

Test et Maintenance de Logiciel

Ileana Ober

Maître de conférences

Université Paul Sabatier

IRIT

<http://www.irit.fr/~Ileana.Ober/>



Année Universitaire 2012-2013

©Ileana Ober

Plan du cours

- ❖ test et spécification des exigences
- ❖ classifications du test
- ❖ processus du test

Spécification des exigences

- ❖ Types d'exigences
 - ❖ fonctionnelle
 - ❖ non-fonctionnelle (durée d'exécution, contraintes matérielles, contraintes physiques, etc.)
 - ❖ contraintes
 - ❖ de processus ou de gestion
- ❖ Définitions
- ❖ Contexte

Spécifications d'exigences

- ❖ Chaque exigence doit être
 - ❖ atomique
 - ❖ réutilisable
 - ❖ testable
 - ❖ non-ambiguë, précise et complète
 - ❖ de haut niveau (ne pas induire des choix de programmation)fonctionnelle
 - ❖ non-fonctionnelle (durée d'exécution, contraintes matérielles, contraintes physiques, etc.)
 - ❖ contraintes
 - ❖ de processus ou de gestion
- ❖ Le cahier de charges est composé d'un ensemble d'exigences
- ❖ Le cahier de charges doit contenir des exigences
 - ❖ non contradictoires
 - ❖ disjointes
 - ❖ non-dupliquées
 - ❖ traçables, complètes et légitimes
 - ❖ utilisant la même terminologie

Spécification des exigences

- ✿ rôle important dans la conception (sinon cette activité est sans objet)
- ✿ rôle important dans le test
- ✿ on ne peut pas faire du test avant que le cahier de charges soit complètement spécifié

Plan du cours

- ❖ test et spécification des exigences
- ❖ classifications du test
- ❖ processus du test

Classification des tests

- ✿ Portée:
 - ✿ unitaire
 - ✿ d'intégration
 - ✿ de validation
 - ✿ de recette
 - ✿ système
 - ✿ de (non) régression
- ✿ Exécution du code
 - ✿ statique
 - ✿ dynamique
- ✿ Accès au code
 - ✿ boîte blanche
 - ✿ boîte noire
- ✿ Structurel, aléatoire, fonctionnel, de mutation

Test unitaire

- ❖ Caractéristiques
 - ❖ tester les différents modules en isolation
 - ❖ vérification d'un composant logiciel (procédures, classes, packages, composants, etc.)
 - ❖ fondé sur la conception détaillée
 - ❖ uniquement test de correction fonctionnelle
- ❖ But
 - ❖ détection précoce des anomalies liées au codage
- ❖ Particularités
 - ❖ détection précoce des anomalies liées au codage
 - ❖ matériel de test important : lanceurs, modules muets



Test d'intégration

- ✿ **Caractéristiques**
 - ✿ tester le bon comportement lors de la composition des modules
 - ✿ vérification d'un sous-ensemble de composants du logiciel
 - ✿ fondé sur la conception globale
- ✿ **But**
 - ✿ détection des anomalies d'interface
- ✿ **Particularités**
 - ✿ détection précoce des anomalies liées au codage
 - ✿ matériel de test important : lanceurs, modules muets



Test de validation

- ❖ **Caractéristiques**
 - ❖ valider l'adéquation au cahier de charges
 - ❖ tester le bon comportement de l'ensemble par rapport à la spécification
 - ❖ vérification de l'ensemble des composants du logiciel
 - ❖ fondé sur la spécification des besoins logiciels et sur la spécification technique des besoins

- ❖ **But**
 - ❖ vérification de la conformité des résultats et du respect des contraintes
 - ❖ préparation de la recette



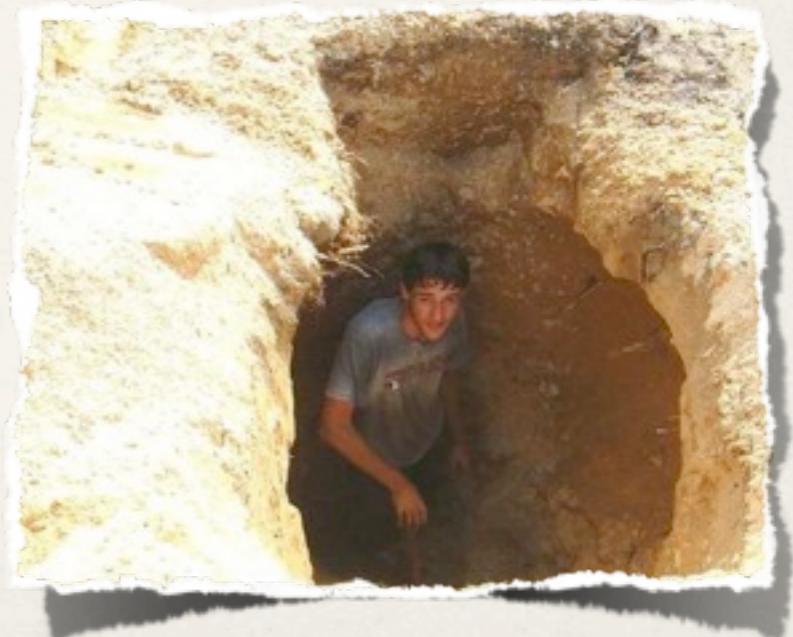
Test de recette

- ✿ **Caractéristiques**

- ✿ vérification du logiciel
- ✿ fondé sur la spécification technique des besoins
- ✿ sous-ensemble, ensemble ou sur-ensemble des tests de validation
- ✿ Souvent effectué par le client

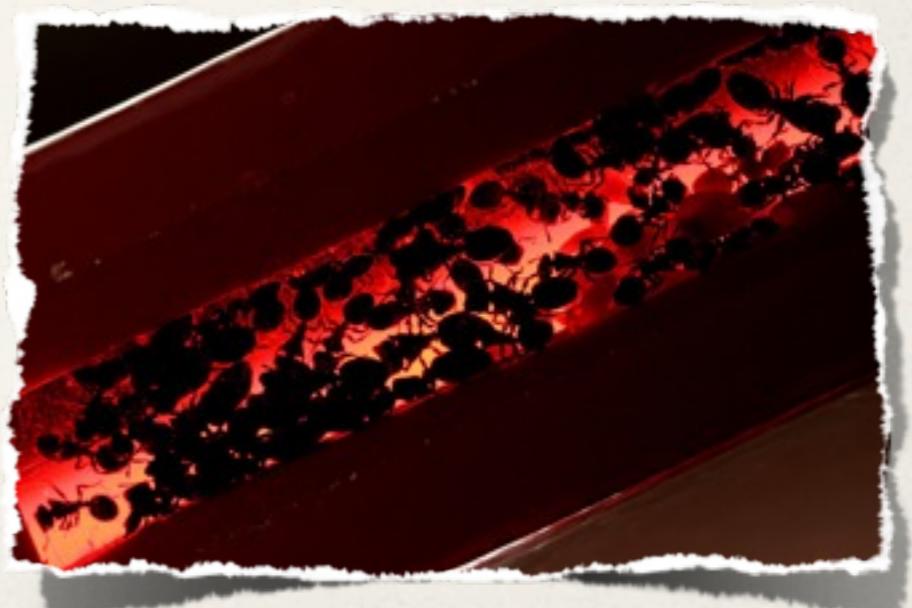
- ✿ **But**

- ✿ Vérifier que le système
 - ✿ Satisfait les demandes du client
 - ✿ Est prêt à être utilisé



Test de système

- ❖ Caractéristiques
 - ❖ valider le bon fonctionnement du code dans l' environnement où il va être exécuté
- ❖ But
 - ❖ Vérifier que le système satisfait la spéc (aspects fonctionnels et globaux)
 - ❖ Tester toutes les caractéristiques émergentes (sécurité, performances, robustesse, etc.)



