

Le test de logiciels

Yves Le Traon

Plan du cours 1

- Introduction: importance du test et positionnement dans un contexte de GL
 - Hiérarchisation des tests
 - Etapes de test
- Le test statique
- Le test dynamique
 - techniques élémentaires de test fonctionnel
 - l'analyse partitionnelle
 - le test aux limites
 - les graphes cause-effet
 - test structurel unitaire
- Test d'intégration: stratégies de base

De la difficulté de la validation intra-composant: Programming-in-the small

```
Acquérir une valeur positive n
Tant que n > 1 faire
  si n est pair
    alors n := n / 2
  sinon n := 3n+1
Sonner alarme;
```

- Prouver que l'alarme est sonnée pour tout n?
- Indécidabilité de certaines propriétés
 - problème de l'arrêt de la machine de Turing...

• Recours au test

- ici, si machine 32 bits, $2^{31} = 10^{10}$ cas de tests
- **5 lignes de code => 10 milliards de valeurs !**

Programming-in-the-Large

- | | |
|--|---------------|
| • Compilateur | 10 KLOC (C) |
| • Système de commutation X25... | 100 KLOC (C) |
| • Contrôle de sous-marin nucléaire ... | 1 MLOC (ADA) |
| • Station spatiale | 10 MLOC (ADA) |

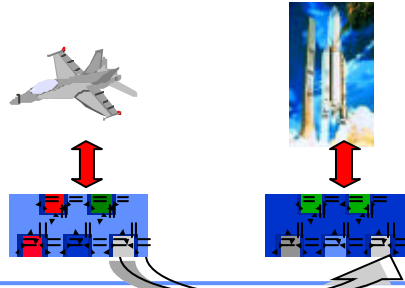
Programming-in-the-Large

Gestion de la complexité due à la taille
(différence entre le bricoleur et l'architecte)

- Méthodes basées sur la modularité (SADT, UML)
- Gestion des ressources (matérielles, humaines) et des synchronisation de tâches

- => multiplication des risques d'erreur
- => impossibilité d'obtenir des certitudes
- => "se convaincre que", notion de "confiance"

Validité inter-composants : Peut-on (ré)-utiliser un composant?



Ex: Ariane 501 Vol de qualification Kourou, ELA3 – 4 Juin 1996,12:34 UT

- H0 -> H0+37s : nominal
- Dans SRI 2:
 - BH (Bias Horizontal) > 2°15
 - convert_double_to_int(BH) fails!
 - exception SRI1 -> crash SRI2 & 1
- OBC disoriented
 - Angle attaque > 20°,
 - charges aérodynamiques élevées
 - Séparation des boosters



Ariane 501 : Vol de qualification Kourou, ELA3 – 4 Juin 1996,12:34 UT

- H0 + 39s: auto-destruction (coût: 500M€)



Pourquoi ? (cf. IEEE Comp. 01/97)

- Pas une erreur de programmation
 - Non-protection de la conversion = décision de conception ~1980
- Pas une erreur de conception
 - Décision justifiée vs. trajectoire Ariane 4 et contraintes TR
- Problème au niveau du test d'intégration
 - As always, could have theoretically been caught. But huge test space vs. limited resources

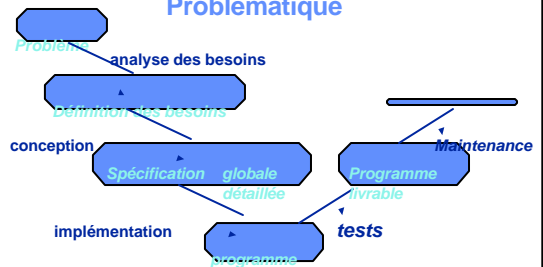
Pourquoi? (cf. IEEE Computer 01/97)

- Réutilisation dans Ariane 5 d'un composant de Ariane 4 ayant une contrainte « cachée » !
 - Restriction du domaine de définition
 - Précondition : $abs(BH) < 32768.0$
 - Valide pour Ariane 4, mais plus pour Ariane 5
- Pour la réutilisation, pour une architecture répartie =>
 - Spécification = **contrat** entre un composant et ses clients

