www.e-naxos.com

Formation – Audit – Conseil – Développement
XAML (Windows Store, WPF, Silverlight, Windows Phone), C#
Cross-plateforme Windows / Android / iOS
UX Design

ALL DOT.BLOG
Tome 4
Design & UX

Tout Dot.Blog par thème sous la forme de livres PDF gratuits !

Microsoft® MVP Most Valuable Professional
Olivier Dahan
odahan@gmail.com

Table des matières

Le défi des nouvelles interfaces - La cassure conceptuelle

J'ai l'habitude de publier ici du code, des tutors et des informations qui sont majoritairement très techniques. Il y a certains sujets où il me semble bon de prendre un peu de recul, les **nouvelles interfaces utilisateurs** le méritent largement. En effet, l'avalanche à la fois technologique et visuelle qui accompagne XAML (Modern UI, Silverlight et WPF, Windows Phone), peut faire oublier l'objectif réel et le changement de paradigme, le nez collé à la vitrine, la tête dans le guidon pour ingurgiter les propriétés de dépendance ou les subtilités de WinRT on ne prend pas assez le temps de réfléchir plus globalement...

Ce billet sera donc une pause dans ce rush, on s'assoit, on prend le temps de réfléchir.

Pourquoi "nouvelles" interfaces ?

Comme vous l'avez certainement compris, quelque chose a changé. Il suffit de regarder Surface et son interface pour comprendre que ce changement n'est pas uniquement cosmétique, il est profond et touche aussi bien à la **représentation des données** qu'à **l'interaction homme / machine**. C'est pourquoi je parle ici de *"nouvelles" interfaces utilisateur*.

N'est-ce qu'une question de mode ?

Oui et non.

Oui car tout est mode chez l'humain... La façon de se coiffer, de s'habiller, la forme des voitures, celle des cafetières, etc. L'informatique, dans sa partie visuelle et interactive n'échappe pas à ce mouvement perpétuel. Le fait qu'un programme puisse se manipuler selon des concepts "à la mode" n'est donc pas "honteux" ni même accessoire ! C'est ainsi que l'humain vit, c'est ainsi qu'il imprime sa marque à l'environnement qui l'entoure. L'humain est le seul animal a modifier son environnement pour le faire correspondre à ses rêves, là où tous les autres animaux subissent de plein fouet la sélection darwinienne et ce qu'on appelle la pression de l'environnement sans rien pouvoir y changer. Il y a donc bien un phénomène de mode dans ces "nouvelles interfaces" mais nous venons de voir que mode et civilisations humaines sont deux choses inséparables...

Non car on a tendance à associer les modes à la simple cosmétique et que le changement qui s'opère est bien plus conceptuel qu'un simple "relookage". J'y reviens plus loin.

Qu'est-ce qu'une mode ?

Pour ce qui nous intéresse ici, car sinon on pourrait en faire largement un sujet de thèse, disons qu'une mode est une façon collective et temporaire de faire les choses ou de les représenter.

Comment passe-t-on d'une mode à l'autre ?

Pour l'illustrer, une petite anecdote :

J'étais chez mon père il y a quelques jours et il venait de s'acheter une machine à café Senseo (lassé qu'il était de se faire racketter par les dosettes Nespresso du bellâtre "What else?"). Surprise ! Ce modèle est rectangulaire alors que les Senseo, jusqu'à lors, était tout en rondeur (le nouveau modèle n'est d'ailleurs même pas encore sur leur site). Un changement de mode s'opère... Mais comment ? Ici on le voit, la mode était aux rondeurs, elles semblent passer aux formes plus carrées. Il est de même en informatique : Windows nous a habitués à ses débuts à des fenêtres bien rectangulaires aux angles bien droits. Ces fenêtres plates et rectangulaires ont fini par lasser. Windows avec XP et plus encore avec Vista et 7 est donc passé aux rondeurs. La "classe" absolue ayant été au début de cette mode de faire tourner ses applications dans une fenêtre non rectangulaire ! Allons-nous revenir à des fenêtres rectangulaires à l'instar des formes de la nouvelles Senseo ? Le nouveau paradigme qui préside à la conception des "nouvelles interfaces" se résume-t-il à cela ? Passer du rond au carré pour mieux y revenir ? On le voit bien avec le « flat ». Au début tout était flat, puis la « mode » a été aux « chromes » voire au skeuomorphisme pour revenir au flat avec Modern UI… N'y-a-t-il vraiment rien de plus ?

La cassure conceptuelle

De nombreuses modes en effet se sont juste résumées à cela : un changement de forme, d'aspect. Quand tout est rond on finit par revenir à quelque chose de plus carré et inversement. Le triangle, malgré sa très grande force symbolique ne se prête guère à l'affichage écran. Exit le triangle, le losange n'étant guère plus pratique, peut-être qu'un jour la mode sera aux fenêtres patatoïdes ? !

Les nouvelles interfaces homme / machine se situent-elles uniquement sur le plan esthétique ?

Encore une fois : Oui et non.

Oui car il est impossible pour une représentation d'échapper à la mode ambiante, ou celle à venir. Il en va de même en écriture par exemple.

On peut écrire des choses très nouvelles sur un sujet, mais il faudra bien se plier aux exigences de la langue de son pays, telle qu'elle est parlée au moment où l'on écrit.
On voit mal un philosophe contemporain ou même un simple romancier publier un ouvrage en latin ou en vieux français. Certains mots sont plus "à la mode" que d'autres, et s'il ne veut pas que sa prose "sente la naphtaline" il faudra qu'il emploie les mots, expressions et formes grammaticales de son temps.

Sinon il risque tout simplement de ne pas être lu/écouté, pire de **ne pas être compris**. Car la mode est un **code culturel** qui fait partie intégrante de la **communication** et donc de la **compréhension** entre l'émetteur et le récepteur d'une information.

Il y a donc toujours une part d'allégeance à la mode en cours dans toute forme de communication. Cette mode pouvant même varier au sein d'une même société selon la cible visée (on ne s'adresse pas à la célèbre ménagère de moins de 50 ans des sondages comme à l'ado qui écrit en langage texto).

La mode, ainsi exprimée, s'intègre dans un corpus de règles sociales et morales plus large qu'on appelle les us et coutumes. Vouloir y échapper n'est pas forcément faire preuve d'originalité. Refuser la mode par simple réaction c'est aussi se couper de ses contemporains. Mais on peut aussi créer la prochaine mode...

Non les nouvelles IHM ne sont basées uniquement sur l'esthétique car on assiste aujourd'hui à une cassure conceptuelle. Une rupture qui peut largement être minorée, voire ignorée, cachée qu'elle est justement derrière l'écran de fumée du simple phénomène de mode perçu comme seule modification esthétique.

Quelle est cette cassure conceptuelle ?

Elle est de taille ! Prenons un exemple concret : vous devez créer un logiciel de surveillance bancaire. Ici tout n'est que séries de chiffres, de pourcentages, de tendances à la hausse ou la baisse. Comment allez-vous créer l'interface d'un tel logiciel ?

La plupart des informaticiens se plongeront d'abord dans le code et poseront ensuite rapidement quelques grilles, ces horribles tableaux rectangulaires que tout éditeur de composants se doit d'avoir perfectionné à sa manière.

Va-t-il suffire de "templater" une DataGrid sous Blend en lui mettant des coins arrondis pour dire qu'on a utilisé correctement XAML ?

Là, en revanche une seule réponse, claire et nette : Non !

Se servir du potentiel créatif de XAML ne consiste certainement pas à relooker les cases à cocher ni à "coller" un drop shadow à une boîte de dialogue ! Agir ainsi est un gâchis énorme, et surtout c'est ne pas avoir compris la cassure conceptuelle qu'imposent ces outils.

Revenons à notre hypothétique application de surveillance. Supposons pour simplifier qu'elle permette à l'utilisateur d'avoir un œil sur les actions de son portefeuille. La surveillance pouvant être activée ou désactivée pour chaque action.

La première tentation sera ainsi de représenter la liste des actions sous la forme d'une grille de données. On aura l'impression de profiter de toute la technologie en templetant la grille pour ajouter le logo de la société, des petites pastilles qui changent de couleur quand le logiciel teste une action, une autre quand l'action est à la hausse ou à la baisse, etc. On ira même jusqu'à ajouter quelques animations, c'est tellement "fun". L'utilisateur naviguera entre les pages qui coulisseront à l'écran de droite à gauche et réciproquement, etc. Là on aura eu l'impression de faire du neuf et d'exploiter à fond les possibilités de XAML !

Hélas, que nenni... En réalité ce qui aura été fait n'aura concerné que la seule apparence. Mettre du rond là où c'était habituellement carré ou l'inverse. C'est la mode dans son sens le plus négatif et le plus péjoratif qui soit : une lubie temporaire qui fait que si on n'a pas les dernières Nike en cour de récré on passe pour un ringard à qui plus personne ne veut parler. Angoisse permanente de l'ado moderne. C'est le côté agaçant, "fashion victim", la mode pour la mode, écervelée. Pourquoi ? **Parce que la mode sans un vrai concept n'est rien**.

La force du concept

Tout est concept, la réalisation n'est finalement que le passage obligé pour donner corps à un concept. Un compositeur quand il possède le concept, l'idée d'une nouvelle œuvre, n'a pas besoin de savoir écrire des partitions ni même de savoir jouer d'un instrument. Le vrai acte de création est purement intellectuel. C'est un pur concept. Mais pour le communiquer à ses semblables il lui faudra passer par la réalisation. Souvent d'ailleurs un compositeur fera appel à un arrangeur pour tout ou partie de son œuvre. Par exemple quelqu'un qui est spécialisé dans les parties de violons ou dans celles des cuivres. Tout le monde n'est pas Mozart, ce qui n'empêche pas des tas de compositeurs d'être connus et reconnus. Pour les films il en va de même. L'auteur n'est que rarement le réalisateur, et encore moins souvent le dialoguiste. Cela enlève-t-il de la valeur à son manuscrit sans qui le film n'existerait pas ?

Le passage au tout conceptuel est justement l'apanage de l'art contemporain. Certains y voient une fumisterie car ils n'ont pas compris qu'ici la cassure a eu lieu : la réalisation n'est que la partie visible de l'iceberg, l'essentiel de l'œuvre se trouve dans le ... concept. C'est pourquoi le bonhomme à tête carré dessiné par votre fils de 5 ans n'a aucune valeur autre que sentimentale alors que celle d'un Picasso s'arrache à coup de millions de dollars (même si ici la spéculation purement financière vient gâcher toute la beauté de l'art, mais c'est encore un autre sujet!). Picasso n'est pas un handicapé moteur ne sachant pas faire mieux que des têtes carrées, il dessine à merveille, comme un Dali ou un Miro. Mais il a choisi de casser le moule, de s'exprimer autrement en échappant au carcan normatif, et pour cela il a beaucoup réfléchit ! Le changement est principalement et purement conceptuel, bien avant d'être dans le geste du pinceau sur la toile.

Une autre représentation du monde

Revenons à notre application bancaire. Que c'est triste comme sujet... Même relooké sous Blend, que cet étalage de chiffres sera rébarbatif. N'y-a-t-il pas moyen de casser le cadre, d'échapper au carcan normatif ? Ne peut-on pas ajouter quelques grammes de douceurs dans ce monde de brutes ? Un poil de poésie ?

Si, cela est possible. A condition de re-conceptualiser le rapport homme / machine, et donc les interfaces.

Un peu de poésie

Très honnêtement la bourse n'est pas un sujet qui m'inspire beaucoup de poésie mais faisons un effort !

Dans notre exemple chaque action se singularise par un nom, une quantité possédée, un cours et par un indicateur signalant si elle est surveillée ou non.

Transformons ces données en acteurs. Car le changement conceptuel est en partie là. Chaque action sera ici un acteur autonome. Ne nous reste plus qu'à choisir comment cet acteur sera **représenté** et dans quel **univers** on va le faire vivre.

Par contraste choisissons un cadre naïf à l'opposé du monde de la finance : une scène champêtre. Notre décor sera une clairière avec une forêt en arrière-plan. La tendance du CAC40 ? Nous allons la représenter par le temps qu'il fait : ciel bleu neutre quand l'indice est stable, soleil caniculaire lorsque que l'indice dépasse un certain pourcentage de hausse par rapport à la veille, pluie voire tempête lorsque l'indice se casse la figure.

Voilà déjà une information conventionnelle et rébarbative représentée d'une autre façon, non conventionnelle. Imaginez-vous un écran géant, quelque part sur un mur du bureau. Un joli décor de campagne animé. Quelques nuages arrivent dans le ciel. L'oeil averti de l'utilisateur saura que le CAC40 est en baisse légère. Avec l'habitude, en comptant les nuages dans le ciel il saura même dire de combien de pourcents. Tout cela sans *datagrid* ni alignement de chiffres....

Mais poussons plus loin les choses et intégrons les actions à ce paysage.
Supposons que chaque action soit représentée par un animal qui va évoluer dans le décor. Total, ce goinfre, sera peut-être representé par un sanglier fouillant le sol. Air France sera représenté par une libellule, les Ciments Lafarge par un castor, etc. Maintenant appliquons à ses animaux des **comportements en fonction des données** chiffrées. Le nom n'est plus une donnée à affichée, le caractère choisi pour chaque action est une identité. Une donnée de moins à afficher. La quantité d'action sera représentée par la position de l'animal : à l'avant-plan quand on possède beaucoup d'action de ce type, loin quand on en possède peu. Encore une colonne du data grid qui devient inutile. Le cours est-il à la hausse ou la baisse que l'animal aura un comportement joyeux, bondissant dans le décor, ou bien qu'il semblera mou, figé, voire couché sur le flanc au sol. Plus besoin de *datagrid* !

Et le caractère actif/inactif de la surveillance d'une action ? Imaginons un enclos dans le décor. Par simple Drag'drop (avec le doigt comme sur Surface) prenons un animal et posons-le dans l'enclos. Le logiciel interprètera cela comme l'exact effet d'une case à cocher "surveillance active" qui sera décochée. Reprendre l'animal et le sortir de l'enclos aura l'effet inverse.

Une idée folle ?

Fermez les yeux et imaginez ce tableau vivant un instant... Une scène de campagne avec des petits animaux qui se promènent, le temps qui change, ah, tiens, une éclaircie. Le pauvre castor a l'air bien malade se matin...

Bien entendu, certains voudront savoir ce que j'ai fumé aujourd'hui, c'est un risque que j'ai pris et que j'assume en écrivant ce papier !

Mais ce que nous venons de faire ici est de transformer une application rébarbative qui, dans le meilleur des cas aurait été rendu sous la forme de tableaux de chiffres aux coins

arrondis servis par quelques boutons animés et une poignée de drop shadow, en une scène champêtre, charmante. Quelque chose qui donne envie de regarder le programme fonctionner juste pour le plaisir. Car **les données sont devenues des acteurs**, car le cadre du **programme est devenu un univers auto-suffisant**.

La cassure est là

Plus de cases à cocher, plus de colonnes de chiffres, plus rien de normatif ni conventionnel. Les données sont des acteurs autonomes qui **racontent une histoire**.

Est-ce si fou ?

Il est évident que l'exemple que j'ai pris des cours de la bourse et sa représentation gentiment niaise façon Walt Disney est un simple contrepied volontairement ironique, mais pas seulement. Imaginez-vous encore ce joli tableau dans le bureau d'un riche spéculateur... Ne pensez-vous pas que je pourrais vendre des installations complètes, très cher, et que cela marcherait ? Certainement que si. Alors que personne ne voudrait acheter le énième logiciel de gestion de portefeuille que je pourrais écrire, même relooké sous Blend ! Certains même parleraient alors de snobisme, de gadget. Ceux qui ne pourraient pas s'offrir mon système bien entendu. Mais pas les utilisateurs qui l'achèteraient et qui en seraient très contents ! La "mode" serait lancée !

Ce qu'il faut donc comprendre ici c'est que les "nouvelles interfaces utilisateurs" l'UX (User eXperience), ce nouveau paradigme dont je parle ici bien souvent et depuis un moment maintenant, n'est pas "qu'une mode", c'est un **changement conceptuel**, donc en profondeur, du **rapport entre l'homme et la machine**. C'est une **nouvelle façon de penser les données** non plus comme des informations passives mais comme des acteurs autonomes évoluant dans un univers vivant.

Et concrètement ?

Tout cela est bien gentil, mais certains se disent (les plus courageux, ceux qui sont arrivés jusqu'à cette ligne et que je remercie au passage !) que bon, ils ont un soft à faire pour hier et qu'ils voient mal comment tout ce gentil délire peut s'appliquer à une compatibilité analytique ou une facturation.

Je leur répondrais que si j'ai pu transformer, même virtuellement dans ces lignes, une application de gestion portefeuille boursier en une scène de campagne tout peut être fait ! Pourquoi ne pas transformer les comptes clients en bulles de couleurs évoluant dans un aquarium, plus la bulle est grosse plus l'encours du client est important, ou bien pour les commerciaux, plus la bulle devient grosse plus il est temps d'appeler le client car il n'a pas passé de commande depuis longtemps. Quelques jolis écrans plats sur les murs du service commercial ou dans le bureau du patron permettraient immédiatement de suivre la vie de l'entreprise, sans chiffres, sans cases à cocher ni boutons radio, sans fenêtre aux coins ronds ou non...

Application exemple

J'avais commencé une application exemple sous Silverlight qui montrait tout cela, mais le temps me manque pour la terminer, c'est malgré tout bien plus long à faire que de placer une grille de données ! Si un jour un gentil mécène sponsorise la création de cet exemple je pourrais prélever le temps nécessaire sur mon emploi du temps déjà bien chargé… Avis aux amateurs !

Un autre exemple existe depuis quelques années, il n'est pas logiciel mais bien du domaine informatique tout de même : le lapin Nabaztag. Même si l'objet en tant que tel n'a pas fait une percée exceptionnelle (ses interactions étaient trop limitées à mon sens) il a su attirer les spots sur lui. Tiens, il baisse l'oreille gauche... le CAC 40 se casse la figure. Ah, il devient rouge, ma chérie a laissé un message sur mon répondeur... Le créateur du Nabaztag a tout compris de cette cassure conceptuelle et des nouvelles interfaces utilisateur...

Si le sujet vous intéresse, et pour voir prochainement une illustration de ce que je viens d'expliquer ici, alors une seule solution : lisez Dot.Blog et n'hésitez à m'appeler pour vos prochains logiciels !

UX != UI

Une petite mise au point s'impose devant l'abus qui commence à être fait de l'acronyme "UX" et de son développé "User Experience" (Expérience Utilisateur). L'UX n'est pas l'UI !

Certes il est tentant d'utiliser les nouveaux acronymes à la mode, mais attention, si cela peut vous faire mousser deux secondes à la machine à café locale, cela peut vous faire passer pour un idiot auprès de ceux qui savent de quoi il s'agit… Jeu dangereux donc. Le mieux reste d'apprendre et comprendre ce qui se cache derrière l'acronyme.

UX != UI

Par exemple faire une démo super cool d'un composant visuel ou d'un effet de transition hypra chouette, voire même connecter une grille totalement templatée à une base de données via RIA Services, tout cela n'est hélas que de l'UI (User Interface, Interface utilisateur). Ce n'est pas de l'UX. Au mieux du mieux, il peut s'agir de Design, mais pas encore d'UX.

Alors c'est quoi l'UX ?

Voilà une bonne question. Au-delà de l'acronyme plaisant et à la mode (les américains appellent cela un "buzzword" - un mot à buzz), se cache toute une réflexion sur l'approche de l'informatique au quotidien par l'utilisateur, la compréhension de ses besoins, comment rendre son travail plus simple, plus plaisant. L'UX est bien plus proche de l'ergonomie qu'elle ne l'est du Design ou de l'Interface Utilisateur. L'UX est un processus complet, pas une tâche parmi d'autres, ce n'est pas une croix à cocher sur une checklist d'opérations pour faire un logiciel. L'UX concerne l'utilisateur, son entreprise, leurs interactions. Le Design et les interfaces utilisateurs ne sont eux qu'une partie de ce grand processus qu'est l'UX.

Mais pour vous faire une idée plus précise, je vous conseille la lecture des quelques références ci-dessous qui vous permettront de mieux comprendre ce qu'est l'UX :

- **1. Mashable: 10 Most Common Misconceptions About User Experience Design**
- **2. UXMatters and their Glossary**
- **3. UX definition by Nielsen Norman group**
- **4. The Elements of User Experience by Jesse James Garrett** (PDF)
- **5. Eric Reiss and FatDUX defintion of user experience**
- **6. UX design versus UI development by UX Matters and Mike Hughes.**

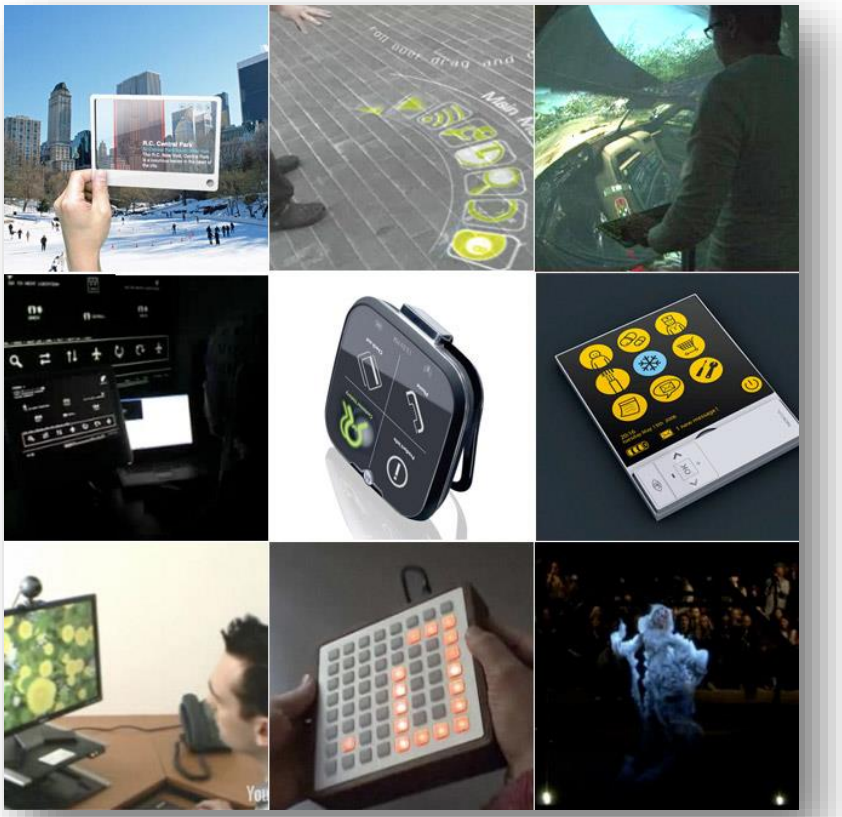
Tout cela est en anglais, of course. Mais instructif.

