



# 01

## Chapitre

# Introduction au génie logiciel

**1I2AC1 : Génie logiciel et Conception orientée objet**

Régis Clouard, ENSICAEN - GREYC

« Si les ouvriers construisaient les bâtiments  
comme les développeurs écrivent leurs programmes,  
le premier pic-vert venu aurait détruit toute civilisation. »

**Gerald Weinberg**

# Organisation de l'enseignement

---

2

- Volume horaire
  - CM : 11 h
  - TD : 14 h
  - Pas de TP, mais
    - ▶ Lien avec le cours « *Java et Programmation objet* »
    - ▶ Katas et Coding dojos
- Plateforme pédagogique (course n°60)
  - Ressources du cours
    - ▶ Polycopié par chapitre
    - ▶ Présentation par CM
    - ▶ Katas et Coding Dojos

# Organisation de l'enseignement

---

## ■ Examen

- Le cahier de TD tient lieu d'annales d'examen
- Il portera sur tout le polycopié (*les diapositives n'en présentent qu'une partie*)
- **Document autorisé : une feuille A4 recto/verso manuscrite**

## ■ Discipline

- Pas d'appel
- Respect de l'enseignant !

# Question du jour

4

- Qu'est ce que le génie logiciel ?
  - « *Le génie logiciel est une science de génie industriel qui étudie les méthodes de travail et les bonnes pratiques des ingénieurs qui développent des logiciels.* »  
Wikipedia

# Plan du chapitre

---

1

Pourquoi  
un cours sur le  
génie logiciel ?

# Pourquoi un cours sur le génie logiciel ?

---

6

- La spécialité informatique de l'ENSICAEN forme des **ingénieurs développeurs logiciels**
  - *alias* **Architectes logiciels**
  - Ce sont des professionnels qui
    - ▶ conçoivent
    - ▶ déploient les systèmes logiciels
    - ▶ maintiennent
    - ▶ administrent
- Le génie logiciel est un enseignement de base pour les métiers liés au logiciel

# Pourquoi un cours sur le génie logiciel ?

---

7

- Pourquoi ne pas se contenter d'un cours de programmation pour développer des logiciels ?

# Question

- Soit le programme suivant d'addition de deux nombres entiers :

```
#include <stdio.h>
/** This program displays the result of adding of 2 integers */
int main() {
    int i1, i2; // The 2 integer operands
    scanf("%d", &i1); // Read the 2 integers
    scanf("%d", &i2);
    printf("Résultat = %d\n", i1 + i2); // Display
    return 0;
}
```



- Est-ce un bon code de programme ?

# Question

- Soit le programme d'addition de deux nombres entiers :

```
#include <stdio.h>
/** This program displays the result of adding of :
int main() {
    int i1, i2; // The 2 integer operands
    scanf("%d", &i1); // Read the 2 integers
    scanf("%d", &i2);
    printf("Résultat = %d\n", i1 + i2); // Display
    return 0;
}
```

Code: V 20  
V Lacunaire  
2/ Dangereux  
3/ Faux  
4/ Sale  
5/ mal structuré

- Est-ce un bon code de logiciel ?

# Confusion

## programmation / développement

---

10

- Analogie avec le génie civil

# Confusion programmation / développement

---

11

- Programmation (Maçonnerie) : accessible à tous



# Confusion programmation / développement

---

12

- Programmation (Maçonnerie) : accessible à tous



# Confusion

## programmation / développement

---

13

- Programmation (Maçonnerie) : accessible à tous



# Confusion programmation / développement

---

14

- Développement (Architecture) : réservé aux professionnels



# Confusion programmation / développement

---

15

- Développement (Architecture) : réservé aux professionnels



# Confusion programme / logiciel

---

16

- Les profanes créent des **programmes**
- Les développeurs créent des **logiciels**
  
- Quelles sont les différences entre programme et logiciel ?

# Différences programme / logiciel

---

17

## 1. Utilisateur

- Programme : un utilisateur averti et bienveillant
- Logiciel : l'utilisateur est un « client »

## 2. Portabilité

- Programme : un OS
- Logiciel : tous les OS

## 3. Complexité

- Programme : résident sur 1 seul nœud
- Logiciel : réparti sur le réseau

## 4. Taille des sources

# Différences programme / logiciel :

## La taille des sources

---

18

- Unité de mesure de la taille d'un logiciel :
  - **LOC** : lines of code
    - ▶ 1 MLOC :  $10^6$  LOC

# Différences programme / logiciel :

## La taille des sources

---

19

- **Programme**
  - Quelques KLOC
- **Logiciel**
  - Commandes de vol A380 : **1 MLOC**
  - OS Android : **15 MLOC**
  - Linux kernel 5.8 (2020): **53 MLOC**
  - Facebook : **62 MLOC**
  - Windows 10: **80 MLOC**
  - Google (tous les services Internet) : **2GLOC**

# Conséquence de la taille : Développement en équipe

---

20

- La taille des logiciels oblige à un travail en équipe
  - Unité de mesure :
    - ▶ **année-homme** (man-year)
    - ▶ ou mois-homme
    - ▶ ou heure-homme ...
  - Par exemple :
    - ▶ Algorithme de recherche de Google est estimé à 1 000 années-hommes

# Conséquence de la taille :

## Coût de développement

- La taille du logiciel entraîne des temps de développement et des coûts élevés
  - Combien de temps pour développer ce bout de code ?
  - Combien facturer ce bout de code ?

```
static void sort( int[] array ) {  
    for (int i = 0; i < array.length - 1; i++) {  
        for (int j = 0; j < array.length - 1 - i; j++) {  
            if (array[j] > array[j + 1]) {  
                int temp = array[j];  
                array[j] = array[j+1];  
                array[j+1] = temp;  
            }  
        }  
    }  
}
```

# Conséquence de la taille :

## Coût de développement

22

- Ordre de grandeur en France :
  - 1 année-homme  $\approx$  1650 h
  - 1h ingénieur  $\approx$  50 €
  - Productivité  $\approx$  2 à 5 LOC/h (4 à 9 KLOC /an)

- Donc, le code coûte :
  - Temps : 1 heure-homme
  - Prix : 50 €

```
static void sort( int[] array ) {
    for (int i = 0; i < array.length - 1; i++) {
        for (int j = 0; j < array.length - 1 - i; j++) {
            if (array[j] > array[j + 1]) {
                int temp = array[j];
                array[j] = array[j+1];
                array[j+1] = temp;
            }
        }
    }
}
```

