

Méthodes de développement

Fabrication et maintenance du logiciel

1 - Objet.....	2
2 - Conduite de la réalisation du logiciel.....	2
3 - La conception détaillée d'un sous-système	2
3.1 But de la conception détaillée	2
3.2 Démarche de la conception détaillée.....	3
3.3 La documentation de la conception détaillée d'un sous-système	3
4 - Codage et essais	3
4.1 But et contenu de l'activité	3
4.2 Les langages de programmation.....	4
4.3 Choix du langage de programmation	4
4.4 Codage en programmation objet	5
4.5 Modifications du source	6
4.6 Outils d'aide au codage.....	6
4.7 Tests unitaires.....	7
4.8 Tests logiques (boite blanche).....	7
4.9 Tests fonctionnels (boite noire).....	8
4.10 Réalisation des tests unitaires.....	8
4.11 Outils d'aide aux tests.....	9
5 - Intégration du logiciel	9
5.1 But de l'intégration	9
5.2 Préparation de l'intégration.....	9
5.3 Conduite d'une intégration	10
5.4 L'intégration dans un processus itératif.....	10
5.5 Le plan d'intégration logiciel.....	10
5.6 La procédure de mise en œuvre des tests d'intégration	11
6 - Essais de validation.....	11
6.1 But de l'activité.....	11
6.2 Déroulement de la validation	12
6.3 Le plan de validation	12
6.4 Procédure de mise en œuvre des essais de validation	12
7 - Recette et livraison.....	12
7.1 Préparation de la recette et de la livraison.....	12
7.2 La procédure de recette	13
8 - Après vente et maintenance	13
8.1 Services après vente	13
8.2 Objectif de la maintenance logiciel	14
8.3 Réalisation d'une nouvelle version de logiciel	14
8.4 Besoins pour la maintenance	15
8.5 Préparation de la maintenance.....	15

1 - Objet

Ce chapitre a pour objet de donner des indications pratiques sur les activités de réalisation du logiciel : conception détaillée, codage et essais, intégration et validation.

Il aborde également la préparation à la maintenance.

2 - Conduite de la réalisation du logiciel

Lors de l'étude de l'architecture du logiciel, le logiciel est découpé en sous-systèmes pour les travaux de réalisation.

Chaque sous-système sera confié à un sous-groupe de réalisation (un sous-groupe peut avoir plusieurs sous-systèmes à réaliser).

Le sous groupe fait d'abord la conception détaillée du sous-système : il découpe le sous-système en modules de réalisation et définit pour chaque module les interfaces du module et les fonctions du module.

Dans une méthode de conception objet, le module de réalisation est la classe. Dans une méthode de conception structurée, le module de réalisation est une fonction ou un sous-programme.

Le sous-groupe fait ensuite les travaux de codage et essais module par module.

Lorsqu'un certain nombre de modules ont été codés et testés par les sous-groupes de réalisation, on met en place un sous-groupe d'intégration pour préparer et conduire les travaux d'intégration.

Lorsque les travaux d'intégration sont avancés et que l'activité de validation est prévue, on met en place un sous-groupe de validation (qui peut être composé des membres du sous-groupe intégration) pour préparer et conduire les travaux de validation.

3 - La conception détaillée d'un sous-système

3.1 But de la conception détaillée

Le codage et les essais (tests unitaires) sont faits sous-programme (classe ou fonction) par sous-programme (classe ou fonction).

Il faut donc en préalable avoir identifié tous les sous-programmes à coder et avoir défini leur rôle et leurs interfaces. C'est le rôle de la conception détaillée du sous-système.

Pour les sous-programmes ayant des fonctions particulièrement complexes, il faudra compléter cette architecture par une description du fonctionnement interne du sous-programme.

3.2 Démarche de la conception détaillée

La démarche comprend :

- identification de tous les sous-programmes du sous-système (fonctionnalités non couvertes par les sous-programmes de base, besoins venant de l'implémentation),
- définition de leur rôle et de leurs interfaces d'appel,
- définition des interfaces externes (protocoles de communication, fichiers accédés, appels au système d'exploitation, IHM...),
- analyse de la structure interne des sous-programmes complexes ou importants.

