

# Testeur certifié CFTL

## Automatisation de test

### niveau avancé

## Formation V2016 FR

Conseil, Assistance et Formation spécialisés en Management de Projet

Testeur Certifié  
Syllabus  
Niveau Avancé



Comité  
Français  
des  
Tests  
Logiciels



International  
Software Testing  
Qualifications Board



- **Chapitre 1 : Introduction et objectifs pour l'automatisation de test.**
  - But de l'automatisation de test .
  - Facteurs de succès de l'automatisation de test.
- **Chapitre 2 : préparation pour l'automatisation de test.**
  - Facteurs du SUT influençant l'automatisation de test .
  - Evaluation et sélection d'outils.
  - Conception pour testabilité et automatisation.
- **Chapitre 3 : l'architecture d'automatisation de test générique.**
  - Introduction à la gTAA.
  - Conception de la TAA
  - Développement de TAS
- **Chapitre 4 : Risques et contingences liés au déploiement.**
  - Sélection de l'approche d'automatisation de test et de la planification du déploiement..
  - Evaluation des risques et stratégies d'atténuation
  - Maintenance des tests automatisé
- **Chapitre 5 : Métriques et reporting sur l'automatisation des tests.**
  - Sélection des mesures de la TAS
  - Mise en œuvre de la mesure
  - Enregistrement de la TAS et du SUT
  - Reporting de l'automatisation des tests

- **Chapitre 6 : Transition du test manuel vers un environnement automatisé.**
  - Critères d'automatisation.
  - Identifier les Etapes nécessaires pour implémenter l'automatisation des tests de régression.
- **Facteurs à considérer lors de l'automatisation de tests de nouvelles fonctionnalités.**
- **Facteurs à considérer lors de l'automatisation de tests de confirmation.**
- **Chapitre 7 : vérification de la TAS.**
  - Vérification des composants de l'environnement de test automatisé.
  - Vérification de la suite de tests automatisés
- **Chapitre 8 : Amélioration continue.**
  - Options pour améliorer l'environnement de test automatisé .
  - Planification de la mise en œuvre des améliorations de l'automatisation de test

**CLI** Interface de Ligne de Commande (Command Line Interface)

**EMTE** Effort de Test Manuel Équivalent (Equivalent Manual Test Effort)

**gTAA** Architecture générique d'Automatisation de Test (fournit un modèle pour des solutions d'automatisation de test) (generic Test Automation Architecture)

**GUI** Interface Utilisateur Graphique (Graphical User Interface)

**SUT** SUT, voir objet de test (System Under Test)

**TAA** Architecture d'Automatisation de Test (une instanciation de gTAA pour définir l'architecture d'une TAS) (Test Automation Architecture)

**TAE** Ingénieur en Automatisation de Test (la personne qui est responsable de la conception d'un TAA, son implémentation, sa maintenance et l'évolution technique de la TAS résultante) (Test Automation Engineer)

**TAF** Framework d'Automatisation de Test (Environnement requis pour l'automatisation de test incluant harnais et artefacts de test tels les bibliothèques de test) (Tests Automation Framework)

**TAM** Manager de l'Automatisation de Test (la personne responsable de la planification et de la

supervision du développement et de l'évolution d'une TAS) (Test Automation Manager)

**TAS** Solution d'Automatisation de Test (la réalisation/implémentation d'une TAA) (Test Automation Solution)

**TAST** Stratégie d'Automatisation de Test (Test Automation Strategy) UI Interface Utilisateur (User Interface)

# Objectifs d'apprentissage du chapitre

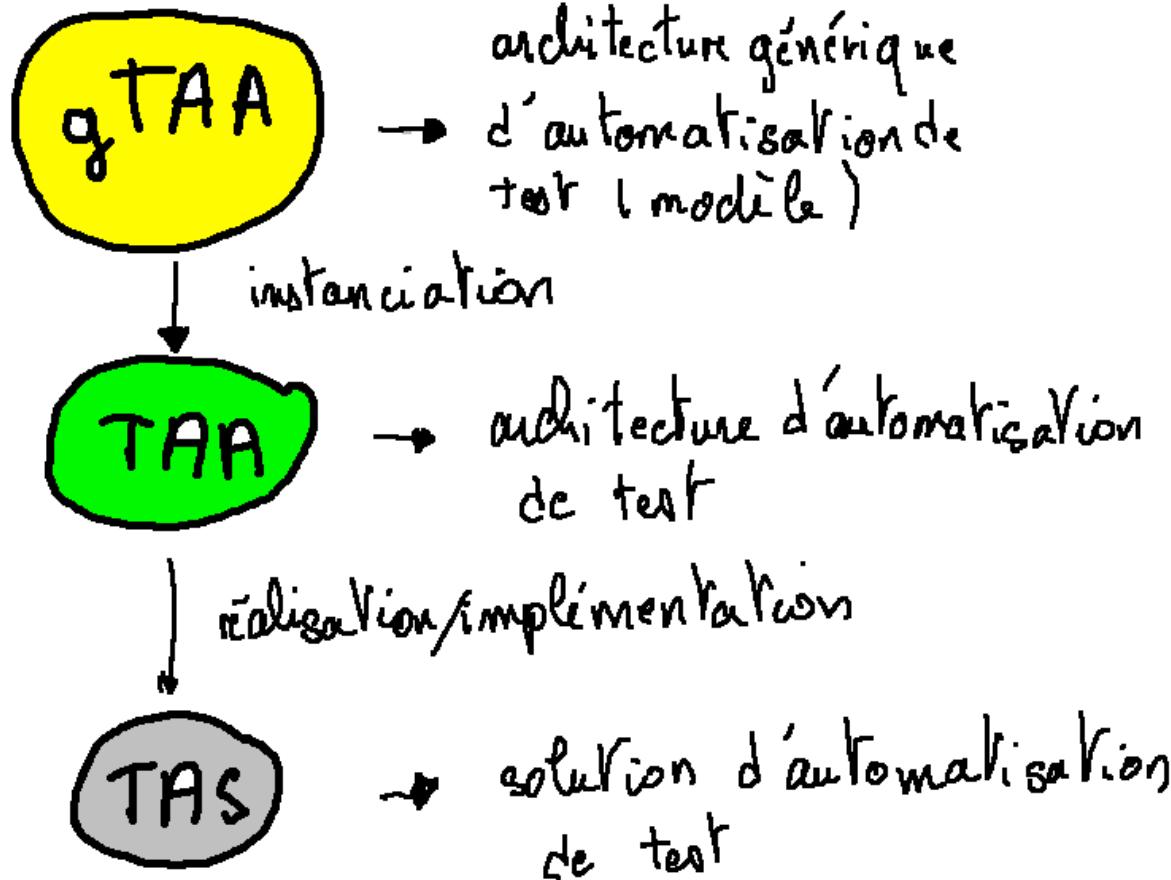
## 1. Introduction et objectifs pour l'automatisation des tests

### 1.1 But de l'automatisation de test

**ALTA-E-1.1.1 (K2) Expliquer les objectifs, avantages et limites de l'automatisation de test**

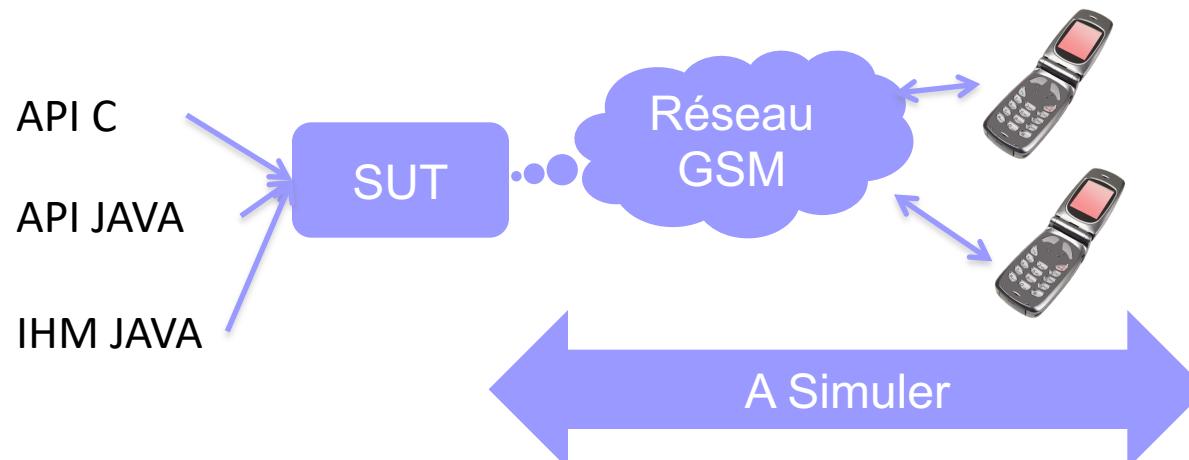
### 1.2 Facteurs de succès dans l'automatisation de test

**ALTA-E-1.2.1 (K2) Identifier les facteurs techniques de succès d'un projet d'automatisation de test**

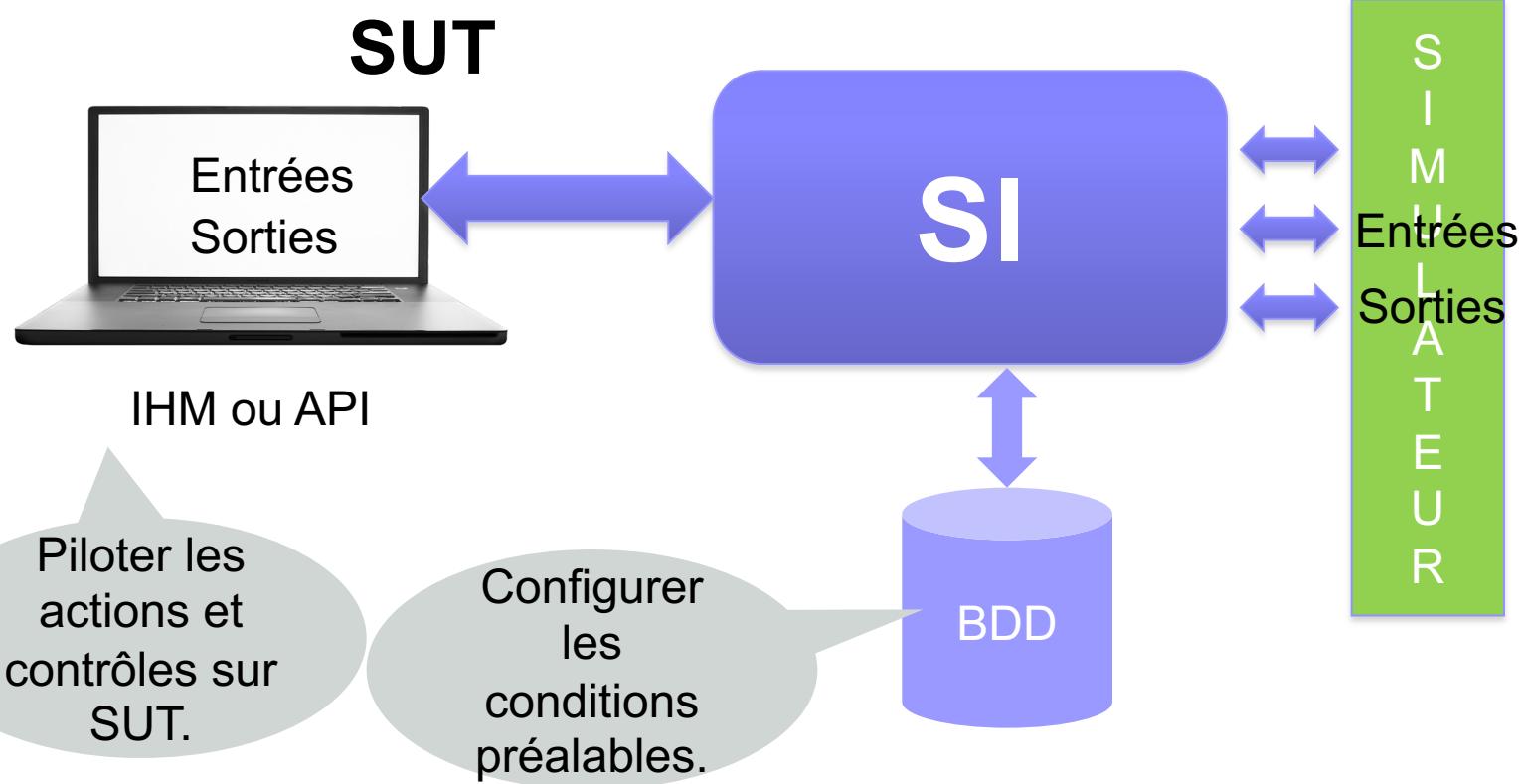


# Exemple fil rouge

- Editeur qui fournit des applications pour des opérateurs de téléphonie permettant de mettre à jour des informations sur une carte SIM.
- On accède à l'application via des API C, java ou une interface client lourd, mais les opérateurs préfèrent les APIs
- Equipes de testeurs plutôt techniques
- Existant



## 1.1- But de l'automatisation de test



## 1.1- But de l'automatisation de test

- **Le pilotage nécessite des logiciels spécifiques pour exécuter les tests.**
- **Le contrôle des sorties nécessite de développer des comparateurs.**
- **Cet outillage permettra de:**
  - **Configurer les conditions préalables.**
  - **Exécuter les tests.**
  - **Comparer des résultats obtenus aux résultats attendus.**
- **Il est préférable de séparer le SUT du système d'automatisation, mais cela n'est pas toujours possible, notamment dans les systèmes embarqués.**

## 1.1- But de l'automatisation de test

- **Un projet de d'automatisation est un projet de développement à part entière:**

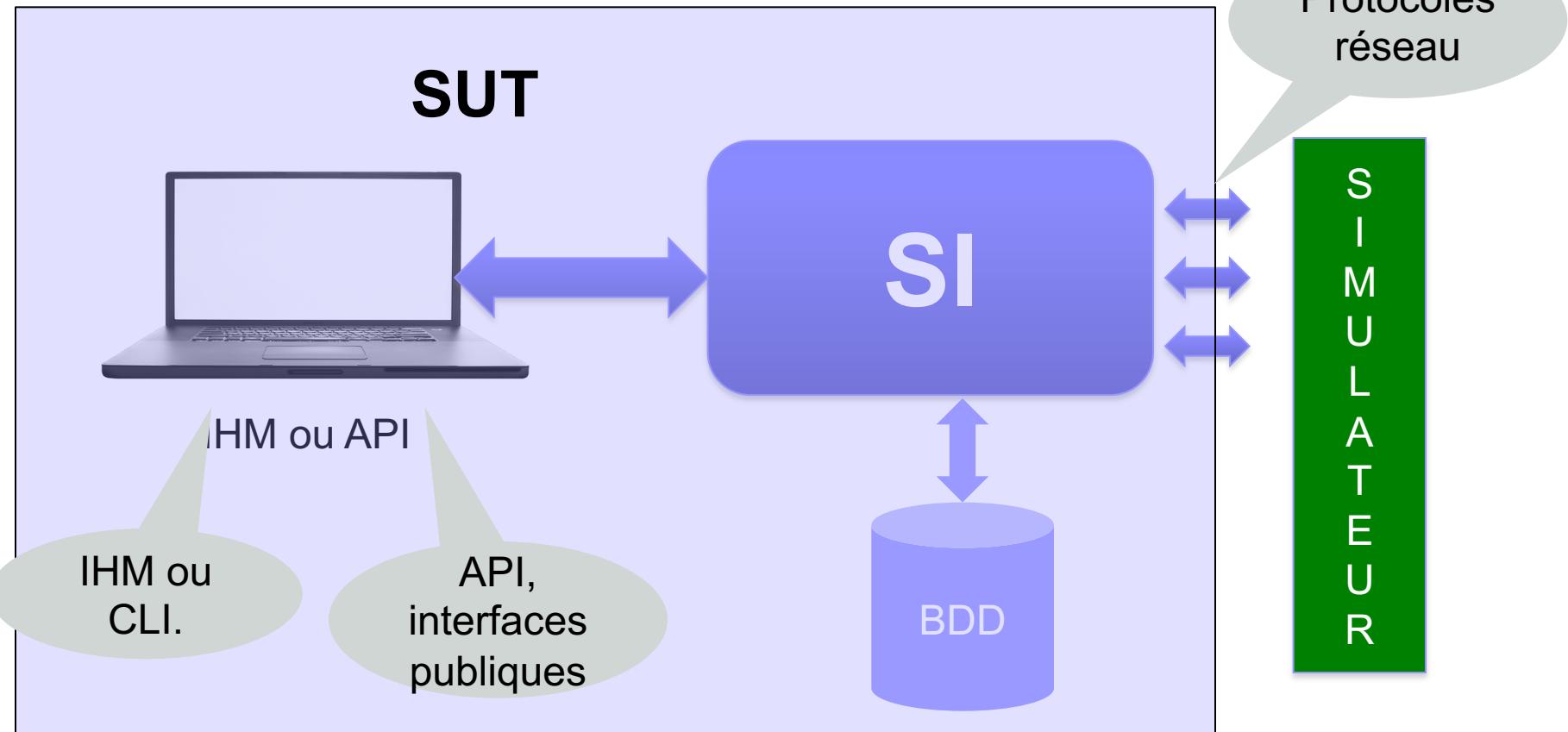
- **Coûts.**
  - **Gestion de projet.**
  - **Planification.**
  - **Ressources spécialisées.**
  - **Maintenance.**

- **Processus de conception de tests incluant:**

- **Logiciels pour créer, exécuter les tests**
  - **Documentations**
  - **Cas de test**
  - **Environnements de test**
  - **Données**

## 1.1- But de l'automatisation de test

identifier comment piloter et observer ...



### Objectif de l'automatisation

- Améliorer l'efficacité du test
- Fournir une couverture plus large
  - Exhaustivité des jeux d'essais
- Réduire le cout total du test
  - Machine versus humain
- Réaliser des tests non faisables par un humain
  - Contrôle de fichier, calculs complexes
- Réduire la période de test
- Augmenter la fréquence/réduire le temps requis pour les cycles de test
  - Intégration continue

### Exercice :

- Avantages
- Désavantages
- Limites

# 1 - Introduction et objectifs pour l'automatisation des tests



## 1.1- But de l'automatisation de test

### Avantages



Un automate:

- Ne se trompe pas
- Peut répéter à l'infini des actions
- Effectue des vérifications complexes

- Un automate fait ce pourquoi il a été programmé
- Si le programmeur se trompe le test est faux → bug
- Test complexe

### Désavantages



# 1 - Introduction et objectifs pour l'automatisation des tests



## 1.1- But de l'automatisation de test

- La diminution des coûts est souvent mise en avant mais la fiabilisation des tests peut-être également un objectif important.

Nombre d'actions

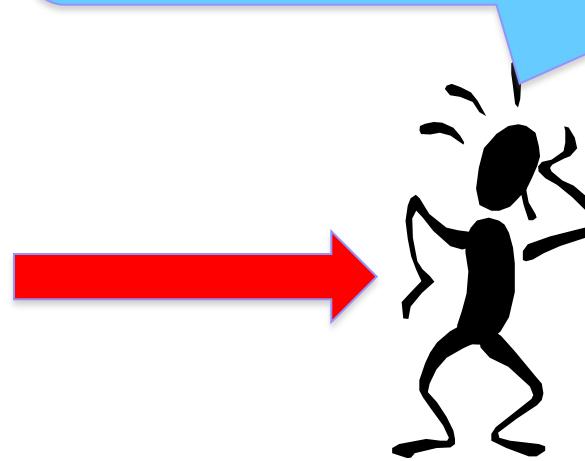
- Procédures IHM
- Batch

Vérifications

- Contrôle de fichiers
- Contrôle de champs
- Contrôle en base



J'ai du me tromper quelque part?



Risque d'erreur élevé



# 1 - Introduction et objectifs pour l'automatisation des tests

## 1.1- But de l'automatisation de test

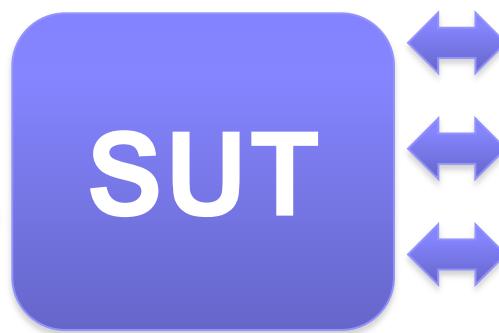
Coût du développement du simulateur.

Coût de la mise en place d'un SUT « test auto ».

Les coûts ne sont pas négligeables.

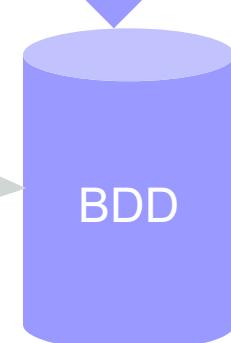


IHM ou API



Coût de l'outillage + coût du développement des scripts et des jeux de données.

Coût de l'outillage pour générer les données, les outils pour la base.



