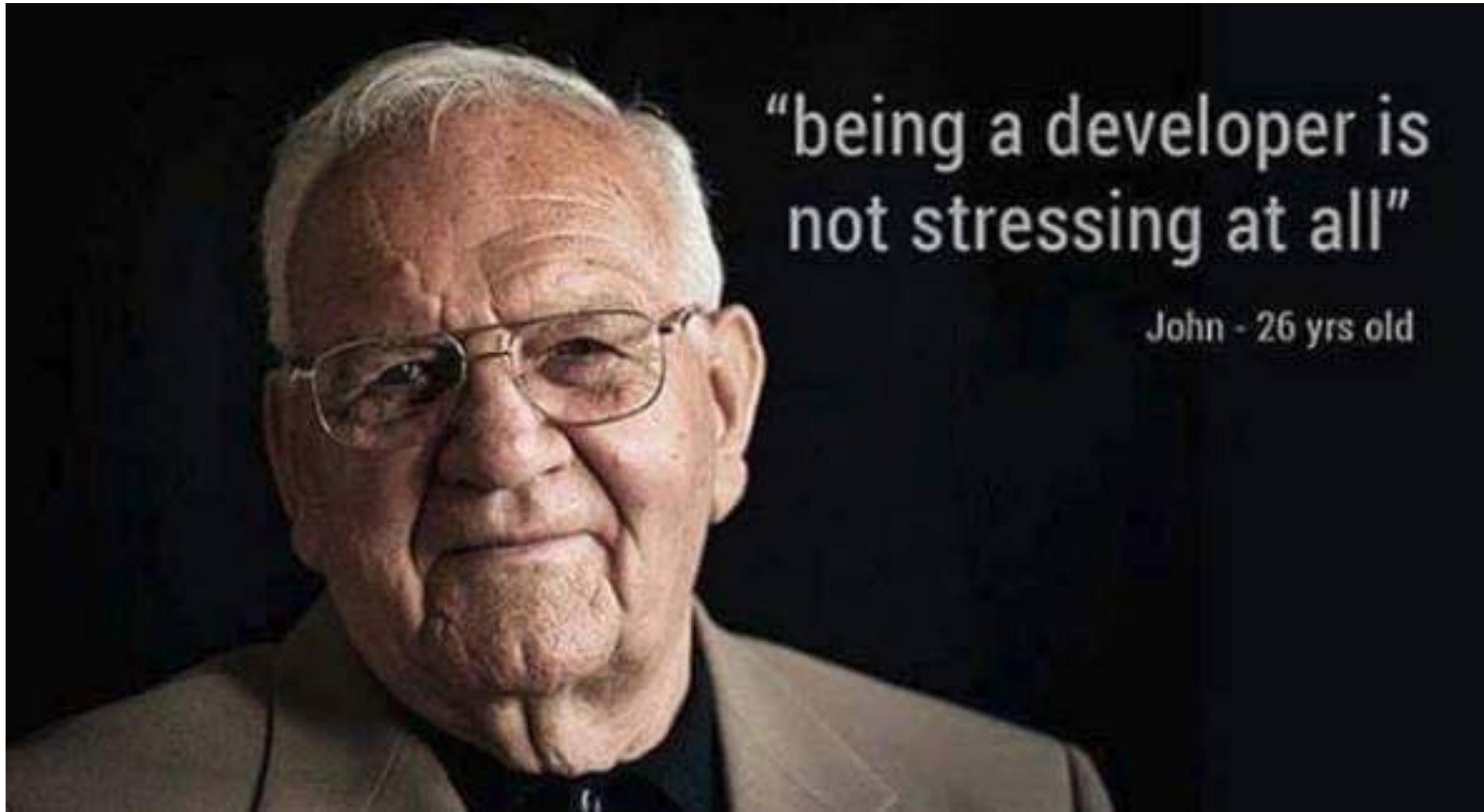


DÉVELOPPEMENT LOGICIEL





**"being a developer is
not stressing at all"**

John - 26 yrs old

Le développement logiciel

Créer un logiciel

L'écrire

Le faire évoluer

Le corriger

En équipe

De manière efficace

Avec des outils

A quoi ça vous sert ?

Mieux maîtriser le développement

Produire un logiciel de qualité

Travailler en mode projet

Mesures

Preuves

Historique

Collaboration

Quelques réflexes à acquérir

Formaliser sa démarche

Utiliser les bonnes pratiques du développement

Connaître quelques outils

Qu'est-ce qu'un logiciel ?

LAROUSSE :

Ensemble des **programmes**, **procédés** et **règles**, et
éventuellement de la documentation, relatifs au
fonctionnement d'un ensemble de **traitement de données**.
(Par opposition au matériel.)

<https://www.larousse.fr/dictionnaires/francais/logiciel/47666>

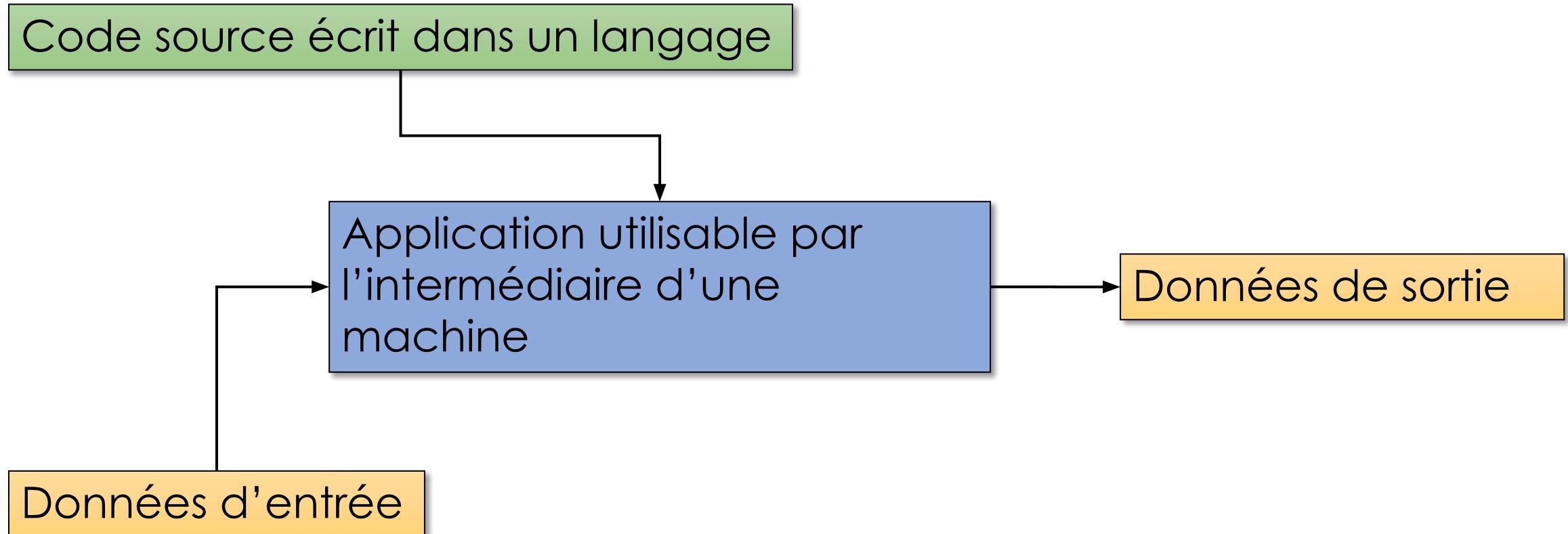
Qu'est-ce qu'un logiciel ?

WIKIPEDIA :

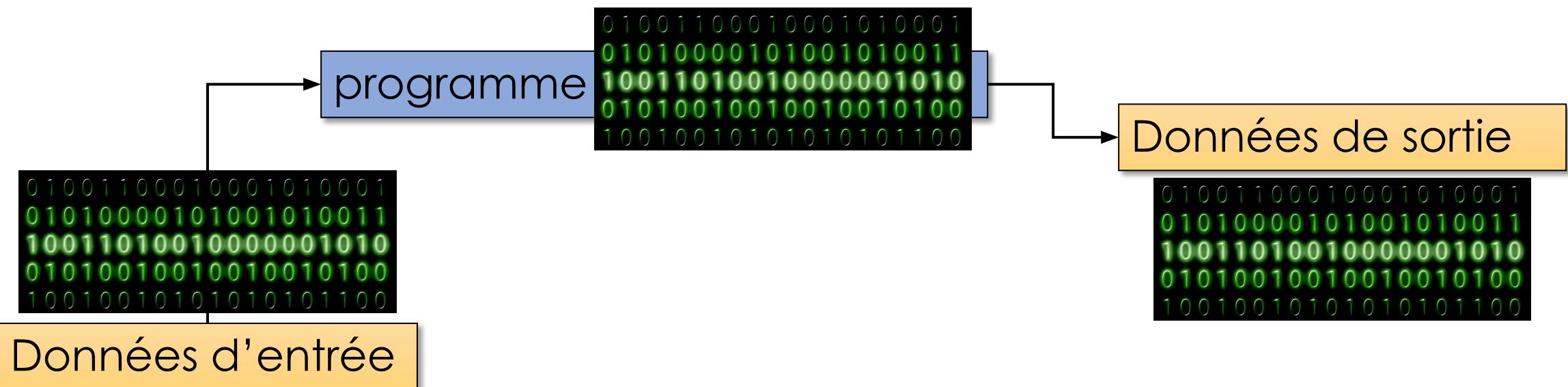
En informatique, un logiciel est un ensemble de **séquences d'instructions interprétables** par une machine et d'un jeu de données nécessaires à ces opérations. Le logiciel détermine donc les tâches qui peuvent être effectuées par la **machine, ordonne son fonctionnement** et lui procure ainsi son utilité fonctionnelle.