On peut alors passer au codage et aux tests unitaires sous-programme par sous-programme.

3.3 La documentation de la conception détaillée d'un sous-système

La conception détaillée d'un sous-système doit être documentée.

Cependant elle peut prendre de multiples formes en fonction d'exigences éventuelles du client, de l'importance et de la criticité du projet, des objectifs de qualité...

Une première solution est de rédiger un document de conception détaillé pour chaque sous-système. Ce document pourra comprendre par exemple :

- présentation du sous-système,
- liste des sous-programmes, de leurs relations et de leur rôle,
- éventuellement, définition de scénarios d'interaction et diagrammes correspondants,
- définition des interfaces d'appel des sous programmes,
- définition des autres interfaces (protocoles de communication, fichiers privés, IHM particulières au sous-système...),
- analyse des classes complexes ou importantes.

Cependant cela fait une documentation très volumineuse qui n'est pas simple à tenir à jour lors des modifications du logiciel.

Une solution alternative est de réduire cette documentation aux informations les plus importantes (rôle, interfaces) et de l'intégrer sous forme de cartouches de commentaires dans le code des sous-programmes.

4 - Codage et essais

4.1 But et contenu de l'activité

Le but de l'activité est de traduire pour un sous-système la conception dans un langage de programmation, préparer les tests unitaires, et les effectuer.

Cette activité s'appuie sur la spécification technique de besoins, le document d'architecture logicielle, et les résultats de la conception détaillée du sous-système.

Pour chaque module de codage (classe en conception objet) :

- l'interface est défini complètement, codé et diffusé aux réalisateurs qui appellent le module,
- le corps du module est codé,

- les jeux de tests sont définis et les programmes de tests préparés,
- les tests unitaires sont effectués,
- la partie du compte-rendu des tests relative au module est rédigée.

L'ordre de codage des modules sera défini en fonction du besoin des tests.

Les résultats de cette activité sont :

- les fichiers de code source des modules,
- les comptes-rendus des tests unitaires.

4.2 Les langages de programmation

Le codage nécessite d'avoir recours à un langage de programmation.

Au début de l'informatique, les premiers langages ont été les assembleurs et macro-assembleurs qui se limitaient à transposer le code du calculateur.

Puis sont apparus dans les années 70 les premiers langages symboliques issus de l'expérience :

- FORTRAN (scientifique),
- COBOL (gestion).

Dans les années 80 les recherches universitaires ont fait apparaître les langages structurés et puis typés :

- ALGOL, le père des langages structurés,
- C, le langage d'UNIX,
- FORTRAN 90, extension du FORTRAN,
- ADA, création du DOD (ministère de la défense des Etats Unis), le plus complet et le plus complexe...

Dans les années 90 sont apparus les langages objet :

- C++, le langage adopté par l'industrie,
- EIFFEL (enseignement),
- SMALLTALK (enseignement),
- JAVA pour Internet,
- C# concurrent Microsoft de Java.

4.3 Choix du langage de programmation

Les critères à prendre en compte pour le choix du langage de programmation sont :

- le domaine d'application (scientifique, gestion, industriel et temps réel),
- la disponibilité de bibliothèques ou de code réutilisable,
- le style de conception (structurée, orientée objet),
- la disponibilité et la maturité des compilateurs et outils associés.

Les écueils à éviter sont :

- de choisir un langage inadapté pour le domaine ou le style de conception,
- de choisir un langage non utilisé largement dans l'industrie,
- de choisir un compilateur très peu diffusé.

Par exemple si on fait de la conception objet, il faut choisir un langage objet, et on prend moins de risques en choisissant les langages C++ , Java ou C# largement utilisés dans l'industrie, que Smalltask ou Eiffel, plus brillants sur le plan conceptuel, mais limités à quelques cercles universitaires.

