Génie Logiciel - Cours 02

Régis WITZ

Cycle de vie

Phases d'existence

Analyse du besoin / Faisabilité / Cahier des charges

Cette phase permet de collecter les données nécessaires pour rendre compte de la situation réelle.

Il est d'abord nécessaire de **comprendre le contexte**, en utilisant toutes les sources d'information pertinentes. La principale source d'information est le client, auquel il faut poser toute question utile, en n'hésitant pas à approfondir : certains éléments vitaux sont parfois tellement évidents/implicites pour le client qu'il ne pense pas à les mentionner.

Ensuite, mettre en perspective les besoins exprimés par avec le contexte permet de redéfinir ou valider les besoins **réels**. Ces besoins réels devraient être ordonnés par degré d'importance et peuvent être regroupés par thèmes.

À cette phase sont aussi établies les différents paramètres de conception, et en particulier les **contraintes** qui peuvent empêcher d'atteindre toute ou partie des objectifs de départ. Les contraintes techniques ou économiques ne sont pas les seules à considérer ; certains domaines sont aussi touchés par des contraintes légales, sociales, environnementales, et ainsi de suite.

La phase d'analyse aboutit à la validation par la MOA d'un **cahier des charges** réaliste et convenant à ses besoins réels. Ce document précise clairement les tâche à effectuer ainsi qu'une estimation du temps nécessaire pour chacune.

Cette phase permet aussi à la MOE d'évaluer qu'elle a accès à toutes les compétences nécessaires au projet, ou si le recours à des ressources externes est nécessaire.

Étude préalable

En amont de la phase d'analyse du besoin, on peut trouver une phase encore antérieure d'étude préalable. Tandis que l'analyse du besoin vise à être précise et exhaustive, l'étude préalable vise simplement à dresser un premier tour d'horizon et à fixer le périmètre.

La phase d'étude préalable ne vise pas l'exhaustivité, mais à orienter les phases suivantes (y compris l'analyse détaillée du besoin).

Exemple d'outils d'analyse

QQOQCCP

- 4 questions : Qui ? Quoi ? Où ? Quand ?
- Complétées par 3 modalités : Comment ? Combien ? Pourquoi ?

Un équivalent anglais est le Five W's : Who, What, Where, When, Why ? Utilisé par les journalistes pour la rédaction de leurs articles.

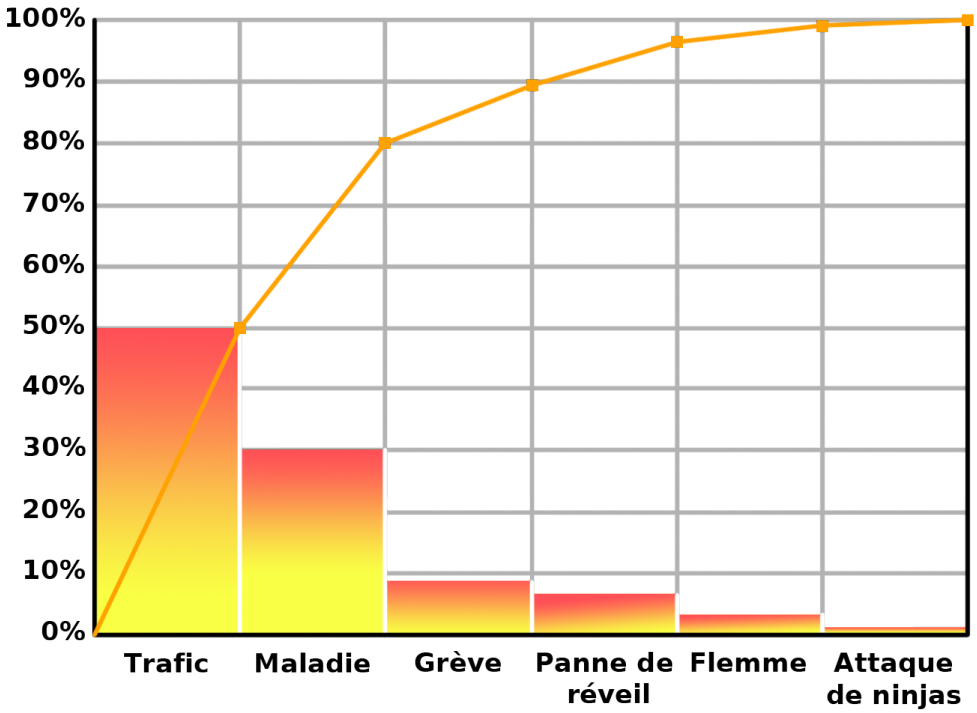
Méthode des 5 pourquoi

Poser successivement cinq questions commençant par *Pourquoi* peut permettre de mieux cerner la cause réelle d'un problème.

Diagramme de Pareto

Un tel diagramme représente sous forme graphique les différentes causes d'un problème, ordonnées par leur importance.

C'est une application de la loi des 80/20 (ou principe de Pareto) : 20% des causes produisent 80% des effets.

