

# Éléments d'histoire de l'informatique

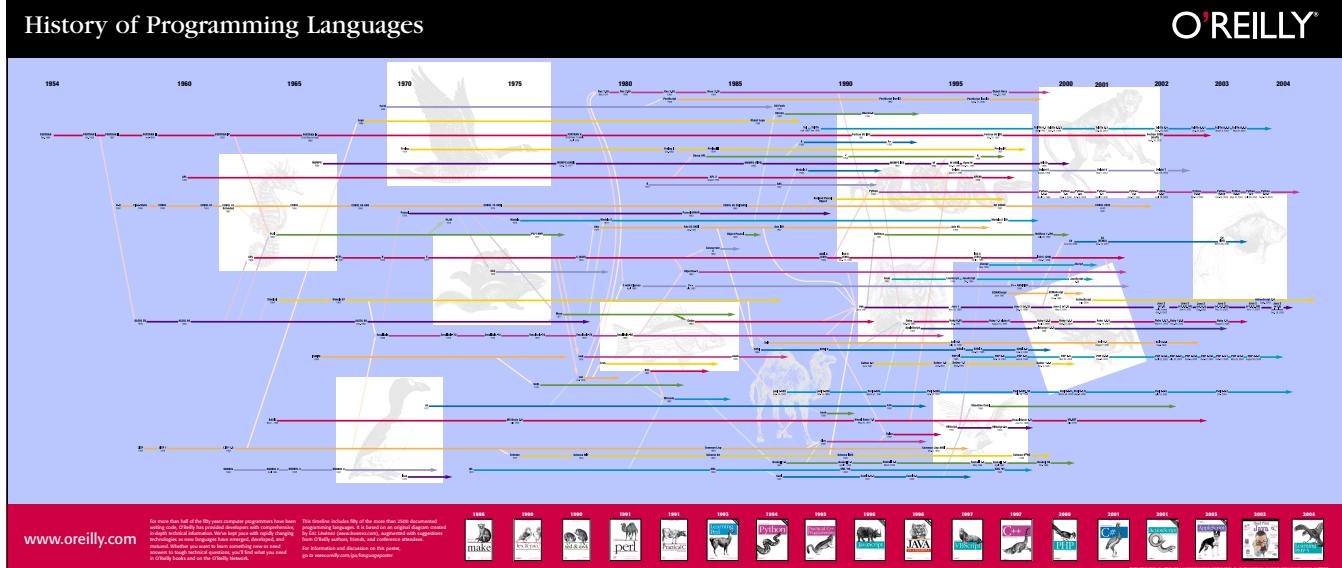
Sacha Krakowiak

Université Grenoble Alpes & Aconit

## 6. Langages de programmation et génie logiciel

CC-BY-NC-SA 3.0 FR

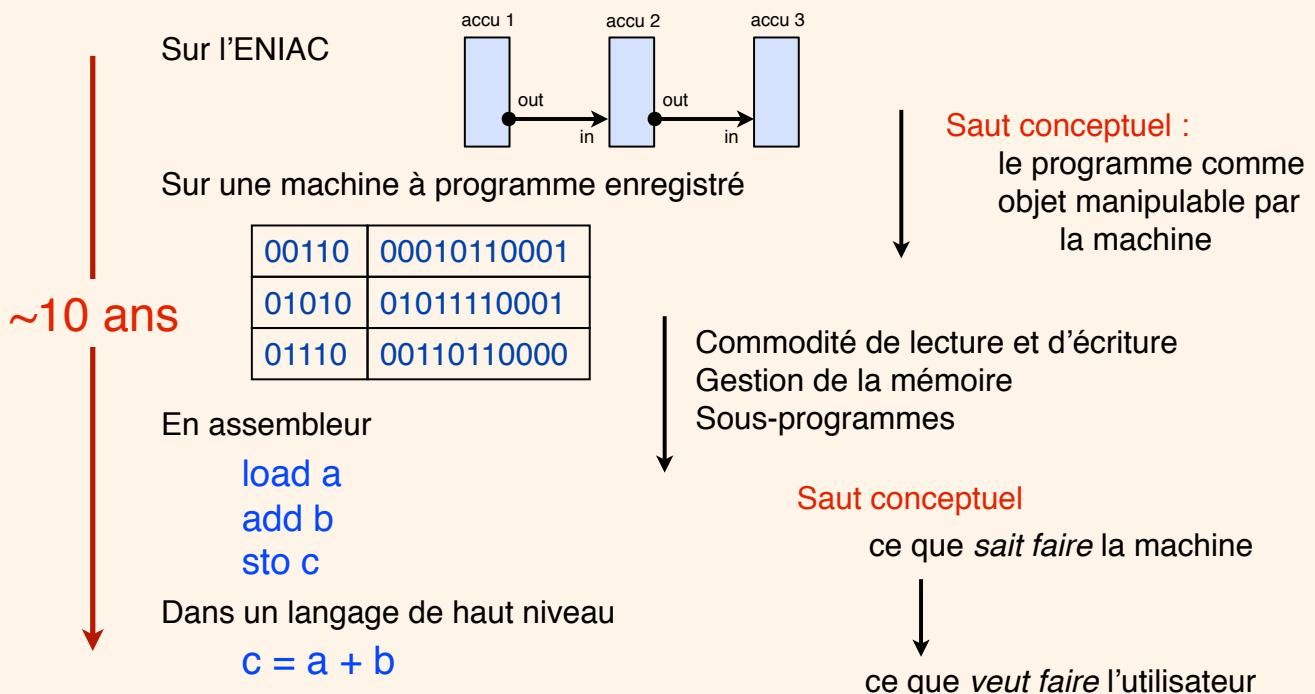
### Les langages de programmation



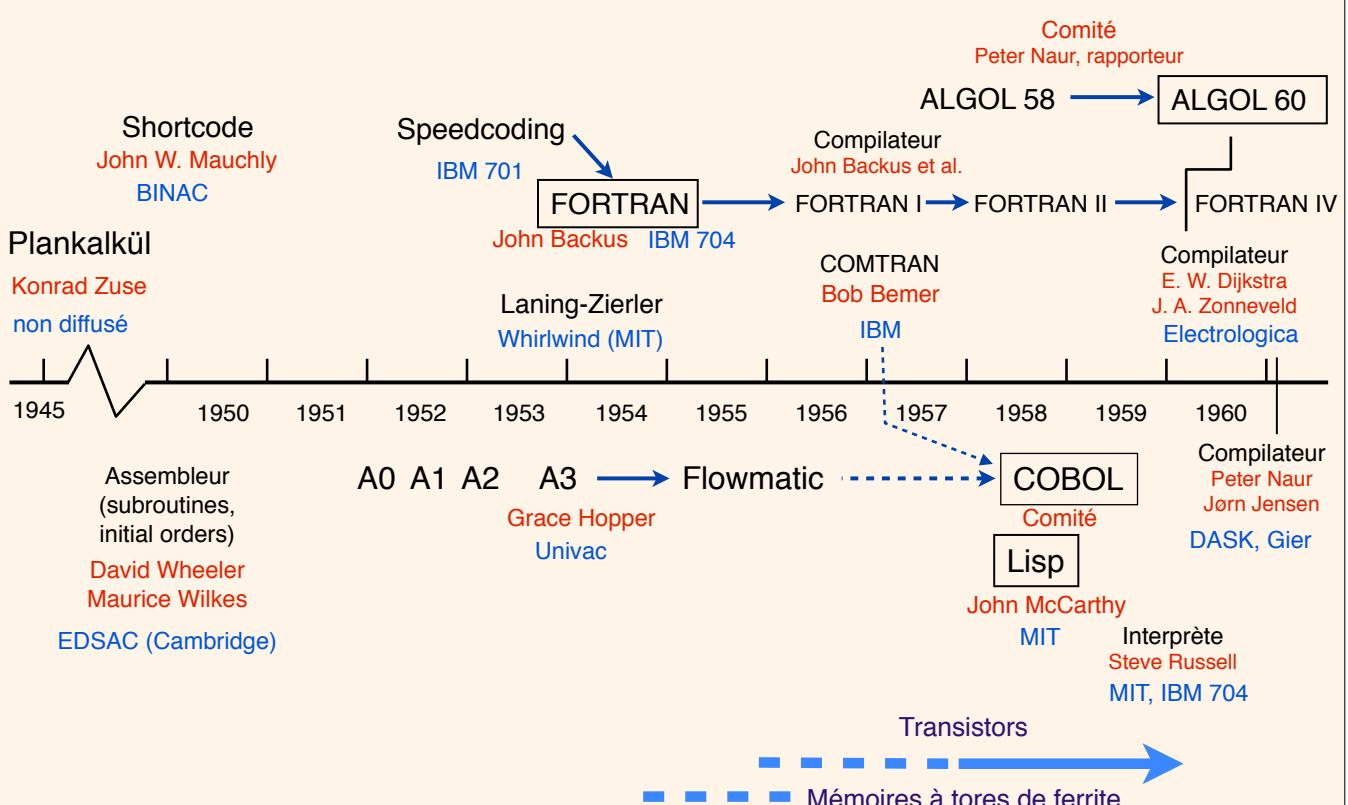
Voir [http://archive.oreilly.com/pub/a/oreilly//news/languageposter\\_0504.html](http://archive.oreilly.com/pub/a/oreilly//news/languageposter_0504.html)

