



NOTES DE COURS

LECTURE NOTES

INF421

Génie Logiciel Avancé

Advanced Software Engineering

Filière : Master 1 INFORMATIQUE

Enseignant

Dr SOH Mathurin

Chargé de cours

Email : mathurin.soh@univ-dschang.org / mathurinsoh@gmail.com

Volume horaire : 60h (30h CM + 10h TD + 10h TP + 10h TPE) 6 credits

Objectif : L'étudiant doit être capable de :

- maîtriser les processus de développement logiciel en s'appuyant sur des techniques formelles et sémi-formelles de modélisation,
- développer des compétences approfondies requises pour intervenir dans toutes les phases du développement d'un logiciel.

Programme :

- 1- **Spécification logicielle:des méthodes informelles aux méthodes formelles**
 - Panorama des méthodes des spécifications
 - Faisabilité et analyse des besoins
 - Méthodes formelles pour la spécification logicielle
- 2- **Les principaux processus de développement logiciel**
 - Grands principes, cycle de vie en Cascade, évolutif, incrémentale, etc
- 3- **Méthodes Agiles**
- 4- **Ingénierie des exigences**
- 5- **Modélisation du système**
- 6- **Conception architecturale**
- 7- **Conception et implémentation des logiciels**
- 8- **Test du logiciel**
- 9- **Evolution du logiciel**
- 10- **Modelisation objet et le langage d'expression de contraintes**

Projet : Le projet consiste à développer un logiciel de façon collaboratif en équipe (et en concurrence avec d'autres équipes), en utilisant les méthodes et outils de génie logiciel qui seront découvertes pendant le cours.

Travaux dirigés :

Les travaux dirigés sont obligatoires. Il est exigé à chaque étudiant de s'attaquer aux exercices proposés, non pas pour absolument proposer des solutions finies, mais d'y appliquer les notions. Les solutions devant être élaborées pendant la correction de ces travaux dirigés.

Travaux pratiques :

La réalisation de travaux pratiques représente une part importante de l'apprentissage. Ils consistent en travaux de programmation ou en problèmes à résoudre qui, la plupart du temps, sont une application concrète des concepts présentés dans le cours, et vérifiant la compréhension de la matière. Ces travaux seront réalisés en utilisant des outils logiciels d'optimisation

Travail personnel :

L'étudiant sera fortement invité à approfondir les concepts vus en classe à travers les exemples du cours et les exercices proposés en travaux dirigés.

Evaluation des connaissances

- Un contrôle continu (30%) sous forme écrite, Participation aux cours, TD, TP (de durée 2h - Vers fin novembre)
- Un examen final (70%) (Forme Ecrit de durée 2h - Vers mi janvier)

Supports pédagogiques

- Notes de cours mises à la disposition de l'apprenant ;
- Études de Cas ; travaux dirigés et travaux pratiques.

A. Comprendre l'activité de spécification logicielle

Définition :

