Polytech Paris-Sud
Formation initiale 4e année
Spécialité informatique
Année 2016-2017

Vérification et validation

Cours 1

Introduction

Delphine Longuet
delphine.longuet@lri.fr

Organisation du cours

Modalités de contrôle des connaissances :

- 2 TP notés : 29 mars et 25 avril
- Contrôle (2h) : 2 mai

Note finale = 40% TP + 60% contrôle

Seuls les transparents du cours sont autorisés au contrôle

Page web du cours :

http://www.lri.fr/~longuet/Enseignements/16-17/Et4-VV

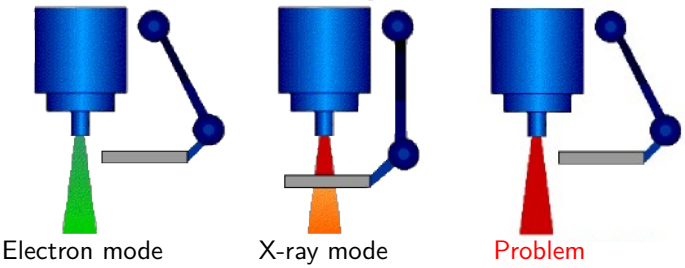
Bugs célèbres

Sonde Mariner 1, 1962

- Détruite 5 minutes après son lancement; coût : $18,5 millions
- Défaillance des commandes de guidage due à une erreur de spécification
- Erreur de transcription manuelle d'un symbole mathématique dans la spécification

Therac-25, 1985-87

- Au moins 5 morts par dose massive de radiations
- Problème d'accès concurrents dans le contrôleur



Processeur Pentium, 1994

- Bug dans la table de valeurs utilisée par l'algorithme de division

Ariane V vol 501, 1996

- Explosion après 40 secondes de vol; coût : $370 millions
- Panne du système de navigation due à un dépassement de capacité (arithmetic overflow)
- Réutilisation d'un composant d'Ariane IV non re-testé

Plus récemment

PlayStation Network, avril 2011

- Des millions de données personnelles et bancaires piratées
- Pertes financières de plusieurs milliards de dollars
- Vulnérabilité du réseau connue mais conséquences mal évaluées ?

Outil de chiffrement OpenSSL, mars 2014

- 500 000 serveurs web concernés par la faille
- Vulnérabilité permettant de lire une portion de la mémoire d'un serveur distant

Bug ?

Erreur : comportement du programmeur conduisant à la faute

Défaut ou faute : élément ou absence d'élément dans le logiciel entraînant une anomalie

Défaillance ou anomalie : comportement inattendu du logiciel, mais il fonctionne toujours

Panne : arrêt total du logiciel ou arrêt partiel entraînant un fonctionnement en mode dégradé

Erreur ⟶ Défaut ⟶ Défaillance ou Panne

Défaut vs. défaillance

Défaut $\not\Rightarrow$ défaillance

Comportement apparent correct du logiciel malgré un grand nombre de défauts, car défauts jamais exercés en fonctionnement

Problème avec réutilisation du code ou nouveaux cas d'utilisation : nouveau contexte d'exécution donc nouveaux défauts exposés

Défaut vs. panne

Défaillance $\not\Rightarrow$ panne

Défaillance : résultats inattendus, mauvais placement d'une fenêtre, message non affiché, mauvais enregistrement des données...

↳ Système toujours opérationnel

Panne : service plus assuré

↳ Nécessite un redémarrage du système, conséquences potentiellement très graves

Raisons de la faible qualité des logiciels

Tâche complexe :

- Taille et complexité des logiciels
- Taille des équipes de conception/développement