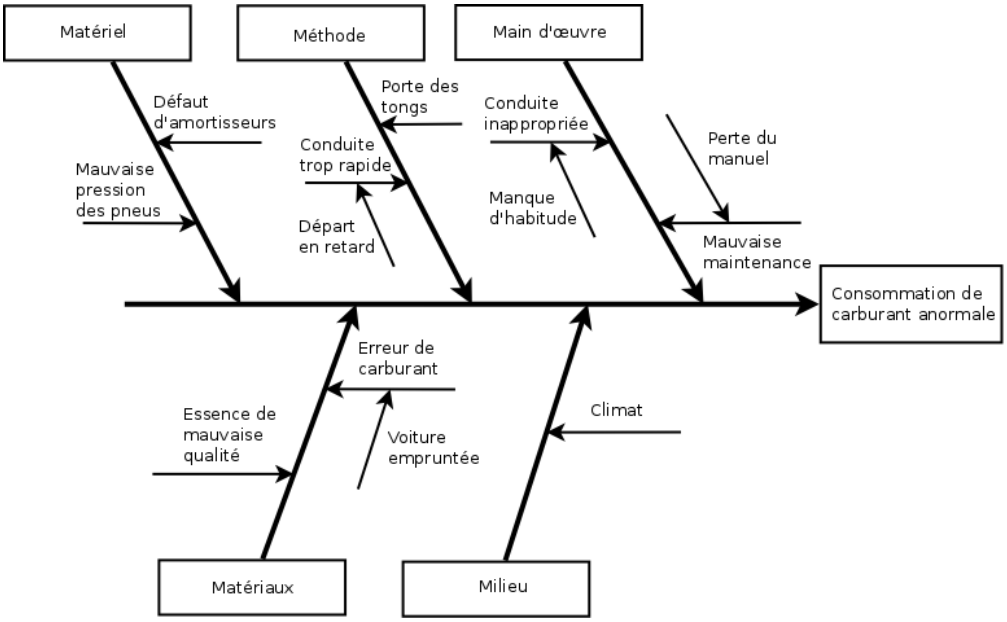


Exemple de diagramme de Pareto:Causes de retards en cours

Diagramme d'Ishikawa

Aussi appelé 5M ou arêtes de poisson, ce diagramme permet de mettre en relation les causes et les effets.

- Matériaux entrant en jeu
- Matériel : équipement, machines, logiciels, technologies
- Méthode opératoire, logique
- Main d'œuvre humaine
- Milieu : environnement, contexte



Exemple de diagramme d'Ishikawa:Causes d'une consommation de carburant anormalement élevée

Spécification

Lors de la phase de spécification, le besoin qui a été analysé précédemment est décrit avec plus de détail, sous forme d'**exigences** que la solution doit impérativement satisfaire.

Un document de spécification peut être de deux types principaux :

- Une spécification **fonctionnelle** décrit les **processus métier** dans lesquels la solution intervient. Par exemple, les unités utilisées, les règles de calcul ou d'interaction, etc. La spécification fonctionnelle représente le **but à atteindre**.
- Une spécification **technique** décrit l'**environnement technique** dans lequel la solution s'inscrit. Par exemple, le design architectural, le format des données d'échange avec les composants déja présents, les langages de programmation utilisés, le format des bases de données, le système hôte, ... peuvent être fixées dans ce document. La spécification technique représente le **moyen d'atteindre le but** fixé par la partie fonctionnelle.

Cette phase débouche souvent sur plus d'un document de spécification. Notamment, les exigences peuvent être raffinées de plus en plus au cours de cette phase jusqu'à atteindre un niveau de détail satisfaisant : on peut alors créer des spécifications générales, puis plus détaillées.

Le plus souvent, c'est la MOA qui est à l'origine des spécifications générales. Il peut cependant être pertinent que les spécifications détaillées soient plutôt écrites par la MOE.

Puisqu'elle décrit aux futurs utilisateurs et développeurs à quoi ressemblera le produit fini, la spécification permet de faire les estimations de coût et de durée. Elle sert donc de base pour établir le planning du projet.

La spécification sert aussi de base contractuelle. Après cette phase, toutes les fonctionnalités qui sont hors-spécification n'ont pas à être ni demandées, ni payées par le client.

Conception

Tandis que la phase de spécification a pour but de décrire la solution vue de l'extérieur, la conception la décrit vue de l'intérieur. Tandis que la phase de spécification décrit les contraintes, la conception apporte les solutions.

C'est le travail de la MOE.

Comme toute documentation, elle peut être raffinée de plus en plus, par exemple en un documents de conception préliminaire/architecturale, puis détaillée.

Implémentation / Développement

Cette phase consiste en la **réalisation** de la solution telle qu'elle a été conçue.

Tests

Tester le logiciel tel qu'il est implémenté a pour objectif d'améliorer la qualité ou de connaître le degré de présence d'une qualité particulière.

