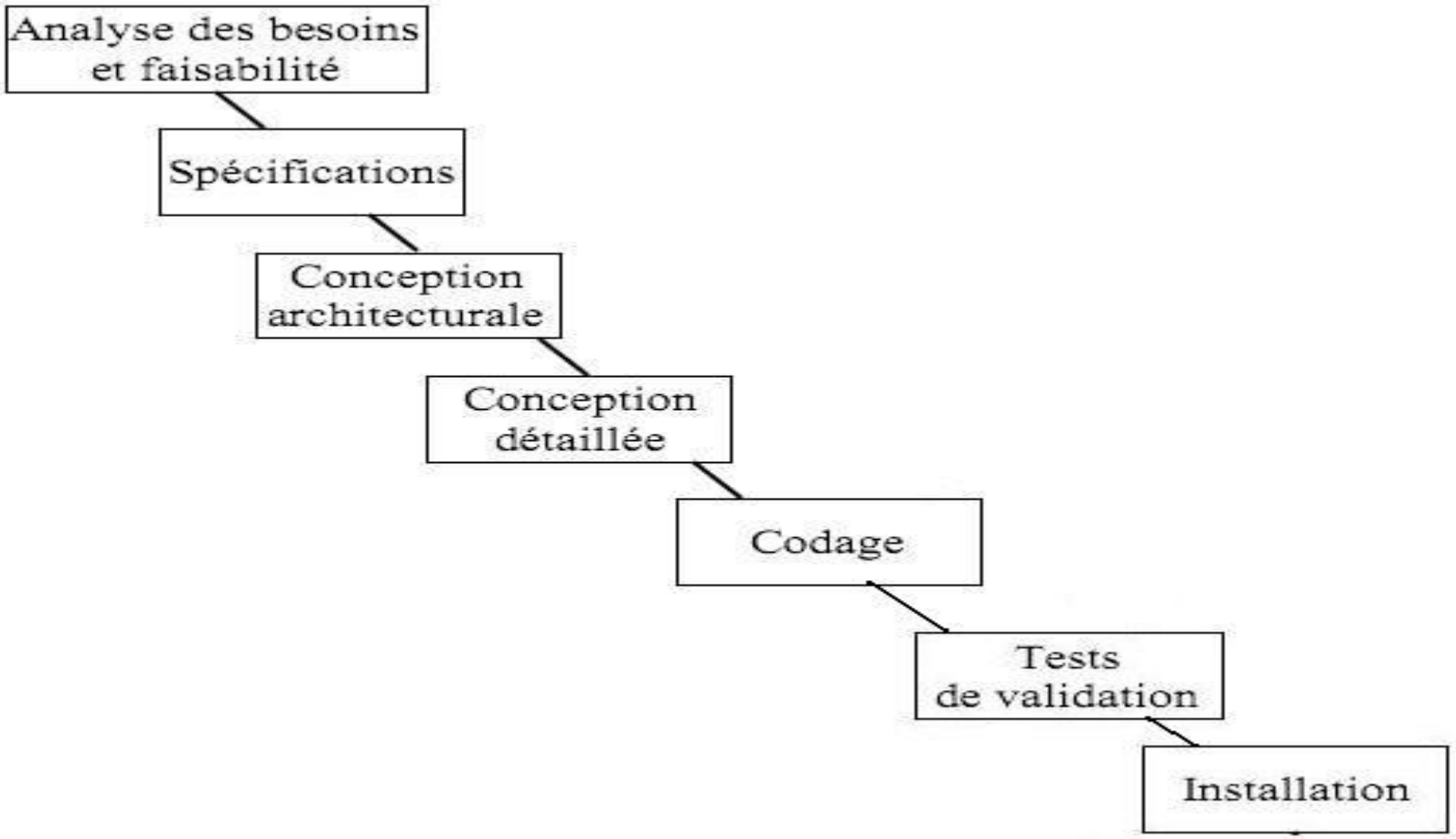
Ingénierie du Logiciel
Master 1 Informatique – 4I502

Cours 8 :
Les méthodes Agiles

Yann Thierry-Mieg
Yann.Thierry-Mieg@lip6.fr

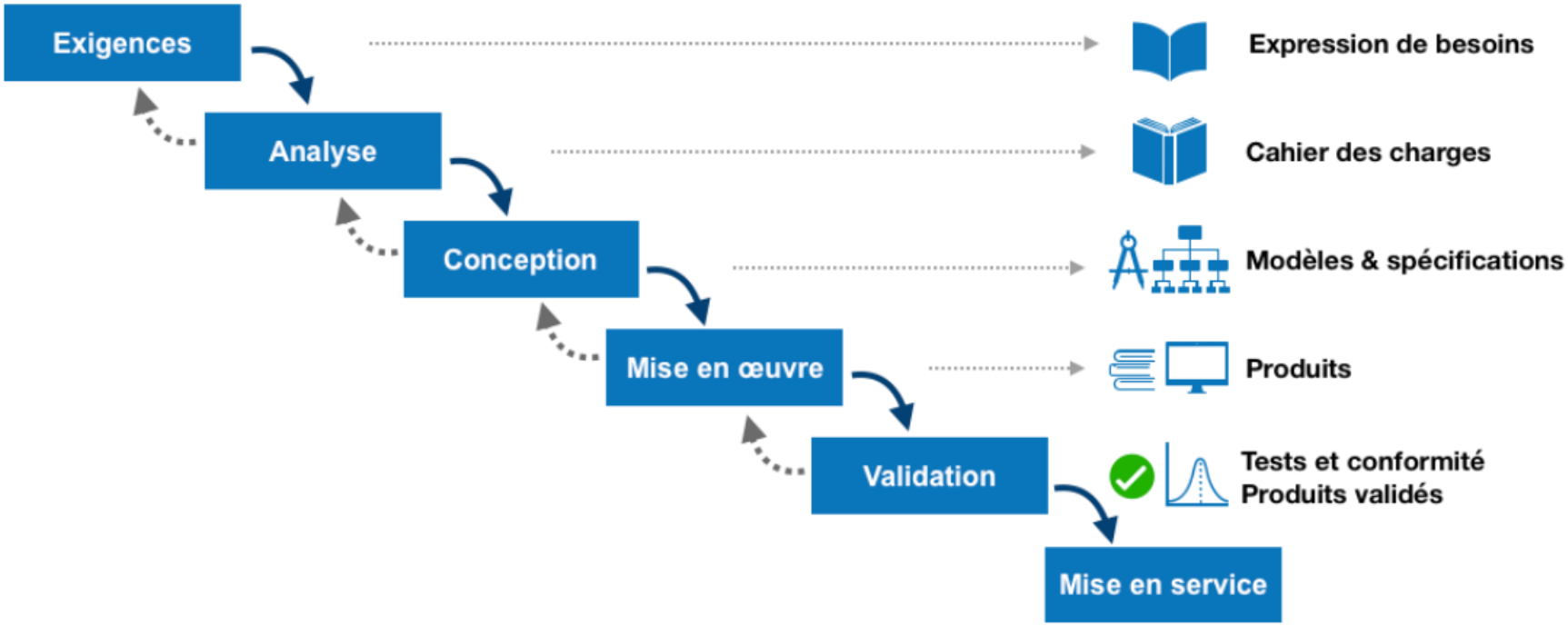
L’Evolution des Méthodes de Développement

‘70 : La cascade



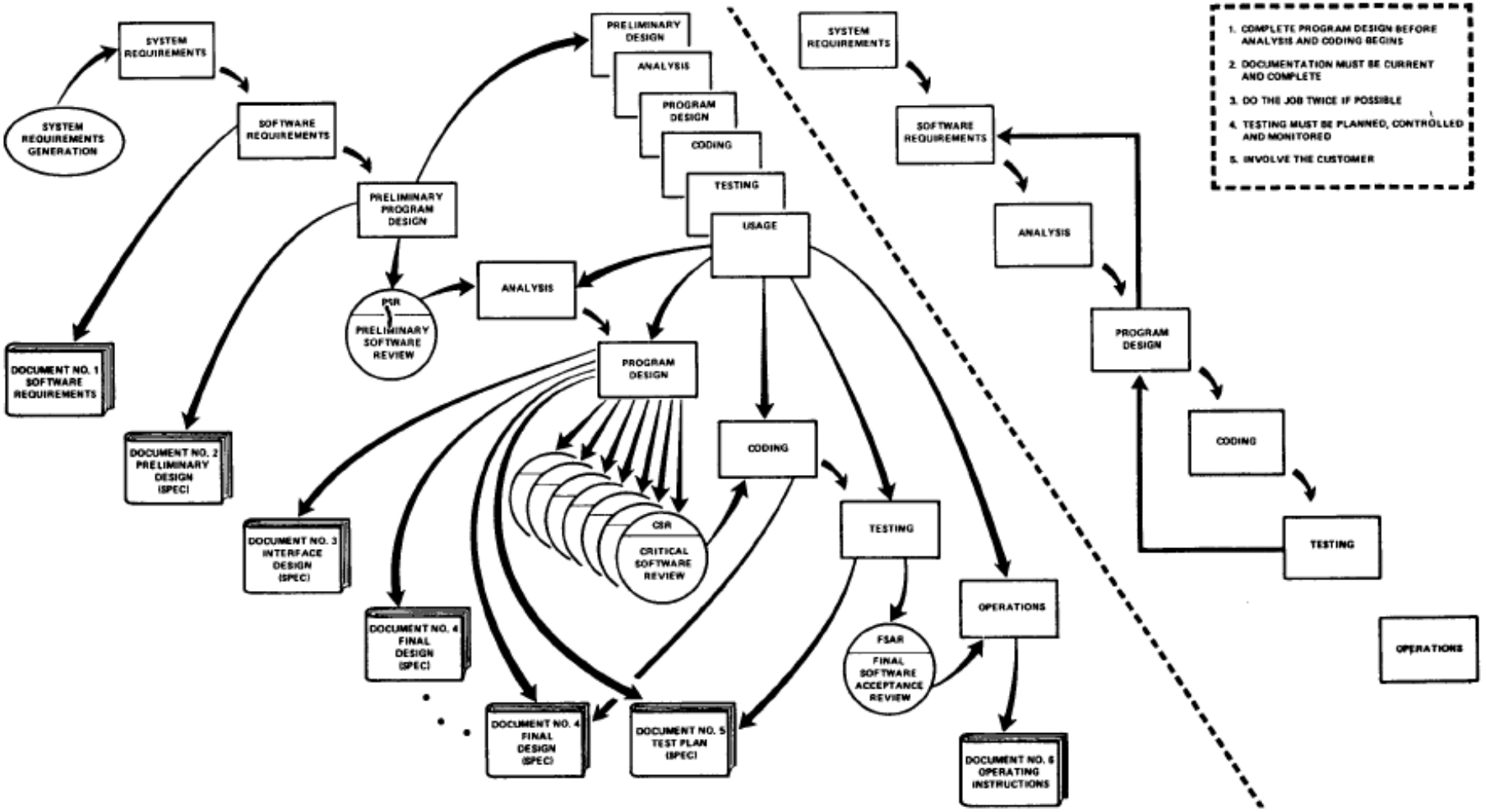
- Etapes séquentielles bien délimitées
- On enchaine l’étape suivante quand la précédente est entièrement terminée

