Partie 2 : le développement logiciel

Chapitre 1 : l'ingénierie logicielle

Table des matières

Définitions

Génie logiciel

Le **génie logiciel** (en anglais *software engineering*) « étudie les **méthodes de travail** et les **bonnes pratiques** des ingénieurs qui développent des logiciels. »**

Ces bonnes pratiques couvrent la conception, l'amélioration et l'installation des logiciels, afin de favoriser la **production et la maintenance de composants logiciels fiables.**

** http://fr.wikipedia.org/wiki/Génie_logiciel, 03-04-2018

Pratique

« Une pratique est une approche concrète et éprouvée qui permet de résoudre un ou plusieurs problèmes courants ou d'améliorer la façon de travailler lors d'un développement* » Aubry, p3

* Lors d'un développement ou de manière générale, dans son métier
** Scrum, Claude Aubry, Dunod, 2011 3ème édition

Méthodes de travail

Une méthode de travail fixe **comment** on fait les choses, quelles sont les **responsabilités de chacun** et les **interactions entre les personnes**.

Une méthode de travail est souvent vaste et complexe. Elle concerne plusieurs employés qui font chacun des activités différentes. Ces activités forment un tout.

L'objectif est de fixer une **démarche** pour réaliser chaque activité dans les meilleures conditions.

Pourquoi décrire
ses méthodes de travail?

Formalisation

(1) Dans une optique de transparence

- Les comprendre
- Les partager
- Les communiquer dans l'équipe / entre équipes

Définir une méthode de travail permet de **comprendre** comment les activités s'articulent ensemble (organisation du travail). Cela permet à chacun de **connaître son rôle**.

➔ Toutes les personnes qui ont le même rôle travaillent de la même façon.

Pourquoi décrire
ses méthodes de travail?

Définir une méthode de travail permet de la **partager** plus facilement et de **faciliter l'apprentissage** d'un nouveau membre de l'équipe.

Définir une méthode de travail permet de **communiquer** plus facilement entre plusieurs employés qui travaillent à des activités différentes mais dans un même but. Cela favorise l'utilisation d'un **langage commun**.

Pourquoi décrire
ses méthodes de travail?

(2) Dans une optique de pérennité

Parce que le temps passe, les équipes changent, les activités évoluent...

- ➔ Permettre de suivre les processus et leurs changements au fil du temps.
- ➔ Avoir **l'assurance de ne pas perdre les connaissances** lorsque quelqu'un quitte l'entreprise.

Pourquoi décrire
ses méthodes de travail?

(3) Dans une optique de satisfaction des clients
approche/démarche qualité (voir chapitre dédié)

- Améliorer la réponse aux attentes du client :
  - **Besoins (Specifications).**
  - **Budget (Budget).**
  - **Délais (Timeframe).**

Processus

Le génie logiciel

touche au **cycle de vie des logiciels**.

Cycle de vie

Cycle de vie des « gros » logiciels :

- Grand nombre de personnes impliquées :
  - Clients, utilisateurs
  - Spécialistes du domaine
  - Responsables de projet
  - Développeurs
  - Testeurs
  - …
- Long temps de vie (développement et utilisation).

Cycle de vie

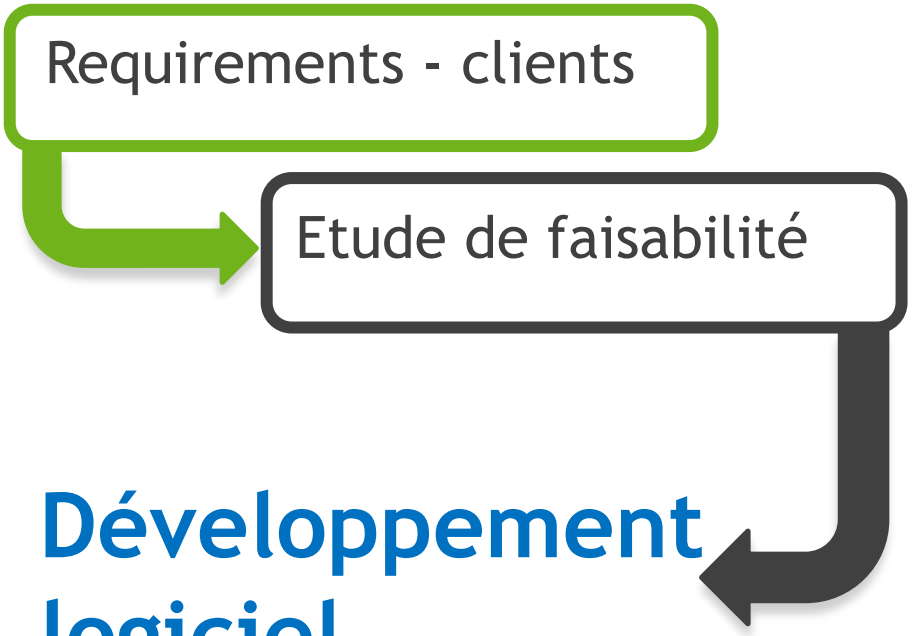
Cycle de vie des « gros » logiciels (suite)

- Différents processus :
  - Construire le logiciel
  - Installer le matériel
  - Gestion de projets (affectation des ressources, budget)
  - …
- Plusieurs versions.
- Beaucoup d'information répartie entre différentes personnes.
- Spécifications et validation essentielles.

Cycle de vie de « petits » logiciels : plus simple.

Activités

Du cycle de vie logiciel



Activités

- Spécifications.
- Analyse formelle.
- Conception (technique).
- Implémentation : codage & tests unitaires.
- Intégration & tests.
- Livraison.
- Installation et mise en production.
- Utilisation – amélioration – maintenance.

Software Development Life Cycle (SDLC)
Multiples activités,
Interactions +/- complexes.

Requirements

Demandes émanant

- du client (dans le cas d'un projet).

- du marché, de la concurrence, de la veille technologique, des partenaires… (dans le cas d'un produit et éventuellement, d'un projet).

- ProjetAE :
  - Document remis par le client.
  - **Input** du processus de développement logiciel.

Requirements

Problème :

- Souvent incomplets.
- Parfois ambigus.
- Ou même parfois incorrects.

ProjetAE – exemple : Tous les utilisateurs pourront effectuer des recherches.

- Responsable/aidant est le seul qui peut effectuer toutes les recherches.

Spécifications

- Traduction des demandes-utilisateurs exprimées dans les « requirements », la demande.
- Traduction orientée vers le développement-logiciel.
- Elaboration du cahier des charges.

- Objectif : définition de ce que le logiciel doit faire.

Spécifications

- ProjetAE
  - Analyse énoncé, définition objectifs de l'application à développer, réflexion sur les IHMS, analyse par les données…
    - Que veut le client ? Gestion site **privé**
    - Que faisons-nous en cas de retrait d'un objet mis en vente ? Conséquences ?
    - Devons-nous prévoir des impressions ?
    - …

Spécifications

- Output :
  - Liste des objectifs.
  - Diagramme de use cases (ou équivalent).
  - Prototypes d'écran.
  - Définition des données.

  - (Si client) Cahier des charges à communiquer au client sur lequel le client va s'engager.
  - (Si produit) liste des demandes.

- ProjetAE
  - **Output** : une grande partie du document d'analyse.

Analyse formelle

- **Que** va faire le système ?
  - Compréhension claire de ce que sera le système et de tous les concepts sous-jacents.
  - Vérification que cela correspond à ce que demande le client.

- Output :
  - Description de chaque fonctionnalité.

Analyse formelle

- ProjetAE
  - En détails, analyse de l'UC « offrir un objet ».
  - **Output** : interaction avec le client ; contrôle de chacun des champs et messages d'erreur associés.

Conception

- Modélise et spécifie le **comment**.
- Architecture : **organisation d'un système logiciel**.

- ProjetAE
  - Réflexion sur l'architecture d'application.
  - Adaptation architecture proposée par Mr Leleux au problème posé.
  - **Output** : architecture de classes, rapport d'infrastructure.

Conception

- Architecture : (vue d'une analyste!)
  - Diagramme de classes – OO – responsabilités, collaboration, communication.
  - Organisation interne : découpe en composants.
  - Intégration des composants externes : comment appeler leurs API ? Comment isoler ces appels ?
  - API que l'on fournit pour que nos clients puissent intégrer leur propre code (exemple : validation d'un champ).
  - API que l'on fournit pour être appelé en tant que composant d'un autre logiciel.

