Remarque : ceci est un travail de maturité = baccalauréat.
Il n'a pas de caution scientifique ou autre, et, bien que cet élève ait fait un travail qui a été accepté dans le contexte scolaire, son contenu n'engage que lui !

Travail de maturité 2009-2010, Collège Calvin

La sécurité informatique et l'ergonomie : Quels enjeux dans les systèmes informatiques aujourd'hui ?

Christian Mouchet, groupe 402

Maître accompagnant : F. Lombard

Table des matières

Signalétique :

Les renvois à la bibliographie sont signalés par le numéro de l'ouvrage sous la forme : [x].

Les mots écrits en bleu sont des termes techniques expliqués dans le **glossaire**.

Introduction

Le 20 janvier 1992 en début de soirée, le vol 148 Air Inter à destination de l'aéroport de Lyon Saint-Exupéry s'écrase près du mont Saint-Odile, en Alsace, faisant 87 morts et 9 blessés graves. Les experts déterminent que la cause du crash était technique. L'Airbus A320 était l'un des appareils les plus informatisés du moment. Le principe sur lequel cet avion était construit était de substituer un maximum l'action du pilote par l'action informatique, réduisant ainsi le risque d'erreur humaine.

Mais l'étude de la boîte noire finira par prouver que l'erreur était en grande partie humaine: il y aurait eu confusion entre deux valeurs, toutes deux affichées sur l'ordinateur de bord. L'une étant la vitesse de descente (VS-Vertical speed), et l'autre l'angle de descente (FPA-Flight Path Angle), les deux valeurs étant affichées au même endroit, suivant le mode de descente choisi par le pilote. Celui-ci aurait alors entré la valeur 33, pensant programmer un angle de descente de 3.3°, mais avait en réalité programmé une vitesse de descente de 3300 pieds/minute (environ 4 à 5 fois trop rapide). Le pilote n'a pas eu le temps de corriger sa trajectoire, provoquant le crash.

Cet accident aurait-il pu être évité ? Comment, en voyant peut-être le système autrement, avec presque 18 ans d'évolution dans le secteur de l'informatique, aurait-on pu empêcher une telle chose de se produire de nos jours ?

C'est sur des problèmes de ce type que planchent les experts de l'informatique aujourd'hui, tentant de faire évoluer le confort et la sécurité d'utilisation en parallèle des autres progrès qui sont fait dans la matière. Avec l'avènement d'internet, nous avons assisté à une hausse considérable de l'utilisation d'ordinateurs à des fins les plus diverses. Cela va de l'utilisation documentaire, divertissante, mais aussi administrative (E-Banking) ou commerciale. Le nombre d'informations, parfois sensibles, circulant sur internet est devenu immense. De nouvelles fonctionnalités, de nouveaux utilisateurs, mais aussi de nouveaux risques et nouvelles menaces : le domaine de la sécurité informatique est en perpétuelle course pour rester à niveau. On se souvient bien de la négligence humaine qui avait failli coûter sa crédibilité au forum de Davos, où l'administrateur avait laissé le mot de passe par défaut de la base de données. Il en avait résulté le vol d'une grosse quantité de données personnelles concernant des personnalités comme Bill Gate, Bill Clinton ou Yasser Arafat.

Dans ce travail, nous allons tenter d'étudier quels sont les enjeux de la sécurité informatique d'aujourd'hui. L'humain est-il vraiment, comme on l'entend souvent, le principal point faible de tout système informatique ? Et quels sont les enjeux pour l'ergonomie ? J'ai commencé par une étude de ce que sont ces deux domaines, puis, fort de cette vision théorique, je suis allé interviewer deux experts ayant chacun leur domaine respectif : l'un plutôt poussé techniquement et passionné de sécurité informatique, l'autre ayant l'expérience de l'ergonome. J'ai ensuite tenté de comparer leurs points de vue.

Ergonomie informatique

Définition :

L'ergonomie a pour but d'améliorer l'interaction entre l'humain et la machine. Le principe de cette science étant de trouver des compatibilités entre les caractéristiques des deux pôles du système : l'humain et la machine (pour autant que l'on considère l'humain comme faisant partie du système). Elles peuvent être physionomiques pour l'humain et techniques pour la machine, par exemple. Le mot employé par Amélie Boucher[2] me paraît tout-à-fait approprié : il est question d'adéquation entre ces caractéristiques.

Toute la difficulté réside essentiellement dans le fait que les caractéristiques évoquées plus tôt ont tendance à varier du côté de la machine et surtout du côté de l'utilisateur. La contrainte la plus évidente à voir, mais aussi la plus difficile à résoudre est bien entendu les connaissances qu'a l'utilisateur dans le programme qu'il va utiliser et même dans l'informatique en général. Mais au fil du temps, des critères ont fini par se dégager. Le document de l'Organisation Internationale de Normalisation ISO 9241-110 définit sa propre « charte ». Elle est celle qui est le plus souvent utilisée. Christian Bastien et Dominique Scapin[3][4] se sont eux aussi penchés sur la question, on retrouve leurs propres critères (très proches de ceux de l'ISO 9241-110 bien entendu) dans le livre « Web Design » de Julien Blanc[6]. Ce dernier définit lui aussi quelques points critiques de l'ergonomie d'un programme. À la page suivante, je vous présente ma synthèse de ces points, reprenant les idées des trois sources mentionnées plus haut. Les critères en gras sont les critères parents des « sous - critères » qui les suivent.

Critères ergonomiques, synthèse:

1. **Guidage :**
   Ce point rassemble tous les moyens **signalétiques** et informatifs qui doivent diriger l'utilisateur vers les bonnes actions par le chemin le plus bref. La qualité de guidage d'un programme se définit par la prise en compte des points suivants :

2. Incitation
   Pour une action donnée, le programme doit montrer à l'utilisateur ce que la machine attend de lui de façon précise. Le type de données que ce dernier va devoir entrer dans la machine, ainsi que le formatage de ces données, doivent être clairs. L'ordre utilisé dans le groupement doit être logique, il doit représenter l'ordre dans lequel l'utilisateur doit entreprendre les actions.

3. Groupement des éléments
   La disposition des éléments sur l'interface utilisateur doit être logique, cohérente et ordonnée. La position dans l'espace doit montrer l'appartenance d'un objet à un groupe d'objets déterminés par son type, son action, ou encore l'ordre chronologique dans lequel les objets doivent être utilisés. Il doit également et surtout montrer de manière logique sur quoi cette action va s'appliquer (justement en les groupant).

4. Feedback

   Chaque action de l'utilisateur doit avoir un effet visuel en plus de l'effet escompté. Il ne faut pas laisser l'utilisateur douter de l'efficacité de son action. Cela va de l'enregistrement du document au survol d'un bouton. C'est un facteur indispensable quand nous parlerons plus tard de la notion de boucle.

5. Lisibilité

   Le formatage du texte doit être lisible le plus rapidement et le plus confortablement possible. Les couleurs doivent êtres adaptée, la police claire et reposante. Les lignes doivent correspondre à la longueur de balayage des yeux C'est-à-dire d'environ 60 caractères avant de passer à la ligne.

   Cela comprend aussi le type de langage utilisé. Un vocabulaire technique conviendra plus à un utilisateur avancé, mais sera incompréhensible pour un utilisateur normal. Comme je l'ai dit plus haut, le tout est de cibler le groupe d'utilisateurs en question.

6. **Charge de travail**

   L'ensemble de toute l'interface utilisateur doit tendre vers la réduction au maximum de la charge intellectuelle sur l'utilisateur. Nous parlerons plus tard de l'utilisateur et des données biologiques qui l'accompagnent. Le principe de base est de reporter un maximum de la charge de travail sur la machine. Les critères suivants participent à cette réduction de charge.

7. Brièveté

   La brièveté consiste en deux facteurs : la concision et la minimalisation des actions.

   Le premier, la concision, vise à optimiser un maximum l'affichage des éléments à l'écran de sorte que l'utilisateur n'ait pas à faire d'effort pour traiter les informations. Le second vise à réduire le nombre d'actions effectuées par l'utilisateur, en fournissant des raccourcis, ou en combinant des actions en une. Il faut aussi éviter toute action inutile du côté humain comme du côté de la machine.

8. Densité informationnelle

   Pour faciliter l'analyse des informations, un programme doit faire le tri dans les informations affichées. Aucune information non-nécessaire ne devrait être affichée. Le regroupement des informations connexes est également très important.