# Qu'est-ce que «programmer une machine ?»

- Soit à ajouter un nombre  $a$  à un nombre  $b$  et à garder le résultat comme  $c$



## La naissance des langages de programmation



# La «programmation automatique»

1949-1954

## ❖ Remédier aux limitations des premiers ordinateurs

- pas de registres d'index
- pas d'opérations câblées en virgule flottante
- répertoire d'instructions incomplet
  - manque d'opérations logiques
- entrées-sorties difficiles à mettre en œuvre

## ❖ Fournir un mode d'expression plus commode que le binaire

- codes d'opérations et adresses symboliques
- organisation des programmes (sous-programmes)

## ❖ Mieux gérer les ressources

- allocation de la mémoire
- entrées-sorties

## ❖ Contrepartie : un coût non négligeable

Une notion primitive  
de «machine virtuelle»

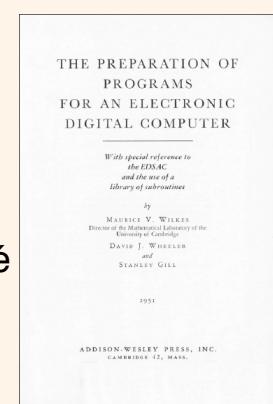
# Les premiers assemblateurs

## ❖ Short Code (John W. Mauchly, BINAC, 1949 puis UNIVAC)

- notation pour les expressions mathématiques (avec conversion manuelle)
- système interprétatif (facteur 50)

## ❖ Ordres initiaux (David Wheeler, Maurice Wilkes, EDSAC, 1949)

- une séquence d'instructions préenregistrée sur support externe (relais)
- remplit les rôles d'amorce (*bootstrap*), assebleur, éditeur de liens, chargeur en 41 instructions !
- code opération d'une lettre, adresses en décimal
- bibliothèque de sous-programmes sur ruban perforé
- séquence d'appel et retour de sous-programme permettant les appels emboîtés



© Association  
for Computing Machinery

# Entre assembleur et langage de haut niveau

La «programmation automatique» et les «pseudocodes»

## ❖ Speedcoding (John Backus, IBM 701, 1953)

«machine virtuelle» à 3 adresses, virgule flottante  
système interprétatif  
gestion des entrées-sorties et de la mémoire secondaire  
aide à la mise au point

## ❖ De A0 à A3 (Grace Hopper, UNIVAC, 1951-53)

A0, A1 : gestion des sous-programmes  
sous-programmes «ouverts» (recopiés)  
A2 : «machine virtuelle» à 3 adresses  
A3 : traduction de formules mathématiques

## ❖ Laning-Zierler (MIT - Whirlwind, 1953)

Notation algébrique des formules  
parenthèses, précédence d'opérateurs  
Noms symboliques  
Langage machine pour les branchements conditionnels

### Autres précurseurs

Rutishauser (Zürich)  
compilateur spécifié, non  
implémenté

Böhm (Zürich)  
compilateur écrit dans le  
langage même, non  
implémenté

# Naissance des langages de haut niveau

(1945 ; 1954-60)

## ❖ Plankalkül, un précurseur sans descendance (Konrad Zuse)

1945. Pas d'implémentation (sinon à titre historique, 1998)

## ❖ FORTRAN (John Backus, IBM)

FORTRAN-0, 1954, pas implémenté  
Compilateur de Fortran I en avril 1957

## ❖ Lisp (John McCarthy, MIT)

Interprète en 1959 ; compilateur en 1962

## ❖ COBOL (Comité ; influence de FLOWMATIC (Hopper), COMTRAN (Bemer))

1959 ; compilateur en 1960

## ❖ ALGOL (Comité ; rôle déterminant de Peter Naur, Regnecentralen)

ALGOL-58, pas implémenté  
Compilateurs d'ALGOL-60 en 1960 et 1961

# Plankalkül

## ❖ Un langage en avance sur son temps

Les programmes sont des procédures (non récursives), avec variables locales

Les types de données sont riches

tableaux (et tuples de tableaux  
de taille variable  
données en virgule fixe et flottante  
enregistrements, listes de paires

Instruction d'affectation

Instruction conditionnelle

Boucles et construction **while**

Pas de **goto**

Opérations logiques

Exceptions dans le calcul arithmétique

## ❖ Une notation à deux dimensions

A + 1 ⇒ A	
V   4	5
S   1.n	1.n

représente  $A[5] := A[4] + 1$

1	A2 = (A9, AΔ1)
2	P1      R(V) ⇒ R
3	V      0      0
4	A      Δ1     Δ1
5	$\sqrt{ V } + 5 \times V^3 \Rightarrow R$
6	V      0      0      0
7	A      Δ1     Δ1     Δ1
8	P2      R(V) ⇒ R
9	V      0      0
10	A      11 × Δ1    11 × 2
11	W2(11)      R1(V) ⇒ Z
12	V      0      0      0
13	K      i
14	A      Δ1     Δ1
15	Z > 400 $\Rightarrow (i, +\infty) \Rightarrow R$ (10-i)
16	V      0
17	K
18	A      9      2      9
19	Z > 400 $\Rightarrow (i, Z) \Rightarrow R$ (10-i)
20	V      0      0      0
21	K
22	A      9      Δ1     2      9

D. E. Knuth and L. T. Pardo. "The early development of programming languages" in *Encyclopedia of Computer Science and Technology*, Marcel Dekker, New York, 1977

# Les débuts de FORTRAN

## ❖ Le contexte (1954)

La «programmation automatique»  
peu de systèmes (A2, Laning-Zierler, Speedcoding)  
faible efficacité  
scepticisme sur la démarche

Une nouvelle machine, l'IBM 704  
virgule flottante câblée  
registres d'index

## ❖ Les motivations

essentiellement économiques : **l'efficacité du code objet avant tout**  
étendre la population des utilisateurs  
initialement : uniquement prévu pour l'IBM 704

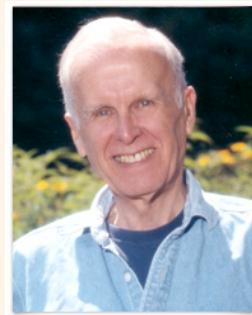
## ❖ Des prévisions optimistes...

plus de débogage (langage de haut niveau)  
compilateur prêt en 6 mois (il a fallu 2 ans, 72 hommes x mois)

# Chronologie initiale de FORTRAN

## ❖ Janvier 1954 : feu vert d'IBM

conception du langage (FORTRAN-0)  
rapport préliminaire en novembre 1954



John Backus (1924-2007)

Image courtesy of the Computer History Museum

## ❖ Fin 1954 - début 1955

présentation à des groupes d'utilisateurs potentiels  
début de la construction du compilateur

## ❖ 1955 - 1957

construction du compilateur pour IBM 704, accent sur l'**efficacité**  
évolution vers FORTRAN I ; *Programmer's Reference Manual* (octobre 1956)  
avril 1957 : compilateur prêt, *Preliminary Operator's Manual* édité  
début 1958 : FORTRAN II (diagnostics, compilation séparée)

## ❖ 1958 - 1962

FORTRAN III, diffusion restreinte : test pour **FORTRAN-IV, version stable 1962**  
COMMON, variables logiques, déclarations de type, noms symboliques des E/S  
**FORTRAN IV compilé pour le 7090**

## À quoi ressemblent les premiers FORTRANs ?

Algol 60

```
begin integer i; real array a[0:10]real
procedure f(t); real t; value t;
    f := sqrt(abs(t))+5*x*t^3;
for i := 0 step 1 until 10 do read(a[i]);
    for i := 10 step -1 until 0 do
        begin y := f(a[i]);
        if y > 400 then write(i, "TOO LARGE");
            else write(i, y);
        end
    end;
```

D. E. Knuth and L. T. Pardo. "The early development of programming languages" in *Encyclopedia of Computer Science and Technology*, Marcel Dekker, New York, 1977