# Différences programme / logiciel

---

23

1. Utilisateur
2. Portabilité
3. Complexité
4. Taille des sources
5. Responsabilité des dysfonctionnements

# Différences programme / logiciel :

## La responsabilité des dysfonctionnements 24

---

### ■ Programme

- L'utilisateur accepte les conséquences des dysfonctionnements lors de l'utilisation du programme

### ■ Logiciel

- Les développeurs sont tenus responsables par les utilisateurs des conséquences néfastes de l'utilisation du logiciel.
- Les développeurs doivent proscrire toutes les conséquences néfastes de l'utilisation du logiciel :
  - ▶ Perte de données
  - ▶ Résultats erronés
  - ▶ Vol de données
  - ▶ Utilisation frauduleuse

# Conséquence comique d'un dysfonctionnement

25

- Windows 95



# Conséquence préoccupante d'un dysfonctionnement

---

26

- Bug sur les serveurs Microsoft Exchange.
  - Les utilisateurs n'avaient plus accès à leur mail suite au changement d'année 2022
  - Le moteur d'analyse des malwares était bugué.
    - ▶ Codage des dates sur un entier signé (31 bits)
    - ▶ Or  $2^{31} = 2\,147\,483\,648$  ne peut pas coder les dates postérieures à 2021
      - 01/01/2022 est codé par 2 201 010 001

# Conséquence tragique d'un dysfonctionnement

---

27

- Mort tragique d'une patiente de 72 ans à l'hôpital de Versailles en novembre 2011
  - Son allergie à un antibiotique, l'amoxicilline, était bien notée dans son dossier médical, mais le logiciel utilisé pour les prescriptions n'a pas intégré cette donnée.
  - → Ne jamais ajouter d'élément dans l'UI qui laisse à penser que la fonctionnalité est implémentée

# Différences programme / logiciel

---

28

1. Utilisateur
2. Portabilité
3. Complexité
4. Taille des sources
5. Responsabilité des dysfonctionnements
6. Maintenance

# Différences programme / logiciel :

## Maintenance

---

29

- Un logiciel ne s'use pas ... mais il se détériore
  - Pas de logiciel sans maintenance
  - Sans maintenance un logiciel a une durée de vie estimée de 6 mois
  - La maintenance doit être préparée dès la conception
  - L'expérience montre même que la majeure partie du travail de développement commence après la livraison du logiciel au client.

# Conclusion en 1968

- Un programme peut se réaliser de façon empirique
- Un logiciel ne peut pas se développer sans une méthode de gestion de projet et un haut niveau d'expertise reposant sur le génie logiciel
  - Projet étalé sur plusieurs mois
  - Projet nécessitant la collaboration de plusieurs personnes
  - Tâche complexe et compliquée
  - La programmation ne représente qu'une très petite partie du logiciel
- Développeur logiciel :
  - Un métier à haut niveau d'expertise
  - Un métier qui s'apprend
  - L'auto-formation donne des programmeurs

# Plan du chapitre

---

1

Pourquoi  
un cours sur le  
génie logiciel ?

2

Création  
du génie logiciel

# Génie logiciel (1968)

32

- Le terme anglais « *software engineering* » a été inventé par une pionnière du génie logiciel : **Margaret Hamilton**
  - Responsable de la partie logicielle embarquée du projet Apollo de la NASA
  - Le premier ingénieur en génie logiciel est une ingénieure



Margaret Hamilton

# Génie logiciel (1968)

---

33

- Génie

- Ensemble de pratiques **régulées** et basées sur des principes **scientifiques** et **économiques** (+ RSE et DD)

- ➔ Génie logiciel

- Le génie logiciel postule donc que l'on peut appliquer le génie au logiciel

- Source d'inspiration : le **génie civil**

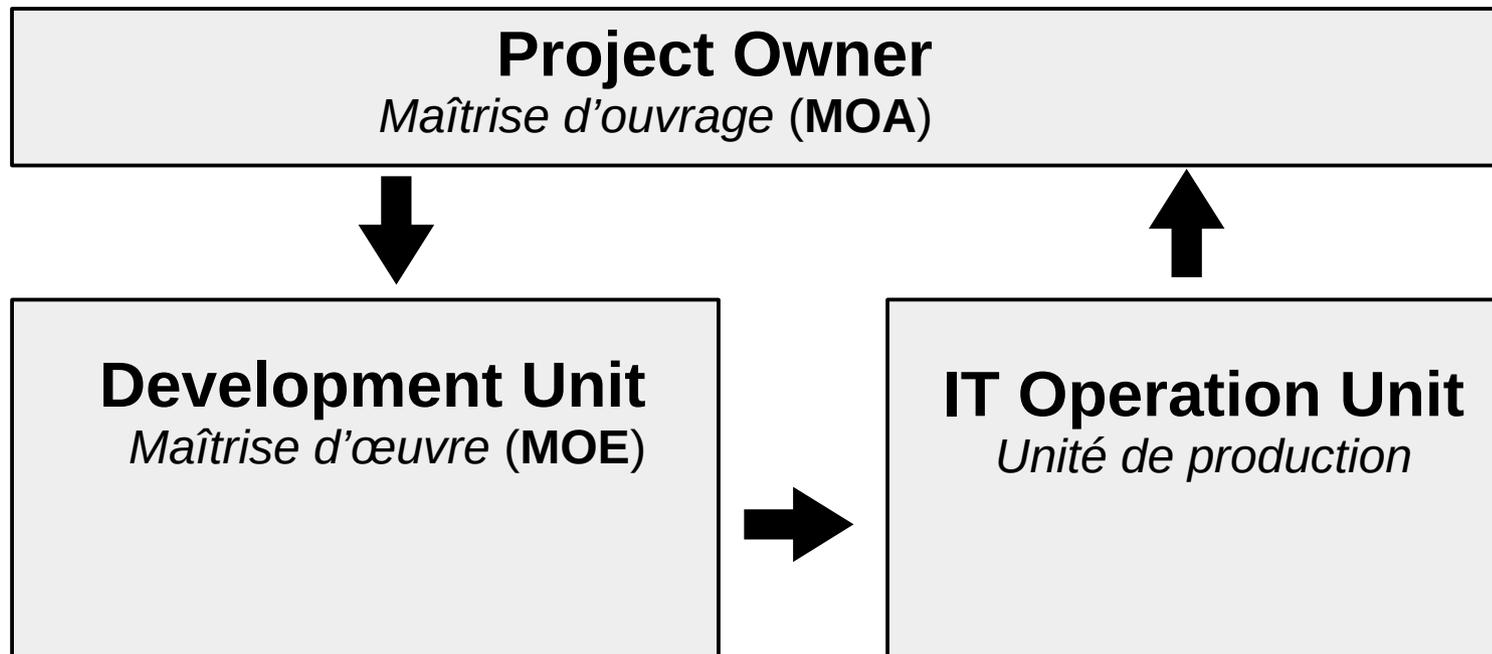
- Le génie civil a démontré toute son efficacité depuis des millénaires

