



Le projet **DELIRE**
Développement par Equipe
de Livrables Informatiques
et Réalisation Encadrée

PStr2 – PQ (Plan Qualité)



Comprendre la démarche qualité

La qualité optimale se situe au point de rencontre des besoins de l'ensemble des parties prenantes: clients, actionnaires, salariés et la compagnie en général.

1. Elle a un coût, censé réduire le coût important de la **non-qualité**
2. Le niveau de qualité optimal ne doit pas produire de coût inadéquat: des fonctionnalités non nécessaires, la **sur-qualité**

Une entreprise est performante lorsque le triptyque Délai - Coût - Qualité, c'est-à-dire l'utilisation de ses ressources, est justifié et efficace. Ceci lui permet de bénéficier d'un ticket d'entrée élevé

L'industrie investit sur les Best Practices pour améliorer la Qualité, tout en réduisant le Délai et le Coût.

On imagine la non qualité comme le Big Problem (exemple du problème de câblage de l'A380). Mais dans la réalité, la non qualité est souvent le fruit d'une collection de problèmes rencontrés lors de la mise au point du produit. :

Or un problème :

1. Détecté en phase d'avant-projet coute 1
2. Détecté en phase de spécification coute 10
3. Détecté en phase de définition coute 100
4. Détecté en phase de fabrication coute 1000
5. Et détecté en cours d'utilisation coute 10000

Certes, la qualité coute cher

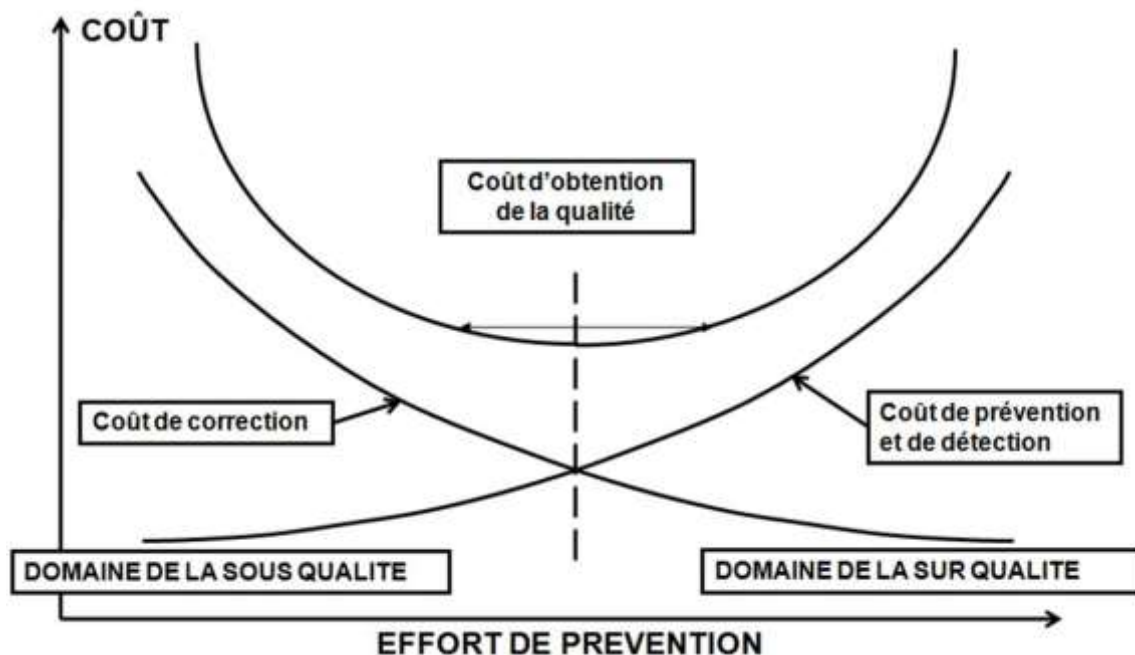
1. Définition des tests
2. Réalisation des moyens de tests
3. Exécution des tests
4. Définition des correctifs à apporter et mise en place des plans de convergence

Mais la non qualité coute encore plus cher

1. En août 2008, GM doit rappeler 1 million de véhicules : défaut de fabrication au niveau du système électrique de l'essuie-glace, pouvant présenter un risque d'incendie.
2. En mars 2009, GM doit rappeler 300 000 véhicules Buick : défaut dans le câblage des boîtes de vitesses, risquant de ne pas bloquer le véhicule en position Parking.
3. En avril 2009, GM doit rappeler 1,5 million de voitures Chevrolet, Buick, Oldsmobile et Pontiac : fuites de carburant sur les conduits d'échappement pouvant présenter un risque d'incendie en freinage.

