Génie Logiciel – Livre Blanc

Version 0.2 – 26 Octobre 2011

Xavier Blanc – *Xavier.Blanc@labri.fr*

Partie I : Les Bases

Sans donner des définitions trop rigoureuses, il faut bien commencer ce livre par énoncer les principaux concepts du génie logiciel. Cette discipline emploie en effet un nombre non négligeable de termes précis qu'il faut absolument connaitre.

Cette première partie présente une vision générale du génie logiciel et donne des définitions informelles pour chacun de ses concepts.

Construction de logiciel

Avant toute chose, il est absolument fondamental de savoir ce qu'est un **logiciel**. D'une manière assez abstraite, on peut dire qu'un **logiciel** s'exécute sur un ordinateur et traite automatiquement de l'information.

Nous connaissons tous des logiciels. Les traitements de textes, les tableurs, les jeux vidéo, les agendas, les moteurs de recherche, les sites sociaux ou les outils de dessins sont tous des logiciels. Ils s'exécutent tous sur un ou plusieurs ordinateurs et traitent de l'information.

Pour autant, s'il est facile de donner une définition abstraite et des exemples de logiciels, il est bien plus compliqué de proposer une définition concrète qui délimite ce qu'est un logiciel et ce que cela n'est pas. En outre, est-ce qu'une version préliminaire d'un logiciel est un logiciel ? Est-ce que le code d'un logiciel est un logiciel ? Est-ce que la documentation fait partie du logiciel ?

Pour répondre facilement à ces questions, nous considérons qu'un logiciel est une entité informatique composée d'un ou plusieurs fichiers (code source, binaire, librairie, documentation, guide d'installation, etc.). Parmi ces fichiers, certains peuvent être exécuté par un ordinateur.

Dans l'ensemble de fichiers qui composent un logiciel, il n'existe pas de fichier obligatoire. En fait, cela dépend de l'état d'avancement du logiciel. En effet, sans code source il n'est pas réellement possible d'exécuter un logiciel. Le code source d'un logiciel n'est pourtant pas présent au début de la vie du logiciel ni même lors de la livraison.

Une fois cette définition posée, même si très simplificatrice, il est aisé de comprendre que le but du génie logiciel est de définir la façon dont construire l'ensemble des fichiers qui caractérisent un logiciel. Autrement dit, le génie logiciel répond aux questions suivantes : Comment construire le code source ? Comment construire les binaires ? Comment construire la documentation ? etc.

*Il n'est pas rare d'utiliser plusieurs autres termes tels que **programme**, **application** ou **produit** à la place du terme **logiciel**. Je trouve que ces termes sont des synonymes.*

Vie du logiciel

Si un logiciel est caractérisé par un ensemble de fichiers, on peut considérer qu'il naît lorsque que le premier fichier de l'ensemble est construit, et qu'il meurt lorsque le dernier fichier de l'ensemble est supprimé.

Curieusement cette présentation de la vie d'un logiciel n'est absolument pas dénuée de sens. En effet, la **naissance d'un logiciel** coïncide avec le moment où une personne émet le souhait de disposer du logiciel. En considérant que cette personne exprime son souhait dans un fichier, un simple fichier texte attaché à un mail par exemple, alors cela correspond à la création du premier fichier de l'ensemble. Pour ce qui est de la mort d'un logiciel, un **logiciel meurt** lorsque la dernière copie du dernier fichier de l'ensemble est supprimée.

Cette présentation de la vie d'un logiciel a un autre avantage. Elle illustre clairement le fait qu'il n'existe pas de terme consacré pour parler du moment où le logiciel devient utilisable. Comme nous le verrons dans les sections suivantes, on peut parler de **déploiement** ou **d'installation** même si ces termes ne sont pas complètement adéquats.

Client et Utilisateur

Si un logiciel traite automatiquement de l'information, c'est parce que ce traitement intéresse quelqu'un. Un logiciel a pour vocation de servir des utilisateurs. **L'utilisateur** d'un logiciel est une personne qui interagi avec le logiciel car elle a besoin des services rendus par celui-ci.

Lors de la naissance du logiciel, l'utilisateur est donc la personne qui émet ses souhaits quant aux services que devra rendre le logiciel. Le terme consacré est **exigences**. Un utilisateur exprime donc ses exigences sur le futur logiciel.

Le terme **Client** est souvent employé pour désigner la personne qui exprime les exigences. L'utilisation du terme **Client** plutôt que du terme **Utilisateur** vise à introduire la notion marchande du logiciel. Le client est la personne qui paye la construction du logiciel.