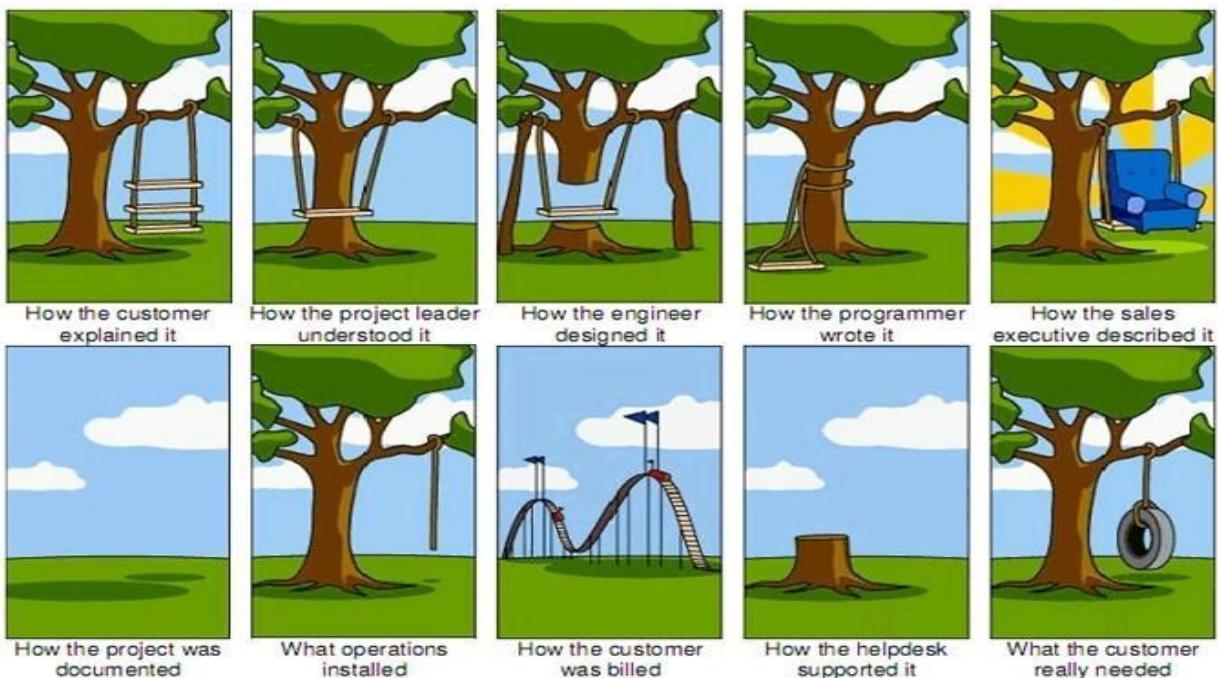
- définir ce que doit faire un logiciel ...
- avant d'écrire ce logiciel !
- => *le quoi, sans le comment !*

S'oppose à **implémentation**,

- comme le plan d'un pont à sa construction.
- Sauf... qu'en informatique, plan et réalisation ne manipulent que de l'écriture...

La spécification est la documentation des exigences du système. Cette vision du système se situe à un niveau très abstrait. Cependant, elle doit être complète et précise, de sorte que tout système qui satisfait ces exigences documentées comble correctement les besoins des utilisateurs. La spécification est comme un véhicule qui transmet les besoins des utilisateurs aux développeurs du système. Pour cela, la spécification doit être utilisée et comprise par toutes les personnes concernées par le développement du système

1) Pourquoi spécifier ?



Définir le travail de réalisation, avant de le faire ...

- la spécification est censée être moins coûteuse, plus rapide à obtenir

Comment vérifier la correction d'un programme,

- si l'on ne sait pas ce qu'il est censé faire ...
- la spécification est la référence pour toutes les activités de vérification et de validation (V&V) :
 - ◆ tests, preuves, mesures, inspections...

Obtenir une description **stable**, plus **abstraite**, moins dépendante des contingences du matériel, des systèmes, des fluctuations de l'environnement.

Spécifications fonctionnelles vs extrafonctionnelles

- En génie logiciel, on distingue :
 - spécification fonctionnelle : décrit le « fonctionnement » du logiciel, le quoi
 - spécification extrafonctionnelle (ou non fonctionnelle) : les conditions de fonctionnement, le comment
- Dans le monde industriel, *spécification* sous-entend, *spécifications techniques*
- En génie logiciel, par défaut *spécification* sous-entend *spécification fonctionnelle*
- Les *spécifications techniques du logiciel* : coût, temps de réponse, performance, robustesse, capacité de charge, consommation de ressource, confort d'utilisation, ergonomie ... sont appelées :
 - qualités de service (QoS)
 - spécifications non fonctionnelles
 - spécifications extrafonctionnelles

Qualités recherchées pour une spécification fonctionnelle

- précision, non ambiguïté, non contradiction,
- concision, abstraction,
- complétude,
- facilité d'utilisation : écriture, lecture, vérification
- réalisable avant l'implémentation,
- si possible à un coût réduit,
- référence contractualisable, pour les litiges...