Un **test** consiste en la vérification *partielle* du logiciel. Il correspond à la combinaison de trois choses:

- des données en entrée
- un objet à tester
- une situation attendue

Si la situation attendue correspond à la situation observée lors du test, c'est un signe de la qualité du logiciel.

Intégration

Durant la phase d'**intégration**, chaque module du logiciel est intégré et testé dans l'ensemble.

Aussi appelé **tests fonctionnels**, cette phase a pour but de vérifier l'aspect fonctionnel (incluant performances, stabilité, etc), parfois non détectable par des tests de plus bas niveau.

Validation

Durant la phase de **validation**, le système est testé dans son ensemble, et dans un environnement se rapprochant au maximum de l'environnement final. Le but est d'évaluer sa conformité avec les exigences spécifiés.

Un type particulier de validation est la **recette**. Elle se déroule en présence de tous les acteurs (MOA et MOE). Elle précède souvent un jalon important de la vie du projet, comme une livraison.

Déploiement

Le déploiement d'un logiciel consiste à sa **mise en production**, c'est à dire à le rendre disponible et utilisable pour le client, ainsi que pour ses utilisateurs finaux.

On peut décomposer cette phase en plusieurs étapes qui s'appliqueront (ou pas) à un projet particulier.

- Livraison (*release*, *packaging*)
  Les différents composants de la solution sont préparés afin de les rendre utilisables.
- Activation (*install*, *activation*)
  La solution est rendue utilisable dans son environnement de production. Ses différents composants sont installés et configurés.
- Désactivation (*uninstall*, *deactivation*)
  Une solution précédente peut avoir à être totalement ou partiellement désinstallée ou désactivée pour permettre à la nouvelle solution de la remplacer.
- Mise à jour (*update*)
  Le nouvelle solution peut nécessiter une version plus récente de dépendances déjà présentes dans son environnement de production. Elle peut aussi faire partie d'un système plus grand, qui doit alors être mis à jour pour permettre l'activation de la solution.

Le déploiement d'une même solution peut être effectué à plusieurs reprises. Cette phase rend indispensable l'utilisation d'un **gestionnaire de version** ainsi que d'un **gestionnaire de configuration**.

Exploitation / Maintenance

Un logiciel peut être amené à évoluer même après avoir été livré, au cours d'actions de **maintenance**.

Une maintenance peut être de plusieurs types :

- La **maintenance corrective** consiste à résoudre une anomalie constatée
  - *maintenance curative*
    Elle corrige l'anomalie de manière permanente.
  - *maintenance paliative*
    Elle empêche l'anomalie d'endommager le système on l'environnement client, tout en permettant au logiciel de continuer à remplir tout ou partie de ses fonctionnalités. Cependant, étant donné que son impact est forcément négatif à un certain degré, ce type de maintenance est souvent de nature temporaire.
- La **maintenance préventive** consiste à intervenir sur un logiciel avant qu'une anomalie ne survienne. Ce type de maintenance peut être *systématique* ou *conditionnel*.
- La **maintenance évolutive** permet de mieux répondre au besoin ou de répondre à de nouveaux besoins, en modifiant le logiciel existant ou en développant de nouvelles fonctionnalités.

La maintenance se différencie des autres phases en ce que le logiciel considéré est déjà en production.

Modèles de développement

Un modèle de développement ordonne de manière structurées les activités de construction du logiciel. Le détail des activités qui se dérouleront dépend du projet.

Caractéristiques

- Clairement défini et implémenté
- Compréhensible par les acteurs du projet
- Accepté par les acteurs du projet
- Observable de l'extérieur (autres acteurs, parties prenantes, …)
- Il doit permettre de détecter les problèmes avant que le produit ne soit mis en service
- Un unique problème imprévu ne doit pas stopper toute la réalisation

Au sein d'un modèle de développement, chaque activité doit détailler :

- Les Tâches à réaliser et leurs auteurs
- Les Décisions à prendre (le cas échéant)
- Les artefacts livrables.
    - Documents
    - Sources
    - Binaires (exécutables, librairies, …)

Il y a au minimum une activité de début et une activité de fin. Il doit y avoir un chemin reliant chaque activité à celle de fin.

Deux activités sont séparées par au moins un artefact. Une activitée ne peut être commencée tant que ses artefacts d'entrée n'existent pas.

Cascade

Ce modèle linéaire se base sur deux idées :

- modifier une étape a des conséquences sur les étapes suivantes, et donc
- une étape ne peut pas être débutée avant que la précédente ne soit achevée

Ce modèle comporte un nombre **fixe et prédéfini** de phases. Chacune des phases produit un certain nombre de livrables, eux aussi définis à l'avance. Chaque phase commence et termine à une date fixe. On ne peut passer à la phase suivante que lorsque les livrables de la phase courante sont validés.

Si une anomalie est détectée, on remonte d'une ou plusieurs phases en arrière.