9. **Actions explicites**

   Dans une grande majorité de cas, l'homme doit garder le contrôle sur sa machine. Il faut à tout prix éviter les actions du système prises de façon implicites (automatisées). Certaines actions répétitives peuvent néanmoins contrarier l'utilisateur s'il doit toujours les autoriser ou les initialiser. Une pratique courante est donc d'autoriser l'automatisation de l'action. (*Voir fig. 1 page suivante*)

Fig.1 Windows propose un système de mise à jour qui peut être automatisé à différents niveaux.

10. Contrôle utilisateur

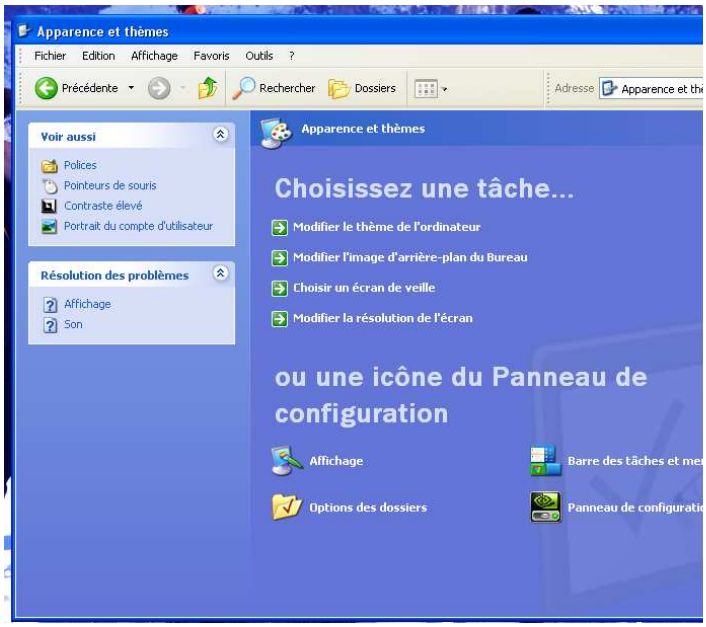
L'utilisateur doit également (au moins) avoir l'impression de contrôler son système. Il doit être en mesure d'annuler un ordre, de revenir sur les conséquences d'une action (annulation) et de réparer les conséquences d'une maladresse.

11. Adaptabilité

L'adaptabilité comprend deux critères : la **flexibilité** et l'**expérience utilisateur**.
La flexibilité définit la capacité d'une interface à s'adapter à l'utilisateur. Elle doit offrir la possibilité de personnaliser son espace de travail avec les actions les plus fréquemment utilisées. Cela permet de viser beaucoup plus d'utilisateurs avec le même produit.
L'expérience utilisateur est donc le critère qui détermine si le programme est adapté aux connaissances de l'utilisateur. En règle générale, une aide doit être apportée aux utilisateurs novices, tandis que des raccourcis doivent être proposés aux expérimentés. (*Voir fig. 2 et 3*)

12. Gestion des erreurs

Tous les critères précédents visant entre autre à réduire au maximum le risque d'erreur, un programme doit également et surtout avoir une gestion d'erreurs très complète. Ce point est critique et sera repris par la suite.



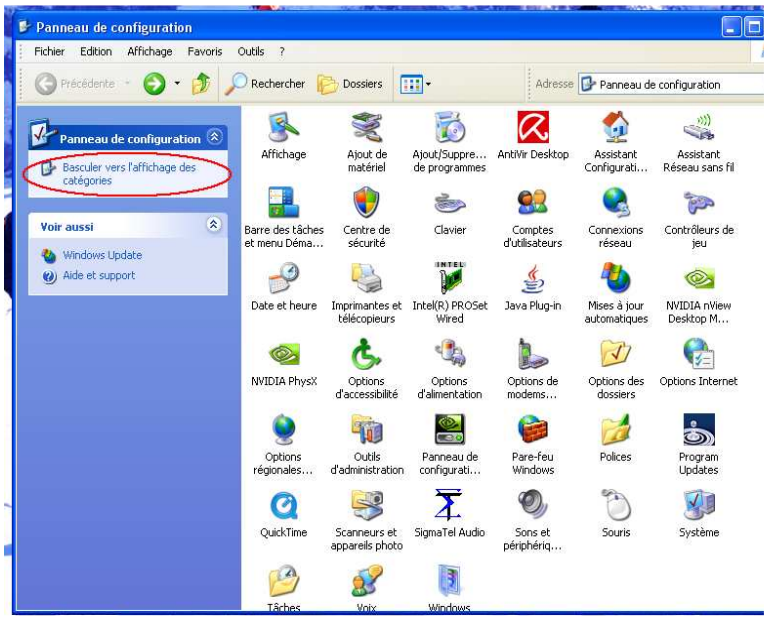


Fig. 2 et 3, le panneau de configuration des options Windows a deux modes d'affichages.

Interface orientée utilisateur (User-Friendly)

Le docteur en ergonomie Jakob Nielsen est le fondateur de l'entreprise Nielsen Norman Group, une société d'expertise en ergonomie. Il a travaillé pour de très grands noms tels que Bell, IBM ou Sun Microsystème.

Son idée générale est de simplifier les interfaces, notamment et surtout les sites web, afin d'en faciliter l'accès et l'*utilisabilité* (du terme anglais de Nielson *utilisability*) à l'utilisateur. Voici, résumées en 10 points critiques, les caractéristiques importantes de Nielsen (2004) quant aux interfaces orientées utilisateur :

1) Visibilité du statut du système (Barre de chargement, sablier, logs)

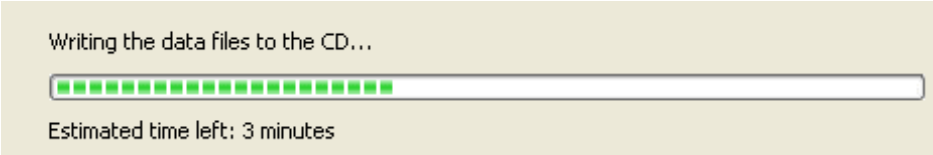


Fig. 4 Une barre de chargement dans l'environnement Windows. Elle peut proposer un pourcentage de l'avancement chiffré, imagé et même une estimation de temps.

2) Points communs entre l'interface et le monde réel. La logique doit rester la même que celle que l'utilisateur a l'habitude de fréquenter. Les éléments doivent être disposés de la même façon et même dans certains cas être contraints aux mêmes règles (physiques).

3) Liberté et contrôle de l'utilisateur. L'utilisateur doit rester maître de son système. Il doit être en mesure de donner les ordres mais aussi de les retirer quand bon lui semble, surtout dans le cas d'une erreur (*Undo-redo*).
4) L'interface doit avoir les mêmes standards que l'environnement dans lequel elle est exécutée. Par exemple, dans l'environnement MacOs d'Apple, les boutons permettant de fermer, réduire ou agrandir la fenêtre sont situés en haut à gauche de cette fenêtre.

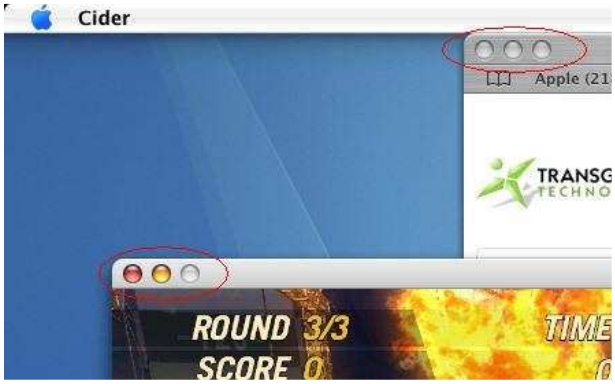


Fig. 5 Entourés en rouge, les boutons cités dans l'exemple.

5) Prévention des erreurs de l'utilisateur. L'interface devrait, dans l'idéal, effectuer une vérification des actions de l'utilisateur. Si celui-ci entreprend une action irréversible, importante ou compliquée, l'interface devrait lui demander s'il est vraiment sûr de vouloir l'entreprendre.

6) Mémoire. Le système ne doit pas attendre de l'utilisateur de retenir les informations d'un dialogue à un autre si cela ne lui est pas demandé explicitement. Toutes les informations doivent apparaître à l'écran ou être facilement retrouvables.

