Les Langages Informatiques

Les évolutions

Michel Riguidel (riguidel@enst.fr)

Table des Matières

Table des Matières 2
Segmentation Informatique 3
    Autrefois 3
    Maintenant 3
Informatique 3
    Informatique, Ordinateur, Logiciel, Langage 3
    Langage 3
Typologie des langages 4
    Mathématique & Théorie des ensembles 4
    Langages de programmation impératifs 4
    Langage procédural 5
    Langage programme orienté objet Eiffel, Java 5
    Langages fonctionnels (Lisp pur, ML) et langages logiques (Prolog) 6
Diversité des langages 6
    La jungle des langages 6
    Taxinomie des langages 6
    Évolution en fonction des évolutions du matériel, de l'Internet ? 7
Conception d'un langage 7
    Puissance d'expression 7
    Simplicité et orthogonalité 8
    Implémentation 8
    Détection d'erreurs et correction d'erreurs 9
    Correction et standards 9
Développement d'un programme 10
    Maîtrise du langage par le développeur 10
    Productivité et Génie logiciel 10
    Modèle de développement 10
    Langages ou systèmes 11
Historique 11
    Survol historique 11
    Machines du début 13
    Premiers langages scientifiques 13
    Langages de traitement de données de gestion 14
    Langages généraux 15
    Développement interactif de programmes 15
    Langages spécifiques 16
    Langages de programmation systèmes 17
    Modules, classes, types de données abstraites, objets 18
    Langage Fonctionnel et Logique 21
Syntaxe et Sémantique 22
    Syntaxe BNF 22
    Sémantique 23
Structure des programmes 23
    Procédures, fonctions, méthodes 24
    Structures de données 24
    Conclusions 25

Segmentation Informatique

Autrefois

Découpage

Matériel et Logiciel : clivage traditionnel hardware-software.

Logiciels

Système d'exploitation avec des applications.
Développement d'applications à partir d'un langage de programmation : compilateur (pas d'éditeur de texte ni d'environnement de développement, au début).

Maintenant

Découpage

Matériel, Logiciel, Contenu (signal, donnée, information, document, connaissance)
Terminal (fixe, mobile) & Réseaux.
L'informatique se conçoit en réseau : apparition dans le domaine de la recherche de métacomputing (concept de grille de calcul), pervasive computing (informatique diffuse).

Logiciels

Système d'exploitation, Browser (Butineur asynchrone), Player (interface synchrone), Middleware (Intergiciels) et protocoles.
Environnement de développement (éditeur, compilateur, bibliothèque de composants, debugger symbolique, framework, …).

Informatique

Informatique, Ordinateur, Logiciel, Langage

Un ordinateur est un outil qui résout des problèmes au moyen de programmes ou logiciels développés dans un (ou plusieurs) langages.
La taille et la complexité des logiciels ont augmenté.
De manière indicative, un logiciel important possède, en moyenne, la taille suivante :

- ✓ Année 70 : 10 000 lignes de code
- ✓ Année 80 : 50 000 lignes de code : début des interfaces graphiques (clavier, souris, écran)
- ✓ Année 90 : 100 000 lignes de code
- ✓ Année 00 : 1 000 000 lignes de code : Netscape communicator, Word, Excel, … occupent 5 Mo.

Langage

Passage de la langue naturelle via le langage de haut niveau au langage machine.
Langage : syntaxe et sémantique => Compilateur, Interpréteur, Analyseur sémantique.
Syntaxe : définie en BNF (Backus-Naur Form)
Sémantique : dépend du langage (définit les opérations algébriques, etc.)
Un système d'exploitation gère le temps, l'espace (mémoire, disque) et les entrées-sorties.
Une application va donner du sens, en transformant un contenu (des entrées) en un autre contenu (des sorties), par un calcul informatique en prenant de la place et en prenant du temps. La valeur ajoutée dépend de la force du langage et de l'intelligence de l'application.
Au fil du temps se sont développées les applications interactives qui agissent en temps réel sur l'environnement ou interagissent avec l'utilisateur.

Un langage informatique est un langage destiné à décrire l'ensemble des actions consécutives qu'un ordinateur doit exécuter. Un langage informatique est une manière pragmatique de donner des instructions à un ordinateur.

Un langage informatique est rigoureux: à une instruction correspond une action du processeur. Le langage utilisé par le processeur est le langage machine. Il s'agit d'une suite de 0 et de 1 (en hexadécimal), pas compréhensible facilement par le commun des mortels. Il est donc plus pratique de trouver un langage intermédiaire, compréhensible par l'homme, qui sera ensuite transformé en langage machine pour être exploitable par le processeur. L'assembleur est un langage proche du langage machine mais il permet déjà d'être plus compréhensible. Toutefois un tel langage dépend étroitement du type de processeur utilisé (chaque type de processeur peut avoir son propre langage machine).
Ainsi un programme développé pour une machine ne pourra pas être « porté » sur un autre type de machine. Pour pouvoir l'utiliser sur une autre machine il faudra alors parfois réécrire entièrement le programme.
Un langage informatique a donc plusieurs avantages:

- ✓ il est plus facilement compréhensible que le langage machine
- ✓ il permet une plus grande portabilité, c'est-à-dire une plus grande facilité d'adaptation sur des machines de types différents.

Typologie des langages

Mathématique & Théorie des ensembles

En mathématique

Définir des ensembles avec des éléments et des opérations et déterminer les propriétés.
Démontrer des théorèmes, des propriétés.
Montrer qu'un élément existe ou qu'une propriété est vraie dans le cas le plus général, sans nécessairement exhiber cet élément (voir les théorèmes d'existence).

En informatique

On va définir des structures de données qui ont des natures complexes, des types hétérogènes, etc.
On veut obtenir un résultat : par exemple, on ne veut pas prouver qu'il existe des nombres premiers, on veut les calculer.
Les langages informatiques vont être conçus pour écrire des programmes qui permettent de définir des structures de données complexes et de mettre en œuvre des calculs et algorithmes efficaces.
Il existe trois grandes écoles de programmations: la programmation impérative, la programmation fonctionnelle et la programmation logique.

1) La programmation impérative est basée sur l'effet de bord, plus connue sous le nom d'affectation. Le programme se déroule en écrasant les valeurs d'une zone de mémoire par une nouvelle valeur, issue d'un calcul. Ce style de programmation garantit de bonnes performances en terme de temps d'exécution.

2) La programmation fonctionnelle est basée sur le lambda calcul. Les effets de bord y sont vus d'un mauvais oeil. On préfère utiliser le concept d'environnement. Un environnement est un ensemble dynamique de paires [nom, valeur]. Une fois qu'une paire est ajoutée à l'environnement elle ne doit (normalement) plus être modifiée. Pour changer la valeur associée à un nom, il suffit de rajouter une nouvelle paire. La paire la plus récemment définie prévaut sur les autres. Quand certaines parties de l'environnement sont devenue obsolètes, elles sont simplement supprimées de la mémoire.

3) La programmation logique est basée sur des résultats mathématiques ; ce mode de programmation est celui qui est le moins utilisé.

Langages de programmation impératifs

Fortran, COBOL, C, C++, Pascal, Ada et Java.
Le programme se déroule en changeant la valeur des variables et les attributs des objets à l'aide d'instructions et d'assignations. L'ordinateur est une machine de von Neumann doté d'un espace géographique (la mémoire vive, les disques) qui a des positions individuelles avec des adresses. Le calcul a un début et une fin et ne prend pas une place illimitée : machine de Turing.
Le calcul est déterministe : le programme donne le même résultat sur toute autre machine (Cependant attention à la précision limitée sur les nombres).

Il existe des entiers (-N à + N) et des flottants (rationnels qui ont une précision donnée de – R à + R).
La plage dépend des ordinateurs (8 bits, 16 bits, 32 bits, 64 bits) et du langage.
On définit des variables qui ont des valeurs données.
Ne pas confondre

- ✓ Le nom de la variable (exemple : valeur-du-compteur)
- ✓ La référence où elle est stockée (son adresse), (exemple : adresse mémoire case 456739087)
- ✓ La valeur stockée : 110

Les algorithmes contiennent des boucles (répétitions, itérations) et des branchements (if then else) qui travaillent sur des variables, tableaux, fichiers.

Langage procédural

Le langage C et le langage Pascal sont deux exemples de langages procéduraux. De tels langages permettent de baser l'écriture du programme sur l'algorithmique : méthode qui permet de transformer une donnée initiale en une donnée finale permettant l'achèvement du processus de calcul. On raisonne donc sur le processus de calcul dans sa totalité, et l'on définit donc une succession d'appel de procédures ou de fonctions.
Un langage procédural est organisé et centré autour de la définition de procédure : séquence définie de traitements, d'algorithmes.
Fortran, Pascal, C++ sont des langages impératifs qui utilisent une séquence d'instructions pour exécuter les opérations désirées. Une instruction est une commande pour effectuer une opération qui change l'état. Les états sont les endroits de stockage et leurs valeurs associées.
Le langage procédural a évolué par extension au cours du temps en incorporant des caractères orientés objets.
C devient C++, Ada 83 devient ADA 95, Pascal devient Object Pascal puis Delphi, Cobol orienté objet devient OOCOBOL, Basic devient Visual Basic.

