lors du développement des nouveaux systèmes, comme elle aide à anticiper les changements futurs.

1.5.3 Mage : Les Concepts

Un modèle Mage est composé de deux parties un génome et un phénotype. Le phénotype représente l'équivalent d'un système logiciel classique. Le génome est une collection de gènes qui façonnent continuellement le phénotype. La figure 1.1 résume l'approche de Mage vis à vis de l'évolution.

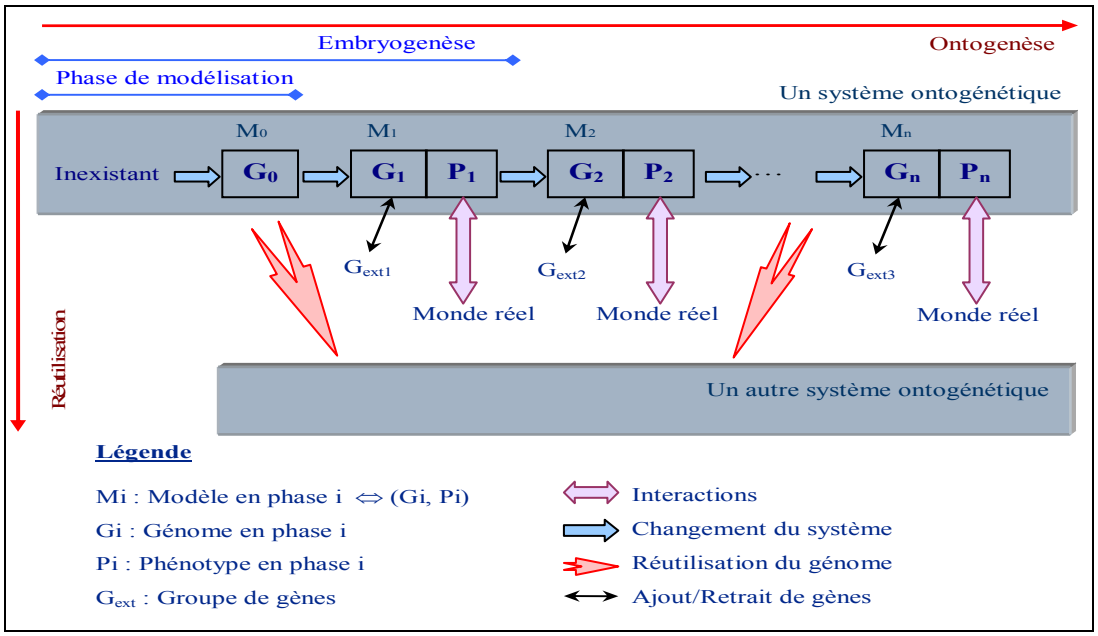


Figure 1.1. L'évolution selon Mage

Selon cette figure, on constate l'existence de trois grandes phases :

La modélisation. Un modèle commence à exister à la fin de la phase de modélisation qui consiste en une conception et une implémentation. Le but de cette phase est de produire un génome qui, une fois exécuté, donne au modèle son premier phénotype.

L'embryogenèse. La fin d'une phase importante du modèle est atteinte lorsqu'il parvient à la fin de l'embryogenèse. Cette fin est marquée par l'aptitude du modèle à interagir avec l'environnement (dans le sens de l'exécution des fonctions).

L'évolution continue. Le génome demeure actif et façonne continuellement le modèle. Les changements non anticipés sont injectés sous forme de gènes dans le génome.

Durant la phase de l'évolution continue, un modèle Mage se présente sous la forme donnée en figure 1.2. Cette dernière fait ressortir quatre notions : le phénotype, le génome, l'interaction et les stimuli. Nous décrivons ci-après le rôle de chacune.