Concevoir des interfaces efficaces et vendeuses

Préambule

Parler de code c'est bien, parler de patterns ou des nouveautés de Silverlight c'est encore mieux, mais parler de Design c'est aujourd'hui tout aussi essentiel. C'est pourquoi j'aborde le sujet régulièrement… Dans le présent article je vous propose de faire le point sur la création des interfaces modernes, l'importance de cette tâche, les motivations de ce mouvement inéluctable.

Il ne s'agit pas d'un « cours » sur le Design qui réclamerait d'entrer plus en profondeur sur chaque point abordé ici, mais d'une présentation des grands concepts qui vous aideront à mieux concevoir vos UI et à garantir un UX de qualité à vos utilisateurs.



La fin d'un mythe, le début d'une aventure

La première chose à comprendre et à admettre c'est que notre beau métier qu'est l'informatique a eu tendance ces dernières décennies à laisser croire à certains qu'ils étaient un peu les "maîtres du monde". C'est un peu vrai… Sans nous, sans notre code, le Monde ne serait pas aujourd'hui tel qu'on le connaît : pas d'Airbus ni d'avion de chasse in-pilotable sans ordinateur (donc sans logiciel), pas d'Internet ni de Web 1 ou 2, pas d'IPhone, ni d'IPad, pas d'information en temps réel, pas de Webcams aux quatre coins du monde, rien, pas même de distributeur de billets, pas de CD, ni de DVD, pas de GPS. Rien. Enfin si, le monde tel qu'il était à la fin des années 50.

Bref dans ce monde de l'information, l'informaticien était un petit roi. Les électroniciens vous diront que ce sont eux les vrais maîtres du monde, sans circuits intégrés, notre métier n'existerait même pas (cogiter toute la journée sur la beauté algorithmique de programmes virtuels sur une machine de Turing purement idéale n'aurait pas changé le Monde).

La vraie **cassure conceptuelle**[1], le vrai nouveau paradigme “à avaler” c’est que depuis quelques années l’informaticien est devenu un OS des temps modernes (sans jeu de mot, ou presque).

Mais il y a encore plus grave : la chute du petit roi ne s’arrête pas là, les vrais maîtres du monde aujourd’hui en matière de technologie ce sont les Designers !

L’IPhone offre-t-il une meilleure écoute que la concurrence ? Non. Offre-t-il une meilleure couverture ? Non, cela ne dépend pas de lui.

Offre-t-il des fonctions téléphoniques plus évoluées que la concurrence ? Non plus, passer un coup de fil se fait aujourd’hui comme dans les années 60 : on décroche et on compose son numéro, ça sonne, on attend que l’autre réponde…

Les problèmes d'antenne de modèles récents, tout comme le fait qu'on reconnait un utilisateur iPhone car c'est le seul à se trimbaler avec son chargeur et à chercher une prise électrique dès qu'il arrive chez vous, démontrent même que comparé à la concurrence comme Nokia les Smartphones Apple sont nettement de qualité moindre. Ils sont d’ailleurs fabriqués en Chine, comme les concurrents, par des entreprises qui ne sont pas plus respectueuses des droits de l’homme, du travail des enfants et de la protection de l’environnement que celles qui fabriquent pour Nokia, Sony-Ericsson ou d’autres…

Pourtant ce sont là des points vitaux pour un téléphone ! Non, réaffirmons-le, *sur l’essentiel de ce qui fait un téléphone moderne l’IPhone ne fait rien de tout cela mieux que les autres* (c’est même plutôt l’inverse !).

Mais il a un Design.

Et ce n’est qu’un exemple parmi des milliers.

On pourrait parler des voitures qui se vendent mieux que les autres, et de milles objets, de la cafetière aux couverts de table, qui eux aussi sont tombés sous la domination du Design. Ce qui a fait le succès d’un Ikea sur ces concurrents "low cost" ? … Le Design.

[1] La « Cassure Conceptuelle » est un concept que j’ai introduit dans le billet « Le défi des nouvelles interfaces Silverlight et WPF - La cassure conceptuelle » du 1er septembre 2009.

(Ci-contre un presse-agrume signé Philippe Starck)

De Charybde en Sylla, notre pauvre informaticien ne cesse de dégringoler de son trône !

Alors certains changements se font dans la douleur au lieu de se faire dans la joie. Il y a de la résistance, du passéisme, on devient réactionnaire, méfiant, traitant tout changement de “mode” pour en minimiser l’impact et l’importance, vénérant des dieux déchus, des langages oubliés. En un mot on se ringardise.

« Puisque ces mystères me dépassent, feignons d’en être l’organisateur »

Heureusement il existe une issue plus heureuse ! N’oublions pas cette pensée sublime de Cocteau.

Le même disait aussi

« L’avenir n’appartient à personne. Il n’y a pas de précurseurs, il n’existe que des retardataires. »

Ce qui, quand on prend la mesure est à la fois d'une profondeur inouïe et un éclairage optimiste sur notre fonctionnement : il nous suffit de vivre avec notre temps pour ne pas être ringard et pour passer pour des visionnaires... Mais nous sommes tellement englués dans le passé que même ceux qui croient vivre "au temps présent" se trompent : le présent n'existe pas, puisque dès qu'on l'évoque il s'agit déjà du passé.

Le poète est souvent un sage qu’on gagne à écouter…

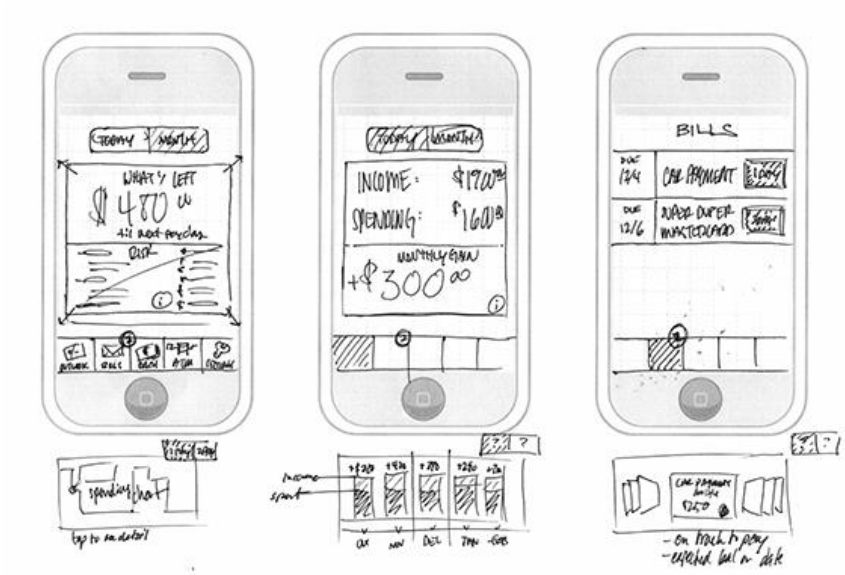
Le rapport avec le sujet ? C’est que les Designers ne peuvent pas faire ce qu’ils veulent, ils ne sont rien sans le fonctionnel ! C’est d’ailleurs ce qui fait d’un simple créateur ou dessinateur un vrai « Designer » *c’est la prise en compte des contraintes d’utilisation* !

Si tout le monde sait ce qu’est une fourchette ou un presse-agrume, et s’il est donc facile à un Starck d’imaginer de nouveaux designs pour de tels objets, en matière d’informatique les choses sont plus complexes. C’est pourquoi les vrais Designers d’interface sont encore bien rares.

Les informaticiens peuvent reprendre le dessus s’ils ouvrent les yeux, s’ils sortent de leur grotte et tombent la carapace.

Un IPhone dont le contenu ne serait que fakes et qui ne passerait aucun coup de fil pour de vrai, où les jeux ne seraient que des plans fixes comme des affiches alléchantes mais sans le film qui va avec et où les applications ne seraient que des dessins d'interface sous Photoshop, vous croyez que ça se vendrait ?

Non. Bien entendu. Et pourtant dans un monde de Designers sans informaticiens, l'IPhone ressemblerait à cette description et serait totalement inutile…



Le vrai pouvoir est au fonctionnel, encore faut-il ne pas se laisser déposséder de ce pouvoir.

Et si les informaticiens d'aujourd'hui ne sont pas les créateurs de cette grande tendance du Design, ils peuvent largement en être les organisateurs (Cf. La citation de Cocteau plus haut…) ! La maîtrise du fonctionnel, la connaissance du besoin de l'utilisateur, aucun infographiste ne l'a. Aussi talentueux soit-il.

Certes il semble aussi difficile de faire apprendre l'art noble du Design à un informaticien que d'expliquer les merveilles de C# à un infographiste. Je vous l'accorde. Mais il n'est pas interdit d'avoir de l'imagination, et, avec les bons outils et les bons collaborateurs l'informaticien du 3ème millénaire peut fort bien devenir, enfin, le maître du monde qu'il pensait être, ou tout du moins *s'associer intelligemment avec les Designers pour accéder ensemble au trône*…

Mais il reste à comprendre l'essentiel, le fait que le Design n'est pas un accessoire, pas plus qu'une mode, *c'est un acte essentiel de la conception d'un produit*, qu'il s'agisse d'une voiture, d'une fourchette ou d'un logiciel. **Les Designers ne sont pas des putschistes venant nous voler la vedette, ce sont des alliés inespérés pour nous aider à conquérir de nouveaux utilisateurs**.

Pour nous tout commence donc avec les interfaces graphiques…

L'importance des interfaces utilisateur

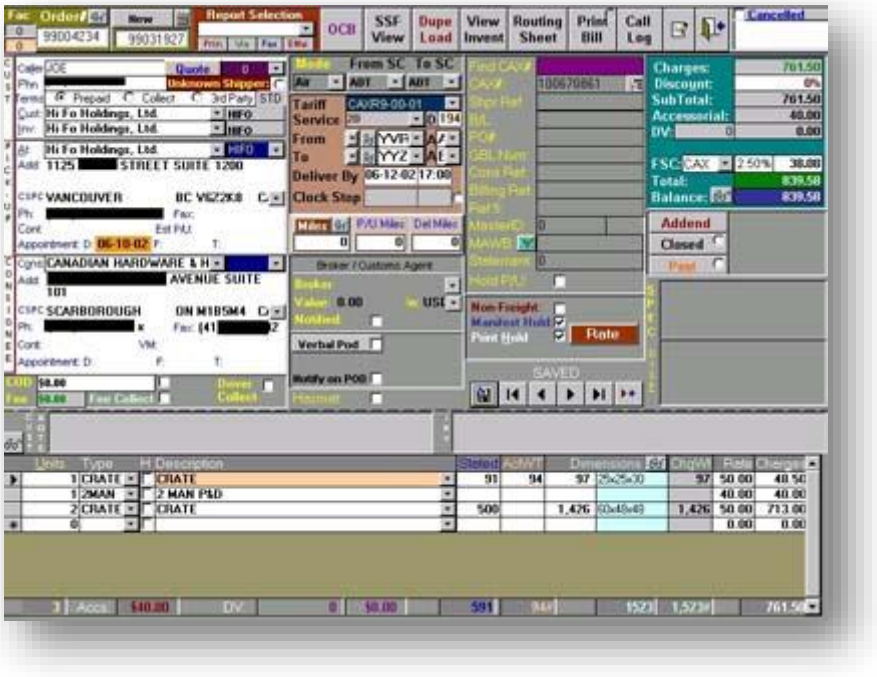
Pourquoi sont-elles si importantes ?

Parce que **c'est la seule partie d'un programme que l'utilisateur peut voir et toucher** !

Parce que nous connaissons tous des applications qui possèdent des fonctionnalités incroyables mais qui ne connaîtront jamais le succès à cause de leur mauvaise interface…

Parce que l'informaticien a toujours du mal à intégrer que l'utilisateur va peut-être passer 8 heures par jour à se fatiguer sur une interface qu'il n'aime pas…

Les utilisateurs ne sont que des humains comme les autres et ils jugent souvent une application en quelques minutes (souvent moins!), juste sur une "première impression". Ce premier contact aura un impact direct et immédiat sur la diffusion de l'application (sa vente, sa fréquentation si c'est un site Web…). Pire, tout cela se fonde presque uniquement sur l'interface utilisateur !



Aimeriez-vous passer des journées entières sur le logiciel ci-dessus ?

Nous sommes plongés, de gré ou de force, dans un monde d'images manipulées, un monde d'apparence où il devient presque impossible de différencier une image réelle d'une image fabriquée. En revanche, et à plus forte raison, habitués que nous sommes à des images "léchées", aux filles aux jambes improbables *photoshopées* des magazines, *il nous est très facile de voir au premier coup d'œil un logiciel moche au look ringard* !

Je connais la chanson pour l'avoir entendu milles fois : "les logiciels hyper lookés sont souvent vides fonctionnellement, moi j'ai fait un soft qui a la super fonction Z que personne n'intègre !". C'est peut-être vrai, mais n'empêche votre zinzin ne se vendra pas. Le relooking d'une application n'est pas une option, c'est une évidence, une partie non négociable de sa fabrication. Le reste n'est qu'excuse pour tenter de rester vainement le maître d'un monde qui n'existe plus…

L'utilisateur potentiel doit aussi percevoir une certaine familiarité, doit se sentir compris, et cela, l'interface peut le communiquer en quelques secondes mieux que 200 pages d'une documentation qu'il ne lira pas et qui tentera de lui faire comprendre la beauté de la fonction Z que personne d'autre n'implémente.

Il faut aussi prendre conscience que la plupart des utilisateurs compareront votre application à d'autres.

Souvent l'informaticien, très (trop ?) content de son "œuvre" oublie de regarder ce qui se fait ailleurs et comment cela est fait. Croyez-vous que Nokia lance la fabrication d'un nouveau téléphone sans avoir démonté tous ceux de ses concurrents avant ? Qui eux-mêmes en ont fait autant.

D'ailleurs vous. Oui vous qui lisez ces lignes. Le dernier logiciel que vous avez créé, avez-vous ne serait-ce qu'une fois tenté de voir comment des produits similaires ou concurrents répondaient au même besoin ? Quel était celui qui se vendait le mieux ? Pourquoi ? Je parie qu'une majorité de lecteur ne pourra que répondre « non ». Hélas…

L'informaticien n'est pas dans une logique industrielle car l'industrie est un système de production de masse alors que l'informatique réinvente tout pour chaque logiciel, c'est de l'artisanat. On entend parfois que c'est une passion de geeks, de no-live, un "trip" ultra-perso. Mais c'est une vision dépassée de notre profession. *Passer de cet isolement à un regard ouvert sur ce qui se fait ailleurs est un bon commencement pour améliorer l'interface de ses applications et devenir un informaticien qui vit avec son temps.*

Qu'est ce qui fait une bonne interface ?

S'il y avait une recette pour fabriquer des IPhones, tout le monde en vendrait sous toutes les marques. Hélas les "bonnes" interfaces sont comme les musiques qui rencontreront le succès, s'il y avait une "recette" cela se saurait depuis longtemps !

Mais en revanche si on ne sait pas dire exactement ce qu'il faut faire pour créer une bonne interface, on peut en cerner les principales qualités :

Une bonne interface semble **facile**, l'application apparait **intuitive** sans lire de manuel;

Une bonne interface est **vendeuse**, au premier contact. Mais elle se révèle aussi **productive** au fil du temps pour les utilisateurs avancés;

Une bonne interface permet à l'utilisateur **d'accomplir les tâches de la façon qu'il souhaite les accomplir** et non en étant forcé de suivre une logique rigide imposée par le concepteur;

Une bonne interface est **cohérente**. La cohérence doit s'étendre à toute l'application mais aussi à tout l'environnement de l'utilisateur et au cadre socio-culturel d'utilisation du logiciel, au profil psychologique des utilisateurs potentiels.

Quelques concepts importants de Design

Connaître ses utilisateurs

Cela peut sembler idiot, mais le plus souvent l'informaticien, même talentueux et ayant une bonne idée de logiciel, n'a jamais vu ni rencontré les utilisateurs potentiels ! Je ne parle pas des informaticiens créant des logiciels dits "standards" comme des comptabilités chez de gros éditeurs et qui travaillent presque à la chaîne, dans un total isolement par rapport à l'utilisateur final.

Bien connaître l'utilisateur final d'une application permet de définir les buts du Design de l'interface et peut modifier les lignes et la stratégie du développement lui-même.

Une application peut se destiner à des utilisateurs ayant un niveau technique et une habileté totalement différents des vôtres. Il faut en tenir compte et cela ne peut se faire qu'en connaissant parfaitement la cible concernée. D'où la nécessité souvent de définir des profils de personnages fictifs et de se mettre dans « leur peau » pour tester l'ergonomie et la conception de l'interface d'un logiciel (Robert le mécanicien pas geek du tout qui doit gérer sa comptabilité, Mauricette, sa secrétaire, la cinquantaine, qui a commencé sur une machine à écrire électrique, ou Carole, sa fille qui lui donne un coup de main de temps en temps, la vingtaine, écouteurs d'Ipod collés aux oreilles, saisissant des factures un œil sur son BlackBerry et les messages qui arrivent…). Ces profils s'appellent des Personas[2].

Une application peut être conçue pour des utilisateurs ayant un mode de pensée totalement différent du vôtre. *Ne pas savoir comment l'utilisateur "fonctionne", comment il pense, ses réflexes, ce qu'il aime, ce qu'il déteste, c'est passer à côté de l'essentiel et forcément prendre le risque de ne pas le séduire ni le satisfaire !*

Les buts de l'utilisateur

Un utilisateur n'est jamais devant une application pour le "fun". Ou alors c'est un informaticien et c'est une autre histoire… Un utilisateur se sert d'un logiciel pour **accomplir une ou plusieurs tâches précises**. Si c'est un jeu, ce n'est pas non plus pour le "fun" qu'il s'en sert, mais bien égoïstement pour se procurer du plaisir. Il n'y a pas d'acte gratuit chez l'humain.

[2] Les personas sont des représentations fictives mais concrètes des utilisateurs pour lesquels le produit est conçu. Ils fournissent aux développeurs une référence pour définir les fonctionnalités et les scénarios d'utilisation.

Connaître parfaitement ces tâches que l'utilisateur cherchera à accomplir permet de proposer des méthodes de travail en accord avec ses buts, sa façon de procéder, de penser. Le fameux sentiment de familiarité, de proximité que j'évoquais plus haut peut naître de cette connaissance et de l'adéquation du Design avec cette dernière.

Un utilisateur n'aime pas se sentir "idiot". Une bonne interface sait le guider s'il ne sait pas comment faire. Elle évite aussi les messages brimant du genre "Action interdite!" ou "Saisie non valide!". Ce sont pourtant des choses qu'on voit tous les jours !

L'utilisateur est parfois forcé d'utiliser un logiciel, auquel cas plus ce dernier saura le séduire, mieux cela sera. Mais souvent l'utilisateur choisit un logiciel parce qu'il pense que cela va l'aider à faire quelque chose qu'il ferait mal ou moins bien sans l'aide d'un ordinateur et d'une application étudiée. *Il ne faut pas le décevoir*…

Les deux types d'utilisateurs

Il y a *l'autonome* et *le survivant*.

L'autonome prend sur lui les fautes et ne les rejette pas sur le système (téléphone, PC, logiciel…). Il sait en apprécier les petits avantages et a tendance à minimiser les gros défauts. Il se débrouille. C'est le genre d'utilisateur qui, si vous lui montrez les faiblesses du système qu'il utilise vous dira "oui mais regarde, quand j'appuie là ça fait clignoter le clavier !". Les informaticiens sont comme ça. C'est aussi le profil des "power users".

Le survivant voit le monde autrement : lui il sait qu'il y a des problèmes dans le système mais ne sait pas comment les régler. Il aimerait d'ailleurs posséder un système *plus simple*, *moins bogué*, plus *proche de ses besoins*, mais il fait avec ce qui existe (ou avec ce qu'on lui a mis entre les mains). Quant au petit gadget qui fait clignoter le clavier, "même mon ours en peluche quand j'avais 4 ans savait faire mieux!".

La grande majorité des utilisateurs appartiennent à cette seconde catégorie !

L'habileté

Les utilisateurs sont des débutants pendant une période qui est généralement très courte. Passé un certain cap, ils se contenteront des solutions qu'ils ont trouvées pour accomplir les tâches usuelles et *n'iront pas chercher plus loin*.

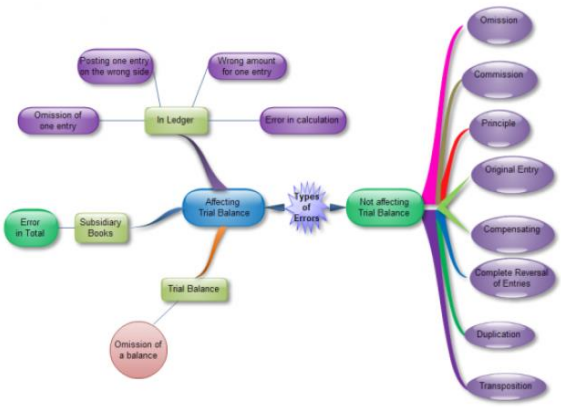
Seule une petite fraction minoritaire des utilisateurs deviendra vraiment “experte”. Le genre d’utilisateur qui peut vous faire voir des raccourcis ou des astuces d’utilisation que vous-mêmes en tant que concepteur du logiciel n’avez jamais imaginé !

En fait, la majorité des utilisateurs seront dans la moyenne, c’est la courbe de Gauss qui s’impose avec son corollaire : une majorité dans le centre de la cloche, une minorité aux extrémités.

Quelques cas font exceptions à cette règle : les applications publiques, de type annuaire téléphonique ou distributeur de billets, et les applications que les utilisateurs n’aiment pas. Dans ces deux cas la majorité des utilisateurs ne fera aucun effort pour comprendre le système et rejettera systématiquement la faute sur ce dernier.

