L'Ergonomie Logicielle en bref

...

GALODÉ Alexandre

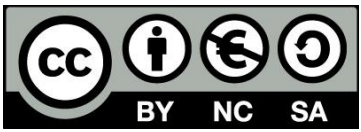
L'Ergonomie Logicielle en bref

...

Ce tutoriel a été rédigé sous Libre Office et écrit avec des polices d'écriture libres.

" La connaissance appartient à tout le monde "
Film Antitrust

Conçu et rédigé par Alexandre GALODÉ.



SOMMAIRE

Introduction

Quel que soit le langage utilisé, ou encore l'utilité d'un logiciel, se pose forcément à un moment donnée la question de l'ergonomie du logiciel.

L'ergonomie influence partiellement la production, tant d'un point de vue qualitatif que quantitatif.

De nombreuse normes ISO existe d'ailleurs à ce sujet: 13407, 16982, 9421-10 à 9241-17, ...

L'ergonomie vise à faciliter la navigation au sein du logiciel et à rendre ce dernier aussi convivial à utiliser et intuitif que possible. Cela passe par l'adaptation du logiciel aux utilisateurs.

En effet, aussi bien la qualité que la facilité d'utilisation compte.

Aucune recette miracle n'existe pour ce faire, et d'ailleurs, de nombreux paramètres entre en jeux.

En effet, chaque logiciel sera différent et devra bénéficier d'adaptation propre à l'environnement, ou au milieu d'utilisation.

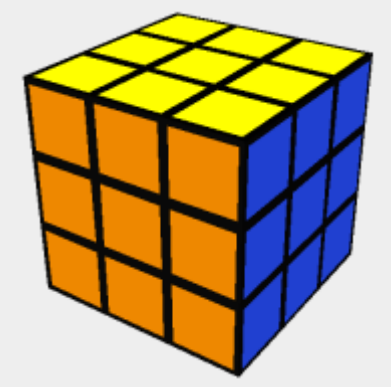
Mais en même temps, certains règles de bases sont à respecter et seront communes à la quasi totalité des logiciels.

Cet ensemble de règles à la fois communes et dédiées constituent les bases d'une ergonomie réussie.

Aussi, même si ce petit tutoriel n'a d'autres but que de vous fournir les fondamentaux, nous tacherons de voir l'essentiel à connaître afin que vos logiciels soient toujours une réussite d'un point de vue ergonomie.

Bonne lecture

1 Les bases communes



Wikimedia Commons,
solved_cube.png

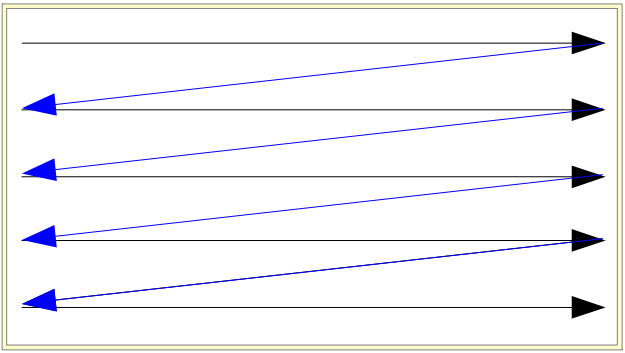
1.1 Nos habitudes

1.1.1 Visualisation d'une page/d'un écran

Notre éducation à un impact dans notre vie de tous les jours. Cela inclut la visualisation d'une page informatique, et donc l'utilisation de logiciel.

Par extension cela aura également un impact sur l'ergonomie que doit adopter un logiciel.

Nous avons l'habitude de lire une page de gauche à droite, et de haut en bas.



Tout logiciel se doit donc d'adopter cette même logique.

Imaginez un logiciel disposant de sa zone de travail en haut, son menu à droite, sa barre de statut à gauche. Vous seriez bien désorienter au moins dans un premier temps. C'est précisément cette situation qu'il faut éviter de produire.