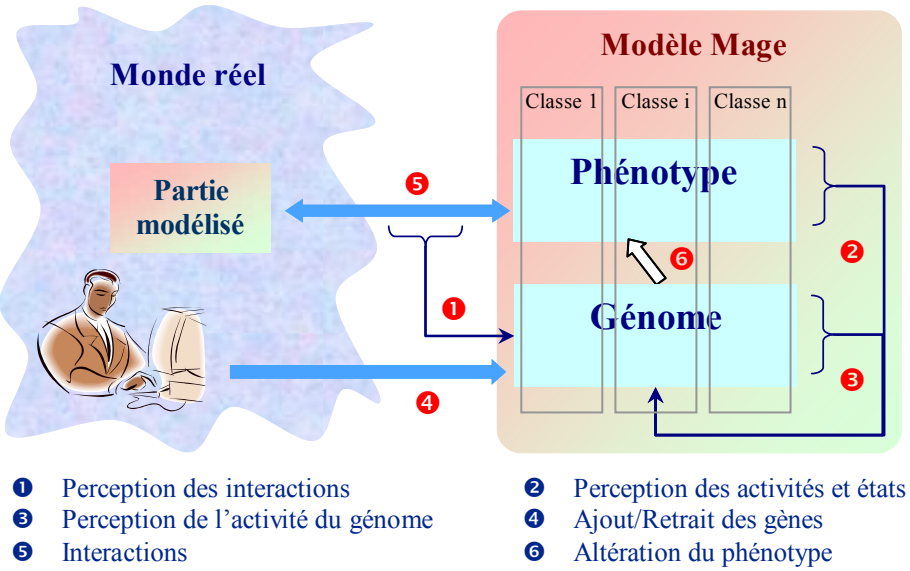


Figure 1.2. Les Concepts en Mage

1.5.4 Mage : Mode Opérationnel

Mage fait usage de quatre notions : le phénotype, le génome, l'interaction et les stimuli. On parle de phénotype à la fin de la phase d'embryogenèse. Le phénotype est décrit à l'aide d'un composant appelé composant universel. Le phénotype est ainsi un graphe constitué d'une multitude de composants universels simples ou composés et interconnectés pour former l'équivalent d'un système objet classique. Il consiste en des composants interconnectés appelés composants universels (UC). Le composant universel est une unité destinée à modéliser et abstraire tous les constituants d'un système : classe, méthode, objet primitif, objet d'une classe, tableau et instruction, graphe, sous graphe (Figure 1.3). Le phénotype est ainsi un graphe constitué d'une multitude de composants universels, simples ou composés, et interconnectés pour former l'équivalent d'un système objet classique.

Les composants universels ont une granularité variable et leur interconnexion fixe à la fois le flux de données et le flux de contrôle. Cette représentation du phénotype permet une expression plus simple et plus uniforme du génome. En effet, au lieu d'altérer un phénotype complexe, le génome altère des connecteurs, des imbrications de composants, des types de composants, etc.

Le génome est partagé en segments dits chromosomes. Chaque chromosome se compose d'éléments de faible granularité qui sont les gènes. On distingue trois types de chromosomes :

Chromosome D. Ce type regroupe les gènes constructeurs ou de développement, responsables des altérations des comportements et structures du phénotype.

Chromosome F. Ce type est composé des gènes qui garantissent des fonctionnalités d'ordre global comme le verrouillage et déverrouillage des composants lors du changement.

Chromosome C. Il matérialise le flux de contrôle de l'évolution et est constitué des gènes contrôleurs qui agissent par activation/désactivation sur tous les autres gènes.

Les gènes sont des objets dont la structure, immuable, se compose de quatre parties : type, état d'activation, condition de déclenchement et informations complémentaires.

La condition d'un gène lui permet de percevoir les stimuli. On en distingue 3 types :

Les stimuli factuels qui reflètent l'état d'un constituant du phénotype ou du génome (présence ou absence de composants universels, de gènes, de connexions, …).

Les stimuli d'activité qui indiquent qu'une activité du phénotype ou du génome est lancée ou terminée. Ils comprennent les stimuli d'interaction qui indiquent un échange entre le phénotype et le monde réel.

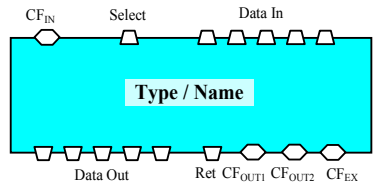
Les stimuli temporels qui permettent aux gènes de se déclencher indépendamment de la structure ou l'état du système, mais conformément à un repère temporel. Ce repère peut être celui d'une propriété, d'un objet ou d'une classe et peut être absolu (celui du système).

L'approche Mage utilise deux modèles d'exécution différents. Le premier est celui du phénotype et le second celui du génome. Au niveau du phénotype, le modèle est celui des langages orientés objets classiques. La sémantique associée à l'UC (composant universel) est la suivante :

Les UC qui matérialisent des données ou des classes n'utilisent pas les flux de contrôle. Il est possible d'y introduire une valeur par l'entrée Select et de lire celle stockée à partir de la sortie Ret.

Les UC qui matérialisent des méthodes ou des instructions utilisent le flux de contrôle et le flux de données. Le déclenchement de l'exécution de l'action associée à l'UC s'opère par l'arrivée d'un jeton sur CF_{IN} et la présence des données nécessaires sur les entrées d'information Select et $DI_0..DI_N$. Après traitement, le jeton est transmis sur CF_{OUT1} (CFOUT2 dans le cas de IF-Else si *Select* est à faux) et le résultat sur les sorties d'information $_{Ret}$ et $DO_0..DO_N$. Toute incompatibilité de données forcera l'UC à produire une exception qui

consiste à transmettre le jeton sur la sortie CFEX qui est reliée à un UC de type Method qui la traite.



