Validation, Vérification et Tests

Objectif :

Faire en sorte que le Logiciel soit de qualité.

« La qualité est l'ensemble des caractéristiques d'une entité
qui lui confèrent l'aptitude à satisfaire des besoins, exprimés ou implicites »

Point de vue du Client :
Il cherche à savoir s'il peut avoir confiance dans le produit qui lui est livré, déjà réalisé;

Point de vue du fabriquant :
Il cherche à savoir s'il peut avoir confiance dans le produit
en cours de fabrication et qui n'existe pas encore,
achevé et à livrer mais non encore utilisé;

Valider et Vérifier

Détecter et corriger les erreurs au plus tôt, pour éviter leur propagation et amplification.

Tester

Chercher à détecter les erreurs résiduelles, en faisant fonctionner le système.

PLAN

I. Terminologie

a) Validation, Vérification et Test

Validation

= confirmation par examen et apport de preuves que *les exigences requises pour l'usage prévu* sont satisfaites

Répond à la question : est-ce **le bon système**, répondant aux besoins effectifs ?

Point de vue externe, s'occupe de :
- l'aptitude du système à accomplir ses missions, atteindre les objectifs assignés
- ce que l'on peut faire avec (le **quoi**)

Se fait en référence à ce que l'on attend :
=> il est impossible de valider qq chose si ce que l'on en attend n'est pas précisément défini.

Exples :
le système final / besoins effectifs,
le système final / la spécification (étape de validation du système),
la conception / la spécification,
le code / l'algorithme.

Vérification

= confirmation par examen et apport de preuves que les propriétés et caractéristiques attendues sont satisfaites

Répond à la question : est-ce un **système bien fait**, conformément aux règles de l'art ?

Point de vue interne, s'occupe de :
- la structure du système, **comment** il est fait,
  en référence à des normes, des propriétés à satisfaire (vérifier le produit)
- des moyens mis en œuvre, le processus de production ;
  en référence à des règles sur la méthode de travail, comment on doit procéder (vérifier le processus),

Exples :
vérifier un document de spécification
  respect du plan type, présence d'un index, au plus 2 propositions par phrase, ...
vérifier un document de conception,
vérifier du code,
vérifier le respect de la procédure de test.

Tester

Exécuter le logiciel, pour trouver les erreurs qu'il contient.

principe :
prouver / convaincre que l'on n'est pas capable d'exhiber de défaut dans le logiciel, et donc qu'il n'y en a pas.
(Cf démonstration par l'absurde : impossibilité de construire un ctre-exple)

L'objectif est donc de :

- découvrir la totalité des erreurs susceptibles de se produire,
- corriger le logiciel pour les supprimer.

Taux de couverture de test
Pour prouver qu'il n'existe pas de défaut, il faut tester toutes les situations,
ce qui est généralement impossible.
Quand tous les tests sont positifs,
on a seulement montré que l'on n'a pas réussi à exhiber un défaut !
Tx de couverture : nb de test réalisés / ceux qu'il aurait fallu faire pour établir la preuve.

Tests de non-régression
Toute correction modifie le logiciel
=> les essais qui étaient positifs avant la modification le sont-ils encore après ?
Les tests de non-régression vérifient
que les tests qui étaient positifs avant la modification le sont encore après.

b) Faute, Défaut et Panne

Les Pannes, ou défaillances

Dysfonctionnement, défaillance du logiciel,
en cours d'utilisation, il ne se comporte pas comme il le « devrait ».

sévérité : étendue, gravité des conséquences de la panne (cf. criticité).

reproductibilité :
possibilité de reproduire « la même » panne lors d'une autre exécution.
Pb **d'identification** des pannes :
peut-on parler de la même panne lors de 2 exécutions différentes ?

Types de Pannes :
Panne d'exécution :
Le logiciel arrête son exécution, le calcul ne progresse plus
arrêt inopiné de l'exécution (« plantage »),
bouclage infini.

Panne fonctionnelle :
Les résultats ne sont pas ce qu'ils devraient être, compte tenu des entrées.
Pour détecter une telle panne, il faut
pouvoir observer les sorties,
disposer d'un **oracle** qui indique ce qu'elles devraient être.

