ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

04
Chapitre

Une méthode : 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)

Rappel

- Le génie logiciel aujourd'hui :
  - Un paradigme de conception : Conception orientée objet
  - Une formalisme de modélisation : UML
  - Une méthode de gestion de projet : Agilité

Rappel

- Méthodes prédictives
  - Très efficaces pour la majorité des projets d'ingénierie
  - Mais peu adaptées au développement de logiciels purs

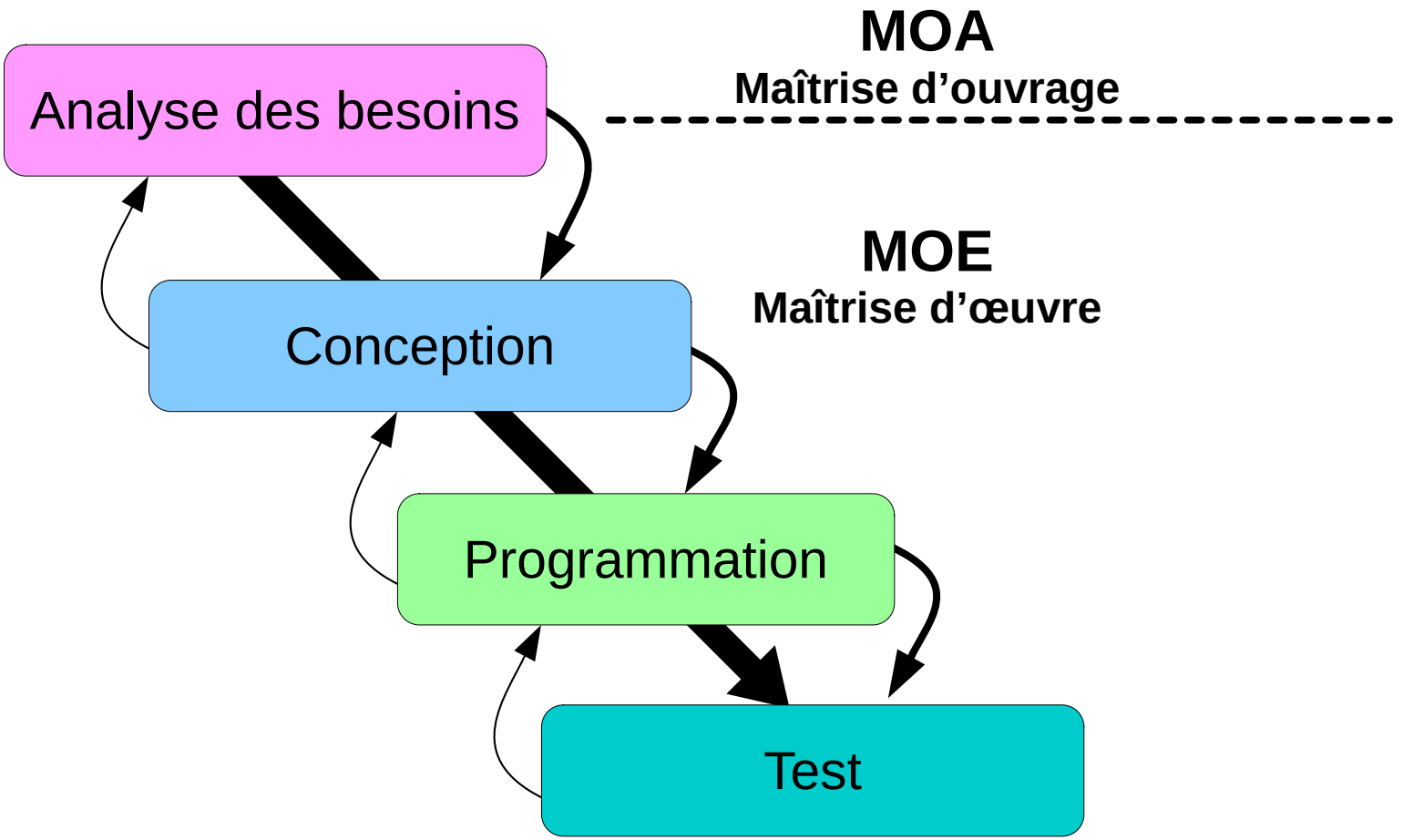
Plan du chapitre

1

Mauvaises pratiques du développement de logiciels

1- Suivre un plan à long terme : le modèle en cascade

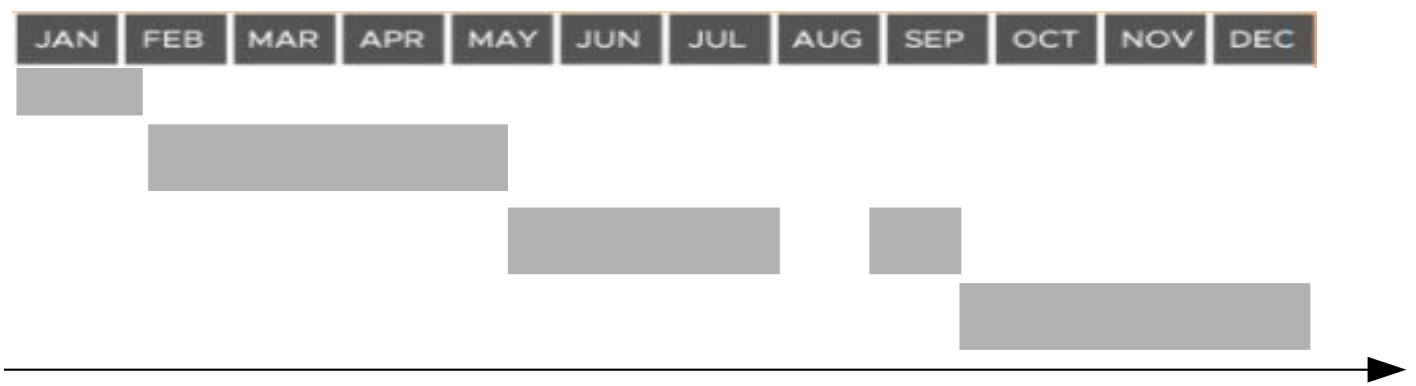
Le modèle (cycle) en cascade



Critique du modèle en cascade

- 1/ Planification complète du projet
  - Quels problèmes posent cette planification ?