Le composant universel (UC)

Type : Indique le rôle de l'UC (classe, méthode, instruction, …)
Name : Nom symbolique ou un nombre unique qui identifie l'UC. Si l'UC est une instruction d'appel, le nom indique l'objet dont la méthode est invoquée.
CF_{IN} : Entrée du flux de contrôle
CF_{OUT1} : Première sortie du flux de contrôle
CF_{OUT2} : Seconde sortie du flux de contrôle
CF_{EX} : Sortie du flux de contrôle dans le cas des exceptions
Select : Entrée non typée qui reçoit une valeur, une référence d'objet ou un sélecteur de méthodes
Ret : Sortie non typée donnant la valeur/référence d'objet ou le résultat d'une méthode
DataIn : DI_0 ..DI_N sont des entrées non typées utilisées pour recevoir des paramètres si l'UC est une méthode ou un appel, ou les indexes d'un élément si l'UC est un tableau
DataOut : Sortie de valeurs ou références d'objets résultat de l'exécution d'une méthode

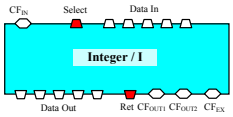
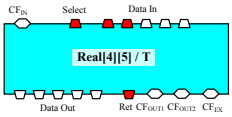
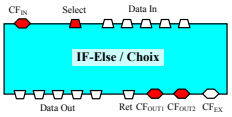
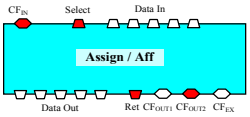
UC				
Ce qu'il représente	Un objet primitif de type Integer. Seuls *Select* et *Ret* sont utilisés. *Select* permet d'assigner une valeur à l'UC et *Ret* permet de la lire	Un tableau à deux dimensions de réels nommé *T*. *Select* et *Ret* permettent d'écrire/lire une valeur. DI_0 *et* DI_1 introduisent les indexes des valeurs.	Instruction *If-Else*. Le *Select* reçoit le *Ret* d'un sous-graphe évaluant une condition. Le flux de contrôle arrive par CF_{IN} puis est transmis sur CF_{OUT1} ou CF_{OUT2} selon la valeur du *Select*.	C'est une affectation qui établit une connexion entre deux objets. Le *Select* de l'objet assigné est connecté au *Ret* de *Aff* et la valeur assignée est donnée sur le *Select* de *Aff*.

Figure 1.3 Le composant universel

1.5.5 L'évolution Anticipée et Non Anticipée

Les systèmes ontogénétiques présentent la caractéristique d'évoluer dynamiquement et de manière autonome pour répondre aux besoins des utilisateurs et leurs changements qu'ils soient anticipés ou non, sachant que :

Les changements anticipés sont ceux identifiés avant la livraison des systèmes logiciels. Ils n'ont de signification que lorsque vient le moment opportun. Ces changements peuvent être anticipés au moment de la conception et doivent être prévus par le programmeur (concepteur) pendant les phases de développement du produit logiciel. Il serait aisé de les inclure moyennant un paramétrage dans le système. Par exemple, le passage d'un objet d'une classe à une autre (e.g. un candidat qui soutient une thèse devient enseignant et passe de la classe *Etudiant* à la classe *Enseignant* [Ous 99].

Les changements non anticipés sont définis dans [Alm 06] comme : "unanticipated software evolution is not something for which we can prepare during the design of a software system". Que l'on peut traduire par "Les évolutions non anticipées ne peuvent être prises en compte durant la conception des systèmes logiciels". Ils visent à corriger les erreurs, ajouter de nouvelles fonctionnalités et en supprimer d'autres. Ces dernières apparaissent une fois le système opérationnel. Ce sont des changements que l'équipe de développement du produit n'a pas considéré dans le processus de conception et d'implémentation.

Les évolutions non anticipées sont une voie de recherche clé en génie logiciel, car elles sont fortement liées au temps et aux coûts du développement et maintenance des systèmes logiciels. Cette dernière est une activité très couteuse et demande pour sa réalisation beaucoup d'effort et de temps pour permettre aux mainteneurs de comprendre les programmes.

