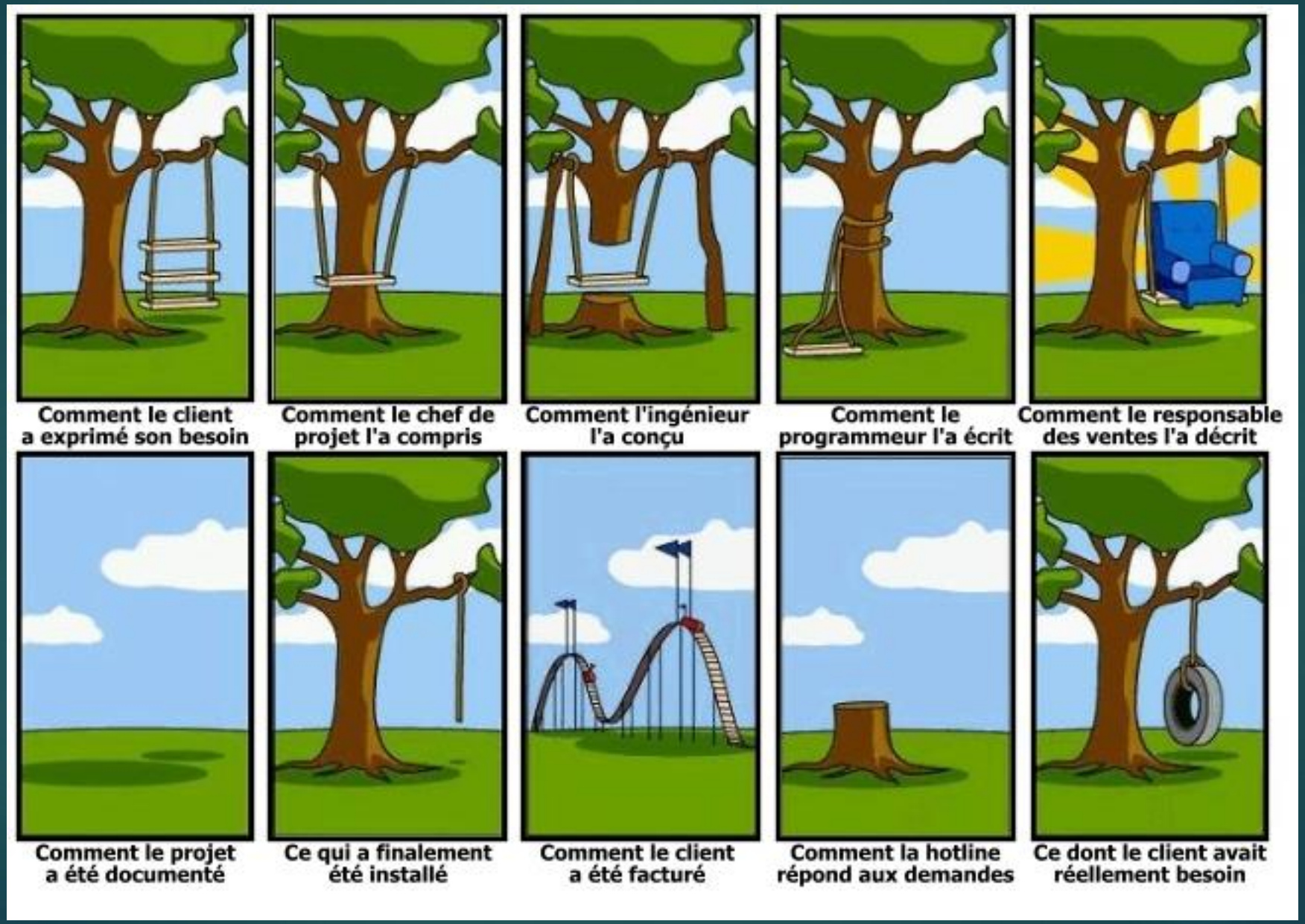
Introduction à la gestion de projet pour le développement logiciel

GUIDE À L'USAGE DES PARTICIPANTS À LA COMPÉTITION MEET-U (2017-2018)

INTRODUCTION



Source : http://www.projectcartoon.com/

INTRODUCTION : Dans le monde du génie logiciel…

- **Logiciel Louvois** : Il a empêché le versement de la solde de dizaine de milliers de militaires durant deux ans, avant d'être abandonné.

460 millions €

- **Vol 655 Iran Air** : Des civils ont été abattus à cause de programmes radar qui n'ont pas fait la différence entre un Airbus iranien (vol 655 d'Iran Air) transportant ces civils et un avion militaire…

290 civils, dont 66 enfants

- **Mission Vénus** : passage à 5 000 000 de Km de la planète, au lieu de 5 000 Km prévus... Cause : remplacement d'une virgule par un point (au format US des nombres).

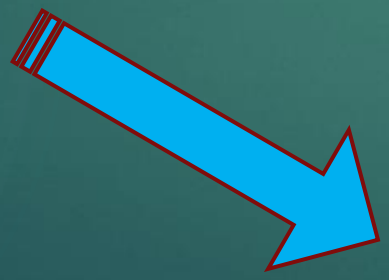
Sources : Le génie logiciel (GL) sur openclassrooms.com, http://www.tisserant.org/cours/qualite-logiciel/qualite_logiciel.html

INTRODUCTION : Dans le monde du génie logiciel…

Etude menée en **1995** sur 8380 projets révèle que :

- 16% des applications sont construites avec succès
- 53% des projets aboutissent mais posent des problèmes (diminution des fonctionnalités fixées, le non-respect des délais spécifiés ou encore l'augmentation des coûts)
- 31% des projets sont purement et simplement abandonnés.

Aux USA en 1993, le projet moyen dépasse son budget de 100 % et est en retard de 1 an !
(source : Yourdon, «CASE and silver bullet of software engineering»)



MODERN RESOLUTION FOR ALL PROJECTS

	2011	2012	2013	2014	2015
SUCCESSFUL	29%	27%	31%	28%	29%
CHALLENGED	49%	56%	50%	55%	52%
FAILED	22%	17%	19%	17%	19%

Source : Standish Group 2015 Chaos Report

PHASES D'UN PROJET : GÉNÉRALITÉS

Objectif : Diviser le projet en différentes étapes ce qui permet d'améliorer (voire de garantir) la **qualité** du produit et le **respect des besoins** des utilisateurs (= objectif) tout en respectant **les délais et les coûts fixés** au départ.

Un projet commence par être découpé en 3 grandes principales phases :

Phase préliminaire

Etablir vos objectifs, définir les rôles de chacun, faire une recherche pour contextualiser son projet et observer ce qui se fait déjà, élaborer un planning d'actions ainsi qu'un budget. Chaque membre de l'équipe doit être en accord avec tous ces éléments.

Phase de réalisation

Mise en œuvre du projet. C'est à ce moment qu'intervient la communication du projet. Veillez à toujours travailler en consultation avec tous les membres de l'équipe projet par des réunions où chacun évoque l'avancée de son action, ses objectifs pour la prochaine réunion, ses difficultés, etc.

Phase de finalisation

Mise en production, c'est-à-dire s'assurer que votre solution est conforme aux attentes.

Temps optimal à consacrer :

PHASES D'UN PROJET : GÉNÉRALITÉS

Objectif : Diviser le projet en différentes étapes ce qui permet d'améliorer (voire de garantir) la **qualité** du produit et le **respect des besoins** des utilisateurs (= objectif) tout en respectant **les délais et les coûts fixés** au départ.

