Réutilisation, livraison pour la réutilisation, Biens logiciels, Bibliothèque de biens logiciels, Référentiel logiciel Patterns, frameworks, architectures à base de composants

Moyens de transmission de la connaissance du logiciel ; Traçabilité ; Bien logiciel ; Contenu du bien logiciel ; Le cas particulier des composants ; Caractéristiques d'un bien logiciel ; Méta-modélisation ; Biens logiciels verticaux et horizontaux ; Réutilisation de type boîte noire/blanche/grise/en verre ; Types de biens logiciels réutilisables ; Bibliothèque de biens logiciels ; Référentiel logiciel ; Catalogue ; Fonctions de la bibliothèque ; Les architectures à base de composants, livraison d'un composant réutilisable ; Des designs patterns, modèles de conception, livraison d'un pattern ; Les frameworks, compréhension du framework ; Ingénierie de domaine, modèle de connaissance ; Stratégie de réutilisation ; Production et consommation de biens logiciels réutilisables ; Pratique effective de la réutilisation ; Méthodologie REBOOT, ouvrage « Software Reuse » ; Gestion de version ; Tests logiciels ; Outils et technologies pour la réutilisation .

Introduction [(^)]

Ce dossier traite du développement logiciel en vue de la réutilisation de tout ou partie du logiciel développé dans d'autres logiciels. Il est question notamment de l'information nécessaire à la réutilisation, qui est à fournir lors de la livraison du produit logiciel réutilisable.

Source [(^)]

Ce dossier utilise le document « Réutilisation des logiciels : faits et mythes » de Philippe Lahaye.

Références du document

CNAM PARIS

Examen probatoire en Informatique - Session de mai 2003

LAHAYE Philippe

« *sujet n°33* **Réutilisation des logiciels : faits et mythes** »

Président du jury : Professeur WATTIAU

Accès URL : **http://lahayenadeau.free.fr/probatoire_reingenierie_reutilisation_logicielle.pdf**

Présentation du document et citations

Ce rapport d'étude bibliographique recense et analyse « les concepts et les techniques mis en avant dans le domaine de la réutilisation logicielle », « afin de voir quelle réalité la réutilisation logicielle recouvre ». « La réutilisation logicielle s'applique à deux sujets majeurs : la **ré-ingénierie** de logiciel et la **réutilisation logicielle**. La ré-ingénierie s'applique plus à la maintenance des applications et leur évolution alors que la réutilisation, dans son acceptation commune, signifie l'utilisation de tout ou partie d'une ou plusieurs applications dans une nouvelle application ».

Exploitation du document dans ce dossier

Il est repris dans la suite de ce dossier des éléments du document « Réutilisation des logiciels : faits et mythes », dont il est fait des citations (texte entre guillemets).

Les parties du document concernant la ré-ingénierie sont ici peu exploitées. Les parties du document auxquelles il est fait référence concernent essentiellement la réutilisation dans le sens d' « utilisation de tout ou partie d'une ou plusieurs applications dans une nouvelle application ». Il s'agit pour une grande part du **chapitre 2.2 « La réutilisation logicielle »**, dans lequel il est abordé « les conditions et les concepts nécessaires à la réutilisation de composants ». Pour plus de détails, se reporter directement au document « Réutilisation des logiciels : faits et mythes ».

La notion de réutilisation (^)

Réutilisation (^)

« La réutilisation consiste à développer du logiciel en partant systématiquement d'un stock de « briques de construction », afin d'exploiter les architectures et besoins communs à plusieurs applications. La réutilisation a pour objectif l'amélioration de la **qualité** et de la productivité, l'industrialisation de la production de logiciel ».

La réutilisation contribue « à l'augmentation de la maturité de l'activité de production logicielle ».

Quand (^)

« La réutilisation se définit comme une nouvelle approche de développement de systèmes selon laquelle il est possible de développer un système à partir de composants existants ». « Cette approche a été mise en œuvre **dans les phases d'ingénierie du logiciel** (programmation, conception d'architectures logicielles…). En revanche cette approche s'est peu développée **dans les phases initiales du cycle de développement** (ingénierie des besoins, spécification conceptuelle…). Pourtant de nombreux auteurs s'accordent pour dire que c'est dans ces phases là que l'approche par réutilisation est la plus prometteuse ».

