La *substantifique moelle* du test de logiciel

Sandrine-Dominique Gouraud,
sandrine.gouraud@sopra.com

Bibliographie (1/2)

- *Syllabus de l'ISTQB*

http://www.istqb.org/

- G. Myers *The Art of Software Testing*
- Les cours de Y. Le Traon et B. Baudry

http://www.irisa.fr/triskell/perso_pro/yletraon/cours/

- Les cours de B. Legeard et F. Bouquet

http://lifc.univ-fcomte.fr/~bouquet/Test/cours_Test.pdf

Bibliographie (2/2)

- Le cours de I. Parissis

http://membres-liglab.imag.fr/donsez/ujf/m1info/tagl/Test-Logiciel-TAGL.pdf

- *Le cours de D. Longuet*

https://www.lri.fr/~longuet/ante_enseignements.html

- Le cours de S. Bardin

http://sebastien.bardin.free.fr/

Pourquoi tester?

- Parce ce que tout être humain peut faire une erreur (méprise) qui produit un défaut (bogue) dans le code, dans un logiciel ou un système, ou dans un document
- Et si un défaut du code est exécuté, il peut générer une défaillance (ou pas)


525 kg 7 PERS.
0
7
8

Pourquoi tester ?

1. Limiter le coût des bogues
   - Baisse de réputation
   - Économique:
     - Perte de contrat
     - Application de pénalités
     - estimation à 64 milliards de $/an rien qu'aux U.S.A. (2002)
   - Humains, environnementaux, etc.

Pourquoi tester?

- Quelques exemples:
  - Ariane 5 (1996):
    - problème de conversion de nombres,
    - Coût : 370 millions de dollars
    - http://fr.wikipedia.org/wiki/Vol_501_d'Ariane_5
  - Machine de radio-thérapie Therac-25, (1985-87) :
    - problème de synchronisation.
    - Coût : plusieurs vies
    - http://fr.wikipedia.org/wiki/Therac-25
  - Panne d'électricité aux États-Unis et au Canada (2003)
    - Impact sur 55 millions d'habitants.
    - Coût : 6 milliards de dollars

Pourquoi tester?

- USS Yorktown (1998)
  - Une division par zéro coupe les moteurs
- Ariane 5 (1996)
  - Mauvaise réutilisation
- Mars orbiter (1999)
  - Plusieurs unités de mesure
- Système de guidage (2002)
  - Initialisation erronée
- The Patriot and the Scud
  - mauvais arrondi dans une soustraction

Pourquoi tester?

2. Assurer la qualité d'une application
   - Pour atteindre les objectifs de normes/lois (ex: DO =178B pour l'aviation)
   - Pour atteindre un bon rapport qualité/prix
   - Pour maintenir la confiance du client

ISO 9126: qualité d'un logiciel

- Capacité fonctionnelle:
  - Le logiciel répond-il aux besoins fonctionnels exprimés?
- Fiabilité:
  - Le logiciel maintient-il son niveau de service dans des conditions précises et pendant une période déterminée?
- Facilité d'utilisation:
  - Le logiciel requiert-il peut d'effort à l'utilisation?

ISO 9126: qualité d'un logiciel

- Rendement/efficacité:
  - Le logiciel requiert-il un dimensionnement rentable et proportionné de la plate-forme d'hébergement en regard des autres exigences?
- Maintenabilité:
  - Le logiciel requiert-il peu d'effort à son évolution par rapport aux nouveaux besoins?
- Portabilité:
  - Le logiciel peut-il être transférer d'une plate-forme d'environnement à une autre?

Quelques chiffres (GARNET)

- La réalité des projets
  - Dérive dans le temps
    - 51% des projets ne respectent pas leurs dates de livraison
    - 16 % des projets livrent dans les temps et avec leur budget prévu
  - Surcoût du projet
    - 43% des projets dépassent leur budget initial
    - dont 53% des projets coûtent 189% du budget initial estimé

Définition(s) du test

- *Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il répond à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus*
  - IEEE (Standard Glossary of Software Engineering Terminology)
- *Tester, c'est exécuter le programme dans l'intention d'y trouver des anomalies ou des défauts*
  - G. Myers (The Art of Software testing)

Définition(s) du test

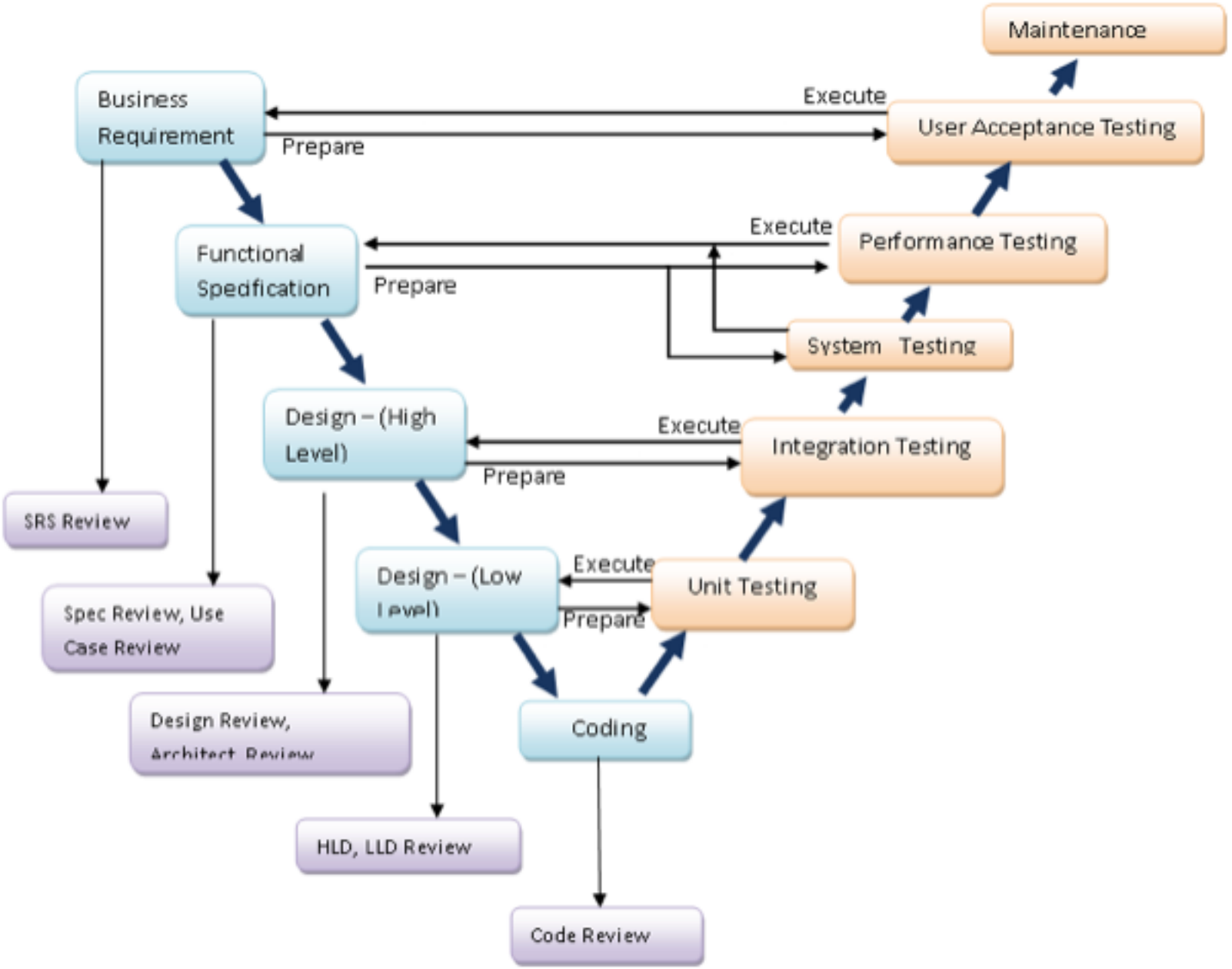
- *Testing can reveal the presence of errors but never their absence*
  - Edgar W. Dijkstra. *Notes on structured programming*. Academic Press, 1972.