FORTRAN 1

FORTRAN 0

```
DIMENSION A(11)
READ A
2 DO 3,8,11 J=1,11
3 I=11-J
Y=SQRT(ABS(A(I+1)))+5*A(I+1)**3
IF (400. >=Y) 8,4
4 PRINT I,999.
GO TO 2
8 PRINT I,Y
11 STOP
```

```
FUNF(T) = SQRT(ABSF(T))+5.0*T**3
DIMENSION A(11)
1 FORMAT(6F12.4)
READ
2 DO 10 J=1,11
I=11-J
Y= FUNF(A(I+1))
IF (400.0-Y)4,8,8
4 PRINT 5,I.
5 FORMAT(I10, 10H TOO LARGE)
GO TO 10
8 PRINT 9,I,Y
9 FORMAT(I10, F12.7)
10 CONTINUE
STOP 52525
```

# L'apport de FORTRAN

- ❖ Premier «vrai» langage de haut niveau
  - une révolution dans les usages : le calcul à la portée des usagers
  - une appropriation au delà des espoirs des réalisateurs
- ❖ Preuve de la possibilité d'une compilation efficace
  - c'était le principal objectif initial (en 1954)
- ❖ Un usage universel pour le calcul scientifique
  - évolution vers le calcul à hautes performances
  - l'outil de base pour les «sciences numériques»
- ❖ Les limitations des premiers temps
  - langage peu sûr (compilateur trop tolérant)
  - sémantique parfois incertaine
  - définitions statiques
  - pas d'allocation dynamique
  - pas de procédures récursives

amélioration (lente !)  
au fil des versions

## La compilation de FORTRAN

- ❖ Conséquences de la primauté de l'efficacité...
  - La définition du langage est influencée par la construction du compilateur
  - Le compilateur est très dépendant de la machine cible (l'IBM 704)
- ❖ Quelques points délicats de la compilation
  - Peu de difficulté pour les expressions
  - Réduire le coût du calcul d'adresse (en particulier des tableaux)
    - procéder par incrément pour réduire le nombre de multiplications
  - Réduire le coût de sauvegarde-restauration des registres d'index (3)
    - estimer la fréquence d'exécution des parties du programme
- ❖ Exemples concrets...
  - La construction DO est privilégiée pour les itérations, avec incrément d'index constants (et positifs)
  - Les indices de tableau doivent être des fonctions linéaires des index
  - Le nombre d'indices des tableaux est limité à 3 (sinon coût trop élevé)
  - La compilation séparée (à partir de FORTRAN II) réduit le coût de compilation

# L'évolution de FORTRAN

(après FORTRAN IV)

- ❖ FORTRAN IV (1961-62)
- ❖ FORTRAN 66
  - premier standard industriel (ANSI)
- ❖ FORTRAN 77
  - amélioration des constructions IF et DO
  - amélioration des entrées-sorties
  - traitement de données de type CHARACTER
- ❖ Fortran 90
  - meilleur traitement des tableaux (ARRAY)
  - procédures récursives
  - modules, gestion dynamique de la mémoire
- ❖ Fortran 95
  - améliorations diverses sur tableaux et chaînes
- ❖ Fortran 2003
  - programmation par objets
  - entrées-sorties élaborées
- ❖ Fortran 2008, 2015
  - améliorations mineures

→ Extensions pour le calcul à hautes performances, 1993 (parallélisme)

## Des avis sur Fortran...

- ❖ Un langage peu aimé des informaticiens puristes...

*“FORTRAN --‘the infantile disorder’--, by now nearly 20 years old, is hopelessly inadequate for whatever computer application you have in mind today: it is now too clumsy, too risky, and too expensive to use.”*

Edsger Dijkstra (1975)

- ❖ ... mais reconnu comme incontournable

*“Fortran is the lingua franca of the computing world. It is the language of the streets in the best sense of the word, not in the prostitutional sense of the word. And it has survived and will survive because it has turned out to be a remarkably useful part of a very vital commerce.”*

Alan J. Perlis (1978)

*“I don't know what the language of the year 2000 will look like, but I know it will be called Fortran.”*

C. A. R. Hoare (1982)

# La genèse d'Algol

## ❖ La situation en 1958

Un langage dominant : FORTRAN (lié à IBM) ; divers autres projets, tous liés à une machine

deux initiatives pour définir un langage scientifique indépendant de la machine : GAMM (Allemagne) et ACM (USA) ; elles fusionnent

## ❖ Étape préliminaire : Algol 58

Création d'un comité et réunion à Zürich, 27 mai - 1<sup>er</sup> juin 1958

Objectif : un langage indépendant de la machine, rigoureusement défini, bien adapté à la description d'algorithmes, compilable

Définition d'Algol 58, inspiré de FORTRAN

NOMBREUSES propositions et tentatives de compilation

Réunions partielles intermédiaires

## ❖ Algol 60

Réunion à Paris, janvier 1960 ; adoption de la BNF pour la description

Premier rapport, mai 1960 ; réunion à Rome (avril 1962), rapport révisé

Rôle prépondérant de Peter Naur, rapporteur du groupe

# Les innovations d'Algol 60

## ❖ Structure de bloc

blocs emboîtés, variables locales

## ❖ Tableaux de taille variable (allocation dynamique sur pile)

## ❖ Définition rigoureuse des procédures

procédures récursives : objet de nombreux débats au sein du groupe

appel par nom : généralité, mais réalisation complexe et pièges sémantiques

## ❖ Déclarations de type

premier effort vers une garantie de sécurité

## ❖ Définition rigoureuse de la syntaxe

any sequence of decimal digits with a decimal point preceding or intervening between any 2 digits or following a sequence of digits, all of this optionally preceded by a plus or minus sign. The number must be less than  $10^{38}$  in absolute value and greater than  $10^{-38}$  in absolute value

```
<digit> ::= 0|1|2|3|4|5|6|7|8|9
<integer> ::= <digit> | <integer><digit>
<realPart> ::= .<integer>|<integer>.|<integer>.<integer>
<real> ::= <realPart>|+<realPart>|-<realPart>
```

# Le rapport Algol 60

## ❖ Un texte-clé de l'histoire de l'informatique

L'introduction de la Backus-Naur Form (BNF)  
un formalisme de définition de la syntaxe

La première définition rigoureuse et complète  
d'un langage de programmation  
la sémantique est encore en langage  
naturel

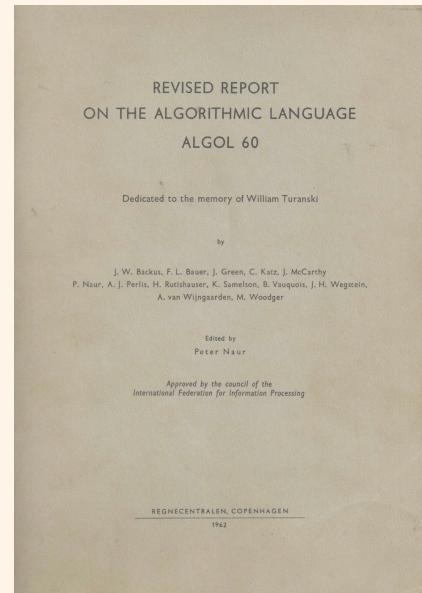


Les langages de programmation deviennent  
l'objet d'une étude scientifique  
Le rôle déterminant de Peter Naur (1928 - 2016)  
prix Turing 2005

CC-BY-SA 2.5, [Eriktj](#)

*Ce qui peut se dire peut se dire clairement ; et sur ce  
dont on ne peut parler, il faut garder le silence.*

Ludwig Wittgenstein



# La compilation d'Algol 60

## ❖ Quelques dates marquantes (non exhaustif !)

Août 1960	E.W. Dijkstra, J. A. Zonneveld	Electrologica X-1
	premier compilateur d'Algol 60	
Fin 1960	E. T. Irons	CDC 1604
	compilateur dirigé par la syntaxe	
1961	Burroughs (avec Hoare, Dijkstra)	gamme Burroughs
	machine B5000 conçue pour Algol	
1961, 1962	P. Naur, J. Jensen	DASK, Gier (Regnecentralen)
	compilateur de grande qualité pour Algol complet	
1964	B. Randell, L. J. Russell	English Electric KDF9
	livre <i>Algol 60 Implementation</i> , description détaillée d'un compilateur	

# Algol 60 : ce qui n'a pas survécu

## ❖ L'appel par nom

général et puissant, mais implémentation complexe et pièges sémantiques

```
real procedure somme(debut, fin, i, terme); value debut, fin;  
    integer debut, fin, i; real terme;  
begin real s;  
    s:=0;  
    for i:=debut step 1 until fin do  
        s:=s+terme;  
    somme:=s;  
end;
```

somme(1, 20, k, 1/k) calcule  $\sum_{i=1}^{20} 1/i$

somme(0, 15, k, (k+2)(2\*k+1)) calcule  
$$\sum_{i=0}^{15} (i+2)/(2i+1)$$

## ❖ La construction *own*

variables rémanentes après la sortie d'un bloc : maniement délicat

## ❖ La construction *switch*

étiquette de branchement calculée dynamiquement  
sémantique complexe, maniement délicat

# Les successeurs d'Algol 60

*Here is a language so far ahead of its time, that it was not only an improvement on its predecessors, but also on nearly all its successors.*

C. A. R. Hoare (1973)

Pourquoi ?

position établie de FORTRAN  
manque de soutien d'IBM  
entrées- sorties non définies

## ❖ Une source d'inspiration

Algol 60 a été relativement peu utilisé, mais...