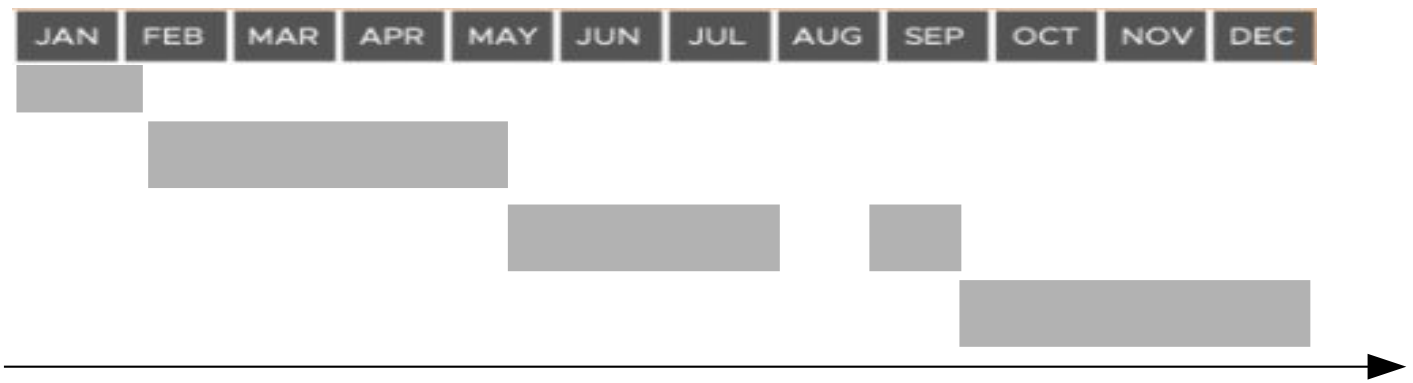
Analyse des besoins
Conception
Codage
Tests



Critique du modèle en cascade

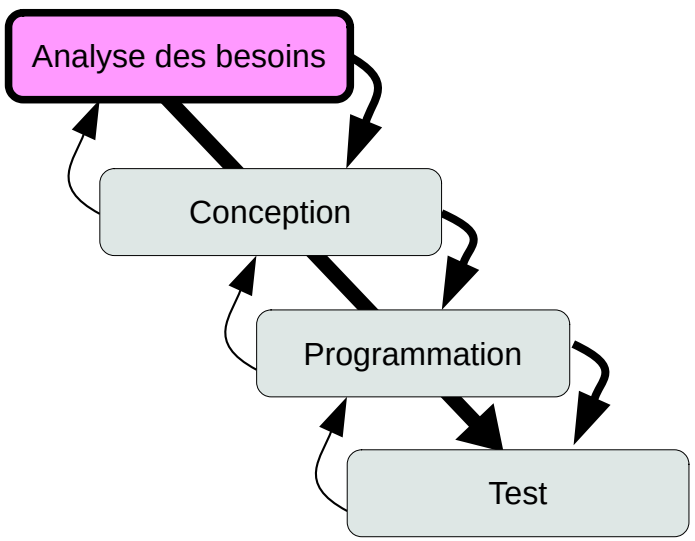
- 1/ Planification complète du projet
  - **Problèmes**
    1) Impossible de **prévoir**
    2) Aucune adaptation aux imprévus ou changements pourtant inévitables

Analyse des besoins
Conception
Codage
Tests



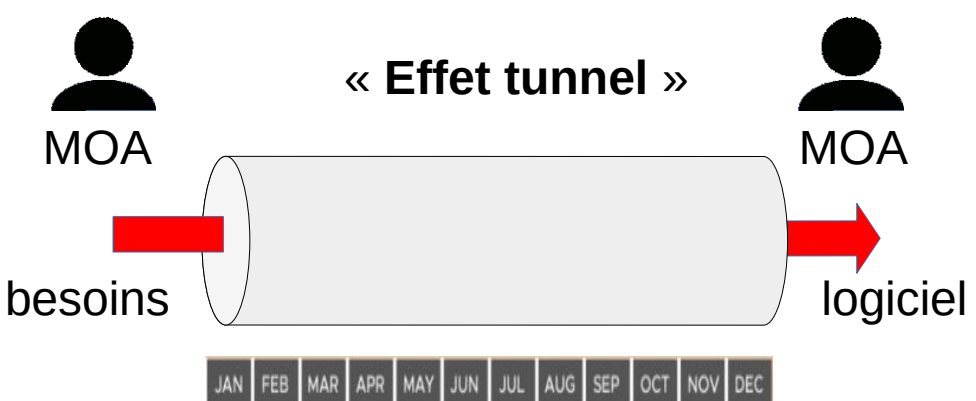
Critique du modèle en cascade

- 2/ Analyse des besoins au début
  - Quels problèmes voyez-vous avec cette organisation ?



