Fiche révisions spécification et conception

Interaction homme-machine :

-Domaine de l'IHM : Étude des phénomènes cognitifs, matériels, logiciels, sociaux mis en jeu dans l'accomplissement de tâches avec un système informatique ;
-Objectifs de l'IHM :
–Concevoir, réaliser des systèmes
–Utiles
–Utilisables
–En contexte

Plusieurs styles d'interaction :
-ligne de commandes (ou conversationnel) : impose une syntaxe précise et le dialogue est imposé par le système ;
-menus, formulaires : le système guide l'utilisateur mais le dialogue reste contrôlé par le système ;
-Navigation : nœuds, ancres et liens ;
-manipulation directe : actions physiques « directes » sur la représentation des objets

Constats sur la conception d'interface homme-machine :
-difficile, long, coûteux ;
-sacrifice de l'utilisabilité ;
-rejet des utilisateurs et concurrence sur le marché (exemple : Windows vs Mac OS)

Pourquoi s'intéresser à l'IHM ? : Parce que tout le monde n'a pas les mêmes capacités.

Conception logicielle mal reconnue et mal appréciée, pas de place dans l'organigramme, considérée comme une sous-partie de l'informatique (on s'intéresse plus à la construction interne qu'à la construction externe).

Psychologie cognitive :

Modèle pour prédire et expliquer le comportement humain.

Modèle du processeur humain : trois sous-systèmes interdépendants qui comprennent chacun une mémoire (capacité, persistance) et et un processeur (cycle de base) d'environ 100 ms :
-système sensoriel : Ensemble des sous-systèmes spécialisés chacun dans le traitement d'une classe de stimuli (phénomène physique détectable) ;
-système moteur : responsable des mouvements (qui sont eux-mêmes des ensembles de micro-mouvements) ;
-système cognitif : mémoire à court et à long terme et processeur cognitif.

Modèle ne fournissant pas des informations intéressantes concernant la conception d'interface homme-machine et ne traitant pas les problèmes de l'erreur et du parallélisme. De plus, ne présente pas de méthode de conception.

Théorie de l'action de Norman sur la réalisation d'une tâche :
1 Etablissement du but
2 Formation d'une intention
3 Spécification d'une suite d'actions
4 Exécution des actions
5 Perception de l'état du système
6 Interprétation de l'état du système
7 Evaluation de l'état par rapport au but fixé

Objectif du concepteur et du réalisateur : réduire les distances mentales par le biais de l'image du système.

Deux distances :
-distance d'exécution : effort cognitif de l'utilisateur pour la mise en correspondance entre la représentation mentale de sa tâche et la représentation physique de l'image du système ;
-distance d'évaluation : effort cognitif inverse.

Modèle perceptuel : modèle mental construit par l'utilisateur grâce aux affordances et aux liens de causalité qu'il perçoit, aux contraintes imposées par le système mais aussi aux correspondances perçues et aux stéréotypes culturels en plus de l'expérience de systèmes similaires et d'instructions reçues.

Modèle conceptuel : description et fonctionnement du système

La distance entre ces deux modèles détermine la performance (utilisabilité) du système.

Une affordance est « a propertyof the world thataffordsaction to appropriatelyequippedindividuals ». Exemples : une chaise est faite pour s'asseoir, une porte peut-être poussée ou tirée, etc...

Le développeur du système doit penser à la logique des utilisateurs et non attendre d'eux qu'ils agissent comme il voudrait. Comme il est toutefois difficile de concevoir un système qui convienne à tout le monde, le compromis est de viser 80 % de la population.

Ergonomie :

Adaptation des conditions de travail à l'homme. Processus de conception d'outils, de machines, etc...

Homogénéité : cohérence globale de l'interface (système famillier perçu comme simple à utiliser), ce qui présente comme inconvénient de freiner ou bloquer l'évolution des standards.

Atteignabilité : capacité du système à permettre à l'utilisateur de naviguer dans l'ensemble des états observables du système. Ainsi, un état q est atteignable à partir d'un état p si une suite de commandes permettre de passer de l'état p à l'état q.

Curabilité : capacité pour l'utilisateur de corriger une situation non désirée, capacité de défaire

Adaptabilité : capacité du système à s'adapter sur intervention explicite de l'utilisateur.

Adaptativité : capacité du système à s'adapter sans intervention explicite de l'utilisateur

Plasticité : capacité du système à s'adapter au contexte d'interaction tout en préservant son utilisabilité

Migrabilité de tâches : capacité de délégation dynamique de tâches entre le système et l'utilisateur ou entre les utilisateurs.

Multimodalité : CARE :
-Complémentarité : plusieurs modalités distinctes sont nécessaires pour exprimer le but ;
-Assignation : une seule modalité est disponible pour exprimer le but ;
-Redondance : plusieurs modalités sont utilisables en même temps et expriment le même but ;
-Equivalence : plusieurs modalités sont utilisables pour exprimer un même but mais on peut n'en utiliser qu'une seule à la fois.