Les schémas directeurs

Un utilisateur va approcher une application avec un certain nombre d’*a priori*, de réflexes, de **schémas directeurs** qui dépendent de ses *connaissances passées relatives au domaine du problème traité par le logiciel*.



Se fondant sur ces schémas, *l’utilisateur va s’attendre, inconsciemment la plupart du temps, à ce que l’application suive un certain “modèle”, un certain “schéma de fonctionnement”.*

Pensez à un comptable, lorsqu’on est passé des livres et journaux physiques à l’informatique il y a eu des grincements de dents… Un informaticien est parfaitement capable de créer une comptabilité qui réclame des connaissances mathématiques du niveau 6ème (additions, soustractions, divisions et multiplications). Mais en revanche il n’est pas comptable et ne connaît pas les habitudes des gens de cette profession. Les premiers logiciels ont reçu un accueil très mitigé.

Pensez aussi à un médecin. Habitué à une relation personnelle avec son patient et à qui “le progrès” impose un “tiers” s’installant entre lui et le patient : l’ordinateur et le logiciel “médical”. Ayant fait partie des pionniers de cette branche particulière des logiciels verticaux dans les années 80 je peux vous garantir que les prévisions et les *a priori* de tous

les informaticiens ont été balayés très vite lorsque leurs logiciels ont été confrontés aux utilisateurs !

Comment une consultation se déroule-t-elle, dans quel ordre un médecin traite-t-il le dossier du patient, combien de temps acceptera-t-il de passer à frapper sur le clavier alors que le patient se tient là, assis en face de lui ou allongé sur la table d'auscultation ? Comment va-t-il faire en visite ?

Les dossiers des patients en papier se transportaient facilement dans la mallette, on y écrivait deux lignes au chevet du patient et quand on remettait le dossier dans l'armoire en rentrant il était à jour. Si le patient téléphonait la secrétaire sortait son dossier de l'armoire pour lui répondre ou le transférer au médecin, pas besoin de "synchronisation", de "réseau", de "portable", ni de Wi-Fi et encore moins d'Internet… Quant aux "backups", la photocopieuse utilisée par la secrétaire était parfaite, pas besoin de disques externes, de NAS, de serveurs, de "scheduler" pour "programmer" les sauvegardes, etc… Et même en cas d'incendie, les piles de papier résistent bien mieux que les ordinateurs (expérience douloureuse faite avec l'incendie d'un appartement que j'occupais il y a de cela une quinzaine d'années).

L'informatique médicale était au final une gêne plus qu'une amélioration !

Cela reste vrai aujourd'hui, et fonctionnellement les logiciels modernes se sont adaptés en offrant nettement moins de possibilités. Il s'agit pour la majorité de "gros bloc-notes" flanqués d'une version électronique du Vidal (le dictionnaire des médicaments) et surtout accompagnés d'un module gérant la carte Vitale (seul vrai argument ayant poussé les médecins à s'informatiser, car si la science pure les enquiquine, les sous, ça les intéresse !). D'une application "scientifique" cherchant à apporter un soutien "intelligent", voire une aide au diagnostic et des systèmes experts, nous en sommes arrivés à des fourre-tout électroniques gérant la relation comptable avec la Sécu. Le médecin n'était pas le "scientifique" qu'on imaginait, il n'avait pas le temps qu'on croyait pour s'occuper d'un logiciel en même temps que d'un patient, et pour lui se faire payer était plus important que de pondre des statistiques sur les cas de grippes qu'il avait vu dans l'année. Enorme désillusion.

Surtout : **mauvaise approche de l'utilisateur** !

Erreur de casting, Le « profilage » de l'utilisateur avait été mal fait au départ.

Les premiers logiciels médicaux ont été développés avec l'aide de médecins "geeks" bricolant souvent eux-mêmes des petits logiciels, *les informaticiens croyaient voir à travers*

eux tous les médecins, ils ne voyaient que des exceptions, pas la règle. L'erreur a souvent été fatale pour beaucoup de sociétés[3] !

Ainsi, *connaitre les schémas directeurs utilisés consciemment et inconsciemment par l'utilisateur n'est pas un luxe, c'est une obligation.* Ne pas se tromper de modèle et choisir un panel d'utilisateurs représentatifs du milieu de la cloche de la courbe de Gauss est tout aussi important… Créer des profils types répondant à la réalité est une tâche essentielle lors du Design.

Cognitive Friction

Ce terme provient du second livre de Alan Cooper, le père de Visual Basic, "The inmates are running the asylum: why High-Tech products drive us crazy and how to restore the Sanity" - *Les aliénés dirigent l'asile : Pourquoi les produits High-Tech nous rendent fous et comment restaurer la santé mentale*.

Dans ce livre que tout Designer et tout informaticien devrait avoir lu[4] (je n'en connais hélas pas de traduction en français), Alan Cooper aborde le délicat sujet de l'interaction homme / machine. Dès le début du livre il enfonce le clou de son thème central qu'il appelle la "*cognitive friction*", qu'on peut traduire par "friction cognitive" mais ce qui ne vous avancera pas beaucoup :-) !

Il définit cette expression de la sorte :

> "*…the resistance encountered by human intellect when it engages with a complex system of rules that change as the problem permutes*."

Ce qu'on peut traduire rapidement par

> *"la friction cognitive est la résistance rencontrée par l'intellect humain quand il est confronté à un système complexe de règles qui changent en même temps que le problème évolue".*

[3] Personnellement j'ai tout simplement stoppé ma gamme Hippocrate© quand ces progiciels sont devenus des comptas connectées à la carte Vitale. Mon truc c'était l'aide au diagnostic, les statistiques, les interactions médicamenteuses, etc, pas la compta… Cela ne m'excitait plus assez pour continuer à éditer des logiciels de ce type. Et puis j'en avais assez de faire des procès à tous les margoulins qui me piquaient le nom de ma marque pourtant déposé à l'INPI.

[4] Alan Copper a aussi écrit « About Face : The essentials od Interaction Design », un excellent bouquin qu'on peut encore trouver, en anglais aussi.

Le livre étant assez ancien, Alan Cooper prenait beaucoup de références dans les appareils high-tech de l'époque, certains ont évolués ou n'existent plus, mais le raisonnement reste valable sur les appareils d'aujourd'hui hélas (et encore plus sur les logiciels) !

Dans sa définition de la friction cognitive il veut insister sur la stupidité qui préside à la conception de certains appareils high-tech, comme les magnétoscopes, les micro-ondes et, bien entendu, les logiciels, se servant d'analogies avec les premiers pour mieux faire comprendre comment on retrouve les mêmes travers dans ces derniers.

Le principe est simple : si vous prenez un violon, aussi difficile qu'il soit d'apprendre à en jouer, vous n'arriverez jamais à un "méta état" dans lequel les "entrées" que vous donnerez au violon le feront sonner comme un cor de chasse, une cloche ou un saxophone. Alors que si vous prenez un magnétoscope (de l'époque donc) les touches changent de sens selon la fonction : les utilisateurs en viennent à conclure que les petits boutons en plastique doivent coûter une fortune, puisqu'il n'y a que quelques malheureux boutons pour faire des dizaines de choses… Par leur nombre limité, les boutons prennent une signification différente selon "'l'état'" dans lequel se trouve l'appareil. C'est pourquoi chez beaucoup de gens (votre grand-mère, la mienne, chez le vieil oncle Robert…) le magnétoscope affiche toujours glorieusement un "12:00" clignotant : comprendre comment régler cette fichue horloge à chaque fois que le courant disjoncte est finalement bien trop compliqué que le petit avantage d'avoir l'heure à cet endroit procure.

Bien entendu le livre date un peu, et tous les magnétoscopes récents règlent leur horloge automatiquement désormais. Et pourquoi ? Parce que jamais elle n'était réglée correctement par les utilisateurs… Et pourquoi ? Parce ces quelques boutons changeant de sens selon le contexte sont un enfer ! Les autres fonctions, notamment la programmation des enregistrements passent aussi aujourd'hui par des commandes visuelles écrites sur l'écran. C'est censé être plus simple. Mais là aussi on rencontre beaucoup de « cognitive friction » !

Prenez certains micro-ondes, si vous n'avez pas le manuel sous les yeux, à quoi correspond le mode "fc-2" par rapport à "fc-3" ? ou "sc-1" et "sc-2". J'ai pourtant un micro-onde dernière génération qui fait four, chaleur tournante et tout le reste, et dont le prix était vraiment élevé, et pourtant en face avant, un mini afficheur à led ne donne que ce genre d'information ! Depuis des années que je l'ai, je suis obligé d'avoir le manuel (qui commence à partir en morceaux) à côté, car sinon, fini la cuisson automatique en mode sc-4 qui utilise la grille du bas mais pas celle du haut et la chaleur tournante et les micro-ondes à 360 W, ce qu'il ne faut surtout pas confondre avec le mode sc-3 qui lui utilise la grille du haut… Alan Cooper est un visionnaire, bien avant que des micro-ondes aussi complexes n'existent il en parlait dans son livre ! Et rien n'a changé, tout a empiré…

La "cognitive friction" est une souffrance, elle créée le rejet. Les logiciels qui sont conçus comme tous ces appareils high-tech sont des atrocités pour l'utilisateur ! Si certains de ces utilisateurs devenaient Empereur, il est certain que l'élu en question commencerait par envoyer au peloton d'exécution tous les informaticiens ayant conçu les logiciels de ce type qu'il aurait dû utiliser durant sa vie ! (personnellement je proposerai quelques noms ☺).

Programmer un magnétoscope est une opération qui fut très longtemps (et le reste parfois encore) une expérience pleine de frustration et de "cognitive friction". Jouer du violon ou du piano, jamais. Et c'est un musicien qui vous le dit. C'est dur. Parfois très dur. Mais jamais on ne connaît de cognitive friction aussi frustrante que celle que créent certains logiciels ou appareils électroniques. **Jamais**.

Peu importent les objets, et peu importe si certains exemples de Cooper ont pris de l'âge en peu de temps, ce qu'il exprime est essentiel, et *comprendre comment créer des logiciels qui n'entraîneront pas de "cognitive friction" est vraiment un besoin urgent, actuel et totalement moderne, au sens d'actuel !*

Cela nous ramène bien entendu aux interfaces utilisateurs. Et pourquoi ? Parce que dès le début je vous ai expliqué que c'est la seule chose "palpable" que l'utilisateur verra de votre travail ! Son seul contact avec votre code, son seul moyen de communiquer avec le logiciel.

Pensez-y, car les logiciels n'ont rien à voir avec les magnétoscopes : ils sont des centaines de fois plus complexes encore, proposent des dizaines, des centaines d'états internes et externes différents, réagissent différemment selon ces contextes. Les logiciels, plus ils se veulent "souples" plus ils créent de la friction cognitive. Tout le monde connaît au moins un logiciel "standard" dont le paramétrage est tellement sophistiqué qu'il faut vendre une formation ou envoyer un professionnel pour adapter le logiciel aux besoins de l'utilisateur. Un bel exemple de ce qu'il ne faut pas faire…

Les conséquences de la friction cognitive

La friction cognitive entraîne aussi son lot de problèmes. Par exemple TOUS les utilisateurs (même les plus geeks) n'utiliseront généralement qu'un sous-ensemble des possibilités du logiciel. Si on prend par exemple les produits phares d'Adobe (PhotoShop et Illustrator), à

chaque fois que je m'en sers je ferai rôtir en enfer les informaticiens pervers qui ont créé ces monstres. Essayer de dédresser une photo avec PhotoShop… Et essayer avec PaintShop Pro. Vous comprendrez ce que je veux dire.

Bien entendu cela ne veut pas dire qu'il faut supprimer des fonctionnalités aux logiciels ! Non, mais en revanche cela oblige à mieux concevoir leurs interfaces !

Parmi les conséquences de la friction cognitive il y a quelque chose de terrible : l'utilisateur se sent idiot. Et quand une chose créée ce genre de sentiment, en toute logique vous la fuyez. Personne n'aime passer pour un idiot. *On choisira donc de préférence des logiciels plus simples, n'offrant pas certaines possibilités ou une certaine "souplesse" mais dont l'utilisation semblera simple et évidente*. Devinez pourquoi tout un tas de gens achètent des Apple au lieu d'un PC ? … Imaginez un utilisateur de base devant XP qui, pour fermer l'ordinateur réclame de cliquer sur le bouton … "démarrer" !!! Le virage à 180° pris par Microsoft depuis le début de l'ère .NET était essentiel. Tardif, mais essentiel. Et c'est pourquoi aujourd'hui Microsoft met les bouchées doubles pour offrir à la fois des OS et des outils de développement qui ne créent pas de friction cognitive. Mais durant des années ils ont laissé la porte ouverte à Apple qui depuis le Mac avait compris qu'on n'éteint pas un logiciel en cliquant sur « démarrer ».

Je pourrais vous citer encore un autre exemple très simple : Prenez la « Table de caractères » de Windows 7, dernière version donc ayant pourtant subi tout le relooking de Vista et le poids de la prise de conscience par Microsoft de cet impératif de faire des UI intelligibles. Avez-vous utilisez la « Table de caractères » ? Elle n'est vraiment utile que dans un cas : les polices de caractères spéciaux, comme Wingdings. Chercher le symbole que vous voulez, certes vous pouvez faire un copier / coller, mais selon le logiciel que vous utilisez, vous voulez utilisez la fonction Windows qui permet de taper le code (ALT+code) pour l'insérer dans un texte, un logiciel de dessin par exemple qui ne gère par le copier/coller de fontes comme Word. Vous avez juste besoin du code ou de la touche équivalente… Et d'une le code dans la touche n'est pas indiqué, et de deux, oui, aussi ahurissant que cela puisse paraître, le code est indiqué en tout petit dans la barre d'état en… hexadécimal, formaté façon C, alors même que la fonction (ancestrale) de Windows permettant de taper le code au clavier réclame un code en décimal !

Cet outil pourtant indispensable de l'OS n'a pas évolué d'un poil en 15 ans. Comment, par exemple, blâmer un graphiste, qui utilise de nombreuses fontes, d'avoir acheté un Mac et non un PC sous Windows…

Hélas, Windows, même en version 7, est encore plein de telles bêtises qui, visiblement, ne choquent pas les informaticiens, ceux de chez Microsoft, et ceux que je croise tous les jours dans mes prestations d'audit, de conseil ou de formation… Je comprends dès lors fort bien pourquoi des gens comme moi sont obligés de ramer pour expliquer les fondamentaux du

Design dans le monde Microsoft. Ce que je dis aujourd'hui, les informaticiens travaillant sur Mac ou IPad le savent depuis toujours. Autre culture…

Autre conséquence de la friction cognitive : le coût exorbitant de la maintenance et du support. Vous allez me dire certains en vivent… Je connais des sociétés ou des indépendants qui vivent juste sur les formations et le paramétrage de certains logiciels "standard"… Quelle que soit la sympathie que j'ai pour certains d'entre eux, il s'agit d'une hérésie.

Enfin, l'acmé de la friction cognitive est atteinte par les informaticiens eux-mêmes. Lorsque leur logiciel est incompréhensible, lorsqu'il créée de la friction cognitive chez l'utilisateur c'est que c'est ce dernier qui est un idiot, un inculte, un paresseux qui n'a pas lu la documentation, un crétin des alpes. D'ailleurs ce sont les mêmes informaticiens qui tenteront de vous démontrer que leur paramétrage infernal est une nécessité absolue parce que, disent-ils le torse bombé par l'orgueil, leur logiciel est "souple" (ou "évolutif", ou "personnalisable") !

> *N'ayez pas cette arrogance de pisseur de lignes qui ne mérite que le chômage de longue durée : si l'utilisateur n'y comprend rien c'est que votre interface est bonne à jeter à la poubelle, c'est tout.*

Les "interfaces utilisateur" du monde réel

Le monde réel nous oblige à de constantes interactions avec des objets de toute nature, avec notre environnement, qu'il soit totalement naturel ou bien artificiel, ou entre les deux. Cela inclut les humains qui s'inscrivent dans "le décor".

Certaines choses sont simples, d'autres sont complexes. Certains objets sont durs à utiliser, d'autres se manipulent sans même y penser. Choses et objets au sens le plus large.

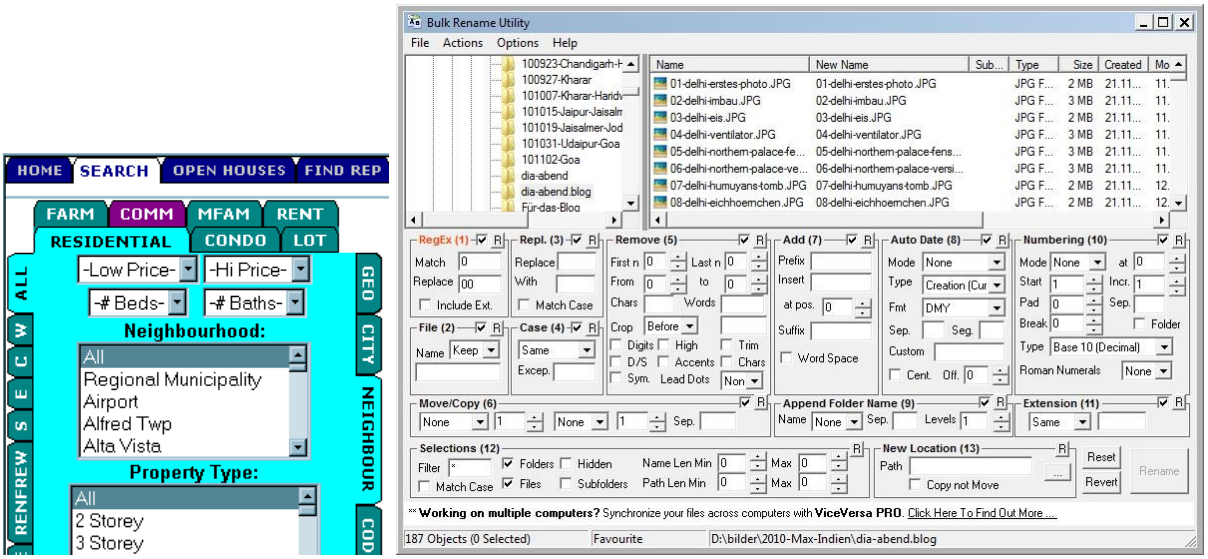
Et cela s'applique aussi aux êtres humains !

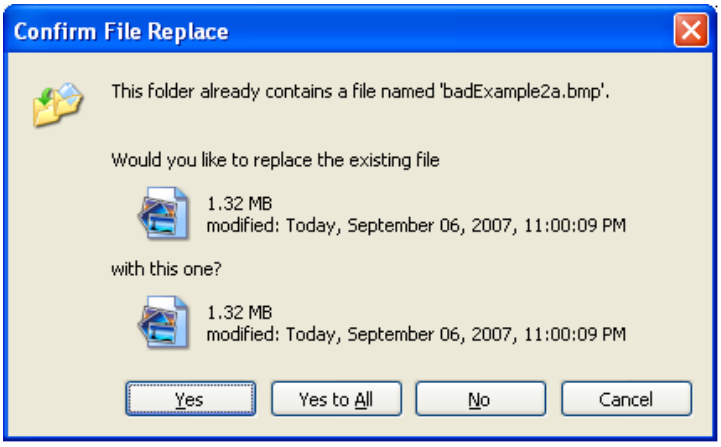
Il en résulte que les concepteurs d'interfaces utilisateur sont souvent de bons communicants. Je parle bien de Designers créant des UI. Pas des infographistes qui, bien que parfois talentueux sont rarement de bon communicants, tout comme les informaticiens d'ailleurs.

Dans la même logique, si vous améliorez votre communication, vous deviendrez certainement meilleurs pour créer des interfaces utilisateurs !

Voici quelques exemples « d'interfaces utilisateur » puisées du monde réel qui nous entoure :

Quelques exemples de friction cognitive en informatique :





“L’apprentissabilité”

Quel horrible mot même s’il existe ! Mais je n’en ai pas trouvé de plus approprié pour faire comprendre le concept…

L’une des questions qu’il faut se poser sur une application c’est de *savoir si l’utilisateur peut rapidement la découvrir et apprendre à s’en servir.*

On peut se dire que cela revient un peu à la fameuse “première impression” dont j’ai déjà parlé. En quelque sorte oui, d’apparence. Dans les faits il s’agit bien de deux choses différentes.

La première impression, comme son nom l’indique, n’est qu’une… impression. Nous nous sommes tous “fait avoir” un jour par un produit (peu importe sa nature) qui “semblait” bien conçu, simple, mais qui, à l’usage, s’est révélé décevant (et je ne parle pas des humains !).

Il a beaucoup de logiciels développés dans cet esprit. Ils sont généralement produits par des sociétés dirigées par des commerciaux… Tout dans la pub, l’apparence, le clinquant, mais pas de budget pour ce “qu’il y dedans”. C’est navrant mais cela existe. Je ne vous parle donc pas ici de ce genre de chose qui, à mon sens, frise l’arnaque.

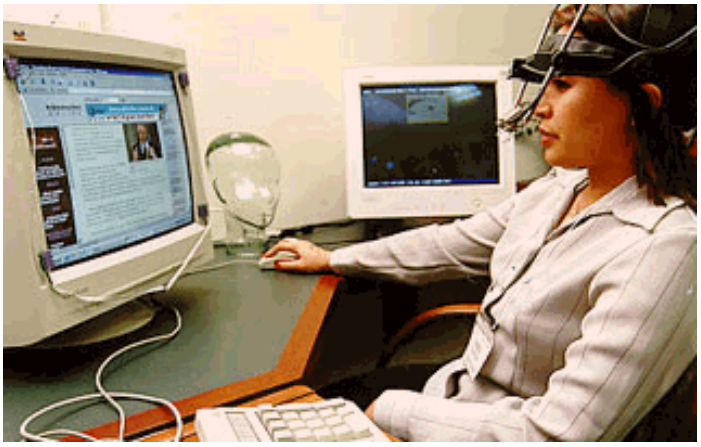
Je vous parle plutôt d’un logiciel faisant bonne impression dès le départ, mais qui à l’utilisation se révèle facile à prendre en main, accompagnant l’utilisateur dans son apprentissage, ne le faisant jamais passer pour un idiot, et même, si possible, lui donnant l’impression d’être plus intelligent qu’il ne l’est en réalité !