7) Flexibilité et efficacité d'utilisation. Le système doit permettre à des utilisateurs avancés de travailler plus vite. Cependant ces raccourcis devraient être invisibles aux utilisateurs novices pour empêcher de fausses manipulations. (*voir fig. 6*)

8) Simplicité du design et esthétique. Le système ne doit pas afficher plus d'éléments qu'il n'en faut pour la bonne compréhension d'un dialogue, mais que le message passe quand même (Une croix rouge pour une erreur, une coche verte pour une réussite).

9) Aider l'utilisateur à agir face aux erreurs. Celles-ci doivent êtres expliquées avec des mots compréhensibles par n'importe quel utilisateur, et surtout il doit être donné une marche à suivre (ou au moins une piste pour les utilisateurs plus aguerrits) sur l'action à entreprendre pour résoudre l'erreur.

10) Aide et documentation. Une documentation complète mais pas trop vaste doit être mise à disposition de l'utilisateur. Celle-ci est dans l'idéal organisée par tâche que l'utilisateur voudra accomplir, sous forme de marche à suivre.

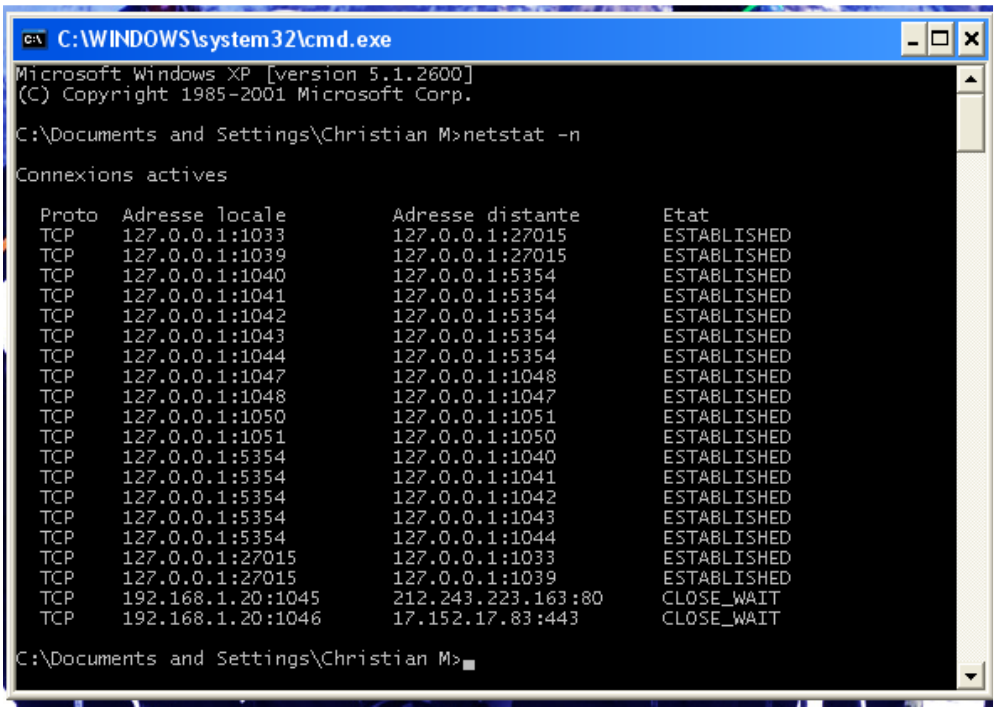


fig. 6 La console Windows est un outil puissant permettant d'effectuer très vite des actions longues à effectuer depuis l'interface normale. Son aspect très repoussant annonce une utilisation réservée aux utilisateurs avancés.

Le *Social Engineering* :

Le *Social Engineering* est, en quelque sorte, l'ergonomie appliquée à des fins malhonnêtes connues sous le nom de hacking. Le livre *L'art de la supercherie* par K. D. Mitnick détaille comment des hackers créent des programmes ressemblant en touts points à ceux que l'utilisateur a l'habitude d'utiliser, ceci afin de récupérer des informations vitales entrées dans ce programme par l'utilisateur croyant avoir affaire au programme qu'il utilise ordinairement.

La notion la plus utilisée ici est un guidage (critère n°1) parfait de l'utilisateur, pour obliger celui-ci à effectuer les actions que le hacker ne peut effectuer d'où il se trouve. Comme par exemple un faux formulaire de payement en ligne qui n'enverrait pas les données vers le site voulu mais directement sur la machine du hacker.

Exemple d'une attaque de type *Social Engineering* :

fig.7 L'exemple prend le cas d'un malware s'exécutant au démarrage de Windows. Il simulait une demande d'activation de la copie de Windows où l'utilisateur devait entrer des informations de payement avec une carte de crédit.

fig. 8 Ces informations étant trop sécurisées pour être accédées par un programme dans l'ordinateur de la victime, le programme fait en sorte que l'utilisateur entre cette information pour lui. Le malware envoie alors ces données au pirate.

La sécurité informatique

Définition :

L'objectif ici n'est pas de donner les réponses et les solutions aux problèmes de sécurité informatiques, mais bien de faire ressortir les enjeux qui y sont rattachés.

Pour commencer, une brève (re)définition de ce qu'est l'informatique s'impose. Au-delà de notre navigateur internet, notre email ou notre traitement de texte, l'informatique a pour principe, et son nom l'annonce bien, d'être une structure d'informations et de traitement. Le principe est de pouvoir entrer des informations dans un système, mais aussi de pouvoir les en sortir. Là où l'informatique dépasse un classeur, c'est qu'elle est capable d'effectuer sur ces informations des opérations qui ne sont pas envisageables pour un humain. En plus, elle permet au même titre qu'un classeur de stocker ces informations. Un bref exemple concret :

Un automobiliste se fait photographier par un radar. Une photographie de la plaque est effectuée et envoyée à un ordinateur, un système informatique, qui va traiter l'information. Ceci comprend notamment l'extraction du numéro de plaque du véhicule. Celui-ci sera alors stocké dans le système, dans l'attente d'être consulté (sorti) par un humain, ou par le système dans le but d'une nouvelle analyse, l'envoi automatique d'une contravention par exemple…

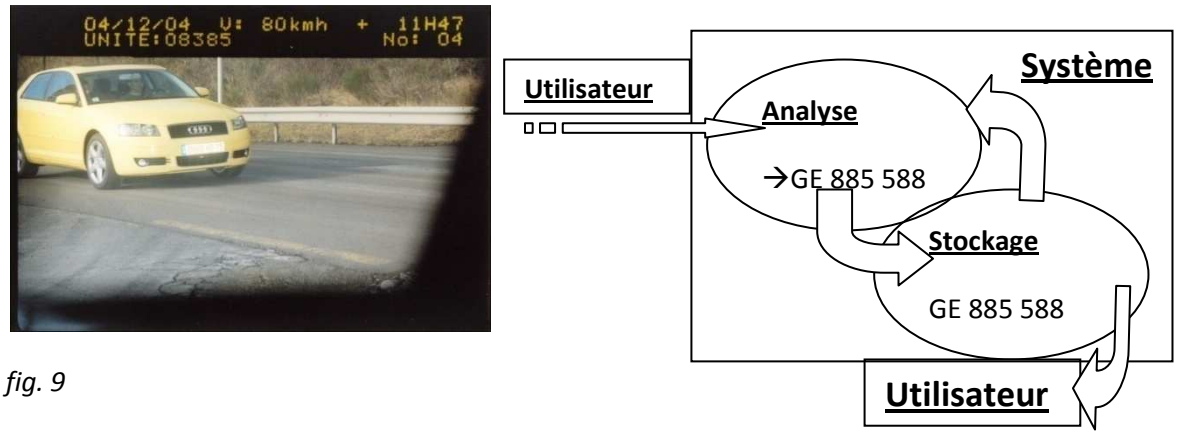


fig. 9

Maintenant que nous avons défini les deux actions principales d'un système, nous allons pouvoir définir ce qu'est la sécurité informatique. *Le petit Larousse illustré* définit le mot « sécurité » par une « *situation dans laquelle quelqu'un, quelque chose, n'est exposé à aucun danger, à aucun risque d'agression physique, d'accident, de vol, de détérioration* »[17]. Deux notions ressortent nettement : le danger et le risque. Bien que souvent employés de manière assez proche, ils ne sont pas identiques, nous verrons plus loin pourquoi. Les premières choses que l'on regarde pour évaluer la sécurité d'un système sont les risques.