Panne opérationnelle :
Les résultats sont correctes, mais ils sont fournis dans de mauvaises conditions.
Exple :
temps de réponse,
utilisation abusive de ressources (saturation du réseau, de la mémoire, , ...),
impossibilité de traiter les volumes nécessaires, indisponibilité.
Pour détecter une telle panne,
il faut pouvoir observer le fonctionnement interne du système.
Souvent difficile à reproduire, car dépendantes de l'état de l'environnement d'exécution.

Une panne peut en cacher une autre

```
   ...
a = X;            // au lieu de 1/X, panne fonctionnelle
...
a = 100 * a;      // overflow, panne d'exécution
```

Les Défauts

Erreur, anomalie dans le logiciel (la spéc., la conception, le code...).
Plus généralement, tout élément du logiciel qui porte atteinte à ses objectifs de qualité.

principe : toute Panne est provoquée par un Défaut du logiciel,
(n'est jamais due à l'usure d'un composant).

Remarque :
Un défaut est parfaitement localisé dans le produit,
les pb d'identification et de reproductibilité ne se posent pas.

Relation Défaut / Panne
Toute panne est provoquée par un défaut dans le code
exécution d'une instruction erronée,
utilisation d'une entité (type, variable, fonction, fichier) erronée.

La liaison Panne / Défaut est n : m.
un défaut peut provoquer plusieurs pannes :

```
var a integer;     // au lieu de real
...
a = a * 10;        // overflow
...
X = Y / a;         // division par 0, du fait de l'arrondi
```

une panne peut résulter de plusieurs défauts :

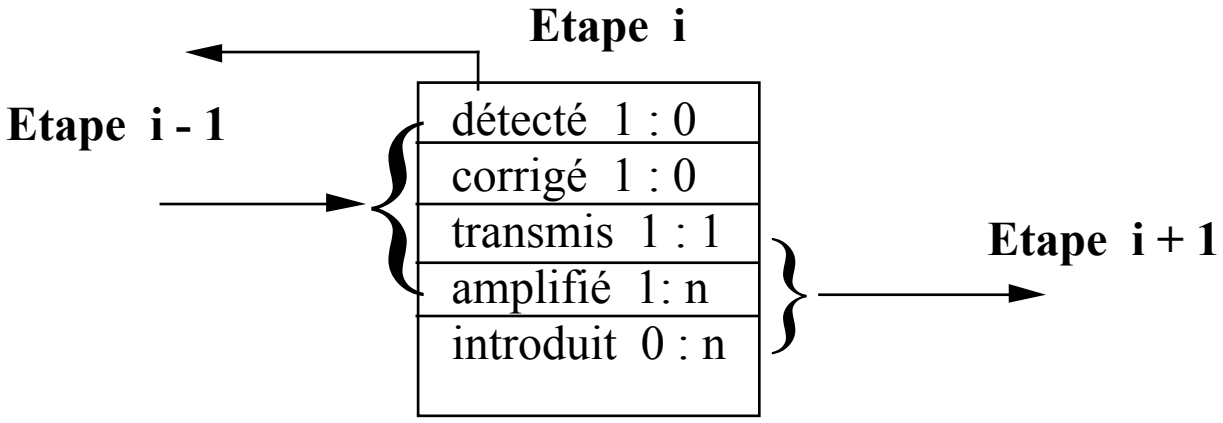
```
b = x - x;    // au lieu de x-y
...
a = b * b;         // au lieu de b*b + 1
...
X = Y / a;         // division par 0
```

Loi de Paréto : 20% des défauts provoquent 80% des pannes,
car 80% du temps d'exécution est consacrée à 20% du code.
=> il est possible d'optimiser le coût de la réduction du taux de panne.

Défaut sans panne
Certains défauts ne provoquent pas de panne,
tout en nuisant à certaines qualités.

Exple :
quitter l'application en laissant une BD ouverte;
incohérence entre la conception (erronée) et le code (correcte)
=> difficulté de maintenance;
défaut dans la documentation utilisateur
=> difficultés d'utilisation;
code mort.

Le Cycle de Vie des défauts
Des défauts sont produits à toutes les étapes du processus de développement.
Une partie des défauts du code provient des défauts
de la conception,
de la spécification,
de l'analyse des besoins.