S  
I  
M  
U  
L  
A  
T  
E  
U  
R

### Avantage de l'automatisation

- Plus de tests exécutés par Build
- Les tests ne pouvant pas être réalisés manuellement sont rendus possibles (tests en temps réel, à distance, en parallèle)
- Les tests peuvent être plus complexes
- Les tests sont exécutés plus rapidement
- Les tests sont moins sujets à des erreurs opératoires
- Utilisation plus efficiente et efficace des testeurs
- Feedback plus rapide sur la qualité du logiciel
- Fiabilité des systèmes améliorée
- Qualité des tests améliorée

## 1.1- But de l'automatisation de test

### Désavantages de l'automatisation

- Implication de coûts supplémentaires
- Investissement initial pour configurer la TAS
- Demande des technologies additionnelles
- L'équipe doit maîtriser des compétences en développement et en automatisation
- Besoin continu de maintenance de la TAS
- Peut distraire des objectifs du test (p.ex. se concentrer sur l'automatisation des cas de test au détriment de l'exécution)
- Les tests peuvent devenir plus complexes
- Des erreurs supplémentaires peuvent être introduites par l'automatisation

## 1.1- But de l'automatisation de test

### ■ Limites de l'automatisation:

- Tous les tests manuels ne peuvent pas être automatisés
- L'automatisation peut uniquement vérifier des résultats interprétables par une machine
- L'automatisation peut uniquement vérifier des résultats obtenus qui peuvent être vérifiés par un oracle de test automatisé
- Ne remplace pas les tests exploratoires

### ■ Ils sont multiples et dépendent de:

- L'architecture d'automatisation
- La testabilité du SUT
- La stratégie d'automatisation
- Le Framework d'automatisation

### L'architecture d'automatisation de test (TAA)

- Elle doit être en ligne avec le produit logiciel à tester.
- Définit tôt dans le cycle de vie du SUT (par exemple la TAA peut être conçue une fois l'architecture du SUT défini).
- Les exigences fonctionnelles et non-fonctionnelles doivent être définies et en particulier:
  - ❖ La maintenabilité
  - ❖ Performance (temps d'exécution des tests)
  - ❖ Facilité d'apprentissage
- Les ingénieurs en capacité à comprendre l'architecture du SUT doivent être impliqués dans la définition de l'architecture d'automatisation

### Testabilité du SUT

- Le SUT doit être pilotable via:
  - ❖ L'IHM
  - ❖ API
  - ❖ CLI
- Quitte à rendre publique des interfaces privées

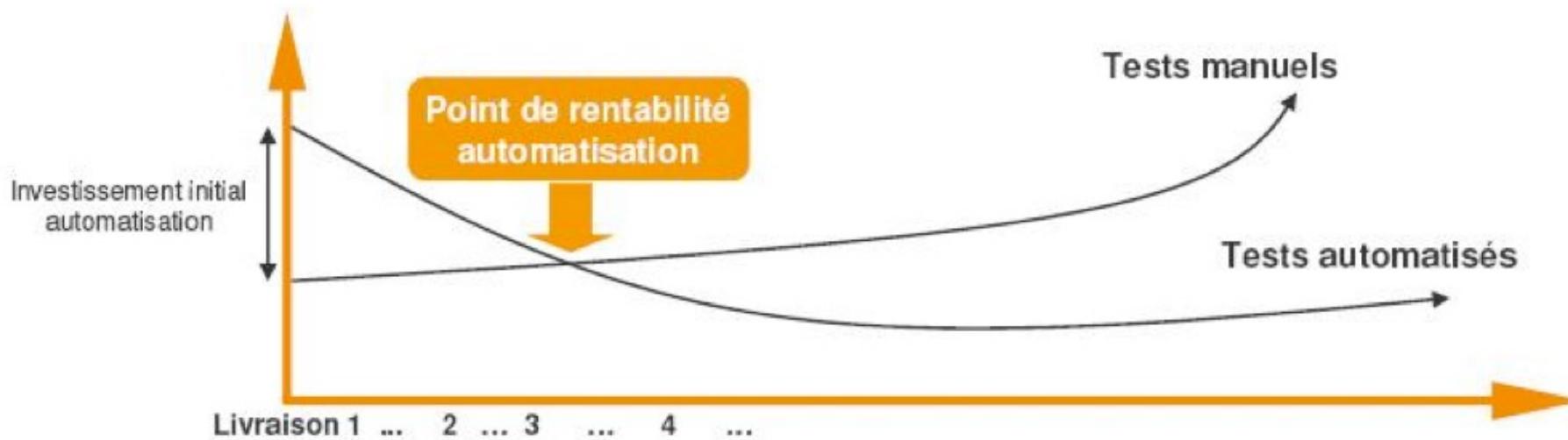
### ■ Il est judicieux de commencer par le « plus facile »

- Preuve de concept
- Montée en compétence

### ■ Un des facteurs de succès est la simplicité d'écriture des scripts de test

### Stratégie d'automatisation de test : Elle doit aborder les notions de

- Coût
- Retour sur investissement
- Maintenabilité



### Définir une Stratégie d'automatisation

Quels objectifs, quel périmètre?

Environnement de test:

Quels outils?

Des simulateurs?

Données?

Formations, expertises?

Métriques? ROI?



### Le Framework d'Automatisation des tests (TAF)

Facile à utiliser et maintenable:

- Reporting
- Enregistrements (log)
  - ❖ Analyse des problèmes
  - ❖ Traçage des étapes
- Gérer les environnements de test de façon cohérente
  - ❖ Outils de test
  - ❖ Environnement de test dédié ...
- Documenté
  - ❖ Facilite l'adoption
- Les scripts de test doivent être maintenables



# 1 - Introduction et objectifs pour l'automatisation des tests



## 1.2 Facteurs de succès de l'automatisation des tests

Gérer les logs.



Pourquoi le test est-il KO?  
Je vais le rejouer en  
manuel.

Les logs doivent fournir les informations nécessaires aux dépouillement des tests. (SUT, TAS ou environnement de test).

- **Quel outil vous permettra de gérer vos environnements de test en cohérence avec le SUT**
  - La gestion de configuration**
  
- **Quelles pratiques avez-vous mis en place pour assurer la maintenabilité de vos tests?**
  - Modularité**
  - Variabilité**
  - Commentaire**
  - Bibliothèques d'utilitaires**
  - ...**

### □ A jour

- ❖ Correction des tests au fur et à mesure

### □ dé ployable

- ❖ Facilement (en utilisant la gestion de configuration par exemple) ou mieux encore automatiquement (CI)

### □ Suppression des tests

- ❖ Ménage

### □ Surveillance du SUT

- ❖ Stratégie de récupération en cas de problème sur le SUT

### □ Maintenabilité du code de test

- ❖ Pas de code sensible à l'interface
- ❖ Pas de code sensible aux modifications de données
- ❖ Pas d'environnement d'automatisation sensible aux contextes

→ Même problématiques qu'en développement

### EXERCICE : que pensez-vous de ce script de test?

```
@Test
public void essai() {
    driver.get("https://www.laredoute.fr/login.aspx");
    driver.manage().window().setSize(new Dimension(1436, 786));
    {
        WebElement element = driver.findElement(By.id("iconProfil"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element).perform();
    }
    {
        WebElement element = driver.findElement(By.tagName("body"));
        Actions builder = new Actions(driver);
        builder.moveToElement(element, 0, 0).perform();
    }
    driver.findElement(By.linkText("Se Connecter")).click();
    driver.findElement(By.cssSelector("#inputEmail .field-label")).click();
    driver.findElement(By.id("textBox_loginPage_alreadyCustomer_loginMail")).sendKeys("dsqsq");
    driver.findElement(By.id("textBox_loginPage_alreadyCustomer_loginMail")).sendKeys(Keys.ENTER);
    assertThat(driver.findElement(By.cssSelector(".help-block:nth-child(6)")).getText(), is("Email incorrect: merci de supprime"))
}
```

### ■ Qu'est qui est un avantage du test automatique

- Meilleure utilisation des ressources de test
- Les tests sont exécutés plus lentement et permettent une meilleure analyse
- Il permet une exécution plus variée des scénarios de test grâce aux variations d'exécutions
- De nouvelles technologies peuvent être utilisées

### ■ Qu'est qui est un facteur technologique de succès

- Les interactions IHM sont fortement couplées à l'implémentation
- Les interactions IHM ne sont pas couplées à l'implémentation
- La stratégie se focalise sur les tests API
- La stratégie se focalise sur les tests IHM

### ■ Qu'est qui est une limite du test automatique

- Les oracles de test doivent être automatisable
- Le coût de la mise en place de la TAA est élevé
- Les testeurs perdent de vue les objectifs de test
- De nouvelles technologies peuvent être utilisées

### ■ Qu'est qui peut mettre en échec un projet d'automatisation

- Les interactions avec le SUT se font via une interface de type CLI
- La stratégie d'automatisation est mise en place lors de la phase de codage
- Les testeurs ont des compétences en développement
- Le framework d'automatisation permet la surveillance du SUT.

## 2. Préparation pour l'automatisation de test

### 2.1 Exigences d'automatisation de test pour le SUT et son contexte

### 2.2 Outil d'évaluation et processus de sélection

### 2.3 Conception pour la testabilité et l'automatisation

# Objectifs d'apprentissage du chapitre

## 1. Introduction et objectifs pour l'automatisation des tests

### 2.1 Exigences d'automatisation de test pour le SUT et son contexte

**ALTA-E-2.1.1 (K4) Analyser un SUT pour déterminer la solution d'automatisation optimale**

### 2.2 Outil d'évaluation et processus de sélection

**ALTA-E-2.2.1 (K4) Analyser les outils d'automatisation de test pour un projet donné et signalez les constatations et recommandations techniques**

### 2.3 Conception pour la testabilité et l'automatisation

**ALTA-E-2.3.1 (K2) Comprendre les méthodes de “conception pour la testabilité” et de “conception pour l'automatisation de test” applicables au SUT**

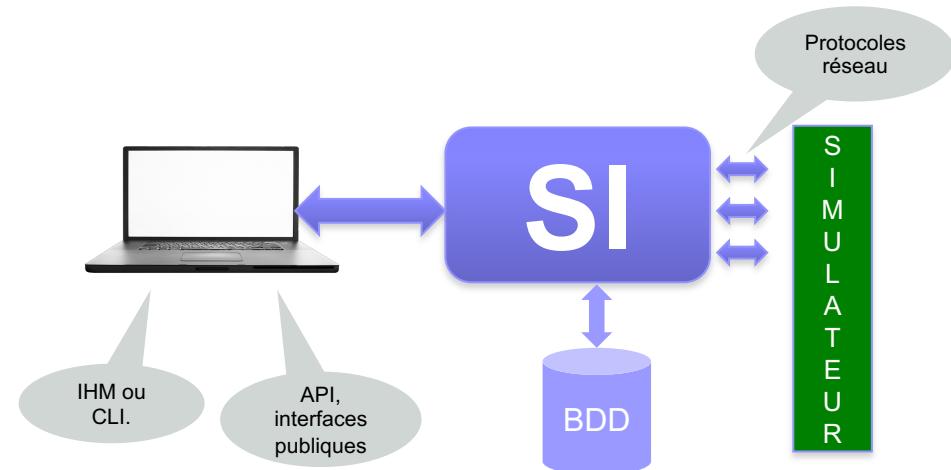
**Les Facteurs influençant l'automatisation et le choix des outils sont:**

- **Interfaces du SUT**
- **Logiciels tiers**
- **Sélection des tests à automatiser**
- **Niveau d'intrusion supporté**
- **Architectures du SUT**
- **Taille et complexité du SUT**

## 2 – Préparation pour l'automatisation de test

### 2.1- Facteurs du SUT influançant l'automatisation des tests

#### Interfaces du SUT:



- **IHM**
- **API publique**
- **API Rest ou SOAP**
- **Protocole de communication spécialisé (ex GSM)**
- **Hook (interfaces privées)**
- **Etc ..**

### 2.1- Facteurs du SUT influançant l'automatisation des tests

#### Interface du SUT=IHM

- L'automatisation se fait avec des outils spécialisés et adaptés à la technologie (WEB, WPF, Qt ...)

#### Exemple pour du web:

```
// @Test
public void testIdentification() throws Exception {
    driver.get(baseUrl + "/timesheet/");
    assertEquals("Connexion", driver.getTitle());
    driver.findElement(By.id("login")).clear();
    driver.findElement(By.id("login")).sendKeys("sqsd");
    driver.findElement(By.id("pw")).clear();
    driver.findElement(By.id("pw")).sendKeys("sqsd");
    driver.findElement(By.name("connecter")).click();
    assertEquals("Connexion", driver.getTitle());
    assertEquals(driver.findElement(By.cssSelector("p")).getText(),"login / pw invalide");
```

#### Interface du SUT= API Publique

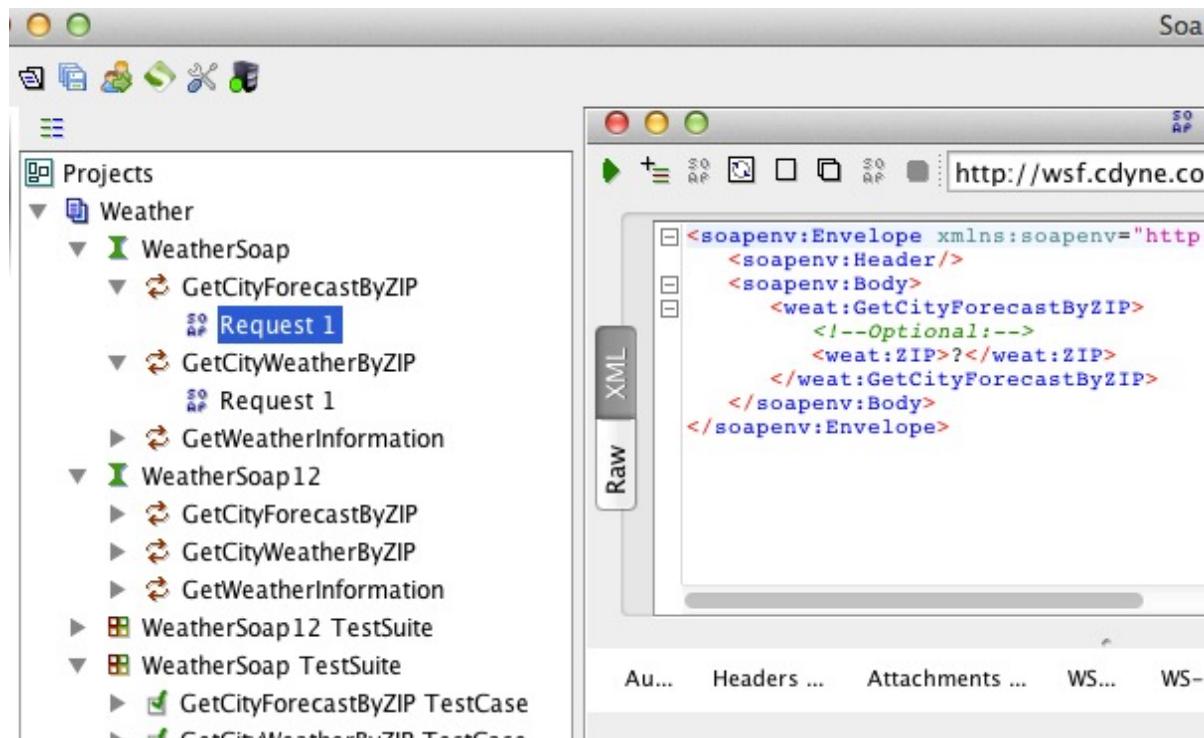
- L'automatisation se fait en général via des langages de programmation, avec des framework de test unitaire, ou avec des langages scriptés.
  - Il est possible d'automatiser des API java avec du jython
  - Il est possible d'automatiser des API C avec du python
  - Ou avec des frameworks de tests unitaires xUnit.

## 2 – Préparation pour l'automatisation de test

### 2.1- Facteurs du SUT influançant l'automatisation des tests

Interface du SUT= Rest ou Soap

- L'automatisation peut se faire via des outils spécialisés ou de la programmation via des bibliothèques adaptées.



Interface du SUT= Protocoles dédiés

- Exemple test d'un mobile GSM
- A l'aide d'un outil spécialisé il sera possible de tester par exemple la bonne réception d'un SMS au niveau protocolaire à l'aide de scripts dédiés.



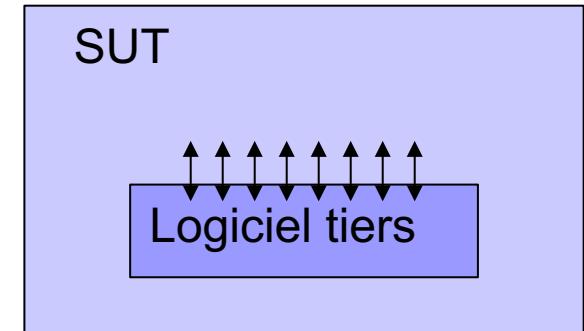
#### Interface du SUT= Hook

- **Hook ou crochet de test, ce sont des interfaces spécialisées non publiques qui permettent d'accéder à certaines fonctionnalités du système.**
- RQ: elles sont publiques pour le test
- Souvent utilisé dans les systèmes embarqués
- On peut alors accéder à des niveaux spécifiques du SUT.

#### Logiciels Tiers

- **Le SUT peut inclure des logiciels fournis par des tiers.**

- **Impact sur la stratégie de test.**
  - Cibler les interfaces avec cette API



- **Bibliothèques → éventuellement automatisation via API avant intégration (recette) : phase de test supplémentaire à définir dans la stratégie de test.**

#### Sélection de test à automatiser.

- **L'automatisation extensive n'est pas utile (ROI).**
- **Il faut choisir parmi tous les tests en fonction de critères prédéfinis dans la stratégie d'automatisation.**
- **L'idéal est de les choisir en phase de spécification.**
- **Le test basé sur le risque peut être une stratégie pour la sélection des tests à automatiser.**

## 2 – Préparation pour l'automatisation de test

### 2.1- Facteurs du SUT influançant l'automatisation des tests

#### ■ Exercice : donner des exemples de critères.

#### Sélection de test à automatiser.

- Exemple de critères:

- Est-ce un test de régression qui doit être joué à chaque livraison?
- Ce test doit-il être joué sur différents environnements?
- Criticité de la fonction associée?
- Est-ce un cas de test nominal ou un cas d'erreur?
- Le cas de test est-il simple à automatiser?
- Le cas de test est-il sujet à erreur si manuel?
- Quelle charge pour automatiser?
- Est-il joué fréquemment?
- L'automatisation du test permet-elle de le rendre plus fiable?

→ Si l'objectif initial est de fiabiliser les tests, le dernier critère aura un poids plus important.

#### Niveau d'intrusion.

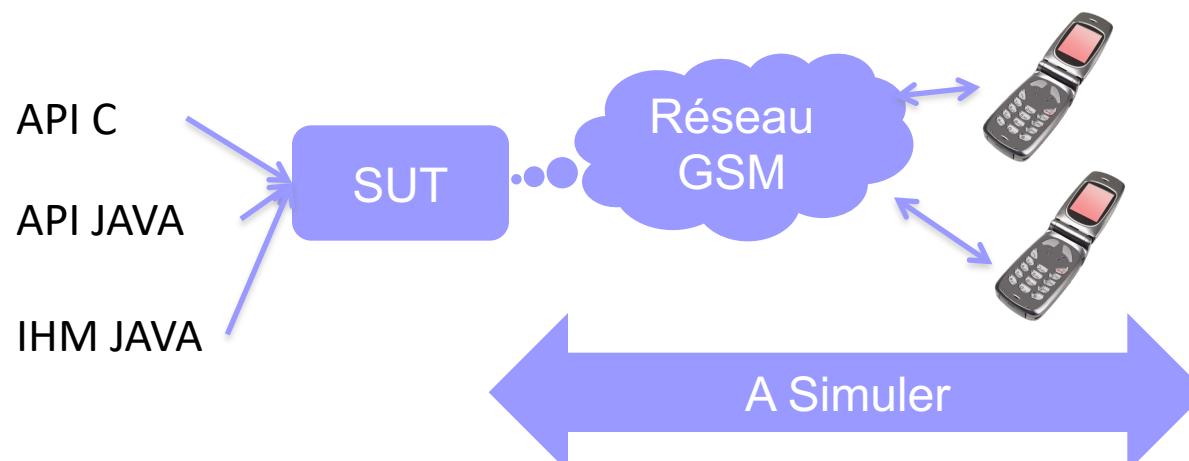
- Certains choix d'outils ou de framework de test nécessitent:
  - des modifications sur le SUT en lui-même (Hook),
  - des « capteurs d'informations » (instrumentation)
  - de recompiler le SUT avec des bibliothèques pour faire fonctionner l'outil de test
- En fonction du niveau d'intrusion le comportement du système peut être différent, par exemple en terme de performance ... et générer des fausses alertes.

#### Taille et complexité du SUT.

- La solution d'automatisation du SUT doit être adaptée à sa complexité de façon à optimiser les coûts.
  - SUT simple  $\leftrightarrow$  solution simple
  - SUT complexe interfacé avec d'autres systèmes nécessitera une solution plus complexe avec par exemple l'emploi de simulateurs.
- Question: Quelle est la différence entre un simulateur et un émulateur?

#### ■ EXEMPLE FIL ROUGE

- Editeur qui fournit des applications pour des opérateurs de téléphonie permettant de mettre à jour des informations sur une carte SIM.
  - On accède à l'application via des API C, java ou une interface client lourd.
- Les opérateurs utilisent plutôt les APIs de façon à intégrer cette application dans leur système d'information.



#### EXEMPLE FIL ROUGE:

##### □ Outil de gestion de test

- Pilotage de test d'IHM intégré.
- Pilotage possible de test API → fixtures à développer.

##### □ Outil d'automatisation IHM

- Permet d'automatiser des IHMs type Swing.
- Jeux d'essais : excel.

##### □ Outil de test d'API

- Langage de script python (API C) et jython pour les (API java)
- Possibilité de réutiliser les mêmes scripts.

##### □ Construction de la base de données

- Manuel dans un premier temps

##### □ Création d'un simulateur pilotable par les tests via les outils de test.

### 2.1- Facteurs du SUT influançant l'automatisation des tests

- Ces différents facteurs ne sont pas forcément connus en avance, néanmoins il est conseillé de penser à l'automatisation du projet en début de projet.
  - Par exemple:
    - Dès que les exigences fonctionnelles sont connues on peut procéder à la sélection des tests à automatiser.
    - Dès que l'architecture est connue, on peut identifier les interfaces logicielles utiles à l'automatisation, ou les manques ...
- Et également identifier des besoins spécifiques à l'automatisation

### 2.2- Evaluation et sélection d'outils

#### ■ Le TAM: test automation manager

- Sélection outil
- Processus information

#### ■ Le TAE: test automation engineer

- Évaluation de la maturité organisationnelle et identification des opportunités pour le support d'outils de test
- Évaluation des objectifs appropriés pour le support d'outils de test
- Identification et collecte d'informations sur les outils qui pourraient convenir
- Analyse des informations sur ces outils par rapport aux objectifs et contraintes du projet
- Estimation du rapport cout/bénéfices basé sur une analyse solide de rentabilité (ROI)
- Émission de recommandations sur l'outil approprié
- Identification de la compatibilité de l'outil avec les composants du SUT

### 2.2- Evaluation et sélection d'outils

- Estimation du rapport cout/bénéfices basé sur une analyse solide de rentabilité
  - Il est nécessaire d'évaluer les bénéfices potentiels et de s'assurer des gains potentiels à court et long terme.  
→ Attention au vision à court terme
- Émission de recommandations sur l'outil approprié
  - Pour améliorer le ROI
  - Utiliser de façon homogène l'outil

### 2.2- Evaluation et sélection d'outils

- Identification et collecte d'information sur les outils qui pourraient convenir
  - Cela peut passer par des preuves de concepts des outils identifiés comme potentiels candidats
  - Via des forums
  - Des démonstrations
- Analyse des informations sur ces outils par rapport aux objectifs et contraintes du projet
  - Les outils sont-ils appropriés aux équipes projets?
  - Permettent-ils d'atteindre facilement les objectifs/périmètre d'automatisation?
  - Limites?



### 2.2- Evaluation et sélection d'outils

Exercices sur les potentiels problèmes rencontrés lors d'une évaluation d'outil.

Problèmes	Solutions
1. L'objet dans l'IHM ne peut pas être capturé.	a. Demander du support au vendeur.
2. Impact sur le SUT (ex plus lent)	b. Faire un pilote au préalable.
3. L'outil a l'air très compliqué	c. Choisir un outil qui ne nécessite pas de modifier le SUT.
4. Les interfaces de l'outil à l'étude ne fonctionne pas avec les outils existants	d. Limiter les fonctionnalités de l'outil.

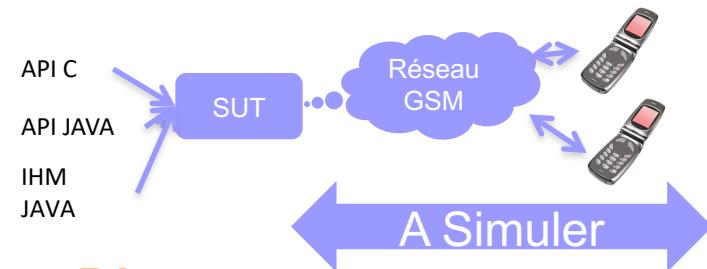
### 2.2- Evaluation et sélection d'outils

Exercices sur les potentiels problèmes rencontrés lors d'une évaluation d'outil.