Si par contre, on a à faire un programme scientifique, la plupart des industriels travaillent en Fortran dans ce domaine, et la quasi totalité des bibliothèques est en Fortran. Il vaut mieux renoncer aux objets et travailler en Fortran.

3.4 Codage en programmation objet

Le module de codage est la classe.

Selon les langages, une classe peut être codée dans un seul fichier, ou répartie sur plusieurs fichiers :

- en C++ on met l'interface dans un fichier à part (fichier h), et le code peut être dans un ou plusieurs fichiers (fichiers cpp),
- en Java et C# interface et code sont regroupés dans le même fichier, mais C# permet aussi de repartir la définition d'une classe sur plusieurs fichiers (mécanisme de partial class).

Il est recommandé de commencer le codage par le codage des interfaces des classes et de diffuser ces fichiers aux autres réalisateurs qui auront à prendre en compte ces interfaces.

Il faut veiller à la lisibilité du code :

- en insérant des commentaires,
- en choisissant des identificateurs significatifs,
- en limitant l'emploi des pointeurs et des références,
- en limitant l'utilisation des mécanismes de redéfinition, liaison dynamique...

Il est recommandé de faire commencer chaque fichier source par un cartouche de commentaire comprenant :

- le nom de la classe concernée,
- son rôle dans le logiciel,
- les références au document d'architecture et aux cas d'utilisation de la STB,
- le numéro de version et la date de création correspondante,
- les méthodes appelées et leurs classes,
- le nom du responsable de la classe,
- et, pour chaque modification, à partir de l'intégration, l'origine, le numéro, la description et la date de modification.

Il est aussi recommandé de faire une cartouche de commentaire pour chaque méthode afin de préciser le rôle de la méthode, et pour chaque paramètre d'appel le rôle et les valeurs possibles.

Il est aussi recommandé de mettre des lignes de commentaires dans le code pour expliciter les opérations réalisées. Un taux de commentaires de 50 % est usuellement recommandé.

4.5 Modifications du source

Lorsqu'on écrit un module source, on fait de nombreuses erreurs qui seront détectées par la suite lors des essais unitaires puis de l'intégration.

On sera donc conduit à modifier de multiples fois un module source. Il ne faut pas modifier un module source sans précautions, sinon on risque de se retrouver devant de multiples versions d'un module source et ne plus pouvoir savoir quelle est la bonne.

Il ne faut pas non plus éliminer systématiquement les anciennes versions, car de temps en temps les modifications introduites ne font qu'empirer la situation et on souhaite retourner à l'ancienne version.

Avant d'introduire un lot de modifications il faut donc archiver l'ancienne version et mettre à jour un identificateur pour reconnaître les versions successives. L'utilisation d'un outil de gestion de configuration facilite grandement cette démarche qui est détaillée dans le chapitre gestion de configuration.

4.6 Outils d'aide au codage

Les principaux outils sont les éditeurs de texte qui facilitent la saisie ou la modification du code. Ces éditeurs peuvent être :

- généraux, indépendants du type de texte entré,
- syntaxiques, prenant en compte la syntaxe du langage de programmation et détectant les erreurs de syntaxe,
- sémantiques, prenant en compte la syntaxe et la sémantique du langage de programmation et détectant les erreurs de syntaxe et de sémantique.

On dispose aussi de générateurs automatiques de code, mais il ne permettent aujourd'hui de générer que des squelettes de programmes (par exemple le générateur C++ de RATIONAL ROSE) qu'il faut compléter à la main, ou des parties particulières de programmes (par exemple code correspondant à l'affichage d'un écran, construction de requêtes d'accès à une base de données...).

Les bibliothèques de classes prédéfinies permettent d'éviter de coder des fonctions standardisées.

On dispose aussi d'ateliers de développement logiciel qui intègrent autour du compilateur les outils d'aide au codage, des outils d'aide au test, et parfois un outil de gestion de configuration.