Cascade



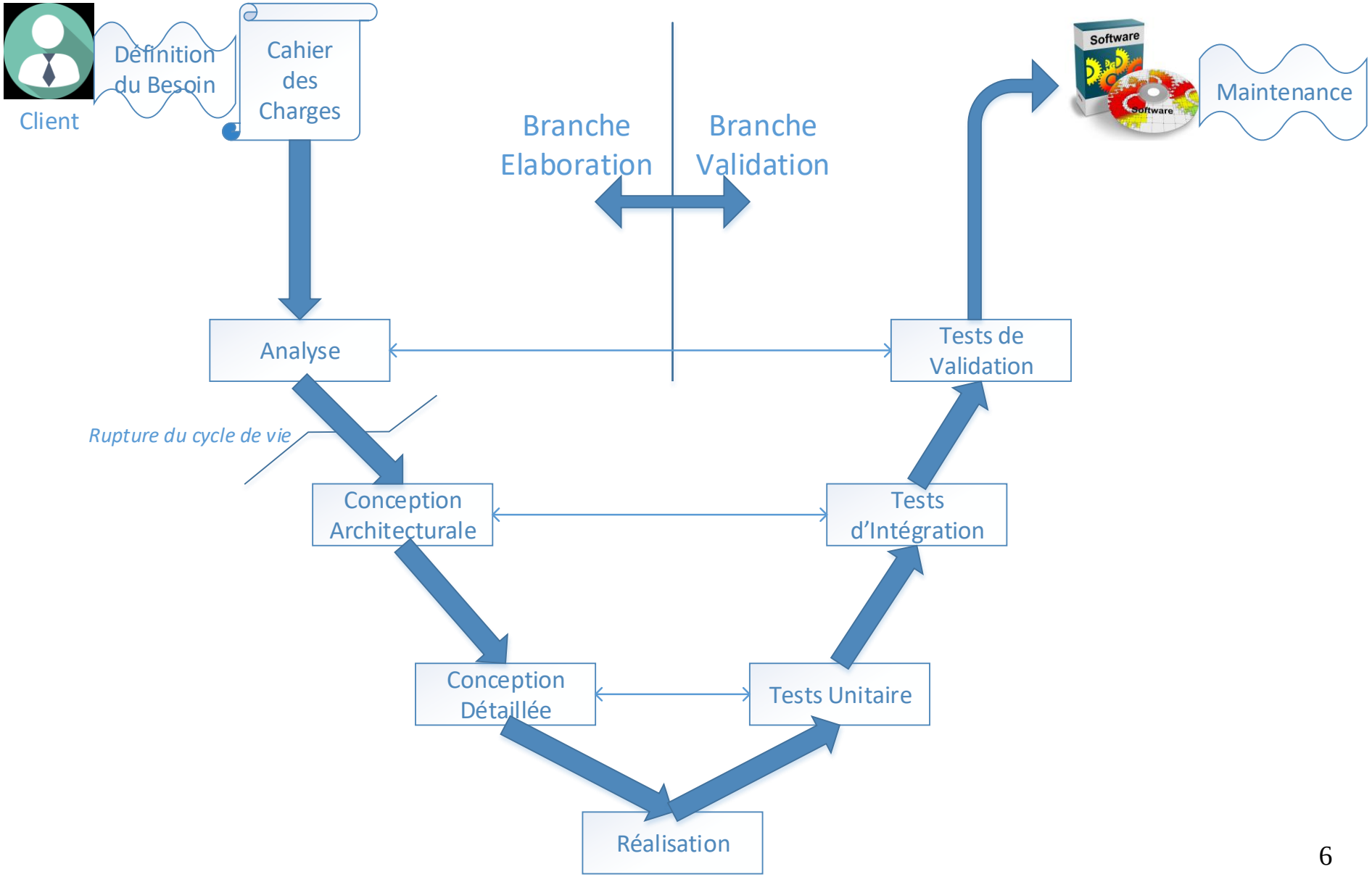
- ++étapes bien identifiées et délimitées
- -- aspect séquentiel forcé
- -- test très tardif
- -- effet « tunnel » assez fort, le client ne voit pas de délivrable intermédiaire, on peut se tromper de cible.

Waterfall (Royce 1970)



- Une méthode complète et bien définie mais qui souffre de défauts: feedback client pauvre, difficulté de corriger les erreurs, validation tardive

Le cycle en V


Client
Définition du Besoin
Cahier des Charges
Branche Elaboration
Branche Validation
Software
Maintenance
Analyse
Tests de Validation
Rupture du cycle de vie
Conception Architecturale
Tests d'Intégration
Conception Détaillée
Tests Unitaire
Réalisation

Cycle en V années 80/90

- Longuement discuté dans l'UE : l'instance SU du V
  - Choix des artefacts à produire à chaque étape,
  - utilisation particulière d'UML
- Qualités Indéniables :
  - ++ Rigoureux, force une certaine qualité logicielle
  - ++ Etapes bien délimitées, séparation des préoccupations
  - ++ Validation mise en place précocement, séparation des niveaux de test
  - ++ Adapté aux projets de grande taille, systèmes complexes,
  - ++ Support du développement orienté composant

Le Cycle en V

- Qualités discutables
  - + Configurable de diverses manières, documentations diverses, niveaux d'abstraction divers, rôles et étapes bien définies
  - - Pas de définition concrète précise, trop de documentation, pas assez concret et orienté prototype, rôles trop figés qui empêchent une bonne transmission verticale des informations
- Défauts majeurs
  - --Effet « tunnel » hérité de la cascade
  - --Séquentialité des étages (feedback du développement),
  - --Pas le droit à l'erreur, la spécification en amont est difficile
  - --Prise en compte tardive de la plateforme technique (cf. le Y de 2TUP)
  - --Mauvaise réactivité au changements
- Reste un modèle de référence important

Les cycles itératifs

Amélioration continue

- Comment :
  - Avoir un meilleur dialogue client
  - Avoir le droit à l'erreur, mais s'améliorer au fil du temps
    - Avoir des incréments stables