Problèmes	Solutions
1. Le code du SUT doit être modifié.	a. Donner des droits supplémentaires sur les postes des testeurs.
2. Conflit avec d'autres systèmes	b. Demander sur les forums.
3. Ressources limitées sur un environnement embarqué.	c. Choisir un outil qui ne nécessite pas de modifier le SUT.
4. Sécurité (droit d'accès)	d. Lire les notes de version

### ■ EXEMPLE FIL ROUGE

- 2 outils:



- WINRUNNER combiné à Test Director

- ❖ Outils connus sur le marché
- ❖ Support en français
- ❖ POC possible avec support
- ❖ Auto + spécification de tests

- Silk Test

- ❖ Moins connus
- ❖ Plus de possibilités techniques
- ❖ Support complexe uniquement anglais

#### ■ La testabilité d'un SUT comporte 3 volets:

##### □ L'observabilité

- ❖ Pour évaluer le résultat d'un test il faut pouvoir observer ou mesurer l'effet d'une action sur le SUT.

##### □ La contrôlabilité

- ❖ Il faut pouvoir réaliser les actions sur le SUT, via une interface, des API ou autres.

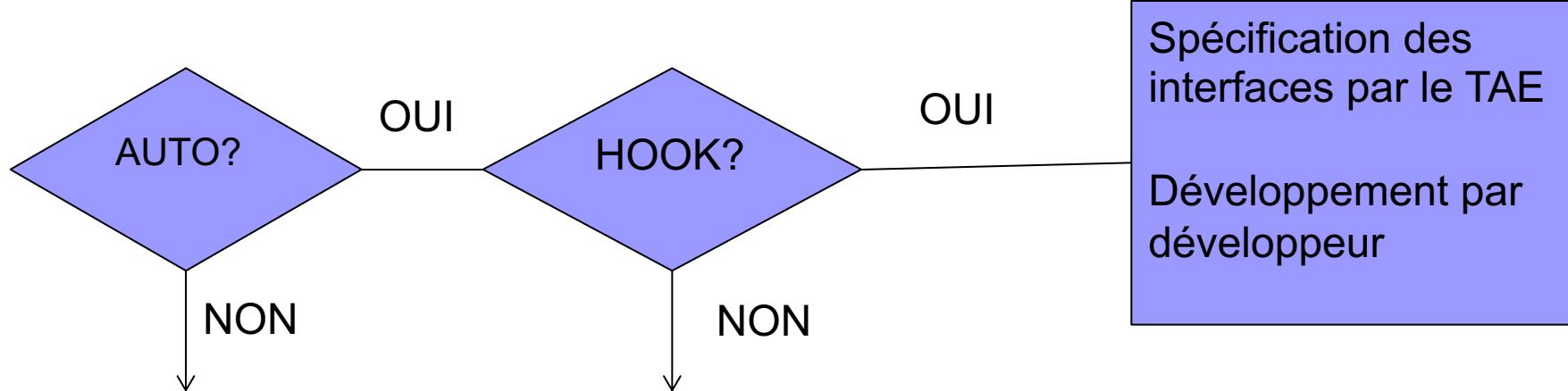
##### □ La connaissance de l'architecture

- ❖ Pour définir l'architecture de test, il est nécessaire d'avoir de la visibilité sur les différentes interfaces du SUT.

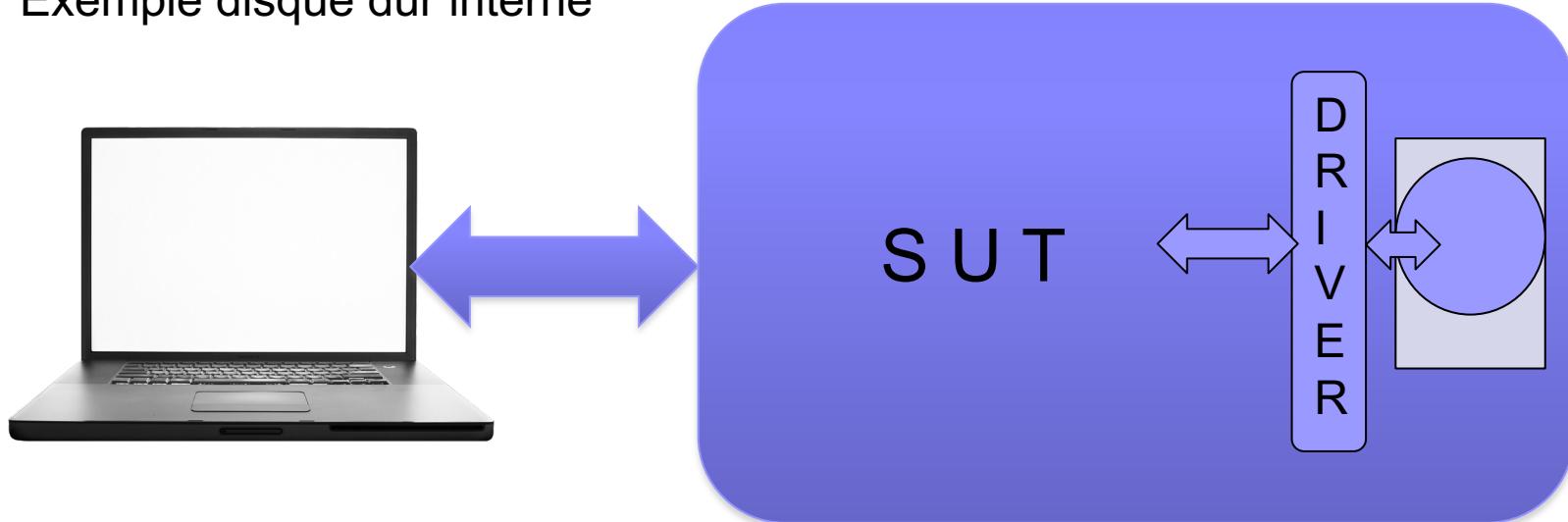
→ L'architecte et le TAE sont impliqués dans la prise en compte et l'amélioration de ces critères.

## 2 – Préparation pour l'automatisation de test

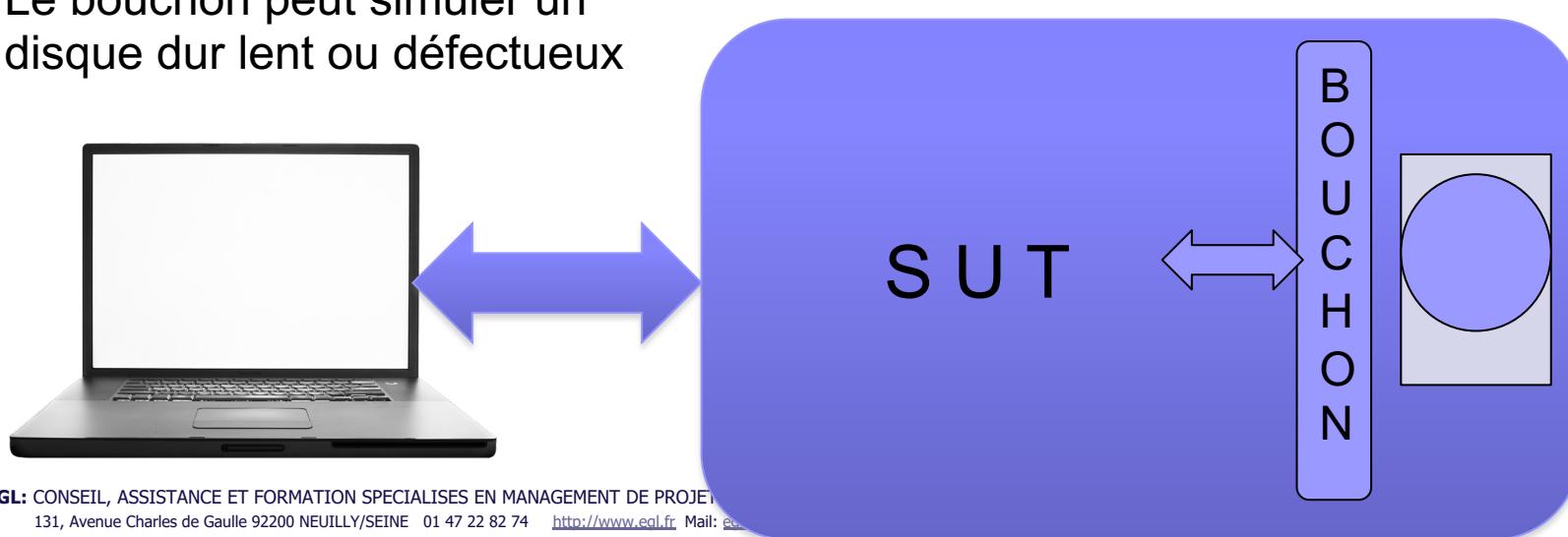
### 2.3- Conception pour testabilité et automatisation



Exemple disque dur interne



Le bouchon peut simuler un disque dur lent ou défectueux



Exemple de solution pour une interface externe non disponible



MOCK

Le simulacre ou bouchon intercepte les messages et répond en fonction du test.

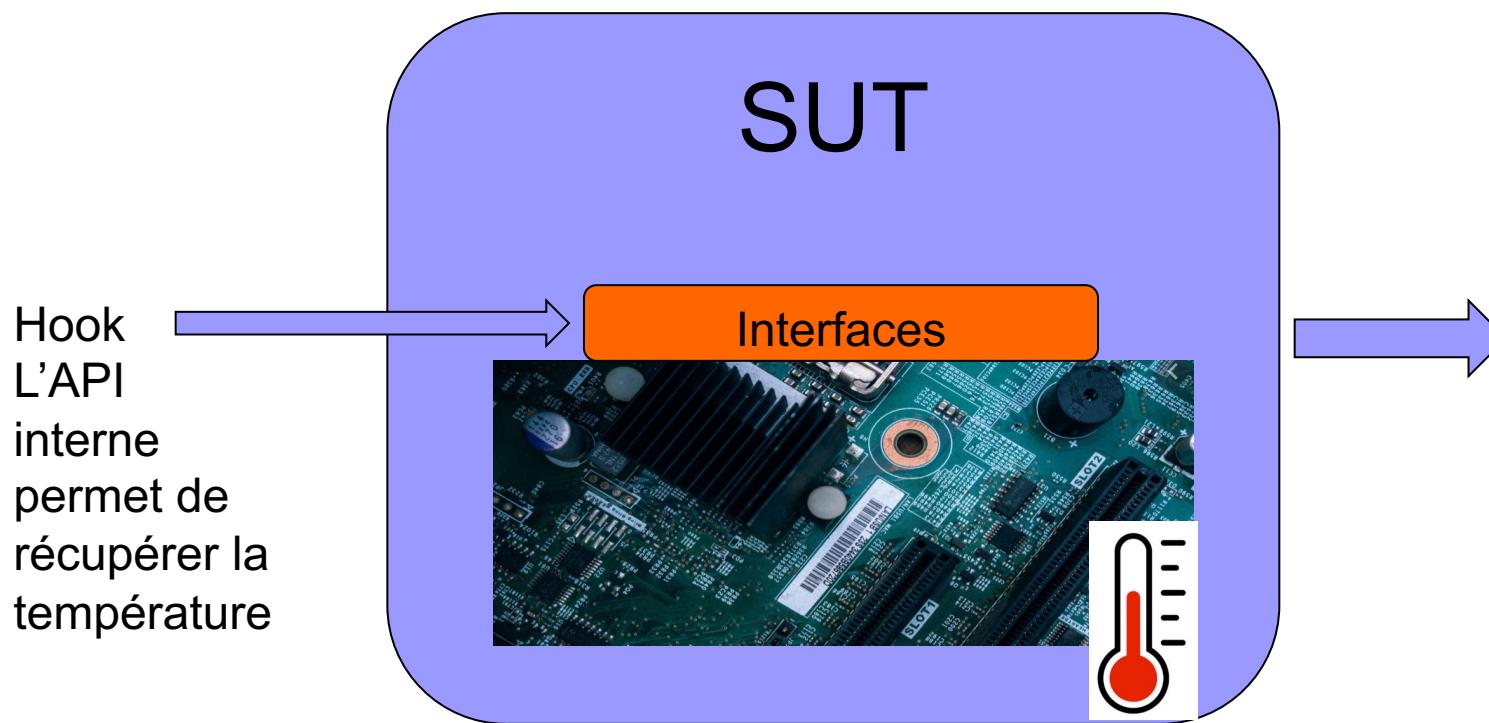
→ Inconvenient: le SUT est modifié.

Ce type de solution permet :

Simuler des interfaces logicielles non disponibles

Remplacer des interfaces matérielles non disponibles

#### Exemple de solution



#### ■ Ces interfaces spécifiques permettent:

- Simuler des parties logicielles et générer tout types d'erreur
- Remonter tout types de valeurs normalement remontées par le matériel
- Piloter le SUT quand aucune interface n'est disponible
- Remonter des états internes du système (test de transition d'état).

#### ■ Attention

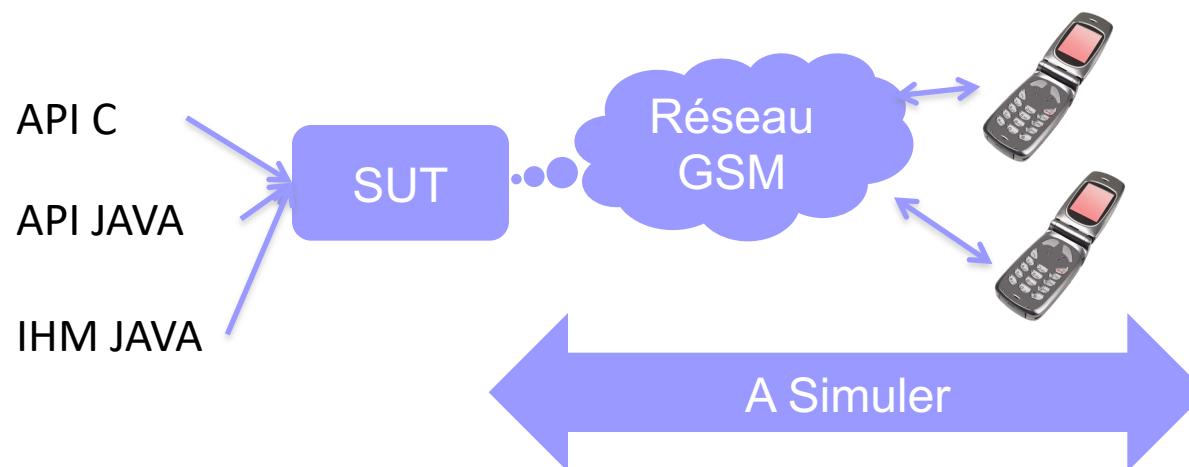
- Le SUT est modifié, le comportement en particulier, temps réel peut être modifié.

#### ■ En résumé

- La conception du framework d'automatisation devrait commencer au plus tôt
- Il est nécessaire de trouver les outils de tests adapté aux interfaces du SUT, sous peine de difficultés à piloter et contrôler le SUT → pénalisant pour les activités d'automatisation.
- Certains SUT nécessitent le développement d'interfaces spécialisées pour l'automatisation.

#### ■ EXEMPLE FIL ROUGE

- En attendant le développement d'un simulateur quelle autre solution serait possible pour automatiser les tests?
- Pour l'automatisation des tests d'API de quoi a-t-on besoin?



## 2 – Préparation pour l'automatisation de test

### Exercice

- Vous travaillez sur un projet qui intègre un software développé en interne avec un logiciel tier. Ce dernier fournit un package de comptabilité qui va être utilisé par votre société. Le projet développé en interne est un produit de point de vente qui va être utilisé dans les différents magasins de détail. L'interface entre les 2 produits se fait par des fichiers batch et API. Les fonctionnalités de reporting du package de compta vont être utilisées pour prendre les décisions budgétaires. Etant donné ces informations quel est l'approche d'automatisation la plus pertinente pour le logiciel tier.
  - a) Pas d'automatisation, car il devrait être testé par les développeurs du produit
  - b) Les tests devraient être limités au reporting, car critique
  - c) Le test devrait se concentrer sur les API utilisées pour passer les données au logiciel tiers, pour s'assurer que les données sont traitées correctement.
  - d) Le test de la préparation des fichiers batch devrait être sur le produit point de vente pour s'assurer que les bonnes données sont incluses dans les fichiers batch.

## 2 – Préparation pour l'automatisation de test

### Exercice

**Vous avez été chargé de mettre en œuvre l'automatisation des tests pour un système qui a les capacités suivantes :**

Reçoit les messages des dispositifs matériels

Traite les messages d'erreur en fonction d'un moteur de règles

Envoi d'un courrier électronique à l'administrateur système avec des recommandations de mises à jour/changements du matériel ou de ses paramètres

**Vous avez étudié les alternatives d'automatisation des tests et vous avez découvert que vous pouvez automatiser la génération de messages, la réception de messages et le traitement des messages par le biais du moteur de règles.**

L'entreprise a investi une somme importante dans un outil d'automatisation des tests, mais l'outil n'est pas capable de récupérer le message du serveur de messagerie et de vérifier que la recommandation est correcte.

**Vous recherchez une solution d'automatisation complète. Que devez-vous faire ?**

- a. Rechercher un nouvel outil qui puisse accomplir toutes les tâches nécessaires
- b. Recherchez un outil complémentaire qui sera en mesure de gérer la validation des e-mails
- c. Vérifier que le traitement des règles est correct et supposer que l'e-mail sera préparé correctement.
- d. Vérifier manuellement les courriels en les faisant envoyer à votre adresse électronique

## 2 – Préparation pour l'automatisation de test

### Exercice

- Vous avez travaillé avec l'outil standard d'automatisation des tests de votre entreprise sur une nouvelle application mobile développée en interne. Cet outil est utilisé depuis plusieurs années et prend en charge l'automatisation des tests pour 20 applications majeures, dont cinq autres applications mobiles. La nouvelle application est dotée d'une interface utilisateur conviviale mais utilise un objet de tableau que votre outil ne peut pas reconnaître. Par conséquent, vous n'avez aucun moyen de vérifier que les données indiquées dans le tableau sont correctes. Quelle est la première chose que vous devez faire pour résoudre ce problème ?
  - a. Voir si l'application peut être exécutée sur un PC et si le tableau peut être reconnu lorsque l'application est utilisée sur le PC.
  - b. Effectuer un test pilote sur un échantillon d'outils de pointe de l'industrie pour voir si l'objet peut être reconnu
  - c. Demandez aux développeurs de supprimer l'objet tableau et de le remplacer par un ensemble de champs de texte
  - d. Demandez aux développeurs de changer l'objet tableau pour un objet tableau standard qui pourrait être reconnu par l'outil

### Exercice

- Quels sont les deux considérations les plus importantes pour la testabilité d'un SUT :
  - a) Contrôlabilité et capacité de changement
  - b) capacité de changement et Maintenabilité
  - c) Maintenabilité et observabilité
  - d) Observabilité et contrôlabilité
- Qu'est qui est important en terme de testabilité lors de la conception du SUT:
  - Compatibilité avec les outils d'automatisation existants
  - Contrôlabilité par des opérateurs
  - Maintenabilité par les développeurs
  - Capacité de changement utilisant les outils d'automatisation existants.

## 3.1- Introduction à la gTAA

### 3.1 Introduction à la gTAA

### 3.2 Conception de l'architecture

### 3.3 Développement de l'architecture

### 3.4 Synchronisation de l'architecture avec l'objet en test

### 3.5 Déployer l'architecture

### 3.1 Introduction à la gTAA

ALTA-E-3.1.1 (K2) Expliquer la structure de la gTAA

### 3.2 Conception d'une TAA

ALTA-E-3.2.1 (K4) Concevoir une TAA appropriée pour un projet donné

ALTA-E-3.2.2 (K2) Expliquer le rôle joué par les couches dans une TAA

ALTA-E-3.2.3 (K2) Comprendre des considérations de conception pour une TAA

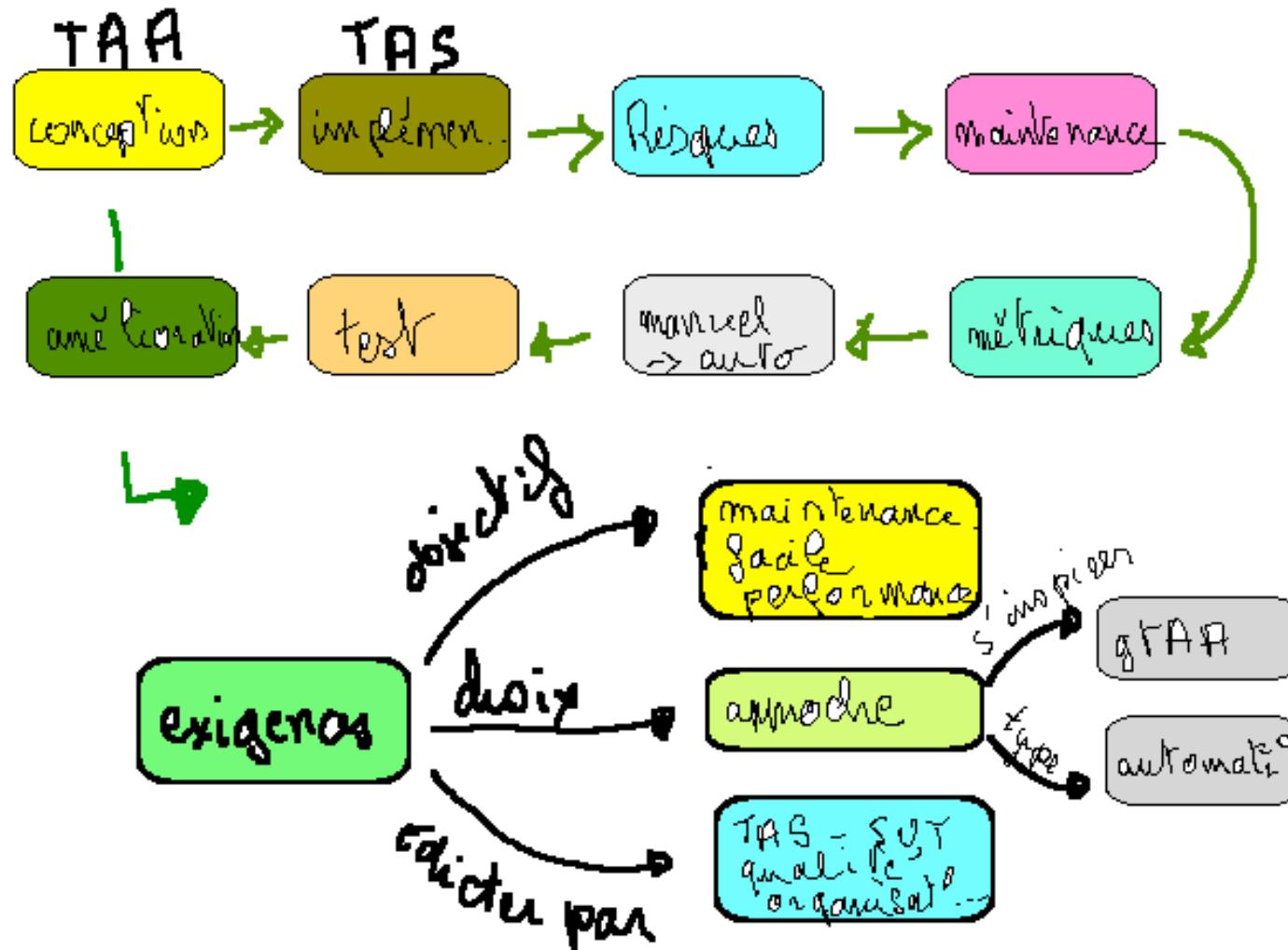
ALTA-E-3.2.4 (K4) Analyser les facteurs de l'implémentation, de l'utilisation, et de la maintenance

(exigences) pour un programme de TAA donné

### 3.3 Développement d'une TAS

ALTA-E-3.3.1 (K3) Appliquer des composantes de la TAA générique (gTAA) pour construire une TAA

ALTA-E-3.3.2 (K2) Expliquer les facteurs à considérer lors de l'identification des composants à réutiliser



- Il s'agit de définir une architecture pour la solution d'automatisation avec les caractéristiques suivantes:

- **Responsabilité Unique**

- ❖ Un composant de la TAS doit n'être en charge que d'une chose comme faire le reporting ...

