

Éléments d'histoire de l'informatique

Sacha Krakowiak

Université Grenoble Alpes & Aconit

7. Les débuts du génie logiciel

CC-BY-NC-SA 3.0 FR

Le génie logiciel

❖ Les motivations

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

... mais il a fallu longtemps pour s'en rendre compte

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».

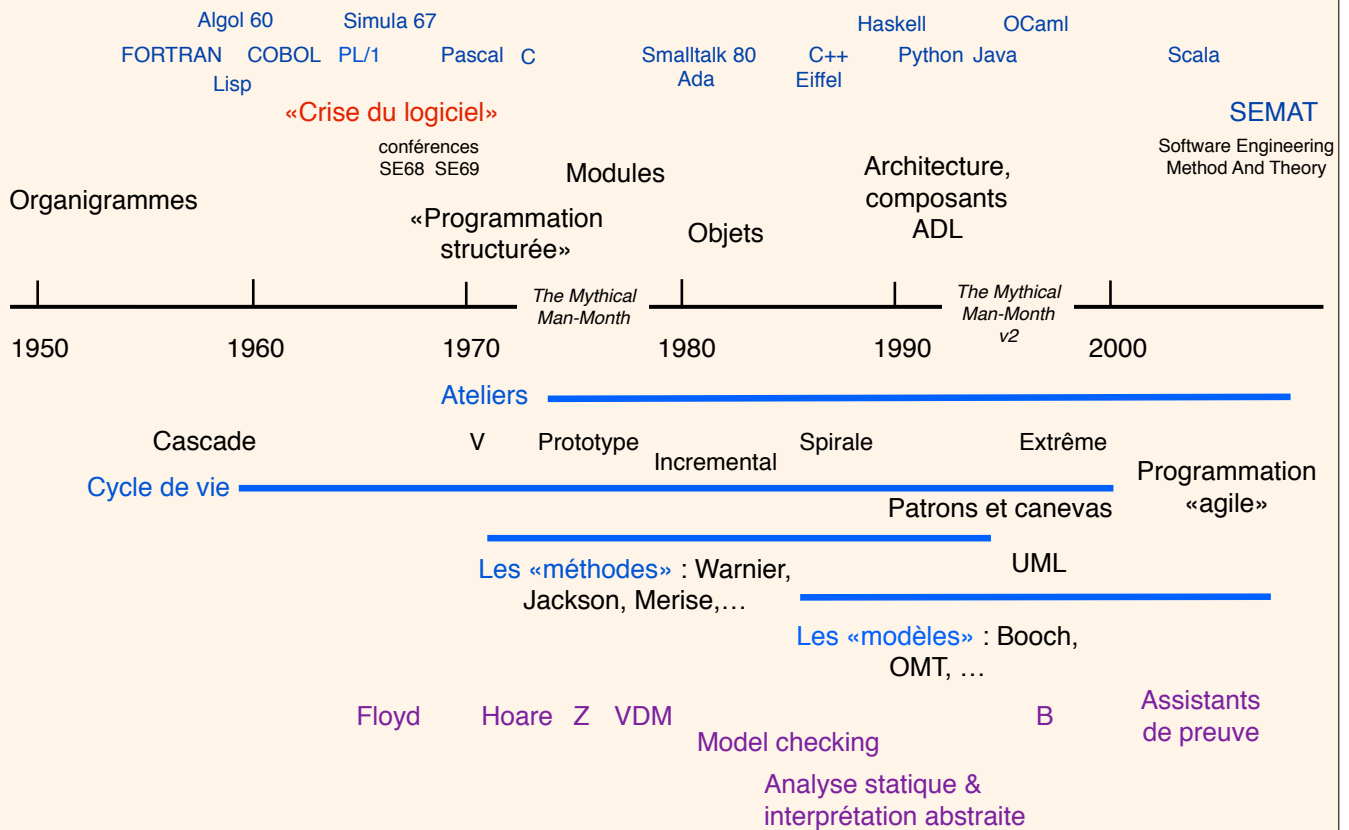
Maurice Wilkes, 1949

❖ 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

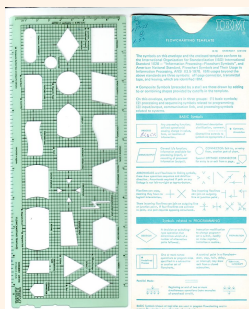


Les premiers outils (années 1950)

