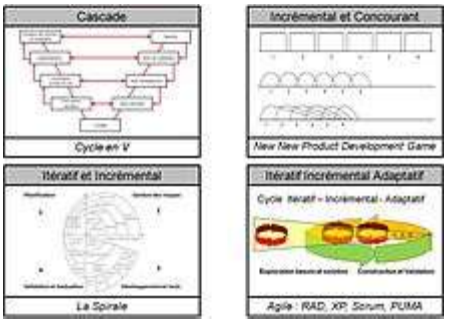
Cycle de développement (logiciel)

Il existe différents types de **cycles de développement** entrant dans la réalisation d'un logiciel. Ces cycles prendront en compte toutes les étapes de la conception d'un logiciel.

Les grandes familles

Modèle en cascade



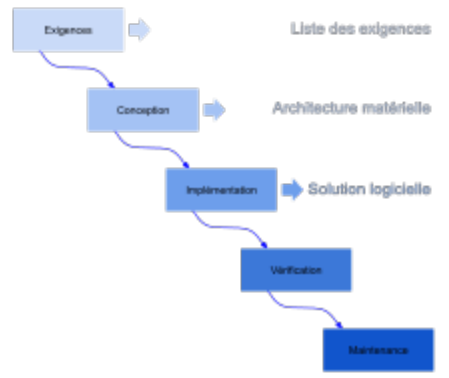
Évolution des cycles basiques

Le modèle en cascade[1] est hérité de l'industrie du BTP. Ce modèle repose sur les hypothèses suivantes :

- on ne peut pas construire la toiture avant les fondations ;
- les conséquences d'une modification en amont du cycle ont un impact majeur sur les coûts en aval (on peut imaginer la fabrication d'un moule dans l'industrie du plastique).

Les phases traditionnelles de développement sont effectuées simplement les unes après les autres, avec un retour sur les précédentes, voire au tout début du cycle. Le processus de développement utilisant un cycle en cascade exécute des phases qui ont pour caractéristiques :

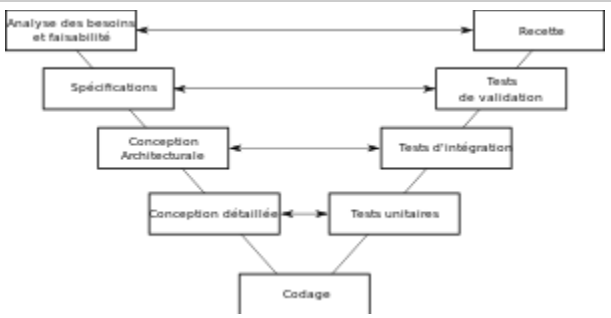
- de produire des livrables définis au préalable ;
- de se terminer à une date précise ;
- de ne se terminer que lorsque les livrables sont jugés satisfaisants lors d'une étape de validation-vérification.



Modèle en cascade

Cycle en V

Le modèle du cycle en V[2] a été imaginé pour pallier le 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 doivent renvoyer de l'information sur les phases en vis-à-vis lorsque des défauts sont détectés afin d'améliorer le logiciel.



Les phases du cycle en V

De plus le cycle en V met en évidence la nécessité d'anticiper et de préparer dans les étapes descendantes les « attendus » des futures étapes montantes : ainsi les attendus des tests de validation sont définis lors des spécifications, les attendus des tests unitaires sont définis lors de la conception, etc.

Cycle en spirale

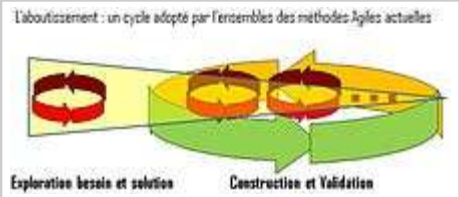
Le développement reprend les différentes étapes du cycle en V. Par l'implémentation de versions successives, le

cycle recommence en proposant un produit de plus en plus complet et robuste.

Le cycle en spirale[3] met cependant plus l'accent sur la gestion des risques que le cycle en V. En effet, le début de chaque itération comprend une phase d'analyse des risques. Celle-ci est rendue nécessaire par le fait que, lors d'un développement cyclique, il y a plus de risques de défaire, au cours de l'itération, ce qui a été fait au cours de l'itération précédente.