- **Extension**

- ❖ Pas de modification directe du composant mais plutôt une extension

- **Remplacement**

- ❖ Possibilité de remplacer un composant par un autre

- **Principes de ségrégation**

- ❖ Composants spécifiques plutôt qu'un couteau suisse

- **Inversion de dépendance**

- ❖ abstraction

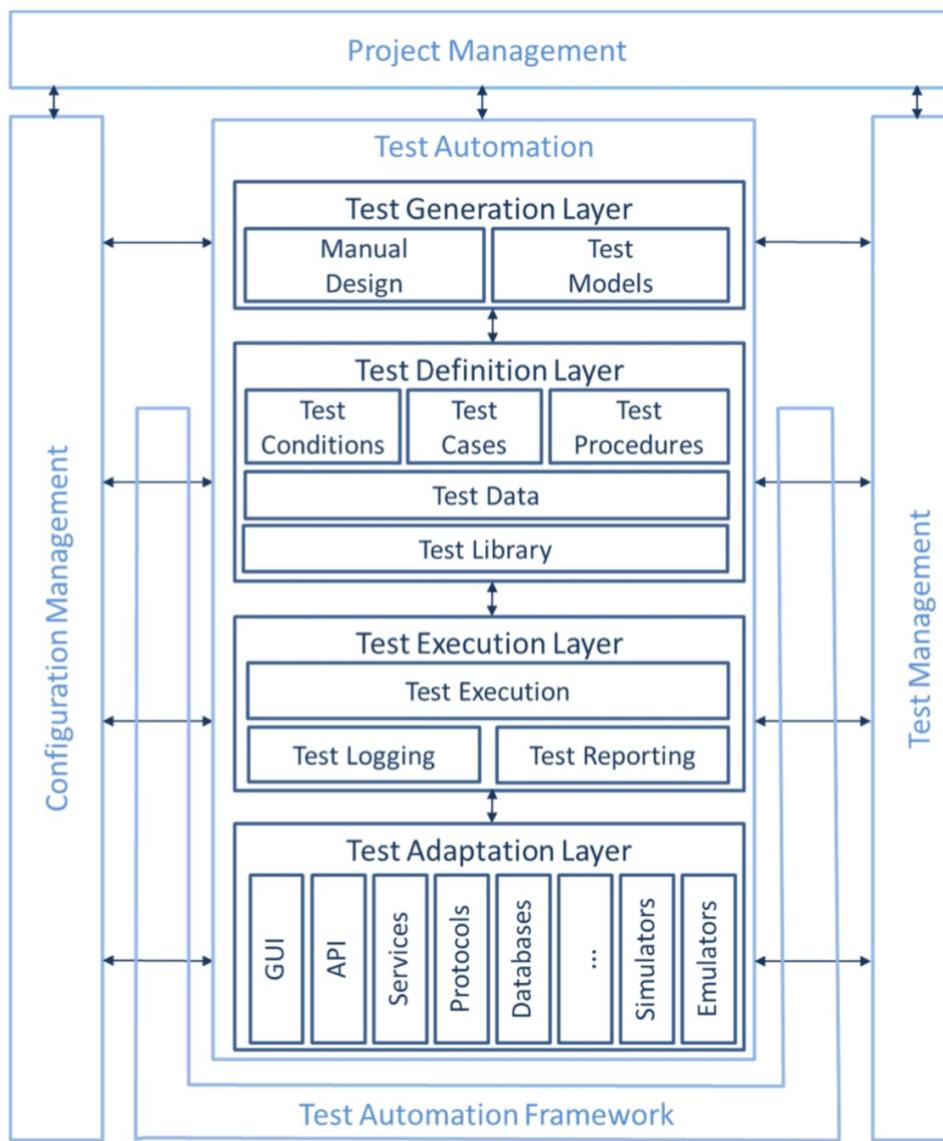
- **gTAA doit être indépendante des fournisseurs**
- **Pas de méthodes concrètes**

- Simple
- Evolutive
- Réutilisabilité
- Maintenance

# 3-Architecture générique d'automatisation de tests



## 3.1- Introduction à la gTAA 3.1.1- Vue d'ensemble



- **Couche génération de tests:**
  - Manuel
  - Modèle
- **Couche définition de tests:**
  - Conditions de test
  - Cas de test
  - Procédures de test
  - Données de test
  - Bibliothèque de test
- **Couche exécution de tests:**
  - Exécution de test
  - Logs de test
  - Rapport de test
- **Couche adaptation de tests:**
  - Gui, api, services, BDD, Protocoles ...



- **En fonction des choix toutes les couches ne sont pas nécessaires:**

- Par exemple pour les tests unitaires on ne va trouver que la couche exécution et adaptation.
  - Si la génération de test doit être automatisée on va trouver cette couche avec des outils style (MBT).
  - Si la couche définition de test doit être automatisée, cette couche sera obligatoirement présente (outils basés mots-clefs) ou collaboratif (fitness).

- A ce niveau on va utiliser des outils pour spécifier les tests soit:

- Manuellement avec des outils de gestion de test

- ❖ Organiser les tests en suite de test
    - ❖ Lier les tests aux objectifs de test

- Automatiquement:

- 1. Modélisation de l'application à tester
    2. Génération des tests à l'aide d'un outil à partir de directives ...

-  projet1 (14)
  - ▶  Technique (3)
  - ◀  fonctionnel (11)
    - ◀  Gestion des posts (8)
      -  projet1-4:Création de post
      -  projet1-5:Réponse à un post
      -  projet1-6:Fermeture d'un post - 30 réponses
      -  projet1-7:Acceptation et rejet d'un post
      -  projet1-8:Réponse à un post 2
      -  projet1-12:Fermeture d'un post 20 jours
      -  projet1-13:Fermeture d'un post 19 jours
      -  projet1-14:Fermeture d'un post 29 réponses
    - ◀  Gestion des comptes (3)
      -  projet1-9:page après login succès
      -  projet1-10:Scroll
      -  projet1-11:page après login echec

## ■ Exemple de structuration avec un outil de gestion de test (testlink).

# 3-Architecture générique d'automatisation de tests



## 3.1- Introduction à la gTAA

### 3.1.2 : couche de génération de test

projet1-6 : Fermeture d'un post - 30 réponses - Version 1 🚧 ⏱

#### Summary

obj : tester la fermeture de la discussion au bout de 30 réponses au post.

Méthode de test utilisé :

-tests aux limites

Données nécessaires à la réalisation du test: un post avec 29 réponses

#### Preconditions



#### Step actions

#### Expected Results

#### Execution

1 Prérequis: être sur un post avec déjà 29 réponses.

Manual



2 Saisir la 30 ième réponse.

La discussion est automatiquement fermée:

Manual



1. Vérifier le statut du post.
2. Il est impossible de saisir une nouvelle réponse.

Create step Resequence Steps

Status : Draft

Importance : Medium

Execution type : Manual



Estimated exec. (min) :  Save



Objectif du test

Etapes du test

Manuel ou auto

Couverture

Keywords: None

Requirements 📜 : 🖊 [Gestion des posts] fct004 [v1] : Fermeture d'un post



- Organiser les tests
  - Identifier les tests à automatiser
  - Lier les tests aux exigences
  - Spécifier les étapes de tests
- 
- Choix des outils: intégrer les outils
    - Vérifier que l'on peut connecter l'outil de gestion au framework de test automatique via
      - ❖ API (ex. REST, SOAP, RCP ...)
      - ❖ Des plugins qui permettent la liaison entre des outils de gestion de test et les outils d'intégration continue.

## ■ Dès la spécification de test on pense automatisation

- Identifier les prérequis
  - Identifier les contrôles
  - Identifier les étapes qui se répètent
  - Organiser les suites de test
- 
- Permet un passage de relais plus simple entre le concepteur de test et le testeur automatique
  - Etape obligatoire, comme en développement, on spécifie avant de coder!

## ■ Les outils de génération automatiques de test (MBT)

- Les tests sont générés automatiquement à partir d'un modèle.

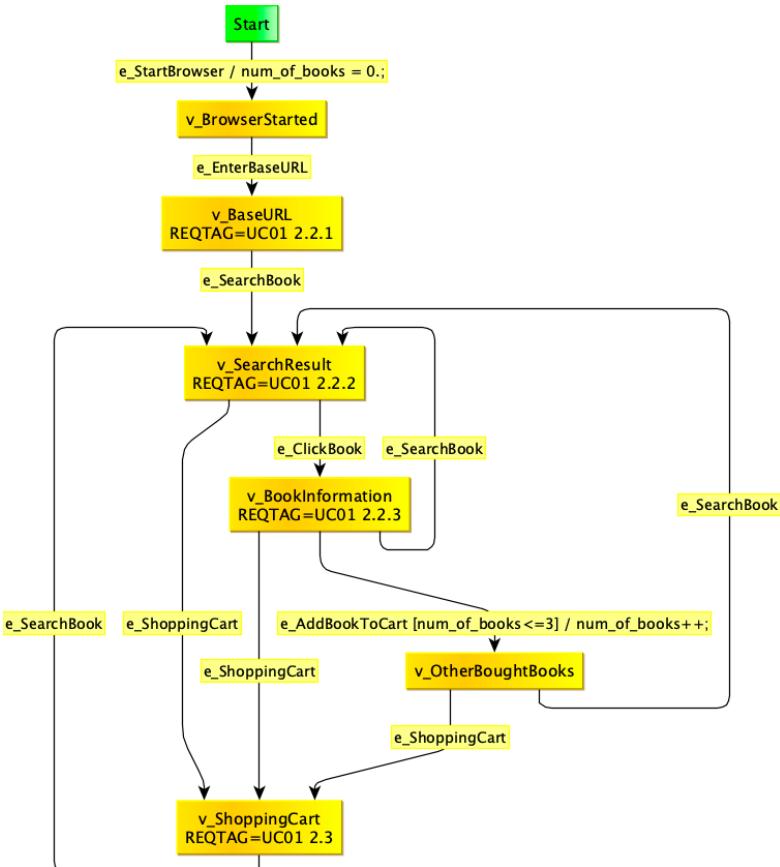
## ■ Exemple avec une modélisation type graphe à états

- Modélisation de façon visuelle du comportement de l'application (diagramme à états)
- Application de règles de génération de cas de test
- Un algorithme mathématique calcule alors les chemins.

# 3-Architecture générique d'automatisation de tests



## 3.1- Introduction à la gTAA 3.1.2 : couche de génération de test



- Les arcs représentent des actions.
- Dans les nœuds on trouve les contrôles et les exigences couvertes.
- L'outil va calculer des chemins en fonction des objectifs de couverture et stratégie de parcours de chemin.

## ■ Action: ajout d'un élément au panier

```
public void e_AddBookToCart() {  
    waiter.until(ExpectedConditions.presenceOfElementLocated(By.id("mediaTab_heading_1")));  
    waiter.until(ExpectedConditions.elementToBeClickable(By.id("mediaTab_heading_1"))).click();  
    waiter.until(ExpectedConditions.presenceOfElementLocated(By.id("newOfferAccordionRow")));  
    waiter.until(ExpectedConditions.elementToBeClickable(By.id("newOfferAccordionRow"))).click();  
    waiter.until(ExpectedConditions.presenceOfElementLocated(By.id("add-to-cart-button")));  
    waiter.until(ExpectedConditions.elementToBeClickable(By.id("add-to-cart-button"))).click();  
    logger.debug("Number of added books by test: " + ++numberOfAddedBooksByProgram);  
}
```

❖ Action: elle est codée en java avec Selenium Webdriver.

→ Fixtures.

## ■ Contrôle: informations sur livres

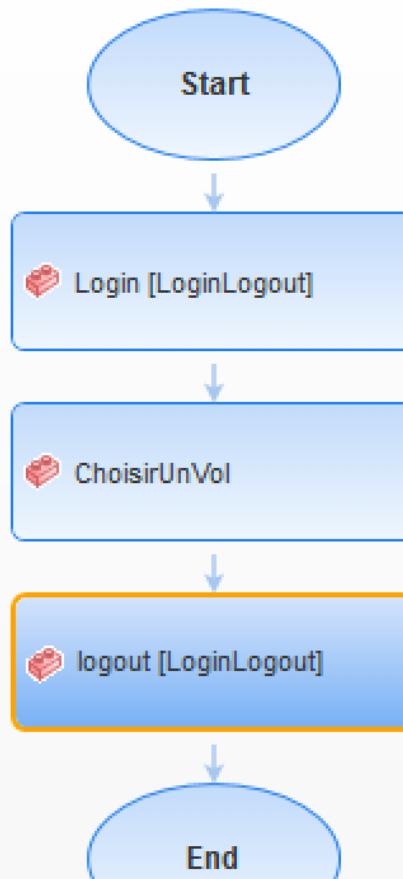
```
public void v_BookInformation() {  
    waiter.until(ExpectedConditions.textMatches(By.tagName("h1"),  
        Pattern.compile("Practical Model-Based Testing: A Tools Approach .*")));  
    waiter.until(ExpectedConditions.visibilityOfElementLocated(By.id("mediaTabsGroup")));  
    waiter.until(ExpectedConditions.visibilityOfElementLocated(By.className("navFooterVerticalColumn")));  
}
```

❖ Contrôle: il codé en java avec Selenium Webdriver.

→ Fixtures.

- **Objectif: définir une couche abstraite qui:**
  - n'expose pas les aspects techniques du SUT.
  - permet de gérer les données de test.
- **Cette couche permet de rendre indépendant les tests du SUT→ maintenabilité des tests.**
- **Elle permet de formaliser les tests avec une vision métier.**
- **Elle permet aux métiers de spécifier les tests avec un langage « automatisation » haut niveau type DSL, Domain Specific Langage.**

Exemple de Séquence/cas de test avec UFT.



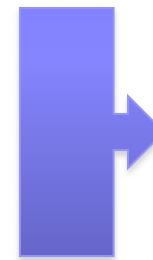
- **Spécification de test sous format visuel, avec réutilisation d'actions métier de façon séquentielle.**
- **La partie accès au SUT est masquée par les actions de haut.**
  - Ici on ne sait pas si on attaque une interface WEB, un client lourd ...

Exemple de Séquence/cas de test avec RobotFramework.

#### \*\*\* Test Cases \*\*\*

T1

```
[Documentation] cas nominal
[Tags] NONREG
Un utilisateur indecis
je fais une recherche avec le mot cle jupes
je trouve 1 résultat a été trouvé.
```



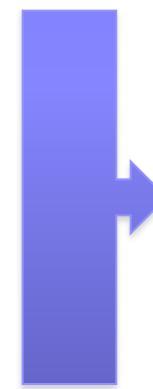
Le testeur utilise des mots-clefs haut niveau pour définir son test, le test peut être documenté et caractérisé.

#### \*\*\* Keywords \*\*\*

```
Un utilisateur indecis
AllerALaPageDaccueil
```

```
je fais une recherche avec le mot cle [Arguments] ${mclef}
lancerUneRecherche ${mclef}
```

```
lancerUneRecherche [Arguments] ${mclef}
Input Text id=search_query_top ${mclef}
Click Element name=submit_search
```



Les mots-clefs implémentés peuvent être fournis sous forme de bibliothèques.

## 3.1- Introduction à la gTAA

### 3.1.3- Couche de définition des tests

```
// Create a test using SignInPage and PageProject to check successful login
//@Test
public void testValidLogin() throws Exception {
    // Actions action;

    PageProjectTest page;
    page = log.LoginValidUser("at", "at", driver);
    assertTrue(page.estLogge());
}

// create a test using SignInPag to check unsuccessful login
//@Test
public void testInValidLogin() throws Exception {
    assertTrue(log.LoginInvalidUser("to", "toto"));
}

// optional create a test checking presence of project
//@Test
public void testProjetExiste() throws Exception {
    PageProjectTest page;
    page = log.LoginValidUser("at", "at", driver);

    assertTrue(page.projetExiste("refonte de notre site pour le passer sous CodeIgniter"));
}
```

Exemple avec  
java+webdriver.  
L'utilisation du concept de  
page object permet de  
séparer la logique du test de  
l'implémentation du test. La  
maintenance en est facilitée.

- **modalités d'insertion des jeux de données en base,**
  - **modalités de vérifications des données en sortie des tests**
- Choix du formalisme:

- ❖ Excel
- ❖ CSV
- ❖ XML
- ❖ Ou autre

E1	E2	E3	E4	....	S1	S2	....
V1	V2	V3	V4		V5	V6	
....	....	....	....		....	....	

- **Spécification de l'outillage pour insérer les données et contrôler les données.**
- **Règle de stockage des jeux d'essais et références.**
  - ❖ Partage des jeux d'essais entre l'outil de gestion de test (ex: pièce jointe) et les scripts d'automatisation.

```
import com.calculFrais.*;  
  
@RunWith(Parameterized.class)  
public class TestCalculFrais4 {  
  
    @Parameters public static Collection<Object[]> val() throws ParserConfigurationException, SAXException, IOException{  
        List<Object[]> TestData = new ArrayList<Object[]>();  
        final DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
        final DocumentBuilder builder = factory.newDocumentBuilder();  
        final Document document= builder.parse(new File("C:/calculFrais/Donnees.xml"));  
        final NodeList jeux = document.getElementsByTagName("jeux");  
        for(int j = 0; j<jeux.getLength(); j++) {  
            final Element jeu = (Element) jeux.item(j);  
            TestData.add(new Object[]{ Double.parseDouble(jeu.getAttribute("total")), Double.parseDouble(jeu.getAttribute("montant")) } );  
        }  
  
        return TestData;  
    }  
  
    //    return Arrays.asList(new Object [][] {{700.0,0.0},{50.0,30.0}});
```

#### ■ Ex avec Sahi:

```
_include("C:/SQUASH-TA/sahi_v50_20141105/userdat");
function doTest($name,$pwd)
{
    _navigateTo("http://dgu-PC/timesheet");
    seLogger($name,$pwd)
    controle("login / pw invalide")
}
var $data = _readCSVFile("data.csv");
_dataDrive(doTest, $data)
```

Ce fichier peut également être une pièce jointe de la spécification de test dans l'outil de gestion de test

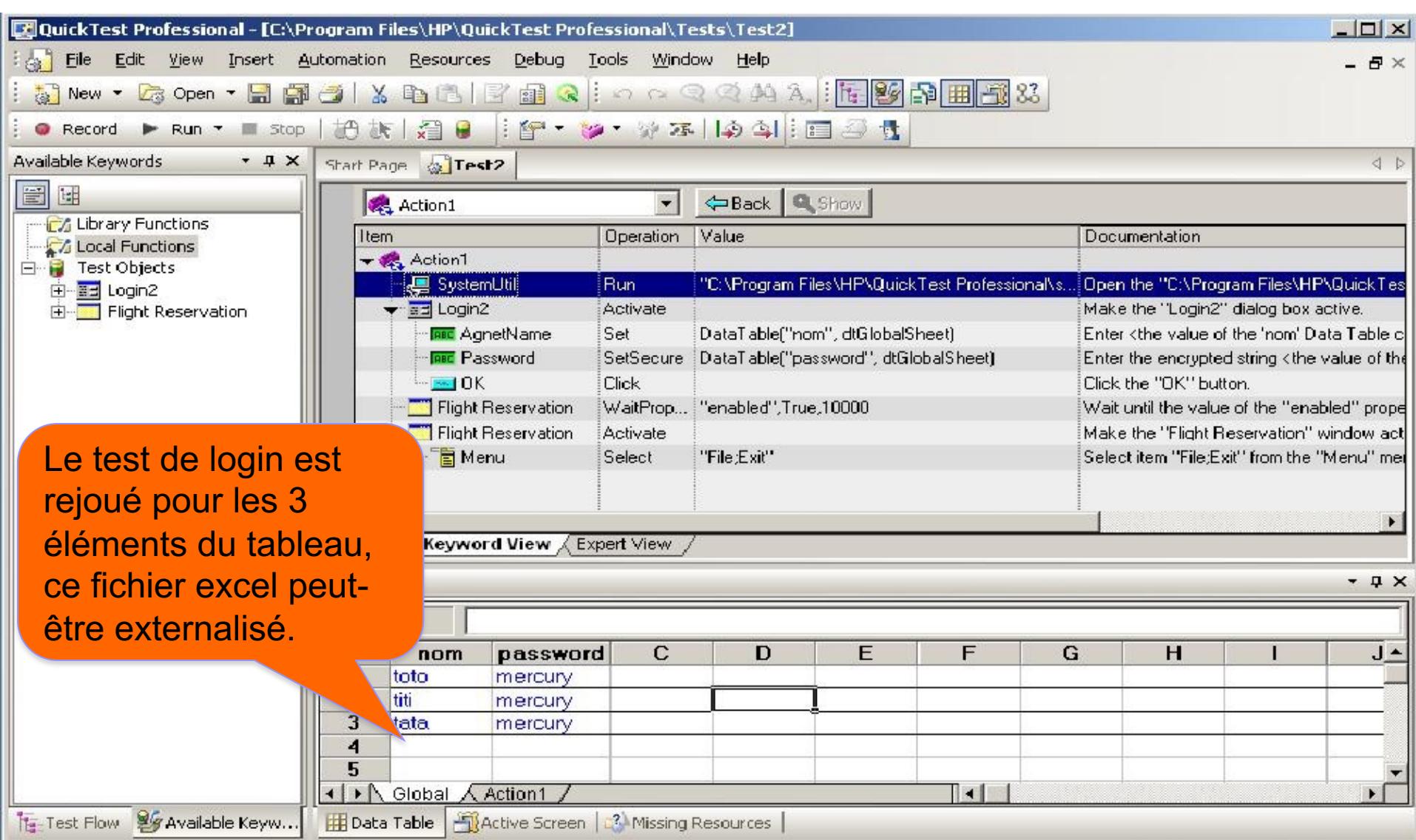
## ■ Intérêt des jeux d'essais définis à l'extérieur des scripts d'automatisation

- Permet au métier de mettre à jour les données sans modifier les scripts d'automatisation
- Permet d'être exhaustif au niveau des jeux d'essais

## ■ Pour aller plus loin

- Introduire une part d'aléatoire dans la génération des valeurs.
- Utiliser un générateur de données:
- ex <http://www.generatedata.com>

	siret	nom	adresse
1	440455269	Dolor Elit Corp.	123 Nulla Av.
2	341551240	Scelerisque Industries	6173 Vitae Rd.
3	691614226	Vehicula Pellentesque Ltd	Appartement 463-4804 ipsum, Chemin
4	165312786	Quis Massa Foundation	Appartement 155-7538 Mi, Impasse
5	221662315	Duis A Mi PC	739-9685 Cursus Rd.
6	278107842	Sit Amet LLP	Appartement 613-2199 Vitae Route
7	616335485	Ultrices Consulting	CP 396, 825 Mauris Rue
8	842315459	Pede Malesuada Vel Foundation	CP 149, 652 Vulputate Rue
9	403081854	Enim Mi Tempor Ltd	545-6072 Portitor Ave
10	263266660	Molestie Sodales Mauris Corporation	626-6969 Consectetur Ave
11	525338927	Iaculis PC	565-6784 Quisque Chemin
12	349401307	Risus Donec Egestas Corp.	947-2014 Eu Ave
13	017880642	In Tincidunt Incorporated	8554 Condimentum Route
14	842877136	Enim Diam Vel Ltd	Appartement 297-1997 Ligula. Ave



Le test de login est rejoué pour les 3 éléments du tableau, ce fichier excel peut-être externalisé.