Car là est la clé du succès : si votre logiciel donne l’impression à l’utilisateur qu’il s’en sort facilement, alors qu’il avait peut-être une certaine appréhension, alors il vous aimera, vous bénira ! Lui qui avait peur de se sentir un peu bête, se rend compte qu’il comprend tout… Si vous arrivez à créer ce sentiment chez l’utilisateur alors votre logiciel se vendra tout seul par le bouche à oreille !

Mesurer “l’apprentissabilité”

Il existe de nombreuses études portant sur le sujet, souvent très universitaires et peu diffusées. Il existe aussi des cabinets d’experts en ergonomie capables de mesurer avec précision les mouvements des yeux des utilisateurs découvrant votre logiciel ou votre site

Web. Le prix de leurs prestations est généralement assez dissuasif, seules de très grosses sociétés pouvant s'offrir de tels services.



Mais il n'est pas interdit de faire dans la simplicité : réussir à trouver deux ou trois utilisateurs types du logiciel, les mettre devant l'écran et leur demander d'exécuter une tâche précise et habituelle pour eux (pour un comptable saisir une note frais par exemple). Prenez une montre et mesurez combien de temps cela nécessite. Testez deux ou trois options différentes de l'interface. Comparez les résultats.

> *Si vous n'êtes pas en mesure de réunir deux ou trois utilisateurs potentiels de votre logiciel, alors laissez tomber la création d'applications. Franchement. Car si vous ne pouvez pas réunir quelques utilisateurs potentiels comment avez-vous fait pour connaître le profil précis et les habitudes de ces mêmes utilisateurs alors qu'il s'agit d'un préliminaire incontournable ?*

Vous le voyez, **l'échec s'inscrit souvent dès le départ** : une conception coupée des utilisateurs ne pose pas seulement que des problèmes d'interface, mais de cahier des charges, de fonctionnel aussi… *Un logiciel conçu dans l'éloignement le plus total des utilisateurs ne pourra jamais être un bon logiciel*. C'est ainsi. A la trappe l'idée du « génial inventeur » créant un super soft tout seul sur son portable posé sur la table de la cuisine !

Mais admettons que ce point essentiel est compris et que vous avez eu à cœur d'être proche des utilisateurs dès l'analyse, dès le cahier des charges, et que maintenant que vous en êtes à la conception de l'interface il vous est donc facile de revoir les mêmes gens et de leur demander de faire quelques tests. Supposons que vous puissiez faire ces mesures, ces comparaisons pour aider à prendre de meilleures décisions.

Il n'en reste pas moins vrai qu'au moment de la prise en main l'utilisateur ne sera qu'un débutant, découvrant tout. Il réclamera donc un **support permanent**, ce que le logiciel doit être capable de fournir de lui-même. Vous ne serez pas à côté de tous les utilisateurs pour leur donner des conseils…

D'où l'importance d'intégrer des dialogues supplémentaires, des explications, des info-bulles, des choix du type "débutant / expert" modifiant l'apparence et offrant plus ou moins d'options, sans oublier les "experts" ou "wizards" permettant de réaliser une tâche donnée en étant guidé. N'oubliez pas qu'*un bon "expert" ne doit pas être une boîte noire*, ce qu'on voit trop souvent ! Un bon expert est conçu pour guider l'utilisateur vers son but, mais *en même temps il doit être capable de lui faire comprendre comment atteindre ce but en se passant de l'expert !*

Un logiciel dont "l'apprentissabilité" est excellente répondra au minimum à 80% des besoins de l'utilisateur.

Arrivé à ce point de véritable "souplesse" (et non pas la fausse souplesse-excuse des usines à gaz), vous aurez développé un vrai logiciel moderne. L'intégration de la conception d'une interface utilisateur pensée, avec l'aide d'un vrai Designer, ne vous apparaitra plus alors comme une "mode", un "gadget". C'est que vous aurez compris ce que doit être un logiciel bien conçu de nos jours…

"L'utilisabilité"

Avec "*apprentissabilité*" je ne suis plus à un mot abominable près hein… Le Design ne s'encombre pas d'esthétisme dans son jargon !

L'utilisabilité n'existe pas en français, mais cela existe dans la réalité. Concernant ce point important il convient de réfléchir à la question suivante :

A quel point l'interface utilisateur est-elle intuitive et productive pour un utilisateur "moyen" ?

Cela peut aussi se mesurer. Cette fois-ci en prenant pour cobaye un utilisateur avancé et en lui demandant, toujours la même chose, d'effectuer une tâche habituelle. Bien entendu on prendra plutôt deux utilisateurs, voire quatre ou cinq, pour que les mesures aient un sens.

Une interface utilisateur ayant une haute "utilisabilité" permet aux utilisateurs d'effectuer toutes les tâches courantes de façon directe et sans détour. Cela implique dans certains cas que le workflow puisse être personnalisé.

Il faut faire attention à ne pas confondre “utilisabilité” et “apprentissabilité”. La confusion est fréquente car l’un va souvent avec l’autre, mais il s’agit bien de deux domaines distincts. Deux objectifs différents qui ont leurs impératifs propres et leurs mesures distinctes.

Les tests d’utilisabilité réclament un panel de testeurs le plus large possible pour pouvoir tirer des conclusions. Les évolutions, les variantes, tout cela doit être mesuré spécifiquement et séparément. C’est vraiment une partie difficile dans la conception d’un logiciel car cela peut coûter très cher à organiser. On comprend facilement pourquoi les plus grands éditeurs du monde préfèrent généralement sortir des versions “bêta” … A défaut d’un test encadré scientifiquement, les “bêta” permettent de s’offrir un large panel d’utilisateurs pour un coût ridicule ! Les plus malins diffusent durant quelques jours une bêta offrant certains choix d’interface, et pendant les jours suivant une version offrant un choix différent. Tout le monde croyant télécharger la même version bêta bien entendu. En fonction des retours du premier et du second groupe vous saurez quel choix est le meilleur…

La clarté

Enfin un mot qui existe ! Et pourtant celui-là, ironie du sort, son orthographe ne m’a jamais paru claire ! J’ai toujours l’impression qu’il manque un “e” entre le “r” et le “t”. Comme quoi le mot n’est pas la chose qu’il désigne (Cf. « Ceci n’est pas une pipe » de Magritte dans le même esprit). L’orthographe est d’ailleurs, dans notre langue, un excellent exemple de friction cognitive ! Beaucoup de règles qui changent selon le contexte avec tout autant, si ce n’est plus, d’exceptions…

Beaucoup d’interfaces surchargent l’utilisateur de données qui n’ont pas toute la même pertinence. Il est en réalité essentiel de viser une conception « pacifiée », « calme ». Le look « Metro » que Microsoft a produit pour Windows Phone 7 et qui sera ré exploité sous Windows 8 est un exemple d’une telle recherche de « calme ». Circulation du blanc ou du noir pour aérer la présentation, des objets aux formes simples, reconnaissables, que l’œil repère, classe, trie facilement. Des fonds évocateurs de nature, de paix intérieure. Metro n’est pas l’exemple parfait, mais c’est une des façons d’appliquer de bonnes règles de conception visuelle.

Il faut aussi guider l'utilisateur en créant des groupes visuels qui ont un sens. Les groupes peuvent être logiques ou fonctionnels. Les horizontales doivent être exploitées pour tout ce qui est en rapport avec un espace, une durée, les verticales pour ce qui est apparenté à des énumérations, que ces verticales et horizontales soient matérialisées ou uniquement suggérées par la mise en page (je n'aborderai pas ici les principes de design en pratique, mais c'est ce qu'on appelle les « closures », créer des formes sans les dessiner).

> *La règle d'or en la matière est qu'il est bien plus important de se concentrer sur la clarté du visuel que d'ajouter des infos-bulles de partout pour expliquer ce qu'on ne comprend pas tout de suite !*

La liberté d'action

Avoir le choix, avoir une grande liberté d'action est une chose essentielle, l'utilisateur ne doit jamais se sentir « piégé » dans une procédure dont il ne peut plus s'échapper.

Mais il y a plus important encore : la liberté de pouvoir changer le comportement du logiciel. C'est la fameuse « souplesse » qu'on retrouve dans des logiciels aux paramètres pléthoriques, mais mise en scène correctement.

Selon le niveau de compétence de l'utilisateur, certaines options sont ou non importantes, et l'accès même à ces options suit cette logique.

Par exemple, un débutant (sur le logiciel considéré) n'aura pas besoin d'avoir accès à de nombreuses options. Il doit pouvoir travailler immédiatement et réaliser les manipulations qu'il souhaite, sans friction cognitive et sans avoir à répondre à des listes d'options dont il ne comprend pas (encore) le sens.

Je vois souvent des logiciels qui, au moment de l'installation (ou de la première utilisation) affichent de nombreux choix qui n'ont pas encore de sens pour l'utilisateur. Comment décider de la mise en page par exemple avant même d'avoir testé celles qui sont proposées ? Visual Studio pose ce genre de question, c'est une erreur. Quelle différence y-a-t-il entre la mise en page « VB.NET » et « C# » ? Vous le savez, vous ? Ne travaillant pas en VB.NET je n'ai jamais choisi cette option et après des années de pratique de VS, je ne sais toujours pas ce qui se passerait si je la choisissais ! Ahurissant ! Du coup je choisi toujours « environnement C# » mais il est fort possible qu'une autre option soit mieux adaptée à mes goûts et ma façon de travailler. Hérésie et Frustration sont les conséquences d'un mauvais Design…

La gestion des options est essentielle : un bon logiciel est souvent paramétrable pour s'adapter aux préférences de l'utilisateur. Mais il doit le faire de façon progressive, graduée, en accord avec les connaissances de l'utilisateur, son habilité.

Cela passe aussi par une phase délicate pour le développeur : choisir les bonnes valeurs « par défaut » pour le débutant.

Il faut se convaincre que si le développeur ne sait pas faire ces choix, alors l'utilisateur débutant le sera généralement encore moins !

Toutes les options doivent être regroupées par thématique, être facilement accessibles et surtout être compréhensibles.

Prenez les options d'Internet Explorer, surtout IE9… C'est un enfer. Il y a trop de lignes pour la petite boîte qui les affiche, aucun moyen de chercher une option, la plupart ont des noms qui n'aident pas à « réellement » savoir ce que cela va faire, et encore je suis informaticien ! Encore un exemple parfait de ce qu'il ne faut pas faire ! Les options de Visual Studio sont déjà mieux structurées, mais ne sont pas toujours plus parlantes, sans parler de l'affreux système de paramétrage du clavier et des commandes.

Comme quoi, il n'y a pas besoin d'aller chercher bien loin les mauvais élèves[5] !

Revenons aux options, en général. Il faut se mettre dans la peau de l'utilisateur. Certains choix deviendront évidents à un utilisateur expérimenté. Il saura où trouver les options « avancées » et les manipuler sans problème. Pour cela le logiciel doit avoir été correctement pensé et offrir différents niveaux d'options.

Utiliser les bonnes métaphores

L'art de la métaphore, comme de la parabole est une sorte de sixième sens. Vous l'avez ou non. Si vous ne l'avez pas, faite appel à un Designer compétent qui saura trouver les métaphores qui font sens.

Les métaphores dans les interfaces sont essentielles car elles font venir une partie du monde réel connu dans le logiciel. Elles introduisent un sens de familiarité entre l'application et l'utilisateur.

Une métaphore réussie se suffit à elle-même, elle supprime le besoin d'un libellé, celui d'une bulle d'aide. Elle peut se passer totalement de texte.

La plus célèbre et parmi les plus réussies, est certainement la métaphore de la corbeille. On comprend immédiatement ce que c'est et à quoi cela sert. Dans le pire des cas on apprend très vite à quelle fonction du logiciel elle fait référence, et on sait s'en servir en quelques secondes.

D'autres métaphores sont tout aussi connues et utilisées régulièrement : la notion de « bureau », de « répertoire », de « document » par exemple.

Il ne faut pas abuser des métaphores. D'abord parce qu'elles sont loin d'être universelles. Il est plus facile de traduire un bout de texte pour internationaliser un logiciel que de refaire toute l'interface en prenant soin d'utiliser des métaphores en phase avec la culture de chaque pays…

[5] A noter que si je « tape » volontiers sur la tête de Microsoft, c'est tout simplement parce que j'en connais bien les produits car j'en suis globalement satisfait. Nul doute, et que le lecteur en soit convaincu, que si je travaillais sur Mac j'aurai certainement autant d'incohérences à vous livrer. J'ai eu un Mac G3, et de cette simple expérience j'aurai déjà toute une liste d'âneries et de trucs générant de la friction cognitive à vous livrer !

Le danger étant qu'une métaphore « convenable » ou « adaptée » dans un pays, une culture, devienne au mieux « curieuse » voire choquante dans une autre culture !

Prenons un petit exemple. En Italie se tapoter le lobe de l'oreille avec l'index signifie (en général en connivence avec son interlocuteur) que la personne dont on parle, ou qui vient juste de passer, est un homosexuel. Dans les régions très au sud, comme la Sicile, au machisme légendaire, un tel geste est donc porteur d'un sens très péjoratif.

Imaginons un instant que vous ayez créé un logiciel qui diffuse du son et que vous ayez tourné quelques petites vidéos pour expliquer les fonctions. On pourrait supposer une séquence où l'acteur se tapote l'oreille comme je l'ai expliqué pour signifier « j'entends mal, comment montez le son ? ».

Je vous laisse comprendre ce qu'une telle métaphore aurait comme effet lorsque vous diffuserez votre logiciel en Italie !

Refaire toutes les vidéos d'aide de votre logiciel risque d'être coûteux, bien plus que de traduire un texte. Plus grave : encore faudra-t-il être capable de déceler ces « anti-métaphores » pour chaque pays où vous diffuserez votre logiciel…

Vidéo, photos, dessins, animations, toutes ces manifestations visuelles peuvent avoir un sens déplacé dans les cultures que vous ne maîtrisez pas. Ne l'oubliez jamais !

Mais il y a aussi les métaphores qui « poussent le bouchon trop loin » notamment en se voulant tellement proches de la réalité qu'elles en deviennent néfastes pour l'application. C'est le fameux skeuomorphisme.

Prenons un exemple simple d'un domaine que vous ne connaissez peut-être pas, mais vous allez comprendre. Dans le monde des synthétiseurs modulaires, ceux des années 70/80 (le célèbre Moog modulaire par exemple) toute la modularité tenait dans le fait que ces

appareils étaient dotés de dizaines, voire centaines de prises Jack femelles et qu'à l'aide d'un cordon Jack Mâle/Mâle on pouvait établir des connexions entre les différents modules. Pour avoir connu cette époque je peux vous assurer que c'était le summum du nec plus ultra du jeune geek ! Je garde d'ailleurs religieusement deux cordons de mon dernier système modulaire… C'est pour dire.

Bref, un modulaire en cours de fonctionnement ressemblait aux anciens (très anciens)

standards téléphoniques des années 50. Kitch et geek (mais il fallait parfois des heures pour fabriquer un son différent, difficile à utiliser sur scène !).

Depuis que la puissance des ordinateurs a rendu la chose possible, de très nombreux synthétiseurs « cultes », comme le Moog Modulaire, ont été simulés en numérique pour être utilisés dans des séquenceurs ou arrangeurs (Cubase ou Live par exemple). Certains éditeurs ont été tellement loin dans le réalisme qu'on peut brancher des cordons Jack comme sur le « vrai » synthétiseur… Réalisme amenant au même problème : quand de très nombreux cordons sont branchés on ne voit plus rien de la face avant du synthétiseur, on ne comprend plus quel cordon va où… Les cordons sont si bien simulés qu'ils bougent et se tortillent comme des vrais. Une horreur ! La capture écran enchâssée dans ce texte vous montre un autre logiciel musical utilisant la même métaphore ultra réaliste des cordons…

Cela « jette » en démonstration. Mais à l'usage c'est plus que pénible. Mauvais choix de métaphore, mauvaise balance entre la reproduction du monde réel et la fameuse utilisabilité du logiciel que j'évoquais plus haut.

Le musicien veut une reproduction parfaite des possibilités et du son de la machine originale, mais pas de ses enquiquinements ! J'attends le jour où certains pousseront la plaisanterie jusqu'à obliger l'utilisateur à attendre presque une heure devant son écran avant de se servir de son synthétiseur… En effet, ces machines étaient analogiques et la « mise en température » était longue. Si vous faisiez de l'enregistrement multipiste, il fallait être sûr que tous les composants soient chauds avant de faire le « La », sinon la fréquence bougeait de trop pendant la session de travail et votre morceau de musique final sonnait désaccordé. Le choix des cordons qui gênent la vue et rendent peu lisibles les branchements effectués est aussi stupide que le serait de simuler ce temps de mise en température des vraies machines de l'époque… *La métaphore ne doit pas reproduire les faiblesses du monde réel.*

En matière de métaphore il faut donc avoir la main légère et le coup de crayon précis. Parvenir à l'universalité est impossible ou presque.

Le réglage du thème de Windows 7 ou celui de l'image de fond sont en revanche des métaphores réalistes tout à fait réussies. On comprend tout de suite qu'on regarde une simulation de son propre écran tel qu'il sera si on applique les changements.

Représenter un écran par un écran est une métaphore évidente et facile à mettre en œuvre, sans risque de mauvaise interprétation. Il existe bien plus de cas où trouver la bonne équivalence est un casse-tête. C'est pour cela que l'aide d'un Designer créatif est plus qu'indispensable !

Les utilisateurs ne lisent rien !

Certaines règles dans la fabrication des interfaces sont si évidentes qu'on les oublie… Par exemple tout le monde sait que les utilisateurs ne lisent jamais les messages. Alors pourquoi peut-on encore voir des logiciels afficher des tartines à l'écran ? Ce refus de la réalité est à la limite de la maladie mentale et de l'autisme, il faut faire attention …

La plupart des boîtes de dialogue se ressemble tellement et est tellement utilisée qu'**elles endorment l'utilisateur**. Jusqu'à ce qu'il comprenne qu'il vient par habitude de cliquer sur « oui » mais que la question n'était pas « voulez-vous sauvegarder le fichier » mais « confirmez-vous l'effacement du disque » !

Bannissez les boîtes de dialogue ! Evitez-les le plus possible. Même « relookées », même « jolies », même en accord avec le thème visuel de l'application !

Si l'interface est bien conçue, l'utilisateur ne risque pas de cliquer sur « supprimer » au lieu de « sauvegarder ». Lui demander de confirmer « voulez-vous supprimer… » est stupide. D'autant qu'un bon logiciel doit permettre l'erreur, donc le « undo ». Sinon on en arrive aux dialogues de confirmation incessants de Vista qui ont usé les nerfs des plus patients. Ou au message systématique qu'on se « prend » (tel un poing en pleine figure) à chaque fois qu'on passe une vidéo en plein écran sur Internet, juste pour vous dire … je ne sais plus quoi d'ailleurs ! Preuve de l'inefficacité totale du procédé (comme tout bon utilisateur j'ai juste mémorisé ce qui m'intéresse : il faut taper sur Escape pour sortir de ce mode).

Cela est stupide car l'utilisateur est endormi par tous les dialogues et qu'il y répond machinalement, **donc en annihilant l'effet recherché**, c'est stupide parce que **cela gêne l'utilisateur**, c'est stupide parce que cela « casse le rythme », entrave la fluidité des actions de l'application.

Si votre logiciel est obligé d'afficher un dialogue, concevez-le comme une publicité : le message doit être court, sans équivoque, percutant. Les tournures alambiquées, les questions interronégatives et autres placards d'explications jamais lus sont plus qu'à éviter : considérez cela comme une interdiction totale, absolue, de la même nature qu'il est interdit d'éteindre son ordinateur en arrachant la prise électrique !

La formulation d'un dialogue doit être positive, claire, et donc basée sur le contexte. Si vous affichez le même dialogue dans deux situations différentes dites-vous que vous êtes dans l'erreur !

Bien entendu, comme pour une publicité ou un discours commercial, toutes les tournures ou mots ayant un sens négatif sont à éliminer. Les messages du type « Fonction Interdite ! » sont des abominations. Tout comme terminer les messages d'erreur par des points d'exclamation. Quelle sale manie ! « Imprimante indisponible ! », « Réseau non détecté ! ». Oh la-la … On se croirait face à une maîtresse revêche ancienne école, prête à vous taper sur les doigts avec sa règle en bois !

Les messages, même d'erreur, doivent être ponctués d'un simple point. Nul besoin non plus de sons d'alarme imitant le Nostromo[6] avant son autodestruction ! Ni même d'icônes angoissantes genre tête de mort ou symbole nucléaire !

Un bon logiciel est un logiciel qui a été conçu comme je le disais aussi plus haut pour inspirer le « calme », la « sérénité ». Vous imaginez Maître Yoda se mettre à courir dans tous les sens en poussant des petits cris aigus parce que Dark Vador est dans le secteur ? Franchement ?

[6] Le cargo spatial dans lequel se déroule le film Alien (le 1).

Non. Un bon logiciel non plus. Il doit donner l'impression à l'utilisateur que tout est sous contrôle. Un « plantage » ? Il s'en gausse ! Tout juste le signale-t-il calmement, sans affoler tout le monde, en ayant pris soin de créer un Log détaillé et de tenter de revenir à la situation précédente sans faire perdre tout le travail de l'utilisateur. Et s'il doit s'arrêter, qu'il le fasse dans la sérénité, en se relançant automatiquement et en tentant de réafficher les informations telles qu'elles étaient avant le plantage.

Ça c'est de la programmation de haute voltige, là il peut y avoir fierté du développeur.

Conclusion

L'informaticien, celui que je rencontre le plus souvent, est trop souvent hélas totalement ignorant de ce bouleversement qui tel un tsunami pousse toute la production de logiciels vers le Design d'UI intelligentes et agréables. Vers une UX réfléchie et agréable.

Pire, certains se rient de tout cela. Se moquant peut-être de ce qui leur fait peur, de voir leur métier leur échapper, se diriger dans une direction où ils savent, au moins inconsciemment, qu'ils n'ont aucune compétence…

Pourtant le Design n'est pas tant être capable de dessiner la Jaconde que de savoir en imaginer les traits et la posture…

Et tout informaticien peut acquérir les compétences qui lui permettront de concevoir des interfaces ergonomiques et agréables, quitte à s'associer à un vrai Designer ou même un simple infographiste pour l'aider dans la partie purement visuelle, dans le dessin qu'il ne saura jamais réaliser lui-même. Créer un concept ne réclame pas de savoir le réaliser. Croyez-vous que Philippe Starck sache façonner lui-même les objets qu'il conçoit ? Qu'il maîtrise la ferronnerie, l'extrusion des plastiques, l'art du menuisier, du tailleur de pierre, du maître verrier ?