❖ L'organigramme

von Neumann - Goldstine

Wikimedia Commons
domaine public



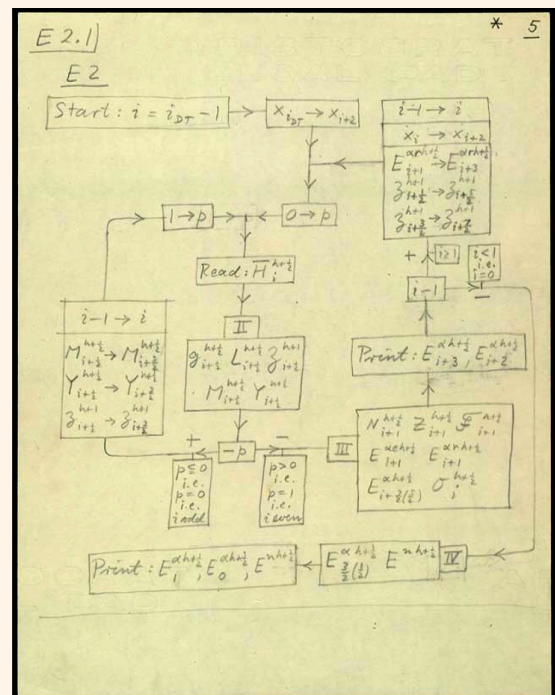
L'outil du programmeur
(années 1960)

❖ La mise au point interactive

Exécution pas à pas

Points d'arrêt

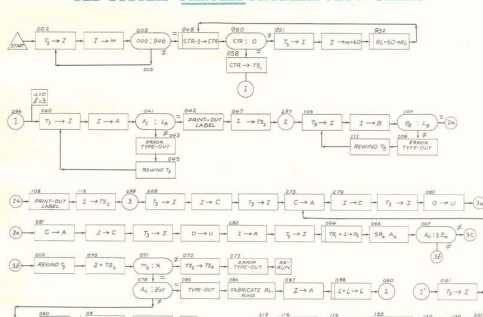
Dump de la mémoire et des registres



Library of Congress, USA

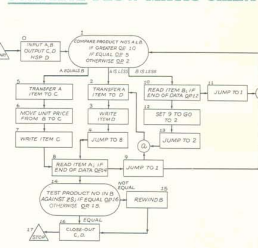
SIMPLIFIES FLOW CHARTING AND CODING

OLD SYSTEM—PARTIAL PROBLEM FLOW CHART

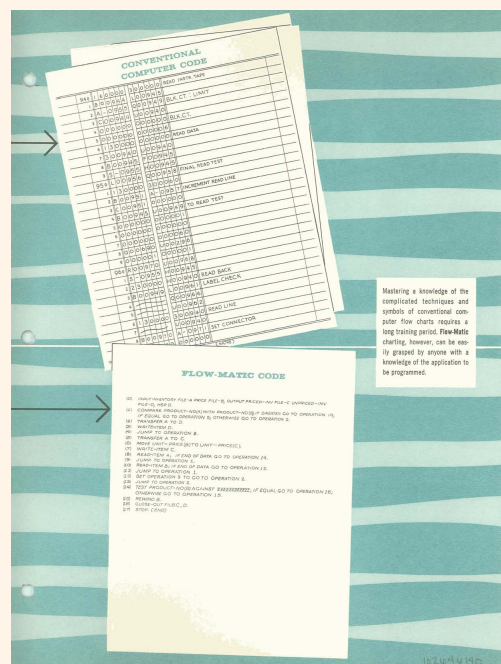


The English-language pseudo-code of Flow-Matic causes the Univac system to automatically compile and write its own computer code. Thus, both flow charts and codes are made intelligible to the non-programmer as well as to the programmer. Many weeks of extensive training and the countless hours spent in coding each application are hurred by this revolutionary new development.

SAME PROBLEM—COMPLETE FLOW-MATIC CHART



UNIVAC Flowmatic (1957)



Images courtesy of Computer History Museum

Les années 1956-69

❖ 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

5.10^5 à 5.10^6 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, le «dépaquetage» (*unbundling*, IBM 1969)

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

L'OS/360 (1963-67)

- ◆ Un système gros et complexe, en langage assembleur (> 10⁶ lignes de code)
- ◆ Un domaine nouveau (multiprogrammation)
- ◆ Une conception défailante
- ◆ Des retards croissants, mal traités

Au total :

- ◆ Un an de retard
- ◆ Budget \$500M (X 4)
- ◆ Système de mauvaise qualité, difficile à maintenir (1 000 bugs/version)