Pour éviter la propagation des défauts,
il faut les détecter et les corriger au plus proche de leur introduction.

Les défauts introduits en début de processus sont les plus graves :
- ils sont les plus amplifiés;
- les défauts semblent être gérés en mode Pile :
  ceux introduits en premier sont détectés en dernier (cf. cycle de vie en V),
  et donc les plus onéreux à corriger.

Typologie des défauts
- action à entreprendre pour le corriger
- procédé utiliser pour le mettre en lumière

Types de Défauts dans le code
Nature des éléments qui peuvent être erronés, absents, ou en trop :

interaction :	appel de fonction du logiciel ou d'un autre système
données :	variable, paramètre ou type
algorithme	
entrée / sortie	
performance	
fonctionnalité	
interface utilisateur	
standard :	non respect d'un standard
documentation :	ambiguïté, clarté, complétude
test	
syntaxe :	erreur de forme (grammaire, orthographe, ...)

La correction des défauts
Très souvent, la correction d'un défaut introduit un nouveau défaut,
moins grave mais plus difficile à trouver.

En phase de test, on a tendance à supprimer ce qui cause la panne (soigner),
plutôt qu'à identifier le défaut (guérir)
Exple :

```
tb : array [1 to tailletb] of untype
....
a = tb(i);      // i = 0
```

le défaut provient-il de la valeur de i ou de la déclaration de tb ???

Il n'est pas toujours facile d'identifier la nature d'un défaut.
Le problème de la **non-régression** est sérieux.

Les Fautes

Comportement qui conduit à l'introduction d'un défaut,
mauvaise façon de travailler.

principe :
Tout Défaut dans le logiciel résulte d'une Faute dans le processus.

La liaison Faute / Défaut est n : m
Exple :
interview utilisateur pas assez approfondie ET
non validation du compte-rendu par l'utilisateur
=> mauvaise définition du besoin
tâche confiée à une personne non compétente
=> plusieurs défauts introduits par incompétence.

L'identification des fautes permet

- d'améliorer le plan qualité,
- de comprendre l'origine du défaut, et donc de bien le corriger,
- de découvrir d'autres défauts provoqués par cette même faute.

II. L'obtention de la fiabilité

Les objectifs

- empêcher toute possibilité d'apparition de pannes : irréaliste, et souvent inutile.
- obtenir le niveau de fiabilité requis
  (fréquence des pannes en fonction de leur degrés de sévérité),
- à moindre coût, c'est-à-dire du premier coup.

Les moyens

pas de faute => pas de défaut => pas de panne

La prévention des défauts

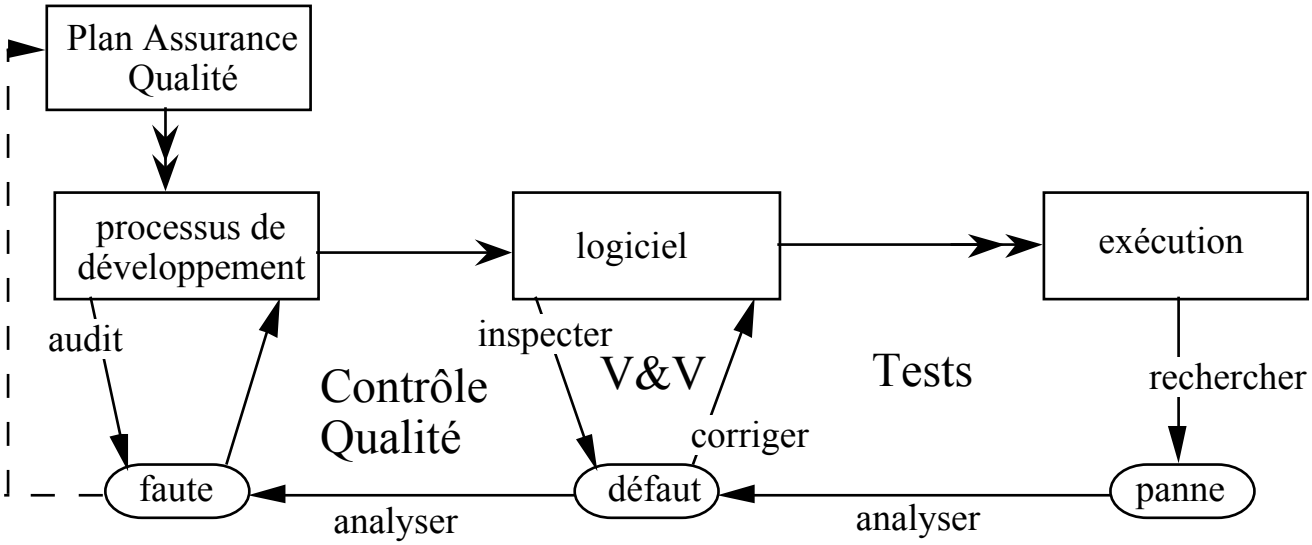
- éviter les fautes : **Assurance Qualité**
- détecter les fautes, et corriger leurs effets : **Contrôle Qualité** du processus

