Licence d'informatique **année 2002**

Génie logiciel I **Cours de Anne Nicolle**

Chapitre 6 : L'ingénierie du logiciel

1. Le cycle de vie du logiciel

Il a été défini par une norme IEEE pour assurer la conformité du processus mis en œuvre dans toutes les entreprises avec les exigences des maîtres d'ouvrage.

Le cycle de vie du logiciel décrit l'organisation entre les processus à mettre en place pour le développement et la gestion des logiciels, du projet jusqu'à leur retrait. Chaque processus du cycle de vie peut être décomposé en une activité principale et des activités annexes. Nous ne verrons dans ce cours que les aspects conceptuels du cycle de vie, c'est à dire les activités qu'on trouve organisées dans toutes les méthodes de conduite de projets sans s'occuper de leur réalisation effective dans le temps et de leur répartition précise sur des équipes ou des individus. Chaque entreprise a sa méthode de conduite de projets qui organise les tâches dont nous parlons ici abstraitement (plannings, réunions d'équipes, rapports intermédiaires…).

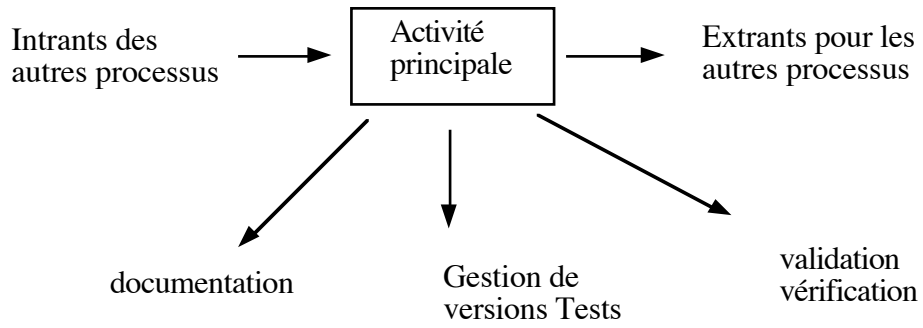

Figure 1

Les 8 étapes du cycle de vie sont :

- L'étude d'opportunité (avantages, risques)
- L'analyse du problème (quoi ?)
- L'architecture (comment ?)
- La conception d'une solution
- La réalisation technique (codage), les tests, et l'intégration des composants développés séparément
- L'implantation dans le lieu d'utilisation et les essais avec les usagers
- L'utilisation, la maintenance, l'évolution
- Le retrait

L'étude d'opportunité est du ressort des économistes de l'entreprise. Si elle est positive, les informaticiens commencent à intervenir à partir de l'étape d'analyse. Dans les deux étapes

suivantes, ils vont être à l'interface entre la technologie informatique et les usagers. les étapes 4 et 5 sont plus techniques. Les étapes 6 et 7 amènent à nouveau à être à l'interface entre la technologie et les comportements humains et sociaux. La 8ème étape relève de la décision des managers de l'entreprise mais doit être assumée par les informaticiens pour les conséquences que la décision de retrait a sur les autres éléments du système d'information et de communication.

Dans la suite, nous ne considèrerons que les étapes 2 à 7 du cycle de vie, dont la figure 4 présente l'organisation. Conduire un projet logiciel amène à connaître d'une part la technologie et d'autre part les sciences humaines (psychologie cognitive, psychologie de l'interaction, sémiotique, ergonomie) afin de concevoir des produits conformes à l'état actuel de la technologie et bien adaptés aux situations de travail et aux usagers. Ces compétences sont souvent le fait d'une équipe plus que d'un individu.

Le travail de l'analyste doit s'inscrire dans un cycle de réutilisation si on ne veut ni faire refaire constamment les mêmes choses, ni mettre à jour plusieurs fois les mêmes informations lors de l'utilisation

- l'approche base de données est une première méthode de réutilisation puisque les données sont ainsi disponibles pour tous les logiciels. Comme les données d'une entreprise sont plus permanentes que les traitements, il est normal d'organiser le système informatique d'une entreprise autour de sa base de données.