Implémentation

- **Construction du logiciel**
  - Codage.
  - Intégration de composants externes (ex: logiciel open-source ; achat d'un composant ; composant développé par une autre équipe…).
- Output : code-source et documentation (justification des choix, code documenté).

- ProjetAE
  - **Output** : code (et documentation code).

Tests unitaires

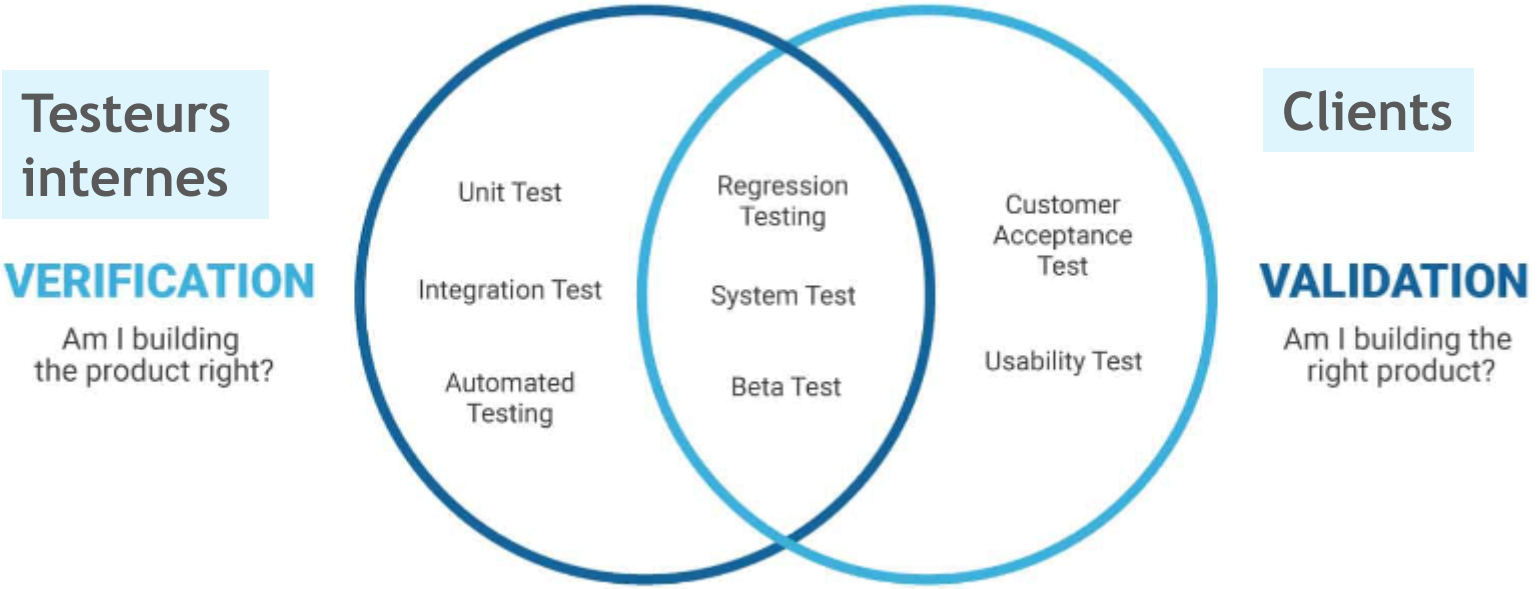
- Tests de blocs de code.
- Tests écrits par les développeurs eux-mêmes pour tester leurs classes ou leur code.
- Tests exécutés par les machines.

  Ces tests font souvent partie de l'implémentation.

- ProjetAE
  - **Output** : tests eux-mêmes.
  - Exécution des tests et enregistrement des résultats de ceux-ci.

Intégrations & Tests

- Représentent le double procédé de **vérification** et **validation** d'un logiciel.



https://www.plutora.com/blog/verification-vs-validation,27/04/2020

Intégrations & Tests

- Assurent que le produit est **conforme aux spécifications**.
- Assurent que le produit fait ce que l'on attend.
- Détectent les erreurs et les bugs.
- Répondent à la question : est-on prêt pour l'étape suivante ?

Vérification

Tests d'intégration

- Vérifient que les composants s'intègrent bien ensemble.
- Vérifient que le produit est compatible avec l'environnement logiciel et matériel prévu chez le client.

Tests fonctionnels

- Vérifient que le produit répond à l'analyse formelle (ou fonctionnelle).

Tests système

- Tests de performance (temps de réponse à une requête).
- Tests en volume.
- Tests de stress (exagération de la demande).

Vérification (2)

Tests système (suite)

- Tests de fiabilité.
  - Ex: résistance aux pannes (coupure réseau…)
- Tests de sécurité.
- Et … Tests d'utilisation en « vitesse de croisière »

- **Output** : tests, exécution de ceux-ci et documentation de l'exécution.

Validation

Tests de validation ou d'acceptation

- Tests formalisés par le client.
- Tests dont le succès assure l'acceptation du logiciel par le client.

- **Output** : exécution des tests par le client, rapport de tests et signature pour acceptation du logiciel.

Livraison, installation & mise en production

- **Mise à disposition du logiciel chez le client.**

- **Output** : logiciel mis en production chez le client, document signé par celui-ci attestant la mise en production.

Maintenance

- Changements apportés au système **après sa mise en production**.
- **Maintenance corrective** : correction de bugs ou de défaillances.
- **Maintenance adaptative** : adaptation de la solution à de nouvelles contraintes techniques.
- **Maintenance évolutive** : modifications du logiciel entraînées par des changements ou ajouts dans les besoins.

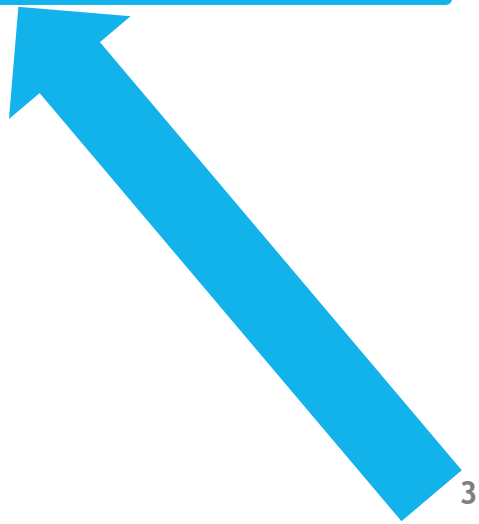
Questions sur les activités?

Le développement logiciel

Développement logiciel

Processus développement

- Créer le logiciel selon les demandes du client (requirements)



Processus Management

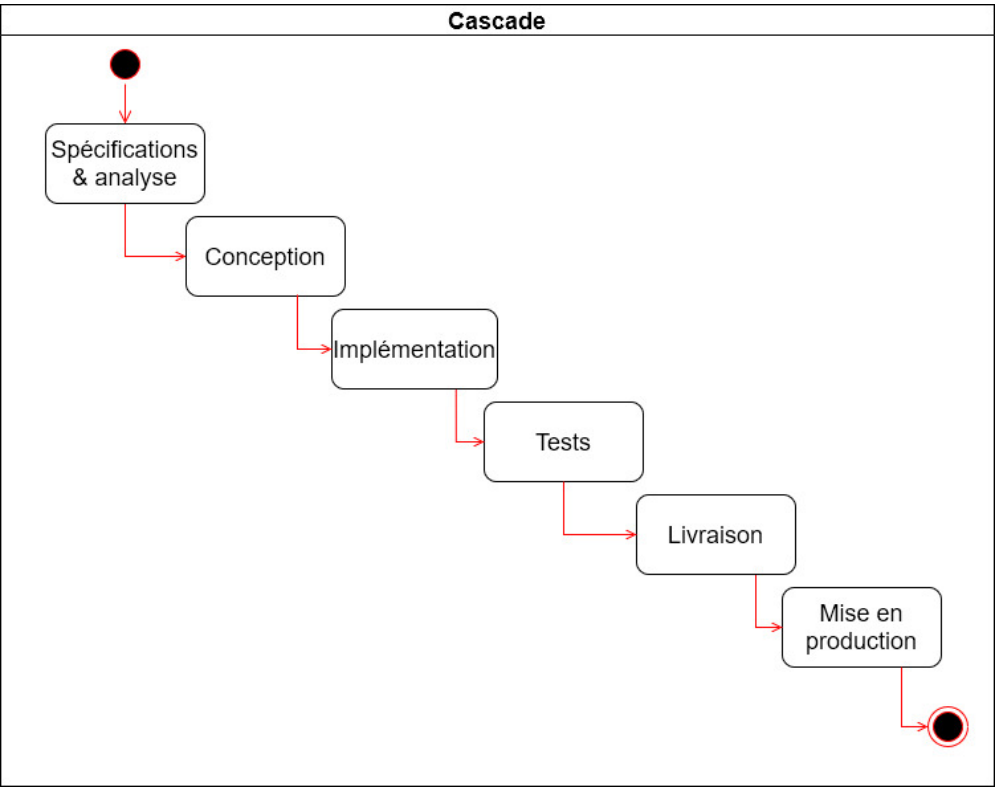
- Planifier le travail
- Planifier les livraisons
- Allouer les ressources
- Gérer le budget, les coûts
- Surveiller l'avancement des travaux
- Gérer les risques.