Roue de Deming

Plan

- Prévoir un incrément, Spécifier, Concevoir

Do

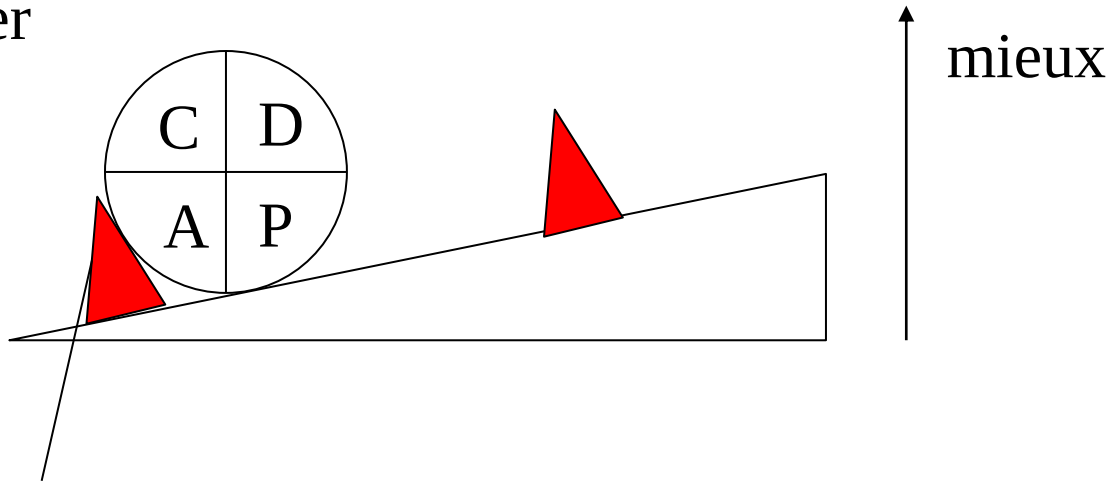
- Réaliser

Check

- Contrôler, mesurer

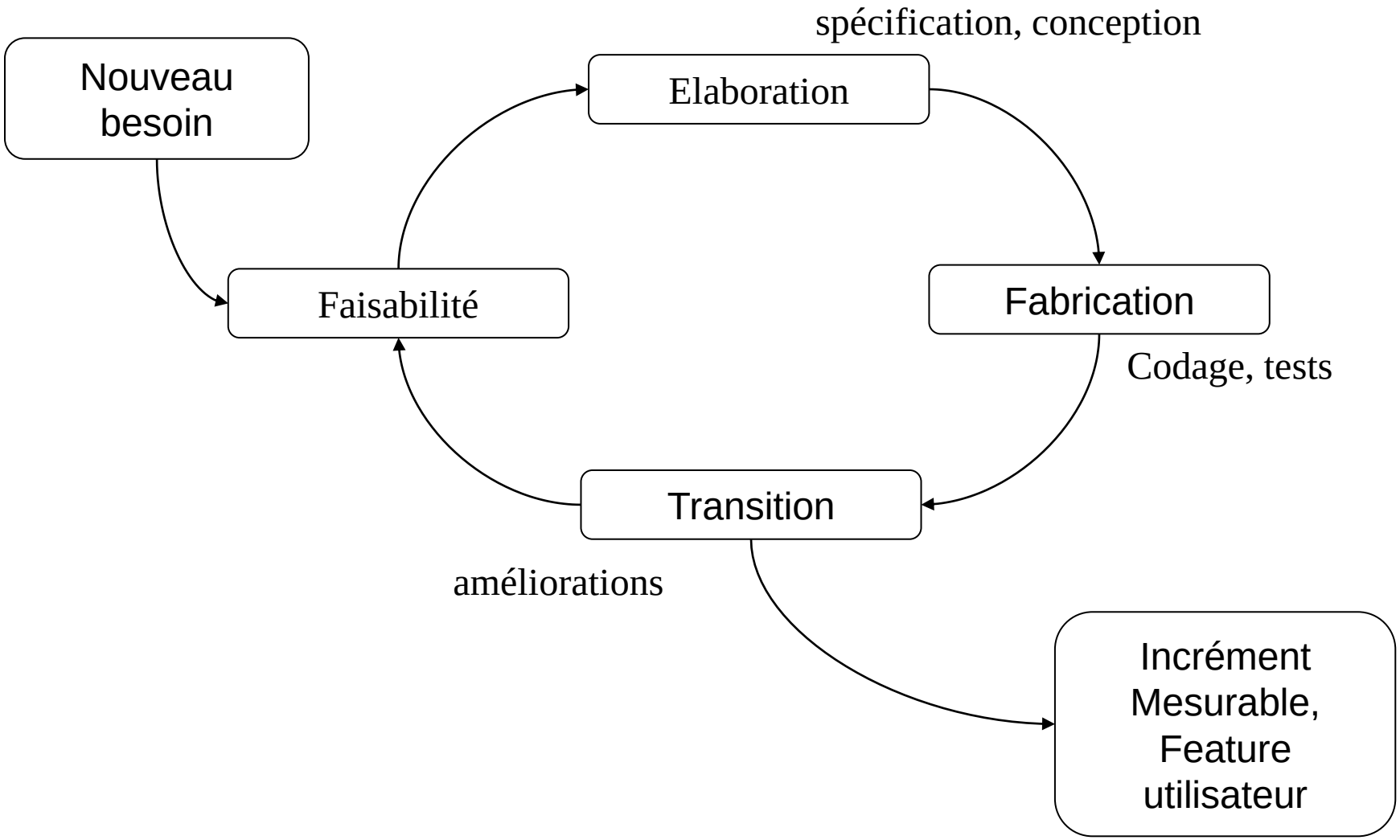
Act

- Améliorer,
- Apprendre
- Corriger le tir

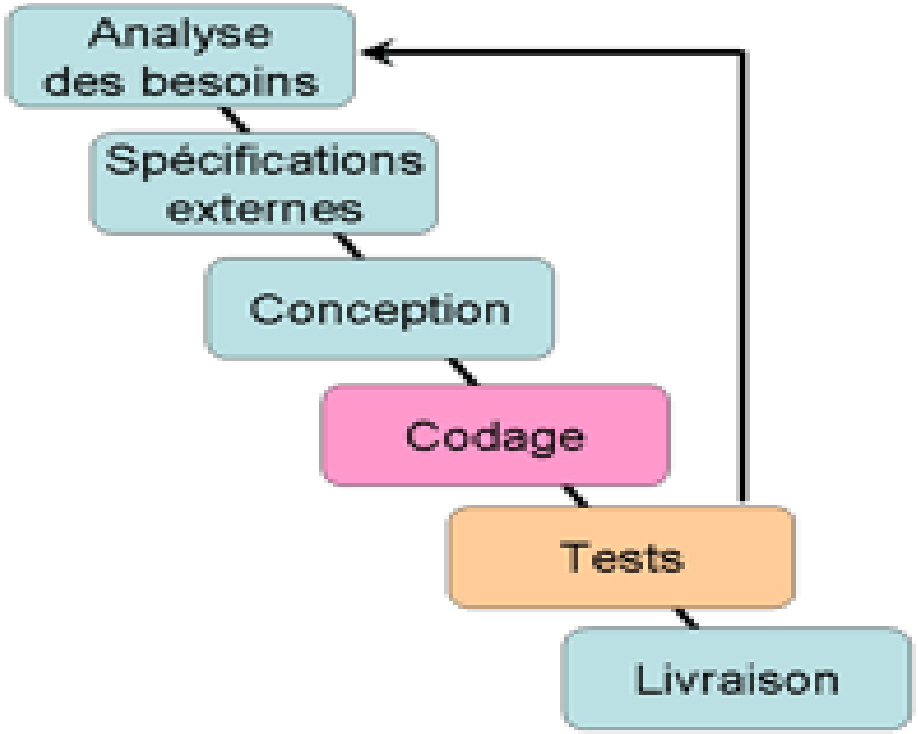


Tests de non régression

Cycle itératif

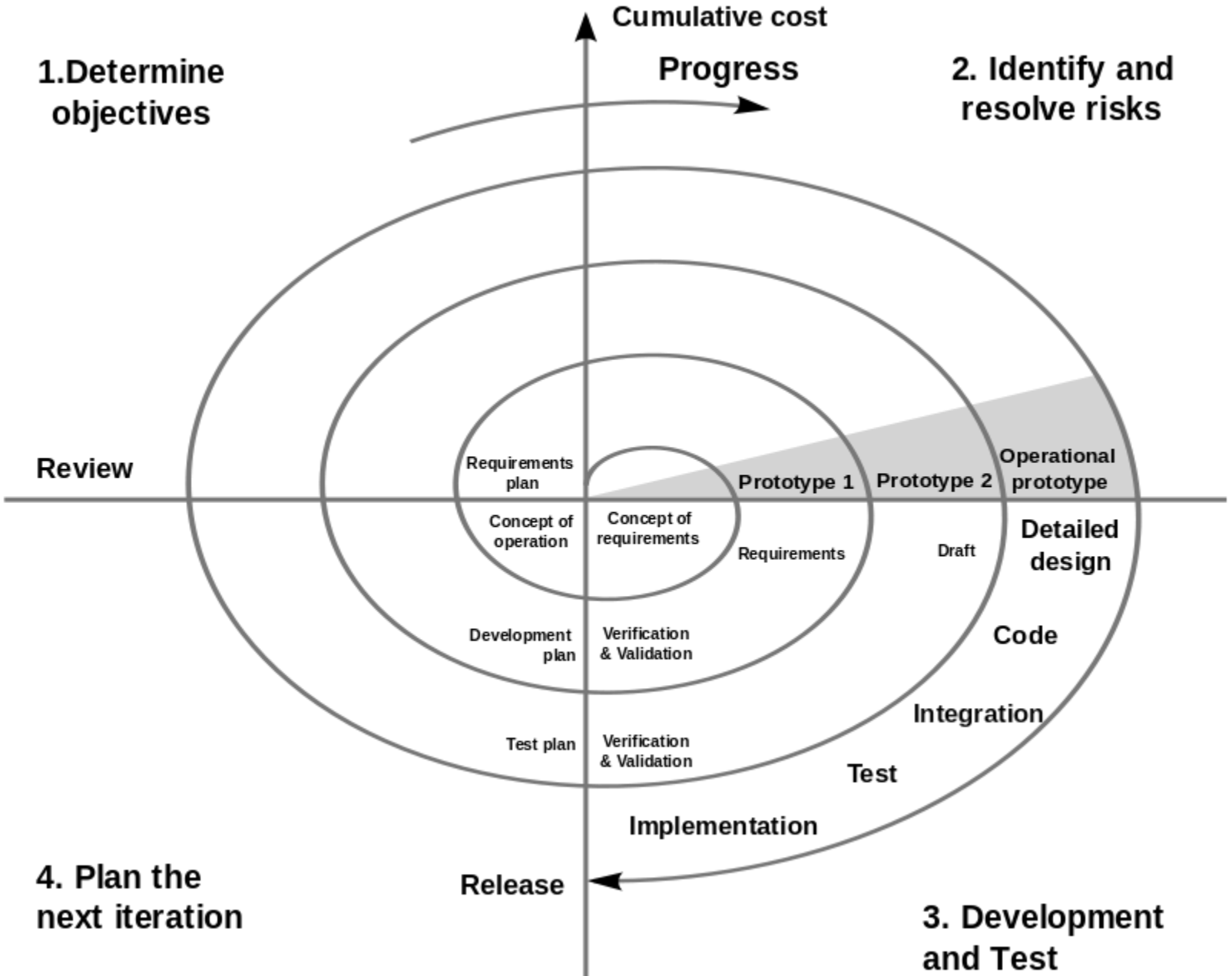


Le cycle reboucle

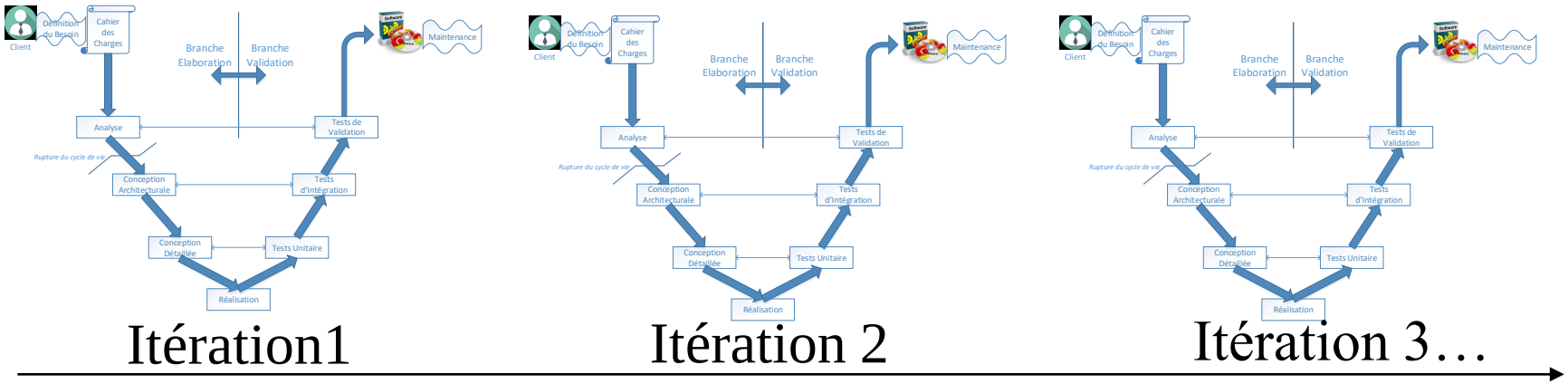


- ++ Droit à l'erreur
- ++ Communication client
- ++ Gestion du changement
- +- Approximations successives, manque de vision long terme

Spirale (Boehm)