<https://fr.wikipedia.org/wiki/Logiciel>

En bref

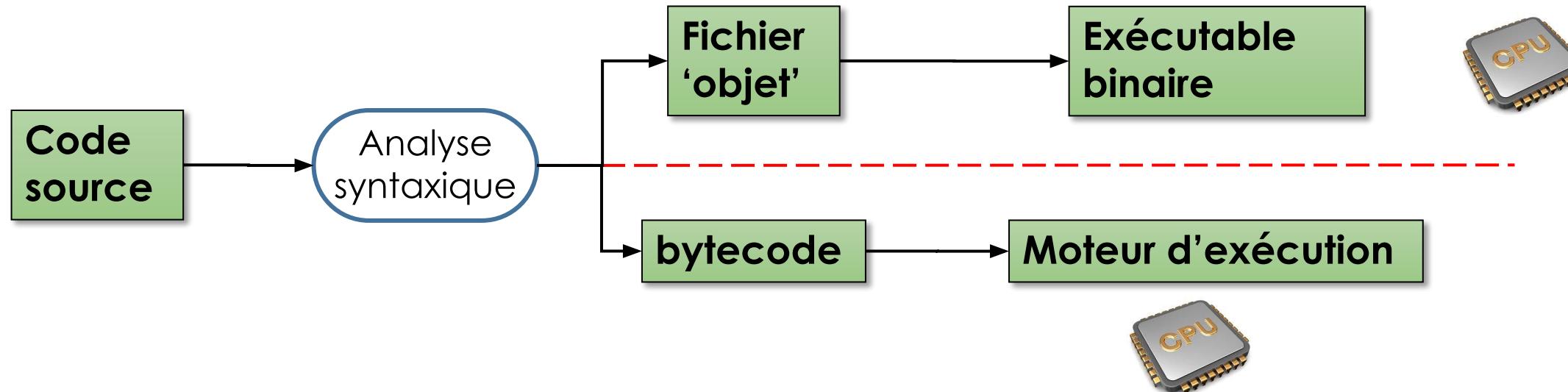


En pratique...



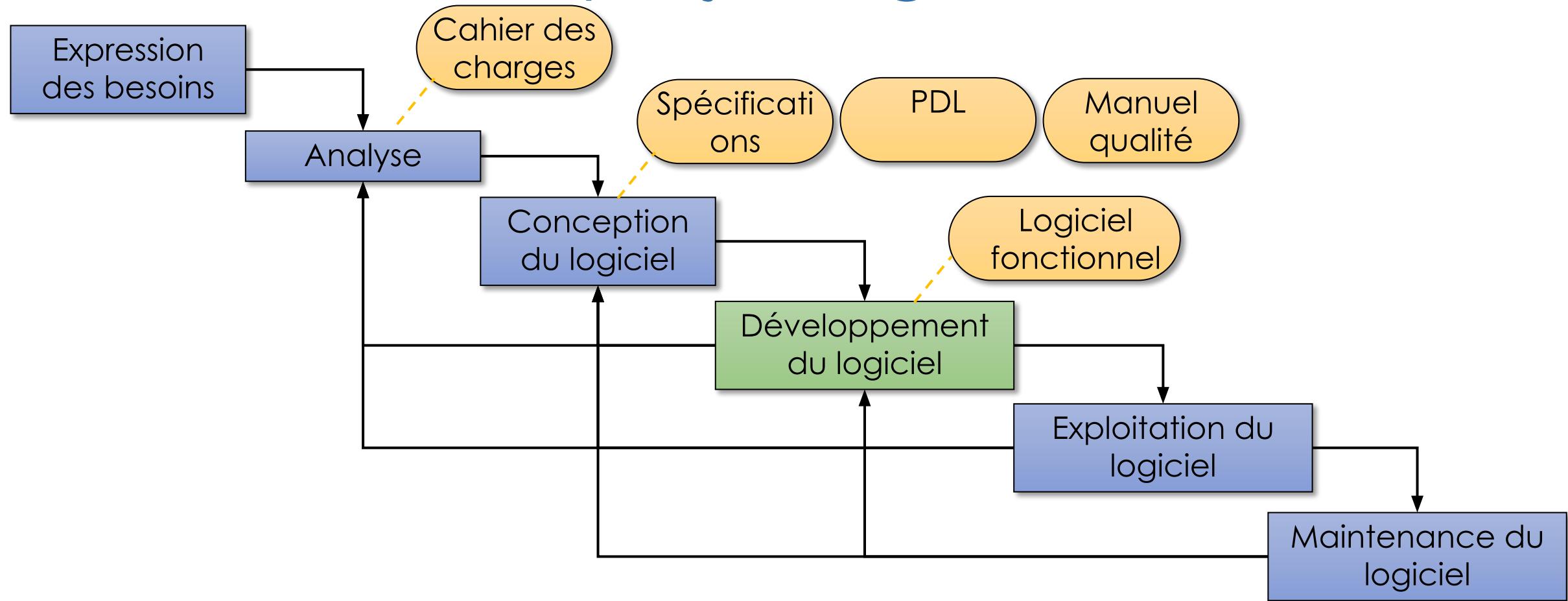
De la source à l'application

Langages compilés

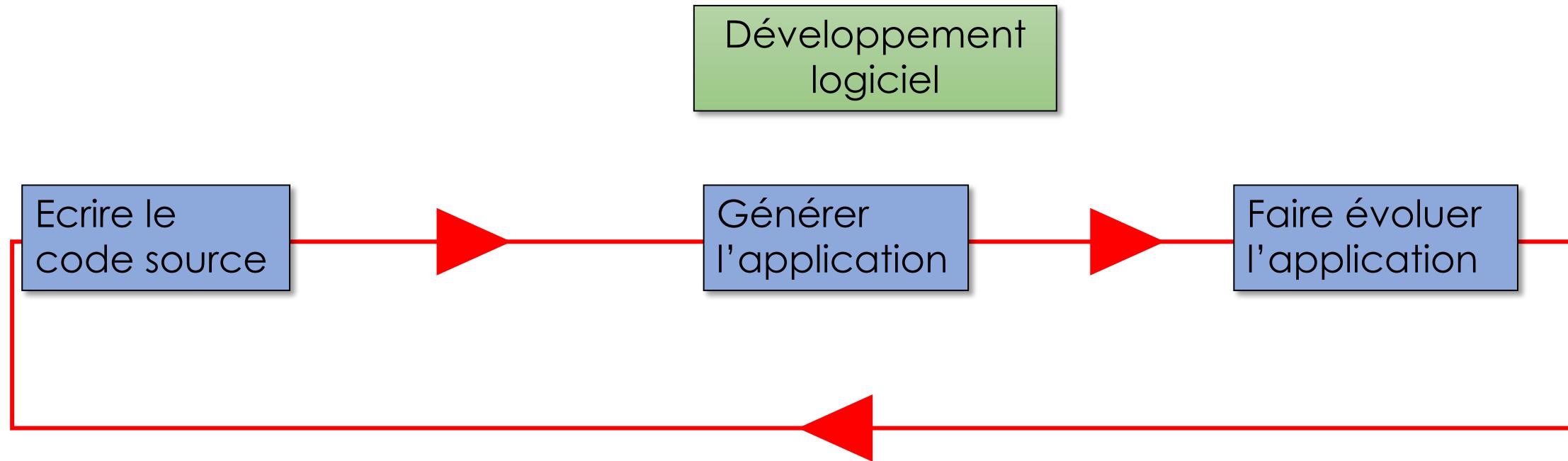


Langages interprétés

Le chemin du projet logiciel



Les activités



Le point de départ

Choix de plateforme technique (architecture et langage)

Documents de conception – algorithmes, classes, modules, interfaces

Pour

Ecrire des fichiers source

REDIGER LE CODE SOURCE

Ecriture de code propre et structuré

Objectif lisibilité - pas assuré par le langage lui-même

Indentation, coloration syntaxique

Règles de bonne pratique

C/C++ : [à la carte](#)

Python : [PEP8](#)

exemple

```
#include<stdio.h>

int a = 256;int main(){for(char b[a+a+a],
*c=b ,*d=b+ a ,*e=b+a+a,*f,*g=fgets(e,(b[
a]=b [a+a] =a- a,a) , stdin);c[0]=a-a,f=c
,c=d ,d=e ,e=f, f= g,g =g,g = fgets(e,a+a
-a+ a -a+a -a+ a- +a,stdin ),f +a-a ; pu\
tchar(+10)) { for( int h= 1,i=1,j, k=0 ,l
=e[0]==32,m,n=0,o=c [ 0]== 32, p, q=0;d[q
];p=(j=k,j)+(k=l,k*2)+(l=(i = i&&e [q] )
&& e[q+1 ] ==32, l *4)+( m=n,m*8)+( n =o,
16* n )+( o =(h =c[ q]&&h) &&c[q+1]==
32,o* (16+16) )+0-0 +0, putchar(" . . . . "
/*\ ( ||| ) | / | / * / . ' ) | ) \\ \\ \\ \\ ' "
" " ||| " " ||| " | " " ) | ) \\ \\ \\ \\ '/ | / ( "
"( / / | / \\ | \\ | / | / ( / ( / / | / \\ | \\ | "[d[q++]==
32?p:0]);}}
```

Ceci est un code source en langage C accepté par un compilateur.