Processus de développement

- Comment s'enchainent les activités du SDLC ?
- Processus
  - Règles de conduite des développements.
  - Contrôle d'avancement.
- Description du processus
  - (Diagramme d'activités ou équivalent - BPMN).
  - Séquence d'activités.
  - Parallélisme.
  - Itération.

Cycles de vie classiques

Cascade - Waterfall



Chaque étape doit être finie avant que l'étape suivante ne commence.

En théorie, l'optimum

En pratique,

- Très mauvaise gestion du risque.
- Pas de gestion du changement.

  Or, les requirements du client changent ou ne sont pas toujours compris.

Risque = danger potentiel, probabilité de survenance, niveau d'exposition

Cascade : pros & cons

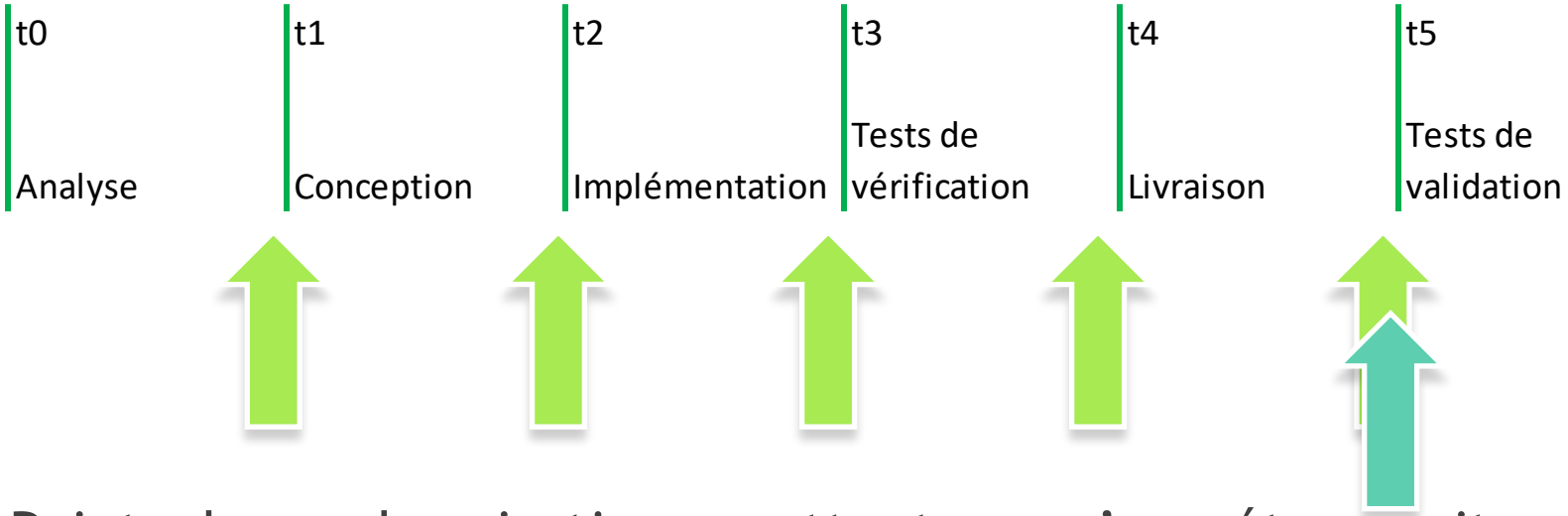
- Documentation produite entre chaque phase.
  - Analyse formelle.
  - Choix de conception.
  - Documents de tests.
  - Description du produit/projet à livrer.
- Bien supporté par les outils de planification.
- Efficace quand les développements sont complexes.

Cascade : pros & cons

- Modèle n'est pas réaliste.
- Spécifications sont figées beaucoup trop tôt dans le processus.
- Spécifications sont validées beaucoup trop tard.
- Pas gestion changement, ni gestion risque.
- Pas de mise à jour de l'analyse en cours de développement.
- Pas de tests en cours de développement *(excepté tests unitaires)*.

Au plus tard les erreurs sont trouvées, au plus cher leur correction !

Cascade : dans le temps



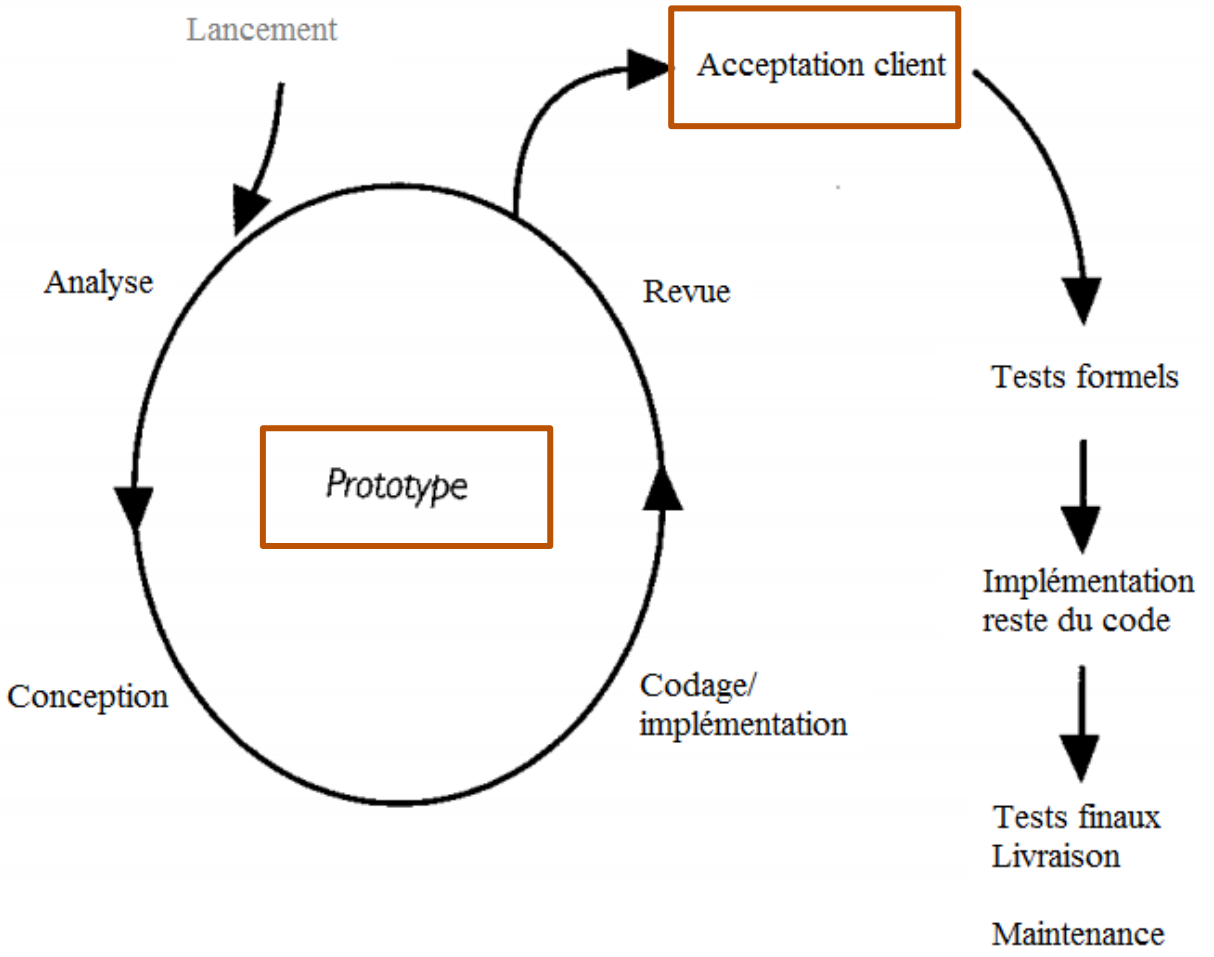
Points de synchronisation : « attente » qu'une étape soit terminée

Outputs bien définis = inputs de l'étape suivante

Interaction avec le client.