Objectifs du test

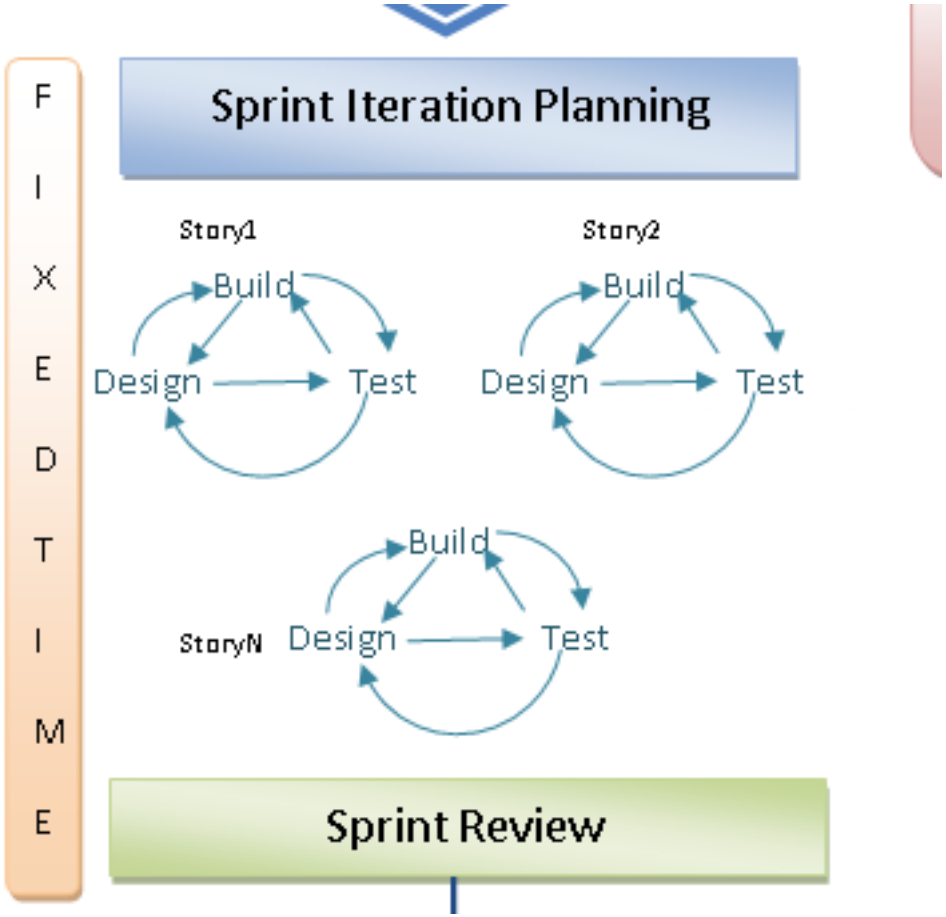
- Trouver des défauts
- Augmenter le niveau de confiance en la qualité d'un logiciel
- Fournir aux décideurs *go/no go* un état le plus précis possible de la qualité du logiciel
- Prévenir des défauts

