# IFT2255 - Génie logiciel

Introduction

Bruno Dufour
dufour@iro.umontreal.ca

# Plan de cours

## Horaire



- Cours magistraux :
  - Jeudi de 12h30 à 13h30, Z-210
  - Mardi de 9h30 à 11h30, Z-337
- Démonstrations :
  - Jeudi 13h30 à 15h30, Pav. André-Aisenstadt salle 1175



## IFT2255 sur le web



- Page web du cours:

  `http://www.iro.umontreal.ca/~dufour/cours/ift2255/`
- Remise de travaux, forums, matériel des démos, etc.

  `http://studium.umontreal.ca/`

# Préalables



- Programmation en Java
  - IFT1025 - Programmation 2



# Plan de cours



| Semaine | Sujet | Remise |
|---|---|---|
| 1 | Introduction, historique du GL | |
| 2 | Cycle de vie des logiciels, systèmes de contrôle de révisions | |
| 3 | Modélisation des logiciels : structure | |
| 4 | Modélisation des logiciels : comportement | TP1 |
| 5 | Modélisation des logiciels : OCL | |
| 6 | Modélisation des logiciels : Réseaux de Pétri | |
| 7 | Changements, analyse des besoins | |
| 8 | Localisation des concepts, impact des changements | TP2 |
| 9 | Architectures logicielles | |
| 10 | Patrons de conception | |
| 11 | Tests et vérification | TP3 |



# Évaluation



- Travaux pratiques (3): 35% (10%, 10%, 15%)
- Examens: 65 %
  - Intra : 30 %
  - Final : 35 %
  - Seuil = 40%



# Retards



- Aucun retard ne sera toléré pour la remise de travaux.
- La note attribuée à un travail en retard sera zéro.

## Plagiat



- Les discussions avec les autres étudiants du cours sont encouragées, mais le plagiat sous toute forme sera sévèrement puni, conformément aux règlements de l'Université de Montréal
  - échec du cours
  - sanctions



## Ouvrage de support



- Vaclav Rajlich, *Software Engineering: The Current Practice*, Taylor & Francis, 2012. ISBN: 1439841225



## Bibliographie pertinente




- Ian Sommerville. *Software Engineering* (9e édition), Addison-Wesley, 2010. ISBN: 0137035152.
- Craig Larman. *Applying UML and Patterns* (3e édition), Prentice-Hall, 2004. ISBN: 0131489062.
- Martin Fowler. *UML Distilled* (3e édition), Addison-Wesley, 2003. ISBN: 0321193687.




# Introduction

## Les logiciels et la société



- Aujourd'hui la société dépend à tous les niveaux de systèmes informatiques
  - **Banque**: comptes, guichets auto., bourse, ...
  - **Bureau**: courriel, Word, Excel, ...
  - **Transport**: véhicules, routes, horaires, ...
  - **Communication**: téléphone, vidéo, web, ...
  - **Recherche**: bioinformatique, nanotechnologie, ...
  - **Divertissement**: films, jeux, musique, ...



## Qu'est-ce qu'un logiciel?



- Un logiciel est:
  - Du code exécutable
  - Les données associées au programme
  - Des documents: besoins des usagers, design, guide d'utilisateur, guide de programmation, etc.
- Les logiciels jouent un rôle clé:
  - Produisent, gèrent et présentent de l'information



## Qualités d'un bon logiciel



- **Correct**: programme fiable sans "bogue"
  - Le programme est conforme à sa spécification
  - Il est testé sur plusieurs cas, normaux et extrêmes
- **Convivial**: facile à utiliser
  - Interface usager intuitive, cohérente et esthétique
- **Performant**: rapide et économe en mémoire
  - Pas besoin d'un ordinateur dispendieux
  - Logiciel "réactif" (contrôle, jeux, ...)
  - "Scalable"



## Qualités d'un bon logiciel



- **Facile à maintenir**: modifiable et extensible
  - Chaque logiciel a un cycle de vie
    1. **Analyse** et conception: analyse des besoins, spécification.
    2. **Implémentation**: développement d'algorithmes, codage
    3. **Maintenance**: réparer bogues, extension
  - Cycle de vie: quelques minutes à des dizaines d'années, un seul programmeur à des milliers
  - Concepts de: portabilité, réutilisabilité, modularité, tests de régression, lisibilité, standards et styles de programmation