Eric Léopold et Serge Lhoste, dans le livre *La sécurité informatique*, définissent 3 grands types de risques[5]. C'est l'approche la plus courante et la plus exhaustive. Les risques portent sur les critères suivants :

- La disponibilité : qui juge si le système est accessible au moment voulu, à un utilisateur ayant le droit de l'utiliser. La **conséquence** d'une défaillance à ce niveau peut être l'inutilisabilité du système par une personne légitime entraînant ainsi le non-accomplissement d'une tâche qu'elle devait effectuer, ou alors plus grave : l'accès au système à une personne non-autorisée, lui permettant d'effectuer des tâches qu'elle n'était pas sensée effectuer.
- L'intégrité : qui juge si les informations stockées dans le système sont exactes et complètes. Nous avons vu plus haut ce qu'était vraiment l'informatique, on se rend donc compte de l'importance de l'intégrité. Ici, la **conséquence** d'un problème est que l'utilisateur du système (ou pire : le système lui-même, ce qui provoquerait un plantage) ne puisse plus accéder aux données qu'il veut, accède à des données erronées, ou accède à des informations contradictoires.
- La confidentialité : elle assure que seules les personnes autorisées à accéder à certaines données (informations) puissent le faire. Ce critère comprend donc une séparation des données par leur sensibilité, étant donné qu'une confidentialité totale est impossible en informatique (par exemple sur le web). Les **conséquences** d'une défaillance de confidentialité sont évidement qu'une personne non autorisée puisse accéder à des informations sensibles qu'elle n'est pas sensée voir, et puisse donc les utiliser à mauvais escient.

D'ailleurs, depuis que les systèmes informatiques disposent de plus de mémoire disponible pour stocker des données, un nouveau critère de traçabilité est couramment utilisé dans l'analyse de la sécurité d'un système. Ce critère veut que chaque action soit enregistrée, ainsi que la cause (qui l'a initialisée), le moment, et son succès (ou non). De cette façon, un administrateur peut facilement déceler les failles de sécurité ou les causes de pannes de son système.

La deuxième partie de la définition du mot « sécurité » est le danger. Pour cette notion, la meilleure définition que j'ai trouvée est la norme française, dans le secteur de la santé, NF-EN-1441. « Le danger est la combinaison entre la conséquence de l'évènement redouté (mis en gras plus haut) et sa probabilité d'occurrence ».

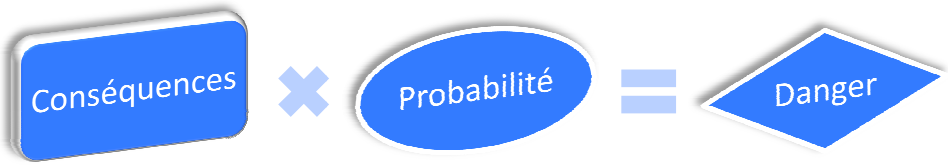


fig. 10

Cette définition est tout à fait applicable en informatique. Plus les conséquences d'une attaque (d'un hacker par exemple) seront grandes, plus le danger qu'elle représentera sera grand. En revanche, plus le système de sécurité est capable de contrer ces attaques, moins le danger sera grand. Ce qui amène inévitablement le développeur du système de sécurité à régler en priorité les problèmes de sécurité ayant des conséquences importantes.

Les menaces

D'où ces dangers viennent-ils ? Les experts en sécurité séparent souvent les menaces par types d'utilisateurs du programme.

- L'utilisateur maladroit : « *l'erreur est humaine* », l'utilisateur risque de modifier, supprimer ou altérer des données accidentellement, exécuter une action non-voulue également. L'intégrité des données est alors mise en péril, et peut même, à l'extrême, rendre le système instable et affecter sa disponibilité.
- L'utilisateur inconscient : qui risque d'introduire un logiciel espion ou malveillant dans le système sans le savoir. Il est aussi une cible facile pour les attaques de type « social engineering » dont il a été question. Cet utilisateur compromet alors la disponibilité du programme, car celui-ci pourrait alors être contrôlé par une personne à l'origine du programme malveillant. Plus couramment, il compromet la confidentialité des données de son système (logiciel espion).
- L'utilisateur malveillant : qui veut délibérément mettre en péril la sécurité d'un système, soit pour en sortir une information confidentielle, soit pour en prendre le contrôle. Il se sert en général des faiblesses de l'utilisateur inconscient pour accomplir ses méfaits.

La lecture d'un livre de Kevin D. Mitnick, un ancien pirate informatique qui travaille maintenant en tant que consultant en sécurité informatique, nous éclaire sur le mode opératoire des utilisateurs malveillants. Dans *L'art de l'intrusion*[7], K. D. Mitnick raconte sous forme de courtes histoires comment il s'est introduit dans différents systèmes informatiques. Ce qui est intéressant, c'est que dans plus de deux tiers des histoires, le pirate a exploité la faille de l'utilisateur inconscient. Dans son autre livre *L'art de la supercherie*, il développe ses procédés pour pousser cet utilisateur à effectuer les actions requises à l'intrusion. Nous avons déjà vu ça, il s'agit du *social engineering*.

Que peut-on ressortir de l'analyse de ces deux approches, sécurité et ergonomie, de l'interaction homme-machine ? On voit bien que l'humain est le point crucial du système qu'il utilise. N'oublions pas que ce système a été créé par l'humain, pour l'humain. Ceci me conduit donc à choisir une vision spécifique du système : La boucle opérateur-machine.

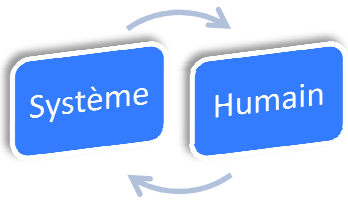


fig. 11

La vision qui ne tient pas compte de l'humain dans le concept de système n'est selon moi, pas exhaustive. Je le répète : par l'humain, pour l'humain. Une vision cyclique du système nous fait déjà nous rendre compte de l'importance de l'humain dans la boucle. Imaginer un système fonctionnant sans prendre en compte l'humain équivaut à négliger non seulement la cause première

de ce système, mais aussi et nous l'avons vu dans la définition de la sécurité informatique, l'une de ses principales causes de défaillance.

Le système avait déjà une première responsabilité qui était une sécurité garantissant intégrité, disponibilité et confidentialité. Mais cette vision lui impose maintenant une deuxième responsabilité : une bonne communication avec son binôme, l'utilisateur, dans le but d'une meilleure sécurité bien entendu. Un utilisateur bien accompagné fera beaucoup moins d'erreurs, et sera aussi moins susceptible d'être l'outil d'une attaque de type *social engineering* car le système devrait l'avoir découragé d'effectuer une action douteuse contre son gré [1]. L'ergonomie, que je voyais avant comme une tension s'opposant à la sécurité, devient un facteur essentiel à celle-ci.

fig. 12 Un autre exemple d'attaque de social engineering, ici un programme malveillant tente d'imiter l'apparence d'une commande afin que l'utilisateur l'utilise par mégarde. Mais un utilisateur plus aguerrit, pour autant qu'il prenne le temps de regarder un peu plus attentivement, se rend facilement compte de la supercherie.

L'exemple est un virus infectant les clés USB. Windows propose automatiquement un choix d'actions possibles en lisant un fichier présent dans la clé USB. C'est en infectant ce fichier que le virus ajoute l'option qui exécutera en fait un programme malveillant à la demande de l'utilisateur .Tout ceci afin de contourner la sécurité posée par Windows empêchant tout exécution automatique de

Interviews

J'ai choisi d'interviewer deux professionnels, tous deux spécialisés dans le développement d'applications web (de sites internet) : Patrick Mettraux et Eric Pasquier. Mon choix s'est porté sur deux visions que je pensais différentes sur l'interaction entre l'utilisateur et le programme. En effet, en ayant brièvement travaillé avec P. Mettraux dans le cadre du développement d'un site web, je me suis rendu compte qu'il préférait laisser le graphisme aux graphistes, pour se consacrer surtout au système de sécurité : sa spécialité.

E. Pasquier a pour particularité de toucher à la fois le domaine de la programmation et celui de l'ergonomie. Je ne connaissais pas M. Pasquier, il m'a été présenté par François Lombard qui travaille avec lui au TECFA de l'université de Genève.

J'ai enregistré les interviews afin de les écouter à nouveau et essayer de retranscrire les propos tenus de la manière la plus fidèle et synthétique possible.

Interview : Patrick Mettraux, programmeur

Patrick Mettraux est un programmeur (essentiellement web, PHP, Ajax, qui sont des langages de programmation pour internet) que j'ai connu sur un projet de développement d'un site web, au sein de l'entreprise Cjonline. Il en a programmé intégralement tout le noyau dur au niveau sécurité. Sécurité pour laquelle il s'est passionné et spécialisé.