Cycle semi-itératif

Le cycle semi-itératif a pour origine les travaux de James Martin publiés à partir de 1989 dans diverses revues Nord Américaines. Ce cycle de développement fut totalement formalisé en 1991 dans le livre RAD[4] (Développement rapide d'applications). À partir de 1994, des travaux complémentaires menés en France (RAD2) et en Angleterre (DSDM) apportèrent des spécialisations aux techniques basiques initiales. L'adoption par RUP[5] Rational Unified Process d'IBM consacra l'apogée de ce cycle toujours en vigueur dans les projets conséquents.



Le cycle Agile est en fait semi-itératif

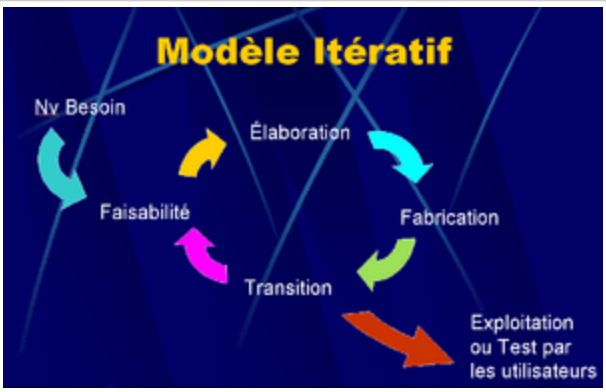
Dans le cycle semi-itératif les deux premières phases classiques (top down, par la structure) consistent en l'expression des besoins et la conception de la solution. C'est lors de la troisième et dernière grande phase, la construction du produit (bottom up, par le besoin) que la notion d'itérations courtes intervient[6]. C'est vers 2001 avec l'apparition de plusieurs méthodes dont ASD, FDD, Crystal, Scrum[7] ou l'extreme programming[8] et la vision uniformisée de leurs auteurs dans le cadre du Manifeste Agile (Agile Manifesto) et de l'Agile Alliance, que le cycle semi-itératif fut généralisé. Toutes les méthodes Agiles débutent par des phases séquentielles, courtes mais bien réelles, d'exploration, d'architecture et de planning. Un usage totalement itératif de ces méthodes n'est cependant pas exclu mais ne peut s'appliquer qu'à de très petits projets.

Cycle itératif

On sépare les activités des artefacts, un artefact étant le produit issu d'une activité. Ainsi, on applique un cycle de type roue de Deming sur la production d'une documentation, d'un composant, d'un test, etc.

Rapportée à une activité de type gestion de projet, la première phase sera celle de

- la faisabilité : l'acceptation d'un nouveau besoin
- l'élaboration : on imagine comment on va le réaliser
- la fabrication : construction
- la transition : tout est mis en œuvre pour livrer au client



Les phases du Cycle itératif

Le cycle itératif **n'est pas** une bijection avec le cycle en V du type

- faisabilité = spécifications
- élaboration = Architecture
- fabrication = développement prototype

- transition = tests

Sachant que chaque itération ne dépasse **jamais** huit semaines, cette tactique est donc impossible. L'idée est de livrer au plus tôt quelque chose qui puisse être testé par le client. On peut en effet réaliser plusieurs itérations sur une documentation telle que l'architecture. De la même manière, si un document n'est qu'un artéfact parmi d'autres, il ne faut pas obtenir un document complet. On préférera utiliser la loi de Pareto : ne pas passer 80 % de l'effort sur les 20 % restants.

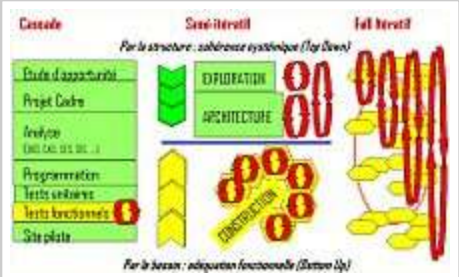
La différence entre un PDCA et une itération est la durée : elle doit être courte et régulière alors qu'une roue de Deming appliquée à une organisation de 300 personnes prend plusieurs mois, voire plusieurs années.