Un projet commence par être découpé en 3 grandes principales phases :

Phase préliminaire

Phase de réalisation

Phase de finalisation

En pratique :

Dans la mauvaise direction…

Dans l'urgence (et mal)

PHASES D'UN PROJET : DÉCOMPOSITION EN ÉTAPES

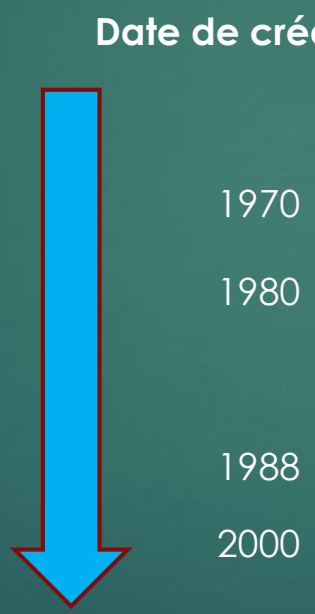
- **Phase préliminaire** :
  - Expression des besoins : Décrire les besoins des utilisateurs finaux
  - Analyse fonctionnelle : Décrire les actions que le système devra pouvoir effectuer
  - (Etude de faisabilité, analyse des risques, analyse de la valeur, …)
- **Phase de réalisation** :
  - Conception : Elaboration de l'architecture du système
  - Implémentation (i.e. programmation)
- **Phase de finalisation** :
  - Tests (unitaire puis d'intégration) et validations
  - Mise en production/maintenance : corrections et stabilisation du système avant livraison

CYCLE DE DÉVELOPPEMENT LOGICIEL

Question : Comment organiser toutes ces étapes en cycle efficace ?

Quelques exemples :

- Linéaire
  - Modèle en cascade
  - Modèle en V
- Modèle itératif
  - Modèle en spirale
- Méthodes AGILE



Remarque : On ne parlera pas dans cette présentation des méthodes AGILE, qui sont bien plus qu'un modèle de cycle de développement (certains diront que ce sont même des philosophies !). Sachez toutefois que si vous comprenez tout ce qui est présent ici, vous aurez de bonnes bases pour les appréhender dans votre avenir professionnel.

CYCLE DE DÉVELOPPEMENT LOGICIEL : CASCADE

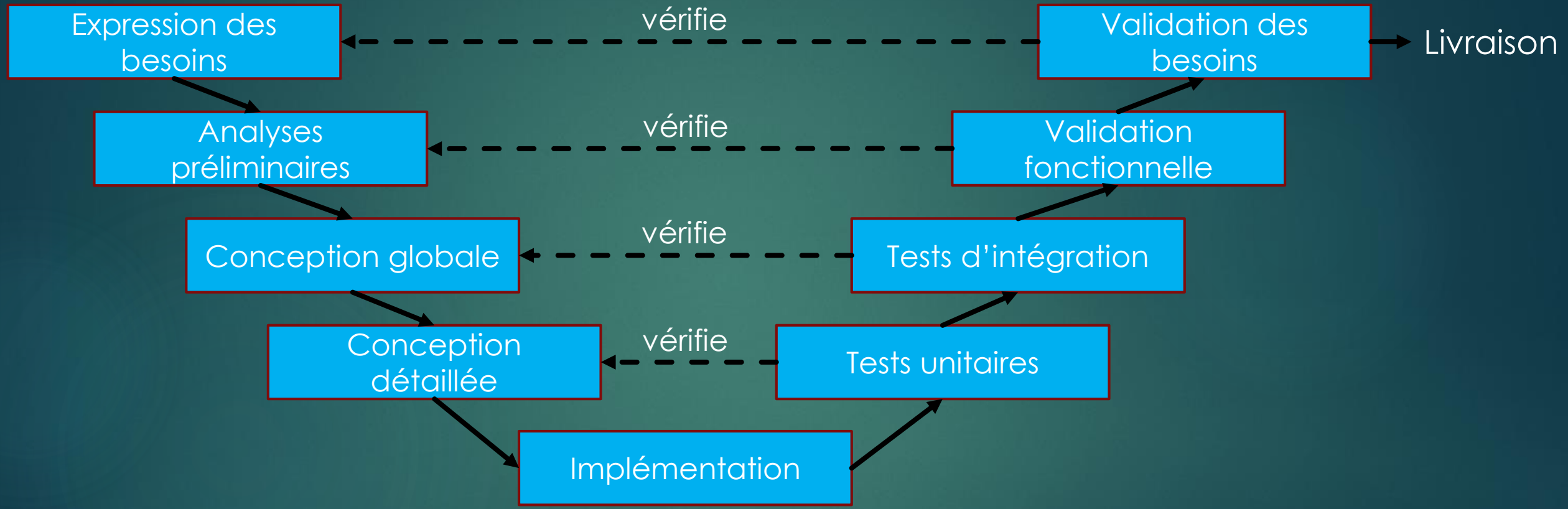
Présenté par Winston W. Royce en 1970, le modèle en cascade originel est hérité du BTP. Il se base sur 2 idées fondamentales :

- Une étape ne peut pas être débutée avant que la précédente ne soit achevée : inutile de monter les murs tant que les fondations ne sont pas coulées
- La modification d'une étape du projet a un impact important sur les étapes suivantes

Inconvénients : nombreux ! (faible réactivité, répercussion importante des erreurs, …). Mais c'est intuitivement le 1er modèle.

Pour aller plus loin :
Lire « Il est temps de rompre avec le cycle en cascade »

