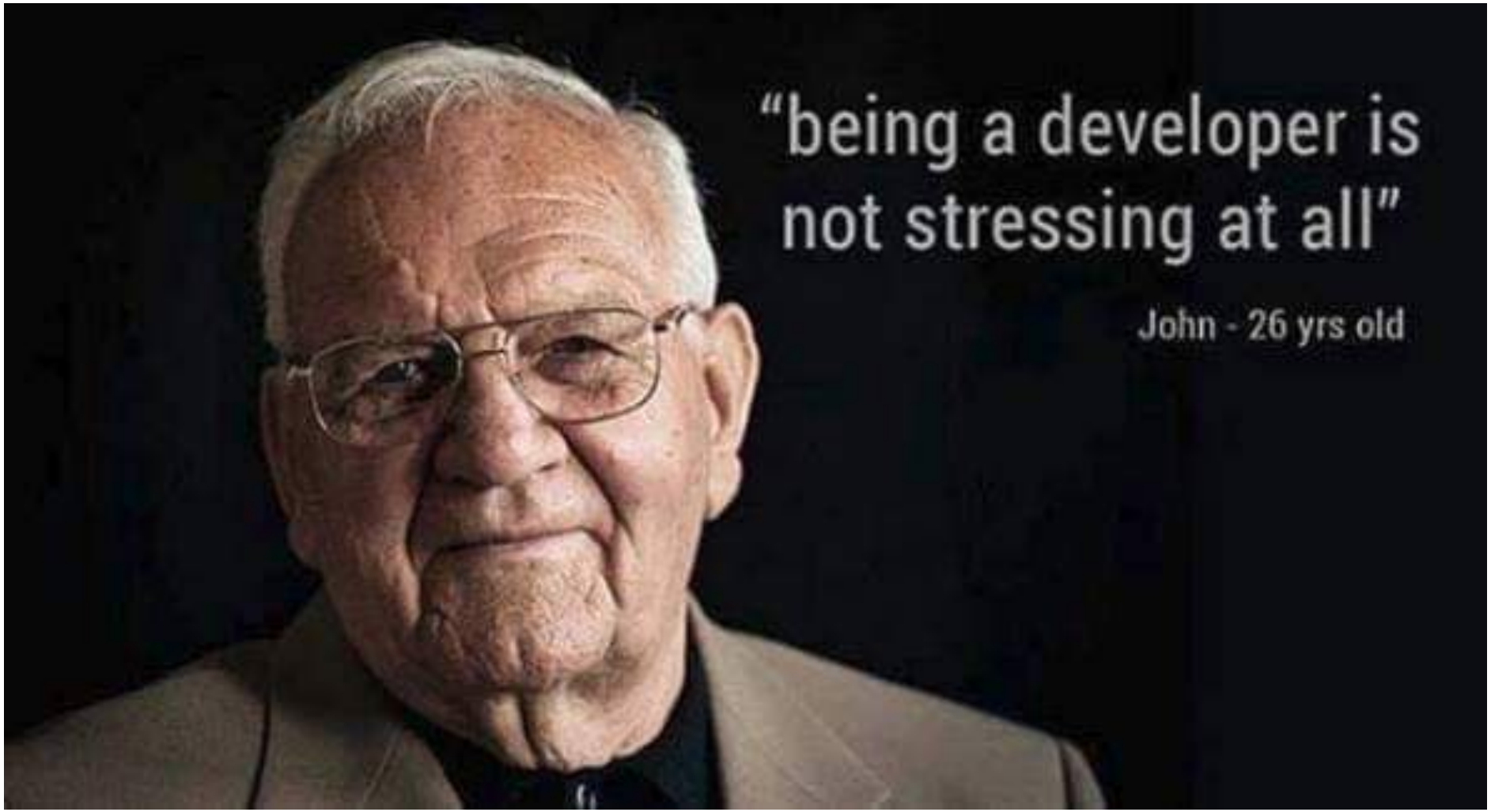
SOFTWARE DEVELOPMENT

efrei PARIS
Ecole d'ingénieurs du numérique


“being a developer is not stressing at all”
John - 26 yrs old

Software development

How to create a software

Write it

Making it evolve

Correct it

In team

In an efficient way

With tools

What is the goal ?

Better control of development

Producing quality software

- Measures
- Proofs
- History

Working in project mode

- Collaboration

Some reflexes to acquire

Formalise your approach

Using good development practice

Know some tools

What is software?

LAROUSSE :

A set of programmes, procedures and rules, and possibly documentation, relating to the operation of a data processing system. (As opposed to hardware.)

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

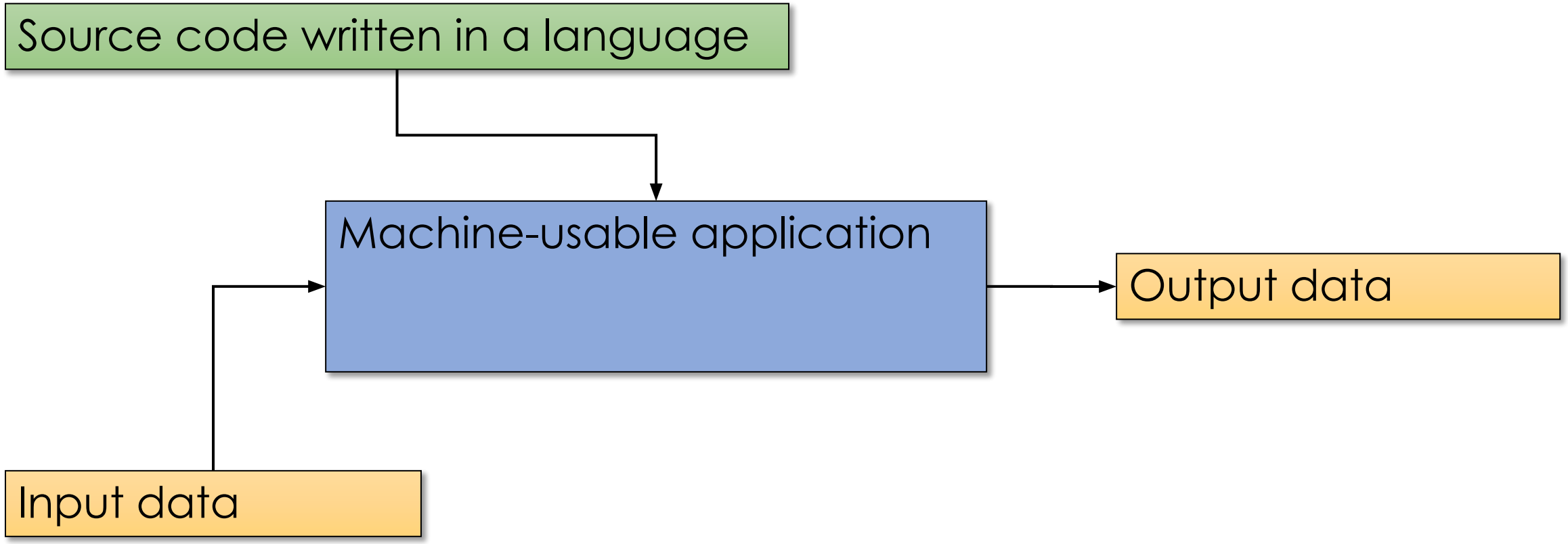
What is software?

WIKIPEDIA :

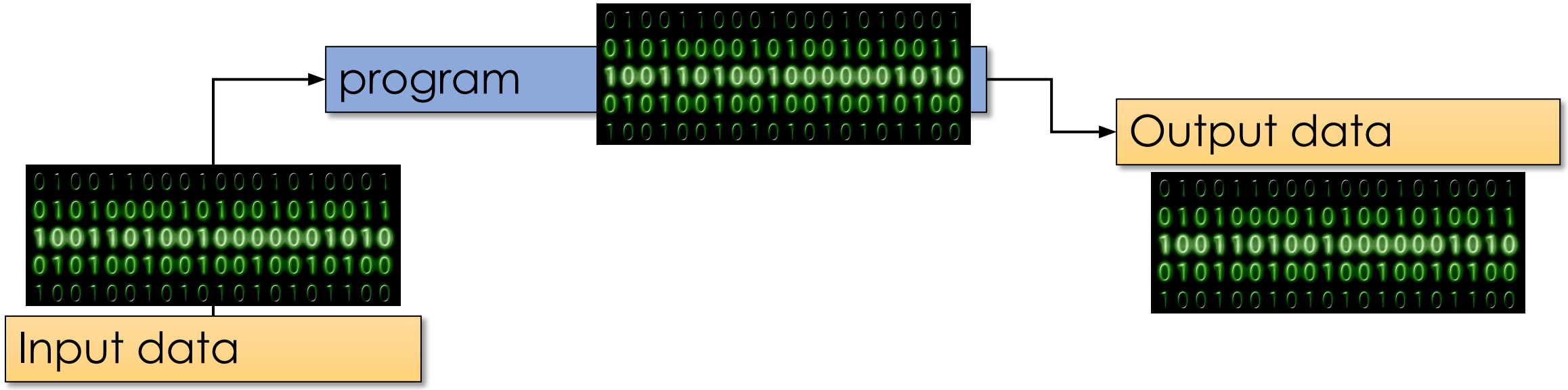
In computer science, software is a set of sequences of instructions that can be interpreted by a machine and a set of data required for these operations. The software therefore determines the tasks that can be performed by the machine, orders its operation and thus gives it its functional utility.