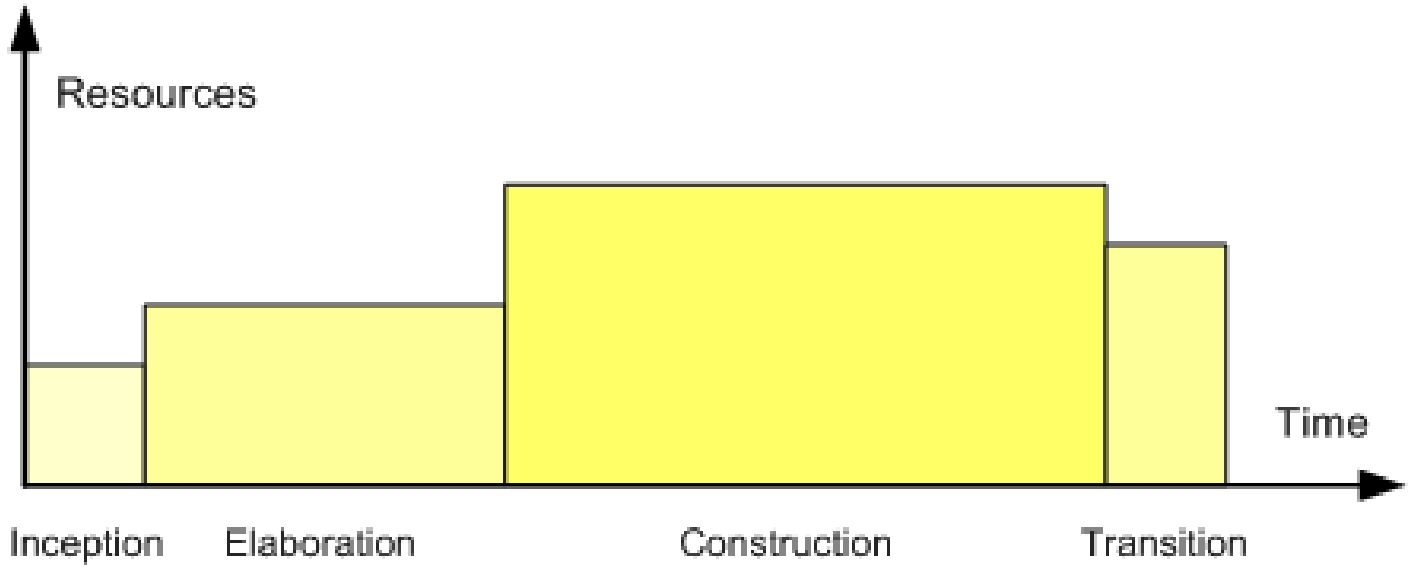


Le V répété



- Chaque itération est un cycle complet
  - Mais un sous ensemble de fonctionnalités utilisateur est traité
- Découpe selon les fonctionnalités :
  - Use cases, scenarios de fiches détaillées
  - Développement transverse des composants
- Présentations client régulières, définition des prochains T.V. en début d'itération
  - La cible est mieux maitrisée, limite le « tunnel »
- Droit à l'erreur
  - Corriger l'analyse à posteriori

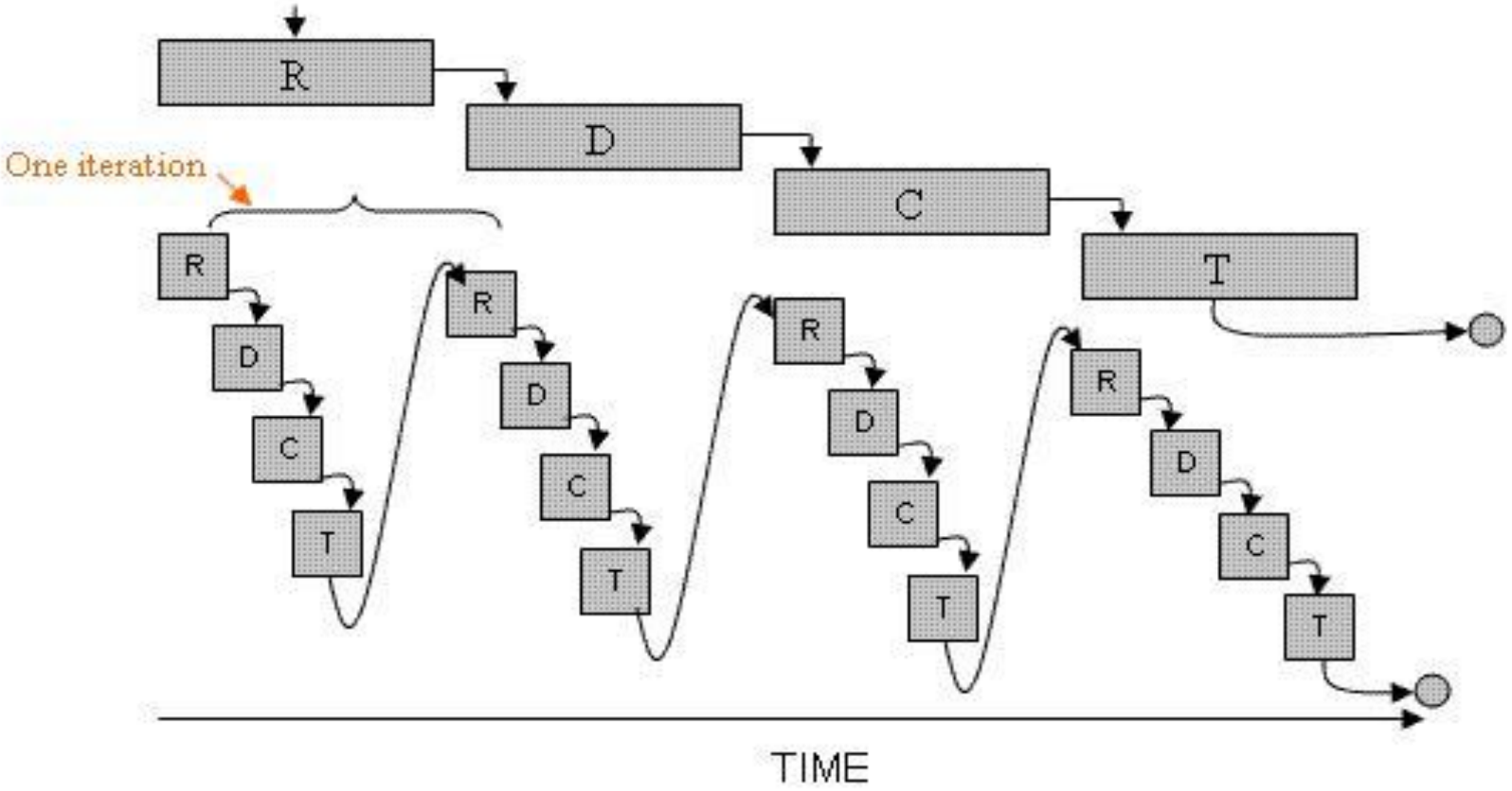
Rational Unified Process



4 Phases :

- Inception
- Elaboration
- Construction
- Transition

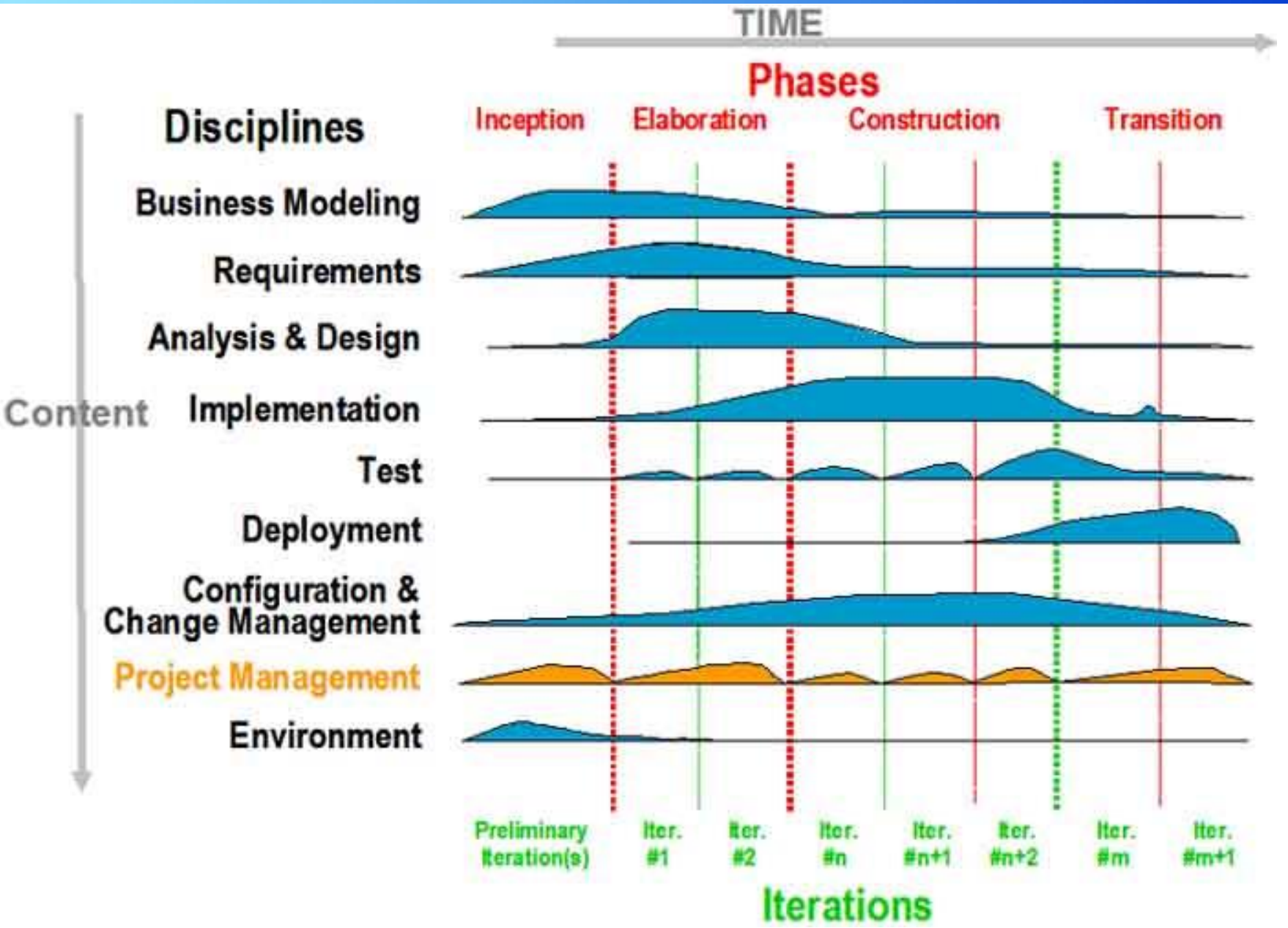
Etapes du RUP



Des étapes dans chacune des phases :