## Que fait ce programme?

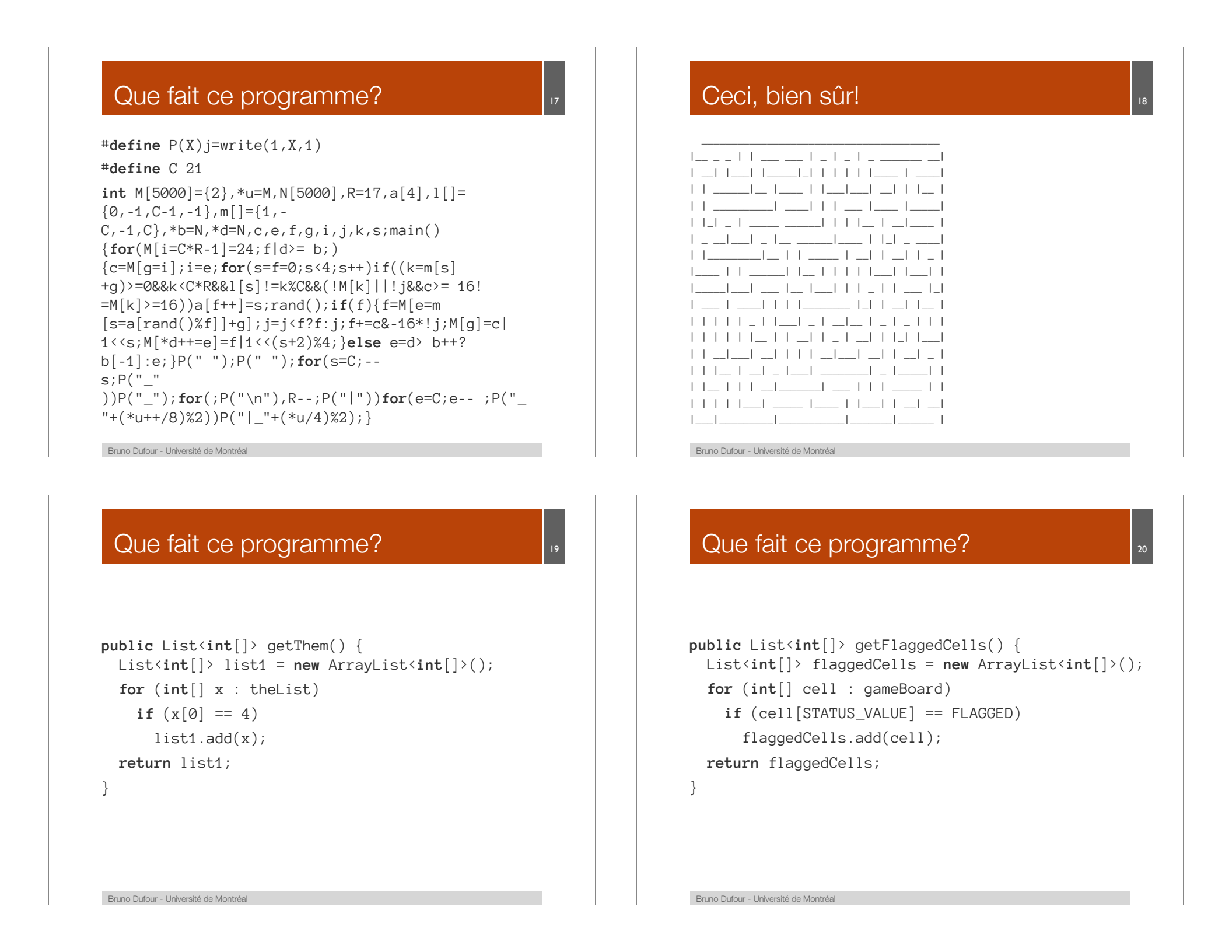


```
#define P(X)j=write(1,X,1)
#define C 21
int M[5000]={2},*u=M,N[5000],R=17,a[4],l[]=
{0,-1,C-1,-1},m[]={1,-
C,-1,C},*b=N,*d=N,c,e,f,g,i,j,k,s;main()
{for(M[i=C*R-1]=24;f|d>= b;)
{c=M[g=i];i=e;for(s=f=0;s<4;s++)if((k=m[s]
+g)>=0&&k<C*R&&l[s]!=k%C&&(!M[k]||!j&&c>= 16!
=M[k]>=16))a[f++]=s;rand();if(f){f=M[e=m
[s=a[rand()%f]]+g];j=j<f?f:j;f+=c&-16*!j;M[g]=c|
1<<s;M[*d++=e]=f|1<<(s+2)%4;}else e=d> b++?
b[-1]:e;}P(" ");P(" ");for(s=C;--
s;P("_"
))P("_");for(;P("\n"),R--;P("|"))for(e=C;e-- ;P("_
"+(*u++/8)%2))P("|_"+(*u/4)%2);}
```