L'élimination des défauts

détecter et corriger les défauts : **Vérification et Validation**
Test indirectement, à partir des pannes

L'inhibition des défauts : **programmation tolérante**

en cours d'exécution, détecter et récupérer les pannes



Validation, Vérification et Test ne doivent pas être réalisés par les développeurs qui :

- sont juge et partie,
- ont une idée préconçue de ce qu'ils attendent.

=> (sur)coût de l'apprentissage du projet.

La tolérance aux Pannes

Observation par le système de son propre comportement
en cas d'anomalie, il gère la situation au mieux.

Exples

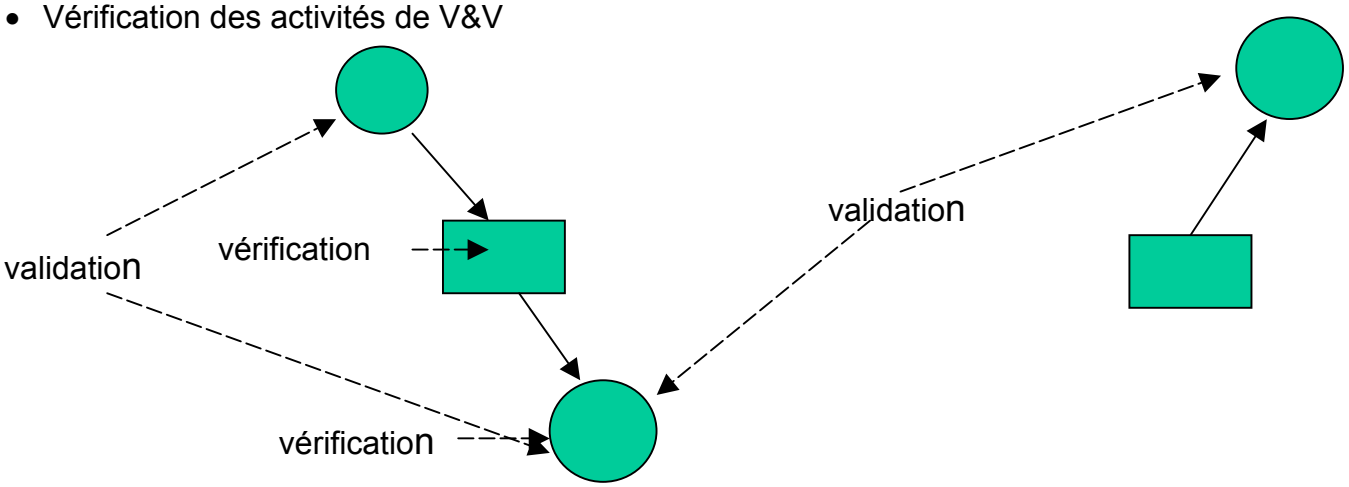
test des pré et post-conditions des fonctions,
test de tous les pointeurs, bornes de tableaux, diviseurs, ...
test des contraintes d'intégrité après chaque mise à jour de la BD,
Redondance : plusieurs logiciels s'exécutent en parallèle et s'observent mutuellement.

