POLYTECHNIQUE MONTRÉAL
UNIVERSITÉ D'INGÉNIERIE

LOG8430 : Attributs de qualité, partie 4

Attributs de qualité – partie 4

Définition des attributs de qualité

1. Disponibilité
2. Interopérabilité
3. Modifiabilité
4. Performance
5. Sécurité
6. **Testabilité**
7. **Facilité d'utilisation**
8. **Autres attributs de qualité**

Attribut de qualité 6: Testabilité

Le coût des tests d'un système logiciel bien conçu peut représenter entre 30 et 50% du coût de développement, et parfois plus.

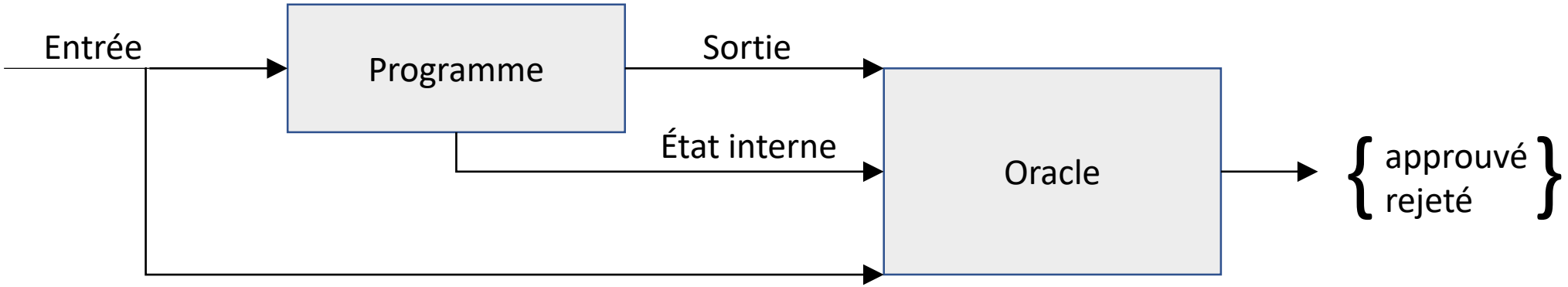
Si l'architecture peut permettre de réduire ces coûts, le retour peut être très important.

La testabilité réfère à la facilité avec laquelle un logiciel peut être amené à révéler ses défauts par des tests.

La testabilité réfère donc à la probabilité, si un défaut existe dans un logiciel, que ce défaut apparaisse à la prochaine exécution d'un test.

Attribut de qualité 6: Testabilité

Un modèle pour le test d'un programme comprend les éléments suivants: des données d'entrée, des résultats de sortie, l'état interne du programme et un oracle qui décide si les résultats attendus sont corrects ou non.



Attribut de qualité 6: Testabilité

- L'oracle est un agent humain ou mécanique qui décide si le résultat est correct ou non en comparant la sortie à la spécification du programme.
- Les sorties ne sont pas uniquement fonctionnelles, mais peuvent inclure des mesures dérivées d'attributs de qualité, p.ex. le temps nécessaire pour produire un résultat.
- L'état interne du programme peut également être exposé à l'oracle qui décidera si le programme est dans un état correct.

  → Établir et examiner l'état interne d'un programme est un aspect des tests très présent dans les tactiques de testabilité.

Attribut de qualité 6: Testabilité
Harnais de test

Pour qu'un système puisse être correctement testé, il faut être en mesure de contrôler les entrées de chaque composante et d'en observer les sorties.

Il peut être aussi nécessaire de manipuler et d'observer l'état interne de la composante.

Ce contrôle et ces observations sont souvent réalisés à l'aide d'un harnais de test, qui permet d'exécuter des procédures de test et d'enregistrer les résultats.

Scénario général de testabilité

Source du stimulus

Testeur unitaire, Testeur d'intégration, Testeur système, Testeur d'acceptation et utilisateur final. Humain ou système automatique.

Stimulus

Un ensemble de tests exécutés suite à l'achèvement d'un incrément de codage: une couche de classes ou de service, l'intégration d'un sous-système, l'implémentation complète du système ou la livraison du système.

Artéfact

Une unité de code (un module), un sous-système ou le système entier.

Environnement

Les tests sont effectués en cours de développement, à la compilation, au déploiement ou en cours d'exécution. Peut inclure le harnais de test.

Réponse

Le système peut être contrôlé pour effectuer les tests désirés et les résultats des tests peuvent être observés.

Mesure de la réponse

Visent à mesurer combien facilement un système sous test divulgue ses fautes. Les mesures peuvent inclure: l'effort nécessaire pour trouver une faute ou une classe de fautes, l'effort requis pour tester un pourcentage donné des énoncés, la longueur de la chaine de test la plus longue, l'effort pour effectuer les tests, pour trouver des fautes, un estimé de la probabilité de trouver des nouvelles fautes et le temps et l'effort nécessaire pour préparer l'environnement de test.