1.1.2 Organisation

Toute IHM se doit d'être bien organiser afin de garantir une utilisation agréable et optimale.
Dans ce but, tout type d'éléments, quels qu'il soit (toolbar, menu, ...), doit être classé.

Différentes options peuvent alors s'offrir, selon les besoins ou souhaits des utilisateurs:

>alphabétique
>logique
>importance / priorité
>...

On affichera les éléments selon l'option choisit de gauche à droite ou de haut en bas.

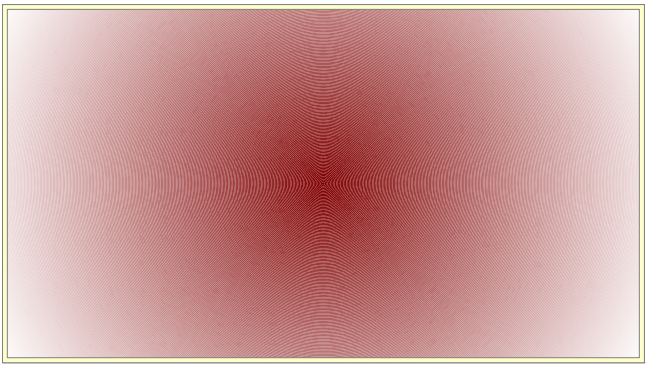
1.1.3 Champ de vision

Le champ de vision de l'être humain est un champ de type rectangulaire.

Notre vision se focalise sur un point central, puis sur la périphérie. Plus on s'écarte du point central, moins l'être humain remarque de détails.

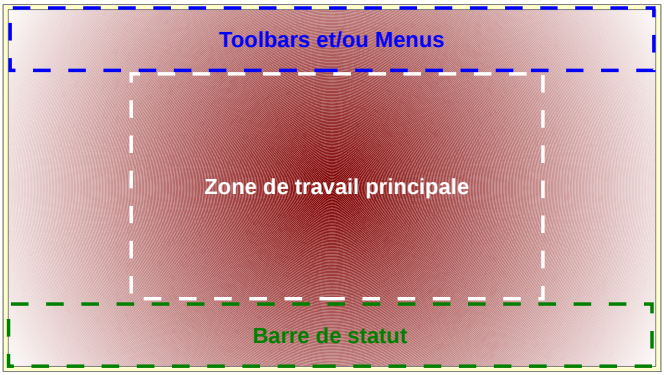
Ce principe doit être pris en compte dans la conception d'IHM.

L'illustration suivante est une représentation de la vue humaine, adaptée à un écran rectangulaire type 16:9.



Sur cette illustration, le point central est rouge. On remarque bien que plus on s'écarte du centre du rectangle, moins on visualise le rouge. On remarque également que nous sommes plus attentifs aux bordures hautes et basses qu'aux bordures gauches et droites.

L'ensemble de ces constats permet de donner des lignes directrices pour une bonne organisation de nos IHM.



La zone de travail principale peut ne concerner qu'une partie ou la totalité de l'espace situé entre les menus et la barre de statut.

1.2 Règles basiques

Une bonne IHM répond positivement aux points suivants:

>Utilisabilité / simplicité:
Est-il facile d'utiliser le logiciel?
>Explicité:
Le logiciel peut-il être utilisé sans notice?
>Convivialité:
Le logiciel est-il agréable à utiliser?
>Fonctionnalité:
Le logiciel répond-t-il aux besoins?
>Erreurs maîtrisées:
Les messages d'erreurs sont-ils explicitent?

1.2.1 Les principaux éléments

1.2.1.1 Fenêtre principale

L'IHM de la fenêtre principale doit être la plus précise possible. La fenêtre doit posséder un titre.

Il faut limiter l'affichage dans la fenêtre à 1 ou 2 tâches à la fois.

1.2.1.2 Fenêtres secondaires

Lors de l'affichage d'un message d'information, d'erreur, …, à l'utilisateur, il faut utiliser une icône standard afin d'indiquer le type de message.

