couverture des décisions.

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 (p.ex. au niveau intégration) où le pourcentage des modules, composants ou classes qui ont été exercées par une suite de cas de tests peut être exprimé comme une couverture des modules, composants ou classes.

Des outils de support sont utiles pour les tests structures de code.

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

Termes
Estimation d'erreur, tests exploratoires.

Background
La pratique probablement la plus utilisée est l'estimation d'erreurs. 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 approche structurée de la technique d'estimation d'erreurs est d'énumérer une liste d'erreurs possibles et de concevoir des tests qui s'attaquent à ces erreurs. Ces listes de défauts et de défaillances peuvent être construites basées sur l'expérience, les 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), basées sur une charte de tests contenant les objectifs de tests, et effectués dans des périodes de temps. C'est l'approche la plus utile dans le cas ou les spécifications sont rares ou non adéquates, et de sévères contraintes de temps, de façon à augmenter ou compléter d'autres méthodes de tests plus formelles. 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 termes spécifiques.

Background
Le choix des techniques de tests à utiliser dépend d'un nombre de facteurs, incluant le type de système, les standards réglementaires, les exigences client ou contractuelles, le niveau de risque, le type de risques, les objectifs de test, la documentation disponible, la connaissance des testeurs, le temps disponible et le budget, le cycle de vie de développement utilisé, l'utilisation 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.

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) — 180 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)
- Reconnaître l'importance du test indépendant. (K1)
- Énumérer les avantages et inconvénients du test indépendant pour une organisation. (K2)
- Reconnaître les différents membres d'une équipe à envisager lors de la formation d'une équipe de test. (K1)
- Rappeler les tâches typiques d'un responsable de test et d'un testeur. (K1)

5.2 Estimation et planification du test (K2)
- Reconnaître les différents niveaux et objectifs de la planification du test. (K1)
- 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)
- Rappeler les facteurs typiques qui influent sur l'effort relatif au test. (K1)
- Faire la différence entre deux approches d'estimation conceptuellement différentes : l'approche basée sur des mesures et l'approche basée sur l'expertise. (K2)
- Faire la différence entre le sujet de la planification du test pour un projet, pour des niveaux de test individuel (par exemple, test système) ou des finalités de test spécifiques (par exemple, test d'utilisabilité), et pour l'exécution de tests. (K2)
- Énumérer les tâches de préparation et d'exécution du test qui nécessitent une planification. (K1)
- Reconnaître et justifier les critères 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)
- Rappeler les mesures habituelles utilisées pour suivre la préparation et l'exécution du test. (K1)
- Comprendre et interpréter les mesures de test pour la documentation et le contrôle du test (par exemple, les défauts trouvés et corrigés, les test réussis et défaillants. (K2)
- Résumer le but et le contenu du rapport de synthèse établi selon le guide « Standard for Software Test Documentation » (IEEE 829). (K2)

5.4 Gestion de configuration (K2)
- Résumer la manière dont la gestion de configuration assiste le test. (K2)

5.5 Test et risques (K2)
- Décrire un risque comme un problème probable qui peut compromettre l'atteinte des objectifs de projet d'un ou de plusieurs acteurs. (K2)
- Se rappeler que les risques sont déterminés par leur probabilité (d'occurrence) et leur impact (dommages en résultant). (K1)
- Distinguer entre les risques liés au projet et ceux liés au produit. (K2)
- Reconnaître les risques typiques du produit et du projet. (K1)
- 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)
- Reconnaître le contenu du rapport d'incident établi selon le guide « Standard for Software Test Documentation » (IEEE 829). (K1)
- 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 :

- 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).
- Des testeurs indépendants peuvent constituer un goulot d'étranglement comme dernier point de vérification.
- Les développeurs perdent le sens de la responsabilité pour la qualité.

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 or 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 du test est intitulé gestionnaire du test ou coordinateur du 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 du test planifie, surveille et contrôle les activités et tâches du test définies dans la section 1.4.

Les tâches typiques du responsable des tests 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 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.
- Programmer les tests.
- Établir des rapports de synthèse du test à partir des informations recueillies pendant le test.

Les tâches typiques 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 à celui d'intégration seront des développeurs, ceux du test de recette seront des experts métier et des utilisateurs et ceux pour la recette opérationnelle seront des opérateurs.