La maintenance : Programming-in-the-Duration

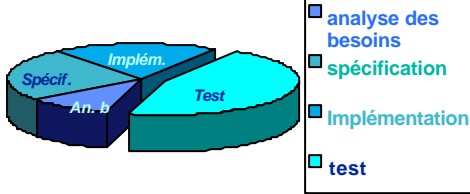
- Etalement sur 10 ans ou plus d'une "ligne de produits"
- Age moyen d'un système : 7 ans
- 26% des systèmes ont plus de 10 ans (Cf. Application bancaire et Cobol)

Problématique



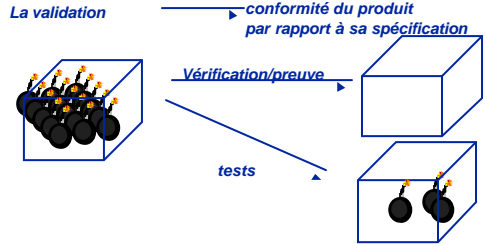
Problématique

Le coût du test dans le développement



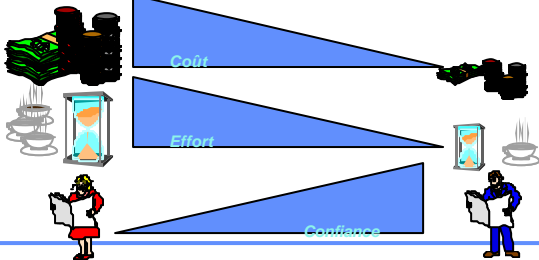
+ maintenance = 80 % du coût global de développement !!!

Problématique: une définition !



Problématique

Pourquoi ?



Vocabulaire

Testabilité	faute		
	erreur	bogue	Fiabilité (Reliability)
Test de Robustesse		défaillance	
			Test statique
Tolérance aux fautes	Séquence de test		Test de non-régression
	Données de test		
Sûreté de fonctionnement (Dependability)	Jeu de test		Test statistique
	Cas de test		

Problématique

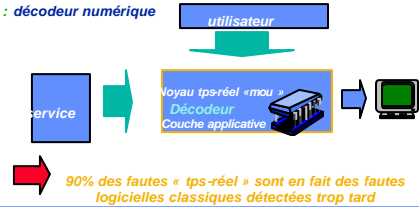
- Objectifs du test
 - examiner ou exécuter un programme dans le but d'y trouver des erreurs
 - éventuellement mise à l'épreuve de :
 - la robustesse (test hors bornes/domaine)
 - des performances (test en charge)
 - de conformité (protocoles normalisés)
 - de propriétés de sûreté

Hierarchisation des tests

But : séparer les plans

Typiquement confusion entre fautes hardware et software
fautes fonctionnelles/structurelles
faute temps-réel / fautes classiques

Exemple : décodeur numérique



Hierarchisation des tests

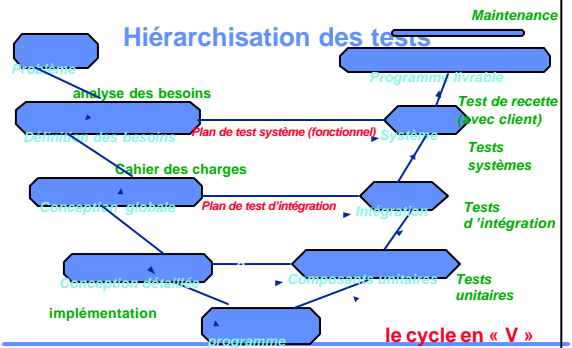
But : séparer les plans

Risques :
perte d'information d'un plan à l'autre

*Mauvaise compréhension des fonctions à réaliser
Introduction de défauts de conception
Tendance à repousser les problèmes aux niveaux inférieurs*

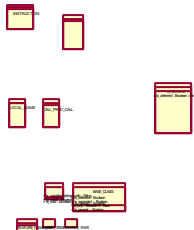
Mais on n'a encore rien trouvé de mieux ...
à moins que l'objet n'offre des solutions

Hierarchisation des tests



Niveau unitaire

- Unit Level Components
 - => implement specified functions
 - how can you trust them ?
 - How using/reusing them safely ? "off-the-shelf"

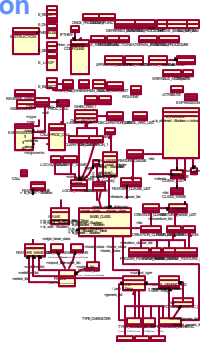


A component has
 → a stability
 → an identified interface

Intégration

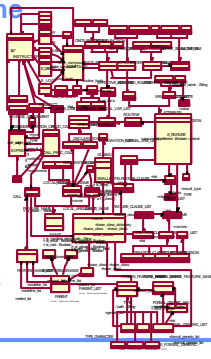
- Integration/Evolution
 - How integrating them safely ?
 - How optimizing integration?