Le tout en pleine débâcle financière (Ne l'oubliez pas, c'est toujours au même moment que les ennuis s'accumulent)

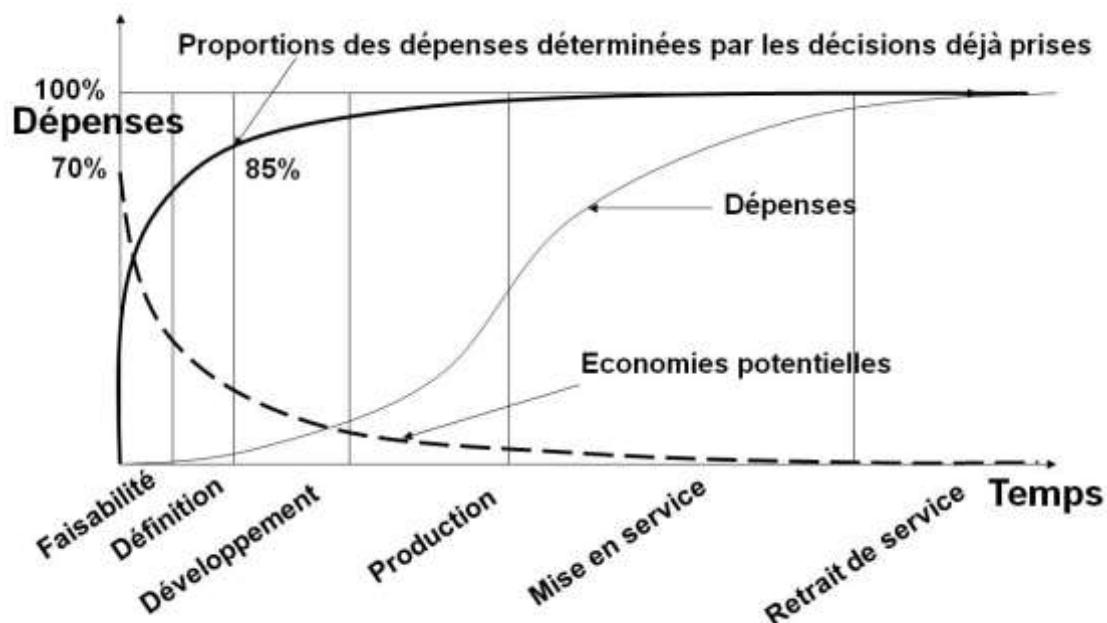




Il est très courant de décrire la qualité comme la satisfaction du client. La qualité semble être floue et relative, car fonction des exigences et de l'appréciation des clients.

Dans la réalité de l'industrie, la démarche de gestion de la qualité va avoir pour objectif de transformer la satisfaction du client en une cible dont les critères sont précisément fixés. En d'autres termes, les objectifs qualité sont toujours des objectifs chiffrés. Une fonctionnalité est chiffrée et mesurable, un objectif qualité est chiffré et mesurable, c'est cette caractéristique qui va permettre le suivi Qualité

La définition et le suivi de la qualité doivent être faits très tôt



Exemple de l'installation du chauffage de votre maison:
Dans votre cahier des charges: une douce chaleur agréable
L'architecte a deux solutions:

1. Une pompe à chaleur:
 - a. c'est cher à l'installation
 - b. mais relativement économique à l'usage
2. Des radiateurs électriques:
 - a. c'est peu cher à l'installation
 - b. mais relativement onéreux à l'usage... sauf si l'appartement est remarquablement isolé

Et l'isolation, c'est relativement facile pour une nouvelle maison, sauf si vous souhaitez avoir beaucoup de baies vitrées....