5.2 Estimation et planification des tests (K2)	*50 minutes*

Termes
Critère d'entrée, critère de sortie, test exploratoire, approche du test, niveau de test, plan de test, procédure 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 de recette. Le schéma des documents de planification de test est défini dans le guide « Standard for Software Test Documentation » (IEEE 829).

La planification est influencée par la politique de test de l'organisation, la portée du test, les objectifs, les risques, les contraintes, la criticalité, la testabilité et la disponibilité des ressources. Au fur et à mesure de l'avancement du projet et de la planification du test, 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 (K2)

Les activités de la planification du test peuvent comprendre les suivantes :

- 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).
- 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 du test, 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 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.

5.2.4 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.
- 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.

5.2.5 Approches des tests (stratégies de test) (K2)

Une façon de classifier les approches ou les stratégies de test repose sur le moment où le travail de conception des tests débute :

- Les approches préventives, où la conception des tests commence le plus tôt possible.
- Les approches réactives, où la conception des tests commence lorsque le logiciel ou système a été produit.

Les approches ou stratégies typiques peuvent comprendre les suivantes :

- 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 fault-attack), 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.

La sélection de l'approche de test doit prendre en compte le contexte, notamment :

- Le risque d'échec du projet, les aléas du produit et le risque couru par des personnes, l'environnement et l'entreprise suite à une défaillance du produit.

- Les compétences et l'expérience des personnes quant aux techniques, outils et méthodes proposés.
- Les objectifs de l'entreprise du test et la mission de l'équipe de test.
- Les aspects réglementés, comme les réglementations externes et internes du processus de développement.
- La nature du produit et du métier.

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

Termes
Densité de défauts, taux de défaillance, contrôle des tests, couverture 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 comprennent les suivantes :

- 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 retest).
- 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 829).

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 pour ce niveau de test.
- L'adéquation des approche 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.

Des exemples d'actions de contrôle de test peuvent être :

- 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 construction.

5.4 Gestion de configuration (K2)	*10 minutes*

Termes
Gestion de configuration, contrôle de versions.

Background
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 inclure l'assurance que :

- Tous les éléments du matériel de test sont identifiés, sous contrôle de versions, que les changements sont identifiés et retraçables, 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 lié au produit, risques lié au projet, risques, test basé sur les risques

Background
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 (K1, K2)
Les risques liés au projet sont les risques menaçant la capacité de ce dernier à atteindre ses objectifs, tels que :

- Problèmes d'acquisition :
  - Défaillance d'une tierce partie ;
  - Problèmes contractuels.
- Facteurs organisationnels :
  - Manque de savoir-faire et de personnel ;
  - Problèmes de personnel et de formation ;
  - Problèmes politiques, tels que
    - problèmes dûs 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 ;
  - Qualité de la conception, du code et des tests.

Pour analyser, gérer et diminuer ces risques, le gestionnaire du test suivra les principes bien établis de la gestion de projet. Le guide « Standard for Software Test Documentation » (IEEE 829) 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).
- 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
Rapport d'incident

Background
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. Les incidents devraient 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 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.

Un testeur ou réviseur consigne en général, les informations sur un incident suivantes si elles sont disponibles :

- Date de l'incident, organisation faisant part de l'incident, auteur, approbations et états.
- Portée, sévérité et priorité de l'incident.
- Références, comprenant l'identification de la spécification du cas de test qui a révélé le problème.

Les détails d'un rapport d'incident peuvent comprendre les points suivants :

- Résultats attendus et effectifs.
- Date de la découverte de l'incident.
- 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.
- 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, tels que la séquence des actions entreprises par des membres de l'équipe du projet afin de localiser l'incident, de le corriger et d'en confirmer la correction.

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

Références

5.1.1 Black, 2001, Hetzel, 1998
5.1.2 Black, 2001, Hetzel, 1998
5.2.5 Black, 2001, Craig, 2002, IEEE 829, Kaner 2002
5.3.3 Black, 2001, Craig, 2002, Hetzel, 1998, IEEE 829
5.4 Craig, 2002
5.5.2 Black, 2001 , IEEE 829
5.6 Black, 2001, IEEE 829

6. Outils de support aux tests (K2) — 80 minutes

Objectifs de connaissance pour Outils de support aux tests
Les objectifs identifient les capacités que vous aurez à la fin de chaque module.