Tiré de : <http://www.ioccc.org/years.html#2018>

Outils de rédaction

Coloration syntaxique et style :

Code:Blocks configuration de l'éditeur

Outil LINTER - SpellCheck

Avec Microsoft Visual Studio pour C/C++

Avec Spyder pour Python – outil **pylint**

Documentation de code

Utilisation de commentaires

Outils d'extraction de documentation

DOXYGEN : exemple d'utilisation en C

Doxygen

```
3  using namespace std;
4
5  void quadratic(float a, float b, float c) {
6
7      float x1, x2, discriminant, realPart, imaginaryPart;
8
9      discriminant = b * b - 4 * a * c;
10
11     if (discriminant > 0) {
12         x1 = (-b + sqrt(discriminant)) / (2 * a);
13         x2 = (-b - sqrt(discriminant)) / (2 * a);
14         cout << "Roots are real and different." << endl;
15         cout << "x1 = " << x1 << endl;
16         cout << "x2 = " << x2 << endl;
17     }
18
19     else if (discriminant == 0) {
20         cout << "Roots are real and same." << endl;
21         x1 = (-b + sqrt(discriminant)) / (2 * a).
```

PRODUCTION DE LOGICIEL

IDE et production de logiciel

Integrated Development Environment

Edition et formatage du code source (vu)

Génération / production de logiciel (cible)

Debug

Gestion de version

Chaine de production

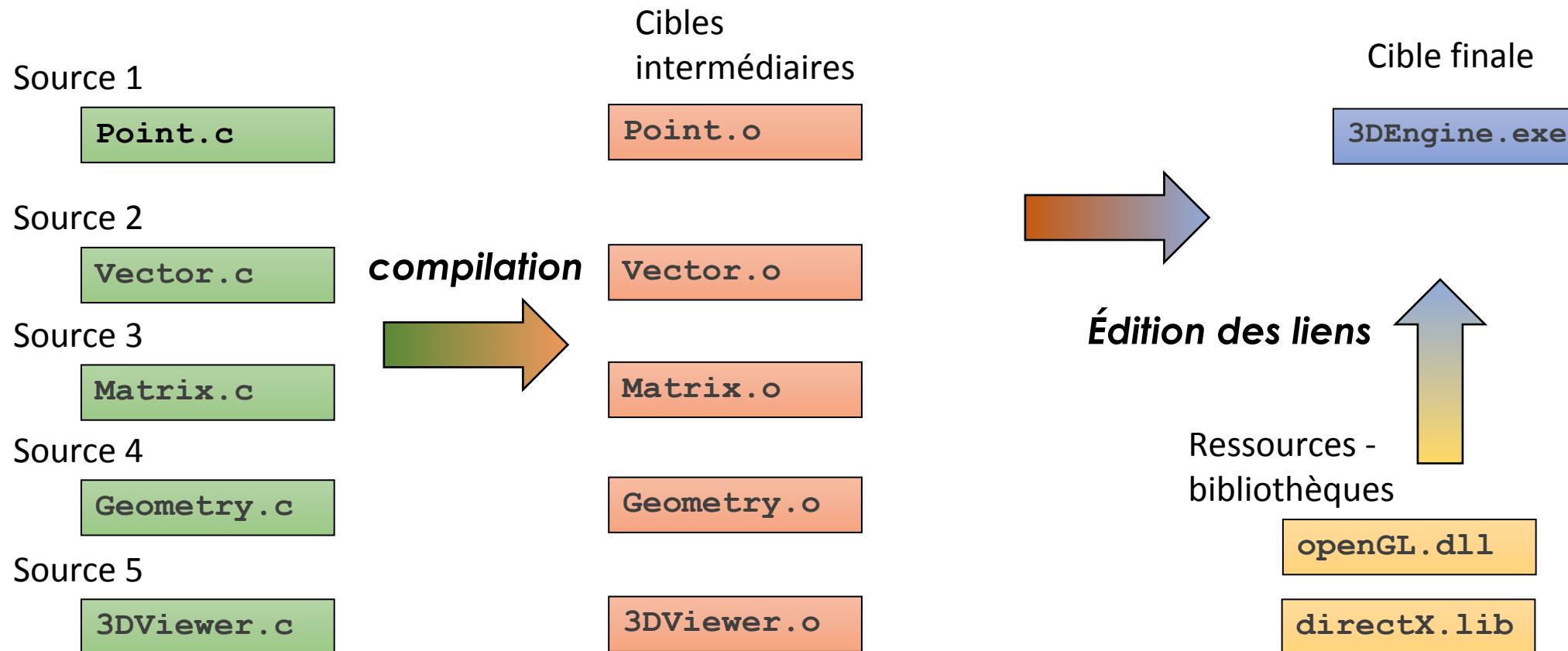
De la source (source) à la cible (target)

Un outil transforme une source en cible

En plusieurs étapes :

Cible intermédiaire / cible finale

Chaine de production en C/C++



Makefile, dépendance

Un makefile indique comment générer les cibles à partir d'une source ou dépendance, avec le format :

cible : dépendance (source)

action

Avec makefile

Source 1

Point.c

Cibles
intermédiaires

Point.o

Source 2

Vector.c

Vector.o

Source 3

Matrix.c

Matrix.o

Source 4

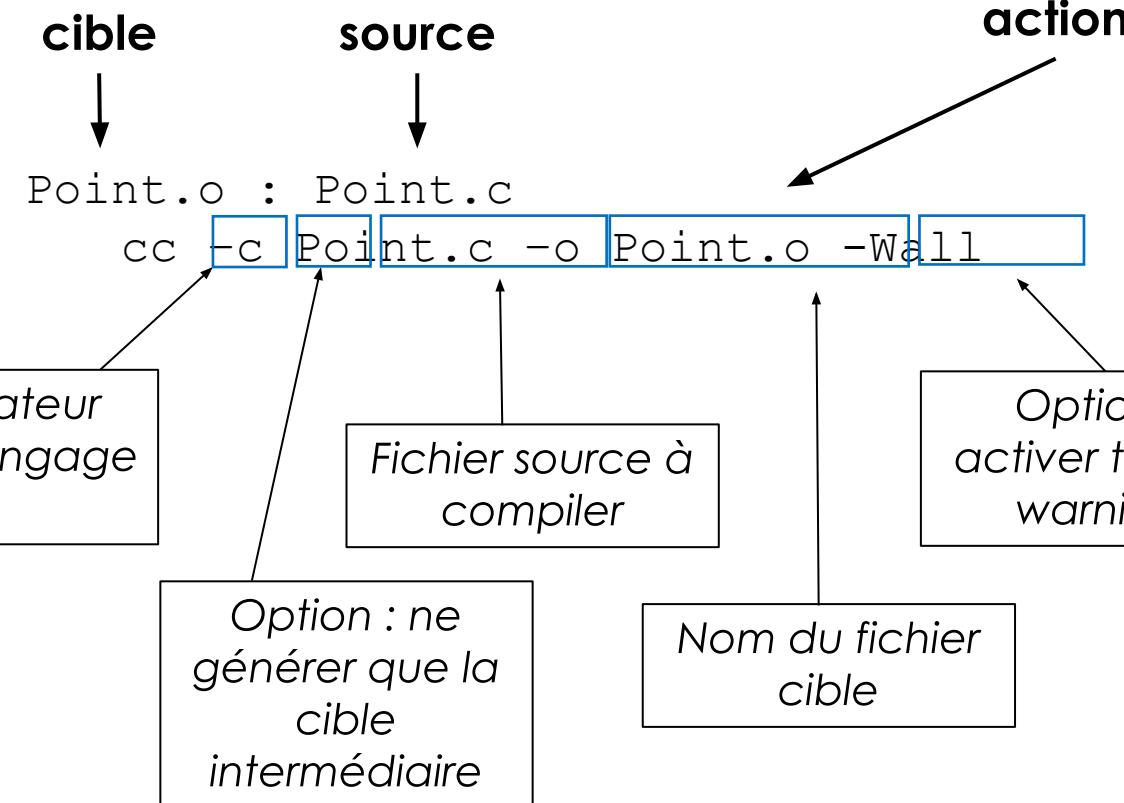
Geometry.c

Geometry.o

Source 5

3DViewer.c

3DViewer.o



Validité cible / source

Timestamping

Une cible doit être générée :