Exemple de scénario concret de testabilité

Source du stimulus

Testeur unitaire

Stimulus
Unité de code complétée

Artéfact
Unité de code

Environnement
Développement

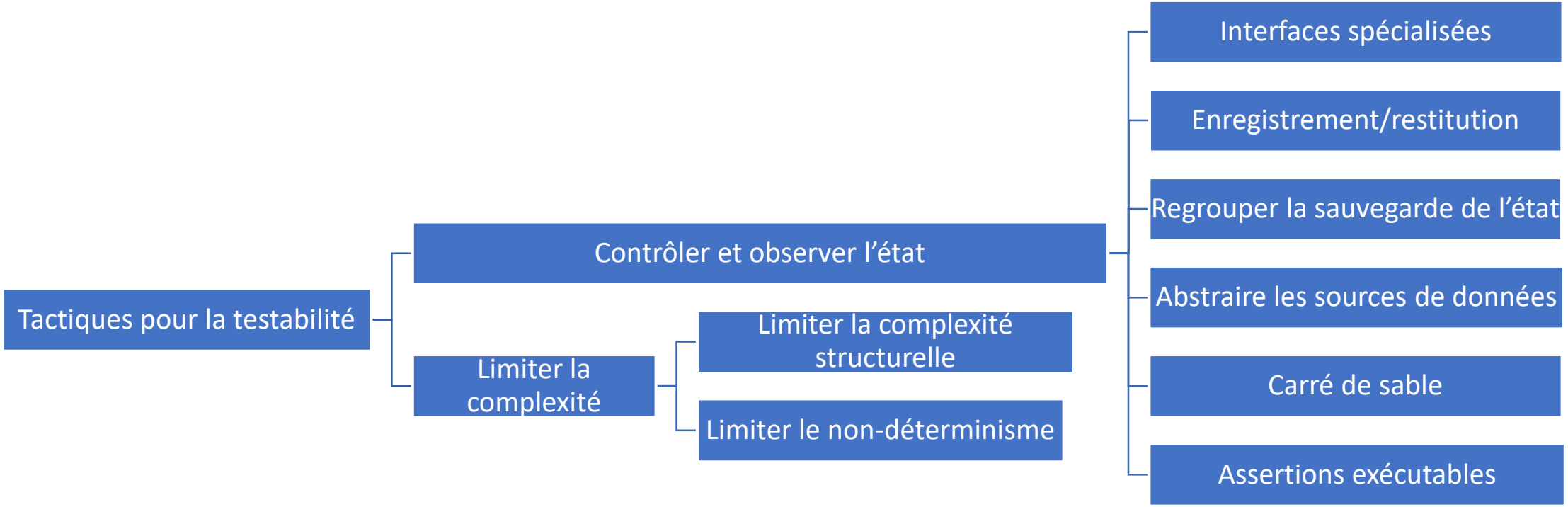
Réponse
Résultats enregistrés

Mesure de la réponse
85% des chemins couverts en trois heures

Attribut de qualité 6: Tactiques pour la testabilité

Les tactiques de testabilité visent à rendre les tests plus facile à effectuer lorsqu'une partie du développement logiciel est complété.

Attribut de qualité 6: Tactiques pour la testabilité



Attribut de qualité 6: Tactiques pour la testabilité: **Contrôler et observer l'état**

- Interfaces spécialisées: prévoir des interfaces spécialisées pour les tests permet de contrôler et d'interroger les valeurs des variables d'une composante par le harnais de test ou en cours d'exécution normale. P.ex.:
  - Des méthodes *get* et *set* pour les variables importantes, les modes ou les attributs (des méthodes qui ne seraient pas nécessairement disponibles sauf pour les tests),
  - Une méthode *report* qui retourne l'état complet d'un objet,
  - Une méthode *reset* pour initialiser l'état interne d'un objet,
  - Une méthode pour activer un mode de sortie « verbeux », différents niveaux d'enregistrement des évènements ou de monitoring des ressources.
- Ces interfaces de test devraient être clairement distinctes des interfaces standard fournies pour implémenter la fonctionnalité, de façon à pouvoir les retirer au besoin.

Attribut de qualité 6: Tactiques pour la testabilité: **Contrôler et observer l'état**

- Enregistrement/restitution: retrouver l'état du système au moment où une faute s'est produite peut être difficile. Enregistrer l'état quand une interface est utilisée permet d'utiliser cet état pour « rejouer » une portion du système pour reproduire automatiquement la faute.
  - Cette tactique réfère tant à l'enregistrement de l'état qu'à l'utilisation de l'état pour tester.

Attribut de qualité 6: Tactiques pour la testabilité: **Contrôler et observer l'état**

- Regrouper la sauvegarde de l'état: pour permettre de démarrer un système, sous-système ou module dans un état arbitraire, il est très pratique que son état soit stocké à un seul endroit. Au contraire, si l'état est profondément enfouit ou distribué, cela devient très difficile ou impossible.
  - L'état peut être à grain fin (jusqu'au niveau des bits) ou à gros grain pour représenter des abstractions globales ou des modes d'opération.
  - Le choix de la granularité dépend de la façon dont l'enregistrement de l'état sera utilisé dans les tests.
  - Une façon pratique d'externaliser le stockage de l'état (rendre l'état manipulable via une interface) consiste à utiliser une machine à état comme mécanisme pour suivre et rendre compte de l'état courant.