6.1 Les types d'outils de tests (K2)
- Classer les différents types d'outils selon les activités du processus de tests. (K2)
- Reconnaître les outils qui peuvent aider les développeurs dans leurs tests. (K1)

6.2 Utilisation efficace des outils: bénéfices potentiels et risques (K2)
- Résumer les bénéfices et risques potentiels d'automatisation et d'utilisation d'outils pour les tests. (K2)
- Reconnaître que les outils d'exécution des tests peuvent présenter des techniques d'écriture de scripts différentes, notamment guidées par les données (data driven) et par mots clé (keyword driven). (K1)

6.3 Introduire un outil dans une organisation (K1)
- Exposer les principes majeurs lies à l'introduction d'un outil dans une organisation. (K1)
- Exposer les objectifs d'une preuve de concept et d'un projet pilote pour l'évaluation d'un outil. (K1)
- 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 *Les types d'outils (K2)* — *45 minutes*

Termes
Outils de gestion de configuration, outils de mesure de couverture, outils de déboguage, pilote, 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, bouchon, 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 Classification des outils de tests (K2)

Il existe un certain nombre d'outils qui assistent des aspects différents des tests. 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. Certains outils du commerce s'adressent à une seule activité ; d'autres vendeurs d'outils offrent un ensemble d'outils qui assistent de nombreuses activités, voire toutes.

Les outils de tests peuvent améliorer l'efficacité des activités de tests en automatisant les tâches répétitives. Les outils de tests peuvent aussi améliorer la fiabilité des tests, par exemple, en automatisant des comparaisons de données volumineuses ou en simulant des comportements du logiciel en cous de test.

Quelques types d'outils peuvent être intrusifs en ce que l'outil lui-même peut affecter le résultat actuel du tests. Par exemple, la mesure temporelle peut varier selon la manière dont il est mesuré avec divers outils de mesure de performances, ou l'on peut obtenir des mesures de couverture de code différentes selon l'outil de couverture utilisé. La conséquence de l'utilisation d'un outil intrusif est appelé 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.

6.1.2 Outils de support et de gestion 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
Les fonctionnalités des outils de gestion des tests incluent:

- Assistance à la gestion des tests et des activités de test effectuées.
- Interface avec les outils d'exécution, outils de gestion des défauts et outils de gestion des exigences.
- Contrôle de version indépendant ou interface avec un outil de gestion de configuration externe.
- Support pour la traçabilité des tests, des résultants de tests et incidents vers les documents source, tels les spécifications d'exigences.
- Enregistrement des résultats de test et génération des rapports d'avancement.
- Analyse quantitative (métriques) liées aux tests (c'est à dire tests exécutés et tests passés) et aux objets de tests (càd. incidents levés), de façon à donner de l'information sur les objets de tests, à contrôler et à améliorer les processus de tests.

Outils de gestion des exigences
Les outils de gestion des exigences archivent les énoncés des exigences, vérifient la consistance et les exigences non définies (ou manquantes), permet d'affecter des priorités aux exigences et permet à des tests individuels d'être reliés à des exigences, des fonctions et/ou des caractéristiques. La traçabilité peut être indiquée dans les rapports d'avancement et de gestion des tests. La couverture des exigences, fonctions et/ou caractéristiques par un ensemble de tests peut aussi figurer dans ces mêmes rapports.

Outils de gestion d'incidents
Les outils de gestion d'incidents archivent et gèrent les rapports d'incident, c'est à dire les défauts, défaillances ou problèmes constatés et les anomalies, ils assistent également à la gestion des rapports d'incidents de diverses façons incluant:

- Faciliter leur organisation.
- Affecter des actions à des personnes (càd. corriger ou effectuer des tests de confirmation).
- Attribuer un état (càd. rejeté, prêt à tester ou reporté à une version ultérieure).

Ces outils permettent le suivi d'avancement des incidents sur une période, et fournissent souvent une assistance à l'analyse statistique et à la génération de rapports sur les incidents. Ils sont aussi connus sous le terme outils de gestion d'anomalies.

Outils de gestion de configuration
Les outils de gestion de configuration ne sont pas strictement des outils de tests, mais sont typiquement nécessaires pour conserver la trace des différentes versions et constructions des logiciels et des tests.

Les outils de gestion de configuration:

- Archivent les informations relatives aux versions et constructions des logiciels et des tests associés.
- Permet la traçabilité entre les fournitures de tests et les livrables logiciels ainsi que les variantes du produit.
- Sont particulièrement utiles lorsque les développements s'effectuent sur plus d'un environnement de configuration matérielle/logicielle (càd. pour diverses versions de systèmes opératoires, diverses librairies ou compilateurs, divers navigateurs ou divers ordinateurs).