Les étapes de développement selon P. Mettraux :

Selon P. Mettraux, l'expérience du programmeur lui permet de ne pas s'attarder sur la conception graphique. Toutefois, il est important de peser le poids de l'utilisateur dans le programme. Par exemple, une interface de communication (ex. un panneau d'administration) ne sera pas pensé la même chose qu'un noyau de programme comme le système de sécurité. Il suggère même que l'interaction avec l'utilisateur est un souci que le programmeur ne peut pas s'offrir le luxe de résoudre lors du développement d'une telle application. Surtout à cause du fait que cette partie du programme se doit de fonctionner quel que soit le comportement de l'utilisateur.

La base de cette sécurité repose toutefois sur cet utilisateur. Ainsi, la première étape pour P. Mettraux est de coder toute la partie qui aura pour but d'identifier l'utilisateur du programme. Le web étant une connexion quasi anonyme, *de tout le monde vers tout le monde*, il est évident que la première action soit de s'assurer de l'interlocuteur. Cette étape demande d'être détachée totalement de toute l'interface graphique.

La seconde étape consiste à construire le système de sécurité autour de cet utilisateur. C'est à ce moment-là que sont définies les actions autorisées, non-autorisées, les accès, les droits, etc. C'est une phase qui demande également à être détachée de l'interface graphique. Cette étape est cruciale car la moindre faille pourra être exploitée. Elle nécessite une idée précise et détaillée du

système de sécurité dans son ensemble, afin de prévoir non seulement les attaques venant du dehors, mais aussi les erreurs et mauvaises manipulations venant du dedans.

L'avantage premier de détacher ces deux « couches » totalement des interactions avec l'utilisateur a déjà été évoqué : il s'agit que le noyau dur du programme soit toujours opérationnel, peu importe les autres parties du programme. Mais il y a autre chose : s'il n'y a qu'un minimum de rapports entre les fonctionnalités du programme et son système de sécurité, alors la plus grande partie de ce système est réutilisable pour d'autres applications à quelques modifications près. C'est un gain de temps pour le développement de l'application suivante du programmeur.

Par-dessus cette couche vient ensuite s'implémenter le programme spécifique, avec tâches et actions spécifiques. Avec la sécurité, c'est le travail le plus long et le plus complexe à réaliser. A côté de cela, le graphisme ne représente qu'une infime partie du travail. Selon P. Mettraux, le gros problème est que, en entreprise, la tendance est de laisser pour la fin ce travail de graphisme et de finir par devoir le « bâcler 2 jours avant la réunion ». Quand il travaille en *freelance*, il préfère de loin mettre sur pied une maquette pour la confronter au client. Comme ceci, ce dernier peut alors y apporter les corrections qu'il désire sans que ce soit un poids pour le programmeur. De plus, au niveau marketing, le client est beaucoup plus satisfait d'avoir sous les yeux quelque chose de concret, « joli », conforme à la représentation qu'il se fait d'un site web, plutôt que de belles explications théoriques. Comme en général la communication avec le client peut être assez longue et laborieuse, le temps de « flottement » de cette confrontation peut être utilisé pour commencer le vrai développement, donc à partir de la sécurité. Car, il le rappelle, il faut qu'il y ait le moins de liens possibles entre le graphisme et le noyau dur du programme.

Les risques liés à l'utilisateur :

La première chose qui a été faite est de séparer différents types d'utilisateurs relativement à la sécurité. L'utilisateur lambda, celui qui utilise le programme normalement, avec l'intention d'utiliser ce programme et rien de plus (il n'est pas question ici d'une séparation par compétences). Puis l'utilisateur un peu plus poussé, qui, en tombant sur une faille de sécurité grossière, va l'exploiter modérément par simple curiosité, l'humain étant naturellement à tendance curieuse. Celui-ci est facilement « neutralisable », selon P. Mettraux, car un minimum de sécurité permet de boucher les trous de sécurité apparents, sur lesquels ce type d'utilisateurs peut tomber par inadvertance. Car le curieux ne cherche pas la faille. C'est donc le troisième type d'utilisateurs, plus communément appelé « pirate » ou « hacker », qui posera le plus de problèmes de sécurité. Car celui-ci recherche la faille, l'exploite, et en retire souvent un profit. Selon P. Mettraux, un site développé depuis le visuel a de fortes chances de ne pas résister à ce type d'utilisateurs.

Puis le problème de la sécurité trop imposante est abordé. L'utilisateur normal et « idiot » (sic), cette espèce « représentant environ 95% du web », il est important de ne pas aller trop loin dans la gêne occasionnée par un surplus de sécurité. Et dans ce cas, même la moindre des choses dérange déjà. L'exemple pris est celui du CAPTCHA, ces lettres qui s'affichent à l'écran et qui doivent être recopiées avant l'envoi d'un formulaire afin de s'assurer que la personne à l'origine de l'action est bien humaine (*voir fig. 13 page suivante*). Il provoque de nombreuses plaintes, mais sa suppression serait catastrophique car elle entraînerait une augmentation incroyable de la quantité de spam et autres robots dans le programme qui deviendrait alors inutilisable pour l'utilisateur.

fig. 13 Le CAPTCHA

Pour l'utilisateur lambda, un des risques évoqués est l'erreur. Et tout particulièrement l'erreur de saisie. Pour P. Mettraux, c'est un gros travail de contrer ce type d'erreur, car le programmeur doit penser à toutes les erreurs possibles. En revanche, la manière de contrer ces erreurs est très simple à réaliser. La priorité absolue ne semble pas être la réussite d'une opération, mais que l'échec de celle-ci fasse planter le programme. D'abord on pense aux dégâts, ensuite à la communication avec l'utilisateur.

Séparation design/programme :

P. Mettraux avoue d'abord ne pas du tout aimer « le graphique », il entend par là la création de l'interface usager. Pour lui c'est un métier à part entière qui doit être séparé du programmeur pour être vraiment efficace. Pour vraiment pouvoir pousser à fond la partie programmation, P. Mettraux est très perfectionniste quant à la fonctionnalité, l'intégrité et l'optimisation de ses programmes. Devoir faire le travail d'un graphiste est donc une perte de temps et aussi un point faible, car cela nécessite de faire quelque chose qu'on ne maîtrise pas bien.

Interview : Eric Pasquier, ergonome

Eric Pasquier est à la fois programmeur web et ergonome. En effet, son expérience dans le domaine de l'enseignement l'a conduit vers une approche tout-à-fait différente de l'interaction homme-machine.

Ancienne définition de l'ergonomie(1950) -> Livre ?

L'ergonomie est un ensemble de connaissances scientifiques, relatives à **l'homme**, qui peuvent servir à la création d'interfaces/espaces de travail, avec un maximum de confort, **sécurité**, efficacité. A ce point, il a été relevé que la sécurité est en fait à la base de l'ergonomie, et que l'idée de les opposer n'est pas forcement logique.

A quels types de problèmes ?

Qui est en danger ?

Dans un système tel qu'un site internet ou une application regroupant les données d'une multitude d'utilisateurs, deux types d'utilisateurs courent un risque. D'une part, les usagers du système qui entrent leurs données dans celui-ci, une perte de données étant l'un des principaux risques pour cet utilisateur. D'autre part, l'administrateur du système, qui doit veiller à son bon fonctionnement. Dans les deux cas, la victime est humaine. Ce qui nous conduit directement à chasser le raisonnement qui rend le système **désireux** (ex. de marcher, ou pas…), il ne peut donc pas vraiment être une victime.

Sur le web : l'usager normal ne court pas vraiment de risques.

Ebanking : l'usager court déjà plus de risques

Quel est le risque ?

Pour E. Pasquier, le risque principal reste quand même l'utilisateur. Il constate que les systèmes dotés d'interfaces (donc comprenant une interaction avec un utilisateur), sont plus sujets à des problèmes de sécurité que des systèmes qui n'ont que des interactions avec des machines. C'est pourquoi il semble crucial de séparer les algorithmes vitaux et importants de ceux qui gèrent la communication avec l'utilisateur.

Approche 1 : Comment voir la conception d'un système du point de vue de l'ergonomie.

E. Pasquier, lors de la conception d'un système, en l'occurrence un site internet, renverse le processus habituel. Ce processus habituel consiste à penser depuis le cœur du programme (100% algorithme) vers l'interface graphique (90% visuelle). Le problème de cette méthode est que l'ergonomie est souvent laissée de côté jusqu'au développement de l'interface. On fait donc « ce qu'on peut » pour rendre le programme facile, confortable à utiliser.