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

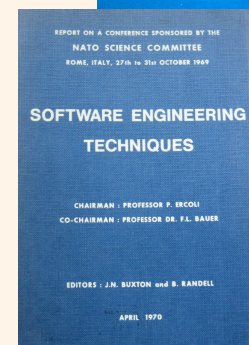
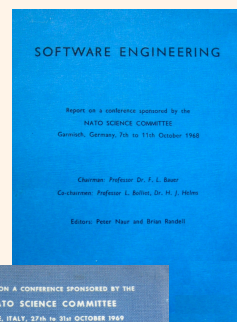
Participation : 50 personnes

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

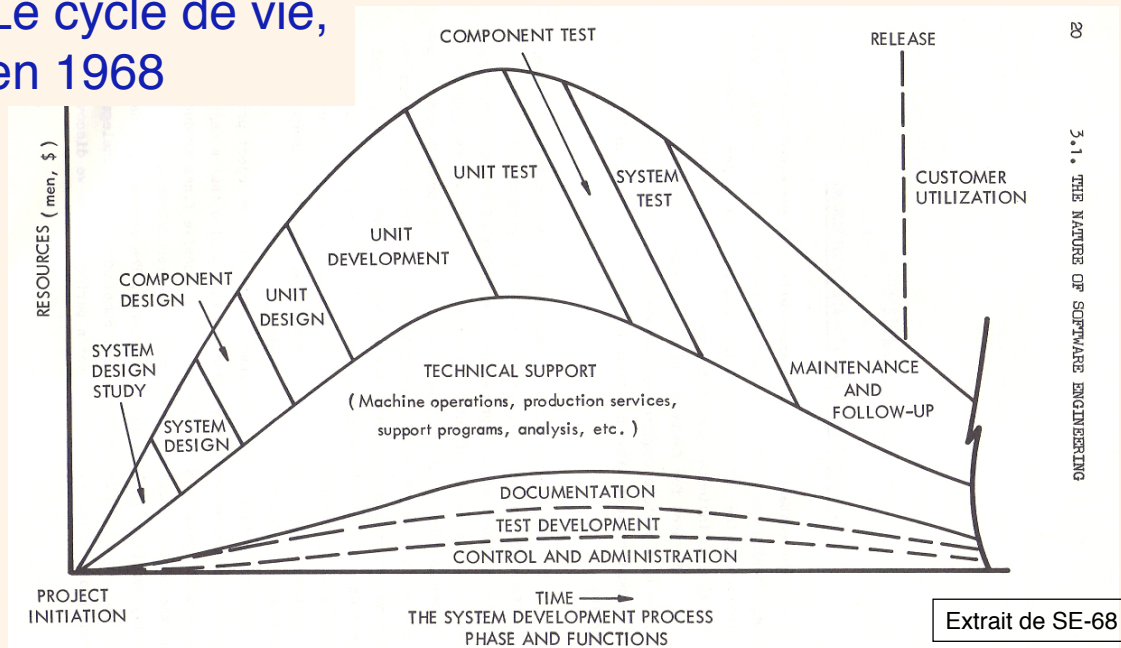
❖ L'importance des facteurs humains

Weinberg

❖ Un premier bilan

Brooks

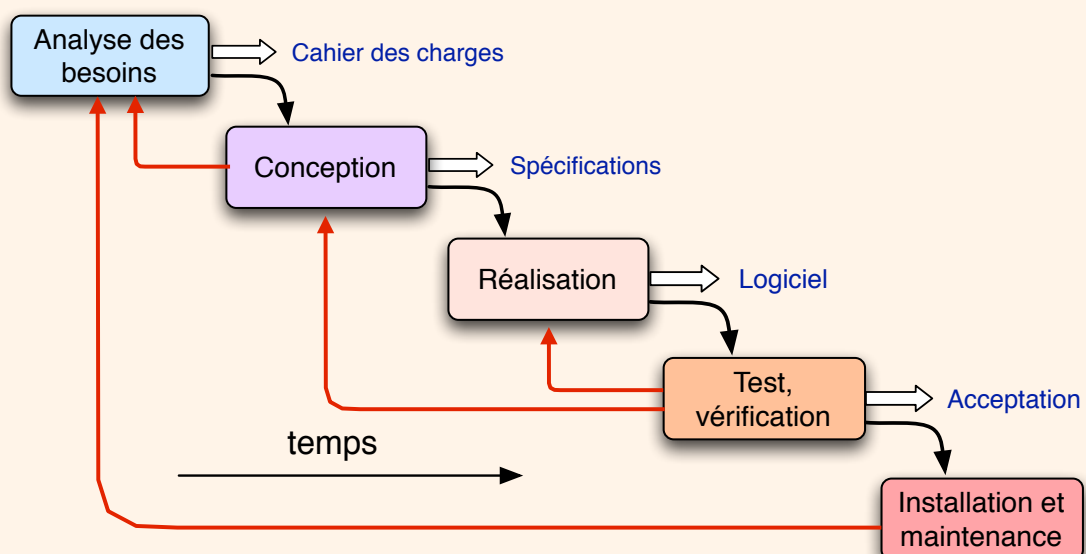
Le cycle de vie, en 1968



- * 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 explicite
- * 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

progrès par itération

Cycle en spirale (vers 1986)