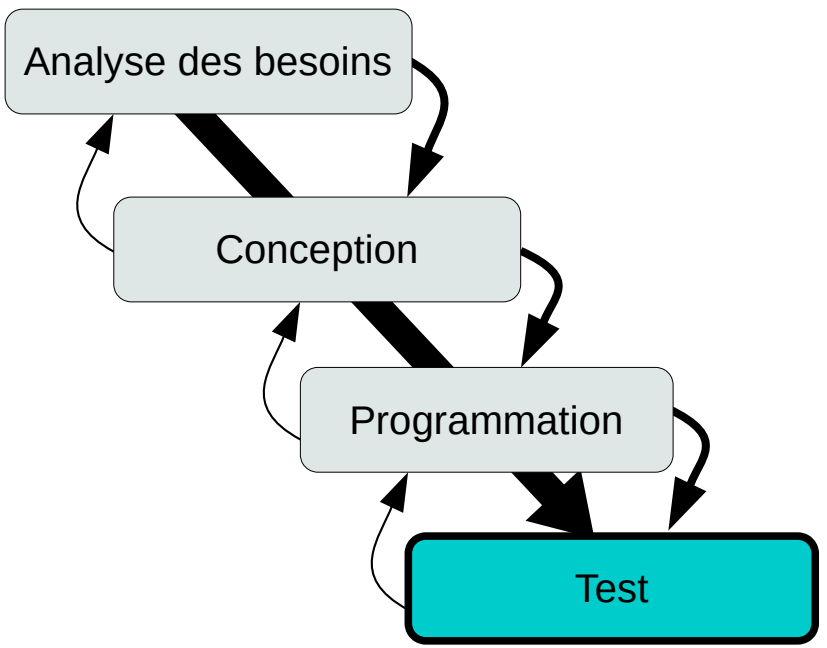
Critique du modèle en cascade

- 2/ Analyse des besoins au début
  - **Problèmes**
    1) **Non satisfaction des clients** : les besoins ont été mal compris ou ont changés entre-temps
    2) Effet tunnel
    3) **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*)



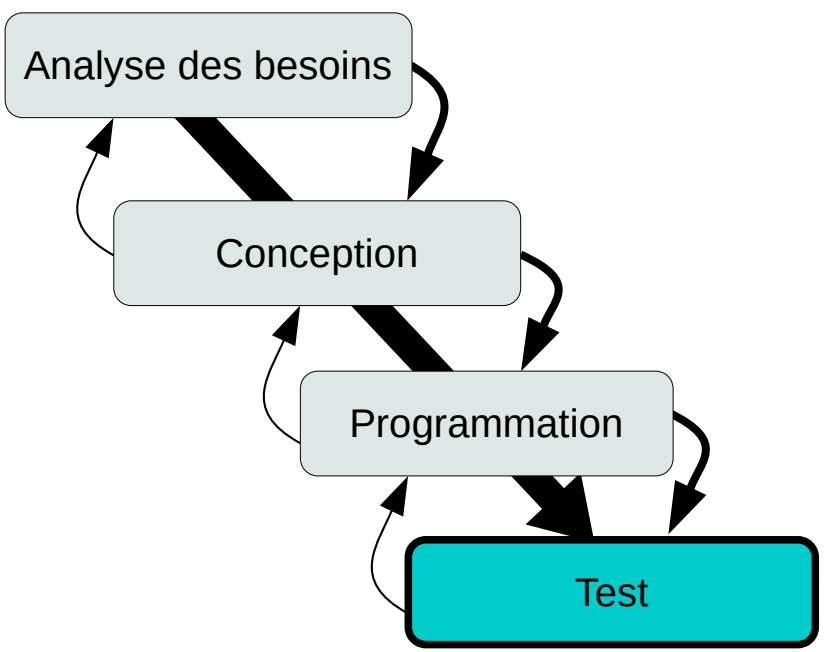
Critique du modèle en cascade

- 3/ Test en fin de projet
  - Quelles conséquences néfastes voyez-vous sur les tests ?