- l'approche objet est une méthode complémentaire, pour décrire de manière unique les objets du système. Mais attention, il peut y avoir plusieurs points de vue sur les mêmes objets, il faut avoir une vue large du système quand on définit les classes. Les bonnes classes, celles qui résistent le plus longtemps, sont souvent des classes abstraites qui définissent des structures de données qu on peut retrouver d'une application à l'autre sans que les objets se ressemblent.

L'analyse du problème n'est pas purement descriptive. Elle propose un modèle du monde dans lequel le logiciel prend place et un modèle du logiciel, qui doit être compréhensible pour les utilisateurs. La modélisation amène à représenter les objets du monde, leur relations, les processus auxquels ils participent. La méthode utilisée doit assurer la consistance entre le monde du problème, externe à la machine, et le monde interne de la machine. Elle doit permettre de bonnes interprétations : le modèle conceptuel est interprété par les personnes, le modèle computationnel est interprété par la machine. Ils doivent être cohérents. Dans le schéma de la figure 2, le trait horizontal sépare le monde du problème et le monde de la machine. On voit que lorsqu'une libraire de composants réutilisables est constituée, elle a un statut d'objet logiciel, mais aussi un statut d'objet du monde utilisable pour la conception du logiciel.

Pour gérer les modèles, on utilise la notion de package, représentée par un rectangle surmonté d'un petit rectangle à gauche portant le nom du package (figure 3). Un package peut contenir des sous packages. On représente par des flèches en pointillés les accès à un package et les importations de packages. Les packages du modèle correspondront à des dossiers contenant la documentation.

La méthode utilisée pour concevoir et développer le logiciel doit assurer la qualité des produits réalisés.