Si elle n'existe pas

Ou

Si elle est plus ancienne que sa source / dépendance

Exemple

Makefile :

```
all : executable
executable : file1.o file2.o
        gcc -o executable file1.o file2.o
file1.o : file1.c file1.h
        gcc -c file1.c
file2.o : file2.c file1.h file2.h
        gcc -c file2.c
clean :
        rm file1.o file2.o executable core
```

Exemple

Compilation :

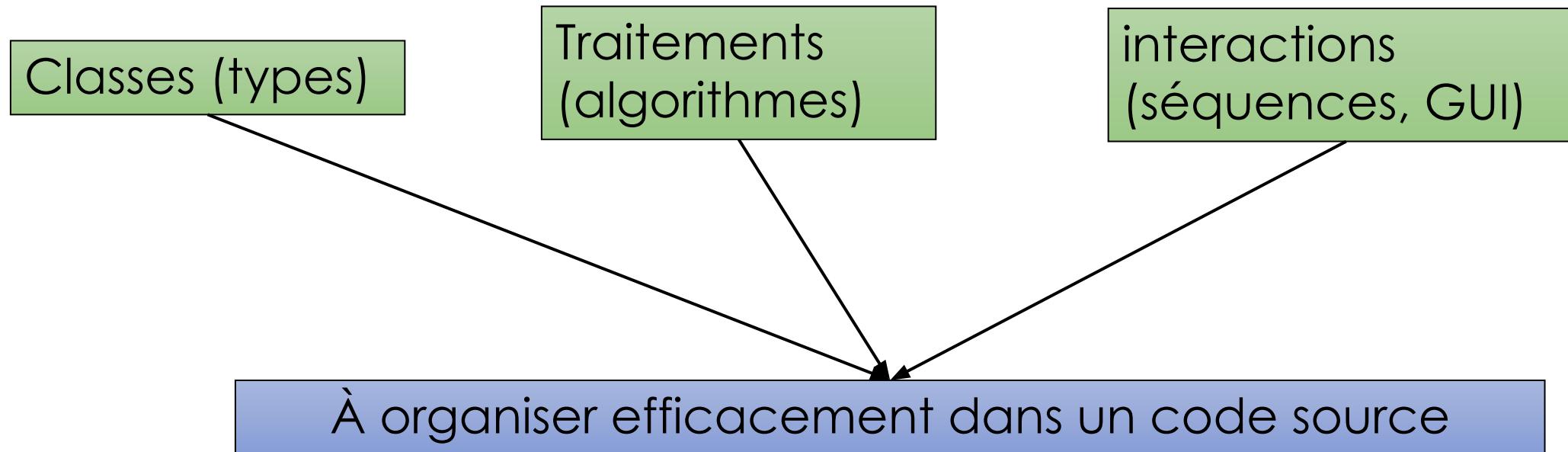
```
% make clean
rm file1.o file2.o executable core
rm: cannot remove `core': No such file or directory
% make
gcc -c file1.c
gcc -c file2.c
gcc -o executable file1.o file2.o
% touch file2.h
% make
gcc -c file2.c
gcc -o executable file1.o file2.o
```

```
% touch file2.o
% make
gcc -o executable file1.o file2.o
% touch file1.h
% make
gcc -c file1.c
gcc -c file2.c
gcc -o executable file1.o file2.o
```

MODULARITE DU CODE SOURCE

Règles de modularité

Conception / Modélisation de l'application



Rôle des fonctions et modules

Découpage logique / fonctionnel

- ✓ Pour organiser un projet de développement
- ✓ Pour exploiter les fonctionnalités d'un projet (API)

Un exemple d'API : OGRE3D

Allons faire un tour sur le site :



Pour le développement

Un module regroupe des fonctionnalités (fonctions) par thème

Un module a donc un rôle précis **architecture**

De même pour les fonctions

De même pour la conception OO et les classes

Rédiger une fonction

4 questions :

Que fait-elle ?

nom

De quoi a-t-elle besoin ?

paramètres

Quel résultat fournit-elle ?

valeur de retour

Comment le fait-elle ?

code interne

Principe KISS

Keep It Simple, Stupid

Une fonction fait **une seule chose**, et le fait bien

Simple à écrire, simple à tester

Souplesse pour la mise au point

Exemple : tri de tableau

Analyser pour séquencer

Découper le problème :

- Créer un tableau
- Le remplir
- Le trier
- Afficher le résultat

Avec une seule fonction

- Créer un tableau
- Le remplir
- Le trier
- Afficher le résultat

Ça paraît confortable ☺

Pas de paramètres

Pas de valeur de retour

Attention aux apparences

Rôle / nom pas pertinent

Code ‘qui fait tout’

Evolution difficile

Remplir le tableau :

aléatoirement par programme ?

par saisie ?

à partir d'un fichier ?

Aucun lien avec la partie 'Trier le tableau'



Découpage en fonctions distinctes

4 fonctions (ou plus)

Array createArray(size)

scanArray(Array)

initArrayFromFile(Array, filename)

sortArray(Array)

displayArray(Array)

saveArrayToFile(Array, filename)

Fichiers d'entête (C/C++)

Séparer le **quoi** du **comment**

À faire en conception

Si possible en développement

Permet de partager des fonctionnalités (**et pas du code**)

imports

En C/C++ : inclusion des déclarations **#include**

En Python : import d'objets (tout est objet) **import**

N'importer que ce qui est nécessaire

Utilisation de fichiers .h en C/C++

Quand on utilise un type qui n'est pas un type de base

Quand on appelle une fonction

exemple avec le programme le plus simple possible

exemple avec un nouveau type

Règles d'inclusion

Ne **jamais** inclure un fichier .c

Jamais

Inclure un fichier .h quand le compilateur en a besoin

appel de fonction ou utilisation d'un nouveau type.

