Section 1

Why Design fails & Design of Everyday Things

Mythe de l'erreur humaine

$\mathscr{H}$ Les *échecs* d'un système $\mathbb{PM}$ sont souvent dus au **design**. Pour $\downarrow$ erreur $\mathscr{H}$:

- ▷ Design qui tient compte des **limitations** et de la **fiabilité** des $\mathscr{H}$.

Principes de design

- ▷ Utilisabilité
- ▷ Expérience de l'Utilisateur (UX)
- ▷ **Psychopathologie** : frustratiosn courantes
- ▶ Permettent de **critiquer**, **analyser** et **convervoir** interfaces.

Causes d'échecs

- ▷ Fonctionnalité
  - ▶ $\mathscr{U}$ ne connait pas fonctions de l'Obj.
  - ▶ L'objet ne fait pas ce que $\mathscr{U}$ désire.
- ▷ Visibilité
  - ▶ $\mathscr{U}$ **voit** pas certaines infos l'Obj.
  - ▶ $\mathscr{U}$ ne sait pas quelle séquence de contrôle est nécessaire pour atteindre son but.
  - ▷ E.g. Lumières enfoncée pour passage piétons
- ▷ Feedback
  - ▶ Comment $\mathscr{U}$ sait si les opérations ont réussi ?
  - ▶ Comment $\mathscr{U}$ sait s'il y a une erreur en cours de route ?

Buts du UX | MAUSSEE

: Mémorabilité, Apprentissage, Utilité, Sécurité, Satisfaction, Efficience, Efficacité.

Définition de l'utilisabilité

Degré selon lequel un produit peut être utilisé par des $\mathscr{U}$ *identifiés*, pour atteindre des **buts** *définis* par l'**efficacité**, l'**efficience** et la **satisfaction**.

- ▷ **Efficacité** : atteindre le but ◎
- ▷ **Efficience** : *effort* et ou *temps* **minimal** ◷
- ▷ **Satisfaction** : évaluation subjective par $\mathscr{U}$

Note :

L'**utilisabilité** implique aussi la sécurité, l'apprentisage et la *mémorabilité*.

Où les desginer se trompent

- ▷ Ne comprennet pas $\mathscr{U}$ et leurs limitations
- ▷ Ne prévoient pas différents **contextes d'utilisation**
- ▷ Absence de **modèle détaillé** du fonct.
- ▷ Absence de **feedback** par l'objet.

Pourquoi le design est-il difficile

Les **interactions** sont complexes et difficile à définir. Par ailleurs, les tâches sont complexes et *implicites*. Il faut distribuer *raisonnablement* les tâches à la machine et à l'$\mathscr{U}$ pour éviter que l'un ou l'autre ne soit pas confronté à une complexité excessive.

Principe de découvrabilité

L'$\mathscr{U}$ doit savoir immédiatement à quoi l'objet sert, comment l'utiliser et quelles sont les opérations possibles.

- ▷ **Affordance** ce que l'O permet de faire. Un *signifiant* est un élément qui permet de rendre **l'affordance** visible.
- ▷ **Signifiants** indiquent que l'affordance $\exists$ et ne doivent pas être **contradictoire**.
- ▷ **Anti-affordance** permettent de masquer visibilité d'un aff. et contribue à la gestion d'erreur. Il s'agit d'une affordance qui est *délibérément supprimée*
- ▷ **Correspondance** Permet de faire l'association lors de l'utilisation (direction volume, mode `on/off`)
  - ▶ Soit une série de lumières alignées et des interrupteurs un à côté de l'autre, quel interrupteur alume quelle lumière.
- ▷ **Contraintes** sont des restrictions physiques ou logiques de l'objet ou l'interface qui contraignent l'$\mathscr{U}$à utiliser l'objet d'une certain façon. P. ex., orientation d'une *clé USB*.
- ▷ **Feedback** permet d'indiquer à l'$\mathscr{U}$ l'effet de son action ou d'une interaction.
- ▷ Modèle conceptuel est une explication très simplifiée du fonctionnement d'un élément.
  - ▶ **Modèle fonctionnel** : on sait *quoi* faire sans savoir *pourquoi* ça marche
  - ▶ **Modèle structurel** : on connait les composants et leurs interactions
  - ▷ Exemple : *Chinese mystery pot*