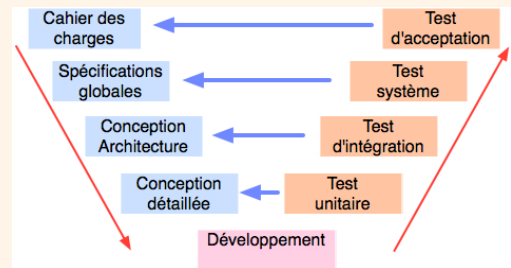
améliore le cycle incrémental

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 ?»

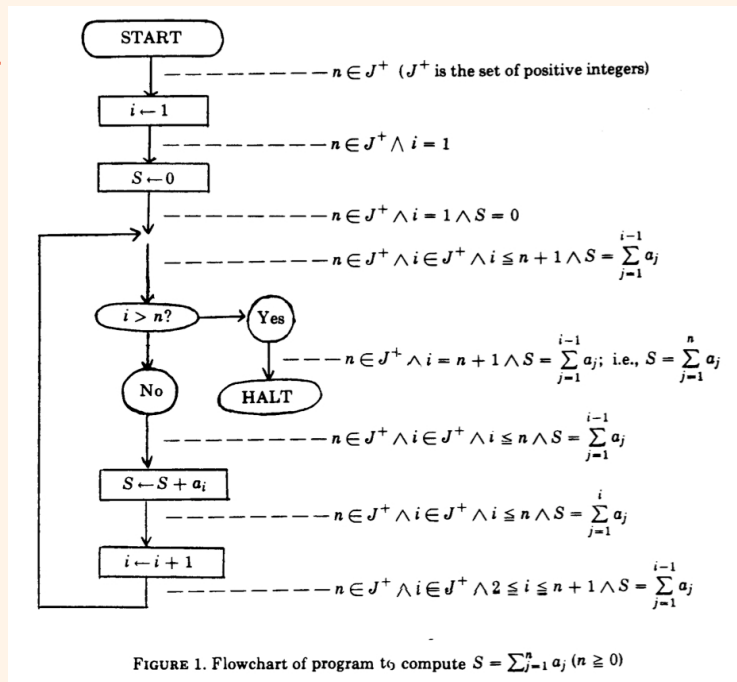
Plus ambitieux : «peut-on prouver que mon programme fait bien ce que je lui demande ?»

❖ Robert Floyd (1967)

Assigning Meaning to Programs

Le programme comme transformateur de l'état

Des assertions comme propriétés de l'état



Les avancées de la théorie (2)

❖ La logique de Hoare (1969)

Étend le travail de Floyd

Construction de base

pré-condition

post-condition

$\{P\} A \{Q\}$

Construction étendue

invariant

$\{P \wedge I\} A \{Q \wedge I\}$

Des règles d'inférences pour les compositions usuelles :

séquence

instruction conditionnelle

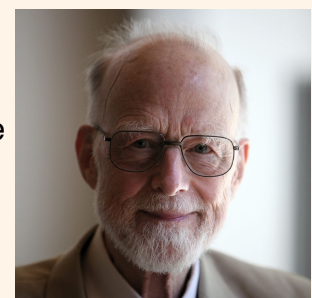
itérations

Terminaison de A

preuve séparée

«variant» décroissant avec plancher

C. A. R. «Tony» Hoare
(1934 -)



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

$\langle \text{expression booléenne} \rangle \rightarrow \langle \text{liste d'instructions} \rangle$
Suite de commandes gardées : exécution non-déterministe
 $\langle \text{exp1} \rangle \rightarrow \langle \text{liste1} \rangle \parallel \langle \text{exp2} \rangle \rightarrow \langle \text{liste2} \rangle \parallel \dots$

❖ Une influence intellectuelle profonde

mais peu d'utilisation dans la pratique industrielle

Avancées architecturales

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

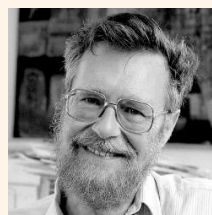
pour réduire sa complexité

❖ Conception descendante

hiérarchie de «machines abstraites»