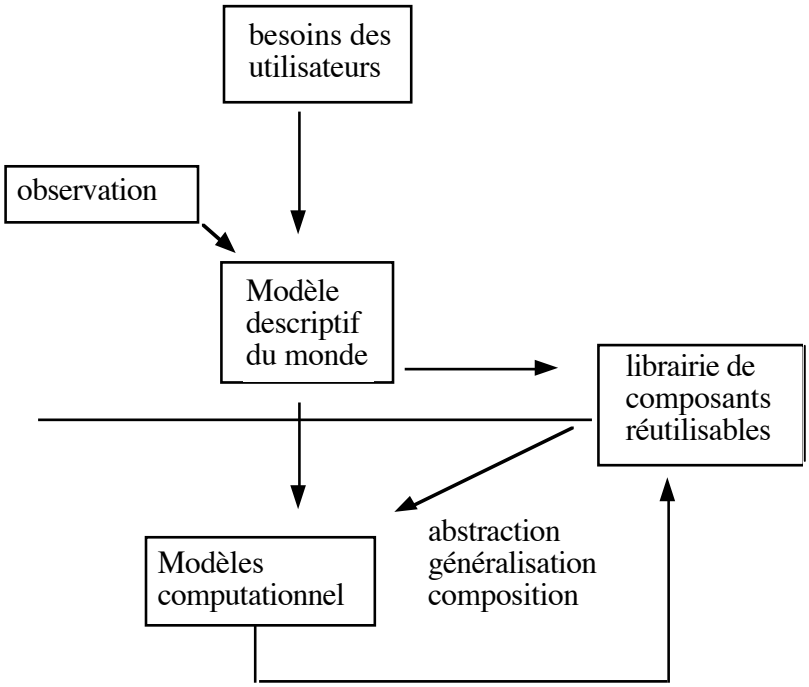

Figure 2

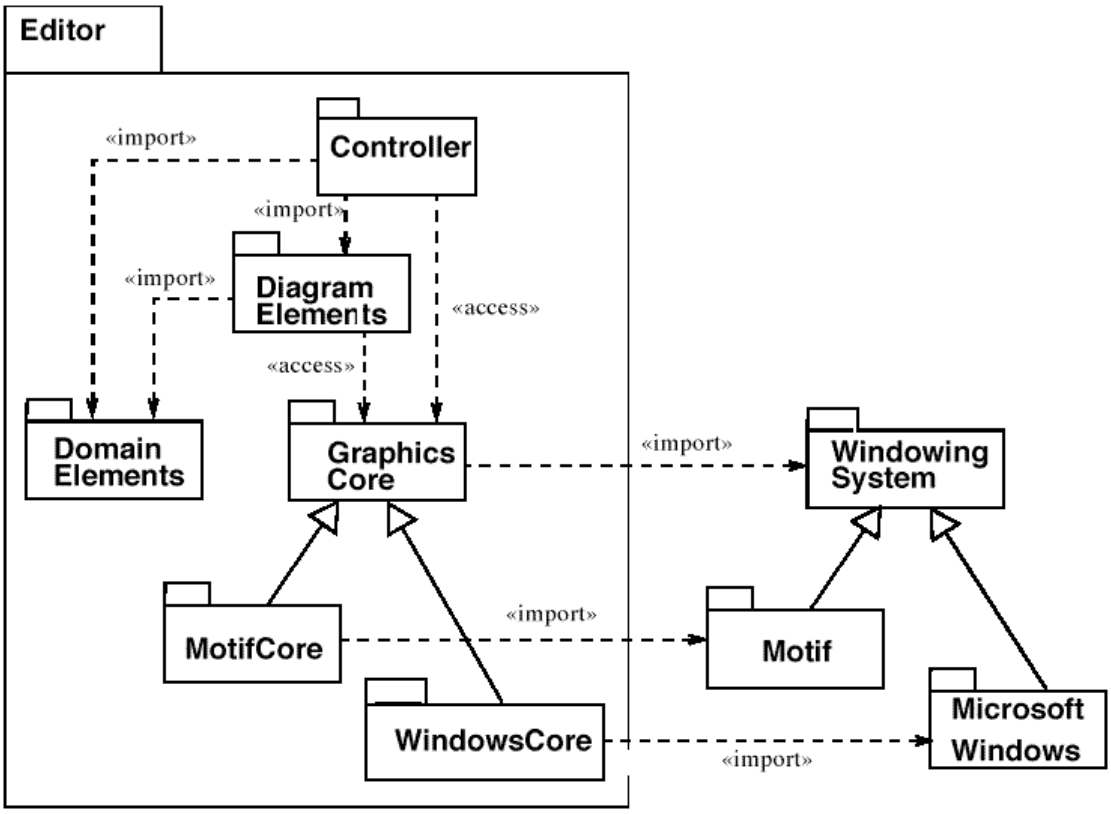

Figure 3

2. La qualité du logiciel

Qu'est-ce qui caractérise un bon produit ?

- la facilité et la qualité d'utilisation
- la facilité et le faible coût d'entretien et de réparation
- la facilité d'adaptation a de nouveaux besoins
- la durée de vie utile.

Ce sont des caractéristiques communes à tous les artefacts utilisés dans les sociétés humaines (véhicules, bâtiments, appareils ménagers...). Comment obtenir ces qualités ? C'est le domaine de l'ingénierie qui, pour chaque domaine d'activité, définit des critères généraux sur :

- la compréhension réelle des besoins et des exigences des utilisateurs.
- la représentation des besoins et des exigences à l'aide d'un modèle qui puisse être l'objet de discussions entre les commanditaires, les usagers et les ingénieurs.
- la représentation des solutions proposées avec un modèle compréhensible par les acteurs du projet et qui facilite la mise en œuvre (plans en architecture)
- la qualité de la documentation
- la qualité du produit obtenu (zéro défaut)

La principale difficulté de la conception est d'obtenir un modèle conceptuel de qualité avant d'entreprendre la conception et la réalisation du système car une étude de l' US Air Force a montré que

41 % des erreurs sont des erreurs de spécifications

28 % des erreurs sont des erreurs logiques

or plus une erreur a été commise tôt, plus elle a de conséquences et plus il est difficile de la réparer.

Si on représente sur un graphique les étapes de la conception, on obtient une approche en V, où les étapes de test sont mises en face des étapes de développement correspondantes. On voit ainsi que les étapes les plus précoces sont testées le plus tard. Il faut donc y faire très attention et chercher à obtenir la confirmation des choix faits à ces étapes sans attendre la phase de test correspondant.