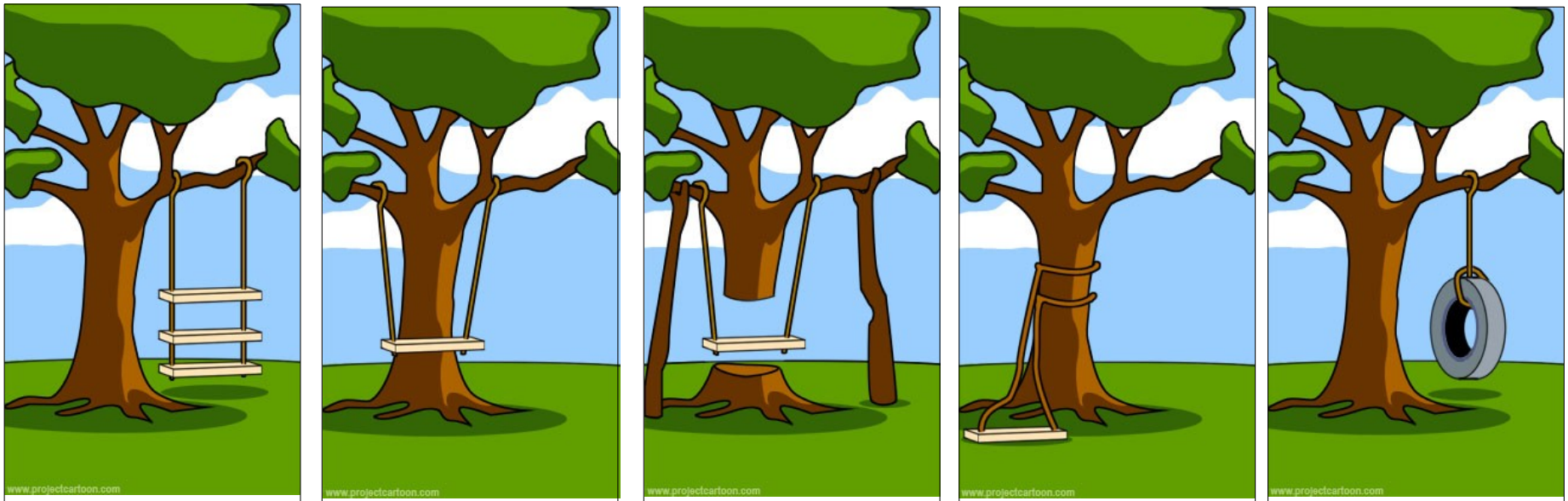
Manque de méthodes et de rigueur :

- Manque de méthodes de conception
- Négligence et manque de méthodes et d'outils des phases de validation/vérification

Mauvaise compréhension des besoins :

- Négligence de la phase d'analyse des besoins du client
- Manque d'implication du client dans le processus

Raisons de la faible qualité des logiciels



Raisons de la faible qualité des logiciels

Difficultés spécifiques du logiciel :

- Produit invisible et immatériel
- Difficile de mesurer la qualité
- Conséquences critiques causées par modifications infimes
- Mises à jour et maintenance dues à l'évolution rapide de la technologie
- Difficile de raisonner sur des programmes

Importance de la qualité des logiciels

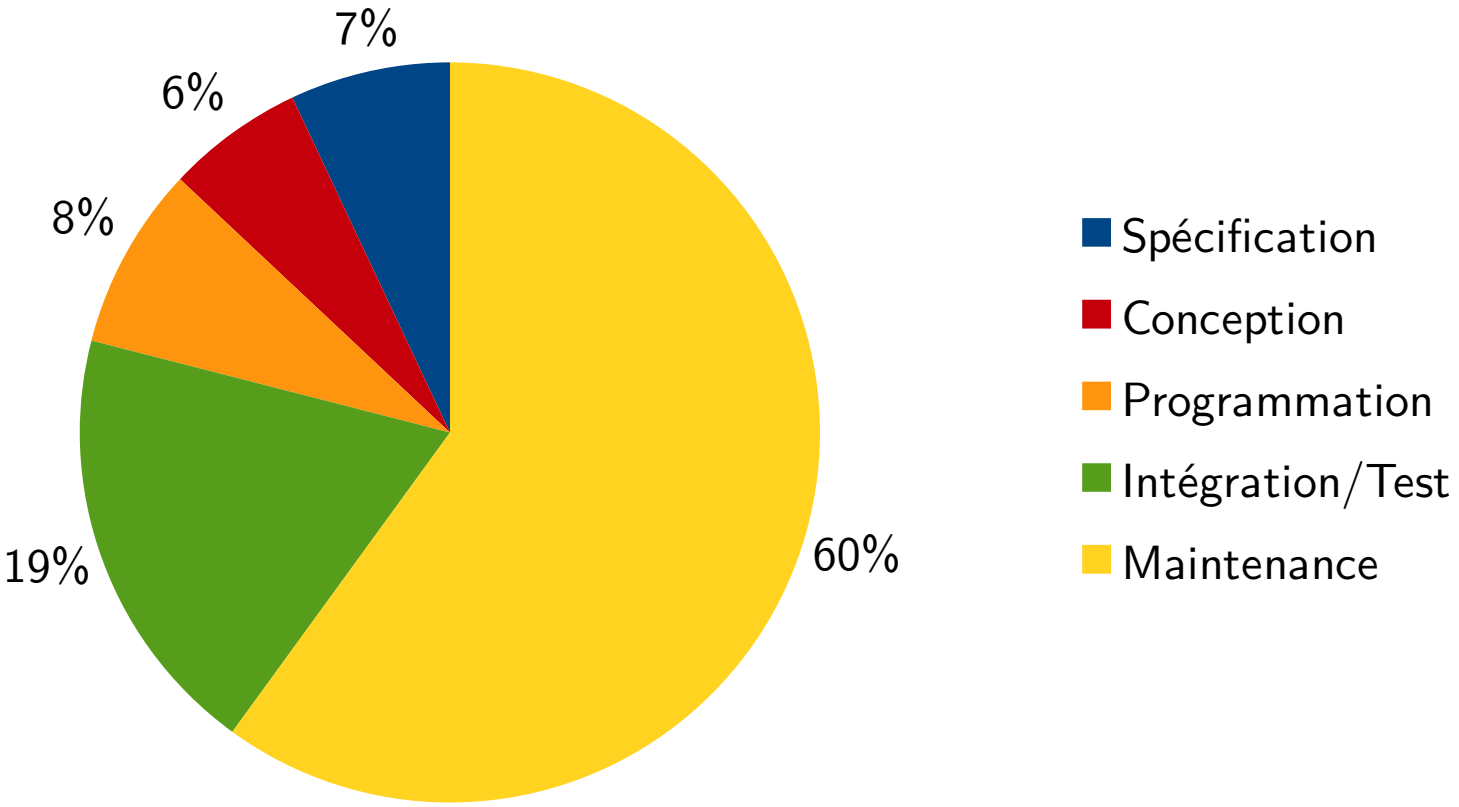
Fiabilité, sûreté et sécurité des logiciels