Le procédé inverse, celui que nous expose E. Pasquier, se soucie d'abord de la façon dont l'utilisateur va interagir avec le système, pour ensuite développer ce système sur la base de ces contraintes. De cette manière, l'ergonomie du programme est vraiment un plus pour la sécurité. Une interface bien conçue limite beaucoup le nombre d'erreurs.

Ce procédé implique donc de très bien connaître l'utilisateur, d'avoir identifié ses compétences et ses besoins. Il faudrait, au maximum, essayer de voir la tâche à accomplir comme l'utilisateur identifié la verrait. De façon générale, ce procédé demande une étape

pré-développement beaucoup plus importante. Car l'écriture du code à proprement dit devra de toute façon commencer par le cœur du programme pour aboutir à l'interface. Mais le fruit d'une telle méthode sera alors non pas un système « user-proof », mais un système dont l'utilisateur fait partie.

Approche 2 : Ergonomie comme le complément d'une faille humaine

La première chose qui est ressortie est que, de manière générale, l'utilisateur ne devrait pas avoir à réfléchir à autre chose qu'à la tâche qu'il est en train d'accomplir, dans le but d'éviter une distraction qui pourrait mener à une erreur. L'exemple évoqué est le défibrillateur public. Une petite valise qui dit à son utilisateur les actions simples à effectuer dans l'ordre et sans confusion possible. Ici on constate aussi que la machine prend en charge tout ce que l'utilisateur, une personne n'ayant aucune compétence médicale, ne sait pas faire.

On ne peut pas attendre de l'utilisateur d'être pleinement concentré sur le système à chaque instant. Une interface semblant alors claire et facilement utilisable peut soudain paraître complexe. Une machine étant, dans la plupart des cas, utilisée comme un accessoire en parallèle à d'autres tâches (conférence, bureau, divertissement), il faut alors garder à l'esprit que l'utilisateur ne dispose pas de toutes ses capacités pour cette machine. On parle alors de **charge cognitive**. C'est la quantité d'informations qu'une personne moyenne est capable de retenir ou de traiter à la fois. De façon chiffrée, les ergonomes se sont rendu compte qu'une personne moyenne était capable de retenir de 5 à 7 items (objets) à la fois. Seulement sur ces 5-7 items, tous ne sont pas disponibles pour une seule chose. Le rôle d'une interface est alors de décomposer les actions complexes en plusieurs actions simples, qui, de façon chiffrée, ne prendraient qu'un ou même zéro items par action.

Définition : Interactivité fonctionnelle et intentionnelle

L'**interactivité intentionnelle** est, en quelque sorte, le spectre des actions qu'un système est capable d'exécuter. Il est crucial qu'une certaine interface propose un certain type d'actions, sans pour autant prendre le risque de perdre l'utilisateur dans un trop grand nombre de choix. L'utilisateur qui cherche à faire quelque chose doit pouvoir trouver une action qui correspond à l'action la tâche à effectuer. Si c'est le cas, alors l'interactivité intentionnelle d'un programme est bonne, sinon non. Elle est donc très relative à l'utilisateur, qui voudra, selon ses capacités, fonctions etc., exécuter un certain type d'action.

L'**interactivité fonctionnelle** intervient après que l'utilisateur ait identifié l'action qu'il veut entreprendre. Elle décrit la façon donc l'action va être présentée, et la facilité d'utilisation. L'interactivité fonctionnelle est plus en relation avec l'interface, le visuel et les actions « physiques ». Plus l'action est facile à effectuer, plus l'interactivité est bonne. Ici il n'y a même pas besoin d'identifier l'utilisateur, les actions ayant été définies par l'aspect **fonctionnel** du programme.

Conclusion de l'interview

L'ergonomie n'est pas vraiment le problème du confort de l'usager. L'enjeu est plutôt l'efficacité et la sécurité. Le confort vient ensuite. De ce fait, l'ergonomie doit être prise en considération dès le début. Un développeur ne doit pas partir de sa propre vision du programme quand il en conçoit le maniement, mais de la vision de l'utilisateur final (utilisateurs finaux).

Analyse

Points communs

- Centralité de l'utilisateur

On constate d'abord que les deux interwievés s'entendent sur la centralité de l'utilisateur. P. Mettraux commence le développement de ses applications sur la base de celui-ci, c'est-à-dire depuis le système qui permet de l'identifier. Tout le système vient ensuite se tisser autour de cette identification.

E. Pasquier a commencé par définir l'ergonomie comme étant « relative à l'homme ». Il a aussi bien défini qui était la victime finale d'une défaillance ou d'une erreur : c'est bien l'humain. Le système, lui, est juste l'outil.

- Sensibilité de l'utilisateur

Si l'humain est la victime finale, elle est aussi la cause principale de défaillances. À nouveau, les deux experts sont d'accord là-dessus. Sauf dans le cas d'un système complexe comprenant de nombreux services ou sous-programmes (comme un système d'exploitation, par exemple), où les notions d'exception et d'incompatibilité rentrent en ligne de compte, une défaillance vient toujours de l'humain.

E. Pasquier ne fait que très peu mention du cas où l'utilisateur est malveillant. Il prend plus en compte le risque d'erreur, contrairement à P. Mettraux qui n'en a fait mention qu'à ma demande (*« comment gérez-vous l'erreur ?»*). Son souci était plus proche des dangers directs du web quant aux utilisateurs malveillants.

- Analyse du comportement utilisateur

Tous deux m'ont également décrit une partie du développement où ils soumettaient une maquette du programme à l'utilisateur, ceci dans le but d'étudier son comportement face au programme. Toutefois, le but recherché semble être différent. E. Pasquier parle plutôt d'une étude poussée, où on demande à l'utilisateur d'exécuter une tâche et en regardant la façon dont cet utilisateur voit le programme, cela pour éviter que le programmeur crée le programme selon sa vision et non celle de l'utilisateur final. Quant à P. Mettraux, l'étude porte plutôt sur l'aspect

visuel et sur l'impression de l'utilisateur sur le programme, ceci dans le but de créer le programme correspondant aux envies du client.

Points divergents

- Traitement de la faille humaine

Les deux experts sont d'accord sur le fait qu'un utilisateur lambda ait plus ou moins souvent des défaillances et commette des erreurs. Cependant, les deux n'ont pas du tout la même façon de résoudre le problème. C'est ici que les visions divergent le plus.

La solution que semble proposer E. Pasquier est d'agir sur l'utilisateur, par le biais de l'ergonomie, en décomposant les actions en petites actions simples ou en « ménageant » l'utilisateur en limitant la densité informationnelle. Les solutions qu'il propose se retrouvent en grande partie dans la synthèse des critères ergonomiques que j'ai faite au début.

Pour ce qui est de la réponse de P. Mettraux à ma question, on dirait qu'il privilégie un développement « user-proof » du programme, en tentant de faire face à toutes les éventuelles erreurs. La priorité est mise sur le système, car celui-ci doit rester opérationnel et intègre pour les autres utilisateurs, en revanche il ne semble pas se soucier de l'intégrité de l'utilisateur qui pourrait bien être en train de « s'arracher les cheveux » devant son écran. La gestion des erreurs est un critère ergonomique (12), toutefois elle devrait donc être accompagnée d'un feedback, comme dit au point 4.

- Séparation par compétences

Seul E. Pasquier a évoqué une séparation des utilisateurs réguliers du programme par leur niveau de compétences. C'est le 11^{e} critère ergonomique que j'ai fait ressortir au début. « L'adaptabilité du programme ». Il me semble que c'est un point critique. Les programmes proposent de plus en plus d'options pour passer d'un « mode normal » à un « mode avancé », qui permet à un utilisateur avancé d'effectuer des tâches plus sensibles tout en dissuadant un novice de faire de même au risque d'une erreur.