III. Les techniques de Validation et Vérification

Les activités de V & V

Quand

- Vérification de chaque activité
- Vérification des résultats de chaque activité (tout type de document, code)
- Validation des activités / étapes de la descente du V :
  les exigences spécifiées par l'étape précédente sont-elles satisfaites, correctement implantées ?
- Validation des activités / étapes de la remontée du V :
  le produit satisfait-il les exigences spécifiées par l'étape descendante correspondante ?
  qualification, recette : étape finale qu projet
- Vérification des activités de V&V



Objectif

Rechercher, aussi bien dans les produits que dans le processus,

- des fautes susceptibles de produire des défauts,
- des défauts susceptibles de produire des pannes,
- des anomalies, statistiquement liées à la présence de défauts,
- des atteintes aux objectifs de qualité (utilisabilité, maintenabilité, ...).

Mais on ne cherche pas les pannes (analyse statique)

Démarche

Pour chaque propriété,
  comparer le résultat obtenu avec ce qui attendu
  tant que ce n'est pas conforme (ou si la distance n'est pas acceptable)
    localiser le défaut
    corriger
    comparer ce qui attendu avec le résultat obtenu
  vérifier la non-régression

- Nécessite que ce qui est attendu soit bien défini, assorti de moyens contrôle
- Plus un processus ou document est complexe, mal structuré, informel, ...
  plus il est difficile à comprendre et donc à valider/vérifier.

Examen informel

- Re-lecture, lecture croisée,
- Inspection (cf. le cycle auteur-lecteur de SADT), en fin d'activité
- Revue de fin d'étape entre tous les partenaires, planifiée
- Audit du projet

Peut être contrôlé par rapport au nombre et au type de défaut que l'on s'attend statistiquement à trouver

C'est un moyen très efficace de trouver des erreurs,
améliore la compréhension collective du projet.

Preuve

L'utilisation des techniques de preuve nécessite
un formalisme mathématiquement défini,
de pouvoir formuler dans ce formalisme les propriétés que l'on veut prouver.

Exple :
invariants de Hoare-Floyd,
Réseaux de Petri,
langages Z, B,
Type de Données Abstraits,
Contraintes d'Intégrité d'une BD relationnelle formulées en logique du 1er ordre.

De nombreux problèmes sont indécidables
=> la preuve est manuelle, elle est spécifique au cas considéré.
Comment s'assurer qu'une preuve manuelle est correcte ?
La preuve de spécifications ou de programmes nécessite un environnement de preuve raisonnablement facile à utiliser.

Analyse automatique

Empiriquement, on constate une corrélation statistique entre
- le taux de défauts,
- certaines caractéristiques du logiciel.

Les normes, standards ont pour fonction d'éliminer ces situations pathogènes.
=>
Utiliser un outil qui vérifie automatiquement certaines propriétés,
ou calcule certains paramètres.

Exple sur un document de spécification :
vérification du plan,
construction d'un index (tous les concepts sont-ils définis, une seule fois, ...),
vérification de la syntaxe, de l'orthographe.

Exple sur le code :
taux de commentaire,
mise en page,
graphe de contrôle,
=> complexité de l'algorithme,
initialisation des variables,
tx d'utilisation de chaque variable en partie gauche / droite,
graphe d'appel entre procédures
... ...

Simulation

Certains formalismes de spécification sont exécutables.
Analyse dynamique : étudier le résultat de l'exécution des spécifications.
Cf. le prototypage.

IV. Les Tests

Chercher à provoquer des pannes, pour :
- trouver les défauts présents dans le code,
- mesurer la fiabilité du code.

L'effort de test croit exponentiellement avec la complexité de l'entité testée;
=> on cherche à tester peu de chose à la fois, et procède de façon incrémentale.

1) Les Types de tests

La qualification du logiciel, recette

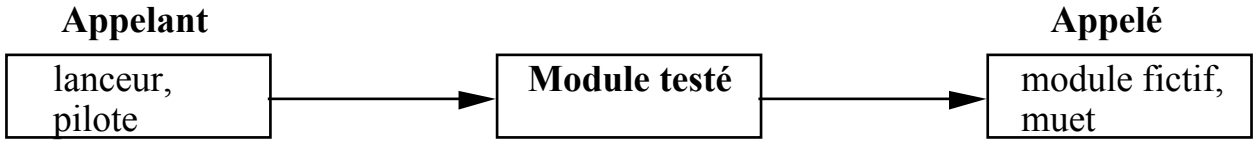
Tests du système réalisés sur site chez le client, dans les conditions normales d'utilisation.
Validation de l'analyse des besoins, de la conformité du logiciel aux besoins effectifs.