Test de (non)régession

- ❖ Caractéristiques
 - ❖ S'effectue en phase de maintenance
 - ❖ Réalisé par développeurs et équipe de test
 - ❖ Intervient après modification d'un composant et test unitaire, d'intégration ou de validation
- ❖ But
 - ❖ vérifier que la maintenance n'introduit pas de nouveaux défauts

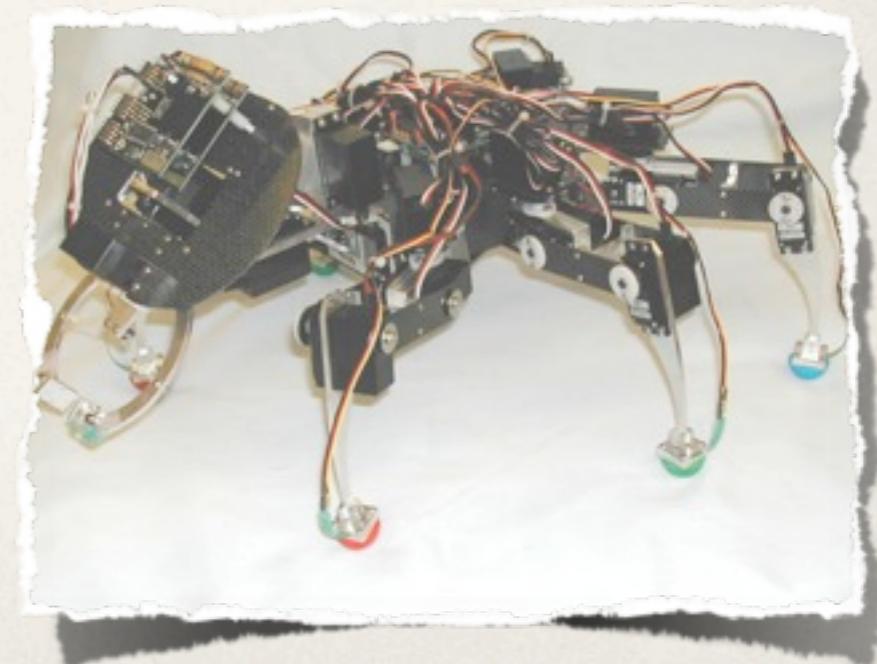


Classification des tests

- ✿ Portée:
 - ✿ unitaire
 - ✿ d'intégration
 - ✿ de validation
 - ✿ de recette
 - ✿ système
 - ✿ de (non) régression
- ✿ Exécution du code
 - ✿ statique
 - ✿ dynamique
- ✿ Accès au code
 - ✿ boîte blanche
 - ✿ boîte noire
- ✿ Structurel, aléatoire, fonctionnel, de mutation

Test statique

- ❖ **Caractéristiques**
 - ❖ fondé sur l'examen du source du logiciel à tester
 - ❖ *pas d'exécution*
- ❖ **But**
 - ❖ Eliminer vite des erreurs due à des erreurs de frappe
(ex: =, à la place de == en C++, clause else des ifs imbriquées, etc)
- ❖ **Avantages**
 - ❖ efficace
 - ❖ peu coûteux
- ❖ **Inconvénients**
 - ❖ dépendant langage
 - ❖ résultats peu réutilisables



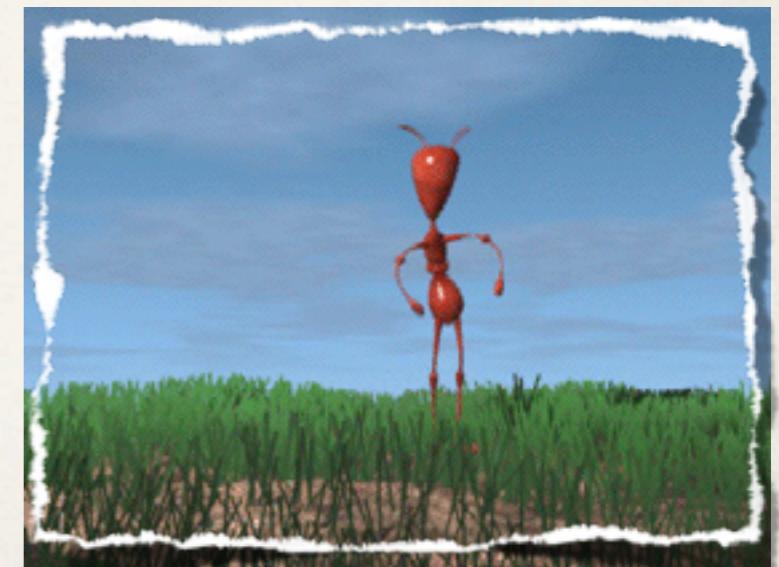
©Ileana Ober, 2012

Test dynamique

- ❖ **Caractéristiques**
 - ❖ fondé sur l'exécution du logiciel à tester, avec un sous-ensemble des entrées possibles (jeu de tests)
- ❖ **Avantages**
 - ❖ efficace
- ❖ **Inconvénients**
 - ❖ long
 - ❖ coûteux
- ❖ **Problèmes posés**
 - ❖ sélection du jeu de tests
 - ❖ soumission du jeu de tests
 - ❖ oracle
 - ❖ arrêt du test

Test dynamique

- ❖ Caractéristiques
 - ❖ fondé sur l'exécution du logiciel à tester, avec un sous-ensemble des entrées possibles (jeu de tests)
- ❖ Avantages
 - ❖ efficace
- ❖ Inconvénients
 - ❖ long
 - ❖ coûteux
- ❖ Problèmes posés
 - ❖ sélection du jeu de tests
 - ❖ soumission du jeu de tests
 - ❖ oracle
 - ❖ arrêt du test



Classification des tests

- ✿ Portée:
 - ✿ unitaire
 - ✿ d'intégration
 - ✿ de validation
 - ✿ de recette
 - ✿ système
 - ✿ de (non) régression
- ✿ Exécution du code
 - ✿ statique
 - ✿ dynamique
- ✿ Accès au code
 - ✿ boîte blanche
 - ✿ boîte noire
- ✿ Structurel, aléatoire, fonctionnel, de mutation

Test boite noire



✿ Caractéristiques

- ✿ vise le comportement des entrées-sorties
- ✿ se fait à partir de spécifications, i.e. basé sur le dossier de conception
 - ✿ ne nécessite pas de connaître la structure interne du système
 - ✿ permet d'assurer la conformité spé - code, mais aveugle aux défauts fins de programmation

✿ But

- ✿ Réduire les cas de test en trouvant des classes d'équivalence

✿ Avantages

- ✿ Pas trop de problème d'oracle pour le cas de test
- ✿ Approprié pour le test du système mais également pour le test unitaire

✿ Inconvénient

- ✿ Pas de règle pour trouver les classes d'équivalence

✿ But

- ✿ Réduire les cas de test en trouvant des classes d'équivalence
- ✿ Faire au moins un test par classe d'équivalence

Test boite blanche

- ❖ Caractéristiques

- ❖ La structure interne du système doit être accessible
- ❖ Se base sur le code : très précis
- ❖ Sensible aux défauts fins de programmation, mais aveugle aux fonctionnalité absentes

- ❖ But

- ❖ exécuter chaque instruction au moins une fois

- ❖ Types de test boite blanche

- ❖ Test d'instruction
- ❖ Test de boucle
- ❖ Test de branche
- ❖ Test de chemin



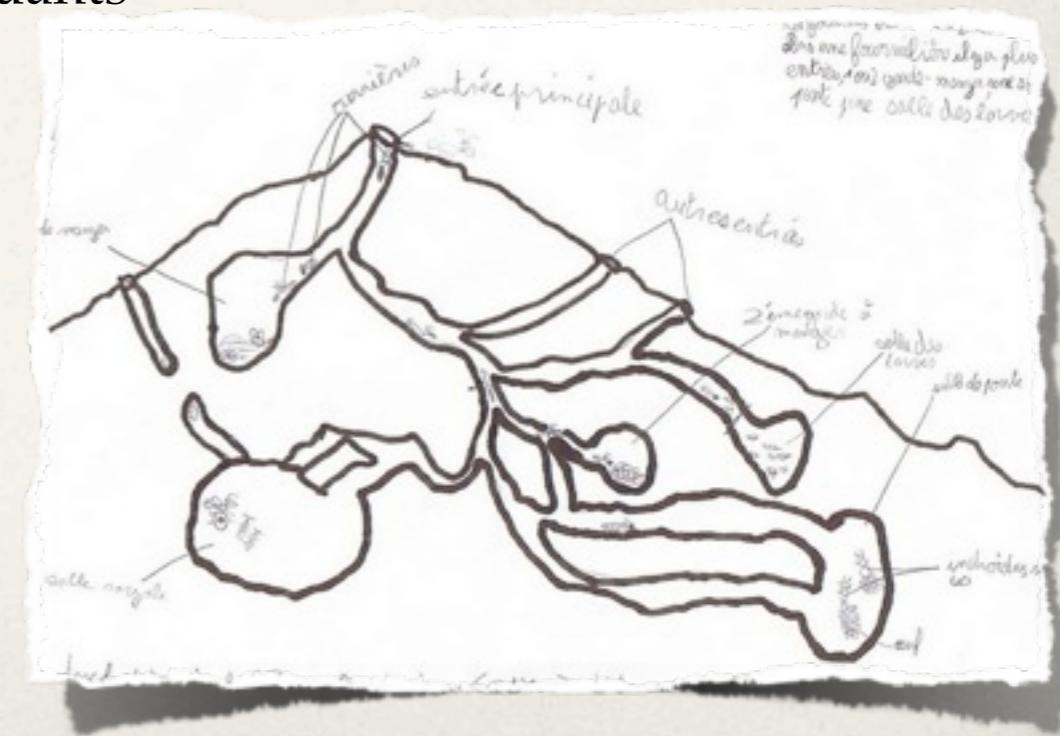
©Ileana Ober, 2012

Classification des tests

- * Portée:
 - * unitaire
 - * d'intégration
 - * de validation
 - * de recette
 - * système
 - * de (non) régression
- * Exécution du code
 - * statique
 - * dynamique
- * Accès au code
 - * boîte blanche
 - * boîte noire
- * **Structuel, aléatoire, fonctionnel, de mutation**