STRUCTURER LA CHAINE DE PRODUCTION

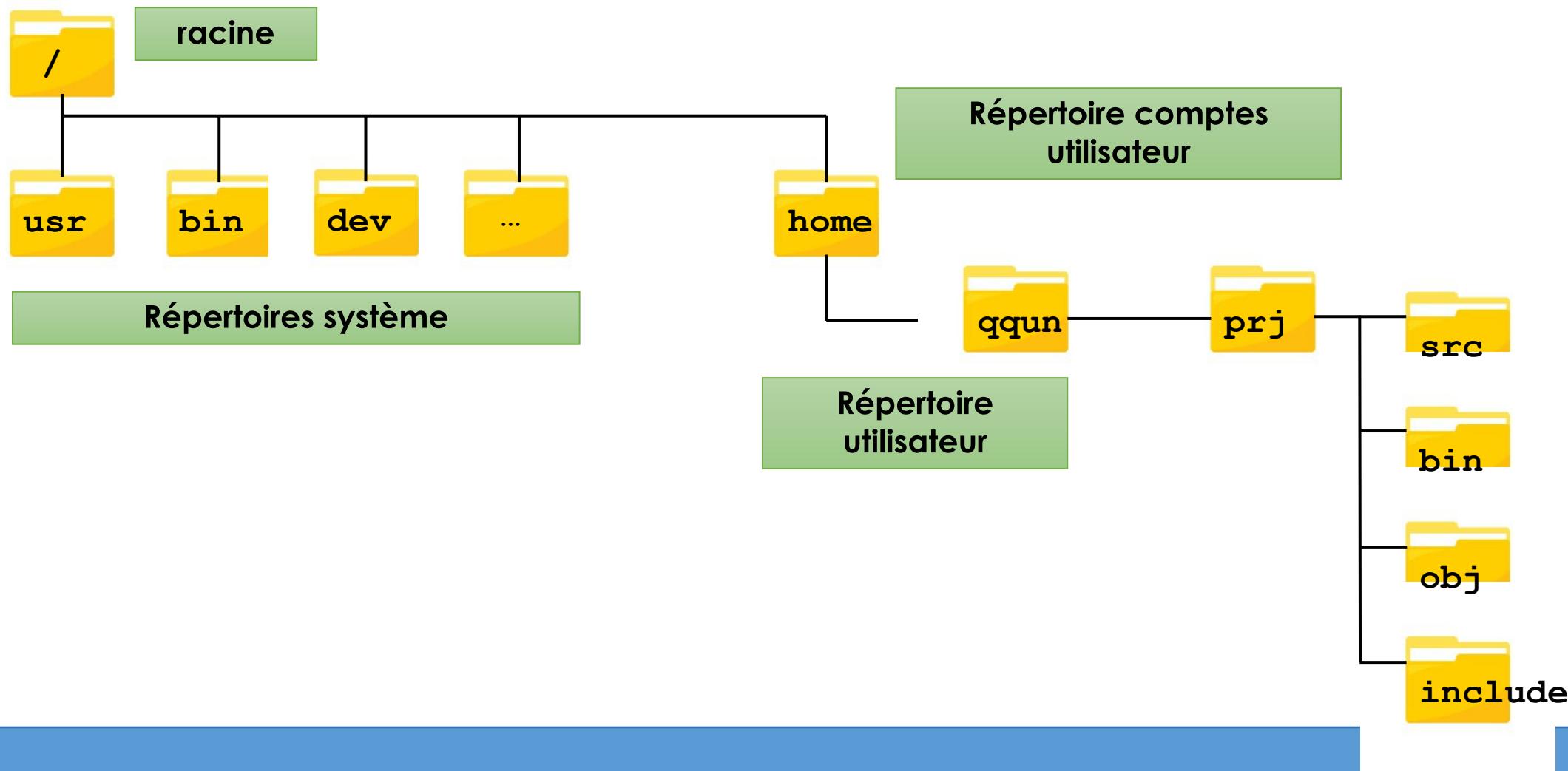
Structure de fichiers

Classer les fichiers selon leurs types

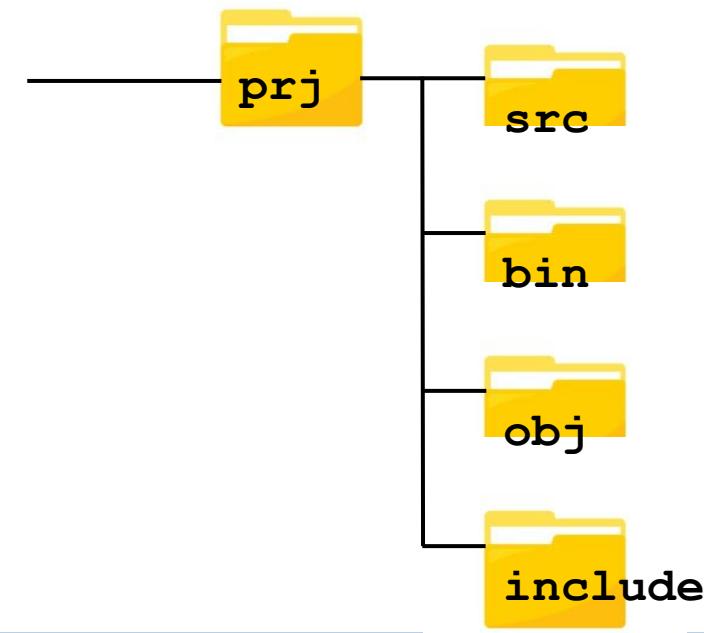
Exemple : le standard du système de fichier Unix

FHS : Filesystem Hierarchy Standard

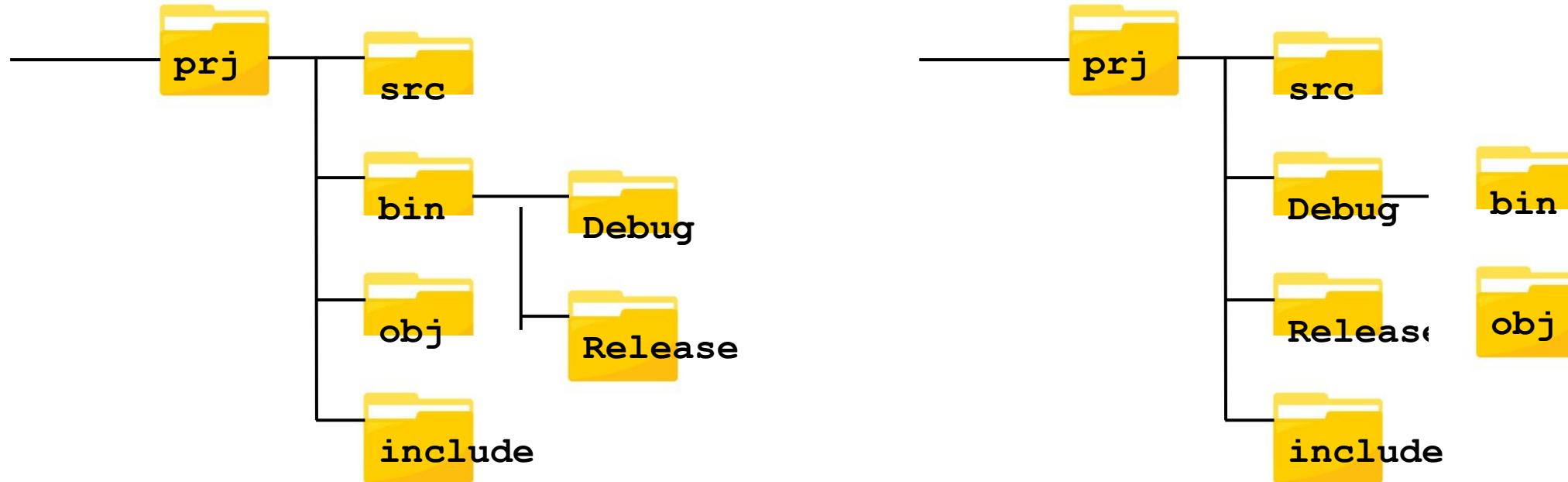
Unix – Structure hiérarchique



Unix – Structure hiérarchique



Sous répertoires de projet



Gestion des configurations

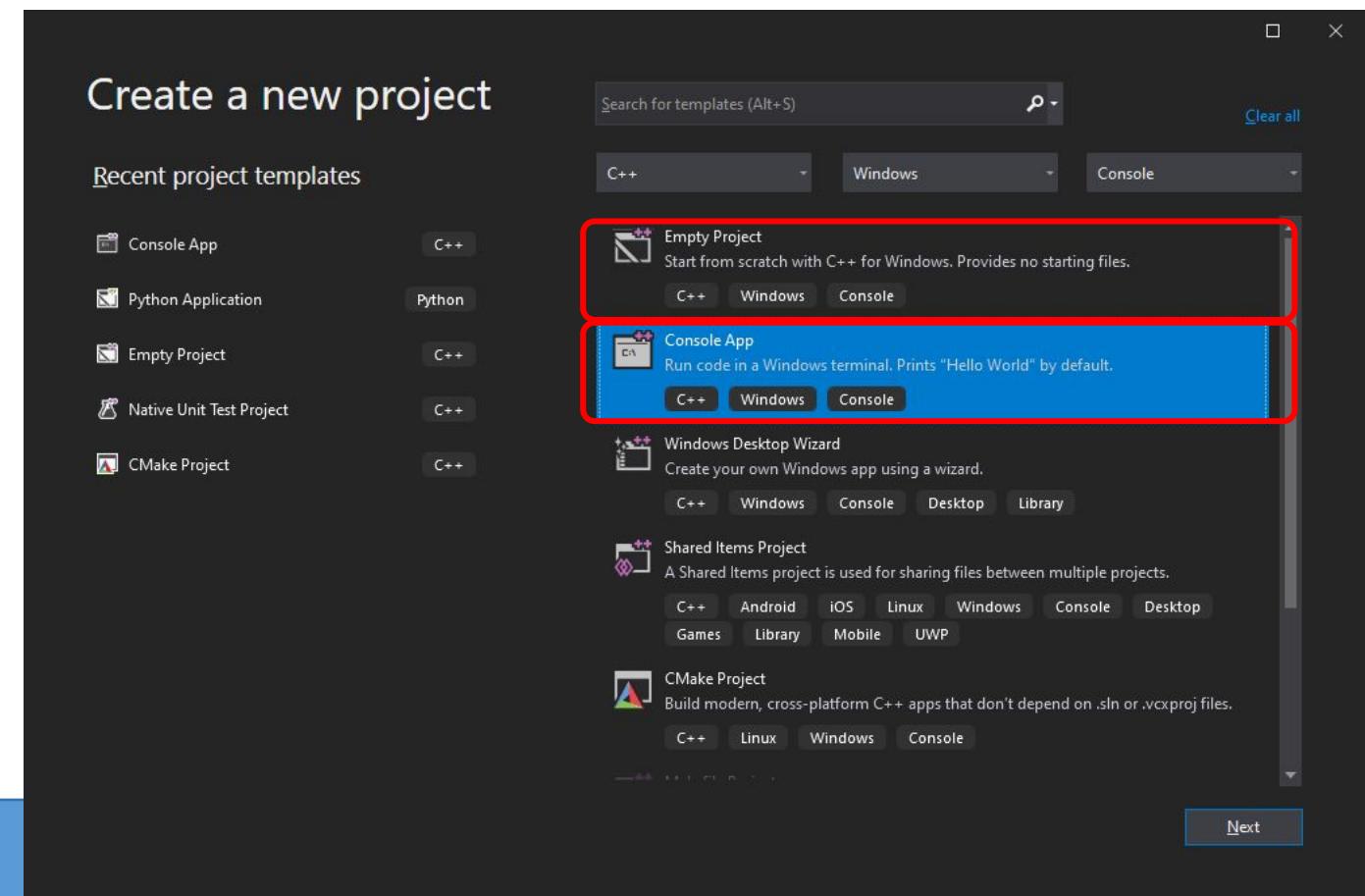
Plusieurs types de cible

A partir d'ensemble et de sous-ensembles de source

Configuration des règles de génération

Types de cible

En C : liste des types de projet Microsoft Visual Studio



Un exemple :