De plus, le texte du message devra être court, explicite, et pertinent, pour tout utilisateur, y compris les novices. On évitera donc de donner trop d'information à l'utilisateur, car trop d'info tue l'info.

En cas de message d'erreur, un numéro pourra être fournit, à destination des informaticiens. Ce numéro servira alors à localiser plus précisément le type d'anomalie constatée, et/ou le code étant tombé en erreur.

1.2.1.3 Menus

La création d'un menu doit s'inspirer de l'environnement, en en extrayant des mots clés.

Outre cela, il faut également utiliser des mots clés standards (Fichier, Edition, …).

Au sein des menus, un tri sera effectué afin de présenté les divers éléments à l'utilisateur en respectant une logique (alphabétique, importance, …).

Pour des questions de lisibilité et de clarté, on se limitera à 2 sous menus.

1.2.1.4 Icônes

Les icônes utilisées au sein de l'application ne doivent pas être choisies au hasard.

Le mauvais choix d'une icône peut engendrer des incompréhension, voir des erreurs d'utilisation par un utilisateur.

Une icône doit s'inspirer de l'action qu'elle vise à déclencher. Toute icône doit pouvoir se passer de texte descriptif. C'est à cette seule condition qu'une icône peut être considéré comme judicieusement sélectionnée.

Les icônes doivent être à la bonne taille, ni trop petite, ni trop grande, et si possible être issue de l'environnement.

1.2.1.5 Barre d'outils

Les barre d'outils peuvent se présenter sous forme verticale ou horizontale. Comme vu sur le chapitre concernant le champ de vision, il est préférable d'utiliser le modèle horizontale, qui sera plus visible.

Comme pour les menus, on tachera de trier les éléments en respectant une certaines logique.

1.2.1.6 Barre de statut

Les barres de statut devront impérativement être placées en bas de l'écran.

Les informations qui y seront affichées devront être pertinente et explicite.

1.2.1.7 Texte

Concernant le texte au sein de l'application, on privilégiera les polices standards et libre de droits.
On se limitera également à 2, voir 3 polices maximum, pour des raisons de clarté.

On oubliera pas d'aérer le texte et de contraster au maximum (texte de couleur sombre sur fond clair par exemple).

1.2.1.8 Boutons

Afin d'être efficace, les boutons doivent être clairement compréhensible.

Les boutons peuvent contenir du texte seul, une icône seule, ou une icône accompagné d'un texte. C'est cette dernière solution qui doit être privilégiée.

Une variante consiste à utiliser l'icône seule, avec le texte en bulle d'aide.

Le texte du bouton devra, encore une fois, être court, et explicite.

1.2.2 Le vocabulaire

Lorsque l'on conçoit un logiciel dédié à un environnement spécifique (mécanique, médical, scientifique, …), il est important de porter attention au vocabulaire utilisé.

En effet, chaque milieu spécifique utilise son propre langage. Ne pas tenir compte de ce langage propre reviendrait à condamner d'avance un logiciel à péricliter.

Si l'équipe de développement n'est pas du milieu visé par le logiciel, il est donc primordial de se documenter sur le vocabulaire adéquat, et de le tracer dans une documentation qui sera à disposition de l'ensemble des développeurs et utilisateurs du logiciel final.

Dans la mesure du possible, il faut également éviter d'utiliser des abréviation, et standardiser et uniformiser le vocable du logiciel.

1.2.3 Homogénéité

Il faut respecter un certains formalisme au sein de l'application afin d'éviter de perdre l'utilisateur.

Pour cela, il faut éviter d'utiliser différents termes pour la même action (ex: quitter et sortir).

De même, en cas de jeux de couleur, on tâchera de rester cohérent dans leurs utilisations.

Il faut que quel que soit l'endroit du logiciel ou se trouve l'utilisateur, il n'est jamais à remettre en question l'interprétation à faire de l'IHM.

1.2.4 Clarté

Toute IHM réussit doit être agréable à utiliser. Cela passe par une clarté de l'écran.

