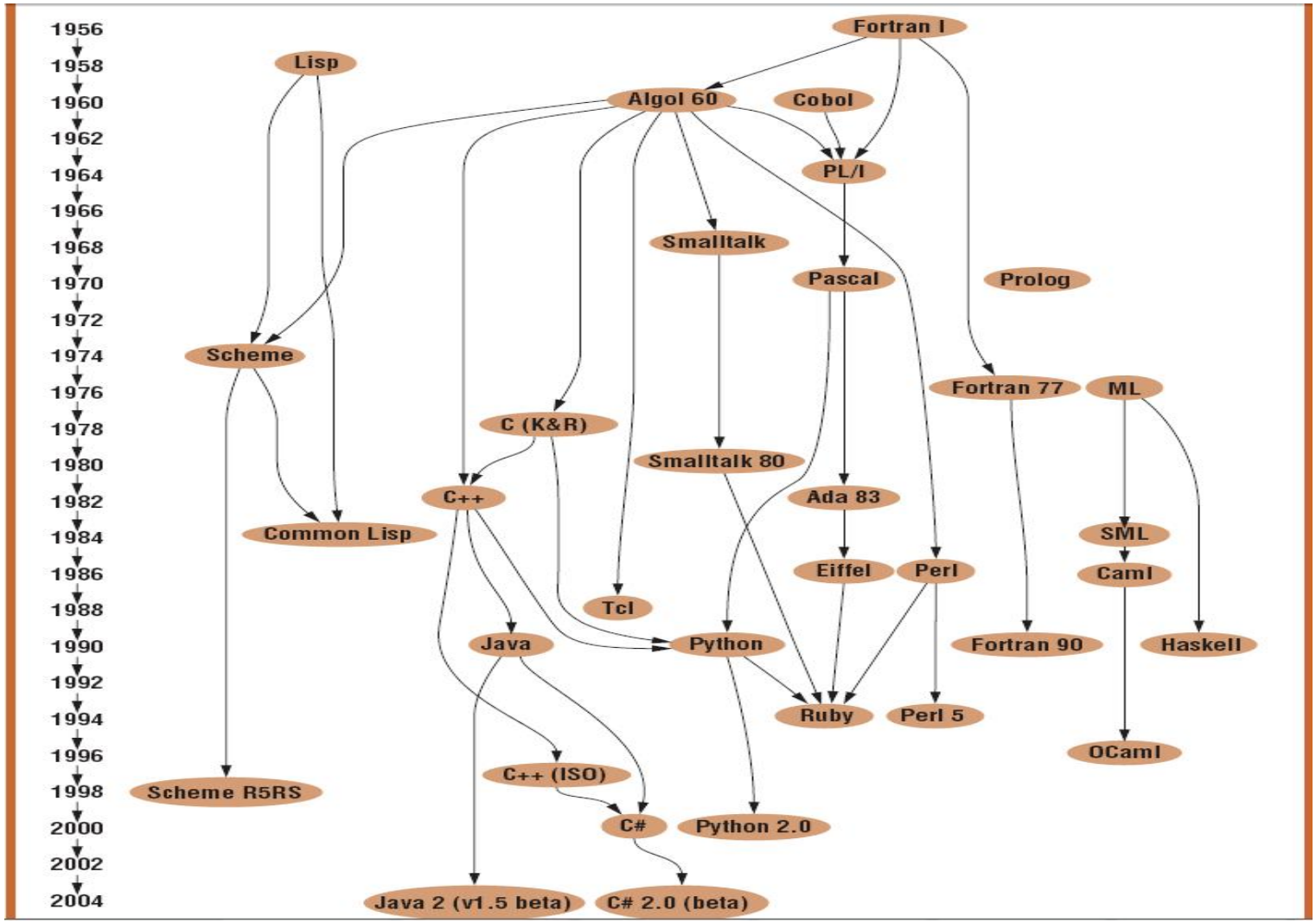
Séance 2

Introduction au langage Ada

Objectifs :