Attribut de qualité 6: Tactiques pour la testabilité: **Contrôler et observer l'état**

- Abstraire les sources de données: comme pour l'état, contrôler facilement les données d'entrée facilite les tests. Rendre les interfaces abstraites permet de substituer plus facilement les données de test.
  - Par exemple, dans un système qui utilise une base de données de transactions client, l'architecture pourrait être conçue de façon à permettre de facilement utiliser une autre base de données ou même un fichier pour effectuer les tests, sans changer le code fontionnel.

Attribut de qualité 6: Tactiques pour la testabilité: **Contrôler et observer l'état**

- Carré de sable: isoler une instance du système pour permettre une expérimentation non contrainte par la nécessité de récupérer des conséquences de l'expérience.
  - Les tests sont facilités par la possibilité d'opérer le système sans que ça ait des conséquences permanentes ou qu'il faille garder la possibilité de revenir en arrière,
  - Permet l'analyse de scénarios, la formation ou la simulation.
- Une forme courante de carré de sable consiste à virtualiser des ressources. Pour tester un système, il faut souvent interagir avec des ressources externes au système. La méthode du carré de sable permet de mieux contrôler les ressources.

Attribut de qualité 6: Tactiques pour la testabilité: **Contrôler et observer l'état**

- Assertions exécutables: placer manuellement des assertions dans le code à des endroits stratégiques pour indiquer que le programme est en faute.
  - Vérifier que les valeurs des données satisfont des contraintes spécifiées.
  - Les assertions sont définies en termes de déclarations spécifiques de données et doivent être placées là où les données sont référencées ou modifiées.
  - Les assertions peuvent être spécifiées comme des pré ou postconditions pour une méthode ou comme invariants de classe.
  - Les assertions augmentent l'observabilité.
  - Les assertions sont une façon d'imbriquer l'oracle directement dans le code.

Attribut de qualité 6: Tactiques pour la testabilité: **Contrôler et observer l'état**

Toutes ces tactiques ajoutent des capacités ou des abstractions au logiciel qui ne seraient pas présentes si on ne voulais pas le tester. Ces ajouts peuvent se faire grâce à différents mécanismes:

- Remplacement de composante, qui échange une composante par une version instrumentée pour faciliter les tests. Généralement au moment de la construction.
- Macros du préprocesseur pour activer des portions de code ou des moniteurs qui fournissent de l'information ou retourne le contrôle à une console de test.
- Options de compilation qui activent les assertions.
- Aspects qui se chargent de rapporter l'état.

Attribut de qualité 6: Tactiques pour la testabilité: **Limiter la complexité**

- Limiter la complexité structurelle: éviter ou éliminer les dépendances cycliques entre les composantes, isoler ou encapsuler les dépendances envers l'environnement externe et réduire les dépendances entre les composantes en général.
- Au niveau des classes, on peut aussi:
  - Simplifier la hiérarchie d'héritage en limitant le nombre de classes dont dérive une classe ou le nombre de classes dérivées d'une classe,
  - Limiter la profondeur de l'arbre d'héritage,
  - Limiter le polymorphisme et les appels dynamiques
  - Limiter la métrique de « réponse » d'une classe (nombre de méthode de la classe et de d'autres classes appelées par les méthodes de la classe) augmente la testabilité.

Attribut de qualité 6: Tactiques pour la testabilité: **Limiter la complexité**

- Limiter le non-déterminisme: la contrepartie de limiter la complexité structurelle est de limiter la complexité comportementale. Du point de vue des tests, le non-déterminisme est une forme pernicieuse de complexité comportementale. Trouver et illimiter les sources de non-déterminisme augmente la testabilité:
  - Parallélisme non contraint,
  - Dépendances à la position en mémoire,
  - Approches stochastiques de planification des tâches.

Liste de vérification pour la testabilité
Allocation des responsabilités

- Déterminer quelles responsabilités du système sont les plus critiques et doivent être testées à fond.
- S'assurer que des responsabilités ont été ajoutées pour:
  - Exécuter les suites de tests et capturer les résultats,
  - Enregistrer la/les activité(s) qui ont résulté en une faute ou en un comportement inattendu, qui n'est pas nécessairement une faute,
  - Contrôler les aspects pertinents de l'état du système pour le tester.
- S'assurer que l'allocation des fonctionnalités engendre une haute cohésion, un faible couplage, une grande séparation des responsabilités et une faible complexité structurelle.

Liste de vérification pour la testabilité
Modèle de coordination

- S’assurer que les mécanismes de coordination et de communication :
  - Supportent l’exécution d’une suite de test et permettent de capturer les résultats dans le système ou entre systèmes,
  - Supportent l’enregistrement des activités qui ont résulté en une faute dans le système ou entre systèmes,
  - Supportent l’injection ou le monitoring d’état sur les canaux de communication utilisés pour les tests,
  - N’introduisent pas de non-déterminisme inutile.