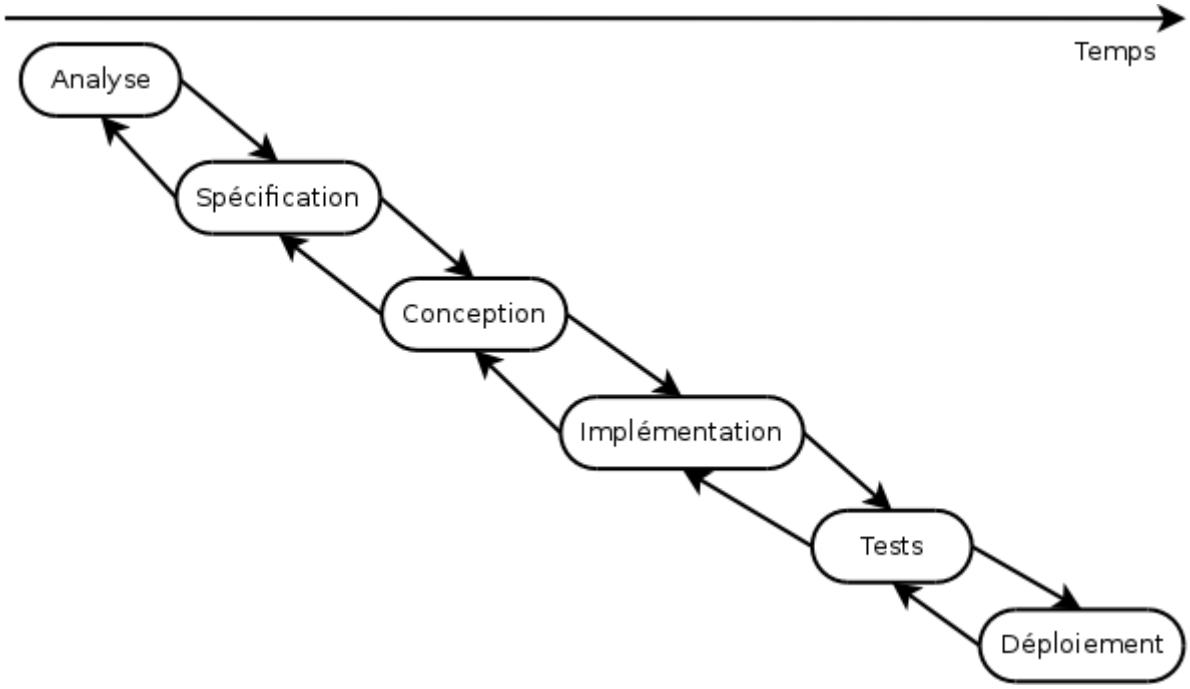


Figure 01:Cascade

Ce modèle suppose que l'on connaisse et maîtrise la plupart des exigences au lancement du projet.

Il nécessite d'accorder une attention très importante à la **documentation**. En particulier, il faut livrer proprement chaque document, puis attendre les retours et les commentaires sur cette livraison, puis faire évoluer ces documents pour y intégrer ces commentaires, et ainsi de suite, jusqu'à ce que chaque document soit accepté par toutes les parties. … Et ce, à chaque étape.

Avantages

Tout est **prévisible** : Les acteurs savent précisément ce qui doit être livré, à quelle date et ce que cela entraîne.

Inconvénients

- Le **temps nécessaire** pour obtenir un logiciel testable est important.
- Les phases les plus **risquées** (tests …) arrivent **à la fin** du cycle. Ce modèle est donc dans les faits très peu tolérant aux erreurs.
  - Que se passe-t'il si un besoin a été mal interprété ?
  - Et si un détail de conception s'avère inadapté lors de l'implémentation ou du déploiement ?
- La durée de vie d'un projet étant souvent de plusieurs années. Pourtant, ce modèle est très **intolérant** aux changements.
  - Que se passe-t'il si le besoin évolue ?
  - Et si la nature du marché change ?

Domaines d'application

Ce modèle peut néanmoins être adapté dans certains cas :

- Les domaines où il est impossible ou très coûteux de revenir en arrière. Par exemple, c'est le monde du BTP qui a donné naissance à ce modèle (peut-on construire un bâtiment avant d'avoir spécifié le terrain et conçu les plans ?).
- Les projets dont le périmètre est faible et la durée très courte. Dans de tels petits projets, le risque de retour en arrière est à priori faible.

Il est à déconseiller pour les nouveaux systèmes en raison des nombreux problèmes de spécification et de conception que la nouveauté entraîne.

En "V"

Ce modèle linéaire tente de mettre en évidence la **complémentarité** entre certaines phases. Chaque phase d'étude et d'analyse est **couplée** avec un phase de tests qui la valide.