- Transports automobile, ferroviaire, aéronautique
- Contrôle de processus industriels, nucléaire, armement
- Médical : imagerie, appareillage, télé-surveillance
- e-commerce, carte bancaire sans contact, passeport électronique

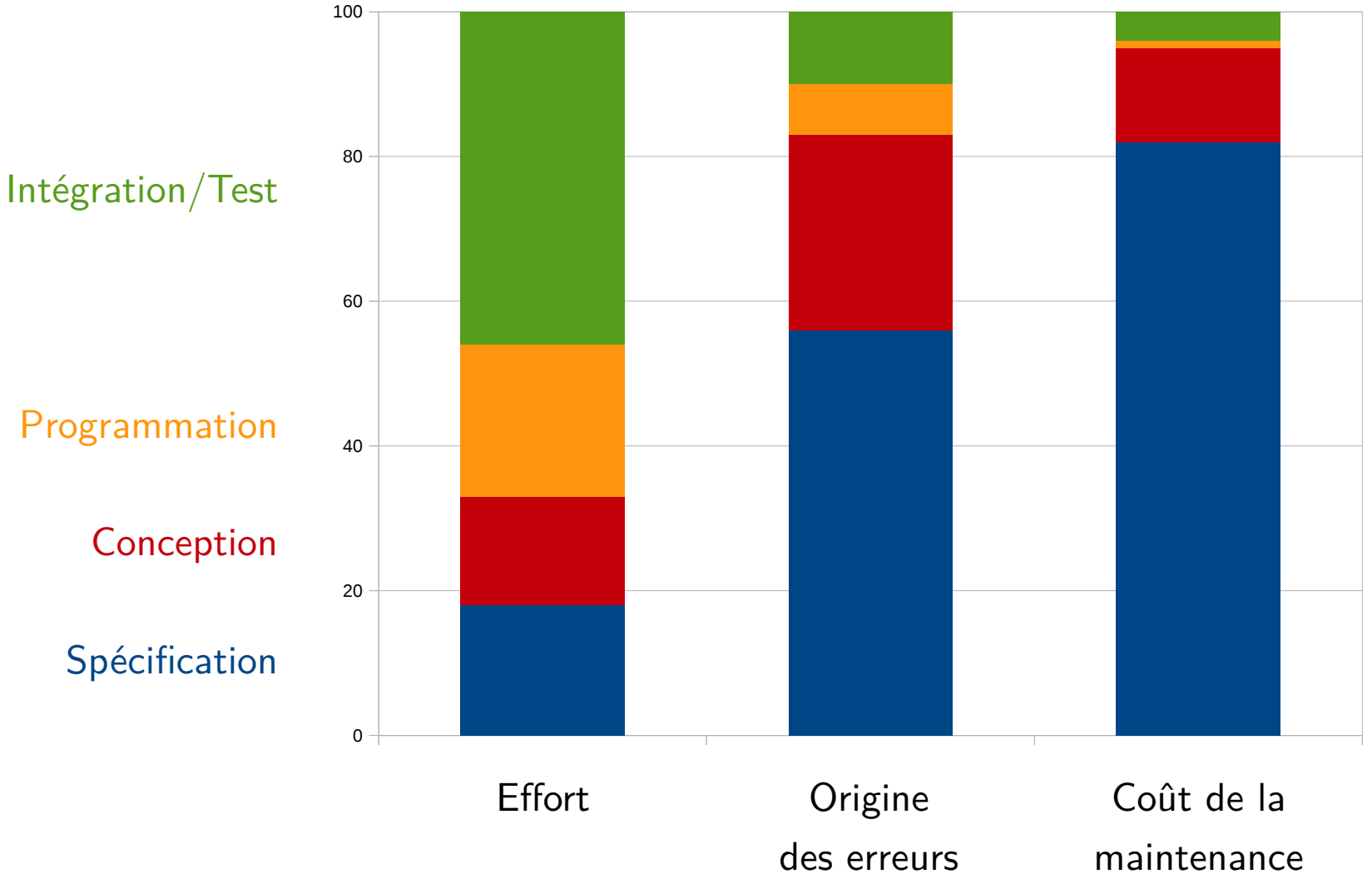
Raisons économiques : coût d'un bug

- coût de la correction, du rappel des appareils défectueux
- coût de l'impact sur l'image, de l'arrivée tardive sur le marché
- coût en vies, coût de l'impact écologique

Répartition de l'effort



Rapport effort/erreur/coût



Pourquoi des méthodes pour valider et vérifier ?

