ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

04
Chapitre

Une méthode de gestion de projet : Agilité

1I2AC1 : Génie logiciel et Conception orientée objet

Régis Clouard, ENSICAEN - GREYC

« Je ne suis pas un grand programmeur.
Je suis juste un bon programmeur avec de bonnes habitudes. »
Kent Beck (créateur de la méthode X-Programming)

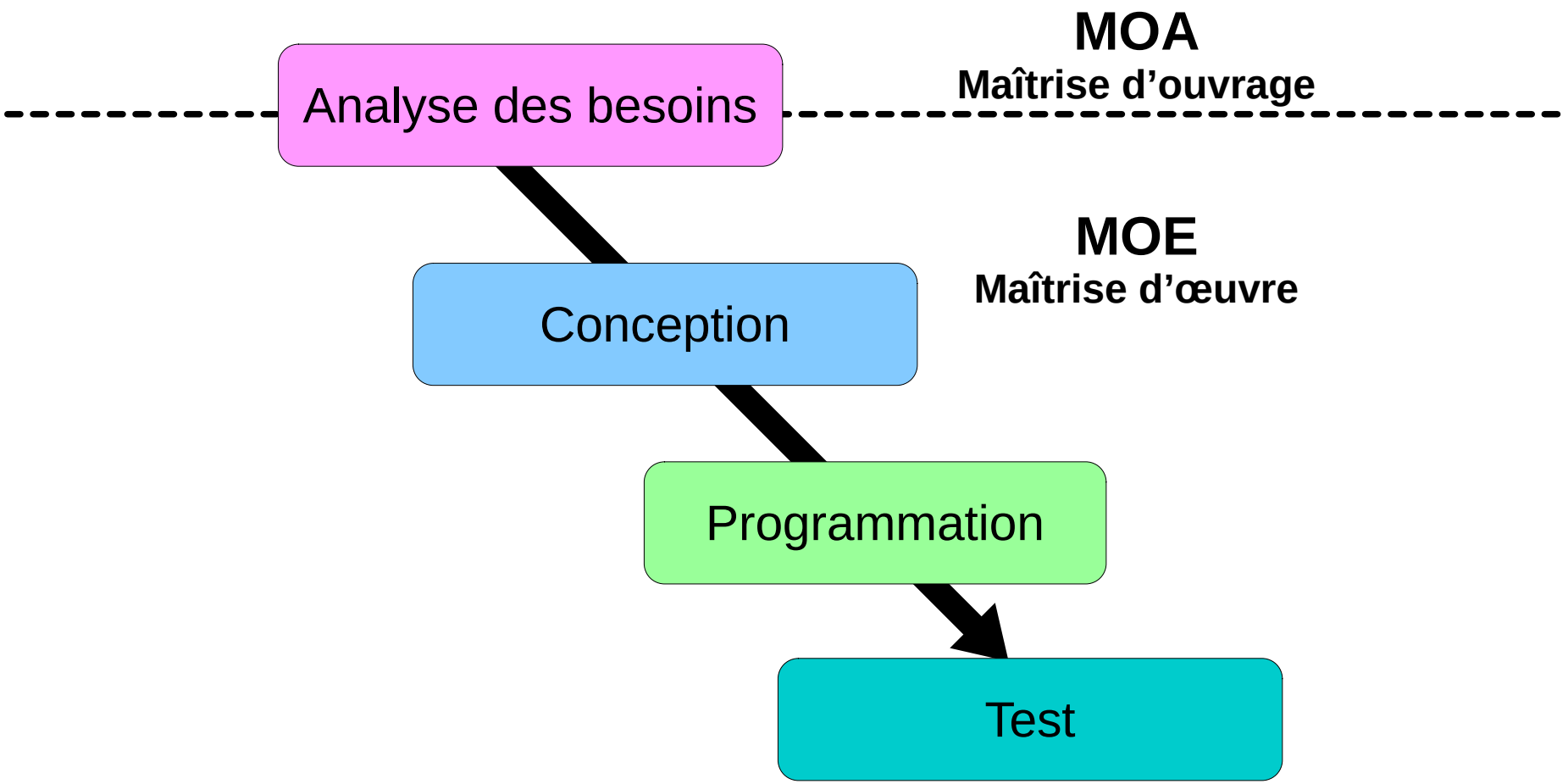
Objectif de ce chapitre

- Se questionner sur la gestion de projet pour le cas spécifique du génie logiciel
  - A mettre en parallèle avec les cours de gestion de projet qui reposent sur des méthodes prédictives
  - Enseigné plus précisément en 2A : Gestion de projet Agile

Rappel

- Méthodes de gestion de projet prédictives
  - Exemples : cycle en cascade, cycle en V
  - Très efficaces pour la majorité des projets d'ingénierie
  - Mais pas du tout adaptées au génie logiciel

Rappel : Le cycle de vie en cascade

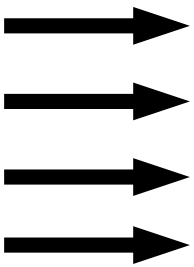


Plan du chapitre

1
De la prédiction
vers l’agilité

Pratique actuelle : agilité

- **Méthode prédictive**
  - Suivi d'un plan
  - Documentation exhaustive
  - Processus et les outils
  - Négociation contractuelle



- **Méthode agile**
  - Adaptation au changement
  - Un logiciel qui fonctionne
  - Les individus et leurs interactions
  - Collaboration avec les clients

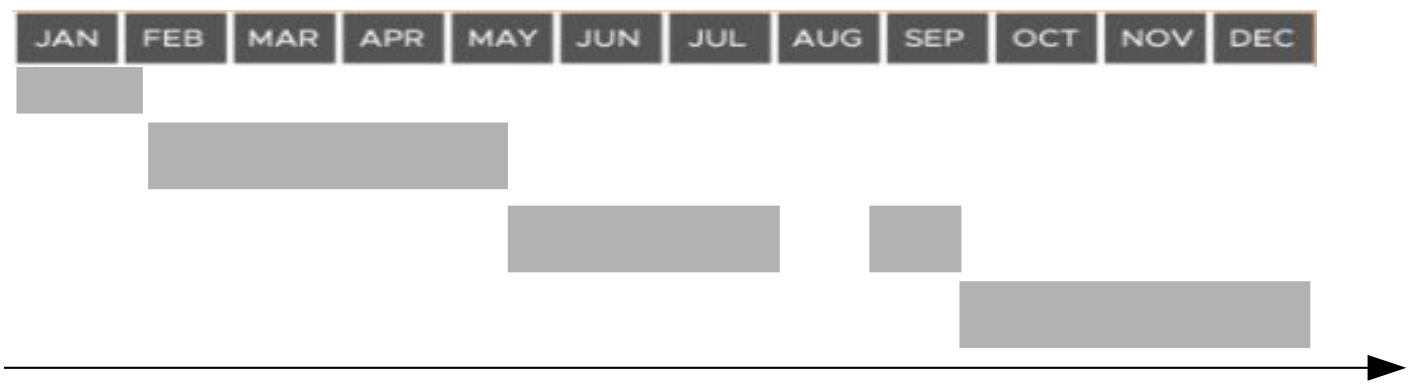
- **Remarque**
  - « Bien qu'il y ait de la valeur dans les éléments situés à gauche, notre préférence se porte sur les éléments qui se trouvent à droite. »

1- Mauvaise pratique
Suivre un plan à long terme : le cycle de vie en cascade

Critique du cycle de vie en cascade

- **Planification prévisionnelle du projet en phases successives**
  - Intérêt : métiers spécifiques à forte expertise
  - Quels problèmes posent cette planification ?