2) Panorama des méthodes des spécifications

Principaux formalismes de spécification

- Texte informel
 - En langage naturel (cahier des charges, commentaires)
 - éventuellement encadré par une méthode ou un formalisme graphique (contraintes UML)
- Graphique
 - sauf exception (réseaux de Petri...) rarement très formel : (SADT, UML)
 - pas très précis, mais utile pour la synthèse et en complément
- Semi-formel
 - Sans ambiguïté, leur expressivité est insuffisante pour établir une preuve, mais on peut souvent les tester :
 - ◆ techniques assertionnelles, spécification axiomatique
- Formel
 - Sans ambiguïté, leur expressivité est suffisante pour tout décrire, donc pour établir des preuves, humaines (« à la main ») ou automatiques

B. Motivation et justification des méthodes formelles et semi-formelles

La construction de systèmes de qualité tout en respectant les contraintes de temps et de budget représente toujours un formidable défi pour les informaticiens. Dans les domaines où la sécurité des personnes et des biens est en cause, ce défi est encore plus grand. Une des approches proposées pour résoudre ce problème consiste à utiliser des outils mathématiques pour spécifier, valider et vérifier les systèmes informatiques. On désigne communément ces approches basées sur les mathématiques comme des méthodes formelles. Le choix des mathématiques s'explique par le besoin de rigueur et de précision dans la description du comportement de systèmes complexes, et par la nécessité de disposer de mécanismes d'abstraction pour juguler la complexité. Les méthodes formelles permettent également de pallier aux faiblesses des méthodes traditionnelles de tests qui ne peuvent traiter de manière exhaustive tous les cas possibles d'utilisation d'un système.

Intérêts et limites des méthodes semi-formelles

- SADT, SA-RT, SSADM, ...
- JSD-JSP,
- Merise, Axial, ...
- OOA, OMT, UML
- ...

L'analyse du problème est faite.

Contribution positive même si suffisante.

Le problème est dégrossi.

→ impossible de raisonner formellement sur le système en vue.

→ Il peut y avoir des ambiguïtés, des erreurs.

C. Enjeux et problématique des méthodes formelles dans les cycles de développement logiciel

Développement de systèmes informatiques

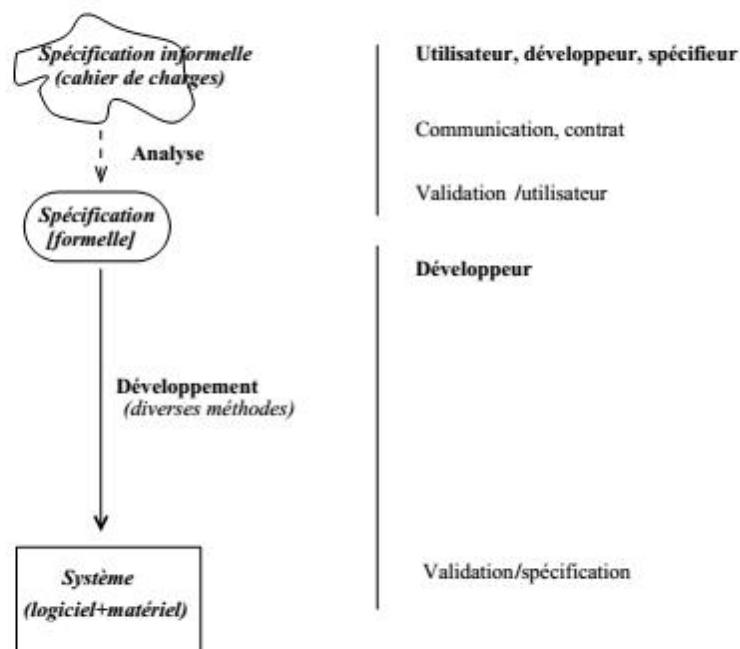


FIG. 3 – Problématique du développement de système

L'industrie développe maintenant des systèmes larges et complexes. Malgré le fait qu'un grand nombre de

logiciels aient été développés, les processus utilisés et la qualité des résultats obtenus sont encore pauvres. Le coût et le temps nécessaires pour développer un logiciel sont imprévisibles et souvent très élevés. Le développement de systèmes larges et complexes est souvent achevé en retard par rapport au plan initial. Pour résoudre ces problèmes, il est indispensable d'apporter des améliorations au processus de développement.