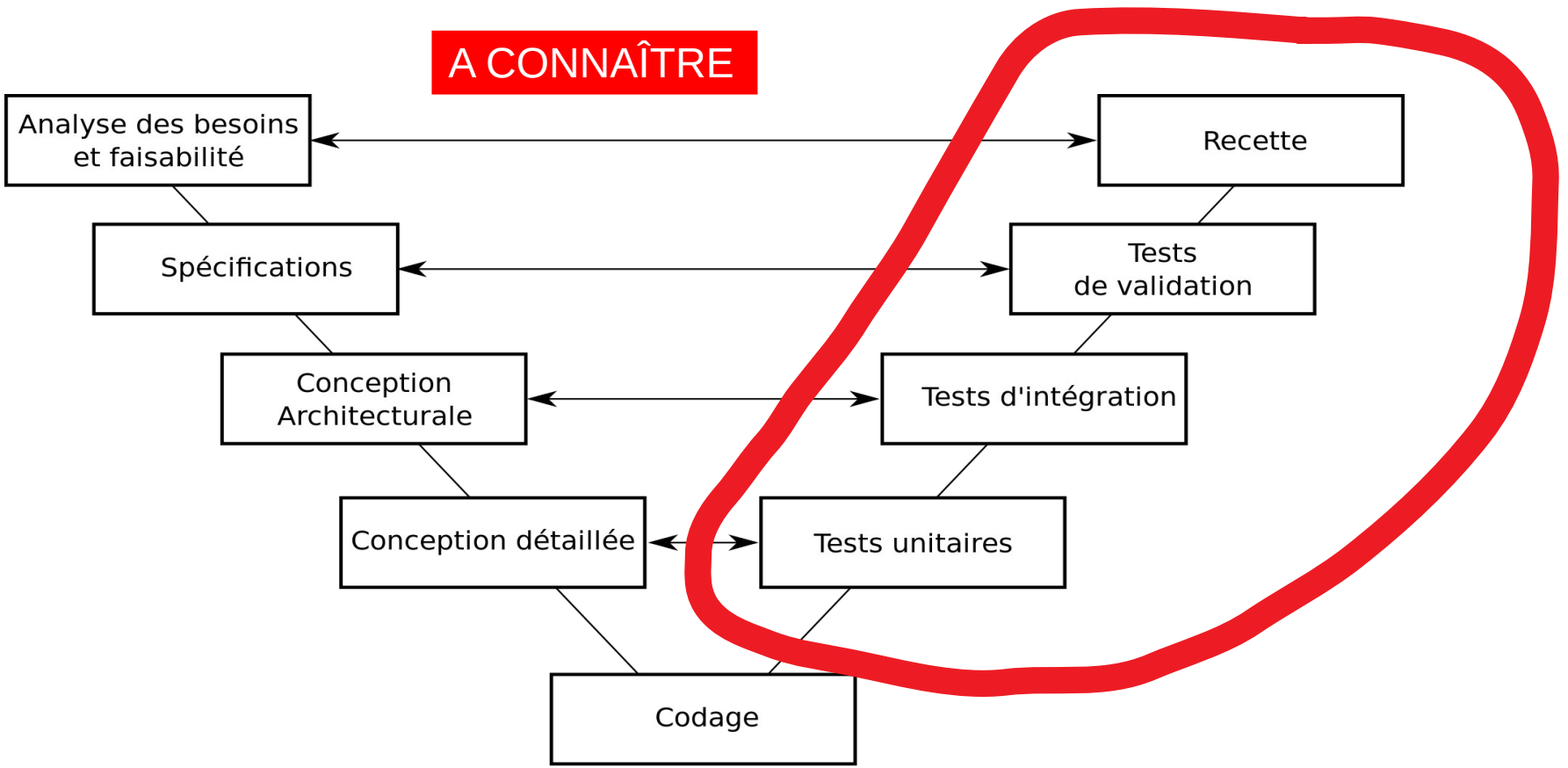
Critique du modèle en cascade

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



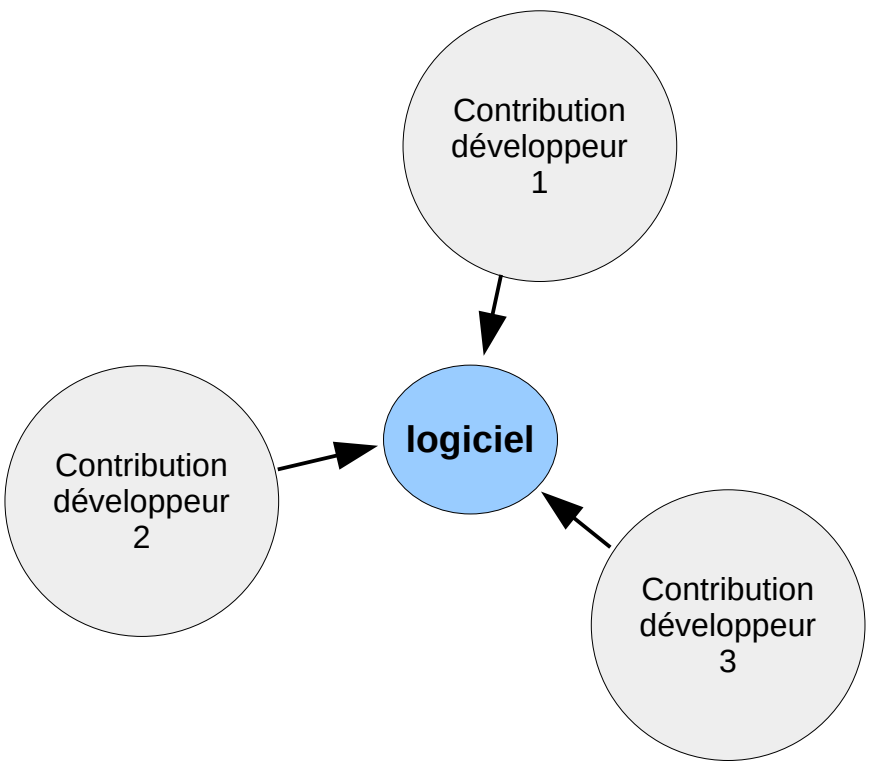
Une amélioration : le cycle en V

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



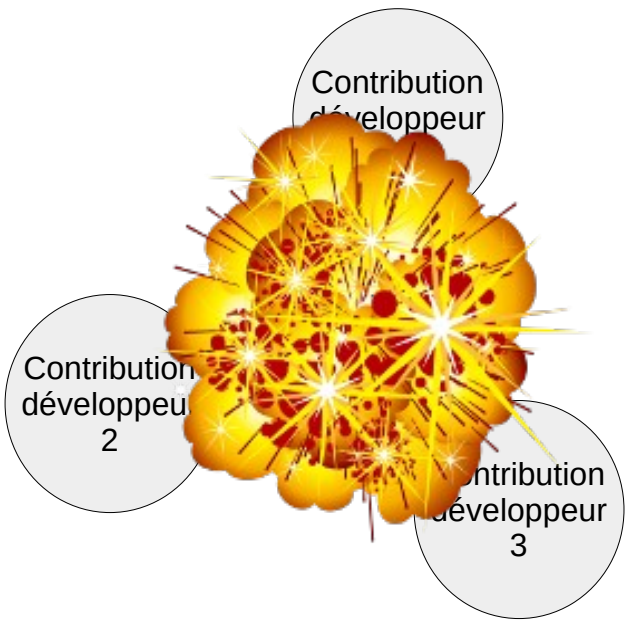
Critique du modèle en cascade

- 4/ Parallélisation de la phase de programmation : le logiciel final résulte de l'intégration des codes de chaque partie
  - Quel problème pose cette intégration finale ?



Critique du modèle en cascade

- 4/ La planification permet le travail en parallèle
  - **Problème**
    - **Intégration avec « Effet big bang »**
      1) Conflits
      2) Incompatibilité



Exemple d’incompatibilité non détectée

- 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 $.