Positionnement du test dans le cycle de vie du logiciel

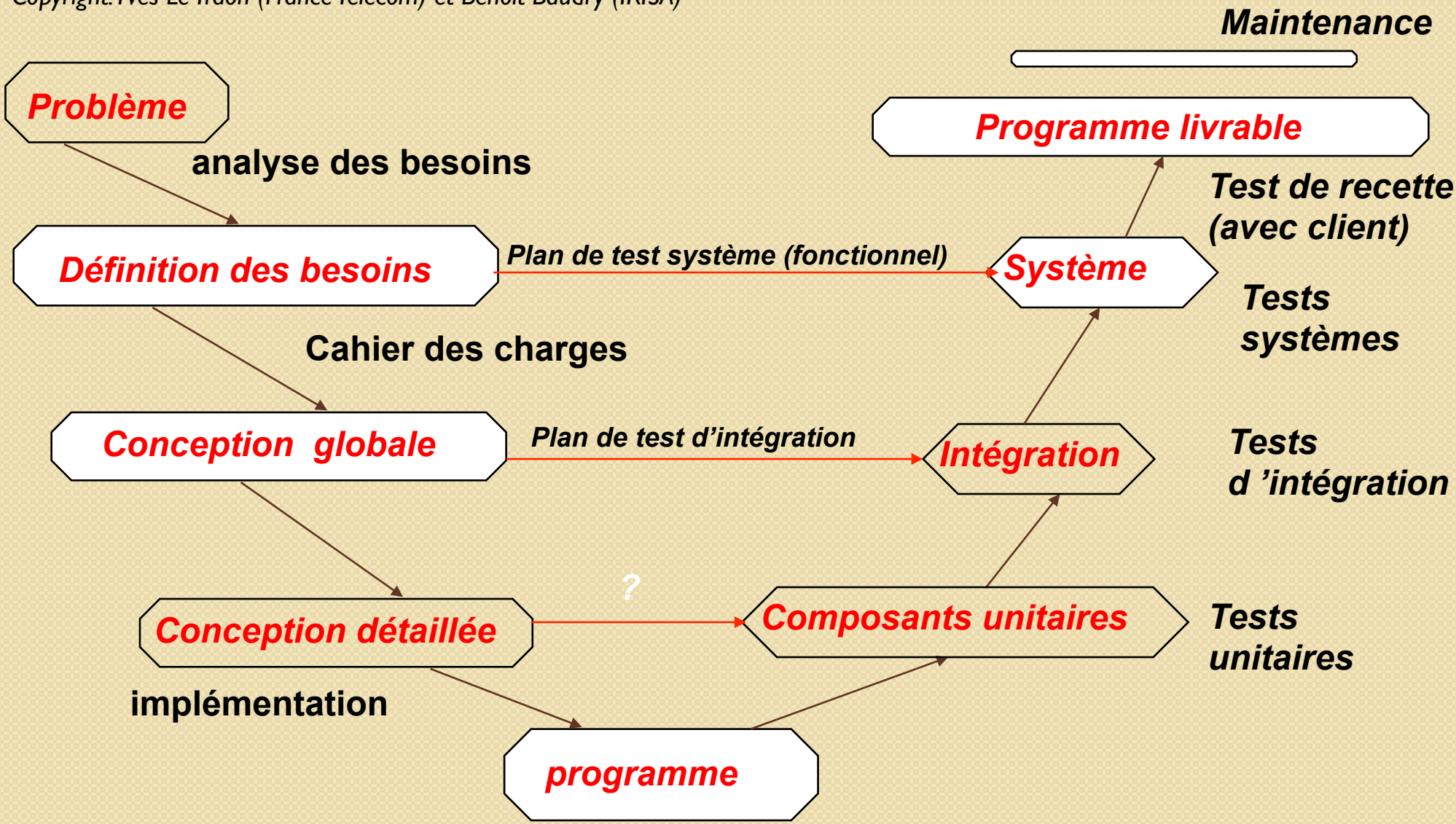
- Source: http://onlinewebapplication.com/agile-test-methodology-model/



Agile Test Methodology Vs V Model Figure 1


Sprint Iteration Planning
F
I
X
E
D
T
I
M
E
Story1
Build
Design
Test
Story2
Build
Design
Test
Build
StoryN
Design
Test
Sprint Review

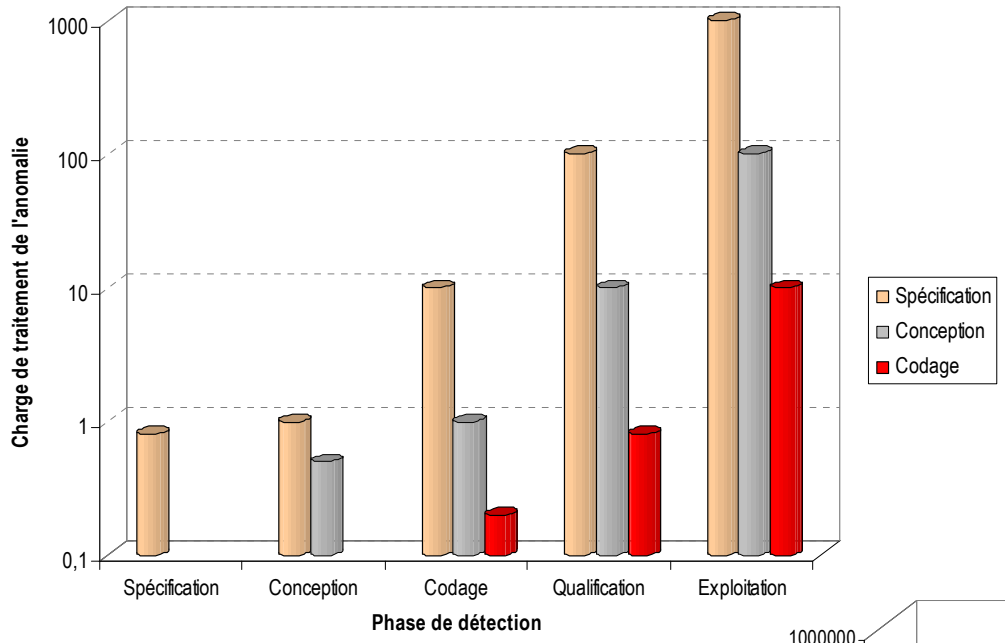
Hiérarchisation des tests



Les 7 principes généraux

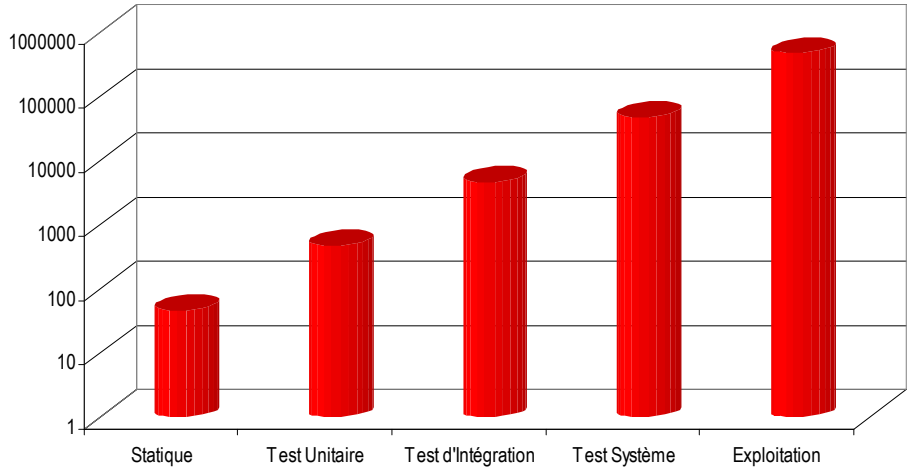
1. Les tests montrent la présence de défauts
   - Jamais leur absence
2. Les tests exhaustifs sont impossibles
   - Exemple Sopra: test d'une IHM contenant 20 écrans, 4 menus, 3 options, 10 champs, 2 types de données, 100 valeurs possibles
   - 20*4*3*10*2*100= 480 000 cas de tests
   - Un expert (10s par test!) mettrait 180 jours
3. Il faut tester le plus tôt possible

