Université Nice Sophia Antipolis

Vérification et Validation

- Principes
- Approches statiques
- Approches dynamiques
- Intégration

Contrôler la qualité

- Contrôle de la qualité = Ensemble d'inspections, de revues et de tests ➡ pour trouver des erreurs, des défauts…
- Idées préconçues :
  - La qualité ne peut être évaluée que lorsque le code est disponible
  - La qualité ne peut être uniquement améliorée par la suppression d'erreurs dans le code
- Mais les produits intermédiaires sont *contrôlables*
  - Prototypes / maquettes
  - Documents de spécification, de conception
  - Code
  - Jeux de tests

Principes de V&V

- Deux aspects de la notion de qualité :
  - Conformité avec la définition : VALIDATION
    - Réponse à la question : faisons-nous le bon produit ?
    - Contrôle en cours de réalisation, le plus souvent avec le client
    - **Défauts** par rapport aux besoins que le produit doit satisfaire
  - Correction d'une phase ou de l'ensemble : VERIFICATION
    - Réponse à la question : faisons-nous le produit correctement ?
    - Tests
    - **Erreurs** par rapport aux définitions précises établies lors des phases antérieures de développement

Qualité et cycle de vie

- Les spécifications fonctionnelles définissent **les intentions**
  - Valider la conformité aux besoins
  - Définir le plan qualité
- A chaque vérification, on vérifie la conformité aux spécifications fonctionnelles par rapport aux **intentions**
- Lors de la phase de qualification, on valide le produit
  - par rapport aux besoins
  - par rapport aux performances requises

Terminologies

- Norme IEEE (*Software Engineering Terminology*)
  - Erreur : commise par le développeur, entraîne un défaut
  - Défaut : imperfection dans le logiciel, pouvant amener une panne
  - Panne : comportement anormal d'un logiciel
- Classification des faits techniques (qualification) :
  - Non conformité : Erreur par rapport au cahier des charges
  - Défaut : Erreur car le comportement du logiciel est différent d'un comportement normal dans son contexte
  - Évolution : Demande de changement sans prise de garantie

Problèmes

- Plus de 50 % des erreurs sont découvertes en phase d'exploitation
  - Le coût de réparation croit exponentiellement avec l'avancée dans le cycle de vie
  - ☞Contrôles tout au long du cycle de vie + Qualification
- Problèmes lors des contrôles :
  - prééminence du planning sur la qualité
  - sous-estimation des ressources
    - par les développeurs (activité inutile ?)
    - par les dirigeants (budgets séparés pour développement et maintenance !)

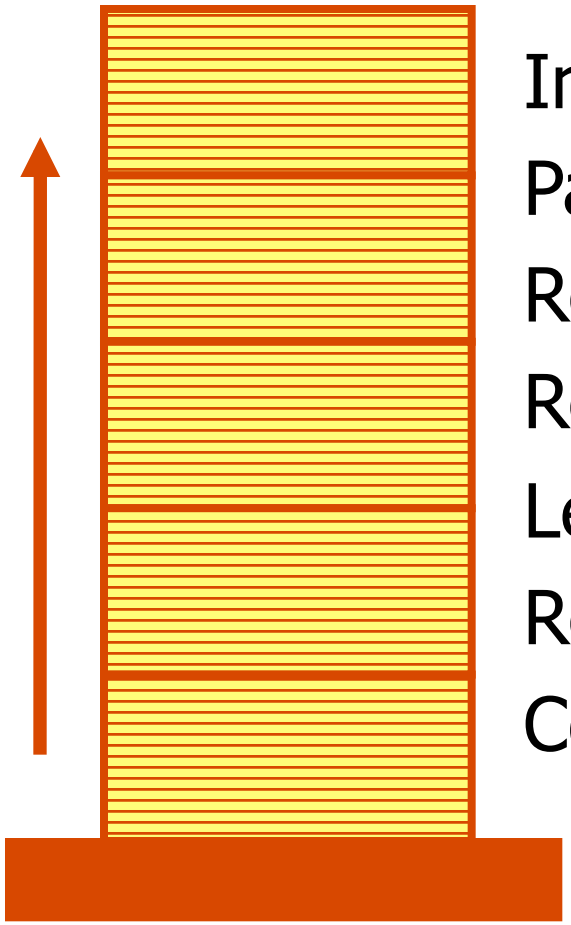
V & V : les moyens

- Statiques :
  - Examen critique des documents : Inspections, revues
  - Analyse statique du code
  - Évaluation symbolique
  - Preuve
- Dynamiques :
  - Exécution du code : Tests
    - Comment les choisir ?
    - Quand arrêter de tester ?