1- Suivre un plan à long terme :
le modèle en cascade

2- Documentation exhaustive

Documentation exhaustive

- La documentation est un pilier des méthodes prédictives
- Documents
  - Cahier des charges
  - Documentation de conception (eg, organisationnelle, fonctionnelle, non fonctionnelle) sous forme de diagrammes UML et de texte
  - Documentation technique (eg, algorithmes, arborescence de configuration)
  - Manuel d'utilisation
- La documentation est indispensable pour :
  - Passer à la phase suivante
  - Revenir en arrière en cas d'erreur
  - Maintenir le logiciel
  - Comprendre comment fonctionne le logiciel

Documentation exhaustive

- Quels problèmes voyez-vous avec 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 voire nuisible**
      - Il y aura toujours un décalage par rapport aux changements
    - **Jamais lue**
      - C'est donc du temps précieux perdu

Critique de la documentation exhaustive

- La documentation n'a de sens que :
  - pour des choses qui n'évoluent plus (p. ex. rapport d'étudiant)
  - pour un instant donné (p. ex. analyse amont d'un problème ou développement d'une fonctionnalité)
- Ce n'est pas le cas d'un logiciel professionnel

Plan du chapitre

1

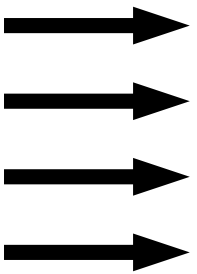
Mauvaises pratiques du développement de logiciels

2

Bonnes pratiques du développement

Pratique actuelle : agilité

- **Méthode prédictive**
  - Le suivi d'un plan
  - Documentation exhaustive
  - Les processus et les outils
  - Négociation contractuelle



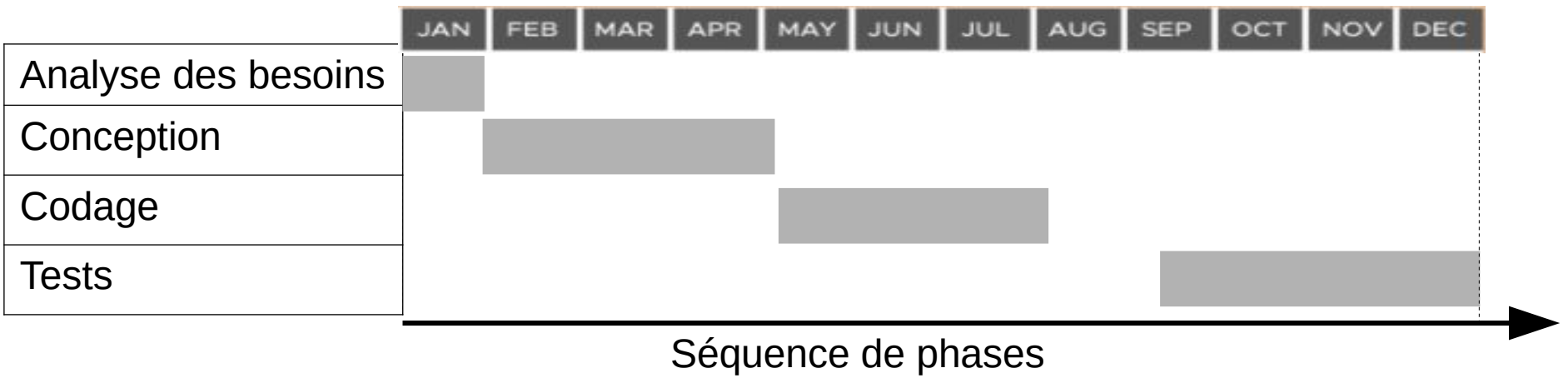
- **Méthode agile**
  - L'adaptation au changement
  - Un logiciel qui fonctionne
  - Les individus et leurs interactions
  - La collaboration avec les clients

- **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. »

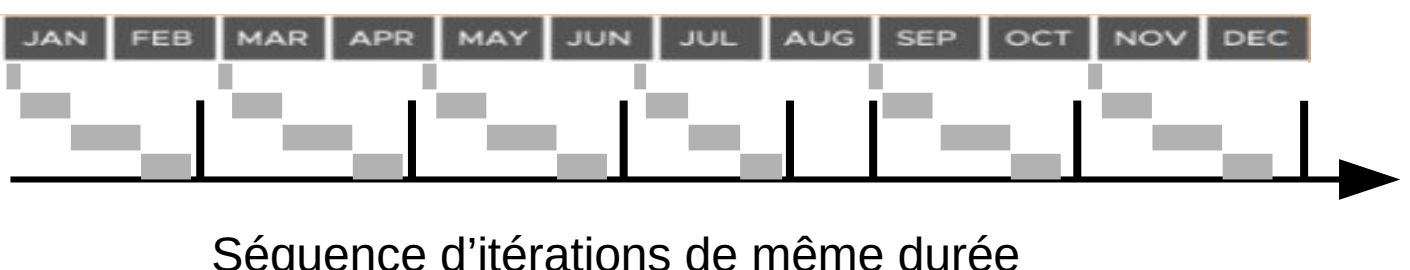
1. Renoncer à la planification à long terme
→ Cycle de développement itératif