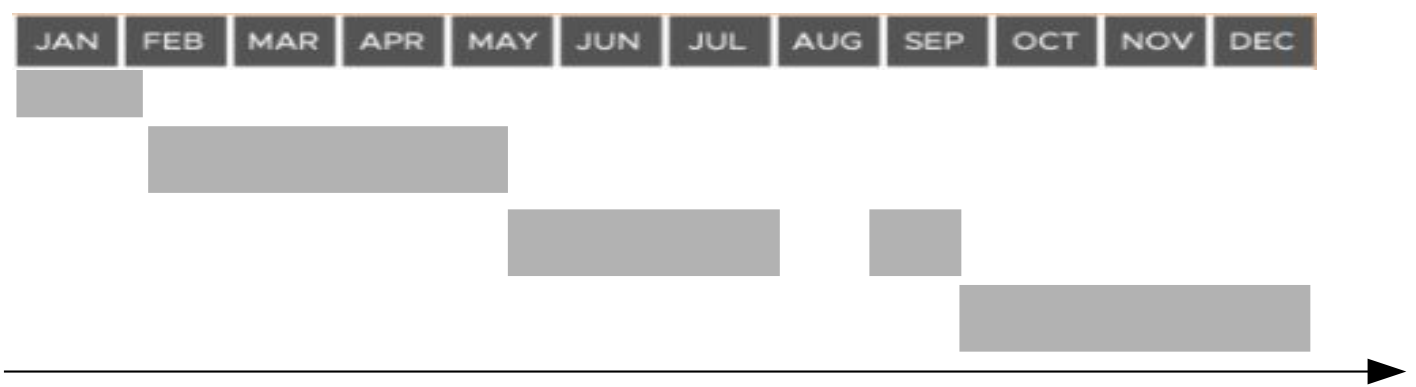
Analyse des besoins
Conception
Codage
Tests



Critique du cycle de vie en cascade

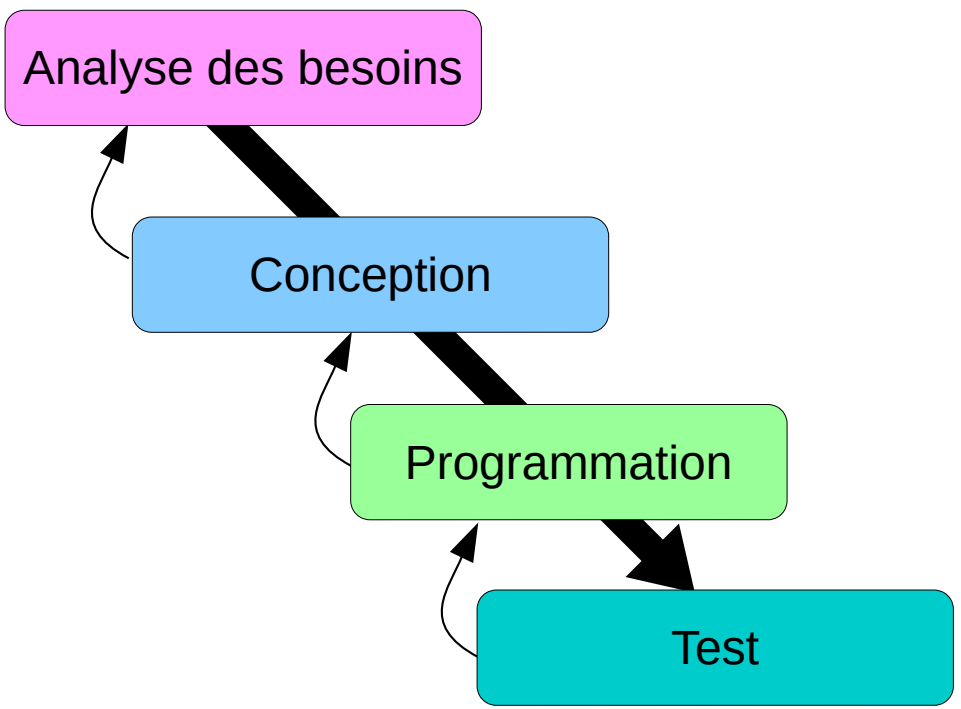
- **Planification prévisionnelle du projet en phases successives**
  - **Problèmes**
    1) Impossible de tout **prévoir** → dépassements systématiques
    2) **Aucune adaptation** aux imprévus ou changements pourtant inévitables
    3) **Dépendance** entre les phases. Le réalisation d'une phase peut remettre en cause une phase précédente

Analyse des besoins
Conception
Codage
Tests



Une amélioration : boucle de retour

- Ajouter un retour vers la phase précédente
  - Mais, allonge mécaniquement la durée des phases avec une prédiction encore plus difficile à estimer que la durée des phases



Bonne pratique

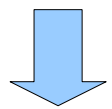
1. Renoncer à la planification à long terme

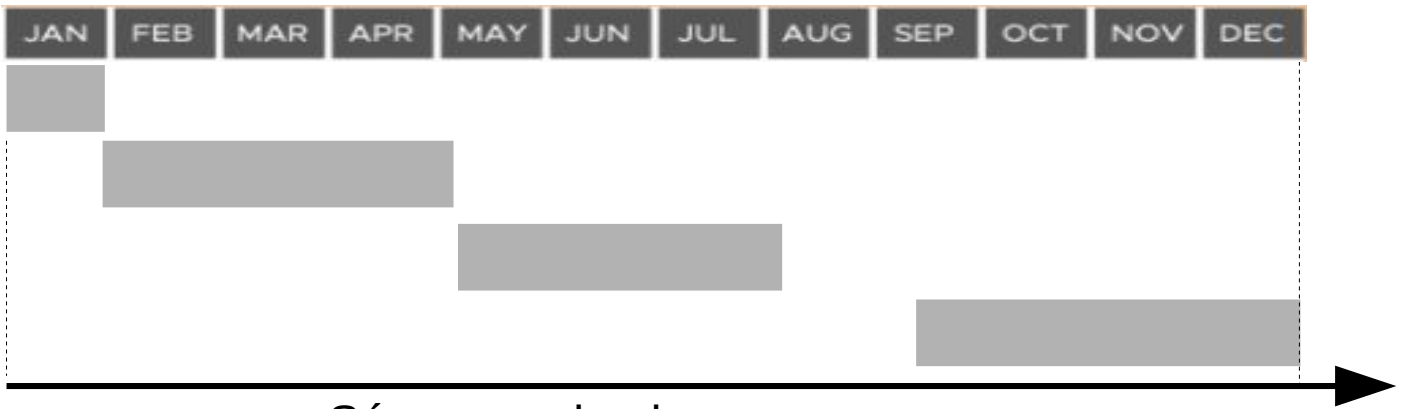
→ Cycle de vie itératif

Cycle de vie itératif

- Cycle de vie en cascade

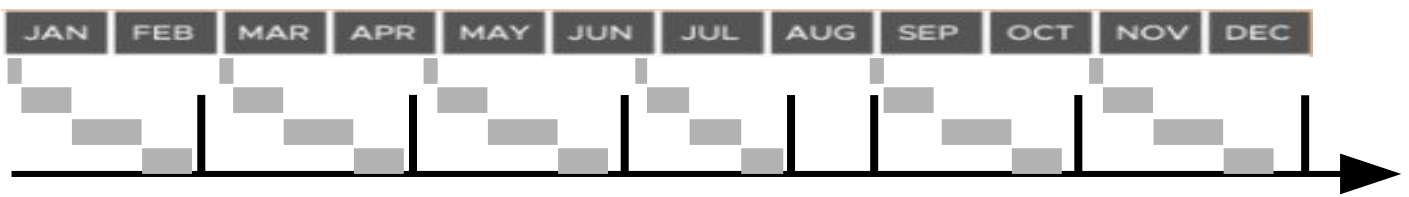
Analyse des besoins
Conception
Codage
Tests





