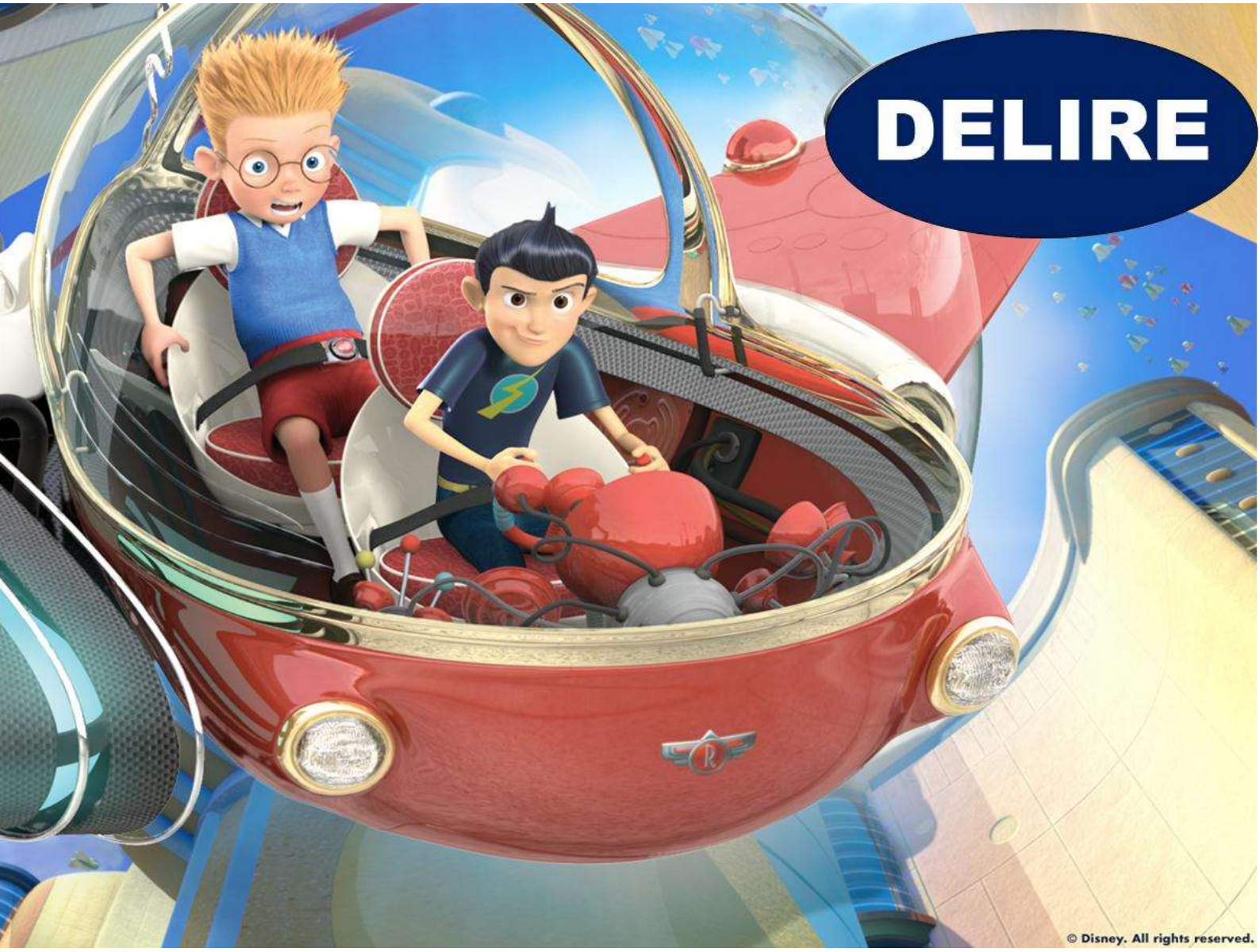


Le projet **DELIRE**

Développement par **E**quipe de **L**ivrables **I**nformatiques et **R**éalisation **E**ncadrée

NS6 – Rôle du Responsable Qualité

La crise du logiciel

La « crise du logiciel » s'apparente aujourd'hui à une maladie chronique de l'industrie du logiciel, dont les symptômes sont les suivants:

- Les délais de livraison des logiciels sont rarement tenus, le dépassement de délai et de coût moyen est compris entre 50 et 70%.
- La qualité du logiciel corresponds rarement aux attente des acheteurs, le logiciel ne corresponds pas aux besoins, il consomme plus de moyens informatiques que prévu, et tombe en panne.
- Les modifications après coup des logiciels coûtent cher, et sont à l'origine de nouveaux défauts. Les adaptations sont bien souvent une nécessité du fait de l'évolution des produits et des attentes des utilisateurs.
- Il est rarement possible de réutiliser un logiciel existant pour en faire un nouveau produit de remplacement, l'amortissement du coût de développement initial est ainsi rendu impossible.

Le coût du logiciel est le coût prépondérant d'un système informatique. et les coûts dus à la correction des défauts font exploser la note.