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

In brief

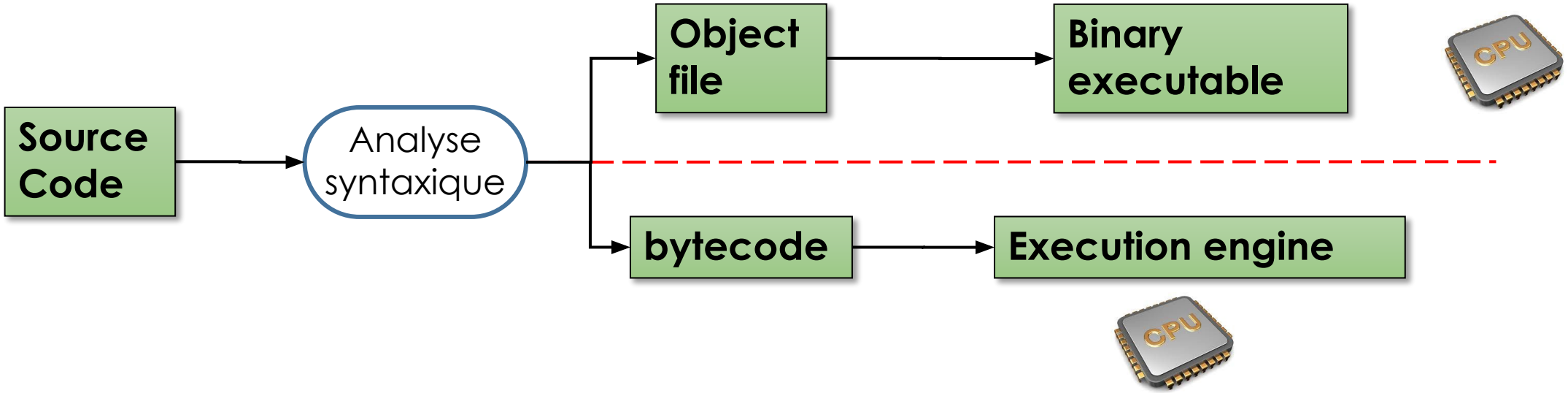

Source code written in a language
Machine-usable application
Output data
Input data

In practice...



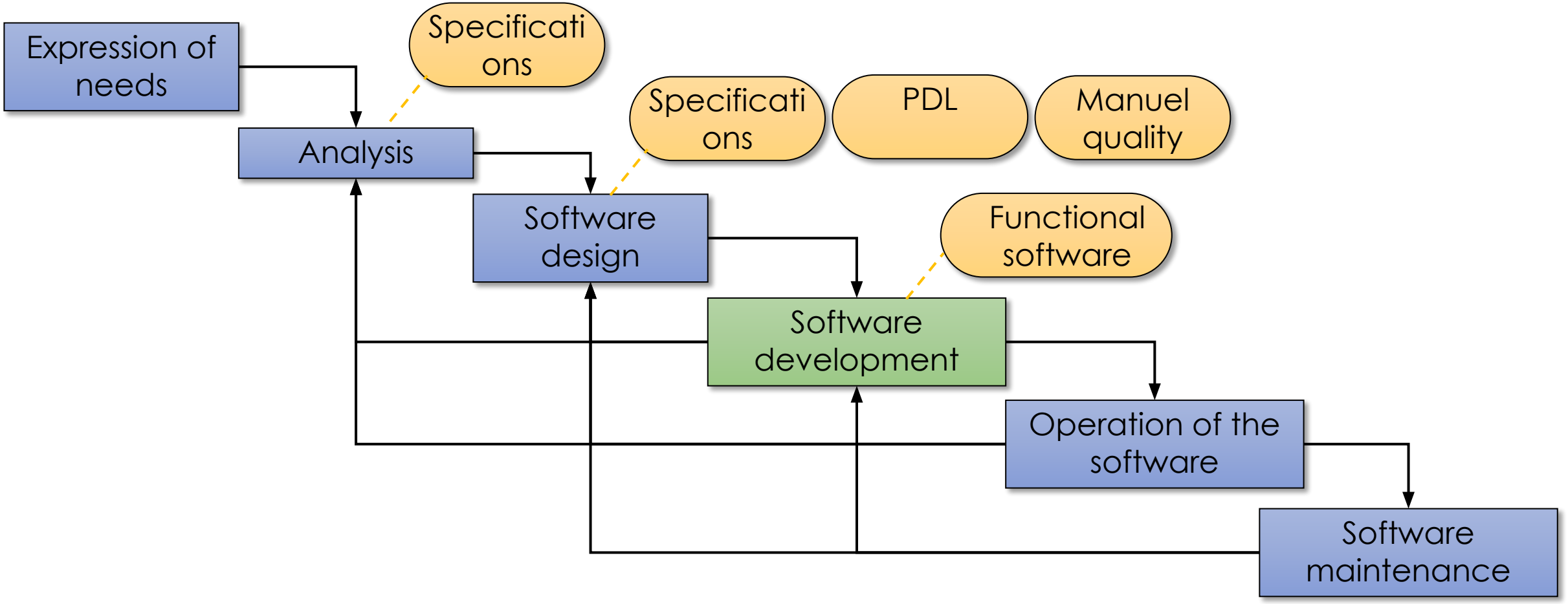
From source to application

Compiled languages

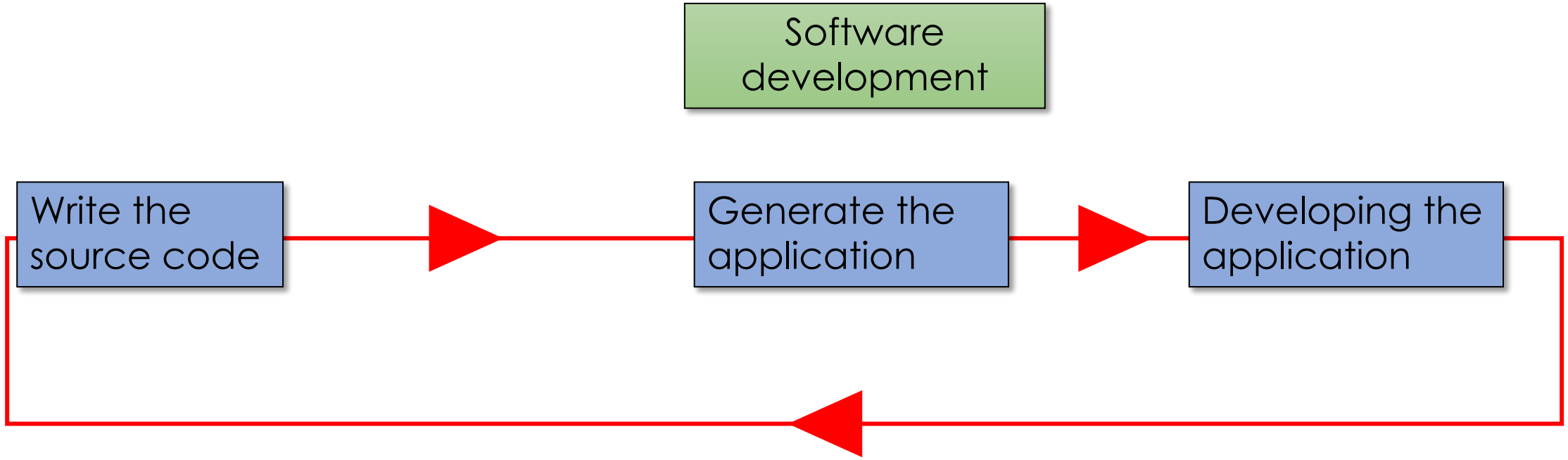


Interpreted languages

The path of the software project



The activities



The starting point

Choice of technical platform (architecture and language)

Design documents - algorithms, classes, modules, interfaces

To Write source files

REWRITE THE SOURCE CODE

Clean and structured code writing

Objective readability - not provided by the language itself

Indentation, syntactic colouring

Rules of good practice

C/C++ : à la carte

Python : PEP8

example

```
#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]));}}
```

This is source code in C language accepted by a compiler.