Examen critique de documents

- Minimisation des problèmes d'interprétation
  - Point de vue indépendant du rédacteur
- Vérification
  - Forme : respect des normes, précision, non ambiguïté
  - Fond : cohérence et complétude
  - Testabilité et traçabilité
- Validation
  - Mauvaises interprétations des documents de référence
  - Critères de qualité mal appliqués
  - Hypothèses erronées
- Quelle méthode pour examiner les documents ?
  - Pouvoir de détection
  - Coût
    - 5 à 10p/h Cahier des charges
    - 20 à 50 LOC/h Code

Échelle d'efficacité des méthodes



Relectures et revues

- Relecture individuelle → qualité faible
- Lecture croisée → qualité assez faible
- Revue en groupe structuré
  - Groupe de 10 pers. Max.
  - Lecture puis discussion → qualité moyenne
- Revue structurée
  - Liste séparée de défauts
  - *Check list* des défauts typiques → bonne qualité
- Revue en *Round Robin*
  - lecture préalable
  - attribution de rôles → qualité variable

Parcours et inspection

- Parcours systématique
  - le plus souvent du code
  - audit par des experts (extrêmement coûteux)

- Inspection
  - Préparation : recherche des défauts
  - 🕐 Cycle de réunions
  - 🕐 Suivi : vérification des corrections
  - ☞ Modérateur + secrétaire  meilleure qualité

Ça marche, les inspections ?

- [Fagan 1976] Inspections de la conception et du code
  - 67%-82% de toutes les fautes sont trouvées par des inspections
  - 25% de temps gagné sur les ressources / programmeur (malgré le temps passé dans les inspections)
- [Fagan 1986] nouvelle étude de Fagan
  - 93% de toutes les fautes sont trouvées par inspections
- Réduction de coût pour la détection de fautes (en comparaison avec les tests)
  - [Ackerman, Buchwald, Lewski 1989]: 85%
  - [Fowler 1986]: 90%
  - [Bush 1990]: 25000 $ gagné PAR inspection

Analyse statique du code

- Évaluation du code par des métriques
  - moins cher, mais résultat souvent approximatif
  - qualité des métriques ?
- Recherche d'anomalies dans le code
  - Références aux données (flots, initialisation, utilisation…)
  - Contrôle (graphe de contrôle, code isolé, boucles…)
  - Comparaisons
  - Respect des conventions de style

Méthodes dynamiques : les tests

> “ Testing is the process of executing a program with the intent of finding errors.”
>
> Glen Myers

Tester, c’est exécuter un programme avec l’intention de trouver des erreurs

Tests : définition...