Liste de vérification pour la testabilité
Modèle de données

- Déterminer les principales abstractions de données qui doivent être testées afin d'assurer les opérations correctes du système.
  - S'assurer qu'il est possible de capturer les valeurs des instances de ces abstractions,
  - S'assurer que les valeurs des instances des abstractions de données peuvent être initialisées lorsqu'un état est injecté dans le système afin de reproduire les circonstances ayant menées à une faute,
  - S'assurer que la création, l'initialisation, la persistance, la manipulation, la traduction et la destruction de ces instances des abstractions de données peuvent être invoquées et enregistrées.

Liste de vérification pour la testabilité
Correspondance entre les éléments architecturaux

- Déterminer comment tester différentes possibilités de correspondances entre les éléments architecturaux, dont:
  - La correspondance des processus sur les processeurs,
  - La correspondance des fils d'exécution sur les processus,
  - La correspondance des modules aux composantes,

  afin d'obtenir la réponse désirée aux tests et d'identifier et d'éliminer les conditions potentielles de compétition.
- Déterminer s'il est possible de tester les correspondances illégales entre les éléments architecturaux.

Liste de vérification pour la testabilité
Gestion des ressources

- S’assurer qu’il y a suffisamment de ressources disponibles pour exécuter les suites de tests.
- S’assurer que l’environnement de test est représentatif de l’environnement d’exécution.
- S’assurer que le système fournit les moyens de:
  - Tester les limites des ressources,
  - Enregistrer les détails d’utilisation des ressources pour fins d’analyse en cas de faute,
  - Injecter des nouvelles limites de ressources dans le système à des fin de tests,
  - Fournit des ressources virtualisées pour les tests.

Liste de vérification pour la testabilité
Choix du moment pour effectuer une liaison

- S’assurer que les composantes qui sont liées plus tard qu’à la compilation peuvent être testées dans un contexte de liaison tardive.
- S’assurer que les liaisons tardives peuvent être enregistrées au cas où il se produirait une défaillance, afin de recréer l’état du système qui a mené à la défaillance.
- S’assurer que l’ensemble de toutes les liaisons possibles puisse être testé.

Liste de vérification pour la testabilité
Choix de technologie

- Déterminer quelles technologies sont disponibles pour aider à implémenter les scénarios de testabilité applicable à l'architecture.
- Des technologies sont-elles disponibles qui aident à effectuer des tests de régression, de l'injection de faute, de l'enregistrement et restitutions, etc.
- Déterminer combien testables sont les technologies choisies ou considérées et s'assurer que les technologies choisies supportent le niveau de testabilité requis par le système.

Attribut de qualité 6: Testabilité
Deux défis: **1 – données de tests**

- Créer des ensembles de données de tests consistants et utiles peut représenter un défi considérable.
- Les données pour les tests unitaires et les tests d'acceptation sont souvent générées à la main.
- Générer des données pour les tests d'intégration, permettant de s'assurer que plusieurs composantes fonctionnent correctement ensemble, et les tests de performance, est particulièrement difficile.

Attribut de qualité 6: Testabilité
Deux défis: **1 – données de tests**

- Deux approches pour créer des ensembles de données:
  - Concevoir un système de génération de données
    - Requiert des efforts considérables,
    - Permet de couvrir tous les cas limites.
  - Enregistrer les données d'exécution dans un environnement de production
    - Plus simple que de générer les données à la main,
    - Nécessite de vérifier la couverture de l'ensemble du système et des cas limites,
    - Requiert souvent un processus d'anonymisation des données,
    - Peut être contraint par des enjeux de confidentialité et de législation.
- Peut avoir un impact important sur l'architecture pour enregistrer les transactions en cours d'exécution ou sauver des captures de données.

Attribut de qualité 6: Testabilité
Deux défis: **2 – automatisation des tests**

- Mise en place d'un environnement de tests automatiques
  - Exécution quotidienne (chaque nuit)
  - Exécution à chaque changement (intégration continue)
- Souvent négligé dans la planification de grands projets
  - Pas de budget et de ressources spécifiques pour mettre en place l'infrastructure de test,
  - Peut être une infrastructure complexe en soit – qui nécessite des tests…
- Tester les interfaces usager est souvent très complexe et fortement sujet à changement.
- Recours à des outils spécialisés.

Attribut de qualité 7: Facilité d'utilisation

La facilité d'utilisation réfère à combien il est facile pour un utilisateur d'accomplir une tâche désirée et le niveau de support offert par le système.

Améliorer la facilité d'utilisation a été identifié comme l'un des moyens les plus efficaces d'améliorer la qualité d'un système – ou du moins la perception des utilisateurs de la qualité d'un système.