- La gestion de projet selon le génie civil :
  - Découper le temps du projet en une suite d'étapes séquentielles
    - ▶ *Livrable : diagramme de Gantt / Pert*
  - Ne faire qu'une seule chose à la fois à chaque étape
    - ▶ *Métiers spécifiques*
  - Récolter le besoin avant de le réaliser
    - ▶ *Livrable : cahier des charges*
  - Bien réfléchir avant d'agir
    - ▶ *Livrable : documentation de conception et de réalisation*

# Génie logiciel

- Les éléments de base du génie civil appliqué au génie logiciel :
  - **Une méthode de gestion de projet**
    - ▶ Planifier une organisation du temps pour le travail en équipe
  - **Un paradigme de conception**
    - ▶ Fournir les briques de base de la conception et les mécanismes pour les assembler
  - **Un formalisme de modélisation**
    - ▶ Fournir un langage formel pour parler du code de manière du code abstraite et non ambiguë

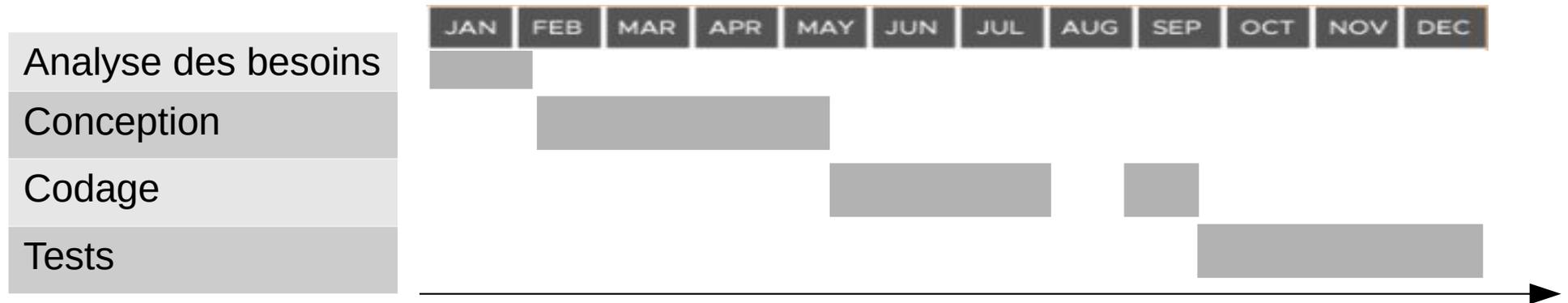
# Écosystème du développement logiciel



# 1/ Une méthode : Le cycle en cascade

37

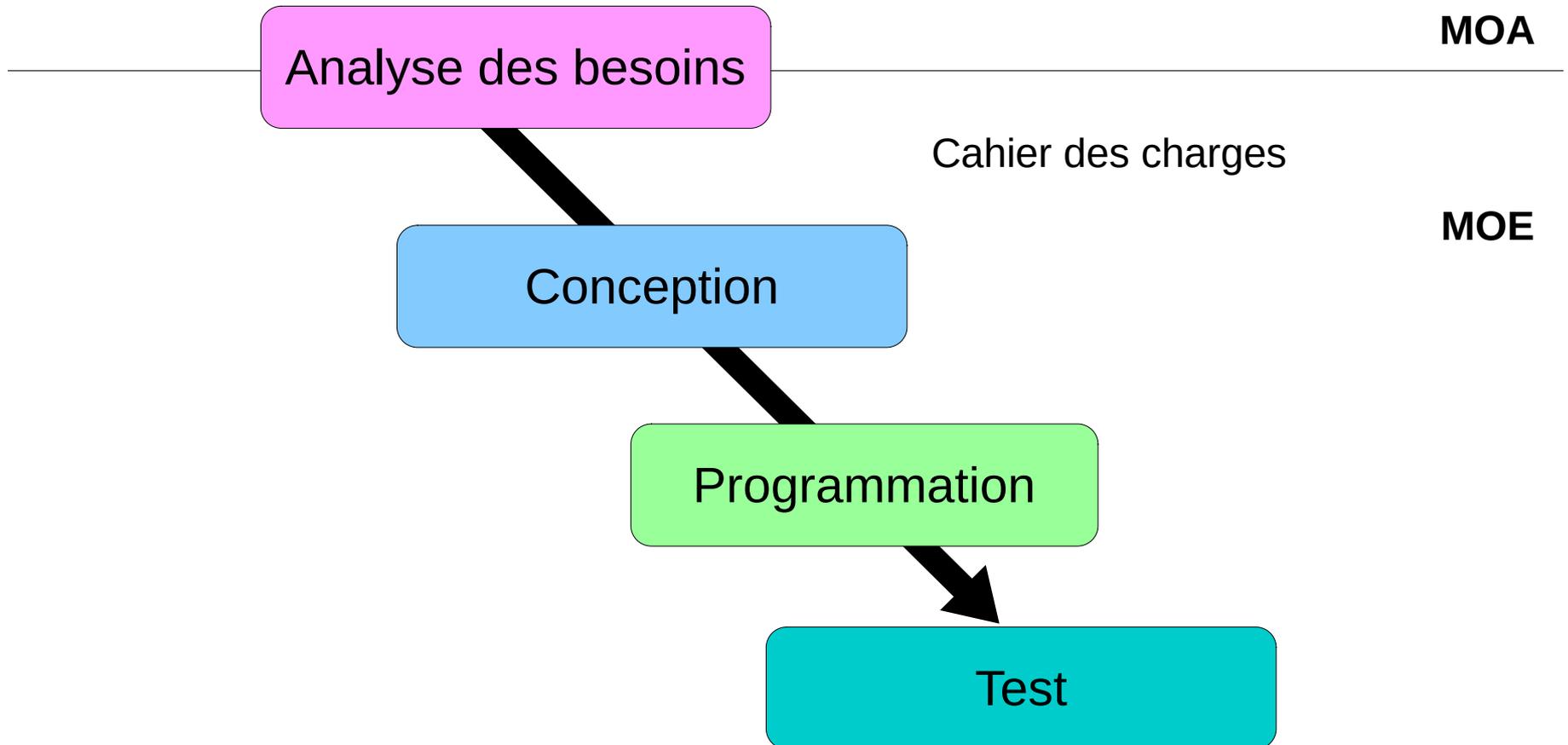
- *Découper le temps du projet en étapes séquentielles*