Malgré l'importance de l'évolution des systèmes logiciel, les modèles et techniques qui offrent un support pour l'évolution des logiciels sont loin d'être idéales. En particulier, les changements non anticipés des besoins qui ne sont pas bien pris en charge, malgré qu'ils représentent des complications techniques et des coûts très importants. L'objectif des travaux de recherche est alors de trouver des méthodes pour que l'évolution non anticipée se fasse au moindre coût. Dans la littérature, la plupart des propositions reposent sur la transformation de code [Sad 03].

Conclusion

Le paradigme Mage présente une nouvelle approche où le modèle d'un système est une représentation qui englobe ses dimensions structurelle, comportementale et ontogénétique. Mage incarne une vision radicale du changement en s'inspirant du développement biologique. En effet, sous Mage, un système est conçu pour changer. Pour forcer cette vision, le phénotype est considéré inexistant au départ et qu'il est le fruit d'une activité continue du génome. Mage modélise uniformément le phénotype en utilisant le composant universel et le génome par trois types de gènes dont les rôles sont bien distincts. En plus de son aspect naturel, Mage traite de façon uniforme les changements anticipés et non anticipés. Les changements anticipés sont prévus, donc déjà étudiée lors de la phase d'analyse alors que ceux non anticipés, sont imprévus lors du développement du système.

La prise en compte de l'ontogenèse constitue un nouveau défi et une vision radicale de l'évolution qui a un effet aussi bien sur notre façon de percevoir les systèmes logiciels que notre manière de les concevoir. En effet, les systèmes ontogénétiques nécessitent clairement des démarches de développement appropriées qui redonnent à l'évolution le statut de concept central. Dans le chapitre suivant nous discutons les deux approches de méthodologies de développement existantes et nous présentons le processus Rational Unified Process basé sur le formalisme UML.

CHAPITRE 2
LES DEMARCHES DE DEVELOPPEMENT ACTUELLES

Afin de faire face à la complexité croissante du développement logiciel, des méthodes de gestion des projets informatiques ont été introduites. Celles dites traditionnelles sont axées-plan et généralement lourdes, ces méthodes ont perdu de plus en plus de partisans en faveur des nouvelles méthodes dites « agiles ». Cependant, ces dernières présentent des limitations pour les projets critiques ou de grande taille nécessitant une équipe importante en nombre.

Ce chapitre vise à faire une présentation succincte permettant de comprendre les processus et les méthodes de développement existants tout en comparant l'école traditionnelle et celle dite Agile. Nous présentons également le processus RUP qui constitue le cadre de la démarche que nous proposons dans le chapitre 4.

2.1 Le Processus de Développement

La recherche en génie logiciel a depuis long temps tenté de mieux comprendre le processus de développement logiciel, minimalement, pour en reproduire les bonnes pratiques, et idéalement pour pouvoir le mécaniser. Un processus est «un ensemble organisé d'activités mises en œuvre dans un but précis».

Le développement de logiciel comprend l'ensemble des étapes et processus qui permettent de passer de l'expression d'un besoin informatique à un logiciel fonctionnel et fiable. L'application de différentes activités allant de la collecte et l'analyse des besoins jusqu'au déploiement et la maintenance du système.

Le processus de développement de systèmes logiciels est alors défini comme étant un ensemble organisé d'activités mises en œuvre dans le but de construire un logiciel. Plus précisément, un processus de développement de logiciels est définis comme «un ou plusieurs ensembles d'activités ordonnées, avec un déroulement séquentiel ou parallèle, qui permettent d'analyser et de concevoir, de rétro-concevoir ou de refondre le système» [Cas 92].

2.1.1 Définitions et Concepts

Système. Un système est généralement décrit comme un ensemble d'éléments en interaction organisés en un tout homogène au sein d'un environnement avec lequel il interagit [Moi 90]. Il évolue dans le temps [Sha 67], il transforme des grandeurs d'entrées en grandeurs de sorties [Lar 93] et il réalise des fonctions afin d'atteindre un objectif donné (Figure 2.1).