Un exemple est VISUAL STUDIO qui comprend autour des compilateurs VISUAL C++ et C# :

- des éditeurs de code,
- la génération automatique de code pour des graphiques, des événements...,
- les bibliothèques de classes MFC et Windows Forms,
- un debugger,
- une aide en ligne.

4.7 Tests unitaires

Pour détecter les erreurs de programmation, chaque module (classe) de code fait l'objet de deux séries de tests unitaires :

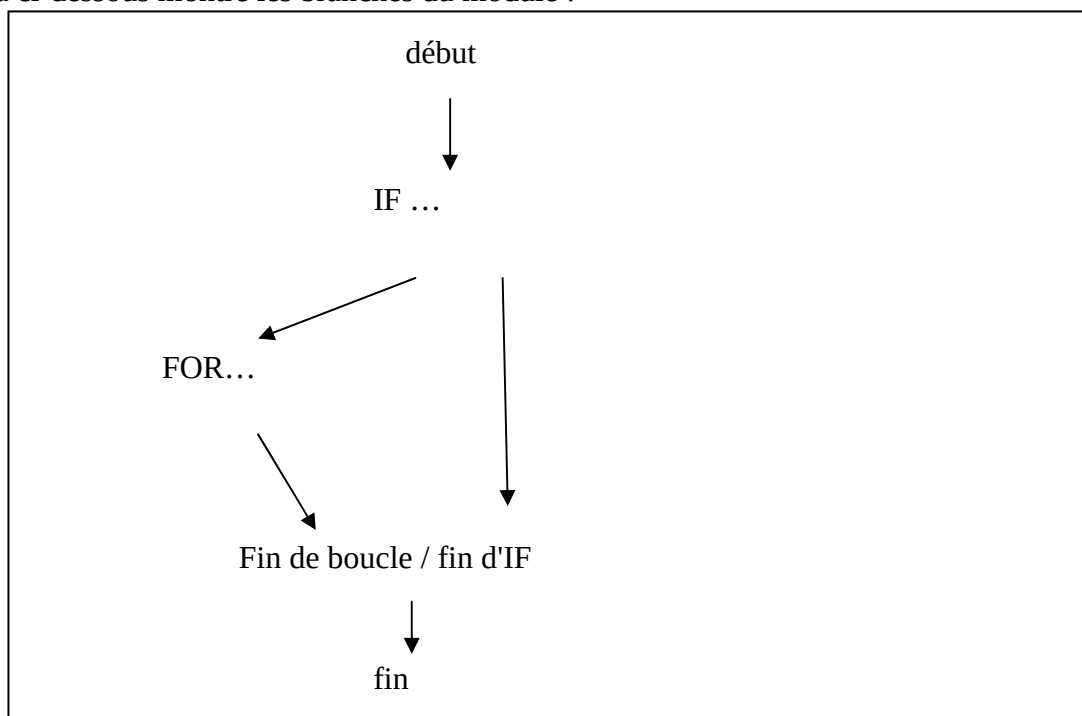
- tests logiques (tests "boite blanche"),
- tests fonctionnels (tests "boite noire").

La définition de certains de ces tests dépend de la manière dont est codé le module, ces tests ne pourront donc être définis qu'après codage du module.

4.8 Tests logiques (boite blanche)

Une première série de tests logiques a pour but de s'assurer que chaque branche du module peut être exécutée.

Le schéma ci-dessous montre les branches du module :



Pour réaliser cette première série de tests logiques :

- on choisit des jeux de variables d'entrée (paramètres d'appel ou variables publiques utilisées) permettant de passer dans chaque branche du programme au moins une fois,
- on fait exécuter le module pour chacun des jeux de variables d'entrée retenus,
- on vérifie qu'il n'y a pas eu d'exceptions levées et que les paramètres de sortie sont cohérents avec les paramètres d'entrée.

Une deuxième série de tests logiques permet de tester les interfaces du module avec les autres modules du sous-système déjà réalisés. Pour cela on fait exécuter ensemble les modules dont on veut tester les interfaces, et on vérifie que les données sont correctement transmises entre les modules.