Test structurel

- ❖ **But**
 - ❖ exécuter tous les chemins du source du logiciel à tester
 - ❖ **Caractéristiques**
 - ❖ Dynamique
 - ❖ Boîte blanche
 - ❖ Fondé sur la structure du source du logiciel à tester :
 - ❖ graphe de contrôle
 - ❖ graphe d'appel
 - ❖ méthode de sélection : critères de couverture correspondants
 - ❖ **Avantages**
 - ❖ facile à mettre en oeuvre
 - ❖ Très étudié, très outillé
 - ❖ **Inconvénients**
 - ❖ Peut demander un matériel de test important
 - ❖ N'identifie pas des chemins manquants
 - ❖ Non réutilisable



©Ileana Ober, 2012

Test fonctionnel

- ❖ Caractéristiques
 - ❖ Dynamique
 - ❖ Boîte noire
 - ❖ Fondé sur la spécification
- ❖ Avantages
 - ❖ Peut mettre en évidence des chemins manquants
 - ❖ Réutilisable
 - ❖ Aucune connaissance de l'implantation
 - ❖ Les jeux de test peuvent être trouvés avant le codage
- ❖ Inconvénients
 - ❖ Peu étudié, peu outillé



Test de mutation

⊕ Caractéristiques

- ⊕ méthode d'évaluation de la qualité du jeu de tests
- ⊕ indépendant de la méthode de sélection
- ⊕ fondé sur :
 - ⊕ la source du logiciel à tester
 - ⊕ le jeu de tests déjà sélectionné
- ⊕ statique, puis dynamique
- ⊕ boîte blanche, puis noire

⊕ Avantages

- ⊕ meilleure confiance dans le jeu de tests
- ⊕ détection de cas de tests manquants
- ⊕ arrêt sur élimination des mutants principaux

⊕ Inconvénients

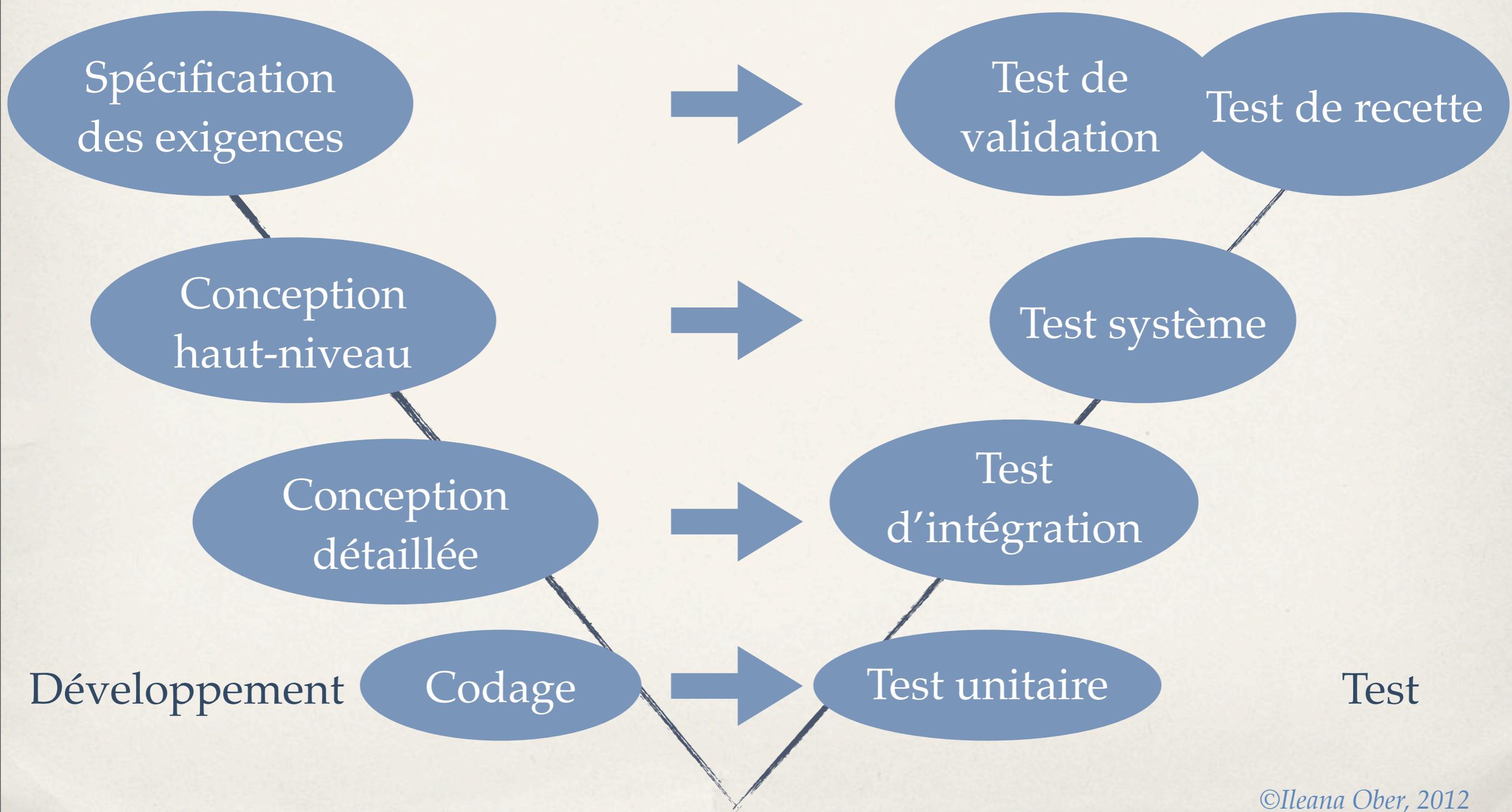
- ⊕ nombre de mutants ⇒ long, coûteux
- ⊕ équivalence initial / mutant
- ⊕ heuristiques de génération des mutants



Tableau récapitulatif

		unitaire	d'intégration	de validation	de recette	système	de (non) régression
statique		X					
	structurel	XX	X				
dynamique	fonctionnel	X	X	XX	XX	XX	XX
	aléatoire	X	X	X	X	X	X