La spécification est un élément critique dans le processus de développement du logiciel. L'écriture de la spécification permet une compréhension approfondie du logiciel à développer. C'est en fait un moyen pour assurer une meilleure communication entre toutes les personnes concernées par le développement. La spécification est la documentation des exigences du système. Cette vision du système se situe à un niveau très abstrait. Cependant, elle doit être complète et précise, de sorte que tout système qui satisfait ces exigences documentées comble correctement les besoins des utilisateurs. La spécification est comme un véhicule qui transmet les besoins des utilisateurs aux développeurs du système. Pour cela, la spécification doit être utilisée et comprise par toutes les personnes concernées par le développement du système.

1) La spécification informelle

Actuellement, la plupart des concepteurs écrivent les spécifications en langage naturel. Le langage naturel est compréhensible par les utilisateurs, les analystes et les développeurs. Chacun d'eux est habitué à lire et à écrire des documents en langage naturel, puisqu'il n'exige pas une spécialisation particulière. L'organisation de ces documents est bien familière: table des matières, chapitres, sections, etc. La spécification informelle s'accorde bien avec les méthodes actuelles utilisées pour développer un logiciel. Cependant, il est possible qu'une spécification en langage naturel ait plusieurs interprétations. Cela signifie que les spécifications informelles sont prédisposées à être incomplètes et incohérentes, à cause de leur ambiguïté et de l'incapacité d'en faire une vérification méthodique et rigoureuse.

2) La spécification formelle

Les méthodes de spécification formelles utilisent des techniques mathématiques pour décrire un problème. L'utilisation de notations formelles résout le problème de la variété d'interprétations par la rigueur du formalisme, l'abstraction, la syntaxe et la sémantique mathématiques bien définies. Elles engendrent des spécifications précises que l'on peut vérifier de manière systématique à l'aide d'outils. En identifiant tôt les problèmes du système, il est encore possible de les corriger avec un coût minimal. Ceci peut réduire le coût et la durée du développement. Ces améliorations pourraient aboutir à un processus prévisible qui produit un logiciel ayant un nombre réduit de fautes.

Actuellement, les méthodes formelles ne sont pas encore largement utilisées dans l'industrie. Une raison est la difficulté d'incorporer la technologie formelle dans l'industrie. Par exemple, l'utilisation d'une méthode formelle allonge typiquement la phase de spécification et réduit la phase de test. En conséquence, le code est produit un peu plus tard dans le cycle de développement, ce qui donne l'impression que le projet n'avance pas aussi vite qu'avec un processus traditionnel.

► Principe des méthodes formelles

- Utiliser les **mathématiques** pour concevoir et si possible réaliser des systèmes informatiques
 - **Spécification** formelle / **Vérification** formelle / **Synthèse** formelle
- Avantage : **non ambiguïté** et **précision** des mathématiques !
 - A et B ou C: $(A \wedge B) \vee C$ vs. $A \wedge (B \vee C)$?
 - $a < 3$ vs. $a \leq 3$?

L'avantage principal des méthodes formelles est l'utilisation de concepts de la logique et de la technique mathématique. Ces concepts fournissent des outils effectifs qui organisent les pensées des concepteurs et qui facilitent la communication entre toutes les personnes concernées par le développement. De plus, ils nous permettent de décrire de manière précise, non ambiguë, les demandes énoncées par l'utilisateur du système logiciel à réaliser. Les notions d'ensemble, de relation, de fonction et leurs différentes propriétés et opérations, avec les quantifications universelles et existentielles, nous permettent d'établir une spécification d'une manière simple et claire et de démontrer mathématiquement les propriétés de la spécification

► Objectif global : améliorer la confiance !

- Dans le *logiciel* et le *matériel*
 - « Program testing can be used to show the presence of bugs, but never show their absence »
– Edsger W. Dijkstra
- *Pas* une « silver bullet » !
 - « Beware of bugs in the above code; I have only proved it correct, not tried it. » – Donald E. Knuth, dans le code source de TeX

Approche générale

En plusieurs étapes

1. *Spécifier* le logiciel en utilisant les mathématiques
2. *Vérifier* certaines propriétés sur cette spécification
 - Corriger la spécification si besoin
3. (parfois) *Raffiner* ou *dériver* de la spécification un logiciel concret
4. (ou parfois) Faire un *lien* entre la spécification et (une partie d') un logiciel concret

De nombreux formalismes

- 1) Spécifications algébriques,
- 2) Machines d'état abstraites,
- 3) Interprétation abstraite,
- 4) Model Checking,
- 5) Systèmes de types,
- 6) Démonstrateurs de théorèmes automatiques ou interactifs,

• ...