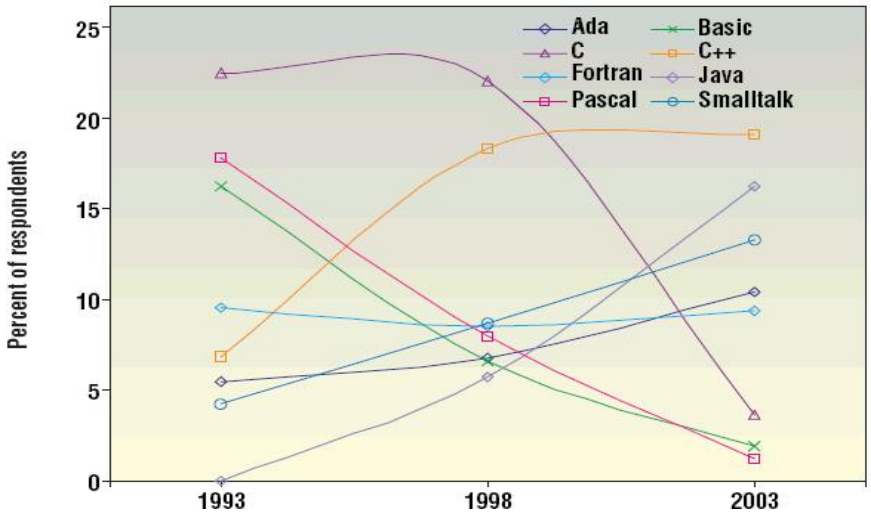
- ✓ Pourquoi Ada ?
- ✓ Évolution des langages de programmation
- ✓ Particularités d'Ada
- ✓ Structure d'un programme Ada.