Cycle de vie du logiciel



test fonctionnel

Comportements du logiciel

U

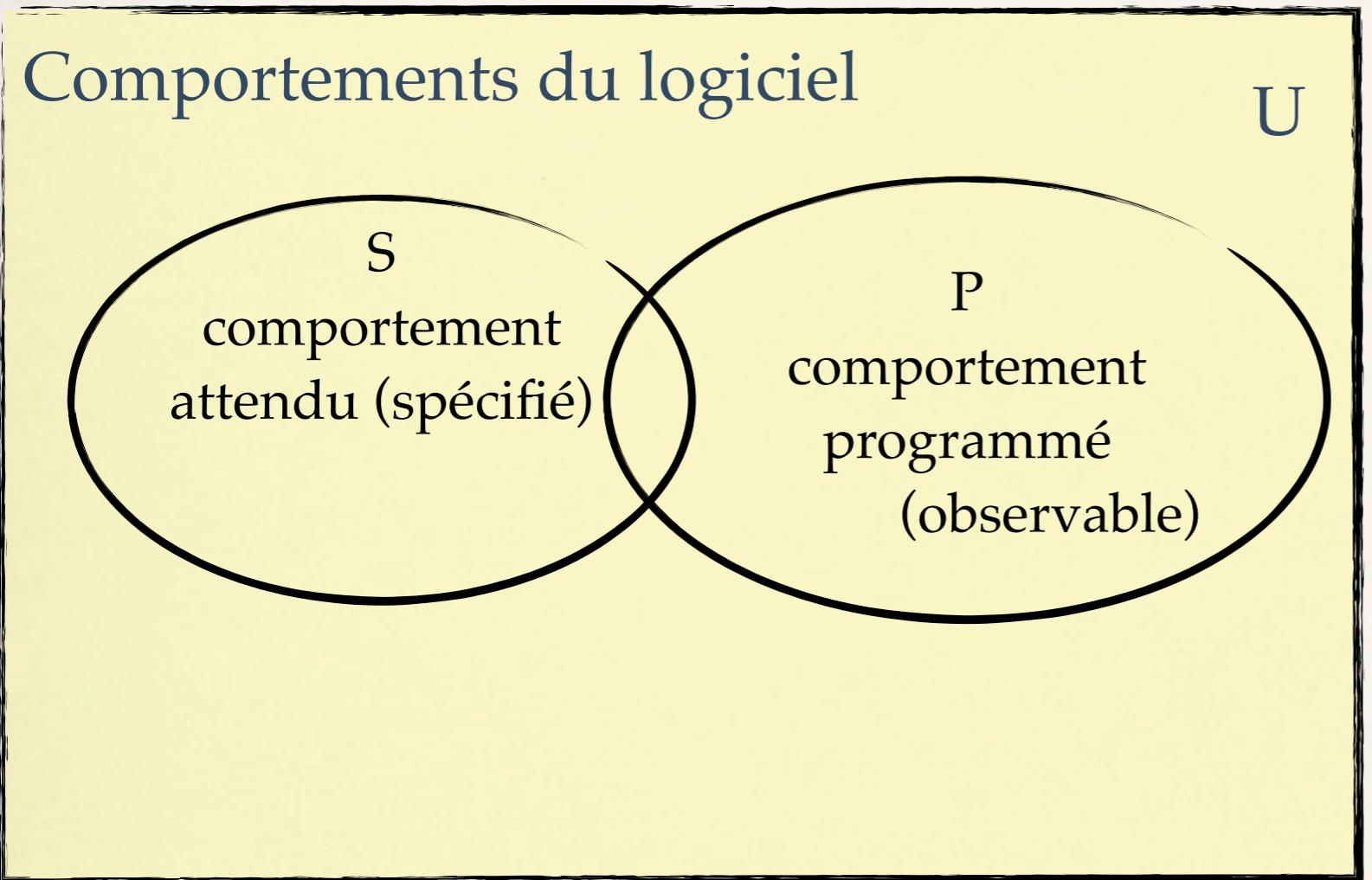
test fonctionnel

Comportements du logiciel

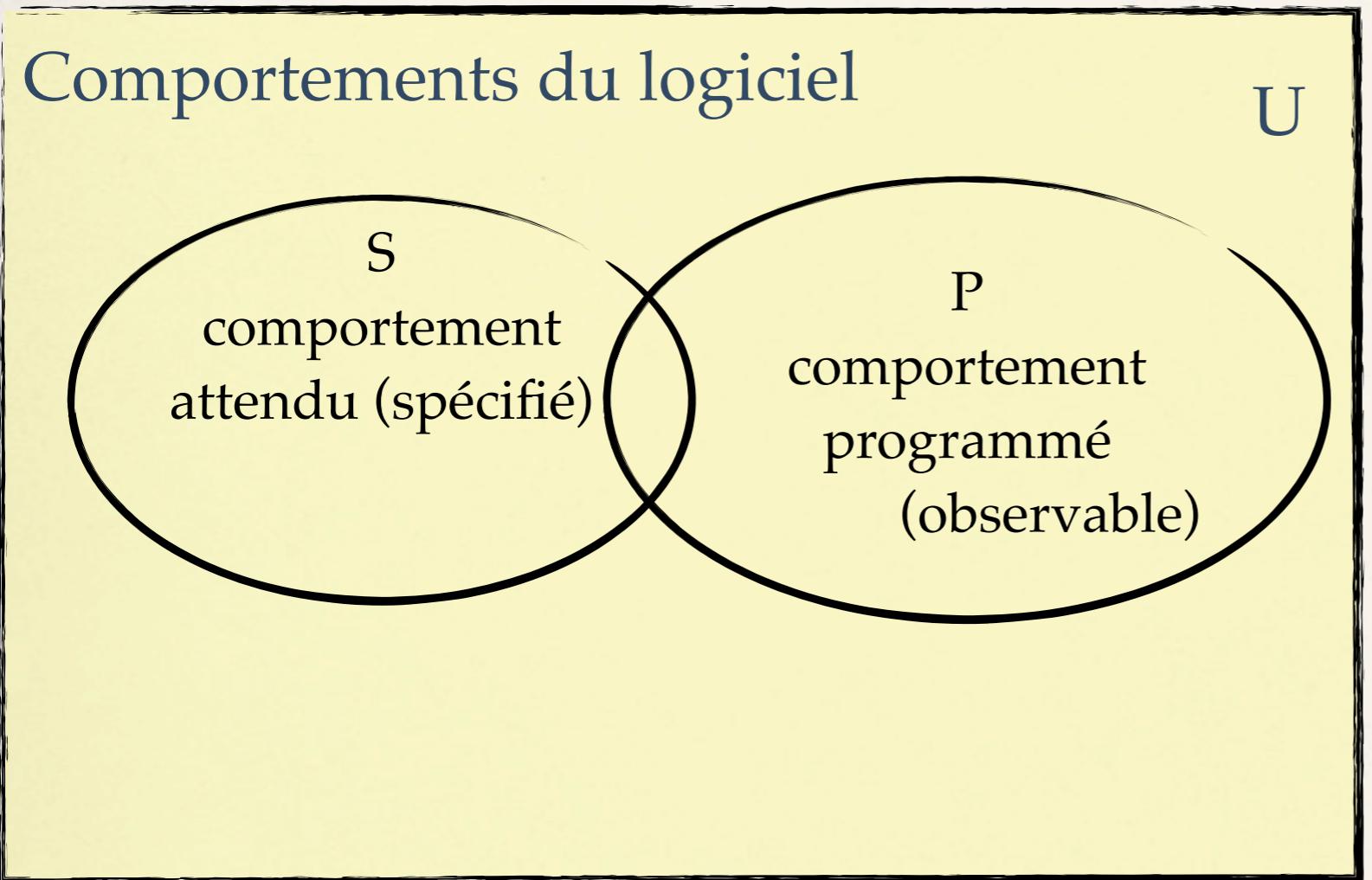
U

S
comportement
attendu (spécifié)

test fonctionnel



test fonctionnel

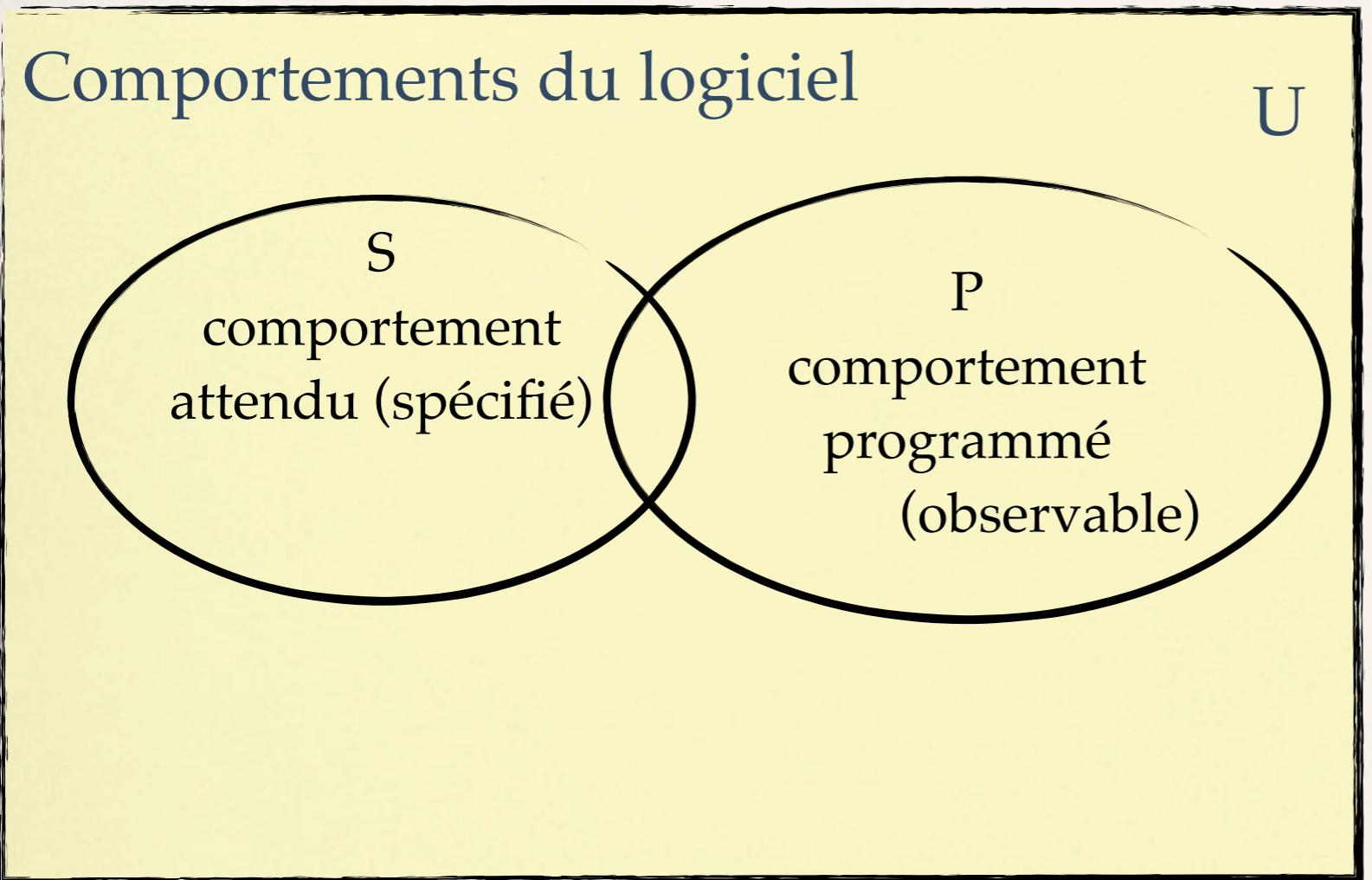


U = univers des comportements

S = comportement attendu (spécifié)

P = comportement programmé

test fonctionnel



U = univers des comportements

S = comportement attendu (spécifié)