Séquence de phases

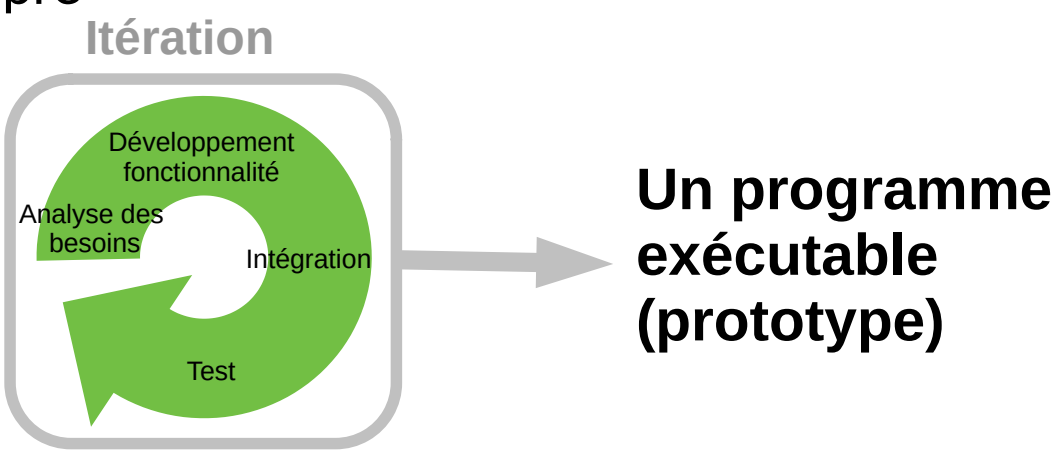
- Cycle de vie itératif



Séquence d'itérations de même durée

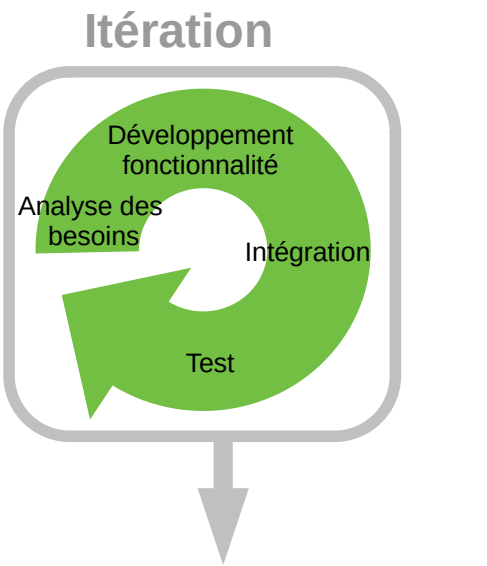
Itération

- Un mini-projet de 2 à 4 semaines
  - On retrouve toutes les activités d'un projet : analyse des besoins, développement, test, intégration et déploiement
- Livrable :
  - Un prototype exécutable et testé
  - Le prototype est validé par le client
  - Le code est propre



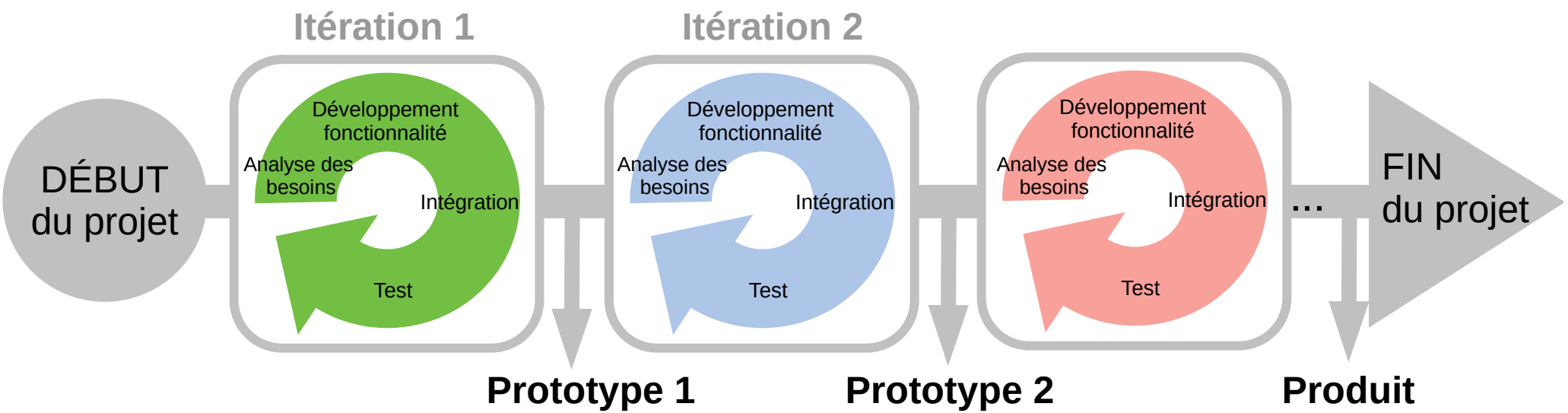
Itération

- Quelle méthode de gestion de projet pour une itération ?
  - Toute, voire aucune
  - Même une méthode à l'arrache peut fonctionner (https://www.la-rache.com/)