Maintenir avec un seul jeu de code source :

Un exécutable de production **Release**

Un exécutable de mise au point **Debug**

Un exécutable pour des tests **Test**

Une bibliothèque API **lib**

CYCLE DE DÉVELOPPEMENT LOGICIEL

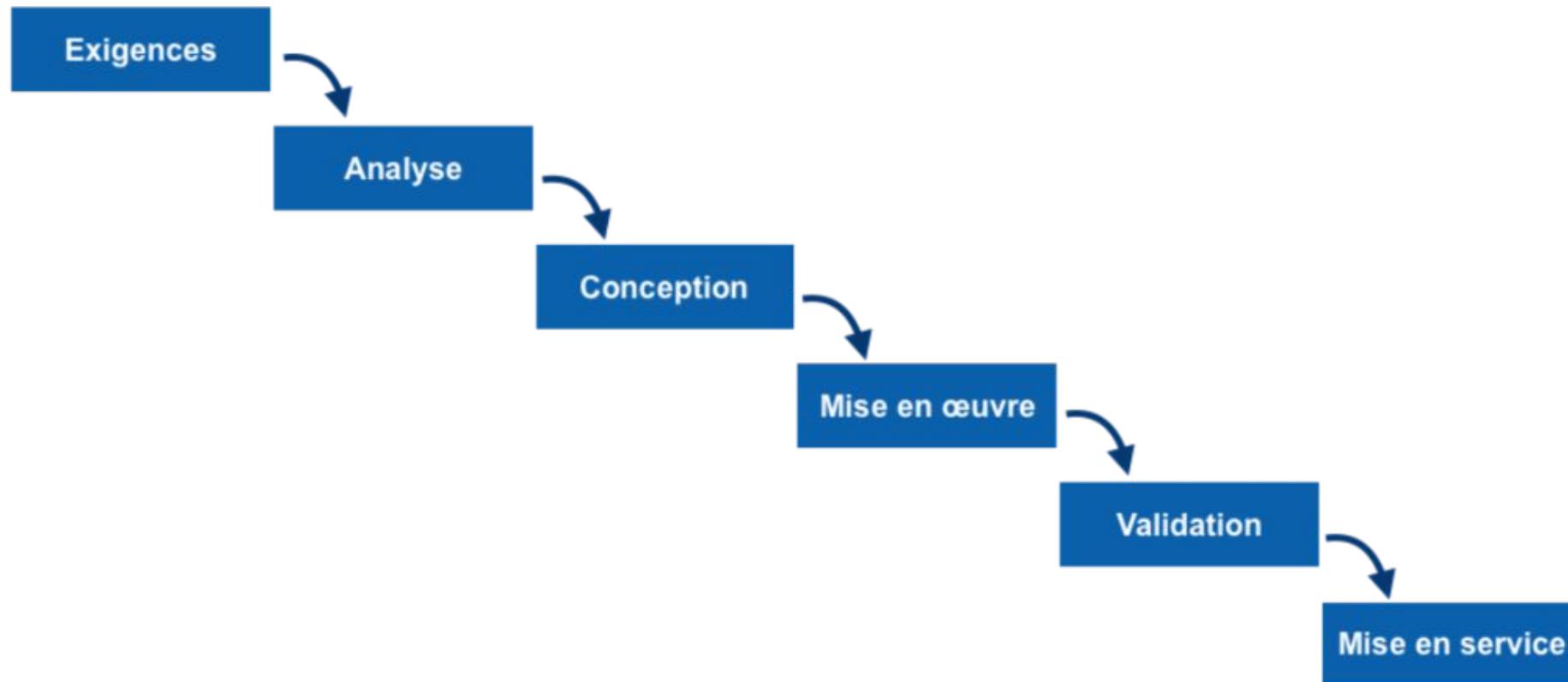
Modèle en cascade (1970)