Attribut de qualité 7: Facilité d'utilisation

La facilité d'utilisation comprend les aspects suivants:

- **Apprendre les caractéristiques du système**. Pour un utilisateur qui n'est pas familier avec un système ou certains aspects particuliers, comment faire pour que le système facilite l'apprentissage ? Ceci peut inclure des mécanismes d'aide.
- **Utiliser le système efficacement**. Comment faire pour que l'utilisateur soit plus efficace dans ses opérations ? Ceci peut inclure la possibilité pour l'usager de rediriger le système après avoir émis une commande. P.ex. suspendre une tâche, effectuer plusieurs opérations, puis reprendre une tâche.
- **Minimiser l'impact des erreurs**. Comment faire pour qu'une erreur de l'utilisateur ait un impact minimal ? P.ex. permettre d'annuler une commande incorrecte.

Attribut de qualité 7: Facilité d'utilisation

La facilité d'utilisation comprend les aspects suivants:

- **Adapter le système aux besoins des utilisateurs**. Comment les système peut-il s'adapter pour rendre les tâches de l'utilisateur plus facile ? P.ex. en fournissant des valeurs de défaut apprises dynamiquement comme choix de réponses.
- **Accroitre la confiance et la satisfaction**. Que fait le système pour confirmer à l'utilisateur que la bonne action est en train de se produire. P.ex. en fournissant une rétroaction qui indique qu'une tâche longue est en cours et en indiquant le progrès accompli.

Scénario général de facilité d'utilisation

Source du stimulus

L'utilisateur final, possiblement dans un rôle spécialisé.

Stimulus

L'utilisateur essaie d'utiliser un système de façon efficace, apprendre à utiliser le système, minimiser l'impact des erreurs, adapter ou configurer le système.

Artéfact

Le système ou la portion spécifique du système avec laquelle interagit l'utilisateur.

Environnement

En cours d'exécution ou de configuration.

Réponse

Le système devrait fournir à l'utilisateur les caractéristiques dont il a besoin ou anticiper les besoins de l'utilisateur.

Mesure de la réponse

Un ou plusieurs parmi les suivants:

- Temps pour effectuer une tâche,
- Nombre d'erreurs,
- Nombre de tâches accomplies,
- Satisfaction de l'utilisateur,
- Accroissement des connaissances de l'utilisateur,
- Ratio d'opérations réussies sur le nombre total d'opération,
- Temps perdu ou données perdues en cas d'erreur.