4.9 Tests fonctionnels (boite noire)

Le but est de s'assurer que les résultats fournis par le module (paramètres en sorties, variables publiques mises à jour, fichiers disques, écrans ou éditions) sont conformes à ce qui était attendu.

Les tests peuvent être effectués au niveau d'un module, ou sur plusieurs modules d'un sous-système qui participent à une même fonction.

Il faut donc connaître les résultats attendus et pour cela trouver une technique indépendante du programme à tester pour calculer ces résultats, ce qui n'est pas toujours simple.

Il faut ensuite faire un échantillonnage des paramètres d'entrée qui permet de couvrir leurs domaines de variation.

Il faut aussi tester les cas aux limites (par exemple pour une fonction racine carrée tester le cas zéro).

4.10 Réalisation des tests unitaires

Les tests unitaires peuvent être réalisés :

- soit en faisant exécuter le ou les modules à tester sous contrôle du debugger,
- soit en écrivant et faisant exécuter des programmes de test.

Pour l'exécution sous contrôle du debugger, on utilise le debugger du langage qui permet en général :

- d'initialiser les paramètres d'entrée,
- de définir des points d'arrêt,
- de choisir l'instruction de début d'exécution et lancer l'exécution au pas à pas ou jusqu'au prochain point d'arrêt,
- à un point d'arrêt de visualiser ou modifier des variables,
- de simuler l'exécution des fonctions externes appelées par le module,
- de visualiser les variables en fin d'exécution.

Pour la réalisation des tests par programme de test, on écrit pour chaque module ou groupe de modules à tester un petit programme principal qui initialise les paramètres d'entrée, appelle le point d'entrée ou les points d'entrée du module, puis affiche ou imprime les paramètres de sortie et éventuellement fait la comparaison avec les résultats attendus. Pour toutes les fonctions externes appelées (méthodes des autres classes), il faut écrire une petite fonction qui fournit le résultat correspondant au cas de test.

On réunit alors les modules à tester et ces petits modules en un programme qu'on exécute pour les différents cas de test prévus.

Que ce soit avec le debugger ou par programme de test, on pourra utiliser des modules déjà testés comme "outillage" pour rendre plus simple la réalisation des tests. Il faut donc prévoir l'ordre de réalisation des modules en fonction du besoin de leur utilisation comme "outillage" pour les tests.

4.11 Outils d'aide aux tests

Pour aider à définir les tests, les exécuter et archiver les résultats, on dispose de nombreux outils :

- debuggers (en général associés à un langage et à un atelier),
- analyseurs structurels permettent de visualiser la structure des modules à tester,
- outils de mesure de couverture structurelle pour les tests logiques,
- outils de gestion des jeux de test et des résultats de test,
- outils de génération de cas de test.

En dernier recours, on peut encore disposer d'un dump mémoire en hexadécimal...

5 - Intégration du logiciel

5.1 But de l'intégration

Le but de l'activité d'intégration est de :

- intégrer ensemble les modules (classes) réalisés séparément pour construire le logiciel complet,
- s'assurer que chaque module rend bien les fonctions attendues et que le logiciel est prêt à être validé.

Pour réaliser l'intégration, les modules issus du codage et essais sont assemblés de façon progressive par étapes, et à chaque étape des tests sont passés pour vérifier que les appels entre modules se font correctement et que les données sont correctement échangées.

Lorsque tous les modules sont assemblés, des tests globaux sont passés pour vérifier le bon fonctionnement global du logiciel.

Les résultats des travaux d'intégration sont consignés dans le compte rendu des tests d'intégration.

5.2 Préparation de l'intégration

L'intégration est préparée en rédigeant un plan d'intégration qui définit les étapes de l'assemblage progressif, et les tests passés à chaque étape.