Constat



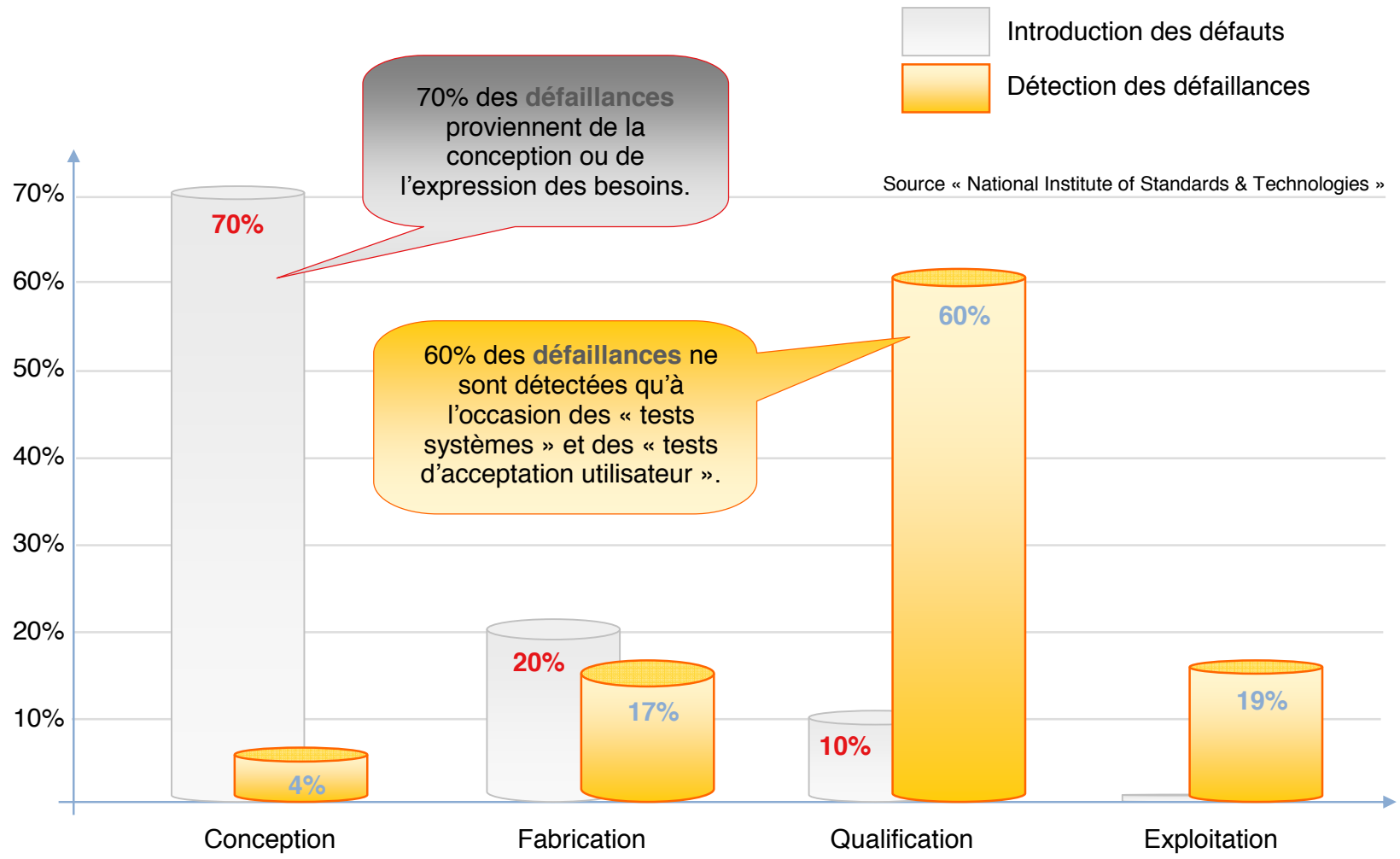
Evolution de la charge du traitement (correction) d'une anomalie détectée pendant une activité

Evolution du coût (en €) du diagnostic et de la correction d'une anomalie par rapport aux niveaux de test (source : UK Orange services)



Constat

Décalage entre injection des défauts et détection des défaillances



Les 7 principes généraux

4. Les défauts sont généralement regroupés
   - Loi de Pareto: 80% des défauts sont concentrés dans 20% des modules
5. Paradoxe du pesticide
   - Les mêmes tests ne décèlent plus de défauts
6. Les tests dépendent du contexte
   - Les tests de sécurité dépendent de l'application testée

Les 7 principes généraux

7. L'illusion de l'absence d'erreur
   - Trouver et corriger des défauts ne sert à rien si le système conçu est inutilisable (opérativité, conformité, attractivité, performance)