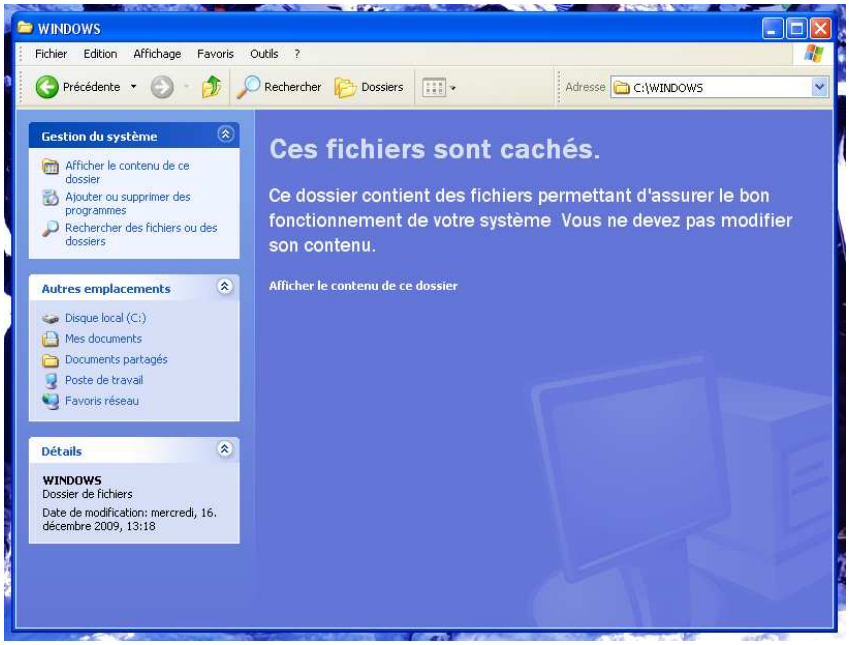


fig. 14 Windows cache ses fichiers systèmes « nativement », mais propose une option permettant de les afficher. Ceci en avertissant bien un utilisateur moins expérimenté du risque que cela comporte.

Conclusion

La façon de gérer l'erreur étant l'un des objets principaux de ce travail, je vais maintenant donner mon avis sur les différentes solutions proposées au problème. Pour commencer, il me semble que les deux approches prises seules sont insuffisantes. Même une très bonne ergonomie ne peut faire disparaître complètement le risque d'erreur. Dans l'optique d'un système important utilisé par beaucoup de personnes, l'approche « sécuritaire » me semble plus appropriée. Le problème de l'ergonomie est qu'elle ne semble pas être nécessaire, mais semble être plutôt comme « un plus ». La sécurité, elle, est unanimement nécessaire. En effet, on constate alors que sécurité et ergonomie ne peuvent pas être mises sur la même échelle d'importance. Une ergonomie non satisfaisante d'un programme sera palliée par une sécurité accrue (au niveau du contrôle des données entrées par l'utilisateur par exemple). En revanche, il est impossible de pallier de façon complète un problème de sécurité avec une bonne ergonomie. Imaginer toutes les erreurs que l'utilisateur peut faire et y remédier par l'ergonomie de manière totale est impossible. Ma propre expérience de programmeur m'a prouvé qu'on ne peut tout simplement pas attendre de tous les utilisateurs de se comporter de manière prévisible. Par contre, ne laisser rentrer dans le système que les données conformes, exploitables et attendues est possible. L'erreur de saisie (la faute de frappe) est toujours envisageable, mais même l'ergonomie ne pourrait résoudre ce problème, et le programme n'est pas menacé car les données seront quand même exploitables. Les autres utilisateurs du programme peuvent donc continuer leur tâche sans subir le « plantage » du programme. N'oublions pas non plus le 3^{e} critère ergonomique de Nielson : un programme doit offrir la possibilité à l'utilisateur de corriger ses erreurs. On ne peut pas corriger l'erreur si le programme a planté.

On constate alors que les deux caractéristiques devraient être présentes : l'une pour diminuer le risque d'erreur, l'autre en cas de dernier recours (persistance à l'erreur). Le manque de l'un ou l'autre n'est donc pas directement dû à la trop forte présence du second, mais souvent à un manque de temps, argent ou implication nécessaires à son développement. Je vois donc la sécurité informatique comme un pré-requis, une nécessité à tout programme. Une sécurité qui garantirait intégrité, disponibilité, confidentialité par divers moyens dont l'ergonomie ferait partie.

De plus, j'avais élargi la « faille humaine » en incluant le problème de l'utilisateur malveillant. Problème que même la meilleure ergonomie du monde ne peut pas entièrement résoudre. On s'aperçoit pourtant que les dangers auxquels fait face le programmeur qui crée un système de sécurité ne sont finalement pas beaucoup plus grands que ceux de l'ergonome. Pensons donc en terme de conséquences-probabilités. Les conséquences d'un piratage ou d'une action frauduleuse sont souvent de loin plus graves en termes de conséquences (vol de données confidentielles, d'argent...) qu'une erreur de la part de l'utilisateur, en revanche, la probabilité d'une telle attaque reste assez faible. À l'opposé, la probabilité d'erreurs de l'utilisateur reste très forte (moi-même il m'arrive de m'y reprendre à 2 ou 3 fois pour envoyer un simple formulaire sur internet, par exemple), mais les conséquences sont souvent minimisées car se limitent souvent au système lui-même et peuvent êtres contrées avec un minimum de sécurité. Je ne simplifie pas au point de dire que les dangers « s'égalisent », mais je pense que l'écart est minimisé.

Pourquoi alors a-t-on tendance à favoriser la sécurité et oublier l'ergonomie ? Selon Daniel Boy, la perception des risques est presque paradoxale. Il est dit qu'à dangers égaux, l'évènement ayant la moins forte probabilité est souvent beaucoup plus craint[9].

Une première conclusion générale que j'en tire m'est venue de la lecture de l'article de la revue *Bio-Tremplin* du 28 Novembre 2009[12] reprenant la définition du danger de la norme française NF-EN-1441 et la perception du risque selon Daniel Boy. L'humain a construit l'informatique de ses mains, il l'a faite à l'image du monde dans lequel il vit. L'un des points cruciaux de l'ergonomie est d'ailleurs le guidage, c'est-à-dire de faire ressembler le monde virtuel dans lequel évolue l'humain au monde qu'il connaît intuitivement, afin que cette intuition le guide aussi vers la bonne décision. Il est donc logique que ce monde soit exposé aux mêmes types de risques que son modèle. La perception du risque suit alors les mêmes règles et explique pourquoi l'ergonomie est souvent laissée de côté.

Prenons le cas d'un système ayant une faille de sécurité pouvant entraîner un plantage à cause d'une exception ne se produisant que très rarement et une maladie ayant des conséquences graves dans seulement une infime partie des cas déclarés (dans l'article, il est question de H1N1). Dans les deux cas, l'exception est beaucoup plus mal acceptée par les masses. Je me permets d'affirmer également que ce parallélisme est largement exploité par les créateurs de logiciels antivirus, présentant leur produit comme on présente un médicament.

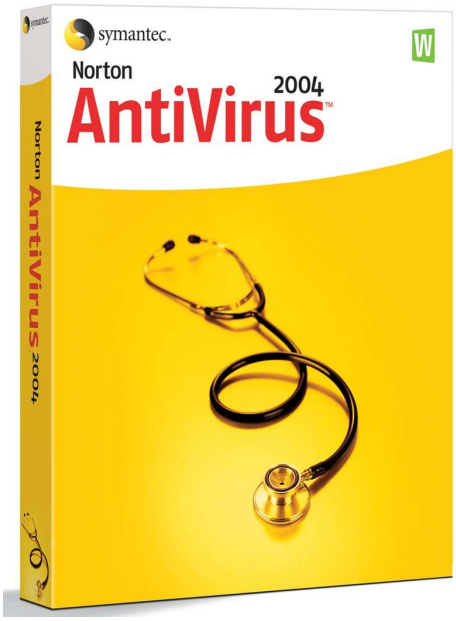


fig. 15

D'ailleurs l'appellation même de « virus » parle d'elle-même. L'informatique a pris d'innombrables expressions au langage courant (souris, dossier, fichier), mais la notion de virus est de loin la plus imagée et frappante.

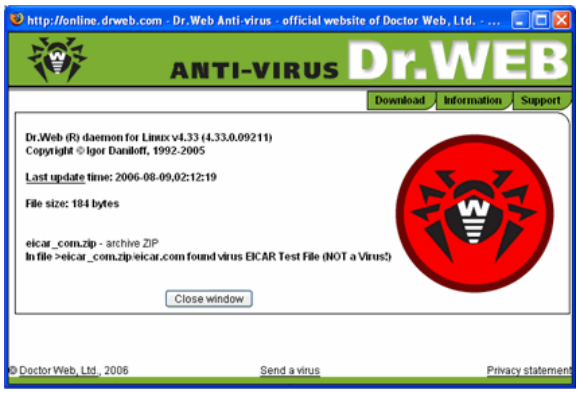


fig. 16

Néanmoins, cette ressemblance reste indispensable afin que les systèmes restent intuitifs et utilisables même par les plus réticents au passage vers une époque informatisée. C'est ici, je le pense, le principal enjeu de l'ergonomie dans le secteur informatique.