Pour réduire le coût

- Vérification et validation =
  - environ 30% du développement d'un logiciel standard
  - plus de 50% du développement d'un logiciel critique
- Phase de test souvent plus longue que les phases de spécification, conception et implantation réunies

Pour certifier la qualité

- Processus de certification, de normalisation
- Obligations légales

Validation et vérification

Validation : assurer que les attentes du client sont satisfaites

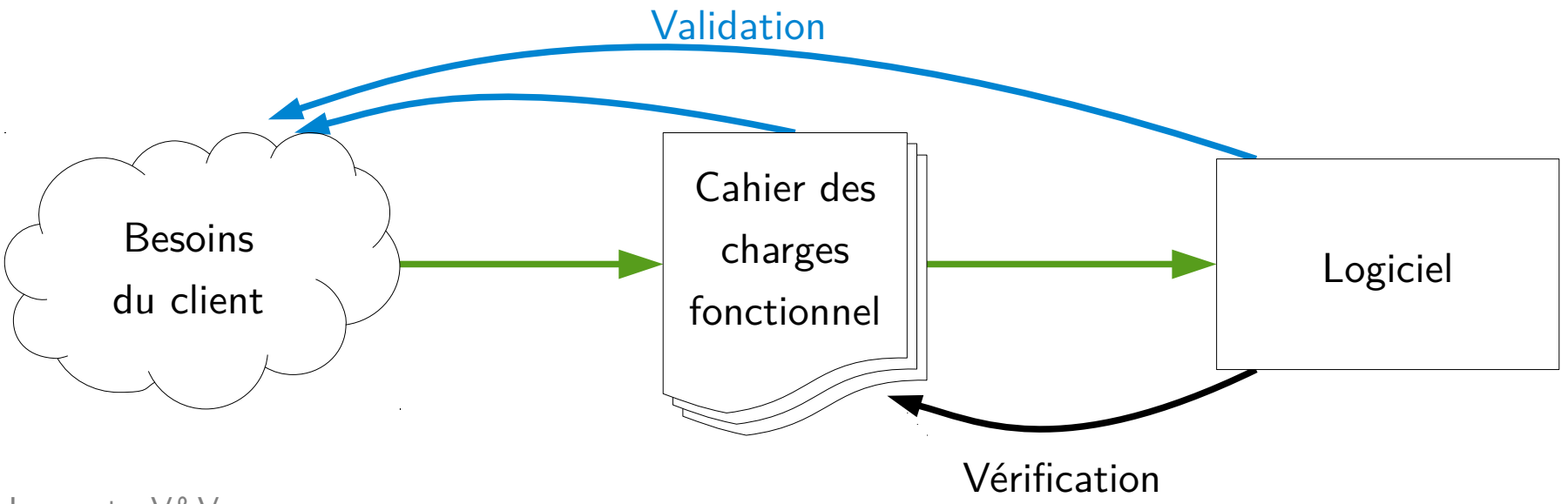
« Are we building the right product ? »