## Ceci, bien sûr!







## Que fait ce programme?



```
public List<int[]> getThem() {
  List<int[]> list1 = new ArrayList<int[]>();
  for (int[] x : theList)
    if (x[0] == 4)
      list1.add(x);
  return list1;
}
```



## Que fait ce programme?



```
public List<int[]> getFlaggedCells() {
  List<int[]> flaggedCells = new ArrayList<int[]>();
  for (int[] cell : gameBoard)
    if (cell[STATUS_VALUE] == FLAGGED)
      flaggedCells.add(cell);
  return flaggedCells;
}
```

## Maintenabilité



- Définition
  - ensemble d'attributs portant sur l'effort nécessaire pour faire des modifications données
- Sous-caractéristiques
  - Facilité d'analyse : effort nécessaire pour diagnostiquer les déficiences et causes de défaillance ou pour identifier les parties à modifier
  - Facilité de modification : effort nécessaire pour modifier, remédier aux défauts ou changer d'environnement
  - Stabilité : risque des effets inattendus des modifications
  - Facilité de test : effort nécessaire pour valider le logiciel modifié



## Qualités du logiciel



- Le génie logiciel doit assurer la qualité
  - Non seulement du logiciel
  - Mais aussi du processus de développement.
    - Productivité
      - diminuer les coûts
      - réutilisation (compromis)
      - utilisation d'outils et environnements
    - Ponctualité
      - livrer logiciel d' "actualité"
      - développement incrémental: livraison précoce du logiciel
    - Visibilité (transparence)
      - documentation



## Exigences de qualité



- Systèmes d'information et de commerce électronique
  - Gestion d'information: création/destruction, consultation et mise à jour de données.
  - Ex: systèmes bancaires, catalogue informatisé (biblio)
  - Caractéristiques importantes
    - Intégrité des données
    - Sécurité
    - Disponibilité des données
    - Performance des transactions
    - Convivialité



## Qualités du logiciel



- Systèmes temps réel
  - Doit répondre aux événements dans un laps de temps limité.
    - Temps de réponse fait partie de la fonctionnalité du logiciel
  - Ex.: système d'injection d'un moteur à combustion, système de guidage d'une sonde
  - Caractéristiques importantes
    - Correction, fiabilité, robustesse
    - Interopérabilité, convivialité

## Qualités du logiciel



- Systèmes distribués
  - Ordinateurs indépendants connectés par un réseau de communication.
  - Aspects à considérer
    - Distribution: données ou contrôle?
    - Tolérance aux fautes
  - Caractéristiques importantes
    - Fiabilité, robustesse
    - Performance
    - Interopérabilité



## Qualités du logiciel



- Systèmes embarqués
  - Système intégré à un dispositif, une machine ou un autre système pour le piloter.
  - Exemples: système de freinage automobile, tableau de commande d'un micro-ondes, etc.
  - Aspect à considérer
    - Interface avec autre système (non humain).
  - Caractéristiques importantes
    - Interopérabilité
    - Réutilisation



## Qualités du logiciel



- Comment mesure-t-on la qualité?
  - Ex. On évalue la fiabilité d'un pont en mesurant par exemple la pression qu'il peut supporter, la résistance au vent, etc.

1. Définir avec précision la caractéristique de qualité à évaluer
2. Établir un ensemble de métriques et un modèle qui permet d'évaluer cette caractéristique à partir des valeurs des métriques



## L'informatique et l'ingénierie



- Les autres disciplines de génie
  - manipulent des composantes:
    - réelles
    - qui interagissent de façon prévisible
    - sont moins sujettes aux changements
- Développer un logiciel n'obéit pas aux contraintes du monde réel
  - Le développement de logiciels est difficile
    - 1/3 des projets sont abandonnés, la moitié dépassent leur budget, et presque tous se terminent avec du retard
  - Le matériel excède souvent ses objectifs alors que le logiciel nécessite plusieurs ajustements

## Ariane 5



- Fusée Ariane 5
- Construite par l'Agence Spatiale Européenne
- Première mise à feu en juin 1996
- A explosé 40 secondes après le lancement
  - Heureusement, pas de passagers à bord
- Coûts:
  - Fusée et contenu : 500 millions $
  - Développement : 7 milliards $
- Cause de l'explosion: faute du logiciel

http://www.ima.umn.edu/~arnold/disasters/ariane.html, http://www.around.com/ariane.html



## Que s'est-il passé?



- Dépassement de capacité durant la conversion de la vélocité d'un entier de 64 bits à un entier de 16 bits
- L'exception n'a pas été captée
  - Le système de référence d'inertie a fait défaut (ainsi que le système de secours)
  - La fusée a dévié de sa trajectoire
  - Le module d'autodestruction s'est (correctement) activé
- Le code provenait d'Ariane 4
  - Même logiciel, mais des environnements d'exécution différents
    - Le code était demeuré actif après la phase de vol auquel il était dédié



## Ariane 5 – Un cas isolé?



Non.