La définition des étapes d'intégration s'appuie sur les diagrammes de dépendance entre classes. Il y a deux techniques possibles :

- commencer par les fins de chaînes de dépendances, ce qui évite d'avoir à simuler des classes absentes, mais nécessite d'écrire des programmes de test d'intégration assez importants pour activer les fonctions et visualiser les résultats,
- commencer par intégrer l'IHM, puis ajouter progressivement les classes en descendant les chaînes de dépendance, l'IHM étant utilisée pour activer les fonctions et visualiser les résultats ; cela évite d'avoir à écrire des programmes de test, mais il faut écrire des petits logiciels pour simuler le fonctionnement des classes appelées et pas encore intégrées.

Selon le type et la taille du système à réaliser, la préparation de l'intégration demande des travaux plus ou moins importants :

- mise en place d'une plate-forme pour réaliser les essais d'intégration,

- réalisation de logiciels de simulation pour simuler des interfaces externes ou des matériels réels non présents sur la plate-forme,
- réalisation de petits logiciels pour simuler les réponses des classes absentes lors des premières étapes d'intégration
- définition des tests d'intégration,
- rédaction de la procédure de mise en œuvre des tests d'intégration, si nécessaire,
- codage et mise au point des programmes de test pour l'intégration (si intégration à partir des fins de chaînes de dépendances).

Tous ces travaux doivent être identifiés dans le plan d'intégration, et lorsqu'il sont importants (dans certains cas la simulation de matériels réels non présents ou d'interfaces est un logiciel complexe) identifiés comme tâches dans le plan de développement logiciel.

5.3 Conduite d'une intégration

Une intégration est conduite en plusieurs étapes.

Pour la première étape, on réunit la configuration à intégrer (modules logiciels, simulateurs de modules en interface, simulateurs de matériels en interface), on passe les tests d'intégration, et on fait des corrections jusqu'à ce que ces tests fournissent des résultats satisfaisants.

On découvre en général beaucoup d'erreurs en intégration. Certaines empêchent de continuer les travaux et il faut les corriger immédiatement. Pour les erreurs moins gênantes il est parfois préférable de différer la correction.

Il est nécessaire d'avoir mis en place une procédure interne de gestion des évolutions pour décider s'il y a lieu de différer ou non la correction des erreurs.

Pour chacune des étapes suivantes, on part de la configuration de l'étape précédente, on ajoute des modules ou on prend en compte des matériels réels et on ajuste les besoins en simulateurs.

On fait les étapes jusqu'à ce que le logiciel soit totalement intégré et intégré aux matériels en interface.

5.4 L'intégration dans un processus itératif

Dans un processus itératif, on fait une intégration dans chaque itération où on réalise un prototype.

Le premier prototype réalisé fait l'objet d'une intégration comme indiquée ci-dessus, mais qui devrait demander peu d'étapes car les fonctions seront limitées.

Pour chacun des prototypes suivants, on utilise le prototype de l'itération précédente comme configuration de la première étape d'intégration.

5.5 Le plan d'intégration logiciel

Le plan d'intégration définit la logique et les étapes de l'intégration et précise les moyens nécessaires pour les travaux d'intégration.

Le plan type suivant est proposé :

1	Introduction
2	Logique d'intégration
3	Moyens d'intégration
4	Contenu des étapes

Pour rédiger le plan d'intégration :

1	Indiquer l'objet du document et le système concerné
2	Indiquer les étapes prévues, leurs objectifs et l'enchaînement des étapes. Dans un processus itératif, la logique sera définie pour chaque itération.
3	Préciser les moyens matériels ou outils logiciels nécessaires pour l'intégration. Indiquer, s'il y a lieu, les développements logiciels significatifs induits par l'intégration (par exemple développement de simulateurs).
4	Indiquer pour chacune des étapes : - les constituants qui seront intégrés ou simulés, - les tests à réaliser, - les moyens spécifiques à l'étape s'il y a lieu.

Une première version du plan d'intégration sera rédigée lors de la première itération, il pourra être limité aux travaux de cette itération. Il sera complété à chaque itération.

5.6 La procédure de mise en œuvre des tests d'intégration

Les procédures de mises en œuvre des tests fournissent la description détaillée des tests et les résultats attendus, et la procédure d'exécution des tests.