Requirement -> Design -> Code -> Transition

RUP : Activités au fil des itérations



RUP Les rôles et tâches

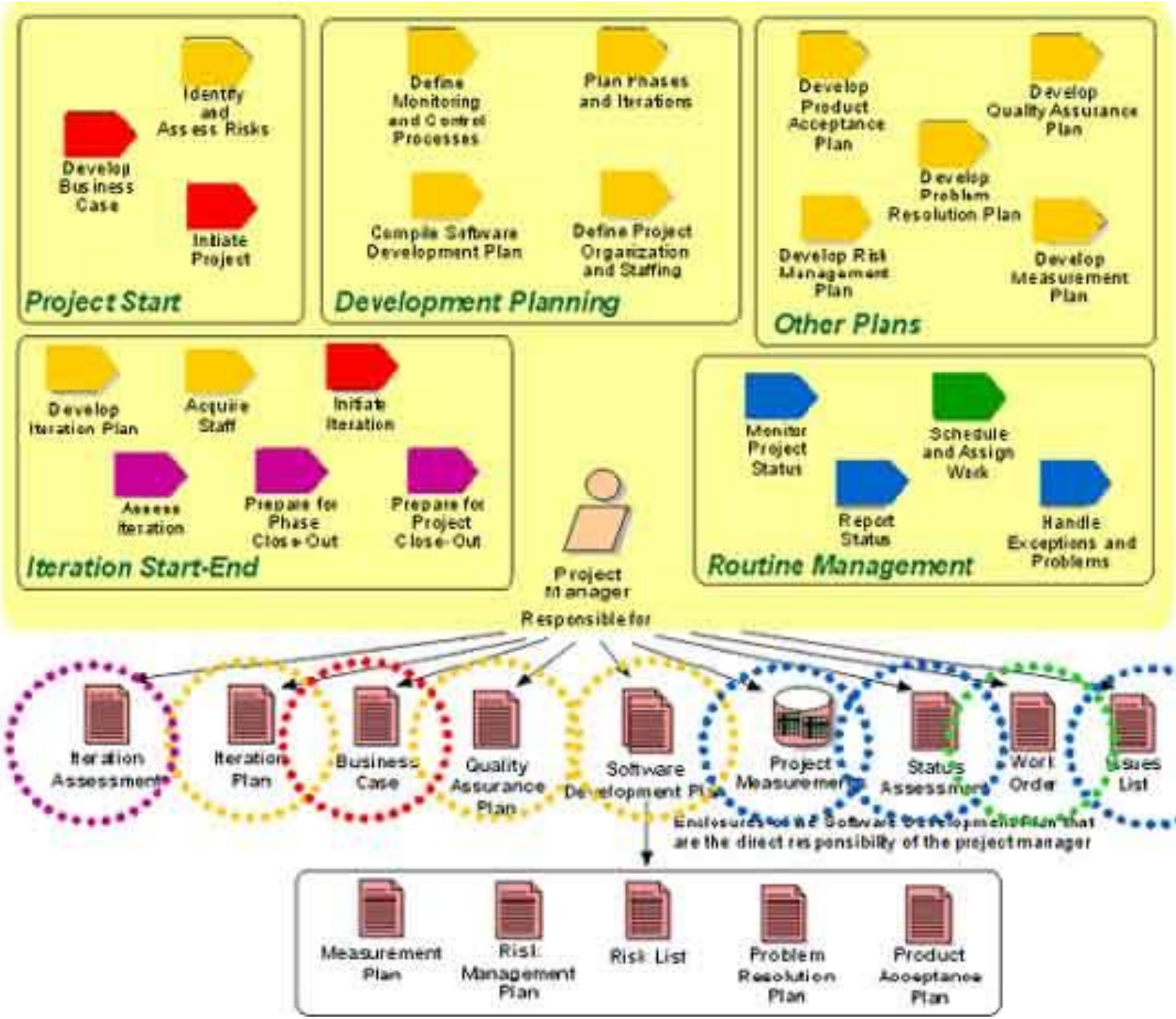
Initiating

Planning

Executing

Controlling

Closing



RUP (2003)

Un framework plutôt qu'une méthode précise

Méta-méthode ?

Qualités :

- Bon support d'emblée pour les modèles, Model Driven Engineering, UML
- Etapes et rôles clairement définis, voire formalisés
- Permet de définir sa propre méthode dans ce cadre -> Certification

Défauts

- Adapté principalement aux grosses structures, gros projets
  - >= 30 homme/an ? 1 M$?
- Lourdeur de mise en place, rigidité et contraintes sur le développement
  - Nombreux artefacts à produire, puis mettre à jour
  - Ratio Spécification / Développement de l'effort consenti

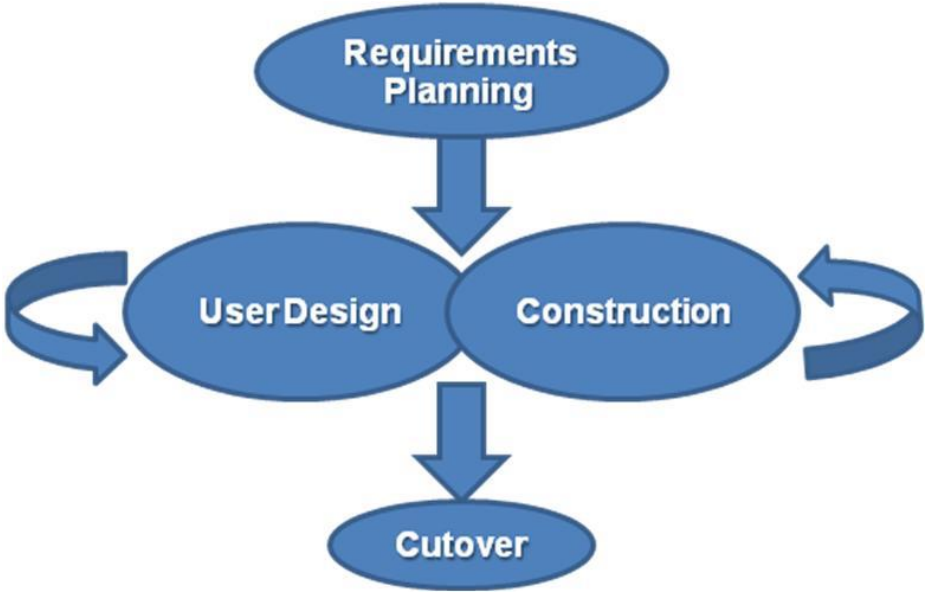
Les méthodes agiles

Rapid Application Development RAD 90+

- Besoins qui évoluent rapidement
  - Time to market important
- Développement des frameworks
  - IDE Environnements de Développement Intégrés
  - Librairies et outils plus largement disponibles
- Prototypage rapide
  - Différent d'une maquette, utile au produit final
- Cycle Itératif
  - Itérations **courtes**
  - Etapes simplifiées, validation par le prototype

RAD : les phases

- Requirements
  - Définir le besoin
  - Accord de principe avec le client sur l'objectif
- User Design
  - Modèles et prototypes démontrables
  - Approbation via des démonstrations fréquentes au client
- Construction
  - Développement du code
  - Mise en place des tests (validation + non-regression) et exécution
  - Intégration dans le prototype
- Cutover
  - Fin du cycle
  - Consolidation d'un résultat stable
  - Transition / Améliorations



Manifesto for Agile Software Development

We are uncovering better ways of developing
software by doing it and helping others do it.
Through this work we have come to value:

Individuals and interactions over processes and tools
Working software over comprehensive documentation
Customer collaboration over contract negotiation
Responding to change over following a plan

That is, while there is value in the items on
the right, we value the items on the left more.

Manifeste Agile

- 17 signataires influenceurs et à la pointe du domaine
  - Auteurs des méthodes SCRUM, eXtreme Programming, …

Nous découvrons comment mieux développer des logiciels par la pratique et en aidant les autres à le faire. Ces expériences nous ont amenés à valoriser :

- Les individus et leurs interactions plus que les processus et les outils
- Des logiciels opérationnels plus qu'une documentation exhaustive
- La collaboration avec les clients plus que la négociation contractuelle
- L'adaptation au changement plus que le suivi d'un plan

Nous reconnaissons la valeur des seconds éléments,
mais privilégions les premiers.

Les 12 principes sous-jacents (2)

Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.

Accueillez positivement les changements de besoins, même tard dans le projet. Les processus Agiles exploitent le changement pour donner un avantage compétitif au client.

Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.

Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet.

Les 12 principes sous-jacents (3)

Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et le soutien dont ils ont besoin et faites-leur confiance pour atteindre les objectifs fixés.

La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face.

Un logiciel opérationnel est la principale mesure d'avancement.

Les processus Agiles encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.

Les 12 principes sous-jacents

Une attention continue à l'excellence technique et à une bonne conception renforce l'Agilité.

La simplicité – c'est-à-dire l'art de minimiser la quantité de travail inutile – est essentielle.

Les meilleures architectures, spécifications et conceptions émergent d'équipes autoorganisées.

À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence.

Une anti-méthode ?

- Non ! Mais des méthodes moins rigide, appuyée par des pratiques plutôt qu'un *process*
- Focus sur les humains et leurs interactions

Agilité : caractéristiques

- Petits incréments
  - itérations courtes, adaptabilité forte
- Rôles flous au sein de l'équipe
  - responsabilité *collective* (!)
- Avancement mesuré par l'avancement d'un prototype fonctionnel
- Communications orales à tous niveaux
  - Open space, tableaux, réunions orales : font partie de la méthode
  - Représentation du client, interactions orales et démonstrations concrètes

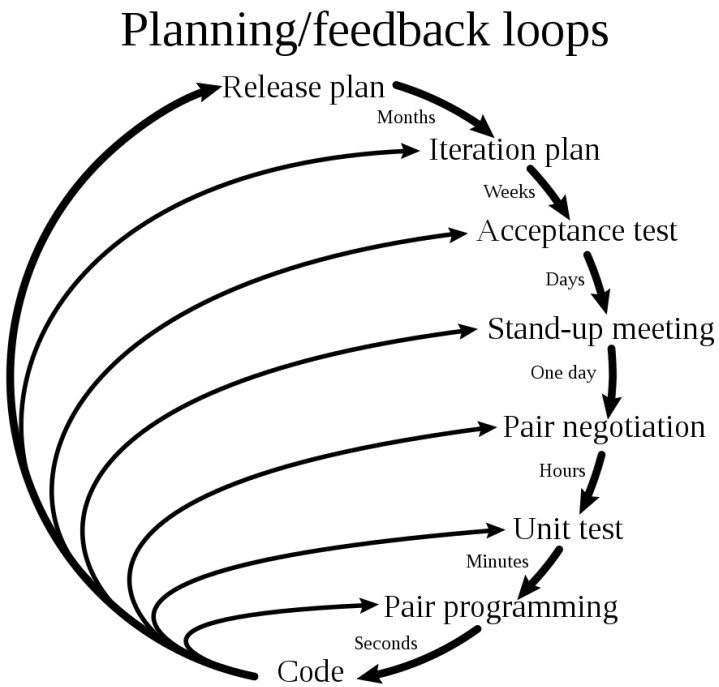
Manifeste agile : impact

- Entériner, justifier, légitimer des pratiques courantes
- Une « contre »-méthode adaptée aux personnes éduquées
  - Reste très supérieur à un développement ad hoc
- Bien adapté aux « petits » projets
- Prise en compte forte du changement, définition de la cible au fil de l'eau
- Nombreuses méthodes dans ce courant agile
  - https://en.wikipedia.org/wiki/Agile_software_development
  - Focus SCRUM, XP dans ce cours

XP : eXtreme Programming

XP

- Moins organisé, une méthode beaucoup plus floue
- Quatre activités :
  - Coding
    - Activité primordiale et majeure
  - Testing
    - Niveau unitaire
    - Niveau « acceptance »/test de validation
  - Listening (!)
    - i.e. connaitre le domaine métier et comprendre le client
  - Designing
    - On préférerait s'en passer mais il faut parfois en faire un peu



XP : Les Valeurs

- Communication
  - De préférence orale, tableau blanc, standup
- Simplicity
  - Pourquoi faire compliqué ? Idée de KISS, Keep It Simple Stupid
- Feedback
  - Via des tests frequents et en continu
  - Au niveau humain à travers des échanges frequents dans l'équipe et avec le client
- Courage
  - De coder dans l'immédiat plutôt que dans un plan bien défini (KISS)
  - De retravailler les codes (*refactoring*), être prêt à en jeter, se remettre en question
- Respect
  - Des autres : ne pas casser la branche master !
  - De soi : toujours faire de son mieux !