Bref, les fonctionnalités offertes dépendent des choix qui sont faits, lesquels nécessitent de connaître globalement l'architecture retenue pour le projet

Le modèle CUPRIMD

Le modèle CUPRIMD a été proposé très tôt (dans les années 70) par IBM comme modèle universel de qualité en logiciel. On peut intelligemment s'en inspirer dans DELIRE.

CUPRIMD signifie :

1. C : Capability
2. U : Usability
3. P : Performance
4. R : Reliability
5. I : Installability
6. M : Maintainability
7. D : Documentation

C : Capability

La Capability est le fait de découvrir qu'il manque une (ou des) fonctionnalité. Attention, ne raisonnons pas d'un point de vue fournisseur. Ne vous réfugiez pas derrière le fait que la liste des fonctions proposées est conforme à ce qui a été défini en SFG. Ce qui est important est le fait que le produit apporte une véritable valeur ajoutée aux utilisateurs.

La bonne solution est donc de définir des scénarios fonctionnels (ceux que j'avais définis sous le vocable scénarios d'utilisation dans le livre 5 – Les SFG (Spécifications Fonctionnelles Générales).

Ceci nous donne une vision un peu plus claire de la définition d'un scénario fonctionnel : c'est la définition d'un ensemble de commandes dont l'exécution séquentielle garantit à l'utilisateur un ROI (Return On Investment)

En tant que responsable Qualité, votre objectif est de vous assurer que les scénarios fonctionnels vont couvrir la définition complète du produit.

En phase de spécification et de structuration, vous devez vous assurer que les scénarios fonctionnels sont validés vis-à-vis :

1. Des SFG
2. De la maquette d'interface utilisateur



En phase de réalisation, vous serez probablement amené à faire des concessions sur le niveau de vos ambitions. Il est donc probable que l'ensemble des scénarios fonctionnels ne pourront pas être couverts. Il est donc nécessaire que vos scénarios fonctionnels soient structurés en fonction des différentes versions de votre produit. J'attire votre attention sur le fait que, dans bien des projets, les contenus de Basic, Standard et Advanced sont fait un peu à la va vite pendant la phase de spécification. Et par la suite, les groupes vont picorer certaines fonction dans Basic, certaines autres dans Standard, et certaines enfin dans Advanced. Il y a mille raisons pour cela : fonctionnalités qui se révèlent chères, fonctionnalité qui n'ont pas d'intérêt. Le problème est que on peut se retrouver ensuite avec aucun scénario fonctionnel qui apporte une véritable plus-value au client. L'identification, lors de l'élaboration des SFG, du contenu de chaque version est une démarche qui se révèle à terme très structurante.

U : Usability

Réussir un logiciel convivial est extrêmement difficile, et demande beaucoup de métier.

Pour une voiture, la convivialité est le fait d'avoir tous les instruments à portée de main, de pouvoir comprendre immédiatement le fonctionnement de la voiture, et que les opérations soit simples (ce qui n'est pas toujours le cas lorsqu'il faut changer une roue crevée).

Dans le cas de DELIRE, la convivialité peut se décliner en 3 critères :

1. Le respect de standard dans la présentation : icônes classiques, présentation identique durant toute l'exécution du produit
2. Facilité d'accès aux commandes : elles doivent toutes être accessible depuis le menu principal (bref pas de commande cachée au sein d'une autre commande)
3. Nombre d'interactions le plus faible possible, pour les commandes usuelles au minimum, et au sein de ces commandes un maximum de clicks et un minimum de saisie clavier.

Il existe d'autres critères pour la convivialité. En particulier la notion de fermeture. L'utilisateur a besoin de se rassurer par le fait qu'une étape est terminée. S'il se retrouve dans des commandes sans fin, l'utilisateur aura tendance à générer lui-même une pseudo fermeture. Nous n'aborderons pas ces aspects dans DELIRE.

P : Performance

Dans la réalité industrielle, la performance recouvre 3 notions

1. Performance : c'est le temps de réponse unitaire de la fonction en utilisation nominale (un seul utilisateur, pas d'autre process en cours, réseau non chargé...).
2. Capacity : c'est la quantité de données qu'il est possible de manager par utilisateur
3. Scalability : c'est le nombre d'utilisateurs que le système est capable de supporter simultanément.