La réutilisation est « une dimension importante que l'on peut développer à tous les niveaux du génie logiciel (conception, développement, test, documentation, exploitation et maintenance) ».

« La réutilisation doit être systématique, c'est-à-dire doit être pratiquée en permanence, si l'on veut atteindre les objectifs de la réutilisation ».

Renseigner le logiciel réutilisable [(^)]

Proposer un logiciel en réutilisation implique de mettre la connaissance du logiciel à la portée des réutilisateurs. Et ce serait faire abstraction de « la partie humaine de l'ingénierie du logiciel » que de sous-estimer les efforts à déployer d'une part pour **donner moyen à quelqu'un de comprendre un logiciel** et d'autre part pour **prendre connaissance d'un logiciel qu'on n'a pas conçu soi-même**.

L'information permettant de connaître et comprendre le logiciel [(^)]

Moyens/supports de transmission de la connaissance [(^)]

« **Le code source** ne contient pas toute l'information nécessaire » pour « appréhender la structure interne et la conception générale d'un système logiciel ». C'est pourquoi il est nécessaire de transmettre par **d'autres moyens** et sous **d'autres formes** des informations comme « la connaissance des architectures, des différences de conception, des contraintes diverses et enfin du domaine de l'application », faute de quoi toute cette information « existe seulement dans l'esprit des ingénieurs informatiques ayant participé à l'élaboration du logiciel ».

Voir aussi « Ingénierie de domaine, modèle de connaissance ».

Dimension historique, traçabilité [(^)]

Etant donné que le processus de développement logiciel s'inscrit dans la durée et suit une certaine logique, « la **traçabilité** est un élément fondamental » pour donner la maîtrise du logiciel. « Tous les changements opérés sur un programme informatique devraient être consignés, tant du point de vue des spécifications qui ont initié ces changements que du point de vue du code source. Un **lien explicite** doit être établi entre les spécifications et les changements de code consécutifs ».

Organiser l'information [(^)]

Face à la diversité de l'information constituant la connaissance du logiciel (l'information se trouve par exemple dans « des enregistrements de discussions, de demandes de changement et de résultats d'inspection »), « le challenge est d'**organiser** et d'exploiter cette richesse d'**information** ».

La notion de bien logiciel [(^)]

Bien logiciel [(^)]

Donner moyen de réutiliser un logiciel ne se réduit pas à remettre « uniquement le code d'une application logicielle », car alors comment comprendre ce code, savoir qu'en faire et comment le mettre en œuvre ? La réutilisation porte sur tout un « bien logiciel ».

Définition : « Un **bien logiciel** est composé d'un ensemble de produits logiciels issus des différentes activités du cycle de vie du logiciel : en particulier, l'expression des besoins, la définition d'architecture, le modèle d'analyse, le modèle de conception, le code, les programmes de test, les rapports de test ».

Contenu du bien logiciel [(^)]

« Les produits logiciels réutilisables doivent être fournis avec l'information nécessaire à leur réutilisation (la description du bien logiciel, également appelée méta-information). De même, les produits logiciels doivent être apparentés : tels tests correspondent à telle version de module de tel logiciel. »

> Un bien logiciel est « constitué de deux types d'information : le **corps** (contenant les produit logiciels effectivement réutilisables) et la **description** (contenant les informations permettant de supporter le processus de réutilisation) ».

Le cas particulier des composants [(^)]

« Le terme composant est plus spécifique » que le terme bien logiciel. « Un **composant** est un bien logiciel exécutable (sous forme binaire ou non) qui peut être intégré sans modification dans une application. Cette notion fait référence aux technologies à base de composants (EJB, Active X / DCOM, CORBA, web services) ».

Caractéristiques, critères respectés par un bien logiciel [(^)]

Généralités :
« On ne va réutiliser que des composants logiciels « **certifiés** » dont la robustesse et la fiabilité ont été éprouvées ».
« On ne réutilise que des composants qui vérifient la célèbre règle des **cinq « R »** (Reliable, Reusable, Replaceable, Resilient, Revisable) ; c'est-à-dire qu'un composant doit alors être fiable, réutilisable, remplaçable, élastique (souple, adaptable), modifiable ».