Cycle de développement itératif

- Modèle en cascade

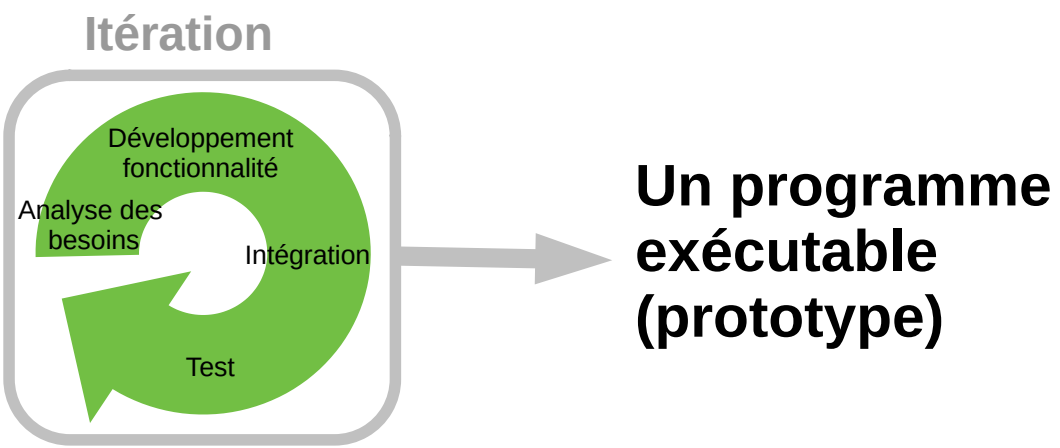


- Modèle itératif



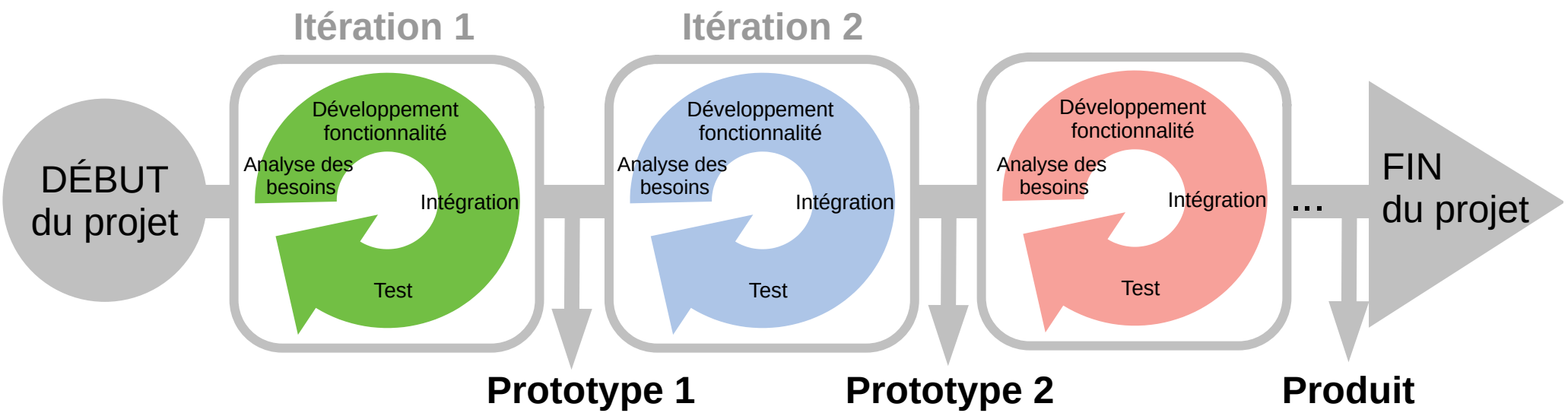
Itération

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



Développement itératif

- Projet : une suite de mini-projet



Avantages du développement 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 testé
- Chaque pas est validé avec le client
- La revue d'itération permet de réviser les besoins
- Nul besoin d'une méthode structurée pour une itération de 2 à 4 semaines
  - Même une méthode à l'arrache peut fonctionner (https://www.la-rache.com/)
- Adaptation aux changements

1. Renoncer à la planification à long terme
   → Cycle de développement itératif
2. **Éviter l'« effet tunnel »**
   → Cycle de développement incrémental

Développement incrémental

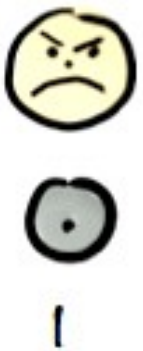
- Le logiciel est construit par **incrémentation** en profitant de la malléabilité du logiciel
  - Impliquer le client dans la définition des besoins
    - Faire un prototype qui soit évaluable par le client
- Incrément :
  - **MVP** (***Minimum Viable Product***) : Un prototype avec juste assez de fonctionnalités 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 çà !

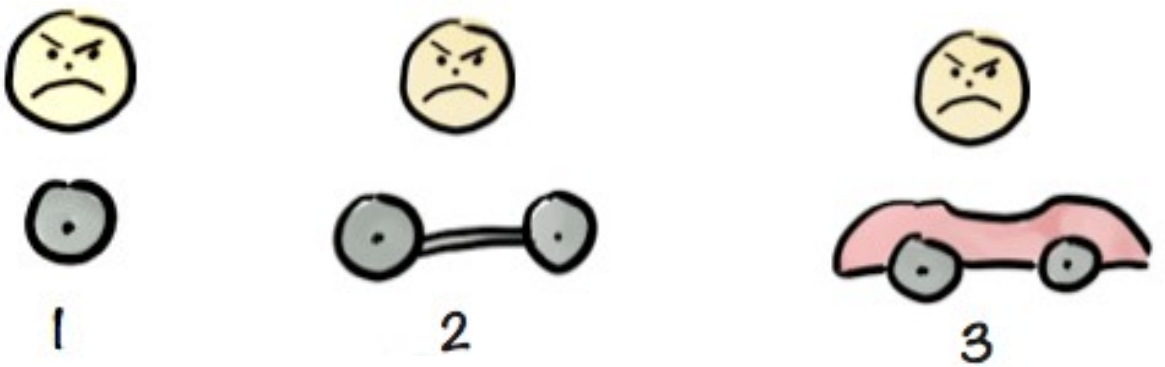
- Itération 1



  - « 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 ? »

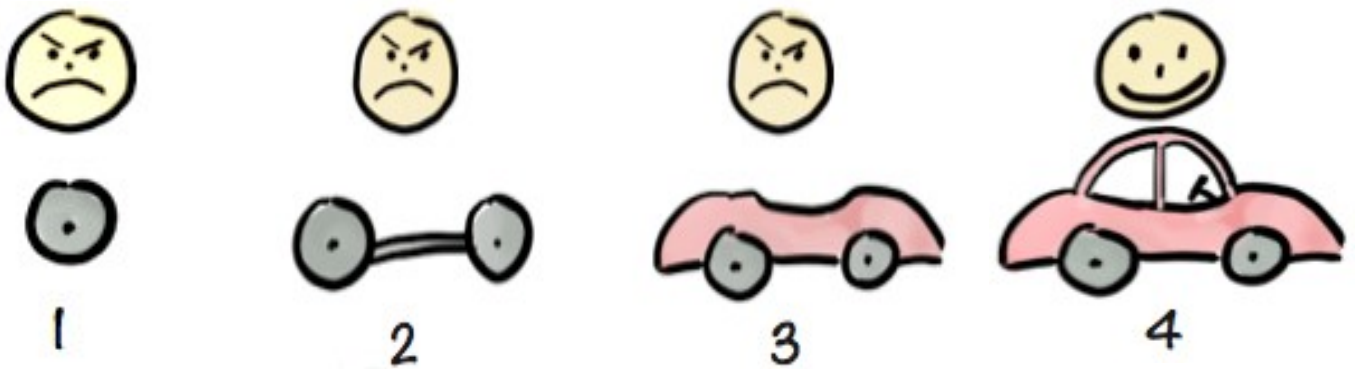
Incrémental : Pas comme çà !