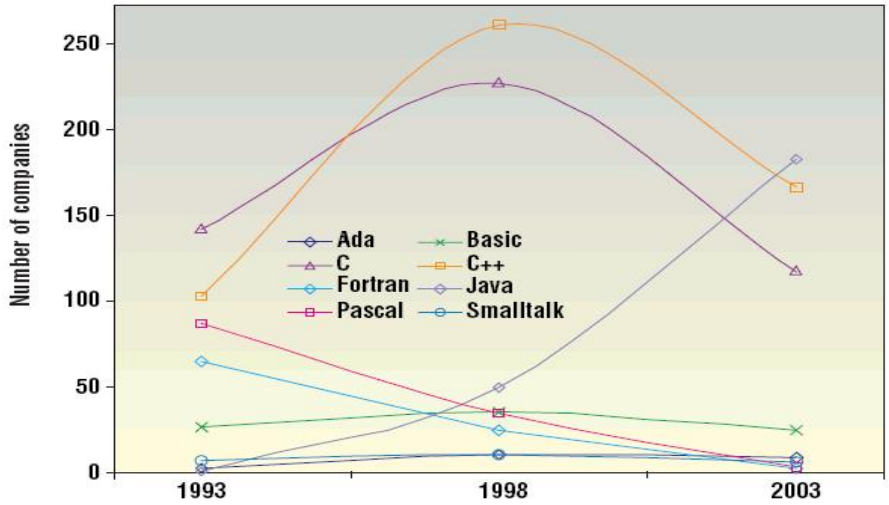
L'arbre généalogique

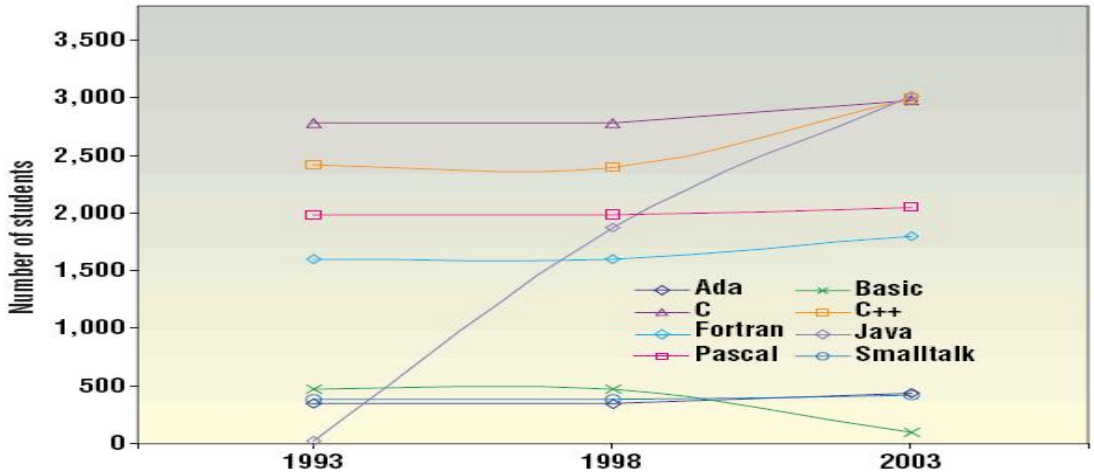


Réf: "An Empirical study of Programming Language Trends"

