

Méthodes de développement

Gestion de la configuration

1 - Activités de la gestion de configuration.....	2
2 - Définition de la configuration	2
3 - Identification et gestion des versions	3
3.1 Impact des modifications	3
3.2 Identification des versions.....	4
3.3 Marquage des produits	4
3.4 Gestion des versions avec outil de gestion de configuration	4
3.5 Gestion des versions sans outil de gestion de configuration.....	5
4 - Procédures d'évolution	5
4.1 Evolutions induites par la réalisation	5
4.2 Evolutions du besoin en cours de réalisation	6
4.3 Evolutions du besoin en utilisation	6
5 - Contrôle des produits pour la recette et les livraisons	7
6 - Le plan de gestion de la configuration.....	7

1 - Activités de la gestion de configuration

Gérer la configuration, c'est prendre en compte les "changements", qu'ils soient demandés ou intervenus :

- venant des réalisateurs qui introduisent des modifications pour corriger des problèmes et créent ainsi de multiples versions,
- venant du client dont le besoin peut évoluer,
- venant des fournisseurs de matériels et outils de production qui font évoluer leurs produits.

Chaque "changement" accepté conduit à la réalisation de nouvelles versions du logiciel, il faut donc :

- gérer les multiples versions, et pour cela les identifier,
- gérer les demandes de changement,
- pouvoir contrôler l'état des produits logiciel.

Ce sont les principales activités de la gestion de configuration

2 - Définition de la configuration

D'après la petit Larousse, la configuration est "la forme extérieure d'un ensemble" ou "l'aspect général, la physionomie".

La notion de gestion de configuration vient du matériel, où il faut gérer tout ce qui contribue aux aspects ou formes extérieurs vus par le client, tant physiques que fonctionnels.

La définition de la configuration pour les systèmes militaires est :

"l'ensemble des caractéristiques fonctionnelles et physiques du produit telles qu'elles sont décrites dans les documents de son système documentaire".

Pour un logiciel, qui est un produit immatériel (séquence d'informations binaires), on ne peut pas parler de caractéristiques physiques, de forme, d'aspect...

La configuration se limite donc aux caractéristiques fonctionnelles, et gérer la configuration, c'est gérer tout ce qui a une incidence sur les caractéristiques fonctionnelles du logiciel.

Toute modification, fut elle minime, de la séquence binaire pouvant avoir une incidence sur le fonctionnement du logiciel, il faut gérer tout ce qui entre dans la production de la séquence binaire (sources, bibliothèques, compilateur...).

3 - Identification et gestion des versions

3.1 Impact des modifications

Une modification de code source conduit à une nouvelle version du code source :



Cependant la modification ne donne pas toujours le résultat escompté :

- la solution retenue pour corriger n'était pas bonne...
- la correction a eu un effet de bord (introduction d'autres erreurs...).

Lorsqu'on corrige un module de code, il faut prendre des précautions :

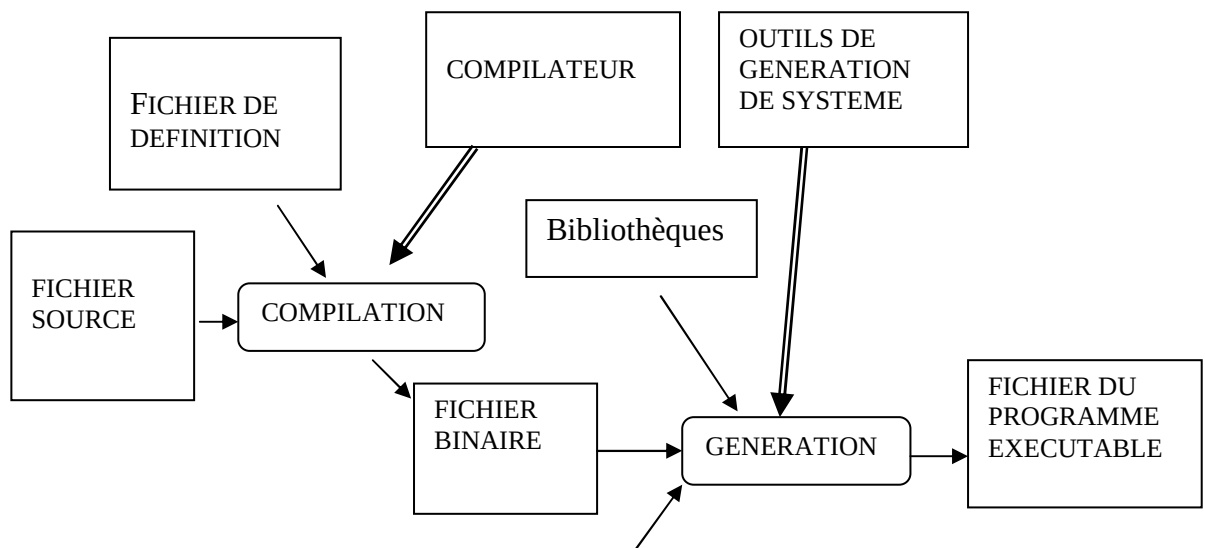
- donner une identification différente à chaque version (même si le changement n'est que minime),
- bien noter toutes les modifications introduites,
- archiver les anciennes versions (possibilité de retour à une ancienne version).