Observabilité : capacité du système à rendre perceptible l'état pertinent du système. Capacité pour l'utilisateur à évaluer l'état actuel du système

Réciprocité : capacité d'observation/inspection mutuelle des variables d'état personnelles

Réflexivité : capacité d'inspecter ou d'observer les variables d'état personnelles publiées par autrui.

Insistance : capacité du système à forcer la perception de l'état du système

Honnêteté : capacité du système à rendre observable l'état du système sous une forme conforme à cet état et qui engendre une interprétation correcte de la part de l'utilisateur.

Prévisibilité : capacité pour l'utilisateur de prévoir, pour un état donné, l'effet d'une action.

Prévisibilité et guidage : inciter à ne pas commettre d'erreurs en grisant certains champs, regrouper des informations de même type ensemble, etc... A chaque retour utilisateur, l'interface doit changer.

Viscosité : l'action de l'utilisateur a un effet sur son plan de tâches ou, en collecticiel, sur celui des autres.

Compatibilité : capacité du système à s'intégrer dans les activités des utilisateurs (exemple : parler la langue de l'utilisateur)

Flexibilité : capacité de l'interface à s'adapter à différents contextes d'utilisation

Concision : ensemble de moyens visant à réduire la charge perceptive et mnésique de l'utilisateur (se baser sur des icônes qui rappellent le comportement à l'utilisateur comme la disquette pour enregistrer).

Traitement des erreurs : messages d'erreurs clairs, concis et compréhensible par la majorité des utilisateurs.

Conception centrée utilisateur :

Utilisabilité : processus itératif : concevoir, implémenter, évaluer

Modèle en cascade non adapté à la conception d'interface homme-machine car on se trompe (c'est sûr), et du coup, cela revient cher. Le modèle en spirale est plus adéquat.

La conception itérative permet une meilleure interface utilisateur (plus il y a d'itérations et mieux c'est). Seule la dernière sera dévoilée au public. La première est prototypique et s'appuie sur des supports papiers alors que les suivantes sont de plus en plus fidèles et de plus en plus riches.

Conception centrée utilisateur : conception itérative où les utilisateurs sont impliqués à chaque itération (en tant qu'utilisateur, voire en tant que concepteurs). L'analyse des utilisateurs, des tâches et du domaine constitue le pilier de cette conception.

Analyse des utilisateurs : qui sont-ils ? (identifier les caractéristiques de la population cible (âge, sexe, culture, langue, niveau d'éducation, limitations physiques, expérience de l'informatique, motivation, attitude, expérience du domaine, expérience de la tâche, environnement et contexte social, relation entre utilisateurs, communication, etc...)).

Plusieurs classes d'utilisateur, chacune d'elles pouvant être représentée par un persona qui reprend les caractéristique de cette classe.

Dans cette analyse des utilisateurs, décrire ce qu'ils sont et non ce qu'ils devraient être.

Analyse des tâches : que doivent faire les utilisateurs ? (pas comment) : détermination du but, des préconditions (ce qui doit être fait avant) et des sous-tâches pour arriver au but qui doivent être analysées récursivement. Déterminer où la tâche doit être effectuée ainsi que sa fréquence de répétition, son environnement, les contraintes liées aux ressources et au temps, l'apprentissage de la tâche, ce qui peut mal se passer et les autres utilisateurs éventuellement impliqués.

Dans cette analyse, penser du point de vue de l'utilisateur et non du point de vue du système.

Analyse du domaine : quel est le contexte dans lequel le(s) utilisateur(s) travaille(nt) ? (identifier les différentes tâches que le système informatique pourrait éventuellement résoudre, découper ces tâches en sous-tâches, etc...). Identifier les humains ainsi que les objets (physiques et/ou non physiques) impliqués ainsi que les relations qu'ils entretiennent entre eux. Ensuite, identifier les classes d'utilisateurs et/ou les tâches éventuellement manquantes.

Analyse des exigences : quelles sont les conséquences des trois analyses précédentes sur la conception ? Qu'est-ce que le système informatique devrait faire ?

Techniques de conception :

Dessiner des croquis pour faire émerger plein d'idées, en travaillant de préférence en groupe (soit sur papier soit avec un logiciel).

Un scénario est une histoire qui parle d'un utilisateur utilisant le système. Il suit l'utilisateur et comment il atteint son but.

Ces scénarios sont représentés grâce à une suite de croquis appelée storyboard.

Modèle de conception :

Appelé communément design pattern (ou patron de conception)

Une bonne solution à un problème courant qui peut être réutilisable mais qui présente l'inconvénient d'être incomplète. Ils servent beaucoup pour la conception de systèmes interactifs (formulaires contenant des boutons, des boîtes de dialogue, des champs de saisie, des cases à cocher, des boutons radio, etc...).