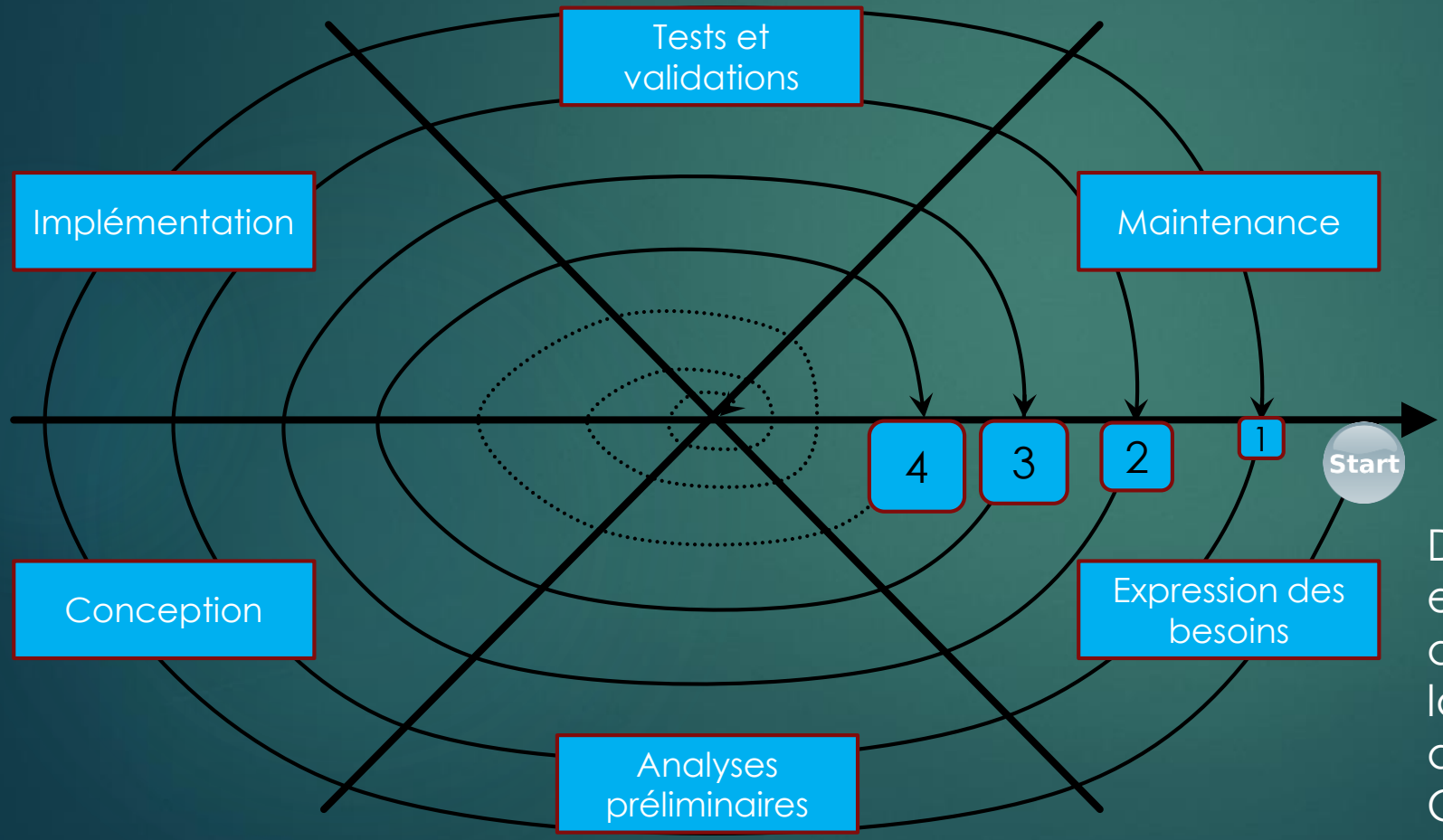
CYCLE DE DÉVELOPPEMENT LOGICIEL : V

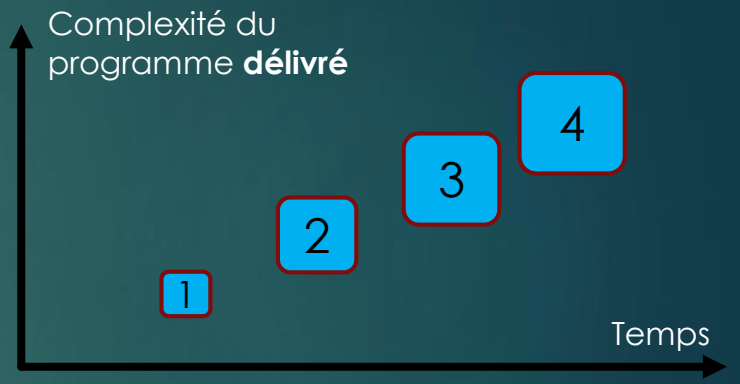


L'industrie informatique a adopté le cycle en V dans les années 80. Il améliore le modèle en cascade en permettant la conception des tests et des validations en amont, ainsi qu'en augmentant la réactivité en cas d'anomalie (i.e. organisation 2 à 2 plus cohérente qui limite le retour sur trop d'étapes précédentes).

Inconvénient : Il est difficile en pratique de spécifier toutes les caractéristiques d'un système en avance !

CYCLE DE DÉVELOPPEMENT LOGICIEL : MODÈLE EN SPIRALE



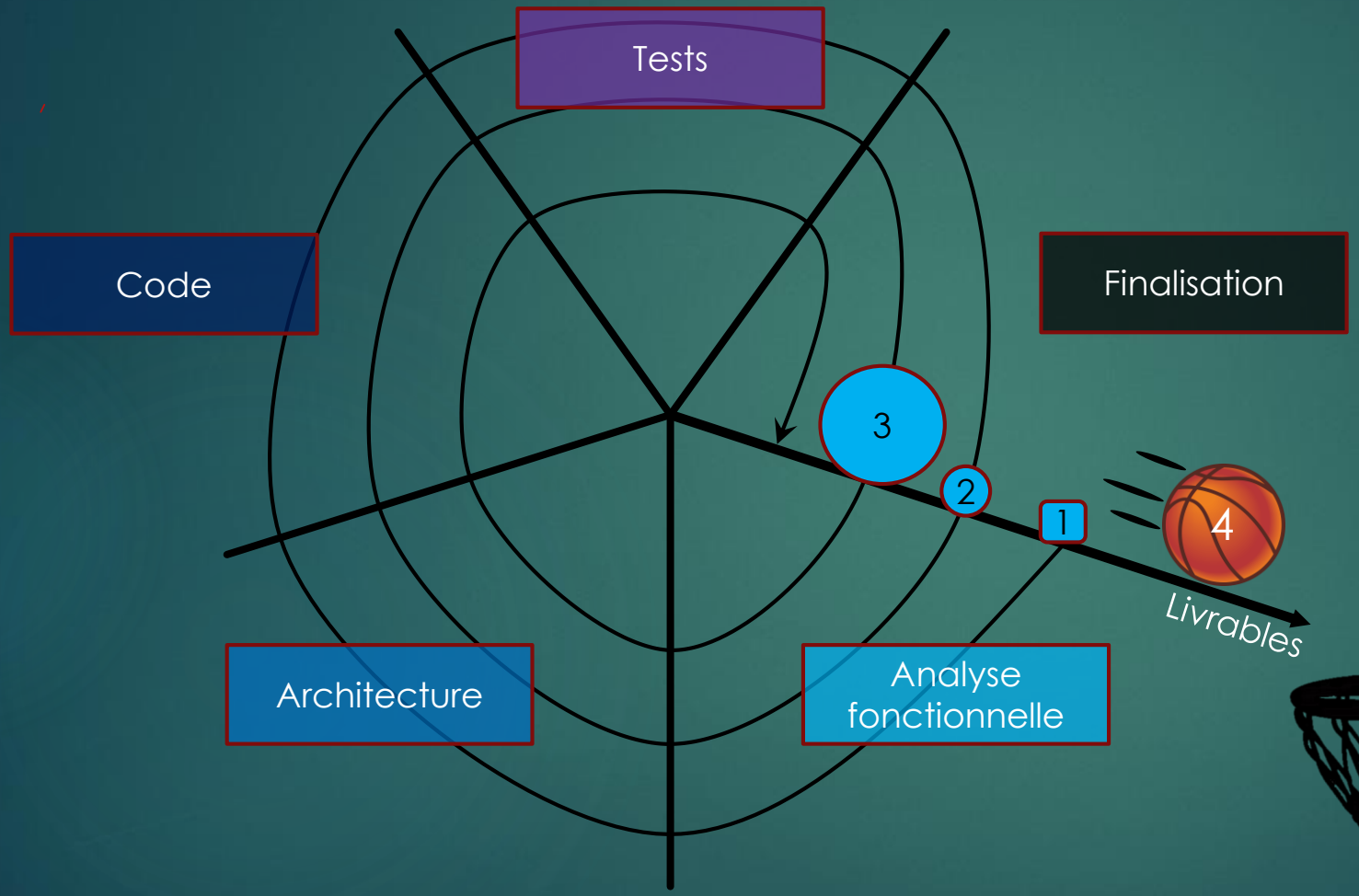


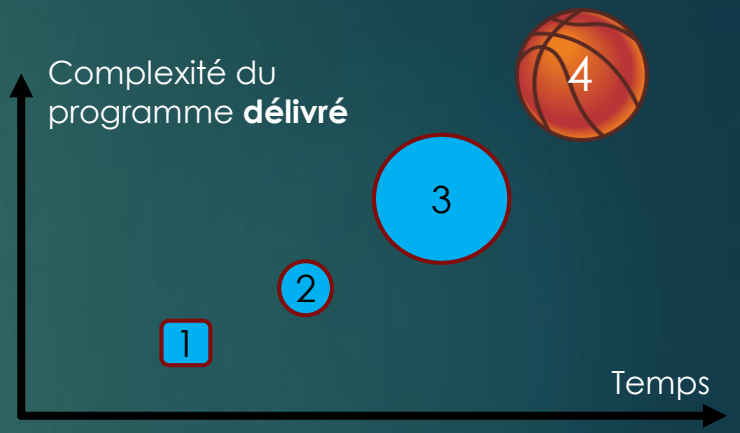
Défini par Barry Boehm en 1988, le cycle en spirale prévoit l'implémentation de versions successives. Il prévoit donc la livraison de prototypes, c'est à dire de versions incomplètes du produit. Cela a l'avantage de rendre le projet très adaptable.