Taken from :
http://www.ioccc.org/years.html#2018

Writing tools

Syntactic colouring and style :

Code:Blocks editor configuration

Tool LINTER - SpellCheck

With Microsoft Visual Studio for C/C++

With Spyder for Python - pylint tool

Code documentation

Use of comments

Documentation extraction tools

DOXYGEN: example of use in 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);
```

SOFTWARE PRODUCTION

IDE and software production

Integrated Development Environment

Editing and formatting the source code (seen)

Software generation / production (target)

Debug

Version management

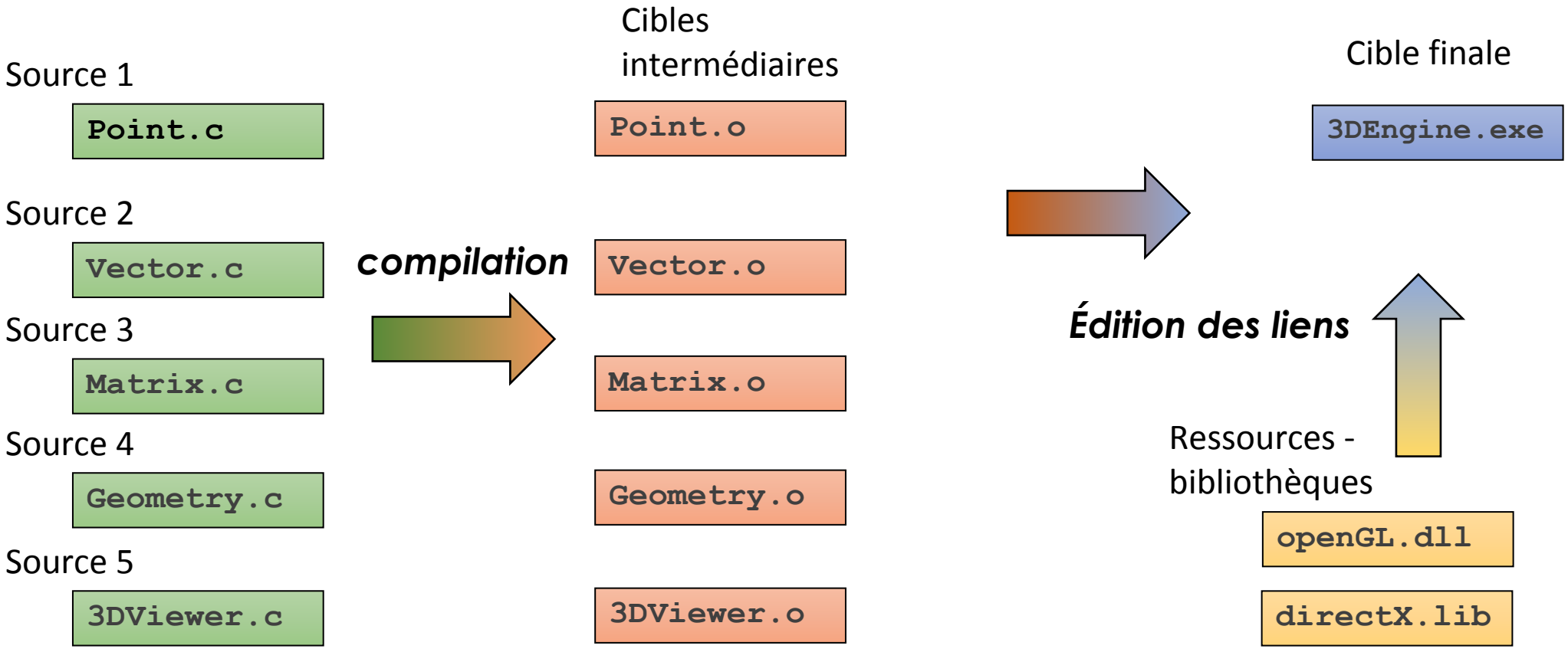
Production line

From source (source) to target (target)

A tool transforms a source into a target

In several stages :
Intermediate target / final target

C/C++ production line

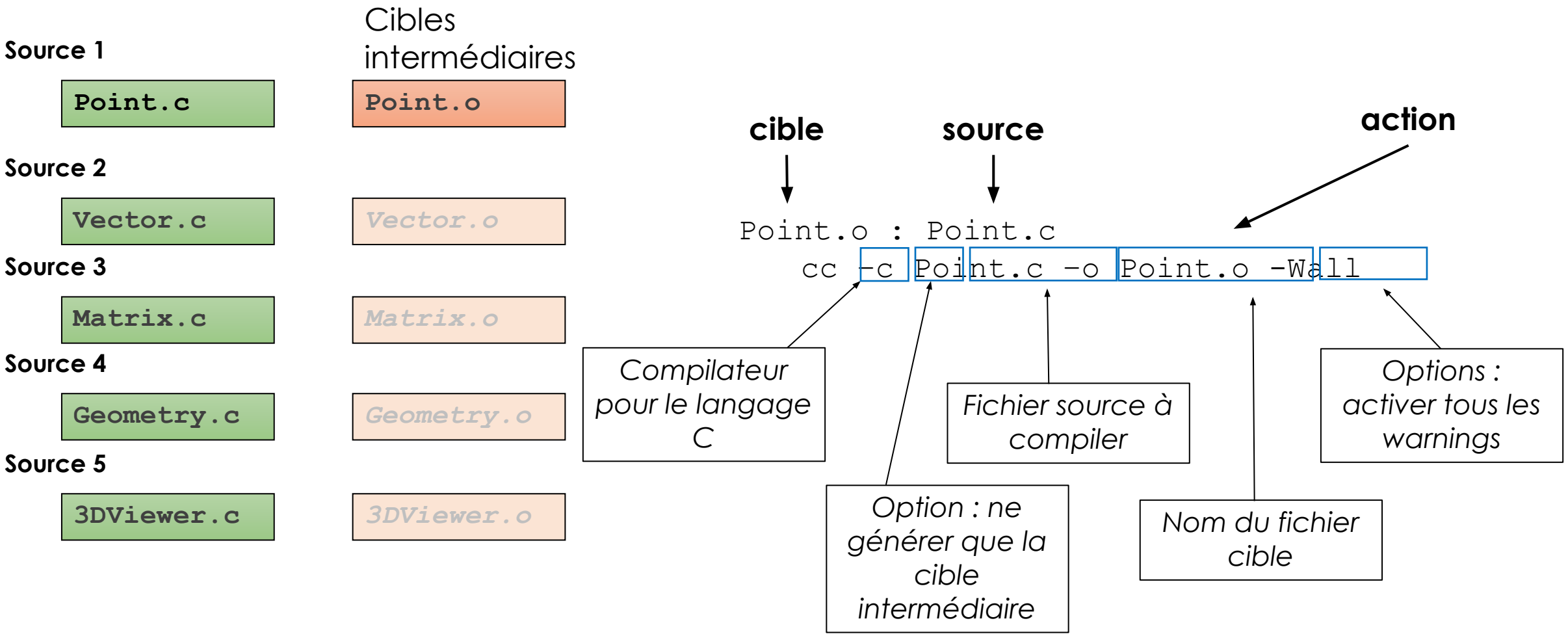


Makefile, dependency

A makefile indicates how to generate targets from a source or dependency, with the format :

```
target : dependency (source)
   action
```

With a makefile



Target/source validity

Timestamping

A target must be generated :

If it does not exist

Or

If it is older than its source / dependency

Example

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
```

Example

Compilation :