Langage programme orienté objet Eiffel, Java

On ne cherche plus à définir un processus de calcul, mais l'on tente plutôt de définir les différentes entités (les objets) entrant en jeu dans la résolution du problème. A partir de cette analyse, il ne reste alors plus qu'à déterminer quels vont être les différents mécanismes de communication qui vont lier ces objets et les faire évoluer (ce qui, globalement, va faire évoluer le système). Ces mécanismes de communication sont toujours rattachés à une donnée (un objet).
Le langage Java ou encore le langage Eiffel sont deux exemples de langages orientés objets. Le langage C++ permet aussi ce style de programmation, cependant, il permet aussi d'écrire de façon procédurale, voire de mixer les deux styles.

Le programme orienté objet est organisé comme un ensemble d'objets qui communiquent entre eux à travers des interfaces définies strictement.
La programmation orientée objet consiste à modéliser informatiquement un ensemble d'éléments d'une partie du monde réel en un ensemble d'entités informatiques. Ces entités informatiques sont appelées *objet*. Il s'agit de données informatiques regroupant les principales caractéristiques des éléments du monde réel. La difficulté de cette modélisation consiste à créer une représentation abstraite, sous forme d'objets, d'entités ayant une existence matérielle ou bien virtuelle (entités informatiques).
Les langages orientés objet constituent chacun une manière spécifique d'implémenter le paradigme objet. La programmation orientée objet implique

- ✓ une conception abstraite d'un modèle objet (c'est le rôle de la méthode objet)
- ✓ l'implémentation à l'aide d'un langage orienté objet.

Une méthodologie objet, c'est une méthode d'analyse du problème et un langage permettant une représentation standard stricte des concepts abstraits (la modélisation).

Procédural & Objet

Un langage Objet est différent d'un langage procédural, mais les principes sous-jacents aux langages sont les mêmes.

Langages fonctionnels (Lisp pur, ML) et langages logiques (Prolog)

L'approche pour les langages fonctionnels et logiques est assez similaire, mais elle est différente de celle des langages procédural et objet. Le principe de base de ces langages est la récursivité.
Les langages fonctionnels se différencient des langages dits procéduraux par le fait que l'appel et la définition de fonctions sont des objets du langage. Le résultat d'une fonction peut être une fonction qu'on pourra plus tard appeler. Il en découle que ces langages ne peuvent être complètement compilés (un résultat de type fonction dépend des paramètres d'appel de la fonction génératrice et son code ne peut être connu qu'à l'exécution). Ils sont donc généralement interprétés, bien que des techniques de compilation avancées résolvent les problèmes.
Le modèle computationnel est le lambda-calcul : Fonctions mathématiques (A Church).
Il fait appel à des fonctions. La notion de position de stockage n'est pas nécessaire.
L'instruction d'assignation (fondamentale dans un langage impératif) n'existe pas ici. On applique des fonctions qui retournent des valeurs.
Un programme fonctionnel comprend une série de fonctions primitives avec un mécanisme pour créer de nouvelles fonctions.
Un langage fonctionnel a pour opération fondamentale d'évaluer des expressions. Une expression fournit une valeur.
L'intérêt pour les langages fonctionnels a été fort dans les années 80s.

Diversité des langages

Bien comprendre le paysage des langages : ce qui les rapproche et ce qui les distingue.
Connaître au moins le nom des langages qui ont une importance historique : Simula, …
Entre Langage machine (Assembleur) et Langage naturel.

La jungle des langages

- ✓ langages anciens encore utilisés (Fortran, Cobol)
- ✓ d'autres émergent (Java en 95, XML en 98)

700 langages en 1969 à la NASA pour la mission Apollo.
2000 langages en l'an 2000. Combien de langages réellement industriellement utilisés ?

Taxinomie des langages

- ✓ Langage orienté objet : Java, Delphi
- ✓ Langage procédural : Pascal, C, Fortran
- ✓ Hybride orienté objet : C++, ADA 95
- ✓ Langage basé objet : Smalltalk, Eiffel, (Java)
- ✓ Langage fonctionnel : FP, ML (CAMLIGHT, OCAML, …), LISP (modernisé en dialecte Scheme)
- ✓ Langage logique : Prolog, Programmation par contraintes
- ✓ Langage à balise : SGML, HTML, XML
- ✓ XML est un métalangage pour créer des langages à balises. Ce n'est pas un langage de programmation
- ✓ Langage de Script : Perl, CGI
- ✓ Langage de description et de conception de programmes
  Ce sont souvent des langages de forme mathématique pour les techniques de preuves.
  Approche par notation graphique : UML (diagrammes de flux de donnée, diagramme d'entité relation, diagramme de transition, dessin de séquence de messages)
  - ✓ Langage de Description d'Architecture (ADL) : Rapid, Darwin, …
  - ✓ Langage formel : Z, B, Estérel, LOTOS, …
  - ✓ Langage semi-formel : SDL, UML
  - ✓ Langage pour les protocoles : ASN1. Ce langage décrit les formats de messages sous forme de type abstrait.
  - ✓ Langage de test pour les protocoles (TTCN)
  - ✓ Il existe d'autres langages pour la conception : à acteurs, pour le parallélisme, à flots de données, synchrones/réactifs, etc.

- ✓ Langage pour concevoir du matériel (System on Chip) : VHDL. Tout le matériel informatique est conçu avec du logiciel.
- ✓ Autres langages : PostScript pour imprimer sur un écran ou sur une imprimante.

Évolution en fonction des évolutions du matériel, de l'Internet ?

Résumé de la liste des principaux langages.

Langage	Applications classiques	Compilé/interprété
ADA	Embarqué	compilé
BASIC	Macro de traitement bureautique	interprété
C	Programmation système	compilé
C++	Programmation système objet	compilé
Cobol	Gestion	compilé
Fortran	Calcul	compilé
Java	Programmation orientée Internet	intermédiaire
LISP	Intelligence artificielle	intermédiaire
Pascal	Enseignement	compilé
Prolog	Intelligence artificielle	interprété
Perl	Traitement de chaînes de caractères	interprété

Conception d'un langage

Il faut réaliser des programmes fiables.
Il faut concevoir des langages conviviaux pour pouvoir réaliser des logiciels faciles à concevoir, écrire, lire, tester, exécuter, documenter et modifier.

Propriétés des langages
- ✓ Puissance d'expression
- ✓ Simplicité et orthogonalité
- ✓ Implémentation
- ✓ Détection d'erreur et correction
- ✓ Correction et standards

Puissance d'expression

Le programmeur doit réfléchir à des solutions de problèmes, exprimées dans les termes du problème plutôt qu'en terme d'informatique sur laquelle la solution est implémentée en langage machine.
Le programmeur doit se concentrer sur le problème à résoudre.
La puissance d'expression se situe dans la notation pour décrire des algorithmes et des structures de données.
Le langage doit supporter des idées de programmation structurée et de modularité.
Un langage informatique est un outil pour des types d'applications donnés. Ce ne peut être un outil universel et intemporel. Il dépend de la technologie informatique tout entière : Ada possède une puissance d'expression forte en calcul numérique, moindre en traitement de données.

Les types

Les langages modernes utilisent des types de plus en plus puissants.
Certains types définis une fois pour toute.
Dans les langages modernes, il existe possibilité de construire des nouveaux types (abstract data types) : package Ada et classes C++ ou Java.
On peut alors définir a priori des types prédéfinis par l'utilisation de bibliothèques standard que le programmeur peut utiliser. Mais le programmeur peut en définir lui-même.
Type Boolean = (false, true).
Type Couleur is (rouge, orange, vert).

Traitement des exceptions

Événements exceptionnels : division par zéro, erreur d'un indice de tableau qui va au-delà des limites, rencontre inopinée de la fin de fichier, …
Le programme doit anticiper les fautes.
PL/I : ON-conditions
Ada a été vraiment le premier à lever les erreurs.
Celles impossibles à détecter à la compilation le seront à l'exécution : Raised (Ada, Delphi) ou thrown (C++ et Java).
L'exécution normale est interrompue et le contrôle est transféré à une partie spéciale appelée traitement des exceptions.
Le traitement des exceptions est crucial pour les programmes tolérant aux fautes et la sûreté de fonctionnement.

Simplicité et orthogonalité

Simplicité

Facilité à lire, écrire et comprendre (succès de Visual Basic)
Problème de lisibilité pour comprendre facilement ce qui est écrit (Commentaires, convention de noms).
Modification des programmes avec leur documentation (difficile à mettre en oeuvre).