Critères généraux : « Tout bien logiciel destiné à être réutilisé doit respecter certains critères généraux » (*pour plus de détails, voir en page 10 du document*) :

- « **Conformité aux standards** » (normes et directives, externes et internes),
- « **Application du processus d'ingénierie** » (…« rédaction de la documentation adéquate », « procédures de tests et de validation »…),
- « **Complétude** des produits logiciels et des informations fournies »,
- « **Simplicité et compréhensibilité** »,
- « **Modularité** ».

Critères techniques : « Afin d'être facilement réutilisés et intégrés dans une nouvelle application, les biens logiciels doivent être mis en œuvre en tant que composants, c'est-à-dire se conformer à certaines caractéristiques techniques favorisant leur assemblage. Ils doivent respecter les critères techniques suivants » (*pour plus de détails, voir en page 11 du document*) :

- « **Interopérabilité** »,
- « **Portabilité** » (…en particulier l'accès par des « interfaces logiques » permet « d'isoler les composants métiers à forte valeur ajoutée des technologies d'implémentation »…),
- « **Séparation de l'interface et de l'implémentation** » (« les services offerts par un bien logiciel et la manière dont il est implémenté » peuvent « évoluer séparément » et ne doivent « pas être mélangées »),
- « **Composition** » (« un composant peut résulter de l'assemblage de plusieurs composants de granularité plus fine », respectant alors les propres règles de « chaque niveau d'imbrication »),
- « **Auto-descriptivité** » (un bien logiciel doit « inclure sa propre documentation ». « Un composant logiciel exécutable se doit de décrire sa propre interface, c'est-à-dire son protocole d'utilisation », donnant ainsi moyen à l'utilisateur « de l'assembler avec d'autres composants »),
- « **Transparence** de localisation »,
- « **Sécurité** »,
- « **Prêt à l'emploi** ».

Méta-modélisation : « Si on ajoute à un corps de bien logiciel, **un modèle décrivant l'architecture et la conception** d'un bien logiciel, on facilite sa compréhension et son intégration dans un système logiciel ».

Des classifications des biens logiciels [(^)]

Biens logiciels verticaux et biens logiciels horizontaux sont associés à des motifs différents de réutilisation (*pour plus de détails, voir en pages 9 et 10 du document*) :

- Les **biens logiciels verticaux** sont « spécifiques à un **domaine fonctionnel** ». *« Un domaine fonctionnel, ou domaine d'application, se définit comme un ensemble d'applications (existantes ou futures) participant à la même activité d'une entreprise ».*
  Les « biens logiciels horizontaux » sont « plus faciles à identifier et à réutiliser car ils représentent des éléments récurrents d'**architecture technique** ». Dans les biens logiciels horizontaux, on peut distinguer :
- Les **biens logiciels horizontaux** qui sont « nécessaires techniquement pour implémenter les couches basses (communications, entrées/sorties, accès au système d'exploitation…) d'un système logiciel ».
- Les **biens logiciels horizontaux** « **génériques** qui fournissent des services à plus d'un domaine fonctionnel (objets graphiques d'IHM, service d'authentification, framework de gestion des erreurs) ».

Types de réutilisation d'un bien logiciel : Il est défini 4 types de réutilisation en fonction de la manière dont le bien logiciel est récupéré, des conditions d'accessibilité et de modification de l'information (*pour plus de détails, voir en page 10 du document*) :

- Réutilisation de type « **boîte noire** ».
- Réutilisation de type « **boîte blanche** ».
- Réutilisation de type « **boîte grise** ».
- Réutilisation de type « **boîte en verre** ».

Types de biens logiciels réutilisables : Il est défini et décrit différents types de biens logiciels en fonction de la forme que prend le bien (bibliothèque, exécutable…) et de son objet (*pour plus de détails, voir en pages 14 et 15 du document*) :

- « **Composant exécutable** ».
- « **Bibliothèque** ».
- « **Pattern** ».
- « **Framework** ».
- « **Modèle métier** ».
- « **Modules logiciels** ».
- « **Générateur d'application** ».
- « **Modèles d'exigences** ».
- « **Référentiel produit** ».