Rapid Application Development

Prototyping SDLC

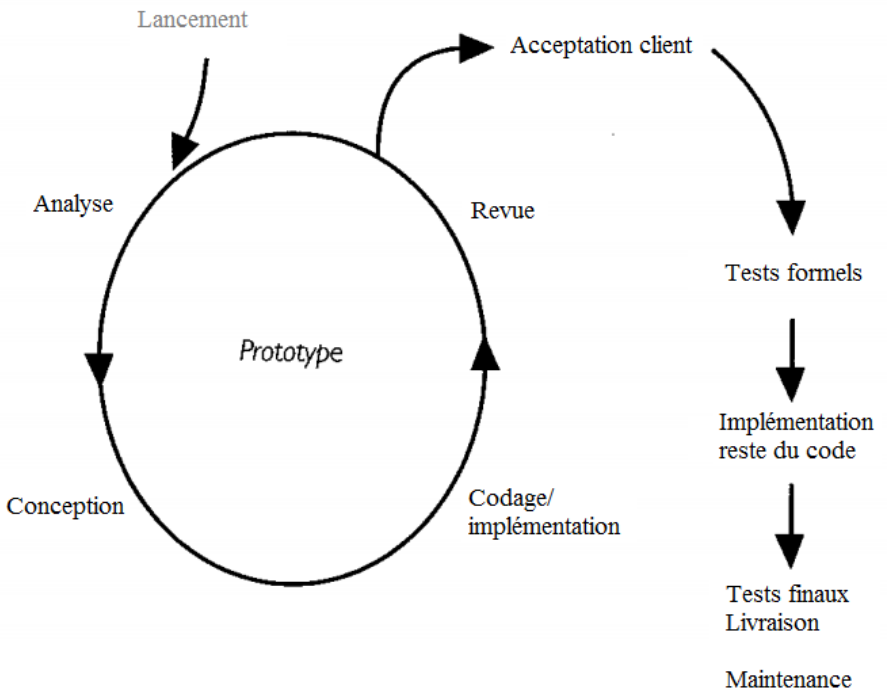


Réaction au modèle en cascade

Mettre la gestion du risque au centre.

- Eviter le risque que le client n'accepte pas le projet livré.
- Tester via prototype les parties difficiles du système et bcp de fonctionnalités.

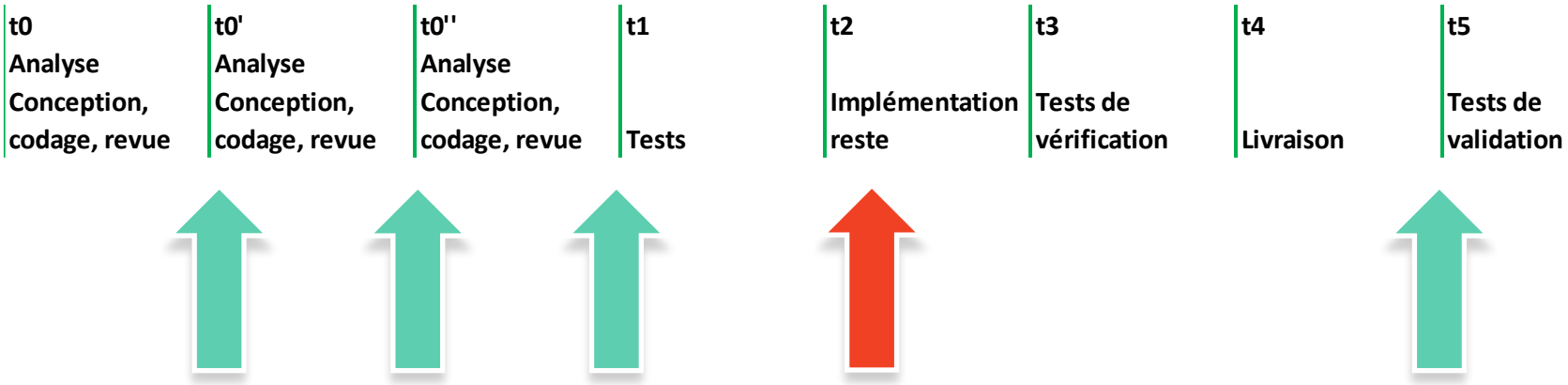
RAD : pros & cons



En pratique,

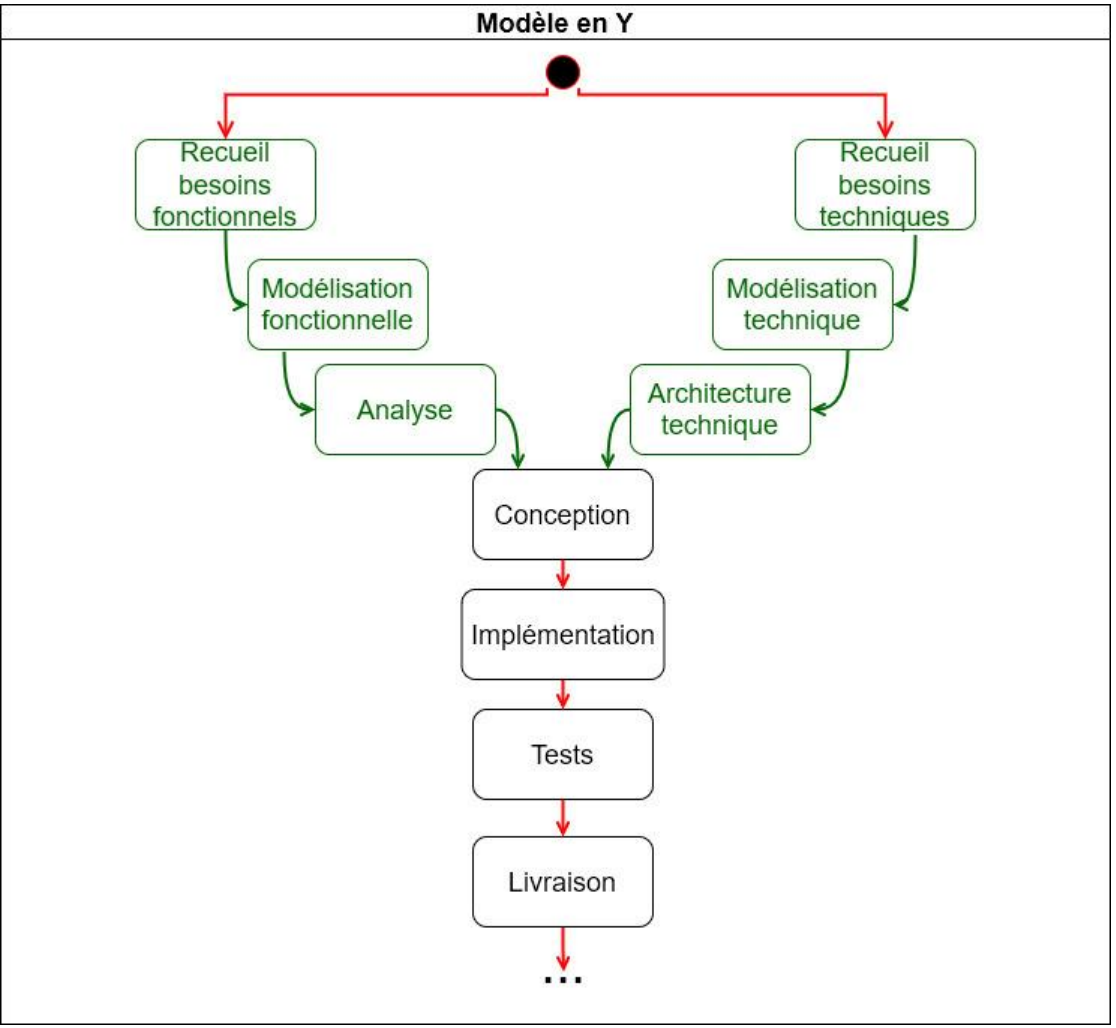
- Livraison à temps.
- Validation dès le début du dév.
- Manque de documentation.
- Système avec des structures très pauvres.
- Changements tout aussi difficiles.