Développement correct de logiciels

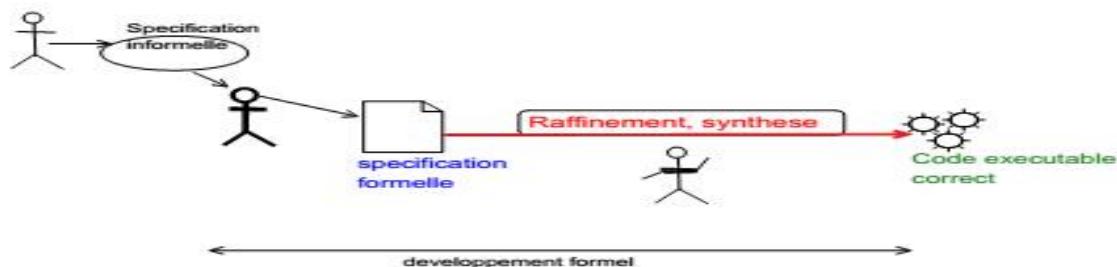


FIG. 2 – Activité de développement formel

Pourquoi utiliser des méthodes formelles ?

Reformuler la spécification en utilisant les mathématiques force à être précis

Un moyen d'avoir des spécifications claires

On explicite le « quoi » mais pas le « comment »

Au passage : aussi crucial d'expliciter le « pourquoi », documentez !

Avec des outils automatiques ou des vérifications manuelles, fournir des preuves de fiabilité

. 60 à 80% du coût total est la maintenance (source Microsoft)

. 20 fois plus cher de gérer un bug en production plutôt qu'en conception

D. Applications des méthodes formelles

Les méthodes formelles ont été utilisées avec succès dans le domaine du transport, où la sûreté des automatismes est primordiale. Les logiciels tiennent une part importante dans le fonctionnement des systèmes de contrôle/commande du mouvement. L'utilisation de méthodes formelles, pour développer ce type de logiciels, est une solution au double impératif: la qualité et la sûreté de fonctionnement. Une des applications, parmi les applications les plus connues dans le domaine du transport, est celle du Métro de Paris, en utilisant la méthode B de Abrial.

Quelques autres exemples

- [NASA](#): Contrôle de satellites, navettes
- [RATP](#): Contrôle du métro de Paris
- [SNCF](#): Limitations de vitesse des trains
- [FAA](#): Prévention des collisions en vol
- [Darlington, Ontario](#): Extinction de réacteurs nucléaires