Référentiel logiciel, bibliothèque de biens logiciels [(^)]

Bibliothèque [(^)]

« En rendant ses logiciels réutilisables, une *entreprise* accroît la valeur de son patrimoine en le rendant pérenne ». Son patrimoine logiciel prend « la forme d'une **bibliothèque** de biens logiciels », parfois aussi appelée « **référentiel** logiciel (en référence au terme anglais correspondant Repository) ».

« La bibliothèque de biens logiciels » offre :

- « Un endroit connu et unique où chercher et déposer les biens logiciels ».
- « L'identification et l'inventaire des biens logiciels ».

- « Un mode homogène de documentation, de recherche et d'administration des biens logiciels ».
- « Une méthode de gestion des évolutions et des améliorations apportées aux biens logiciels, en particulier les procédures de gestion de configuration ».

Catalogue [(^)]

La bibliothèque contient « les biens logiciels (composants) » et aussi « un **catalogue** », qui « associe une classification (taxonomie) permettant de « ranger » logiquement les biens logiciels ».

« Il faut pouvoir agréger les catalogues de composants » (interne, externes) « ou bien encore centraliser l'interface de recherche des composants ». « Ainsi on peut par rapport à un besoin, rechercher un composant, comparer ceux existants et choisir la meilleure solution : acquérir, créer de rien ou récupérer d'une application existante le composant désiré ».

Fonctions de la bibliothèque [(^)]

« Les fonctions de la bibliothèque » sont :

- « Publier les biens logiciels disponibles ».
- « Parcourir le catalogue ».
- « Récupérer un bien logiciel ».
- « Classification et recherche (recherche textuelle, par mot clé, par attribut, thésaurus) ».
- « Archivage et versioning ».
- « Fournir des rapports, des indicateurs sur l'utilisation des biens logiciels stockés ».
- « Contrôler l'accès (droits d'accès en lecture, écriture, suppression) ».
- « Travail collaboratif (mécanismes de workflow pour la validation) ».
- « Notification des modifications aux utilisateurs abonnés ».

Des solutions de la réutilisation [(^)]

Patterns, frameworks et composants [(^)]

« Les patterns, les frameworks et les composants peuvent être complémentaires. Les patterns peuvent être caractérisés comme des descriptions plus abstraites des frameworks, qui sont eux implémentés dans un langage particulier. Des frameworks sophistiqués incorporent de nombreux patterns. De la même manière, les patterns peuvent servir à documenter les frameworks. Aussi les frameworks peuvent être utilisés pour développer des composants. »

« L'intégration de biens logiciels réutilisables (utilisant une architecture à base de composants) est synthétisée et récapitulée dans une démarche élaborée et promue par l'OMG (Open Management Group) : il s'agit de la **MDA** (Model Driven Architecture). »

Les architectures à base de composants [(^)]

Voir les pages 15 et 16 du document.

Développement et exécution des systèmes à base de composants : Le « développement de l'application » se rattache à la notion d'« assemblage ». Le développeur d'application construit « l'application en assemblant des composants logiciels existants ». Il fournit « l'architecture d'intégration, alors que la conception et l'implémentation de chaque composant sont déjà fixés ». Le « déploiement » et « l'exécution » de l'application se rattachent eux à la notion de « transaction ». « Le

principe de dissociation des programmes clients, des serveurs et des services permet de créer un environnement d'exécution propre aux applications sans avoir à retoucher les objets et les composants sur lesquels elles s'appuient ».

Technologies : « Des technologies permettant d'élaborer des architectures à base de composants » : « COM / DCOM / ACTIVE X / .NET, EJB, CORBA et les Web Services ».

Composant réutilisable : « Les composants sont des unités de programmes disponibles sous forme binaire. Ils sont réutilisables comme des boites noires en proposant une interface prédéfinie qui expose ses propriétés et ses méthodes (ainsi que les événements pour les EJB). La granularité des composants peut varier de petits objets graphiques d'IHM jusqu'à des applications complètes, comme des traitements de texte. »

Autonomie du composant réutilisable : Un composant est « dissocié de son entourage », il est « constitué d'un objet ou d'un groupe d'objets « autosuffisants » ». Il « fournit un ensemble de services à une application cliente, mais n'a pas de dépendance externe (sauf avec le serveur d'application) ».