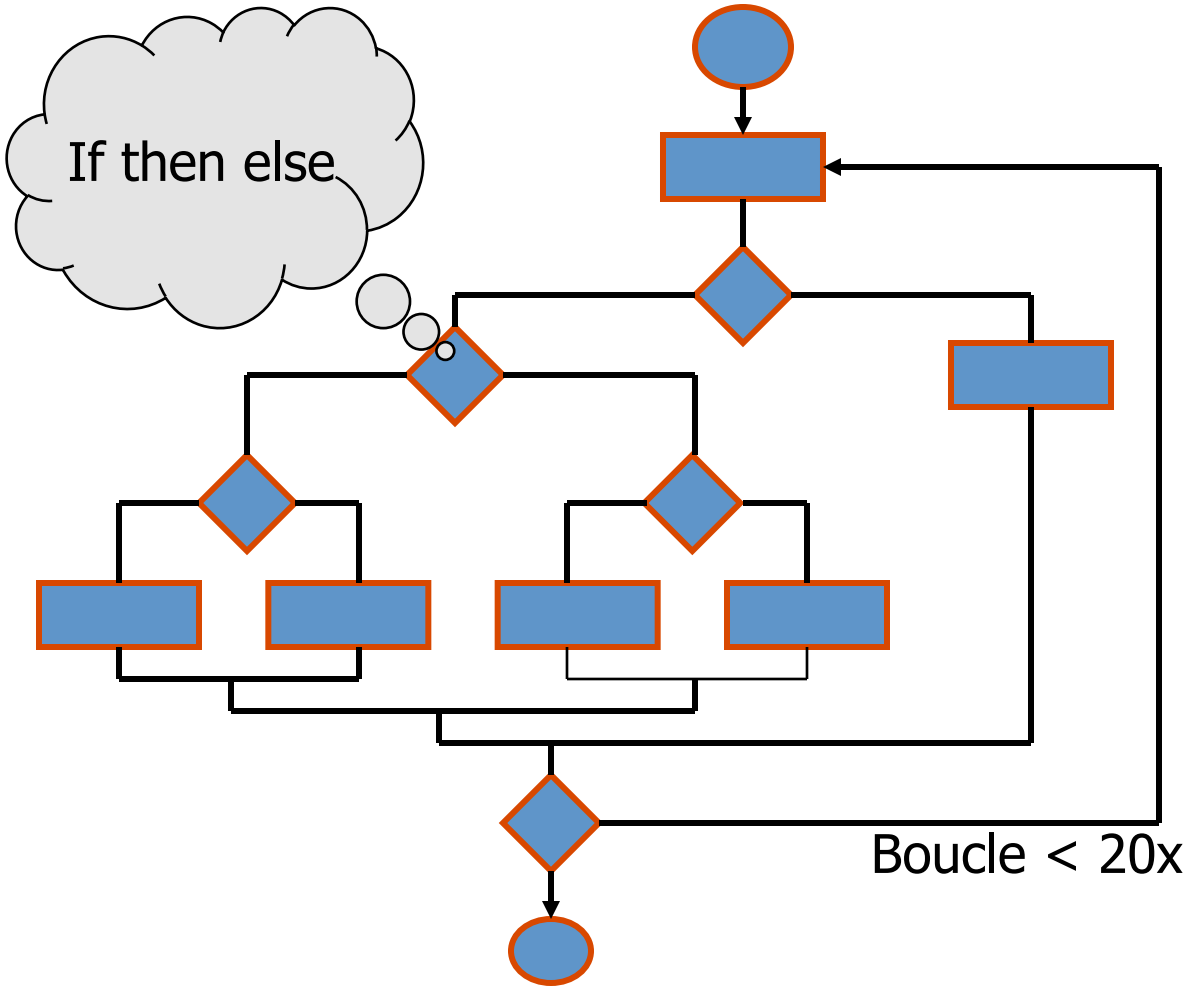
- Une expérience d'exécution, pour mettre en évidence un défaut ou une erreur
  - Diagnostic : quel est le problème
  - Besoin d'un **oracle**, qui indique si le résultat de l'expérience est conforme aux intentions
  - Localisation (si possible) : où est la cause du problème ?

☞ *Les tests doivent mettre en évidence des erreurs !*

☞ *On ne doit pas vouloir démontrer qu'un programme marche à l'aide de tests !*

- Souvent négligé car :
  - les chefs de projet n'investissent pas pour un résultat négatif
  - les développeurs ne considèrent pas les tests comme un processus destructeur

Tests exhaustifs ?



Il y a 5^{20} chemins possibles
En exécutant 1 test par milliseconde, cela prendrait **3024** ans pour tester ce programme !

Constituants d'un test

- Nom, objectif, commentaires, auteur
- Données : jeu de test
- Du code qui appelle des routines : cas de test
- Des oracles (vérifications de propriétés)
- Des traces, des résultats observables
- Un stockage de résultats : étalon
- Un compte-rendu, une synthèse…