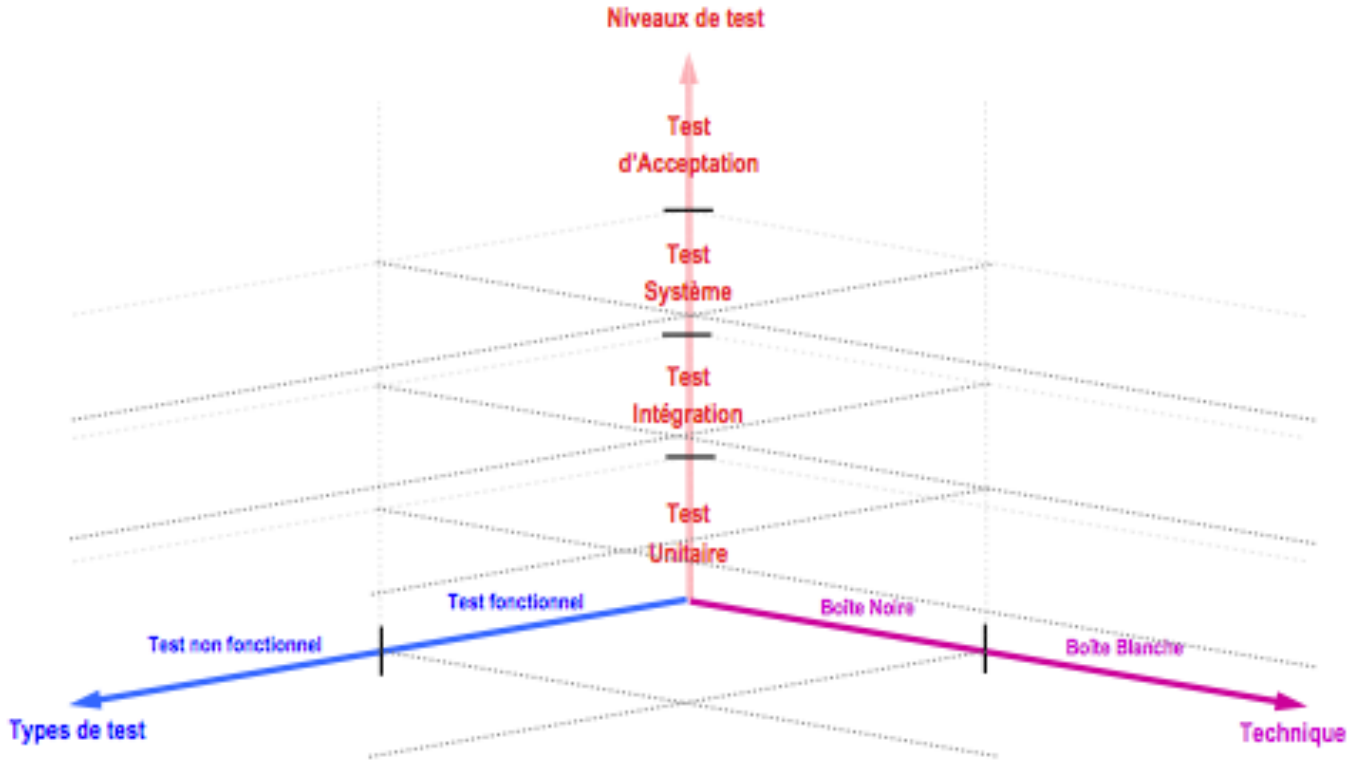
Tester <> Déboguer

- Tester et déboguer, ce n'est pas la même activité et les responsables sont différents:
  - Les testeurs testent
  - Les développeurs déboguent
- Déboguer: activité de développement permettant d'analyser, trouver et supprimer les causes de la défaillances

Typologie des tests

Typologie

Vue d'ensemble



Les différents niveaux de test

- **Tests unitaires ou de composant** : tester chaque élément de manière isolé
  - Élément: méthode, classe, composant, etc.
  - Acteurs: les développeurs
  - Le code est accessible
  - Aucun rapport d'incident
  - Vocabulaire: bouchon/simulateur/pilote, test statique/dynamique

Les différents niveaux de test

- **Test d'intégration**: tester le bon comportement/l'intéraction des éléments suite à une composition d'éléments
  - Acteurs: les développeurs
  - Le code est accessible ou pas
  - Vocabulaire: big-bang, bottom-up, top-down, ihm

Les différents niveaux de test

- **Test système ou de conformité**: assurer que l'application présente les fonctionnalités attendues
  - Acteurs: l'équipe dédiée
  - Le code n'est pas accessible
  - Vocabulaire: qualification, performance

Les différents niveaux de test

- **Test de validation ou d'usine ou d'opération**: valider l'adéquation avec les besoins du client
  - Acteurs: l'équipe dédiée, le client
  - Le code n'est pas accessible
  - Vocabulaire: Alpha tests, Beta tests

Les différents types de test

- **Test des fonctionnels**: pertinence, exactitude, interopérabilité, sécurité, conformité
- **Test des caractéristiques non-fonctionnelles**: fiabilité, facilité d'utilisation, rendement/efficacité, maintenabilité, portabilité

Les différents types de test

- **Test de la structure**: architecture logicielle
- **Test lié au changement:** test de confirmation, test de régression
- **Test de maintenance**: modification, migration, suppression

Après la théorie, la pratique

Avec le test unitaire

Le test en entreprise

- Le test effectué généralement en entreprise est un test dynamique: il s'agit d'exécuter du code pour s'assurer d'un fonctionnement correct
- Il appartient aux méthodes dites V & V:
  - Validation: l'application répond-t-elle aux besoins du client?
  - Vérification: l'application fonctionne-t-elle correctement?