- A Transition Step



Test système

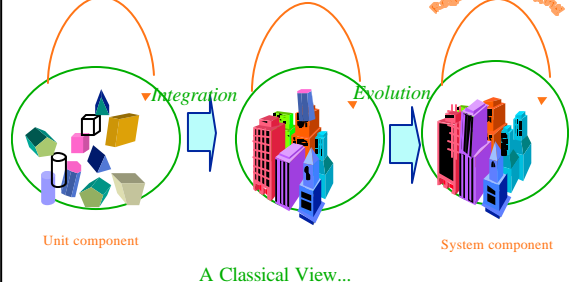
- System
 - = a coherent components set
 - => implement specified functions
 - => a more elaborated component



A finalized system has
 → a stability
 → an identified interface

System = Component

Des composants au système?



Hiérarchisation des tests

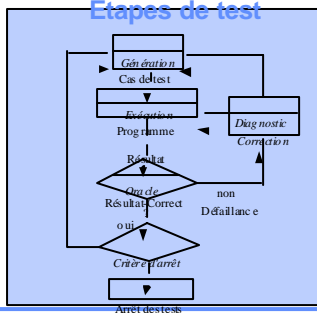
Développements spécifiques au test

- Test unitaire : drivers (lanceur des tests), oracle (succès/échec), instrumentation (mesure couverture)
- Test intégration: idem + "bouchons" de tests (stubs), pour simuler les modules non disponibles
- Test système : test des fonctions + environnement matériel + performances.

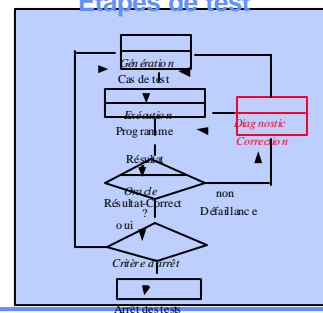
Conditions (théorique) de possibilité du test

- H1 : programmeur "compétent"
 $\text{Réalisé} = \delta(\text{Idéal})$
- H2 : Axiome de couplage (optionel)
 anomalies complexes = combinaison d'anomalies simples

Étapes de test



Étapes de test



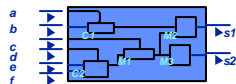
Etapes de test

☐ Test fonctionnel



Etapes de test

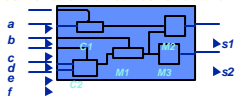
☐ Test fonctionnel ou test structurel



☐ Génération aléatoire ou déterministe

Etapes de test

☐ Test fonctionnel ou test structurel

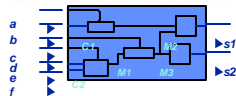


☐ Génération aléatoire ou déterministe

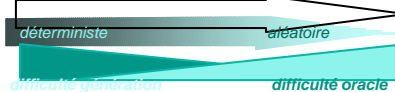


Etapes de test

☐ Test fonctionnel ou test structurel



☐ Génération aléatoire ou déterministe



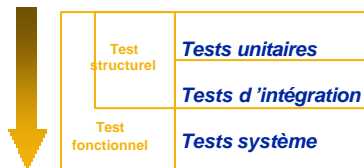
Etapes de test

☞ **Test fonctionnel**
 (« black-box », boîte noire) *Indépendant de l'implémentation*
Planifier à partir de la spécification fonctionnelle
Réutilisables

☞ **Test structurel**
 (« glass box », boîte blanche ou « de verre ») *Dépend de l'implémentation*



Etapes et hiérarchisation des tests



Le test en général

Problème : Génération des cas de test

- trouver les données efficaces pour révéler les fautes
- minimiser le nombre des données de test