6.1.3 Outils d'aide aux tests statiques (K1)

Outils d'assistance au processus de revue
Les outils d'assistance au processus de revue peuvent archiver des informations relatives aux processus de revue, garder et diffuser les commentaires des revues, informer sur les efforts et défauts, gérer les références vers les règles de revue et/ou les checklists, ainsi qu'assurer le traçabilité entre les documents et le code source. Ils peuvent aussi fournir de l'aide aux revues en ligne, utiles en cas d'équipes de révision dispersées géographiquement.

Outils d'analyse statique (D)
Les outils d'analyse statique aident développeurs, testeurs et personnel dédié à l'assurance qualité dans la recherche de défauts avant l'analyse dynamique. Leur objectifs principaux incluent :

- Le respect des standards de codage.
- L'analyse des structures et de leurs dépendances (càd. liens des pages web).
- L'aide à la compréhension du code.

Les outils d'analyse statique peuvent calculer des métriques à partir du code (p.ex. complexité), qui peuvent fournir une information précieuse pour, par exemple, l'analyse de risque ou la planification.

Outils de modélisation (D)

Les outils de modélisation peuvent aider à valider des modèles du logiciel. Par exemple, un vérificateur de modèle de bases de données peut trouver des défauts ou inconsistances dans le modèle de données ; d'autres outils de modélisation peuvent découvrir des défauts dans un modèle d'états ou un modèle objet. Ces outils peuvent souvent aider à générer quelques cas de test basés sur les modèles (voir aussi « outils de conception des tests » ci-après).

Le principal avantage des outils d'analyse statique et des outils de modélisation est la rentabilité (Retour sur Investissement) procurée en découvrant d'avantages de défauts plus tôt dans le processus de développement. En conséquence, le processus de développement peut être accéléré et amélioré par des reprises moins importantes.

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

Outil de conception des tests
Les outils de conception des tests génèrent des entrées de test ou des tests réels à partir des exigences, d'une interface utilisateur graphique (GUI), de modèles de conception (états, données ou objets) ou à partir du code. Ce type d'outil peut aussi générer les résultats attendus (p.ex. utilisation d'un oracle de tests). Les tests générés à partir d'un modèle d'état ou d'un modèle objet sont utiles pour vérifier l'implémentation du modèle dans le logiciel, mais sont rarement suffisants pour vérifier tous les aspects du logiciel ou du système concerné. Ils peuvent faire gagner un temps précieux et fournir une exhaustivité accrue des tests par la complétude des tests que l'outil peut générer.

D'autres outils de cette catégorie peuvent faciliter la génération des tests en offrant des cadres structurés, parfois appelés "test frame", qui génèrent des tests ou des bouchons, et ainsi accélèrent le processus de conception des tests.

Outil de préparation des données (D)
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. Un avantage de ces outils est de s'assurer que les données réelles transférées vers l'environnement de tests sont rendues anonymes de façon à protéger leur contenu.

6.1.5 Outils d'aide à l'exécution et au suivi des tests (K1)

Outils d'exécution des tests
Les outils d'exécution des tests permettent l'exécution automatique, ou semi-automatique, des tests en utilisant des entrées archivées et des résultats attendus, par le biais d'un langage de script. Ce langage permet de manipuler les tests avec des efforts limités, par exemple, pour répéter les tests avec des données différentes ou tester une partie différente du système avec des étapes similaires. En général, ces outils comprennent des possibilités de comparaison dynamique et proposent des information de suivi de chaque exécution des tests.

Les outils d'exécution des tests peuvent aussi être utilisés pour enregistrer des tests, au quel cas ils peuvent être référencés comme des outils de capture et de rejeu. Capturer des données d'entrée des tests pendant des tests exploratoires ou des tests non scriptés peut être utiles de façon à reproduire et/ou documenter un test, par exemple, en cas d'apparition d'un défaut.

Harnais de test / Outils de tests dans un cadre unitaire (D)
Un harnais de tests peut faciliter le test des composants ou parties de systèmes en simulant l'environnement où s'exécute l'objet en cours de test. Ceci peut être fait soit parce que les autres composants de cet environnement ne sont pas disponibles et sont remplacés par des bouchons et/ou pilotes, ou simplement en fournissant un environnement prévisible et contrôlable dans lequel un défaut peut être localisé au sein de l'objet en cours de test.