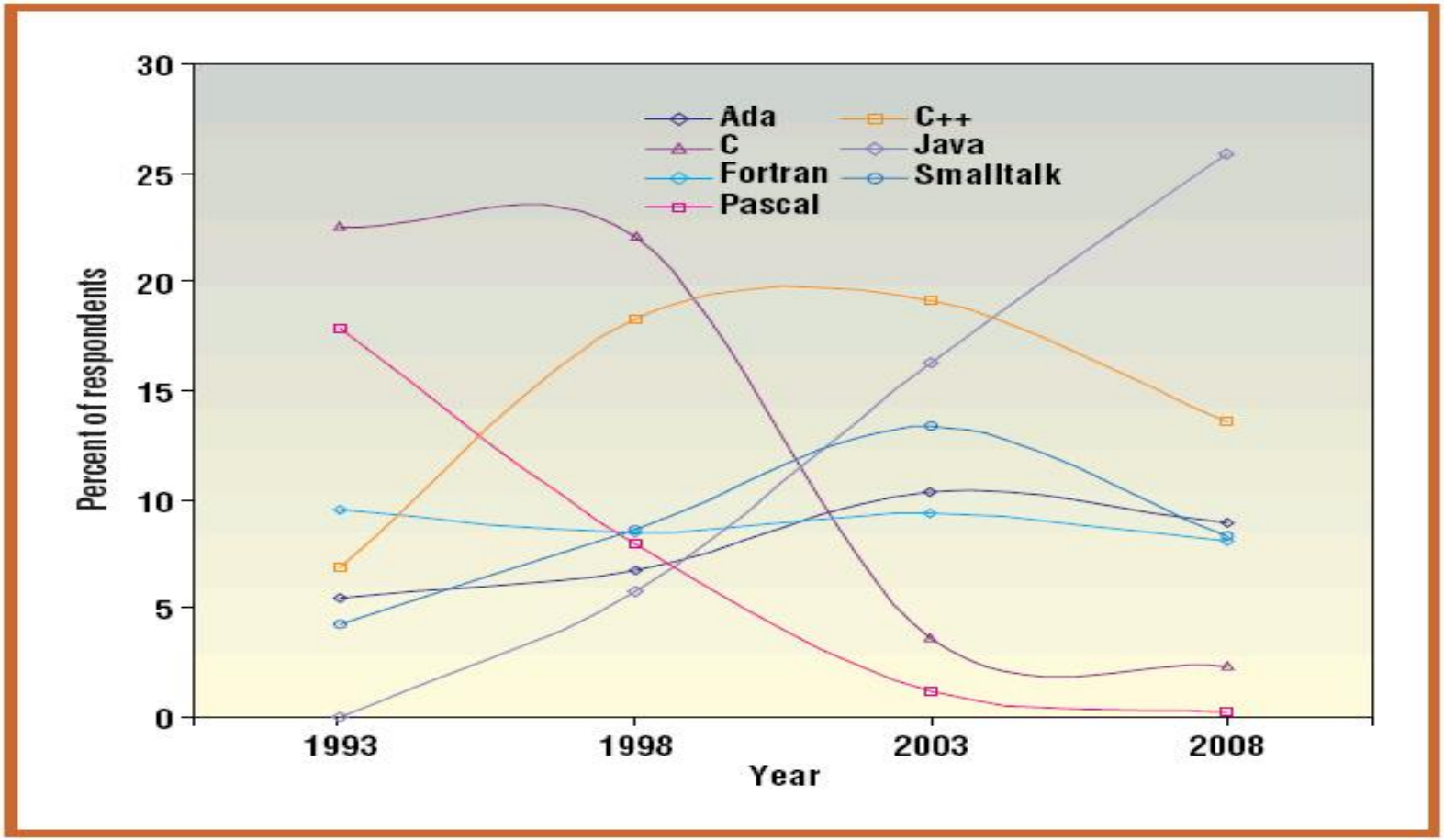
Comparaison chiffrée







Modèle predictif



ANCÊTRES D'Ada

- Ada se veut un langage évolué de haut niveau rassemblant les concepts et les constructions de ses prédécesseurs.

- La plupart des constructions d'Ada sont des constructions qui ont été éprouvées dans de nombreux langages qui ont précédé Ada.

HISTORIQUE

- En 1973, déjà plus de 500 langages étaient utilisés par le Ministère de la Défense Américaine (DoD)
- Plusieurs langages étaient mal utilisés et vieux
- Cette diversité était un des facteurs importants de la montée des coûts du logiciel

Buts d'Ada

- Contrairement aux autres langages, Ada a été développé avec des buts précis et des besoins explicites
- Les besoins sont apparentés et quelques fois en conflits! e.g. les unités de compilation séparées et la programmation concurrente entrent en conflit avec la simplicité visée du langage
- Les buts principaux sont:
  - fiabilité
  - simplicité
  - modularité
  - efficacité

BUTS -- FIABILITÉ ET FACILITÉ DE MAINTENANCE

- Accent sur la lisibilité plutôt que sur la facilité d'écriture
- Notations favorisant les erreurs ont été évitées
- Constructions doivent avoir une sémantique simple et non ambiguë

BUTS -- SIMPLICITÉ

- Le langage a été gardé aussi petit que possible
- Les écueils d'une complication excessive ont été évités.
- Les constructions du langage sont intuitives pour l'utilisateur.

BUTS -- MODULARITÉ

- Le langage doit supporter et favoriser
  - les logiciels modulaires et réutilisables,
  - les unités de compilation séparées.

BUTS -- EFFICACITÉ

- Ada a évité des constructions qui nécessitent
  - des compilateurs trop élaborés,
  - une utilisation inefficace de la mémoire, ou du temps d'exécution,
  - du matériel spécialisé.

PROGRAMMATION EMBARQUÉE (EMBEDDED SYSTEM)

- Les sous-systèmes embarqués sont des composantes de système qui doivent souvent satisfaire des contraintes sévères de :
  - Temps réel
  - Fiabilité.

- Les logiciels systèmes (embarqués) sont habituellement
  - gros
  - ont une grande longévité
  - habituellement en ASM

PRINCIPES D'INGÉNIERIE DU LOGICIEL POUR LES LANGAGES

- Les buts d'Ada de simplicité, de fiabilité, de maintenance, et d'efficacité peuvent être atteints en utilisant les principes du génie logiciel:
  - abstraction,
  - dissimulation d'information,
  - modularité,
  - programmation embarquée,
  - uniformité,
  - complétude,
  - confirmabilité.

LANGAGES STRUCTURÉS

- Les premiers langages, comme FORTRAN et COBOL, permettaient et favorisant au début l'écriture de "programmes spaghetti" qui étaient difficiles à mettre au point et à maintenir.

STRUCTURES DE CONTRÔLE

- Ada supporte les énoncés structurés:
  - énoncés conditionnels (***if*** et ***case***)
  - boucle (***for*** et ***while***)

- Ada fournit des sorties (***exit***) à différents niveaux afin d'éliminer les besoins des "***goto***"

PROTECTION ET TYPE DE DONNÉES

- Ada exige que chaque variable ou expression ait un type explicite, à la compilation
- La protection des données est mise en force par la possibilité d'interdire certaines opérations sur certaines données.