... il a eu une influence profonde sur la quasi-totalité des langages impératifs  
(et même sur certains langages fonctionnels)

## ❖ Les dérivés directs

Algol W (N. Wirth, C. A. R. Hoare)

chaînes (*string*), structures (*records*), paramètres par valeur/résultat

Algol 68 (Comité ; A. van Wijngaarden)

grammaire à deux niveaux

influence conceptuelle

Pascal (N. Wirth, 1970)

un but initialement pédagogique

une large diffusion

# La genèse de COBOL

## ❖ Reconnaissance d'un besoin

un langage pour applications de gestion (par opposition au calcul scientifique)  
création d'un comité (initiative du DoD, USA) : réunion le 29-29 mai 1959  
un premier rapport en décembre 1959 sur «COBOL 60»  
des sources d'inspiration : FLOWMATIC (G. Hopper), COMTRAN (B. Bemer)

## ❖ Un démarrage difficile

3 révisions de la définition en 5 ans  
des compilateurs initialement peu efficaces

## ❖ Les innovations

DEFINE : première construction pour macros  
DATA DIVISION

définition de structures de données hiérarchiques (*records*)  
définition précises de formats de représentation des variables  
spécification de structure des fichiers

tentative (peu réussie) d'imiter le langage naturel



Grace Hopper  
1906-1992

Image courtesy of the  
Computer History Museum

```
$ SET SOURCEFORMAT"FREE"
IDENTIFICATION DIVISION.
PROGRAM ID. Iteration-If.
AUTHOR. Michael Coghlan.

DATA DIVISION.
WORKING STORAGE SECTION.
01 Num1    PIC 9  VALUE ZEROS.
01 Num2    PIC 9  VALUE ZEROS.
01 Result   PIC 99 VALUE ZEROS.
01 Operator  PIC X  VALUE SPACE

PROCEDURE DIVISION.
Calculator.
    PERFORM 3 TIMES
        DISPLAY "Enter First Number      : " WITH NO ADVANCING
        ACCEPT Num1
        DISPLAY "Enter Second Number     : " WITH NO ADVANCING
        ACCEPT Num2
        DISPLAY "Enter Operator (+ or *) : " WITH NO ADVANCING
        ACCEPT Operator
        IF OPERATOR = "+" THEN
            ADD Num1, Num2 GIVING Result
        ENDIF
        IF OPERATOR = "*" THEN
            MULTIPLY Num1 BY Num2 GIVING Result
        ENDIF
        DISPLAY "Result is = ", Result
    END PERFORM
STOP RUN
```

À quoi ressemble  
un programme COBOL ?

Un langage verbeux  
et peu élégant

# Un bref bilan de COBOL

- ❖ **Une adoption universelle, après un démarrage lent**
  - premiers compilateurs peu efficaces
  - des insuffisances initiales (procédures sans paramètres, ...)
  - soutien actif du DoD, publication de normes successives (61, 65, 68, 74, ...)
- ❖ **Vers 1970, devient le langage le plus largement utilisé**
  - on estime que 80% des programmes existants sont écrits en COBOL
- ❖ **Un langage méprisé par la communauté académique**
  - "The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense". E. W. Dijkstra*
- ❖ **Un langage sans successeurs...**
  - ... mais une évolution progressive
  - un déclin face aux progiciels, mais reste le poids de l'héritage
  - I'activité autour de COBOL se résume à la maintenance

... mais PL/I  
tentative de mariage  
FORTRAN-COBOL

## La genèse de Lisp

- ❖ **Influences initiales et premiers efforts**
  - IPL-2, Newell, Shaw, Simon (1956) : structure de liste
  - l'IBM 704, en particulier la structure d'adressage
  - le lambda-calcul (mais essentiellement pour la notation des fonctions)
  - un langage de traitement de listes en FORTRAN : FLPL (1957-58)
    - H. Gelernter, C. Geberich (IBM) sur suggestion de J. McCarthy (MIT)
    - pas d'expressions conditionnelles ni de fonctions récursives
- ❖ **Première implémentation**
  - Fin 1958 : début d'implémentation (John Mc Carthy, MIT)
    - idée du ramasse-miettes**
  - Lisp comme langage et comme description de fonctions calculables (\*)
    - absence d'effets de bord : langage fonctionnel**
  - La fonction **eval** et le premier interpréte (inattendu) : Steve R. Russell

(\*) John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *Communications of the ACM* (3:4), April 1960 , pp. 184-195

# Un programme en Lisp

```
(DEFUN DEPLACER (d X Y)
  (LIST (LIST 'DÉPLACER 'DISQUE d 'DE X 'VERS Y)))
```

```
(DEFUN HANOI (N X Y Z)
; Déplace une tour de hauteur N de X vers Z
(COND ((> N 0)
  (APPEND (HANOI (- N 1) X Z Y)
    (DEPLACER n X Z)
    (HANOI (- N 1) Y X Z)))))
```

L'appel (hanoi 3 '1 '2 '3) produit le résultat suivant :

```
((DÉPLACER DISQUE 1 DE 1 VERS 3) (DÉPLACER DISQUE 2 DE 1
VERS 2) (DÉPLACER DISQUE 1 DE 3 VERS 2) (DÉPLACER DISQUE 3
DE 1 VERS 3) (DÉPLACER DISQUE 1 DE 2 VERS 1) (DÉPLACER
DISQUE 2 DE 2 VERS 3) (DÉPLACER DISQUE 1 DE 1 VERS 3))
```

Les disques sont numérotés dans l'ordre croissant de leur taille.

## Les apports de Lisp

### ❖ Les concepts

- Programmation fonctionnelle
- Manipulation symbolique
- Récursivité

### ❖ Les techniques

- Représentation et manipulation de listes
- Méta-circularité (l'interprète de Lisp en Lisp)
- Le ramasse-miettes

### ❖ Les usages

- «Intelligence artificielle» : démonstration et raisonnement, calcul formel, ...
- Systèmes experts

### ❖ La suite...

- Dialectes courants de Lisp : Scheme (1975) et Common Lisp
- Héritage de Lisp, les langages fonctionnels : ML, Haskell, OCaml, ...



John McCarthy (1927-2011)  
vers 1987

Image courtesy of the Computer History Museum

# APL, un langage hors du commun

## ❖ Histoire

Développé entre 1958 et 1964 par Kenneth Iverson, à Harvard puis à IBM  
Au départ, une notation mathématique pour les tableaux  
Utilisation majoritaire en temps partagé



## ❖ Caractéristiques

Une structure de données unique : le tableau (ou vecteur)  
Des opérateurs sur les tableaux, en particulier un itérateur (opération unique sur tous les éléments)  
Une évaluation de droite à gauche  
Une notation utilisant des symboles spéciaux  
Une exécution le plus souvent interprétative

Kenneth Iverson  
1920-2004

Image courtesy of Computer History Museum

## ❖ Programmation

Une expression extrêmement concise (*one-liners*)

$(\sim R \in R \circ . x R) / R \leftarrow 1 \downarrow i R$  calcule les nombres premiers jusqu'à R

# Retour vers la machine

## ❖ Motivation

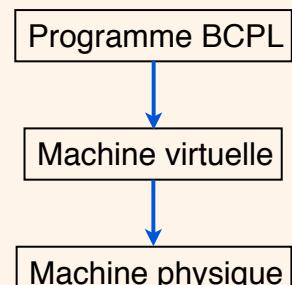
programmer le logiciel de base (compilateurs, systèmes d'exploitation), dépendant des caractéristiques de la machine

## ❖ Première approche : l'assembleur déguisé

Exemple : PL360 (Wirth, 1968) et langages analogues (LP10070, etc.)  
Visibilité des registres, constructions de haut niveau (IF, WHILE, etc.)  
Procédures