Les raisons de la non-qualité des logiciels sont les suivantes :

- Un logiciel étant un bien immatériel, les seules représentations observables du logiciel sont le code source, l'interface utilisateur et la documentation (spécification). La quantité de code source (nombre de lignes) est rarement connu, ce qui entraîne souvent une sous-estimation de la complexité du logiciel.
- Pour chaque module d'un logiciel il existe de nombreuses conditions d'utilisation. La combinaison des différentes conditions d'utilisation des différents modules d'un logiciel amène à une explosion combinatoire, et lors de sa construction un logiciel n'est jamais contrôlé dans la totalité des conditions d'utilisation qu'il rencontrera durant son cycle de vie.

Il n'y a pas de séparation claire entre un défaut mineur et majeur, et modification mineure ou majeure. Et l'effort de détérioration d'un logiciel n'est pas proportionnel à l'effort de construction.

- Un défaut mineur peut entrainer un incident majeur, et nécessiter une correction mineure : dans l'incident du vol 501 d'Ariane 5, une correction mineure a suffi à éviter la destruction de la fusée.
- De même une modification mineure d'un logiciel peut le mettre hors d'usage; un phénomène largement exploité par les virus informatiques.

Pour être considéré comme produit de qualité par l'usager, un logiciel doit répondre aux besoins exprimés explicitement par l'usager aussi bien qu'aux besoins implicites (non exprimés). Or les vœux implicites évoluent avec le marché, et il arrive bien souvent que ces vœux implicites des usagers ne soient pas connus des ingénieurs logiciels.

Amélioration de la qualité

En génie logiciel, l'abstraction, la dissimulation, la modularité, la structuration, l'uniformité, la complétude et la confirmabilité sont des principes destinés à améliorer la qualité du logiciel.

- La modularité, c'est-à-dire la qualité d'un logiciel d'être découpé en de nombreux modules, permet l'abstraction et la dissimulation. Associée avec un couplage faible, elle vise à augmenter la maintenabilité du logiciel en diminuant le nombre de modules touchés par des éventuelles modifications, ainsi que la fiabilité en diminuant les dépendances entre les modules.
- En jargon de programmation, un plat de spaghetti est un terme péjoratif pour désigner un logiciel au couplage trop fort et au code source difficile à lire, dans lequel toute modification même mineure demande un intense travail de programmation.
- L'abstraction vise à diminuer la complexité globale du logiciel en diminuant le nombre de modules et en assurant l'essentiel. Elle peut également apporter une uniformité du logiciel qui augmente son utilisabilité en facilitant son apprentissage et son utilisation.
- La dissimulation vise à séparer complètement les détails techniques du logiciel de ses fonctionnalités selon le principe de la boîte noire, en vue d'améliorer sa maintenabilité, sa portabilité et son interopérabilité.
- La structuration des instructions et des données rends clairement visible dans le code source les grandes lignes de l'organisation des instructions et des informations manipulées, ce qui améliore sa maintenabilité et laisse moins de place aux bugs.

De nombreux langages de programmation soutiennent, voire même imposent un code source fait de structuration, de modularité et de dissimulation. C'est le cas des langages de programmation structurée et de programmation orientée objet.

Vous l'aurez compris, la qualité du logiciel passe préalablement par une architecture Logique et des STDs (Spécifications techniques Détaillées) robustes, ce qui représente un investissement conséquent à court terme mais très payant à long terme.

Qualité Produit et Qualité Process

Dans le Cahier des Charges, le MOA a tenté de formaliser les besoins du client en termes fonctionnels. Mais dans la réalité il y a un tas de besoins implicites et donc non formalisés.

Le client s'attend naturellement, sans avoir ressenti le besoin de l'exprimer, à avoir un produit

- Convivial
- Fiable
- Performant

Jusque-là tout va bien. Là où cela se complique, c'est de savoir ce que signifie Convivial, Fiable ou performant, et surtout si client et fournisseur y mette le même contenu.

  - Pour Usain Bolt, performant au 100 mètres veut dire moins de 10 secondes
  - Pour ma mère, 96 ans, 2 minutes est une belle performance.

Sinon, bonjour la catastrophe quand le produit sera remis au client. Celui-ci sera insatisfait, et le fournisseur devra soit réinvestir sur le développement, soit baisser le prix de vente. Dans les deux cas :