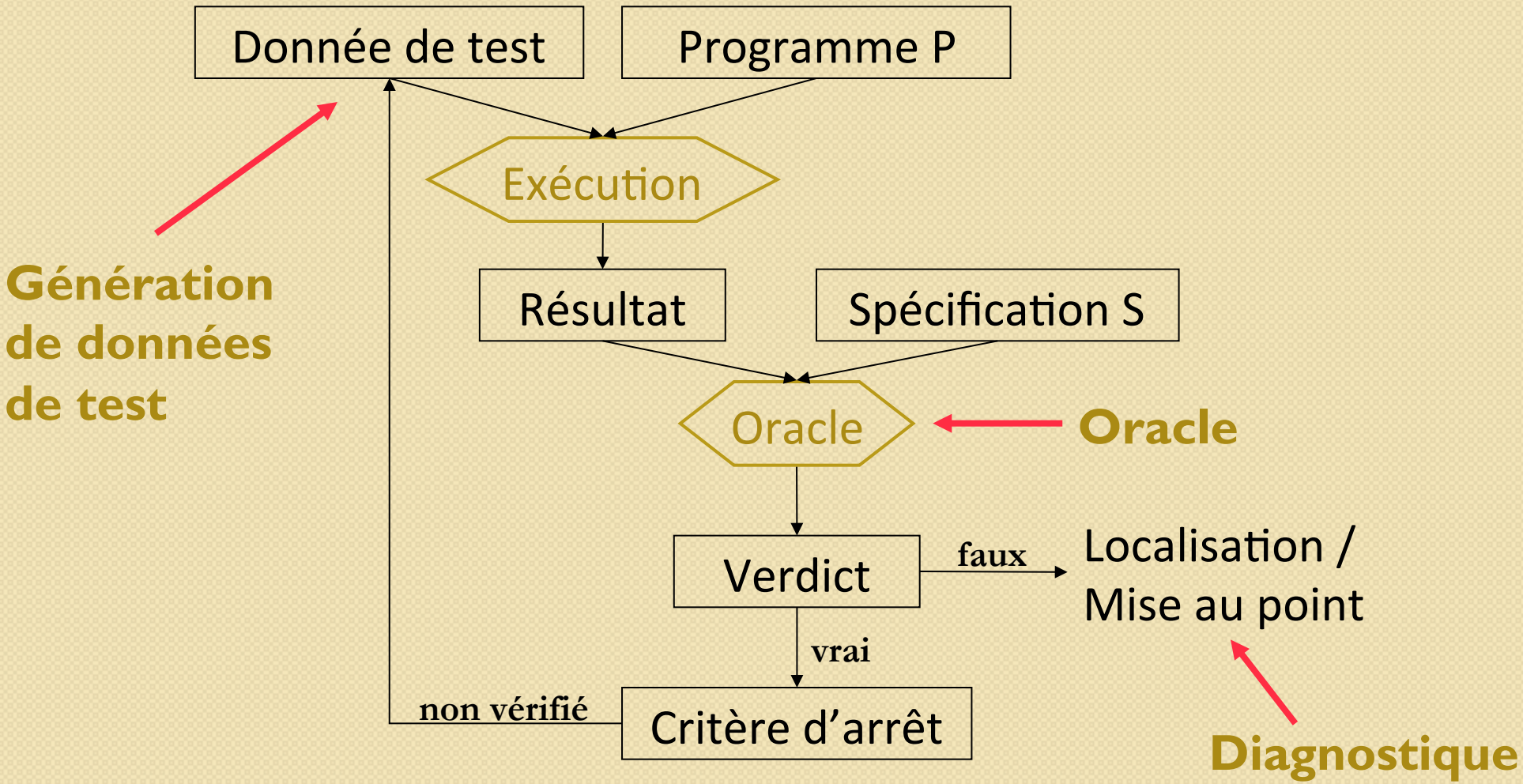
Les différentes méthodes V&V

- Test statique: revue de code ou de spécifications, etc.
- **Test dynamique**
- Vérification symbolique: exécution symbolique, etc.
- Vérification formelle: preuve, model-checking, etc.

Cycle de vie d'un test

1. Choisir un scénario à exécuter (cas de test)
2. Estimer le résultat attendu (oracle)
3. Déterminer les données du test et le résultat concret attendu
   - Données du programme
   - Suite d'actions à exécuter
4. Exécuter le test
5. Comparer le résultat attendu et le résultat obtenu

Le test dynamique : processus



Exemple (1/2)

Spécification: *trier un tableau **T** par ordre croissant sans doublon*

1. Choisir un scénario à exécuter (cas de test)
    1. S1: **T** est vide
    2. S2: **T** contient des doublons
    3. S3: **T** ne contient pas de doublons
2. Estimer le résultat attendu (oracle)
    1. O1: le résultat est un tableau vide
    2. O2, O3: le résultat est un tableau trié par ordre croissant sans doublon

Exemple (2/2)

Spécification: *trier un tableau **T** par ordre croissant sans doublon*

3. Déterminer les données du test et le résultat concret attendu

S1	T= []	R=[]
S2	T=[3;3;4;5;0;4;-4]	R=[-4;0;3;4;5]
S3	T=[-3;-5;7;3;100]	R=[-5;-3;3;7;100]

Comment choisir les bons scénarios?

- Idéalement, il faudrait qu'ils soient exhaustifs mais cela est généralement impossible.
- Ils doivent donc permettre d'exécuter un maximum de comportements différents de l'application
  - Couverture des cas dits nominaux : les cas de fonctionnements les plus fréquents
  - Couverture des cas limites ou délicats
  - Entrées invalides
  - Montée en charge
  - Sécurité

Comment choisir les bons scénarios?

- Ils doivent aussi permettre de contribuer à assurer la qualité de l'application: en trouvant rapidement le plus de bogues possibles
- Ils doivent aussi permettre de démontrer la qualité de l'application à un tiers: en étant le plus représentatif possible

Comment choisir les bons scénarios?

- L'expérience des anomalies: certains tests sont effectués en fonction d'anomalies récurrentes, de cas récurrents
- La connaissance du développeur:
  - ses défauts/qualités,
  - ses affinités par rapport aux fonctionnalités,
  - ses classiques

Critères de sélection des tests

Comment sélectionner les tests?

- **Test fonctionnel ou boîte noire**: la sélection se fait en se basant sur la spécification
  - On teste ce que doit faire l'application
- **Test structurel ou boîte blanche/ transparente**: la sélection se fait en se basant sur le code
  - On teste la manière dont l'application le fait
- **Test probabiliste**: la sélection se base sur le domaine d'entrée en fonction d'arguments probabilistes

Critères de sélection des tests

Comment sélectionner les tests?

- Ces différentes approches sont utilisées de manières complémentaires car elles ne remontent pas les mêmes bogues.
- Ils existent même des approches qui les combinent pour combler les défauts des unes par les qualités de l'autre.

Test fonctionnel

- Avantages:
  - Ne dépend pas du code: langage, technique, style
  - Peut donc être défini avant le développement
  - Basé sur l'interface et les fonctionnalités, le nombre de scénarios est de taille raisonnable
  - Assure l'adéquation du code avec la spécification
- Défauts:
  - Ignore les défauts de programmation
  - La concrétisation des scénarios n'est pas toujours évidente
- Utilisé pour le test unitaire et le test système

Test fonctionnel

- Ce type de tests aurait pu éviter le crash d'Ariane 5 qui était lié à un problème métrique indétectable par test unitaire

Spécification: *retourne la somme de 2 entiers modulo 15000*

Somme(x,y)=

```
If (x=123 & y=421)
    then resultat:= x-y
    else resultat:=x+y
Return resultat
```

Test structurel

- Avantages:
  - Basé sur le code: le nombre de scénarios est plus important mais plus précis
  - Sensible aux défauts de programmation
- Défauts:
  - Le code doit être accessible
  - Problème d'Oracle: le résultat est-il celui réellement attendu?
  - Ignore les fonctionnalités absentes
- Utilisé pour le test unitaire

Test structurel

Spécification: *retourne la somme de 2 entiers modulo 15000*

Somme(x,y)=

If (x=123 & y=421)

then **resultat:= x-y**

else **resultat:=x+y**

Return resultat

Test aléatoire

- Avantages:
  - Ne dépend pas du code: langage, technique, style
  - Basé sur l'interface et les fonctionnalités, le nombre de scénarios est de taille raisonnable
  - Permet du test en masse
- Défauts:
  - Ignore les défauts de programmation
  - La concrétisation des scénarios n'est pas toujours évidente
- Utilisé pour le test unitaire et le test système

Test aléatoire

Spécification: *retourne la somme de 2 entiers modulo 15000*

Somme(x,y)=

If (x=123 & y=421)

then **resultat:= x-y**

else **resultat:=x+y**

Return resultat

Critère de test

Quand arrêter de tester ou comment déterminer un bon jeu de test?