Il ne faut donc pas hésiter à regrouper des actions communes par thème par exemple, ou encore d'ajouter des onglets au besoin.

De même, il faut penser à aérer l'IHM et à bien aligner les divers éléments afin d'obtenir quelque chose d'agréable à l'œil.

1.2.5 Temps de réponse

Lorsqu'on conçoit un logiciel, il n'est pas rare de tomber sur des fonctionnalités qui mettent un peu de temps à s'exécuter.

Si on peut se permettre de laisser l'utilisateur attendre 1 ou 2 secondes, au delà, il faut penser à l'attente qu'il devra subir.

Pour cela, il faut mettre en place, idéalement, une barre de progression, accompagné ou non d'explication textuelle, permettant à l'utilisateur de connaître le niveau d'avancement de la tache.

1.2.6 Jeux de couleurs

Les jeux de couleurs permettent de mettre en évidence des données ou des informations liées à des données.

Par exemple, le rouge est habituellement associé à tout ce qui est critique, important, ou élevé. De même, le vert est associé généralement à un traitement qui s'est bien déroulé, à une variable dans la plage, …

Le choix de jeux de couleurs peut s'avérer judicieux à conditions de ne pas rentrer en conflits avec des jeux de couleurs utilisés dans l'environnement (carte météo par exemple).

Il faut alors faire attention au contraste, afin de ne pas agresser les yeux, et limiter ces jeux. En effet, choisir des jeux disposant de trop de possibilité va amener l'utilisateur à se perdre. Tâchez de vous limiter au strict minimum.

1.2.7 Procédure

Dans certains environnement, un process consiste en un enchaînement de traitement.

Lors de la retranscription de ces process dans un logiciel, il vaut respecter la chronologie afin de rester cohérent vis-à-vis des utilisateurs (ex: catalogue->sélection->commande->paiement).

1.2.8 Erreurs

Une des règles de base (PEP) en PYTHON est " Une erreur ne devrait jamais passer silencieusement ".

Cette règle résume bien ce genre de situation. Tout développeur sait parfaitement que une des plus grandes source d'erreur est l'interface chaise / clavier.

Pour éviter tout problème, il faut anticiper les erreurs potentiels en essayant de se mettre à al place des utilisateurs. Les erreurs doivent alors être indiquée aux utilisateurs en les accompagnant de numéro unique permettant de localiser l'erreur au sein du code.

1.2.9 Assistance des utilisateurs

Il existe différente façon de guider un utilisateur. Les points suivants peuvent être fournis au format PDF.

La première est de fournir une notice fonctionnelle avec le logicielle. Afin d'être efficace, il faut que cette notice soit concise. Il ne faut pas s'y perdre en explication interminable qui aura pour effet de donner envie à l'utilisateur d'arrêter de la lire.

Chaque exemple / chapitre, devra être agrémentée d'exemple, si possible visuel, afin de casser la monotonie du texte.

Afin de rester cohérent, cette notice devra utiliser les mêmes termes que l'application. De même, on respectera également la logique de process.

La seconde méthode consiste à insérer directement dans le logiciel des informations utiles pour l'utilisateur. Par exemple, si on attend une saisie de valeur, on peut préciser l'unité ou le format attendu.

La troisième méthode consiste à insérer au sein de l'application des infobulles. Ainsi, pour savoir comment renseigner ou utiliser une partie du logiciel, l'utilisateur aura simplement à positionner sa souris sur l'élément concerné.

Outre cela, pour un logiciel Open Source, on tâchera autant que possible de fournir une documentation technique.

1.2.10 Feedback ou retours utilisateurs

Un point important dans le développement d'un logiciel est le feedback, ou retour utilisateur.

A l'exception du logiciel dédié à un usage exclusivement personnel, un logiciel est censé servir à un minimum d'utilisateur.

Aussi, afin d'assurer la pérennité du logiciel développé, les retours utilisateurs ont une importance non négligeable.