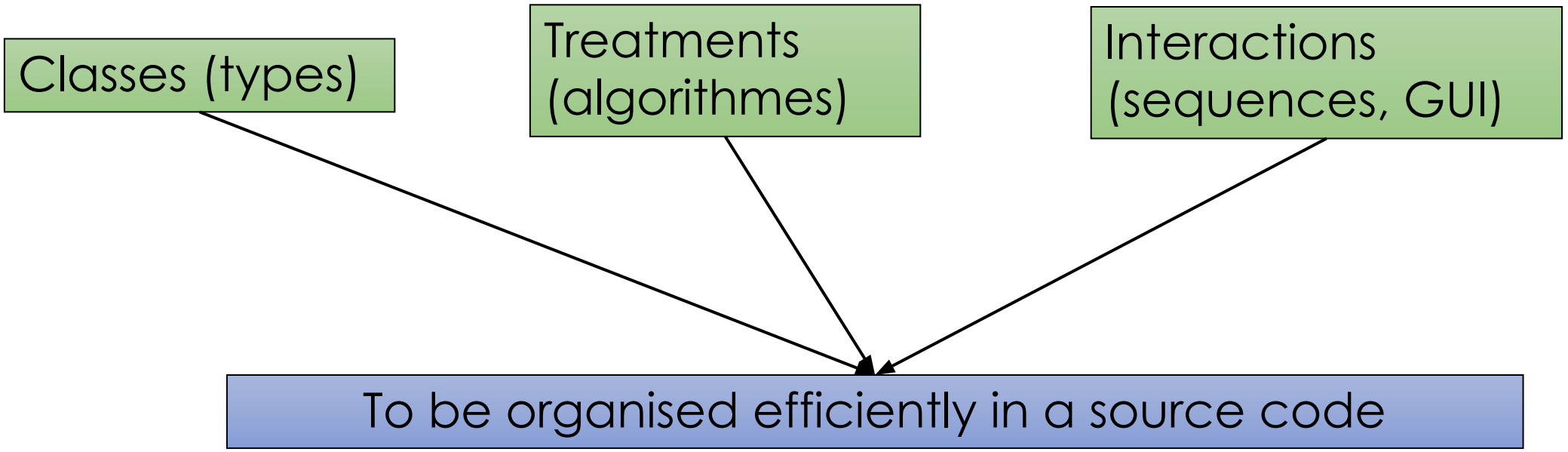
```
% make clean
rm file1.o file2.o executable core
rm: cannot remove `core': No such file or directory
% make
gcc -c file1.c
gcc -c file2.c
gcc -o executable file1.o file2.o
% touch file2.h
% make
gcc -c file2.c
gcc -o executable file1.o file2.o
```

```
% 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
```

MODULARITY OF THE SOURCE CODE

Modularity rules

Design / Modeling of the application



Role of functions and modules

Logical / functional division

- ✔ To organise a development project
- ✔ To exploit the functionalities of a project (API)

An example of API: OGRE3D

Let's have a look at the site:

For development

A module groups functionalities (functions) by theme

A module therefore has a precise architectural role

The same applies to the functions

The same applies to the OO design and the classes

Write a function

4 questions :

What does it do?	name
What does it need?	parameters
What result does it provide?	return value
How does she do it?	internal code

Principle KISS

Keep **I**t **S**imple, **S**tupid

A function does one thing, and does it well

Simple to write, simple to test

Flexibility in focusing

Example: table sorting

Analyze to sequence

Cutting out the problem :

- Create a table
- Fill it in
- Sort it out
- Display the result

With a single function

- Create a table
- Fill it in
- Sort it out
- Display the result

Seems comfortable ☺

No parameters

No return value

Beware of appearances

Role / name not relevant

Code 'that does it all'

Difficult evolution

Fill in the table :

randomly by programme?

by entry?

from a file?

No link with the 'Sort table' part

Divided into separate functions

4 functions (or more)

Array createArray(size)

scanArray(Array)

initArrayFromFile(Array, filename)

sortArray(Array)

displayArray(Array)

saveArrayToFile(Array, filename)

Header files (C/C++)

Separate the what from the how

Things to do in design
If possible in development

Allows to share features (not code)

imports

In C/C++: inclusion of statements `#include`

In Python: importing objects (everything is an `import` object)

Importing only what is necessary

Use of .h files in C/C++

When using a type that is not a basic type

When calling a function

- example with the simplest possible programme
- example with a new type

Rules for inclusion

Never include a .c file

Never

Include an .h file when the compiler needs it function call or use of a new type.