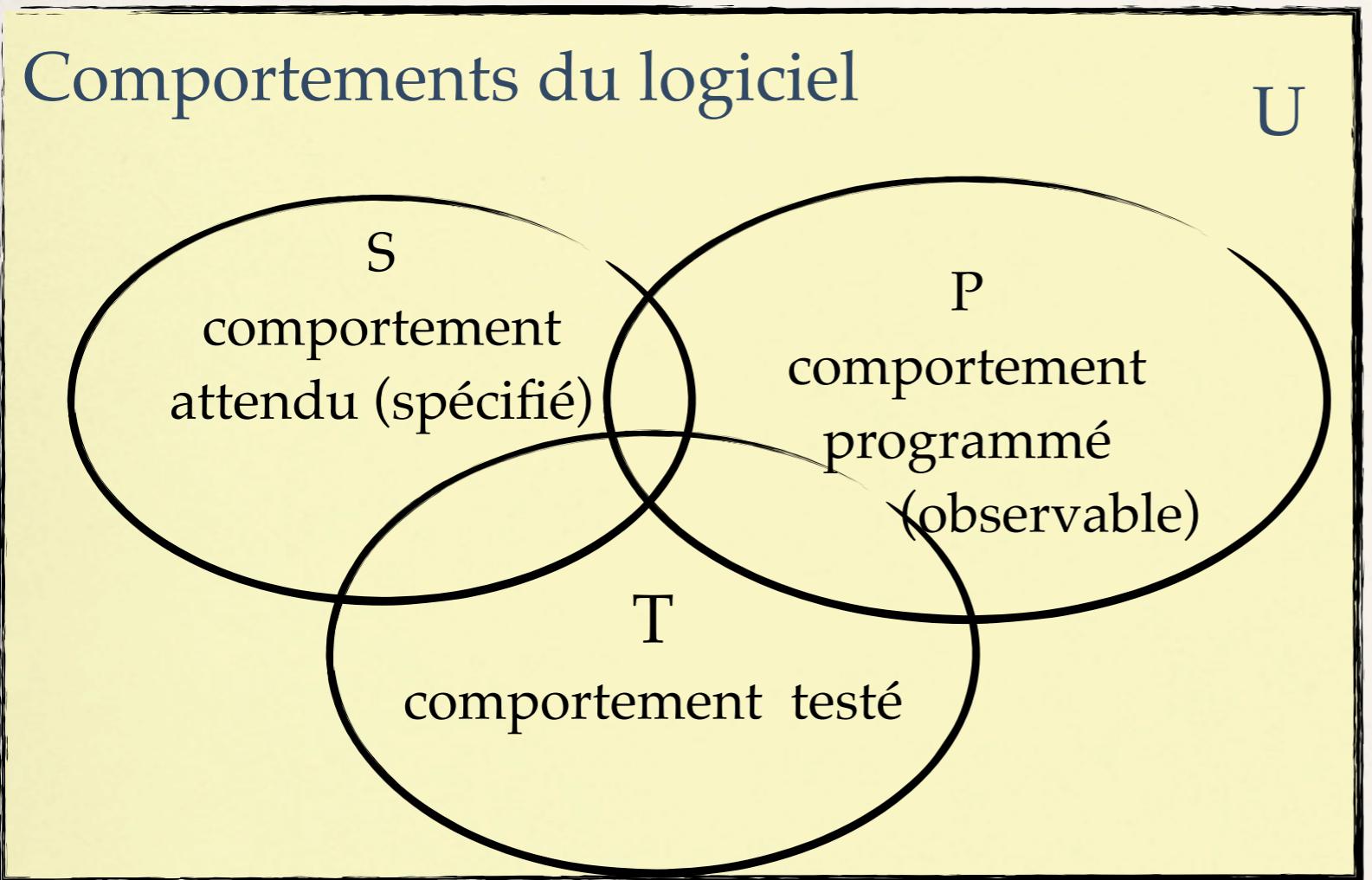
P = comportement programmé

$S - P$ = fonctionnalité oubliées (erreur par omission)

$P - S$ = erreur par commission (fonctionnalité non voulue)

$P \cap S$ = comportement correct

test fonctionnel



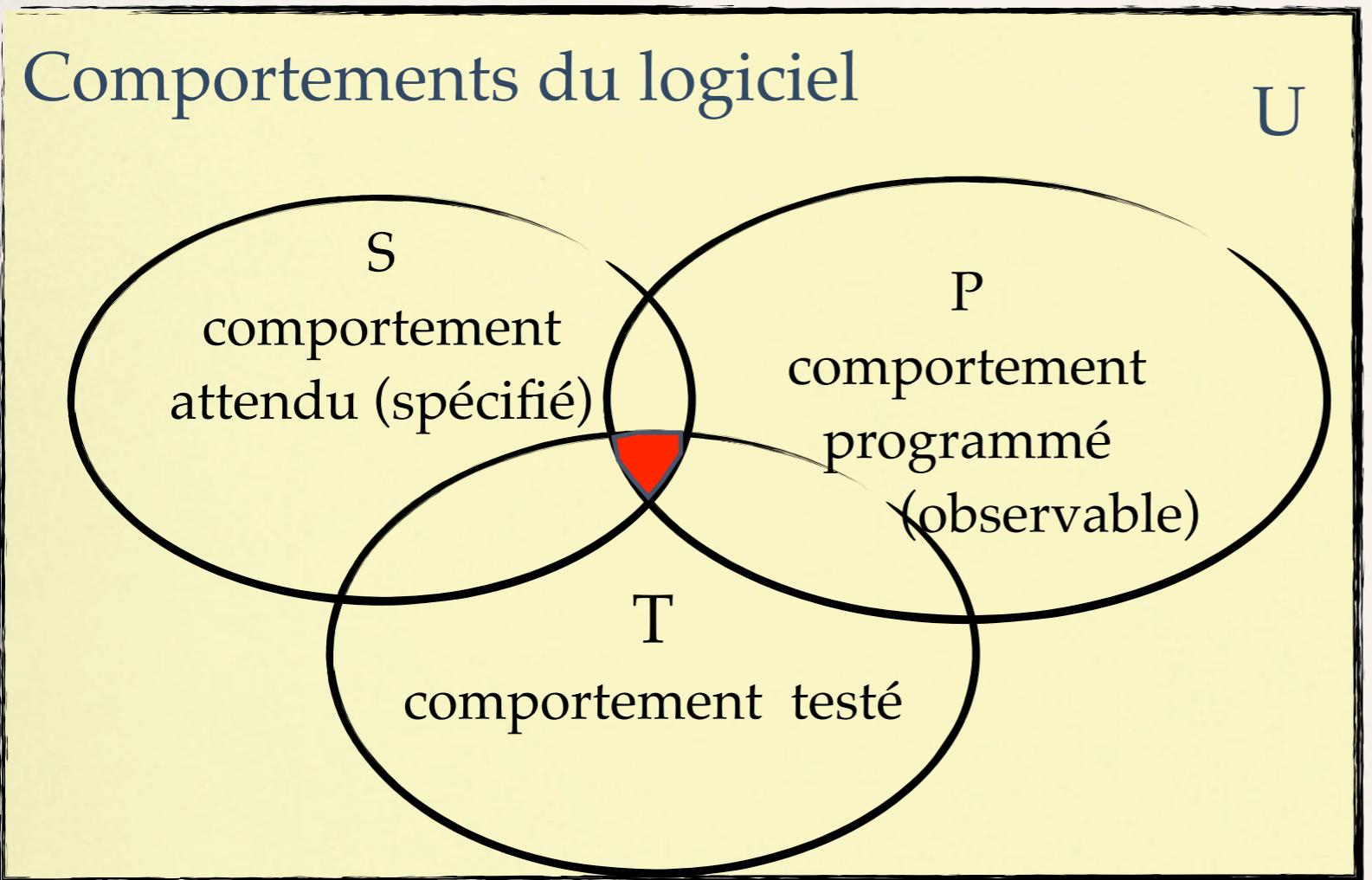
U = univers des comportements

S = comportement attendu (spécifié)