#### 3.1- Introduction à la gTAA

##### 3.1.3- Couche de définition des tests

Formalisation de test à l'aide d'une table de décision (Fitness). L'outil permet à la fois de spécifier les tests puis à l'aide de Fixture (implémentation) d'exécuter les tests

should I buy milk			
cash in wallet	credit card	pints of milk remaining	go to store?
0	no	0	no
10	no	0	yes
0	yes	0	yes
10	yes	0	yes
0	no	1	no
10	no	1	no
0	yes	1	no
10	yes	1	nope

# 3-Architecture générique d'automatisation de tests



## 3.1- Introduction à la gTAA

### 3.1.3- Couche de définition des tests

```
package milk;
import java.util.*;

public class ShouldIBuyMilk {
    private int dollars;
    private int pints;
    private boolean creditCard;

    public void setCashInWallet(int dollars) {
        this.dollars = dollars;
    }
    public void setPintsOfMilkRemaining(int pints) {
        this.pints = pints;
    }
    public void setCreditCard(String valid) {
        creditCard = "yes".equals(valid);
    }
    public String goToStore() {
        return (pints == 0 && (dollars > 2 || creditCard)) ? "yes" : "no";
    }
}
```

should I buy milk			
cash in wallet	credit card	pints of milk remaining	go to store?
0	no	0	no
10	no	0	yes
0	yes	0	yes
10	yes	0	yes
0	no	1	no
10	no	1	no
0	yes	1	no
10	yes	1	nope

#### Exécution des tests

##### ■ Objectifs:

- Exécuter les tests automatiquement.
- Tracer l'exécution des tests
- Reporter les résultats de tests

##### ■ Framework de test avec fonctionnalités de

- Set Up et Tear Down
- Comparaison de résultats (attendus versus obtenus)
- Traces

## ■ Set Up et Tear Down

- Permettent de capitaliser des actions communes avant et après chaque test ou des suites de test.

```
@Before  
public void setUp() throws Exception {  
  
    System.setProperty("webdriver.chrome.driver", "C:/chrome/chromedriver.exe");  
    driver = new ChromeDriver();  
    baseUrl = "http://www.qualifiez.fr/examples/Selenium/";  
    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);  
    driver.get(baseUrl + "");  
    log = PageFactory.initElements(driver, SignInPageTest.class);  
};
```

**@Before Le « set up » permet ici de créer un driver de type « chrome » pour exécuter des tests sur un navigateur chrome.**

```
@After  
public void tearDown() throws Exception {  
    driver.quit();  
}
```

**@After → Le « Tear Down » permet ici fermer le navigateur à la fin du test.**

Dans Junit4 il existe les annotations **@BeforeClass** et **@AfterClass** qui permettent d'exécuter des actions avant et après une suite de test.

# 3-Architecture générique d'automatisation de tests



## 3.1- Introduction à la gTAA

### 3.1.4- Couche d'exécution de test

Données : les frameworks de test comme ici JUnit ont des mécanismes permettant de gérer les jeux d'essais.

```
@Parameters public static Collection pos() {
    return Arrays.asList(new Object[][] {{'c',7,'c',7},{'c',9,'c',1}});
}
private int posHor;
private int posVer;
private int posHora;
private int posVera;
public testPieceEchiquier4(int posHor,  int posVer, int posHora, int posVera)
{
    this.posHor = posHor;
    this.posVer = posVer;
    this.posHora = posHora;
    this.posVera = posVera;
}
@Test
public void testConstructeur()
{
    PieceEchiquier piece = new PieceEchiquier("piece","blanc", posHor,posVer);
    assertTrue((piece.getPositionHorizontale() == posHora) && (piece.getPositionVerticale() == posVera) );
}
```

2 jeux d'essais  
Valeurs +  
Valeurs attendues

Le test sera joué pour les 2 jeux d'essais

## ■ Comparateur:

- Les framework de test possèdent également des mécanismes de comparaison, les assertions:

```
@Test  
public void test() {  
    paramAppli p = new paramAppli();  
    CalculFrais f = new CalculFrais(p);  
    assertEquals(montant,f.montant(total),0.0);  
}
```

Comparaison de flottants.

- Il existe des comparateurs spécialisés: ex avec dbunit

```
// comparer les éléments de la colonne id_category_default en utilisant Assertion.assertEqualsByQuery  
Assertion.assertEqualsByQuery (dataSetAtt,connexion,"SELECT id_category_default from ps_product ","ps_product", new String[] {});
```

- Dans certains cas il faudra développer ses propres comparateurs.

# 3-Architecture générique d'automatisation de tests



## 3.1- Introduction à la gTAA

### 3.1.4- Couche d'exécution de test



Pourquoi le test est-il KO?  
Je vais le rejouer en  
manuel.

Les logs doivent fournir les informations nécessaires aux dépouillement des tests.

1. **Vérifier les prérequis pour exécution**
  - Permet de distinguer défaut et incident d'exécution
2. **Avoir des logs de tests qui trace les étapes du test**
  - Je sais où j'en suis dans l'exécution
3. **Sauvegarder des informations en cours de test**
  - Copies d'écran
  - Horodatage pour lien avec les logs du système
  - Etat d'élément du système
  - Valeurs constatées versus attendues
  - Permet de rédiger l'anomalie

# 3-Architecture générique d'automatisation de tests



## 3.1- Introduction à la gTAA

### 3.1.4- Couche d'exécution de test

#### TEST CASE: Cas de test 5

**Full Name:** Bdt.BTD.Cas de test 5

**Start / End / Elapsed:** 20110403 13:52:53.022 / 20110403 13:53:11.828 / 00:00:18.806

**Status:** PASS (critical)

**KEYWORD:** resource.Given un utilisateur google

Start / End / Elapsed: 20110403 13:52:53.024 / 20110403 13:53:10.908 / 00:00:17.884

**KEYWORD:** SeleniumLibrary.Open Browser <http://google.fr>, Firefox

**KEYWORD:** SeleniumLibrary.Location Should Be <http://www.google.fr>/

**KEYWORD:** resource.When il fait une recherche sur RobotFramework

Start / End / Elapsed: 20110403 13:53:10.909 / 20110403 13:53:11.018 / 00:00:00.109

**KEYWORD:** SeleniumLibrary.Input Text q, RobotFramework

**KEYWORD:** SeleniumLibrary.Click Button btnG, No\_Wait

**KEYWORD:** resource.Then il doit retrouver le site de robotframework

Start / End / Elapsed: 20110403 13:53:11.019 / 20110403 13:53:11.742 / 00:00:00.723

**KEYWORD:** SeleniumLibrary.Wait Until Page Contains code.google.com/p/robotframework/, 5

**TEARDOWN:** resource.Quitter

Vérifier la pertinence des recherches sur google.

Pré requis du test:  
Accéder à google

Action

Vérification

## ■ Rapport de test

- Des mécanismes existent pour générer les rapports de tests comme par exemple avec Maven, junit : le plugin surefire.
- Pour la publication elle peut se faire via un outil d'intégration continue style jenkins.
- Il est également possible avec certains outils de gestion de test de ramener les résultats de tests automatiques dans l'outils de gestion de test. Ce qui permet d'avoir un rapport unique qui combine les tests manuels et les tests automatiques.

- Exemple
- Le plugin surefire de maven permet de générer des résultats de tests sous format xml.
- Ces fichiers peuvent être exploités par le plugin Junit de jenkins.

 Publier le rapport des résultats des tests JUnit 

XML des rapports de test  

Une configuration du type Fileset 'includes' qui indique où se trouvent les fichiers XML des rapports de test, par exemple 'myproject/target/test-reports/\*.xml'. Le répertoire de base (basedir) du fileset est la racine du workspace.

Retain long standard output/error 

Health report amplification factor    

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Additional test report features  

Allow empty results  Do not fail the build on empty test results 



## 3.1- Introduction à la gTAA

### 3.1.4- Couche d'exécution de test

 Retour au projet

 État

 Modifications

 Console Output

 Informations de la construction

 Historique

 Tagguer ce build

 TestLink

 Résultats des tests

 Build Précédent

 Build suivante

## Résultats des tests

1 échecs

1 tests

A pris 9,8 s.

 Ajouter une description

### Tous les tests qui ont échoué

Nom du test	Durée	Age
<a href="#">com.selenium.TestRecherche.testRecherche1</a>  Error Details row count (table=ps_product) expected:<[2]> but was:<[7]> Stack Trace Standard Output Standard Error	9,8 s	1

### Tous les tests

Package	Durée	Échec	(diff)	Sauté	(diff)	Pass	(diff)	Total	(diff)
<a href="#">com.selenium</a>	9,8 s	1	+1	0	0	0	0	1	+1



- Il s'agit des éléments/library qui permettent d'interagir avec le SUT.
  - Interface Homme-Machine (web, swing, WPF ...)
  - API (SOAP, Rest, RCP ...)
  - CLI (commande Line Interface)
- Également les simulateurs (remplacent les systèmes qui interagissent avec le SUT).
- Nécessite des compétences techniques.

## 3.1- Introduction à la gTAA

### 3.1.5- Couche d'adaptation de test

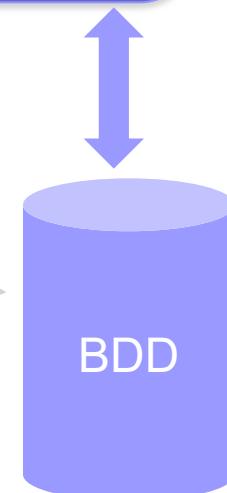
## ■ SUT: System Under Test



IHM ou API

Accès au SUT via des API, l'IHM ou des commandes en lignes pour activer des fonctionnalités du SUT ou récupérer de l'information en vue de contrôles.

Accès à la base de données pour gérer les données en base ou effectuer des contrôles.



Simulation du monde extérieur.

## 3.1- Introduction à la gTAA

### 3.1.5- Couche d'adaptation de test

Ex: Avec robot framework vous créez vos mots clefs métier et vous les implémentez via des librairies qui proposent des mots clefs vous permettant d'interagir avec le SUT.

#### RESTInstance

Robot Framework test library for HTTP JSON APIs.

#### SeleniumLibrary

Web testing library that uses popular Selenium tool internally.

#### ExtendedSelenium2Library

Web testing library that uses Selenium2Library internally, providing AngularJS support on top of it.

#### SudsLibrary

A library for functional testing of SOAP-based web services based on Suds, a dynamic SOAP 1.1 client.

#### TFTPLibrary

Library for interacting over **Trivial File Transfer Protocol**.

#### SapGuiLibrary

Library for testing the SAPGUI client using the internal SAP Scripting Engine

#### Selenium2Library

Web testing library that uses Selenium 2. Library is deprecated; users should upgrade to SeleniumLibrary described above.

#### SikuliLibrary

Sikuli Robot Framework Library provide keywords to test UI through **Sikulix**. This library supports Python 2.x and 3.x.

#### SwingLibrary

Library for testing Java applications with Swing GUI.

#### WhiteLibrary

Library for automating Windows GUI. Based on White framework, supported technologies include Win32, WinForms, and WPF.

#### Selenium2Screenshots

Library for capturing annotated screenshots with Selenium2Library.

#### Selenium2Library for Java

Java port of the Selenium2Library.

#### SSHLibrary

Enables executing commands on remote machines over an SSH connection. Also supports transferring files using SFTP.

#### TestFX Library

Library to enable to test Java FX applications using the **TestFX framework**. Has also remote interface support.

#### watir-robot

Web testing library that uses Watir tool.

#### Exemple de Mot clef de la library Selenium de robotframework

##### \*\*\* Test Cases \*\*\*

```
T1
[Documentation] cas nominal
[Tags] NONREG
Un utilisateur indecis
je fais une recherche avec le mot cle jupes
je trouve 1 résultat a été trouvé.

*** Keywords ***
Un utilisateur indecis
    AllerALaPageDaccueil

je fais une recherche avec le mot cle [Arguments] ${mclef}
    lancerUneRecherche ${mclef}

lancerUneRecherche [Arguments] ${mclef}
    Input Text id=search_query_top ${mclef}
    Click Element name=submit_search
```



Input Text est un mot cle bas niveau qui permet d'interagir avec le SUT, ici saisir du texte dans un champ repéré par son ID (attribut HTML) de la balise Input.

## ■ Exemple avec Selenium Webdriver:

```
public void setMotCle(String mot) {  
    driver.findElement(By.name("s")).sendKeys(mot);  
    driver.findElement(By.name("s")).sendKeys(Keys.RETURN);  
}
```

- Cette bibliothèque fournit une API qui permet d'interagir avec une Interface WEB.
- Ici on repère le champ texte identifié par l'attribut name et on saisit une chaîne de caractères contenue dans la variable « mot ».

- **Exemple avec la bibliothèque Requests de robotframework:**

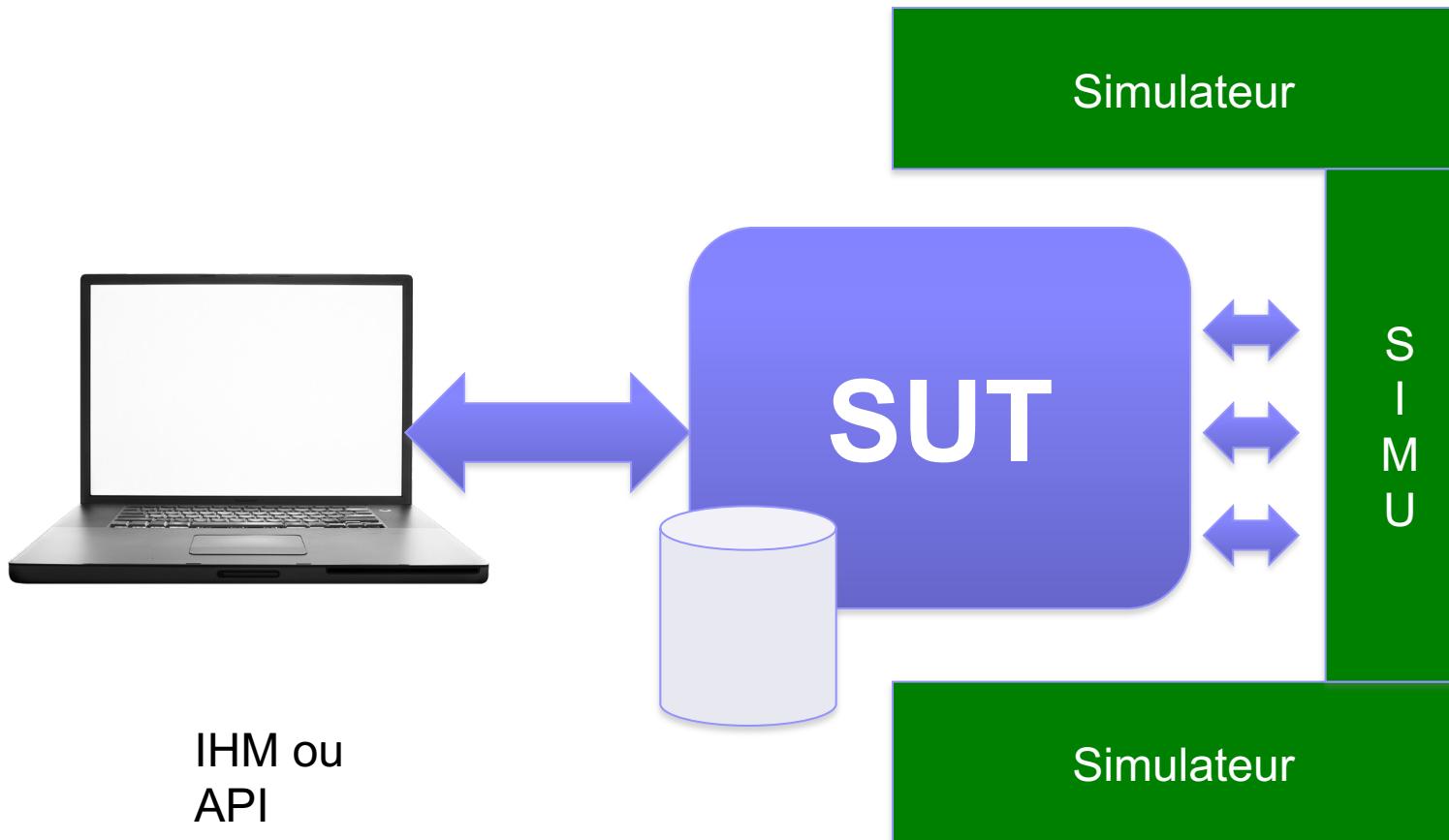
```
Create Session pays http://api.zippopotam.us
${resp} Get Request pays /us/90210
${obj} To Json ${resp.content}
```

- **Avec cette bibliothèque il est possible de communiquer avec le SUT via des requêtes REST.**
- **Ici on initialise une session puis on envoie une requête et on récupère la réponse.**

# 3-Architecture générique d'automatisation de tests

## 3.1- Introduction à la gTAA

### 3.1.5- Couche d'adaptation de test



## Le ou les simulateurs:

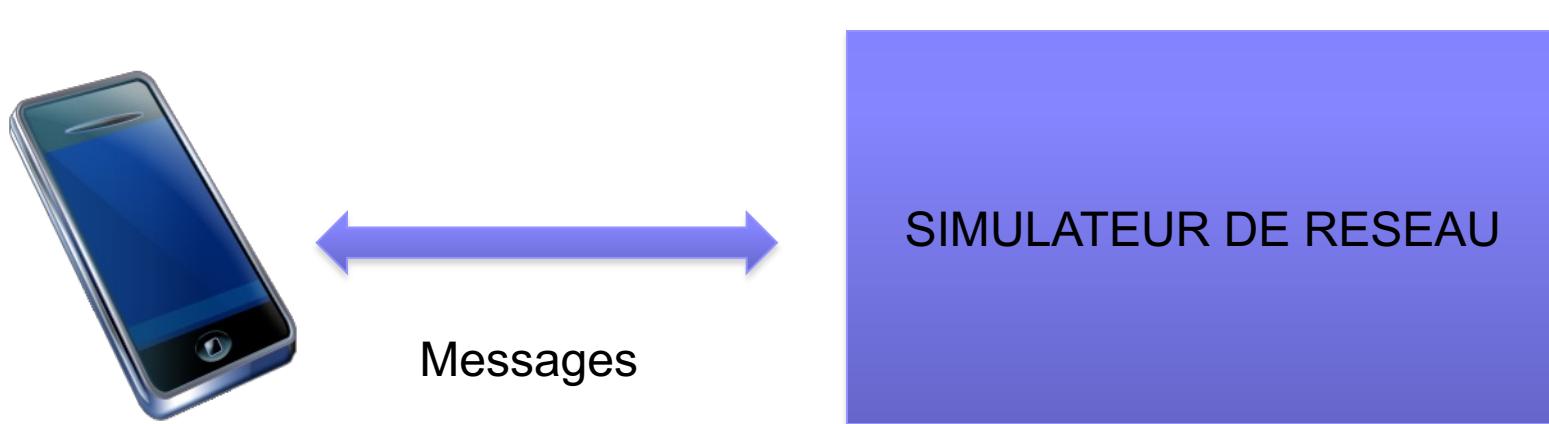
### ■ Complexé ou simple?

- Simple simulacre (ex: SOAP UI Mock)
- Développement spécifique?
- Achat supplémentaire?

### ■ Pilotage du simulateur

- Est-il pilotable via le framework de test?
- Me permet-il de réaliser tous les tests?
- Est-il représentatif du système?