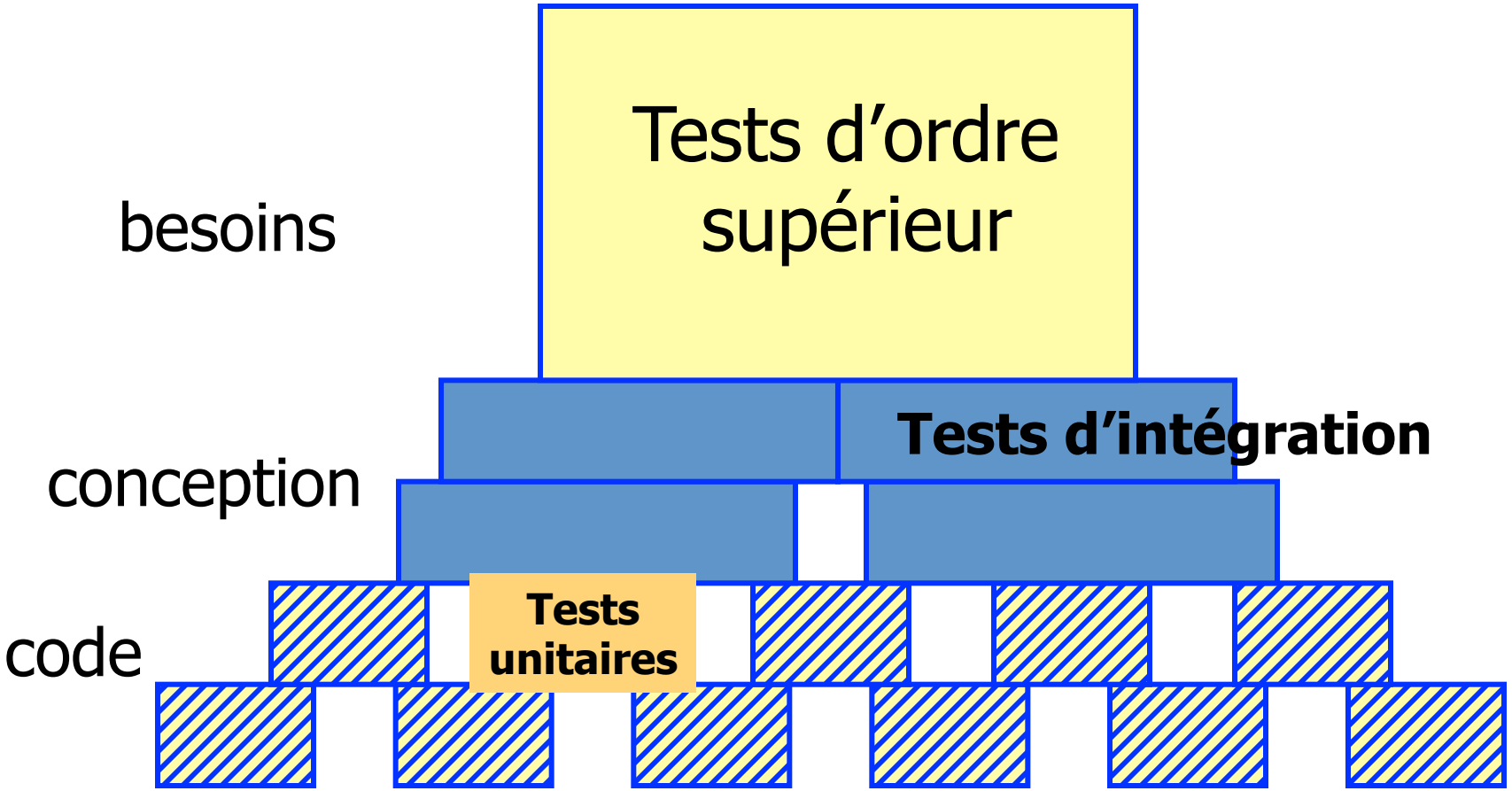
- Coût moyen : autant que le programme

Test vs. Essai vs. Débogage

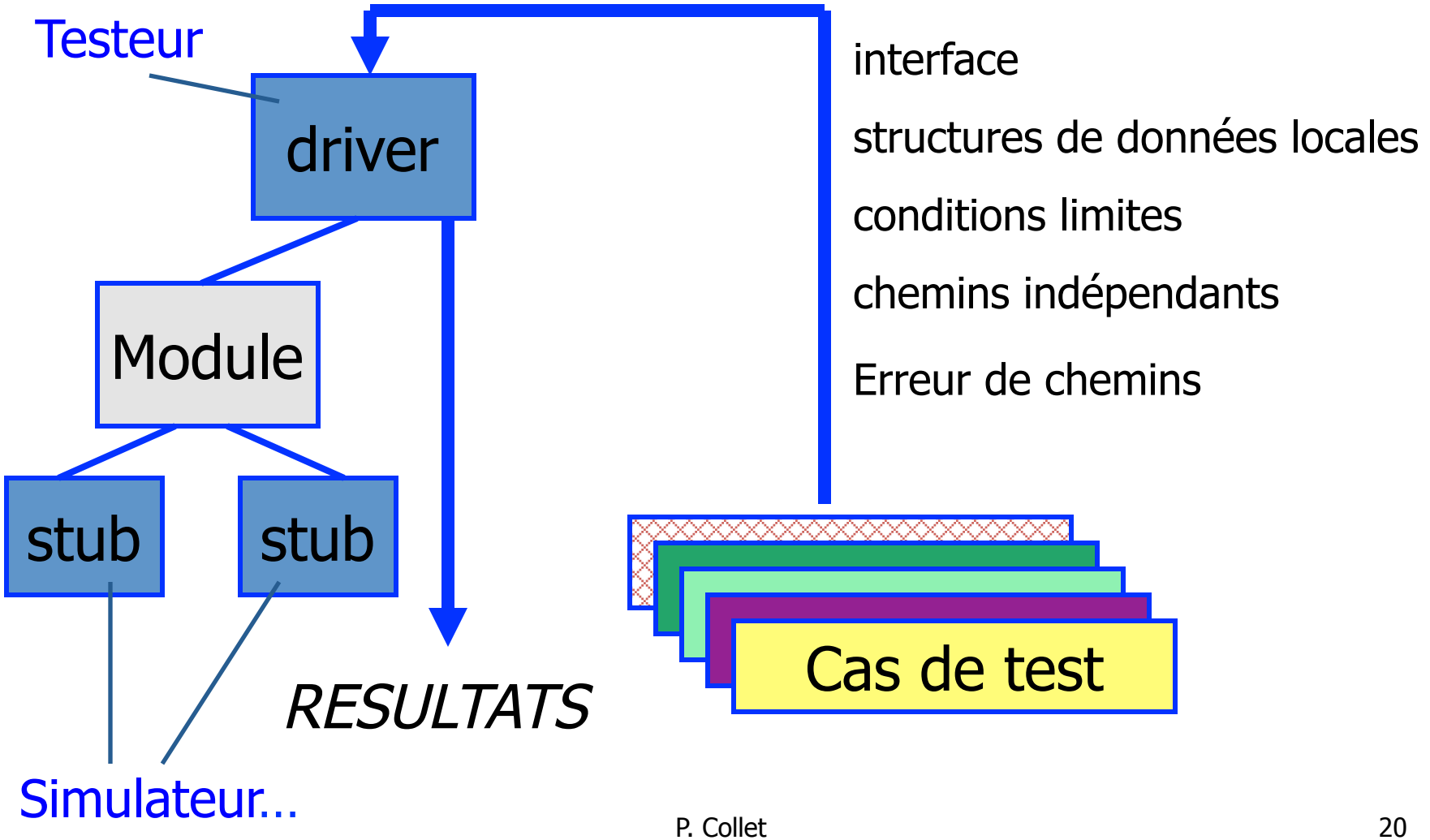
- On converse les données de test
  - Le coût du test est amorti
  - Car un test doit être **reproductible**
- Le test est différent d'un essai de mise au point

- Le débogage est une enquête
  - Difficilement reproductible
  - Qui cherche à expliquer un problème

Les stratégies de test



Test unitaire



Tests d'intégration

Si tous les modules marchent bien séparément, pourquoi douter qu'ils ne marcheraient pas ensemble ?

Réunir les modules : **Interfacer**

Big Bang

Stratégie de construction incrémentale

Intégration partielle

Test de non-régression

Tests de charge et de performance

- Charge :
  - Tests de vérification des contraintes de performance en **pleine charge** : avec les contraintes maximales
    - Mesure des temps d'exécution depuis l'extérieur
    - Vision de l'utilisateur face au système *chargé*
- Performance :
  - Analyse des performances du logiciel en charge **normale**
  - Profilage d'utilisation des ressources et du temps passé par instruction, bloc d'instructions ou appel de fonction
    - Instrumentation des programmes par des outils pour effectuer les comptages

Tests de validation et qualification

- Rédigés à partir des spécifications fonctionnelles et des contraintes non fonctionnelles
- Composition :
  - Préconditions du test
  - Mode opératoire
  - Résultat attendu
  - Structuration en ***Acceptation, refus et panne***
  - Résultat des passages
  - Fiche(s) d'anomalie liée(s)