Orthogonalité

Toute combinaison de constructions du langage est possible avec seulement des restrictions.
L'utilisation de constructions doit être cohérente. L'effet doit être similaire où qu'elles soient utilisées.
La combinaison des constructions est prévisible (loi de surprise minimum).
Exemple de langages orthogonaux : Algol 68, Smalltalk. Le nombre de concepts de base est minimum.

Implémentation

Wirth, concepteur de Pascal et de Modula-2, a voulu concevoir des langages pour des programmes efficaces.
Les caractéristiques qui sont difficiles à compiler sont difficiles à comprendre.
Algol 68 inventé par un comité qui a sous-estimé les problèmes d'implémentation.
Les concepteurs d'Ada étaient des implémenteurs des années 70.

Compilateur

Un programme écrit dans un langage dit "compilé" va être traduit une fois pour toutes par un programme annexe (le compilateur) afin de générer un nouveau fichier qui sera autonome, c'est-à-dire qui n'aura plus besoin d'un programme autre que lui pour s'exécuter. De plus, la traduction étant faite une fois pour toute, il est plus rapide à l'exécution. Toutefois il est moins souple que programme écrit avec un langage interprété car à chaque modification du fichier source (fichier intelligible par l'homme: celui qui va être compilé) il faudra recompiler le programme pour que les modifications prennent effet.
Certaines applications sécurisées nécessitent la confidentialité du code pour éviter le piratage.

- ✓ Un programme compilé a pour avantage de garantir la sécurité du code source.
- ✓ Un langage interprété, étant directement intelligible (lisible), permet à n'importe qui de connaître les secrets de fabrication d'un programme et donc de copier le code voire de le modifier. Il y a donc risque de non-respect des droits d'auteur.

L'exécution d'un programme écrit dans un langage impératif se fait par traduction (compilation) du programme source dans un programme en code machine qui est « équivalent ».
Le code en langage machine est exécuté.
La facilité de la traduction du langage et l'efficacité du code résultant est un facteur majeur pour le succès de ce langage.
Les « gros » langages auront des compilateurs qui seront importants, lents et coûteux.
La vérification des types a lieu à la compilation, pour détecter des erreurs logiques.

Interpréteur

Un langage informatique est différent du langage machine. Il faut donc le traduire pour le rendre intelligible du point de vue du processeur. Un programme écrit dans un langage interprété a besoin d'un programme auxiliaire (l'interpréteur) pour traduire au fur et à mesure les instructions du programme.
Alternative au compilateur : interpréteur.
Un interpréteur peut directement exécuter le programme source. Mais plus communément, le programme source est plutôt traduit dans une forme intermédiaire qui peut être exécutée par l'interpréteur. Dans ce cas, l'interpréteur implémente une machine virtuelle.
Exécuter un programme sous le contrôle d'un interpréteur est beaucoup plus lent qu'exécuter le code équivalent en langage machine mais cela donne plus de flexibilité à l'exécution.
La flexibilité est importante dans les langages où le propos est plus la manipulation symbolique que le calcul numérique.
Exemple : langage de traitement de chaîne SNOBOL4, le langage orienté objet Smalltalk, le langage fonctionnel Lisp, le langage logique Prolog, le langage de script Perl.
Quand on utilise un interpréteur qui supporte un environnement de programmation interactif (Lisp), on écrit la fonction et on l'évalue immédiatement (détection et correction d'erreur facile). Quand le programme est terminé, on peut le compiler pour une exécution plus rapide.

Notion de machine virtuelle

Java est impératif et devrait donc être compilé mais les programmes sont interprétés (pour animer les pages web). Le programme source en Java est traduit en code pour une machine virtuelle pour se faire exécuter sur une machine différente.
Certains langages appartiennent donc aux deux catégories compilables et interprétables (LISP, Java, ...) car le programme écrit avec ces langages peut dans certaines conditions subir une phase de compilation intermédiaire vers un fichier écrit dans un langage qui n'est pas intelligible (donc différent du fichier source) et non exécutable (nécessité d'un interpréteur). Les applets Java, petits programmes insérés dans les pages Web, sont des fichiers qui sont compilés que l'on exécute à partir d'un navigateur internet.

Détection d'erreurs et correction d'erreurs

Ce serait idéal si toutes les erreurs apparaissaient à la compilation. En fait, c'est impossible (problème de l'arrêt d'une machine de Turing indécidable). Quand on met au point un programme, on le modifie au fur et à mesure ; différentes erreurs apparaissent à la compilation et à l'exécution. Il faut veiller à ne pas détruire la structure du programme quand on corrige.

Correction et standards

Correction et rectitude : le programme fait ce qu'il doit faire (conforme à la spécification).

- ✓ Langage fonctionnel : raisonnement mathématique.
- ✓ Langage impératif : programmation structurée avec des modules (éviter les instructions « go to »).

Pour prouver qu'un programme est correct, il est nécessaire d'avoir une définition rigoureuse de chaque construction du langage (méthode pour définir la syntaxe et la sémantique du langage).
Il est facile de définir la syntaxe.
Il est difficile de définir la sémantique d'un langage.
Il est vital d'avoir une description informelle compréhensible par les développeurs (compromis entre exactitude et informel, caractéristique pragmatique de l'informatique).

Standardisation d'un langage

La notion de définition de standard pour un langage hélas n'existe pas souvent au bon moment. La définition du standard vient trop tard.
Les éditeurs de compilateurs ajoutent des caractéristiques pour « améliorer » le langage, ce qui créent ensuite un problème de portabilité.
L'exception est le langage Ada : le compilateur doit être validé sur un sur-ensemble et sous-ensemble du langage par une suite de tests.

Développement d'un programme

Maîtrise du langage par le développeur

Il est difficile de bien maîtriser un langage.
Certains parlent de la maîtrise d'un langage et d'un OS par génération : la génération Pascal, la génération C et Unix, la génération Java et Internet ?
Il existe une attitude irrationnelle vis-à-vis de « son » langage (adatollah, javatollah, …).
Il faut s'approprier (au moins) un langage et ne pas tomber dans le piège de devenir un fanatique de son langage ou de son environnement de développement.
Barrière d'abstraction : tout le monde ne peut pas programmer.
Développer un programme est un travail difficile qui demande de la concentration.
Les qualités sollicitées sont d'être pragmatique et être suffisamment abstrait (il faut être efficace).
Développer un programme ce n'est pas un travail d'artisan ou d'artiste. Il faut le considérer comme un travail industriel intégré dans une équipe.

- ✓ Un développeur peut maîtriser parfaitement 10 000 lignes de code.
- ✓ Pour développer un programme de 50 000 à 100 000 lignes de code, il faut 5 à 10 personnes en parallèle avec une bonne méthodologie.
- ✓ Pour réaliser un programme de 1 million de lignes de code (Netscape, Excel 5, Word, etc.), il faut toute une équipe car une seule personne ne peut maîtriser ces programmes. C'est l'organisation et la méthodologie qui pilote le cycle de vie.

Productivité et Génie logiciel

Le problème de l'informatique aujourd'hui, c'est la faible productivité.
Le génie logiciel a émergé en 1968 : la productivité en logiciel depuis 68 a été multipliée par 4 (?) seulement.
Entre temps, les logiciels se sont complexifiés (d'un facteur beaucoup plus grand que le facteur de productivité), d'où la volonté de réutilisation, de structuration par modules.
Aujourd'hui, on ne crée plus ex nihilo un programme, on « reprend » un vieux programme (avec avantages et inconvénients).
Un développeur a une productivité de 3 à 5 lignes par heure (tout compris) pour un programme difficile et de 2000 lignes par mois (quand on sait parfaitement ce qu'il faut faire) pour un prototype facile (à titre indicatif).
Pour améliorer la productivité :

- ✓ il faut travailler en parallèle : méthodologie pour travailler en équipe ;
- ✓ il faut réutiliser le travail des autres : notion de composants logiciels et méthodologie logiciel (modularité).

En général, on n'écrit pas un programme seul, on modifie ou adapte le programme d'un autre.

Modèle de développement

Les différentes étapes sont les suivantes.