## ■ EX: test d'un nouveau téléphone mobile



## ■ Le Simulateur émet et reçoit des messages conformément au protocol GSM.

## ■ Développer son propre simulateur de test

- N'existe pas
- Prix
- Fonctionnalités manquantes

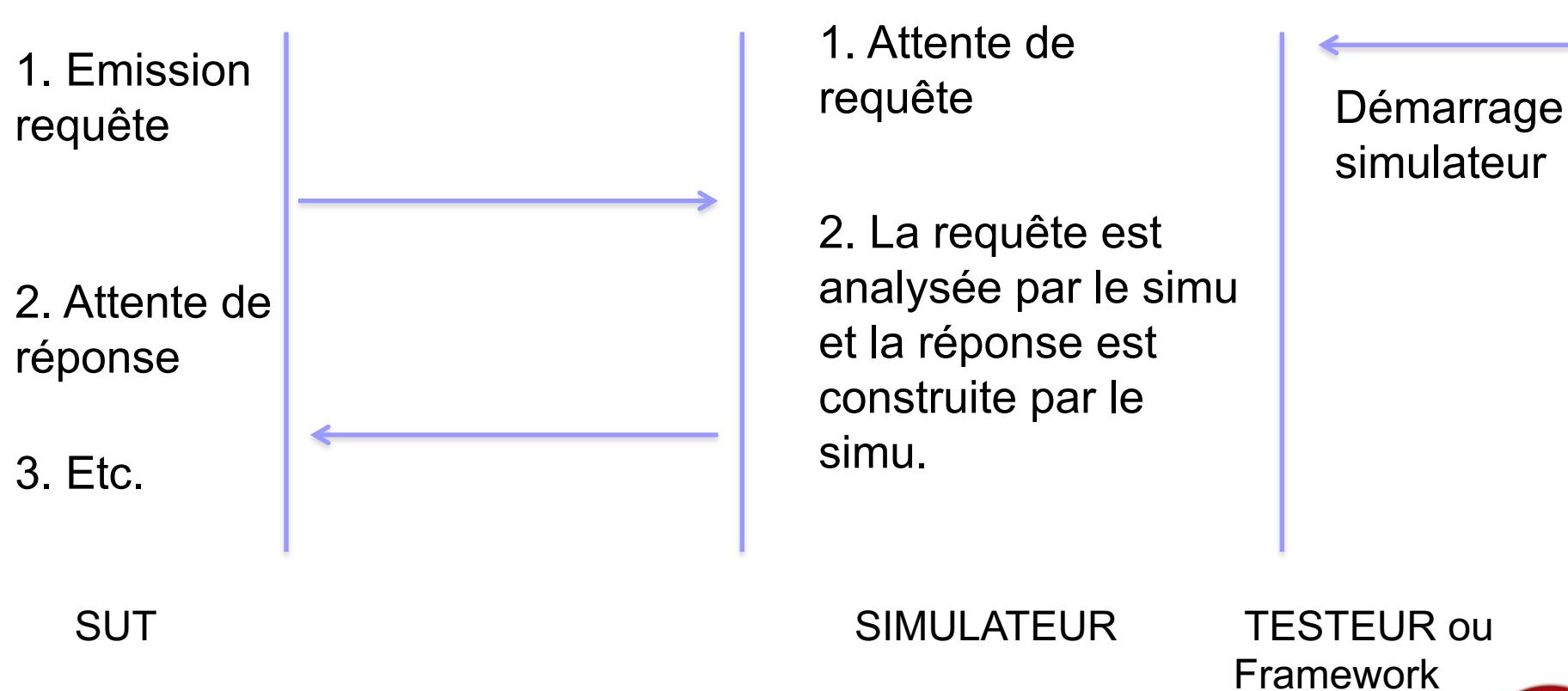
## ■ CONTRAINTES:

- Prévoir une phase de débogage de l'outil
- Fiabilité
- Projet de développement de logiciel

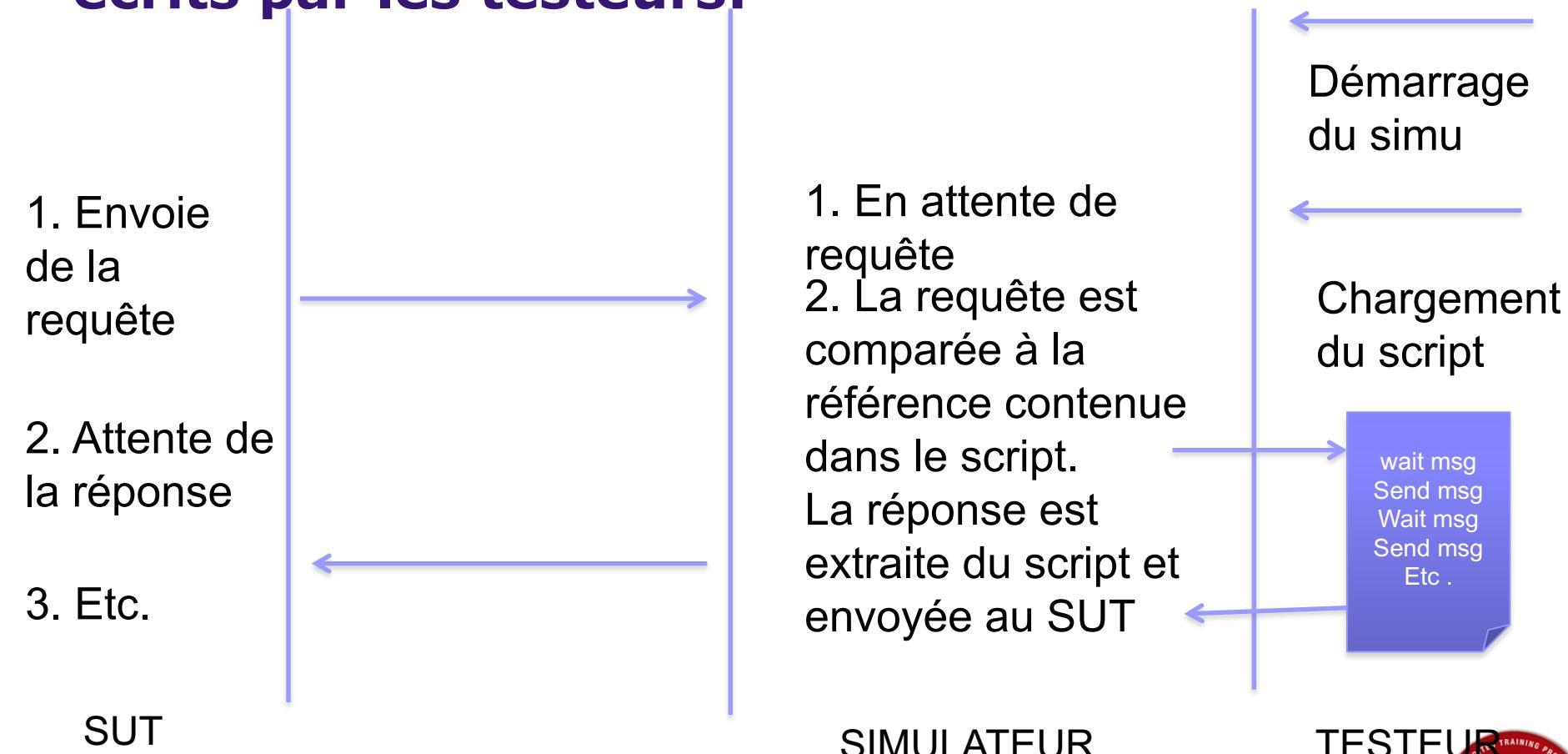
## ■ Implémentation, plusieurs choix sont possibles

- Simulation via logiciel
- Simulation avec scripts
- Mock de l'interface de communication

## ■ Le comportement est simulé via logiciel:



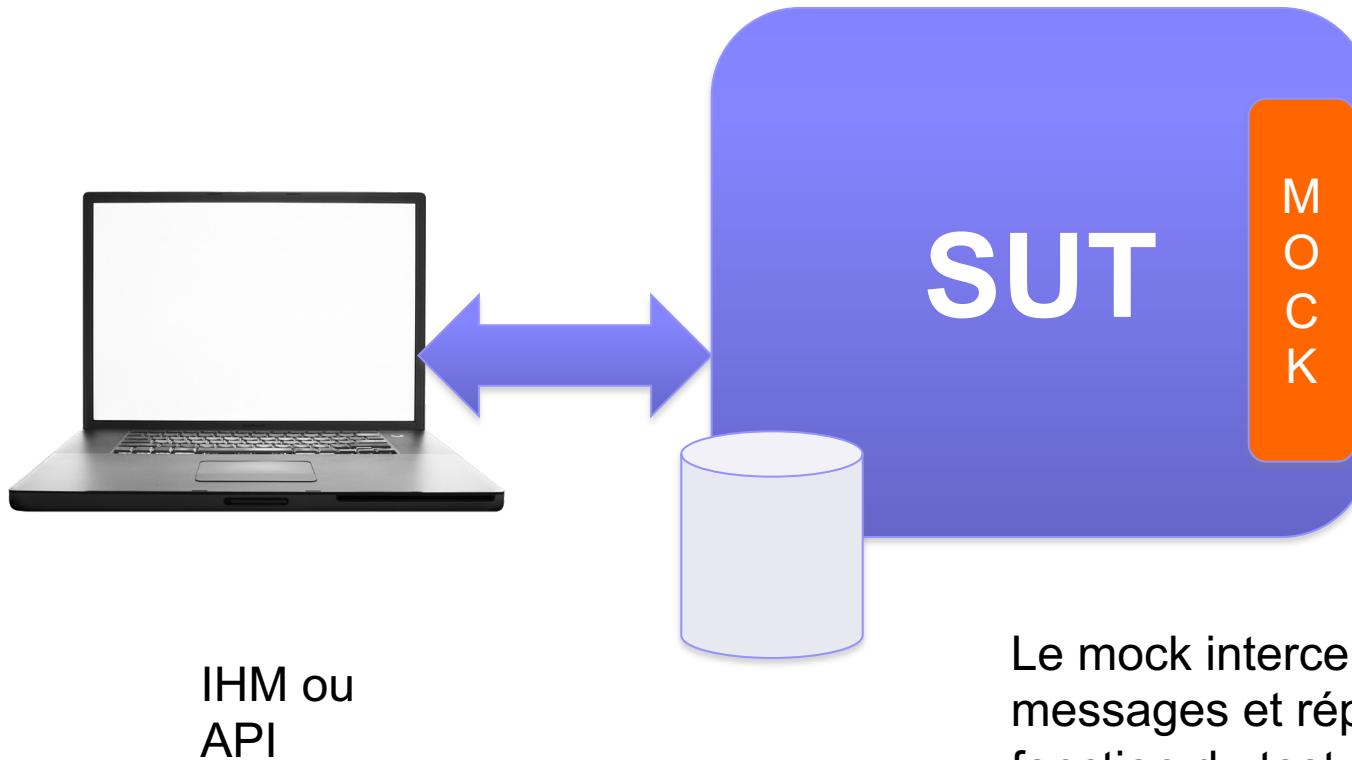
## ■ Le comportement est simulé via des scripts écrits par les testeurs:



## SCRIPT

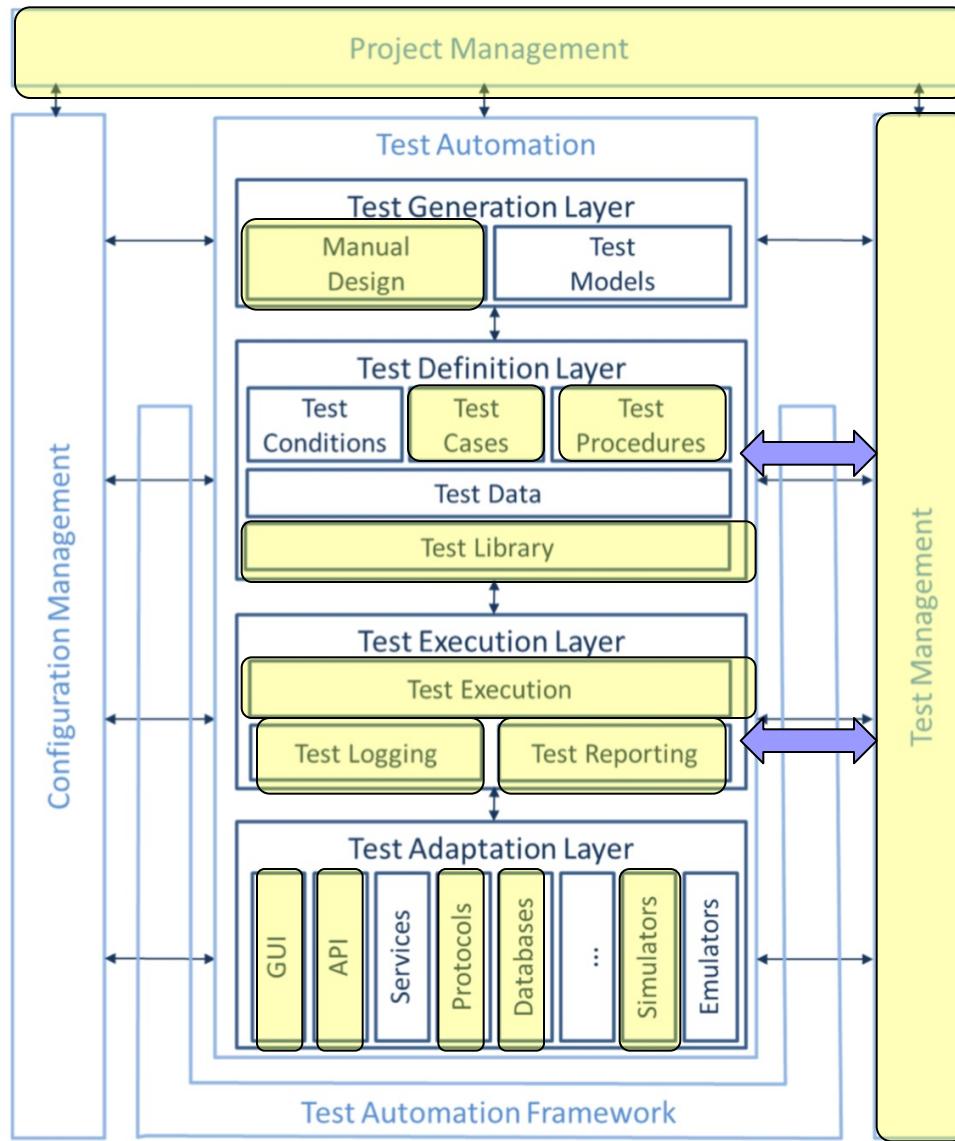
```
Wait for msg  
(timeout)  
Send msg  
Wait for msg  
(timeout)  
Send msg  
Wait for msg  
(timeout)  
Send msg  
...
```

- **Le script reflète le comportement du système.**
- **Le message reçu est comparé à celui du script**
- **Le message utilisé pour la comparaison est la référence pour calculer le résultat du test.**
- **Nécessité d'outil de comparaison intégré.**



Le mock intercepte les messages et répond en fonction du test.  
→ Inconvenient: le SUT est modifié.

# 3-Architecture générique d'automatisation de tests : EXEMPLE fil rouge



- **Le développement de la TAS se fait en parallèle du SUT. Ses évolutions doivent être compatibles.**
- **La gestion de configuration englobe les éléments suivants:**
  - Des modèles de test
  - Des définitions/spécifications incluant les données de test, les cas de test et les librairies
  - Des scripts de test
  - Des moteurs d'exécution de test et des outils et composants supplémentaires
  - Des adaptateurs de test pour le SUT
  - Des simulateurs et émulateurs pour l'environnement du SUT
  - Les résultats et les rapports de test

**Ces éléments constituent le testware.**

**En cas de problème détectée sur une version antérieure**

**Il sera possible de déployer l'environnement de test**

**Rejouer les scripts de test à la bonne version pour tester les corrections**

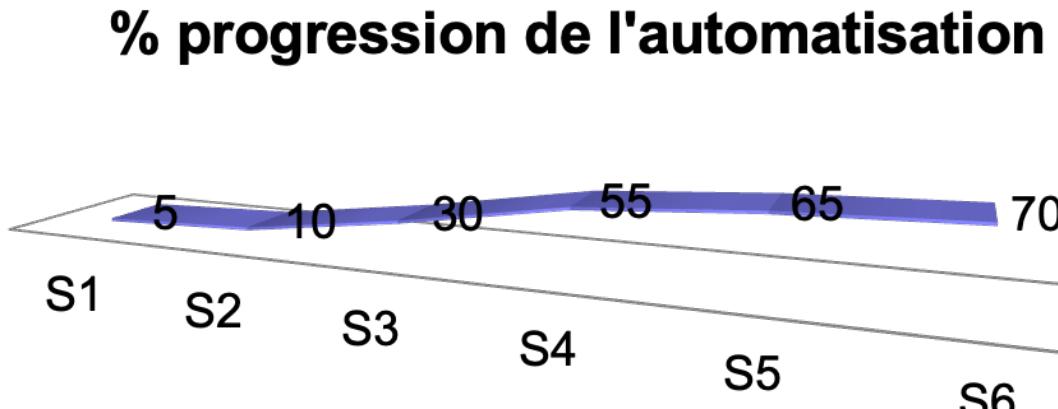
**Comparer avec les anciens logs de test pour vérifier la correction.**

## ■ Gestion projet d'une TAS

- Projet de développement
- Métriques associées
- Informations de suivi extraites facilement

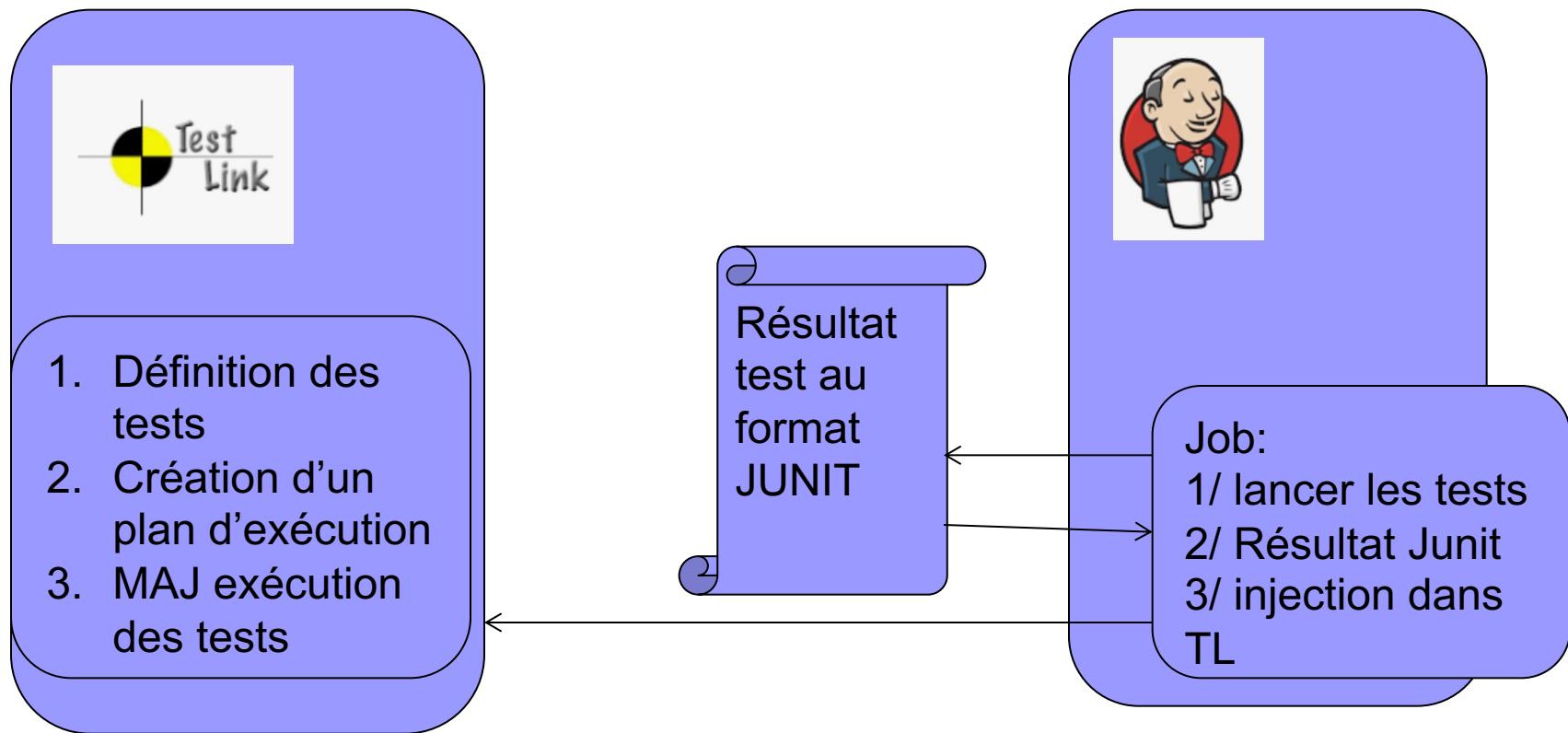
### Exemple:

- Nombre de test automatisés/ nombre de test à automatiser.
- Etc ...



- **Fourniture de rapport et log de test:**
- **Mise à jour automatique des résultats de test**
  - **Les outils de gestion de test offrent généralement des interfaces pour mettre à jour les résultats de test.**
  - **Contrôler la possibilité de d'interfacer outil de gestion de test avec outil d'automatisation.**
    - Via des API types RCP, Rest ou SOAP
- **Consolidation tests manuels/tests automatiques**

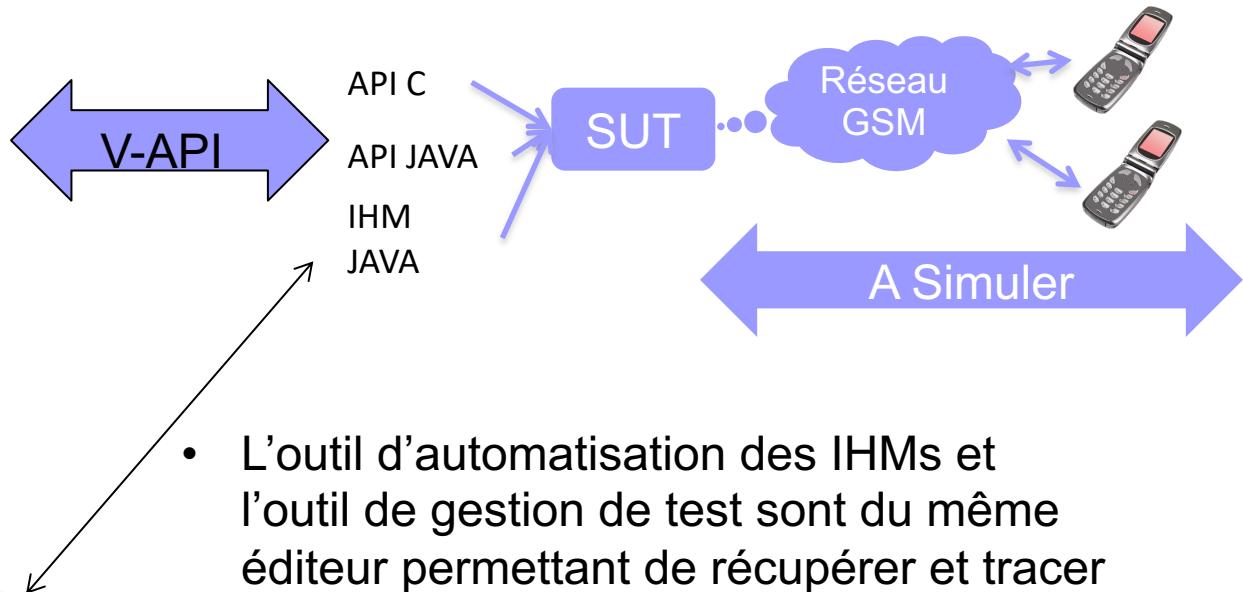
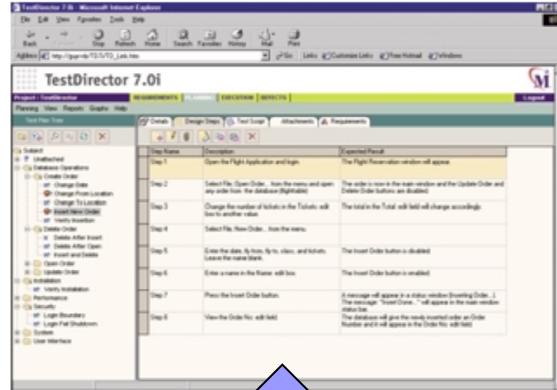
## ■ Exemple: Testlink expose des API types RCP



# 3-Architecture générique d'automatisation de tests : EXEMPLE fil rouge

## 3.1- Introduction à la gTAA

### 3.1.8 Support de la TAS pour la gestion de test



The screenshot shows the WinRunner interface with a script editor. The script contains VBA-like code for automating a flight reservation process:

```
# Flight Reservation - Open a new order
win_activate ("Flight Reservation");
set_window ("Flight Reservation", 1);
obj_mouse_click ("Button", 4, 15, LEFT);

# Flight Reservation - Start the order
obj_type ("EditMaskUndClass", "123123");
list_select_item ("Fly From:", "London"); # Item Number 2;
list_select_item ("Fly To:", "Paris"); # Item Number 3;
obj_mouse_click ("FLIGHT", 61, 35, LEFT);

# Flights Table
set_window ("Flights Table", 1);
list_select_item ("Flight", "12506 LON 10:24 AM PAR 12:24");
button_press ("OK");
```

- L'outil d'automatisation des IHMs et l'outil de gestion de test sont du même éditeur permettant de récupérer et tracer les résultats de test.
- L'outil de gestion de test permet également d'intégration du test automatique d'API.

