II
SPÉCIFICATION ET VÉRIFICATION FORMELLE DE LOGICIELS

MODÈLES DE DÉVELOPPEMENT LOGICIEL

- Le modèle du cycle de développement est un des concepts les plus importants dans l'ingénierie moderne du logiciel. Sa prémisse est que le développement et l'implantation d'un système sont réalisés en plusieurs phases distinctes, chacune ayant une tâche bien définie.
- Plusieurs modèles ont été proposés: le modèle en cascade (Waterfall), le modèle en spirale, le modèle en V, le modèle en W, etc.
- Une des motivations de ces modèles est de minimiser le nombre des fautes résiduelles dans les premières phases, car il est moins coûteux de corriger une faute le plus tôt possible.

MODÈLES DE DÉVELOPPEMENT LOGICIEL

- **Par exemple**, il est souvent facile de corriger une faute de programmation durant les tests unitaires et il est également facile d'insérer une exigence durant la revue de la spécification.
- Par contre, il est très coûteux d'insérer ou de corriger une telle exigence si elle est détectée après la phase de codage ou durant la phase de test d'intégration

MODÈLES DE DÉVELOPPEMENT LOGICIEL

- **Analyse** : comprendre le problème, les besoins
- **Conception** : décomposer en sous-problèmes, trouver une architecture pour résoudre le problème
- **Réalisation** : développer les différents morceaux
- **Intégration** : faire marcher ensemble les différents morceaux
- **Validation** : s'assurer qu'on a bien répondu au problème