Exemple :

générateur aléatoire
 une addition sur 32 bits
 un ensemble

Le test en général

Méthodes de génération des tests :

1) Génération déterministe

Génération « à la main » des cas de test et des oracles

Deux critères :

- couvrir une portion précise du logiciel (critère structurel)
- couvrir les fonctions du logiciel (critère fonctionnel)

Pas de méthode miracle

Avantage ? : nécessitant une forte implication humaine

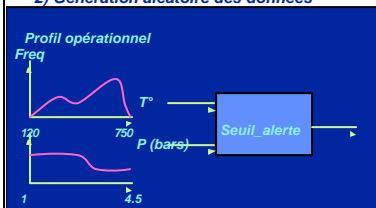


les tests plus efficaces ?

Le test en général

Méthodes de génération des tests :

2) Génération aléatoire des données



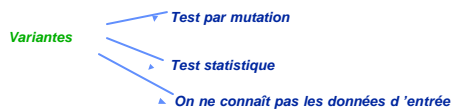
Performances

Crash ou grosses défaillances

Le test en général

Méthodes de génération des tests :

2) Génération aléatoire des données



Exemples : Le flot d'entrée est fourni par le client
 Le flot d'entrée est obtenu via une antenne etc.

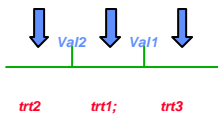
Le test en général

2) Génération guidée par les contraintes (analyse symbolique)

```

Procedure lissage(in: integer; var out :integer)
begin
  if in < Val1 then
    begin
      if in > Val2 then out := trt1;
      out:=trt2;
    end
  else out:=trt3;

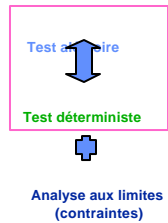
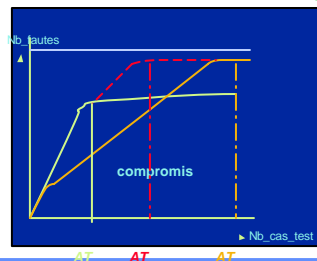
```



Contraintes rapidement trop complexes
 Uniquement test structurel (en s'accès au code)

Le test en général

Limites du test aléatoire : couvre toujours les mêmes cas



Le test en général

Problèmes : exécution des tests

- environnement de développement « propre »
- environnement de test
- debugger (points d'arrêt, suivi de l'état des données du programme sous test)
- instrumentation automatique du code/outils d'observation (couverture etc.) -> logiscope & Co.

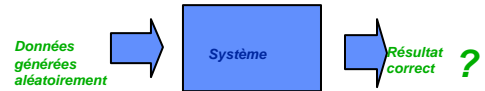
! on ne teste que rarement le système tel qu'il sera chez le client (partie simulée, systèmes embarqués, parties non-terminées ou sous traitées ailleurs ...) -> environnement de simulation (potentiellement bogué)

Le test en général

Problèmes : oracle (ou verdict)

Oracle : expression booléenne caractérisant la détection d'une erreur
 Généralement = (résultat_attendu = résultat_obtenu)
 Ou bien : levée d'une exception

Exemple : génération aléatoire des données de test



Avant le test... mieux vaut prévenir que guérir

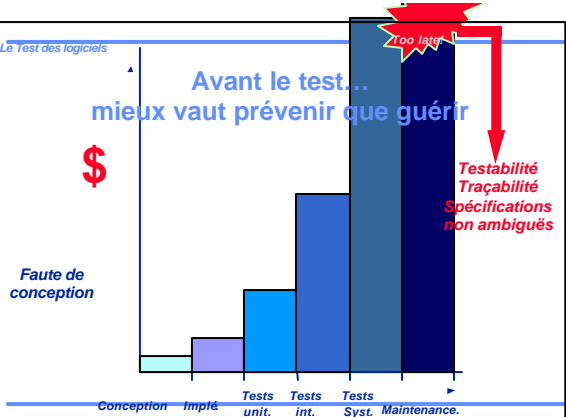
Un principe : plus une faute est détectée tard, plus elle coûte cher

Moralité : Mieux vaut prévenir que guérir

2 moyens principaux

- a) Analyse de testabilité
- b) « test statique »