P = comportement programmé

T = comportement testé

test fonctionnel



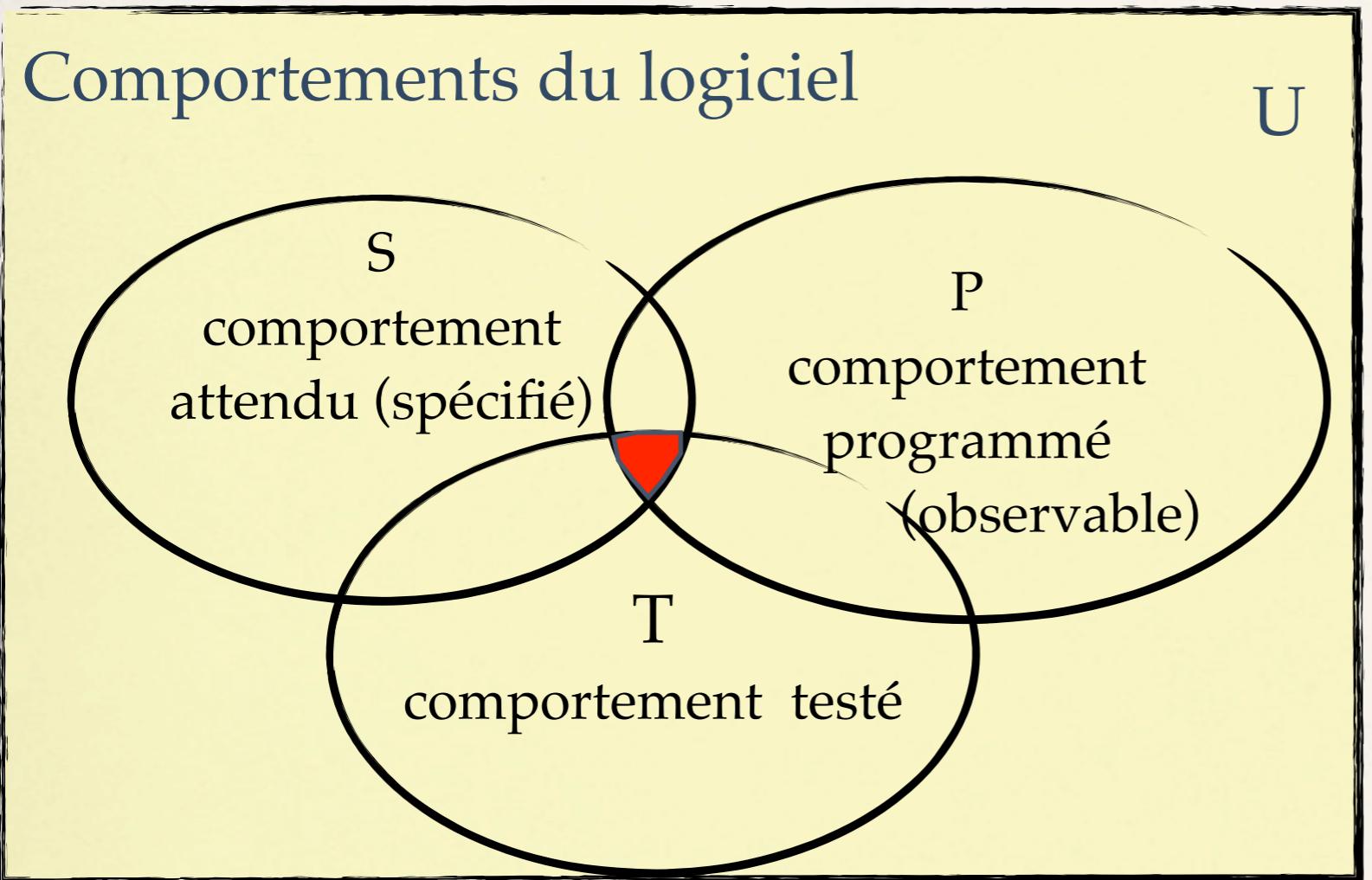
U = univers des comportements

S = comportement attendu (spécifié)

P = comportement programmé

T = comportement testé

test fonctionnel



U = univers des comportements

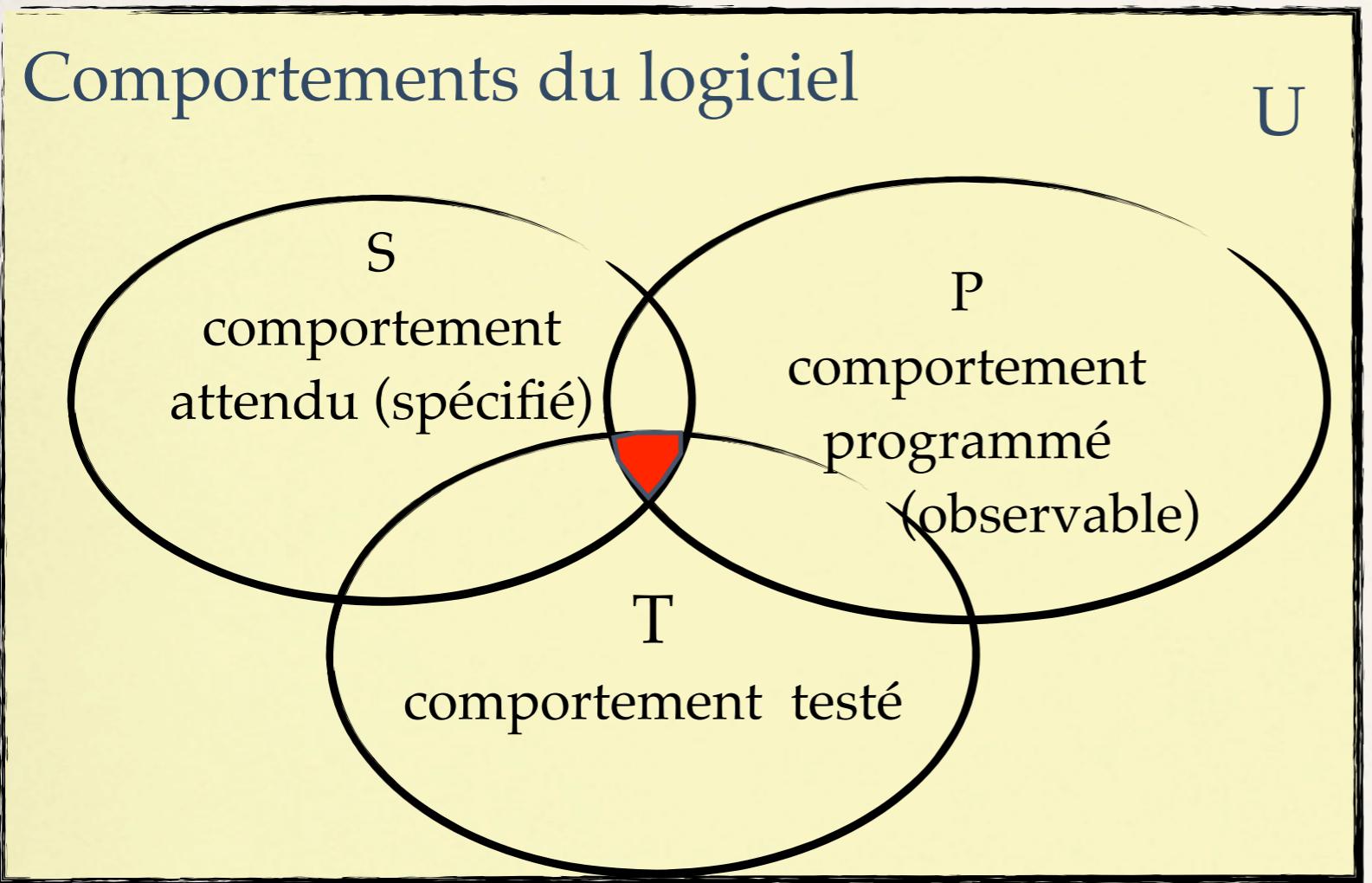
S = comportement attendu (spécifié)

P = comportement programmé

T = comportement testé

$S - P =$ fonctionnalité oubliées (erreur par omission)

test fonctionnel



U = univers des comportements

S = comportement attendu (spécifié)