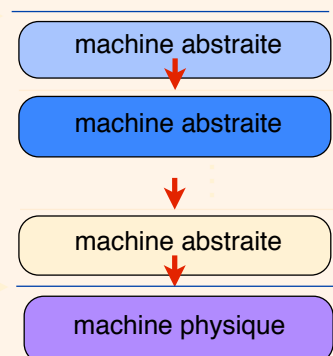
réalisation en logiciel

raffinement progressif



Edsger W. Dijkstra
(1930 - 2002)
crédit : U. of Texas at Austin

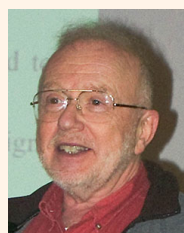
interface
souhaitée



❖ Décomposition en modules

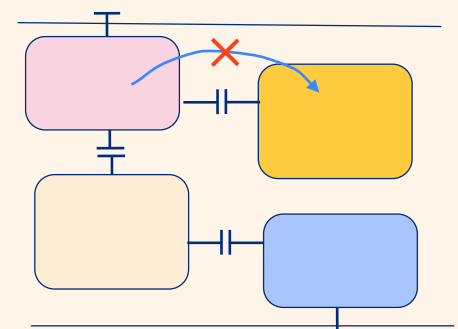
interface visible,
réalisation cachée

tout doit passer
par l'interface



David L. Parnas
(1941 -)

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



Avancées méthodologiques

- ❖ 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

Méthodes de conception

- ❖ 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... do, *case of...*)

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, 1996

❖ 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

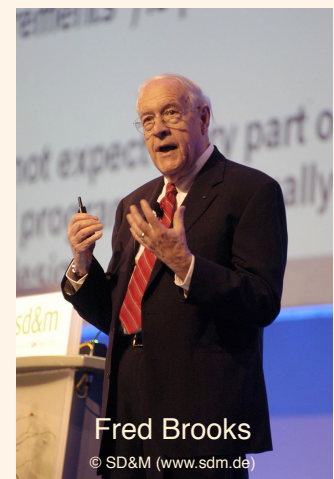
❖ L'effet «second système»

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

Après 1975...

❖ Avancées architecturales

Modularité, composition

❖ Avancées méthodologiques

Méthodes et outils de conception

❖ Avancées dans l'organisation

Nouvelles pratiques du cycle de vie

❖ En quête de la validité

Théorie et pratique

La modularité

❖ Un objectif ambitieux

Doug McIlroy (1968, Software Engineering Conf.)
Une bibliothèque de composants réutilisables

❖ Avancées méthodologiques

David Parnas (1972)
Critères de décomposition

Isolation des
choix de
conception

❖ En quête d'outils

DeRemer & Kron (1974)
Programmation globale vs programmation détaillée

Module
Interconnection
Languages

❖ Une nouvelle discipline

Architecture du logiciel (années 1990)

Composants
logiciels

❖ Un problème toujours ouvert...

Architecture logicielle

❖ Description globale d'un système

Composants, connecteurs, configurations
Règles de composition
Langage de description d'architecture (ADL)

Base pour construction, vérification et preuve, administration

❖ L'architecture logicielle, base pour l'administration

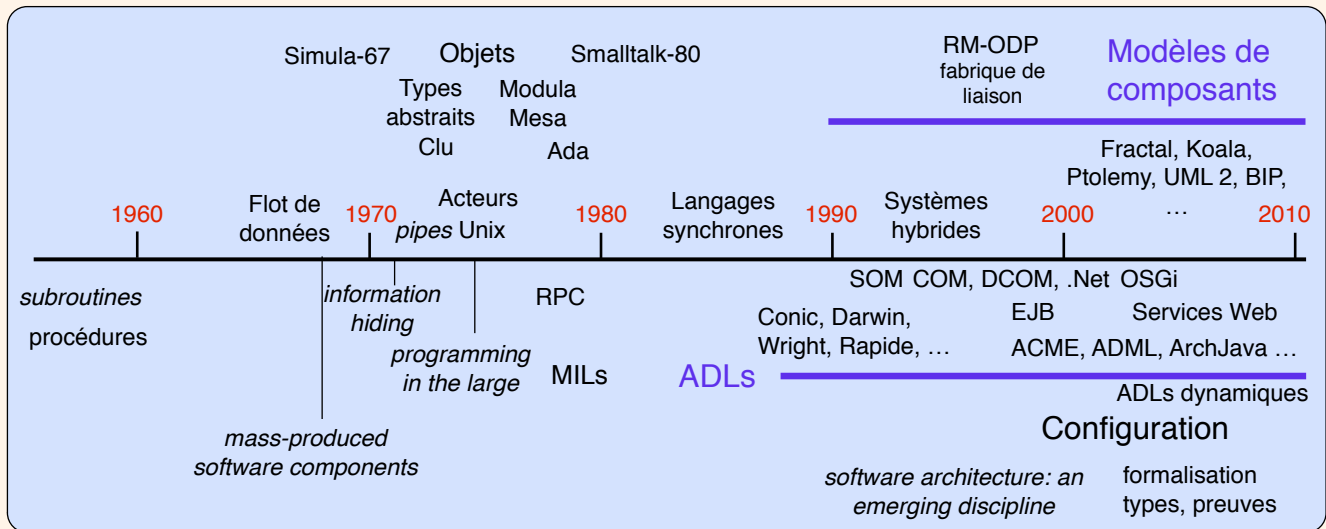
Configuration
Déploiement
Reconfiguration