XP « Pratiques » Feedback

- Planning Game (~1 semaine)
  - Planification des besoins :
    - user stories
    - Visibilité sur quelques itérations
  - Planification des tâches qui en découlent
    - Tâches de développement
  - ✓ 3 phases pour chaque planning : exploration, commitment, steering

Planning game

- Release Planning
  - Exploration : User story + Estimer (maquette ?) + redécouper
  - Au client de travailler ses scenarios
  - Commitment :
    - Valeur client : critique, forte valeur ajoutée, pratique
    - Risque : 2 points chaque pour : complétude, volatilité, complexité
  - Steering : adapter les éléments existant
- Iteration Planning
  - Exploration : Construire des “task card”
  - Combiner / redécouper jusqu’à pouvoir estimer la charge
  - Commitment : s’affecter une tâche
    - On fait attention à l’équilibre de charge et au développement durable
  - Steering : realisation, Test-Driven, Pair Programming

XP Pratiques

- Conventions de codage
  - Soutenue par des outils
- Responsabilité collective du code
  - E.g. un git accessible à tous
  - On peut tous patcher un problème n'importe où
- KISS : conception minimaliste
  - Toujours aller à l'essentiel
  - Accepter qu'il faudra parfois faire du refactoring
- Releases incrémentales
  - Itérations de l'ordre de la semaine, releases stables régulières

XP « Pratiques »

- Intégration Continue
  - Un serveur qui reconstruit, teste et déploie les artefacts à chaque modification du code
  - Tests de non regression
  - Outils de qualimétrie, e.g. SonarQube
  - Cf. travis-ci, gitlab « pipelines », circle-ci, appveyor, teamcity,…
- Test-Driven Development
  - Tests écrits en premier lieu, il faut donc implanter pour les passer
  - Red -> Green -> Refactor
- Refactoring : modifications du code à sémantique constante
  - Recherche des « code smells »
  - A cause du KISS nécessaire à divers moments

Pair Programming

- Programmation en binomes
  - Un qui tapes, l'autre qui regarde et donne des indications
  - Augmente la qualité, évite les blocages, les fautes bêtes
  - Force à rester sur le sujet (mail…)
  - Bilan industriel mitigé :
    - Senior/Senior : perte de productivité

Conclusion XP

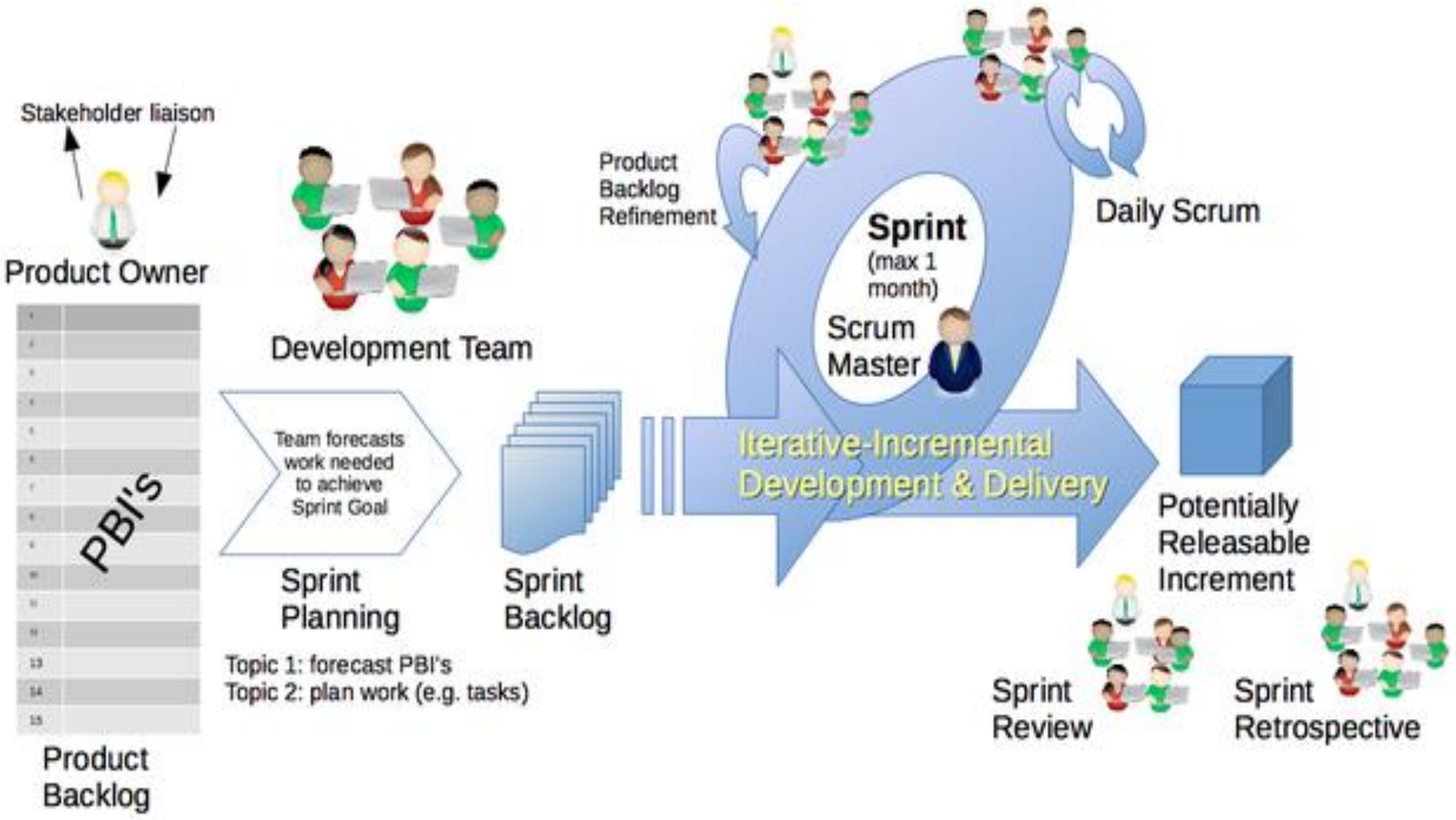
- Reste une « méthode » pour développer des systèmes
  - Beaucoup moins rigide et contraignant qu'un cycle classique
  - S'appuie sur des qualités humaines de l'équipe de développement, beaucoup de confiance mutuelle
  - Beaucoup des pratiques sont un peu controversées, mais certaines sont devenues standard (e.g. intégration continue)
  - Adoption industrielle assez faible au final

SCRUM

Position de SCRUM

- Archétype d'une méthode « Agile »
- Gros succès « médiatique »
- Adoption importante au niveau industriel
- Une méthode bien définie, raffinée

Vue d'ensemble



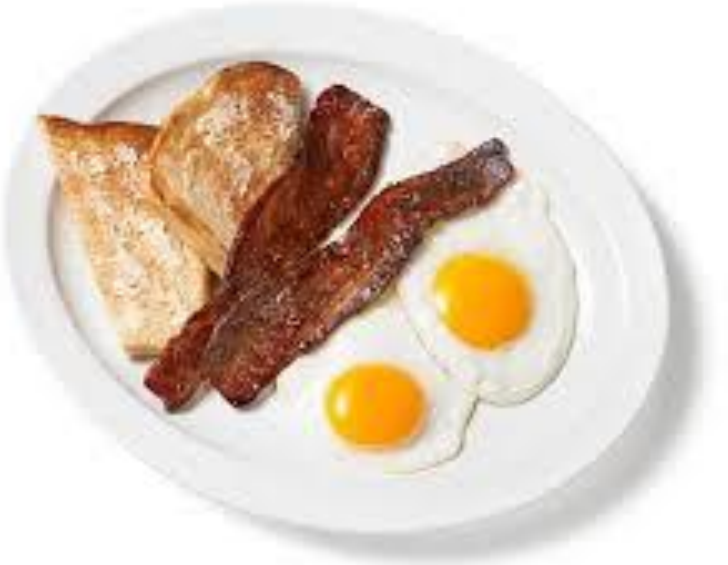
Rôles en SCRUM

- Team member :
  - Un seul rôle pour tous les participants au développement
  - L'équipe est réduite, <= 10 personnes
- Scrum Master :
  - Facilitateur de l'équipe; pas son chef
  - Aplanit les difficultés logisitique, interface avec l'extérieur
  - Maître de Cérémonie, lance et dynamise les échanges
- Product Owner :
  - Réprésentant du client, mais rattaché à l'équipe
  - Intégré au processus de développement

- Les autres :
  - Stakeholder : a un intérêt financier au développement de l'outil
  - Manager : ressources humaines, logistique,…
  - Customer : le client

Rôles en SCRUM

- Team member :
  - Un seul rôle pour tous les participants au développement
  - L'équipe est réduite, <= 10 personnes
- Scrum Master :
  - Facilitateur de l'équipe; pas son ch
  - Aplanit les difficultés logisitique, ir
  - Maître de Cérémonie, lance et dyna
- Product Owner :
  - Réprésentant du client, mais rattach
  - Intégré au processus de développen