Bien heureusement, j'en croise d'autres qui ont pris conscience de ce mouvement et qui s'y intéressent, même confusément.

Pour certains c'est une simple « curiosité », ils regardent tout cela avec un intérêt détaché comme si « eux » n'étaient pas concernés par ce changement… Mais c'est un bon début !

L'informatique est un outil, un simple outil. Comme une pelle ou un marteau.

La seule différence est dans le fait que les outils classiques permettent d'étendre la force ou l'habilité physique, alors que l'informatique est un outil destiné à étendre la force et l'habilité intellectuelle.

Un marteau est bien plus dur qu'une main, et plus lourd, mieux adapté que celle-ci pour enfoncer un clou. Mais ce n'est qu'un prolongement de la main qui le tient. Mieux, le marteau n'est qu'une main, façonnée par l'esprit de l'homme pour être adaptée à une tâche. Une greffe temporaire.

Un ordinateur est bien plus précis qu'un cerveau humain, il ne se lasse pas des tâches répétitives, là où le cerveau perd sa vigilance par habitude et lassitude. Il n'est lui aussi qu'un

prolongement, celui de l'esprit qui l'utilise, une extension de sa mémoire, de ses capacités de calcul.

L'informatique n'est qu'un outil. Tous les outils, depuis le jour de leur création, tous, ont été voués à évoluer, à se perfectionner. C'est dans la nature de l'humain de transformer un caillou en outil pour casser ou tailler d'autres outils, puis de transformer ce caillou en marteau, et enfin d'ajouter à ce marteau une partie gommée sur le manche pour mieux le saisir, d'équilibrer son poids pour qu'il ne fatigue pas le poignet : de l'outil naturel, le caillou, l'humain arrive à un objet composite complexe, utilisant des matériau différents réclamant chacun une maîtrise technologique différente, à une évidente prise en compte du confort de son utilisateur. Pas pour la « beauté » ou l'esthétique, mais d'abord pour lui-même, son propre confort égoïste. Puis celui des autres, parce qu'après tout, l'humain est avant tout un animal social.

Pourquoi cette évidence échappe-t-elle autant aux informaticiens lorsqu'il s'agit des logiciels ?

Soyons lucides. Il est évident que l'informatique ne pouvait, comme tout outil, en rester à ses formes et apparences de ses débuts.

Aujourd'hui, tel le papillon émergeant de sa chrysalide, l'informatique se dépouille de ses écrans verts, de ces grilles de valeurs numériques rébarbatives, de ses lourdeurs, de son manque d'adaptation aux utilisateurs.

Soyez l'entomologiste de ce nouveau monde en prenant soin de ce papillon qui ne réclame que votre aide pour s'envoler…

Bon Développement !

Vidéo Session DES150 en ligne (concevoir des interfaces utilisateur efficaces et vendeuses)

Lors des Techdays Microsoft (peu importe de quelle année, cela n'a guère d'intérêt ici), j'y ai présenté la session DES105 sur le Design. Tous les ans Microsoft propose en ligne les Webcast des sessions. En attendant cette probable diffusion je vous propose de visionner cette conférence immédiatement si vous n'avez pas pu y assister !

Lien direct vers la vidéo youtube

Nota : il s'agit d'un enregistrement "de test" effectué avant les Techdays et non de la session live du 8 février. L'avantage est que le son est meilleur et que le discours est débarassé des pauses ou private jokes avec l'audience présents dans la version Live ce qui peut être plus intéressant pour une écoute hors contexte.

Sommaire

Concevoir des Interfaces Utilisateur efficaces et vendeuses

Conférence Niveau 100

Catégorie DESIGN, ERGONOMIE ET INTERFACE NATURELLE

- Connaitre ses utilisateurs
- L'utilisateur
  - Les buts de l'utilisateur
  - Les deux types d'utilisateurs
  - L'habileté
  - Les schémas directeurs
  - Les utilisateurs ne lisent rien !
- Cognitive Friction
- Les "interfaces utilisateur" du monde réel
- L'apprentissabilité (learnability)
- L'utilisabilité
- La clarté
- La liberté d'action
- L'universalité de ces concepts
- Utiliser les bonnes métaphores
- Metro, un langage de design à maitriser
- Metro sur WP7
- Metro Style Windows 8

Le but de cette présentation n'est bien entendu pas de faire un cours de Design, en une heure cela ne serait pas sérieux, mais de faire un tour d'horizon de ce qu'est le Design aujourd'hui dans les applications modernes, pourquoi il est essentiel, comprendre un minimum les utilisateurs et les grandes lignes à respecter pour éviter l'échec.

Si vous n'y connaissez rien, ce n'est pas grave, cette conférence est faite pour vous ! (niveau 100 : débutant, c'est écrit dessus !).

Je ne vous mentirais pas en affirmant qu'à la fin de la conférence vous saurez "concevoir des interfaces utilisateurs efficaces et vendeuses" comme par magie. Non. Je vise seulement à vous sensibiliser aux nombreuses facettes du Design appliqué aux applications pour vous mettre sur la voie qui vous permettra d'atteindre peut-être un jour cet objectif. Ce n'est déjà pas si mal !

A bas les grilles !

A bas les grilles ! De tout genre et de toute provenance !

Ahhhh ça fait du bien :-)

Un petit coup de gueule en passant donc… Je lisais un forum MSDN et encore des questions sur les grilles, les DataPager, etc…

Alors soyons clairs : Je hais les grilles comme Jean Yanne haïssait les départementales dans un sketch aussi vieux qu'incontournable. Et encore lui jouait-il l'indignation et la répulsion, un rôle d'acteur. Moi je hais *réellement* les grilles. Pour être exact *je déteste l'utilisation qui en est faite*, les grilles ne sont que des objets et on ne peut leur en vouloir… (Je parle des grilles de données et de toutes leurs cousines bien entendu).

Pourquoi tant de haine ?

Parce que sauf en de rares occasions (et je dis rares, au sens le plus restrictif du terme) les grilles qu'on place dans les applications ne sont que la traduction d'une énorme paresse du développeur soit, tout aussi souvent, d'un travail bâclé de l'analyste qui n'a pas pris le temps de comprendre *ce dont avait réellement besoin l'utilisateur*.

"Alors dans le doute, hop! j'te colle une grille, comme ça y'a tout ! "

Le problème n'est pas récent, la pression monte autant que la moutarde à mon nez depuis des lustres, au sens propre, donc depuis plusieurs fois 5 ans. Autant dire que j'ai les narines qui piquent !

Il y a deux principaux types de grilles :

La grille fourre-tout

C'est la plus courante. On ne sait pas ce que veut l'utilisateur, ni comment il se servira du logiciel, alors on balance une grille avec cinquante colonnes, voire plus, il n'aura qu'à faire son choix et à passer son temps à scroller… On s'en fout, le développeur n'est pas obligé d'utiliser les trucs affreux qu'il pond parfois (et c'est bien dommage !).

Le pire c'est que "ça" pompe des mégas sur les réseaux ces saletés ! Qu'il s'agisse de celui de l'entreprise ou d'internet quand les données proviennent de services distants. Car si sous les pavés il y a la plage, *sous la grille il y des requêtes souvent aussi abominables que les grilles elles-mêmes*… Des requêtes remontant des centaines, des milliers d'enregistrements (ou d'objets) !

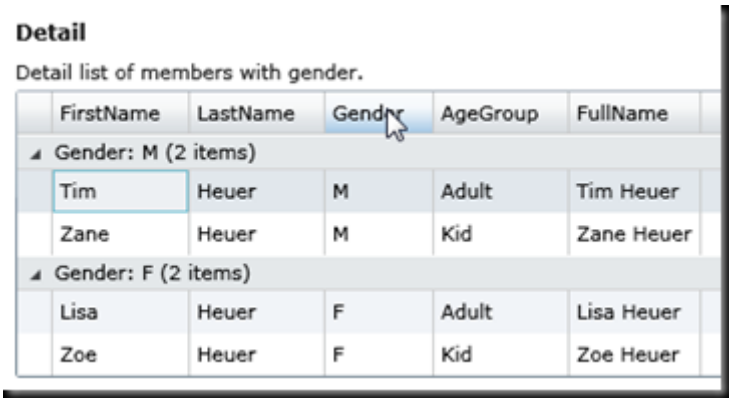
Mais l'informaticien qui aime les grilles se moque de ces problématiques. Autant que la consommation mémoire de ses frivolités (car toutes ces lignes en mémoire, virtualisées ou non, ça consomme comme un poivrot au bar du coin dès potron-minet).

Autant le dire tout de suite : Je n'ai jamais vu un seul utilisateur lire plus de deux ou trois pages dans une grille.

Non, je rêve à voix haute en fait, j'ai hélas vu trop souvent de pauvres utilisateurs s'esquinter les yeux à parcourir des dizaines de pages car il n'y avait *aucun autre moyen de trouver l'information* ! Vous dire ce qu'ils pensaient des informaticiens ne serait pas convenable pour les lecteurs chastes qu'un langage vert choquerait… A peu près la même chose que ce que vous pensez des garagistes ou des dentistes qui achètent 10 euros des prothèses fabriquées en Chine et qui vous les vendent 1500 euros sans la pose (rhétorique de garagiste, c'est fait exprès).

> *La grille fourre-tout est la mort du Design, de la User Experience, c'est le degré zéro de la bavure technique pas même un chef-d'œuvre de nullité, juste la goutte qui fait déborder le vase dans lequel se soulage le pisseur de lignes (et l'analyste du dimanche) !*

Une grille moche :



La grille sapin de Noël

"Elle est fraîche elle est belle ma grille ! Achetez-en !"

La grille sapin de Noël est peut-être celle qui me dégoutte le plus. La grille fourre-tout est nulle, moche, inutile, mal pensée, mais au moins c'est du travail de sagouin et ça se voit tout de suite.

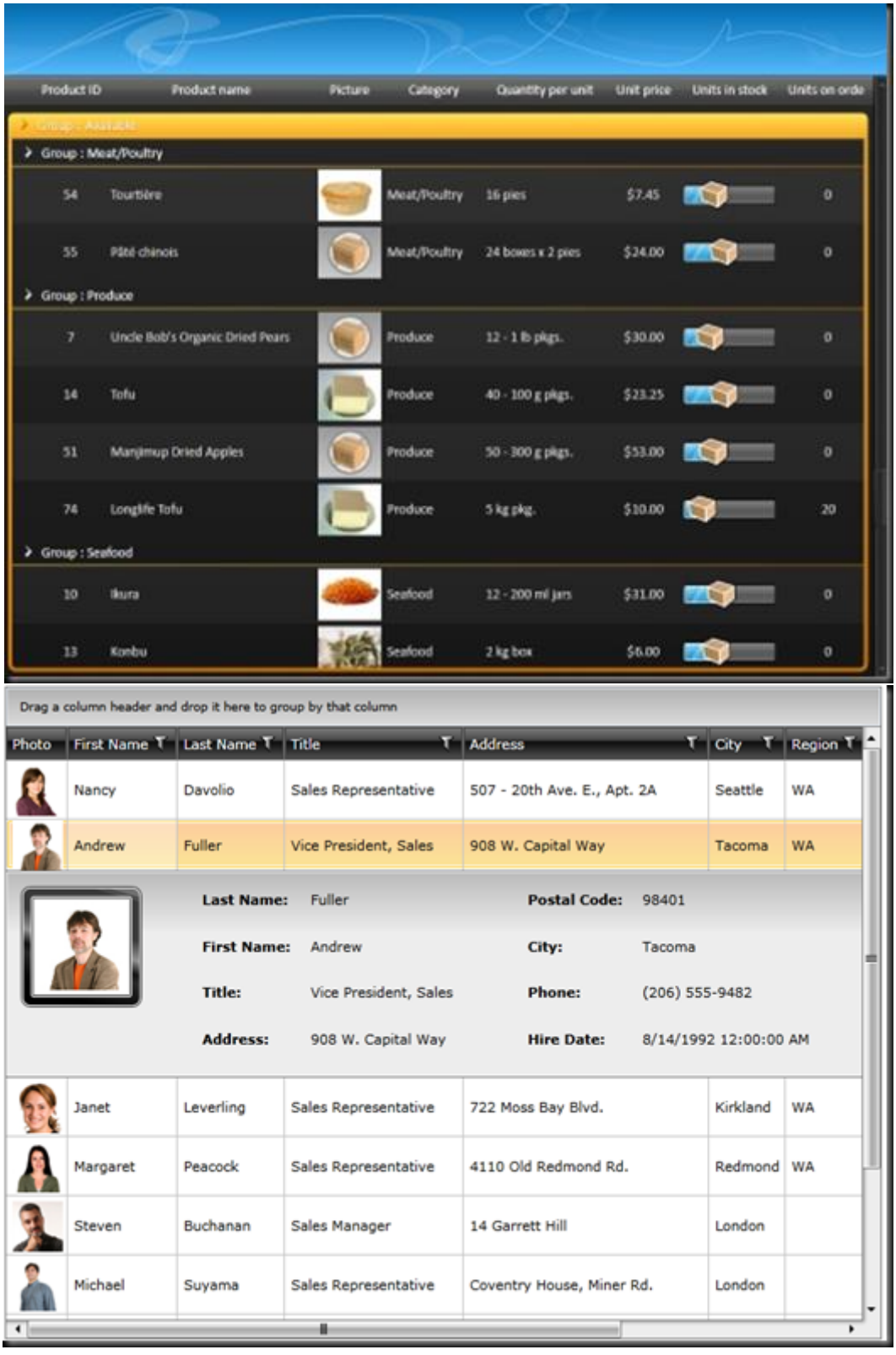
La grille sapin de Noël c'est plus vicelard, plus pervers. Il y a des types qui passent leur temps à développer des grilles pas possible, des trucs à sous-niveaux rétractables et double piston inversé.

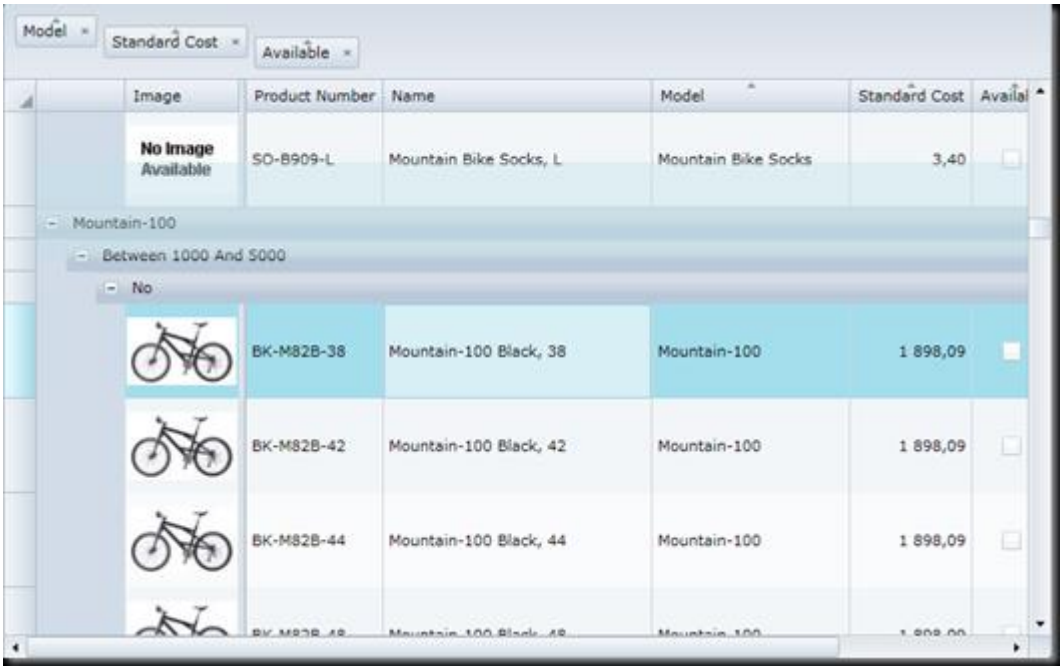
Il y en a même qui fournissent des "skins" avec. Vous pouvez changer la couleur du sapin de Noël, et même laisser l'utilisateur choisir et changer quand il veut. Génial !

En dehors du fait que ces grilles sont congénitalement frappées des mêmes tares que les grilles moches, à savoir qu'elles traduisent une mauvaise analyse doublée d'une programmation paresseuse, elles sont impossibles à utiliser sans fournir un manuel de 200 pages. Autant pour le développeur d'ailleurs que pour l'utilisateur. Le développeur c'est bien fait pour lui. Mais le pauvre utilisateur ? Qu'a-t-il fait pour mériter un tel châtiment ? Rien. Le pire c'est que le développeur est généralement fier de son œuvre.

> *Les grilles sapin de Noël sont donc aussi nuisibles que les grilles moches, avec en plus une lenteur (d'exécution et de chargement) généralement proportionnelle à la liste des fonctionnalités extraordinaires qu'elles exposent.*

Des grilles sapin de Noël (avec sous niveaux, groupes, modèle TDI 16 soupapes) :


Product ID
Product name
Picture
Category
Quantity per unit
Unit price
Units in stock
Units on orde
Group : Meat/Poultry
54 Tourtière Meat/Poultry 16 pies $7.45 0
55 Pâté chinois Meat/Poultry 24 boxes x 2 pies $24.00 0
Group : Produce
7 Uncle Bob's Organic Dried Pears Produce 12 - 1 lb pkgs. $30.00 0
14 Tofu Produce 40 - 100 g pkgs. $23.25 0
51 Manjimup Dried Apples Produce 50 - 300 g pkgs. $53.00 0
74 Longlife Tofu Produce 5 kg pkg. $10.00 20
Group : Seafood
10 Ikura Seafood 12 - 200 ml jars $31.00 0
13 Konbu Seafood 2 kg box $6.00 0
Drag a column header and drop it here to group by that column
Photo
First Name
Last Name
Title
Address
City
Region
Nancy Davolio Sales Representative 507 - 20th Ave. E., Apt. 2A Seattle WA
Andrew Fuller Vice President, Sales 908 W. Capital Way Tacoma WA
Last Name: Fuller
First Name: Andrew
Title: Vice President, Sales
Address: 908 W. Capital Way
Postal Code: 98401
City: Tacoma
Phone: (206) 555-9482
Hire Date: 8/14/1992 12:00:00 AM
Janet Leverling Sales Representative 722 Moss Bay Blvd. Kirkland WA
Margaret Peacock Sales Representative 4110 Old Redmond Rd. Redmond WA
Steven Buchanan Sales Manager 14 Garrett Hill London
Michael Suyama Sales Representative Coventry House, Miner Rd. London



Les bonnes grilles, ça existe ?

Oui : une bonne grille est une grille morte.

Celles qui frétillent encore un peu peuvent passer si elles sont suffisamment occis.

C'est à dire lorsqu'elles se résument à une page ou deux de 10 à 15 lignes maximum et qu'elles servent à *retourner les premières informations qui correspondent au vrai besoin de l'utilisateur dans le contexte donné*. Que cette adaptation soit automatique (le luxe !) ou bien qu'elle découle d'une saisie de critères par l'utilisateur.

Bref, **une bonne grille est une grille bien utilisée**, c'est à dire qu'elle ne sert qu'à présenter **peu d'informations** déjà **filtrées** avec le **minimum de colonnes** nécessaires (celles qui sont **utiles** pour que l'utilisateur puisse faire son choix).

> *Une bonne grille est ainsi une grille de sélection. C'est à dire que sous XAML on peut systématiquement la remplacer par une Listbox et un DataTemplate adapté. Exit les grilles donc.*

Les exceptions qui confirment la règle

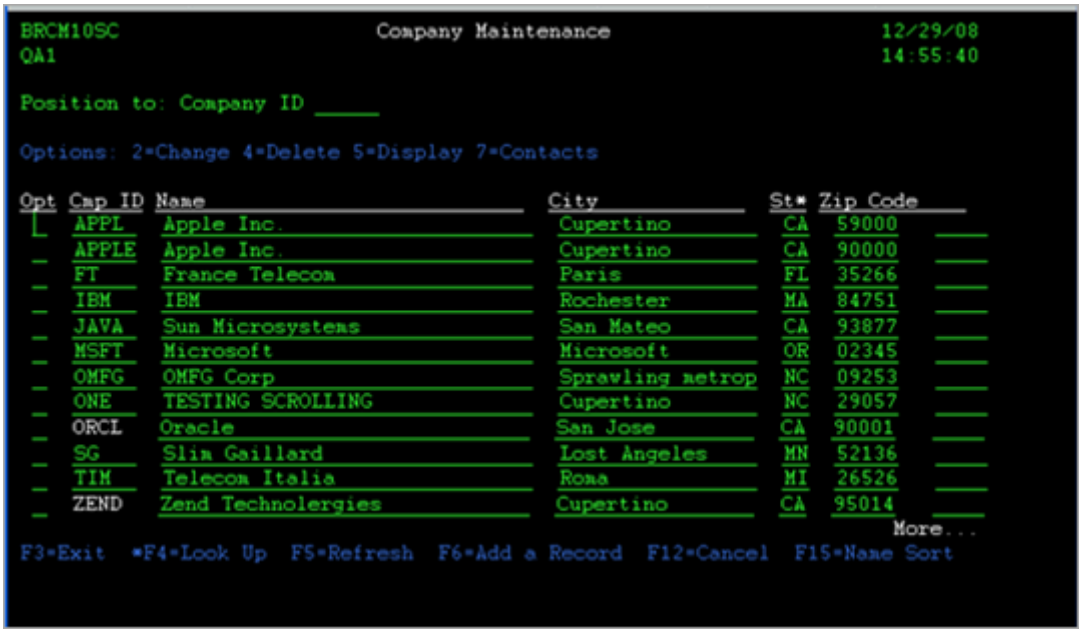
Forcément, il y a des exceptions. Pour la saisie principalement.

Dans certains cas, en effet, il se peut qu'une grille soit un élément d'interface tout à fait adapté à la circonstance. Notamment pour certaines saisies rapides de données. Mais dans ce cas la grille est vide : c'est l'utilisateur qui la remplit et jamais, lui, n'ira la remplir de milliers d'enregistrements en quelque secondes. Ca, seuls les informaticiens savent le faire avec les grilles fourre-tout ou sapin de Noël.