Cycle de vie itératif

- Projet : une suite de mini-projet



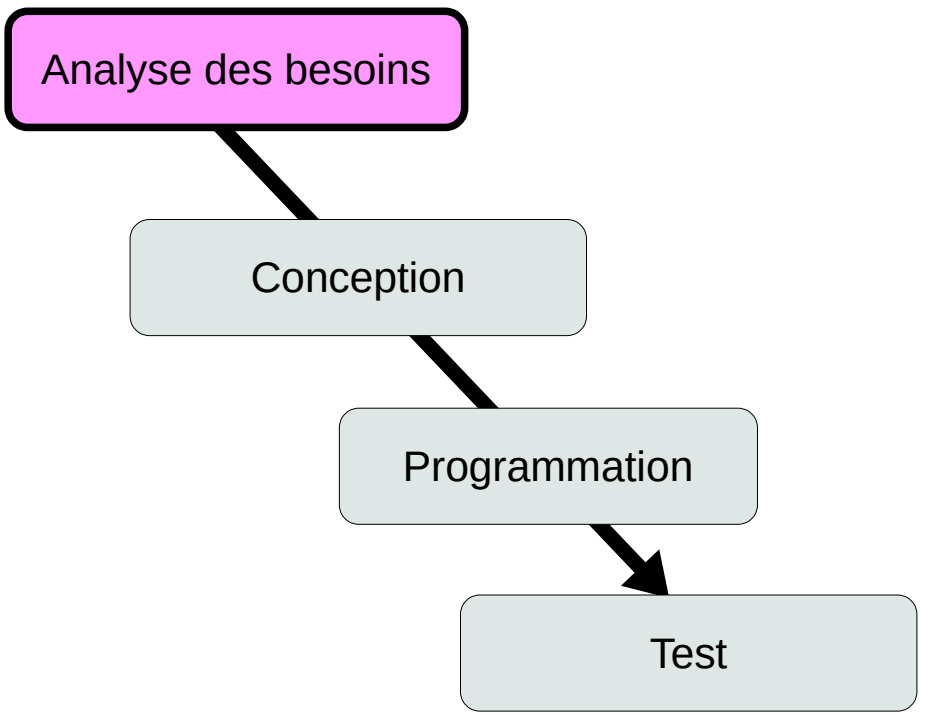
Avantages du cycle de vie itératif

- On avance à petits pas testés
- Si on rate un pas, on n'a raté qu'un pas, sans grosses conséquences
- Chaque pas est validé avec le client
- La revue d'itération permet de réviser les besoins
  - Adaptation aux changements
- Chaque pas est testé
- On peut arrêter le projet n'importe quand, il y aura toujours une version opérationnelle
- Plus de métiers spécifiques. Un seul métier : ingénieur développeur

2- Mauvaise pratique
Analyse des besoins au début du projet

Critique du cycle de vie en cascade

- **Analyse des besoins au début du projet et « effet tunnel »**
  - Intérêts : formalisation et contractualisation
  - Quels problèmes posent cette organisation ?



Critique du cycle de vie en cascade

- **Analyse des besoins au début du projet et « effet tunnel »**
  - **Problèmes**
    1) **Non satisfaction des clients** : les besoins ont été mal compris ou ont changés entre-temps
    2) Effet tunnel

  - **Inflation de fonctionnalités** : obligés de tout définir au début du projet, les client inventent des besoins
    - Résultat : 45 % des fonctionnalités développées ne sont jamais utilisées (*Standish group, 2006*)

Bonne pratique
2. Éviter l'« effet tunnel »
→ Cycle de vie incrémental

Cycle de vie incrémental

- Le logiciel est construit par **incrémentation** en profitant de la malléabilité du logiciel
  - Impliquer le client dans la définition des besoins
    - Faire un prototype qui soit évaluable par le client

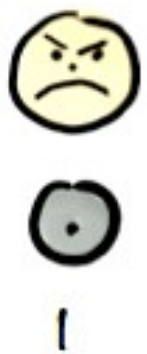
- Incrément :
  - **MVP** (***Minimum Viable Product***) : Un prototype avec juste assez de fonctionnalités pour **satisfaire les premiers clients** et fournir une **rétroaction** pour poursuivre le développement
  - **POC** (***Proof Of Concept***) : Un prototype pour **tester** quelque chose sans retour concret vers le client.

Incrément : MVP

- Exemple
  - Besoin : le client veut une voiture (*métaphore*)

Incrémental : Pas comme ça !

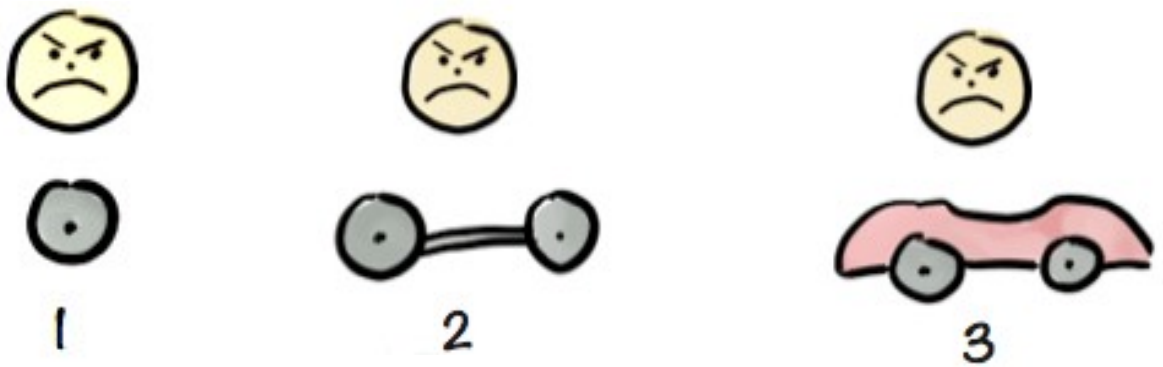
- Itération 1



  - « Hé monsieur, voici notre première itération, un pneu avant. Qu'en pensez-vous ? »
  - « Mais qu'est ce que vous voulez que je fiche d'un pneu ? »

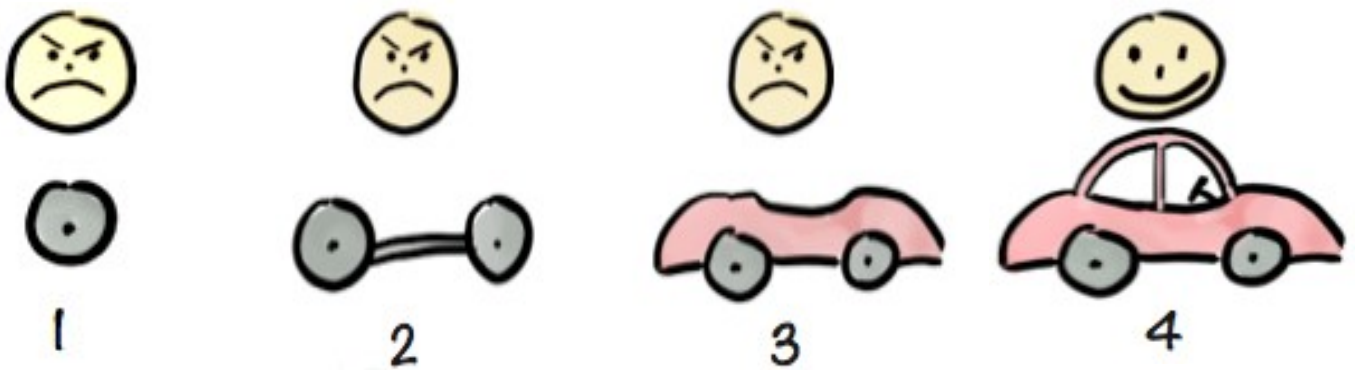
Incrémental : Pas comme ça !

- Itérations 2 et 3



Incrémental : Pas comme ça !

- Itération 4 finale



  - « Merci, Enfin ! Pourquoi n'avez-vous pas simplement livré ça directement en sautant toutes les autres livraisons inutiles ? »

Incrémental : Comme ça !

- Besoin : le client veut une voiture
- Itération 1 : discussion initiale pour comprendre le besoin sous-jacent
  - On cherche le **pourquoi** et pas le **quoi**
  - « J'ai besoin de pouvoir me rendre plus vite d'un point A à un point B. »

MVP 1

- Ce qui est appris avec l'exécution de ce MVP 1 :
  - Parfait pour aller du bureau à la salle à café
  - Mais, le véhicule est instable

Incrémental : Comme ça !