E. En bref

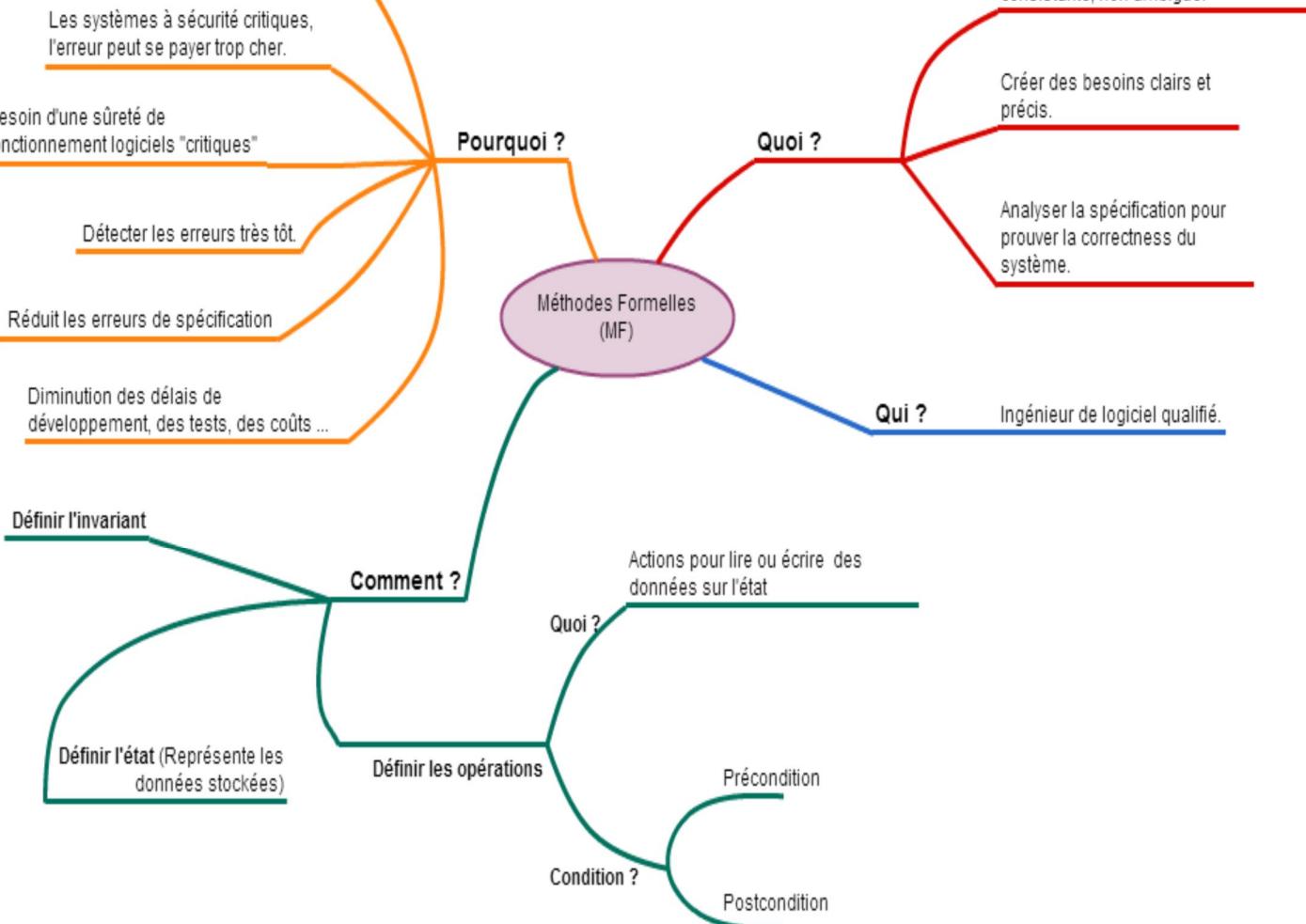
Les méthodes formelles sont des techniques permettant de raisonner rigoureusement, à l'aide de la logique sur des programmes afin de démontrer leur validité par rapport à leur spécification • Ces méthodes permettent d'obtenir une très forte assurance de l'absence de bug • elles sont coûteuses en ressources (humaines et matérielles) et actuellement réservées aux logiciels les plus critiques.

L'utilisation de spécifications formelles permet:

une compréhension approfondie du logiciel à développer;

- de dériver une maquette réalisant partiellement les fonctionnalités désirées;
- pour des applications critiques, de valider formellement la correction du logiciel;
- la génération automatique de jeux de tests.

Pour produire un logiciel « correct par construction », c'est-à-dire conforme à sa spécification.



F. Conclusion

Les Méthodes de spécification formelles sont utiles et nécessaires pour spécifier clairement et précisément le comportement attendu d'un système. Elles sont adaptées essentiellement pour des applications sensibles: nucléaire, avionique, médical, finances, électronique

G. Exercices

- qu'est-ce qu'une méthode formelle?
- qu'est-ce que la spécification formelle?
- quel est l'intérêt des méthodes formelles?
- qui utilise les méthodes formelles?