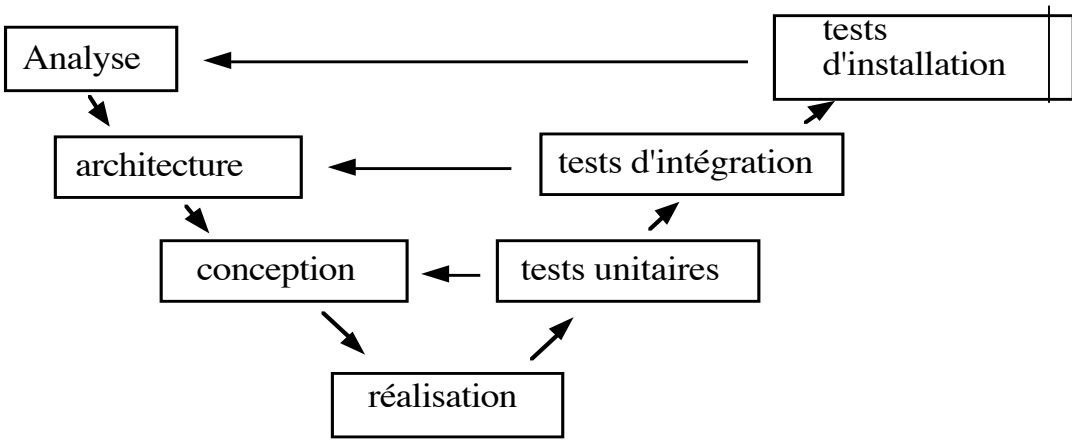

Figure 4

3. La spécification

Elle résulte de l'analyse du problème et définit les exigences et les préférences du client. Elle décrit le produit qui va être conçu et réaliser sans préjuger de sa solution. Elle constitue un contrat entre le commanditaire et le concepteur (cahier des charges). Ce contrat décrit les fonctionnalités du produit et les contraintes entre des choix possibles, qui devront être respectées dans la mesure du possible. Le produit peut être composé de logiciel seulement (ex : aide à la décision) ou de logiciel et d'un matériel spécialisé (ex : terminaux bancaires). On obtient donc le schéma de la figure 5 comme cadre de la spécification.

C'est au cours de la spécification qu'il faut concevoir les tests qu'on fera passer au programme pour s'assurer de sa conformité avec le cahier des charges. Une bonne spécification doit être :

- complète (il ne reste rien à décrire)
- sans ambiguïté (tous les acteurs du projet ont la même interprétation)
- correcte : vérifiée par l'usager
- compréhensible aussi bien par les informaticiens que par les usagers
- vérifiable sur le produit obtenu
- consistante : aucune des contraintes n'est en contradiction avec les autres
- indépendante de la technologie (mais pas du niveau de la technologie car elle doit être réalisable)
- traçable : on peut savoir où elle est réalisée dans le programme
- modifiable => modularité
- catégorisée par ordre d'importance pour le cas où des contradictions entre les contraintes seraient révélées à la conception, ou pour le cas où des solutions bien meilleures seraient possibles avec des modifications mineures.