Pour résumer

Les grilles, de quelque type soient-elles sont laides, rectangulaires comme les fenêtres de Windows 1.0, et font régresser l'expérience utilisateur au temps des consoles IBM 5250 connectées aux mainframes, ou aux AS/400 (une bonne trentaine d'années donc, au

minimum). Ci-dessous un bon exemple (issu d'un émulateur moderne avec des couleurs pour faire plus jeune). On ne rit pas, c'est le genre d'affichage qu'on trouve aujourd'hui encore très fréquemment dans les banques ou même chez un assureur (j'ai vu les mêmes il n'y a pas longtemps à la MACIF…).



Elles sous-entendent le plus généralement une consommation excessive des ressources réseau et de la mémoire des machines clientes.

Elles sont lentes, et plus elles veulent en faire, plus elles sont lentes.

Le pire c'est qu'elles sont aussi lentes à charger en tant que composant (regardez les principales grilles sapin de Noël vendues pour XAML et essayez de les utilisez avec une connexion web française dans la norme, soit 512K à 1Mb, et vous me direz combien de bières vous avez eu le temps boire avant que l'application n'arrive à l'écran attendu…).

> *Il n'y a finalement que deux utilisations des grilles qui rendent la présence de ce composant acceptable : soit la* ***saisie rapide de données de même nature et comprenant peu de champs****, soit la* ***présentation de résultats de recherche filtrés*** *ne dépassant pas 30 à 50 lignes grand maximum. Et encore la « grille » doit se faire discrète visuellement.*

De fait, en dehors de la saisie qui impose un composant particulier, la grille peut être remplacée par une Listbox templatée dans 99% des cas.

Conclusion

N'utilisez pas de grilles ! Sauf dans le cas où cela s'avère un choix judicieux, ce qui est fort rare. Ne succombez pas aux sirènes des grilles sapin de Noël. Quitte à utiliser une grille, faites dans la simplicité d'utilisation et la sobriété du look et des fonctionnalités. Vouloir intégrer Excel dans un composant et le connecter à des millions de données stockées sur un réseau et croire qu'on a "programmé un logiciel" est stupide, en plus d'être un mensonge.

> *Un logiciel bien pensé est bien designé, et les éléments d'interface pour l'utiliser apparaissent aisément (et rapidement). (Emprunté à Nicolas Boileau)*

La sobriété est mère de toutes les vertus, autant que les grilles sont mères de tous les vices…

Pensez-y la prochaine fois que vous serez tenté d'une coller une petite par-ci ou par-là !

A bas les Datagrid !

Un an environ après mon article « A bas les Grilles ! » (Ci-dessus) Eric Ambrosi, le seul en France avec votre serviteur à avoir été MVP Silverlight, a commis en début du mois d'aout 2012 un petit papier intitulé "Supprimons les DataGrid de 90% de nos écrans ça fera du bien... aux utilisateurs". Sachant que je défends depuis 15 ans au moins cette position (mode née avec des logiciels comme Borland Paradox ou MS Access) car les grilles sont toujours ou presque l'expression d'une méconnaissance des besoins de l'utilisateur (sauf rares exceptions), je ne peux que vous inciter à lire le billet de Eric qui justifie en designer cette position radicale mais ô combien salutaire. Ce n'est pas long, et c'est une lecture saine et édifiante !

Les bases du design Modern UI (ex Metro)

Ne m'appelez plus jamais Metro... Donc pour l'instant il semblerait qu'il faille dire tout simplement "Windows 8 App", même pas "Style". En réalité le terme est désormais Modern UI, voir Windows Store Apps Style. Mais laissons la guerre des noms d'un côté et regardons plutôt ce qui n'a pas changé : l'ergonomie Metro et ses principes de base.

Une histoire assez longue déjà

En réalité "Metro Style" ou "Modern UI" est un Design assez ancien qui remonte à Zune (jamais vendu en France et arrêté de production avant d'avoir franchi l'Atlantique, mais dont le logiciel de synchro est resté pour les Windows Phones) et qui a été "popularisé" (c'est un bien grand mot !) par Windows Phones 7.

C'est ensuite, lors de la préparation de Windows 8 que ce langage de Design a été formalisé et baptisé "Metro Style" avec livret explicatif à l'appui. Toute une communication déclinée sur le côté "urbain", "clean", signalétique de métro. Mais sans se méfier que le nom était déposé par ailleurs… Amateurisme surprenant.

Les démêlés jamais réglés avec Metro AG, un gros distributeur allemand, auront eu raison de cette machine à gagner communicationnelle qui perd ce qu'il y a de pire pour un produit ou une idée, son identité.

Windows 8 app, puis « Modern UI », c'est une appellation un peu longue et peu vendeuse. D'autant qu'on peut faire des applications Windows 8 en WPF 4.5 sous .NET pour bureau classique faisant par exemple des appels aux API spécifiques Windows 8 et que donc de telles applications sont bien des "Windows 8 Apps" puisqu'elles seraient conçues

uniquement pour Windows 8. Mais elles ne fonctionneraient pas pour autant sous “Metro”...La confusion est dans l’air pour Microsoft, et c’est peu dire !

Toutefois ces aléas patronymiques ne changent rien à ce qu’est réellement Metro Style.

Un but, une UX nouvelle et agréable

Le but ultime de Metro Style (je l’appellerai encore comme ça un petit moment pour éviter qu’on le confonde avec les fraises de bois) est de fournir à l’utilisateur une “expérience utilisateur” (UX) nouvelle, agréable, réactive et intuitive.

Pour la nouveauté, c’est gagné. On ne peut reprocher à Microsoft d’avoir copier sur ses concurrents, il y a bien innovation.

Pour la réactivité, force est de constater que les équipes Windows ont fait un travail de fond remarquable et que l’OS répond vite et bien. On l’avait déjà vu dans Windows Phone où la réactivité était plutôt bonne malgré des machines techniquement moins gonflées que des équivalents Android ou Apple.

Agréable... On entre dans le subjectif. Et dans la zone de discorde. Pour une mamie qui fait du mail à ses petits enfants ou un gamin qui passe ses journées sur Youtube et FaceBook sur une tablette, certainement, c’est très agréable. Pour le professionnel qui doit switcher entre le menu Metro Style et le Bureau classique, et qui devrait rester le bras tendu toute la journée pour faire du tactile sur une machine Desktop, le côté agréable m’échappe totalement, ainsi qu’à tous ceux qui ont testé la bête dans ces conditions... Mais nous ne sommes visiblement pas la cible du produit. Et c’est à nous de rendre tout ça agréable finalement...

Intuitive... Là aussi ça coince un peu. Metro Style réclame une formation de l’utilisateur lorsque ce dernier est déjà rodé à la métaphore du Bureau... C’est une des raisons des réticences des entreprises dans un récent sondage d’ailleurs. Je comprends parfaitement les craintes des DSI face aux utilisateurs dits “de base” (secrétaires, techniciens, infirmières, etc), tous ces gens dont l’informatique est très loin de leur enseignement et qui auront du mal à comprendre qu’il faut mettre sous pouce ici et pas là, qu’il faut rester appuyer sur ce bouton mais pas trop longtemps... Toutefois le succès des Smartphones qui utilisent déjà le tactile à tout va doit tempérer cette appréhension. L’utilisateur de base n’est pas si idiot que ça, surtout si les applications sont bien designées !

Deux points assurés sur quatre. 50% de réussite certaine, 50% à voir, c’est bien ça qui d’ailleurs rend la période si électrique : si MS avait atteint les 80% au moins on aurait l’assurance du succès de Windows 8. Avec un fifty/fifty, c’est le jeté de pièce en l’air. Ca rend tout le monde nerveux dans le monde de l’édition, ça angoisse les développeurs qui se demandent s’il faut rester sur les acquis .NET ou bien se tourner vers Apple ou bien chercher une bonne poutre pour se pendre avec le câble de leur box internet...

Il n’en reste pas moins vrai que tout ce qui est nouveau chamboule les habitudes et que nous souhaitons tous que Windows 8 soit un succès immense mettant fin à cette période de doute et d’hésitation qui n’a que trop durée tant elle est nuisible au business. [Un an après

nous le savons, c'est un échec. Mais Windows 8.1, en tout point identique va être lancé, répéter la même chose serait donc la clé du succès ? Rendez-vous dans un an !]

Certains malins, qui semblent avoir fait l'impasse sur les cours de logique durant leurs études d'informatique voudraient prouver que Metro Style c'est forcément bien et révolutionnaire puisque ça bouscule les habitudes. Mais tout ce qui bouscule les habitudes n'est pas forcément un progrès. Tout ce qui choque n'est pas forcément moderne. Tout ce qui est hérétique n'est pas forcément juste ! N'est pas Giordano Bruno ou Galilée qui veut ! Si proposition logique originale est vraie sa négation ne l'est pas automatiquement !

Bref, il faudra voir, car c'est justement l'utilisateur de base qui va trancher, et il n'y a plus longtemps à attendre. Mais une chose est sure :

Metro Style sera un succès si les applications suivent. Et cela implique un design très étudié. [Cela a manqué à Windows 8 et posera le même problème à 8.1]

D'autant que cette nouvelle interface a de nombreux aspects positifs et novateurs qu'on ne peut lui retirer et qu'elle se repose sur un OS dont on a aujourd'hui l'assurance de la qualité (en tout cas en termes de réactivité, consommation électrique, consommation mémoire, etc).

Moi j'aime bien Metro, j'ai juste peur qu'on se lasse vite des tuiles, trop radicalement simplistes, déjà "trop vues" avant même d'être sur le marché (enfin si, WP7 existe depuis longtemps mais reste toujours très confidentiel, ne parlons pas de 7.5 ou 7.8 ni même de Windows Phone 8 qui semble juste profiter de la disparition de BlackBerry sans inquiéter Android ni Apple). Mais c'est un doute personnel qui engage mes gouts et je sais que sur ce terrain je ne représente pas la masse du marché visé donc que mes sentiments ne peuvent ici être traduits en prédictions, sauf à passer pour un guru à la manque si jamais Metro style fait un tabac auprès du public ! [même si l'échec de Windows Phone 7 et 8 et de Windows 8 donne un certain relief à mes prévisions avec le recul].

En tout cas, "Modern UI", Metro donc, change la donne et force le marché et ses acteurs à se poser des questions, à remettre en doute ses certitudes, et c'est positif.

Reste à voir quels sont les fondamentaux de ce nouveau "Style".

Concevoir dans l'esprit Modern UI

La conception d'applications Windows 8 Modern UI est très différente de celle des applications de bureau traditionnelles. Ceux qui ne sont jamais passés par la case WPF ou Silverlight vont avoir un drôle de saut à faire !

Les lecteurs de Dot.Blog qui suivent mes articles notamment sur le Design et ceux qui m'ont fait l'honneur de venir écouter ma conférence sur le Design aux TechDays à Paris sont peut-être un peu mieux préparés que les autres. C'est un juste retour sur investissement pour eux, supporter mes billets et mon humour va enfin payer 

En effet, concevoir des écrans qui sont assez petits (Smartphones ou tablettes) et qui se manipulent avec les doigts n'a que peu de chose en commun avec la conception d'une

application Windows Forms ou ASP.NET pour un écran de bureau et une souris ! Même le clavier n'est pas forcément présent ou devient alors virtuel, occupant une partie du champ de vision.

Ces contraintes très fortes obligent à penser la mise en page et les commandes du logiciel de façon très spécifique.

Des modèles pour comprendre

Vous trouverez ici des ressources intéressantes pour vous lancer sous Modern UI.

Vous y trouverez des assets pour concevoir simplement des interfaces qui correspondent tout de suite aux contraintes à la fois de look & feel et de matériel (pas de clavier, pas de souris, etc...). Beaucoup de choses utiles sont à télécharger :

Une variété de styles (clairs ou foncés) des tous les contrôles

Des tuiles en ratio 1x1 ou 1x2, les principaux contrats de l'interface comme les charmes, la recherche, la configuration, les barres de menu, etc...

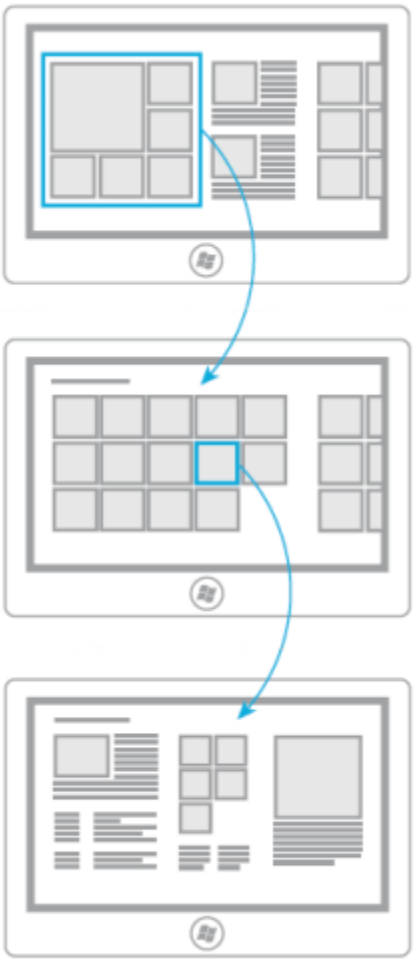
Des mises en page de base conçues pour le plein écran, le portait ou le mode paysage, les vues dockées

Des templates de projets en thème foncé ou clair.

C'est un bon point de départ pour qui veut se faire une idée de ce fameux look qui n'a plus de nom mais un caractère unique...

Les design patterns de l'UX Metro

Parlons maintenant des patrons de conception de l'UX. Ils englobent les principes les plus importants pour la conception d'applications Windows 8 Modern UI et sont des guides précieux pour la conception finale. Ces patrons vous aideront à trouver les réponses aux questions liées à l'organisation du contenu des pages, au placement des boutons et des commandes ainsi qu'à la bonne intégration de la gestuelle et ses interactions avec l'utilisateur.



La navigation

Les patterns de navigation guide le développeur pour l'aider à organiser le contenu de ses pages, sous-pages, sections et catégories de telle façon à ce que l'utilisateur se sente à l'aise et puisse manipuler le logiciel de façon intuitive, notamment lorsqu'il s'agit de changer de page, de naviguer dans toute l'application.

Ces patterns se focalisent sur deux modes principaux de navigation.

Vous trouverez ici des informations plus détaillées et d'autres assets à télécharger.

Le design en Hub

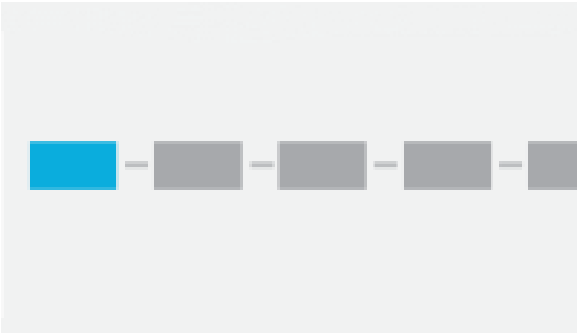
On l'appelle aussi pattern hiérarchique. C'est celui qui a été le plus utilisé dans la Consumers Preview par les applications de démonstration. Il est particulièrement bien adapté aux applications présentant beaucoup d'information dont la nature peut elle-même variée sensiblement.

Le contenu est organisé en “niveaux” et chaque niveau est responsable de la présentation d’un contexte spécifique. La page la plus haute de la hiérarchie, aussi appelée “Hub page” ou “page initiale” est l’emplacement où l’information est présentée en différentes sections et différents contextes. Généralement la Hub page propose beaucoup de catégories et de contextes mais peu de commandes et de fonctionnalités. Son but principal est de présenter de façon synthétique l’ensemble de l’information proposé par l’application.

Chaque section de la page Hub est associée à une page de Section qui est un peu plus spécifique dans sa présentation. Une page de Section contient généralement de nombreux items, chacun d’entre eux faisant l’objet d’un affichage détaillé dans une page dite de Détail. C’est le 3eme et dernier niveau de ce pattern de navigation.

Le design plat (flat design)

Ce pattern de navigation est principalement utilisé par des applications qui ont un but précis et facilement identifiable. Les interactions et le flux navigationnel sont clairs. C’est le cas des applications de création de documents par exemple qui possèdent un nombre restreint de pages qui doivent s’exécuter dans un ordre logique quasi immuable.



Les commandes

Les patterns de gestion des commandes que vous pouvez trouver ici décrivent les meilleures pratiques dans les applications Metro pour placer et utiliser des boutons de commande de toutes les façons possibles.

L'accent est mis sur ce que l'utilisateur devrait être capable de faire et comment cela devrait se produire. Lorsque cela est possible vous devez permettre de manipuler les éléments et d'exécuter les commandes directement à partir du canvas principal d’affichage et non par les charmes ou la barre de l'application. Les boutons de commande doivent être utilisés

systématiquement dans toutes les vues de l'application d'une même façon constante et intuitive pour que les utilisateurs puissent exécuter les commandes partout sans avoir à se poser de questions. Afin de limiter la complexité, le nombre de commandes doit rester assez faible. On doit toujours tenir compte de l'emplacement des boutons pour améliorer l'ergonomie et la rapidité d'action.

Il existe des guidelines pour les opérations impliquant les opérations tactiles, vous les trouverez ici.

La zone de confort des interactions

Si sur l'écran tactile d'un PC le confort reste moyen quelle que soit la position de l'utilisateur (à moins de mettre son moniteur 23" sur ses genoux...), il y a une façon de tenir une tablette tactile qui, *de facto*, rend certaines zones plus propices aux interactions et d'autres moins commodes d'accès.

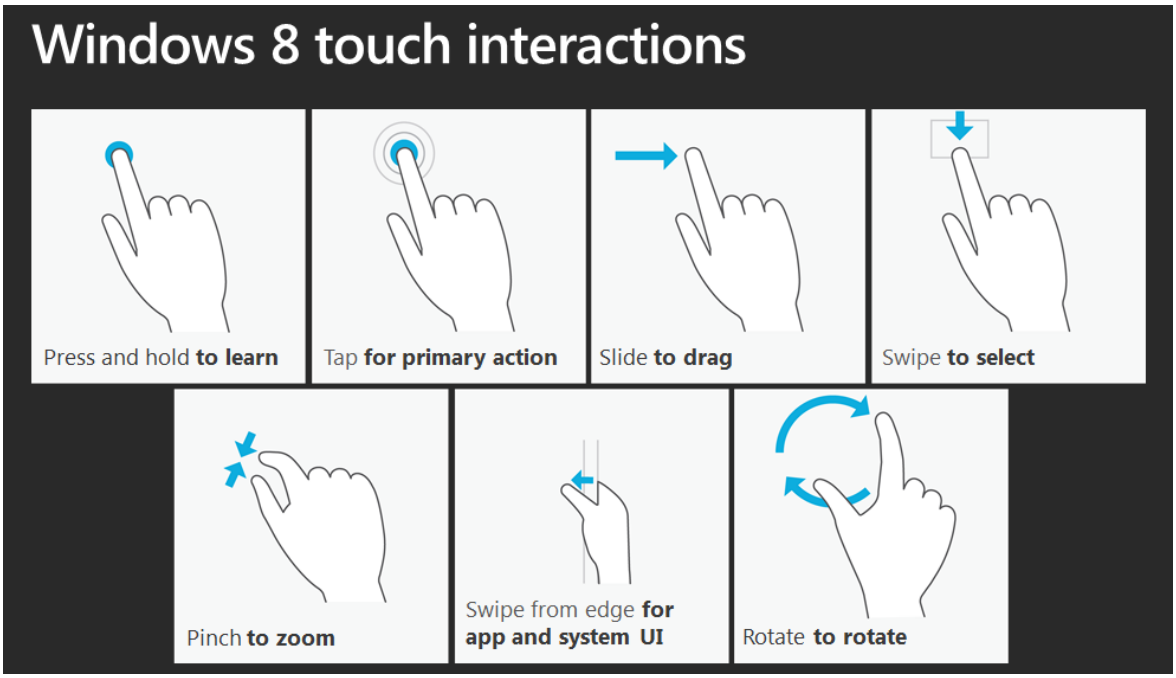
Ci-dessus on voit un schéma des trois zones de confort sur une tablette et un smartphone, de la zone la meilleure pour les interactions "best" à la zone la moins pratique "ok" en passant par la zone intermédiaire "better".

La zone de confort pour la lecture

Hélas cette zone de confort ne se confond pas réellement avec la précédente, ce qui implique une conception bien pensée pour satisfaire toutes ces contraintes ! On peut même dire qu'en fonction de la contrainte de confort pour les commandes la zone de confort de lecture est quasiment inversée :

La gestuelle standard

Windows 8 possède un nombre de commandes tactiles standard, des gestuelles qui doivent conserver le même sens tout le temps pour ne pas rendre fou l'utilisateur !



Les éléments de l'interface qui peuvent être déplacés doivent suivre la direction du doigt tant que l'affichage est touché, tout comme dans le monde réel – le stylo est dans votre main jusqu'à ce que vous le reposiez sur le bureau.

Toutes les interactions doivent être pistées par l'application, c'est-à-dire que l'utilisateur doit sentir ou voir une indication (rétroaction) de ses actions.

Il n'est pas acceptable d'appuyer sur un bouton avec un doigt et de ne voir aucun flash ou un autre type de rétroaction. L'utilisateur sera alors induit en erreur, pourra penser que l'action n'a pas été prise en compte ce qui l'incitera à refaire le geste (ce qui peut perturber l'application si elle ne gère pas convenablement les commandes en les désactivant tant qu'elles ne doivent plus être manipulées, etc).

Enfin, il toujours tenir compte de la taille des objets manipulables. Le taux d'erreur augmente quand la taille diminue. La taille minimale recommandée pour une cible de touch est de 7 x 7 mm (40 x 40 pixels) avec au moins 2 mm de padding. Ce genre de "détail" est essentiel et possède un impact gigantesque sur le design des écrans.

Les interactions tactiles sont importantes surtout pour les unités mobiles (smartphones et tablettes). Pensez bien au fait que ces interactions forment un véritable langage pour communiquer avec l'utilisateur, il faut que ce langage soit parlé de façon fluide et limpide...

Conclusion

Vous devez concevoir vos applications en vous mettant à la place de l'utilisateur, c'est un facteur essentiel de réussite et même de viabilité pour vos logiciels.

Pour cela les simulateurs de développement ne suffisent pas. Il faut des machines réelles sous la main pour tester et surtout “faire tester” par vos amis, votre famille, surtout pas vos collègues !