STRUCTURING THE PRODUCTION CHAIN

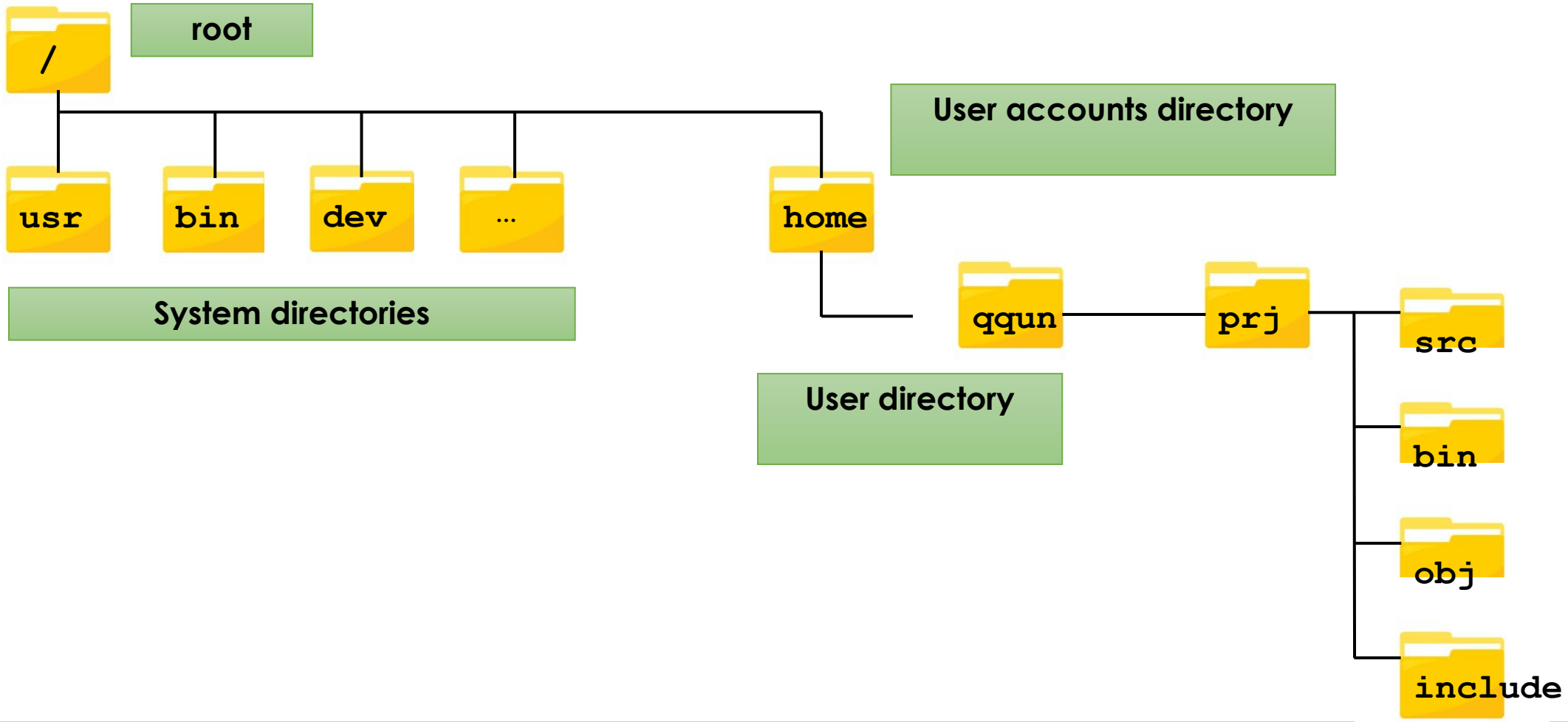
File structure

Classify files according to their type

Example: the Unix file system standard

FHS : Filesystem Hierarchy Standard

Unix – Hierarchical structure



Unix – Hierarchical structure

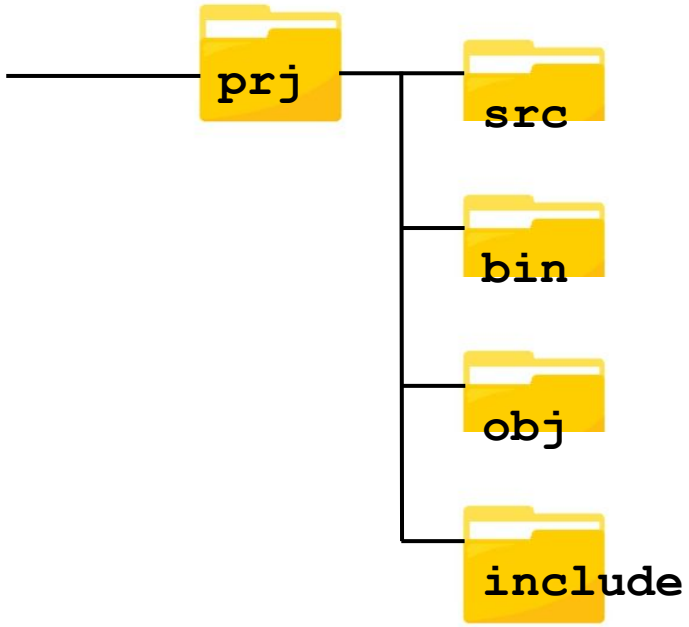
Project directory

Source files

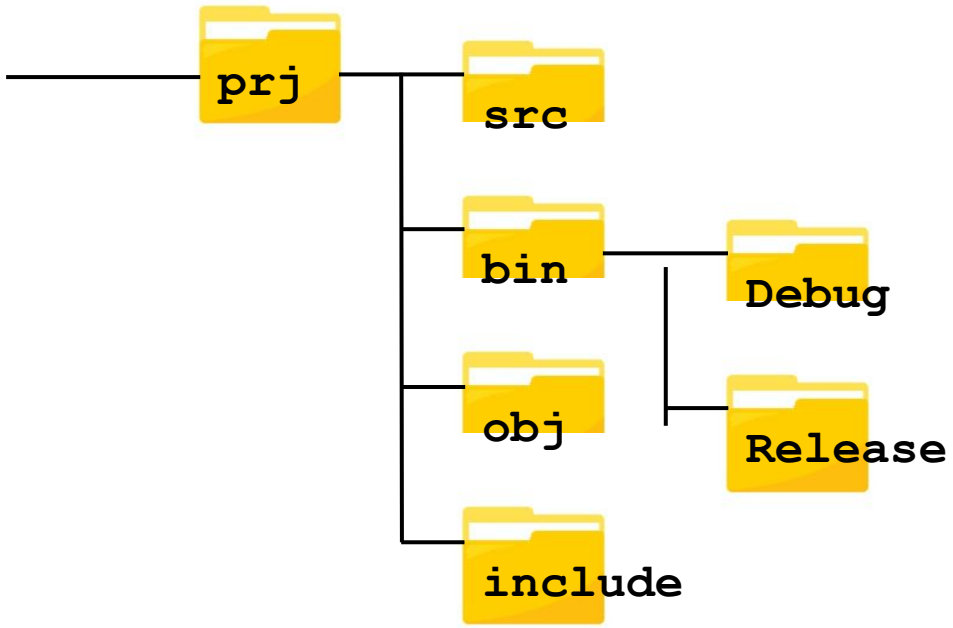
Binary files

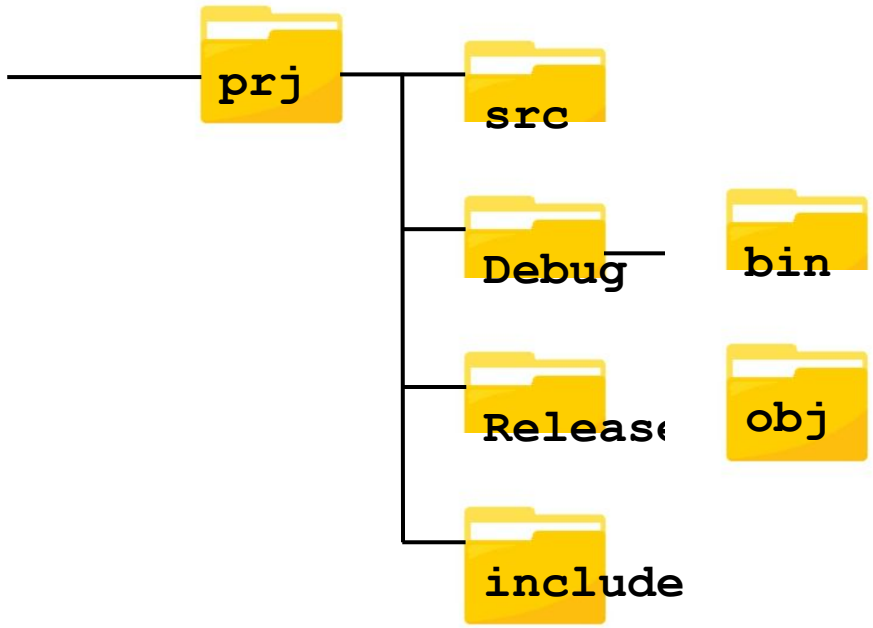
Files .o / .obj

Header files



Project sub-directories





Configuration management

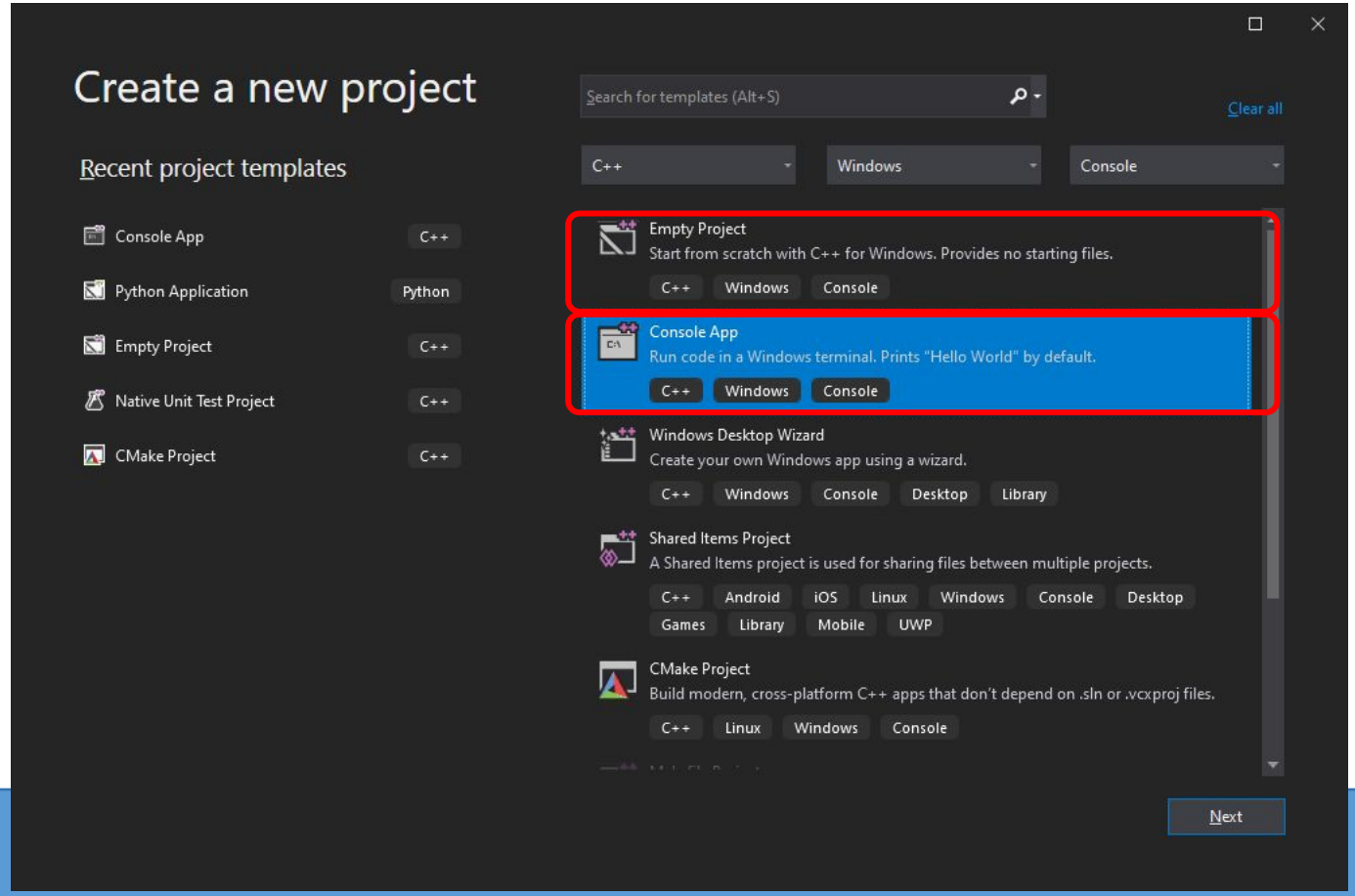
Several types of target

From source sets and subsets

Configuration of generation rules

Types of target

In C: list of Microsoft Visual Studio project types



An example :

Maintain with a single set of source code :