C'est ce qu'on appelle le PCS.

Dans le projet DELIRE, nous ne nous intéresserons qu'à la dimension Performance, mais à titre personnel, vous pouvez vous poser la question de savoir si vous vous sentez confortables vis-à-vis de Capacity et de Scalability.



Dans un site web, la performance est le temps de réponse lors de l'exécution d'une commande.

En général on considère qu'un produit avec un temps de réponse supérieur à 1 seconde est un produit qui colle. Bien entendu, ce temps doit être mesuré avec tous les composants en interne, on ne peut prendre en compte la latence du réseau.

On se polarisera sur les commandes les plus utilisées. Ainsi :

1. La commande d'inscription d'un nouvel utilisateur sur le site n'est exécutée qu'une seule fois par utilisateur, il est donc acceptable qu'elle ait un temps de réponse un peu plus long
2. Par contre, une commande de consultation doit absolument être rapide.

Définir un temps de réponse pour une commande n'est pas une fin en soi. Il n'y aurait plus qu'à attendre la phase d'intégration du produit pour mesurer si effectivement le temps de réponse est conforme aux attentes. Et on serait bien ennuyé si tel n'était pas le cas.

Un fois connu le temps de réponse pour les commandes les plus utilisées, et en s'appuyant sur l'architecture logique du produit, on doit être capable de définir un temps maximum d'exécution au sein d'un module donné. Et ceci est testable durant les tests unitaires du composant. Si d'aventure le temps de réponse se révèle trop important, il est encore temps de retravailler l'algorithme pour rectifier le tir, ou de modifier en dernière extrémité l'architecture du produit.

R : Reliability

La fiabilité d'un système, c'est à dire le temps avant de rencontrer un carton, se mesure traditionnellement en MTBF (Mean Time Between Failures). Et les valeurs traditionnellement admises sont par exemple de l'ordre de 200 heures : en d'autres termes, un utilisateur doit pouvoir utiliser le logiciel pendant 1 mois sans être confronté à un bug.

Problème : on admet généralement que pour pouvoir démontrer un MTBF de 200 heures il faut globalement passer au moins 1000 heures de test. Il est clair que ceci n'est pas compatible avec le budget de DELIRE.

Dans la réalité, on se contentera de corriger tous les problèmes rencontrés durant les tests et la mise au point de la démonstration. Et on priera pour qu'il n'y ait pas de problème pendant la démonstration de la soutenance ni durant les tests basiques que les professeurs pourront exécuter par la suite.

Oui Antoine, mais que met-on comme objectif. Il est clair que vous ne pouvez pas mettre de valeur, puisque les objectifs de qualité sont des engagements vis-à-vis du client : il pourrait être en droit de vous en demander la démonstration.

Si vous deviez mettre en production votre produit, il faudrait prévoir une période (par exemple 2 mois) où les problèmes trouvés seraient corrigés gratuitement. Passé cette période, la maintenance deviendrait un service payant.

Ce que je vous propose comme critère de Reliability est de définir comme objectif le fait d'avoir exécuter l'ensemble de vos tests fonctionnels sans une seule erreur. Vous serez surpris de voir que c'est un objectif très ambitieux.



I : Installability

L'installation du produit est le premier contact du client avec le produit. Autant que cela se passe le mieux possible.

La solution idéale est bien sûr :

1. D'avoir une installation automatique.
2. Dans le cas où la saisie de paramètres est nécessaire, de le faire au travers de panels simples à comprendre et à renseigner.
3. Dans le cas où un problème est détecté pendant l'installation, d'avoir une explication claire du problème rencontré et la documentation du mécanisme à utiliser pour le contourner.

Vous pouvez penser qu'un site web ne s'installe pas, mais dans la réalité beaucoup d'entre vous répondent à un appel d'offre, et le produit devra être installé en avril sur le serveur du client.

Pour DELIRE, une documentation papier d'installation pourra être suffisante.