Organiser l'activité de tests

- Qui teste le logiciel ?
  - Développeur : comprend bien le système mais, testera « gentiment » et est motivé par la *livraison*
  - Testeur indépendant : doit apprendre le système mais, essaiera de le casser et est motivé par la *qualité*
- Mettre en place les différents types de tests :
  - tests unitaires
  - tests d'intégration
  - tests de validation
  - tests de qualification
  - tests de suivi d'exploitation

Organiser l'activité de tests (suite)

- Les jeux de test sont des produits :
  - Spécification et développement des tests
  - Contraintes de **reproduction** des tests
  - Taille et coût minimum pour une probabilité de détection maximum
- Les tâches associées aux tests
  - Planification
  - Spécification et réalisation des jeux de tests
  - Passage des tests et évaluation des résultats

☞ *Commencer le plus tôt possible*

Jeux de test (= $\sum$ cas de test)

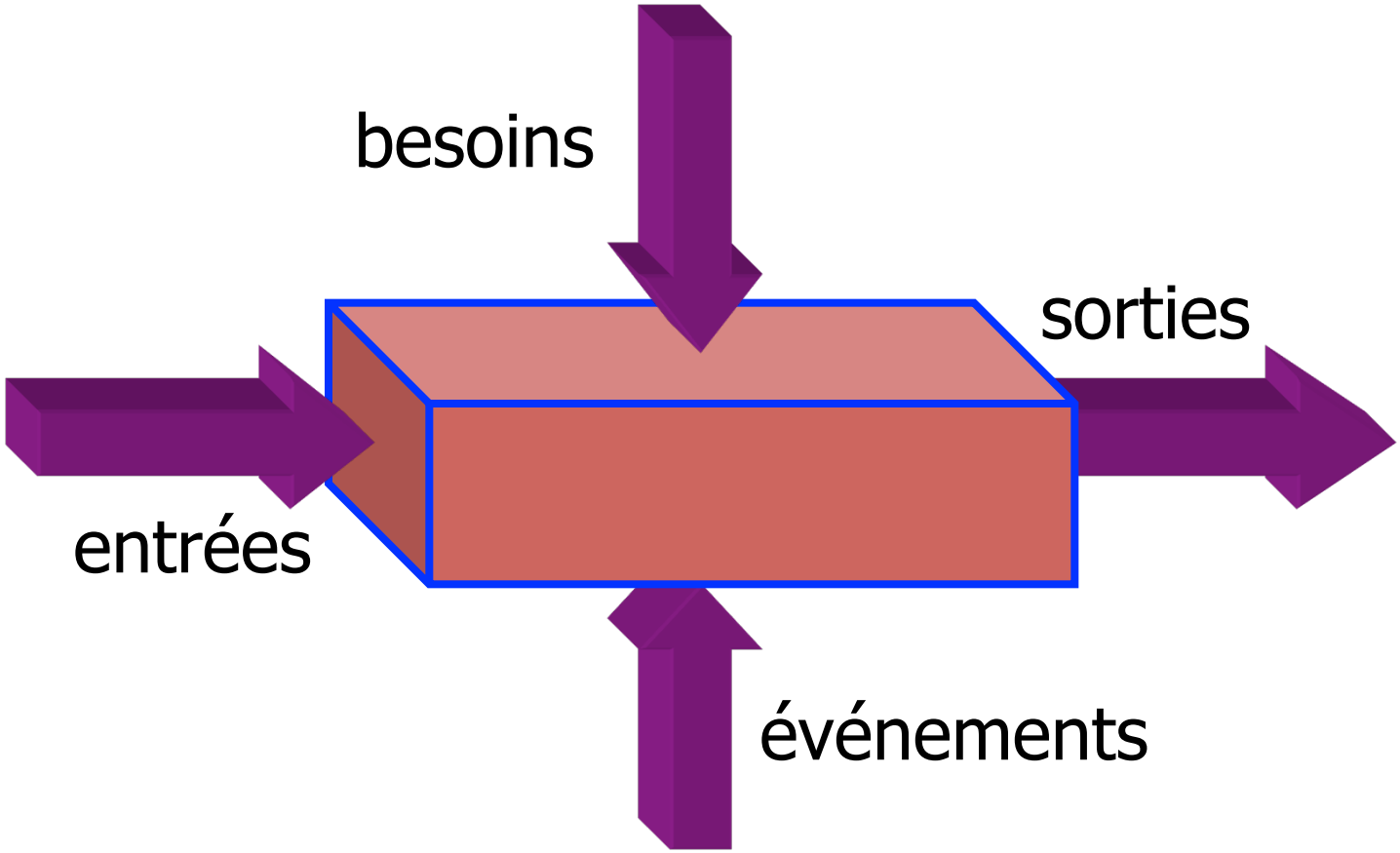
- Décrivent comment tester un système/module
- La description doit faire apparaître :
  - L'état du système avant l'exécution du test
  - La fonction à tester
  - La valeur des paramètres pour le test
  - Les résultats et sorties attendus pour le test
- Objectif ⇒ *Découvrir des erreurs*
- Critère ⇒ *de manière complète*
- Contrainte ⇒ *avec un minimum d'effort et dans un minimum de temps*