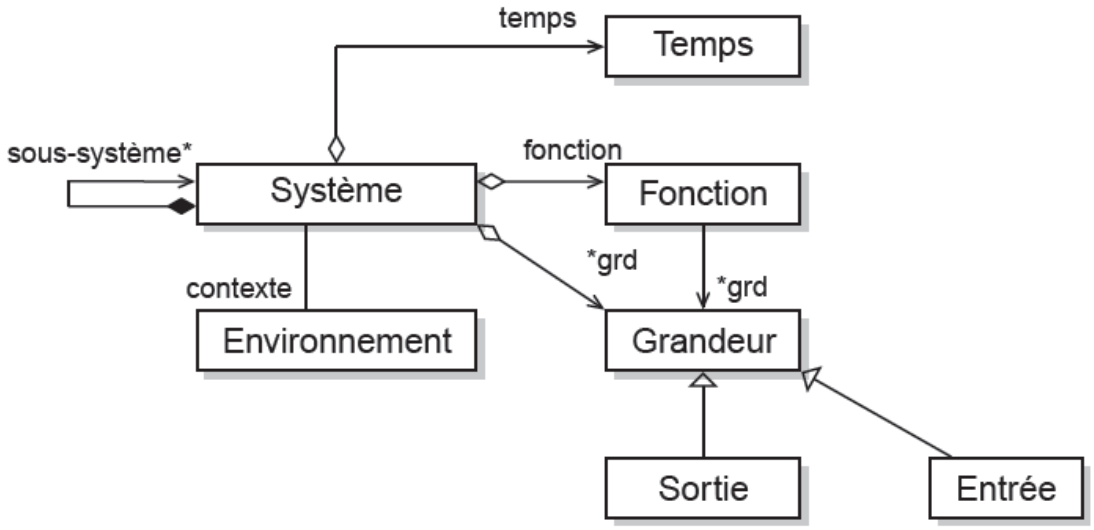


Figure 2.1 Vue conceptuelle de la notion de système

Système logiciel. Les systèmes logiciels, comme tout système, sont constitués de sous systèmes en interactions et évoluent dans un environnement donné. A cette caractérisation, se greffe un certain nombre de concepts clefs tels que : *l'architecture, l'état, le comportement, la réactivité, la concurrence, la communication, l'émergence ou encore la complexité* [Per 07].

Méthode. Pour Booch [Boo 91], une méthode est « un processus rigoureux permettant de générer un ensemble de modèles qui décrivent divers aspects d'un logiciel en cours de construction en utilisant une certaine notation bien définie ».

Les modèles de processus proposent des démarches de développement de systèmes logiciels. Ils décrivent une démarche méthodologique pour atteindre la cible que sont les produits [Rol 05].

Humphrey définit le processus d'ingénierie logicielle comme suit : «Le processus d'ingénierie logicielle est l'ensemble des activités d'ingénierie logicielle nécessaires à la transformation des besoins d'un utilisateur en un logiciel» [Hum 88].

2.1.2 Cycle de vie de logiciel

Le cycle de vie d'un logiciel est un ordonnancement de différentes étapes du processus de développement. Il désigne toutes les étapes de développement d'un logiciel, de sa conception à sa disparition. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la vérification du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre. L'origine de ce découpage provient du constat que les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de réalisation. Le cycle de vie permet de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés.

Le cycle de vie du logiciel comprend au minimum les étapes suivantes (Figure 2.2):

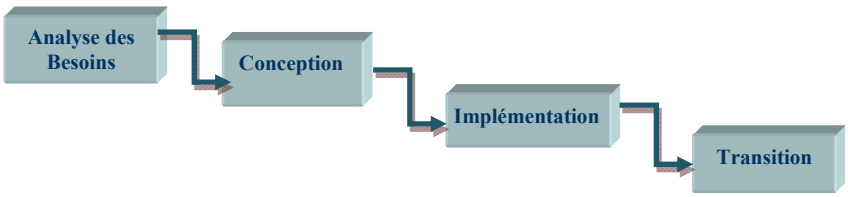


Figure 2.2 Etapes essentielles d'un cycle de vie de logiciel

2.1.2.1 Le cycle de développement en cascade

Le modèle de cycle de vie en cascade a été mis au point dès 1966, puis formalisé aux alentours de 1970. Il définit des phases séquentielles à l'issue de chacune desquelles des documents sont produits pour en vérifier la conformité avant de passer à la suivante (Figure 2.3).

Le cycle en « cascade » se caractérise par des phases séquentielles, qui le succèdent après la validation des livrables produits lors de la phase précédente :

- Tous les besoins sont exprimés et recueillis lors de la première phase, puis vient l'analyse détaillé de ces besoins dont la conception du système est très dépendante.
- La conception du système, bien que textuelle ou représentée sous forme de diagrammes, doit être validée avant le démarrage du développement.