M : Maintainability

La maintenance d'un logiciel (c'est-à-dire la correction des bugs) a deux défauts majeurs :

1. elle coûte cher et est difficile à facturer
2. elle est source de découragement pour les équipes de développement et peut être à l'origine d'un turnover coûteux pour l'entreprise.

Pour en réduire le coût, il y a 3 voies :

Livrer un produit avec peu de bugs : ceci suppose un travail important en phase de SGF et d'architecture logique

Réduire le temps d'analyse d'un incident

Réduire le temps de correction d'un incident.

Vue de l'utilisateur, la maintenabilité est la facilité à pouvoir décrire un problème et transmettre à l'équipe de maintenance les données qui lui permettront de reproduire le problème pour pouvoir en faire l'analyse et le corriger. En général, on fournit avec le produit une documentation qui permet de bien documenter le problème soumis.

Vue du développeur, la maintenabilité est la facilité à pouvoir analyser et corriger un problème. Ceci suppose :

1. De pouvoir reproduire facilement un problème
2. D'identifier rapidement dans quel composant se situe le problème.
 - a. Ceci suppose que chaque composant analyse les entrées qui lui sont passées, et renvoie si besoin un cas d'erreur.
 - b. Ceci suppose également que le composant renvoie une erreur si un traitement se passe mal ou si un composant qu'il a appelé a lui-même levé une erreur.
 - c. Et il peut être bien d'avoir un mode trace pour pouvoir comprendre l'exécution d'un module
3. D'avoir un code facile à lire et à comprendre pour pouvoir y intervenir avec confiance.
4. D'avoir un ensemble de tests qui permettront de certifier que l'erreur est effectivement corrigée, et que la correction n'a pas entraîné de régression.

Les cas d'erreur qui seront renvoyés par un composant :



1. seront a priori des erreurs internes, car un composant appelant peut avoir un comportement nominal ou correctif en cas d'erreur levé par un composant appelé
2. peuvent devenir des erreurs externes (qui devront être documentées) si l'erreur remonte jusqu'à un composant de présentation.



D : Documentation

La documentation User est généralement un document qui a deux objectifs :

1. présenter le fonctionnement nominal du produit
2. définir la démarche à avoir face à un comportement anormal du produit (absence de réponse, message d'erreur...)

La documentation User ne présente généralement pas des exemples de best practices du produit : dans certains cas, cette information pourra faire l'objet d'un service payant.

Il existe 3 façon de faire accéder l'utilisateur à la documentation user :

1. Une documentation papier : elle a le défaut de se perdre et de ne jamais être disponible quand on en a besoin
2. Une documentation en ligne : elle apparait (c'est généralement un .pdf) lorsqu'on clique sur une icône de type 
3. Une documentation contextuelle : la documentation de la commande en cours ou du cas d'erreur en cours apparait) lorsqu'on clique sur l'icône 

Le plus agréable est évidemment la documentation contextuelle. Mais celle-ci suppose de prendre le problème très tôt, et de truffer tant les commandes que la documentation de mots clé.

Ceci me semble incompatible avec le projet DELIRE. Je vous propose de viser une documentation en ligne.

La compatibility

Dans la réalité, il existe un autre critère de qualité pour un logiciel: sa compatibilité. Bien entendu, cette notion n'a pas de signification pour le projet DELIRE, puisque votre proposition est totalement originale.

Je vais tenter de nuancer cette affirmation. Même si cela peut vous décevoir, il y a peu de chance que votre produit soit l'unique produit software utilisé par votre client. Il y a de bonne chance que votre client utilise par exemple la suite Office (Word, Excel, Powerpoint...). Dans ce cadre, je vous recommande fortement d'identifier quels sont les standards de menus et d'icônes qui se sont imposés, et de vous y confirmer. Je vous rappelle qu'un des principes de base de la convivialité est de se conformer à un standard qui réduit le travail de réflexion de l'utilisateur.

Maintenabilité et Evolutivité

Le modèle CUPRIMD est un modèle orienté utilisateur : c'est-à-dire qu'un utilisateur est capable de noter un produit selon chacun des critères de CUPRIMD au bout d'un certain nombre d'heures d'utilisation. Ceci définit une vision statique de la qualité du produit : selon CUPRIMD, le produit vaut tant à un instant donné.