- ✓ Analyse du besoin utilisateur : parfois vague, incohérent, ambigu, incomplet.
- ✓ Analyse des exigences pour régler les conflits des différents points de vue.
- ✓ Spécification pour définir ce qu'on fait : document pour dire ce que le logiciel fera (et ne fera pas).
- ✓ Définitions, en parallèle de la spécification, des plans de test et de validation.
- ✓ Conception d'une solution et implémentation en utilisant un ou plusieurs langages de programmation.
- ✓ Validation et vérification (ou preuve de programme) : ce que fait le programme c'est ce qui est spécifié.
- ✓ Test avec des données (prouve la présence d'erreurs, ne prouve pas l'absence d'erreurs).
- ✓ Maintenance curative (correction d'erreurs)
- ✓ Modification du programme pour des évolutions ou des additions ou des changements de spécifications

- ✓ Certification de logiciels en sécurité et en sûreté de fonctionnement pour les télécoms, l'avionique, …

Méthodologie & cycle de vie d'un logiciel

La conception d'un logiciel est souvent influencée par le langage qu'on va utiliser à l'implémentation.
Le modèle en cascade (waterfall model) : les étapes sont réalisées en séquence.
Le modèle en spirale de Boehm (1988) : le modèle séquentiel waterfall est remplacé par un mode itératif et incrémental pour la gestion du risque.
Avec la méthode orientée objet on utilise un « framework » d'objets communicants pour spécifier, concevoir et implémenter. L'orienté objet est assez bien adapté pour une approche incrémentale. Les objets sont définis à différents degrés d'abstraction : à mesure que le développement se déroule, on ajoute des caractéristiques à la description des objets.
Développement orienté objet avec les notations UML (Booch, 1998) : diagramme de classe.

Modularité

Relation entre manière de programmer et le langage
- ✓ Pascal et langage structuré : construction de modules indépendants
- ✓ Packages en ADA
- ✓ Classes en langages orienté objet

Langages ou systèmes

Les langages modernes supportent la programmation en réseau et la création d'interfaces graphiques à travers des bibliothèques. On peut se demander si ces bibliothèques font partie du langage ou de l'environnement de développement.
Souvent ces librairies sont dépendantes du système d'exploitation.
- ✓ Delphi et Visual Basic dépendent de Microsoft Windows, langages liés à l'OS et à l'implémentation donc à l'environnement de développement système.
- ✓ Java librairies indépendantes de l'OS et du réseau (bibliothèque IP).

Les capacités de Java sont définies en terme de JVM.
Programmes avec des interfaces graphiques : ils sont « event driven », on attend un clic de l'utilisateur pour interagir.
La structure des programmes interactifs est différente de celle des programmes procéduraux.

Historique

Important de comprendre l'évolution des langages, pour comprendre les erreurs du passé et éviter de les répéter.
Fortran et Cobol sont encore utilisés 40 ans après leur apparition.
Inertie énorme en programmation.
Difficile de faire apparaître un nouveau langage.
Énorme investissement dans les systèmes stables (voir le « bug de l'an 2000 »).

Survol historique

Années 50

Se débarrasser de l'inflexibilité du matériel (von Neumann).

Années 60

Mission Apollo : 1 million de lignes de code, ordinateurs de 0,5 mips avec mémoire de quelques kilo-octets.
Course à la performance (pas assez de mips)
Tous les ordinateurs sont différents (structure du matériel, des systèmes d'exploitation, …)
Fortran, Algol, Cobol, PL/I, Basic.
Fortran, Cobol et Basic sont restés (ils ont beaucoup évolué depuis).

En 70

Matériels très différents (IBM, HP, DEC, Control Data, Univac, …).

Problèmes de portabilité, nécessité d'avoir un standard et un compilateur (motivation pour ADA du DoD).
Logiciels assez petits (pas d'interface graphique, peu de données, …).
Problème du temps réel (Ada).

Année 70 - 80

3 langages sont restés importants dans les années 90.

1) Pascal était le langage enseigné dans les universités. Il n'est plus enseigné dans les universités comme le langage de base bien qu'il soit pédagogique.
2) Le langage C est le langage des programmeurs systèmes et a été avec Unix le langage des stations de travail. Il est devenu C++ qui a servi ensuite à créer Java.
3) Ada a démarré fin 70, Ada 83 et Ada 95 (le langage et ses évolutions sont venus toujours trop tard).

Tous ces langages ont eu une extension orienté objet.

Années 75 - 85

Mini-ordinateurs (PDP, VAX, SUN, HP, etc.) et Unix avec la station de travail et l'interface graphique : Xerox, puis Sun, Apple (Macintosh).

Années 80

Années de l'Intelligence Artificielle.
Programmation fonctionnelle et logique.
Niveau d'abstraction plus grand, donc plus facile de raisonner.
Langages fonctionnels : moderne Scheme (Lisp), ML, StandardML, OCAML et Haskell.

En 85

Bouleversement avec l'arrivée des Personal Computers (Apple, Microsoft).

Années 85 - 95

Les langages orientés objet.
Smalltalk, Eiffel, C++, Delphi, Ada 95 et Java.

Années 90

L'utilisation des ordinateurs a changé radicalement.
Apparition du PC à la maison.
Les interfaces graphiques sont développées en 70 (Xerox), sur Apple en 80 mais c'est en 90 que les IHMs (Interfaces Homme machine) sont devenues la norme pour dialoguer avec des ordinateurs (grâce aussi à la programmation objet très adaptée aux IHMs).
En 90 est aussi apparu le séisme du Web avec Internet (qui n'était pas neuf !).
On s'écarte des langages de programmation, on parle de systèmes de programmation.
Les librairies standard de Java deviennent la partie centrale du langage.
Delphi est un ensemble de bibliothèques de classes et un environnement de développement au-dessus de Object Pascal.
Même chose pour Visual C++ ou Visual Basic.

En 95

Uniformisation : Intel + Microsoft + Cisco
Réaction Unix => Linux, logiciel libre, Java
Sur Internet, idée de Machine Virtuelle avec du bytecode pour la portabilité (à cause de la puissance des machines).

Années 00

Intel + Microsoft + Cisco dominent les stations de travail et l'Internet.
L'approche objet joue un rôle central, marquée par :

- ✓ Les architectures réparties de l'Internet (Web, XML et Java)
- ✓ Les architectures embarquées des Télécoms mobiles (terminaux à faible consommation)

Apparition des architectures mobiles : Erlang (utilisant le Pi-calcul).
Mouvement vers l'interopérabilité des applications réparties ou mobiles (M2M, P2P, etc.).
Nécessité de faire intervenir des langages à balises pour gérer l'hétérogénéité.

Machines du début

Spéculation du 19ème siècle

Ada, comtesse de Lovelace, est considérée la première développeuse car elle a travaillé avec Charles Babbage sur la Machine Analytique (spéculation académique) au 19ème siècle. Son prénom a été utilisé pour le langage Ada.

1er Travaux

Fin des années 1940 travaux sur machine et langage (Turing, von Neumann).
Il n'y a pas de développement de langage de programmation de haut niveau.
Ce sont des « codes instruction » très primitifs par rapport à l'assembleur.
Les données sont dans le code, il n'y a pas de boucles.
Les programmes de l'époque ont des erreurs avec la nécessité de créer des patchs pour les corriger.
Au milieu des années 50, émerge une nécessité d'améliorer la programmation des 2 côtés de l'Atlantique (États-Unis, Grande Bretagne, France).

Premiers langages scientifiques

Fortran

IBM : Mathematical FORmula TRANslating system.
Fortran 1 en 1956, compilateur en 58
Concepteur : Backus. Il émet quelques principes nouveaux à l'époque :

- ✓ Ce qui compte ce n'est pas la beauté de programmer mais son économie.
- ✓ La version codée à la main doit aller moins vite que le programme compilé.
- ✓ Le coût de développement et le coût de debugging est plus élevé que le coût de l'exécution.
- ✓ D'où l'idée d'utiliser des notations mathématiques.

Fortran conçu pour les calculs scientifiques sur IBM 704 et ses cartes perforées.

- ✓ Pas de facilités de traitement de chaînes de caractères
- ✓ Structure de données : uniquement le tableau
- ✓ Apparition des commentaires dans le programme.
- ✓ Instruction d'assignation avec des instructions mathématiques complexes à la droite.
- ✓ Simplicité de la boucle.
- ✓ Subroutines et fonctions : notation symbolique proche des mathématiques.
- ✓ Format d'entrée et de sortie simplifié.
- ✓ Langage indépendant de la machine.

Premier compilateur pas efficace mais succès quand même.
Efficace pour les programmeurs, facile à apprendre.
Supporté par IBM, devient la compagnie du calcul : Fortran et IBM se sont développés ensemble.
Les applications de l'époque étaient scientifiques.
A facilité les entrées et les sorties.
Évolution : Fortran en 66, en 77 (développé en 78) et Fortran 90 (développé en 91).
Fortran 90 supporte l'allocation d'espace pour tableau, enregistrement, modules, pointeurs et supporte Fortran 77 : combinaison d'ancien et de nouveau.

Algol

Profonde influence sur la définition et la conception de langages.
Défini par un comité européen et américain (International Algebraic Language IAL ALGOrithmic Language).
Proche du langage mathématique et lisible.
Algol 58 au début mais jamais implémenté.
Rapport : Report revised par Naur en 63.