Exemple de scénario concret de facilité d'utilisation

Source du stimulus

Utilisateur

Stimulus

Télécharge une nouvelle application

Artéfact

Système

Environnement

En cours d'exécution

Réponse

L'utilisateur utilise l'application de façon productive

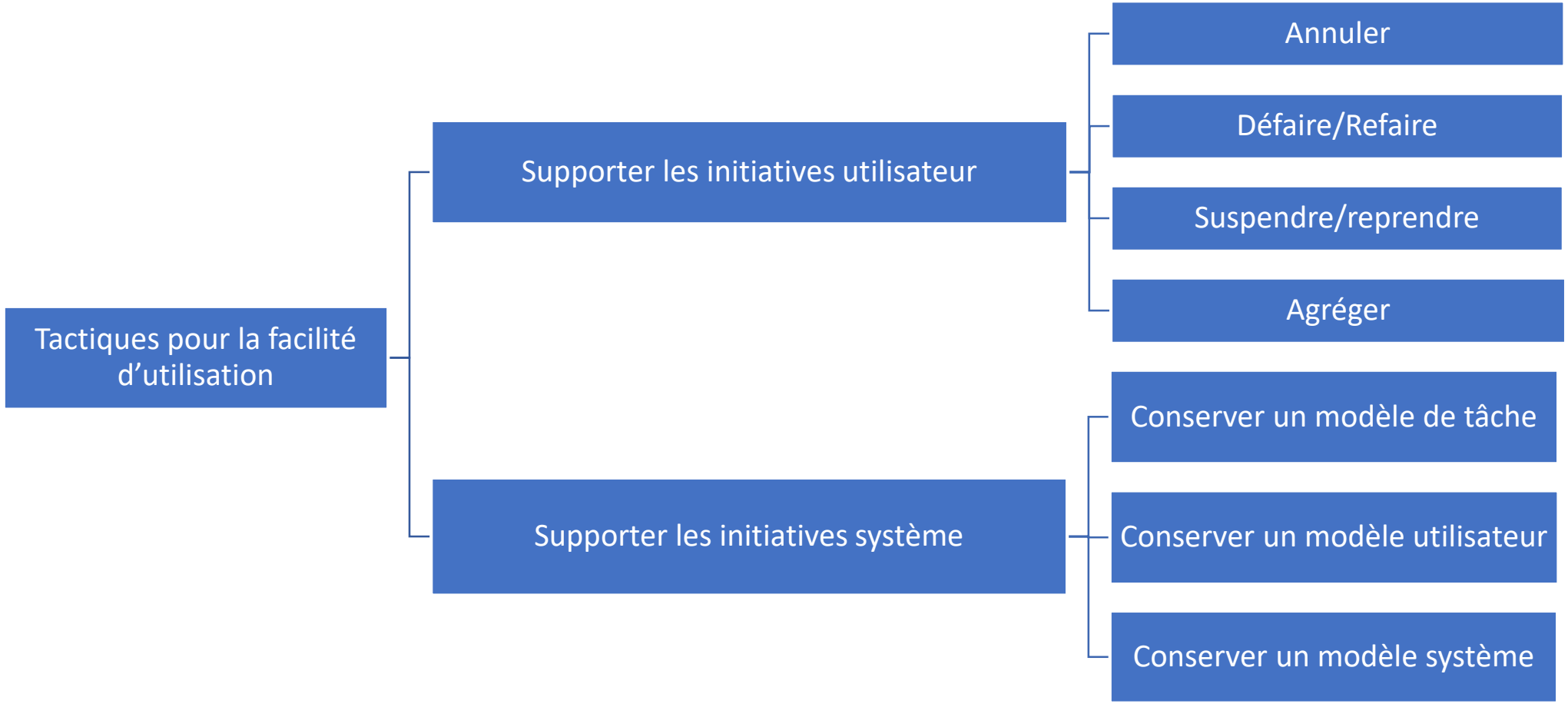
Mesure de la réponse

En moins de 2 minutes d'expérimentation

Attribut de qualité 7:
Tactiques pour la facilité d'utilisation

Les tactiques pour la facilité d'utilisation visent à rendre le système plus facile à utiliser efficacement et à apprendre.

Attribut de qualité 7:
Tactiques pour la facilité d'utilisation



Attribut de qualité 7: Tactiques pour la facilité d'utilisation: **Supporter les initiatives utilisateur**

- Annuler: lorsque l'utilisateur émet une commande d'annulation, le système doit être en train d'écouter pour pouvoir agir immédiatement:
  - Une composante du système doit avoir la responsabilité de constamment écouter les évènements de l'utilisateur et ne doit pas bloquer, quelque soit la commande à annuler,
  - La commande qui est annulée doit être interrompue immédiatement et les ressources utilisées par la commande doivent être libérées,
  - Les composantes qui collaboraient avec la commande annulée doivent être averties afin qu'elles prennent les actions appropriées.

Attribut de qualité 7: Tactiques pour la facilité d'utilisation: **Supporter les initiatives utilisateur**

- Défaire/Refaire: pour permettre de défaire ou refaire une commande, le système doit conserver suffisamment d'information à propos de son état antérieur pour être en mesure de le restaurer.
  - L'enregistrement de l'état peut être sous la forme d'une capture, d'un point de vérification ou d'une série d'opérations réversibles.
  - Ce ne sont pas toutes les opérations qui sont réversibles. Pour les opérations non-réversibles, le système doit enregistrer plus d'information à propos des changements effectués.
  - Certaines opérations ne peuvent tout simplement pas être défaites. P.ex. émettre un son.

Attribut de qualité 7: Tactiques pour la facilité d'utilisation: **Supporter les initiatives utilisateur**

- Suspendre/reprendre: lorsqu'un usager déclenche une opération longue – calcul, téléchargement, etc. – il est souvent utile de lui permettre de suspendre et de reprendre l'opération.

  → Implique la capacité de libérer temporairement des ressources qui pourront être réallouées à d'autre tâches.

Attribut de qualité 7: Tactiques pour la facilité d'utilisation: **Supporter les initiatives utilisateur**

- Agréger: lorsqu'un usager effectue des tâches répétitives ou qui affectent un grand nombre d'objets de la même façon, il est utile de fournir un mécanisme pour rassembler les objets de plus bas niveau dans un groupe et d'appliquer l'opération au groupe.

Attribut de qualité 7: Tactiques pour la facilité d'utilisation: **Supporter les initiatives système**

- Conserver un modèle de tâche: un modèle de tâche est utilisé pour déterminer un contexte de façon à ce que le système ait une idée de ce que l'utilisateur tente d'accomplir et puisse lui fournir de l'aide.
  - P.ex. correction automatique dans les systèmes de composition.

Attribut de qualité 7: Tactiques pour la facilité d'utilisation: **Supporter les initiatives système**

- Conserver un modèle utilisateur: ce modèle représente explicitement les connaissances de l'utilisateur à propos du système, le comportement de l'utilisateur en terme de réponse attendue et d'autres aspects spécifique d'un utilisateur spécifique ou d'une classe d'utilisateurs.
  - P.ex. le système peut modérer les mouvements de la souris durant une sélection pour permettre une sélection précise d'une section de texte.
  - Le modèle peut contrôler la quantité d'aide ou les suggestions offertes à un utilisateur.
  - Un cas particulier de cette tactique est de fournir à l'utilisateur la possibilité de personnaliser l'interface du système, ce qui agit directement sur le modèle utilisateur.

Attribut de qualité 7: Tactiques pour la facilité d'utilisation: **Supporter les initiatives système**

- Conserver un modèle système: dans ce cas, le système conserve un modèle explicite de lui-même, afin de prédire son propre comportement et fournir une rétroaction appropriée à l'utilisateur.
  → P.ex. fournir un indicateur de progrès indiquant le temps restant avant de compléter une opération.