Elles présenteront pour chaque étape de l'intégration :

- l'état du système à tester : composition du système à tester, logiciels ajoutés pour les tests, moyens matériels,
- un tableau des tests à réaliser : numéro de test, données d'entrée, traitements, résultats attendus, et une colonne vierge pour inscrire les résultats obtenus.

Les comptes rendus de tests sont faits en complétant ces tableaux.

6 - Essais de validation

6.1 But de l'activité

Le but de l'activité de validation est de vérifier que le logiciel répond à toutes les exigences de la STB et valider le manuel d'utilisation.

Cette activité s'appuie sur la spécification technique de besoins (STB) et le manuel d'utilisation (MU).

Cette activité est préparée en rédigeant un plan de validation, et une procédure de mise en œuvre des essais de validation.

Les résultats sont consignés dans le compte rendu des essais de validation.

Dans le cas d'un prototype, cette activité n'est pas à prévoir explicitement. Les essais montrant que le logiciel répond aux exigences de la STB sont faits en fin d'intégration.

6.2 Déroulement de la validation

Pour s'assurer que le logiciel aura un code source bien identifié et sera reproductible, les codes sources du logiciel sont transférés dans un répertoire de référence

Le logiciel est régénéré à partir de ce répertoire de référence

Une plate-forme de validation représentative de l'utilisation opérationnelle est constituée.

Une série d'essais est effectuée sur la plate-forme de validation conformément au plan de validation pour vérifier que le logiciel répond aux exigences de la STB et valider le manuel d'utilisation.

Une revue de validation est tenue pour examiner les résultats des essais de validation.

S'il n'apparaît pas d'erreurs ou de non conformités, ou qu'elles apparaissent acceptables pour l'utilisation, le logiciel peut être recetté et livré.

Dans le cas contraire, on retourne au codage et essais pour corriger les problèmes importants puis on reprend l'intégration puis la validation.

6.3 Le plan de validation

Le plan de validation définit les tests à réaliser lors des essais de validation, ainsi que les moyens nécessaires pour ces tests.

Il présente :

- les étapes de la validation, leurs objectifs et l'enchaînement des étapes,
- les moyens de validation, moyens matériels ou outils logiciels nécessaires pour la validation, et, s'il y a lieu, les développements logiciels significatifs induits par la validation (par exemple développement de simulateurs),
- le contenu des étapes : configurations d'essais, tests à réaliser, et moyens spécifiques à l'étape s'il y a lieu.

6.4 Procédure de mise en œuvre des essais de validation

Elle se présente comme la procédure de mise en œuvre des tests d'intégration.

7 - Recette et livraison

7.1 Préparation de la recette et de la livraison

Pour préparer la recette et la livraison :

- le support de livraison est préparé à partir du logiciel archivé dans le répertoire de référence,
- le manuel d'utilisation est mis à jour si besoin,

- un manuel ou une notice d'installation sont préparés,
- une fiche de version précisant le contenu de la version livrée et les éventuelles réserves d'utilisation est rédigée si besoin,
- la procédure de recette est rédigée et soumise à l'approbation du client.

Pour exécuter la recette, le logiciel est chargé sur la plate-forme de recette à partir du support de livraison en suivant la procédure d'installation. Les essais de recette sont exécutés en présence du client en suivant la procédure de recette. Ces essais sont en général un échantillonnage des essais de validation.

Le procès verbal de recette est rédigé et visé par le client.

Nota :

Dans certains cas la recette peut être faite en deux parties :

- *recette usine dans les locaux du réalisateur,*
- *recette site dans les locaux du client avec les matériels qui seront utilisés par le client.*

7.2 La procédure de recette

La procédure de recette définit le déroulement de la recette du logiciel. Elle est en général préparée par le fournisseur et soumise au client pour approbation.

Elle présente :

- le déroulement de la recette,
- les moyens nécessaires,
- les essais de recette,
- les jeux d'essais de recette.