### ■ Quelles sont les 4 couches horizontales de la gTAA

- a) adaptation, exécution, conception, définition
- b) Génération, exécution, définition, API
- c) Génération, définition, exécution, adaptation
- d) Définition, exécution, reporting, adaptation

## Les étapes nécessaires:

- 1. Capturer les exigences**
- 2. Comparer les différentes solutions**
- 3. Identifier les zones où un niveau d'abstraction permet un gain**
- 4. Comprendre les aspects techniques du SUT et comment ils sont connectés au système d'automatisation**
- 5. Comprendre l'environnement du SUT**
- 6. Estimer son coût et le temps de développement**
- 7. Documenter**

## ■ Capturer les exigences

- Quelle activité de test veut-on automatiser (gestion de test, conception de test ...)**
- Quel niveau de test (test unitaire, intégration, système)?**
- Quel type de test (fonctionnel, conformité ...)?**
- Quel profil de testeur?**
- Quel produit ou famille de technologie?**
- Quelle technologie?**

## ■ Ex: Si je décide d'automatiser les tests unitaires et je travaille en java je vais plutôt utiliser un framework de test Unitaire style JUNIT ou TestNG.

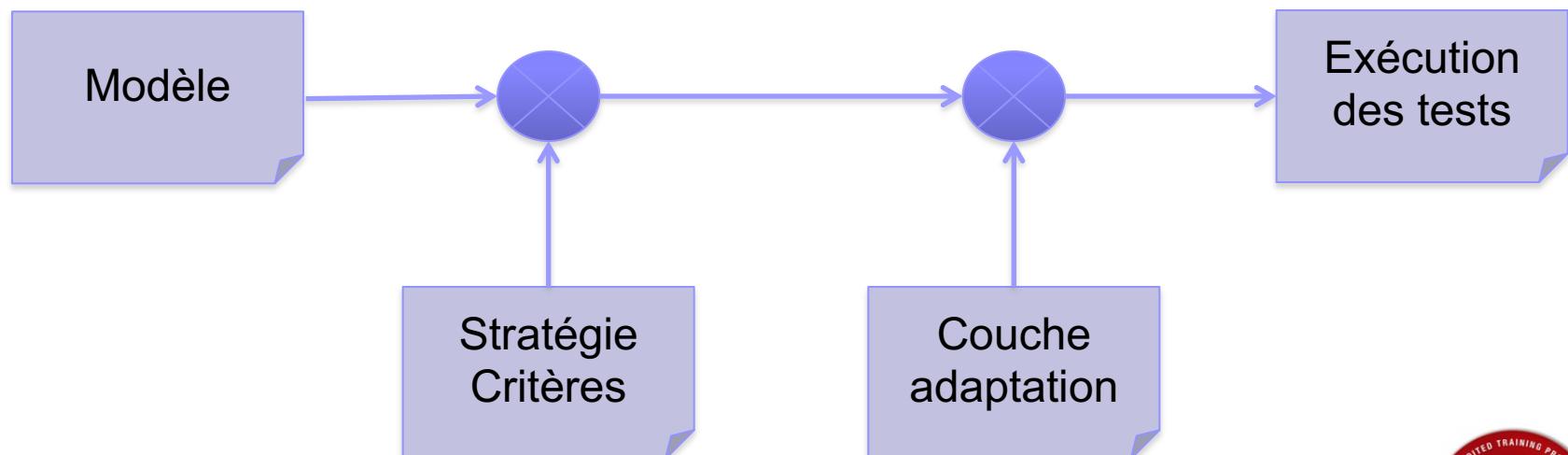
- Comparer et contraster les différentes approches de conception/architecture

- **Considérations à prendre en compte pour la couche de génération de test.**

- ❖ Automatisation de la génération de test (modèle)
    - ❖ Génération basée sur exigences, données, scénarios, comportements ...
    - ❖ Sélection de la stratégie de génération (couverture d'un modèle basée données, scénarios nominaux/exception, parcours de graphe ...)

- **Il est indispensable de définir des critères de couverture, poids, risques pour limiter la génération de cas de test qui peut devenir exponentielle.**

- Exemple
- GraphWalker définit différents critères d'arrêt pour son parcours de graphe.
  - Pourcentage de Nœuds couverts
  - Pourcentage d'Arcs couverts
  - Pourcentage d'exigences couvertes.



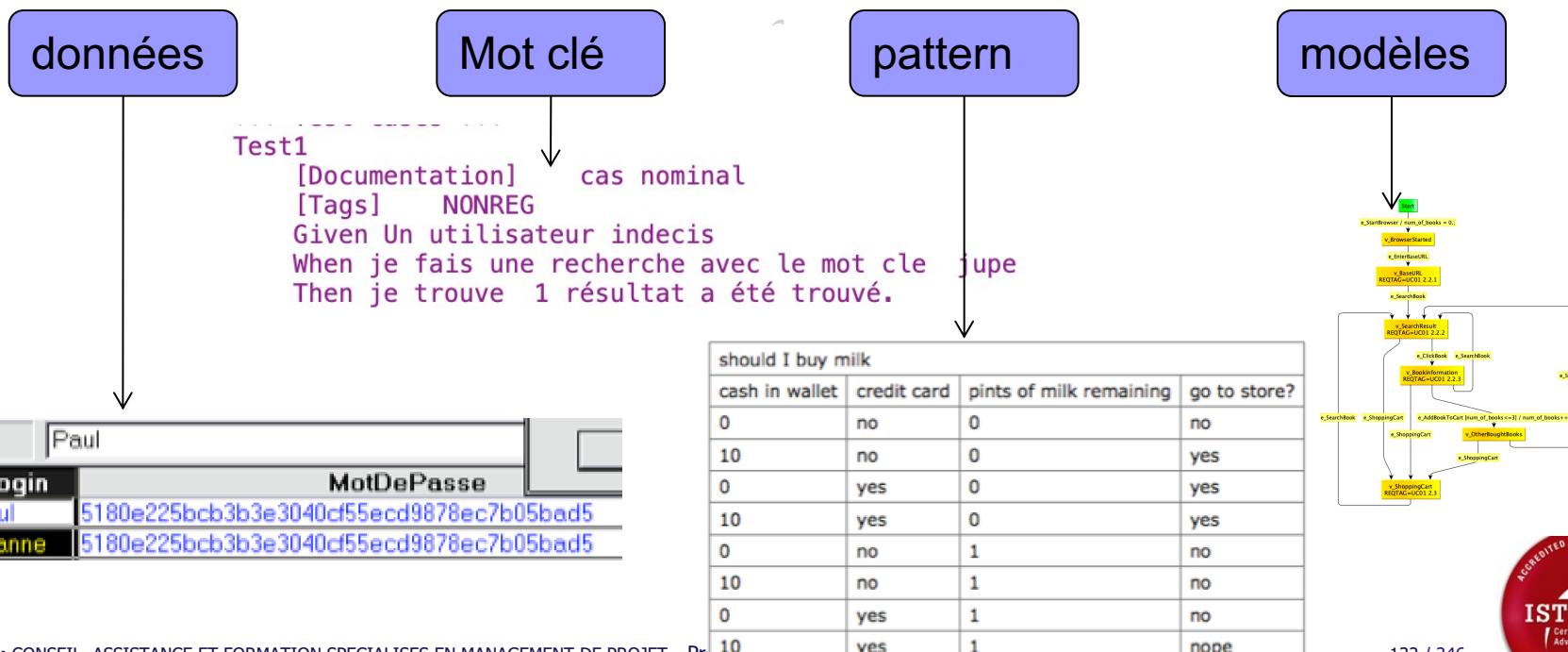
# 3-Architecture générique d'automatisation de tests



## 3.2- Conception de la TAA

### 3.2.1 Introduction à la conception de la TAA

- Comparer et contraster les différentes approches de conception/architecture
  - Considérations à prendre en compte pour la couche de définition de test



- Comparer et contraster les différentes approches de conception/architecture

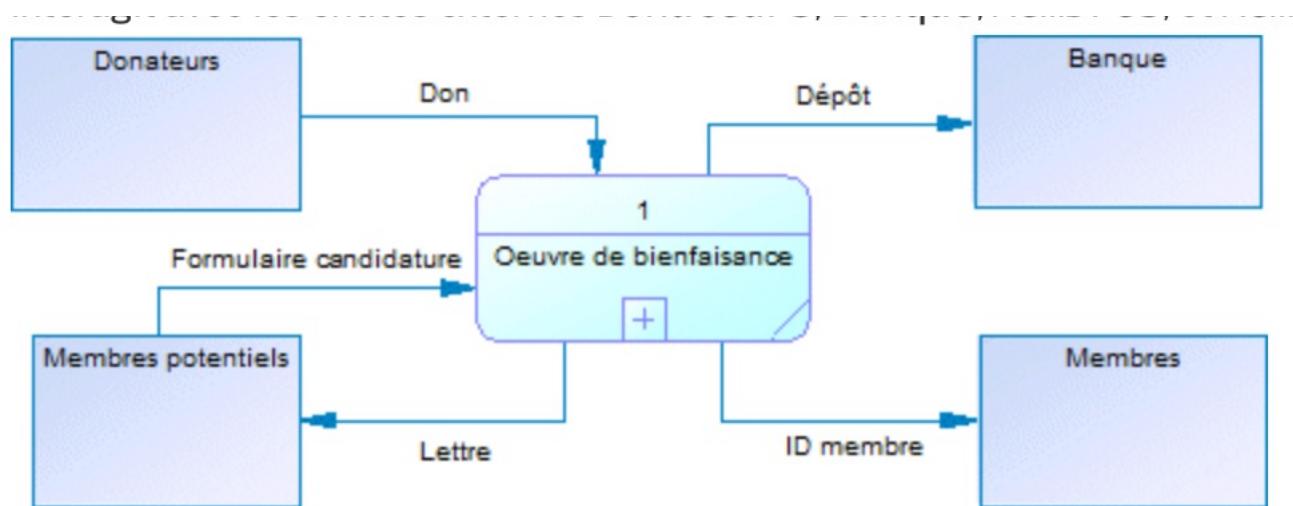
- **Considérations à prendre en compte pour la couche de définition de test → notation**

- ❖ Tableaux (excel, xml ...)
    - ❖ Notation basée sur les états
    - ❖ Notation basée sur les flux de données
    - ❖ Langage spécifique à un domaine TTCN

- **Considérations à prendre en compte pour la couche de définition de test → stockage des tests**

- ❖ Tableur
    - ❖ Outil (UFT, Ranorex ...)
    - ❖ BDD
    - ❖ ...

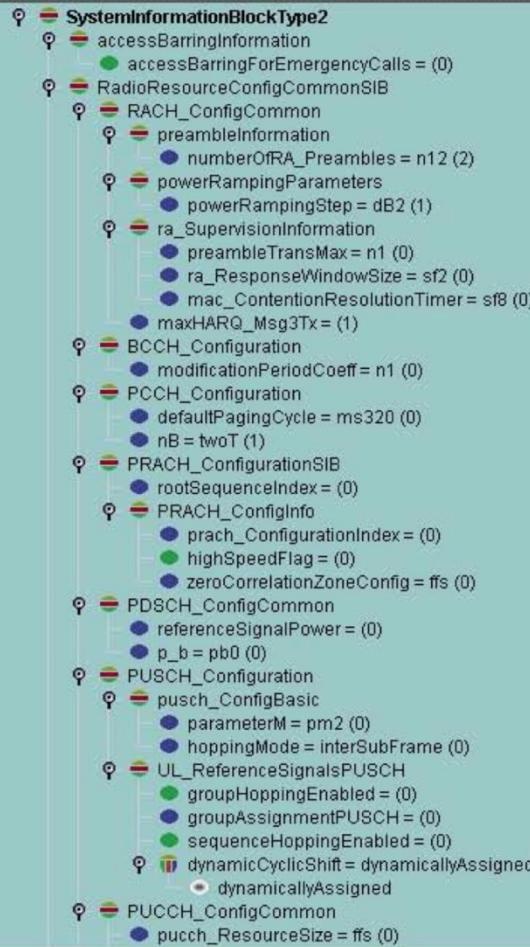
## ■ Basée sur flux de données



## 3.2- Conception de la TAA

### 3.2.1 Introduction à la conception de la TAA

## ■ TTCN → test mobile



Byte	Bitstream	Identifier	Decimal	Interpreta
		CRRC SystemInformationBlock Configura		CrrcSibConfigReq
32	00000101	Cell Handle	5	
33	00000100	Number of ASN.1 structures	4	
		ASN.1 structure list		AsnlStruct
		ASN.1 structure information		
34	00000001	SIB Type used for CRRC primitives	1	SystemInformationBlockType1
35	00000000	Length of encoded ASN.1 structure in 101		
36	00000000			
37	00000000			
38	01100101	Encoded SystemInformationBlock ASN.1		SystemInformationBlockType1
39	00-----	SystemInformationBlockType1		
39	--0-----	cellAccessRelatedInformation		
39	--000--	PLMN IdentityList		value
		value		
39	-----0-	PLMN Identity		
39	-----0	MNC		value
40	0000----	MCC MNC Digit	0	
40	----0000	MCC MNC Digit	0	
41	1-----	cellReservedForOperatorUse	1	notReserved
		TrackingAreaCode		
41	-0000000	CellIdentity		
42	00000000			
43	00000000			
44	00001---			
44	-----1-	cellBarred	1	notBarred
44	-----1-	cellReservationExtension	1	notReserved
44	-----0	csg Indication		
45	0-----	cellSelectionInfo		
45	-000000-	q Rxlevmin	-70	
45	-----0	frequencyBandIndicator	1	
46	00000---			
46	-----000	SchedulingInformation		value
47	01-----			
		value		
47	--001---	si Periodicity	1	rfl6

- Comparer et contraster les différentes approches de conception/architecture
  - **Considérations à prendre en compte pour la couche de définition de test → guide**



Les guides d'écriture de script de test sont les reflets de l'implémentation de la stratégie de test. Ils permettent d'avoir des scripts de test maintenable, homogène, accessible pour les nouveaux venus ...

- Comparer et contraster les différentes approches de conception/architecture

- **Considérations à prendre en compte pour la couche de exécution de test**

- ❖ Outil d'exécution de test
    - ❖ Compilé, scripté
    - ❖ Language C, C++, python etc ...
    - ❖ Librairies tierces (calcul, manipulation BDD, logging ....)

- Comparer et contraster les différentes approches de conception/architecture

- **Considérations à prendre en compte pour la couche de d'adaptation de test**

- ❖ Sélection d'interfaces de test du SUT
    - ❖ Sélection d'outils pour simuler et observer les interfaces de test
    - ❖ Sélection d'outils pour surveiller le SUT pendant l'exécution de test
    - ❖ Sélection d'outils pour tracer l'exécution de test (par exemple, incluant le timing de l'exécution de test)

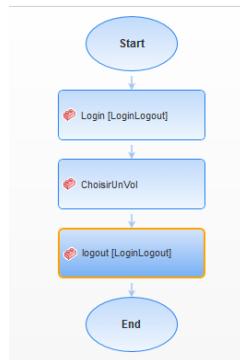
- Identifier les domaines où l'abstraction peut offrir des avantages
  - l'abstraction est intéressante pour la TAA (test Automation Architecture) et permet une meilleure maintenabilité et flexibilité.
  - Gain pour portage sur différentes technologies
  - Indépendance vis à vis du fournisseur
  - Plus facile d'accès pour le métier.

- Identifier les domaines où l'abstraction peut offrir des avantages : exemple

- Utilisation d'un DSL implémenté via des mots-clés

```
T1 →  
    → [Documentation] → cas · nominal  
    → [Tags] → NONREG  
    → Un · utilisateur · indecis  
    → je · fais · une · recherche · avec · le · mot · cle · jupes  
    → je · trouve · 7 · jupes
```

- Utilisation d'un langage propriétaire



- Identifier les domaines où l'abstraction peut offrir des avantages : conclusion

- Nécessite un investissement plus important (au départ)
  - Architecture du TAA plus complexe
  - Montée en compétence des automaticiens plus longues
  - ROI moins rapide

→ Etude du retour sur investissement à faire, vision long terme.

- Comprendre les technologies des SUT et comment ils s'interconnectent avec la TAS interface:
  - Niveau Logiciel, p.ex., le SUT et le logiciel de test sont liés ensemble.
  - Niveau API, p.ex., la TAS invoque les fonctions/opérations/méthodes fournies par une interface de programmation d'application (à distance) (RCP).
  - Niveau Protocole, p.ex., la TAS interagit avec le SUT via HTTP, TCP.
  - Niveau Service, p.ex., la TAS interagit avec les services du SUT via les Web Services, Services.
  - RESTful.

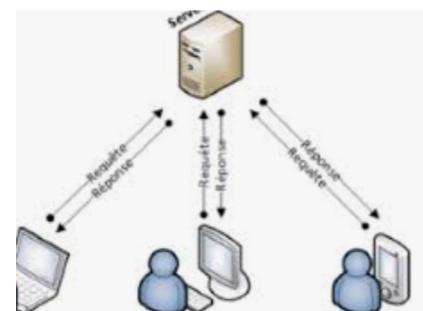
- Comprendre les technologies des SUT et comment ils s'interconnectent avec la TAS :

- Identification du paradigme

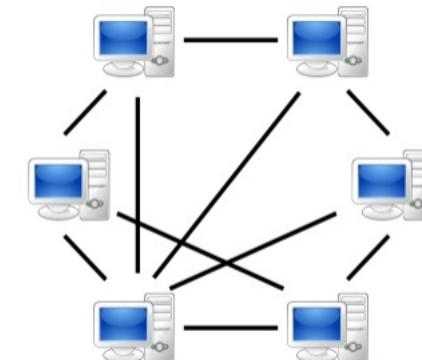
- ❖ Evénement
  - ❖ Invocation de service type Client Serveur
  - ❖ Pair à Pair



Événements : réception  
SMS, appel, via clavier ...



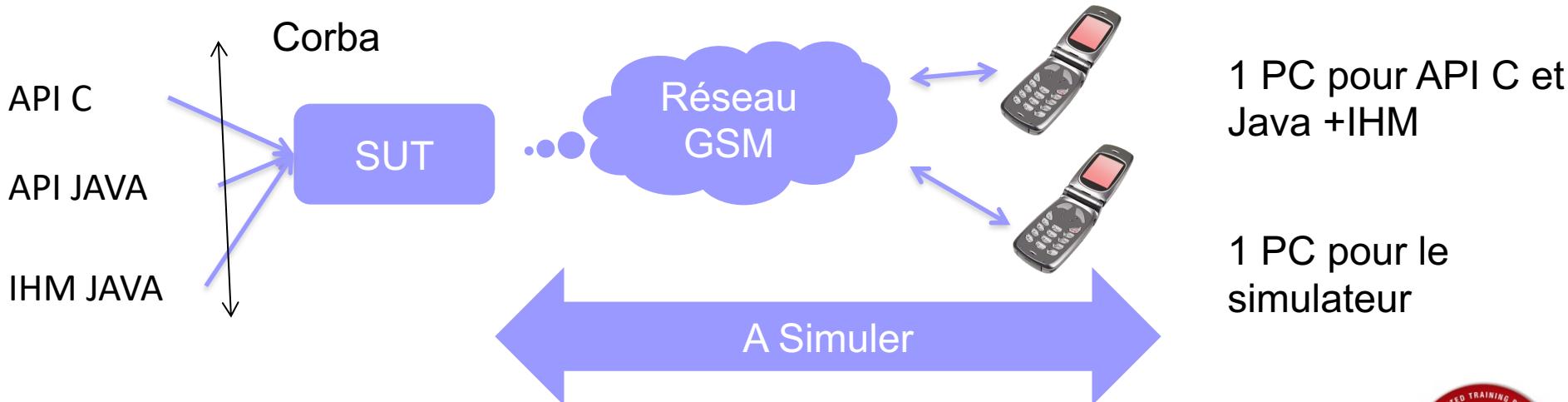
Requêtes http,  
web services ...



Appel de  
services



- Comprendre les technologies des SUT et comment ils s'interconnectent avec la TAS : Environnement
  - Il est important d'identifier le SUT et ses interactions avec d'autres systèmes, matériel et composants environnementaux.
  - Exemple Fil Rouge:



- Comprendre les technologies des SUT et comment ils s'interconnectent avec la TAS : Environnement
  - Boitier décodeur (set top box) → prévoir du matériel de test supplémentaire (télécommande, Tv ...)
  - Simulateur de hardware pour simuler l'environnement technique du SUT (logiciel embarqué)
    - ❖ Les maquettes peuvent être couteuse
    - ❖ Ces environnements facilitent les tests, et tests automatiques.
  - Routeur : dispositif en réseau pour tester le routeur.

- Durée et complexité pour l'implémentation d'une architecture de test
- Responsable TAM, données fournies par le TAE
  - ❖ Estimation basée sur l'analogie comme l'analyse des points de fonction, estimations à 3 points
  - ❖ Wide Band delphi, estimation basée sur l'expertise
  - ❖ Estimation en utilisant les structures de décomposition du travail comme celles des logiciels de gestion ou des modèles projet
  - ❖ Estimation avec des paramètres utilisant “Constructive Cost Model (COCOMO)”
  - ❖ Estimations basées sur la taille utilisant l'analyse des points de fonction, l'analyse des points de User Story ou l'analyse des cas d'utilisation
  - ❖ Les estimations de groupe utilisant le Planning Poker

#### ■ Analyse par point de fonction:

Type de fonction	Multiplicateur		
	Simple	Moyen	Complex
Entrée de données externes	3	4	6
Sortie de données externes	4	5	7
Fichier logique interne	7	10	15
Fichier d'interface externe	5	7	10
Fichier de requête externe	3	4	6

Estimation à 3 points : Technique basée sur les statistiques (distribution normale).

• Une estimation optimiste = O

- en référence au projet le moins coûteux qui puisse être rapproché du projet à estimer.

• Une estimation pessimiste = P

- en référence au projet le plus coûteux qui puisse être rapproché du projet à estimer.