Mary Shaw, David Garlan, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996

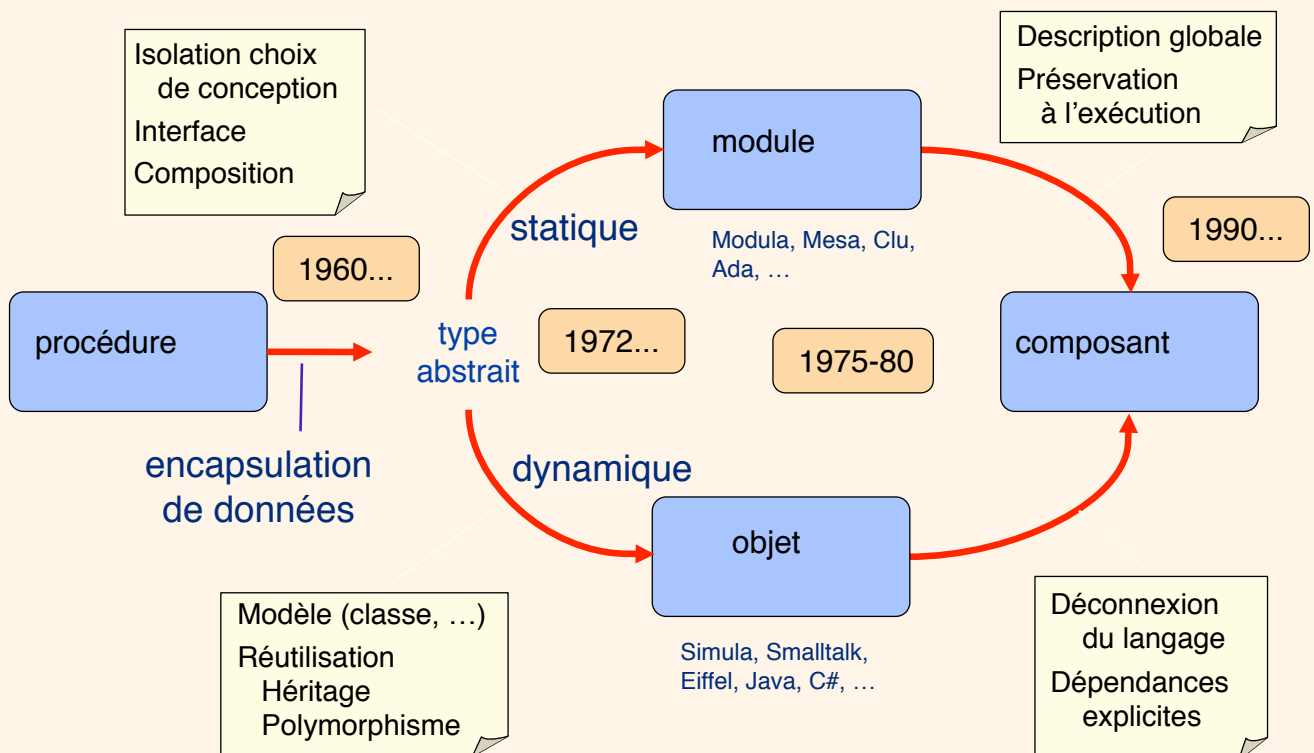
❖ Vers une formalisation de l'administration de systèmes

Réaction autonome aux pannes et surcharges

Une brève histoire de la (dé)composition



Des procédures aux composants



Patrons et canevas

❖ Patrons de conception (*design patterns*)

Ensemble de règles de conception (définition d'éléments, modes de combinaison des éléments, modes d'emploi) permettant de répondre à une classe de besoins identifiés

Un patron définit des règles de conception, non une mise en œuvre spécifique de ces règles

❖ Canevas logiciels (*software frameworks*)

Un canevas est un «squelette» de programme réutilisable (et paramétrable) pour une famille d'applications

Dans le cadre des langages à objets : un ensemble de classes destinées à être adaptées à des cas d'utilisation spécifiques

❖ Un outil de description et de conception : UML

Des schémas pour différentes vues d'un logiciel

En quête de la validité ...