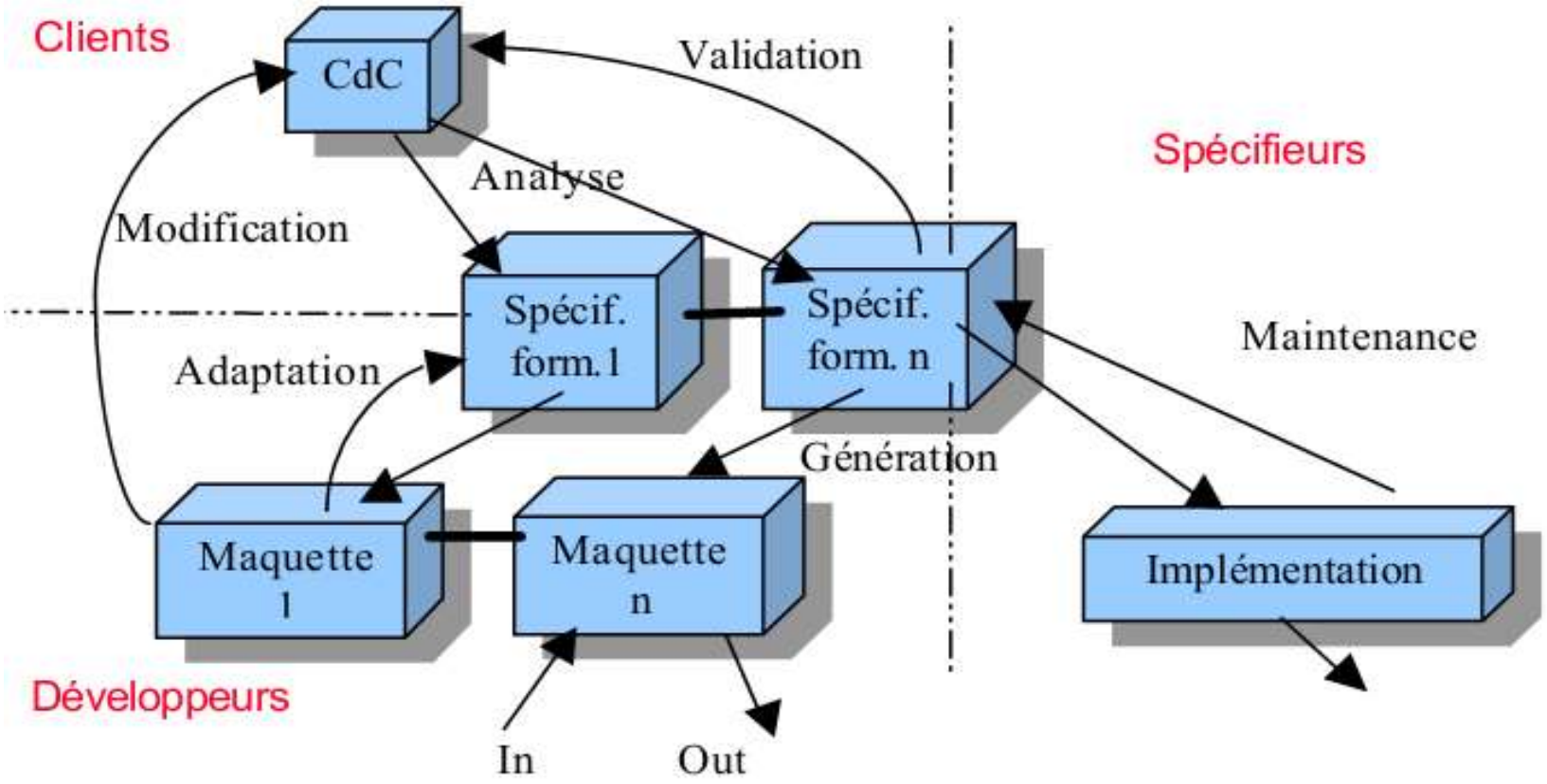
MODÈLES DE DÉVELOPPEMENT LOGICIEL

- Chaque phase produit son lot de spécifications c'est-à-dire un document qui décrit tout ou partie de l'application, à un niveau d'abstraction donné, dans un formalisme qui peut être plus ou moins formel.
- Une spécification est un ensemble explicite d'exigences que doit satisfaire le programme/système.

CYCLE DE VIE DE BALZER

- Avec les méthodes formelles les cycles de vie classique ne marchent plus. Il faut les adapter. C'est du cycle de vie de Balzer
- Le cycle de vie de Balzer représente la nouvelle famille de cycles de vie adaptés au développement formel (ou rigoureux !).
- Le modèle de Balzer associe développement incrémental et utilisation de spécifications formalisées, elles même développées de manière incrémentale et maintenues.

CYCLE DE VIE DE BALZER



SPÉCIFICATION

- 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.

SPÉCIFICATION

- 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.
- La spécification peut être :
  - Implicite : donnée par le langage (e.g., pas de division par zéro) (on parle de propriété non-fonctionnelle)
  - Explicite : donnée pour chaque programme (e.g., la fonction de tri renvoie une liste triée) (on parle de propriété fonctionnelle)
  - plus ou moins abstraite : indique le résultat d'une fonction, pas comment il est calculé
  - informelle (e.g., document de spécification en langue naturelle)
  - encodée dans le programme (e.g., assertions, contrats)
  - une formule mathématique (pour les méthodes formelles)

SPÉCIFICATION INFORMELLE

- On parle de spécification informelle lorsque le langage utilisé n'a pas une syntaxe précise
- Certains 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.
- 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.

SPÉCIFICATION FORMELLE

- La spécification formelle est définie en génie logiciel ISO/AFNOR comme « une spécification pouvant être utilisée afin de démontrer mathématiquement la validité de la mise en œuvre d'un système ou encore de dériver mathématiquement la mise en œuvre du système » ;
- C'est une spécification écrite en notation formelle, souvent utilisée pour une démonstration d'exactitude.
- Selon Gaudel, Marre, Bernot et Schlienger , une spécification est dite formelle si:
  - « elle est écrite en suivant une syntaxe bien définie, comme celle d'un langage de programmation » ;
  - « la syntaxe est accompagnée d'une sémantique rigoureuse qui définit des modèles mathématiques représentant les réalisations acceptables de chaque spécification syntaxiquement correcte ».

SPÉCIFICATION FORMELLE

Selon Choppy, on parle de :

- spécification informelle lorsque le langage utilisé n'a pas une syntaxe précise,
- spécification semi-formelle lorsque la syntaxe du langage utilisé est définie de façon précise, et
- spécification formelle lorsque la syntaxe et la sémantique du langage utilisé sont définies .

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.

SPÉCIFICATION FORMELLE

- L'addition de méthodes formelles au cycle de développement améliore le processus de développement et la qualité du logiciel, en imposant un contrôle dès les premières phases du développement, notamment la spécification.
- Les méthodes formelles ne sont pas utiles si elles ne sont pas compréhensibles et faciles à utiliser.
- Différentes notations sont utilisées sous différentes formes : tabulaire, graphique, mathématique, etc.
- Différents outils ont été développés (des éditeurs, des animateurs et des vérificateurs) pour manipuler ces notations.

EXERCICE

La spécification de la règle de notation à un examen est la suivante : « L'examen est un ensemble de 20 questions à réponses multiples. Chaque bonne réponse à une question rapporte 1 point. Chaque mauvaise réponse fait perdre 1/3 de point. Chaque question sans réponse donne 0 point. »

Pensez vous que cette spécification est claire ?

Pour le vérifier, calculez les notes des 3 étudiants suivants :

	Réponses correctes	**incorrectes**	**sans**	**doubles réponses**
Toto	10	5	5	
Tata	4	16		
Titi	10	3	4	3 (1 juste, 1 fausse)

Recensez les résultats possibles.

Donnez une spécification plus précise.