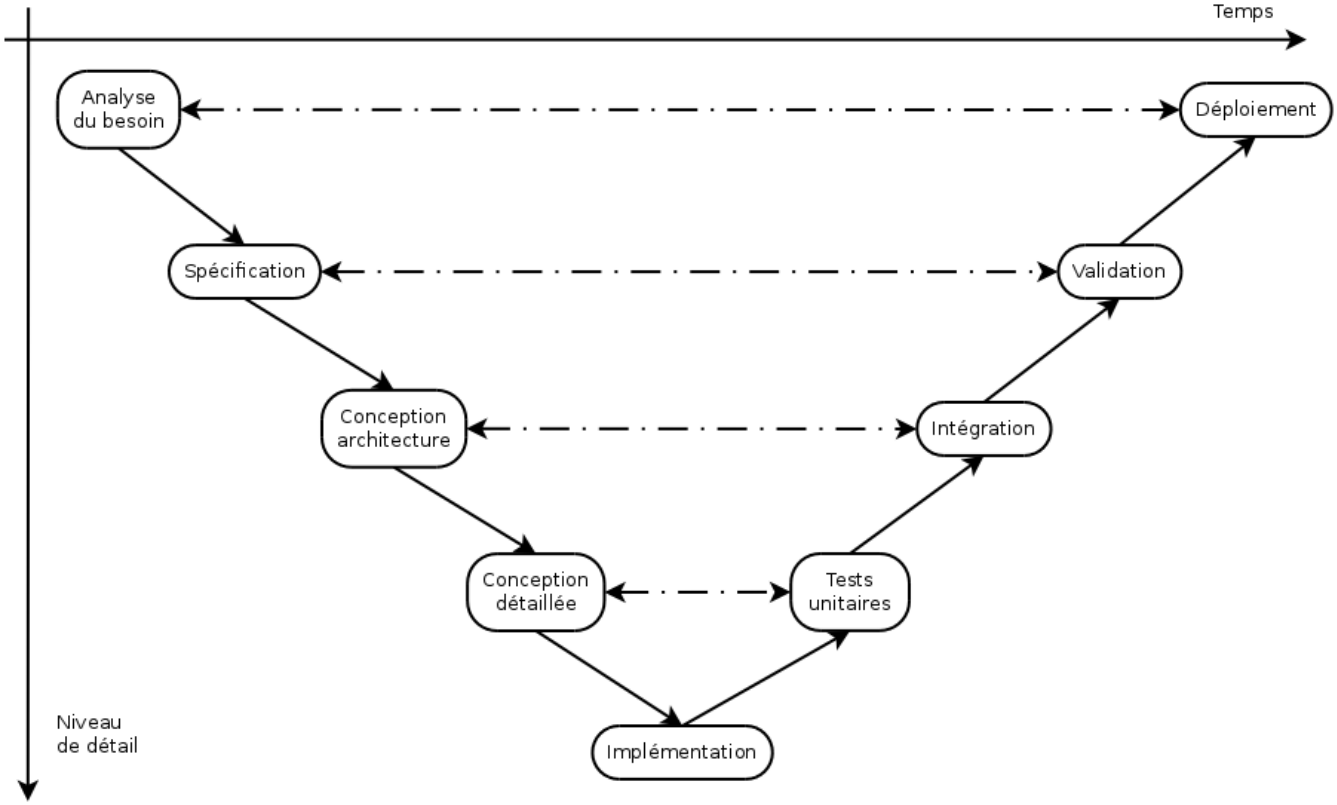


Figure 02:Cycle en V

Ce modèle est calqué sur la production industrielle classique. Il est donc de loin le plus utilisé dans le **domaine industriel**.

Avantages

- Modèle **éprouvé** :
  - Vaste taux d'usage en entreprise depuis les années 80
  - Supporté par de nombreux standards
  - Appuyé par de nombreux outils
- Les phases de test sont aussi importantes que les phases de réflexion. De plus, il est plus facile

de décrire de manière exhaustive comment tester une fonctionnalité au moment où celle-ci est conçue. La synergie entre analyse et description du test est profitable.

- Puisqu'il favorise la décomposition hiérarchique et fonctionnelle, il permet l'organisation du travail, des équipes et la maîtrise des coûts (voir la méthode COCOMO). Cela lui offre une bonne visibilité. Le suivi de projet est facilité.

Inconvénients

- Ne fait qu'amenuiser les inconvénients du modèle en cascade sur lequel il est basé. Le principal problème restant le **manque de souplesse**.
- Différence entre la théorie et la pratique. Les phases durant lesquelles le niveau de détail est accru, en particulier celles de spécification détaillées et d'implémentation, permettent parfois de se rendre compte que les analyses issues des phases précédentes sont incomplètes ou carrément irréalisables en l'état.

Domaines d'application

Ses inconvénients persistants en dépit de son vaste taux d'utilisation font que le cycle en V est davantage un **idéal** vers lequel certains aimeraient tendre mais est rarement appliqué tel quel.

Il est donc en général utilisé pour de grands projets industriels avec plus ou moins de bonheur.