Un cadre peut être créé lorsque des parties du code, d'objets, de méthodes ou de fonctions, unités ou composants peuvent être exécutées, en appelant l'objet à tester et/ou en fournissant des informations à cet objet. Ceci peut être effectué en affectant des moyens artificiels de fourniture de données à l'objet en test, et/ou en fournissant des bouchons pour obtenir des résultats de l'objet, en lieu et place des cibles réelles des résultats.

Les harnais de tests peuvent aussi être utilisés pour fournir un cadre d'exécution où les langages, systèmes d'opération ou matériels doivent être testés ensemble.

Ils peuvent être appelés outils de tests dans un cadre unitaire quand ils se focalisent sur les niveaux de tests de niveau composant. Ce type d'outil aide à l'exécution des tests de composants en parallèle avec la construction du code.

Comparateurs de tests
Les comparateurs de tests déterminent les différences entre les fichiers, bases de données ou résultats de tests. Les outils d'exécution de tests comportent typiquement des comparateurs dynamiques, mais une comparaison post-exécution peut être effectuée par un outil de comparaison séparé. Un comparateur de tests peut utiliser un oracle de tests, en particulier s'il est automatisé.

Outils de mesure de couverture (D)
Les outils de mesure de couverture peuvent être soit intrusifs, soit non-intrusif selon la technique de mesure utilisée, ce qui est mesuré et le langage de développement. Les outils de couverture de code mesurent le pourcentage de types de structures de code exercés (p.ex. instructions, branches ou décisions, et d'appels de modules ou de fonctions). Ces outils indiquent de quelle manière le type de structure analysé a été parcouru par un jeu de tests.

Outils de sécurité informatique
Les outils de sécurité recherchent des virus informatiques et des attaques de déni de service (denial of service). Un pare-feu, par exemple, n'est pas strictement un outils de tests, mais peut être utilisé dans des tests de sécurité. D'autres outils de sécurité informatique stressent le système en y cherchant des vulnérabilités spécifiques.

6.1.6 Outils de support des performances 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 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 tests de performances / outils de tests de stress
Les outils de tests de performances surveillent et indiquent le comportement du système soumis à une variété de conditions d'utilisation simulées. Ils simulent une charge sur une application, une base de données, ou un environnement système, tel qu'un serveur ou un réseau. Ces outils sont souvent nommés en fonction de l'aspect des performances qu'ils mesurent, telles que la charge ou le stress, et sont donc connus sous le terme outil de tests de charge ou outil de tests de stress. Ils sont souvent basés sur une répétition automatisée de tests, contrôlée par des paramètres.

Outils de surveillance
Les outils de surveillance ne sont pas strictement des outils de tests, mais fournissent de l'information pouvant être utilisée à des fins de tests et qui n'est pas disponible par d'autres moyens.

Les outils de surveillance analysent continuellement, vérifient et informent sur l'utilisation de ressources système spécifiques, et alertent an cas de problèmes de service potentiels. Ils archivent des informations sur la version et la construction du logiciel et des composants de tests, et assurent la traçabilité.

6.1.7 Support outillé pour des domaines d'applications spécifiques (K1)

Des exemples individuels des types d'outils classifiés ci-dessus peuvent être dédiés à une utilisation dans un domaine d'application particulier. Par exempleil existe des outils de tests de performances spécifiques pour des applications reposant sur le web, des outils d'analyse statique pour des plates-formes de développement spécifiques, et des outils d'analyse dynamique dédiés à des aspects du test de sécurité informatique .

Des suites d'outils de test du commerce peuvent viser des domaines d'application particuliers (p.ex. les systèmes embarqués).

6.1.8 Support outillé avec d'autres outils (K1)

Les outils de tests listés ici ne sont pas les seuls types d'outils utilises par les testeurs – ils peuvent aussi utiliser des tableurs, SQL, des ressources ou des outils de déboguage (D), par exemple.

6.2 *Usage efficace d'outils : bénéfices potentiels et risques (K2)* — *20 minutes*

Termes
Test dirigé par les données (data-driven testing), test dirigé par les mots clé (keyword-driven testing), langage de script.

6.2.1 Bénéfices potentiels et risques lies aux outils de tests (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.

Les bénéfices potentiels à l'utilisation d'outils incluent :

- 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).
- Répétitivité et cohérence accrues (p.ex. tests exécutés par un outil et tests déduits des exigences).
- Evaluation objective (p.ex. mesure statiques, couverture et comportement du système).
- Facilité d'accès aux informations au sujet des tests ou de leur exécution (p.ex. statistiques et graphiques sur l'avancement des tests, taux d'incident et performances).