RAD : dans le temps



Interactions avec le client

Modèle en Y



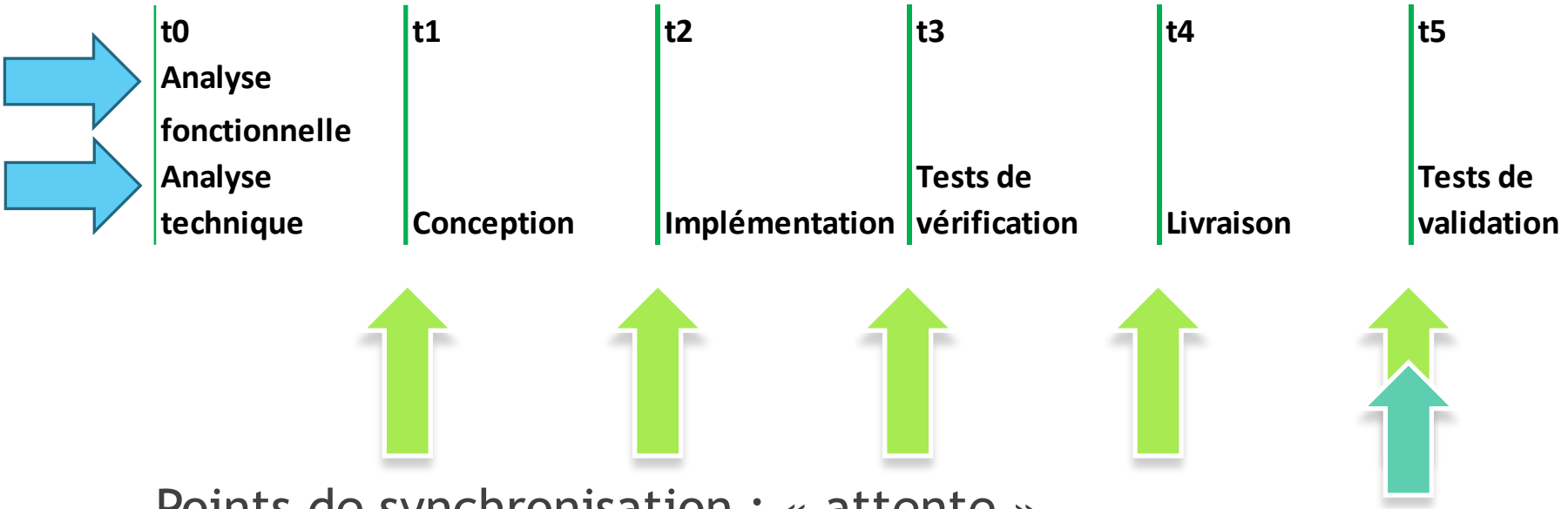
Réaction au RAD

Montée en puissance de l'architecture.

Réduire le risque technologique.

Réintroduire les « bonnes pratiques ».

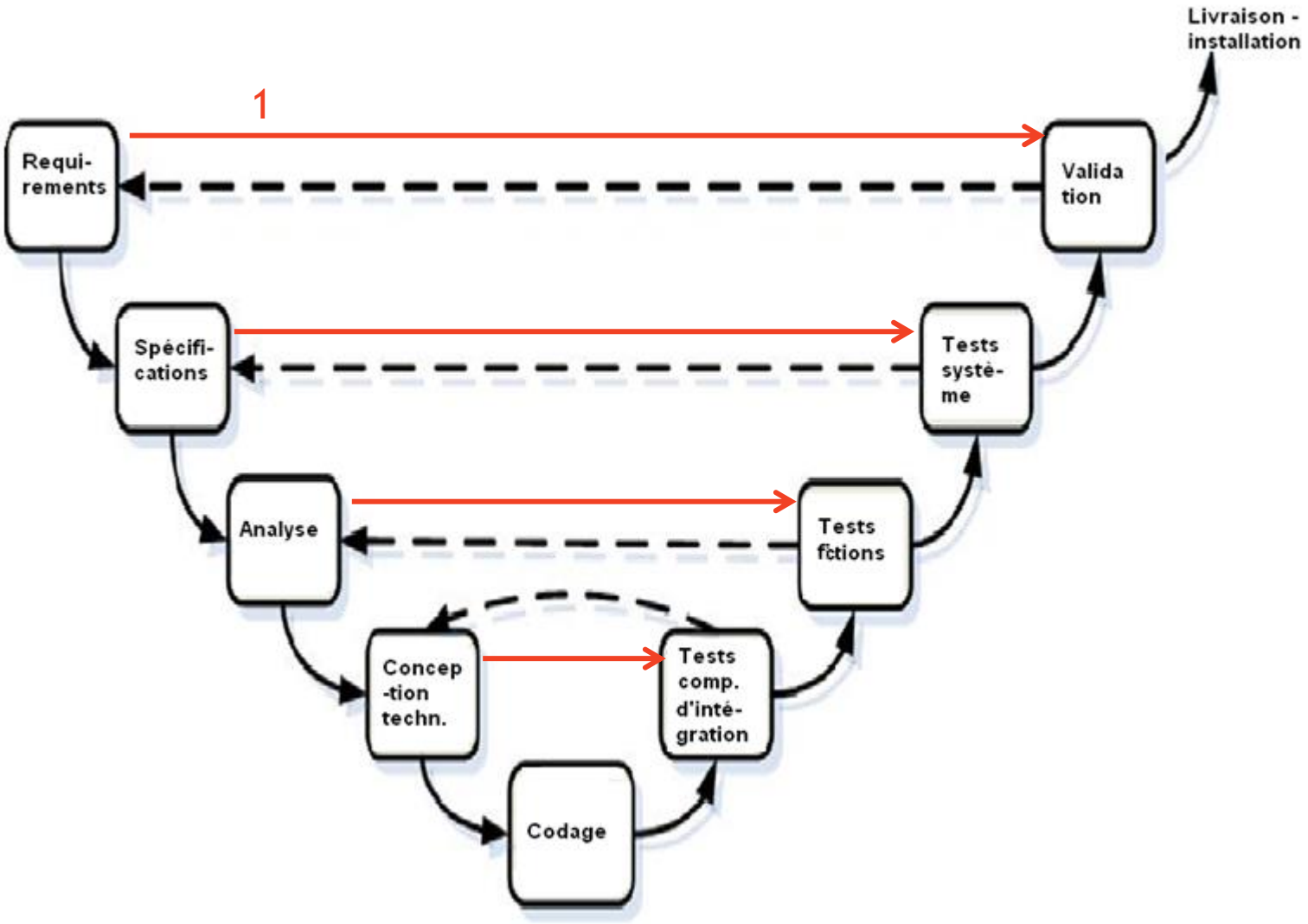
Y : dans le temps



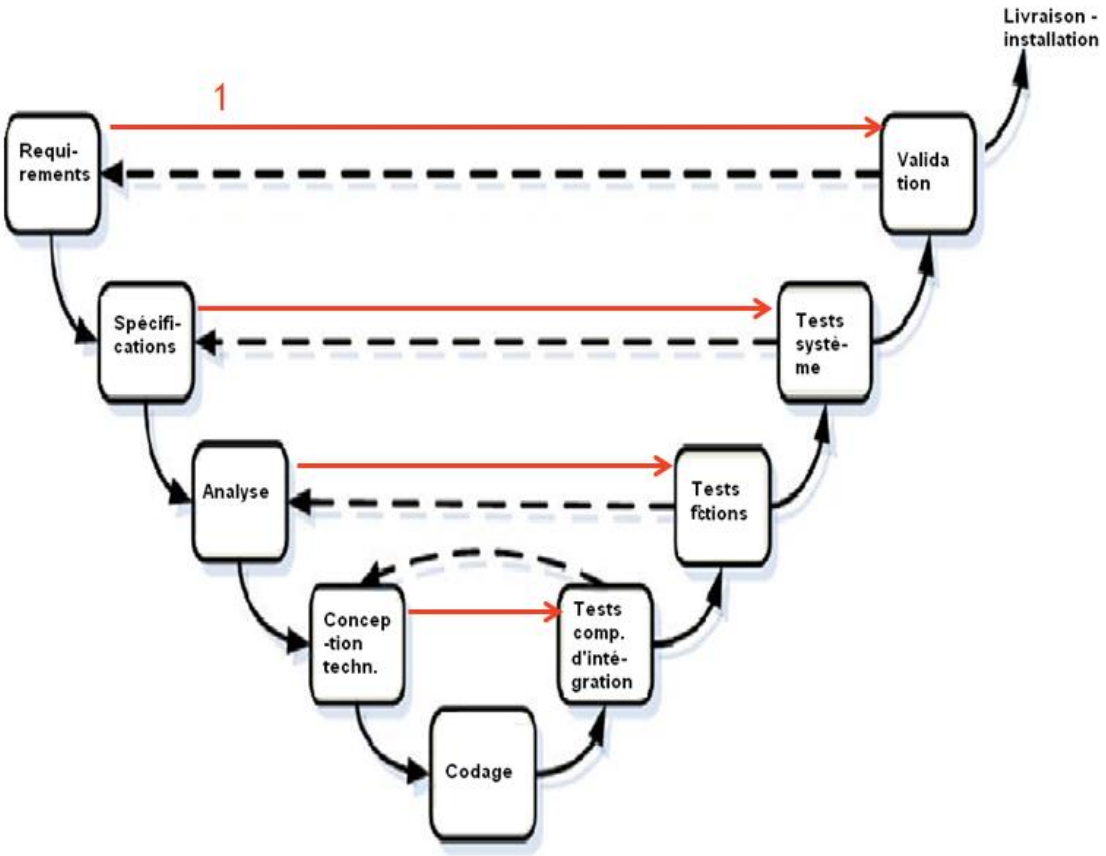
Points de synchronisation : « attente »
qu'une étape soit terminée.