Concept ALGOL 60

Définition du langage en langage formel BNF pour définir la syntaxe qui a conduit à définir des compilateurs dirigés par la syntaxe.
La sémantique était définie en anglais et avait des ambiguïtés.
Algol 60 est structuré en blocs, les variables ne sont pas visibles en dehors des blocs.
Algol 60 instructions de contrôle structurées améliorées par rapport au Fortran.

Algol 60 introduit la récursion : procédures récursives
A conduit à publier les algorithmes via The ACM pour communiquer entre utilisateurs.
Algol n'a pas dépassé Fortran (compilateur apparu 3 ans après).
IBM ne supporta pas Algol. Les compilateurs Fortran étaient plus efficaces.
Algol n'avait pas d'entrée et de sorties officielles.

Algol et ses suites

Introduction des complexes, des bits.
Algol 68 se veut universel (traitement de données).
Économie de construction alors que PL/I essaya d'être un langage qui prend en compte tout.
Orthogonalité : pas d'interaction entre les constructions.

Pascal

Niklaus Wirth fin 60, début 70 grâce à son travail sur Algol W (Wirth).
Accent sur l'implémentation du programme bien structuré et bien organisé.
Pour défier le Fortran utilisé aux USA.
Dimension du tableau statique (contrairement à l'Algol) car on améliore ainsi l'exécution des programmes.
On introduit les types de données qui peuvent être construits à partir des types non structurés integer, real, boolean, char et enumeration type.
Type structuré par l'utilisateur : tableaux, enregistrement, ensembles et fichiers.
Types de pointeur avec des enregistrements pour faire des arbres et des listes liées.
Les chaînes peuvent être manipulées mais c'est difficile !
Pascal a été adopté par les Universités dans les années 70 pour enseigner l'informatique.
Et mi 80 c'est le langage enseigné pour les étudiants. C'est le langage qui introduit un ensemble complet de structures de données et qui encourage un bon style de programmation.
Pascal a influencé les autres langages.
Wirth conçut ensuite Modula puis Modula-2, c'était l'introduction du concept des modules, une extension de Pascal.
Pascal a profondément influencé Ada.

Langages de traitement de données de gestion

A l'origine, on a des « computers » pour le calcul mais très vite on a vu qu'on pouvait manipuler des symboles. « computer » se dit en français ordinateur …
Le problème, c'est qu'à l'époque, il n'y avait pas de notation de gestion et de manipulation (comme en mathématique). On a pensé à la langue anglaise pour les notations mais une langue naturelle est trop ambiguë.
1955 : Flow-Matic et a conduit au COBOL implémenté sur Univac 1.

COBOL

(Common Business Oriented Language) fin des années 50 conçu par des représentants américains de constructeurs.
Le DoD a une grand influence sur son succès. Les contrats devaient utiliser les programmes en COBOL. DoD en a fait un de facto standard bien avant sa définition standard en 1968.
Importance du traitement de fichier et de données.
Les calculs sont très simples.
COBOL est verbeux si bien que les programmeurs et les managers peuvent lire les programmes écrits en Cobol.
Cobol n'était pas innovant mais a introduit l'idée de base de données : séparation entre la description de données, l'environnement physique dans lequel le calcul est réalisé et le calcul réellement effectué sur les données.
La description logique des données du problème peut être décrit indépendamment des caractéristiques physiques du support physique sur lesquels elles sont stockées et manipulées.
Un programme COBOL est divisé en 4 parties :

1) Division identification : pour la documentation
2) Division environnement : définit les spécifications du programme dépendant de la machine (données extérieures)

3) Division data : logique description des données
4) Division procédure : contient les algorithmes nécessaires pour résoudre le problème

L'informatique est divisée en 2 mondes : à Cobol la gestion, à Fortran le calcul scientifique.
Évolution lente et maîtrisée du langage qui a peu changé en 25 ans :
- ✓ Changements en 74 et 85.
- ✓ Changement radical en fin 90s : on a introduit l'orienté objet et le ramasse-miettes
- ✓ OOCobol inclut les versions précédentes.

Langages de 4ème génération

1ère génération : Code machine
2ème génération : Assembleur symbolique
3ème génération : Langages de haut niveau Fortran, Cobol, Pascal, assez indépendant du hardware
4ème génération : Le but des programmes est d'être orienté vers le problème à résoudre.
Les Langages de 4ème génération ont été développés à cause du mécontentement devant Cobol (informatique de gestion) : Report Program Generator (RPG) dans les années 60.
D'autres problèmes exigent et utilisent des feuilles de tableau et des bases de données.
C'est une application qui interagit avec un système plutôt que l'utilisation d'un langage de programmation.
Les systèmes de base de données utilisent SQL (Structured Query Language) : le langage est caché par une interface.
Les langages comme Java et Delphi incluent des bibliothèques avec des classes qui supportent des appels à des commandes SQL, si bien que ces langages peuvent être utilisés avec des bases de données et avec un éventuel accès distant sur Internet.

Langages généraux

PL/I

Dans les années 60
- ✓ Informatique scientifique : Fortran arithmétique avec des points flottants, tableau, procédures rapides et des calculs, souvent avec des calculs par lot.
- ✓ Informatique de gestion : Arithmétique décimale, rapide et entrées sorties asynchrones, facilité de traiter les chaînes et des routines de tri et de recherche.

Mais on demandait du temps réel, du calcul de listes§.
IBM était en avance et inventa l'IBM 360 avec le PL/I.
Les principes sont :
- ✓ Économiser le temps du programmeur
- ✓ Unité pour réconcilier la gestion et le scientifique
- ✓ Un langage très complet

Derrière PL/I, il y a cette idée de « default ». Chaque attribut, option d'une variable a une valeur par défaut, chaque spécification a une interprétation par défaut.
Venaient de Fortran et de Cobol mais aussi d'autres langages, le traitement des listes, les structures de contrôle et les méthodes de management de stockage.
Il y avait aussi du nouveau : le traitement des exceptions avec des ON-conditions.
PL/I essayait d'être efficace et flexible : la pénalité était la complexité.
A l'opposé de PL/I il y avait Algol 68.
Débat (début des années 70, fin années 60) entre « core » et « comprehensive language » mais sans conclusions.
Les langages orientés objet (comme Java) qui peuvent avoir un petit cœur de base et couplés avec la possibilité de s'étendre avec des types, existant ou à définir avec de nouveaux types, répondent à ce problème.
PL/I a eu de nombreuses difficultés et a été soutenu par IBM (coût énorme) : résistance des utilisateurs pour abandonner le Fortran et le Cobol et les remplacer par PL/I. IBM a abandonné.

Développement interactif de programmes

Autrefois les ordinateurs étaient mono-tâches. Les vitesses relatives différentes des entrées – sorties, du processeur central et des mémoires permettent le temps partagé pour utiliser au mieux les éléments de l'ordinateur.

Hypertrophie du processeur central avec terminaux distants reliés à la machine centrale.
Commença en 60s jusqu'à mi 70s. Les langages ont évolué et les compilateurs étaient aussi bien dans les stations de travail que dans l'ordinateur central. Le programme devient un fichier : gestion interactive, compilation, gestion des erreurs.
Certains langages permettent un développement interactif (langages fonctionnel et logique).

Le langage BASIC

Beginner's All Purpose Symbolic Instruction Code
Développé à Dartmouth College par Kemeny et Kurtz mi 60s pour les débutants, facile à apprendre et à se souvenir et à traduire, on n'oublie jamais.
Succès qui a surpris : langage simple pour les micro-ordinateurs des années mi 70.
Visual Basic : évolution vers l'orienté objet pour les applications avec une interface graphique sous Windows.

APL

A Programming Language
Défini par Iverson, langage mathématique très concis, avec un grand nombre d'opérateurs.
Langage riche.
APL : grand alphabet (52 lettres, 10 chiffres et 52 autres caractères)
Les objets primitifs sont des « arrays » : liste, tables, ou matrices.
Langage piloté par un opérateur : peu de branchement.
Pas de précédence d'opérateurs : toutes les instructions sont parsées de droite à gauche.
APL a attiré des « fanatiques », beaucoup de controverses.

Langages spécifiques

Langage spécifique à un domaine.
Augmente la productivité à cause de leur degré d'abstraction plus grand.
Intéressant pour les experts du domaine de l'application.

Langage de manipulation de chaînes

COMIT (MIT 57-61, puis SNOBOL (Bell Labs) et SNOBOL4 (67).
Influence sur les éditeurs de texte : allocation dynamique de mémoire pour changer une chaîne de caractères par une chaîne plus grande.