## Autres bogues célèbres



- Mars Pathfinder (1997)
  - Concurrence
- Mars Spirit Rover (2003)
  - Manque de mémoire
- Inversion dans le système de bord du F16
  - L'avion se serait retourné à chaque fois qu'il traversait l'équateur
- Panne de courant en Amérique du nord (2003)
  - Problème de concurrence a paralysé le système d'alarme durant plus d'une heure
- Samy / MySpace XSS (2005)

## Évolution du génie logiciel



- 1950: les logiciels sont développés comme du matériel
  - Ex: avions, circuits, etc.
  - Économie: le temps d'ordinateur coûte plus cher que le salaire des opérateurs (e.g. $600/hr vs $2/hr)



## Évolution du génie logiciel



- 1960: les logiciels ne sont plus du matériel
  - Propriétés:
    - Invisibles
    - Complexes
    - Exécutés par des ordinateurs
    - Difficiles à modifier
  - Meilleurs outils: compilateurs, systèmes d'exploitation
  - Processus: coder puis corriger
  - Plusieurs succès (ex. Apollo)
  - Problèmes: échecs de la majorité de systèmes de grande taille, code impossible à maintenir



## Évolution du génie logiciel



- 1970: Approches formelles sont développées
  - Programmation structurée, élimination des "goto"
  - Modèle de développement en cascade (Waterfall)
  - Notions d'étude des besoins et de conception dans le processus de développement
  - Problèmes des méthodes formelles
    - Succès limités aux petits programmes critiques
    - Preuves démontrent la présence de fautes, pas leur absence



## Évolution du génie logiciel



- 1980 : Productivité, réutilisation, objets
  - Développement de processus permettant de produire des logiciels
  - Augmentation de la productivité
  - Travailler plus rapidement: outils et environnements
  - Travailler plus intelligemment : processus et méthodologie
  - Éviter le travail inutile: réutilisation, objets
    - Librairies orientées objet (Smalltalk, Eiffel, C++)

## Évolution du génie logiciel



- 1990: Modélisation
  - Logiciels libres (open source)
  - Systèmes hérités (legacy systems)



## Évolution du génie logiciel



- 2000: “gros” logiciels, “frameworks”
  - La maintenance est très importante pour les logiciels de grande taille
  - Quelques examples:
    - F-22 Raptor: 1.7M LOC
    - Linux kernel: ~6M LOC
    - Boeing 787: 6.5M LOC
    - Windows Vista: ~50M LOC
    - Automobile: ~100 MLOC (30-100 microprocesseurs)



## Pourquoi pas de “code & fix”?



- Écart croissant entre l’utilisateur et le programmeur.
- Taille et complexité des systèmes augmentent
  - Plus grande puissance de calcul
- Développement du logiciel en équipe élargie
- Temps d’accès au marché de plus en plus critique
- Changement de l’économie informatique:
  - Coût du matériel diminue
  - Coûts de développement et maintenance augmentent
- Réseaux locaux et grande distance étendus
- Adoption des technologies orientées objets
- Conception d’interface graphiques utilisateurs



## Génie logiciel



- Une discipline qui comprend:
  - Le processus de développement de logiciels
  - La méthodologie pour l’analyse, la conception, le développement, la vérification et la maintenance de logiciels
  - Des outils qui supportent le processus et la méthodologie

# Les outils du génie logiciel

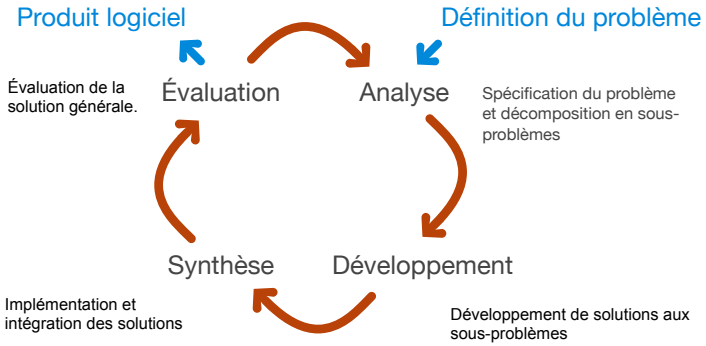


- Lorsqu'un logiciel ne se comporte pas correctement, il est difficile de trouver la cause
- Le génie logiciel propose une méthodologie et des outils qui permettent :
  - de prévoir les problèmes
  - de prévenir les problèmes
  - d'identifier et de corriger les problèmes
  - d'éviter les défaillances en présence de problèmes
- La technologie développée pour les compilateurs est très utile pour ces tâches