BESOINS DE DONNÉES STRUCTURÉES

- Les premiers langages, e.g. permettaient des constructions de contrôle structurées, mais ne permettaient pas la structuration des données

- PASCAL a introduit des types structurés de données
  - enregistrement
  - tableau
  - pointeur

- Ada possède des types structurés de données, des types étiquetés (***tagged***) dont on peut hériter, en plus des types privés (***private***) qui permettent de protéger les données.

NOMS ASSOCIÉS À DES ENTITÉS

- Ada associe des noms à des entités, e.g. variable, constante, exception, type, paquetage, tâche, etc....

- Cette nomenclature améliore
  - la vérification des types,
  - la lisibilité du programme,
  - la modularité,
  - les abstractions.

PROGRAMMATION CONCURRENTE

- Les premiers langages ne supportaient pas la programmation concurrente.
- Avec ces langages, les logiciels embarqués ("embedded system") étaient une collection de programmes s'exécutant en parallèle.
- Ada permet explicitement des constructions de tâches (***task***) pour la programmation concurrente et permet l'exclusion mutuelle grâce aux objets protégés (***protected type***).
- Les tâches peuvent être créées dynamiquement, et sont logiquement exécutées en parallèle.

MULTI-TÂCHES -- MULTI-PROCESSEURS

- Un multi-processeur est un ordinateur muni de plusieurs processeurs qui s'exécutent physiquement en parallèle.
- Les tâches d'Ada permettent le développement de logiciels contrôlant plusieurs processeurs.
- Les tâches permettent l'implémentation d'applications synchronisées et communiquant entre elles.

MINIMISER L'UTILISATION D'ASM

- ASM a toujours été fortement utilisé pour les logiciels systèmes (embarqués),
- Ada fournit des outils évolués pour accéder au matériel et le contrôler,
- Les constructions d'Ada
  - minimisent les besoins d'ASM,
  - permettent d'isoler les dépendances de la machine,
  - simplifient l'entretien du logiciel.

LIBRAIRIES (BIBLIOTHÈQUES) DE MODULES

- Les bibliothèques de logiciels sont des ensembles de modules partagés ou utilisés par des projets logiciels.

- Les bibliothèques de logiciel sont un facteur majeur dans la réduction des coûts du logiciel.
- Ils ont évolué vers des Framework (Exp: Framework .Net 3.5).
- Attention plus efficaces mais le développement devient plus complexe.

RARETÉ DE LIBRAIRIES

- Les bibliothèques de logiciels réutilisables ne sont pas très importantes, sauf pour FORTRAN et maintenant C, C++ et STL.

- Les bibliothèques de logiciels sont utiles en pratique, si le langage est
  - bien normé,
  - utilisé abondamment,
  - et s'il existe des outils, lors des compilations séparées et de l'édition des liens, permettant de vérifier la consistance et l'intégrité des modules.

Ada FAVORISE LA RÉUTILISATION DES LOGICIELS

- Les paquetages (***package***) d'Ada permettent de regrouper les déclarations apparentées et de les encapsuler.
- Les paquetages peuvent être compilés séparément et favorisent la vérification ("*checking*")
- Les paquetages génériques permettent de définir des squelettes de programmes (sous-programmes) semblables.

NORMALISATION DE LA MAINTENANCE ET PORTABILITÉ

- Une grande partie des langages ne sont pas normés, (e.g. LISP, PROLOG)
- ou deviennent standards après plusieurs années (e.g. PASCAL)
- ou possèdent plusieurs standards ou sous-ensembles (e.g. FORTRAN & COBOL)

Avantages et inconvénients du langage Ada

- *Les avantages :*
  - Très proche de l'algorithmique
  - Fortement typé
  - Nombreuses vérifications faites par le compilateur
  - Programmation modulaire obligatoire
  - Structuration
  - Abstraction (encapsulation, compilation séparée)
  - Temps réel
  - Interfaçage
  - une programmation plus propre avec moins d'erreurs
- Les inconvénients :
  - Contraignant
  - Pas très à la mode.