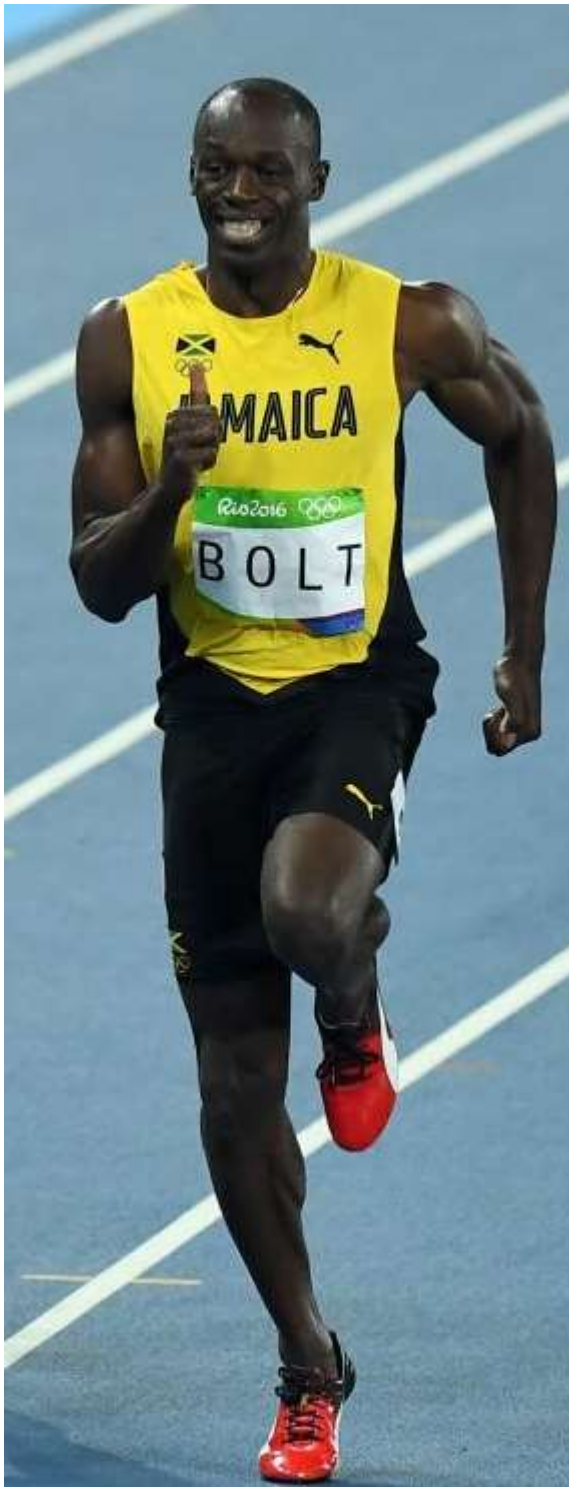
- une perte pour lui à court terme
- et surtout une perte à long terme car le client ne voudra sans doute plus faire appel au même fournisseur.

La méthodologie classiquement employée dans l'industrie est de définir des objectifs chiffrés et mesurables

On se met d'accord, lors de la négociation entre le client et le fournisseur, sur un certain nombre de scénarios, sur la façon de mesurer la Convivialité, la Fiabilité ou la Performance, et sur les résultats attendus.

Et dans la réalité, cela se révèle beaucoup plus difficile que prévu

- Dire : mon produit est fiable, c'est facile
- Dire mon produit est fiable de tant, se révèle plus compliqué.



Mais le rôle d'un projet n'est pas uniquement de réaliser un produit qui réponde aux besoins du client. Il est également de faire progresser le savoir-faire de l'entreprise

- Un projet c'est toujours de l'innovation et donc des mécanismes de structuration méthodique et progressive d'une réalité à venir
- Mais une entreprise doit favoriser l'exécution d'opérations : des taches maîtrisées, dont le workflow, le coût et la durée sont connus.
- Ceci permettra à l'entreprise de se lancer par la suite dans des projets encore plus ambitieux,
- Soit pour conquérir de nouveaux marchés

- Soit pour ne pas être éliminée par des concurrents sur son marché actuel.

Le savoir-faire de l'entreprise est donc formalisé dans des « Best practices », qui permettent aux projets futurs de ne pas refaire les erreurs des projets antérieurs, mais de faire tout de suite les bons choix.
C'est pourquoi, il est nécessaire en fin de projet de formaliser les bons résultats obtenus.

- Comme le projet DELIRE n'est pas réalisé dans le cadre d'une entreprise, ce point ne sera pas traité.
- Néanmoins, formaliser pour chacun ce qu'il a appris durant le projet lui sera utile dans le cadre de sa carrière : ce sera le rôle des Post Mortem en fin de projet.

Par contre, un objectif clé de DELIRE est de favoriser la coopération. Une part importante de cette mission est du ressort du MOE et je l'ai traité dans le document NS5 : Rôle du MOE. Mais une part importante revient au Responsable Qualité.
La coopération passe par des facilités d'échange. Et ces facilités d'échange passent par des standards communs.
J'en distingue 6 :

- Le standard de Livrable
- Le standard de Présentation
- Le standard de Coding
- Le standard de STD
- Le standard de Fiche de Livraison
- Le standard de Fiche de Test

Qualité Produit
On définit traditionnellement la qualité d'un produit par sa capacité à satisfaire les besoins du client

- Les besoins fonctionnels ; c'est l'objectif des SFG, et ce rôle est dévolu au MOE
- Les besoins non fonctionnels : c'est l'objectif de la Qualité Produit, et ce rôle est dévolu au Responsable Qualité

Le modèle souvent utilisé comme référence dans l'industrie du logiciel est celui d'IBM : le CUPRIMDSO

- Capacity - Fonctionnalités
- Usability - Convivialité
- Performance - Performance, Capacité et Scalabilité
- Reliabilitry - Fiabilité
- Installability - Installabilité (Capacité à être déployé).
- Maintenability – Maintenabilité
- Documentation/Information
- Service
- Overall – Qualité Globale

Capability est sous le contrôle du MOE, au travers des SFG.