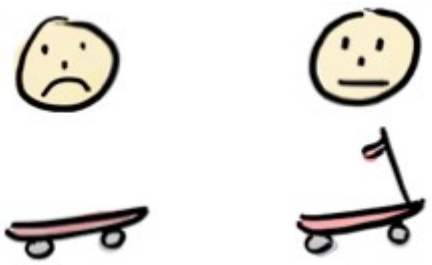
- Et pourquoi pas un simplement un ticket de bus

MVP 1

- Dans ce cas, le projet s'arrête là avec la solution plus adéquate à moindre frais. Le client repart comblé

Incrémental : Comme ça !

- Itération 2

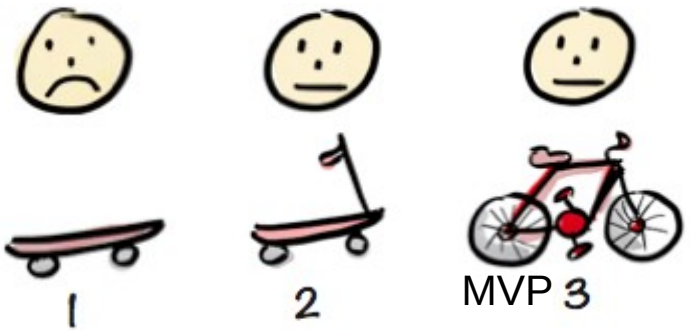


MVP 2

- Nouvelle chose apprise :
  - Difficile de parcourir de plus longues distances entre deux bâtiments de l'école à cause des petites roues et de l'absence de freins

Incrémental : Comme ça !

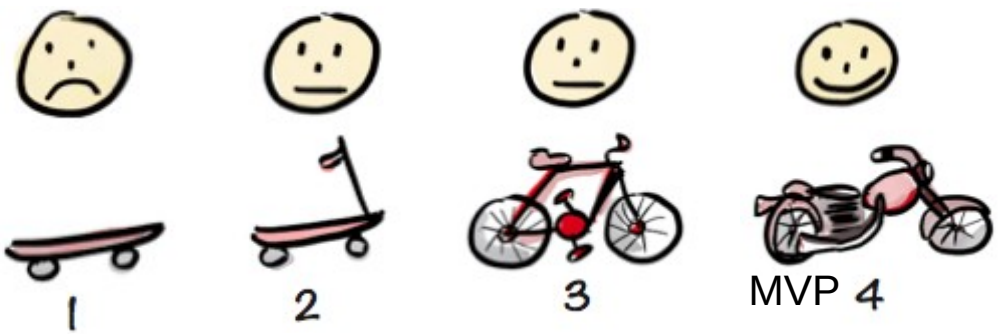
- Itération 3



- Nouvelles choses apprises :
  - Le client peut se déplacer sur le campus à toute vitesse
  - Le client aime le contact de l'air frais sur son visage
  - Mais, le vélo fait suer

Incrémental : Comme ça !

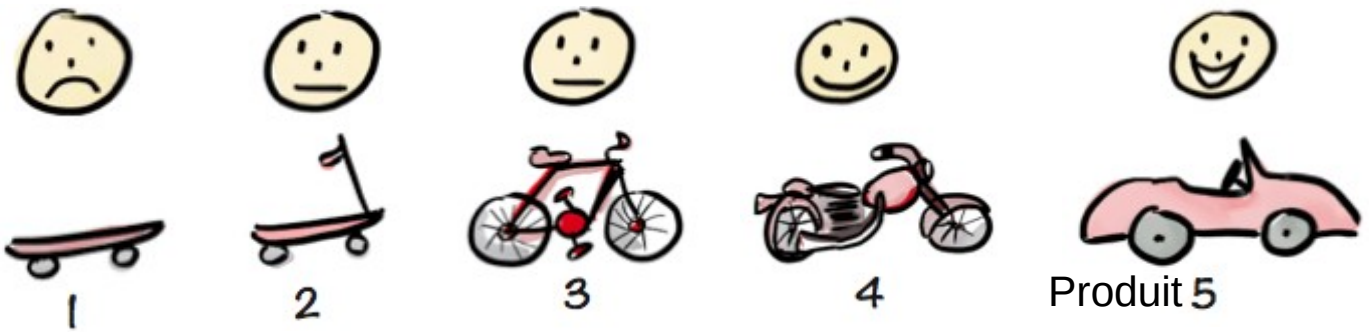
- Itération 4



- Nouvelle chose apprise :
  - On apprend qu'il peut être amené à transporter du matériel encombrant
- On pourrait s'arrêter là si le client le désire

Incrémental : Comme ça !

- Itération 5 finale



  - Nous avons appris en cours de route que le client apprécie l'air frais sur son visage, donc nous avons fait un véhicule décapotable, léger mais qui peut transporter du matériel

MVP : la métaphore de la part de gâteau

- MVP : un peu de tout

C’est quoi votre skateboard ?

- **Application de course orientation**
  - Problème sous-jacent
    - Visualiser la carte d’orientation et comptabiliser les balises visités
  - MVP 1
    - Affichage d’une carte stockée en mémoire + validation manuelle des balises par bouton
    - Déjà utilisable pour une vraie course

C’est quoi votre skateboard ?

- **Domotique**
  - Problème sous-jacent
    - Piloter des périphériques en ligne
  - MVP 1
    - Allumage d’une lumière installée sur un client à partir du serveur

Avantages du cycle de vie incrémental

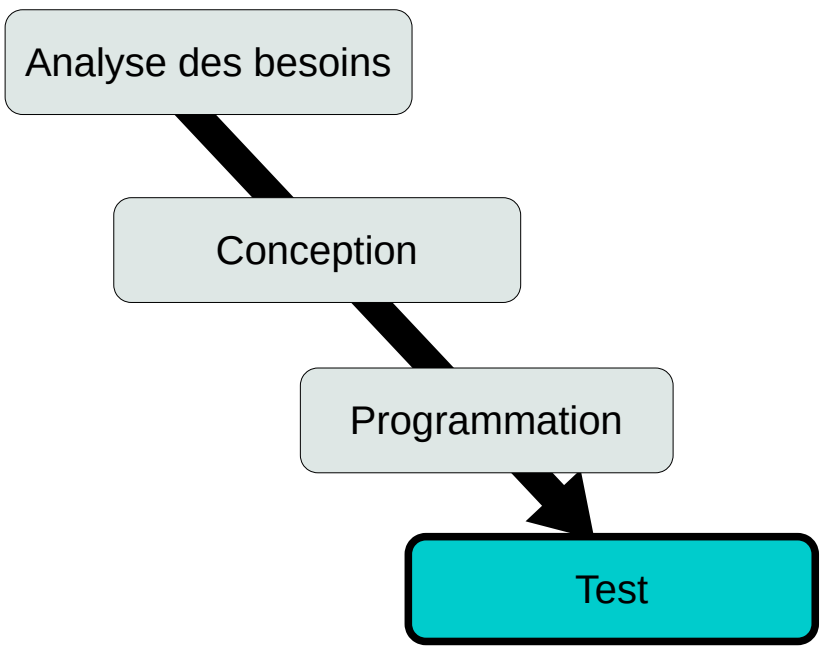
- Fournir une version testable qui permet d'apprendre sur le problème
- Donner rapidement de la valeur au produit
- Le manque de temps conduit à un déficit de fonctionnalités et pas à un échec total du projet ni à un projet non testé
- Le client voit une évolution rassurante et constante de son produit
- Impliquer le client dans le développement alors qu'il n'est pas informaticien