Livraison d'un composant réutilisable [(^)]

Un composant doit être livré avec :

- Un « paquet contenant ses propriétés, méthodes et événements associés »,
- Un paquet « décrivant les services qu'il requiert » : « des données supplémentaires ayant trait cette fois aux services (annuaires, sécurité, transaction, persistance…) qu'il requiert pour son exécution ».

« Ces deux paquets de données supplémentaires qui s'ajoutent au code exécutable illustrent une approche déclarative de l'assemblage de composants logiciels. »

Des designs patterns, modèles de conception [(^)]

Voir les pages 16 à 18 du document.

Pattern : « Un pattern documente un savoir-faire réutilisable, concernant un problème récurrent. Il est souvent indépendant d'un langage de programmation ou des détails de mise en œuvre. Cependant il ne décrit pas la structure complète d'un système mais seulement une vue de la structure qui a un rapport avec un problème particulier. L'intégration de différents patterns doit être faite au moment du développement ». « Les patterns permettent une abstraction de haut niveau ».

Ensemble de patterns : « Un assemblage structuré de design patterns pour un domaine d'application spécifique se nomme **langage de pattern** ». « On peut aussi retrouver les patterns structurés dans un **catalogue**. Le catalogue est classifié en domaine (domaine de classe et domaine objet) et par rôle » (un pattern peut avoir un rôle « créateur », « structurel », « comportemental »).

Livraison d'un pattern [(^)]

« La définition la plus générale d'un pattern décrit le **contexte** d'application du pattern, le **problème** de conception qui est traité et une **solution** validée par l'expérience. Certains auteurs décrivent également les avantages et les inconvénients du pattern ainsi qu'un exemple d'application sur un cas particulier. »

Un pattern est « composé :

- d'un nom et de l'énoncé de son but principal,
- d'un problème et son contexte,
- des points forts adressés,

- de la solution, constituée de la description abstraite de la structure et des collaborations entre ses composants,
- des conséquences négatives et positives de son utilisation,
- une démarche de mise en œuvre et du code source en exemple,
- ses utilisations connues et des patterns apparentés ».

Les design patterns contribuent à la **compréhension des systèmes** : Les design patterns aident « à documenter des systèmes et améliorent ainsi la compréhension des systèmes ». « Les patterns capturent explicitement la connaissance des experts et rendent ce savoir plus partageable. Ils permettent une meilleure communication entre développeurs : les noms de pattern forment un vocabulaire ».

Les frameworks [(^)]

Voir les pages 18 à 20 du document.

Framework : « Un framework comporte à la fois du **code** et une **architecture** réutilisable ». « Un framework est un ensemble intégré de composants qui collaborent pour fournir une architecture réutilisable pour une famille d'applications similaires. Dit autrement, un framework est un ensemble intégré de produits logiciels réutilisables, extensibles et personnalisables **pour un domaine d'application spécifique**. Un framework objet est constitué de classes et de relations décrivant des structures d'objets coopérants. »

Utilisation, formes d'utilisation du framework : « Des applications concrètes sont générées en adaptant les éléments variables du framework. Il y a principalement deux façons d'adapter un framework » (« ces deux formes d'utilisation sont fréquemment disponibles dans le même framework ») :

- « La personnalisation **boite noire** » (« *instanciation* ») : « le framework fournit une implémentation spécifique de chaque élément variable. Le développeur génère une application en faisant un choix (paramétrage) parmi les différentes implémentations fournies par le framework ».
- « La personnalisation **boite blanche** » (« *héritage* ») : « le framework fournit une implémentation générique de chaque élément variable, que le développeur spécialise pour son application. Cette forme de spécialisation est plus flexible, mais demande une très bonne compréhension du domaine traité, lors de la conception et de l'implémentation du framework ».

Types de frameworks : On distingue « deux types de framework » :

- « Les **frameworks d'application** (ou **verticaux**) » : « des applications semi-complètes et adaptables (les problèmes de conception et d'analyse caractéristiques du domaine d'application y ont déjà été résolus) ».
- « Les **frameworks techniques** (ou **horizontaux**) » : offrent « des services particuliers (service de persistance, de sécurité, de gestion de la concurrence d'accès par exemple) ».