A production executable **Release**

A focusing executable **Debug**

An executable for testing **Test**

An API library **lib**

SOFTWARE DEVELOPMENT CYCLE

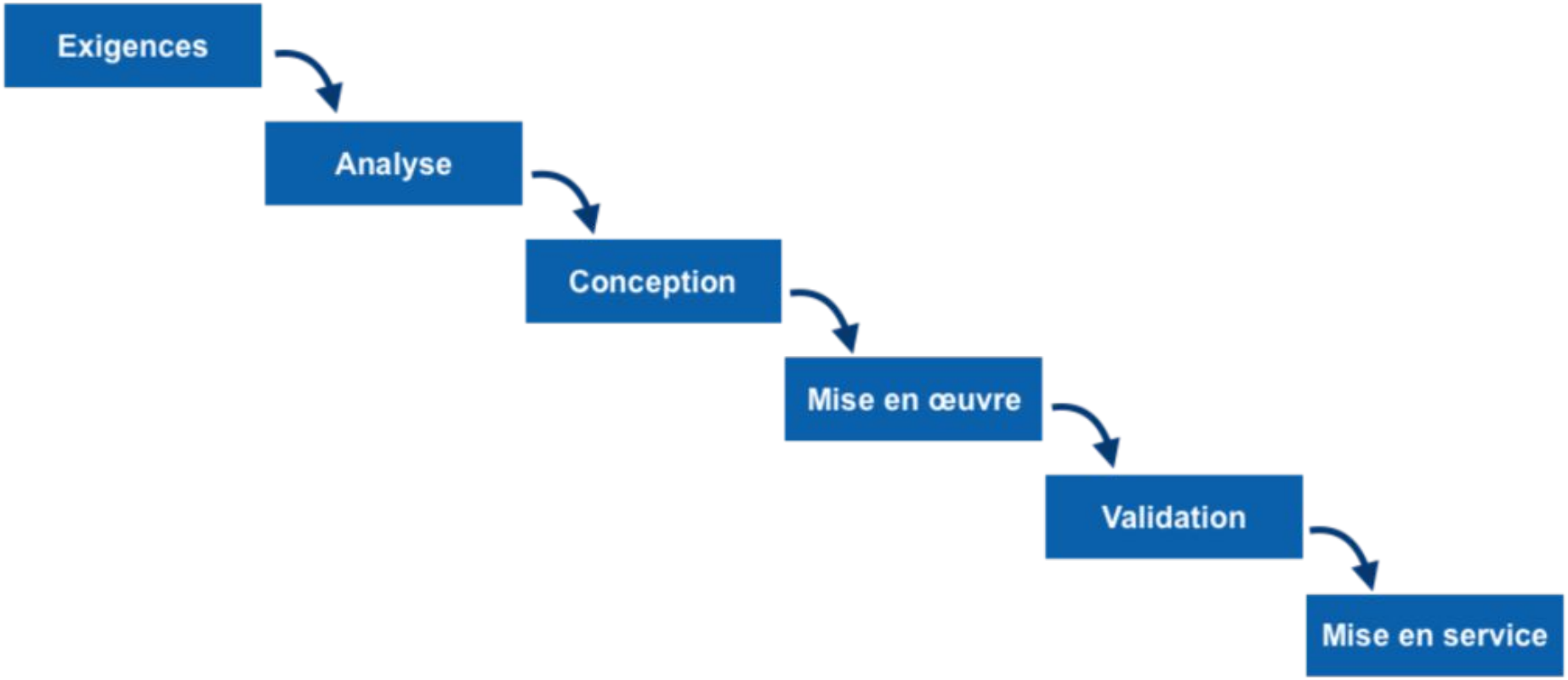
Cascade model (1970)