## ❖ Deuxième approche : la machine virtuelle

BCPL (Martin Richards, Cambridge, 1967)  
Structure de données unique : le mot  
Le précurseur de C

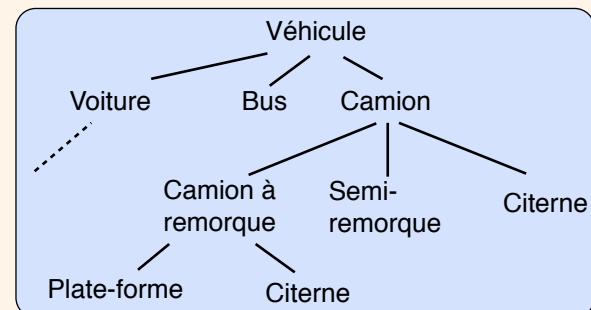


# Nouveaux paradigmes

## ❖ Langages à objets : Modéliser le monde réel

Simula (Dahl - Nygaard, 1967)  
regrouper données-fonctions d'accès  
objets, classes, héritage

Smalltalk (Kay et al., 1978-80)  
métaclasses, polymorphisme  
machine virtuelle à objets  
environnement de programmation



## ❖ Langages logiques : Modéliser le raisonnement

Représentation des connaissances  
(McCarthy, Minsky, Hewitt, années 1960)  
**declaratif vs procédural**

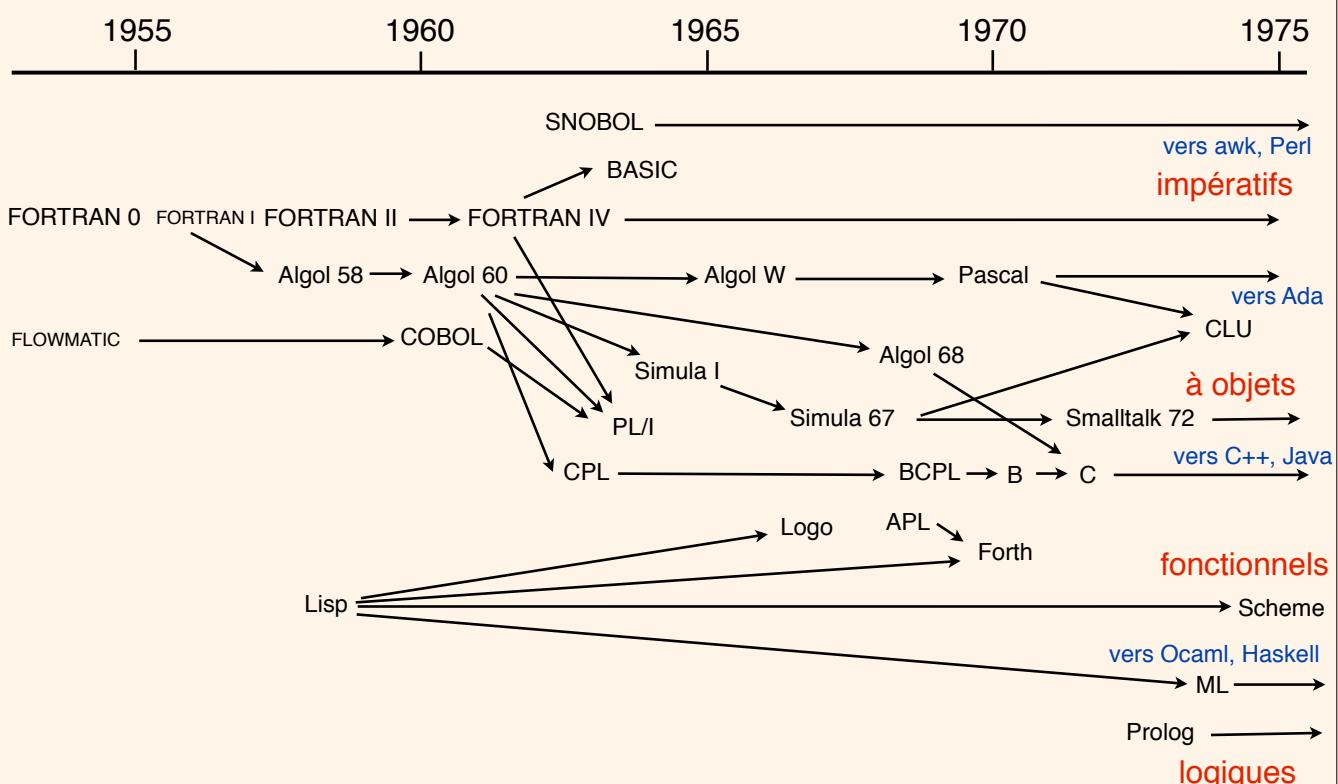
Prolog (Colmerauer - Kowalski, 1972)  
**faits et règles**  
**moteur d'inférence**

Programmation par contraintes

Pierre est fils de Paul ; Julie est fille de Paul  
Marie est fille de Pierre ; Cécile est fille de Pierre  
Émile est fils de Julie  
A enfant de B = A fils de B ou A fille de B  
A cousin germain de B = A enfant de X  
et B enfant de Y et X enfant de Z  
et Y enfant de Z et X différent de Y  
Quels sont les cousins germains d'Émile ?  
--> Marie, Cécile

# Langages de programmation

## Les 20 premières années



# Qualités d'un langage de programmation

Priorités aujourd'hui

## ❖ Qu'est-ce qu'un «bon» langage de programmation ?

Un langage sûr

qui empêche de faire les erreurs les plus courantes

exemple : ne pas ajouter des pommes et des oranges ...

Un langage rigoureux

une “sémantique” bien définie (on sait précisément ce qu’on fait)

idéalement : on peut prouver que le programme fait bien ce qu’on veut qu’il fasse

Un langage élégant et lisible

un programme est fait autant pour être lu (et compris) que pour être exécuté

Un langage expressif

Un langage efficace



Priorités des années 1950

# Le génie logiciel

## ❖ Les motivations

Produire des programmes corrects est une entreprise **difficile**

Il est déjà difficile de définir un «programme correct»...

Un visionnaire :

«Je pris conscience, et cette révélation me frappa avec toute sa force, que j'allais passer une bonne partie du restant de ma vie à trouver des erreurs dans mes propres programmes».