Compréhension du framework : « Souvent les frameworks sont construits en suivant plusieurs design patterns. La connaissance de ces patterns permet de mieux comprendre le fonctionnement du framework. »

Ingénierie de domaine, modèle de connaissance [(^)]

« Pour produire des biens logiciels réutilisables, l'*entreprise* doit investir dans l'**ingénierie de domaine**. L'ingénierie du domaine consiste à définir une architecture, à analyser les besoins et à développer du logiciel, en vue, non pas d'une seule application, mais d'une famille d'applications (existantes, à

construire ou potentielles). L'**analyse du domaine** fait partie du processus d'ingénierie du domaine. Cette dernière consiste à identifier les objets et les opérations d'une classe de systèmes similaires appartenant à un domaine particulier ».

Le fait de chercher à expliciter un logiciel « partage une grande partie de la problématique de la **gestion de la connaissance** où l'on doit tenter de joindre à l'analyse des éléments de connaissances (le code source) des éléments de **connaissance du domaine** (i.e. **ontologies**, **taxonomies**). »

« Les solutions proposées [pour décrire un **modèle de connaissance**] font appel aux méthodes des sciences cognitives (**ontologies** principalement, **classification**, **dictionnaires** du domaine) ». Il s'agit aussi d'utiliser « un langage de définition d'architecture complet et reconnu ».

Organisation et pratique effective de la réutilisation [(^)]

Stratégie de réutilisation [(^)]

La réutilisation fait « l'objet d'une stratégie » établie à partir de ce qui est déterminé concernant les « buts de la réutilisation », les « objectifs à atteindre » et les « moyens » attribués à l'activité de réutilisation.

Production et consommation de biens logiciels réutilisables [(^)]

« S'organiser pour produire un composant logiciel réutilisable est plus lourd que s'organiser pour réutiliser un composant produit par une autre entité ».

Le fait d'utiliser un composant logiciel produit par une autre entité peut rendre l'entité consommatrice dépendante de l'entité productrice. Le degré de **dépendance** varie en fonction du type de réutilisation (boîte noire, blanche, grise, en verre), du type du bien logiciel (composant exécutable, bibliothèque, modules logiciels…).

Toucher un bien réutilisable peut avoir un impact sur tous les logiciels qui s'en servent. Aussi, pour être le moins tributaire possible des choix de l'entité productrice, l'entité consommatrice a besoin de garanties quant aux conditions dans lesquelles ont lieu la réutilisation : la distribution du bien logiciel s'inscrit-elle dans un processus stable et permanent, quel est l'engagement de distribution de l'entité productrice vis-à-vis des versions existantes du bien logiciel (lorsqu'une nouvelle version sort), les conditions de distribution risquent-elles de changer plus tard, quelle est la stabilité de l'entité productrice, etc.

Pratique effective de la réutilisation [(^)]

« Concrètement, l'*entreprise* doit mettre en œuvre et gérer un référentiel de biens logiciels ». Il faut donc impliquer « les ressources nécessaires pour insérer les biens logiciels dans le référentiel logiciel et maintenir à jour cette base d'informations ». Il faut aussi que « tous les producteurs et consommateurs de biens réutilisables dans l'*entreprise* respectent les procédures qui auront été élaborées pour que le système soit géré correctement. C'est une approche un peu similaire à une approche « qualité » dont les bénéfices et les contraintes sont bien identifiés ». « La production de biens logiciels réutilisables (composant, framework, pattern) est très difficile et complexe. En tout cas, le développement, en tenant compte de sa réutilisabilité, d'un composant demande un effort de développement plus important ».

« Une stratégie de réalisation correspond à un projet d'*entreprise* de **long terme** et se mène sur plusieurs années ». Elle doit être « accompagnée d'une évaluation » (des moyens, objectifs, résultats obtenus), afin de valider la stratégie de réutilisation suivie et d'adapter le projet en cours de réalisation ».

Unification des biens logiciels : « Une vision unifiée du patrimoine logiciel de l'*entreprise* nécessite l'intégration des différentes applications de l'*entreprise*, au moins au niveau de la conception ». Il s'agit d'éviter ou du moins d'étudier les redondances et duplications.

Patterns : « L'intégration des patterns dans un processus de développement logiciel est une activité à haute intensité humaine car elle nécessite aujourd'hui beaucoup de savoir-faire ».