Installability, Maintenability et Documentation seront adressées lors de la phase de Convergence, par

- Rédaction d'une documentation Utilisateur
- Rédaction d'une documentation de Maintenance
- Rédaction d'une documentation d'Installation

Service mesure la valeur ajoutée pour le client, et suppose la disponibilité de l'ensemble des versions du produit
Overall mesure la satisfaction des utilisateurs, et suppose que le produit ait été déployé en production

Nous ne nous intéresserons donc ici qu'à Usability, Reliability et Performance

Les deux questions à se poser

- Comment définir des objectifs chiffrés et mesurables
- Comment être sûr de les atteindre et donc de ne pas avoir une très mauvaise surprise en fin de projet

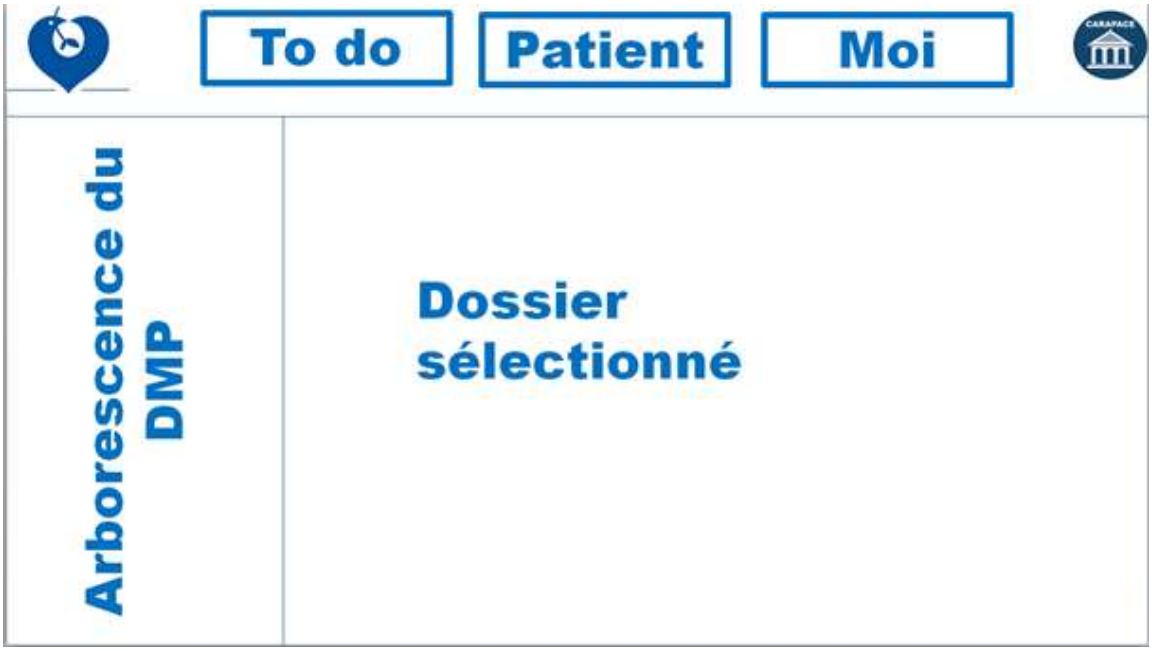
Définir des objectifs chiffrés et mesurables
La Convivialité
C'est-à-dire la facilité d'utilisation. OK, mais facile en quoi :

- Facile à prendre en main ?
- Facile à mémoriser ?
- Minimisant l'effort de l'utilisateur ?

Dans la réalité c'est tout cela

Facile à prendre en main signifie que la structure globale est stable dans son utilisation, c'est-à-dire qu'il se présente toujours de la même façon, et qu'il suffit de comprendre un écran pour avoir une bonne idée du fonctionnement du produit.
Exemple :

1. Quelle que soit la commande, l'écran se présente toujours de la façon suivante :



2. Le nom de la commande apparait toujours quand on passe la souris sur l'icône
3. On peut toujours faire UNDO, ou alors on doit confirmer l'opération

Facile à mémoriser veut dire que les mêmes causes produisent les mêmes effets, et que les mêmes effets sont produits par les mêmes causes. En d'autres termes, que le même type de commande se présentera toujours de la même façon dans toute l'utilisation du produit.

- Le nom de la personne connectée et du patient suivi est affiché en permanence
- La sélection d'un DMP est identique quelle que soit la commande en cours

Minimiser l'effort de l'utilisateur veut dire que le nombre d'interaction pour une commande donnée est minimum, et que au sein de ces commandes, la sélection (par exemple d'une liste déroulante) est privilégiée par rapport à une saisie clavier
Exemple :

- Une fois saisi le code postae, on propose de choisir la ville dans une liste.
- Ceci permet de ne pas avoir d'incohérence entre code postale et ville, et de vérifier que le code postal entré est pertinent.

Concernant les commandes à optimiser, on se focalisera en priorité sur les commandes les plus utilisées : il vaut mieux réduire d'une interaction une commande utilisée dans 95% du temps, que de réduire de 3 interactions une commande utilisée un fois par an. Le mieux étant bien entendu de faire les deux.
On définira les objectifs comme ceci :

- La sélection d'un DMP nécessite 4 interactions dont 2 saisies clavier maximum

- La saisie des données sociodémographiques d'un patient est de 27 interactions au maximum, dont 9 saisies clavier.