Comparaison des approches Cascade, V et Itératif

Le cycle en cascade a pour origine l'industrie lourde. La particularité de ce milieu est que la phase qui suit nécessite bien plus de ressources que la précédente.

Par exemple, pour fabriquer un objet en matière plastique,

1. un bureau d'étude va concevoir le produit,
2. puis des empreintes de moules seront usinées et placées dans des carcasses pour recevoir de la matière plastique par injection,
3. et une fois que le prototype est correct, on passe à une phase de production.



Diverses formes d'itératif

Il faut savoir que pour un objet simple tel qu'un gobelet en plastique, la conception est une affaire d'une poignée de semaines (soit quelques milliers d'euros) alors qu'un moule (empreinte + carcasse) nécessite plusieurs mois de fabrication et plusieurs centaines de milliers d'euros.

Par conséquent, dans un tel contexte, pour bien gérer son projet, il est important de ne pas négliger la validation de chaque étape sous peine de le voir déraper.

Ce phénomène intervient sur des chantiers logiciels réunissant des dizaines voire des centaines de personnes. Les décisions de l'équipe de direction ou de l'architecte projet impactent tellement d'ingénieurs pour de telles durées qu'il vaut mieux s'assurer de la validité de chaque étape.

Par ailleurs, pour limiter l'entropie (désordre) du système constitué par l'équipe-projet, il est nécessaire de formaliser par des documents (voire des outils) :

- les processus
- les besoins
- les spécifications logicielles
- l'architecture logicielle
- les tests

Dans le cas d'un projet logiciel impliquant une douzaine de personnes pendant une à deux années, la configuration n'est plus la même ; en effet, avec de tels projets on dispose :

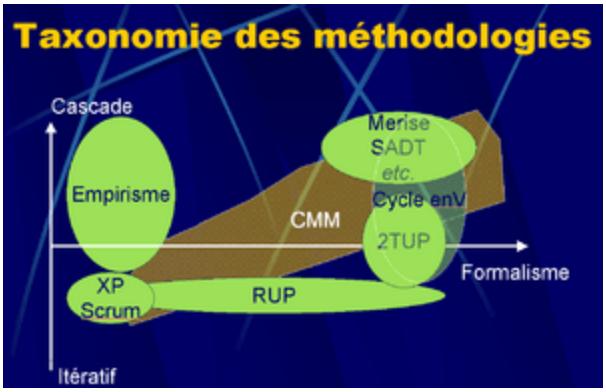
- d'une plus grande réactivité, due à :
  - une proximité géographique

  - une facilité (relative) de communication
- d'un facteur de coût limité entre chaque étape

Aussi, il est possible de s'orienter vers des méthodes de développement dites *agiles* en diminuant le formalisme et en multipliant le nombre de cycles (fonctionnement itératif).

Quelques méthodologies

- Développement rapide d'applications
- eXtreme Programming
- Scrum
- RUP
- 2TUP
- Merise
- SADT
- HERMES
- Processus urbanisant les méthodes agiles (PUMA, voir Méthode agile)
- CMMI



Taxonomie des méthodologies

Références

1. ↑ Winston W. Royce. Managing the Development of Large Software Systems. IEEE Wescon, p. 1-9, 1970
2. ↑ John McDermid et Knut Ripken. Life cycle support in the ADA environment. University Press, 1984.
3. ↑ Barry W. Boehm. A Spiral Model of Software Development and Enhancement. IEEE Computer, 21(5), p. 61-72, 1988.
4. ↑ James Martin. Rapid Application Development. Macmillan Coll. Div., 1991.
5. ↑ Philippe Kruchten. The Rational Unified Process: An Introduction. Addison-Wesley, Longman Publishing, Co., Inc. Boston, Massachusetts, 2000.
6. ↑ http://www.rad.fr/phasprin.htm
7. ↑ Ken Schwaber et Mike Beedle. Agile Software Development with SCRUM. Prentice Hall, Upper Saddle River, New Jersey, 2001.
8. ↑ Kent Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley Professional, Longman Publishing Co., Inc. Boston, Massachusetts, 1999.

Voir aussi

- Méthode agile
- Catégorie:Méthode de développement logiciel
- Qualité des systèmes informatiques

media files.