Le Test système,

Test du système, chez le développeur
Validation des spécifications.

Les Tests unitaires

On teste un seul module, isolément.



Validation du codage des modules.

Le codage des **pilotes** et **modules fictifs** fait partie des «échafaudages logiciels» non livrés

Les modules d'interface (avec l'utilisateur, l'environnement système, d'autres applications) sont difficiles à simuler => on ne les code pas de façon fictive.

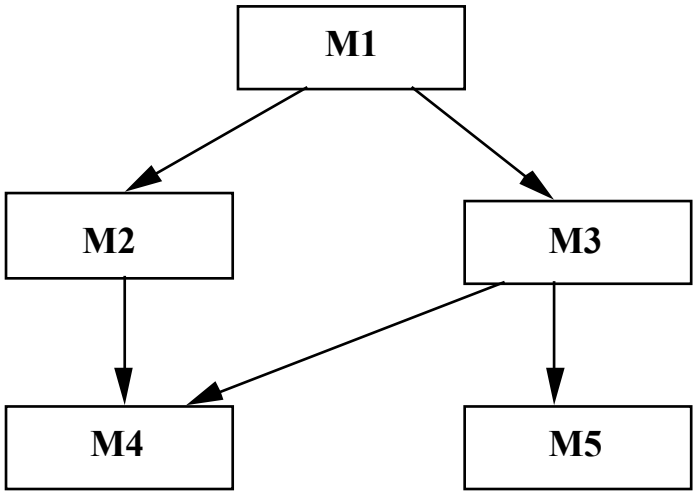
Les Tests de non-régression

Après chaque correction, pour vérifier que l'on n'a pas introduit de nouveaux défauts.

Les Tests d'intégration

Une fois que l'on est assuré que chaque module fonctionne bien,
on s'assure qu'il en est de même de leur composition.
Validation de la conception du logiciel.

Réalisé incrémentalement : on ne teste qu'une seule intégration à la fois
en se basant sur le graphe d'appel entre les modules



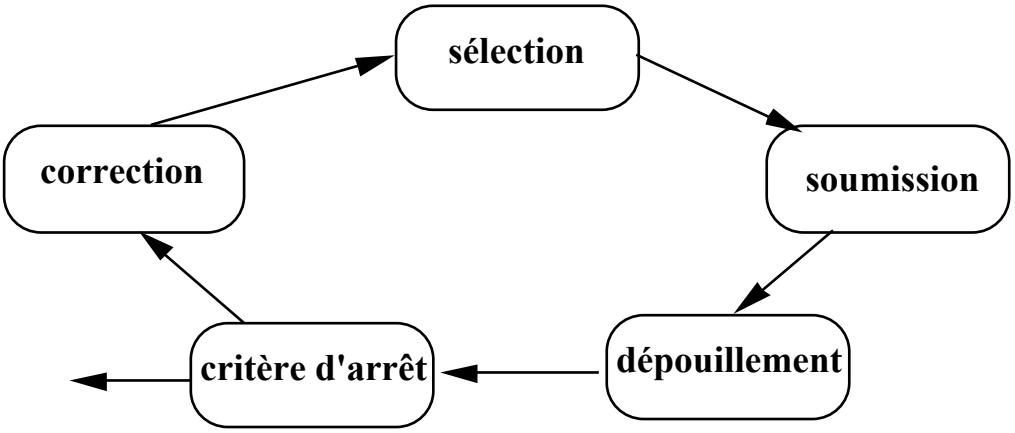
Stratégie descendante
On commence par les modules appelants, pour terminer par les feuilles
tests unitaires de M1, M2, M3, M4 et M5
M1 + M2
M1 + M2 + M3 //on procède en largeur
M1 + M2 + M3 + M4
M1 + M2 + M3 + M4 + M5

Stratégie ascendante
On commence par les feuilles
tests unitaires de M1, M2, M3, M4 et M5
M5 + M3
M5 + M3 + M4
M5 + M3 + M4 + M2
M5 + M3 + M4 + M2 + M1

Stratégie ascendante paresseuse
tests unitaires de M4 et M5
M5 + M3 // test simultané de M5 et de son intégration avec M3
M5 + M3 + M4
M5 + M3 + M4 + M2
M5 + M3 + M4 + M2 + M1

Remarques
La récursivité croisée interdit les tests unitaires.
L'effort de test est multiplié par le nombre de versions de chaque module
(Cf. la Gestion de Configuration).