Et maintenant…

C’est très bien tout ça, mais concrètement ???

MÉTHODE PROPOSÉE :

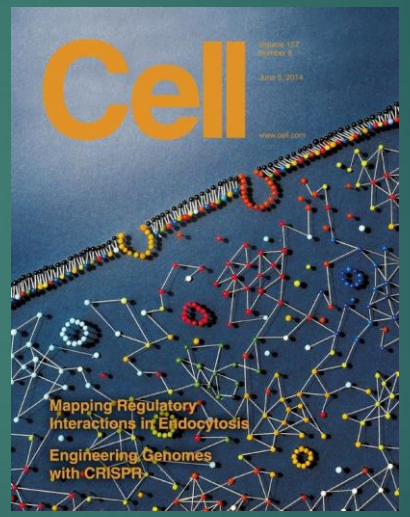




MÉTHODE PROPOSÉE :
1) Analyse fonctionnelle

Comment savoir quoi faire, où commencer ???

Aller fouiller la littérature du domaine (+ vos connaissances + interventions et documents MEET-U !)



NCBI

PubMed.gov

US National Library of Medicine
National Institutes of Health

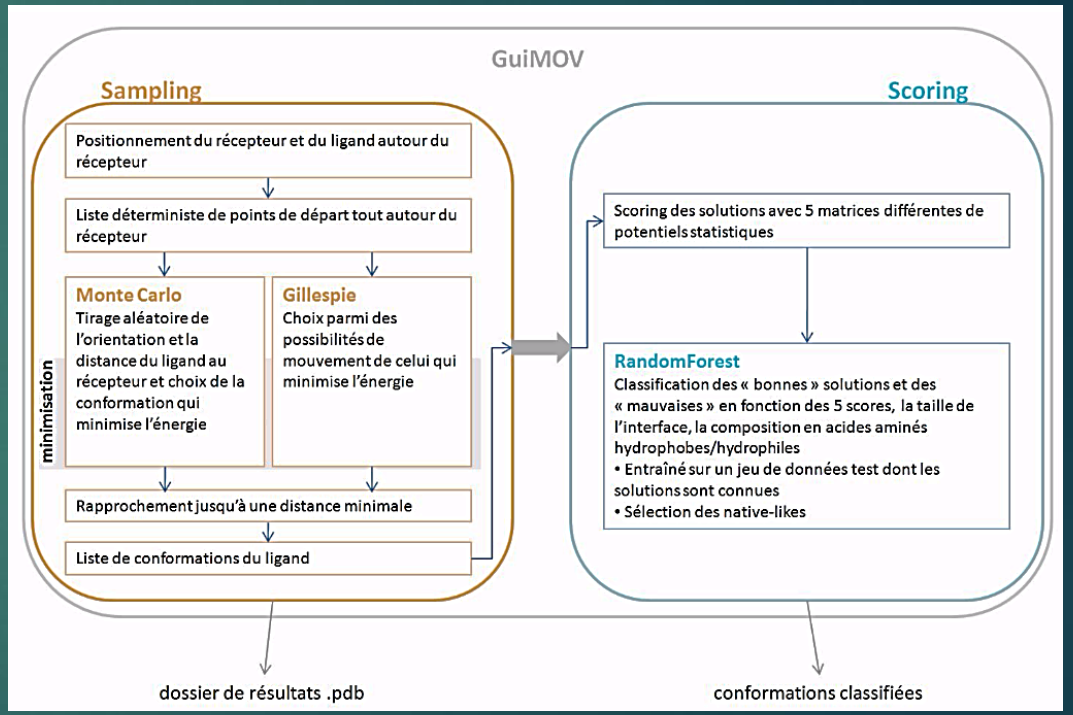
MÉTHODE PROPOSÉE : 1) Analyse fonctionnelle

Définition (Wikipedia) :

« Un diagramme est une représentation visuelle **simplifiée et structurée** des concepts, des idées, […] etc. employé dans tous les aspects des activités humaines pour visualiser et éclaircir la matière. Un diagramme permet aussi de décrire des phénomènes, […] ou de représenter des parties d'un ensemble. »

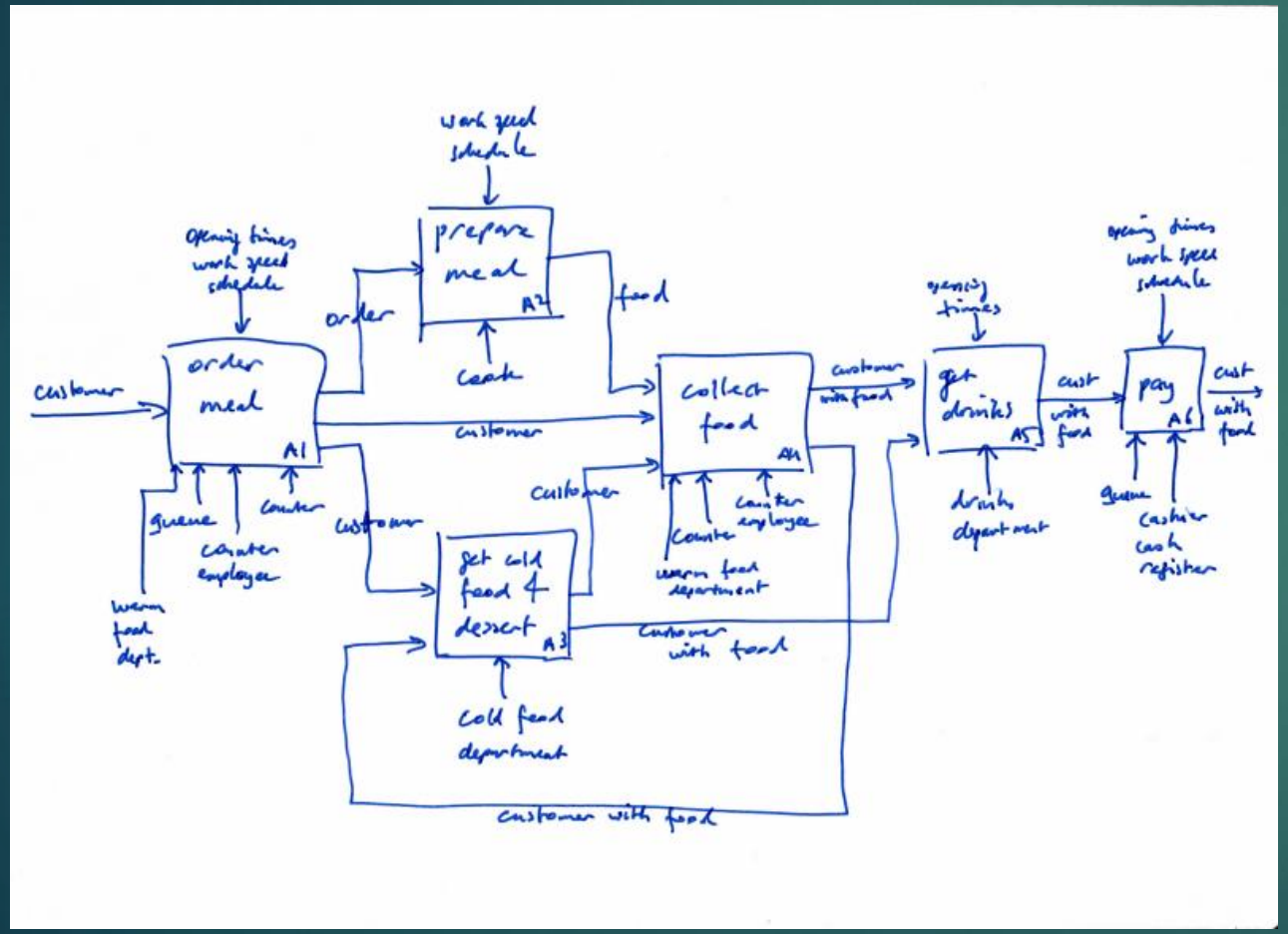
Les avantages principaux sont :