Langages de traitement de Listes

En Fortran et Algol la seule structure de données est l'Array du même type (et de taille fixe en Fortran).
Cobol structure de données avec des éléments de différents types mais il est impossible de changer la structure de base de l'array. C'est ce que fait un processeur de liste.
Un élément est divisé en une partie information et une partie pointeur.
Dans le cas d'une liste liée, le pointeur donne la référence à l'élément suivant.
Dans les structures d'arbres, il y a deux pointeurs au moins.
Dans Lisp les pointeurs sont implicites.
IPL-V (Carnegie institute of Technology 57 – 61).
Fait apparaître le concept de free space list : espace de stockage pour les nouveaux éléments.

Lisp

John McCarthy (MIT fin des années 50) langage utilisable dans les débuts des années 60.
Exécute des calculs avec des expressions symboliques plutôt qu'avec des nombres.
Plutôt pour les applications d'Intelligence Artificielle : jeux, preuve de théorème, traitement du langage naturel.
Lisp est non typé.
Représente des expressions symboliques sous la forme de structure de listes.
Petit ensemble d'opérations de construction et de sélection pour créer, extraire des informations d'une liste.
Les opérations sont des fonctions et utilisent l'idée de composition de fonctions pour construire des fonctions plus complexes.
Le contrôle est récursif plutôt qu'itératif.

Les données et les programmes ont des formes équivalentes : les programmes peuvent être modifiés comme des données et les structures de données peuvent être exécutées comme de programmes.
Gestion de la mémoire par ramasse-miettes.
C'est un langage fonctionnel (Lisp pur).
Lisp a survécu et est toujours utilisé dans les systèmes experts.
Il est élégant et mathématique, avec une structure fonctionnelle de base, capable d'exprimer les idées du Lambda calcul.
Emacs (l'éditeur de texte) est écrit en Lisp.
A permis le développement de Scheme et un standard Common Lisp.
Algol 68, PL/I et Pascal commençaient à introduire du traitement de listes dans le langage.

Langages de simulation

La simulation des systèmes discrets a été une des premières applications de l'ordinateur.
Ces systèmes sont modélisés par une série de changements d'états qui arrivent souvent en parallèle. Il peut y avoir des interactions entre les divers éléments.
Ces simulations servent à prédire le comportement, par exemple, simulation de trafic pour faire apparaître des goulets d'étranglements dans un réseau.
GPSS (General Purpose Simulation System) en 61.

Simula 67

Début des années 60 de Ole-Johan Dahl & Kristan Nygaard (Norvège).
Collection d'objets indépendants dans un but commun.
On étudie le cycle de vie de ces objets.
Simula basé sur Algol 60 avec le concept de classe.
Il est possible de déclarer une classe et de générer des objets à partir de cette classe.
Classe : Idée de base. Les données (et les structures de données) et les opérations formées sur ces donnés appartiennent à cette classe.
Ceci est la base de la programmation objet et l'implémentation des data types.
Simula est en fait un langage général avec 2 classes spécifiques pour simuler.
Simula a introduit l'idée de concurrence.
Simula est le père des langages orientés objet.
Le concept de classe a été repris dans C++, Ada, Smalltalk, Eiffel et Java.

Langages de Script

Tâche classique de l'informatique : analyser un texte très grand ou le reformater
Nécessité de faire du « pattern matching » sur des chaînes de caractères.

- ✓ Langage awk : Pattern matching
- ✓ Langage Perl : puissance de awk avec la syntaxe de C : Perl et Awk pour Unix au départ utilisé pour le WWW.
- ✓ Common Gateway Interface pour dialoguer entre un client et un serveur.
- ✓ Autres langages de script : Python, Tcl, JavaScript.

Ce sont des langages de glu pour faciliter la communication entre programmes écrits dans des langages différents.

Langages de programmation systèmes

Pour operating system, noyau temps réel et compilateur et synchronisation de processus :
BCPL fin des années 60, BLISS.
Maintenant c'est le C.

Le Langage C

Le paradigme d'Unix est : « tout est fichier ». Unix banalise le matériel (une imprimante est un fichier /dev/lpr) et le logiciel par la notion de fichiers. Un fichier est un ensemble de caractères. Le langage C va permettre de traiter ces caractères, et les langages de script pour Unix vont exploiter et reformater les chaînes de caractères.

Langage C : évolution de BCPL en B (69-73) utilisé pour implémenter l'OS UNIX.
Unix et C développé aux Bell Labs par Ritchie et Thompson.
C est un langage petit. Il existe des compilateurs portables.
ANSI standard en C démarrage en 83 et publié en 89.
La force de C : alliance entre avantages de langage de haut niveau et l'efficacité d'un langage assembleur.

Programmation système: pas de vérification de type et capacité de traiter de l'arithmétique sur les adresses de stockage et d'exécuter des opérations au niveau du bit et des patterns de bits.
Large catalogue d'opérateurs qu'on peut combiner de manière libre.
Les défauts du langage C : difficile à lire, facile d'écrire des bugs (à cause du manque de vérification de type).
Sa structure est un mélange d'Algol et de Fortran.
C++ version orientée objet par Stroustrup au début des années 80.
Les compilateurs prennent en compte C et C++. C disparaît comme langage séparé en 90.

Occam

Développé pour les Transputers : ordinateurs puissants à partir d'un réseau de Transputers opérant en parallèle.
La communication et la synchronisation sont donc essentielles.
Le processus est au centre du langage et les processus s'envoient des messages sur des canaux.
Similaire à ce qui est fait en Ada.
Large intérêt à la fin des années 80.

Modules, classes, types de données abstraites, objets

Module

Année 70 : idée de module pour maintenir les programmes.
Modules avec des interfaces publiques assez petites et une implémentation cachée.
Besoin de réutiliser les logiciels et de construire des logiciels à partir de composants.
La modularité : décomposition physique en fichiers, décomposition logique pour élever le niveau d'abstraction avec laquelle les programmeurs conçoivent leurs programmes.
Un problème peut être spécifié comme un ensemble de types de données abstraits.
Un module doit implémenter un abstract data type avec la représentation et l'implémentation des opérations cachées.
Langage orienté objet : Smalltalk, C++, Eiffel, Java moins de concentration théorique sur les types de données abstrait mais en plus héritage et liens dynamiques.
Modula 2 et Ada : les modules structurent le programme (le package en Ada sert à définir des types abstraits de données).
La programmation objet a décollé dans les années 80 et est à la fin des années 90 le paradigme dominant.
Ada a des types de données abstraites.
Approche orientée objet : classes, héritage et des liens dynamiques.
Small talk pour le prototypage mais marche industriellement pour l'implémentation d'interface graphique.
C++ a ajouté des caractéristiques orientées objet à C.
Eiffel a eu une grande influence. C'est un langage purement objet qui a pour but les grands systèmes et la syntaxe est basée sur Pascal.
La position des langages orientés objet s'est renforcée avec Java.
En fait Eiffel, Delphi, C++ et Java sont responsables de la conversion des programmeurs en procédural à une approche orientée objet.

Objet

Simula a été le premier langage de programmation à implémenter le concept de classes en 67.
En 76, Smalltalk implémente les concepts d'encapsulation, d'agrégation, et d'héritage (les principaux concepts de l'approche objet).
Un objet est caractérisé par plusieurs notions:

- ✓ **Les attributs**: Il s'agit des données caractérisant l'objet. Ce sont des variables stockant des informations d'état de l'objet
- ✓ **Les méthodes** (*fonctions membres*): Les méthodes d'un objet caractérisent son comportement, c'est-à-dire l'ensemble des actions (appelées *opérations*) que l'objet est à même de réaliser. Ces opérations permettent de faire réagir l'objet aux sollicitations extérieures (ou d'agir sur les autres objets). Les opérations sont étroitement liées aux attributs, car leurs actions peuvent dépendre des valeurs des attributs, ou bien les modifier

- ✓ **L'identité**: L'objet possède une identité, qui permet de le distinguer des autres objets, indépendamment de son état.

Classe

On appelle *classe* la structure d'un objet, c'est-à-dire la déclaration de l'ensemble des entités qui composeront un objet. Un objet est issu d'une classe. En réalité on dit qu'un objet est une **instanciation** d'une classe, c'est la raison pour laquelle on pourra parler indifféremment d'*objet* ou d'*instance* (éventuellement d'*occurrence*).
Une classe est composée de deux parties:
- ✓ **Les attributs** (*données membres*): il s'agit des données représentant l'état de l'objet
- ✓ **Les méthodes** (*fonctions membres*): il s'agit des opérations applicables aux objets

ADA

DoD le plus grand utilisateur des ordinateurs de l'époque.
Dans les années mi 70, le DoD a voulu favoriser et financer le développement d'un nouveau langage.
700 langages pour la mission Apollo.
Volonté d'avoir un seul langage de programmation standard pour les différents systèmes embarqués.
Logiciels embarqués :
- ✓ logiciel de contrôle commande (avion, hôpitaux, usines, etc.)
- ✓ application avec de la concurrence et des changements en fonction du temps, « automates »
- ✓ ces systèmes demandent de la fiabilité avec la possibilité de décider s'il y a une erreur.