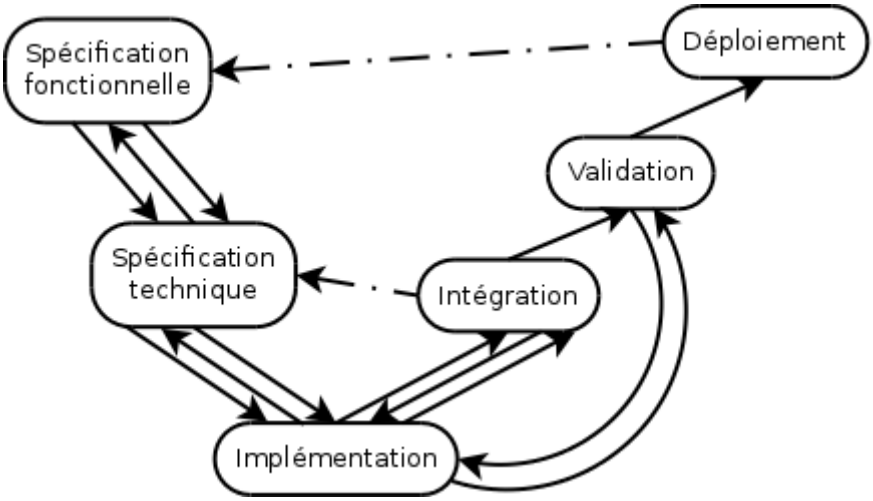


Figure 03:Cycle en V, trop fréquemment

En spirale (de Boehm)

Inspiré par la Roue de Deming, ce modèle itératif met l'accent sur la **gestion des risques**. Il reprend les étapes du cycle en V, mais prévoit la création de versions successives au cours de **cycles** de développement.

Chaque cycle peut être découpé en 4 étapes distinctes représentées par l'acronyme PDAC (*Plan-Do-Check-Act*) :

- Planifier
  - Analyse des besoins

    - Détermination des objectifs
    - **Analyse des risques**
    - Analyse des alternatives
- Développer
    - Spécification
    - Conception
    - Implémentation
    - Tests unitaires
- Contrôler
    - Intégration
    - Validation
- Ajuster
    - Livraison
    - Définition du prochain cycle