Les deux fossés d'interaction

La conception doit permettre de résoudre **2** ensembles de questions que l'$\mathscr{U}$ se pose lorsqu'il interagit :

- ▶ Comment ça marchee et *qu'est-ce que je peux faire avec l'objet ?*
- ▶ Qu'est-ce qui s'est passé et *est-ce que c'est ce que je voulais faire ?*

Les sept étape d'une actions

1. Définitir l'*objectif*, 2. Former l'*intention* 3. Spécifier la séquence d'actions, 4. Exécuter l'action 5. Percevoir l'état du système 6. Interpréter l'état du système 7. Évaluer l'état du syst. *par rapport aux intentions.*

Analyse de la cause originelle

Permet de déterminer si un *objectif* est réelle la fin souhaitée ou est simplement un sous-objectif d'un but à atteindre.

Septs questions garantissant l'atteinte de l'O

Un *bon design* implique qu'à tout moment, l'$\mathscr{U}$ parvient à répondre aux **sept questions** suivantes.

1. Que puis-je faire, 2. Quelles sont les alternatives 3. Comment puis-je le faire, 4. Le fais-je bien ? 5. Qu'est-ce que ça veut dire 6. Que s'est-il passé ?

Section 2

Design Centré sur $\mathscr{U}$

Risque du modèle en cascade Il permet de s'assurer que les implémentation sont *conformes aux engiences*, mais ne garantit pas qu'elles sont optimales pour $\mathscr{U}$. Par ailleurs, les problèmes sont parfois identifiés plus tard et revenir en arrière dans la modèle cascade peut être **couteux**

- ▷ Problèmes identifiés tard.
- ▷ Manque d'input et *feedback* de l'$\mathbb{U}$
- ▷ Problèmes → modif. exigences et conception.

Design itératif Étale le projet sur plusieurs petites itérations de *conception*, *prototypage* et *évaluation*.

Modèle en spirale Il utilise le principe de design itératif et réduit graduellement les risques à travers le *itérations*. Seules les itérations matures sont présentée à l'$\mathscr{U}$.

Design centré sur l'$\mathscr{U}$ Marque un changement de paradigme où l'opinion de l'$\mathscr{U}$ a précédence sur la *technologie* ou l'intuition du designer. On conçoit en fonction de ce que les $\mathscr{U}$ *doivent*, *peuvent* ou *veulent* faire.
▷ **Prédesign** pour comprendre le problème
▷ **Premier design** pour explorer l'espace de design
▷ **Mi-design** Développe l'approche choisie
▷ **Design avancé** lorsqu'on intègre et déploie L'évaluation par les $\mathscr{U}$ se fait de façon *continue* ; ils accompagnent les développement à chaque itération :

Design Il faut **1.** analyser les utilisateurs, les tâches *qu'ils cherchent à accomplir* ; il faut comprendre le problème et s'assurer que l'idée de solution est importante ou au moins *nécessaire* pour les $\mathscr{U}$. Il faut estimer le niveau d'expertise des $\mathscr{U}$. Il faut aussi **2.** suivre les principes de conceptions liés à l'*utilisabilité* Finalement, il faut **3.** assurer la prévention et gestion des erreurs.

Implémentations brouillons Elle peut être sur papier ou de style *Wizard of Oz* C'est *rapide*, *simple* et suffisamment *abstrait* pour se concentrer sur l'essentiel.

Évaluation Permet de relier la *progression de la conception* aux *besoin indentifiés* et aux contextes de l'utilisateur. L'évaluation peut être effectué *tôt* ou *tard*, selon le besoin.