Il faut aussi de vrais designers. On n’engage pas un infographiste pour lui faire concevoir les écrans d’une application, ce n’est pas possible. Il faut des spécialistes qui connaissent toutes les contraintes techniques, que cela soit sous Silverlight, sous WPF et encore plus sous WinRT.

Il y a quelques informaticiens très doués pour le dessin, ils sont très rares, comme les infographistes étant capables de prendre en compte les contraintes techniques de plateformes techniques complexes comme .NET ou WinRT.

Ce qui est rare est cher. Mais surtout c’est rare...

Au sein d’E-naxos je cherche toujours à proposer les meilleurs services au prix le plus juste. Ce qui est rare est cher, mais ne doit pas être hors de prix non plus... Pour régler ces problèmes de design j’utilise mon savoir-faire de compositeur (la musique et le dessin ont beaucoup en commun quand on les pratiques en les conceptualisant sans cesse) mais pour la qualité du tracé, l’idée graphique que je n’aurai pas, je m’appuie sur ma collaboration avec un infographiste. Plutôt que de chercher la perle rare sachant tout faire, je préfère ainsi proposer un tandem qui regroupe les deux compétences, cela coute cher, forcément, mais c’est accessible. Et mes clients ont l’assurance que graphiquement et techniquement le design de leurs applications sera pensé dans les règles. Car un beau dessin d’interface qui ne prend pas en compte les contraintes techniques coute très cher à (mal) réutiliser, de même qu’un design approximatif reste mauvais même si techniquement il respecte toutes les contraintes...

Dans un monde où l'utilisateur est au pouvoir et définit les normes, la conception et à la convivialité de votre application est l'emballage. Et un mauvais packaging dans un tel monde ne doit même pas être considéré comme une option.

Les utilisateurs zappent très vite sur un market place. Le look & feel de vos application est l’élément essentiel si vous voulez avoir ne serait-ce qu’une petite chance que vos applications soient téléchargées et testées... Et lorsque l’utilisateur vous aura fait cet honneur, vous n’aurez que quelques secondes, quelques minutes au plus pour le séduire et ne pas le décevoir. Personnellement quand je cherche une application faisant quelque chose de précis sur un market place il m’arrive de télécharger 5 ou 10 applications l’une derrière l’autre. Je ne leur laisse qu’une poignée de secondes pour me donner envie d’aller plus loin, et j’agis ici comme la grande majorité des consommateurs. Pensez-y !

Guide de l’UX pour les applications Windows 8

Microsoft a publié le “Windows 8 UX Guidelines for Windows Store apps”. A lire pour créer des applications WinRT respectant les “normes” Windows 8 et pouvoir passer les tests du Windows Store sans trop de mauvaises surprises.

UX : Le Syndrome de la porte de Parking appliqué à Surface et au Windows Store

Vous le savez, Dot.Blog ne véhicule pas que des billets purement techniques, j'aime de temps en temps vous livrer le fruit de mes réflexions. Aujourd'hui je vais vous parler du Syndrome de la Porte de Parking appliqué au développement de logiciels pour Windows Store…

Qu'est-ce que le "Syndrome de la porte de parking" ?

Qu'est-ce que c'est cette histoire de porte de parking ? C'est fort simple et je vais par le menu vous la conter.

Une histoire banale

Pendant de nombreuses années j'ai habité en immeuble dans différents quartiers de Paris, mais en prenant soin à chaque fois de disposer d'un parking pour garer ma voiture. Tous ces parkings même au plus loin que je puisse me souvenir, tous étaient dotés d'une porte électrique.

De la plus ancienne et simple (une vraie clé à tourner dans un boitier, comme une porte de maison), à la plus sophistiquée (détection de badge sans contact, système de contrôle infrarouge de présence, etc), toutes ces portes de parking ne demandaient pas de sortir de la voiture pour les ouvrir et les refermer.

Ce fut une telle habitude que jamais je n'avais fait attention à ce "détail".

Puis, dans les années 2000 je me suis installé à Vincennes, du côté jouxtant l'avenue de Paris à St Mandé. Un endroit agréable, près de tout et déjà hors de la foule parisienne. L'appartement que j'avais choisi répondait à mes différents critères et possédait au rez de chaussée un parking privatif. Certes ce n'était qu'une cour intérieure sans toit, mais ce n'était pas pire que de laisser la voiture dans la rue après tout, avec l'avantage justement de pouvoir se garer de façon sûre à toute heure du jour et de la nuit, ce qui, en région parisienne est un luxe qui n'a pas de prix (enfin, si, ça en a un, assez élevé en général…).

Un détail qui n'en n'est pas un

Le petit “détail” auquel je n’avais pas porté attention concernait la fameuse porte de ce parking. N’ayant jamais été confronté à la situation je n’avais pas “tilté” que cette porte n’était pas automatique, de prime abord cela ne me gênait pas.

Erreur.

Les premiers temps furent ceux de la joie, de la nouveauté, de la découverte à pied de mon nouveau quartier, puis rapidement vinrent les contraintes du quotidien et leur gestion très terre à terre. Aller chez un client, en revenir, aller chercher les enfants, faire des courses dans les supermarchés de la périphérie… Revenir avec 2 pack d’eau plus toutes les courses ne se fait pas en bus ni à vélo… Il fallait donc prendre la voiture.

Et là j’ai commencé à comprendre la nature de ce que j’allais appeler plus tard le “**syndrome de la porte de parking**”.

Le Syndrome de la porte de parking

Avec le temps, de plus en plus souvent et sans trop m’en rendre compte au départ, je me suis mis à prendre le vélo pour faire des courses proches, quitte à “galérer” avec des sacs en plastiques accrochés au guidon… Pour les déplacements plus lointains ou dans un Paris déjà devenu impraticable à cette époque, j’utilisais de plus en plus ma moto. Quitte à l’utiliser à des moments où je ne l’aurais jamais prise : pluie légère, puis plus battante réclamant de s’harnacher avec la combinaison étanche, petits froids inconfortables jusqu’aux froids polaires impliquant un équipement digne de “Premier de Cordée” …

Pourquoi petit à petit me suis-je ainsi soumis à des désagréments de plus en plus grands ?

A cause de cette fichue porte de parking.

Le simple fait qu’elle ne soit pas automatique m’obligeait à une séquence, une “expérience utilisateur” qui, sans en avoir l’air, devenait intolérable :

- Monter dans la voiture, la démarrer
- Avancer jusqu’à la porte
- Descendre de la voiture (fermer la portière s’il pleut)
- Aller ouvrir la grille (trouver la clé, ouvrir, enclencher les grilles dans leur système de blocage)
- Revenir vers la voiture, ouvrir la portière, s’installer à nouveau, fermer la portière
- Franchir le seuil du parking
- Arrêter la voiture
- En descendre, refermer les portières, couper le moteur, retirer la clé, verrouiller la voiture car cette fois-ci elle était côté rue (un pc portable ça se vole aussi rapidement que ça, et même la voiture toute entière)
- Se diriger vers cette satanée porte
- Débloquer les grilles, les fermer

- Refermer à clé (zut, je l'ai laissée sur le siège passager… revenir plusieurs lignes plus haut !)
- Revenir à la voiture, la rouvrir
- Enfin, s'installer, mettre la ceinture

Partir, enfin !

… Penser qu'au retour le même cinéma sera à recommencer…

Et encore je vous fais la séquence "courte". Si j'étais chargé au départ ou à l'arrivée (retour de courses par exemple), et s'il pleuvait (à Paris c'est fréquent) tout cela devenait encore plus pénible.

Le détail devient frontière

Et c'est là que naît le Syndrome de la Porte de Parking. Ce simple détail, ce simple petit automatisme d'ouverture et de fermeture de la porte qui manquait, quelque chose de dérisoire en soi, allait prendre au fil du temps une telle importance en impliquant une "expérience utilisateur" tellement mal vécue, que pour diminuer les frustrations qui naissaient de cette situation j'étais prêt à m'infliger d'autres frustrations comme utiliser le vélo ou la moto même sous la pluie ou dans le froid…

Pourquoi le vélo ou la moto ? La porte de parking devait être franchie aussi pourtant ! ?

Simplement parce que vous pouvez, avec quelques contorsions vous rendre directement devant la porte et l'ouvrir et la fermer tout en restant en selle (plus délicat avec la moto qui est plus lourde et plus grosse, mais faisable) !

De simple détail, l'absence d'un automatisme à la porte de parking était devenue **une frontière infranchissable** de nature à vous **décourager de faire certaines choses**, **vous obligeant à couper votre vie en deux pays séparés par cette barrière** : **tout ce qui mérite vraiment de passer la porte** de parking avec la voiture, **et le reste**.

Quel rapport avec Surface, le Windows Store ou l'UX ?

Cette petite histoire peut de cent façons être rattachée à l'UX (que j'ai mentionné volontairement quelques fois) et donc au marché de Surface et plus généralement à celui du Windows Store (Smartphones, tablettes et PC).

Comment tirer parti de cette leçon, quelle est la morale de cette histoire ?

Rappelez-vous : je franchissais plus aisément la porte du parking en vélo ou en moto, quitte à subir d'autres désagrément plutôt que d'avoir à ouvrir la grille en grand pour faire passer la voiture (et jouer toute la séquence infernale décrite plus haut).

Vous ne voyez toujours pas le rapprochement ?

Mais si voyons…

Votre Smartphone est toujours allumé, toujours sur vous ou à côté de vous. C'est le vélo ou la moto. Certes il a un clavier virtuel ridicule, certes son écran est tout petit pour faire la moindre chose sérieuse (même sur un Galaxy SIII ou Note 2 je peux vous l'assurer), mais malgré le vent, le froid et la pluie, vous préférerez toujours utiliser une application Smartphone qu'une application tablette ou pire PC si les contraintes sont psychologiquement vécues comme moindre que le boot de la tablette ou du PC.

En d'autres termes, cela signifie que les applications pour tablettes ne sont pas à copier dans la masse des applications pour Smartphone…

Les applications pour tablettes doivent être capable de passer le test du Syndrome de la Porte de Parking, c'est à dire qu'elles doivent apporter un service tel que les avantages sont supérieurs aux désagréments qu'entraîne le boot d'un PC ou d'une tablette…

J'entends bien quelques voix qui parleront de la mise en veille (autant sur tablette que sur PC). C'est vrai, ça existe. Mais ce n'est pas sans inconvénient non plus (ne serait-ce que l'autonomie qui diminue). Et puis cela est plus ou moins pratique à mettre en œuvre dans la pratique, selon le matériel. Mais cet avantage sur une Surface et son clavier rabattable peut créer une situation intermédiaire, c'est vrai, d'autant que Windows 8 est conçu pour se réveiller très vite.

Mais au final la porte de parking n'est jamais bien loin…

Et puis ce n'est pas tout.

Un problème de taille

Porte de parking ou pas, les tablettes (et là je ne parle plus des PC tellement cela est évident) pour justifier leur existence doivent avoir un écran assez grand pour permettre de faire ce qu'un Smartphone ne peut pas faire avec un confort visuel suffisant (comme taper un rapport sous Word par exemple).

Une tablette qui devient assez petite pour tenir dans une poche n'est plus qu'un Smartphone qui ne peut pas téléphoner, aucun intérêt. Autant qu'un Smartphone qui grossirait pour avoir écran 23 pouces ne trouverait guère d'utilisateurs..

Mais dans le même temps les tablettes doivent restée plus petite que les portables classiques… Sinon ce sont des portables !

Du coup une tablette ne se range pas dans une poche, elle ne peut prétendre être un compagnon de chaque instant comme un Smartphone. Ce qui au final renforce le Syndrome de la Porte de Parking en augmentant **la nécessité d'applications vraiment utiles et attractives** pour justifier d'aller chercher la tablette et la sortir de veille (alors qu'on a son Smartphone allumé dans sa poche ou sur son bureau)…

C'est ce que j'appellerai le Syndrome de la Poche de Veste.

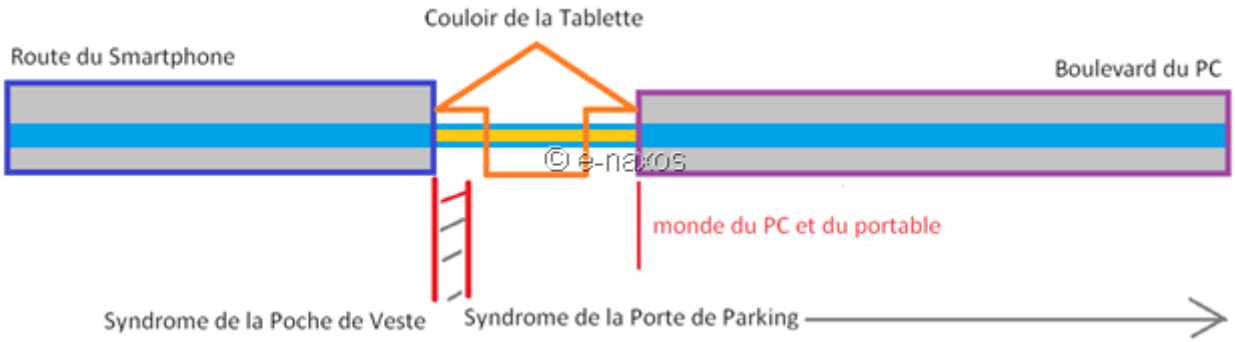
Une Surface réduite à un couloir…

Le jeu de mot était facile. Mais c'est pourtant une réalité bien plus qu'une tournure de phrase !

En réalité le marché de la tablette se trouve coincé entre ces deux syndromes : celui de la porte de parking et celui de la poche de veste. Les tablettes ne peuvent se réduire à la taille d'un Smartphone ce qui serait un moyen d'échapper à ce couloir, et de par leur réactivité à l'allumage moindre qu'un Smartphone toujours en fonctionnement les applications qu'elles proposent doivent franchir le seuil du syndrome de la porte de parking.

Cela laisse beaucoup de place aux tablettes sur un marché vierge mais pourrait vite devenir un couloir trop étroit pour en faire un marché de masse passé l'engouement des geeks de la première heure…

Regardons cette vue d'ensemble :



Les deux "syndromes" que j'évoque dans ce billet forment deux barrières très proches sur la longue voie des logiciels et de leurs supports matériels. La tablette peut être très réactive à l'allumage elle se place malgré tout à droite du syndrome de la porte de parking, ce n'est pas aussi rapide et pratique à utiliser qu'un Smartphone déjà allumé qu'on a sur soi.

Une toute petite portion (hachurée) se trouve entre les deux syndromes. Il s'agirait de tablettes pouvant être aussi réactives qu'un Smartphone, aussi disponibles et rapide à utiliser, mais dont la taille resterait celle d'une tablette. Je ne suis pas certain que même Surface puisse se faufiler dans ce "conduit" (ce n'est même plus un couloir !) malgré sa rapidité de boot.

D'un côté le "Boulevard du PC". C'est une évidence, il y a des centaines de millions de PC dans le monde, ils sont équipés de grands écrans confortables, de grands claviers permettant une frappe longue, d'une puissance raisonnable et sont installés un peu partout. Ils sont certes placés du "mauvais côté" de la frontière du syndrome de la porte de parking, mais les logiciels proposés sur PC sont en majorité des outils qui justifient d'avoir à ouvrir cette fichue porte… Word et Excel en sont d'excellents exemples. Leur puissance, la nécessité d'avoir un affichage assez grand et le service rendu d'une haute valeur ajoutée justifient largement de franchir la porte du parking… J'évoquerai aussi PhotoShop, Illustrator dont le nombre de palettes réclament plusieurs écrans pour avoir assez de place pour le dessin qu'on créée. Que dire de Cubase ou Live en musique et de milles autres types de logiciels qui tous justifient de franchir la porte du parking car le service rendu le justifie.

De l'autre côté se trouve la "Route du Smartphone". Une route et non un boulevard car les restrictions sont malgré tout importantes : puissance, taille de l'écran, absence de clavier physique, nécessité d'une certaine autonomie, etc… Mais cette route est longue et assez large pour drainer des milliers d'applications qui ne passeraient pas la porte de parking ni même la limite de la poche de la veste. Prenez par exemple dans les Top 100 des différents market places : gérer sa pilule, c'est génial, ça colle parfaitement aux contraintes du Smartphone. Jamais une femme n'ira allumer un PC ou une tablette pour gérer ça, en revanche son Smartphone c'est comme sa trousse de maquillage qui est dans son sac à main : toujours là, toujours prête. Pensez aux applications de localisation (où ai-je garé ma voiture ? repérer les coins où j'ai trouvé des champignons, mesurer une hauteur ou une distance, …) et aux petits jeux qui distraient dans le métro, le bus, dans la salle d'attente. Pensez aux applications de type mp3 player, et même aux fonctions d'appareil photo et caméra vidéo. Tout cela colle parfaitement avec le monde des Smartphones car pour leur grande majorité toutes ces applications se trouvent du bon côté des deux syndromes, donc **à leur place sur la bonne machine**.

Au milieu… non il ne coule pas une rivière. C'est un couloir, étroit. Borné par les deux syndromes d'un côté et par le monde du PC de l'autre, frontières derrière lesquelles s'arcboutent avec force les PC et les Smartphones qui campent sur leurs positions en essayant de ne rien lâcher !

La cible spécifique des tablettes

Ce couloir n'est pas extensible. Il limite de fait et pour toujours la place réservée aux tablettes. Cela implique que les applications qui seront conçues pour tablettes et qui en feront le succès seront toutes **des applications qui respectent le couloir de la tablette**.

Nous sommes sur un marché naissant et je ne vais pas vous livrer comme ça les exemples que je pourrais éventuellement développer demain… Mais regardez bien tous les logiciels qui sont aujourd'hui présents sur le marché des unités mobiles et demandez-vous lesquelles passent la porte du parking…

Pourquoi Surface pourrait bouger les lignes

Quand je dis que ce couloir à tablettes est là pour toujours c'est certainement vrai. Mais avec de bons coups de butoir il est peut-être possible de l'élargir un peu.

C'est à mon sens ce que fait Surface tout en respectant les limites du syndrome de la porte de parking…

D'abord surface c'est son clavier magnétique. Une astuce unique qui permet de sortir une tablette vendue comme telle alors qu'en réalité cela devient un hybride… donc un pc portable. Du coup, la … Surface du couloir s'étend en grignotant un peu sur le boulevard du PC. C'est ce que mon petit dessin (plus haut) montre.

Mais en même temps, Microsoft dans sa grande sagesse, et même sur les modèles RT, intègre de base Office. C'est pile le premier exemple de logiciels que je citais plus haut qui justifient d'avoir à ouvrir la porte du parking…

Grâce à cette approche **Surface se démarque de toutes les autres tablettes** en jouant subtilement sans grand changement apparent avec les limites du syndrome de la porte de parking !

Comment en tirer parti ?

Maintenant que je vous ai expliqué la nature du couloir des tablettes il ne vous reste plus qu'à trouver des idées de logiciels qui respectent cet espace étroit et bien délimité mais assez grand pour s'y faire un nom et remplir son compte en banque !

Les applis de dessin avec stylet sont un parfait exemple d'adéquation avec les limites du couloir. Elles justifient d'ouvrir la porte du parking à la fois parce qu'elles utilisent un écran mieux adapté au dessin que celui d'un Smartphone (syndrome de la poche de veste) et qu'elles offrent un service qui justifient d'ouvrir la porte du parking.

Il y a quelques bonnes idées qui existent déjà mais je suis convaincu que malgré les centaines de milliers d'applis des market places Google ou Apple, il n'y a que peu de logiciels qui correspondent à la cible. D'ailleurs peu de développeurs / éditeurs gagnent réellement de l'argent au point d'en vivre et encore moins on fait fortune au point qu'on connaisse leurs noms. C'est une indication assez forte de la difficulté de respecter le couloir de la tablette tout autant que la route du Smartphone !

Conclusion

Sous Windows 8 et Windows Store tout reste à faire.

Ne perdez pas votre temps à publier des applis qui ne respectent pas le couloir de la tablette. De bonnes applications peuvent faire le succès d'une architecture, d'un OS, d'une machine, bien plus que dix milles productions bas de gamme ou mal adaptées à la cible.

La créativité, par définition, n'a pas de limites. Mais les tablettes en ont deux : par force elles ne peuvent franchir le syndrome de la poste de veste, et elles doivent offrir des services dont la valeur ajoutée leur permet de passer le Syndrome de la Porte de Parking.

N'oubliez ni l'un ni l'autre avant de vous lancer dans la conception et l'édition d'une application pour le Windows Store !

Quelques conseils de design (UserControl, Blend, Visual State manager, Xaml)

L'une des avancées les plus intéressantes introduite dans XAML (ajout datant de Silverlight 2) est très certainement le Visual State Manager. Gestionnaire des états visuels simplifiant la

conception visuelle des contrôles (UserControl). Bien utiliser le VSM, outre de rendre plus simple la représentation des états visuels d'un composant, apporte aussi une clarification essentielle à la gestion des transitions entre ces derniers.

Etats et transitions

Un contrôle peut être ou non visuel. Un `Timer` n'est pas visuel. Une `Combobox` l'est. Nous parlerons bien entendu ici uniquement des contrôles qui possèdent un visuel.

Un Etat visuel peut être compris comme une allégorie d'un ou plusieurs états logiques du contrôle. Les états logiques sont ceux définis dans le code fonctionnel, comme par exemple `IsEnabled` ou `IsChecked`. Il ne faut pas confondre *état* et *propriété*. Les états sont le plus souvent représentés par des propriétés (ils peuvent simplement être des champs internes ou le résultat d'un calcul) mais toute propriété ne représente pas un état (ou alors faut-il adopter une définition si large pour « état » qu'on ne peut plus rien en dire, la couleur `Foreground` ou le type de curseur sont des exemples d'états au sens global et objet mais pas dans le sens que j'entends ici).

je parle d'*allégorie* car comme je l'évoque dans le billet "Le défi des nouvelles interfaces Silverlight et WPF – La cassure conceptuelle", la représentation visuelle d'un contrôle (et de ses états) est l'aboutissement d'une démarche intellectuelle et conceptuelle qui a justement pour but de créer un univers dans lequel les acteurs (les contrôles) doivent chacun avoir leur place et leur comportement. Entre visuel et fonctionnel il n'y a pas de relation d'identité (au sens d'un schéma conceptuel de données - relation de type 1-1). Le visuel créé une identité pour l'acteur, ce qui est différent et ne s'entend pas dans le même sens.