Les utilisateurs sont ceux qui vont permettre de faire vivre un logiciel, et les écouter et tenir compte/appliquer leurs demandes/recommandations jouera en faveur du logiciel. Se sentant écouter, ces derniers continueront d'utiliser un logiciel, et le recommanderont autour d'eux.

Sur le principe du donnant/donnant, le feedback est une base importante dans le développement logiciel.

2 Processus de conception

Wikimedia Commons,
brainstorming.gif

2.1 L'idée de base

Tout comme n'importe quel projet, la création d'un logiciel part d'un besoin (personnel ou professionnel) ou bien d'une idée.

Afin d'éviter toute mauvaise surprise, il est important de se détailler au maximum le besoin/idée.

Une fois tout cela effectué, il faut passer à la partie IHM afin de savoir comment organiser le logiciel. C'est ce qu'on appelle les maquettes.

2.2 Les maquettes

Une maquette est un aperçu visuel (un simple dessin par exemple) qui permet de représenter grossièrement, ou finement au choix, le rendu final que doit posséder le logiciel à développer.

Les maquettes présentent un intérêt dans le sens, ou avant même de coder votre logiciel, elles vous aide à concevoir l'IHM et à valider cette dernière auprès des utilisateurs finaux.

De plus, une fois les maquettes validées, ces dernières vont vous simplifier le codage, car vous pourrez alors facilement coder l'IHM.

2.3 La confrontation au monde réel

2.3.1 Le point de vue du développeur

Un développeur est quelqu'un de plonger dans son monde et ayant sa propre vision de la façon dont doit être développé le logiciel.

Malheureusement, cette vision, principalement technique, peut ne pas aboutir au résultat escompté par l'utilisateur final.

2.3.2 Les points de vue utilisateur

L'utilisateur final a également sa propre vision des choses. Pour lui peu importe la manière dont est construit un logiciel (du moins en général). Seul le côté fonctionnel de ce dernier l'intéresse.

Le logiciel est-il simple d'emploi? Remplit-il bien ses besoins? Enfin, est-il ergonomique?

Bref, une vision à l'opposé du développeur.

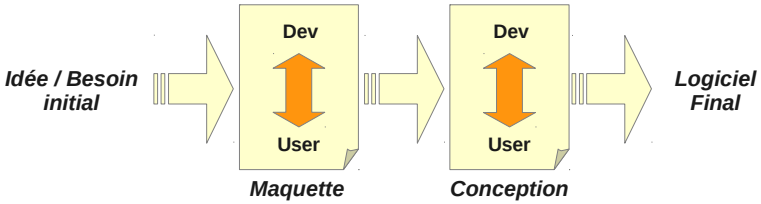
2.3.3 La communication: la clé

Comme indiqué dans le titre, la clé d'un développement réussi est la communication entre chaque partie.

Chaque partie possède un élément primordiale au développement: le développeur possède la clé technique, et l'utilisateur final la clé fonctionnel.

A moins de disposer d'un développeur ou d'un utilisateur compétent à tous niveaux (ce qui existe mais sera malheureusement rarement reconnu d'un point de vue professionnel) l'échange d'informations ainsi que d'avis est indispensable.

Cet échange est indispensable et fait partie intégrante du cycle de création d'un logiciel, qui peut se simplifier et se résumer ainsi:



2.3.4 La qualification

La qualification vise à déterminer si le logiciel créé répond bien aux attentes, aussi bien en terme de fonctionnel que d'ergonomie. C'est ce dernier point qui nous intéresse naturellement ici.

Pour cela, de nombreuses méthodes existent. Pour la plupart, elles consistent en une série de question, bien plus ciblé et étayé que celles posées au tout début de ce tutoriel.

Ces questionnaires sont remplis, en général, par une équipe dédiée, indépendant des développeurs, afin d'éviter toute influence.

On peut également imaginer un questionnaire à destination des utilisateurs finaux.

Je vous renvoie à votre moteur de recherche internet avec des mots clés tels que Meinadier, Shneiderman, Bastien & Scapin, ...