Le binaire issu de la compilation dépend :

- du fichier source à compiler,
- des fichiers de définition référencés dans le code source (`#include C++`, `import Java`, `using C#...`),
- de la version du compilateur.

De même le binaire du programme à exécuter dépendra :

- des binaires issus de la compilation,
- des fonctions de bibliothèque ajoutées par la génération de système (ou édition de liens),
- parfois de la version des outils de génération de système.



Une modification d'un fichier source conduit à une nouvelle version du fichier binaire issu de la compilation puis du ou des programmes exécutables qui en dépendent.

Si l'on est amené à modifier des fichiers de définition référencés, ou changer de version de compilateur, de bibliothèques ou d'outil de génération de système, cela conduira à de nouvelles versions de toute une série de binaires qui en dépendent :

modification de	nouvelle version de	nouvelle version de
fichier de définition référencés	binaires dépendants	exécutables dépendants
fichier de compilation	binaire correspondant	exécutable dépendant
compilateur	tous binaires	tous exécutables
bibliothèques		tous exécutables
génération système		tous exécutables

Les fichiers binaires font aussi l'objet de multiples versions dont il faut gérer les dépendances avec les sources, les fichiers de définition référencés et les versions des outils.

3.2 Identification des versions

Chaque élément de la chaîne de production (source à inclure, module source, module binaire, exécutable) correspond à un fichier et doit être identifié par un nom et un numéro de version.

Chaque modification, même la plus minime, d'un module ou d'un élément dont il dépend, doit entraîner une évolution du numéro de version.

La dénomination usuelle des versions est de la forme Version m.n ou Version Pm.n :

- m est mis à 1 au départ puis incrémenté à chaque évolution majeure de fonctionnalités,
- n est mis à 1 au départ et à chaque modification de m, et incrémenté à chaque correction ou modification mineure de fonctionnalité.

3.3 Marquage des produits

Pour les matériels, il est demandé de placer une étiquette d'identification sur chaque matériel...

A défaut de pouvoir étiqueter le logiciel qui est immatériel, il est recommandé :

- pour les fichiers de sources, d'inscrire le numéro de version sous forme de commentaire dans le code, et modifier ce commentaire à chaque évolution,
- lorsqu'on dispose d'un outil de gestion de configuration, faire correspondre la référence de version du fichier au numéro de version que l'outil associe à chaque version de fichier,
- sans outil de gestion de configuration, tenir à jour un tableau des versions et dépendances des fichiers.

3.4 Gestion des versions avec outil de gestion de configuration

Les outils de gestion de configuration prennent en général en charge :

- l'identification des versions (numéro de version),
- l'archivage sous forme compactée des versions successives de sources,
- la fourniture de différentiels entre versions,
- le suivi des dépendances et des recompilations suite à une modification.

Un outil de gestion de configuration est fortement recommandé pour tout développement important.

Sous Unix on a un outil de base en standard : SCCS, et plusieurs outils évolués sur le marché, par exemple : SPMS.

On trouve en logiciel libre les outils CVS (Concurrent Version System) et SVN(Sub Version Number) qui fonctionnent sous Unix et sous Windows, et permettent de regrouper et gérer les versions de sources d'un groupe de programmeurs, sur un serveur local ou à travers Internet.

3.5 Gestion des versions sans outil de gestion de configuration

Chaque module (source, binaire, exécutable) correspondra à un fichier.

En général les fichiers ne peuvent être identifiés que par le nom et la date/heure de dernière modification.

Recommandations :

- ne pas faire évoluer les noms des fichiers entrant dans le répertoire de la chaîne de production (problème des "#include", des procédures de production...),
- faire une fiche pour chaque version de source en notant les modifications introduites,
- avant de commencer à modifier une version, en faire une copie dans un répertoire d'archives,
- pour les modules source, mettre le numéro de version sous forme de commentaire dans le code,
- tenir à jour un tableau des versions et dépendances de chaque fichier de module source, binaire ou exécutable.

type	fichier	version	date/heure modif	fichier d'archive	dépendant des fichiers/versions

4 - Procédures d'évolution

4.1 Evolutions induites par la réalisation

Lorsqu'on code un logiciel, on introduit de multiples erreurs qui sont ensuite détectées lors des tests, soit par les tests unitaires, soit par les tests d'intégration, soit par les tests de validation.