Reprenons maintenant nos deux affaires de crash et de vol d'informations. Pour le premier, une réponse ergonomique aurait sûrement suffi à éviter la catastrophe (les autres facteurs ayant entraîné le crash n'ayant pas pour conséquence de faire descendre l'avion). Si au moment de la conception de l'ordinateur de bord, les ingénieurs d'Airbus avaient fait une étude approfondie de la communication pilote-machine, ils auraient certainement buté sur des points comme l'incitation et le groupement des éléments (cf. critères ergonomiques). En effet, positionner les deux valeurs au même endroit et

de façon identique ne pouvait qu'entrainer la confusion entre ces deux valeurs. De plus, une analyse du critère n°6, la charge de travail, aurait conduit à une réflexion sur le fait que le pilote, ayant bien plus à gérer que l'ordinateur de bord, pourrait ne pas retenir, durant tout le vol, le mode de descente qu'il avait choisi auparavant. La réponse ergonomique est donc une différenciation évidente et visible des deux instruments.

De plus, une bonne sécurité au niveau de l'entrée des données aurait pu éviter que le fait d'entrer 3.3 soit interprété par l'ordinateur comme 3300. C'est une faille importante de sécurité car, même si l'entrée de la valeur ne provoqua pas de *crash* du système, elle provoqua le *crash* de l'avion.

Au niveau du vol des informations personnelles du forum de Davos, la réponse est encore plus simple. Aujourd'hui, la plupart des systèmes informatiques protégés par mot de passe ont un système d'évaluation de celui-ci inclus. Ce système avertit l'utilisateur lorsque le mot de passe employé est insuffisant (trop court, trop simple, **inchangé**).

De plus, de façon ergonomique, le système aurait pu guider (critère n°1) l'administrateur vers le choix d'un nouveau mot de passe dès l'installation même du système de base de données, en insistant sur l'importance d'une telle opération.

Le fait que ces deux incident auraient pu être évités aujourd'hui nous fait donc réfléchir d'avantage sur les problèmes actuels en matière de sécurité informatique et d'ergonomie. En effet, ils sont cruciaux pour empêcher les incidents de demain.

Glossaire :

Algorithmes (en informatique): Série d'actions effectuées de manière systématique. Ils peuvent avoir plusieurs buts. Par exemple, un algorithme de sécurité effectuera systématiquement une série d'actions destinées à sécuriser le système.

Application : Partie du logiciel dont l'utilisateur se sert pour effectuer les tâches pour lesquelles ce logiciel a été conçu. L'application comprend à la fois les actions de l'humain et l'automatisation de celle-ci.

CAPTCHA : Petit programme affichant une image avec des lettres ou des mots. L'utilisateur doit recopier ces lettres/mots dans une case pour confirmer qu'il n'est pas un robot. Ce système mise sur la difficulté qu'à une machine à reconnaitre dans un temps très court une lettre depuis une image.

Conception graphique : Il s'agit d'une étape dans le développement d'applications. Cette étape construit la communication visuelle entre l'utilisateur et sa machine. C'est l'étape où l'ergonomie est la plus importante.

Développement : Etape durant laquelle un logiciel est construit. Elle comprend de multiples étapes. Notamment la conception graphique de l'interface utilisateur, l'écriture du code et le test.

Exécution (informatique) : L'exécution d'un programme désigne le lancement du programme sur l'ordinateur, son utilisation.

Hack (er) : Le hack est un terme anglophone désignant l'activité du Hacker. Cette activité est principalement de s'introduire dans des systèmes informatiques privés pour voler des informations qu'il contient, ou en prendre le contrôle.

Interface utilisateur : En anglais « HMI, Human-Machin interface » elle a pour rôle la communication entre l'humain et sa machine.

Malware : C'est un programme malveillant. Il s'installe sur un ordinateur, souvent pour récupérer des données sensibles, ou effectuer des actions non-autorisées. Ils se propagent essentiellement par internet et les clés USB.

Programmeur : Profession consistant à écrire de manière textuelle les instructions que l'ordinateur devra suivre dans le cadre d'un programme. Ces instructions peuvent être écrites en plusieurs langages dépendamment de la machine, du système d'exploitation et du cadre d'utilisation du programme.

User-proof : Dans le jargon du programmeur, un système user-proof est un système qui. Confronté à l'utilisateur, ne comporte aucune faille de sécurité pouvant conduire l'erreur d'un utilisateur à un crash du système.

Remerciements :

Je commence par remercier mes trois collègues de TM Morgan Humbert, Benjamin Millet, et Christopher Harless pour la bonne dynamique de groupe qui s'est instaurée durant la réalisation de nos travaux. Cela n'aurait pas été pareil que d'avancer seul dans cette recherche.

Je remercie M. Patrick Mettraux, pour m'avoir accueilli chez lui et m'avoir consacré une partie de son temps pour répondre à mes questions.

Je remercie également M. Eric Pasquier, qui en plus d'avoir répondu à mes questions, a continué à m'envoyer des informations intéressantes.

Et tout particulièrement à mon professeur accompagnant : M. François Lombard, qui a dépassé de loin tout ce que j'attendais d'un accompagnant de travail de maturité. Sa disponibilité, son implication et sa motivation ont été des éléments-clé dans l'achèvement de ce travail.

Bibliographie :

[1] Andrew Rae, "Helping the Operator in the Loop: Pratical Human Machine Interface Principles for Safe Computer Controlled Systems," Adelaide, Australia: 2007, p. 10 (PDF)

[2] Amélie Boucher, "C'est quoi l'ergonomie informatique ?," Nov. 2003 (PDF)

[3] Amélie Boucher, "Les critères ergonomiques de Bastien & Scapin - Partie 1," Déc. 2003 (PDF)

[4] Amélie Boucher, "Les critères ergonomiques de Bastien & Scapin - Partie 2," Déc. 2003 (PDF)

[5] Eric Léopold et Serge Lhoste, *La sécurité informatique*, 73, avenue Rondsard, 41100 Vendôme: 2003

[6] Julien Blanc, *Web Design*, 44021 NANTES: 2005.

[7] Kevin D. Mitnick et William L.Simon, *L'art de l'intrusion*, 47 bis, rue des Vinaigriers 75010 Paris: 2005

[8] Rebé Hanouz, *Sécurité et qualité des systèmes d'information*, 75010 PARIS: 1993

[9] Boy, Daniel. (2007) La perception des risques, Les dossiers de La Recherche, N° 26, Février-Avril 2007

[10] CNRS, "Ergonomie -Définition - DSI du CNRS," in *dsi.cnrs.fr* [Juin 2009]

[11] Silicon, "'Stupid' administrators left Davos open to hack attack" in *Silicon.com*, http://www.silicon.com/management/cio-insights/2001/03/05/stupid-administrators-left-davos-open-to-hack-attack-11023056/ [Décembre 2009]

[12] TECFA, "Bio-Tremplin", in tecfa-bio-news.blogspot.com, http://tecfa-bio-news.blogspot.com/2009/11/h1n1-pourquoi-une-telle-inquietude-du.html [Décembre 2009]

[13] Wikipedia, "Ergonomie - Wikipedia" in *Wikipedia.org*, http://fr.wikipedia.org/wiki/Ergonomie [Juin 2009]

[14] Wikipedia, "Intégrité (cryptographie) - Wikipedia" in *Wikipedia.org*, http://fr.wikipedia.org/wiki/Int%C3%A9grit%C3%A9_(cryptographie) [Juin 2009]

[15] Wikipedia, "Système d'information - Wikipedia" in *Wikipedia.org*, http://fr.wikipedia.org/wiki/Syst%C3%A8me_d%27information [Juin 2009]

Wikipedia, "Jakob Nielsen (usability consultant) - Wikipedia" in Wikipedia.org, http://en.wikipedia.org/wiki/Jakob_Nielsen_%28usability_consultant%29 [Octobre 2009]

[16] BEA, "F-GGED" in *Bea-fr.org*, http://www.bea-fr.org/docspa/1992/f-ed920120/htm/f-ed920120.html [Décembre 2009]

[17] « *Sécurité* » in Le petit Larousse illustré, Larousse, 2004, p.928