Avant le test... mieux vaut prévenir que guérir



Test statique

Techniques de « test statique »

Définition : ne requiert pas l'exécution du logiciel sous-test sur des données réelles

réunions de 4 personnes environ pour inspecter le code

1 modérateur, le programmeur, le concepteur et 1 inspecteur



Test statique

Préparation : Distribution des documents quelques jours avant la séance (code, spec, éventuellement cas de test prévus)

Durée : 1h30 à 2h max

But : le programmeur lit et explique son programme
le concepteur et l'inspecteur apportent leur expertise
les fautes sont listées
(pas corrigées-> à la charge du programmeur)

Test statique

Efficacité : plus de 50 % de l'ensemble des fautes d'un projet sont détectées lors des inspections si il y en a (en moyenne plus de 75%)

Défaut : mise en place lourde, nécessité de lien transversaux entre équipes, risques de tension...tâche plutôt fastidieuse

Test statique

- Règles
 - être méthodique (cf. transparents suivants)
 - un critère : le programme peut-il être repris par quelqu'un qui ne l'a pas fait
 - un second critère : les algorithmes/l'architecture de contrôle apparaît-elle clairement ?
 - > décortiquer chaque algo et noter toute redondance curieuse (coller) et toute discontinuité lorsqu'il y a symétrie (ce qui peut révéler une modif incomplète du programme)

Test statique

Exemple: vérification de la clarté (procédural)
R1 : Détermination des paramètres globaux et de leur impact sur les fonctions propres

```

program recherche_tricho;
uses crt;
const
  max_elt = 50;
  choix1 = 1;
  choix2 = 2;
  fin = 3;
type
  Tliste = array[1..max_elt] of integer;
var
  liste : Tliste;
  taille, choix, val : integer;
  complex : integer;

```

But du programme non exprimé
Manque de commentaires
Identificateurs non explicites

Test statique

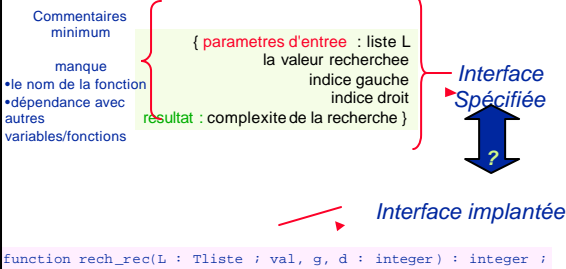
Exemple: vérification de la clarté
R2 : Existence d'un entête clair pour chaque fonction

```

{-----
Recherche recursive d'une valeur dans une liste trie
-----}

```

Test statique: pour chaque fonction



```
function rech_rec(L : Tliste ; val, g, d : integer) : integer ;
```

Test statique

```

var i, pt, dt : integer;
begin
  affiche(L, g, d);
  if g<d
  then
    begin
      pt :=g+(d-g) div 3;
      if val > L[pt]
      then
        begin
          dt := (pt+1+d) div 2;
          if val > L[pt]
          then rech_rec:=2+rech_rec(L, val, dt+1, d)
          else rech_rec:=2+rech_rec(L, val, pt+1, dt)
          end
        else rech_rec:=1+rech_rec(L, val, g, pt)
        end
      else rech_rec:=0
    end; { rech_rec }

```

Test statique

Autre méthodes :

- + revues,
- + métriques (contraintes etc.),
- + analyse d'anomalies
(du graphe de contrôle/de données),
- + règles (de bon sens!)
 - identificateurs clairs,
 - découpage fonctionnel « naturel »
 - limiter les effets de bords (interfaces, variables globales)
- + preuves d'algorithmes,
- + exécution symbolique,
- + simulation de modèles, etc.