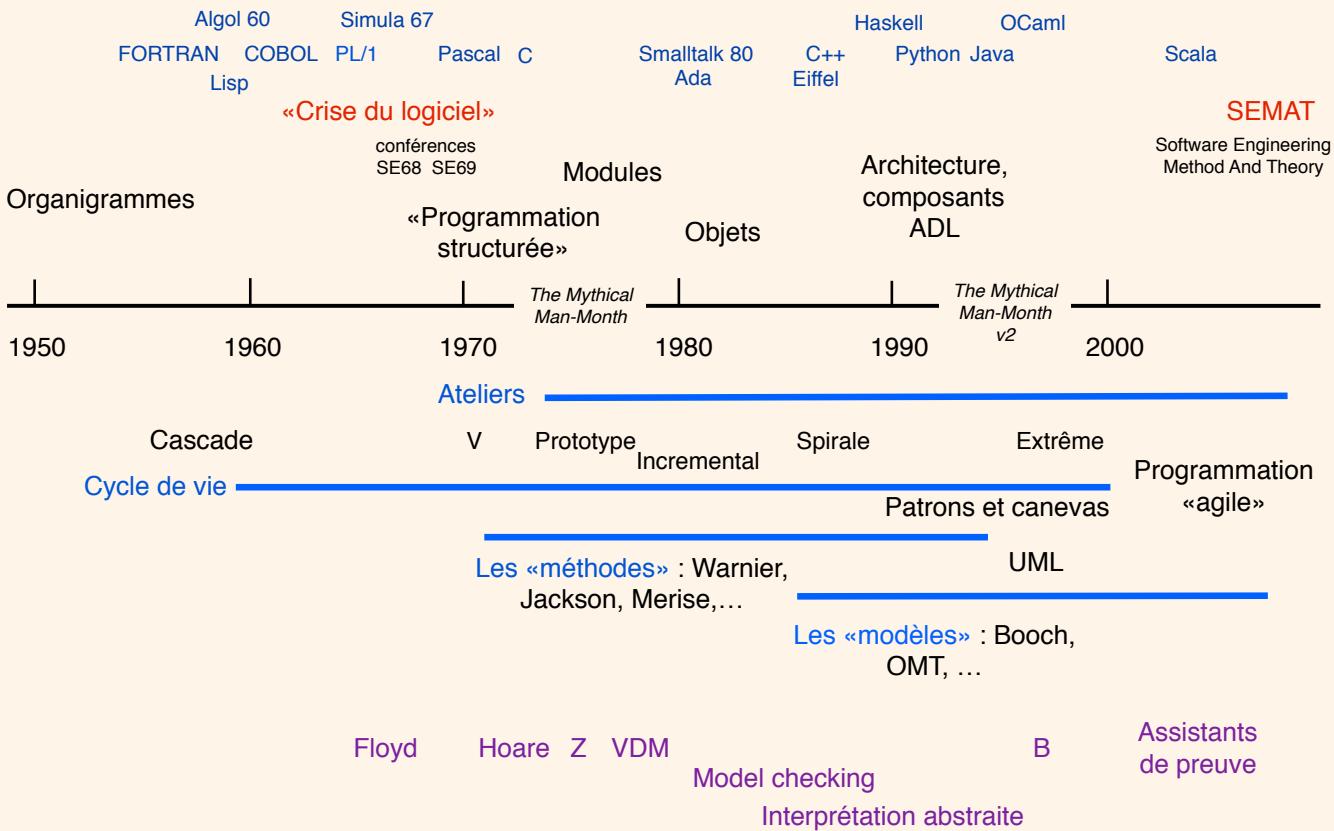
Maurice Wilkes, 1949

## ❖ L'objectif

Définir et appliquer des méthodes, des outils et des pratiques, *fondés sur des principes scientifiques*, et propres à assurer la production de logiciel répondant à des besoins spécifiés et respectant certains critères de qualité, eux-mêmes spécifiés, ainsi que des contraintes économiques.

## ❖ A-t-on réussi ?

# Le génie logiciel



CC-BY-NC-SA 3.0 FR - S. Krakowiak, 2016/17

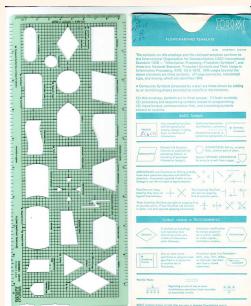
Histoire de l'informatique

7 - 35

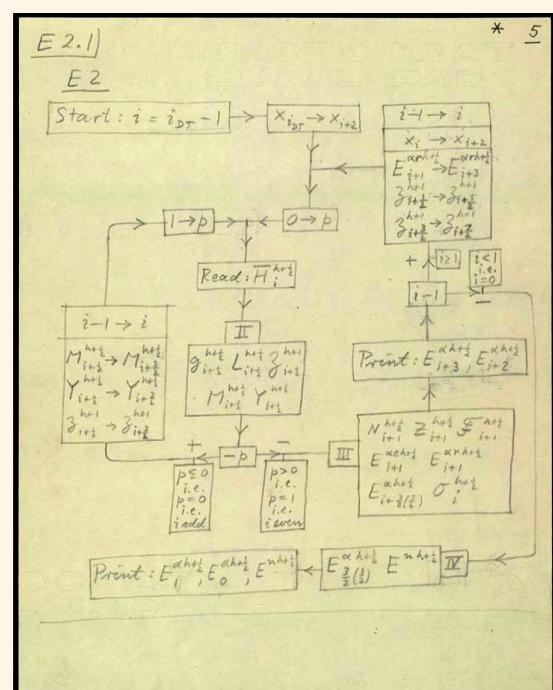
## Les premiers outils (années 1950)

### ❖ L'organigramme von Neumann - Goldstine

Wikimedia Commons  
domaine public



L'outil du programmeur  
(années 1960)



Library of Congress, USA

### ❖ La mise au point interactive Exécution pas à pas Points d'arrêt Dump de la mémoire et des registres

CC-BY-NC-SA 3.0 FR - S. Krakowiak, 2016/17

Histoire de l'informatique

7 - 36

## Les années 1956-66

### ❖ Les premiers langages de programmation

FORTRAN, COBOL, Algol 60...

Utilisés pour les applications...

... mais pas pour le logiciel de base

Améliorent les choses, mais n'éliminent pas les erreurs

### ❖ Les premiers logiciels de grande taille

$5 \cdot 10^5$  à  $5 \cdot 10^6$  lignes de code ; des dizaines, voire centaines de programmeurs

SABRE, SAGE

première notion de «cycle de vie»

OS/360

«*a multi-million dollar mistake*» (Fred Brooks)

### ❖ Les débuts de l'industrie du logiciel

Les sociétés de service

## La «crise du logiciel»

(fin des années 1960)

### ❖ Un constat

On ne maîtrise plus le développement des grands projets logiciels

délais non tenus

budget dépassé

qualité médiocre, voire échec total

### ❖ Une prise de conscience

Produire des programmes corrects est difficile

On en est encore à un stade artisanal

### ❖ Des conclusions

Il faut développer des méthodes et des outils

Il faut mieux former les gens

Il faut passer à un stade industriel

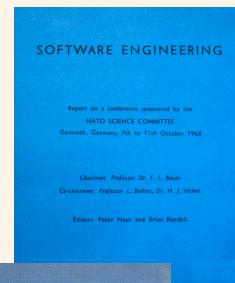
## Les conférences fondatrices (1)

### ❖ Une réaction à la crise du logiciel

Promouvoir un «génie logiciel» (*software engineering*)

Fonder la pratique sur des principes scientifiques

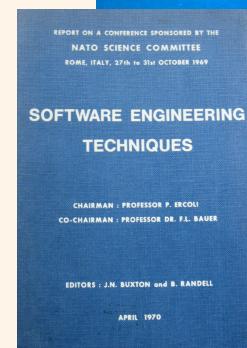
Définir les traits de cette nouvelle discipline



### ❖ Une préoccupation partagée

Initiative : division des affaires scientifiques de l'OTAN

Participation : 50 personnes scientifiques, constructeurs, sociétés de service



### ❖ Deux conférences

Octobre 1968, Garmisch

Octobre 1969, Rome

## Les conférences fondatrices (2)

### ❖ SE 68 : un état des lieux

une prise de conscience de la gravité de la «crise du logiciel»...

inadéquation des outils et pratiques

problèmes de management

problèmes de communication

quelques pistes de réflexion pour y remédier

éducation et formation

avancées conceptuelles

outils et pratiques

une proposition :  
composants réutilisables

### ❖ SE 69 : un constat d'incompréhension...

entre constructeurs et utilisateurs

entre praticiens et théoriciens

entre chercheurs et industriels

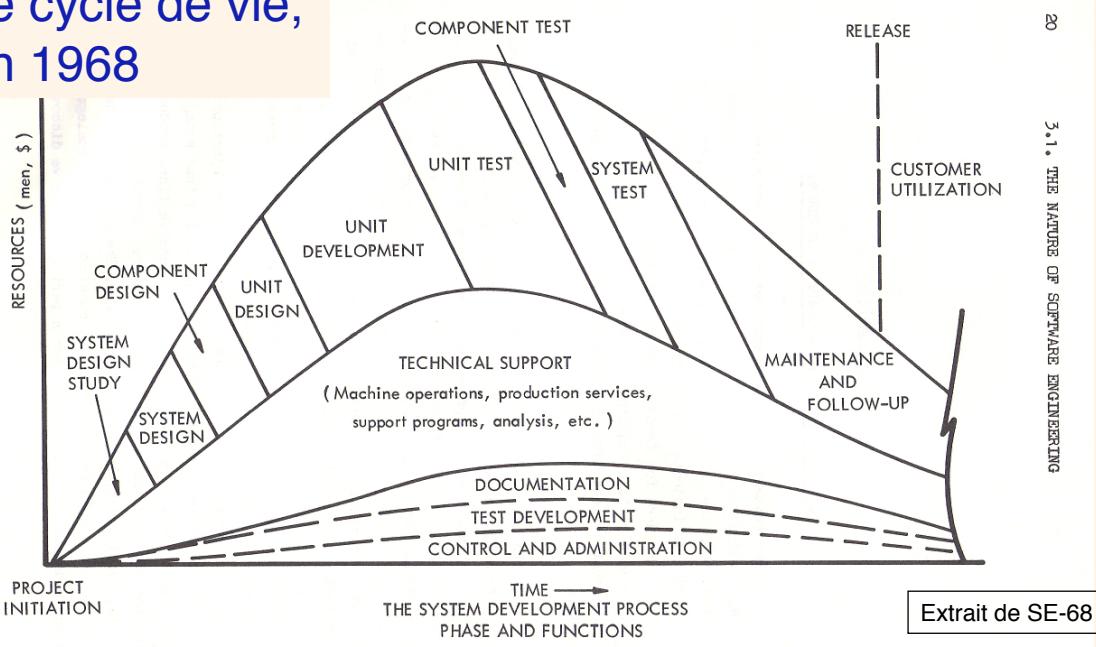
... un reflet de la situation dans le monde réel