Outputs bien définis = inputs de l'étape suivante/
Interaction avec le client.

Le plus courant : Modèle en V



V : Prévoir les tests



- **AVANT de les exécuter!**
- Toute description d'un composant est accompagnée de tests qui permettront de s'assurer qu'il correspond à sa description.

V : Pros & cons

- Toute étape descendante est **accompagnée de l'écriture des tests** qui permettront de s'assurer qu'un composant correspond à sa description.
- Cette méthode de travail rend explicite la préparation des dernières phases (validation-vérification) par les premières (construction du logiciel).
- Toute propriété du logiciel DOIT être vérifiable objectivement après la réalisation.

Remarque : ceci vaut pour toutes les méthodes qui décrivent les tests au début dès la construction du logiciel.

V : Pros & cons

- La planification est aisée.
- L'organisation est facilitée entre les différentes équipes.
- Les outputs sont clairement définis.
- Le client est associé, dès le début du projet pour écrire les tests de validation.

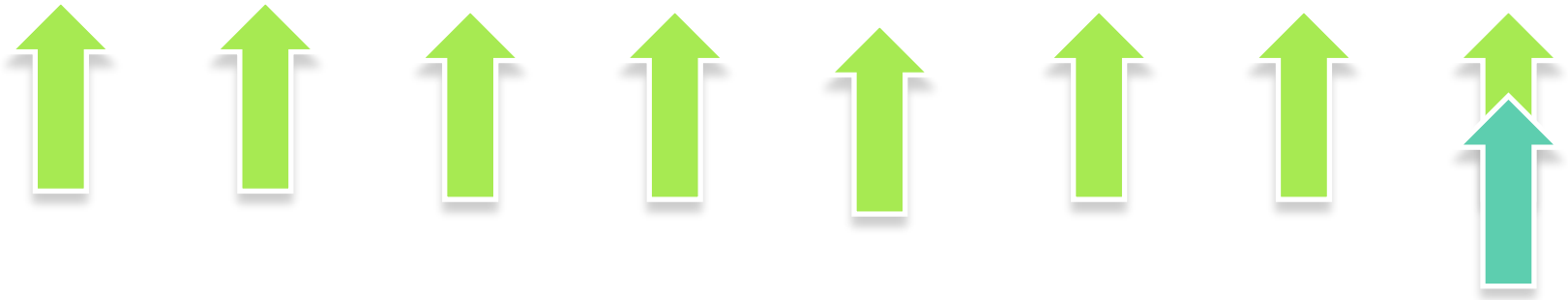
V : Pros & cons

- Le client n'intervient pas en cours de projet.
- Le client découvre souvent trop tard les erreurs.
- Les équipes de développement doivent être expertes pour évaluer le projet.

V : dans le temps

Quand tout se passe bien

t0	t0'	t1	t2	t3	t3'	t3''	t4	t5
Spécifications	Analyse	Conception	Implémentation					
Ecriture des tests système	Ecriture des tests fonctionnels	Ecriture des tests d'intégration	Ecriture et exécution tests unitaires	Exécution tests d'intégration	Exécution tests fonctionnels	Exécution tests système	Livraison	Tests de validation



Points de synchronisation : « attente »
qu'une étape soit terminée

Outputs bien définis = inputs de l'étape suivante

Interaction avec le client.

V : dans le temps

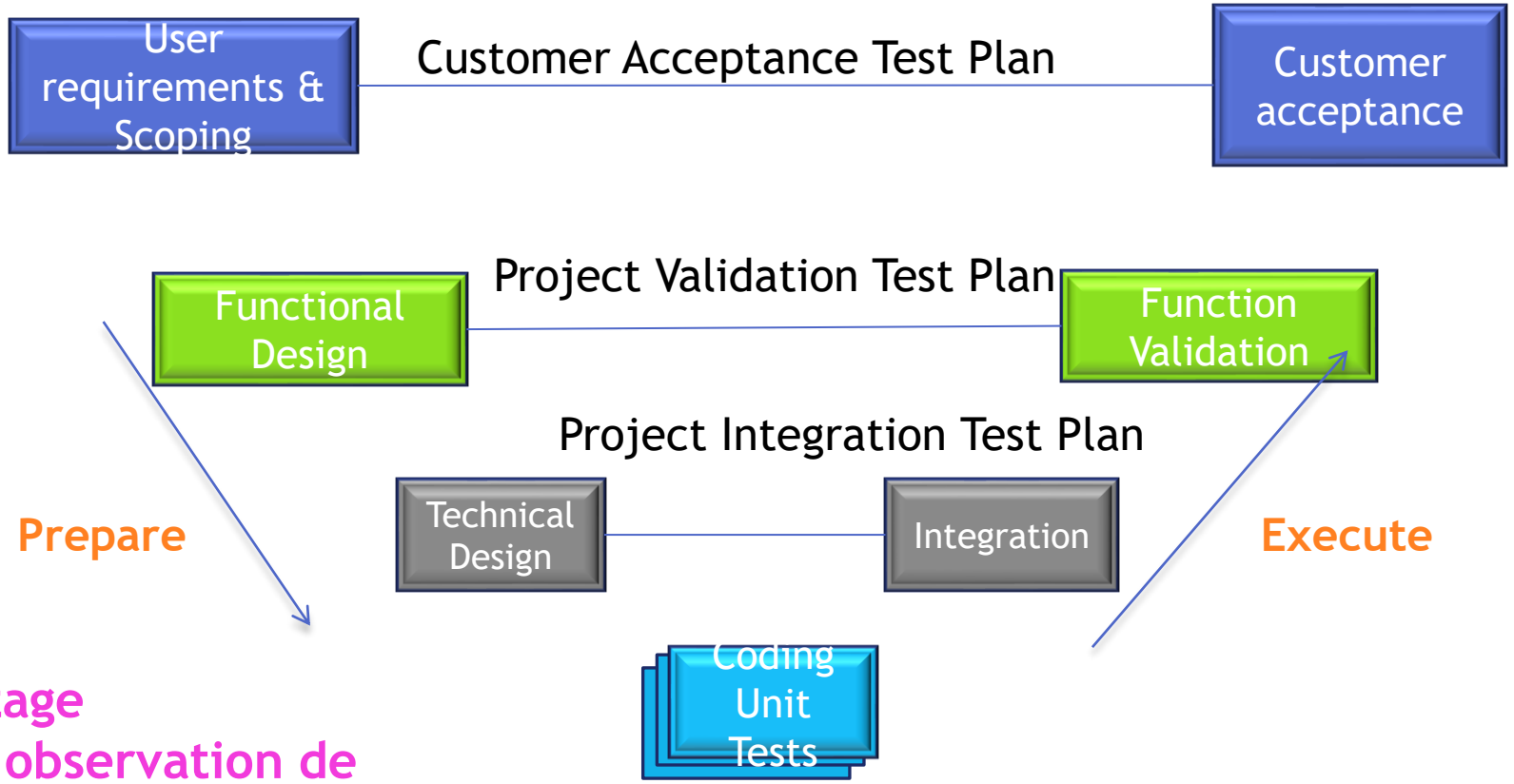
Quand tout se passe bien

t0	t0'	t1	t2	t3	t3'	t3''	t4	t5
Spécifications	Analyse	Conception	Implémentation					
Ecriture des tests système	Ecriture des tests fonctionnels	Ecriture des tests d'intégration	Ecriture et exécution tests unitaires	Exécution tests d'intégration	Exécution tests fonctionnels	Exécution tests système	Livraison	Tests de validation