J'emploie indifféremment les termes Client et Utilisateur. Je considère que la distinction entre ces deux termes n'est pas fondamentale en ce qui concerne le génie logiciel. Notez que les anglophones utilisent le terme Stakeholder pour désigner indifféremment le client et l'utilisateur. Dans le reste de ce livre, j'emploierais uniquement le terme Utilisateur afin de ne pas créer de confusion.

Si l'utilisateur exprime ses exigences sur le futur logiciel, c'est pour que celui-ci soit développé. Plusieurs termes existent et permettent de catégoriser les différents métiers nécessaires au développement du logiciel. Le terme **développeur** est le terme le plus générique. Un développeur est une personne qui participe au développement du logiciel.

Les termes maitrise d'ouvrage (MOA) et Maîtrise d'œuvre (MOE) sont très souvent employés dans l'industrie et dans le tertiaire. La **MOA** désigne la société utilisatrice alors que la **MOE** désigne la société qui prend en charge les développements. Le terme métier est parfois utilisé pour désigner l'utilisateur. Une personne du **métier** est un utilisateur, souvent non informaticien, qui utilise le logiciel dans son métier de tous les jours.

Le plus grand problème du génie logiciel est de faire en sorte que les développeurs construisent un logiciel qui répond parfaitement aux exigences des utilisateurs. C'est un problème de traduction.

Comment traduire les exigences dans des fichiers qui constituent un logiciel ? Autrement dit, **comment rendre les exigences des utilisateurs interprétables par un ordinateur** ?

Il est important de noter que le génie logiciel s'adresse principalement aux développeurs et pas aux utilisateurs. Comme nous le verrons, les exigences des utilisateurs sont principalement exprimées en langage naturel. La traduction vers des fichiers interprétables par un ordinateur est donc à la charge du développeur. Le génie logiciel a pour objectif de professionnaliser les développeurs et propose des solutions afin de faciliter ce travail de traduction.

Développement d'un logiciel

Nous avons vu que la vie d'un logiciel commence dès qu'un utilisateur exprime ses exigences. En génie logiciel, le terme consacré pour cette activité est **expression des exigences**. Le terme anglais est requirement pour parler des exigences. L'expression des exigences est à la charge de l'utilisateur qui a pour objectif de lister toutes ses exigences, de vérifier leurs compatibilités et de fixer les priorités entre elles. Le **cahier des charges** est le document d'aboutissement de cette activité. Le cahier des charges contient toutes les exigences exprimées par un utilisateur.

Dès que des exigences sont exprimées, le développement du logiciel commence véritablement. Trois activités sont alors réalisées par les développeurs :

1 – **L'analyse du besoin** (**Analysis**) : Lors de cette activité, les développeurs ont pour objectif de bien comprendre les exigences des utilisateurs afin de savoir quel logiciel ils peuvent développer. Cette activité permet de spécifier les contours du logiciel, de définir les services qu'il devra fournir mais aussi les services qu'il n'a pas à fournir parce que ses utilisateurs n'en ont pas besoin ou parce qu'ils sont trop couteux à développer. C'est en réalisant l'analyse que les développeurs peuvent estimer le **coût de développement** du logiciel. A l'issue de l'analyse du besoin, les développeurs peuvent rédiger les **tests de validation**. Ces tests sont des scénarios d'utilisation du logiciel. Ils expriment ce que fera le logiciel. Ces tests sont aussi appelés des **tests de recette** car si le logiciel passe ces tests lorsqu'il sera développé, alors l'utilisateur pourra payer le développement.

2 – La définition de **l'architecture** ou **conception** (**Design**): Cette activité permet aux développeurs de prévoir l'organisation du logiciel en différents modules de code. L'organisation en modules à deux objectifs. Le premier vise à définir les qualités du logiciel en termes de performance, de tolérance aux pannes ou de réutilisabilité par exemple. Le deuxième vise à préparer la séparation des taches de développement afin de diminuer les délais de développement. Pour autant, lorsque les différents modules seront développés, il faudra alors les intégrer afin de constituer le logiciel. La conception doit alors absolument anticiper tous les problèmes d'intégration. A cet effet les **tests d'intégration** sont rédigés pendant la conception. Ils ont pour objectif de spécifier la manière dont les différents modules pourront communiquer et donc être intégrés.