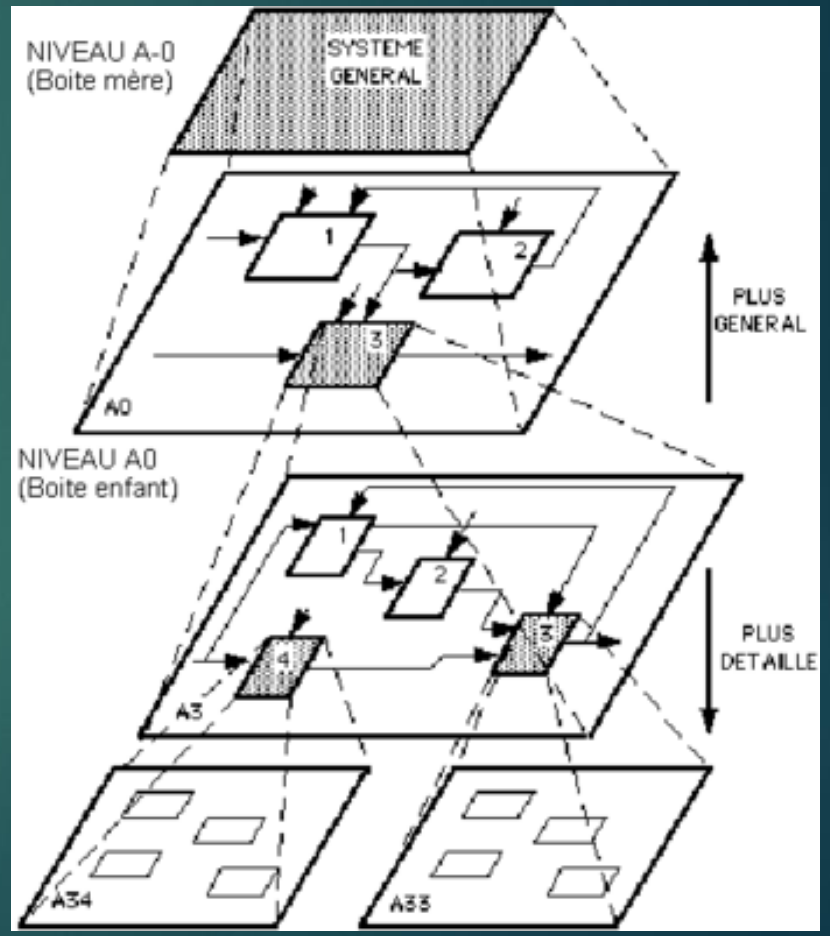
- Simplicité, tout en portant beaucoup d'information
- Structuré, ce qui permet de bien conserver l'information (mieux que de le faire « dans sa tête »)
- Facilement communicable et compréhensible, car visuel (et relativement simple)



Docking rigide protéine-protéine (Meet-U 2016-2017)