Liste de vérification pour la facilité d'utilisation
Allocation des responsabilités

- S'assurer que des responsabilités additionnelles ont été allouées, au besoin, afin d'assister l'utilisateur à:
  - Apprendre comment utiliser le système,
  - Réaliser efficacement une tâche,
  - Adapter et configurer le système,
  - Récupérer d'erreurs utilisateurs et système.

Liste de vérification pour la facilité d'utilisation
Modèle de coordination

- Déterminer si les propriétés de coordination des éléments du système telles que:
  - la ponctualité, l'actualité, la complétude, l'exactitude, et la consistance

  affectent la façon qu'a un usager:
  - D'apprendre à utiliser le système,
  - D'atteindre ses objectifs ou de compléter une tâche,
  - D'adapter ou de configurer le système,
  - De récupérer d'erreurs utilisateur ou système,
  - D'acquérir plus de confiance et de satisfaction.
- Par exemple:
  - Est-ce que le système peut répondre aux évènements de souris et fournir une rétroaction en temps réel ?
  - Est-ce qu'une tâche nécessitant un long temps d'exécution peut être interrompue dans un délais raisonnable ?

Liste de vérification pour la facilité d'utilisation
Modèle de données

- Déterminer les principales abstractions de données qui sont impliquées dans des comportements perceptibles par les utilisateurs.
- S'assurer que ces abstractions, leurs opérations et leurs propriétés ont été conçues de façon à assister l'utilisateur à:
  - Accomplir ses tâches,
  - Adapter et configurer le système,
  - Récupérer d'erreur utilisateur ou système,
  - Acquérir plus de confiance et de satisfaction.
- Par exemple:
  - Les abstractions devraient être conçues pour permettre de défaire/refaire une opération,
  - La granularité des transactions ne devrait pas être si fine qu'il soit extrêmement long de défaire une opération.

Liste de vérification pour la facilité d'utilisation
Correspondance entre les éléments architecturaux

- Déterminer quelles correspondances entre les éléments architecturaux sont visibles pour l'utilisateur. Par exemple:
  - Vérifier dans quelle mesure l'utilisateur est conscient des services qui sont locaux et de ceux qui sont distants.
- Pour les correspondances visibles des utilisateurs, déterminer de quelle façon ou avec quelle facilité l'utilisateur peut:
  - Apprendre à utiliser le système,
  - Atteindre ses objectifs ou de compléter une tâche,
  - Adapter ou de configurer le système,
  - Récupérer d'erreurs utilisateur ou système,
  - Acquérir plus de confiance et de satisfaction.

Liste de vérification pour la facilité d'utilisation
Gestion des ressources

- Déterminer comment un utilisateur peut adapter et configurer l'utilisation des ressources par le système.
- S'assurer que les limites sur les ressources pour toutes les configurations contrôlées par l'utilisateur ne vont pas rendre moins probable l'atteinte de ses objectifs par l'utilisateur. Par exemple:
  - Tenter d'éviter les configurations qui entraîneraient des temps de réponse excessifs.
- S'assurer que le niveau de ressources n'affectera pas la capacité d'un utilisateur d'apprendre à utiliser le système ou ne diminuera pas son niveau de confiance ou de satisfaction envers le système.

Liste de vérification pour la facilité d'utilisation
Choix du moment pour effectuer une liaison

- Déterminer quelles décisions de liaison devraient être contrôlées par l'utilisateur et s'assurer que les utilisateurs peuvent prendre des décisions qui contribuent à la facilité d'utilisation.
- Par exemple, si l'utilisateur peut choisir, à l'exécution, la configuration du système, ses protocoles de communication ou sa fonctionnalité grâce à des plugins, s'assurer que ces choix n'affectent pas négativement la capacité de l'utilisateur à :
  - Apprendre à utiliser le système,
  - Atteindre ses objectifs ou de compléter une tâche,
  - Adapter ou de configurer le système,
  - Récupérer d'erreurs utilisateur ou système,
  - Acquérir plus de confiance et de satisfaction.

Liste de vérification pour la facilité d'utilisation
Choix de technologie

- S'assurer que les technologies choisies contribuent à réaliser les scénarios de facilité d'utilisation qui s'appliquent au système. Par exemple, ces technologies aident-elles dans la création d'aide en ligne, à la production de matériel de formation et à la récolte de rétroaction des utilisateurs ?
- Combien les technologies choisies sont-elles faciles à utiliser ? S'assurer que les technologies choisies n'affectent pas négativement la facilité d'utilisation du système.

Autres attributs de qualité

En plus des attributs déjà présentés, plusieurs autres attributs de qualité peuvent devenir importants selon les caractéristiques spécifiques d'un système. Parmi ceux-ci, on retrouve souvent:

- La variabilité,
- La portabilité,
- La distributivité du développement,
- La capacité d'être mis à l'échelle,
- La facilité de déploiement,
- La mobilité,
- La capacité à être monitoré,
- La sureté.