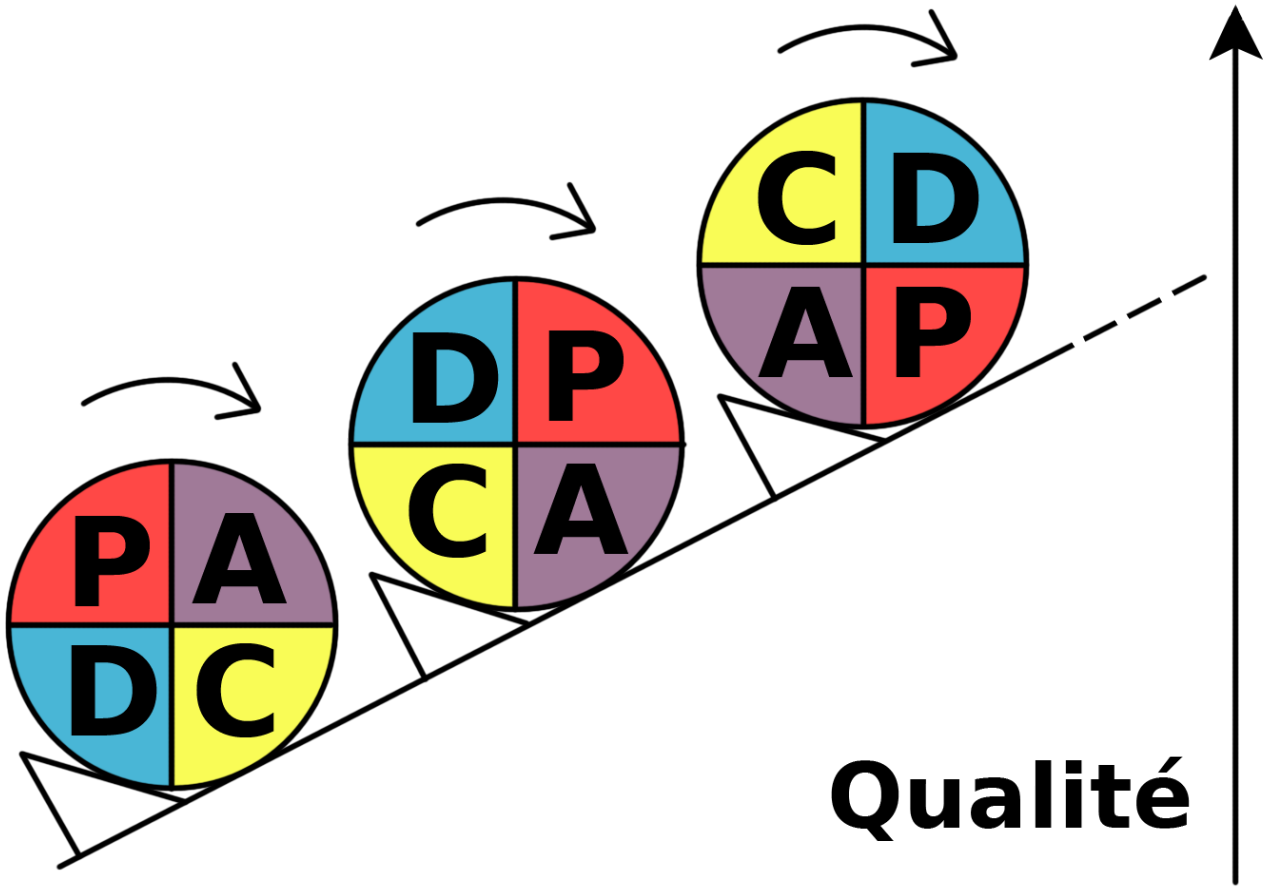


Figure 04:Roue de Deming

Chaque cycle ne se base pas forcément sur les artefacts livrés par le cycle précédent : différents risques peuvent être addressés l'un après l'autre.

Comme dans tout modèle itératif, le nombre de cycles n'est pas déterminé à l'avance.

Avantages

- Maîtrise des risques : on se concentre sur les aspects les plus incertains du développement.
- Mise en avant des objectifs de qualité.
- Intègre maintenance et développement.
- Possibilité d'intervertir l'ordre de certains cycles indépendants
- Compatible à de nombreuses approches et outils existants.

Inconvénients

- Nécessite une vraie expertise sur l'évaluation des risques : la nature des risques peut être différente d'un cycle à l'autre.
- Mettre en place ce modèle assez complexe nécessite une grande expérience.
- Les premiers cycles de la spirale ne produisent en général pas de solution exploitable.
- Verbosité inadaptée pour les petits projets ou des domaines suffisamment connus. Dans le pire des cas, la stricte application de ce modèle (en particulier l'évaluation des risques) engendre un coût plus élevé que la réalisation du projet elle-même.

Domaines d'application

Ce modèle est logiquement approprié aux projets où le périmètre et le risque sont importants.

Développement exploratoire

Le principe de ce modèle itératif est de perfectionner à plusieurs reprises (on parle d'itérations) la spécification du projet, ainsi que son développement et sa validation. L'objectif est de **collaborer** avec le client et de **raffiner** de plus en plus la solution.

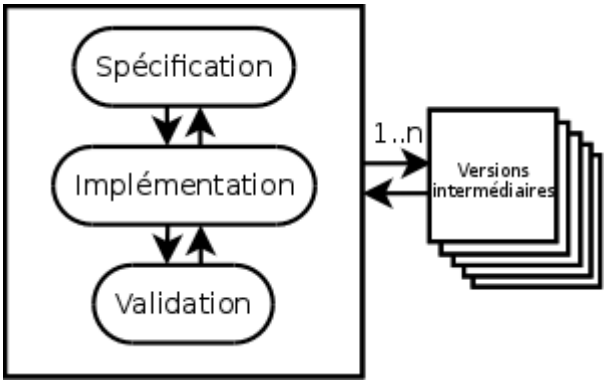


Figure 05:Modèle exploratoire

En général, c'est le délai accordé au projet qui décide du nombre d'itérations.

Avantages

- Minimise le risque pour les nouvelles applications : il est possible d'explorer certaines spécificités du systèmes, de les évaluer afin d'opter pour la meilleure stratégie.
- Le résultat est souvent pleinement compréhensible et satisfaisant pour le client.
- Théoriquement livrable à chaque itération :
  - Favorise les tests et les validations intermédiaires.
  - Il est possibile d'arrêter le processus n'importe quand.

Inconvénients

- Tentation d'abréger le processus et de se contenter d'une solution incomplète.

- Peu structuré, donc impossible à appliquer tel quel à grand échelle.
- Faible visibilité pour les intervenants extérieurs.

Domaines d'application

Le domaine d'application de ce modèle est donc les petits systèmes (ou parties d'un système) interactifs dont le résultat peut être visible rapidement. En particulier, les IHM.

Évolutif

Ce modèle itératif vise a réaliser une version provisoire de la solution aux besoins connus afin de pouvoir la mettre en exploitation au plus vite. De **nouvelles versions** seront ensuite déployées, chacune remplaçant la version précédente. Chaque version apportera de nouvelles fonctionnalités ou modifiera les fonctionnalités existantes.

Les versions intermédiaires sont réalisés avec tous les principes de qualité d'une version finale : ce ne sont donc *pas* des prototypes jetables !

Modèle incrémental

La première version constitue un système partiel. Chaque nouvelle version ajoute une nouvelle fonctionalité complète.