- Itérations 2 et 3



Incrémental : Pas comme çà !

- Itération 4 finale



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

Incrémental : Comme çà !

- 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 à un point B. »

MVP

- 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 çà !

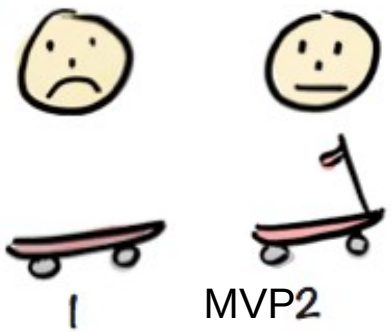
- Et pourquoi pas un simplement un ticket de bus

MVP

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

Incrémental : Comme çà !

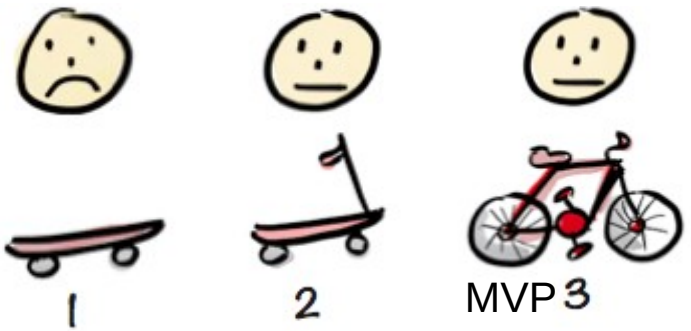
- Itération 2



- 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 çà !

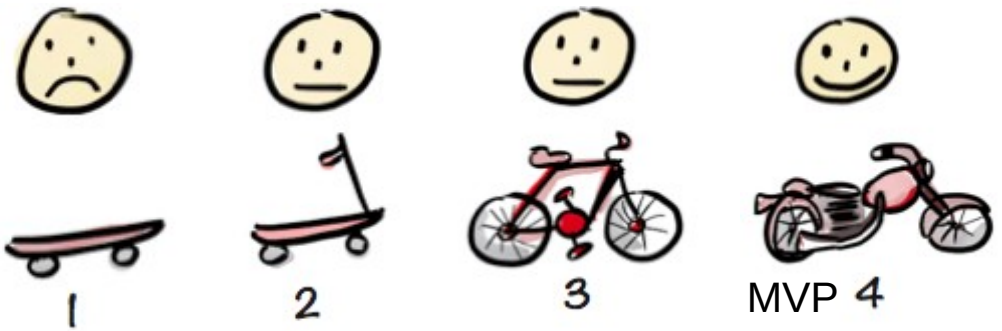
- Itération 3



- 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 çà !

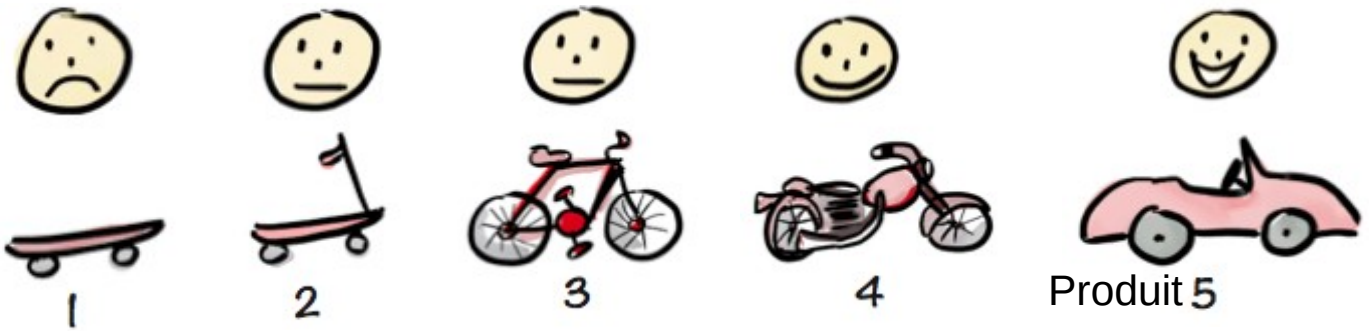
- Itération 4



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

Comme çà !