Je disais aussi qu'un état visuel représente un ou plusieurs états logiques. Dans certains cas deux (ou plus) états logiques peuvent être "résumés" par un seul état visuel. Cette combinaison prend son sens ponctuellement, au cas par cas, et ne peut pas être généralisée en une règle absolue. Néanmoins, lorsqu'on créé le visuel d'un contrôle, il faut garder à l'esprit cette possibilité. L'œil peut discerner de nombreuses subtilités dans les formes et les couleurs alors qu'il lui faut plus de temps pour interpréter des séries de données chiffrées ou textuelles…

Les transitions ont aussi leur importance. On ne passe pas d'un état visuel à un autre de façon abrupte, sauf si cela est volontairement assumé. Les transitions fluidifient l'interface, créent une continuité visuelle. Si les transitions ne portent pas de sens en elles-mêmes, à la différence des états, elles jouent un rôle important dans l'expérience utilisateur (UX).

Trois phases

La conception des états visuels et des transitions peut se découper en trois phases différentes.

Phase statique

Cette phase de la conception consiste à créer des états dans le VSM et à définir l'aspect du contrôle dans chacun d'eux. On en profite pour identifier les **groupes d'états**.

Au sein de chaque groupe les états sont mutuellement exclusifs, les états de différents groupes étant indépendants et simultanés (une `Checkbox` peut à la fois avoir le focus et être cochée ou décochée par exemple. `IsChecked` appartient à une groupe d'états différent de celui définissant l'aspect visuel pour le focus).

Lors de la phase de conception statique on se préoccupe de l'aspect que prend le contrôle dans chacun des états. On vérifie aussi la compatibilité visuelle entre les états des différents groupes d'états. Si cette étape est qualifiée de statique c'est pour souligner qu'on ne se soucie pas encore des transitions (la dynamique visuelle), mieux vaut rester concentré sur les groupes d'états et les états eux-mêmes.

> *On veillera à ne pas manipuler une même propriété du contrôle dans plusieurs groupes d'états pour des raisons de cohérence. D'ailleurs si on commet cette imprudence Blend le signalera par un petit symbole de danger (triangle assorti d'un message de type tooltip). On peut assumer une telle situation si on en mesure toutes les conséquences, c'est pourquoi Blend signale le problème mais n'interdit pas la situation. Mais en règle générale cette alerte trahit une mauvaise conception !*

Etat "Base"

Vous l'avez peut-être remarqué, dès qu'un groupe d'états existe le VSM ajoute systématiquement à la liste un état spécial appelé **Base**. Cet état permet de visualiser le contrôle hors de tous les états visuels définis. Toute modification effectuée dans l'état Base se propage à tous les états définis et n'est enregistré dans aucune time-line.

Etat par défaut

Un contrôle bien conçu devrait posséder pour chaque groupe d'états un état par défaut. En effet l'état Base n'existe pas en tant qu'état visuel, il s'agit juste d'un mode d'édition spécial du VSM. Lorsqu'une instance du contrôle est créée il est forcément dans un état donné (par exemple `IsChecked=False` pour une case à cocher). Il se trouve même souvent dans plusieurs états précis qui seront représentés par plusieurs groupes d'états. De fait il ne faut pas se reposer sur l'état Base mais plutôt créer un état par défaut pour chaque groupe qui traduit l'aspect visuel du contrôle lorsqu'il vient d'être initialisé (ou "remis à zéro" si une telle fonction est disponible dans le contrôle). Dans les contrôles existants vous pouvez remarquer que de tels états par défaut existent. Par exemple dans le groupe `CommonStates` de la classe `Button` on trouve un état `Normal`, dans le groupe `CheckedStates` (d'une case à cocher) on trouve `UnChecked`, etc.

L'état par défaut de chaque groupe ne doit pas forcément porter un même nom (ce qui poserait un problème pour le VSM de toute façon, il n'autorise pas que deux états portent le même nom). Au contraire, cet état par défaut de chaque groupe doit, comme dans les exemples donnés ci-dessus, porter un nom ayant un sens en rapport avec le groupe.

Il est parfaitement valide de créer un état par défaut (ou non) qui ne fait que recopier la situation de l'état Base. Il suffit de créer l'état et de ne rien modifier… Cela permet souvent de clarifier les choses. Prenons l'exemple de la `Checkbox`, la croix est ajoutée dans l'état Base mais est cachée. En revanche cela ne lève pas l'obligation de créer un

état `Unchecked` dans le groupe `CheckedStates`. Cet état est juste créé mais non modifié puisqu'il reprend l'aspect de l'état Base (croix cachée).

Validation des états

Arrivé ici il faut tester tous les états. Cela peut se faire sous Blend mais il ne faut pas hésiter à créer une **fiche de test**, y placer le contrôle, et ajouter des boutons, sliders, et autres éléments d'interface pour tester au runtime le changement de chaque état, la cohérence entre les groupes, etc.

Dans certains cas très simples la conception des états visuels peut s'arrêter là. On peut vouloir des transitions franches et immédiates et il n'y a alors plus rien à ajouter.

Dans d'autres cas il s'avère essentiel d'aborder la seconde phase.

Phase des transitions

La version simple consiste à utiliser le réglage par défaut que le VSM propose pour chaque groupe d'états. Dans de nombreux cas cela peut être suffisant. Il suffit alors de définir un temps pour l'ensemble des transitions. Le VSM calculera à l'exécution les animations correspondantes.

La version plus évoluée consiste à utiliser les options du VSM pour chaque état afin de régler les transitions d'entrées et de sorties de celui-ci. Il peut en effet s'avérer nécessaire d'avoir des timings différents selon le sens de la transition et l'état précédent. Par exemple il est courant que les états visuels de type clic souris soient instantanés pour ne pas donner l'impression à l'utilisateur que le logiciel est "mou", lent à répondre, alors que le relâchement de la souris peut au contraire accepter un temps assez long, donnant une sensation de douceur, d'amorti "luxueux".

Encore une fois les choses peuvent s'arrêter là. Bien entendu après avoir testé toutes les transitions. N'oubliez pas que Blend propose une option permettant de visualiser en conception l'effet des transitions lorsque le VSM est actif. Cela fonctionne très bien, sauf pour les transitions utilisant des storyboards.

Phase des transitions dynamiques

Justement le troisième niveau de personnalisation consiste à créer des storyboards au lieu de se contenter des timings réglés dans le VSM. Dans ce cas on créé une animation complète pour la transition en utilisant les fonctions des storyboards (time line, boucle for-ever, autoreverse…).

Pouvant s'utiliser dans les storyboards ou sur les animations par défaut créées par le VSM, les fonctions de *ease in* et *ease out* permettent d'ajouter une touche "organique", plus naturelle, aux animations. Xaml propose de nombreux modes (comme le rebondissement par exemple) qui, bien utilisés, finissent le visuel et le rende plus "pro".

Relation avec le code

Il y a toujours eu deux aspects à la création d'un contrôle personnalisé, même sous Win32 avec les MFC ou sous Delphi avec la VCL. La création du visuel d'un côté et celle du code fonctionnel de l'autre. Tout a changé mais pas cette séparation qui s'est, au contraire, renforcée.

Le designer (l'infographiste) conçoit le visuel d'un contrôle, il prévoit comment le contrôle se comportera pour l'utilisateur, comment il s'animera, comment ses différents états seront visualisés.

Pour l'informaticien le point de vue est différent. Il ne doit pas se soucier de l'aspect visuel mais du comportement du contrôle, de sa logique sous-jacente.

Cela implique de prendre en compte tout ce qui peut modifier les états internes du contrôle, la cohérence de la machine logique lors de son fonctionnement. Que les changements d'état soient le fait de l'utilisateur, d'une animation créée par le designer ou de tout autre acteur, peu importe.

Il faut oublier le visuel qui d'ailleurs n'existe pas forcément (la séparation du travail design / codage est telle qu'on peut aujourd'hui commencer un projet par une réflexion purement conceptuelle et graphique avec un designer plutôt que d'écrire le cahier des charges avec un informaticien ou un électronicien quand il s'agit d'une device. C'est le mode de travail chez Apple par exemple).

Repérer les ilots d'états (qui deviendront ou non des groupes d'états dans la partie visuelle), le ou les graphes de changement d'état, vérifier quels sont les graphes d'états interconnectés (et qui forment un groupe ou des sous-groupes) de ceux qui sont totalement indépendants, tout cela est essentiel.

L'initialisation

On retrouve ici une préoccupation déjà évoquée quand nous parlions de la partie visuelle. Pour le designer il s'agissait de ne pas confondre l'état *Base* avec les états par défaut qu'il faut créer pour chaque groupe d'états.

Côté code il est indispensable que toute nouvelle instance d'un contrôle se "positionne" correctement. Normalement son initialisation comporte d'une façon ou d'une autre des valeurs par défaut. Sous C# il est possible d'atteindre cet objectif de multiples façons : soit par le biais de champs initialisés lors de leur déclaration, soit par le biais de valeurs par défaut mises en place dans les métadonnées des propriétés de dépendance, soit par code dans le constructeur de la classe, soit par code dans le gestionnaire de l'événement `Loaded` du contrôle (ou de l'une de ses sous-parties), soit encore par code XAML.

Le fait que ces différentes solutions puissent être utilisées simultanément dans un même contrôle n'aide pas forcément à rendre les choses claires… C#, comme tout langage, n'interdit pas le code spaghetti, hélas !

Mais ici en dehors de la pure stylistique, de l'académisme, voire même des bonnes pratiques, c'est aussi la stabilité visuelle qui risque d'être compromise si le contrôle ne s'initialise pas clairement dans une suite d'états bien déterminés (généralement les états par défaut).

Un contrôle a ainsi la charge lors de son initialisation de se positionner sur la case "départ" de son graphe d'états. Cela semble évident mais parfois les portes ouvertes sont celles qui méritent le plus d'être enfoncées. Tout le monde peut voir qu'une porte est fermée. Il faut être très perspicace pour s'apercevoir que l'absence d'un obstacle ne signifie pas qu'il n'y a rien à faire…

Car si rien n'est fait, le contrôle va être frappé d'une sorte de schizophrénie : il peut être doté d'un code qui s'initialise correctement et d'une interface qui en fait tout autant, hélas les deux personnalités existent simultanément et ne se connaissent pas forcément. Les cas de personnalités multiples sont passionnants en psychanalyse (je vous conseille d'ailleurs un excellent vieux livre écrit par une patiente atteinte de ce syndrome : Joan, autobiographie d'une personnalité multiple) et même s'il est plus facile de déboguer un programme qu'un cerveau humain, ce type de désordre du comportement ruine totalement les efforts pour créer une interface riche et cohérente… Alors autant y penser ! Cela signifie dans la pratique que vous devez **centraliser l'initialisation logique** de votre contrôle (l'ensemble de ses états) **et** l'initialisation visuelle. Cette dernière s'effectue en général en appelant `GotoState` avec le paramètre d'animation à `false` (il s'agit d'une bonne pratique. On évite d'animer les contrôles lorsqu'ils s'initialisent). Ainsi, code et visuel sont initialisés conjointement et sont en phase.

Encore une fois ne prenez pas *Base* pour un état. Notamment il n'existe aucune transition gérée par le VSM entre Base et les autres états. Si votre contrôle n'est pas volontairement initialisé dans ses états par défaut aucune animation ne sera jouée lors du premier changement car Base vers un état ne génère aucune transition.

Conclusion

La création d'un contrôle est une opération longue et minutieuse car elle doit être réfléchie. Et plutôt deux fois qu'une : à la fois sous l'angle visuel et sous l'angle du code. Ces deux aspects bien séparés aujourd'hui introduisent la nécessité de penser à leur indispensable synchronisation pour garantir la cohérence du contrôle.

Mais avec un peu d'habitude on s'aperçoit bien vite qu'il est mille fois plus facile de créer un nouveau contrôle visuel avec Blend qu'avec les technologies précédentes et même avec les plus modernes comme sous iOS ou Android qui sont des catastrophes préhistoriques comparés à XAML. Alors cela vaut bien un peu d'attention au départ !

N'oubliez pas que seul Blend permet sérieusement de travailler sous XAML. N'hésitez pas à apprendre à vous en servir…

Un peu d'UX pratique : Majuscules sans accents…

La plateforme .NET prend en charge l'Unicode et s'occupe parfaitement des problèmes de localisation pour les dates, les monnaies, ... Beaucoup pense qu'un tel système est suffisant pour assurer un bon fonctionnement de leurs logiciels sans se poser de question. Erreur. Et l'UX alors ?

Exemple

Prenons un exemple, cela sera plus simple.

Imaginons un logiciel qui permet de classer des documents en posant des mots-clés. Supposons un logiciel de classement de photos permettant de marquer ces dernières. L'utilisateur possède une liste de mots-clés dans laquelle figurent les mots suivants : "Français" et "Élégant" (on a le droit de classer ce qu'on veut après tout).

L'utilisateur a en effet pris le temps, lors de la création de la liste, de bien taper les mots clés en respectant les accentuées propres à notre langue. Il s'est un peu embêté d'ailleurs, car aucun clavier français à ma connaissance ne permet de taper les majuscules correspondantes aux voyelles accentuées ou au "ç" par exemple alors que, officiellement, ces lettres doivent aussi avoir leur majuscule... Mais notre utilisateur est consciencieux.

Maintenant, il importe de nouvelles photos et lorsqu'il clique sur chacune il dispose d'une boite de propriétés lui permettant de saisir les mots clés qui seront sauvegardés dans celles-ci.

Bien entendu, le logiciel étant bien fait, il peut aussi par drag'n drop déposer les photos sur un mot clé pour faire l'affectation (ou dans l'autre sens, déposer le mot clé sur une sélection de photos). Dans un tel cas le logiciel recopie servilement le mot clé original dans chaque photo, il n'y a donc aucun souci.

Mais voilà que notre utilisateur, rompu à l'usage du clavier, et devant effectuer de nombreuses mises à jour, préfère taper directement les mots clés dans la fenêtre des propriétés plutôt que d'utiliser les manipulations de la souris...

Il sélectionne une photo et saisit "élégant". Pas de problème.

Il en prend une autre et tape "elegant". Aie... le logiciel créé un nouveau mot clé et l'affecte car il ne voit pas que "élégant" et "elegant" sont identiques. J'y reviendrais.

Sa petite sœur, moins à l'aise avec le clavier, classe elle aussi ses photos avec le même logiciel, mais elle se moque de savoir que la touche majuscule du clavier est restée enfoncée. Du coup, elle sélectionne une photo et entre le mot clé "FRANCAIS". Aie... un nouveau mot clé est créé et affecté à la photo.

Dans sa liste de mots clés, notre utilisateur méticuleux se retrouve à la suite de ces opérations avec les mots clés suivants : "Français", "Élégant", "elegant", "FRANCAIS". Et bien entendu les mots clés qu'il a saisis proprement ne sont affectés à presque aucune photo...

Une vraie salade. Il décide d'appeler le support technique du logiciel...

Le développeur de base

“Mais, cher utilisateur, notre logiciel fonctionne sous .NET qui est Unicode et gère parfaitement toutes les spécificités de toutes les langues, si vous ne savez pas taper les mots de votre propre langue proprement nous n’y pouvons rien !”

C’est en général ce que répondra un développeur dans ce cas là...

Mauvaise réponse. Car très mauvaise UX. Et oui, la fameuse “expérience utilisateur”... Eviter les frustrations, rendre les choses claires et limpides, et tout ça... ça vous revient ? ...

Et l’UX alors ?

Dans un tel cas il est évident que le logiciel devrait reconnaitre “FRANCAIS” et “Français” et corriger automatiquement, voire proposer un dialogue avertissant l’utilisateur et lui demandant s’il souhaite vraiment créer un nouveau mot clé ou plutôt utiliser celui qui en semble le plus proche...

Pourquoi cela ne marche-t-il pas automatiquement ?

Tout simplement parce que la sophistication du système utilisé par le Framework .NET (et d’autres dans le même cas) ne s’encombre pas de l’usage courant, seule la pratique “officielle”, sérieuse, normalisée et orthodoxe est prise en compte, et ce n’est déjà pas si mal lorsqu’on se souvient du pénible de la situation avec des environnements anciens comme Delphi ou VB sous Win32 qui se moquaient pas mal de la localisation des chaines de caractères ou des dates.

Ainsi, “français accentué” sera traduit par la méthode String.Upper() par “FRANÇAIS ACCENTUÉ”, ce qui est parfaitement exact. Rien à redire.

Mais la plupart des logiciels ne sont pas conçus pour être manipulés par les membres de l’Académie Française disposant de toute une retraite dorée pour travailler sur une seule lettre de l’alphabet avec des sbires s’occupant de transcrire tout cela au rythme ronronnant d’une vieille institution engluée dans les rouages grippés de l’administration à la française...

Hélas, le français moyen, qui ne sait taper du texte qu’avec deux doigts et encore, en phonétique SMS, n’en a que faire de la beauté des symboles diacritiques et il se fiche royalement de nos code page 1252 et de l'Unicode du Framework .NET.

Lui, c’est simple, ce qu’il veut c’est que la machine “comprenne” ce qu’il lui dit. Et que ce soit la machine qui s’adapte à lui et non pas l’inverse.

L’utilisateur est un idiot inculte pour nous autres geeks, c’est possible, mais d’un autre côté il a bougrement raison !

Une solution avec le Framework

Le Framework .NET n’est pas si rigide que ça... Seul le développeur est en cause, à lui d’avoir conscience des problèmes potentiels qui peuvent se poser dans certains contextes, à lui d’apprendre à se servir de ce qui est à sa disposition.

Car même si un simple “Upper()” ne règle pas la question ici, nul besoin d’inventer des bricolages plus ou moins alambiqués qui, de plus, risquent fort de ne pas marcher dans tous les cas.

Dans notre cas ce que nous souhaitons c’est simplement supprimer les symboles diacritiques dans une chaine. Ensuite il est facile de la passer en Lower() ou en Upper() et de faire des comparaisons pour rendre l’expérience utilisateur simple et agréable. Et lui donner envie de consommer encore plus de logiciels et d’en acheter plein au lieu de pirater en se donnant pour bonne conscience la raison que les produits sont mal fichus...

Voici le code d’une méthode qui fera exactement ce que nous souhaitons, elle utilise des fonctions du Framework .NET qui ne sont pas forcément utilisées tous les jours, raison de plus pour s’y intéresser, car souvent la solution est déjà dans le Framework.

```
public static String RemoveDiacritics(String s)
{
  String normalizedString = s.Normalize(NormalizationForm.FormD);
  StringBuilder stringBuilder = new StringBuilder();

  for (int i = 0; i < normalizedString.Length; i++)
  {
    Char c = normalizedString[i];
    if (System.Globalization.CharUnicodeInfo.GetUnicodeCategory(c)
         !=  System.Globalization.UnicodeCategory.NonSpacingMark)
      stringBuilder.Append(c);
  }

  return stringBuilder.ToString();
}
```

Et voici un exemple d’utilisation :

```
var original = "Français accentué";
var simpleUpper = original.ToUpper();
var noDiacritics = RemoveDiacritics(original);
var upper = noDiacritics.ToUpper();

Console.WriteLine("Original: "+original);
Console.WriteLine("Simple Upper: "+simpleUpper);
Console.WriteLine("W/o diacritics: "+noDiacritics);
Console.WriteLine("Upper: " + upper);
```

qui donnera la sortie suivante :

```
Original: Français accentué

Simple Upper: FRANÇAIS ACCENTUÉ

W/o diacritics: Francais accentue

Upper: FRANCAIS ACCENTUE
```

Conclusion

L'UX est un domaine passionnant, mais à côté des grandes théories, des envolées lyriques et des discours philosophiques il y a la pratique. Et l'UX en pratique cela va se nicher partout, même dans un simple problème de majuscules et de symboles diacritiques...

Avertissements

L'ensemble de textes proposés ici est issu du blog « Dot.Blog » écrit par Olivier Dahan et produit par la société E-Naxos.

Les billets ont été collectés fin septembre 2013 pour les regrouper par thème et les transformer en document PDF cela pour en rendre ainsi l'accès plus facile.

Les textes originaux ont été écrits entre 2007 et 2013, six longues années de présence de Dot.Blog sur le Web, lui-même suivant ses illustres prédécesseurs comme le Delphi Stargate qui était dédié au langage Delphi dans les années 90.

Ce recueil peut parfois poser le problème de parler au futur de choses qui appartiennent au passé… Mais l'exactitude technique et l'à propos des informations véhiculées par tous ces billets n'a pas de temps, tant que les technologies évoquées existeront …

Le lecteur excusera ces anachronismes de surface et prendra plaisir j'en suis certain à se concentrer sur le fond.

E-Naxos

E-Naxos est au départ une société éditrice de logiciels fondée par Olivier Dahan en 2001. Héritière de *Object Based System* et de *E.D.I.G.* créées plus tôt (1984 pour cette dernière) elle s'est d'abord consacrée à l'édition de logiciels tels que la suite Hippocrate (gestion de cabinet médical et de cabinet de radiologie) puis d'autres produits comme par exemple MK Query Builder (requêteur visuel SQL).

Peu de temps après sa création E-Naxos s'est orientée vers le Conseil et l'Audit puis s'est ouverte à la Formation et au Développement au forfait. Faisant bénéficier à ses clients de sa longue expérience dans la conception de logiciels robustes, de la relation client, de la connaissance des utilisateurs et de l'art, car finalement c'en est un, de concevoir des logiciels à la pointe mais maintenables dans le temps.

C#, Xaml ont été les piliers de cette nouvelle direction et Olivier a été récompensé par Microsoft pour son travail au sein de la communauté des développeurs WPF et Silverlight. Toutefois sa première distinction a été d'être nommé MVP C#. On ne construit pas de beaux logiciels sans bien connaître le langage…

Aujourd'hui E-Naxos continue à proposer ses services de Conseil, Audit, Formation et Développement, toutes ces activités étant centrées autour des outils et langages Microsoft, de WPF à WinRT (Windows Store) en passant par Silverlight et Windows Phone.

A l'écoute du marché et offrant toujours un conseil éclairé à ses client, E-Naxos s'est aussi spécialisée dans le développement Cross-Plateforme, notamment dans le mariage des OS Microsoft avec Android, les deux incontournables du marché d'aujourd'hui et de demain.

N'hésitez pas à faire appel à E-Naxos, la compétence et l'expérience sont des denrées rares !