- Un bon critère doit être: efficace, objectif, simple, automatisable
- Test fonctionnel:
  - couverture de scénarios d'utilisation
  - couverture des partitions des domaines d'entrée
  - couverture des cas limites
- Test structurel:
  - couverture des instructions, des enchaînements
  - couverture des conditions
  - couverture des dépendances de données

Couverture des partitions des domaines d'entrée

- À partir de la spécification
  - déterminer le domaine des entrées
- Partitionner le domaine des entrées en classes d'équivalences
  - identifier des classes d'équivalence pour chaque domaine d'entrée
  - les classes d'équivalence forment une partition du domaine de chaque donnée en entrée
  - choisir une donnée dans chacune

Couverture des cas limites

- Intuition:
  - de nombreuses erreurs se produisent dans les cas limites
- Pour chaque donnée en entrée
  - déterminer les bornes du domaine
  - prendre des valeurs sur les bornes et juste un peu autour
- Souvent utilisée avec la technique de partitionnement : les valeurs aux limites peuvent être des valeurs aux frontières des partitions

Exemple

- Le programme lit trois nombres réels qui correspondent à la longueur des côtés d'un triangle.
  - Si ces trois nombres ne correspondent pas à un triangle, imprimer un message d'erreur.
  - S'il s'agit d'un triangle, le programme détermine
    - s'il est isocèle, équilatéral ou scalène
    - et si son plus grand angle est aigu, droit ou obtu.

Couverture des partitions des domaines d'entrée

	aigu	obtu	droit
scalène	6,5,3	5,6,10	3,4,5
isocèle	6,1,6	7,4,4	√2,2,√2
équilatéral	4,4,4	impossible	impossible

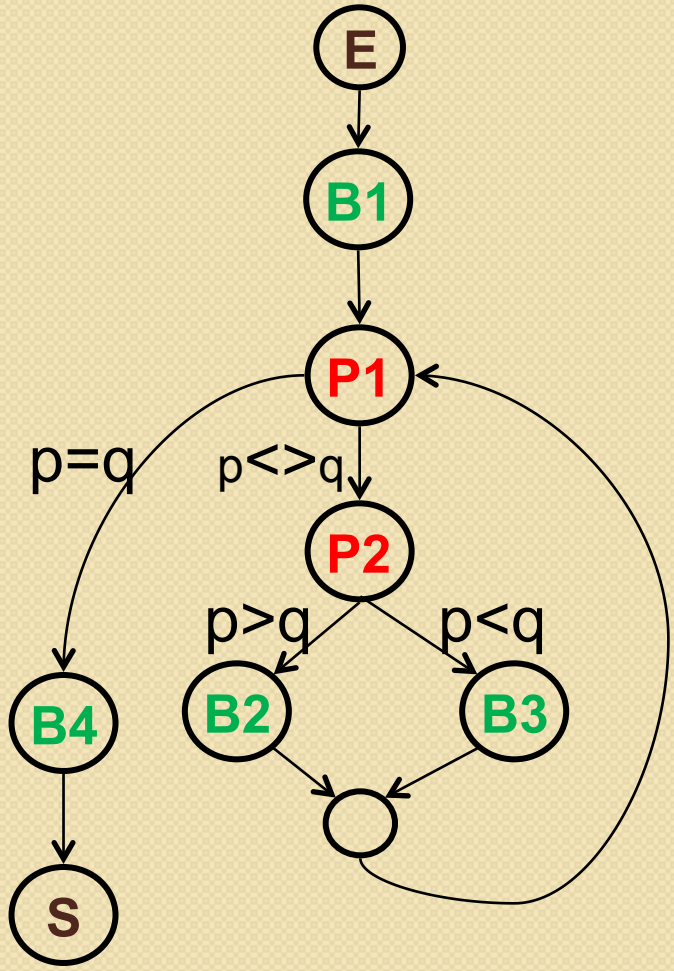
Couverture des cas limites

1, 1, 2	non triangle
0, 0, 0	un seul point
4, 0, 3	une des longueurs est nulle
1, 2, 3.00001	presque triangle
0.001, 0.001, 0.001	très petit triangle
88888, 88888, 88888	très grand
3.00001, 3, 3	presque équilatéral
2.99999, 3, 4	presque isocèle
3, 4, 5.00001	presque droit
3, 4, 5, 6	quatre données
3	une seule donnée
5, 5, A	une lettre
	pas de donnée
-3, -3, 5	données négatives
...	

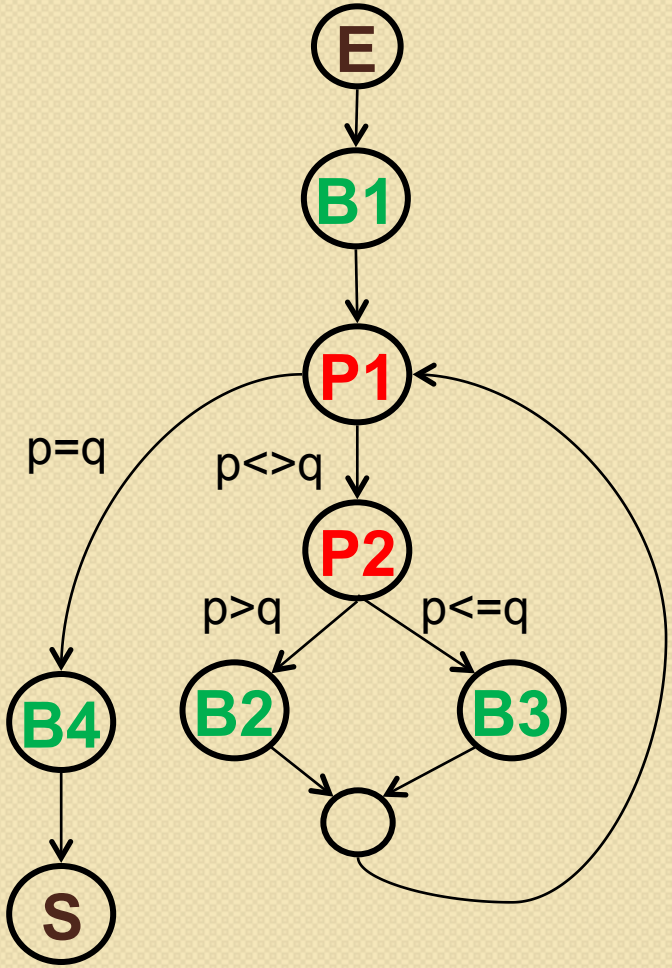
Exemple: critère de test structurel

PGCD de 2 nombres
Précondition: p et q entiers naturels positifs

```
pgcd: integer is
local p,q : integer;
do
    read(p, q) B1
    while p<> q do P1
          if p > q  P2
                    then
                      p := p-q   B2
                    else
                      q:= q-p    B3
          end -- if
    end -- while
    result:=p B4
end-- pgcd
```