VALIDATION ET VÉRIFICATION

- **Validation** : Le processus d'évaluation du logiciel au cours ou à la fin du processus de développement fin de déterminer s'il satisfait les exigences spécifiées. C'est à dire, « ***construisons-nous le système correct ?*** »
- **Vérification** : le processus d'évaluation du logiciel pour déterminer si le produits d'une phase de développement donnée satisfont les condition imposées au début de cette phase. C'est à dire, «***construisons-nous le système correctement ?*** »

VALIDATION ET VÉRIFICATION

- **Validation** (validation) : Le processus d'évaluation du logiciel au cours ou à la fin du processus de développement fin de déterminer s'il satisfait les exigences spécifiées. C'est à dire, « ***construisons-nous le système correct ?*** »
- La validation peut intervenir à une étape intermédiaire ou finale
- La validation, qu'il s'agisse d'une étape intermédiaire ou finale, revient à « s'assurer qu'on a construit le bon produit ».
- Les critères appliqués sont donc externes au produit lui-même.

VALIDATION ET VÉRIFICATION

- Valider un produit final revient à s'assurer qu'il est conforme à son cahier des charges et qu'il donne satisfaction à ses utilisateurs.
- Pour qu'une conception soit valide, il faut qu'elle prenne en compte tous les éléments énoncés lors de l'analyse.
- **Vérification** (verification) : le processus d'évaluation du logiciel pour déterminer si le produits d'une phase de développement donnée satisfont les condition imposées au début de cette phase. C'est à dire, «***construisons-nous le système correctement ?*** »

VALIDATION ET VÉRIFICATION

- La vérification d'un produit « consiste à s'assurer que le produit est bien construit », ou autrement dit, que sa construction satisfait les exigences de qualité spécifiées.
- La vérification d'un document revient à s'assurer que les règles de structuration et de présentation sont respectées, que le document est complet, non ambigu et cohérent, par exemple.
- Pour un programme, on peut vérifier que les règles de style ont été respectées et que les tests qu'il a subis sont suffisants.
- La différence entre validation et vérification n'est en pratique pas aussi nette. On les utilise donc souvent de façon interchangeable, donnant un sens plutôt plus général à vérification, et réservant parfois le terme validation à la validation du produit final.

VALIDATION ET VÉRIFICATION

- En informatique, la vérification de modèles consiste à vérifier si le modèle d'un système satisfait une propriété.
- Elle consiste à vérifier qu'un programme obéit à sa spécification.
- La spécification indique ce que le programme doit et ne doit pas faire.
- Par exemple, on souhaite vérifier qu'un programme ne bloque pas, qu'une variable n'est jamais nulle, etc.
- Généralement, la propriété est écrite dans un langage formel. La vérification est généralement faite de manière automatique.

MÉTHODES TRADITIONNELLES DE VÉRIFICATION

Afin de s'assurer de l'absence d'erreur, les méthodes traditionnelles de l'industrie sont :

- des règles de programmation strictes, par exemple
  - MISRA C (système embarqués)
  - interdiction de récursion et d'allocation dynamique (systèmes critiques)
- la relecture de code par les pairs
  - technique statique : inspection manuelle du code, pas d'exécution du logiciel.
  - détecte entre 31% et 93% de défauts avec une médiane de 60% environ.
  - les erreurs subtiles (défauts de concurrence et dans l'algorithme) sont dicibles à découvrir.

MÉTHODES TRADITIONNELLES DE VÉRIFICATION

- **le test**
  - test unitaire
  - Test d'intégration
  - technique dynamique dans lequel le logiciel est exécuté.
  - de 30% à 50% du coût des projets des logiciels est dédié au test.
  - plus de temps et d'efforts sont consacrés à la validation qu'au codage.
  - densité de défauts acceptée : environ 1 défaut pour 1.000 lignes de code

METHODES TRADITIONNELLES DE VÉRIFICATION

- vérification dynamique (exécution du code)
  - le code peut être instrumenté avec des tests (e.g., valgrind)

Il s'agit souvent d'assurance basée processus. Le code est de bonne qualité parce qu'un ensemble de règles bien définies a été suivi.

LIMITES DU TEST

Exemples de couverture :

- **couverture d'instruction :**
  tester cond1 && cond2 et !cond1 &&!cond2
  (ne trouve pas l'erreur)
- **couverture de chemin :**
  tester tous les cas de cond1 et cond2
- **couverture totale :**
  - tester toutes les valeurs de N, cond1 et cond2
  - La couverture totale est impossible en pratique
- → **le test ne trouve pas toutes les erreurs !**

```
int a[N], x = 0;
if (cond1) x++;
else       x--;
if (cond2) x--;
else       x++;
a[x];
```

LIMITES DU TEST

- Augmenter la couverture coûte cher.
- Dans l'industrie critique, le test d'un logiciel atteint déjà jusqu'à 50% de son coût total de production.
- Enfin, certaines propriétés ne peuvent pas être vérifiées par le test.

Exemple : la terminaison !