Organisation sous forme de phases linéaires et séquentielles

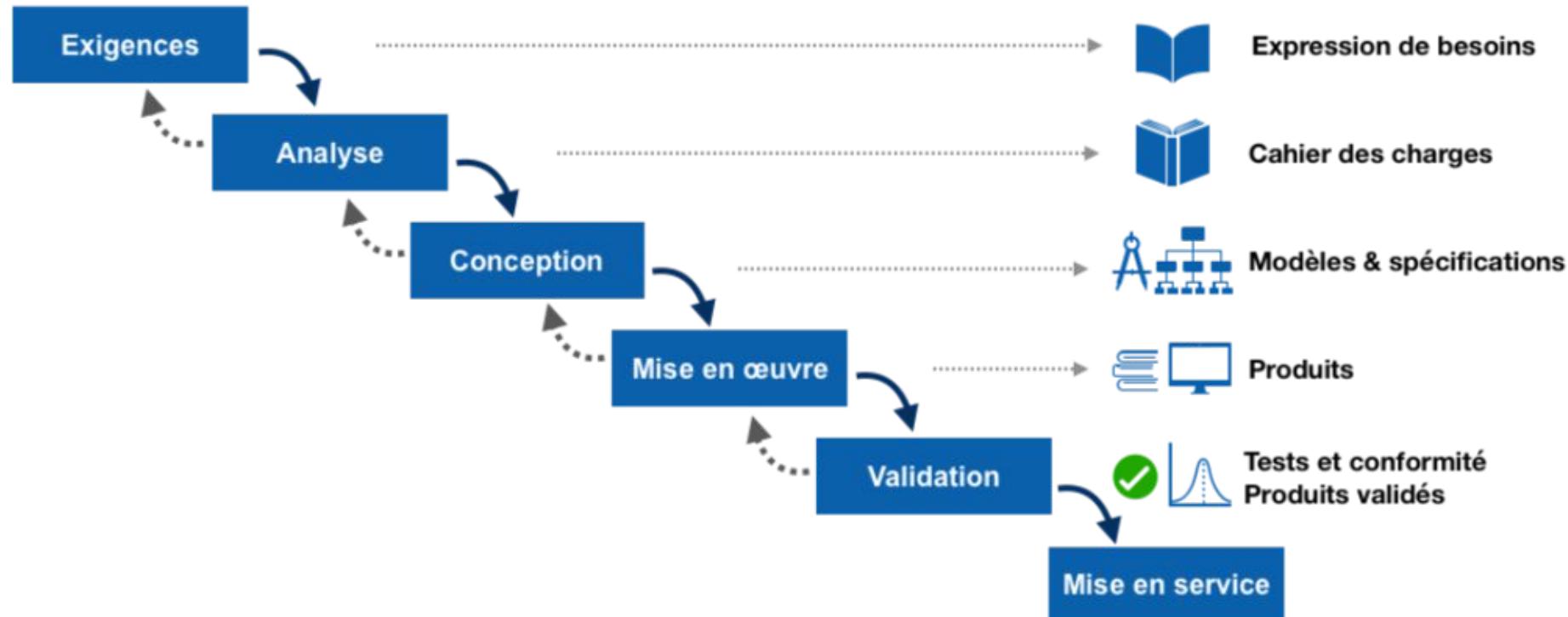
Phase ->

spécialisation des tâches et dépend des résultats de la phase précédente

Modèle en cascade



Modèle en cascade

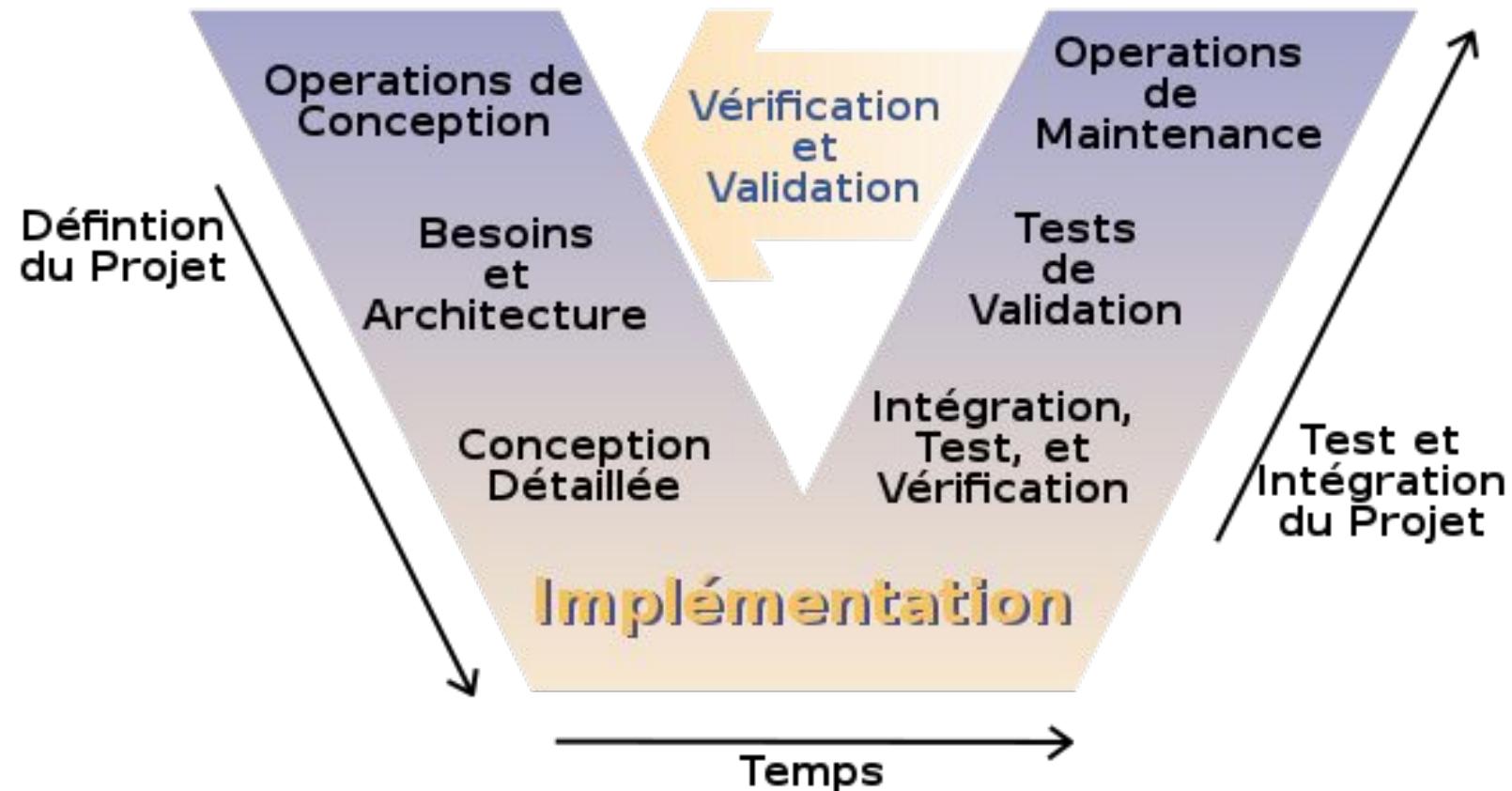


Cycle en V

- flux d'activité descendant : du produit jusqu'à sa réalisation
- flux ascendant : assemble le produit en vérifiant sa qualité

Issu du modèle en cascade approche séquentielle et linéaire de phases + activités d'intégration de système et met en regard chaque phase de production successive avec sa phase de validation correspondante

Cycle en V



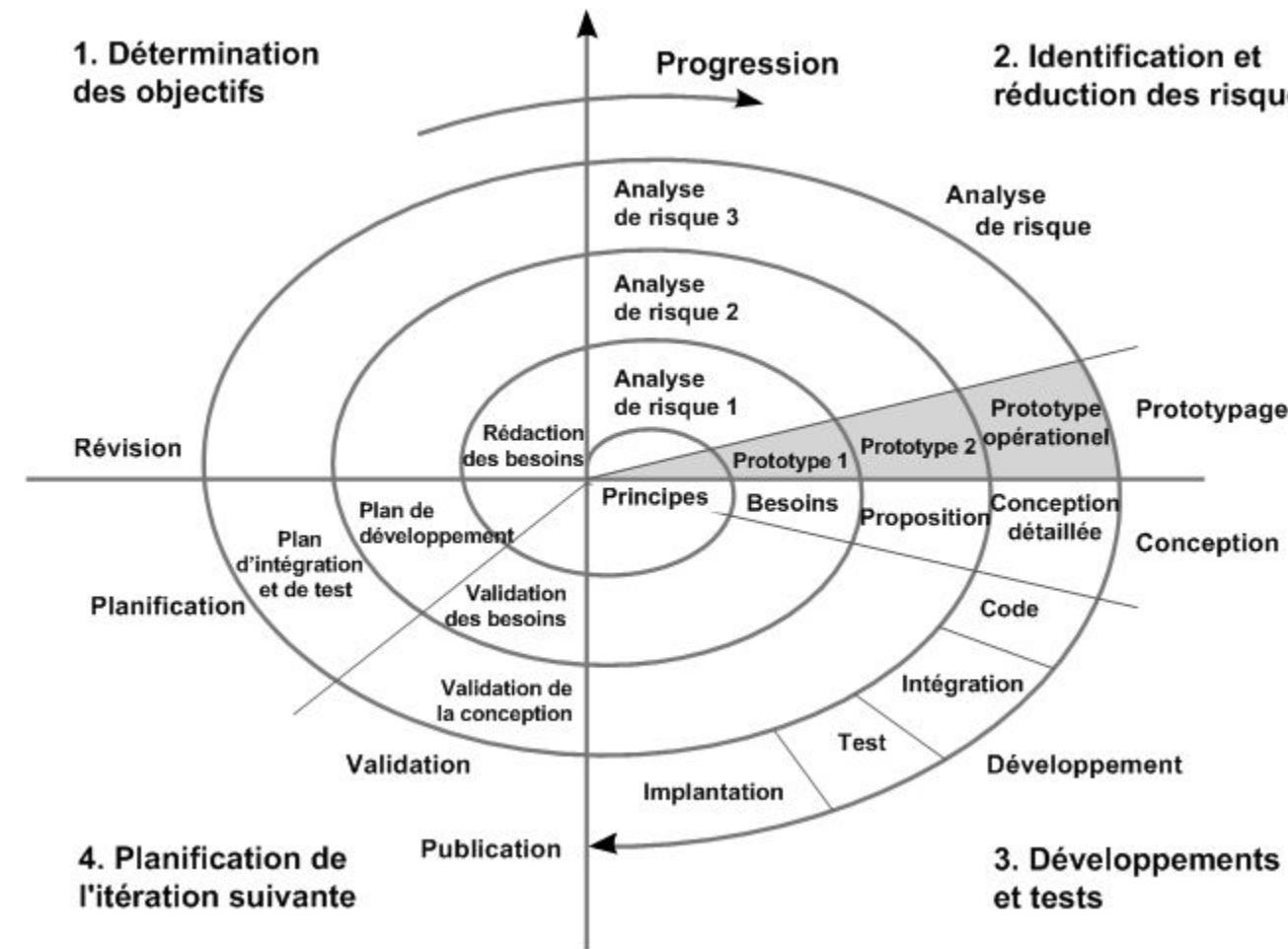
Modèle en spirale ou incrémental (1988)

Réponse aux inconvénients du modèle en cascade

Spirales répétées jusqu'à ce que le produit fini puisse être livré

Le produit est continuellement travaillé et les améliorations se déroulent souvent en petites étapes

Modèle en spirale ou incrémental



Méthode Agile (2001) : définition

- groupes de pratiques de pilotage et réalisation de projets
- référence de multiples méthodes existantes
- implique au maximum le demandeur (client) et permettent une grande réactivité à ses demandes
- cycle de développement itératif, incrémental et adaptatif

Méthode Agile (2001) : implémentation

Scrum (1995/2001) : amélioration continue

XP extrem programming (1999) : réingénierie immédiate des processus

=> lean management

Amélioration continue

Vérifie à chaque modification de code source que le résultat des modifications ne produit pas de régression dans l'application développée

But de cette pratique est de détecter les problèmes d'intégration au plus tôt lors du développement

Automatiser l'exécution des suites de tests et de voir l'évolution du développement du logiciel

Scrum VS Cycle en V

Thème	Cycle en V	Scrum
Cycle de vie	Phases séquentielles	Processus itératif
Livraison	À la fin de la réalisation de toutes les fonctionnalités → livraison tardive	Utilisation partielle du produit suite à la priorisation des besoins → livraison plus rapide
Contrôle Qualité	À la livraison finale (fin du cycle de développement) → effet tunnel	À chaque livraison partielle au client
Spécification	Pas de changement possible sans revenir à la phase de spécifications et repasser par toutes les autres phases → délais et coûts supplémentaires	Spécifications plus souples en ajoutant/modifiant les fonctionnalités aux sprints suivants qui n'étaient pas prévues au départ → principal atout de la méthode Agile
Planification	Plans détaillés basés sur des exigences stables définies dès le début du projet	Planification adaptative et ajustements si nécessaires en fonction des nouvelles demandes
Équipe	Intervention uniquement dans la phase de développement, pas de vision globale du projet	Engagements, échanges et prises de décisions collectives par l'équipe
Documentation	Quantité importante	Strict nécessaire