Frameworks : Les frameworks « nécessitent une courbe d'apprentissage initial importante (beaucoup de classes, beaucoup de niveaux d'abstraction) ».

Méthode pour mettre en œuvre la réutilisation [(^)]

Voir les pages 12 à 15 du document.

« La méthodologie **REBOOT** fournit un support complet pour la réutilisation. Une autre méthodologie pour la réutilisation logicielle est proposée par Jacobson, Griss et Jonsson et présentée dans leur ouvrage intitulé « **Software Reuse** » ».

Mise en référence, gestion de version (gestion de configuration) [(^)]

« En évitant la redondance (ou la duplication) de programme, on évite la maintenance sur les multiples « copies » de fonctions égarées à droite et à gauche. Ce sont des réductions de coût de maintenance à court terme qui peuvent être évitées. »

Tests logiciels [(^)]

Tests, fiabilité et validation logicielles : Un référentiel dont les biens logiciels contiennent leurs « cas de test » « permet d'automatiser la validation des différentes versions d'un produit, et en particulier de garantir la **non régression** ».

Test des design patterns : « Il est difficile de juger de la qualité d'un pattern. Les patterns sont jugés par l'expérience et la discussion plutôt que par les tests automatisés (les composants génériques sont difficiles à valider dans l'abstrait) ».

Tests des biens logiciels réutilisables : Pour « les applications réalisées à partir de pattern, framework ou d'assemblage de composants », les « tests peuvent s'avérer plus difficiles à mener que pour des développements complets à partir de zéro où l'équipe de développement maîtrise l'intégralité du code ».

Outils et technologies pour la réutilisation [(^)]

Outil de mise en œuvre [(^)]

« La réutilisation doit s'intégrer au cycle de développement classique du logiciel (conception, développement, test, maintenance). Ainsi, le processus logiciel fait référence à des activités comme l'ingénierie de produit, la gestion de projet ou groupware, le développement (AGL) et la maintenance ou la gestion de la configuration ». « *L'entreprise* doit disposer d'un outil permettant d'assembler les composants (parfois appelé framework d'intégration) tout en gérant le référentiel logiciel ».

Un panorama d'outils, méthodes, environnements technologiques existants [(^)]

Voir les pages 27 à 34 du document.

- « Les web services et l'architecture « .Net » de Microsoft ».
- « Les ateliers de Génie Logiciel (CASE) et leurs outils de ré-ingénierie ».
- « Logiciels libres et composants logiciels ».
- Les normes ISO du développement logiciel, « les écoles de programmation : Rational Unified Process, Extreme Programming, Méthode AGILE ».

Concepts et standards de réutilisation [(^)]

La réutilisation a été portée par « l'apparition de la programmation et de la conception orientée objet. On peut penser que lorsque l'adoption des concepts de la programmation objet sera largement effective la réutilisation se diffusera. Cependant, les web services, avec XML, représentant un nouveau standard d'architecture à base de composants, on est confronté au problème du nombre de standard trop important et cela nuit à une bonne utilisation et diffusion des techniques de réutilisation ».

Technologies, interopérabilité et obsolescences [(^)]

Le référentiel logiciel doit prendre en compte le fait que « même au sein d'une *entreprise*, on voit se côtoyer des architectures et des environnements hétérogènes (différents systèmes d'exploitations, différents environnements de développement…) ». D'autre part « des biens logiciels, réutilisables à un moment, deviennent obsolètes le jour où la technologie qui le sous-tend est abandonnée ».

Rythme rapide d'innovation dans le domaine informatique [(^)]

« La programmation orientée objet a été un vecteur majeur de l'introduction de la réutilisation dans les pratiques informatiques. Cependant elle a rendu obsolètes (ou moins attractive) d'autres langages et tous les outils qui vont avec ». « La programmation orientée aspect amène une vue pratique supplémentaire, mais elle nécessite d'adapter les outils de conception et de ré-ingénierie. XML, Internet et les web services sont là encore des techniques qui nécessitent l'adaptation de l'ensemble des outils du génie logiciel ». « Ces changements très fréquents et d'importance, parfois conjoints, nuisent donc à une maturation des procédés logiciels, qui, s'ils s'améliorent, sont toujours aussi complexes et retardent l'adoption généralisée des techniques de la ré-ingénierie et de la réutilisation logicielles »