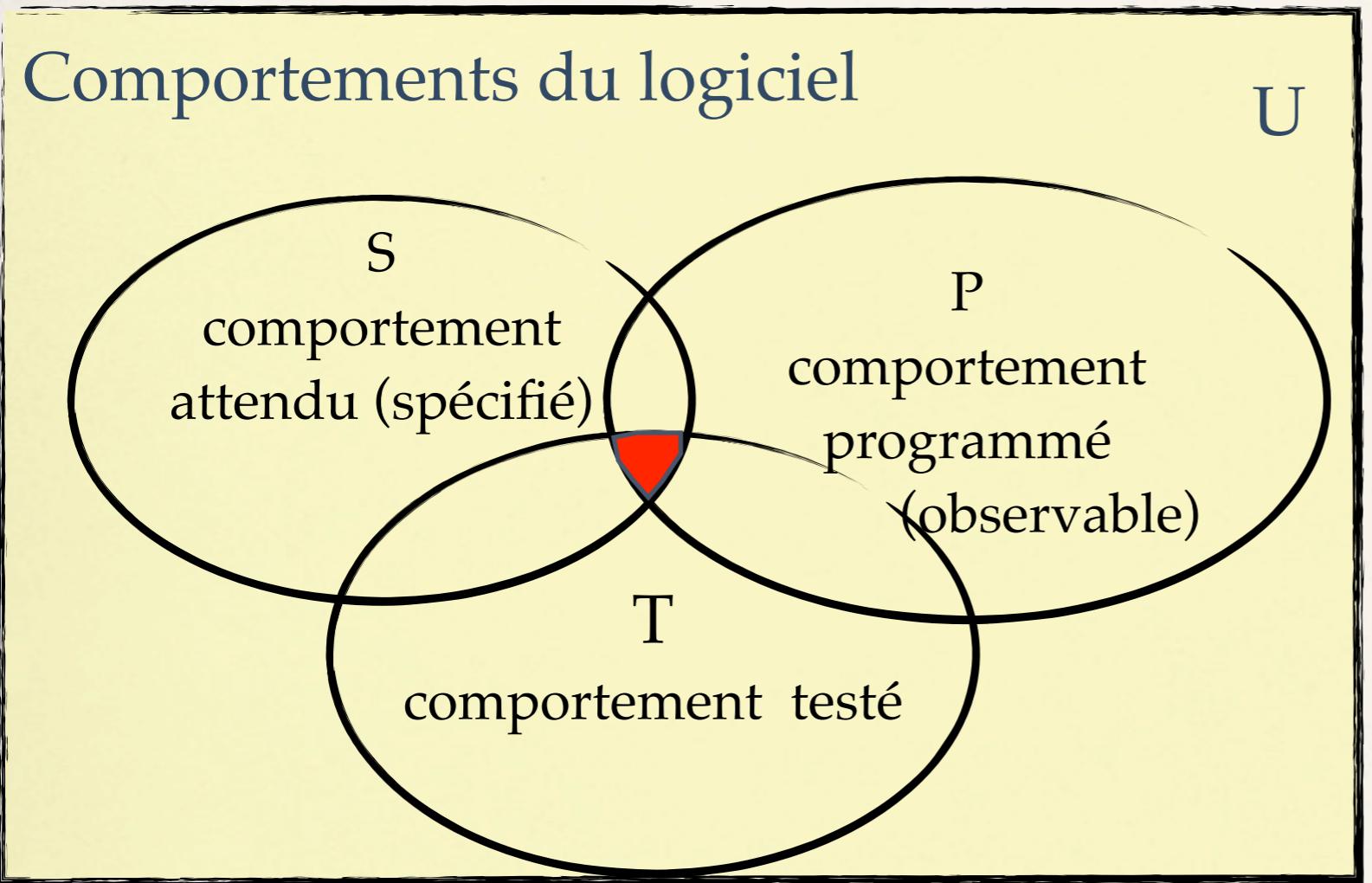
P = comportement programmé

T = comportement testé

$S - P$ = fonctionnalité oubliées (erreur par omission)

$P - S$ = erreur par commission

test fonctionnel



U = univers des comportements

S = comportement attendu (spécifié)

P = comportement programmé

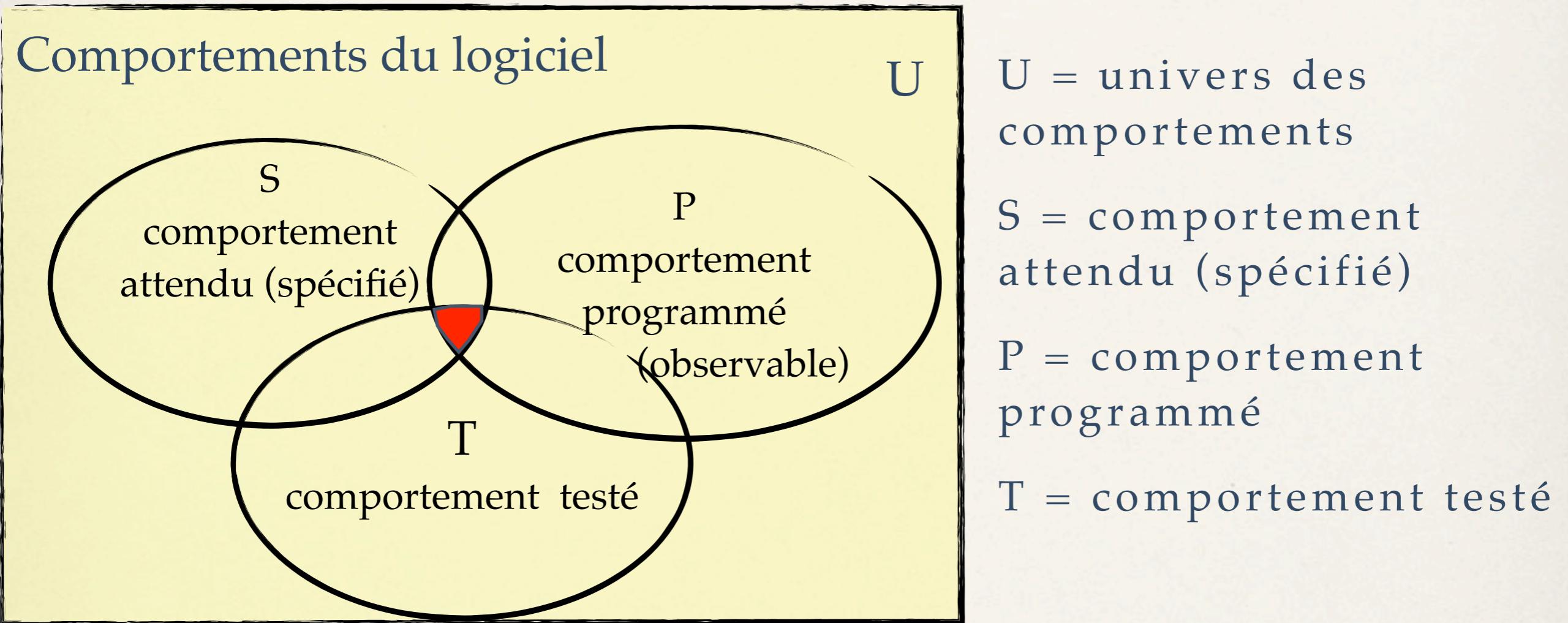
T = comportement testé

$S - P$ = fonctionnalité oubliées (erreur par omission)

$P - S$ = erreur par commission

$P \cap S$ = comportement correct

test fonctionnel



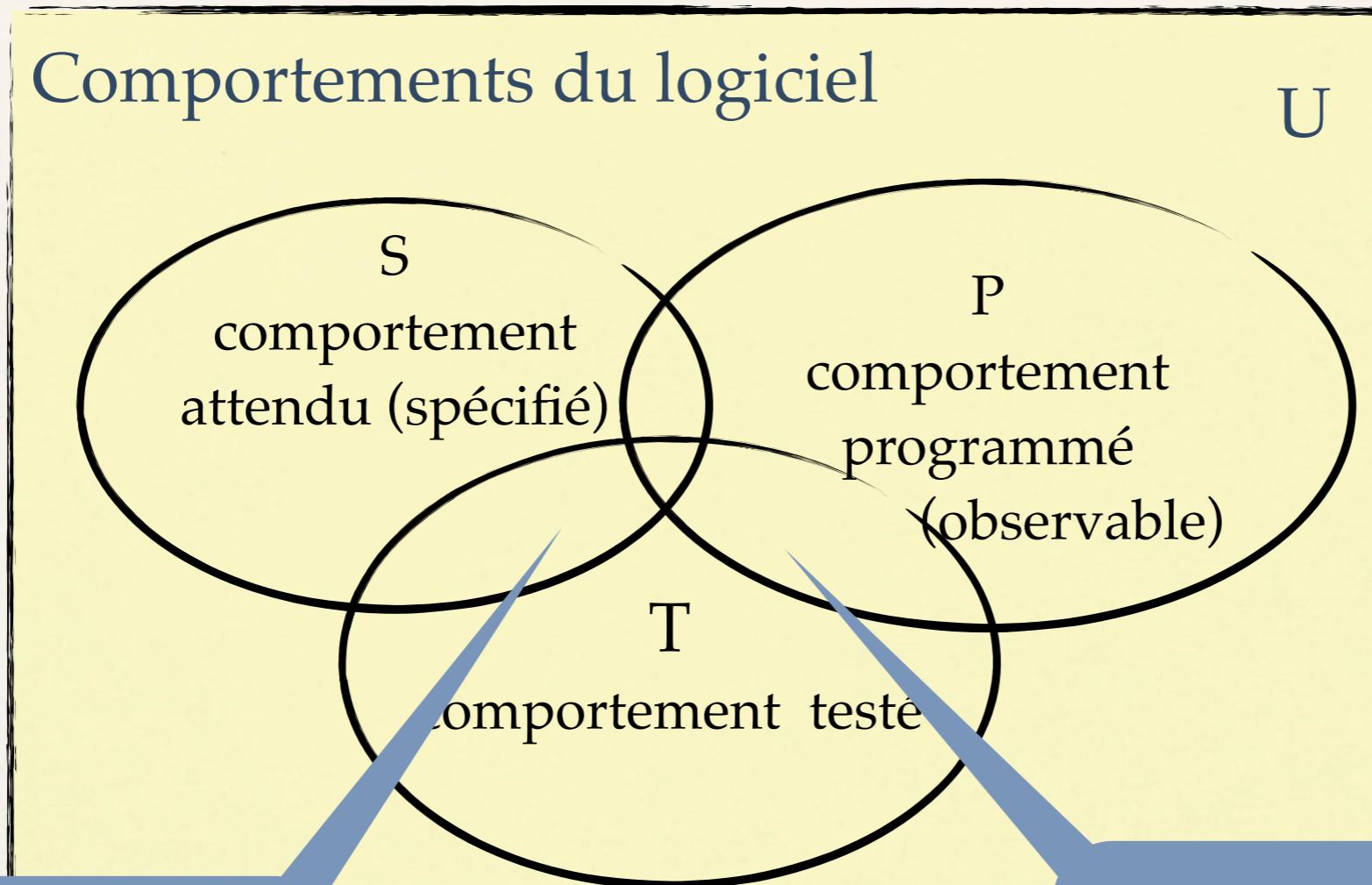
$S - P$ = fonctionnalité oubliées (erreur par omission)