Organisation in the form of linear and sequential phases

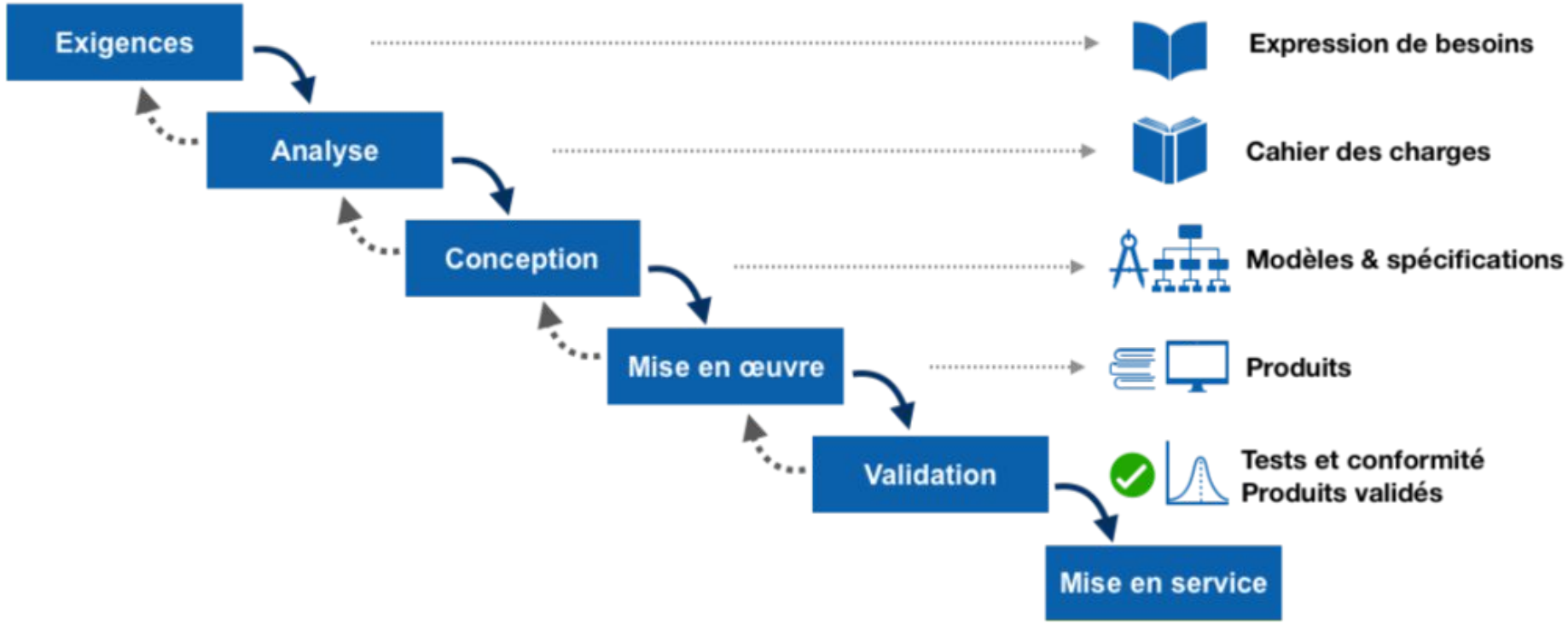
Phase ->

specialisation of tasks and depends on the results of the previous phase

Cascade model



Cascade model

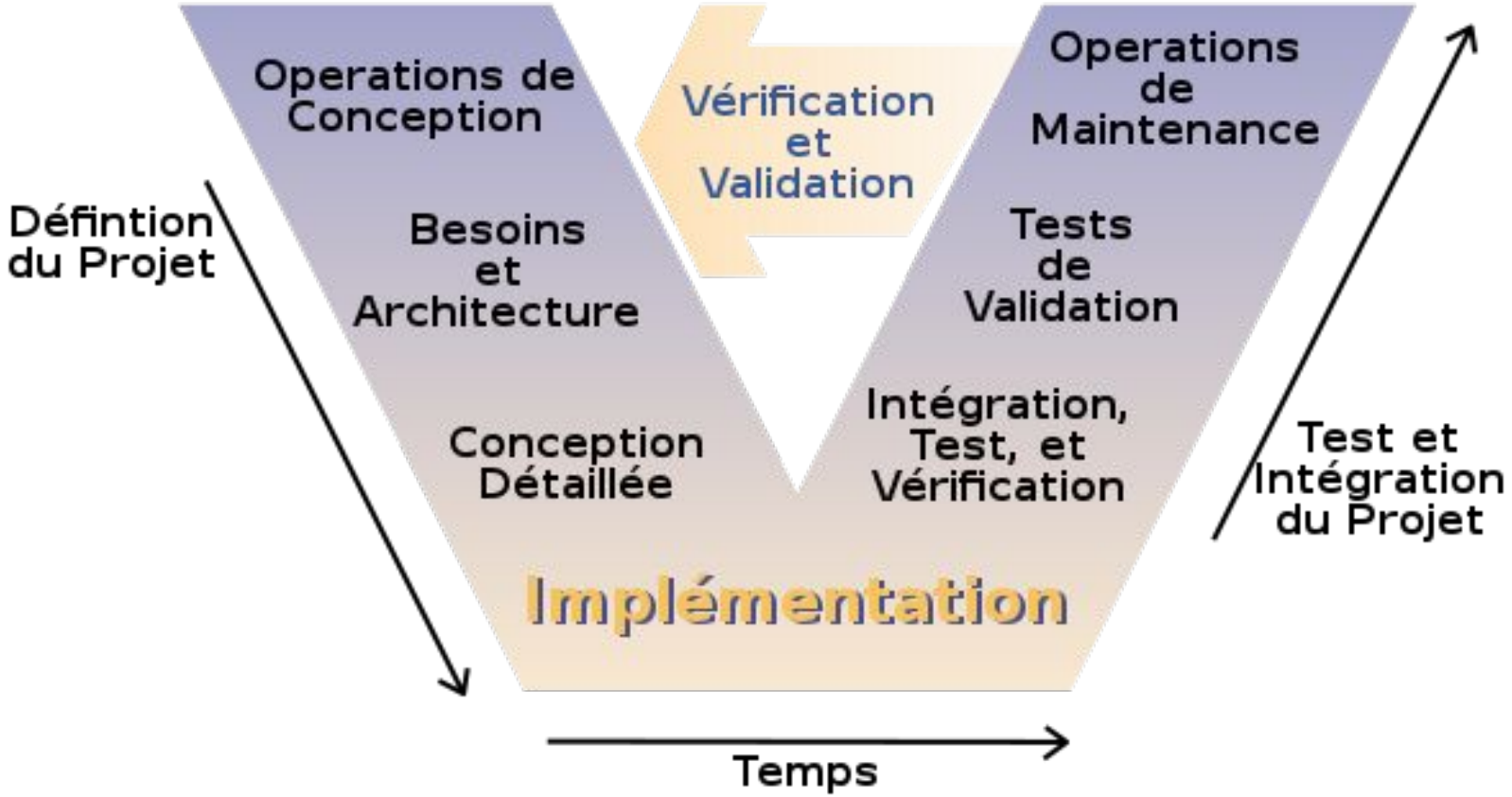


V-cycle

- top-down activity flow: from product to completion
- upward flow: assembling the product by checking its quality

Derived from the cascade model sequential and linear approach of phases + system integration activities and compares each successive production phase with its corresponding validation phase

V-cycle


Operations de Conception
Vérification et Validation
Operations de Maintenance
Défintion du Projet
Besoins et Architecture
Tests de Validation
Conception Détaillée
Intégration, Test, et Vérification
Test et Intégration du Projet
Implémentation
Temps

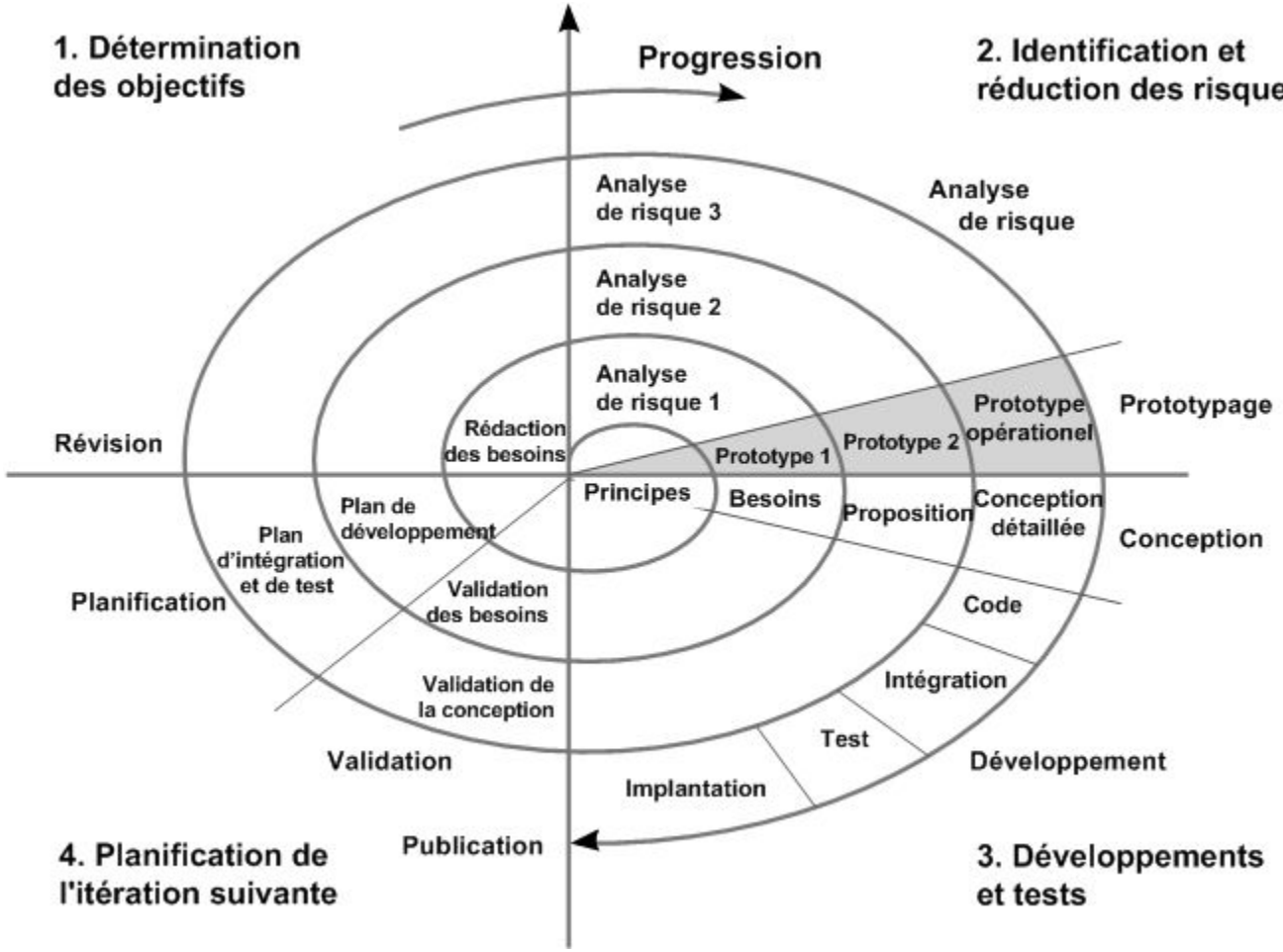
Spiral or incremental model (1988)

Response to the disadvantages of the cascade model

Repeated spirals until the finished product can be delivered

The product is continuously worked on and improvements are often made in small steps.

Spiral or incremental model



Agile method (2001) : definition

- steering practice groups and project implementation
- reference of multiple existing methods
- involves the applicant (client) as much as possible and allows for a high level of reactivity to his requests
- iterative, incremental and adaptive development cycle

Agile method (2001) : implémentation

Scrum (1995/2001): continuous improvement

XP extrem programming (1999): immediate process reengineering

=> lean management

Continuous improvement

Checks each time the source code is modified to ensure that the result of the modifications does not produce regression in the developed application.

The aim of this practice is to detect integration problems as early as possible in the development process.

Automate the execution of test suites and see the evolution of software development