MÉTHODE PROPOSÉE :
1) Analyse fonctionnelle

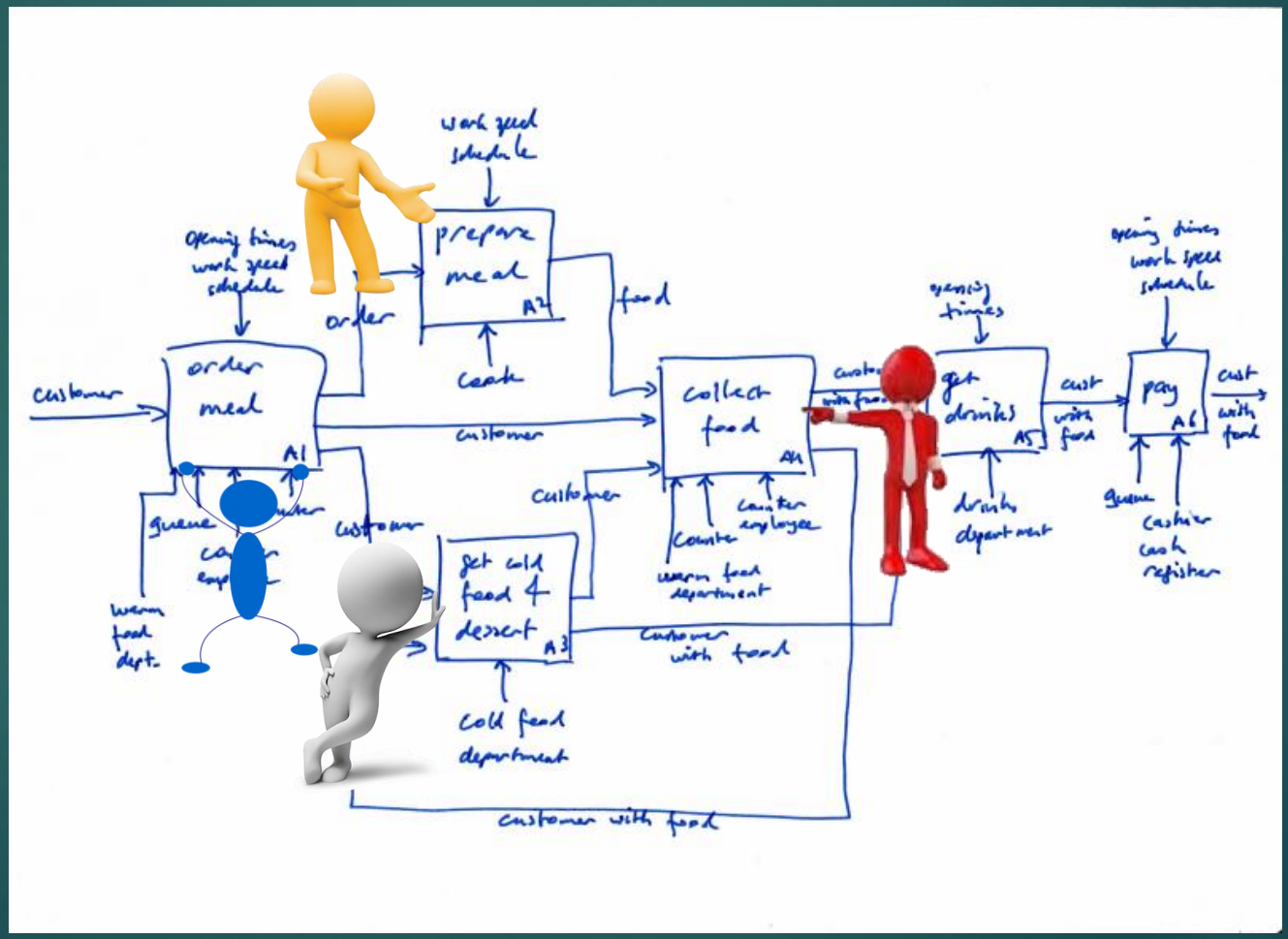




Conseil de plateforme collaborative de dessin : https://www.draw.io/

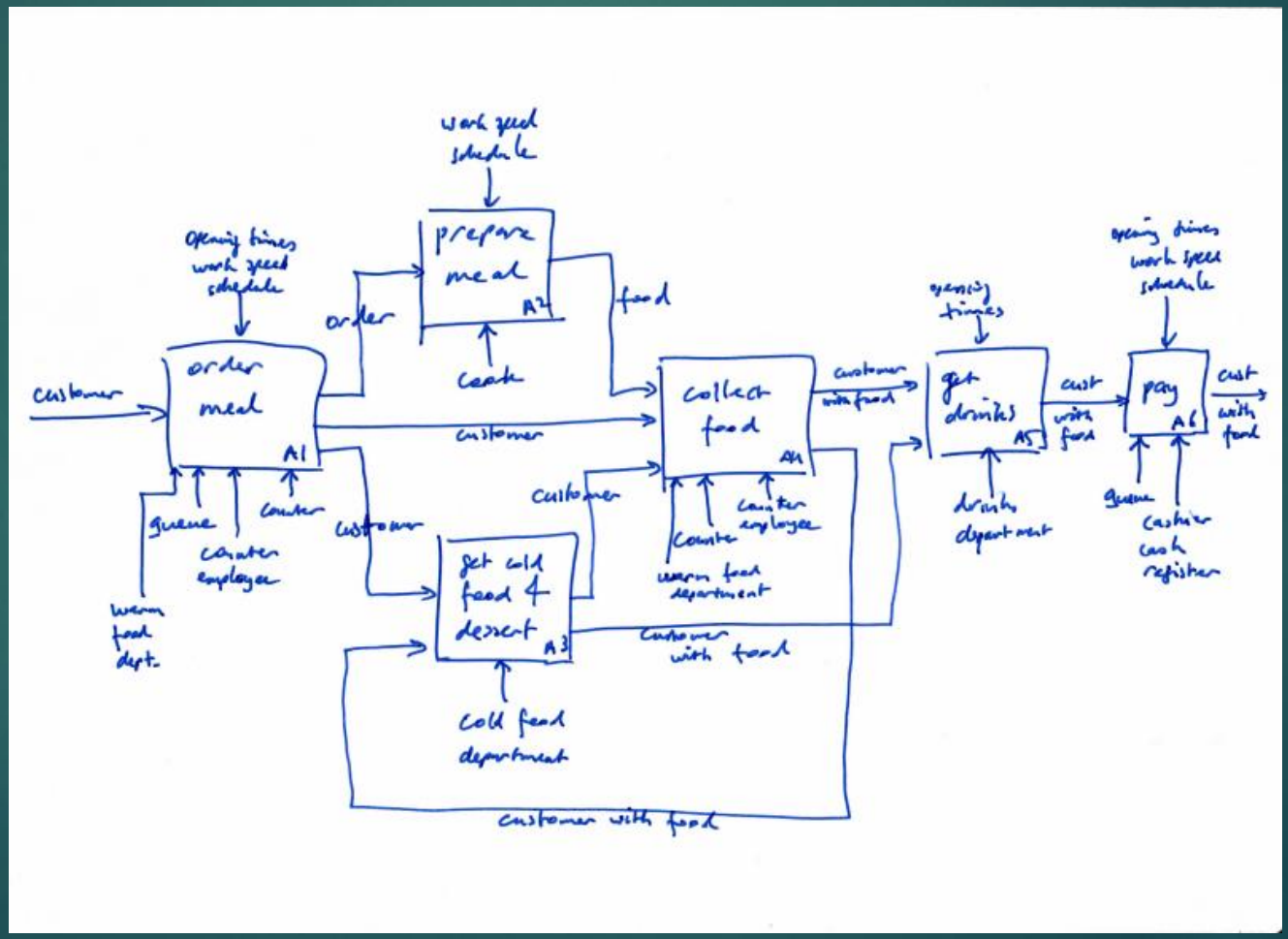
MÉTHODE PROPOSÉE :
1) Analyse fonctionnelle

Maintenant que vous avez votre plan, vous pouvez vous répartir les tâches :