Prévoir les erreurs (retour)

				Retour en conception - amélioration tests Implémentation - amélioration tests unitaires et exécution TU	Retour en analyse - amélioration tests Conception - amélioration tests d'intégration Implémentation - amélioration tests unitaires et exécution TU	Retour aux spécifications - amélioration tests systèmes Analyse - amélioration tests fonctionnels Conception - amélioration tests d'intégration Implémentation - amélioration tests unitaires et exécution TU		
				Exécution tests d'intégration	Exécution tests d'intégration & fonctionnels	Exécution tests d'intégration, fonctionnels & système	Livraison	Tests de validation

Illustration : test Methodology



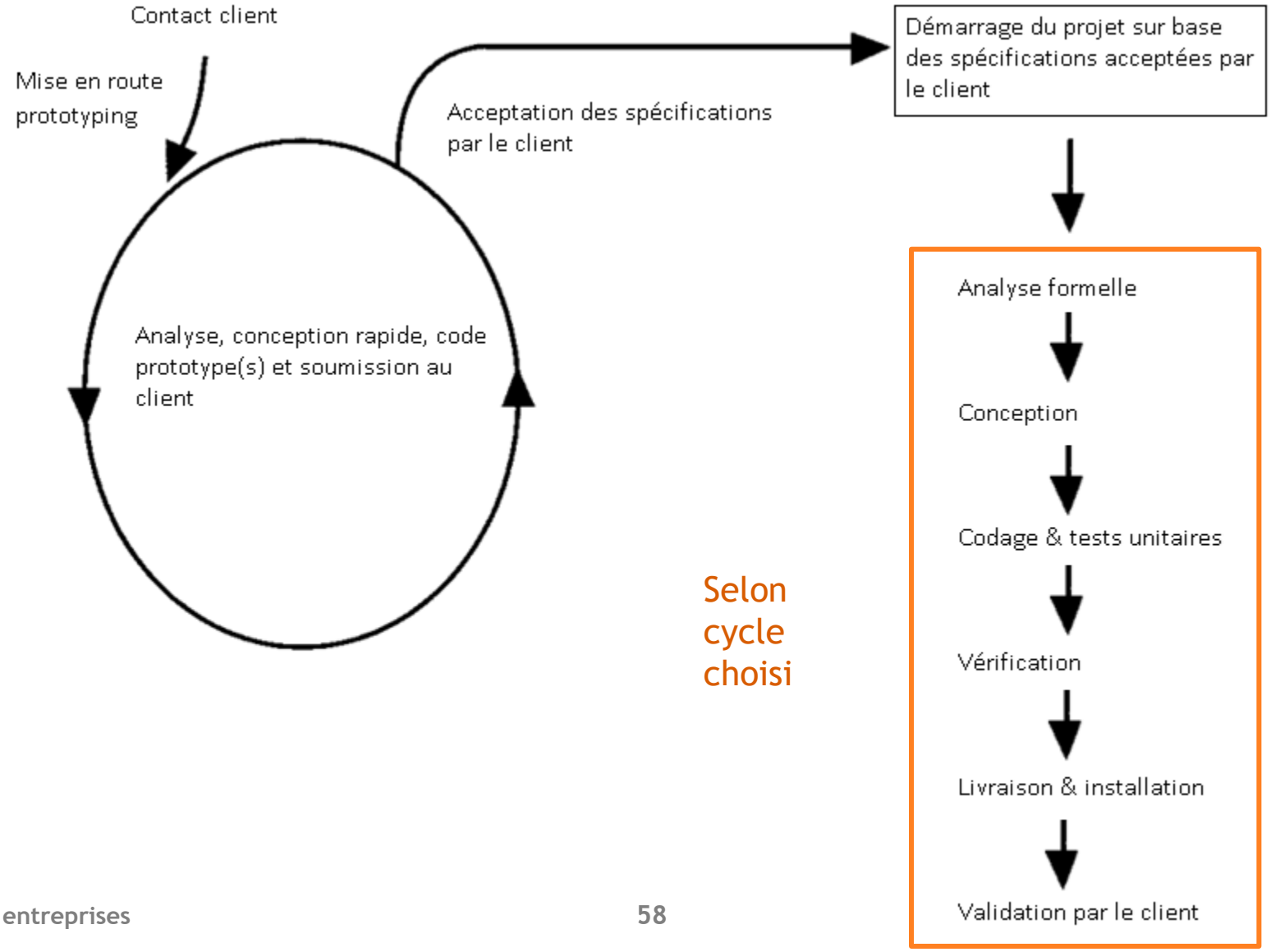
Stage d'observation de Alexandre Coste-Gandrey 2017

Modèle exploratoire

L'idée est de **combiner**

- **Le prototyping** (partie exploratoire).
  - Les spéc sont obtenues à partir de la maquette.
  - Objectif :
    - Spéc adéquates que le client accepte, le plus rapidement possible.
    - Meilleure adéquation entre les spéc et les besoins réels.
- **Et une autre méthode de développement** (Cascade, V, Y).
  - On jette le code écrit précédemment.
  - On utilise l'expérience du prototyping pour valider les spécifications et on démarre ensuite le SDLC selon la méthode choisie.

Modèle exploratoire



Selon cycle choisi

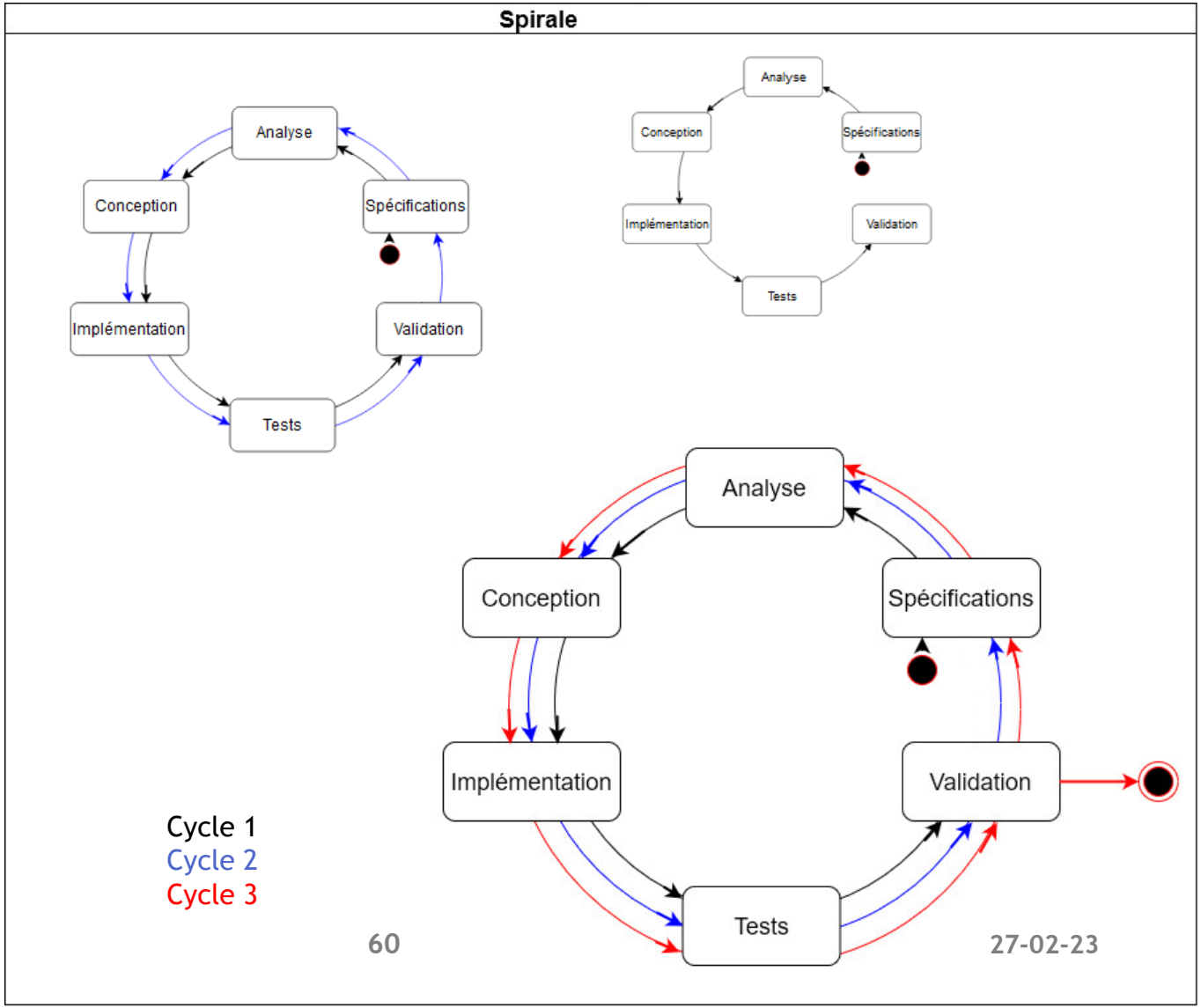
Modèle exploratoire: pros & cons

- Les spécifications sont bien comprises par toutes les parties (client, chef de projet, développeurs, testeurs…).