La variabilité

- Une forme particulière de modifiabilité qui vise à supporter un ensemble de variantes d'un système, qui diffèrent entre-elles de façon planifiée.
- Cette qualité est particulièrement importante dans le contexte d'une ligne de produits qui doivent être supportés et dont l'utilisation doit être adaptée à chaque type de produit.
- Le but de la variabilité est de rendre facile la construction et la maintenance des produits dans la ligne de produit pour une période de temps donné.
- Les scénarios de variabilité analyse les moments où les liaisons sont effectuées, et les acteurs qui effectuent ces liaisons.

La portabilité

- Aussi une forme particulière de modifiabilité qui réfère à la facilité avec laquelle un système construit pour une plateforme donnée peut être changé pour s'exécuter sur une plateforme différente.
- La portabilité est obtenue en minimisant les dépendances à la plateforme dans le logiciel, en isolant les dépendances dans des endroits bien identifiés, et en écrivant le logiciel pour qu'il s'exécute sur une machine virtuelle (p.ex. JVM) qui encapsule les dépendances à la plateforme.
- Les scénarios de portabilité décrivent la migration d'un logiciel vers une nouvelle plateforme en investissant un certain niveau d'effort maximum ou en comptant le nombre d'endroits dans le logiciel qui devraient être modifiés.

La distributivité du développement

- La distributivité réfère à la qualité d'une architecture logicielle de supporter un développement par des équipes distribuées.
- Un problème à résoudre concerne la coordination des activités entre les équipes, dont le besoin doit être minimisé.
- Cette minimisation de la coordination doit être réalisée tant pour le code que pour le modèle de données.
- Implique des négociations entre les équipes de développement sur les interfaces des modules qui doivent communiquer entre eux.
- Les scénarios de distributivité analysent la compatibilité des structures de communication et des modèles de données du système à développer et les mécanismes de coordination des organisations impliquées.

La capacité d'être mis à l'échelle

- Deux types de mise à l'échelle:
  1. Mise à l'échelle horizontale, où des nouvelles ressources sont ajoutées aux unités logiques, p.ex. un nouveau serveur dans une grappe.
  2. Mise à l'échelle verticale, où des nouvelles ressources sont ajoutées aux unités physiques, p.ex. ajout de mémoire dans un ordinateur donné.
- Jusqu'à quel point l'architecture du système permet d'utiliser efficacement les nouvelles ressources. Est-ce qu'une amélioration mesurable peut être observée, sans efforts démesurés ni interruption inacceptable de service?
- Dans les environnements d'infonuagique, la capacité de mise à l'échelle horizontale est appelée Élasticité.

La facilité de déploiement

- Comment un exécutable arrive-t-il sur un ordinateur hôte et comment est-il invoqué ?
- Parmi les questions à examiner:
  - Comment les mises à jour arrivent-elles ? Mode poussé, sans intervention de l'usager, ou mode tiré, où l'usager déclenche la mise à jour.
  - Comment les mises à jour sont-elles intégrées ? Est-il possible d'intégrer les mises à jour pendant que le système existant s'exécute ?
  - Comment la bande passante est-elle gérée sur les appareils mobiles ?
- Les scénarios de déploiement analysent les types de mises à jour, leur format, l'intégration dans le système existant, l'efficacité du processus et les risques associés.

La mobilité

- La mobilité examine les problèmes liés aux déplacements et aux accommodements nécessaires au support d'une plateforme:

  → taille, type d'écran, type de dispositif d'entrée, disponibilité et largeur de la bande passante, durée de vie des batteries.

- Requiert une capacité à se reconnecter en cas d'interruption des communications et différentes interfaces utilisateur pour supporter des plateformes multiples.
- Les scénarios examinent les accommodements nécessaires pour supporter le mouvement et la variabilité des plateformes.

La capacité à être monitoré

- Capacité pour le personnel de soutien de suivre les opérations d'un système durant son exécution.
- Différents types d'informations pertinentes:
  - La longueur et le nombre d'évènements dans des queues,
  - Le temps moyen de traitement des transactions,
  - L'état des différentes composantes du système.
- Les scénarios de monitoring analysent les problèmes potentiels, leur visibilité pour le personnel de soutien et les actions correctives potentielles.

La sureté

- La sureté logicielle réfère à la capacité d'un système d'éviter d'atteindre un état qui cause ou peut mener à des dommages, des blessures ou des pertes de vies, et à sa capacité à récupérer et limiter les dommages lorsqu'il entre dans un mauvais état.
- La sureté se préoccupe de la prévention et de la récupération en cas de faute. La sureté emprunte donc beaucoup de ses scénarios et de ses tactiques à la disponibilité, à laquelle elle est très liée.
- La sureté ne doit pas être confondue avec la fiabilité. Un système peut être fiable s'il respecte ses spécifications, mais non sûr, si ses spécifications lui permette d'effectuer des opérations dangereuses.