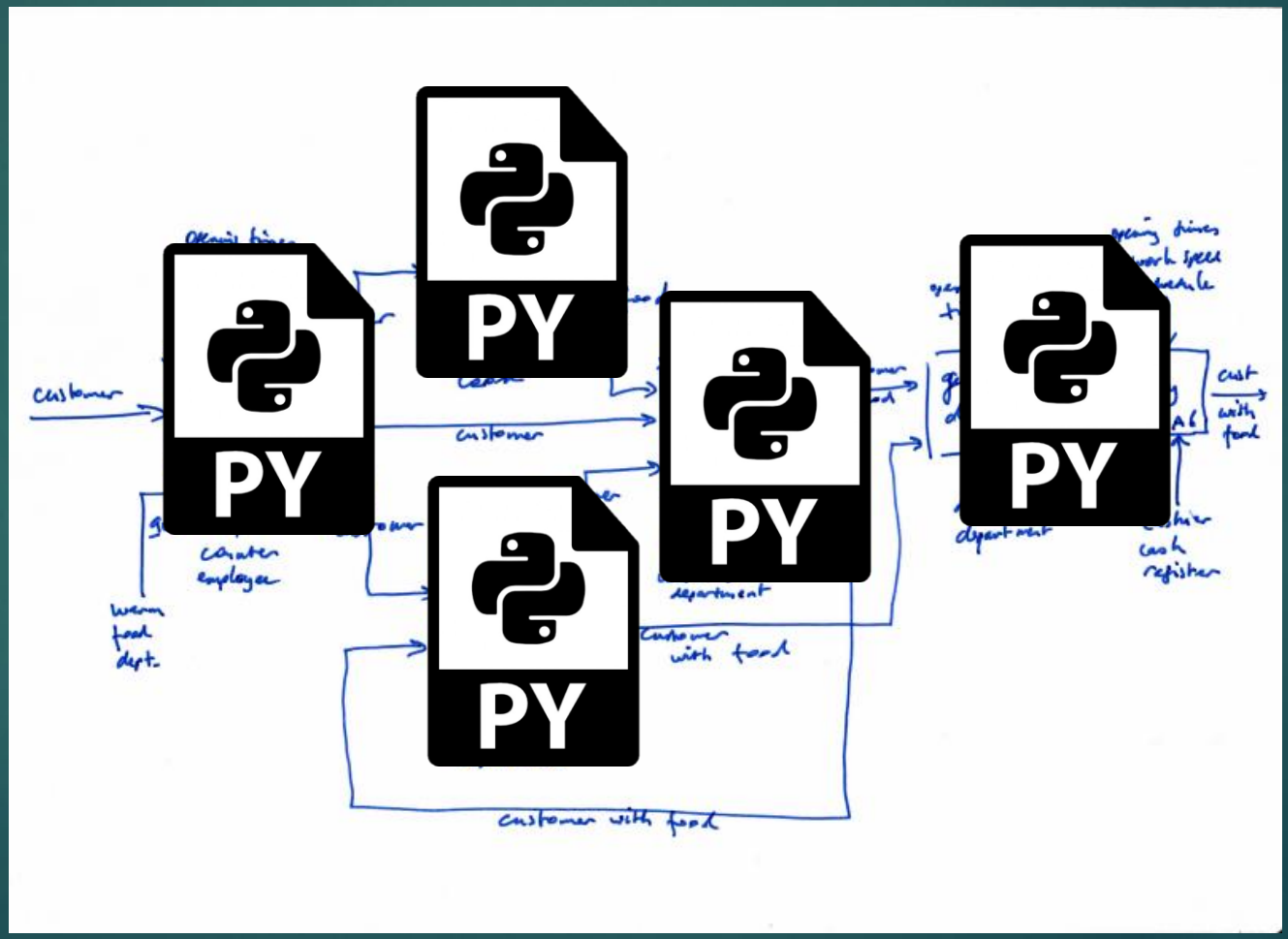


MÉTHODE PROPOSÉE : 2) Architecture

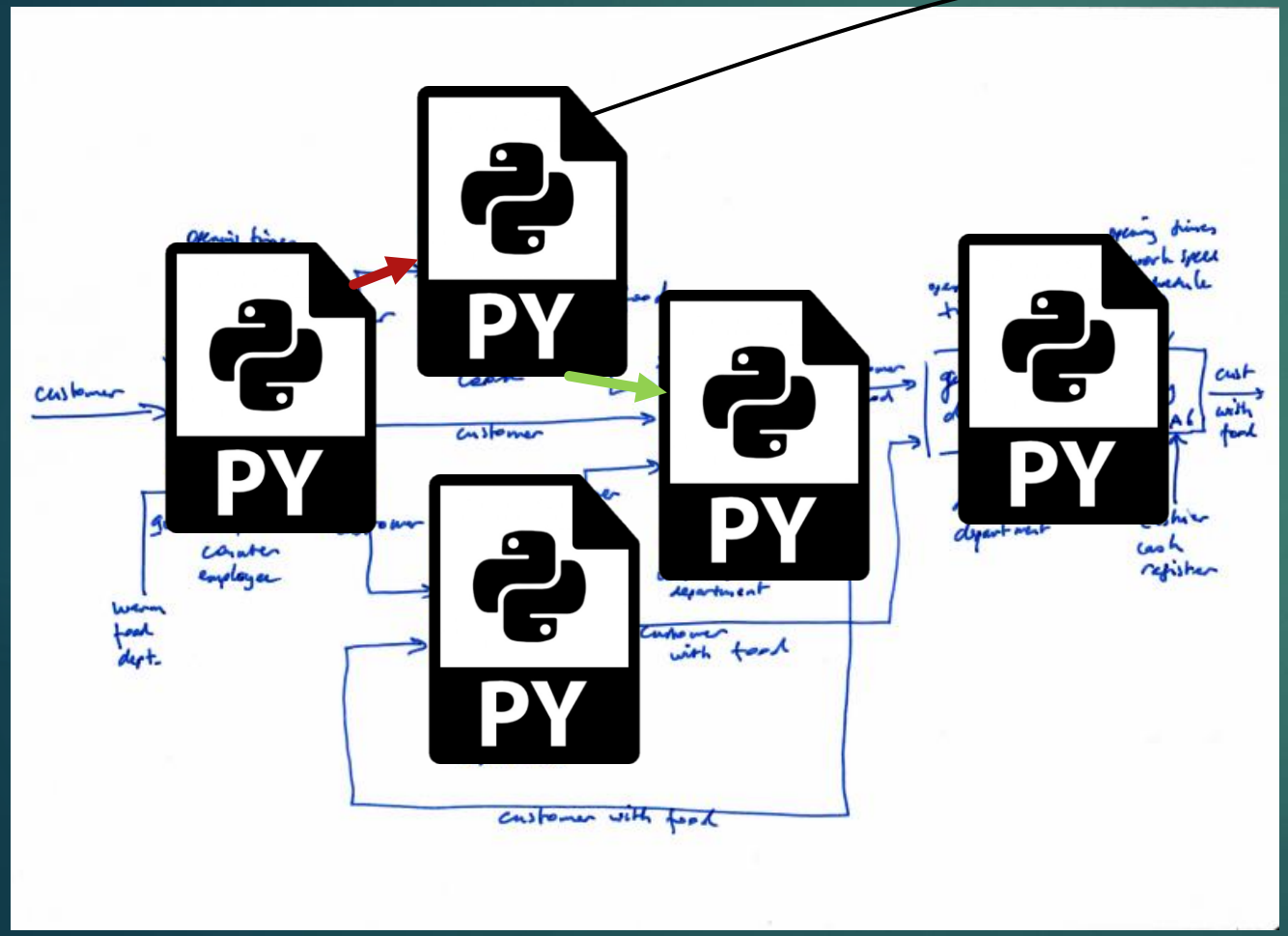
Étape suivante ?



MÉTHODE PROPOSÉE :
2) Architecture



MÉTHODE PROPOSÉE : 2) Architecture




```
!/usr/bin/env python
# coding: utf-8

"""
Author: Arnaud Ferré
Description: Just a file
Licence: CC-BY
"""

# Internal imports
import otherFunction from MyProject.MyOtherFile

# External imports
import sys, numpy, math, random

########################################################
# Functions:
def function1():
    varResult = None
    # ToDo (will call otherFunction)
    return varResult

def mainFunction(var):
    varResult = dict()
    # ToDo
    return varResult

########################################################
# Test and execution area:
if __name__ == '__main__':
    var1 = None
    print(mainFunction(var1))
```

MÉTHODE PROPOSÉE : 3) Code

- Logiciel de gestion de versions : « […] permet de stocker un ensemble de fichiers en conservant la chronologie de toutes les modifications qui ont été effectuées dessus. »
- Intérêts principaux :
  - Sécuriser la sauvegarde des fichiers
  - Mutualiser le développement
  - Retour arrière possible dans le développement
- Conseillé/populaire et gratuit : **Git**
- Un cran plus loin : Utiliser le service web GitHub
  - Interface visuelle pour utiliser Git
  - Plein de fonctionnalités en plus (wiki, bug tracking, **forum**, graphes de suivi, …)
  - User-friendly
  - Avec compte gratuit, le code doit être opensource (attention aux licences des scripts externes utilisés !)

MÉTHODE PROPOSÉE : 3) Code

- Codage :
  - **Faites un code compréhensible et propre !!!** (indentation, nom de variable / fonction / classe informatif, modulaire, commenté – mais pas trop !, aéré, …)
  - En début de script, mettez une zone de commentaire qui, au minimum, décrit ce qu'il contient, et identifie ses auteurs principaux.

- Gestionnaire de version :
  - Toujours commencer une session de travail par un « *git pull* » (= récupérer le code sur le serveur central)
  - Toujours vérifier que son *commit* (= remonter son code sur le serveur) ne rend pas non-fonctionnel le programme entier (compilation, warning, erreur, etc.). Le mieux étant de le tester avant…
  - Toujours mettre un commentaire **informatif** lors d'un *commit* !
  - (Evitez d'écrire sur le même fichier en même temps)

MÉTHODE PROPOSÉE : 4) Tests (unitaires et d'intégration)

- **Test unitaire** : Permet de vérifier le bon fonctionnement d'une partie précise d'un programme (souvent une fonction/méthode)
  - Idée minimaliste : Programmer des scénarios d'exécution d'une fonction/méthode pour différentes valeurs appropriées en entrée et pour lesquelles on attend une sortie précise
  - Il existe des logiciels ou des méthodes spécifiques à un langage/IDE (Xunit, Junit, ..., programmation par contrat, assertions, …) pour programmer plus facilement des tests unitaires (plutôt que de les programmer soi-même) -> attention à la courbe d'apprentissage…

- Test d'intégration : Permet de vérifier le bon fonctionnement de l'ensemble du programme
  - Idée minimaliste : idem que test unitaire mais pour l'ensemble du programme
  - Conseil : Créer des jeux de données simplistes à tester (donc rapide à traiter) + des méthodes de visualisations de vos données à chaque étape importante
  - A effectuer au minimum à chaque itération du cycle de développement (optimal : avant chaque *commit*, mais difficile si long temps de calcul…)

MÉTHODE PROPOSÉE :
4) Tests (unitaires et d'intégration)

- Une astuce pour le Python :

```
if __name__ == '__main__':
```

Tout ce qui se trouve dans ce *if* ne sera exécuté que lors de l'exécution du script qui le contient directement.