Cas de test : exemples

- État du système avant exécution du test
  - *ResourcePool* est non vide
- Fonction à tester
  - *removeEngineer(anEngineer)*
- Valeurs des paramètres pour le test
  - *anEngineer* est dans *ResourcePool*
- Résultat attendu du test
  - ResourcePool = ResourcePool \ anEngineer

- État du système avant exécution du test
  - *ResourcePool* est non vide
- Fonction à tester
  - *removeEngineer(anEngineer)*
- Valeurs des paramètres pour le test
  - *anEngineer* **N** 'est **PAS** dans *ResourcePool*
- Résultat attendu du test
  - EngineerNotFoundException est levée

Test en boîte noire



Tests fonctionnels en boîte noire

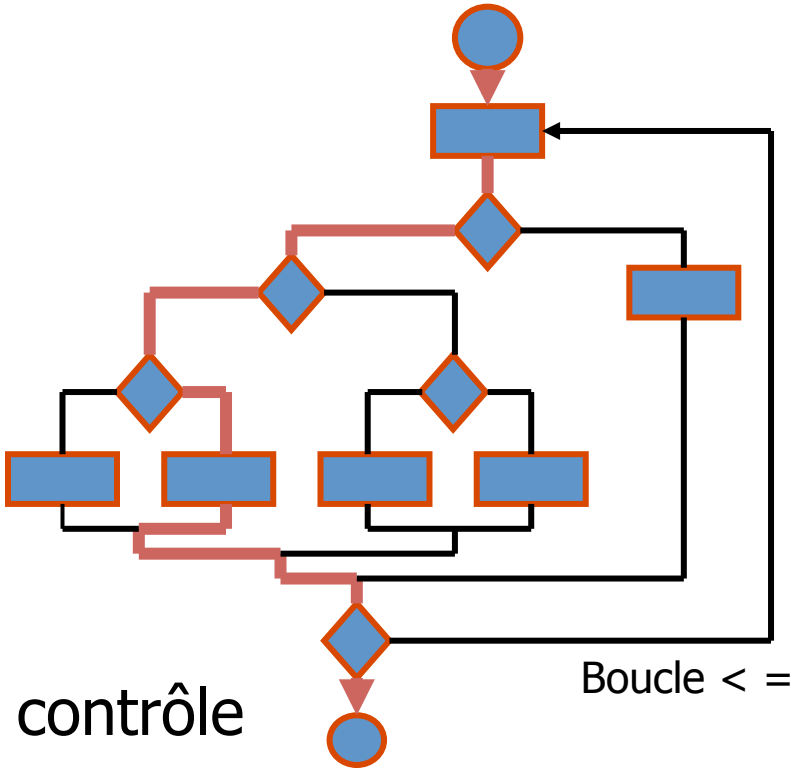
- Principes
  - S'appuient sur des spécifications externes
  - Partitionnent les données à tester par **classes d'équivalence**
    - Une valeur attendue dans 1..10 donne [1..10], < 1 et > 10
  - Ajoutent des valeurs « pertinentes », liées à l'expérience du testeur
    - Tests aux bornes : sur les bornes pour l'acceptation, juste au delà des bornes pour des refus

Pourquoi faire des tests en boîte blanche ?