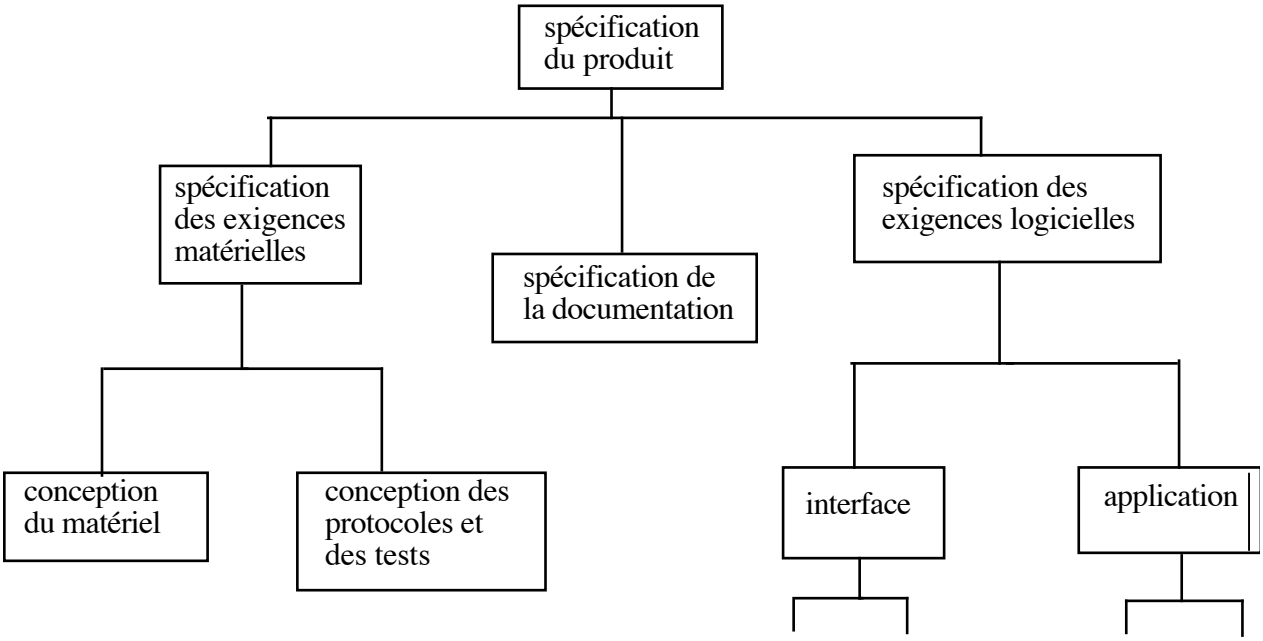

Figure 5

Il y a de plus en plus d'outils, appelés ateliers de génie logiciel (AGL) pour aider à faire les schémas correspondant à l'analyse du problème et pour aider à vérifier que la spécification

est bien respectée par la conception. Il existe une certification UML qui permet de savoir qu'un AGL respecte bien les normes imposées par le modèle objet et les schémas UML. Un AGL comporte un outil de dessin pour les diagrammes, des conventions d'écriture qui permettent de faire les liens entre les diagrammes, en particulier par les noms associés par l'analyste aux boites et aux flèches : le même nom dénote le même objet quand il est réutilisé dans plusieurs diagrammes. Un AGL comporte aussi des outils de vérification de cohérence et un interprète qui génère du code. Une partie du code sera généré automatiquement (définition des classes, signature des méthodes). Les schémas faits dans des étapes du début du projet seront réutilisés et développés dans les étapes suivantes. Les diagrammes doivent être complétés dans l'étape de réalisation pour indiquer si les variables sont publiques ou privées par exemple. La version des diagrammes faite pour la spécification doit cependant être conservée en vue de la documentation. Les fonctions peuvent parfois être écrites dans un langage de programmation existant (C++, Java) ou alors dans un langage d'expressions propre à l'atelier. Dans ce cas elles seront traduites dans le langage visé et ce n'est pas à recommander car on ne maîtrise plus la qualité du code.

4. la documentation

Une bonne documentation est la clé d'un usage facile du logiciel et des possibilités de maintenance et d'évolution. Elle doit être prévue dès le départ du projet. Si elle est bien faite, elle évite le recours à l'assistance téléphonique (hot-line). C'est un élément essentiel du confort des usagers et de la rentabilité de l'application pour l'entreprise qui la commercialise.

Elle comporte plusieurs parties indépendantes en fonction des types d'utilisateurs qu'elle vise :

- Pour les usagers :

- le manuel d'installation et de démarrage,
- le manuel d'utilisation sommaire avec des exemples d'interaction, qui permet de commencer à utiliser le logiciel sans passer du temps à lire tout.

En principe, ces deux manuels ne sont utilisés qu'une fois par chaque usager. Ils peuvent donc être dans un volume à part.

- le manuel de référence, qui documente de manière précise tous les objets et toutes les fonctions. Il doit comporter un lexique où tous les termes sont définis et un index associant à chaque notion les pages où elle apparaît, pour retrouver facilement chaque notion. Dans l'index, il est bien de distinguer (en italique par exemple) les pages où les notions sont définies de celles où elles sont utilisées.

La documentation usagers est souvent faite par une équipe extérieure à l'équipe de développement, qui va interroger les développeurs et les usagers pour faire le pont entre les deux logiques de travail.

- Pour les clients (maîtres d'ouvrage) :

- la spécification, en texte et en diagrammes
- les choix de conception,
    - structure concrète des données,
    - algorithmes,

• contrôle,

- comment et où les spécifications sont prises en compte,
- les tests passés et leurs résultats.

- Pour les développeurs :

- commentaires précis sur tous les objets du code. L'API Java est un bon exemple de documentation pour les développeurs car elle est organisée en parallèle avec l'organisation des répertoires et des classes. Si les commentaires sont écrits au bon endroit dans le code des classes, la documentation peut être générée automatiquement au format de l'API par Javadoc.