3- Mauvaise pratique

Test en fin de projet

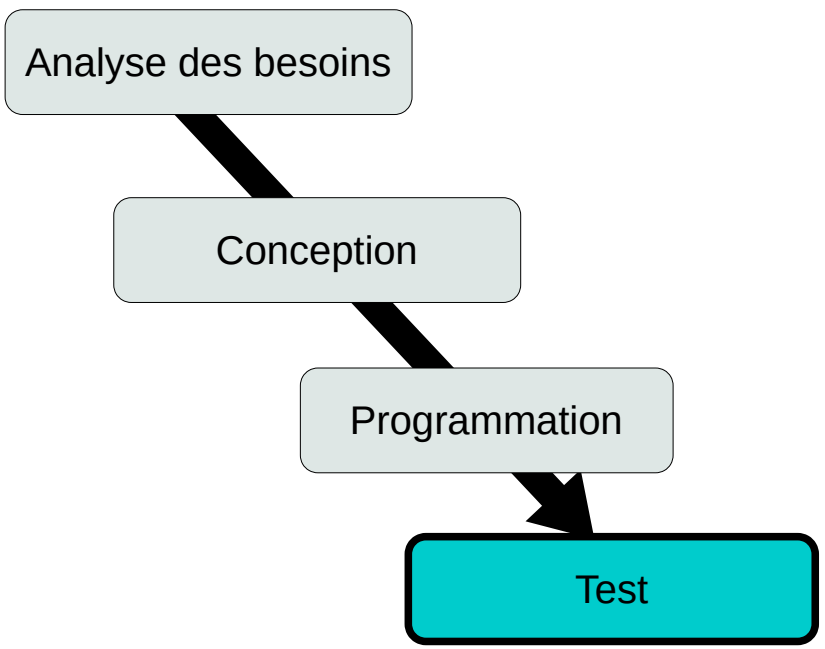
Critique du cycle de vie en cascade

- **Test en fin de projet**
  - Intérêt : tester le logiciel final avec toutes les contraintes
  - Quels problème posent cette organisation ?



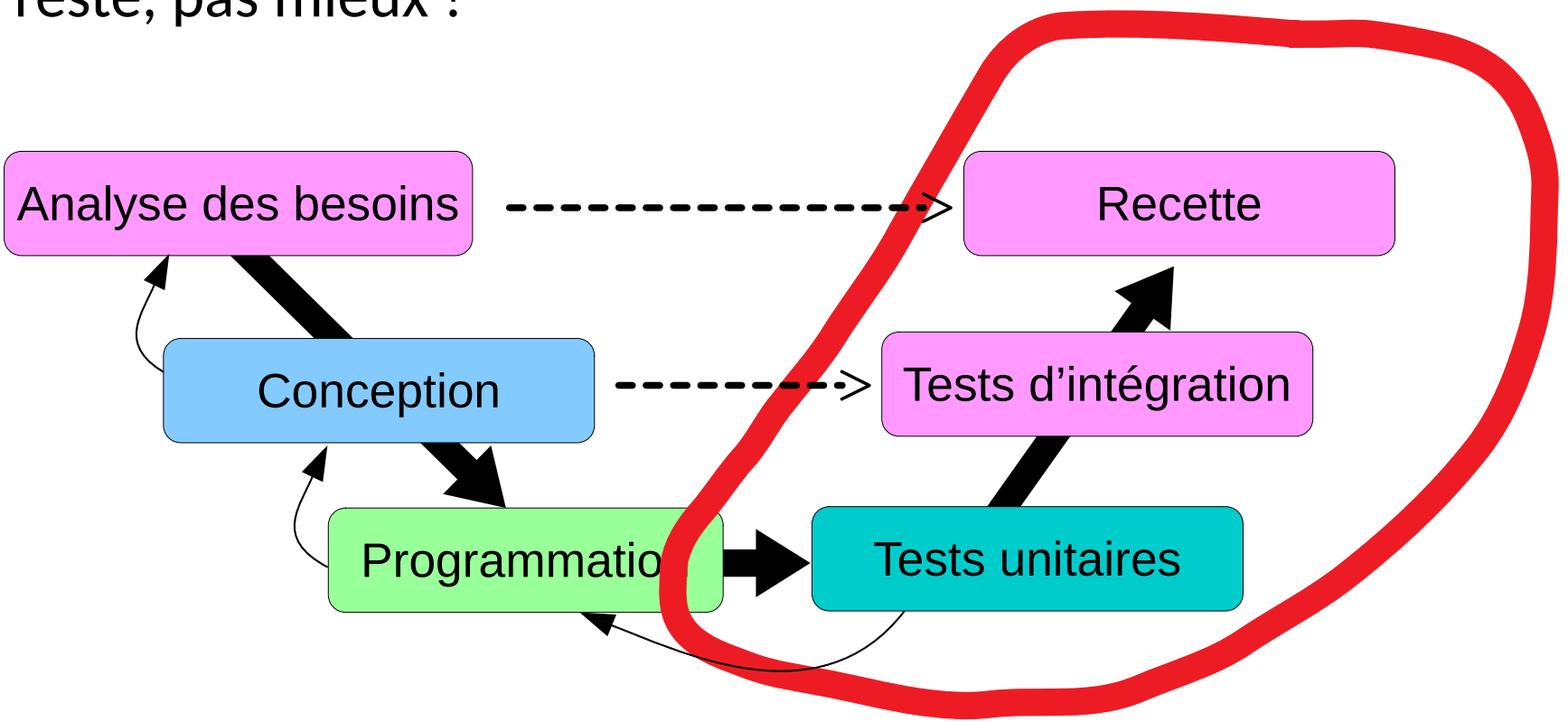
Critique du cycle de vie en cascade

- **Test en fin de projet**
  - **Problèmes**
    1) La phase de test devient la **variable d'ajustement** du temps
    2) Le développeur a **oublié** ce qu'il faut tester et comment le tester



Une amélioration : le cycle de vie en V

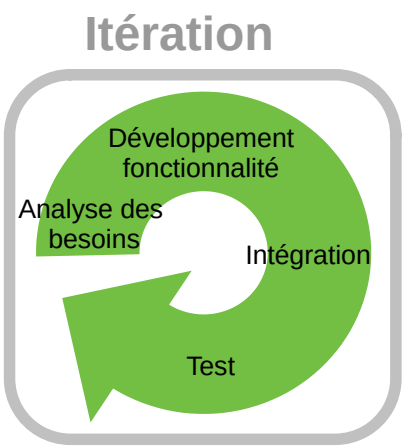
- Amélioration surtout pour la définition des tests
- Pour le reste, pas mieux !