3 – La **réalisation de code** : Cette activité consiste à réaliser le code source du logiciel. C'est lors de cette activité que des choix de codage sont effectués (choix des bibliothèques, choix des conventions de codage, etc.). Les **tests unitaires** sont effectués lors de cette activité. Un test unitaire vise à tester un élément atomique d'un code (une classe pour les langages objet).

Une **méthode de développement** précise la façon dont ces trois activités doivent être réalisées. Historiquement, les première méthodes préconisaient de réaliser ces trois activités

séquentiellement, les unes après les autres. Il fallait commencer l'analyse du besoin, s'attaquer à la conception une fois l'analyse terminée et enfin commencer la réalisation de code dès que la conception avait bien délimitée les modules ainsi que la séparation des taches.

Aujourd'hui plusieurs méthodes existent. La plupart d'entre elles sont itératives. Elles préconisent la réalisation de plusieurs boucles de développement : analyse, conception, codage, analyse conception, codage, etc.

Production et maintenance

Les personnes qui construisent le code source d'un logiciel considèrent trop souvent que logiciel est fini lorsque le code source est écrit. Pour les personnes qui travaillent à l'exploitation du logiciel, c'est au contraire à ce moment que le logiciel prend réellement vie. C'est en effet à ce moment que le logiciel est utilisable et que la moindre anomalie ou la moindre panne doit être traitée.

Classiquement avant de déployer un logiciel sur un ou sur plusieurs ordinateurs, il faut effectuer un test de déploiement. La **pré-production** consiste à déployer le logiciel sur des ordinateurs afin de réaliser un test grandeur nature de son utilisation. Bien souvent, la pré-production utilise exactement les mêmes ordinateurs que ceux sur lesquels le logiciel sera exécuté. Si cela n'est pas le cas, ces ordinateurs se doivent d'être très similaires. Les **béta-test** sont effectués lors de la pré-production. L'objectif est de fournir le logiciel à des utilisateurs / testeurs. Ceux-ci remontent les anomalies qu'ils détectent lors de l'utilisation qu'ils font du logiciel. Le **stress-test** est aussi effectué lors de la pré-production. L'objectif est de chercher les conditions limites d'exploitation du logiciel (en termes de nombre d'utilisateur ou de fréquence d'utilisation).

Une fois la pré-production passée, le logiciel est alors mis en **production**. Lorsqu'il est en production un logiciel peut subir des pannes. Le rôle de la maintenance est d'assurer le traitement de ces pannes.

Qui n'a pas vu son logiciel préféré planter sans que l'on sache trop pourquoi ? Tout bon utilisateur qui se respecte sait qu'il faut parfois redémarrer la machine afin de pouvoir à nouveau utiliser son logiciel. Dans le cas de logiciel complexe, s'exécutant sur plusieurs machines, les plantages arrivent aussi. Il faut alors relancer le logiciel en s'assurant qu'aucune donnée n'a été perdue. C'est là une des taches réalisée par la maintenance.

La maintenance est à dissocier de la gestion d'évolution. En effet, la maintenance n'effectue aucune modification des services que rend le logiciel. En maintenance, il est possible de corriger des bugs du logiciel mais cela ne couvre pas l'extension du logiciel afin qu'il assure de nouvelles fonctionnalités. L'ajout de fonctionnalité est une évolution du logiciel.

Lorsqu'une demande d'évolution est exprimée par un utilisateur, il faut alors reprendre le cycle de développement. L'évolution doit être analysée. Son impact sur la conception doit être définit. Puis enfin l'évolution doit être codée.

On peut entendre les termes de gestion d'ano et de gestion d'évo. Les ano sont des anomalies. Les évo sont des évolutions.

Schéma récapitulatif

Le schéma suivant représente tous les termes que nous venons de présenter. Il propose une vue très générale du génie logiciel. Cette vision générale permet alors de mieux comprendre la diversité du génie logiciel.

Une compréhension globale permet aussi de mieux comprendre des aménagements particuliers. Prenons par exemple le cas de l'outsourcing qui a pour objectif de faire développer un logiciel à moindre coût. Dans une vision génie logiciel, l'outsourcing consiste à délocaliser les activités de codage. Pour ce faire, il faut alors bien préparer la conception mais aussi la pré-production. D'un autre côté, prenons le cas du cloud computing qui a pour objectif de diminuer les coûts d'exploitation d'un logiciel. D'un point de vue génie logiciel, l'objectif est de délocaliser l'activité production. Pour ce faire, il faut alors bien préparer la pré-production mais aussi et surtout, aménager le cycle de développement afin qu'il prenne en compte les contraintes de ce mode d'exploitation.