Les documentations papier, en particulier les manuels de référence, sont peu à peu supplantées par les documentations en ligne qui assurent de plus grandes possibilités de recherche grâce aux liens hypertextes et aux moteurs de recherches. Des assistants logiciels peuvent assister les usagers dans la réalisation de certains tâches en leur indiquant pas à pas ce qu'ils doivent faire et en s'assurant que chaque étape a réussi avant de proposer la suivante (aide de Word).

5. L'architecture

La définition de l'architecture d'un système informatique s'inscrit entre l'analyse du problème et la conception du logiciel. Elle fixe les choix technologiques sur lesquels la conception va s'appuyer. Certains choix technologiques sont fixés par le client, en fonction de ses contraintes, de ses préférences et aussi de l'homogénéité de l'ensemble de son système informatique. Par exemple, la DARPA aux USA a conçu et développé le langage ADA dans les années 70 pour mettre fin à l'anarchie qui régnait dans les langages de programmation utilisés dans ses services qui étaient plus de 300. Ensuite, ce langage était obligatoire pour tous les développements.

Dans cette phase, on fixe les machines et les réseaux sur lesquels le logiciel devra tourner, les conditions physiques dans lesquelles il sera utilisé : postes de travail, synchronisation des tâches, accès aux bases de données existantes. On fixe aussi le ou les systèmes d'exploitation hôte, les SGBD à utiliser, les langages de programmation, les grands principes à respecter pour les interfaces avec les usagers.

La stratégie de mise en œuvre peut alors être fixée : utilisation d'un progiciel, adaptation de programmes existants ou développement autonome. Avec le système Socrate de la SNCF, qui a pris comme stratégie la réutilisation et l'adaptation d'un programme de réservation aérien, on a pu voir l'importance des choix fixés à cette étape.

L'architecture matérielle fixe les machines et les réseaux qui seront utilisés. Il faut bien vérifier la viabilité des solutions envisagées par rapport aux pratiques des usagers, au débit des réseaux, à la vitesse et à la mémoire des machines dont on dispose, au débit des imprimantes, relativement aux délais dans lesquels les traitement doivent être faits.

L'architecture logicielle détermine les composants du système, leurs relations, les flots de données qui transitent entre les composants et la dynamique des processus à réaliser avec les contraintes temporelles à respecter. Elle fixe les relations entre l'application et les interfaces, la décomposition de l'application en modules autonomes et l'implantation des composants sur les

machines. La définition des relations que ces modules entretiendront est visualisée par les flots de données qui circulent entre les différents composants.

Plusieurs modèles d'architecture logicielle peuvent être envisagés :

- architecture monobloc quand les composants sont très liés, qu'ils ont beaucoup de relations dans les données qu'ils utilisent et dans les processus mis en œuvre,

- architecture pipe-line quand les composants peuvent être mis en ligne de façon que la sortie d'un composant soit l'entrée du composant suivant,

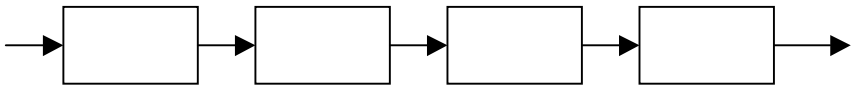

Figure 6

- architecture hiérarchique quand le composant de tête répartit les tâches entre les composants de niveau inférieur et fusionne les résultats quand ils sont obtenus,

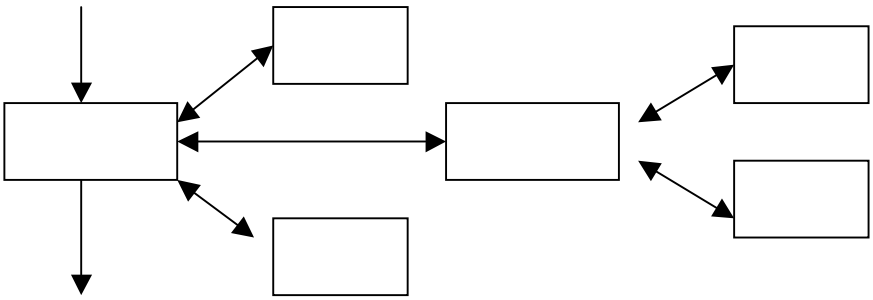

Figure 7

- architecture distribuée quand les composants travaillent de manière autonome, asynchrone, sans ordre prédéfini, au fur et à mesure que des données sont disponibles.