Bonne pratique
3. Tester tout au long du développement
→ cycle itératif

Tester chaque incrément

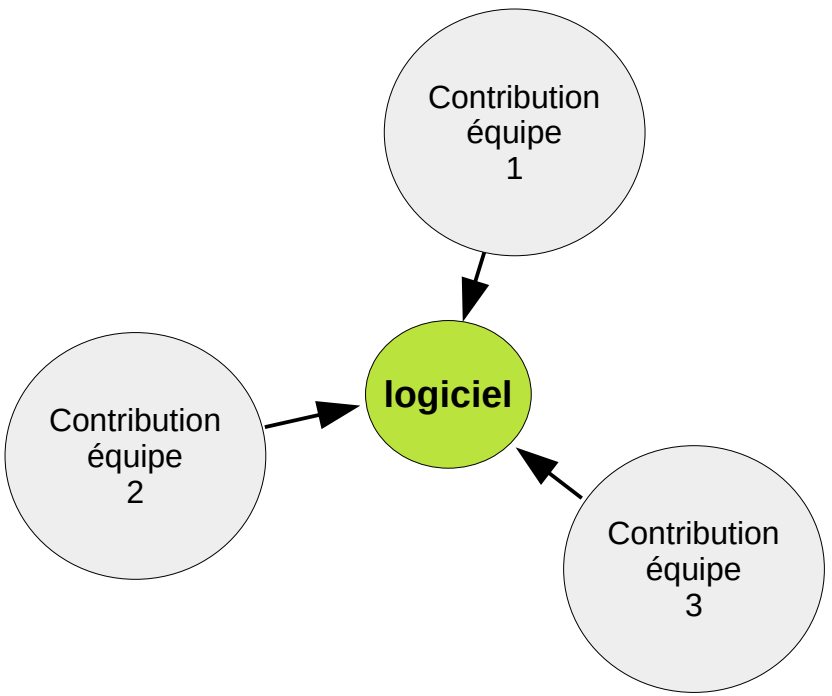
- Chaque développeur teste ses développements avant l'intégration
- Vérifié au moment de l'intégration du développement d'un développeur dans le logiciel commun



4- Mauvaise pratique
Intégration finale

Critique du cycle de vie en cascade

- **Parallélisation du développement**
  - Une des ambitions de la planification est d'identifier les tâches parallélisables
    - Le logiciel final résulte de l'intégration des codes de chaque partie
  - Quel problème pose cette intégration finale ?

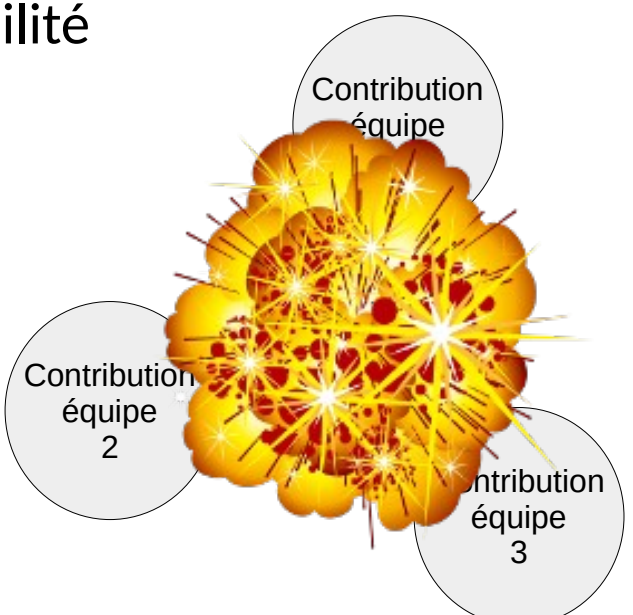


Critique du cycle de vie en cascade

- **Parallélisation du développement**
  - **Problème**
    - **Intégration avec « Effet big bang »**

      1) Conflits

      2) Incompatibilité



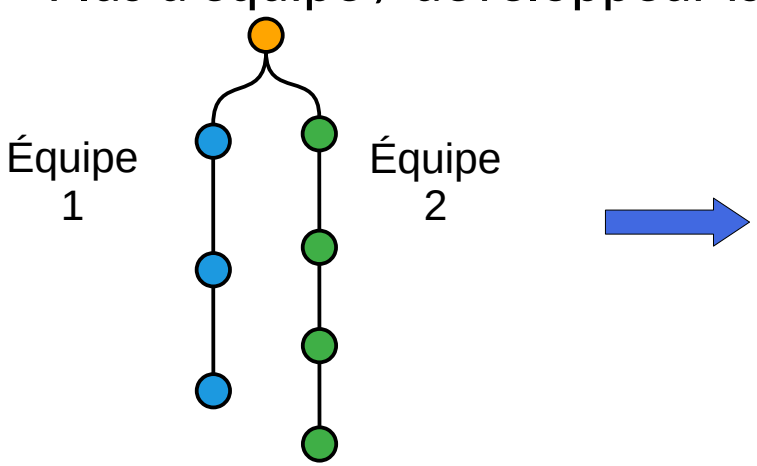
Exemple d’intégration « big bang »

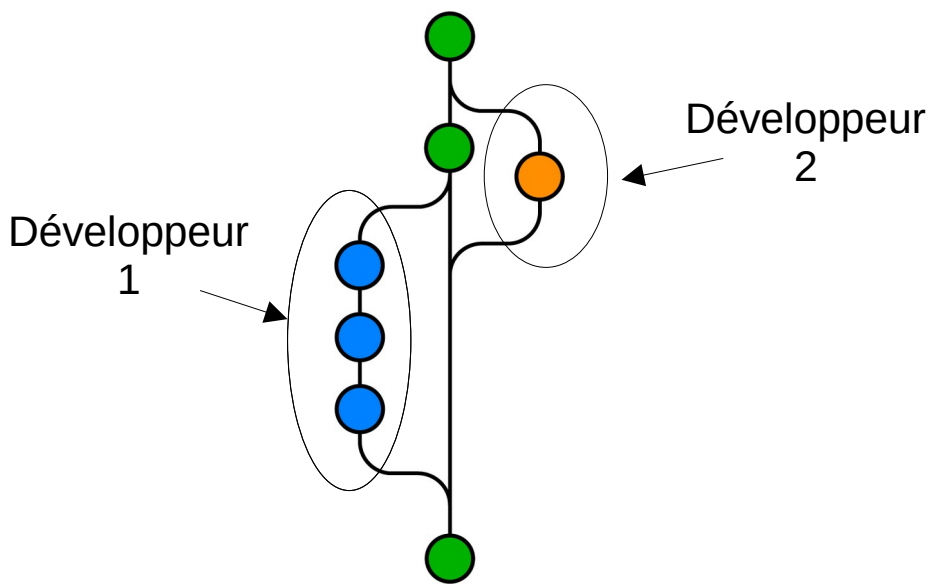
- Perte de la sonde Mars Climate Orbiter (1999) lors de son entrée dans l'orbite de Mars
  - C'est une **erreur de traduction entre systèmes d'unités anglo-saxons et métriques** pour le calcul de sa trajectoire qui est à l'origine de la défaillance. L’alunissage était réalisé par la coopération de **deux composants**, l’un réalisé par **Lockheed Martin** (qui fonctionnait avec le système anglo-saxon) et l’autre par la **NASA** (qui fonctionnait avec le système métrique). L’erreur n’a pas été mise à jour lors de l’intégration finale.
  - Coût de **327 millions** de dollars.