Exemple: critère de test structurel



Tous les noeuds:

(E, B1, **P1**, **P2**, B2, **P1**, B4, S)
(E, B1, **P1**, **P2**, B3, **P1**, B4, S)

Tous les arcs : idem

Tous les chemins élémentaires (1-chemin) :

idem + (E, B1, **P1**, B4, S)

Tous les 2-chemins :

idem +
(E, B1, **P1**, **P2**, B2, **P1**, **P2**, B2, **P1**, B4, S)
(E, B1, **P1**, **P2**, B2, **P1**, **P2**, B3, **P1**, B4, S)
(E, B1, **P1**, **P2**, B3, **P1**, **P2**, B2, **P1**, B4, S)
(E, B1, **P1**, **P2**, B3, **P1**, **P2**, B3, **P1**, B4, S)

Tests en utilisant UML

- Voir le diagramme de séquence comme un graphe: un test, un chemin, une séquence
- Cas d'utilisation: au moins un test par cas
- Diagramme états-transitions
- Diagramme d'activité

Automatisation des tests

- C'est très important car l'activité est **difficile** et **très chère**.
- Difficultés:
  - Trouver des défauts n'est pas naturel (surtout chez les développeurs)
  - La qualité dépend de la pertinence des jeux de test
- Coûts:
  - Logiciels critiques: >50% du coût total du développement.
  - Logiciel standard: environ 30% du coût total

  Exemple: environ 10 milliards de $/an sont dépensés en tests rien qu'aux U.S.A

Automatisation des tests

- Une bonne automatisation est synonyme:
  - D'amélioration de la qualité
  - De maintenance simplifiée
- Il faut donc:
  - beaucoup de scénarios efficaces
  - une plateforme dédiée (serveur, base de données)
- Quelques outils: Sélénium, TestComplete, Quality Test Plan, etc.
- Bien choisir son outil: exécution, dépouillement, mise à jour

Pourquoi encore des tests manuels?

- Application très paramétrable
- Application qui doit tourner:
  - Sur toutes les plateformes (Windows, Mac, Unix, Linux)
  - Sur tous les navigateurs (de IE6 à IE10, Firefox, Chrome, etc)
  - Avec une base de données (sous divers Oracle, divers SQLServer, DB2)
  - Dans plusieurs langues

Et pour finir, un peu de psychologie

L'ennemi de mon ennemi (le bogue) est mon ami…

Le testeur, un destructeur ?

- Il trouve les cas où le logiciel peut être défaillant
- Il a une vue d'ensemble
- Il vérifie que le logiciel répond bien aux exigences
- Il anticipe les problèmes possibles
- Il donne de l'importance au détail surtout s'il s'agit d'une exigence
- Il est limité par le temps et n'a pas le droit aux débordements
- Plus il trouve de bogues, plus il est bon
- Il voit son travail comme une activité constructive

Le développeur, un bâtisseur

- C'est un expert qui trouve les bons algorithmes et la meilleur solution aux problèmes
- Il interprète les éventuelles ambiguïtés de la spécification
- Il peut oublier les détails visuels
- Il n'a pas une vue d'ensemble
- Il travaille sans limite de temps
- Il n'aime pas les erreurs qui montrent qu'il est faillible
- Il voit le testeur comme un messager porteur de mauvaise nouvelle et voit donc son travail comme une activité destructive

Et pourtant

- Testeurs et développeurs travaillent pour un même objectif: améliorer la qualité du logiciel
- Les testeurs ne sont pas là pour juger le travail des développeurs, d'ailleurs, que dire:
  - des tests qui remontent peu de bogues:
    - Les tests sont mauvais ou en nombre insuffisants
    - Le développement est de bonne qualité
  - des tests qui remontent beaucoup de bogues:
    - Les tests sont complets: la majeure partie des bogues a été remontée
    - Le développement est de très mauvaise qualité: quid de son avenir.

Pour finir ou presque

- Le test est un métier à part
- Il doit être effectuer à tous les niveaux
- Il doit être effectuer avec le plus d'indépendance possible
- Il faut accepter qu'il reste des bogues…

Un petit conseil les développeurs

- Importance des commentaires:
  - Au moment du développement
  - Au moment d'une correction
- Mise en place de tests *blindant* les points critiques
- Appliquer les bonnes pratiques

Ai-je le profil d'un testeur?

- Il faut:
  - De la curiosité
  - Du pessimisme professionnel
  - Un œil critique
  - L'attention du détail
  - Savoir être neutre et factuel
  - Savoir être diplomate
  - Savoir communiquer
  - Ne pas donner de solution
  - Ne pas être critique ou juge

Standards/Normes (1/2)

- ISTQB: International Software Testing Qualification Board
  - http://www.istqb.org/
  - Organisme de certification, leader mondial sur la certification du test logiciel
- CFTL: Comité Français des Tests Logiciels

Standards/Normes (2/2)

- IEEE (Institute of Electrical and Electronics Engineers)
  - IEEE 610: standard définissant les termes de l'ingénierie logicielle
  - IEEE 829: standard définissant la documentation pour le test de logiciel
- ISO (International Organization for Standardization)
  - ISO 9126: norme définissant un langage commun pour modéliser les qualités d'un logiciel

Mon ordinateur, j'essaie de faire tout ce qu'il me dit mais lui il fait rien de ce que je veux.

Anne Roumanof, Internet

Cycle de vie d'
une nouvelle version

1. Définition du cahier des charges du client
2. Définition du besoin du chef de produit
   - Obligation légale, réponse à la concurrence
   - demande interne
3. Rédaction de l'analyse
4. Développement
5. Recette du développement
6. Recette du client

C’est quoi un bogue?

- **Anomalie** (fonctionnement) : différence entre comportement attendu et comportement observé
- **Défaut** (interne) : élément ou absence d'élément dans le logiciel entraînant une anomalie
- **Erreur** (programmation, conception) : comportement du programmeur ou du concepteur conduisant à un défaut

Cycle de vie d'
un nouveau patch

- Déclaration d'incident produit:
  - client via le support, client via la recette, intégrateur, testeur
- Diagnostique de l'incident
- Correction du produit ou réponse fonctionnelle
- Validation des corrections

Définitions

- **Jeu de test**: ensemble de cas de test
- **Script de test**: code/script qui exécute automatiquement les étapes 4 et 5 en fonction de plusieurs données issues de l'étape 3