Les risques liés à l'utilisation d'outils incluent:

- 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).

6.2.2 Considérations particulières pour certains types d'outils (K1)

Outils d'exécution des tests
Les outils d'exécution des tests rejouent des scripts conçus pour mettre en œuvre des tests qui ont été archivés électroniquement. 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 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.
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 scripts 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 de tests de performances
Les outils de tests de performances requièrent la présence d'une personne experte en tests de performances pour aider à concevoir les tests et interpréter les résultats.

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 (Warning) 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 constituerait une approche efficace.

Outils de gestion des tests
Les outils de gestion des tests doivent 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. Les rapports doivent être conçus et suivis de façon à ce qu'ils apportent des bénéfices.

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

Termes
Pas de termes spécifiques.

Background
Les principes principaux lies 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.
- Preuve de concept pour tester les fonctionnalités requises et détermination du respect des objectifs par le produit.
- Evaluation du vendeur (y compris les aspects relatifs à la formation, au support et aux pratiques commerciales).
- Identification des exigences internes pour assister et guider l'utilisation de l'outil.

Une preuve de concept peut être effectuée sur un projet pilote de petite taille, de façon à minimiser les conséquences si des obstacles majeurs sont découverts et que le projet pilote n'est pas concluant.

Un projet pilote 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 les éléments de tests (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 (voire du support) aux nouveaux utilisateurs.
- Établir des guides d'utilisation.
- Implémenter une manière à tirer des enseignements de l'utilisation de l'outil.
- Surveiller l'utilisation de l'outil et les bénéfices recueillis.

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

7. Références

Normes

Glossaire CFTL/ISTQB des termes utilises en tests de logiciels version 1.0FR
ISTQB Glossary of terms used in Software Testing Version 1.0

[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 829] IEEE Std 829™ (1998/2005) IEEE Standard for Software Test Documentation (en cours de révision)
Voir sections 2.3, 2.4, 4.1, 5.2, 5.3, 5.5, 5.6

[IEEE 1028] IEEE Std 1028™ (1997) IEEE Standard for Software Reviews
Voir section 3.2

[IEEE 12207] IEEE 12207/ISO/IEC 12207-1996, Software life cycle processes
Voir 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. et Jaskiel, Stefan P. (2002) Systematic Software Testing, Artech House: Norwood, MA
Voir sections 1.4.5, 2.1.3, 2.4, 4.1, 5.2.5, 5.3, 5.4

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

[Gilb, 1993]: Gilb, Tom et 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. et Pettticord, B. (2002) Lessons Learned in Software Testing, John Wiley & Sons:
Voir sections 1.1, 4.5, 5.2

[Myers 1979] Myers, Glenford J. (1979) The Art of Software Testing, John Wiley & Sons:
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

Annexe A – Informations sur le syllabus

Historique de ce document

Ce document a été préparé en 2004-2005 par un groupe de travail composé de membres appointés 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 dans la production 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). A ce moment (2005), l'ISTQB comprend les membres suivants : Autriche, Danemark, Finlande, France, Allemagne, Inde, Israël, Japon, Corée, Norvège, Pologne, Portugal, Espagne, Suède, Suisse, Pays Bas (Hollande), Royaumes Unis (GB) et USA.

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.
- 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 qualification

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).

Background 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.

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.

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 à l'aspect traité, et peut résumer, comparer, classifier et donner des exemples sur le concept de test.

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

- Pour trouver des défauts quand il est meilleur marché 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 peut 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'appliqué à un contexte donné.

Exemple

- Peut identifier les valeurs limites pour des partitions valides et invalides.
- Peut sélectionner les cas de tests pour un diagramme de transition donné de façon à couvrir toutes les transitions.

Références

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

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. (AMERICAN-ENGLISH)

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. (SHELF-LIFE).

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) et ceux que le candidat doit être capable d’utiliser et de mettre en pratique (K3). (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 meme 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 reflèter 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 claires, plus de détail 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 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.

Annexe D – Note aux fournisseurs de formations

Chaque titre de sujet principal dans ce syllabus est associé à une durée affectée 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 devraient couvrir les quatres 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.

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 couvertures des instructions et 100% de couverture des décisions et aussi pour concevoir des cas de tests 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.

Index