- Les autres :
  - Stakeholder : a un intérêt financier au développement de l'outil
  - Manager : ressources humaines, logistique,…

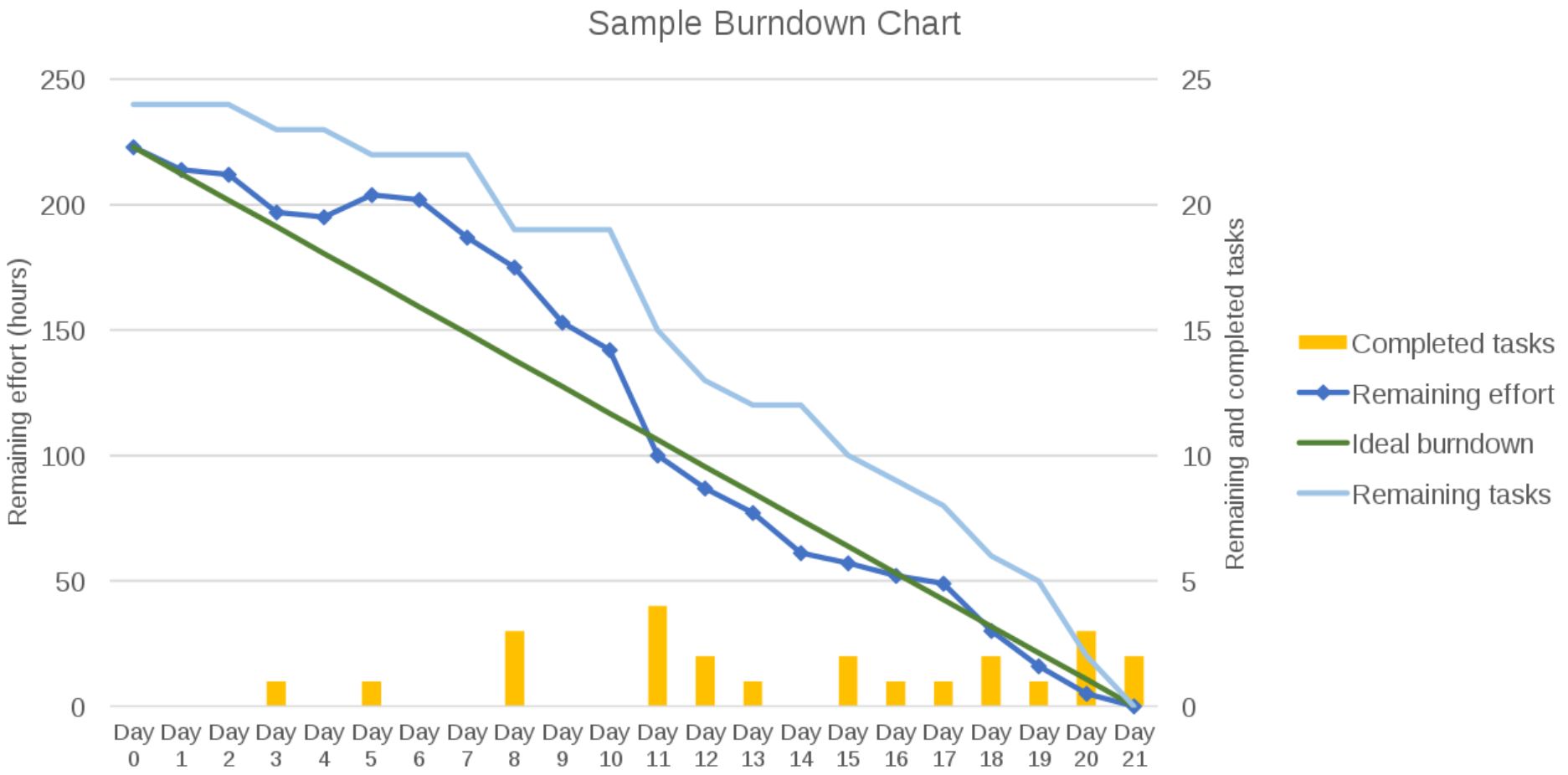
Eléments de SCRUM

- Product Backlog
  - Composé d'items (PBI) correspondant à des *features* utilisateurs
  - Commence plein, le but est de le vider !
  - Evalués en termes d'importance pour le client, de difficulté de développement
  - Descriptions assez informelles mais précises
  - Mis à jour (ajouts, raffinements, redécoupages) au fil du cycle

Sprint Planning

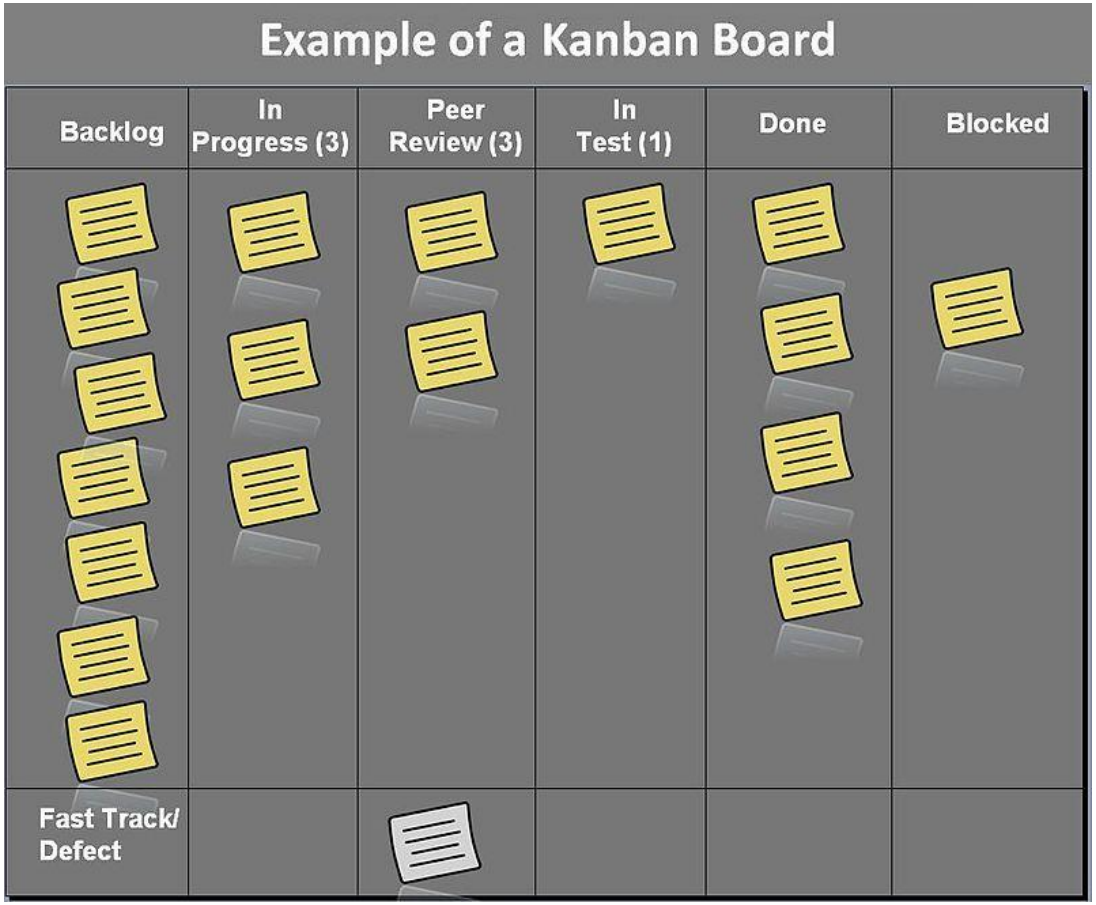
- Sprint Planning ~4h / Sprint 2 semaines
  - Définition des besoins traités dans l'itération (team + PO +SM)
    - e.g. User Stories, I as XXX want YYY so that ZZZ
    - Discussion, schémas, role-play
    - Sélection consensuelle de ceux qui seront traités
  - Définition des taches de développement (team)
    - Raffinement des PBI sélectionnés pour l'iteration :
      - Le besoin fonctionnel induit des charges de développement
    - Pricing des taches
      - E.g. Fibonacci : 1,2,3,5,8,13,21 X
      - Importance pour le client
    - Estimer la quantité raisonnable pour le sprint
      - Célérité de développement de l'équipe
    - Construction du Sprint Backlog

Exemple Célérité : BurnDown Chart



Le Sprint

- Sprint :
  - Phase de développement, on vide le Sprint Backlog
- Gestion de l'avancement avec tableaux Kanban
  - Physique et/ou outillé, e.g. Trello



Daily Scrum

- La « mêlée » quotidienne
  - Stand-up
  - Time-box ~15 minutes
  - Team + Scrum Master (autres OK en spectateur)
  - Horaire précis, régulier.
- Round Robin :
  - Qu'ai-je fait hier ?
  - Que vais-je faire aujourd'hui ?
  - Quels sont les problèmes que je rencontre ?
    - Traitement offline, mais mobilisation des compétences
- Rôle du Scrum Master :
  - Facilitateur, résoudre les blocages, gestion des problèmes d'environnement
  - Organise la réunion sans la diriger

Pratiques de Développement

- Open space
- Responsabilité *collective*, revue de code
- Approche Dirigée par les tests
  - On rédige le test, il échoue, on implante pour que le test passe
- Suivi du développement
  - Visuellement à tout instant Kanban physique
  - Burndown charts…
- Suivi individuel ?
  - Forte pression de résultat
  - Suivi de la célérité des membres de l'équipe
  - Popularité de SCRUM dans le milieu des prestataires

Transition

- Sprint Review (team+PO)
  - Démonstration d'un prototype opérationnel au PO
  - Bilan vis-à-vis du Backlog de Sprint
    - Qu'as-t-on réussi à implanter ?
- Sprint Retrospective (team+PO)
  - Qu'est-ce qui s'est bien passé ?
  - Que peut-on encore améliorer ?