La Fiabilité

"*La simplicité est un prérequis de la fiabilité*." Edsger W. Dijkstra

Classiquement en informatique on définit comme critère de Fiabilité le MTBF : Main Time Between Failure
Un produit est considéré comme très fiable s'il atteint un MTBF de 200 heures, c'est à dire un mois d'utilisation en conditions normales (c'est-à-dire conditions industrielles mais non pas Stress Tests)

Problème : pour valider un MTBF de 200 heures, il faut faire à minima 1000 heures de tests : inenvisageable dans le cadre de DELIRE.
A titre d'information, nous avons à Dassault Systèmes près de 1500 engineers qui testent en permanence nos produits en exécutant des scénarios fonctionnels : cela représente entre 50,000 et 60,000 heures de tests pas semaine et permet, en divisant le nombre d'heures par le nombre de cartons obtenus dans la semaine d'avoir une bonne estimation du MTBF : croyez-moi, 200 heures ce n'est pas du gâteau.

OK Antoine, et pour DELIRE on fait quoi ?
On ne cherche pas à avoir nécessairement un produit béton en utilisation extrême. Mais néanmoins l'utilisateur peut se tromper : donc avoir un produit qui ne oart pas en sucette en cas d'utilisation erronée, mais qui donne un message correcte de l'erreur détectée ne peut nuire
Nous sommes focalisés sur une utilisation industrielle de votre produit. Et à ce jour, ce qui fait foi en termes d'utilisation, ce sont les scénarios fonctionnels.
Et dans tous ceux-ci, il y a un scénario qui doit vraiment être béton : c'est celui du scénario de la démonstration finale.

Bref, voici le type d'objectifs de fiabilité que je peux vous proposer

- Avoir corrigé tous les problèmes trouvés en Stress Tests
- Avoir exécuté chacun des Function Tests au moins une fois sans erreur (ce qui présuppose d'avoir corrigé tous les problèmes trouvés en Function Tests)
- Avoir exécuté le scénario de la démonstration 3 fois de suite sans erreur (ce qui présuppose d'avoir corrigé tous les problèmes trouvés lors de l'exécution du scénario de la démonstration finale)

Pour cette partie Fiabilité du document Plan Qualité, rapprochez-vous du Responsable des Tests et du Designer

Mais en quoi ces objectifs sont chiffrés et mesurable ?

- Ils sont mesurables puisqu'on vient d'exprimer comment les obtenir : en exécutant les scénarios fonctionnel entre 1 et 3 fois
- Ils sont chiffrés (dans un espace booléen) : le nombre de problème doit être égal à 0

Performance

Performance est un terme générique qui englobe la notion de PCS

- P pour Performance : la réponse dans le cas d'une utilisation où tout est en local et seule l'application tourne sur la machine doit être inférieure à un temps déterminé
- C pour Capacity : la réponse du système ne doit être que faiblement dégradé quand on travaille sur de gros volumes de données. Cette dimension n'a que peu de signification dans le projet DELIRE : peu de chance que le médecin traite 200 DMP simultanément.
- S pour Scalability : le système ne doit pas s'effondrer lorsque le nombre d'utilisateurs simultanés grimpe. Cette problématique doit être traité par l'Architecture Logique et ne pourra être valablement testé que dans la phase de déploiement de la version finale, en architecture définitive.

Bref, dans la première (et probablement unique) version de DELIRE, ce sur quoi il faut se polariser est la notion de performance

Dans l'imaginaire de l'utilisateur, il y a des interactions courtes, et d'autres sur lesquelles il est prêt à accepter un certain laps de temps.
Exemple, dans la saisie des données sociodémographiques

- Le passage de l'adresse au numéro de téléphone est vu comme une interaction courte
- La sauvegarde de toutes ces données est vue comme une interaction lourde : on sollicite la base de données

On peut se poser comme base de travail :

- Moins d'un 1/10 de seconde pour in interaction courte
- Moins d'une ½ seconde pour une interaction supposée lourde.

On peut certes définir des objectifs pour chaque interaction de chaque commande de chaque scénario fonctionnel.
Mais le mieux, on va le voir dans le paragraphe suivant, est de trouver les interactions les plus représentative, qui vont permettre de fixer par la suite pour chaque composant logiciel un temps de réponse.

Atteindre les objectifs Qualité Produit définis en début de projet

Ce qu'il faut comprendre est que les objectifs qualité ne se découvrent pas à la fin du projet, où on mesure et… ça passe ou ça casse.
Pour que cela ne casse pas, il faut anticiper.

- Concernant la convivialité, on peut tester que le nombre d'interactions et de saisies clavier est respecté dès que l'IHM et l'Architecture Fonctionnelles, sans qu'une ligne de code ne soit écrite
- Concernant les performances on peut definer, en s'appuyant sur l'Architecture Logique, le temps que doit consommer chaque composant au maximum pour être sûr que tous les objectifs seront atteints après intégration de tous les composants : il sera dès lors possible de valider l'atteinte par chaque composant de son objectif en phase de Réalisation grâce au Unit tests, et de savoir sur quels composants faire porter les investissements en priorité

L'objectif de fiabilité ne pourra certes être validé qu'en phase de Convergence, mais le fait que tous les Unit Tests fonctionnent est de bonne augure pour la phase de Convergence.