Lorsqu'on corrige une erreur, il faut non seulement tester si la correction est efficace, mais aussi contrôler que la correction n'a pas entraîné d'autres erreurs (tests de non régression).

Corriger une erreur coûte d'autant plus cher que la réalisation est plus avancée, et peut conduire à ne plus pouvoir respecter les délais, le remède peut être pire que le mal...

Il est souhaitable de corriger les erreurs détectées en tests unitaires, mais à partir de l'intégration il est parfois préférable de ne pas corriger certaines erreurs. Il est préférable de

livrer un logiciel avec des erreurs connues que de livrer en retard voire ne pas pouvoir livrer...

De même si l'on n'est pas dans la dernière itération d'un processus itératif, il peut être préférable de terminer l'itération et reporter la correction des erreurs à l'itération suivante.

Il faut donc mettre en place une procédure à partir de la phase d'intégration pour :

- évaluer les conséquences de la correction des erreurs,
- que le chef de projet décide de la correction ou non correction des erreurs.

Les erreurs non corrigées seront identifiées dans la documentation d'utilisation, pourront faire l'objet de réserves lors de la recette, et pourront être corrigées dans une nouvelle version réalisée au titre de la garantie

4.2 Evolutions du besoin en cours de réalisation

Le besoin d'un client évolue, et il arrive qu'il demande des évolutions en cours de réalisation.

La prise en compte de telles demandes peut conduire à des augmentations de coût et de délais, et ce d'autant plus que la demande est plus tardive.

Il faut donc d'abord minimiser le risque de demandes tardives :

- en faisant au départ une revue de contrat et cahier des charges avec le client,
- en lui soumettant pour avis des qu'ils sont rédigés la STB et le DAL,
- en lui présentant pour avis des que possible des maquettes ou prototypes.

Il faut aussi mettre en place dès le départ une procédure d'évolution avec le client pour :

- évaluer l'impact sur les coûts et délais des demandes du client,
- proposer une solution au client (prise en compte sans conséquences contractuelles, prise en compte avec modifications de coût et/ou de délais, report à une version ultérieure),
- demander l'accord explicite du client avant le commencement de la prise en compte d'une demande ayant des conséquences contractuelles.

4.3 Evolutions du besoin en utilisation

Lors de l'utilisation d'un logiciel, il peut se révéler de nombreux besoins d'évolutions :

- corrections d'erreurs identifiées mais non corrigées avant livraison,
- corrections d'erreurs détectées en utilisation,
- corrections de problèmes rencontrés en utilisation (utilisation malcommode...),
- évolution du besoin des utilisateurs, souhaits d'amélioration.

La prise en compte de ces besoins nécessite la réalisation de nouvelles versions regroupant les modifications qui seront retenues.

Il est donc recommandé de mettre en place après livraison une "procédure de modification" client pour :

- recueillir tous les problèmes et besoins d'évolution (retour d'expérience),
- trier les problèmes et besoins qui méritent une prise en compte, et évaluer la difficulté de cette prise en compte,
- décider du contenu (modifications retenues) des nouvelles versions.

5 - Contrôle des produits pour la recette et les livraisons

Le logiciel est immatériel et ne peut être étiqueté et il n'est pas facile de vérifier si un logiciel correspond à ce qui était attendu (il y a de multiples exemples d'erreurs de produits lors de la livraisons de logiciels).

Le client doit pouvoir s'assurer en particulier que le logiciel qu'on lui livre est bien celui qui a été recetté.

Il est recommandé pour cela de préparer pour chaque recette ou livraison un dossier d'identification du produit donnant les références (nom et version) de tous les fichiers et outils entrant dans la composition et la fabrication du produit, et les références inscrites sur les supports de livraison.

Il faudrait aussi, dans la mesure du possible, permettre au client de contrôler le contenu de la livraison par rapport au dossier d'identification.

6 - Le plan de gestion de la configuration

Selon l'importance du projet ou les demandes du client, les règles de gestion de la configuration font l'objet d'un plan de gestion de configuration ou d'un chapitre du plan qualité.

Sujets à aborder :

- produits gérés en configuration (logiciels, fichiers source, outils de production, documents...),
- règles d'identification des produits et de leurs versions,
- méthode de gestion des versions, outils de gestion,
- procédures de modification interne,
- procédure de modification client,
- contrôles de configuration.

Dans le cas d'un document indépendant, il faut aussi indiquer l'objet, les documents de référence et le domaine d'application.