# Les sept premières années

1968-1975

- ❖ Premières idées sur le cycle de vie
- ❖ Des avancées de la théorie...  
... mais sans impact sur la pratique Floyd, Hoare
- ❖ Des avancées architecturales...  
... pas toujours bien intégrées Dijkstra, Wirth, Parnas
- ❖ Les tout premiers outils de conception...  
... une influence encore limitée Z, VDM
- ❖ Les premières «méthodes» Warnier
- ❖ Les premiers ateliers de génie logiciel Maestro
- ❖ L'importance des facteurs humains Weinberg
- ❖ Un premier bilan Brooks

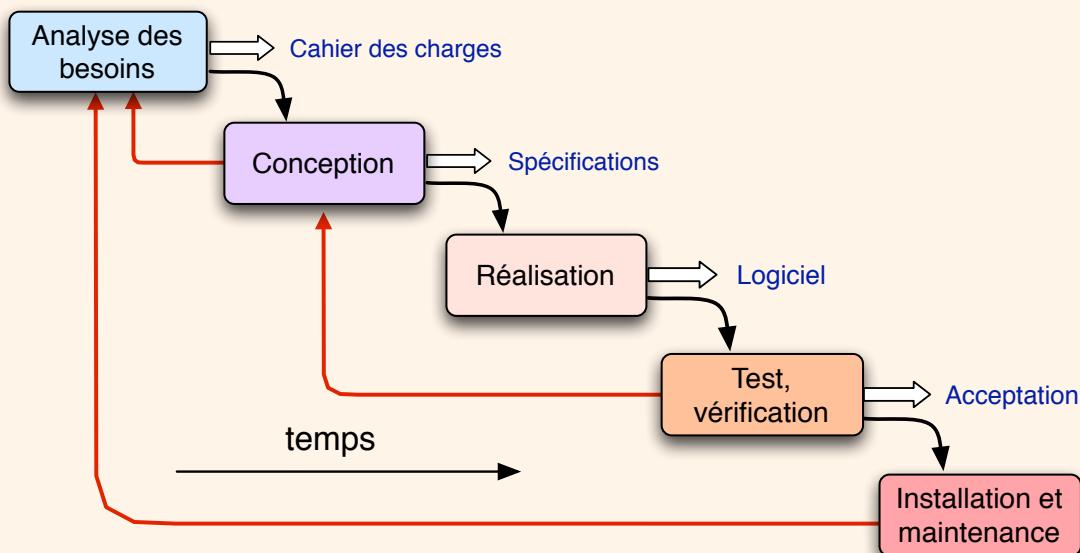
## Le cycle de vie, en 1968



- \* Faible poids de l'analyse des besoins et de la conception
- \* Fort poids des phases de test (indice d'une qualité douteuse)
- \* Retour en arrière non explicité
- \* Faible importance de la documentation, pourtant essentielle

# Une vision initiale du cycle de vie

## ❖ Le modèle de la cascade



## ❖ Faiblesses

Manque de souplesse face aux changements

Livraison tardive du logiciel, retarde le retour d'expérience

# Évolution du cycle de vie

## ❖ Objectif : une plus grande réactivité...

Cycle en V (vers 1970)

peu de différence avec cascade

Cycle incrémental (dès 1960)

une version disponible tôt

Cycle en spirale (vers 1986)

améliore le cycle incrémental

Cycle prototype

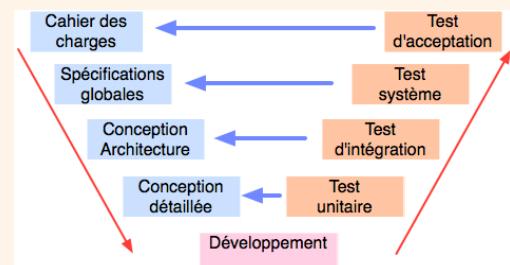
le projet peut se redéfinir en fonction de l'expérience

Cycle «agile» (vers 1995)

cycle court, participation active du client

## ❖ ... mais peu de support théorique

Effets de mode, redécouverte, peu d'évaluation



# Avancées en génie logiciel

Années 1968-75

## ❖ Avancées théoriques

Sémantique des langages (R. W. Floyd)

Logique de la programmation (C. A. R. Hoare)

Programmation «disciplinée» (E. W. D. Dijkstra)

## ❖ Avancées architecturales

Conception descendante, machines abstraites (E. W. D. Dijkstra)

Programmation modulaire (D. Parnas)

## ❖ Avancées méthodologiques

Approches scientifiques : Z (J.-R. Abrial) ; VDM (IBM Vienne)

Approches semi-empiriques (dirigées par les données)

Warnier, Merise, Jackson, SADT... : succès mitigé

## ❖ Support matériel

Ateliers de génie logiciel : Maestro, ...



## Les avancées de la théorie (1)

### ❖ Objectif : définir une sémantique des programmes

Pour répondre à la question «mon programme fait-il bien ce que je lui demande ?»

Plus ambitieux : «peut-on prouver que mon programme fait bien ce que je lui demande ?»

### ❖ Robert Floyd (1967)

*Assigning Meaning to Programs*

Le programme comme transformateur de l'état

Des assertions comme propriétés de l'état

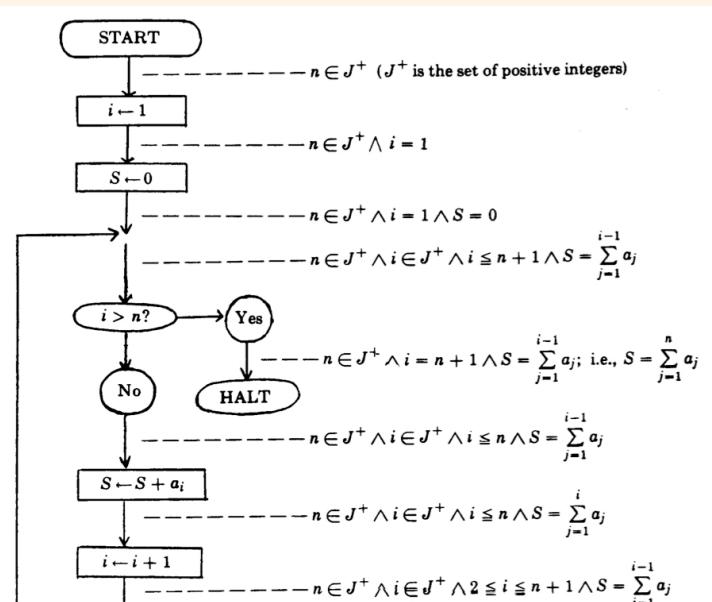


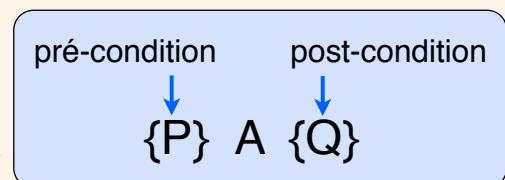
FIGURE 1. Flowchart of program to compute  $S = \sum_{j=1}^n a_j$  ( $n \geq 0$ )

## Les avancées de la théorie (2)

### ❖ La logique de Hoare (1969)

Étend le travail de Floyd

Construction de base



Construction étendue

Des règles d'inférences  
pour les compositions usuelles :

séquence

instruction conditionnelle

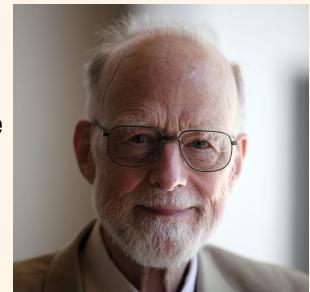
itérations

Terminaison de A

preuve séparée

«variant» décroissant avec plancher

C. A. R. «Tony» Hoare  
(1934 - )



by Rama, Wikimedia CommonsCC-BY-SA 2.0

## Les avancées de la théorie (3)

### ❖ Dijkstra : *A Discipline of Programming* (1976)

Une démarche rigoureuse de construction de programmes corrects

Une méthode générale : la séparation des préoccupations

Une notion de base : la transformation de prédicats

Soit une opération S et un prédicat final (souhaité) Q.

On définit wp(S, Q) comme le prédicat P le plus faible assurant  $\{P\} S \{Q\}$  (wp = weakest predicate)

Un outil : les commandes gardées

<expression booléenne> → <liste d'instructions>

Suite de commandes gardées : exécution non-déterministe

<exp1> → <liste1> || <exp2> → <liste2> || ...