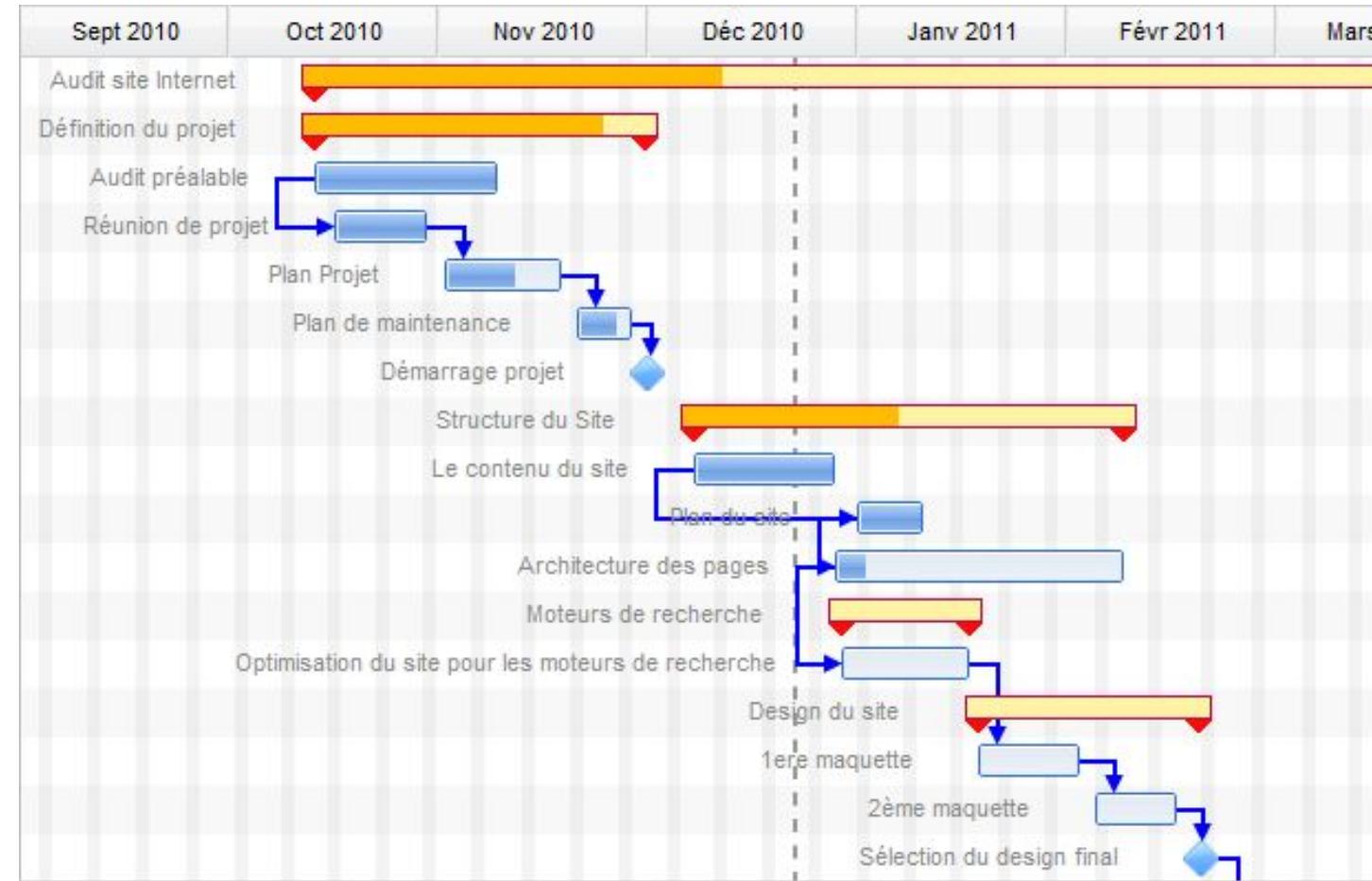
Diagramme de Gantt

Gantt + réseau PERT (ordonnancement, gestion de projet)
=> permet de visualiser dans le temps les diverses tâches composant un projet

Objectifs : planifier de façon optimale et communiquer sur le planning établi et les choix qu'il impose

- de déterminer les dates de réalisation d'un projet
- d'identifier les marges existantes sur certaines tâches
- de visualiser le retard ou l'avancement des travaux.

Diagramme de Gantt



Méthode Kanban

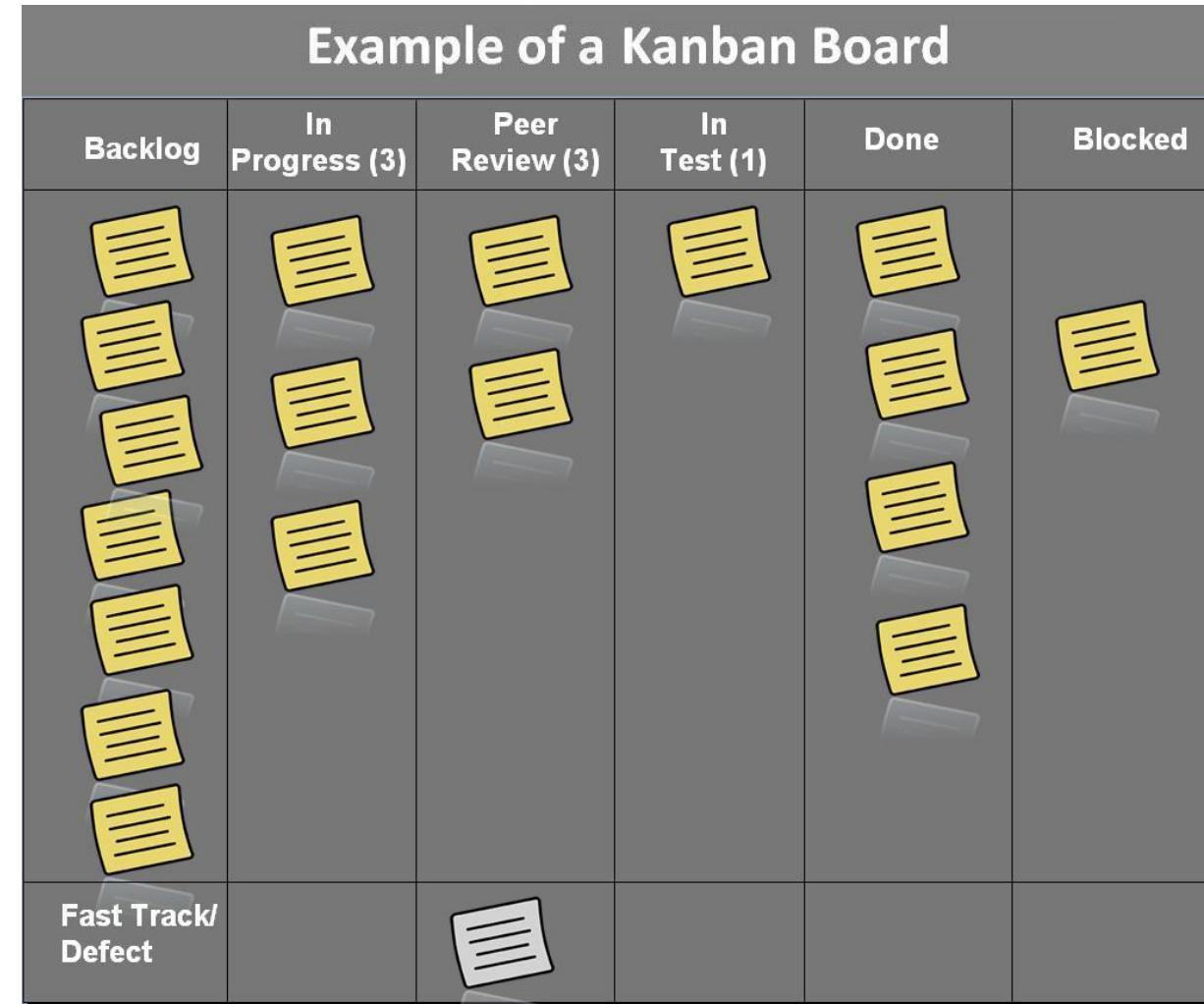
Méthode de gestion des connaissances avec une organisation de type Juste-à-temps en fournissant l'information ponctuellement aux membres de l'équipe afin de ne pas les surcharger

Processus complet : de l'analyse des tâches jusqu'à leur livraison au client est consultable par tous les participants, chacun prenant ses tâches depuis une file d'attente

Système visuel de gestion des processus qui indique quoi produire, quand le produire et en quelle quantité

Approche est inspirée du système de production de Toyota et des méthodes lean

Méthode Kanban



DevOps (2007)

Unification du développement logiciel et de l'administration système des infrastructures informatiques

Cycles de développement courts, augmentation de la fréquence des déploiements et livraisons continues

Automation et du suivi du :

- développement
- intégration
- tests
- livraison et déploiement
- exploitation
- maintenance des infrastructures

Choix du cycle de développement

Dépend de différents facteurs :

- portée du projet
- budget
- niveau requis de soutien et de maintenance
- flexibilité => méthodes agiles

Bonnes pratiques de développement

- Programmation en binôme
(<https://fr.slideshare.net/YvesHanouille/pair-programming-is-like-sex>)
- Burndown chart
- Planning poker