Pour un composant élémentaire

❖ Écrire l'algorithme, et se convaincre qu'il est correct

Par le test

Par la vérification

Par la preuve

❖ Construire en même temps l'algorithme et la preuve

❖ Engendrer l'algorithme à partir des spécifications

Par un procédé dont on a prouvé la validité

Pour un système complexe

❖ Composer les preuves

Déterminer les propriétés du système à partir de celles de ses composants et des règles de composition

Vérification par analyse statique

Idée : déterminer des propriétés **dynamiques** d'un système, **sans exécuter** son programme

❖ *Model checking* (Clarke, Emerson, Sifakis)

Modéliser le système par un graphe de transition d'états

Vérifier que le modèle satisfait une spécification (en logique temporelle)

❖ *Interprétation abstraite* (Cousot)

Modéliser l'évolution du système par ses traces d'exécution

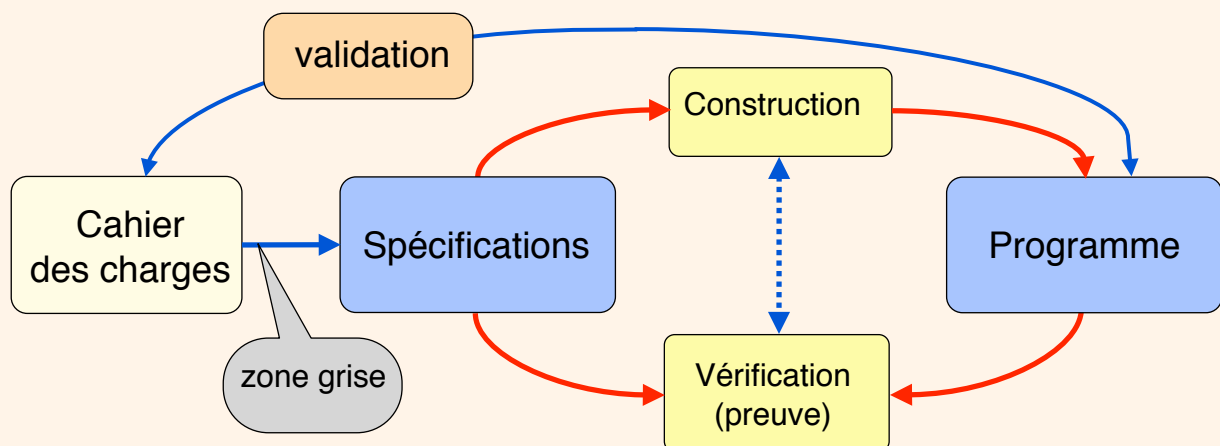
Définir des sémantiques à divers niveaux d'approximation

Résoudre l'équation (de point fixe) traduisant une propriété

❖ *Difficultés communes*

Explosion des états

Spécification, construction, preuve



❖ *Ingrédients de la spécification*

Une base formelle (logique)

Un langage d'expression

Des outils d'aide à la construction

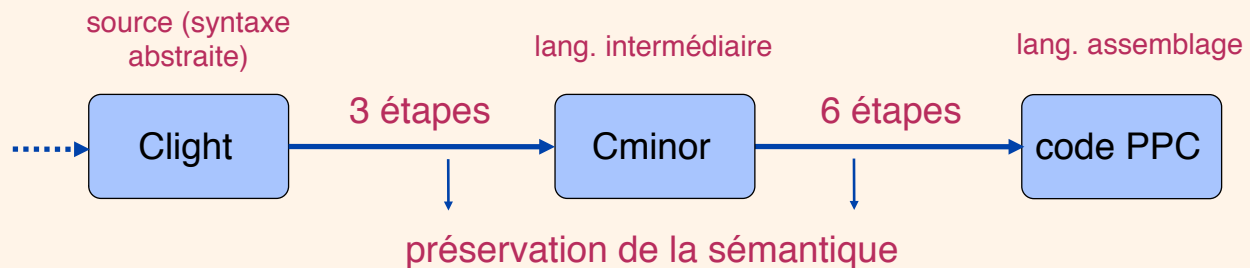
Des outils d'aide à la preuve

Pendant longtemps :
pas de base formelle
langage naturel
pas ou peu d'outils

Une piste prometteuse :
les assistants de preuves

Vérification d'un compilateur

❖ Un compilateur pour un large sous-ensemble de C



❖ Preuve et construction (avec Coq) pour chaque étape

Preuve que le comportement du programme objet est identique à celui du programme source (si pas d'erreur)