# 1/ Une méthode : Le cycle en cascade

38

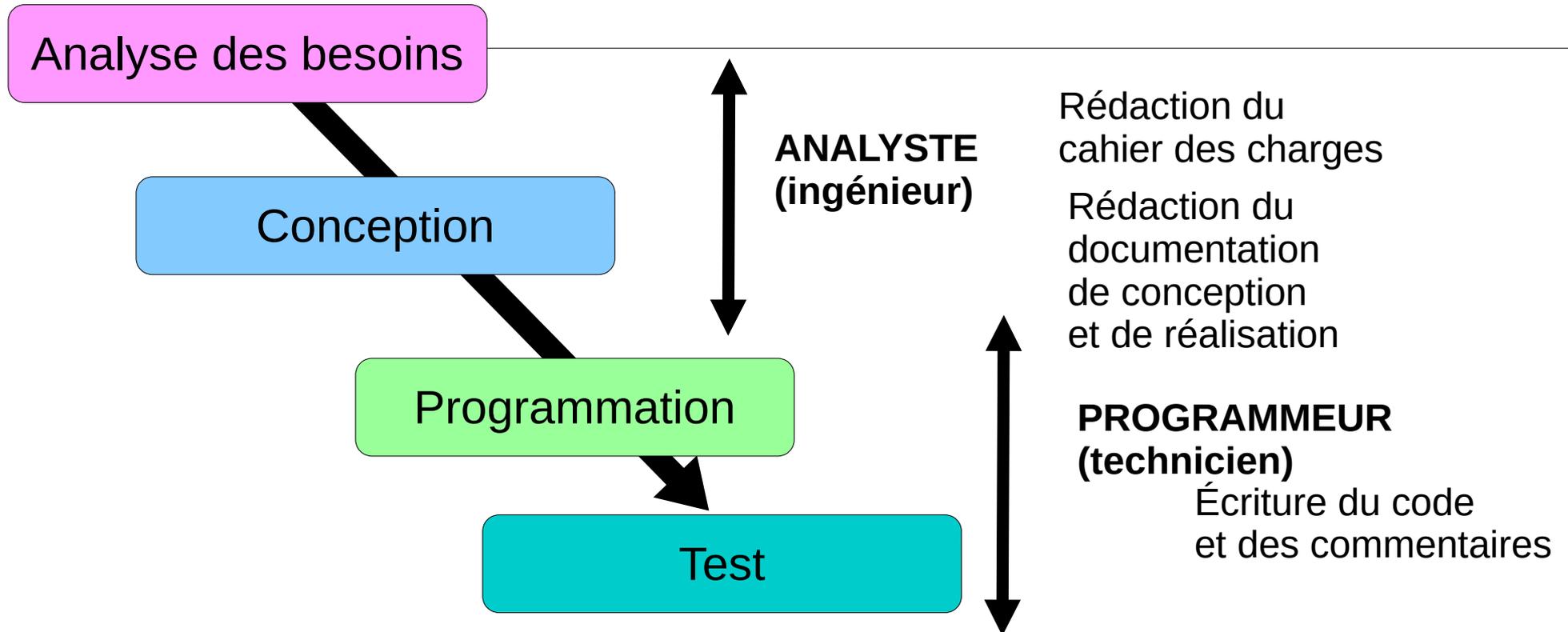
- *Récolter le besoin avant de la réaliser*



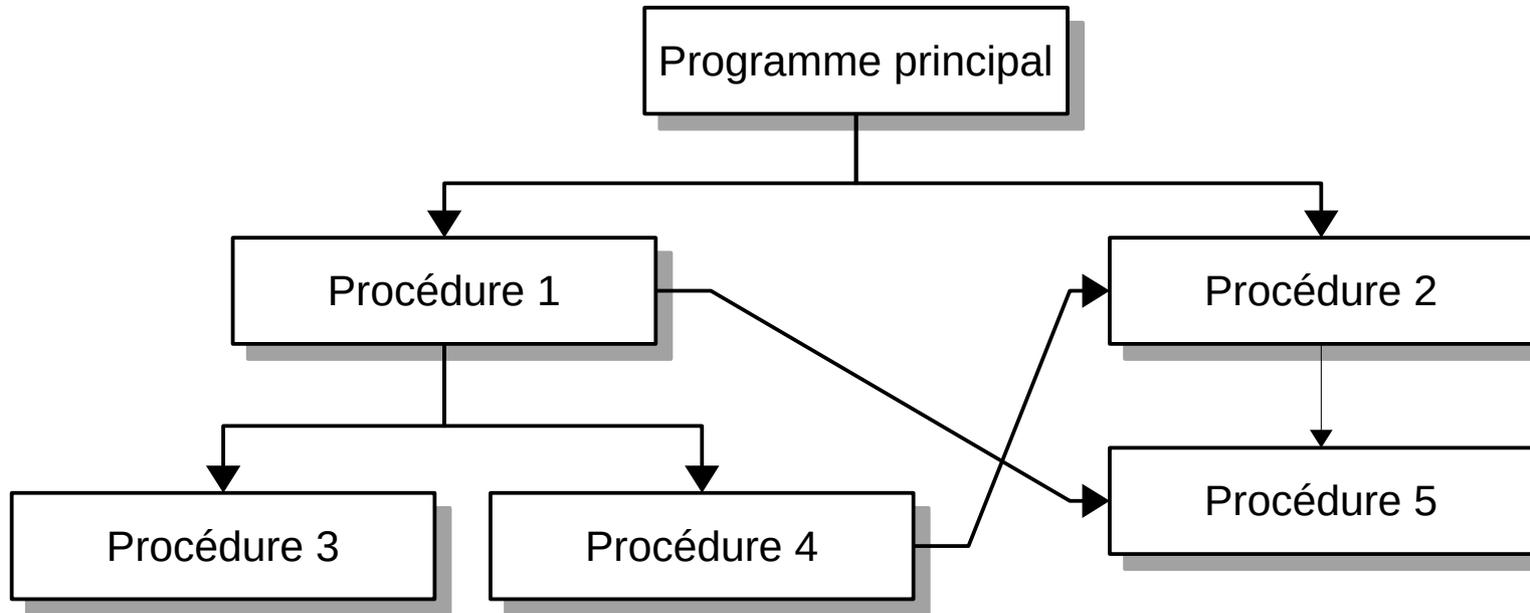
# 1/ Une méthode : Le cycle en cascade

39

- *Ne faire qu'une seule chose à chaque étape*
  - Il en résulte deux métiers dans la MOE :