Qualité Process
Votre objectif n'est pas de définir le process que va suivre votre équipe pendant le projet DELIRE : c'est une méthodologie basée sur le cycle en V et qui s'appuie sur 4 phases : Spécification – Structuration – Réalisation – Convergence. Je le sais, c'est moi qui l'ai imposée.

Par contre, un process est généralement basé sur des procédures. Votre objectif est de définir les procédures qui vont être suivies par l'équipe pour pouvoir réaliser le process défini par mes soins.
Une procédure est en général une bonne pratique qui a été capitalisée par l'entreprise au cours de son histoire. Pour votre entreprise, cela risque d'être compliqué.

On va donc simplifier la donne : l'objectif est de définir des procédures qui vont permettre aux membres de votre équipe, d'améliorer leur productivité et la qualité de ce qu'ils produisent (en d'autres termes, le contenu des livrables) en réduisant l'effort à fournir pour réaliser des tâches à faible valeur ajoutée.
Bref, vous allez définir des standards que les membres de l'équipe n'auront plus qu'à utiliser pour optimiser l'efficience des différents activités de chacune des phases du projet.

L'objectif de la suite de cette note est de comprendre les objectifs dévolus à chacun de ces standards

Le standard de Livrable
L'ensemble de vos livrables va constituer votre dossier projet
A la fin décembre, vous remettrez vos livrables de Spécification et de Structuration pour remporter l'appel d'offres
Fin mars - début avril vous remettrez votre dossier complet

- Les livrables de Spécification et de Structuration repris et amendés en fonction du déroulement du projet en phases de Réalisation et de Convergence
- Les livrables de la phase de Réalisation
- Les livrables de la phase de Convergence

J'aurai l'occasion de revenir sur les livrables de la fin de projet

Pour l'instant, l'objectif est que vos livrables de Spécification et de Structuration apparaissent comme un dossier complet, logique et cohérent :

- Complet :
  - A vous de valider qu'une (et une seule) personne est responsable de chaque livrable
  - Sauf pour les STDs où chaque développeur est responsable de la STD de son composant.
- Logique
  - Comme vous remettrez vos documents au travers d'une clé USB, le mieux est de leur donner une nomenclature qui permette de les classifier naturellement.
  - Exemple pour le projet Archi-Med
    - Archi-Med - Part 1 Specification - Document 1 Cahier des Charges

        - Archi-Med - Part 2 Structuration - Document 3 Plan de Gestion des Risques
- Cohérent
    - Il faut donner l'impression que le dossier est le résultat du travail d'un groupe qui travaille en synergie, et non pas un tas de documents disparates

Chacun doit respecter le standard de livrable

- Polices de caractères,
- Header et Footer
- Mise en page
- Indentation
- Page de garde
    - Nom du livrable
    - Nom de l'auteur, des relecteurs…
    - Numéro éventuel de version…

Ce standard doit mettre en valeur votre entreprise et votre projet. Par exemple, le logo de l'entreprise sur la page de garde, et le logo du projet sur chaque page du livrable.

Bref

- Il faut que vos livrables soient si beaux que vous voudrez les garder dans votre bibliothèque pour les montrer avec orgueil à vos arrières petits-enfants.
- Il faut que vos livrables soient si clairs, si faciles à lire et si logiques que le lecteur (en l'occurrence moi-même) ne puisse plus s'arrêter une fois la lecture entamée et repose à 5 heures du matin le dernier livrable en disant : « il a tout compris »

Le standard de Présentation
Votre présentation sera le faire valoir de votre travail. Elle se doit donc d'être réussie.

Vous avez eu ou aurez des recommandations pour bien construire une présentation au cours de vos cours de communication

Qu'il me soit donc juste permis de faire 3 recommandations

- Adoptez un standard pour chacun des slides que l'un des membres de l'équipe construit : cela permet ensuite de rassembler ces différents slides et d'avoir tout de suite une présentation avec de l'allure
- Faite vos slides d'aujourd'hui en pensant qu'il vous seront utiles pour la présentation intermédiaire de fin décembre et la présentation finale de fin mars / début avril.
- Ne mettez pas en avant l'UPEM ou le projet DELIRE, mais votre entreprise Médi-Hommes ou votre projet Archi-Mèd.

Le standard de STD
La pulsion naturelle d'un développeur est de coder. Cela lui démange les doigts.
Dès lors

- Pourquoi formaliser sur papier ce que j'ai en tête : c'est un peu compliqué à expliquer, et je perds du temps
- Le mieux est de le traduire tout de suite en lignes de code.

Oui certes mais…

Nicolas Boileau disait que
« *Ce qui se conçoit bien s'énonce clairement,*
Et les mots pour le dire vous viennent aisément »
Si vous avez du mal à l'exprimer, c'est ce n'est pas encore bien conçu : il est donc trop tôt pour coder

Vous n'allez pas perdre de temps : lorsque vous démarrerez la phase de Réalisation, le guide de méthodologie que définiront les STDs vous permettront d'atteindre une productivité qui vous surprendra.