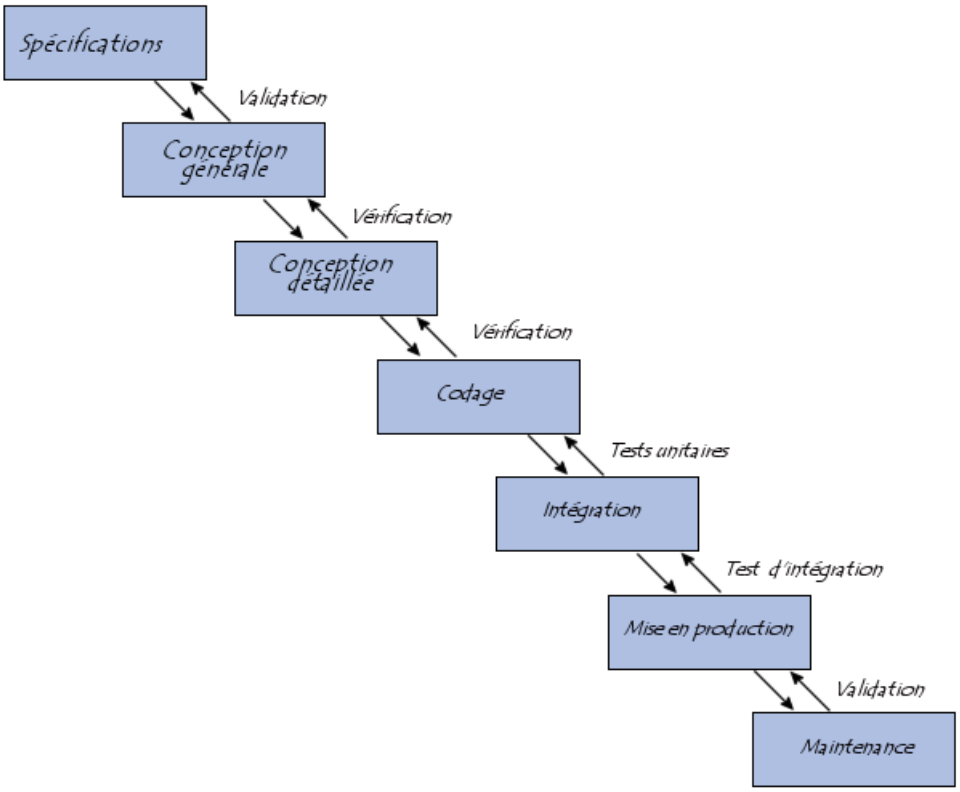


Figure 2.3 Cycle de vie de logiciels en cascade

- Le développement doit être achevé pour permettre à l'équipe de testeurs de lancer ses campagnes de tests fonctionnels et techniques.
- Enfin, une fois les anomalies ont été corrigées, on peut procéder à l'intégration globale finale et à la mise en production du système.

2.1.2.2 Le modèle de cycle de vie en V

Le modèle du cycle en V a été imaginé pour pallier au problème de réactivité du modèle en cascade. Ce modèle est une amélioration du modèle en cascade qui permet en cas d'anomalie, de limiter un retour aux étapes précédentes. Les phases de la partie montante ci-contre doivent renvoyer de l'information sur les phases en vis-à-vis lorsque des défauts sont détectés afin d'améliorer le logiciel. Le cycle en V est devenu un standard de l'industrie du développement de logiciel et de la gestion de projet depuis les années 1980 [McD 84].

Le cycle en V améliore le cycle de la cascade pour mettre l'accent sur les aspects vérification et validation. Les jeux de tests sont préparés dès les phases de spécification et permettent ainsi d'améliorer la production de logiciels corrects et valides (norme AFNOR Z67-130) [Gir 07].

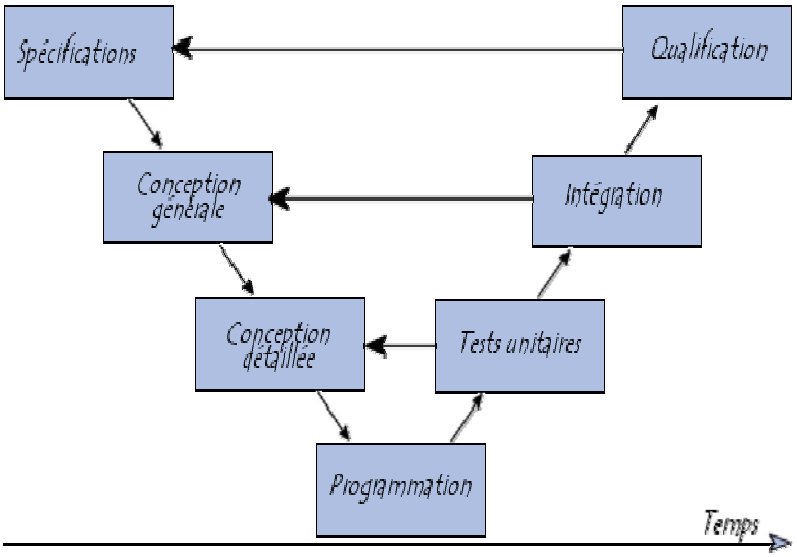


Figure 2.4 cycle de vie en V [Boe 81]

L'inconvénient évident du modèle en V est qu'il faut attendre la fin du processus de développement pour avoir le retour des utilisateurs sur le système logiciel réellement implanté.