En conséquence, il ne sera pas exécuté lors d'une importation par exemple.

Très pratique pour faire des tests unitaires et les garder à la fin d'un script sans diminuer la performance du système !

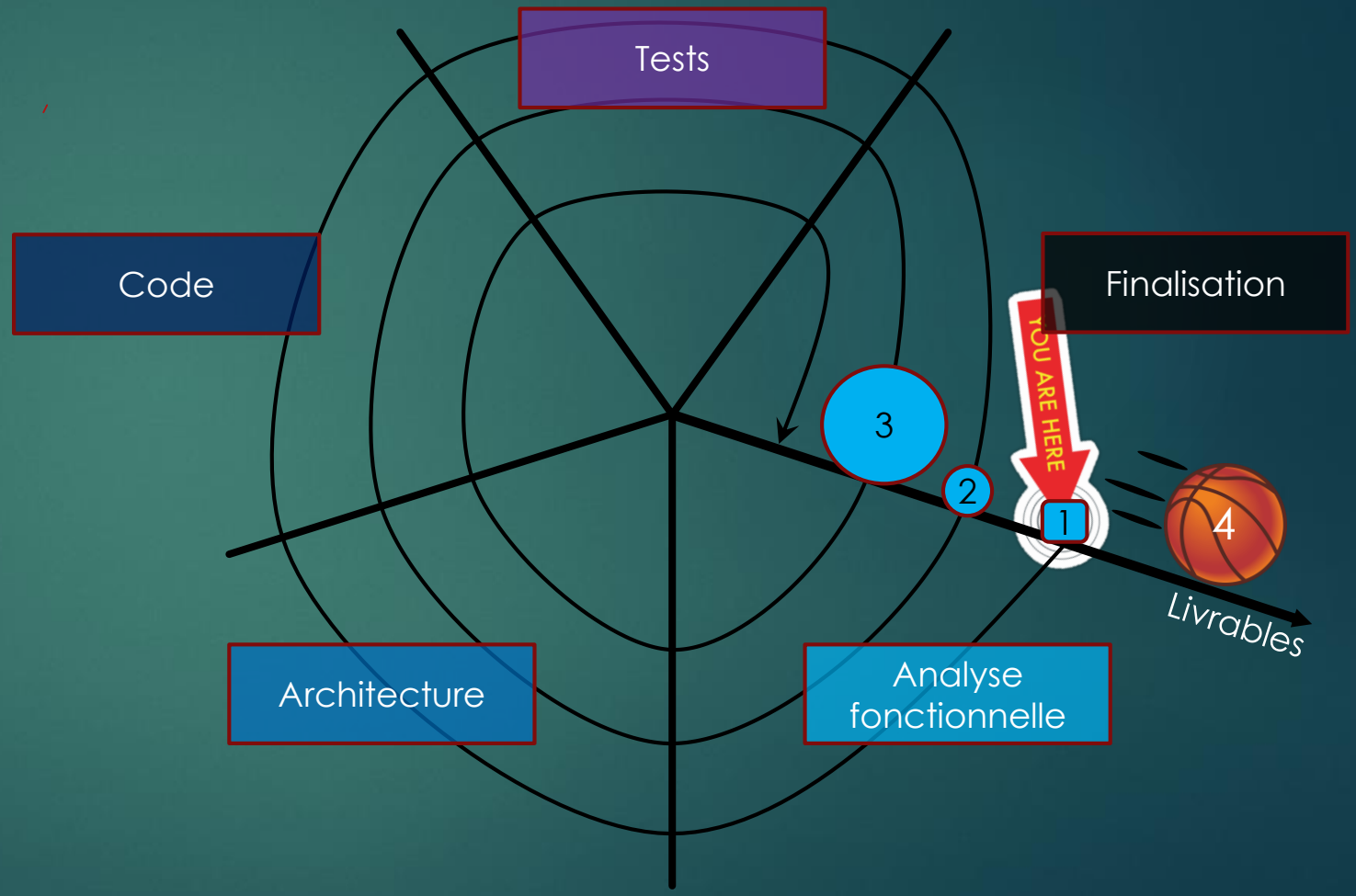
Conseil : Chacun de vos scripts de classes ou de fonctions devraient contenir une zone de test de ce genre !

MÉTHODE PROPOSÉE : 5) Finalisations

- Tests de validation :
  - Vérifier que son programme respecte bien les spécifications définies
- Maintenance :
  - Effectuer les dernières possibles corrections vis-à-vis du bon fonctionnement du programme : pas d'ajout de nouvelles fonctionnalités !
  - Lister les possibles évolutions du programme
  - Faire une documentation d'utilisation (au minimum, un petit README.txt explicatif !)
- Et pour finir : Livraison d'un prototype **fonctionnel** :
  - Le programme doit être fonctionnel (compilation, warning, erreur, etc.).
  - Et fourni avec une documentation (-> utilisez minimalement le README.md de GitHub ou le Wiki)

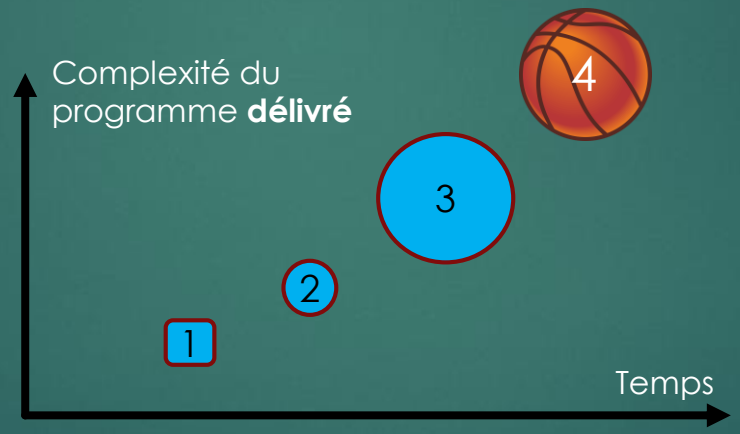
MÉTHODE PROPOSÉE :

- On reboucle !
- Remarque : Ce n'est donc pas grave si votre étude préliminaire n'était pas parfaite !
- Pas d'inquiétude : Vous verrez que chaque phase prend moins de temps (à part celle d'implémentation bien-sûr) ! Et au final, vous avancerez plus vite et avec de meilleurs résultats !



MÉTHODE PROPOSÉE : 0) Organisation/communication

- Faites une réunion d'équipe régulièrement (même via Skype/Hangout) pour faire un point d'avancement et pour distribuer de manière **consensuelle** les tâches et les délais associés
- Privilégiez des étapes de développement courtes et « simples » (donc moins ambitieuses) qui fournissent néanmoins un logiciel opérationnel (même si très basiques)



- Programmez en binôme sur une même partie (un qui code, puis un qui test / corrige / améliore / optimise / etc.), au moins sur les parties les plus importantes