# 2/ Un paradigme : La conception procédurale



# 3/ Un formalisme : L'algorithmique

41

- *Bien réfléchir avant d'agir*

INSERTION-SORT( <i>A</i> )	<i>cost</i>	<i>times</i>
1 <b>for</b> <i>j</i> = 2 <b>to</b> <i>A.length</i>	$c_1$	$n$
2 <i>key</i> = <i>A</i> [ <i>j</i> ]	$c_2$	$n - 1$
3     // Insert <i>A</i> [ <i>j</i> ] into the sorted sequence <i>A</i> [1.. <i>j</i> - 1].	0	$n - 1$
4 <i>i</i> = <i>j</i> - 1	$c_4$	$n - 1$
5 <b>while</b> <i>i</i> > 0 and <i>A</i> [ <i>i</i> ] > <i>key</i>	$c_5$	$\sum_{j=2}^n t_j$
6 <i>A</i> [ <i>i</i> + 1] = <i>A</i> [ <i>i</i> ]	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7 <i>i</i> = <i>i</i> - 1	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8 <i>A</i> [ <i>i</i> + 1] = <i>key</i>	$c_8$	$n - 1$



Ada Lovelace  
vers 1843

- Le premier programmeur est une programmeuse :  
Ada Lovelace

- Algorithme de calcul des nombres de la suite de Bernoulli  $\sum_{k=0}^{n-1} k^m$

# Plan du chapitre

---

1

Pourquoi  
un cours sur le  
génie logiciel ?