2.1.2.3 Le modèle de cycle de vie en Spirale

Le développement se base sur les différentes étapes du cycle en V pour permettre de produire un produit de plus en plus complet via la réalisation de versions successives. Le cycle en spirale généralise le principe d'incrément et évite « l'effet tunnel » où le temps est trop long entre les premières spécifications de besoins et la livraison de tout le système [Boe 86]. Il introduit la notion de prototype (Figure 2.5). Ce cycle de vie correspond à une adaptation du principe de la roue de Deming [Gir 07] (Figure 2.6). La roue de Deming a été créée par Willian Edwards Deming, aussi connue sous le nom de méthode PDCA. La PDCA tire son origine des premières lettres des mots qui la composent : Plan-Do-Check-Act. Ces derniers peuvent être interprétés tel qu'il suit :

- Plan : Préparer, Planifier ;
- Do : Développer, réaliser, mettre en œuvre ;
- Check : Contrôler, vérifier ;
- Act (ou Adjust): Agir, ajuster, réagir.

La méthode PDCA est une démarche cyclique d'amélioration qui consiste, à la fin de chaque cycle, à remettre en question toutes les actions précédemment menées afin de les améliorer [PCD 14].

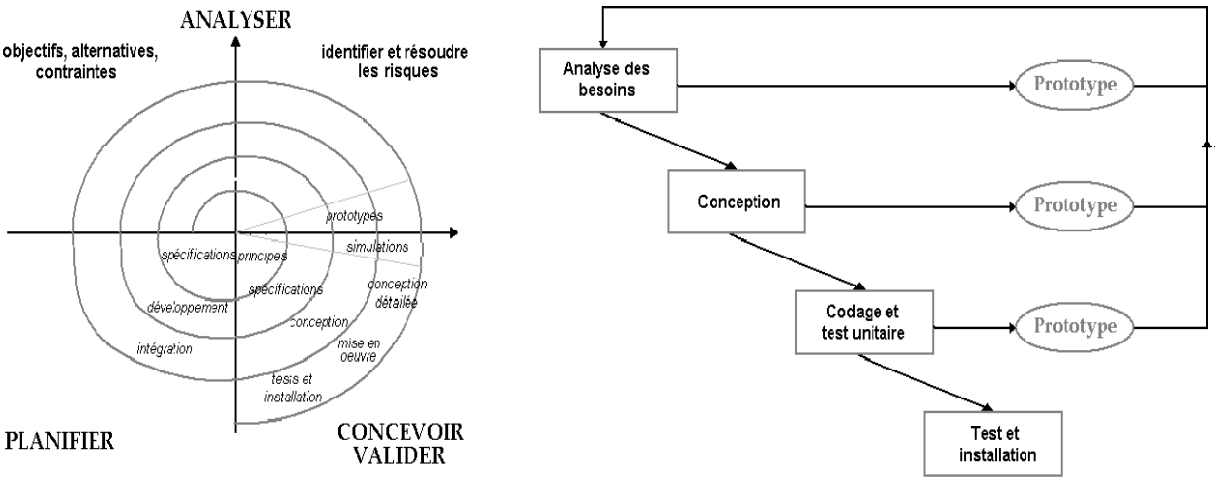


Figure 2.5 Modèle en Spirale de Bohem

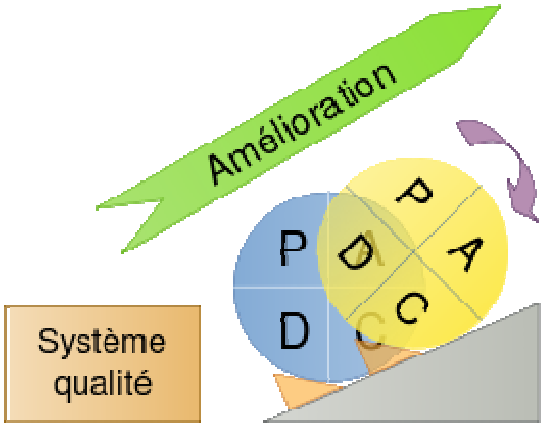


Figure 2.6 Roue de Deming

2.1.3 Contexte du développement d'un système logiciel

Il est important de mieux situer le contexte général de tout développement d'un système logiciel. Un système logiciel est le résultat d'un processus d'ingénierie mettant en œuvre les technologies courantes pour satisfaire les exigences spécifiques d'un client.