Génération d'un programme OCaml traduisant les spécifications

L'efficacité du code produit est très acceptable (-12% / gcc-02)

Xavier Leroy, "Formal Verification of a Realistic Compiler", *Comm. ACM*, 52:7, July 2009

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)

Les avancées architecturales

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

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 ?



Software Engineering Method And Theory

Pour aller plus loin (1)

La conférence fondatrice et sa suite (textes complets disponibles en : <http://homepages.cs.ncl.ac.uk/brian.randell/NATO/index.html>)

- Naur, P. and Randell, B., (Ed.). Software Engineering: Report on a Conference sponsored by the NATO Science Committee, Garmisch, Germany, 7th to 11th October 1968, Brussels, Scientific Affairs Division, NATO, January 1969, 231 p.
- Buxton, J.N. and Randell, B., (Ed.). Software Engineering Techniques: Report on a Conference sponsored by the NATO Science Committee, Rome, Italy, 27th to 31st October 1969, Brussels, Scientific Affairs Division, NATO, April 1970, 164 p.

Mythes et réalités

- Brooks, F. P. The Mythical Man Month, Addison-Wesley (1975). Anniversary Edition, 1995.

Cycle de vie

- Royce, W. [Managing the development of large software systems](#), Proceedings of IEEE WESCON 26 (August 1970) : 1–9.

Ouvrages généraux sur la conception

- Alexander, C. Notes on the Synthesis of Form, Harvard University Press (1964).
- Simon H.A. The Sciences of the Artificial, MIT Press (1969 ; rééditions : 1981, 1996).

Architecture du logiciel, modularité

- Dijkstra, E. W. [The structure of the THE multiprogramming system](#), Communications of the ACM 11 (5): 341–346 (1968).
- McIlroy, M. D. Mass produced software components, in [SE'68].
- Parnas, D. L. [On the criteria to be used in decomposing a system into modules](#), Communications of the ACM 15 (12): 1053–58 (December 1972).
- Liskov, B. and Zilles, S. [Programming with abstract data types](#), Proc. ACM SIGPLAN Symposium on Very High Level Languages, Santa Monica, pp. 50-59 (1974).
- DeRemer, F. and Kron, H. [Programming-in-the large versus programming-in-the-small](#), Proceedings of the International Conference on Reliable Software (ACM), Los Angeles, pp. 114—121 (1975).

Pour aller plus loin (2)

Les premiers articles sur la « bonne programmation »

- Dijkstra, E. W. [Go to statement considered harmful](#), Communications of the ACM, 11 (3): 147–148.
- Dijkstra, E. W. [Notes on structured programming](#), in Dahl, O.J., Dijkstra, E.W. and Hoare, C.A.R. Structured Programming, Academic Press (1972).
- Wirth, N. [Program development by stepwise refinement](#), Communications of the ACM, 14, 4, (Apr 1971) 221-227 (1971).

Les premiers articles sur la sémantique des programmes

- Floyd, R. W. [Assigning meaning to programs](#), Proceedings of Symposia in Applied Mathematics, American Mathematical Society, Vol. 19, pp. 19-32 (1967).
- Hoare, C. A. R. [An axiomatic basis for computer programming](#), Communications of the ACM, 12(10):576–580 (1969).

Spécifications, méthodes formelles

- Abrial, J. R. Data semantics, IFIP Working Conference on Database Management, Klimbie; Koffeman, ed., North Holland, pp. 1-60 (1974). [NB : les bases du langage Z]
- Bjørner, D. and Jones C. B. (1978). The Vienna Development Method: The Meta- Language, Lecture Notes in Computer Science 61. Springer (1978).

La programmation comme activité humaine

- Sackman, H., Erikson W. J., Grant E. E. Exploratory experimentation studies comparing on-line and off line programming performance, Communications of the ACM, 11(1): 3-11 (1968).
- Weinberg, G. M., The Psychology of Computer Programming, Van Nostrand Reinhold, 1971. Silver Anniversary Edition: Dorset House Publishing, 1998.