Identifier les parties intéressées Cela dépend de plusieurs questions :

- Parties intéressées
  - qui l'utilisera?
  - qui décidera de l'utiliser?
  - qui va payer pour cela? .2 qui doit le faire (concevoir / construire)?
  - qui doit en tirer profit?
  - qui rendra votre vie misérable

Section 3

Évaluation heuristique

Types d'erreurs

► **Bévues** et **Lapsus** sonts causés par *manque reflexion*
▷ **Bévues** ⟶ échec d'exécution.
▷ **Lapsus** ⟶ faille de mémoire lors de l'exec.
▷ **Méprise** est un échec de mémoire et ou planif. P. ex., planifier procédures selon objectif, mais procédures utilsées ne mènent pas à l'objectif (méprise sur le choix de procédure).

- Type d'erreurs
  - Erreur d'*exécution*
    - Lapsus
      - Perte d'intention
      - Omission due à l'interruption
      - Omission due à la satisfaction
    - Bévues
      - De Capture
      - De Description
      - De Mode
  - Méprises

▷ **Erreur de capture** Survient lorsqu'un *comportement fréquent* capture notre attention et nos action. P. ex. vouloir taper `facial recognition` sur Google et finir par naviguer sur facebook.
▷ **Erreur de description** Survient lorsqu'une *description* n'est pas suffisamment précise. P. ex. confondre les logos `Reply` et `Replay All`.
▷ **Erreur de mode** Survient lorsqu'on utilise un *contrôle* dans un contexte différent de celui qu'on souhaite. P. ex. utiliser `j` ou `k` dans d'autres application que `Vim`.

Stratégies pour prévenir les erreurs

- ▷ **Bévues** Éviter actions aux démarches similaires
- ▷ **Lapsus** Fonctions de *forçage*
- ► Obliger retrait carte ATM pour prendre `$`
- ▷ **Erreur de mode** Éliminer mode ou ↑ *visibilité* modes.

Stratégies générales

- ▷ Désactiver *commandes illégales*
- ▷ Rendre informations *nécessaires* visibles
- ▷ Préférer les listes déroulantes aux zones de texte
- ▷ Utiliser les *dialogue de confirmation*
- ► Vous êtes sur le point de `rm -rf` !

Contrôle par l'utilisateur et liberté L'interface doit encourager l'exploration en rendant les choses visibles et en assurant que les conséquences des erreurs ne soient pas sévères.

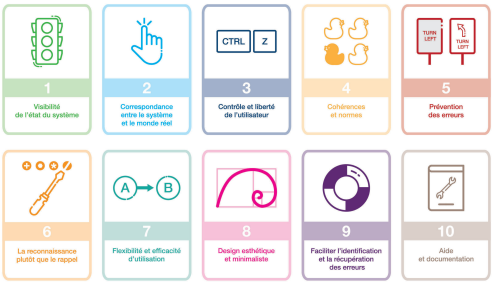
$$\text{Exploration} \implies \uparrow \text{ Découvrabilité}$$

.
▷ S'il y a **automatisation**, les changements doivent aussi pouvoir être effectués manuellement. L'automatisation ne doit pas `override` l'autonomie de $\mathscr{U}$. Les saisies `(input)` doivent être. Un $\mathscr{U}$ doit pouvoir créer, modifier, lire, mettre à jour et supprimer toute donnée. **modifiables**

Le droit d'annuler implique que L'$\mathscr{U}$ doit pouvoir décider ; les opérations doivent être annuables, et toutes boîtes de dialogue doit avoir un bouton d'annulation, pour ne pas **piéger**

3.1 Évaluation heuristique

Avantages Peut être fait rapidement, sans équipement, sans $\mathscr{U}$ et pour peu de $.

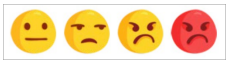


Étapes d'une évaluation heuristique

- **Mise en place** préparation du matériel (scénarios, protoype, liste d'heuristiques)
- **Évaluation** pas exploration de l'interface, application des heuristiques génération de liste de problèmes
- **Agrégation** retour et discusson entre experts.