TECHNIQUES DE VÉRIFICATION FORMELLE POUR LA PROPRIÉTÉ P

- Méthodes déductives
  - méthode : fournir une preuve formelle que P est vrai.
  - outil : démonstrateur des théorèmes / assistant de preuve ou proof-checker (Coq, Isabelle, Pandora)
  - applicable si : le système se présente sous forme d'une théorie mathématique (par exemple, tableaux)
- Model-checking
  - méthode : contrôle systématique de P dans tous les états
  - outil : model checker (Spin, NuSMV, UppAal, . . .)
  - applicable si : le système généré un modèle fini de comportements
- Simulation ou test basé sur des modèles
  - méthode : tester P en explorant les comportements possibles.
  - applicable si : le système définit un modèle exécutable.

SIMULATION ET TEST

- Procédure de base :
  - prendre un modèle (simulation) ou une réalisation (test)
  - introduire certains inputs (à savoir, des tests)
  - observer la réaction et vérifier si cela est « désiré"
- Inconvénients importants :
  - le nombre de comportements possibles est très grand (voire infini)
  - les comportements inexplorées peuvent contenir le bug fatal
- A propos du testing . . .
  - Le test / simulation peut montrer la présence d'erreurs, pas leur absence

ETAPES DE LA VÉRIFICATION FORMELLE

- Exactitude des programmes mathématiques (Turing, 1949)
  - Technique basée sur la syntaxe pour les programmes séquentiels (Hoare, 1969)
  - pour une donnée d'entrée, est-ce que un programme d'ordinateur génère la sortie correcte ?
  - fondée sur des règles de preuve compositionnelles, exprimées dans la logique des prédicats.
- Technique basée sur la syntaxe pour les programmes concurrents
  - elle manipule des propriétés concernant les états pendant la computation
  - fondée sur des règles de preuve exprimées en logique temporelle
- Vérification automatique de programmes concurrents
  - approche basée sur les modelés au lieu des règles de preuve
  - est-ce que le programme concurrent satisfait une propriété (logique) donnée ?

MÉTHODES FORMELLES DE VÉRIFICATION

- **Principe** : Utiliser des outils de la logique pour raisonner sur les programmes avec des garanties mathématiques, des théorèmes de correction →aucune ambiguïté sur ce qui est vérifiée.
- Technique fortement liée au domaine de la sémantique i.e., donner un sens mathématique aux programmes.
- Développées dans les années 60, plusieurs variétés de méthodes existent maintenant :
  - logique de Hoare, preuve de programmes ;
  - calculs de préconditions, méthodes déductives ;
  - vérification de modelés (model-checking) ;
  - analyse statique par interprétation abstraite.

MÉTHODES FORMELLES DE VÉRIFICATION

- Une méthode de vérification ne peut être utile que si elle est (au moins en partie) automatisée.
- Progrès considérables des outils :
  - assistants de preuve (Coq, Isabelle, etc.)
    =)automatise la vérification des preuves (pas leur rédaction)
  - solveurs SAT (formules booléennes)
  - solveurs SMT (arithmétiques, certaines théories logiques simples)
  - analyseurs statiques, naturellement automatiques

OUTILS DE SPÉCIFICATION : EXEMPLES

- La spécification **ANSI / ISO C Langue (ACSL**) est un langage de spécification pour les programmes C, en utilisant le style Hoare avant et postconditions et invariants, qui suit la conception par contrat paradigme. Les spécifications sont écrites sous forme de commentaires d'annotation C au programme C, qui peut donc être compilé avec un compilateur C. L'outil de vérification en cours pour ACSL est Frama-C.
- Le **Java Modeling Language (JML)** est un langage de spécification pour Java programmes, en utilisant le style Hoare pré et postconditions et invariants , qui suit la conception par contrat paradigme. Les spécifications sont écrites comme annotation Java commentaires aux fichiers source, qui peut donc être compilé avec tout Java compilateur . Il comprend divers outils de vérification, comme un vérificateur d'assertion d'exécution et le vérificateur statique étendu (ESC/Java).
- Etc.

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 (pour le moins !)
  - Spécification algébrique, Machines d'état abstraites, Interprétation abstraite, *Model Checking*, Systèmes de types, Démonstrateurs de théorèmes automatiques ou interactifs, …

COMMENT LES APPLIQUER ?

- Quel est votre *problème* ? Qu'est-ce que vous voulez *garantir* ?
  - Trouver une méthode formelle qui corresponde
    - À votre domaine d'application
    - À vos problèmes
    - À vos contraintes de temps et de coûts
- Comment les mettre en œuvre ?
  - Les *intégrer* à votre *cycle de développement*
    - Retour à moyen/long terme
    - Par ex. : mettre à jour la spécification formelle quand la spécification ou le système réalisé changent
  - Prendre les *conseils d'un expert* dans la méthode choisie

FIN