Réponses :

- qu'est-ce qu'une méthode formelle? Méthode basée sur les mathématiques. Les méthodes formelles sont des techniques basées sur les mathématiques pour décrire les propriétés d'un système. Elles fournissent un cadre systématique pour développer le système et pour valider et vérifier le système
- qu'est-ce que la spécification formelle? spécification utilisant une méthode formelle
- quel est l'intérêt des méthodes formelles? clarté, concision, jeux d'essai et automatisation
- qui utilise les méthodes formelles? l'industrie de pointe, projets sensibles

Ex1 : ce code contient-il une erreur ?

► Calcul de la valeur absolue d'un nombre en langage C

```
int z_abs_x(const int x)
{
    int z;
    if (x < 0)
        z = -x;
    else /* x >= 0 */
        z = x;
    return z;
}
```

Ex1 : ce code contient-il une erreur ?

► Calcul de la valeur absolue d'un nombre en langage C

```
int z_abs_x(const int x)
{
    int z;
    if (x < 0)
        z = -x; // Solution : si x = -231 ?
    else /* x >= 0 */
        z = x;
    return z;
}
```

Solution : si $x = -2^{31}$?
 2^{31} n'existe pas, seulement $2^{31}-1$

Ex2 : ce code contient-il une erreur ?

► En Java, recherche par dichotomie dans un tableau trié

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2;
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1;
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1); // key not found.
}
```

Ex2 : ce code contient-il une erreur ?

► En Java, recherche par dichotomie dans un tableau trié

```
public static int binarySearch(int[] a, int key) {
    int low = 0;
    int high = a.length - 1;

    while (low <= high) {
        int mid = (low + high) / 2; // dépassement si low + high > 231-1
        int midVal = a[mid];