Trois aspects essentiels (Figure 2.7), correspondant globalement à trois métiers différents, sont à considérer [Col 10] :

1. **le *métier***, qui requiert des compétences liées au domaine d'application du logiciel,
2. **l'*ingénierie logicielle***, qui requiert des compétences en conception de logiciels,

3. **la *gestion du processus*** *de développement*, qui requiert des compétences en génie logiciel.

Le métier ou domaine d'application impose la prise en compte de connaissances métiers ou de connaissances théoriques relevant d'un domaine d'application particulier. Il détermine des classes d'applications et peut avoir un impact important sur l'aspect spécificité d'un processus de développement (intégration d'activités ou d'outillage spécifiques, conception ou utilisation d'un Framework métier, etc.).

L'ingénierie logicielle regroupe l'ensemble des techniques, technologies, solutions logicielles ou encore méthodes de développement à utiliser pour concevoir le logiciel. Elle a un impact sur la spécificité du processus de développement (choix d'une approche orientée objets, ou à base de composants ou d'agents, conformité à un style d'architecture, intégration de Framework techniques, etc.).

La gestion du processus impose un cycle de développement ou une démarche méthodologique qui doivent être suivis par l'équipe de développement.

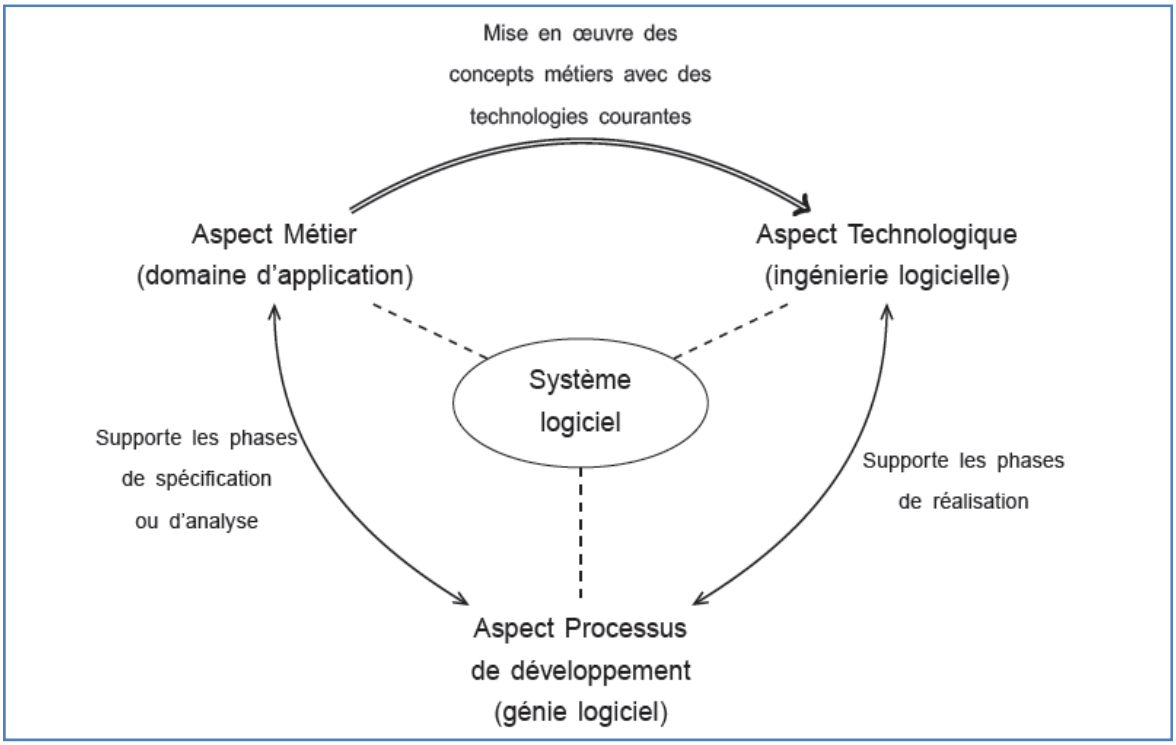


Figure 2.7 Contexte général du développement logiciel [Col 10]

Ces trois aspects sont en interaction et visent à répondre aux questions suivantes concernant le système en développement : *pourquoi* et *quoi* pour l'aspect métier, *avec quoi* pour l'aspect ingénierie logicielle et enfin *comment faire* pour l'aspect processus. Un processus spécifique à une classe d'applications relevant d'un domaine métier particulier devra alors intégrer ces différents aspects.