# Le processus du génie logiciel

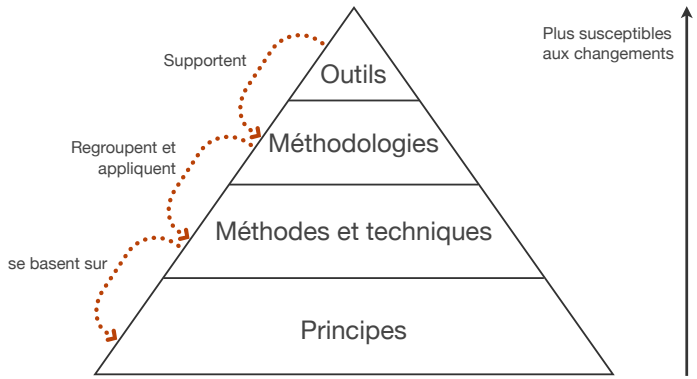







# Méthodes, techniques et outils








# Défis du génie logiciel



- Assurer la qualité des logiciels produits
- Minimiser les coûts de développement tout en répondant aux exigences croissantes
- Réduire les temps de développement
- Instituer l'usage des nouvelles technologies (méthodes et outils) du génie logiciel
- Arriver à simultanément:
  - Assurer la maintenance du nombre croissant de logiciels (vieillissants!)
  - Conserver un rythme de production logicielle qui puisse réponde à la demande

## Principes du génie logiciel



- Rigueur et formalité
- Séparation des préoccupations
- Modularité
- Abstraction
- Anticipation du changement
- Généralité
- Construction incrémentale



## Principes du génie logiciel



- Rigueur et formalité
  - Rigueur:
    - Logiciel doit offrir une solution précise.
    - Développement sérieux, soigné, non approximatif.
  - Formalité
    - Summum de la rigueur
    - Descriptions et validations s'appuient sur des lois mathématiques.



## Principes du génie logiciel



- Séparation des préoccupations
  - Consiste à considérer séparément différents aspects d'un problème pour en maîtriser la complexité.
  - Peut prendre différentes formes:
    - séparation dans le temps (cycle de vie)
    - séparation des qualités à optimiser (ex. fonctionnalité avant performance)
    - séparation des vues d'un système (vue des données, vue du contrôle)
    - séparation du système en parties (cf. modularité)



## Principes du génie logiciel



- Modularité (séparation des parties logiques)
  - Principe qui consiste à décomposer un système en sous-systèmes plus simples
  - Permet de considérer séparément
    - contenu du module
    - relations entre modules
  - Caractéristiques d'une bonne conception modulaire:
    - Regroupement des aspects logiques semblables
    - Indépendance des modules

## Principes du génie logiciel



- Abstraction (séparation des aspects)
  - Consiste à mettre les détails de côté pour ne considérer que les aspects jugés importants d'un système.
  - Il existe différents niveaux d'abstraction:
    - Ex.: circuit électronique
    - Niveau 1: Equation mathématique
    - Niveau 2: Circuit logique des composants
    - Niveau 3: Plan physique des composants réels au sein du circuit intégré.



## Principes du génie logiciel



- Anticipation du changement
  - Consiste à prévoir les changements de façon à faciliter la gestion de ces évolutions inévitables.
  - Nature des changements:
    - perfectifs (nouveaux besoins)
    - correctifs
    - adaptatifs
  - Nature des problèmes:
    - localisation des erreurs (cf. modularité, couplage)
    - gestion des versions multiples du système



## Principes du génie logiciel



- Généralité
  - Consiste à résoudre un problème plus général que le problème spécifique qui est adressé.
  - Solution pourra être réutilisée
  - Ex. On veut convertir une chaîne de caractères en un tableau contenant les mots de cette chaîne…



## Principes du génie logiciel



- Construction incrémentale (raffinement)
  - Consiste à construire une solution étape par étape en s'approchant de plus en plus du but
  - Chaque nouveau résultat est construit en étendant le précédent
  - Ex.: réaliser les fonctions essentielles d'un système puis ajouter aspects secondaires
  - Possibilité de produire des prototypes intermédiaires

# Les acteurs du génie logiciel