8 - Après vente et maintenance

8.1 Services après vente

Les services à assurer sont usuellement :

- la formation des utilisateurs,
- l'assistance à l'utilisation, en général une "hot line",
- le recueil et l'analyse des problèmes,
- la fourniture de solutions de contournement (dépannage en quelques jours),
- la réalisation de nouvelles versions (demande plusieurs mois en général),

La réalisation de nouvelles versions est la maintenance du logiciel.

Tout ou partie de ces services peut être assuré directement par le client, ou par le fournisseur

- dans le cadre de la garantie,
- ou dans le cadre d'un contrat de maintenance.

8.2 Objectif de la maintenance logiciel

L'objectif de la maintenance est de pouvoir réaliser de nouvelles versions d'un logiciel pour corriger des problèmes, introduire de nouvelles fonctionnalités, permettre l'utilisation de nouveaux matériels et logiciels de base...

Il est généralement nécessaire de prévoir une maintenance du logiciel :

- le logiciel peut avoir été livré avec des problèmes recensés à corriger ultérieurement,
- un logiciel comporte de nombreuses erreurs résiduelles qui peuvent être découvertes en utilisation et s'avérer gênantes,
- les besoins des utilisateurs évoluent et ils peuvent avoir besoin de modifications de fonctions ou de nouvelles fonctions,
- les constructeurs de matériels et fournisseurs de logiciel de base font évoluer sans cesse leurs produits.

Sans maintenance, un logiciel risque de devenir rapidement inutilisable, ou avoir son utilisation limitée à des matériels anciens et figés...

On peut envisager deux types de maintenance :

- maintenance continue : une équipe travaille en permanence pour réaliser de nouvelles versions successives du logiciel... c'est coûteux et justifié uniquement pour de très gros logiciels...
- maintenance discontinue : périodiquement on définit le contenu d'une nouvelle version du logiciel, et on charge une équipe de la réalisation de cette nouvelle version.

8.3 Réalisation d'une nouvelle version de logiciel

Un cahier des charges doit préciser les objectifs de la nouvelle version (nouvelles fonctions, problèmes à corriger, matériels et versions de logiciel de base à prendre en compte...)

Le processus de développement de la nouvelle version est le même que celui de la version initiale avec quelques aménagements :

- analyse des exigences : mise à jour des STB ou rédaction d'un complément à la STB initiale,
- conception de l'architecture et conception détaillée : mise à jour du DAL ou complément au DAL initial,
- codage et essais : modifications de modules et écriture de nouveaux modules, passage de tests unitaires pour valider les modifications introduites et vérifier la non régression,
- intégration : les étapes seront adaptées aux modifications introduites et pourront en général être réduites,
- essais de validation : on ne revérifie pas toutes les fonctions, mais seulement celles nouvelles ou modifiées, et on complète de quelques essais globaux.

Dans une méthode itérative, en fonction de l'importance des modifications, ce nouveau développement pourra faire l'objet d'une ou de plusieurs itérations.

8.4 Besoins pour la maintenance

Documents

Recommandé	Spécification technique de besoins Document d'architecture logiciel Manuel d'utilisation
Souhaité	Plans d'intégration, de validation Comptes rendus des tests unitaires, d'intégration, de validation Procédé de génération du code exécutable Fiche de version

Programmes

Indispensable	Code source du logiciel (répertoire de référence)
Souhaité	Programmes de tests et jeux de tests

Outils

Indispensable	Outil d'archivage et gestion du code source
Souhaité	Compilateur, générateur de programme exécutable Atelier de production Outils d'aide aux tests

8.5 Préparation de la maintenance

Documents

Tous les documents recommandés ou souhaités sont déjà prévus dans la méthode de développement, il faut donc les rédiger effectivement et les archiver.

Programmes

Bien archiver le répertoire de référence du code source.

Archiver les programmes de test et jeux de test.

Outils

Archiver les outils indispensables ou recommandés pour la maintenance.

L'application d'une bonne méthode de développement prépare la maintenance, il suffit d'archiver les documents, programmes et outils.