$P - S$ = erreur par commission

$P \cap S$ = comportement correct

Objectif: maximiser $P \cap S \cap T$

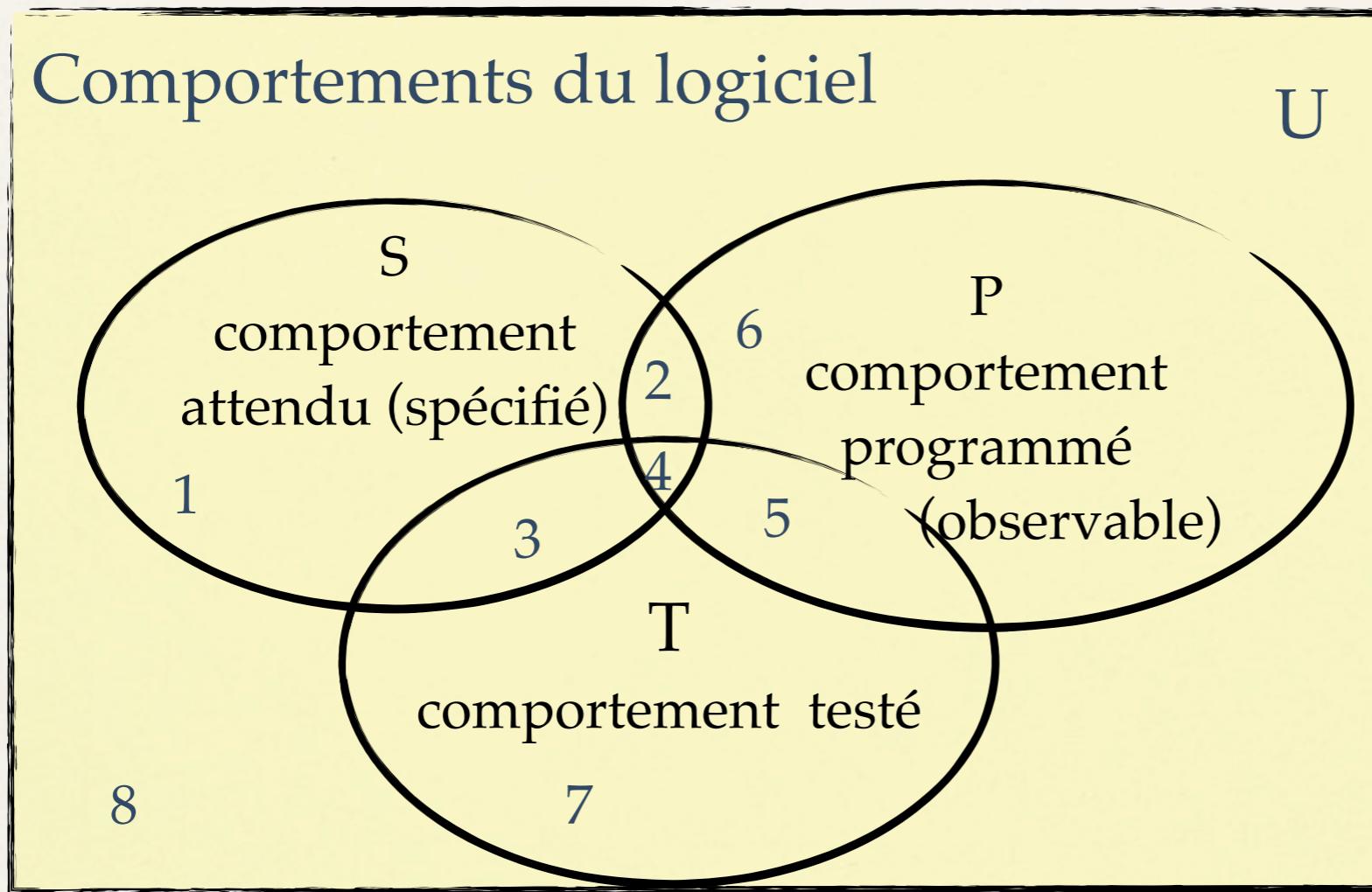
Techniques de test



Test fonctionnel
boîte noire
(exploration du
comportement spécifié)

Test structurel - boîte blanche
(exploration du comportement programmé)

Exercice



Identifier:

- le test incomplet

Plan du cours

- ❖ test et spécification des exigences
- ❖ classifications du test
- ❖ processus du test

Processus du test (1 seul test)

1. Choisir un **cas de test** (\mathcal{C}) - scénario à exécuter
2. (Oracle) Estimer le résultat attendu du \mathcal{C}
3. Déterminer
 1. une donnée de test (un jeu de test) \mathcal{H} qui correspond à \mathcal{C}
 2. son Oracle concret
4. Exécuter le programme sur \mathcal{H} (script de test)
5. Comparer le résultat obtenu avec le résultat attendu
(verdict: pass / fail pour le test)

Vocabulaire

- ❖ Cas de test = scénario à exécuter
- ❖ Script de test = code / script qui lance le programme à tester sur le DT choisi, observe les résultats, calcule le verdict
- ❖ Jeu de test / Donnée de test = Valeurs des données des entrées du programme, qu'on utilise dans le script de test
- ❖ Suite de test = Ensemble de jeux de test

Exemple

Spécification : tri de tableaux d'entiers + enlever la redondance
Interface : `int[] my-sort (int[] vec)`

Quelques cas de test et leurs oracles

c_1	tableau d'entiers non redondants	tableau trié
c_2	tableau vide	tableau vide
c_3	tableau avec deux entiers redondants	tableau trié sans redondance

Concrétisation : jeu de test obtenu

Jt_1	<code>vec = [12, 28 ,2 ,0]</code>	<code>[0, 2 ,12, 28]</code>
Jt_2	<code>vec = []</code>	<code>[]</code>
Jt_3	<code>vec = [2, 8, 9, 8]</code>	<code>[2, 8, 9]</code>

Script de test

```
void testSuite ( ) {  
    int [ ] c1 = [ 5 , 3 , 1 5 ] ; /* préparer données */  
    int [ ] oracle1 = [ 3 , 5 , 1 5 ] ; /* préparer oracle */  
    int [ ] res1 = my-sort( c1 ) ; /*exécuter CT et observer le résultat*/  
    if ( array-compare(res1 , oracle1)) /*vérifier si l'exécution est ok*/  
    then print ("test1 pass")  
    else print ("test1 fail");  
  
    int [ ] c2 = [ ] ; /* préparer données */  
    int [ ] oracle2 = [ ] ; /* préparer oracle */  
    int [ ] res2 = my-sort( c2 ) ; /*exécuter CT et observer le résultat*/  
    if ( array-compare(res2 , oracle2)) /*vérifier si l'exécution est ok*/  
    then print ("test2 pass")  
    else print ("test2 fail");  
  
    int [ ] c3 = [2, 8, 9, 8] ; /* préparer données */  
    int [ ] oracle3 = [2, 8, 9] ; /* préparer oracle */  
    int [ ] res3 = my-sort( c3 ) ; /*exécuter CT et observer le résultat*/  
    if ( array-compare(res3 , oracle3)) /*vérifier si l'exécution est ok*/  
    then print ("test3 pass")  
    else print ("test3 fail");  
}
```

Script de test

la lecture des jeux de test se fait en général dans un fichier externe

```
void testSuite ( ) {  
    int [ ] c1 = [ 5 , 3 , 1 5 ] ; /* préparer données */  
    int [ ] oracle1 = [ 3 , 5 , 1 5 ] ; /* préparer oracle */  
    int [ ] res1 = my-sort( c1 ) ; /*exécuter CT et observer le résultat*/  
    if ( array-compare(res1 , oracle1)) /*vérifier si l'exécution est ok*/  
    then print ("test1 pass")  
    else print ("test1 fail");  
  
    int [ ] c2 = [] ; /* préparer données */  
    int [ ] oracle2 = [] ; /* préparer oracle */  
    int [ ] res2 = my-sort( c2 ) ; /*exécuter CT et observer le résultat*/  
    if ( array-compare(res2 , oracle2)) /*vérifier si l'exécution est ok*/  
    then print ("test2 pass")  
    else print ("test2 fail");  
  
    int [ ] c3 = [2, 8, 9, 8] ; /* préparer données */  
    int [ ] oracle3 = [2, 8, 9] ; /* préparer oracle */  
    int [ ] res3 = my-sort( c3 ) ; /*exécuter CT et observer le résultat*/  
    if ( array-compare(res3 , oracle3)) /*vérifier si l'exécution est ok*/  
    then print ("test3 pass")  
    else print ("test3 fail");  
}
```

Bilan

- ❖ Le test peut commencer lieu uniquement quand les spécifications sont faites
- ❖ Il y a plusieurs classifications du test et plusieurs types de test
- ❖ Ces classifications ne sont pas disjointes
- ❖ Le test est un produit en soi-même (processus, développement, maintenance)