2) Le processus de test



sélection
Déterminer le jeu de tests, ou jeu d'essais.
un essai fixe
la valeur des entrées,
les conditions d'exécution (pour la détection des pannes opérationnelles).
Les essais sont choisis en fonction de ce que l'on cherche à tester.

soumission
exécuter chacun des essais.

dépouillement
Examiner le résultat de chacun des essais, et déterminer les pannes.
pannes d'exécution : facilement observable;
pannes fonctionnelles : comparer les résultats observés avec ceux attendus
il faut un **oracle**, indiquant ce que doit être le résultat;
pannes opérationnelles :
observer le comportement interne du logiciel,
déterminer s'il est conforme au comportement attendu;
Un même essai peut donner lieu à plusieurs pannes.

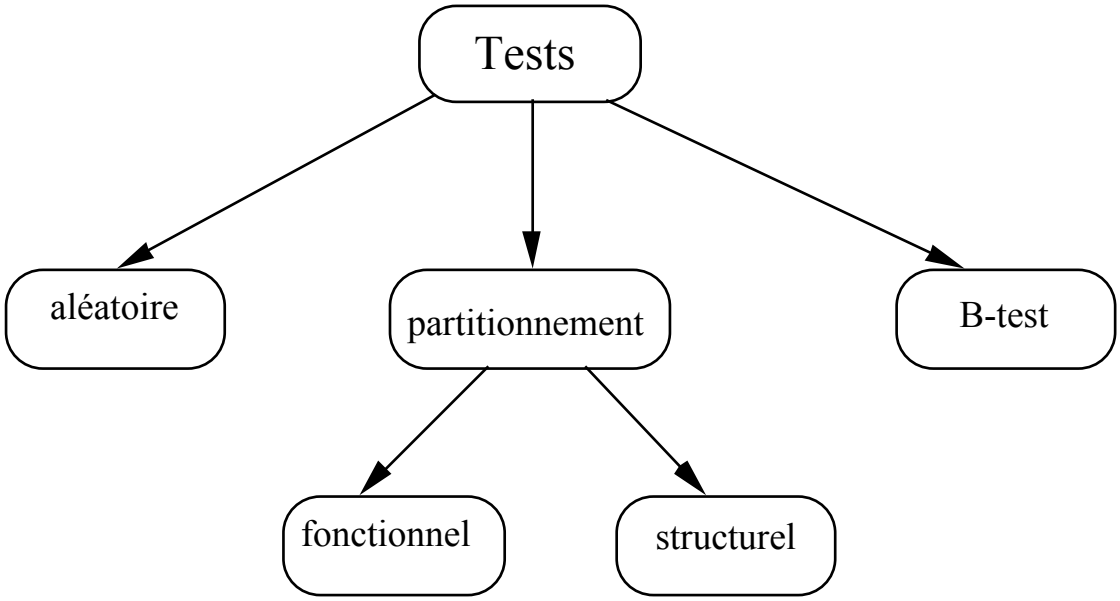
analyse
Etudier chaque panne, identifier le (ou les) défaut qui en est la cause.

critère d'arrêt
la mesure de la fiabilité obtenue,
le tx de couverture de test,
l'urgence des délais ...

Les **environnements de test**
automatisent ± la sélection des jeux d'essais et leur soumission.

3) La sélection des jeux d'essais

Principe : sélectionner les essais les plus susceptibles de provoquer une panne,
compte tenu des essais précédemment réalisés.
Les différentes **stratégies de test** sont complémentaires.



Tests aléatoires

1. l'essai est choisi aléatoirement
Aucun biais du fait de la connaissance de
ce que doit faire le logiciel ,
comment il est fait,
comment il doit être utilisé.
Test du singe, pour la robustesse.