Les composants d'un logiciel distribué peuvent communiquer par des flots de données et des commandes ou par des événements. Quand le contrôle est géré par des événements, les composants d'abonnent à des événements et s'activent quand les événements qu'ils attendent se produisent. Les événements sont produits par des capteurs, par les usagers ou par certains composants quand ils arrivent dans un état déterminé.

Chaque composant d'une architecture peut lui même être décomposé en composants qui utiliseront un modèle propre d'architecture.

L'articulation entre l'architecture matérielle et l'architecture logicielle se traduit souvent par des schémas. Les diagrammes de composants (fig 3.81) décrivent la structure du logiciel. Chaque élément logiciel est représenté par diagramme de déploiement fig 3.82). Une fois l'architecture définie, on peut répartir le travail entre plusieurs équipes, fixer des délais et des points de contrôle de l'avancement des travaux.

6. Le prototypage

Pour modéliser des systèmes ouverts et interactifs, il faut bien séparer l'analyse du problème et la conception de la solution. Mais faire tester aux usagers des hypothèses de solution, en montrant des schémas, en faisant des prototypes, par des jeux de rôles où certains jouent le rôle de la machine, peut permettre de comprendre les situations et les besoins et d'affiner l'analyse du problème. L'important est de ne pas fixer trop tôt des éléments de solution alors que le problème n'est pas maîtrisé.

Lorsque le logiciel à développer ne relève pas d'un problème routinier, il est bon de valider les spécifications et les choix de conception le plus tôt possible en les mettant en œuvre dans un prototype. Le prototypage a pour but de vérifier la faisabilité et la viabilité des solutions envisagées. Il consiste à prendre des solutions très simples pour chaque composant logiciel afin de tester :

1) qu'il existe une solution,

2) que la spécification correspond bien aux attentes des usagers en terme de fonctionnalités et d'ergonomie,

3) que le comportement de l'ensemble de l'architecture est bien ce qu'on attend.

Dans cette étape, on ne traite que le cas général, pas les cas particuliers, on met seulement quelques cas dans les bases de données, on ne se préoccupe pas de l'efficacité, ni de la sécurité, ni de la portabilité.

Certaines activités peuvent être simulées dans un prototype, soit par des tirages aléatoires qui retournent des valeurs du domaine, soit par un compère qui joue le rôle d'un des composants logiciels. On teste ainsi souvent les spécifications des interfaces en faisant jouer le rôle de l'application par un compère. C'est la technique du Magicien d'Oz.

On distingue trois types de prototypes selon leur objectif principal :

1) Les prototypes exploratoires qui clarifient le problème à résoudre et qui sont utilisés pour l'analyse du problème :

"Programmez d'abord, vous penserez ensuite".

2) Les prototypes expérimentaux qui permettent de tester et d'améliorer l'ergonomie du travail et du logiciel sur les points essentiels.

3) Les prototypes évolutifs qui accompagnent les processus de spécification et de conception pour voir concrètement les implications des définitions et des choix.

Les prototypes peuvent servir à présenter des logiciels avant qu'ils soient réalisés pour les vendre, pour obtenir des contrats, pour permettre la réorganisation des postes de travail en vue de l'installation du produit fini. C'est un moyen de motiver les utilisateurs pour obtenir leur coopération afin de définir complètement les contraintes, les exigences et les préférences. Les prototypes dans un état de développement avancé peuvent être utilisés en simulation, en parallèle avec le fonctionnement antérieur des choses, pour voir si les résultats obtenus sont corrects.

Conclusion : Le cours de Génie logiciel de licence a permis d'étudier de façon approfondie les méthodes de modélisation des phénomènes qui permettent la conception des logiciels. Il a présenté de façon sommaire les étapes du cycle de vie du logiciel, la documentation et la conception des interfaces qui seront développées dans le cours de maîtrise. Ce cours abordera aussi la gestion des projets, la réalisation du logiciel et les tests, la maintenance et la réutilisation.