Test dynamique : Vocabulaire

- DT : Donnée de Test = une (séquence) d'entrée(s)
- JT = {DT}
- D = domaine d'entrée du programme P sous test
- Soient F, les fonctions spécifiées du logiciel
 $F : D \rightarrow \text{Sorties}$

Le problème du test dynamique est :

$$D \xrightarrow{F=P?} \text{Sorties}$$

Pratiquement : $T \subset D : t \in T \rightarrow P(t) = F(t)$

Test dynamique

- DEUX REGLES :
 - Conserver les tests déjà écrits
 - Conserver les réponses correctes correspondantes

Test dynamique

Exemple simplissime :

exe < fichier_testi

- quand correct : exe < fichier_testi > res_testi
- Lorsque intégration : Driver de test existe déjà
- Lorsque évolution : Tests de non-régression
 newexe < fichier_testi > res_test_newexe
 diff res_test_newexe



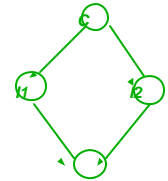
C'est plus simple en OO (classes autotestables)

Test dynamique structurel

“Testing can prove the presence of bugs, but never their absence” (Dijkstra)

Le test structurel: abstraire pour obtenir un critère formel

si C alors I1
sinon I2



Le test unitaire structurel

- Graphe de Contrôle (langages procéduraux/actionnels)
 - But : représenter tous les chemins d'exécution potentiels
 - Graphe orienté : avec un noeud d'entrée E et un noeud de sortie S (noeud fictif ajouté si plusieurs sorties)
 - Sommets :
 - blocs élémentaires du programme, ou prédicats des conditionnelles /boucles, ou noeuds de jonction "vide" associé à un noeud prédicat*
 - Bloc élémentaire : séquence maximale d'instructions séquentielles*
 - Arcs :
 - enchaînement d'exécution possibles entre deux sommets*

Le test unitaire structurel

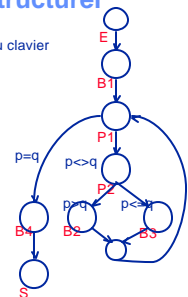
Exemple : PGCD de 2 nombres

Précondition: p et q entiers naturels positifs lus au clavier

Effet : result = pgcd(p, q)

En pseudo-langage

```
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
```



Le test unitaire structurel

- Question : Sur ce modèle imaginer un critère de couverture
 - minimal
 - maximal

Le test unitaire structurel

- Chemins : suite d'arcs rencontrés dans le graphe, en partant de E et finissant en S.
 - en général représenté sans perte d'information par une liste de sommets
 - Prédicat de chemin : conjonction des prédicats (ou de leur négation) rencontrés le long du chemin.
 - Pas toujours calculable
- $E B1 P1 P2 B2 S : p_1=q_1 \ \& \ p_2 > q_2$ ($p_i = i^{\text{ème}}$ valeur de p)
 $E B1 P1 P2 B3 S : p_1=q_1 \ \& \ p_2 \leq q_2$ ($p_i = i^{\text{ème}}$ valeur de p)
 $E B1 P1 (P2 B2)^n S : p_1=q_1 \ \& \ (p_i > q_i, i \in \overline{1, n})$
- ➡ Chemins infaisables : problème en général indécidable

Le test unitaire structurel

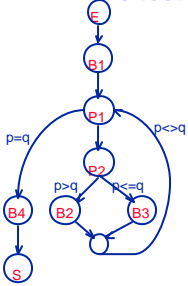
Sélection des tests fondés sur le flot de contrôle

- Couverture des instructions : chaque bloc doit être exécuté au moins une fois
- Couverture des arcs ou enchaînements
- tous les chemins élémentaires ou 1-chemins
- tous les i -chemins : de 0 à i passages dans les boucles
- tous les chemins : si boucles, potentiellement infinis

Le test unitaire structurel

- Question : différence entre couverture arcs et couverture instructions ?
- Faire Exemple

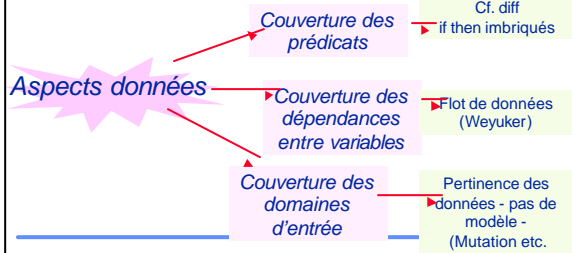
Le test unitaire structurel



- 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)

Test unitaire structurel

- Questions : qu'est-ce que ne capture pas ce modèle ?