Assurance d'un certain niveau de confiance dans le système

Vérification : assurer que le système satisfait sa spécification

« Are we building the product right ? »

Preuve formelle de propriétés sur un modèle du système



Méthodes de validation et vérification

Méthodes de validation :

- Revue de spécification, de code
- Prototypage rapide
- Développement de tests exécutables (*extreme programming*)
- Recette (tests d'acceptation faits par le client)

Méthodes de vérification :

- Test formel (à partir de la spécification)
- Preuve de programmes
- Model-checking

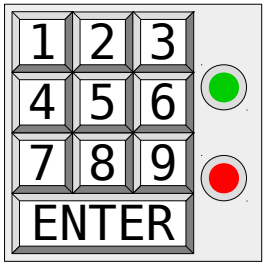
Quelques méthodes de V&V

Test : exécution d'un programme sur des données choisies dans le but de détecter des non-conformités par rapport à la spécification (cahier des charges fonctionnel)

Model-checking : analyse d'un modèle du programme dans le but de prouver mathématiquement qu'il vérifie certaines propriétés dynamiques

Preuve de programmes : preuve mathématique qu'un programme satisfait sa spécification en termes de pré et post-conditions

Exemple : test

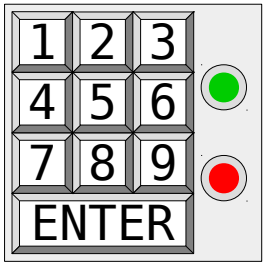


Code d'accès :

- 4 chiffres consécutifs dans l'ordre croissant
- Saisie des chiffres puis validation.
- Accès autorisé s'il existe une suite de 4 chiffres consécutifs dans la suite saisie.

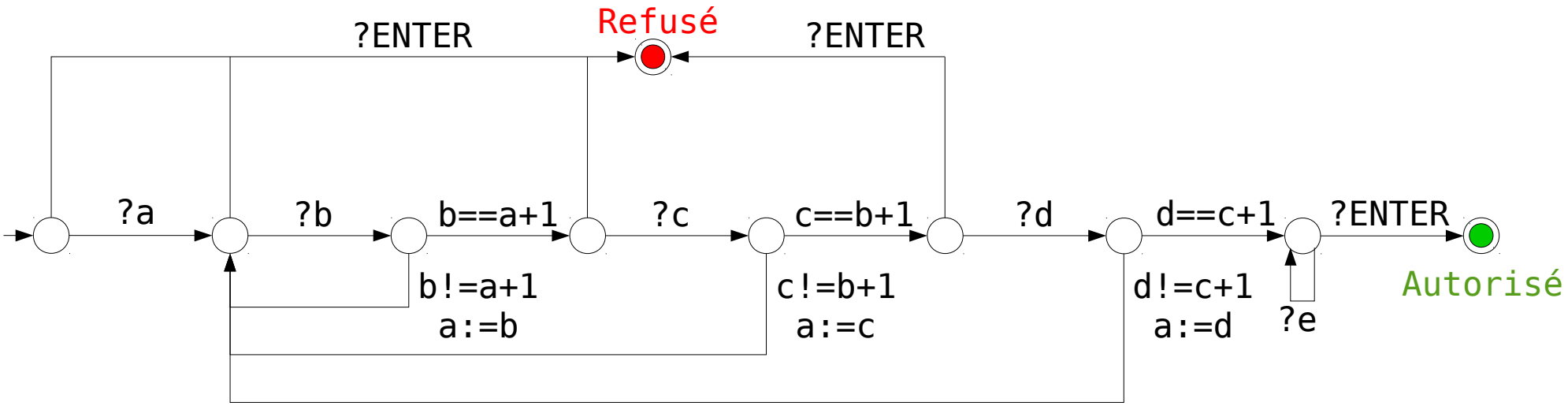
Objectif de test	Données d'entrée	Résultat attendu
4 chiffres consécutifs	1234 + ENTER	Autorisé
Avec des chiffres avant	846789 + ENTER	Autorisé
Avec des chiffres après	3456298 + ENTER	Autorisé
Avec des chiffres avant et après	32345698 + ENTER	Autorisé
Dans l'ordre décroissant	6543 + ENTER	Refusé
Pas consécutifs	2356 + ENTER	Refusé
Moins de 4 chiffres	345 + ENTER	Refusé
Vide	ENTER	Refusé
Oubli de ENTER	5678	Aucun