Scrum VS V-Cycle

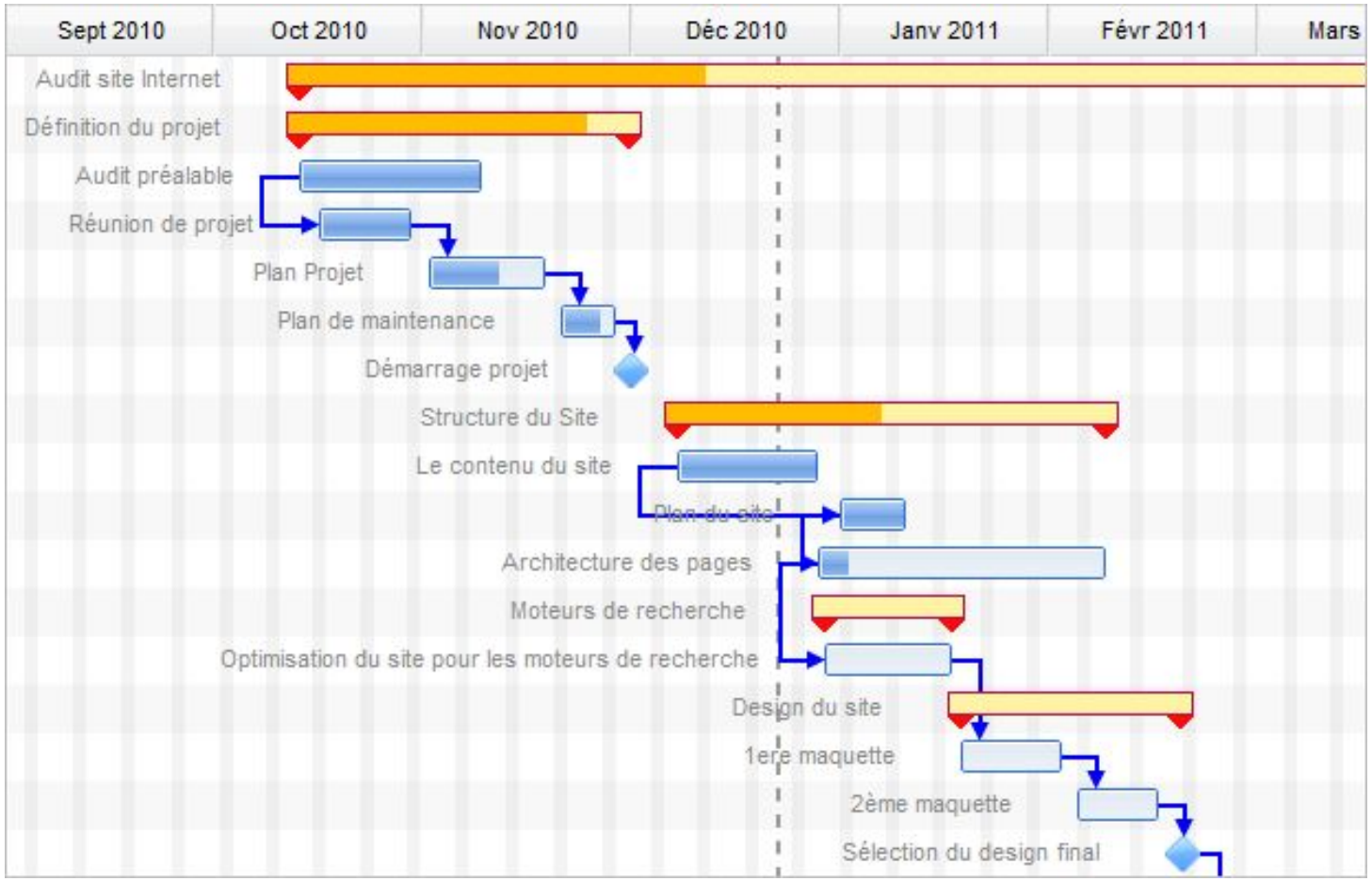
Theme	V-Cycle	Scrum
Life cycle	Sequential phases	Iterative process
Delivery	At the end of the completion of all → features late delivery	Partial use of the product due to prioritisation of needs → faster delivery
Quality control	On final delivery (end of development cycle) → tunnel effect	For each partial delivery to the customer
Specification	No change possible without going back to the specification phase and going through all the other phases → additional time and costs	More flexible specifications by adding/changing functionality to the following sprints that were not originally planned → main advantage of the Agile method
Planning	Detailed plans based on stable requirements defined at the start of the project	Adaptive planning and adjustments if necessary to meet new demands
Team	Intervention only in the development phase, no global vision of the project	Commitments, exchanges and collective decision-making by the team
Documentation	Large quantity	Strictly necessary

Gantt chart

Gantt + PERT network (scheduling, project management)
=> allows you to view the various tasks making up a project over time.
Objectives: to plan optimally and to communicate on the established schedule and the choices it imposes.

- to determine the dates for carrying out a project
- to identify the existing margins on certain tasks
- to visualise the delay or progress of the work.

Gantt chart


Sept 2010
Oct 2010
Nov 2010
Déc 2010
Janv 2011
Févr 2011
Mars
Audit site Internet
Définition du projet
Audit préalable
Réunion de projet
Plan Projet
Plan de maintenance
Démarrage projet
Structure du Site
Le contenu du site
Plan du site
Architecture des pages
Moteurs de recherche
Optimisation du site pour les moteurs de recherche
Design du site
1ere maquette
2ème maquette
Sélection du design final

Kanban method

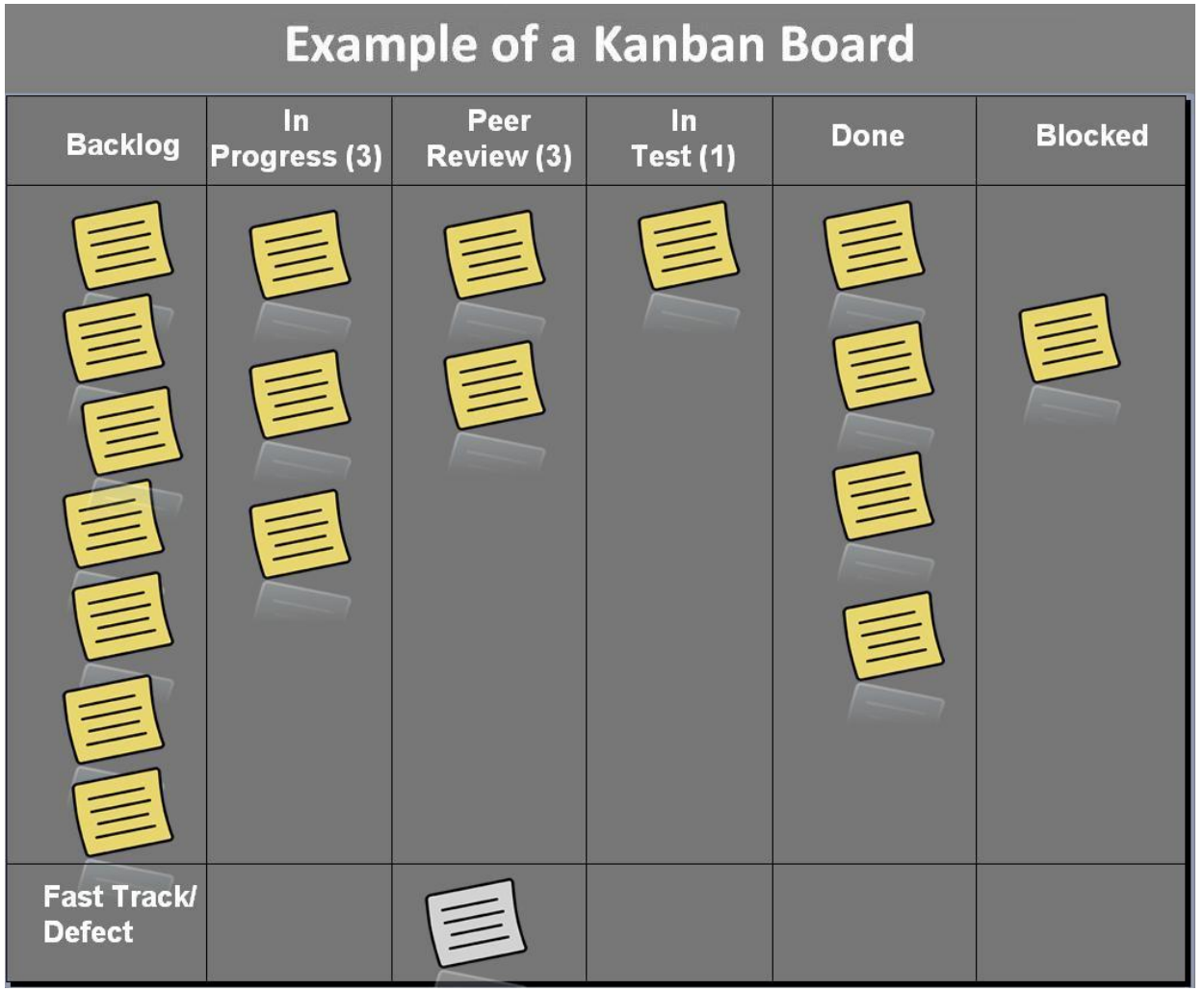
A method of knowledge management with a just-in-time organisation by providing information to team members in a timely manner so as not to overburden them

Complete process: from job analysis to delivery to the client is available to all participants, each taking their tasks from a queue.

A visual process management system that indicates what to produce, when to produce it and in what quantity.

The approach is inspired by the Toyota production system and lean methods.

Kanban method



DevOps (2007)

Unification of software development and system administration of IT infrastructures

Short development cycles, increased frequency of deployments and continuous deliveries

Automation and monitoring of the :

- development
- integration
- tests
- delivery and deployment
- operation
- infrastructure maintenance

Choice of development cycle

- Depends on various factors :
- scope of the project
- budget
- required level of support and maintenance
- flexibility => agile methods

Good development practices

- Programming in pairs (https://fr.slideshare.net/YvesHanoulle/pair-programming-is-like-sex)

- Burndown chart

- Planning poker