- Le choix du cycle de développement après acceptation des spécifications conditionne les autres avantages et inconvénients de ce modèle.

Modèle en Spirale

- Ex:
  projet découpé en 3 cycles.
- Chaque cycle contient les 6 phases.
- Après chaque cycle, livraison au client.



Spirale : Pros & cons

- On ne doit pas avoir analysé l'entièreté de la demande du client dans les détails, on le fera pour une partie du développement à chaque cycle.
- On peut profiter de l'expérience du développement des cycles précédents pour **évaluer** les alternatives offertes pour le développement du nouveau cycle.
- Chaque cycle se termine par une version livrable.
- Le client peut valider le logiciel à chaque cycle.
- Le temps de développement d'un cycle est plus court et les risques mieux gérés à l'intérieur du cycle.

Spirale : Pros & cons

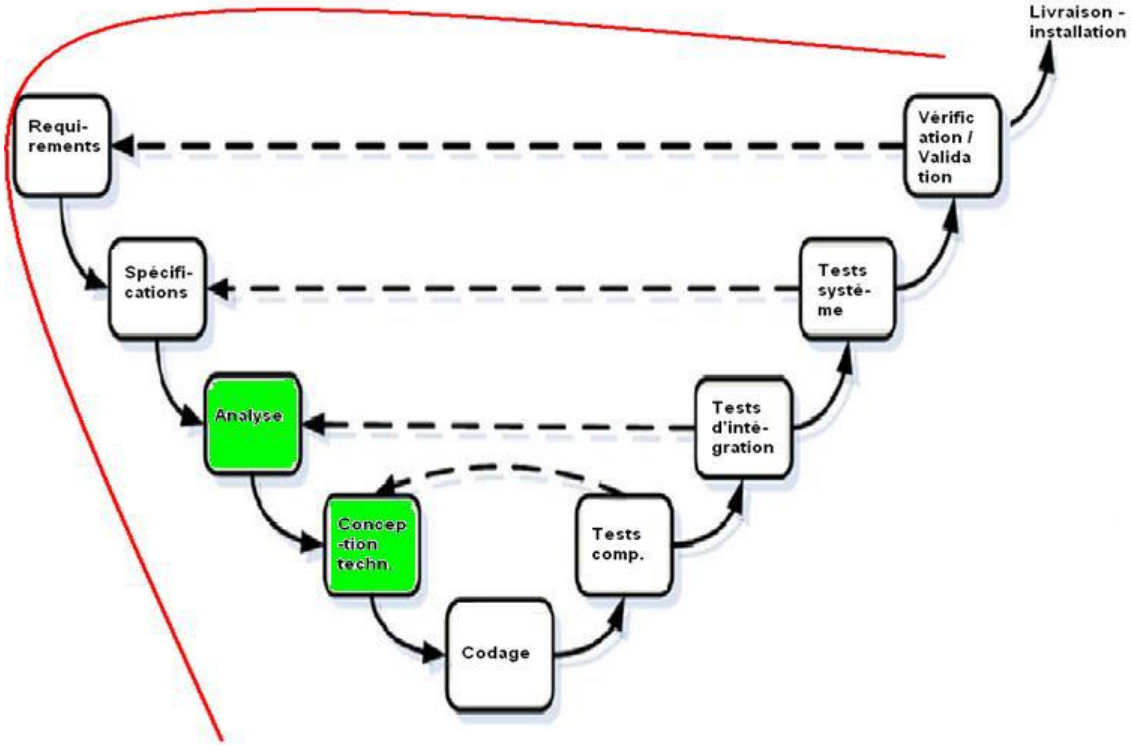
- La gestion de la découpe des spécifications est complexe, il faut tenir compte des :
  - Interactions entre les fonctionnalités.
  - Difficultés rencontrées pendant les cycles précédents.
  - Objectifs et contraintes du cycle courant (C).
  - Alternatives de réalisation des objectifs du cycle C.

  ➔ la décision du développement à faire pendant un cycle doit tenir compte du risque.
- La planification est délicate.
- La conception peut être revue à cause des nouvelles spécifications.

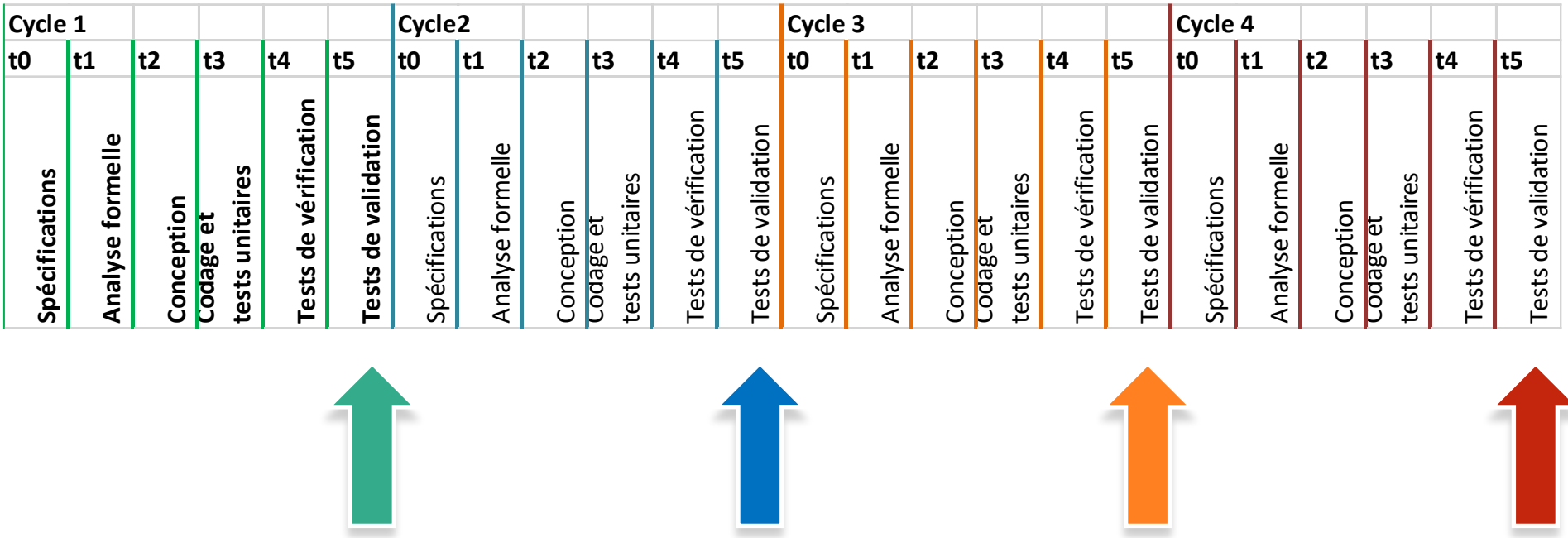
Spirale : Pros & cons

- Chaque cycle de la spirale peut être développé selon n'importe quel autre procédé.

Chaque cycle de la spirale reprend le modèle en V



Spirale : dans le temps



Validation par le client

Questions?

ProjetAE

Y a-t-il un cycle de vie appelé « classique » qui décrive ce que vous avez fait en ProjetAE ?

- Cascade
- Prototyping
- Y
- V
- Spirale ?