- Tests en boîte noire:
  - Les besoins sont satisfaits
  - Les interfaces sont appropriées et fonctionnent
- Pourquoi s'occuper de ce qui se passe à l'intérieur ?
  - Les erreurs de logique et les suppositions incorrectes sont inversement proportionnelles à la probabilité d'exécution du chemin !
  - On croît souvent qu'un chemin ne va pas être exécuté ; en fait, la réalité va souvent à l'encontre des intuitions
  - Les erreurs de saisie sont aléatoires; il est vraisemblable que certains chemins non testés en contiennent

Test en boîte blanche

- Les données de test sont produites à partir d'une analyse du code source
- Critères de test
  - Tous les chemins
  - Toutes les branches
  - Toutes les instructions

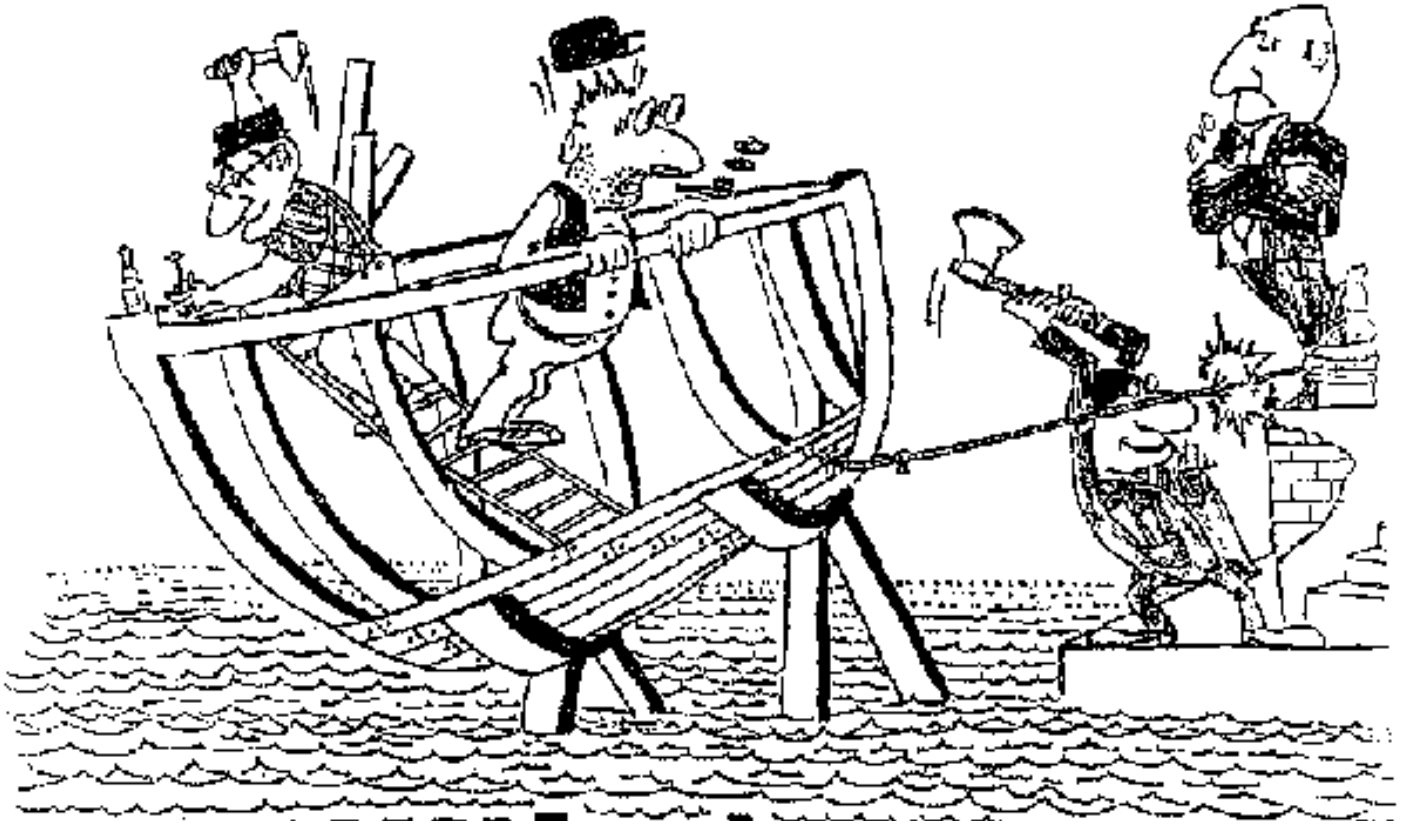


☞ Analyse du graphe de flot de contrôle

☞ Analyse du flux de données

Conclusion

Ne jamais être trop ambitieux…



La date limite, c'est la date limite !