Une compétition internationale a été organisée, gagnée par CII-Honeywell Bull en France, Jean Ichbiah : Ada Comtesse de Lovelace qui a travaillé sur la machine analytique de Babbage et qui était considérée comme la première programmeuse.
Le langage Ada a été publié en 83.
Ada est basé sur Pascal mais est beaucoup plus gros.
Ada a des packages pour décrire des grands composants.
Le composant logiciel contient des définitions de type et des opérations pour manipuler des variables de types.
La représentation des données et l'implémentation des opérations peuvent être cachées à l'utilisateur.
Ada contient la notion de bibliothèques de packages pour assembler des composants sans connaître l'intérieur.
Ada est un langage fortement typé.
Task permet le parallélisme.
Clock data type pour contrôler le temps réel.
Traitement des interruptions et des exceptions.
Le DoD a décrété qu'il ne pouvait y avoir de dialectes d'Ada (pas de sous-ensemble et de sur-ensemble) d'où la nécessité de valider le compilateurs sur une suite sévère de tests.
Le compilateur Ada est gros.
Le langage a eu du mal à être adopté par les programmeurs : difficile face à la compétition de C et ensuite C++.
Ada ne pouvait pas évoluer (par définition). Ada a été obligé de subir une révision en 95 pour des extensions objets (passage de Ada 83 à Ada 95).

Modula 2

Modula 2 conçu par Wirth fin des années 70.
Pour étendre Pascal avec le concept de module.
Modules et des capacités de multiprogrammation.
Moins compliqué que Ada, il était utilisé pour l'enseignement.
En compétition avec C++ et Ada puis récemment Java.

Smalltalk

Alan Kay : idée de fournir un PC à des non experts pour communiquer à travers une interface graphique conviviale.
Développé à Palo Alto par Xerox début 70 sur des stations puissantes en graphique.
Plusieurs versions dont Smalltalk80 : Smalltalk intègre le langage avec les outils, fournissant un environnement de programmation.

Avec Smalltalk on ne sait pas où finit le langage et où commence l'environnement.
Dispose de menus, de fenêtre d'icône d'entrée par la souris.
Smalltalk a développé le concept de classe de Simula mais sa philosophie a été influencée par Lisp.
Dans les années 80, Smalltalk était « le » langage orienté objet.
Un objet est un ensemble de données locales avec un ensemble de procédures qui opèrent sur ces données.
Tous les calculs sont exécutés en envoyant des messages aux objets.
On identifie des vrais objets du monde réel et on les modélise par des objets Smalltalk.
Tout dans Smalltalk est objet.
Plus tard Java et Eiffel ne sont pas allés si loin et ne traitent pas les classes comme des objets. Contrairement à Smalltalk, ils ont des vérifications de type statique. Ils sont ainsi moins flexibles mais plus adaptés aux plus grands produits.
Le passage par message, le support d'environnement de développement riche et les interfaces graphiques sont coûteux en terme de ressources informatiques.
Dans les débuts de l'informatique le temps ordinateur était plus précieux que le temps humain. Les langages et systèmes étaient faits pour optimiser les ressources ordinateur (factures = cpu + mémoire + IO).
La philosophie de Smalltalk est que les ressources informatiques ne sont pas chères et que le temps humain est en revanche précieux. Valable pour l'informatique aujourd'hui.

Eiffel

Bertrand Meyer dans les années 80s.
Fournir un langage qui supporte la création de grands systèmes robustes à partir de composants testés et réutilisables.
La vérification de type est réalisée à la compilation.
Eiffel est un pur langage orienté objet.
Le type des objets est défini par des classes.
Les objets sont créés dynamiquement et accédés par référence.
La correction de programme est un thème majeur de Eiffel qui inclut des pré- et des post-conditions pour donner la signification des opérations et des invariants de classes qui doivent être vraies pour toutes les instances de classes (tous les objets)

- ✓ Précondition vraie si une opération est appelée.
- ✓ Postcondition garantie vraie quand l'opération a vraiment terminé son exécution.

Fournit une méthode systématique pour la construction de classes.
On spécifie l'effet d'une classe comme un type de donnée abstrait en utilisant les pre et post conditions et les invariants de classes.
Ensuite on implémente la classe pour satisfaire la spécification.
Les utilisateurs de la classe n'ont à connaître que la spécification, l'implémentation étant cachée.

C++

Langage efficace pour ajouter à C le concept de classe de Simula.
Développé à Bell Labs par Stroustrup, la patrie de C.
A commencé en 79 la 1ère version C avec des Classes.
C++ a été commercialisé en 85, ANSI standard en 98.
C++ a eu un impact majeur sur l'évolution de C.
La classe construct est construite sur le struct (record) de C.
On ajoute des membres de fonctions aux membres de données.
C++ supporte l'héritage multiple et le lien dynamique (dynamic binding).
Les objets contiennent des membres fonction statique et dynamique, contrairement aux autres langages orientés objet où les calls functions sont liés dynamiquement.
Les classes génériques sont supportées (class templates).
Vérification de type plus sévère qu'en C.
C++ a été un changement dans la continuité.
Ce n'est pas un langage orienté objet pur mais il supporte les caractéristiques de l'orienté objet.
Beaucoup de développeurs utilisent C++ comme un C plus sûr. Ils utilisent les classes pour créer des types de données abstrait et n'utilisent pas les traits de l'orienté objet comme le lien dynamique.
C++ est un gros langage (il garde les capacités de C pour le bas niveau de programmation).

C++ a été à la base de Java.

Delphi

Extension de Pascal orienté objet Object Pascal.
Delphi développement de Object Pascal avec une extension d'interface graphique, de base de données et de programmation Internet.
Delphi est un RAD (Rapid Application Development) avec un environnement de développement graphique.
Delphi utilise Windows.

Java

La diffusion de Java a été impressionnante.
A débuté à Sun Microsystems en 90-91 (Oak) pour concevoir un langage petit pour les appareils de la maison.
Avec le développement du Web, le langage a été reciblé pour le langage de l'Internet et a changé de nom (Java) en 95.
Idée d'applet (petite application) pour animer des pages web : a été un succès immédiatement.
Est devenu le langage pour l'enseignement dans toutes les universités.
Java joue un rôle fondamental en faisant de l'orienté objet le paradigme dominant pour les langages.
C'est un petit langage et qui a un très grand nombre de bibliothèques, appelées les Java Application Program Interfaces (Java API).
Dans les autres langages, les bibliothèques sont un extra. Ici pour Java, c'est au centre du langage.
Java est conçu comme une simplification de C++ : les programmeurs de C++ l'ont adopté très facilement.
Dans Java, il y a un ensemble de Classes de bibliothèque qui permettent de faire des interfaces graphiques.
Java permet d'écrire un programme à partir de composants prédéfinis.
Java est portable (important pour Internet).
Java est transformé en un langage intermédiaire le JavaBytecode de telle manière qu'il puisse être interprété par l'Interpréteur.
JavaBytecode est un langage machine pour une Java Virtual Machine.
L'environnement de développement, le Java Development Kit peut être téléchargé gratuitement à partir d'Internet.
Les browsers Internet Explorer et Netscape sont « Java enabled » (ils ont une JVM à l'intérieur) et on peut exécuter des applets Java (applications petites).
Java diffère de C++

- ✓ C'est un langage purement orienté objet
- ✓ Il est moins performant en terme de rapidité
- ✓ Toutes les opérations sont parties d'une classe
- ✓ Accès à des objets par référence comme dans Eiffel
- ✓ L'arithmétique des pointeurs n'est pas permise
- ✓ Il y a un garbage collecter
- ✓ Il y a un héritage simple

Le multiple héritage se fait à travers des interfaces multiples. Une interface est comme une classe abstraite qui n'a pas d'attributs et dont toutes les opérations sont abstraites.

Langage Fonctionnel et Logique

Lisp existait déjà depuis longtemps mais n'était pas vraiment utilisé.
Backus en 78 a introduit le langage FP et remarqua que les langages impératifs avaient un gros défaut, ils reposaient sur des effets de bord. Comme un langage fonctionnel évalue des expressions, on peut raisonner formellement sur les effets du programme et prouver qu'il est conforme aux spécifications.
FP est simple mais a généré de la réflexion sur le paradigme fonctionnel.

ML

Fins des années 70 et début 80 ML et Hope ont vu le jour.
Ils ont fusionné et donné Standard ML.
Caractéristique de ML : système d'inférence de type.

ML est fortement typé.
Le programmeur n'est pas obligé de les déclarer car le système infère le type en scannant le texte du programme et en vérifiant des incohérences
ML n'est pas purement fonctionnel, il y a des entées-sorties.