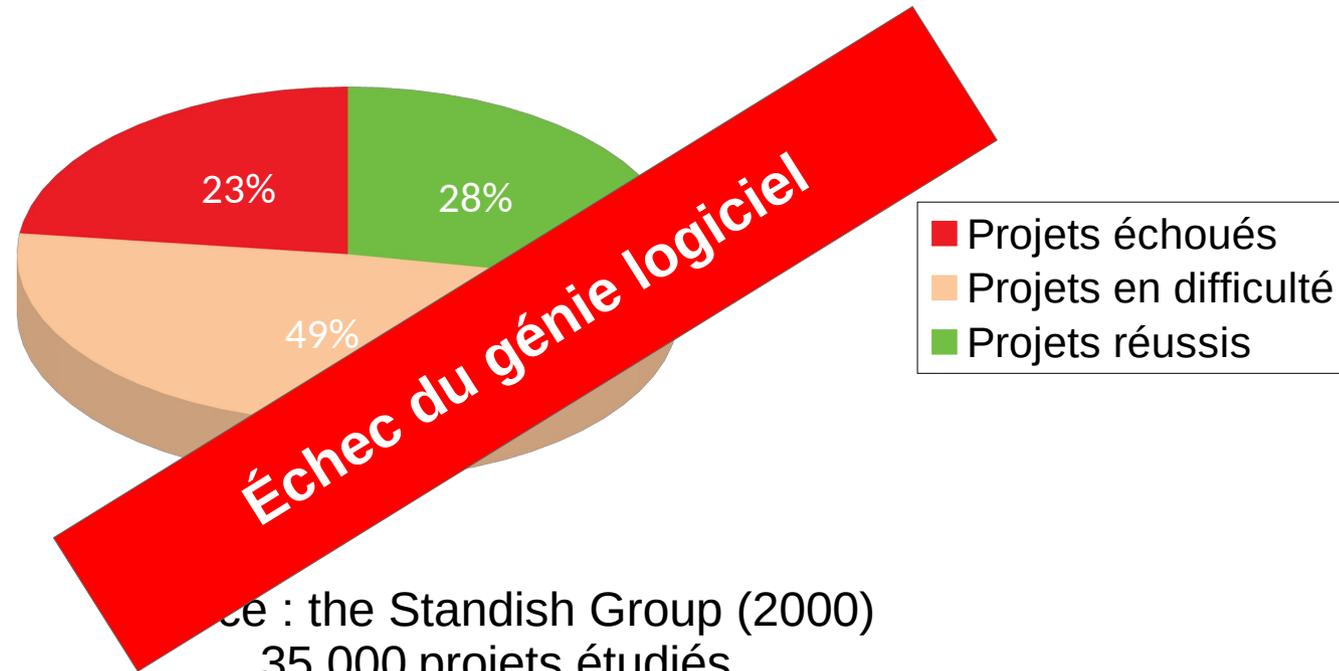
2

Création  
du génie logiciel

3

Bilan

# Bilan en 2001



Source : the Standish Group (2000)  
35.000 projets étudiés

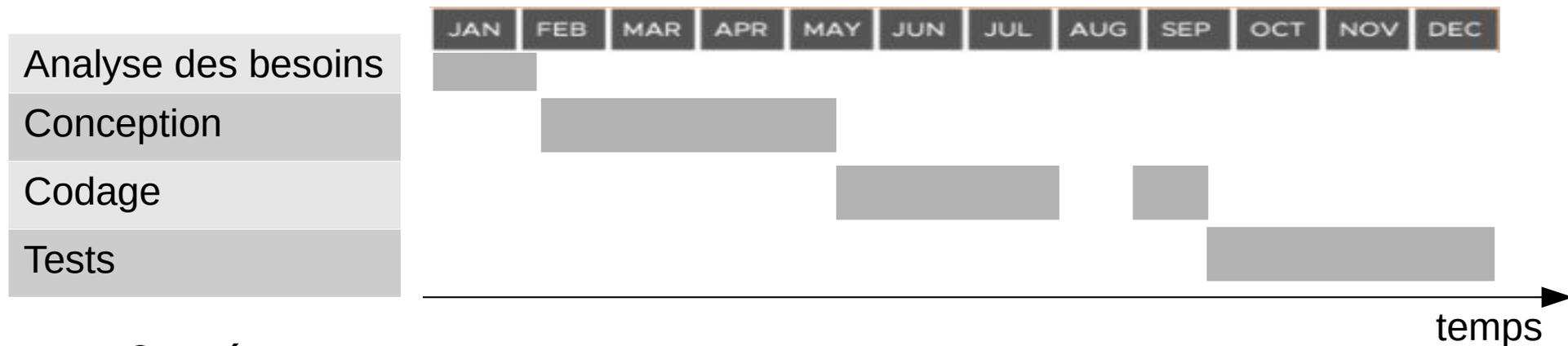
# Causes de l'échec

# Cause d'échec n°1 :

## Suivre un plan coûte que coûte

45

- Le cycle en cascade définit des étapes séquentielles précises
  - Engagement sur plusieurs mois
    - ▶ cf. diagramme prévisionnel de Gantt



- Conséquences
  - ▶ La moindre modification du planning met le projet en danger



« Bon, les gars, il y a eu un accident au quai.  
Une petite auto blanche est tombée à l'eau.  
Qu'est ce qu'on fait ? »



« Pas de problème, on va appeler une grue pour remonter l'automobile. »



« Ça va très bien, la voiture est toute petite,  
dans une demi-heure tout est terminé. »



« ARGH!!!! La grue est tombée aussi à l'eau avec la petite voiture ! »



« Ça va vraiment mal.  
Et là, qu'est ce qu'on fait ? »



« Pas de problème, on a appelé une autre grue, une GROSSE celle-là. »



« Bon, la petite automobile est déjà sortie,  
on aurait du appeler cette grue la première fois... »



« Il ne reste qu'à sortir la petite grue maintenant, dans une demi-heure tout devrait être fini. »

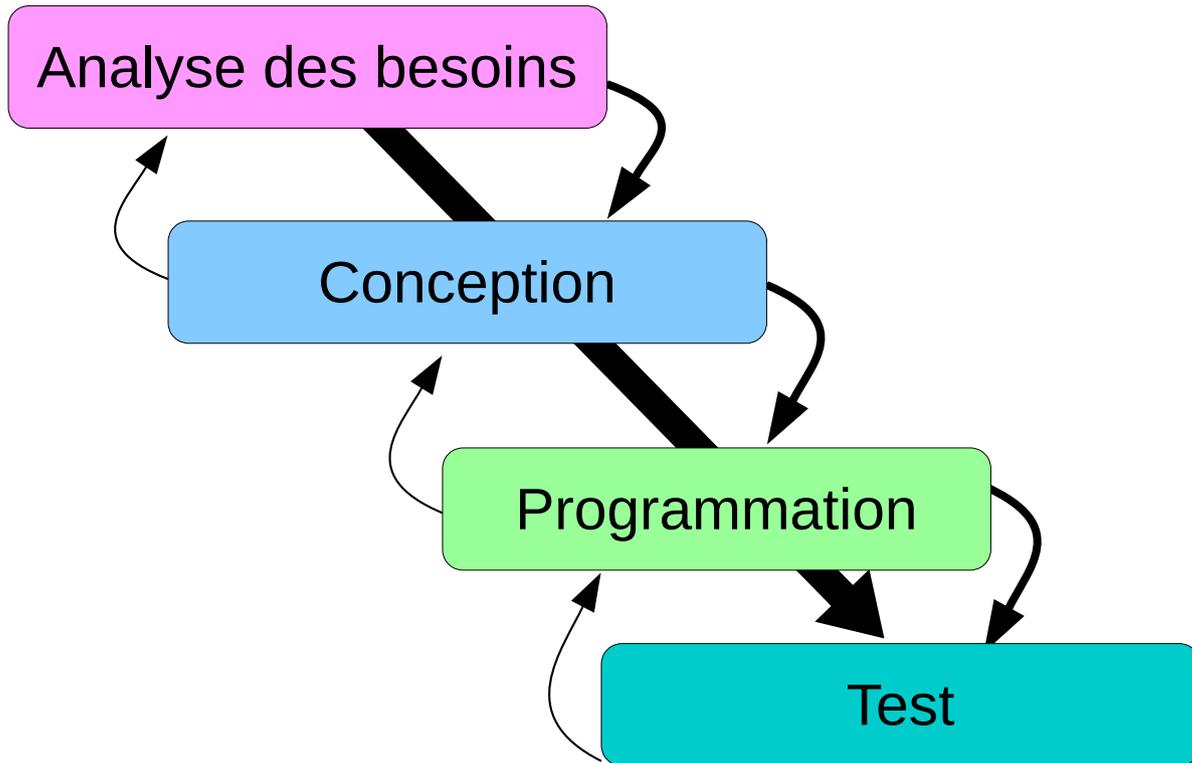


« ARGHHHHHH !!! ENCORE !!!!!!!!!!! »



« Pas de problème, on a appelé une autre grue, plus grosse encore.. »

# Un plan est rarement respecté



# Cause d'échec n°2 :

## Estimer la durée d'un projet

---

57

- Déterminer le nombre d'années-hommes est extrêmement difficile
  - Trop d'aléas et d'incertitudes
  - Pas de règles et pas de mesures systématiques

# Cause d'échec n°2 :

## Estimer la durée d'un projet

58

- Mythe de l'**année-homme**, cf. «The Mythical Man-Month», Fred Brooks, 1995
  - Non-linéarité de la charge de travail (source *Borland Software Corporation*) :

Taille équipe	Productivité par personne (KLOC/année)	Productivité de l'équipe (KLOC/année)	Gain
1	15.0	15.0	
2	11.9	23.8	<b>1.6</b>
10	7.0	69.6	<b>4.6</b>
25	5.1	128.2	<b>8.5</b>

- *Note : 15 KLOC/année → 9 LOC/h*
- Certaines tâches ne sont pas parallélisables :
  - ▶ « Neuf femmes ne font pas un enfant en un mois »

# Cause d'échec n°2 :

## Estimer la durée d'un projet

---

59

- Rattraper le retard
  - Contrairement à l'intuition, ajouter des personnes à une équipe d'ingénieurs ne permet pas de rattraper le retard, au contraire :
    - ▶ Les nouveaux doivent être formés par les anciens qui ne sont donc plus à leur tâche et le retard s'accroît encore