Evaluation :

Trois évaluations dans les tests utilisateurs :
-l'évaluation formative : trouver des problèmes pour l'itération suivante, évaluation de prototypes ou d'implémentations complètes, environnement contrôlé, données fictives, méthodes qualitatives (problèmes d'utilisabilité). Durant l'évaluation formative, prendre quelques utilisateurs représentatifs de chacune des classes d'utilisateurs. Durant cette phase, il doit y avoir un utilisateur à qui une ou plusieurs tâches ont été confiées, un facilitateur et des observateurs. Le facilitateur doit encourager l'utilisateur à penser à haute voix, contrôler la progression du test, donner les informations du test et les tâches. Les observateurs, quant à eux, doivent prendre des notes, et particulièrement les les incidents critiques négatifs. Ils ne doivent parler sous aucun prétexte ;
-l'évaluation sur le terrain : trouver les problèmes en contexte, évaluation d'une implémentation complète, dans un contexte réel avec des tâches réelles, méthode plutôt qualitative ;
-l'expérience contrôlée : tester une hypothèse, évaluation d'une implémentation fonctionnelle de la partie à évaluer, environnement contrôlé, données fictives, méthode quantitative (rapidité, taux d'erreur, satisfaction des utilisateurs, etc...).

Ne pas oublier que les utilisateurs sont des êtres humains. Pendant les tests, ils ne doivent pas se sentir idiots ou se comparer avec d'autres ou se mettre en compétition avec eux. On teste un système et non un utilisateur. L'utilisateur doit avoir le contrôle sur le test (partir quand il veut, arrêter une tâche pour passer à une autre, etc...), être mis à l'aise, être informé sur le test et qu'il consente. Enfin, sa vie privée doit être préservée. Par ailleurs, les résultats du test doivent être anonymisés.

Pendant le test, les recruteurs ne doivent pas avoir l'air déçu et ne doivent faire effectuer qu'une seule tâche à la fois aux testeurs. Les tâches doivent être de difficultés graduelles.

A la fin du test, expliquer aux testeurs à quoi servait le test et en quoi ils ont aidé.

Deux manières pour tester plusieurs versions d'un logiciel :
-plan emboîté (between subjects) : on divise les utilisateurs en groupes. Chaque groupe teste une seule version d'un logiciel ;
-plan croisé (within subjects) : tous les groupes testent toutes les verssions d'un logiciel.

Concernant le plan emboîté, il faut donc que l'utilisateur qui teste une version se retrouve toujours sur la même version quand il reprend le test.

Erreur standard à la moyenne : quand les deux barres se chevauchent, une hypothèse donnée est confirmée et n'est pas significative dans le cas contraire.

UML (Unified Modelling Language) :

Ensemble de diagrammes qui représentent tous les aspects d'un système informatique dans son environnement. C'est un langage à destination des développeurs, des décideurs et des utilisateurs. Ce langage est basé sur des modèles conceptuels.

Diagramme de cas d'utilisation : décrit les fonctionnalités du système à sa bordure. Chaque cas d'utilisation représente une unité discrète et indivisible d'interaction entre le système et l'environnement. Chaque cas d'utilisation comprend une description générale, les exigences, les contraintes (pré-conditions, post-conditions, invariants) et les scénarios. Les cas d'utilisation sont représentés par un ovale avec un nom.

Un acteur est un élément extérieur qui utilise ou interagit avec le système.

Include : un cas d'utilisation peut inclure la fonctionnalité d'un autre cas d'utilisation.

Extend : un cas d'utilisation peut étendre le comportement d'un autre cas d'utilisation.

Un acteur peut en généraliser un autre et les relations entre acteur et cas d'utilisation peuvent optionnellement avoir une multiplicité.

Diagramme de séquence : représentation graphique des interactions entre les objets du système dans le temps. On doit avoir un diagramme de séquence pour chacun des scénarios du diagramme de cas d'utilisation. Les noms des objets correspondent à des noms de classes du diagramme de classes et le nom des opérations correspond à des noms de méthodes de la classe.

Diagramme d'activité : permet de mettre l'accent sur les traitements. Ils permettent donc de représenter graphiquement le cheminement de flots de contrôle et de flots de données et donc le comportement d'une méthode.

Diagramme d'états-transitions : Il ne concerne qu'un seul objet à la fois. Il traduit tous les états de cet objet durant sa ligne de vie (de sa création à sa destruction).

Diagramme de classes : vision statique du système représentant toutes les classes (nom, attributs et méthodes).

Diagramme d'objets : Exemple de diagramme de classes avec de « vrais objets ».

Diagramme de déploiement : description physique plus ou moins détaillée du système (serveurs, capacité, réseau, etc...).