Exemple : model-checking



Code d'accès :

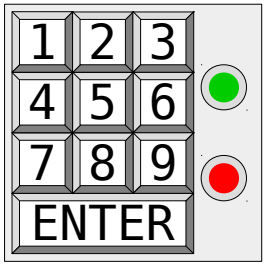
- 4 chiffres consécutifs dans l'ordre croissant
- Saisie des chiffres puis validation.
- Accès autorisé s'il existe une suite de 4 chiffres consécutifs dans la suite saisie.



Propriété à démontrer : `Autorisé ⇒ a!=0 ∧ b=a+1 ∧ c=a+2 ∧ d=a+3 ∧ ENTER`

(Variables : `a, b, c, d, e = 0` ; `ENTER = false`)

Exemple : preuve



Code d'accès :

- 4 chiffres consécutifs dans l'ordre croissant
- Saisie des chiffres puis validation.
- Accès autorisé s'il existe une suite de 4 chiffres consécutifs dans la suite saisie.

```
/*@ requires \valid(t+(0..n-1)) && 4 <= n;
  @ ensures \result <==> \exists integer k; 0 <= k < n
                         && \forall integer j; 0 < j <= 3 ==> t[k+j] == t[k]+j;
  @*/

int digicode(int t[], int n) {
    int res = 0;
    int i = 1;
    /*@ loop invariant \forall integer j; 0 < j <= res ==> t[i-1] == t[i-1-j]+j;
      @ loop invariant \forall integer k; 0 <= k < i-res-1
                        ==> \exists integer j; 0 < j <= 3 && t[k+j] != t[k] + j;
      @ loop invariant 0 <= res <= 3 && 1 <= i <= n;
      @*/
    while(res < 3 && i < n) {
        if(t[i] == t[i-1]+1) { res = res+1; i = i+1; }
        else { res = 0; i = i+1; }
    }
    return(res == 3);
}
```

Comparaison des méthodes de V&V

Test :

- ✔ Nécessaire : exécution du système réel, découverte d'erreurs à tous les niveaux (spécification, conception, implantation)
- ✗ Pas suffisant : exhaustivité impossible

Model-checking :

- ✔ Exhaustif, partiellement automatique
- ✗ Mise en œuvre moyennement difficile (modèles formels, logique)

Preuve :

- ✔ Exhaustif
- ✗ Mise en œuvre difficile, limitation de taille

Comparaison des méthodes de V&V

Méthodes complémentaires :

- Test non exhaustif mais facile à mettre en œuvre
- Preuve exhaustive mais très technique
- Model-checking exhaustif et moins technique que la preuve

Mais :

- Preuve et model-checking limités par la taille du système et vérifient des propriétés sur un modèle du système (distance entre le modèle et le système réel ?)
- Test repose sur l'exécution du système réel, quelles que soient sa taille et sa complexité

Méthodes formelles de V&V

Base : spécification ou modèle formel du système

- Spécification formelle : ensemble de formules logiques qui doivent être vérifiées par le système
- Modèle formel : représentation du comportement du système, description abstraite de sa dynamique

Intérêt : raisonner sur le système avec des outils mathématiques

- Non ambigu, hypothèses explicitées
- Automatisable au moins en partie
- Fiable et reproductible

Langages de spécification formelle

Orientés contrôle : représentation de la dynamique du système

- Automates finis étendus, machines à états UML
- Réseaux de Petri, diagrammes d'activité UML
- Langages synchrones (Lustre)

Orientés données : représentation de l'évolution des données

- Méthode B
- OCL pour UML
- JML (pour Java), ACSL (pour C), Spec# (pour C#)

Plan du cours

I. Spécification formelle orientée objet

II. Test de logiciels : test fonctionnel et test structurel

III. Preuve de programmes