# 01

## Chapitre

# Introduction au génie logiciel

**1I2AC1 : Génie logiciel et Conception orientée objet**

Régis Clouard, ENSICAEN - GREYC

« Si les ouvriers construisaient les bâtiments  
comme les développeurs écrivent leurs programmes,  
le premier pic-vert venu aurait détruit toute civilisation. »

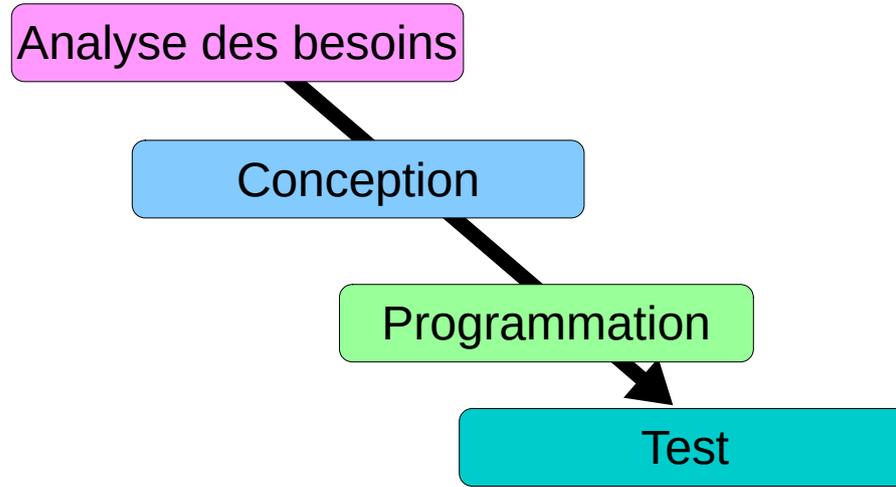
**Gerald Weinberg**

# Cause d'échec n°3 :

## Définir les besoins au début

---

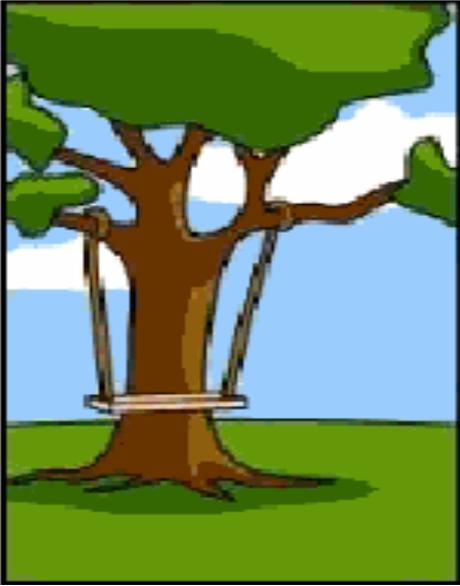
61



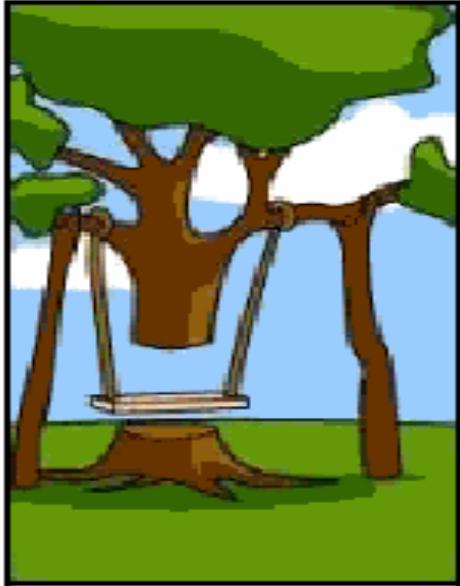
- Les clients ne savent pas identifier exactement ce qu'ils veulent
- Les clients ne savent pas exprimer clairement les besoins identifiés
- Les clients changent leurs besoins au cours du projet