Il existe un autre critère de qualité qui est l'évolutivité du produit, c'est-à-dire la capacité à intégrer de nouvelles fonctionnalités pour répondre au demande d'évolution : c'est un critère dynamique.

L'histoire du logiciel montre que tant la maintenabilité que la capacité d'évolution sont fortement facilitées par la qualité d'écriture du code et la documentation interne du code. En d'autres termes, la qualité d'écriture du code est un critère majeur de qualité du produit.

La mémoire du groupe, l'importance de la communication

Un des problèmes classiques rencontrés dans les projets est l'absence de suivi des décisions prises. Dans la réalité, les décisions sont prises oralement et ne sont pas consignées. Ou bien elle sont bien rédigées mais non communiquées. Et les problèmes qu'elles étaient sensées résoudre continuent à perdurer. Dans le projet DELIRE, les décisions importantes seront généralement prises durant les réunions projet. Il est donc important que les comptes rendus de réunions de projet soient clairs, synthétiques et exhaustifs. Le rôle du responsable de qualité est de définir un compte rendu type. Ma recommandation est que les comptes rendus ne soient pas toujours rédigés par la même personne.

N'oubliez pas également que les livrables écrits au premier semestre seront les documents de travail du second semestre. La lisibilité, la cohérence tant de forme que de fond, et la « complétude » du travail doit être une priorité du responsable de la qualité

Le Plan Qualité dans DELIRE

Je rappelle que bien entendu vous pouvez et vous devez concevoir votre propre Plan Qualité.

Et vous devez, si vous êtes responsable qualité, comprendre que votre travail est principalement

1. de vérifier que rien n'a été oublié pendant la phase de spécification et de structuration
2. de permettre que les membres de l'équipe gardent constamment à l'esprit les objectifs chiffrés et les méthodologies de travail définies, durant la phase de réalisation.

Voici une démarche que je peux vous proposer

1. Vérifiez que les SFG et l'architecture logique ont été validées au travers de validation de scénarios de tests fonctionnels
2. Définissez et/ou validez le standard de présentation, validez les icônes choisies, validez que toute les commandes sont accessibles immédiatement, définissez un nombre maximum d'interactions pour une commande usuelle. Vérifiez que ces propositions ont été prises en compte dans l'architecture fonctionnelle.
3. Définissez et/ou validez les temps de réponse pour les commandes usuelles. Déduisez-en les performances attendues pour chaque composant unitaire.
4. Validez que dans le planning du second semestre les rédactions de la documentation d'installation et de la documentation User ont été prévues.



5. Définissez et/ou validez une norme de programmation : nom des variables, indentation, gestion des erreurs, pourcentage de commentaire, nom des fonctions....
6. Définissez et/ou validez une norme de rédaction des compte rendus de réunion de projet.
7. Définissez et/ou validez une norme de rédaction des livrables : police de caractères, haut et bas de page, numérotation, lexique, termes employés...
8. Définissez une norme de fiche de livraison d'un composant logiciel.

Le responsable du composant doit en particulier s'engager sur le fait que

- a) le code fonctionne
- b) les signatures des méthodes sont conformes aux spécifications de l'architecture
- c) les tests unitaires ont été faits
- d) les erreurs trouvées ont été corrigées
- e) que les performances sont conformes aux valeurs attendues.

Dans les fiches de livraison, on doit également trouver le temps passé en développement, le nombre d'erreur trouvées, le temps passé en correction.

Vous l'avez compris, le responsable de qualité, comme le responsable de la gestion des risque, a un rôle fort de contrôle sur le projet. L'objectif n'est pas de faire le travail des architectes ou des développeurs, mais de s'assurer que les objectifs qualité, qui traduisent l'adéquation aux besoins du client, restent perpétuellement une priorité de tous. Pour ce faire, il met à leur disposition des outils de contrôle.

Dernier message.

Il n'est pas possible pour le responsable qualité de tout contrôler: ce serait malsain et chronophage.

Faites confiance à chaque membre de l'équipe pour s'autocontrôler. Mais pour ce faire, définissez clairement ce qu'il a à contrôler, et donnez-lui les moyens de ne pas oublier de le faire.