        if (midVal < key)
            low = mid + 1;
        else if (midVal > key)
            high = mid - 1;
        else
            return mid; // key found
    }
    return -(low + 1); // key not found.
}
```

Solution
6: **int mid = low + ((high - low) / 2);**
Impact
Bug présent de le JDK de SunOracle ! Il a impacté des utilisateurs
http://googleresearch.blogspot.com/2006/06/ext_a-extra-read-all-about-it-nearly.html

1. Exemple de méthodes formelles

Beaucoup de notations et méthodes différentes

- CSP (Hoare, 1987)
- LOTOS (Bolognesi Brinksma, 1987)
- Z (Spivey, 1988)
- CCS (Milner, 1989)
- Spec (Berzins Luqi 1990)
- VDM (Jones, 1992)
- ONE (Meer Roth Vuong 1992)
- Larch (Guttag Horning 1993)
- Méthode B Jean-Raymond Abrial (1990)

2. Classification des méthodes formelles

Il ya plusieurs critères possibles : Le type de raisonnement, l' aspect du système et le langage

Critères possibles :

- ▶ type de raisonnement :
 - ▶ structure (par modèle, opérationnel) :
 - ▶ modèle mathématique : Z, B, VDM, AMN, automates, réseaux de Petri...
 - ▶ modèle algorithmique : CLU
 - ▶ théorie des types : Coq,
 - ▶ algèbres de processus : CSP, CCS, π -calcul, ...
 - ▶ files d'attentes : SDL, QNAP, ...
 - ▶ propriétés (axiomatique) : réécriture, LP, spécifications algébriques (LPG, ASL, CLEAR, OBJ, PLUSS, LARCH, CASL), λ -calcul, logiques temporelles (TLA, LTL...), logiques diverses...

Pour chaque cas, il existe des approches hybrides (LOTOS, COLD, RSL...)



▶ aspect du système :

- ▶ statique : Z, B, VDM, AMN...
- ▶ dynamique : automates, réseaux de Petri, SDL, CSP, CCS, TLA, π -calcul ...
- ▶ fonctionnel : spécifications algébriques, λ -calcul

Pour chaque cas, il existe des approches hybrides (LOTOS, COLD, RSL...)

▶ langage :

- ▶ général : Z, B, VDM, spécifications algébriques, CLU...
- ▶ dédié : automates, réseaux de Petri, TLA, SDL, CSP, CCS, π -calcul, λ -calcul...

Pour chaque cas, il existe des approches hybrides (LOTOS, COLD, RSL...)

En général, les approches s'accompagnent d'outils de preuve.

Les approches sont des méthodes si : **méthode + outils de développement**

Premier contact avec une méthode formelle : La méthode B

C'est une Méthode de Construction de logiciels corrects par construction proposée par Jean-Raymond Abrial et permettant de Raffiner une spécification abstraite en utilisant la logique de Hoare.

La méthode B tout au long du cycle de vie d'un logiciel utilise un formalisme unique connu sous le nom du Langage B .En effet B découpe le système en plusieurs machines abstraites AMN (Abstract Machine Notation) ;chaque AMN subira des raffinages successifs jusqu'à obtention d'un pseudo - code qui sera facilement transcodé en un langage de programmation impératif .Ainsi l'implantation peut Se faire sur les spécifications d'une ou de plusieurs machines abstraites raffinables ; de cette manière ,un projet se construit peu à peu selon une architecture en couches.

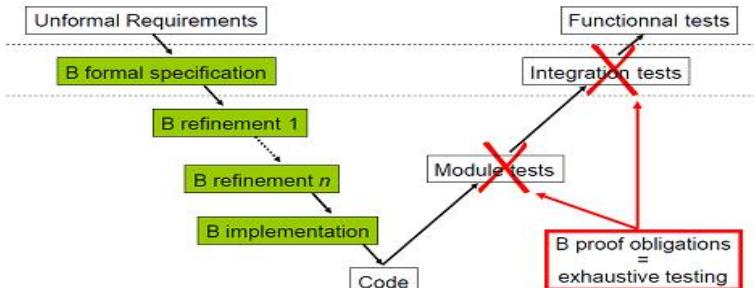
La particularité de B par rapport aux autres méthodes formelles c'est qu 'elle couvre tout le cycle de vie du logiciel à développer dans un cadre formel uniforme. L'apport de la preuve dans B présente l'avantage que le logiciel produit respecte la spécification puisque il en découle totalement .Elle a toutefois l'inconvénient de ne pas fournir de guide de réalisation aussi précis et mûr que certaines méthodes* du marché et c'est dans ce but que les études de cas peuvent être d'un grand intérêt dans l'utilisation de la méthode.

Méthode B : principe

- ▶ Partir d'une spécification de *haut niveau*
 - ▶ Spécification *descriptive*
- ▶ Ajouter des détails jusqu'à arriver à un programme exécutable
 - ▶ Préciser l'*algorithme*
 - ▶ Préciser les *structures de données*

▶ Avantages

- ▶ Substituer des preuves aux tests
 - ▶ *Exhaustivité !*
 - ▶ *Coûts réduits !*
- ▶ Logiciel *correct par construction*
- ▶ *Maintenance* facilitée



Fondements mathématiques :

Cadre homogène pour tout le cycle de développement (analyse, conception et réalisation)

Approche :

Raffiner le modèle initial d'une machine abstraite (sa spécification) en un module exécutable (son code).

Validation basée sur des preuves :

- Preuve de la spécification de chaque opération.
- Preuve du raffinement d'une machine en une autre.

Langage de modélisation :

- Abstraction du système
- Changement d'états du système
- Spécification des invariants du système (à vérifier)

<pre> MACHINE réservation(max_siège) VARIABLES siège INVARIANT siège ∈ 0..max_siège INITIALISATION siège:=max_siège OPERATIONS réserver ≡ PRE siège > 0 THEN siège := siège-1 END; annuler ≡ PRE siège < max_siège THEN siège := siège+1 END; END </pre>	<pre> MACHINE MA-PERSONNE SETS PERSONNES , SEXE={M,F} VARIABLES Personnes , Sexe ,Nom INVARIANT Personnes ⊆ PERSONNES ∧ Sexe ∈ Personnes → SEXE ∧ Nom ∈ Personnes → STRING OPERATIONS Ajout-personne(p,sex,p) Pre p ∈ PERSONNES – Personnes ∧ sex ∈ SEXE ∧ nom ∈ STRING Then Personnes := Personnes ∪ {p} Sexe(p) := sexe Nom(p) := nom End ; Supp_personne(p) Pre p ∈ Personnes Then Personnes := Personnes – {p} Sexe := {p} ≤ Sexe_ Nom := {p} ≤ Nom end END </pre>
---	--

Exemple de machines abstraites