Prolog

PROgramming in LOGic.
A été créé à l'université de Marseille en 73.
Les développements ont été faits à Marseille et Edimbourg fin 70.
L'approche a été de programmer des problèmes d'Intelligence Artificielle.
Le programmeur en Prolog écrit des faits avec des relations entre les objets et des relations vraies. Contrairement aux langages procéduraux, il n'y a pas de contrôle explicite de la manière dont le problème est résolu.
La solution du problème est produite par inférence à partir des faits et des règles.
Un programme en Prolog consiste en une série de clauses qui sont de 3 types :

1) Faits - Déclaration des faits sur les objets et leur relation ; les faits sont toujours vrais
2) Règles - La définition des règles sur les objets et leurs relations
3) Questions - La demande de questions sur les objets et leurs relations. Les questions demandent si les instructions sont vraies ou fausses.

L'intérêt de Prolog a été fort dans les années 80 en particulier quand les Japonais ont lancé leur initiative sur la 5ème génération des ordinateurs.
Bataille entre Prolog (logique) et Lisp (fonctionnel) pour résoudre le même genre de problèmes.
Lisp est utile quand on traite des listes, Prolog est bien pour des applications de recherche avec des bases de données et des systèmes-experts.
Prolog n'est pas adapté pour le calcul, manque d'expression composée et n'est pas forcément efficace (en terme de cpu).
Prolog est adapté aux stratégies d'essais et erreurs, aux problèmes de planification, aux systèmes experts.

Syntaxe et Sémantique

Syntaxe (grammaire) : décrit la forme correcte dans laquelle les programmes doivent être écrits.
Sémantique est la signification qui est attachée aux différentes constructions syntaxiques.
Méthodes et notations pour l'expression formelle de la syntaxe : facile.
Syntaxe BNF Backus-Naur Form (Backus-Normal Form).
Formaliser la sémantique difficile.
Description sémantique : en langue naturelle.

Syntaxe BNF

Ce qui est légal et ce qui ne l'est pas.
BNF métalangage : langage pour définir un autre langage (utilisé pour la 1ère fois avec Algol 60)
Définition

- ✓ Un ensemble de nonterminals : Un nonterminal est définie dans une règle de production
- ✓ Un ensemble de terminals : Un terminal est un symbole dans le langage qui est défini ainsi
- ✓ Partie gauche : nonterminal
- ✓ Partie droite : Séquence de terminal et nonterminals définie dans la partie droite
- ✓ Une série de production de règles

nonterminal défini entre brackets
symbole ::= « is defined as »
| signifie ou bien
<digit> ::= 0|1|2|3|4|5|6|7|8|9
<letter> ::=a|b|….x|y|z

Par exemple : ab2 est un identifier si identifier ::=
<letter>|<identifier><digit>|<identifier><letter>

Une grammaire a un nombre fini de règles de production
Mais on peut utiliser la récursion : <identifier> ::= <identifier> <digit>
Une BNF grammaire est « context free » : la partie gauche n'a qu'un nonterminal

EBNF (extended BNF)

::= remplacé par → et <> remplacé par ‘’
exemple :
Exp → term {(‘+’ | ‘-’) term}
Term → factor {(‘*’ |’/’)factor}
Factor → ‘(‘exp’)’ | identifier
Les compilateurs contiennent aujourd'hui ces productions de règles (dirigés par la syntaxe).

Ambiguïté

Dans Algol 60 else est optionnel.
Dans “If a then if b then c else d” else fait référence à quel then ?
La grammaire est ambiguë.

```
If a then
        Begin
        If b then c else d
        End
```

Ou bien

```
If a then
        Begin
        If b then c
        End
else d
```

Sémantique

Peu de progrès.
Sémantique ISO et ANSI encore en anglais.

Sémantique opérationnelle

Définit une machine abstraite simple qui a un ensemble de structures de données et des opérations dont la sémantique est connue.
La sémantique d'un programme est alors définie par les règles qui dictent comment un programme est écrit dans ce langage et comment il peut être exécuté par cette machine abstraite.

Autres approches

Approche dénotationnelle : fait pour les concepteurs de langage, pas pour les implémenteurs.
Approche axiomatique : basée sur la logique symbolique utilise des règles d'inférence (precondition et post condition).

Structure des programmes

Dans un langage avec des blocs structurés comme Pascal, les procédures sont déclarées dans un environnement créé par des blocs fermés. Un appel de procédure est une instruction de plein droit tandis qu'un appel de fonction fait partie d'une expression et redonne une valeur.
Dans un langage procédural, une procédure peut exister comme une entité séparée, tandis que dans un langage purement orienté objet, une procédure est toujours un composant d'une classe.

Pour résoudre un grand problème, il faut le découper en parties distinctes qui interagissent entre elles uniquement à travers des interfaces strictement définies. Pour s'assurer qu'un programme sera modifiable facilement, il doit être conçu de telle manière que les changements soient localisés. Ceci est réalisé par l'utilisation de modules.
Un module est une collection de déclarations de variables, type et procédure en relation.
Un module en Ada ou Fortran 90 peut contenir la définition d'un type.
Un module (une classe) en C++ et Java est la définition d'un type.

Un module est généralement découpé en 2 parties : la partie publique décrit ce que le module fait alors que l'implémentation privée donne les détails de comment les opérations sont faites.
En Ada, ces deux parties sont la spécification et le corps (body) du package.
Les modules supportent une information cachée. Les identifiants déclarés dans l'interface peuvent être utilisés par d'autres modules alors que les identifiants déclarés dans l'implémentation sont cachés. La possibilité de cacher des détails est un outil majeur pour contrôler la complexité.

Un type est caractérisé par sa portée de valeurs et les opérations qui peuvent être exécutées sur les objets de ce type.
Les packages en Ada et les classes en C++ et Java peuvent être utilisées pour définir de nouveaux types.
Les opérations permises sont données dans la partie de l'interface alors que l'implémentation des opérations et la représentation du type sont cachées. Les types définis ainsi sont des types de données abstraits.
Les packages Ada sont compilés séparément mais dans le contexte des autres packages dont ils importent l'information. De cette manière la vérification de type peut être exécutée à travers les limites de module.

Procédures, fonctions, méthodes

En Pascal, C, Java, le mécanisme de passage de paramètres d'un appel par valeur est utilisé pour transmettre l'information à un sous-programme.
Dans un appel par valeur, le paramètre formel agit comme une variable locale.
Les appels par valeur sont coûteux en terme d'espace et de temps quand le paramètre est un objet structuré.
Les appels par référence et appels par résultat de valeurs sont utilisés pour mettre à jour l'information passée dans une procédure.
Ada, C++ et Java permettent aux procédures et méthodes d'être surchargées.
Le seul effet d'un appel de fonction sur le reste du programme se fait à travers le résultat de la fonction.
La gestion du stockage de piles est nécessaire pour supporter les programmes récursifs.
Si la quantité de stockage nécessaire pour les objets locaux peut être déterminée à la compilation, il est possible d'implémenter le langage de telle manière que la position relative de chaque objet dans la zone de données est connue à la compilation (c'était une exigence de Pascal).
Pascal était défini pour être traduit en une seule passe de compilateur.

Structures de données

Un choix judicieux de structures de données appropriées est crucial dans la conception et la production de programmes bien structurés.
Tous les composants d'un array ont le même type et sont accédés par un indice calculable.
En Pascal et C, la taille de l'array est déterminé à la compilation.
En Ada, on peut différer la taille du tableau jusqu'à l'entrée dans le bloc.
En Java, on peut attendre l'exécution de l'instruction.
En C et C++ les pointeurs peuvent être utilisés pour accéder à des éléments de tableaux.
En Pascal (et ses successeurs), un tableau à 2 dimensions est un tableau de tableaux.
Les composants d'un enregistrement (record) peuvent avoir différents types et sont accédés par des noms.
Les enregistrements permettent de relier logiquement des termes de données pour les grouper en une variable structurée.
Pascal est fortement typé.
Les classes en C++ ont des parties fonctions et des parties données. Les classes permettent que les opérations sur les structures de données soient incorporées avec la définition des structures de données.
Les structures dynamiques sont construites à partir d'enregistrement ou de classes qui ont un ou plusieurs champs ou attributs de type pointeur.
La représentation ou la manipulation de chaîne de caractères est possible si la conception du langage permet des opérations qui potentiellement augmentent la longueur de la chaîne.

Conclusions

Bien apprendre et maîtriser 2 langages : par exemple, C et Java (au minimum).
Connaître un langage fonctionnel.
Utiliser et maîtriser si possible une grande palette des possibilités du langage (puissance d'expression).
Écrire des programmes simples, clairs, lisibles et capables d'être repris et retravailler par quelqu'un d'autre.
Ne pas se lancer dans l'optimisation à tout crin, car les compilateurs font un travail excellent en matière d'optimisation.
Ce qui se conçoit bien se programme clairement et les instructions pour sa réalisation et son exécution s'écrivent aisément …