Le jour où vous devrez transmettre votre code, à des fins de maintenance, d'évolution, ou parce que vous changez de poste dans l'entreprise voire quittez l'entreprise, les STDs seront une bonne documentation de la structure de votre composant

Si vous-même devez replonger dans votre code dans quelques années voire simplement quelques mois, vous découvrirez que la logique qui a prévalu dans la réalisation du composant n'est plus si apparente que vous le pensiez, et vous serez heureux de la retrouver dans les STDs

Enfin, bien formaliser vos STDs vous permettra potentiellement de confier la réalisation à un autre membre de l'équipe, voire plus tard en entreprise d'externaliser le codage de vos applications.

Ne l'oubliez pas, le codage est une phase d'expertise, mais non pas de créativité.

OK Antoine, mais cela milite simplement pour faire les STDs, non pas pour définir un template pour les STDs.
Oui, mais le problème est que le nombre de points à prendre en compte durant l'écriture des STDs est assez important. Un document qui va permettre au développeur de ne pas oublier un point crucial ne peut donc nuire :

- La définition des Unit Tests, boite noire et boite blanche, est-elle effectuée ?
- Toutes les interfaces allouées au composant ont-elles été couvertes
- Les services utilisés pour décrire les méthodes sont-ils tous internes ou exposés par les composants prerequisites
- Les cas d'erreurs sont-ils tous documentés
- Les entrées/sorties de méthodes internes sont-elles toutes documentées
- Les algorithmes à mettre en œuvre sont-ils tous documentés ?
- Les structures de données sont-elles toutes documentées ?

Plus tout ce qui vous semblera important.

Le standard de Coding

Beaucoup de développeurs pensent qu'ils codent très bien, et même qu'ils sont les meilleurs codeurs du monde.
Si on leur parle de norme de programmation, on a pour réponse :

- Cette façon d'uniformiser les individus me déplait, je ne suis pas un numéro !
- Cette norme imposée par ma hiérarchie est une entrave à mon imagination et à mes prises d'initiatives.

Ils sont d'autre part une peu possessifs, ne souhaitent pas que quelqu'un puissent lire leur code en dehors d'eux-mêmes, truffent leur code d'astuces difficiles à comprendre et limitent au maximum les commentaires.

Résultat :

- Eux seuls peuvent maintenir leur code : c'est la panique si un problème survient durant leurs congés
- Et ils sont coincés lorsqu'ils veulent évoluer au sein de l'entreprise.

Elle est loin l'époque des génies dans leurs garages où l'informatique s'apparentait plus à l'artisanat.
Aujourd'hui la création de logiciel doit tendre vers un processus industriel bien rodé où sont normalisés : méthodologie de projet ; méthodologie de spécification, codage, méthodologies de test.

On oublie trop souvent que :

- 80% du temps passé dans le cycle de vie d'un logiciel est passé en maintenance de celui-ci
- Il est très rare qu'un code soit maintenu par son auteur original

De bonnes conventions adoptées par tous permettent à un développeur de s'y retrouver plus facilement dans un code inconnu, pour simplifier la maintenance et faciliter une compréhension universelle.

Un bon code c'est un code où :

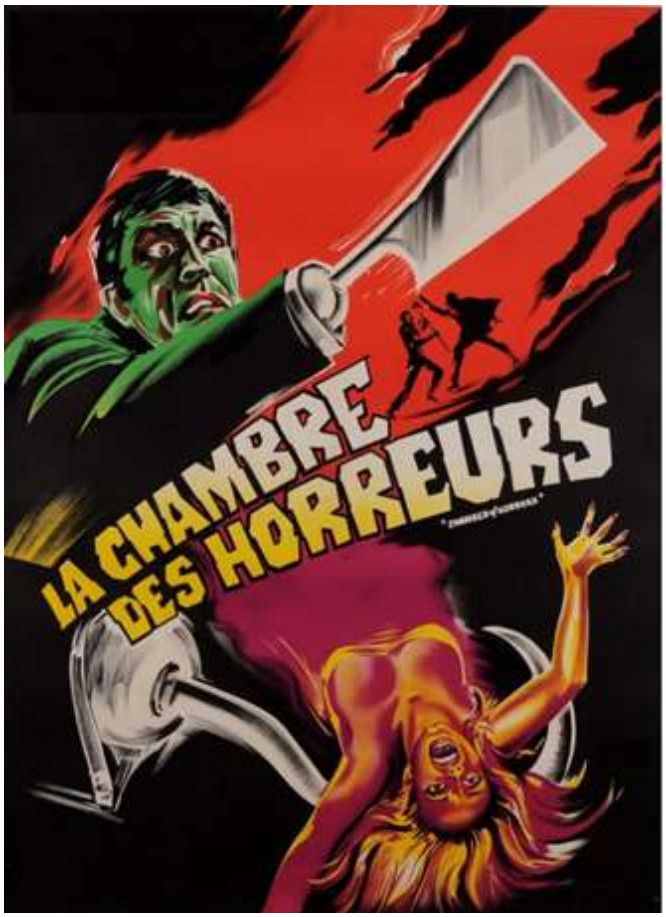
- Les entrées et les sorties sont clairement documentées
- Les cas d'erreurs sont clairement documentés
- La structure du code est clairement documentées
- La façon de coder (typologie des variables, nomenclature des points d'entrée…) répond à un standard

Vous allez définir

- Des normes générales d'organisation du code
  - Taille des fichiers
  - Taille des méthodes
  - Entête documentaire des classes et méthodes
  - Pourcentage de commentaires dans le code
- Des normes générales de nommage
  - Packages
  - Classes
  - Variables
  - Constantes
  - Artefact (jar)
- Des conventions relatives à la lisibilité
  - Indentation du code
  - Taille des lignes
  - Encodage des fichiers
  - Retour à la ligne Windows/Unix

Plus tout ce qui vous semblera bon de préciser

Autre point important, il existe des normes de programmation avec des outils en freeware pour vérifier le respect de ladite norme.