2. introduction aléatoire de défauts
En mesurant le taux de pannes provoquées par ces défauts,
on évalue le nombre de défauts restants.

β-tests

Utilisation en vraie grandeur par quelques utilisateurs pilotes.
Pour la validation du logiciel,
c'est le seul moyen de détecter les pannes rares.

Partitionnement

On partitionne le domaine D de tous les essais possibles en classes d'équivalence,

$$D = \mathbf{U}_{i=1..n}\ D_i$$

de telle sorte que

$$\forall\ i = 1...n,\ \forall\ e \in D_i,\ \forall\ e' \in D_i\ (e \text{ positif} \Rightarrow e' \text{ positif}).$$

Dans chaque classe d'équivalence, si un essai est positif alors ils le sont tous.
Jeu d'essai complet : un essai dans chaque classe.

Tests fonctionnels, ou boîte noire

La détermination des classes d'équivalence est basée sur la spécification (fonctionnelle ou opérationnelle), sans tenir compte du codage.
On teste chacun des « cas » que fait apparaître la spécification.

Exple pour une fonction de recherche d'un élément dans une liste :
la liste est vide, contient 1, 2, nbmax éléments;
l'élément ne figure pas (inférieur au plus petit, supérieur au plus grand, dans un trou);
l'élément est le premier, le dernier, au milieu,
l'élément est le plus petit, le plus grand, au milieu,
l'élément figure plusieurs fois.

Exple pour logiciel multi-utilisateurs :
1, 2, nbmax, nbmax + N utilisateurs,
ordonnancement des connections / déconnections (queue, pile, aléatoire).

Tests structurels, ou boîte blanche

On teste chacune des séquences d'instructions logiquement possibles.

Tests unitaires basés sur le **graphe de contrôle** du module :
toutes les instructions : exécuter au moins 1 fois chaque instruction,
tous les chemins : exécuter toutes les combinaisons de branchement
(N branchements => 2^N chemins : l'effort de test croît exponentiellement avec la complexité),
toutes les boucles : passer 0, 1, 2, N fois dans chaque boucle

Tests unitaires basés sur le **flux de données** du module :
On construit l'hyper-graphe indiquant toutes les façons de dériver les résultats à partir des entrées, et on essaye tous les parcours possibles.

Tests d'intégration basés sur le graphe d'appel des procédures.

	aléatoire	fonctionnel	structurel	β-test
T. unitaires	+	++	++	
T. d'intégration	+	++	++	
T. système	+	++		+++

En quelles entités peut-on avoir confiance, Comment fonder sa confiance ?

Au bout du compte, la confiance repose toujours sur une « intime conviction ».

1. Qq chose de «simple», que l'on comprend parfaitement (évidence cartésienne).
=> Etudier, chercher à comprendre, analyse statique

2. Se ramener à, comparer avec qqch en lequel on a déjà confiance,
montrer que c'est en tout point équivalent.
de façon informelle : analogie,
de façon formelle : prouver l'équivalence
(suppose que l'on ait aussi confiance dans la preuve).
Repose sur le principe : si X a confiance en A et A équivalent à B, X a confiance en B.
=> développement par transformation

3. Expérimenter : faire fonctionner, utiliser,
constater dans toutes les situations possibles qu'il n'y a pas de dysfonctionnement;
test : expérimenter le système lui-même,
simulation : expérimenter un modèle, une maquette, un prototype du système.
Repose sur la reproductibilité des phénomènes, le déterminisme :
si un essai est positif, toute utilisation dans les mêmes conditions sera positive.
=> tester

4. Examiner le processus de production, et non le système lui-même,
pour avoir confiance en la façon dont le système a été fait (ISO 9 000).
Repose sur le principe : la qualité du processus est garante de la qualité du produit.
=> assurance / contrôle qualité

5. La redondance : utiliser plusieurs systèmes
le recoupement : les faire fonctionner en parallèle,
le rechange : si l'un devient défaillant, on utilise l'autre.
Repose sur le principe : si les systèmes sont indépendants, leurs défaillances aussi,
la confiance croit exponentiellement avec le nombre de systèmes.
=> programmation tolérante aux pannes

6. Se reposer sur la confiance d'autres personnes (en qui on a confiance)
=> réutiliser