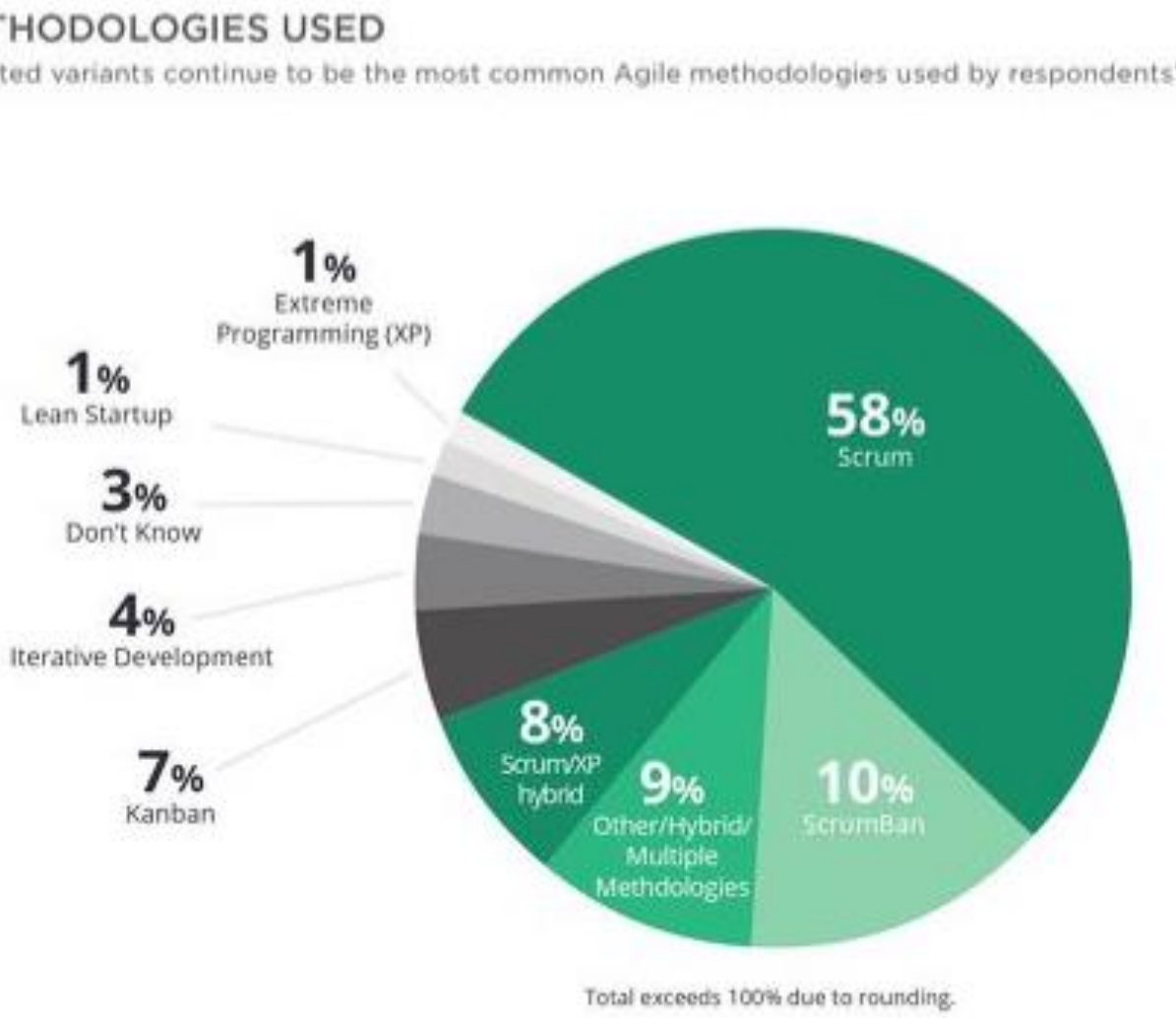
Scrum : valeurs

- Commitment : chacun des membres est engagé dans la réalisation des objectifs de l'équipe
- Courage : ensemble nous aurons le courage de résoudre les problèmes pour avancer
- Focus : on se concentre uniquement sur le backlog et les tâches identifiées dans le planning game
- Openness : tous les participants au projet s'engagent à être ouverts et transparents vis-à-vis du projet
- Respect : les membres de l'équipe se respectent mutuellement et se font confiance professionellement

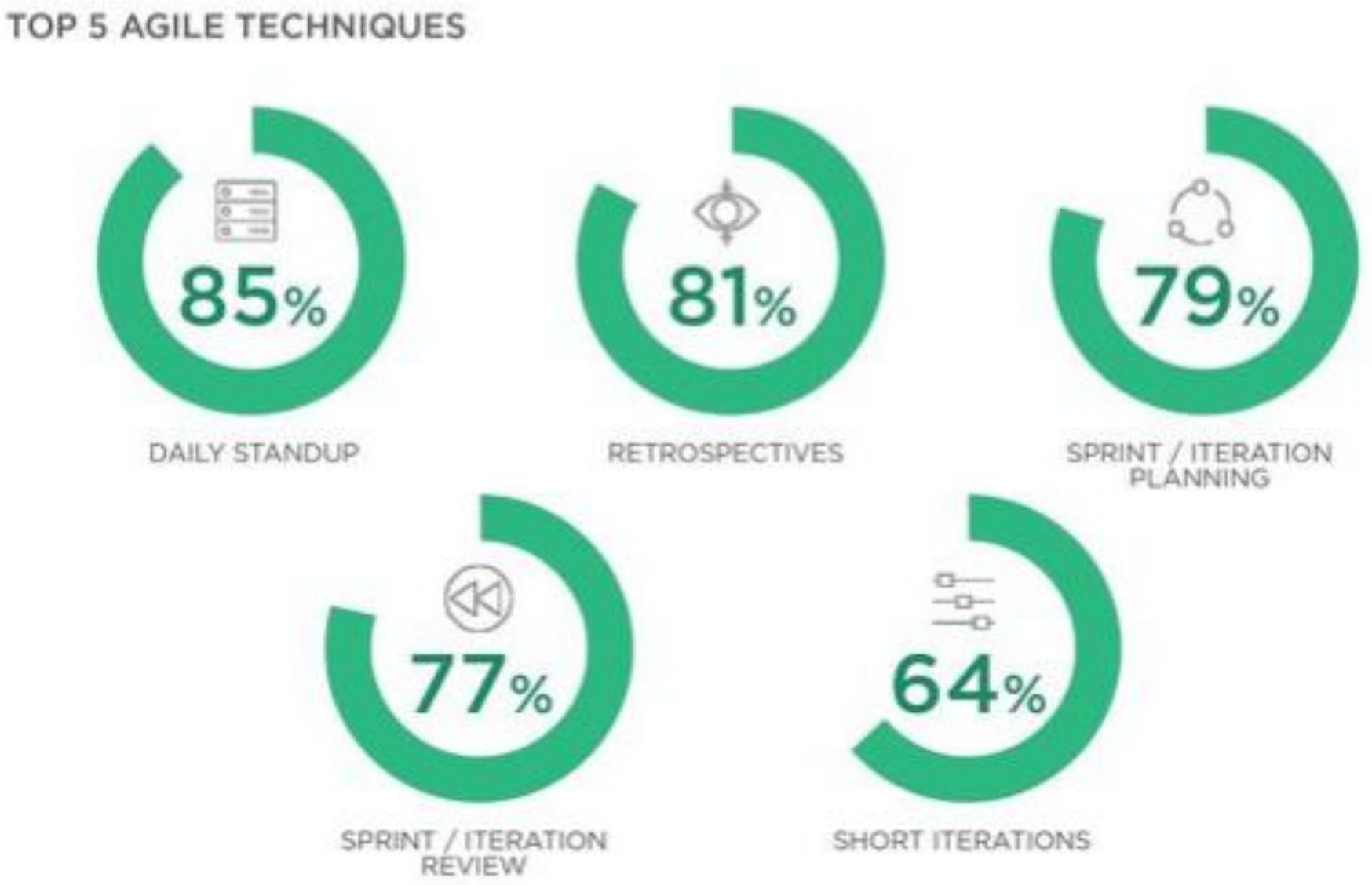
Scrum : bilan

- Une méthode bien adaptée aux équipe de taille modérée
  - Une organisation de la journée bien réglée
  - /!\ aux dérives de management
  - Ne passe pas forcément à l'échelle
- Bilan industriel
  - Adoption assez forte, pas mal d'entreprise qui se réclament appliquer du scrum
  - Le rôle du scrum master est plus ou moins bien observé (!=chef de projet)
  - La partie planning, backlog etc… connait une forte adoption
  - Des méthodes dérivées de Scrum existent e.g LeSS + Scrum/V…

Adoption (state of agile report)



Pratiques agile



Conclusion Agilité

- L'itération est primordiale
  - Toutes les méthodes modernes sont itératives
- Agilité
  - ++ Réponse au changement,
  - ++ cibles mouvantes et/ou mal définies OK
  - - La qualité en general diminue
  - - Le problème de *dette technologique* peut se poser
  - ++ meileure prise en compte du besoin client
  - +- Nécessite des petites équipes à la fois très motivées et très compétentes
- Application/Mise en oeuvre
  - Transition sur ce mode de fonctionnement (e.g. SCRUM) assez rude
  - Reste partiellement idéalisé (*très* optimiste ?) comme les méthodes plus Classique
  - Forte adoption en partie pour l'aspect (micro)managérial et tous les outils de suivi

References/En savoir plus

La plupart des images ont été prises sur Wikipédia

Essential Scrum: A Practical Guide to the Most Popular Agile Process (Rubin)

Extreme Programming Explained: Embrace Change (Beck)

On recommande ce portail : https://en.wikipedia.org/wiki/List_of_software_development_philosophies

Les pages SCRUM et XP sont de bonne qualité.

Il peut être intéressant pour certaines pages de regarder aussi les version françaises (effet local important).

Structure de l'épreuve

Le problème : ~15 points

- Use case :
  - Un peu de commentaire bienvenu
  - Subsystem, acteurs, use case, liens
  - Evitez de trop surcharger (extend, include)
- Classes Métier
  - Classes, attributs de type simple
  - associations bi-directionnelles + cardinalités + rôles
  - Pas de dynamique, pas d'acteurs, pas de système
- Fiche Détaillée
  - On demande *une* fiche en général
  - Attention au début/fin de l'action
  - Titre, Acteurs, Pré-condition, SN, Post-Condition, Alt, Ex
    - Bien numéroter
    - « Le système » « L'acteur » + trigger

ER 1

- Séquence d'Analyse
  - C'est assez simple !
  - Ligne de vie de(s) l'acteur, ligen de vie du système
  - Messages porteurs de valeurs concrètes
  - En déduire les signatures => visibles sur ce diagramme
- Test de Validation
  - Reproductible ! Non-ambigu
  - Contexte : Etat de l'applicatif faisant que c'est ce R.A. particulier
  - Entrée : toutes les entrées/saisies etc…
  - Scenario : que les étapes du testeur, reproductible
  - Résultat Attendu : au niveau métier, mise à jour des données
  - Moyen de Vérification : essayer de proposer autre chose que visuel
- Examen complété en général par 4 ou 5 questions à 1 point
- Attention à la gestion du temps !
  - 1 point = 5 minutes si on compte 20 minutes pour bien lire le sujet