Bref un bon code c'est un code où chaque développeur de l'équipe n'a pas l'impression de pénétrer dans la chambre des horreurs mais au contraire se sent chez lui,

Le (ou plutôt les) standard(s) de Fiche de Livraison
L'objectif du projet DELIRE est de vous permettre de vous constituer une expérience dans certains domaines du développement de produit.
Si je vous pose des questions comme :

- Taille en ligne de codes d'un panel en HTML ?
- Nombre de lignes de code pissées par jour
- Temps de préparation d'un Unit Test
- Temps de détection et de réparation d'un bug
- Nombre d'erreur par millier de de lignes de code ?

Vous serez sans doute bien en peine de me répondre

Et pourtant, des standards existent pour y répondre.

- 50 erreurs par kloc (Kilo Lines of Code)
- Coût de préparation et de detection/correction pour un logiciel supérieur à 50 klocs.

Etape opération	Préparation coûts	Dét./Répar. coûts
DR2	05H par Kloc	03H par bogue
I0	11H par Kloc	05H par bogue
I1	15H par Kloc	05H par bogue
I2	18H par Kloc	06H par bogue
U.T.	25H par Kloc	10H par bogue
Func.Test	40H par Kloc	24H par bogue
Comp.Test	60H par Kloc	24H par bogue
Syst.Test	20H par Kloc	35H par bogue

Vous allez être amenés à développer quelques milliers de lignes de code pour votre produit, à corriger quelques dizaines de bugs, à passer quelques dizaines d'heures pour ce faire.
Cela vaut le coup d'en profiter pour capturer des données et les capitaliser.

Vous avez deux raisons de livrer du code

- Vous livrez votre composant parce qu'il est arrivé à un bon niveau de maturité, et qu'on peut désormais envisager de l'intégrer.
- Vous relivrez votre composant suite à une ou des corrections d'erreurs

Dans le premier cas, cela vaut le coup d'obtenir des réponses aux questions suivantes

- Nombre de lignes de code de l'application
- Temps passé à coder l'application
- Nombre de ligne de code des Unit Tests
- Temps passé à coder les Unit Tests
- Typologie du composant (écran, commande, objet…)

- Nombre de classes gérées (nombre d'écrans, nombre de commandes, nomnbre d'objets …)
- Nombre d'erreurs trouvées en Unit Tests
- Temps passé en correction des erreurs trouvées en Unit Tests
- Toutes les erreurs trouvées en Unit Tests sont-elles corrigées
- Le standard de coding est-il respecté ?
- Les temps de réponse du composant ont-ils été mesurés ? Sont-ils conformes aux objectifs définis dans le PQ ?

Plus tous les points qui peuvent vous sembler pertinents

Dans le second cas, cela vaut le coup d'obtenir des réponses aux questions suivantes

- Nombre d'erreurs corrigées
- Temps passé en correction des erreurs trouvées
- Nombre de lignes de code modifiées
- Y a t'il eu de nouveaux Unit Tests développés pour prévenir le retour d'une des erreurs corrigées
- Si oui
    - Nombre de ligne de code des Unit Tests
    - Temps passé à coder les Unit Tests
- Les Unit Tests ont-ils tous été rejoués ? Sont-ils tous OK ?
- Toutes les erreurs trouvées en Unit Tests sont-elles corrigées
- Le standard de coding est-il respecté ?
- Les temps de réponse du composant ont-ils été mesurés ? Sont-ils conformes aux objectifs définis dans le PQ ?

Plus tous les points qui peuvent vous sembler pertinents

Vous pouvez certes espérer que vos collègues vont consigner naturellement ces informations

Vous pouvez surtout préparer des fiches de livraisons standard qu'ils rempliront à chaque livraison ou relivraison, et être ainsi sûr de capitaliser une expérience précieuse.

Le standard de Fiche de Test

On est dans une démarche similaire à la fiche de livraison
Vous allez exécuter des scénarios fonctionnels et trouver des erreurs dans le produit : je rappelle que, selon ma proposition, vous devez répéter ces scénarios jusqu'à ce que chacun des scénarios fonctionnels ait été réalisé sans erreur détectée, et que le scénario de la démonstration ait été réalisé 3 fois de suite sans erreur.

Il est donc intéressant de capturer un certain nombre d'information dans l'exécution de ces tests fonctionnels, selon un protocole conceptuellement assez similaire à la fiche de livraison.

En conclusion, vos objectifs sont

- Définir un but et un chemin pour atteindre des critères de Qualité Produit conforme à l'attente du client
- Investir sur l'ergonomie et l'efficience du process de développement de l'équipe