Bonne pratique
4. Intégration tout au long du développement
→ Intégration continue

Intégration continue

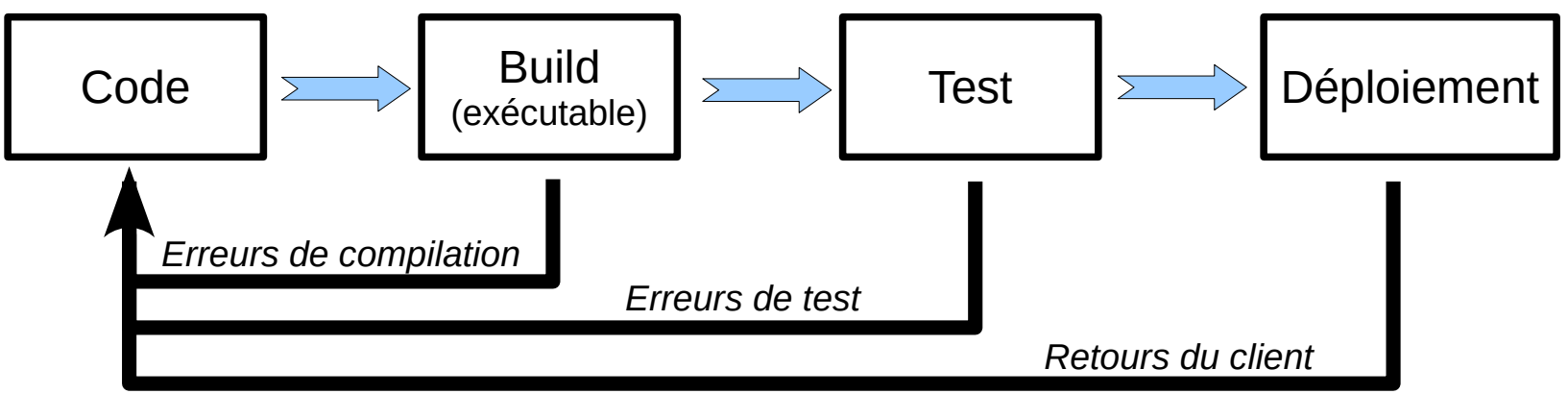
- Intégration **quasi-quotidienne** du travail des développeurs
- Il n'existe qu'une version courante du logiciel partagée par tous les développeurs et toutes les équipes
  - Version opérationnelle et testée
  - Chaque développeur en possède une copie de travail sur laquelle il ajoute ses modifications à la version commune
  - Plus d'équipe / développeur isolée





Cycle d'intégration continue : DevOps

- Quand un développeur pousse son travail sur la version commune cela déclenche :
  - Vérification de la compilation
  - Lancement des tests → **La version commune doit toujours être fonctionnelle**
  - Déploiement automatique
- Nommé **DevOps** (activités de développement et d'opération)



5- Mauvaise pratique
Documentation exhaustive

Documentation exhaustive

- La documentation est un pilier des méthodes prédictives
- Documents
  - Cahier des charges
  - Documentation de conception (eg, organisationnelle, fonctionnelle et non fonctionnelle) sous forme de diagrammes UML et de texte
  - Documentation technique (eg, algorithmes, arborescence de configuration)
  - Manuel d'utilisation

Documentation exhaustive

- Intérêts :
  - Conduire le passage d'une phase à la suivante
  - Revenir en arrière en cas d'erreur
  - Maintenir le logiciel tout au long de sa vie
  - Comprendre comment fonctionne le logiciel
- Quels problèmes posent la documentation ?

Critique de la documentation exhaustive

- **Problèmes** :
  - La documentation est inutile voire néfaste :
    - **Laborieuse à rédiger**
      - Elle a tendance à être bâclée et se révèle confuse et incomplète
    - **Freine les changements**
      - Il faut maintenir la documentation en même temps que les changements
    - **Obsolète**
      - Il y aura toujours un décalage par rapport aux changements
    - **Jamais lue**
      - C'est donc du temps précieux qui est perdu

Critique de la documentation exhaustive

- La documentation n'a de sens que :
  - pour des choses qui n'évoluent plus (eg, rapport d'étudiant)
  - à un instant donné (eg, analyse amont d'un problème ou développement d'une fonctionnalité)
- Ce n'est pas le cas d'un logiciel professionnel qui évolue sans cesse (sinon il meurt)

Bonne pratique

4. Renoncer à la documentation exhaustive

→ Auto-documentation

En particulier : cahier des charges

- Logiciel opérationnel
- Tests d’acceptation
- Le cycle itératif et incrémental permet de réviser constamment le cahier des charges

En particulier : documentation de conception et documentation technique

- Constat :
  - La documentation ment
  - Le code ne ment pas
- Conclusion
  - Faire reposer les documentations de conception et technique sur le code
  - S'il y a besoin de documentation, c'est que le code n'est pas clair… il faut le retravailler : « Don't comment bad code — rewrite it. » Brian W. Kernighan

Quid de la documentation UML

- Les diagrammes UML constituent un moyen de communication, jetable, et pas un moyen de documentation
- Les diagrammes UML ne sont produits que pour :
  - Avancer dans la compréhension du domaine et des besoins à un instant donné
  - Échanger entre développeurs
- Et si on a besoin de documentation, en particulier lors de la reprise de code ?
  - Rétro-ingénierie du code (reverse engineering)

En particulier : manuel utilisateur

- Manuel utilisateur → logiciel intuitif (user friendly)
  - User Experience (**UX**) : grâce aux MVP (intégrer l'utilisateur dans l'équipe de développement)
    - Exemple : En cas d'erreur, ne pas se contenter d'afficher « erreur » → donner une solution possible
    - Motto UX : « *Il ne faut pas prendre les gens (utilisateurs) pour des cons, mais il ne faut jamais oublier qu'ils le sont* » Les inconnus

Auto-Documentation

- Méthodes prédictives

		Méthodes agiles
• Cahier des charges	→	Logiciel fonctionnel + Tests
• Diagramme de Gantt	→	Itérations
• Documentation de conception	→	Architecture du code
• Documentation technique	→	Code propre
• Manuel utilisateur	→	Expérience utilisateur (UX)

Plan du chapitre

1
De la prédiction vers l'agilité

2
Exemples de méthodes Agiles

Exemples de méthodes Agiles

- Scrum
- Kanban
- eXtreme programming (XP)

- Dans un projet : mixer plusieurs méthodes
  - Par exemple :
    - Cadre général : Scrum
    - Gestion des tâches : tableau Kaban
    - Codage : eXtreme Programming

Que retenir de ce chapitre ?

- Les méthodes agiles proposent de nouvelles façons d'aborder le développement logiciel
  - Le développement suit un cycle **itératif** et **incrémental**
    - Chaque itération correspond au développement de plusieurs petites fonctionnalités qui sont intégrées aussitôt au logiciel et testées
    - Chaque itération doit se terminer par la livraison d'une version opérationnelle et testée du logiciel en cours
    - Chaque incrément doit ajouter une valeur pour le client au prototype : MVP
    - Chaque itération est une fin en soi : un mini-projet
  - **Intégration continue** du travail des développeurs
  - Maximiser l'**auto-documentation**
- **L'agilité est plus une méthode de gestion de produit que de gestion de projet**