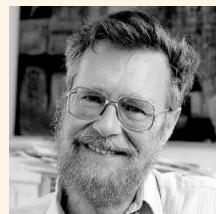
### ❖ Une influence intellectuelle profonde

mais peu d'utilisation dans la pratique industrielle

# Avancées architecturales

## ❖ Objectif : organiser l'architecture globale d'un logiciel

pour réduire sa complexité



Edsger W. Dijkstra  
(1930 - 2002)

crédit : U. of Texas at Austin

## ❖ Conception descendante

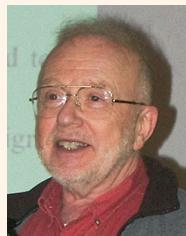
hiérarchie de «machines abstraites»

réalisation en logiciel raffinement progressif

## ❖ Décomposition en modules

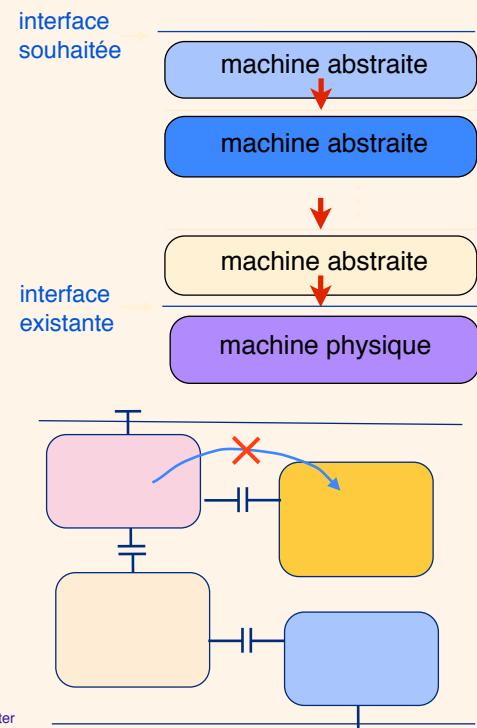
interface visible,  
réalisation cachée

tout doit passer  
par l'interface



David L. Parnas  
(1941 - )

Wikimedia Commons, CC-BY-SA 3.0 H. Baumeister



# Avancées méthodologiques (1)

## ❖ Objectif : définir des méthodes de conception et de développement visant à produire du logiciel correct

Deux démarches : scientifiques vs semi-empiriques

## ❖ Approches «scientifiques»

Fondées sur la logique de Hoare

La méthode Z (Jean-Raymond Abrial, 1974)

Une démarche rigoureuse de spécification, fondée sur les ensembles

Un raffinement avec preuves (formelles ou non)

Précurseur de la méthode B

La méthode VDM (*Vienna Development Method*, 1976)

Au départ : génération d'un compilateur d'une partie de PL/1 à partir de la description formelle du langage

Un langage de spécification

Une méthode et des outils de développement

## Avancées méthodologiques (2)

### ❖ Approches «semi-empiriques»

Utilisées surtout en informatique de gestion

Une pratique codifiée

analyse fonctionnelle (quoi ?)

analyse organique (comment ?)

Une démarche dirigée par l'organisation des données (fichiers, BD)

### ❖ Exemples [avant introduction des objets]

La première méthode : Warnier (1970)

s'applique bien aux opérations sur fichiers (tri, fusion)

Une méthode d'analyse et de modélisation : Merise (1975-80)

Autres : Jackson, SADT, DeMarco, ...

### ❖ Conclusion

NOMBREUSES «méthodes», aucune ne s'impose

Un succès mitigé, manque de bases scientifiques

## Méthodes de programmation

### ❖ Objectif : réduire la complexité des programmes

### ❖ Deux voies

Améliorer l'outil (le langage de programmation)

Les langages à objets (voir plus loin)

Améliorer la pratique du langage

La programmation structurée

### ❖ La «programmation structurée»

Au départ : éliminer le *go to* (Dijkstra, 1968)

Utiliser les constructions plus synthétiques (*if... then... else, while, case...*)

Utiliser une démarche descendante, au niveau du code (Wirth, 1971)

Une démarche souvent mal comprise

application de règles mécaniques, sans effort  
vers une «preuve», même informelle

## Les facteurs humains

### ❖ Le coefficient individuel

Les expériences de Sackman, Erikson et Grant (1968)

Mesure de l'influence des outils sur la productivité des programmeurs

En fait les différences individuelles prédominent (facteur > 10)

### ❖ Psychologie et sociologie de la programmation

Le livre de Gerald Weinberg : *Psychology of Computer Programming* (1971) ; nouvelle édition, 1976

### ❖ L'organisation

Dans un grand projet, l'organisation et le management sont plus importants que les aspects techniques

Le rôle central de la communication et de la documentation

Autonomie et auto-organisation

## Les ateliers de génie logiciel

### ❖ Objectif : intégrer les outils dans un ensemble organisé...

... pour éviter les incohérences

... pour augmenter le confort et la productivité

réduire le temps de réaction

### ❖ Les outils

gérant de versions, éditeurs, compilateurs, metteurs au point, documentation, ...

### ❖ Le premier atelier : Maestro (1975)

Développé par SoftLab (Munich)

Support : système dédié (ordinateur + terminaux)

Un grand succès

### ❖ La suite

Ateliers portables sur tout système

## Fred Brooks : un premier bilan (1975)

### ❖ *The Mythical Man-Month*

Une revue critique de la situation du génie logiciel  
Quelques points saillants

### ❖ Si le projet est en retard, ajouter de la force de travail

**Faux !** Cela ne fait qu'aggraver le problème

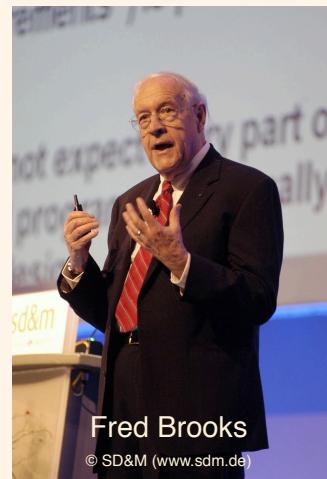
### ❖ L'effet «second système»

Une confiance injustifiée, facteur de risque

### ❖ L'intégrité conceptuelle, qualité principale d'un projet

L'expérience de l'OS/360  
Le rôle central de l'architecte  
L'importance de la documentation

### ❖ Prévoyez de mettre une version à la poubelle : vous le ferez de toutes façons



## Autres mythes des années 70

### ❖ Le mythe de la solution «par les outils»

On résoudra les problèmes en créant des outils plus raffinés

**Très discutable !** la qualité des équipes est le facteur dominant

### ❖ Le mythe de la «programmation automatique»

Les programmes du futur seront créés automatiquement  
Le métier de programmeur va disparaître

**Faux !** Le métier va se transformer, mais dans le sens d'une plus grande qualification

## *The Mythical Man-Month*, vingt ans après...

nouvelle édition (1995)

### ❖ L'essentiel vs l'accidentel

L'essentiel : la complexité intrinsèque

L'accidentel : tout le reste...

### ❖ Ce qui a changé depuis 1975

L'environnement de travail (WIMP, réseaux, ateliers)

Une avancée conceptuelle : les objets, la modularité

Les cycles de vie (incrémental, prototype, ...)

Les progiciels

L'usage de l'informatique

### ❖ Ce qui n'a pas changé

Le rôle central des personnes et de l'organisation

L'importance de l'intégrité conceptuelle et du rôle de l'architecte

Conclusion : *No Silver Bullet*

## La situation en 1995

### ❖ L'architecture du logiciel

Une tentative de formalisation (composants, connecteurs, etc)

### ❖ Les objets, facteur de structuration

### ❖ UML, l'unification des modèles

### ❖ Les patrons et canevas

Modèles et code réutilisables

### ❖ Les ateliers

### ❖ Cahier des charges et spécifications

### ❖ Des outils et méthodes fondés sur la théorie

*Model Checking*, analyse statique et interprétation abstraite, B  
mais influence encore limitée

# Situation et perspectives du génie logiciel

## ❖ 20 ans après 1995...

*No Silver Bullet !*

Des défis persistants

sécurité

tolérance aux fautes

adaptation

Nouveaux modes de construction de logiciel

«La cathédrale et le bazar»

*L'open source*

## ❖ La disjonction persiste entre théorie et pratique

Des avancées dans les pratiques, avec peu de bases théoriques

Des avancées dans la théorie, mais peu d'impact pratique à court ou moyen terme

Quelques espoirs tout de même

## ❖ Vers une refondation ?