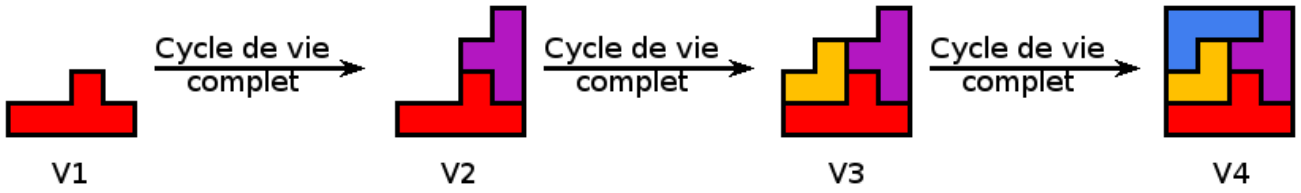


Figure 07.1:Incrémental, par prototypage vertical

Modèle itératif

Le système doit être dès le départ découpé en fonctionalités bien définies. La première version constitue une coquille complète du système. Chaque fonctionnalité qui n'est pas implémentée est remplacée par un **bouchon**. Chaque nouvelle version modifie ou améliore une fonctionalité.

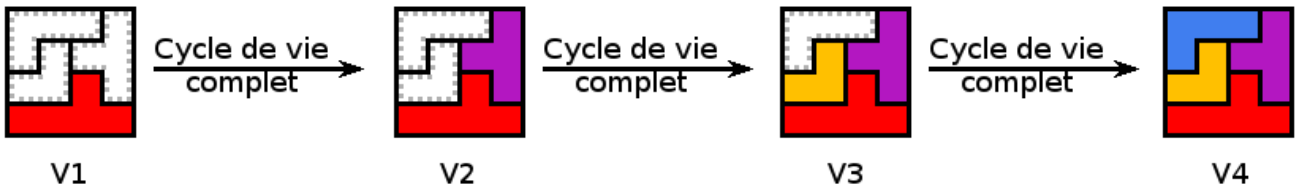


Figure 07.2:Évolutif, par prototypage horizontal

Avantages

- Augmente la **compétitivité** en réduisant le temps de mise sur le marché.
- Formation précoce des utilisateurs.
- Détection précoce des imprévus.

Inconvénients

- Le processus de deploiement de chaque version doit être maîtrisé.
- Risque d'aboutir à un parc hétérogène :
  - Assurer la tracabilité de chaque version.
  - Assurer la tracabilité de chacun de leurs composants.
- La manière d'obtenir un retour des utilisateurs doit être maîtrisée.
- Risque de remise en cause du noyau assurant les fonctionnalités de base.

Domaines d'application

Ce modèle est approprié à tout domaine **fortement concurrentiel**, et où les utilisateurs sont disposés à utiliser un produit incomplet.

Unifié

Le processus unifié (*UP*: *Unified Process*) est un modèle itératif, **générique**, et **orienté-objet**. Il complète les modèles UML, et tente de formaliser une approche plus flexible des réalités de l'entreprise que supportent mal le modèle en cascade ou le cycle en V.

Différentes implémentations de cette méthode ont vu le jour : modèle en "Y" (*2TUP*), *RUP*, *AUP*, *XUP*.

Ce modèle voit le produit comme la somme de ses **fonctionnalités**. Chaque itération produit un certain nombre de fonctionnalités.

Le processus s'arrête quand les fonctionnalités implémentées satisfont le client. Le nombre de fonctionnalités final n'est pas connu à l'avance ; à l'inverse, la situation est examinée à la fin de chaque itération.

- Expression du besoin. Ce besoin peut varier à chaque itération.
- Déroulement de l'itération. Chaque itération doit se concentrer sur l'essentiel ; quelles fonctionnalités sont essentielles pour cette itération dépend du besoin exprimé.
    - Spécification
        - Décision de quelles fonctionnalités vont être développées, et desquelles vont être laissées de coté.
        - Traduction du besoin en langage technique.
    - Développement : Réalisation de ce qui a été spécifié.
        - Implémentation
        - Tests unitaires
        - Tests d'intégration
    - Validation client : vérification que le résultat de l'itération est conforme au besoin exprimé.
    - Évaluation : comprendre l'état actuel du projet
        - Analyse des difficultés rencontrées
        - Plan d'amélioration.
- Déploiement : mise à disposition du client des artefacts qu'il a validés.

Avantages

- Pragmatisme : chaque itération se consacre sur l'essentiel. Ce processus évite de perdre du temps sur des tâches sans valeur ajoutée pour le client. Cela permet de s'approcher d'un mode de fonctionnement optimal.
- Réactivité offerte par des itérations courtes.
- Met l'accent sur la satisfaction du client.

Inconvénients

- Le coût en temps pour le client n'est pas à négliger.
- Moins facile à mettre en place qu'il n'y parait.
- Comment justifier des tâches nécessaires à l'organisation mais sans bénéfice immédiat pour le client ?
- Un nombre étonnant de freins d'ordre organisationnel ou personnel peuvent apparaître. Cette méthode de travail peut en effet être difficile d'approche pour certains (résistance au changement, difficulté à communiquer, etc).

Domaines d'application

Les modèles implémentant ce processus nécessitent que le besoin client soit clairement exprimable. Il est adapté aux entreprises ouvertes d'esprit et où la communication inter- et intra- équipe est bonne.