> **Note :**
> Il est important d'avoir **plusieurs évaluateurs** parce chacun d'eux peuvent identifier *différents problèmes*

Évaluation Il faut au moins **2 passages** par évaluation ; le premier sert à découvrir le flux de la plateforme et le second permet de *cibler* des éléments spécifiques.
▷ **Liste de problèmes** est produite par l'évaluateur ; il y mentionne *l'heuristique qui est violée* et attribue un **cote** de gravité.



▷ **Impact du problème**
▷ **Fréquence du problème**

Marqueurs de bon design Les $\mathscr{U}$ n'attendent pas trop longtemps et on offre du *feedback* (visibilité) ; on utilise des mots, symboles et conventions qui *match* avec le monde réel, p.ex *trashbin* ; on *ne force pas* l'utilisateur sur un chemin fixe ; on permet la révision, correction, modification, supression ; on *rend les erreurs impossible* et valide les entrer ; on s'assure que les tâche courantes se font de façon efficiente grâce aux accélérateurs de calvier et souris, abréviations, double-clic, raccourcis de menu et touches de fonction, etc. ; on aide l'$\mathscr{U}$ à reconnaître, diagnostiquer et récupérer les erreurs ; on offre de la documentation.

Section 4

Design centré sur les tâches

Persona Représentation symbolique de *l'*$\mathscr{U}$ décrivant ses caractéristique et les *tâches* qu'il doit accomplir.

Parties intéressées *Stakeholders* Réfère à toute personne ayant une **raison** de se soucier à l'interface.

- Souvent nombreux
  - Les $\mathscr{U}$
  - Les `dev.`
  - Les constructeurs
  - Les sponsors
- Intérêts parfois en conflits
  - Ces efficience, utilité
  - Facilité de dev.
  - Facilité de construction
  - Coût de financement

4.1 Workflow du design centré sur les tâches

`Identification` ⟶ `Exigences` ⟶ `Design Conceptuel` ⟶ `Parcour cognitif`

Identification Identifier l'$\mathscr{U}$; analyser des *exemples de tâche*

Exigences Décider quelles tâches le design soutiendra.

> **Note :**
> Il *faut* décrire les tâches à l'aide **d'exemple de tâches imaginées** pour pavenir à déterminer les exigences. Les tâches peuvent être décrites par leur fréquence et leur importance.

En pratique, pour identifier les tâches, on peut contacter les potentiels $\mathscr{U}$ et discuter avec eux du fonctionnement du système pour avoir leur feedback. Il faut demander à l'$\mathscr{U}$ de réviser les exemples de tâches :

- Omissions
- Corrections
- Clarifications
- Suggestions

Clarification de la distinction tâche-activité Dans le *design centré sur les tâches*, le mot *tâche* a une signification plus proche du mot **activité**, puisque le design comprends prend en compte une grande quantité de tâche et leur contexte d'exécution.

Bon exemples de tâches Ils sont décrivent **quoi** sans extrapoler **comment** ; il sont *spécifiques* ; ils sont *exhaustif* et décrivent le travail de l'$\mathscr{U}$ dans son ensemble ainsi que la transmission globale de l'information. Dans l'ensemble, ils reflètent les intérêt des vrais $\mathscr{U}$.

Design conceptuel Décrire la base d'interaction et de représentation (modèle mental). On utilise un **scénario** :

`tâche + design` = `scénario`

Le scénario montre comment une tâche est gérée par le design.

Tâche : `Elle choisit une recette dans la liste`

Scénario: `Elle fait défiler la liste d'images et tape sur...`

Parcours cognitf Parcourir le tâches *pas à pas* en utilisant le design et **évaluer**.

Avantages de design centré sur les tâches

- Permet de baser le design sur $\mathscr{U}$ et *tâches réelles*
- Permet d'identifier les exigences lors de la *phase pré-design* (avant même design conceptuel).

Section 5

Les exigenbces du design

Types d'exigences

- **Fonctionnelle** : ce que le produit doit faire
- **De données** : caractéristiques des données requises
- **D'environnement** : Contexte physique et non physique nécessaires au fonctionnement.
- **Propres à l'**$\mathscr{U}$
  - P. ex. une *mère célibataire*