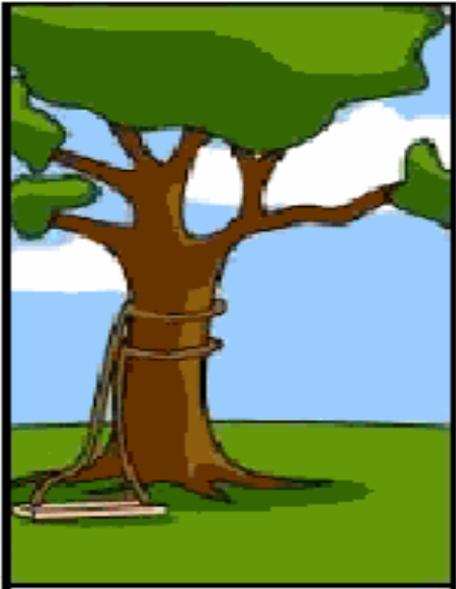
Ce que le client explique



Ce que le chef de projet comprend



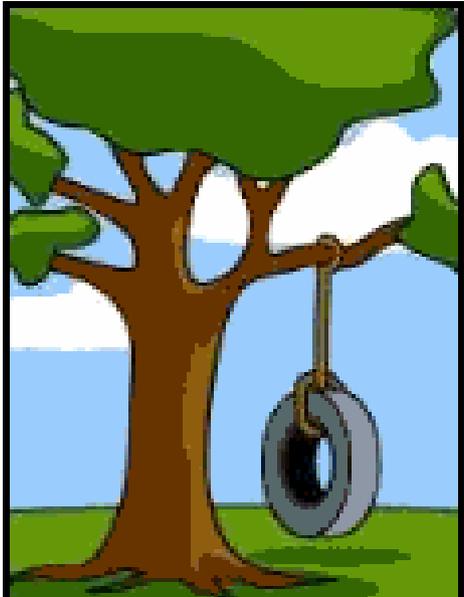
Ce que les analystes modélisent



Ce que les développeurs programment



Ce que les commerciaux vendent

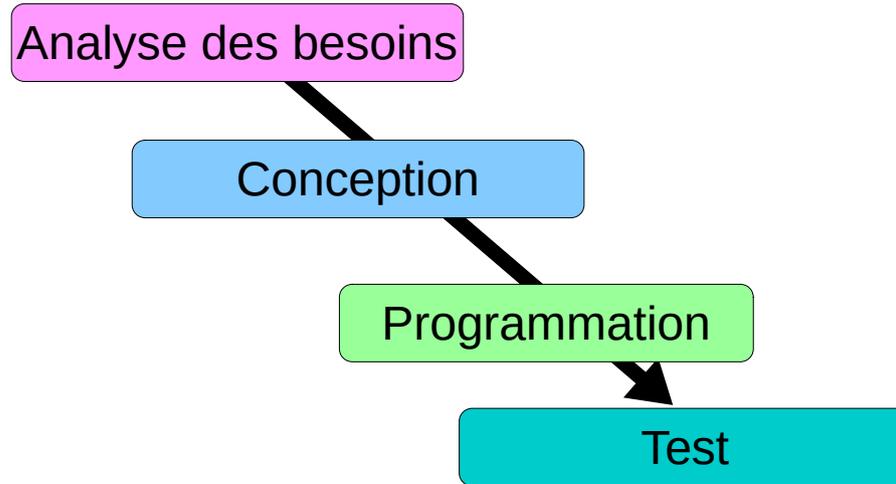


Ce que le client a réellement besoin

**Le produit final est le résultat  
d'une séquence d'incompréhensions**

# Cause d'échec n°4 : Réaliser les tests à la fin

65



- Les tests sont généralement la variable d'ajustement du temps
  - Les tests sont faits selon le temps restant (ou pas)
- Pourtant, il est impossible de garantir des logiciels sans défaut
  - Estimation : 1 à 10 bugs / KLOC
    - ▶ Windows 10 (80 MLOC) → **80 000 bugs !**

# Cause d'échec n°5

## Raisonner au niveau procédure

---

66

- Le paradigme procédurale est inadapté à la conception de programmes de très grande taille
  - Programmer au niveau procédural est une activité complexe qui revient à construire une voiture au niveau moléculaire ; le niveau atomique correspondant à l'assembleur

# Constat préliminaire

- La logiciel se prête mal aux méthodes du génie civil
  - Suivre un plan élaboré au début du projet
  - Estimer la durée des étapes d'un plan
  - Définir les besoins au début
  - Réaliser les tests à la fin
- Le logiciel n'est pas le matériel : il est mou
  - On peut construire une tour de 200 m en commençant par un bungalow
  - On peut construire un bungalow en commençant par la salle de bain
  - On peut construire une salle de bain en commençant par le papier peint
- Il doit donc être possible de repenser le génie logiciel en s'émancipant du génie civil

# Plan du chapitre

1

Pourquoi  
un cours sur le  
génie logiciel ?

2

Première génération  
du génie logiciel

3

Bilan : échec

4

Le génie logiciel  
aujourd'hui

# Révolution du génie logiciel (2001)

- Nouvelle définition du génie logiciel : **Agilité**

Le génie logiciel est l'art et l'ensemble des moyens techniques, industriels et humains qu'il faut réunir pour construire, distribuer et maintenir des logiciels de qualité

- Ajoute une dimension **artisanale** au développement (cf *software craftsmanship*) en prônant qu'il ne suffit pas qu'un logiciel soit fonctionnel, mais il faut qu'il soit **bien conçu**
- Le développeur travaille comme un artisan, guidé par son talent, son expérience et ses connaissances théoriques puisées dans le **partage** et la **formation**
- Le développeur s'appuie sur l'expérience des développeurs expérimentés. Il existe aujourd'hui près de 60 ans d'expérience en génie logiciel dont il est nécessaire de s'inspirer

# Le génie logiciel aujourd'hui

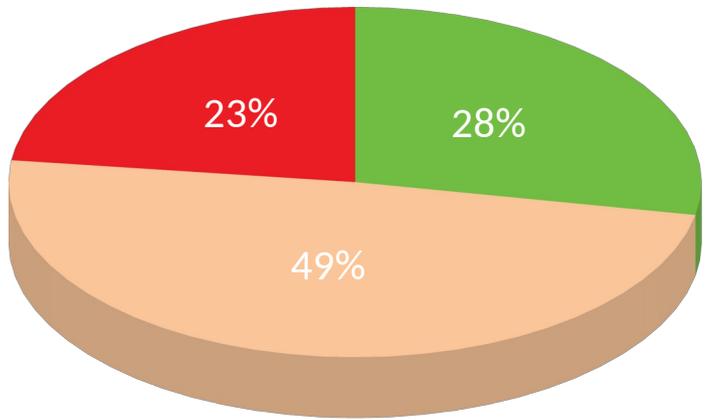
---

70

- Nouvelle méthode de gestion de projet
  - **Cycle itératif et incrémental**
- Nouveau paradigme de conception logicielle
  - **Conception orientée objet**
- Nouveau formalisme de modélisation
  - **UML**
- Ne reste qu'un seul métier : développeur logiciel
- Ces points sont développés dans la suite du cours

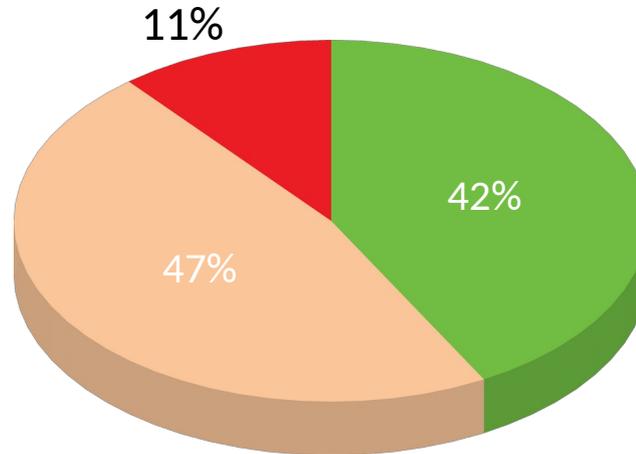
# Effet sur la réussite des projets

Rappel 2000

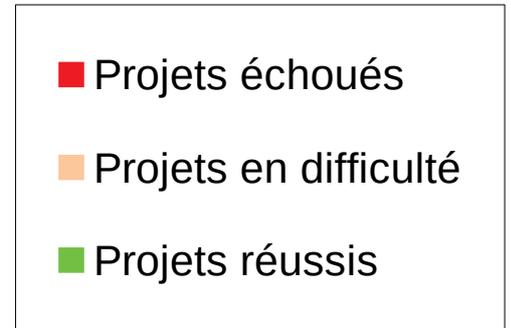


(35.000 projets étudiés)

2020



(10.000 projets étudiés)



Source : the Standish Group