- 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

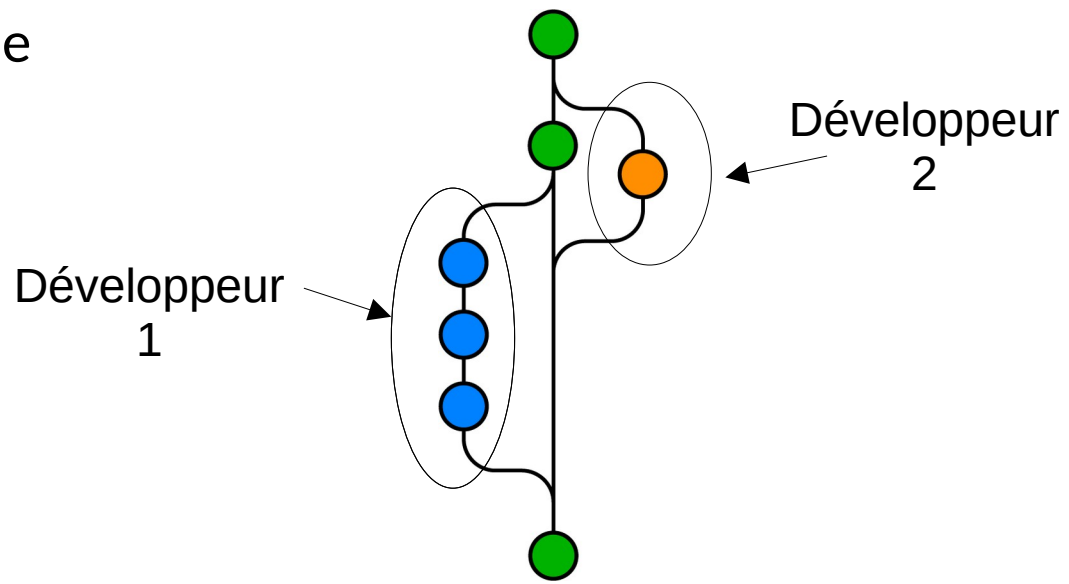
Avantages du développement incrémental

- 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

1. Renoncer à la planification à long terme
   → Cycle de développement itératif
2. Éviter l'« effet tunnel »
   → Cycle de développement incrémental
3. Éviter l'intégration « big bang »
   → Intégration continue

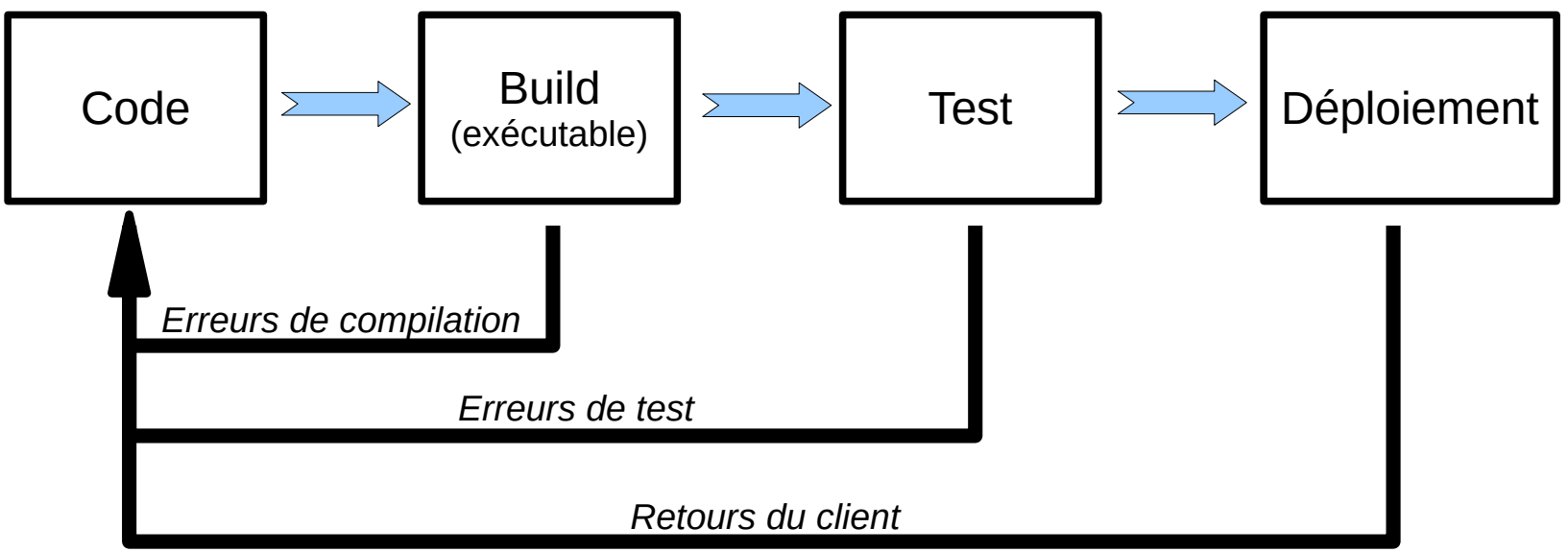
Intégration continue

- Intégration **quasi-quotidienne** du travail des développeurs
- Il n'existe qu'une version courante du logiciel partagée par tous les développeurs
  - 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 isolée



Cycle d'intégration continue

- Quand un développeur pousse son travail sur la version commune cela déclenche :
  - Vérification de la compilation
  - Lancement des tests
  - Potentiellement, déploiement chez le client
- Nommé **DevOps** (activités de développement et d'opération)



1. Renoncer à la planification à long terme
   → Cycle de développement itératif
2. Éviter l'« effet tunnel »
   → Cycle de développement incrémental
3. Éviter l'intégration « big bang »
   → Intégration continue
4. **Limiter la documentation**
   → Auto-documentation

Auto-documentation

- Constat :
  - La documentation ment
  - Le code ne ment pas
- Conclusion
  - Faire reposer la documentation 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

En particulier cahier des charges

- Limité aux Use Cases (Use Stories)
  - Peu formel et pas contractuel
  - Le cycle itératif et incrémental permet de réviser constamment le cahier des charges

En particulier documentation UML

- Les diagrammes UML constituent un moyen de communication, jetable, et pas de la 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, par exemple 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 aussi une solution possible.
    - « *Il ne fait jamais prendre les gens (utilisateurs) pour des cons, mais il ne faut pas oublier qu'ils le sont* » Les inconnus.

Mise en pratique de l'agilité en TP

- Démarche qui s'inspire du développement agile :
  - Approche itérative
    1) Coder une fonction seulement
    2) Faire une fonction dans le main() qui permet de la tester
       - Compiler
       - Exécuter
       - Tester
    3) Recommencer avec une autre fonction
  - Approche incrémentale
    - Version simplifiée du problème avec peu ou aucune contrainte
    - Ajouter des contraintes

Plan du chapitre

1

Mauvaises pratiques du développement de logiciels

2

Bonnes pratiques du développement

3

Des méthodes Agiles

Exemples de méthodes Agiles

- Scrum
- Kanban
- eXtreme programming (XP)

- Dans un projet : mixer plusieurs méthodes
  - P. ex.
    - Cadre : Scrum
    - Visualisation 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**