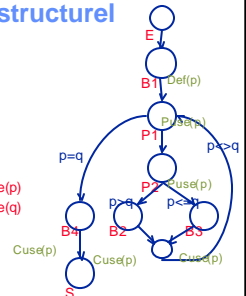
Le test unitaire structurel

- Graphe de Flot de Données (Weyuker)
 - But : représenter les dépendances entre les données du programme dans le flot d'exécution.
 - Graphe de contrôle **décoré** d'informations sur les données (variables) du programme.
 - Sommets : idem GC
 - une définition (= affectation) d'une variable v est notée def(v)
 - une utilisation d'une variable est v notée P_use(v) dans un prédicat et C_use(v) dans un calcul.

Exemple Le test unitaire structurel

```

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



Le test en général: critères d'arrêt unitaire : flot de données

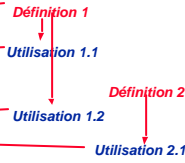
Autres critères structurels : lien definition-utilisation (Weyuker)

Program pilote_automatique

```

var
  probleme: boolean;
  direction : integer in [1..360]
begin
  ...
  saisir_direction direction);
  ...
  probleme := false
  ...
  while not probleme do
  begin piloter(avion, auto, direction
  ...
  if capteur_pression < 120 then probleme := true
  ...
  if probleme then traiter_probleme(capteur_pression)
  ...
end

```

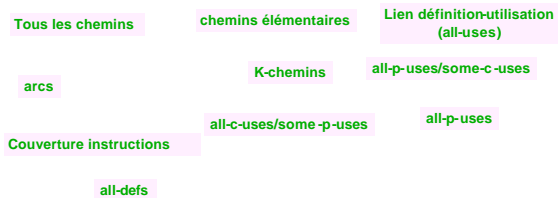


Test structurel : relation d'implication entre critères

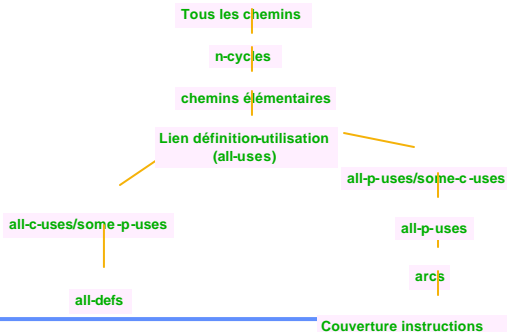
- Définition : $C1 \Rightarrow C2$ ("subsumes") ssi $\forall P, \forall JT$ satisfaisant $C1$ on a JT satisfait $C2$

Test structurel : relation d'implication entre critères

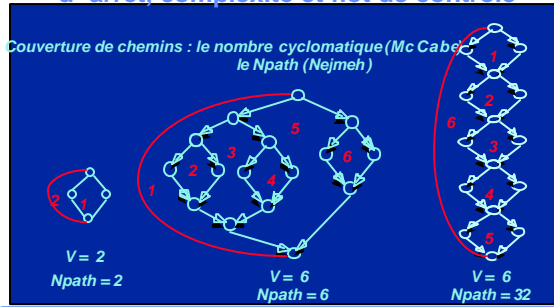
- Question : ordonner les critères selon cette définition



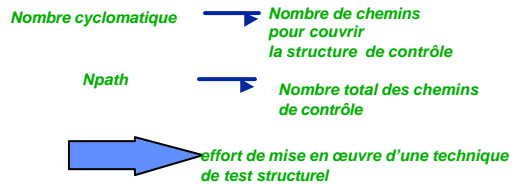
Le test unitaire structurel: classement



Le test unitaire structurel: critère d'arrêt, complexité et flot de contrôle

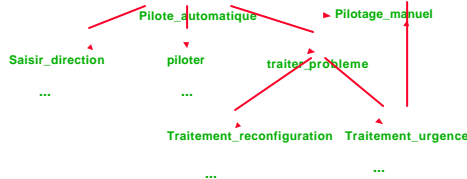


Le test en général: critères d'arrêt



...vers le test structurel d'intégration

Couverture du graphe d'appel



Le test d'intégration

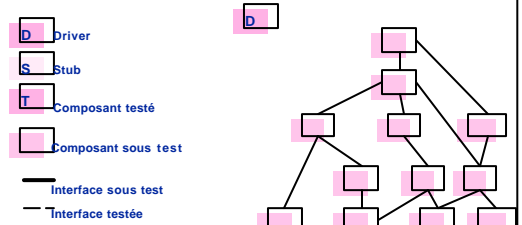
- But : vérifier les interactions entre composants (se base sur l'architecture de la conception)
- Difficultés principales de l'intégration
 - interfaces floues
 - implantation non conforme au modèle architectural (dépendances entre composants non spécifiées)
 - réutilisation de composants (risque d'utilisation hors domaine)

Le test d'intégration

- Stratégies possibles (si architecture sans cycles)
 - big-bang : tout est testé ensemble. Peu recommandé !
 - top-down : de haut en bas. Peu courant. Ecriture uniquement de drivers de tests.
 - bottom-up : la plus classique. On intègre depuis les feuilles.

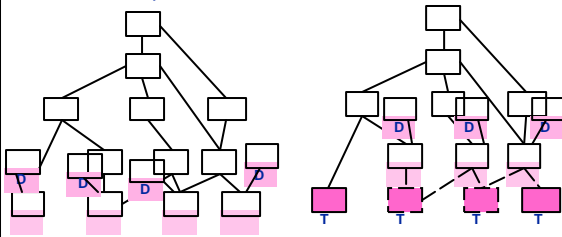
Le test d'intégration

- 3 stratégies: bottom-up, Top-down, Big-Bang.

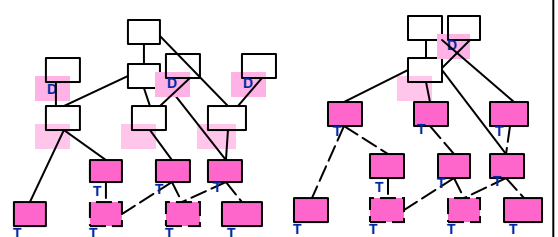


Le test d'intégration

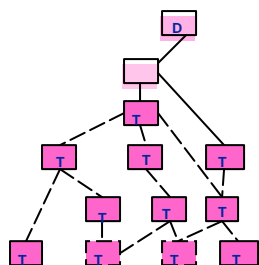
- Bottom-up



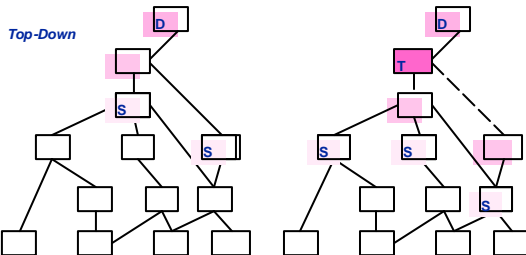
Le test d'intégration



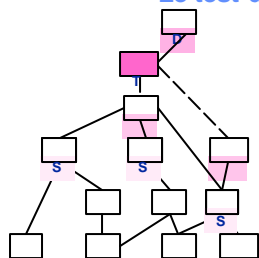
Le test d'intégration



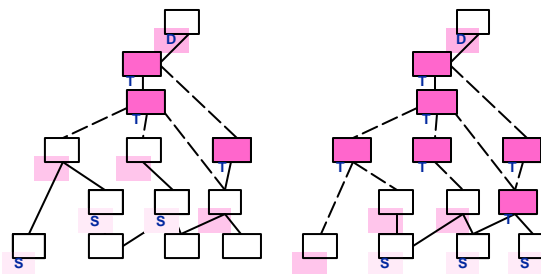
Le test d'intégration



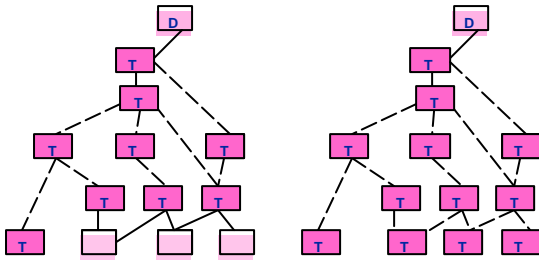
Le test d'intégration



Le test d'intégration



Le test d'intégration



Le test d'intégration

- intégration Big-Bang
- intégration Top-down
- intégration bottom-up
- intégration backbone
- intégration par domaines de collaboration