• Une estimation moyenne = M

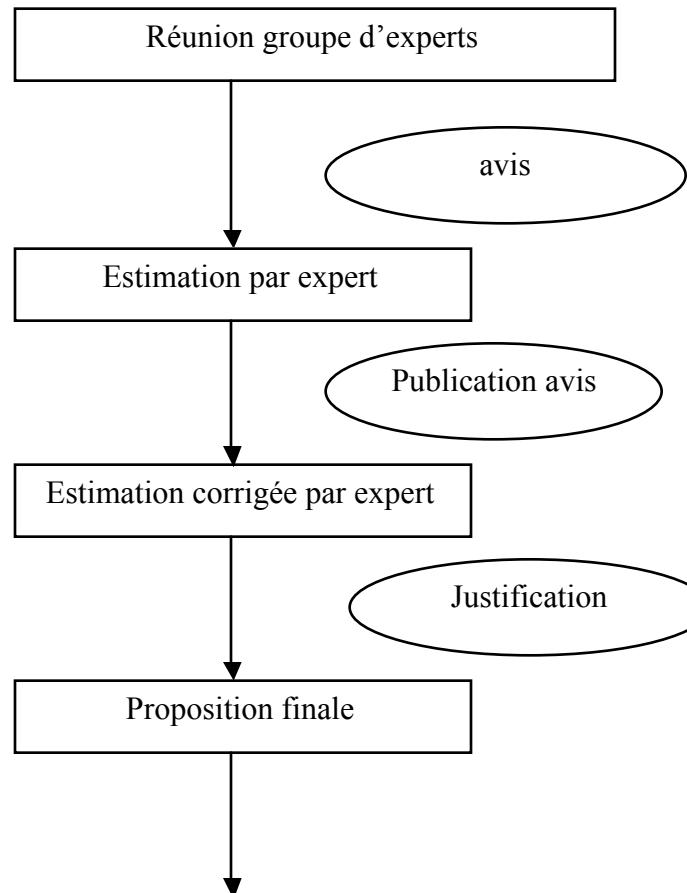
- en référence au cas le plus probable, soit un projet moyen le plus proche possible du projet à estimer.

$$\rightarrow E = (O + 4 * M + P) / 6$$

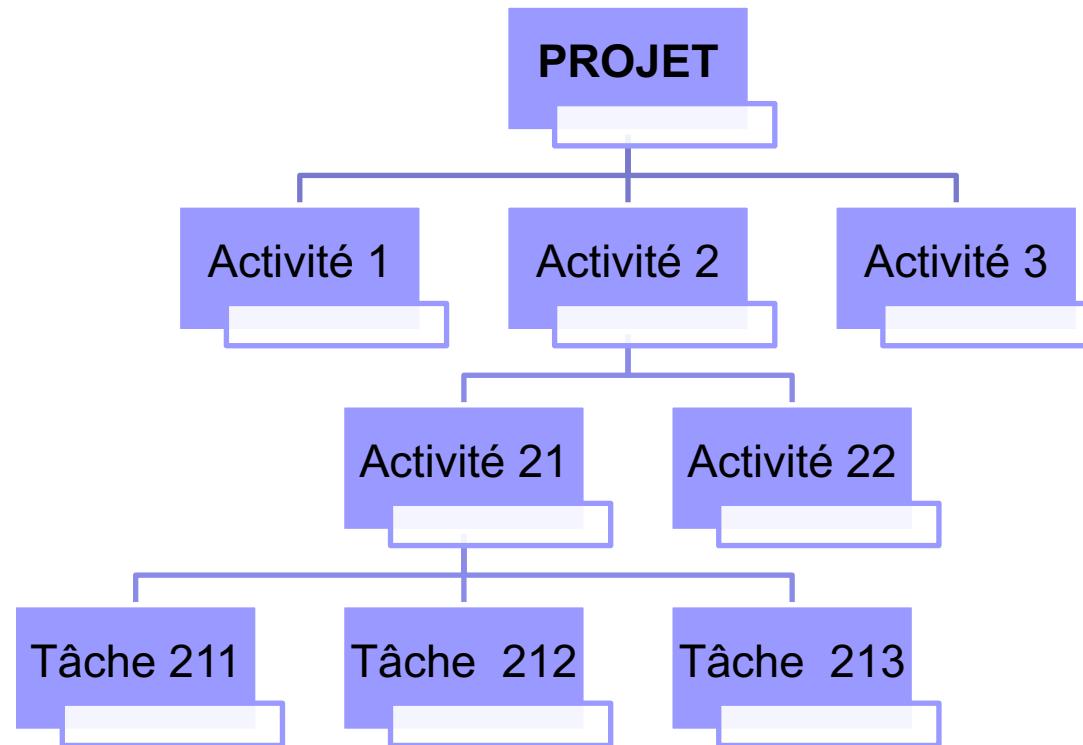
## 3.2- Conception de la TAA

### 3.2.1 Introduction à la conception de la TAA

#### ■ Delphi large Bande



## Analytique

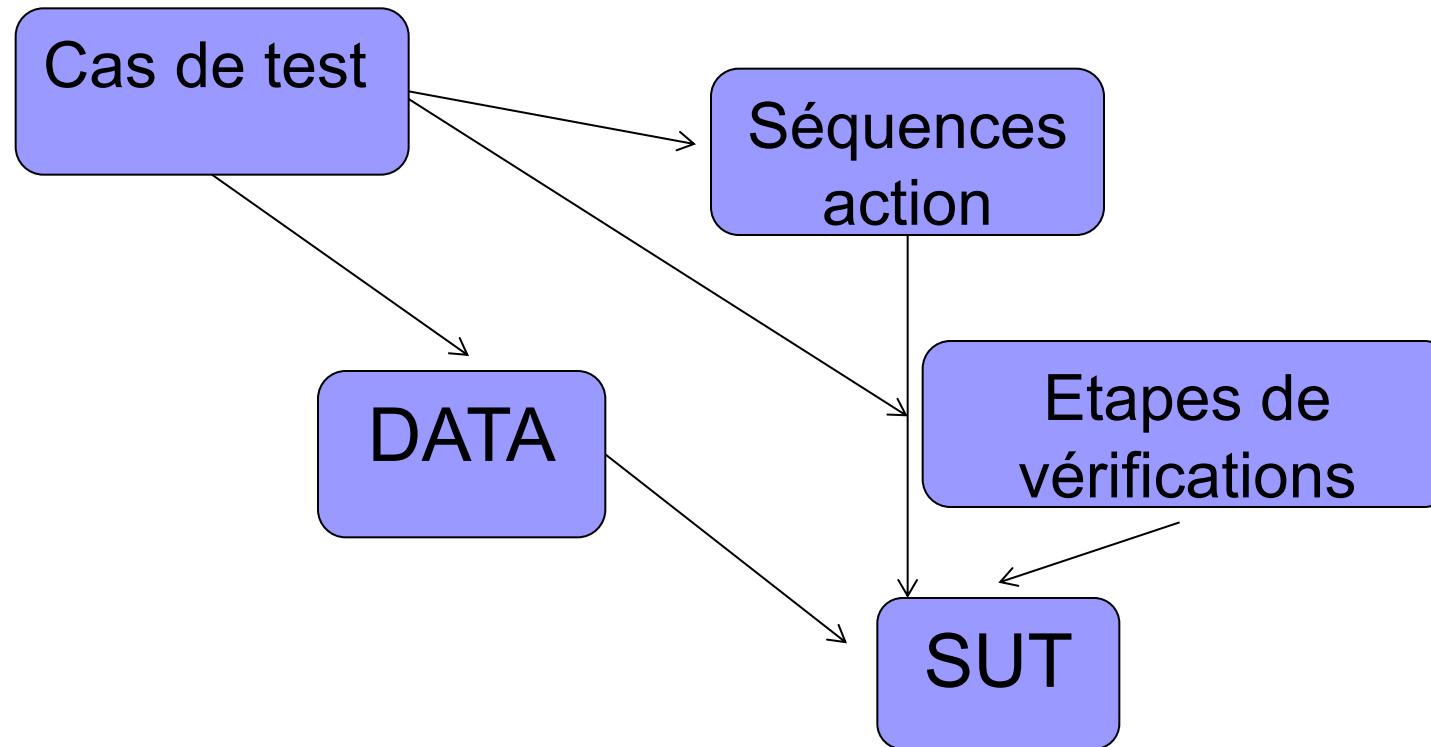


## ■ Evaluation de la complexité en point (planning poker)



## ■ Facilité d'utilisation

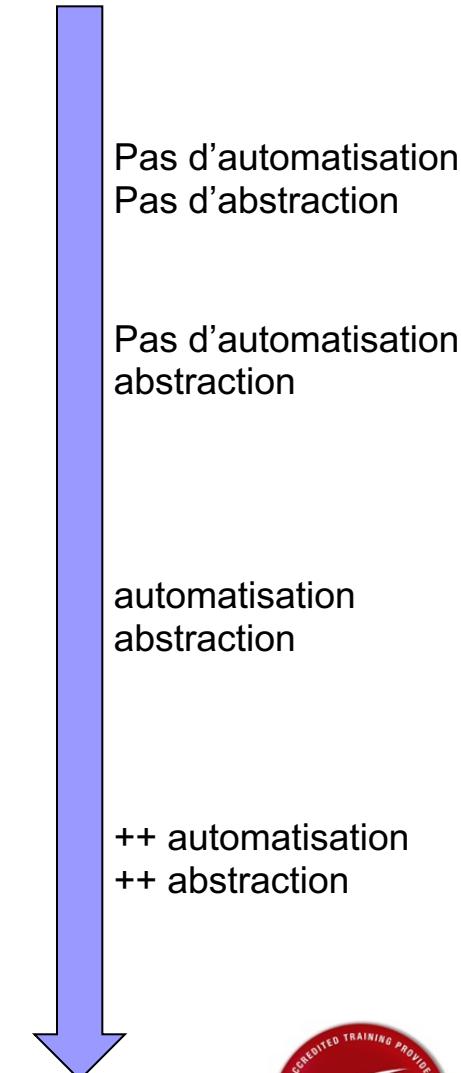
- Conception orientée testeur → souvent orienté développeur
- Facilité d'utilisation de la TAS
- Support de la TAS à d'autres acteurs du développement logiciel, assurance qualité et gestion de projet
- Organisation efficace, navigation et recherche dans/avec la TAS
- Documentation utile, manuels et aide pour la TAS
- Reporting pratique fourni par la TAS et sur la TAS
- Conception itérative pour traiter les feedback et avancer empiriquement sur la TAS



## ■ Options possibles (outils):

1. **Le TAE implémente directement les cas de test en script de test automatique**
2. **Le TAE conçoit les procédures de test et les transforme en script de test automatique**
3. **Le TAE utilise un outil pour transcrire les procédures de test en script de test automatique**
4. **Le TAE utilise un outil qui génère des script de test automatique à partir d'un modèle.**

**Le choix dépend du contexte et peut être évolutif.**



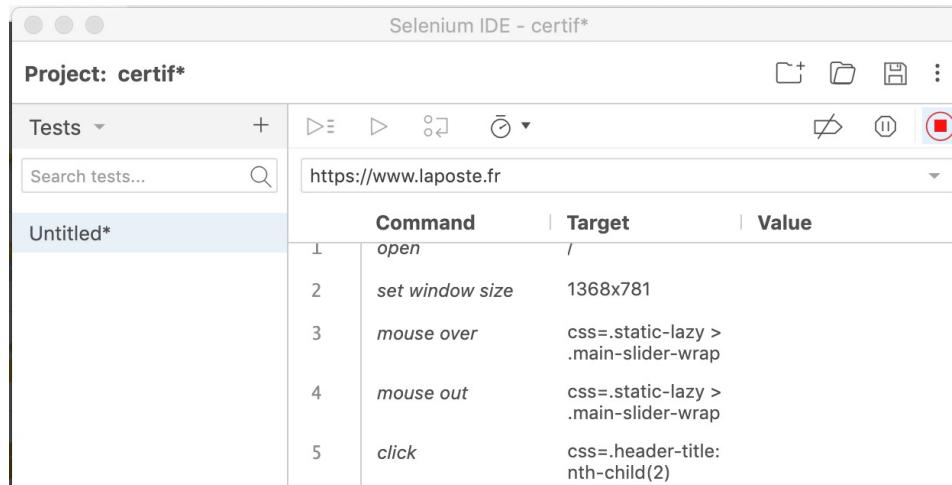
## ■ Les solutions actuelles existantes:

Solution	Options
Basée enregistrement/rejet	Option 1
Approches structurées: Mot clé Piloté par les données Scripting structurée	Option 2 et 3
Test basé sur les modèles	Option 4

## ■ Capture/Rejeu

- **les outils capturent les interactions avec le SUT, les entrées et les sorties peuvent également être capturées.**
- **Néanmoins les vérifications peuvent être**
  - ❖ Manuelles, le testeur regarde
  - ❖ Complètes : toutes les sorties ont été enregistrées
  - ❖ Exactes : toutes les sorties ont été enregistrées selon le niveau de détail sélectionné
  - ❖ Checkpoint : le testeur sélectionne les sorties à enregistrer

## ■ Capture/Rejeu : exemple d'outil: selenium



The screenshot shows the Selenium IDE interface with a project named "certif\*". A test titled "Untitled\*" is displayed with the following steps:

Step	Command	Target	Value
1	open	/	
2	set window size	1368x781	
3	mouse over	css=.static-lazy > .main-slider-wrap	
4	mouse out	css=.static-lazy > .main-slider-wrap	
5	click	css=.header-title: nth-child(2)	

Le testeur peut sélectionner le contrôle qu'il veut mettre en place.

Toutes les actions utilisateurs sont enregistrées

Ouvrir le lien dans un nouvel onglet  
Ouvrir le lien dans une nouvelle fenêtre  
Ouvrir le lien dans la fenêtre de navigation privée

Enregistrer le lien sous...  
Copier l'adresse du lien

Selenium IDE

Inspecter

Voix

LE GR  
LA PO  
ment La Poste  
LE GR  
LA PO

Mouse Over

Verify Text

Verify Title

Assert Text

Assert Title

Store Text

Store Title

Transport et logist

s et livraisons en

colissimo

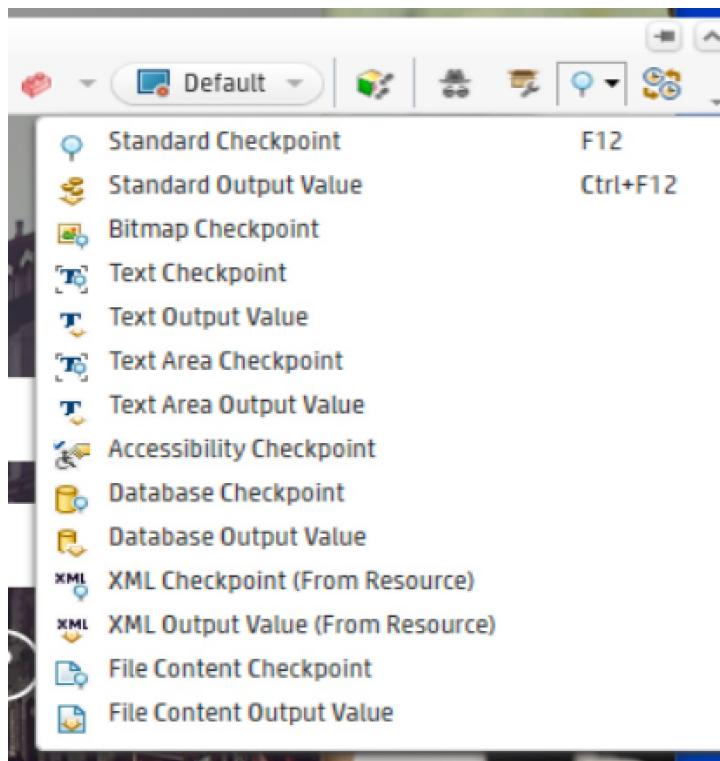
Mediapost

ISTQB

Certified Tester

Advanced Level

## ■ Capture/Rejet : exemple d'outil UFT



Exemple de contrôle spécialisé.

## ■ Capture/Rejeu

### Avantages

Facile à mettre en oeuvre

### inconvénients

Coût de maintenance important du à la fragilité du script (trop proche de l'implémentation).

## ■ Scripting linéaire

- **Les procédures de test sont connus mais pas forcément formalisées.**
- **L'outil enregistre la séquence d'actions quand le testeur exécute manuellement le test**
- **Les scripts sont ensuite édités pour rajouter des commentaires ou des vérifications.**
- **Les scripts peuvent être rejoués à l'infini.**
- **Ces scripts coûtent chers en maintenance, car sensible au changements.**

## ■ Scripting linéaire: exemple

```
_navigateTo("http://qualifiez.fr/examples/Selenium");
SetValue(_textbox(0), "dcfsdc");

.SetValue(_textbox("maClasse[0]"), "cdfsdcsd");
.click(_submit(0));
.assertEqual("Time sheet", _getText(_heading1("Time sheet")));

.assertEqual("", _getValue(_textbox("pw")));
.assertExists(_paragraph("login / pw invalide"));

.click(_submit("connecter"));
.click(_paragraph("login / pw invalide"));
.click(_submit("connecter"));
.click(_paragraph("login :"));
```

## ■ Scripting linéaire

### Avantages

Facile à mettre en œuvre

Nécessite peu de compétence en programmation

### Inconvénients

Coût de développement élevé car pas d'optimisation.

Nécessité de connaître le langage de script qui peut-être propriétaire.

Coût de maintenance important du à la fragilité du script (trop proche de l'implémentation).

## ■ Scripting structuré

- Utilisation de la réutilisabilité pour créer des bibliothèques d'actions et de contrôles.

## ■ Exemple:

```
_include("C:/SQUASH-TA/sahi_v50_20141105/userdata/scripts/global_include.sah")
function doTest($name,$pwd)
{
    _navigateTo("http://dgu-PC/timesheet");
    seLogger($name,$pwd)
    controle("login / pw invalide")
}
var $data = _readCSVFile("data.csv");
_dataDrive(doTest, $data)
```

## ■ Scripting structuré: exemple

```
@Test
public void testRecherche1() throws Exception {
    pageRecherche p2;
    p2 = p1.rechercher("JUPE");
    assertEquals("1 rÈsultat a ÈtÈ trouvÈ.", p2.res());

}
@Test
public void testRecherche2() throws Exception {
    pageRecherche p2;
    p2 = p1.rechercher("ROBES");
    assertEquals(7, p2.nbMeilleursVentes());

}
```

## ■ Scripting structuré

### Avantages

Coût de maintenance réduit.

Coût de développement réduit.

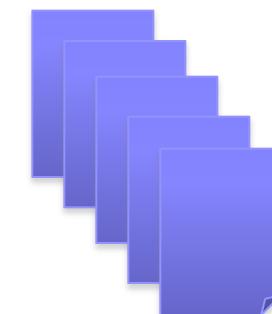
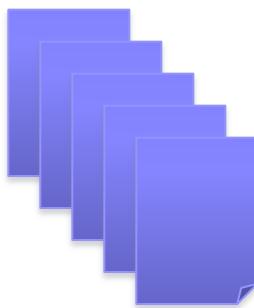
### inconvénients

Investissement initial important.

Compétences en programmation requises.

## ■ Tests pilotés par les données

- Basé sur le scripting structuré
- Les données sont gérées via des fichiers



Jeux d'essais :  
Entrées

Script de contrôle

Référence:  
Sorties

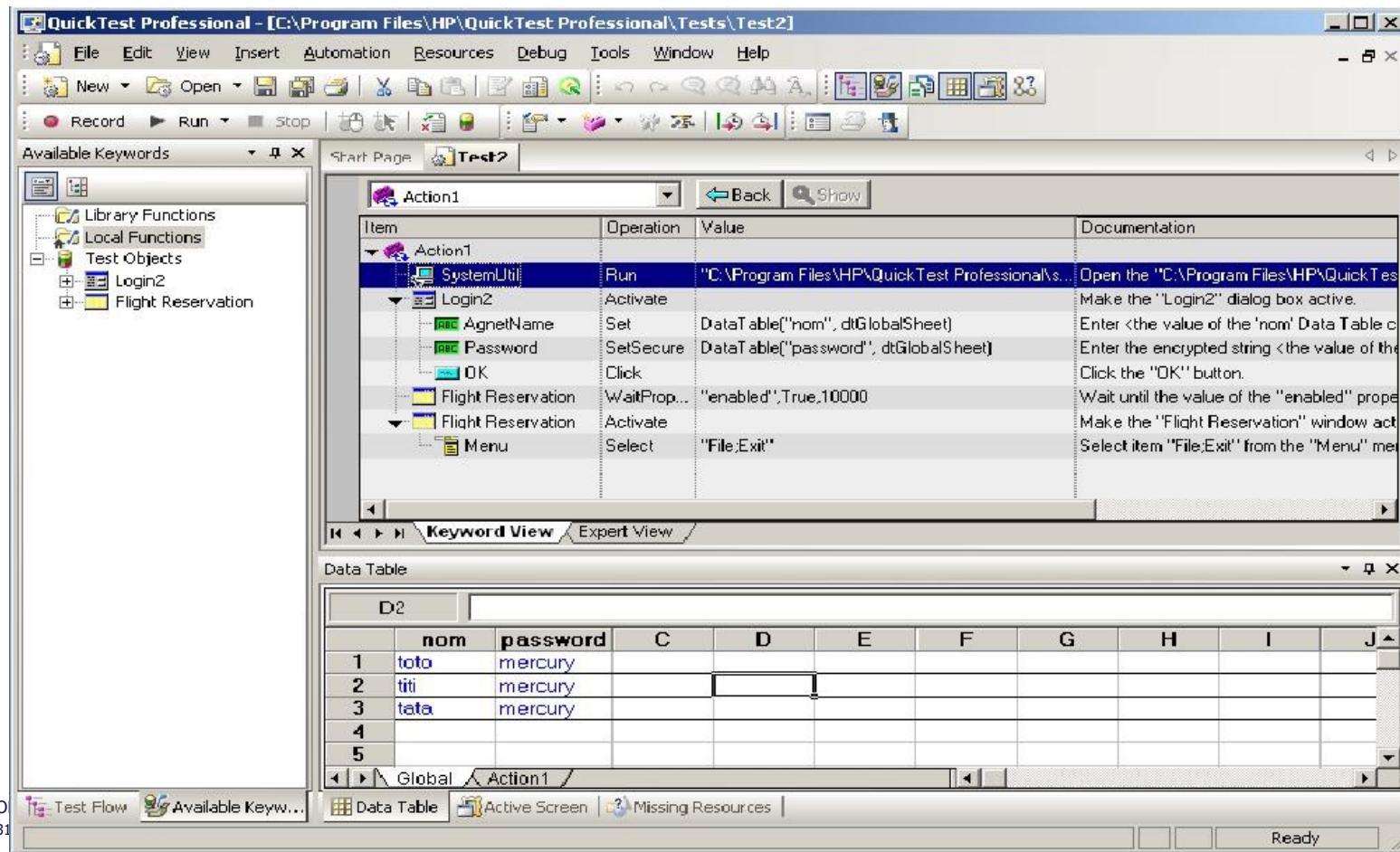
Fichier de  
données, xml,  
excel ...



## 3.2- Conception de la TAA

### 3.2.2 Approche pour l'automatisation des tests

## ■ Test piloté par les données: exemple



The screenshot shows the HP QuickTest Professional interface. The main window displays a test action named "Action1" under the "Test2" test plan. The action tree shows a sequence of operations: Run "SystemUtil", Activate "Login2", Set "AgnetName" to DataTable("nom", dtGlobalSheet), SetSecure "Password" to DataTable("password", dtGlobalSheet), Click "OK", WaitProperty "Flight Reservation" until "enabled" is True, Activate "Flight Reservation", and Select "File;Exit" from the menu. The "Documentation" column provides a brief description for each step. Below the action tree is a "Data Table" window titled "D2" containing a table with columns "nom", "password", C, D, E, F, G, H, I, J. The rows show data for users "toto", "titi", and "tata" with "mercury" as the password. The bottom navigation bar includes tabs for "Test Flow", "Available Keyw...", "Data Table", "Active Screen", and "Missing Resources".



## ■ Test piloté par les données

### Avantages

Augmentation de la profondeur de tests  
(variation d'un même test)

L'analyste de test peut facilement créer  
des tests via les jeux de données

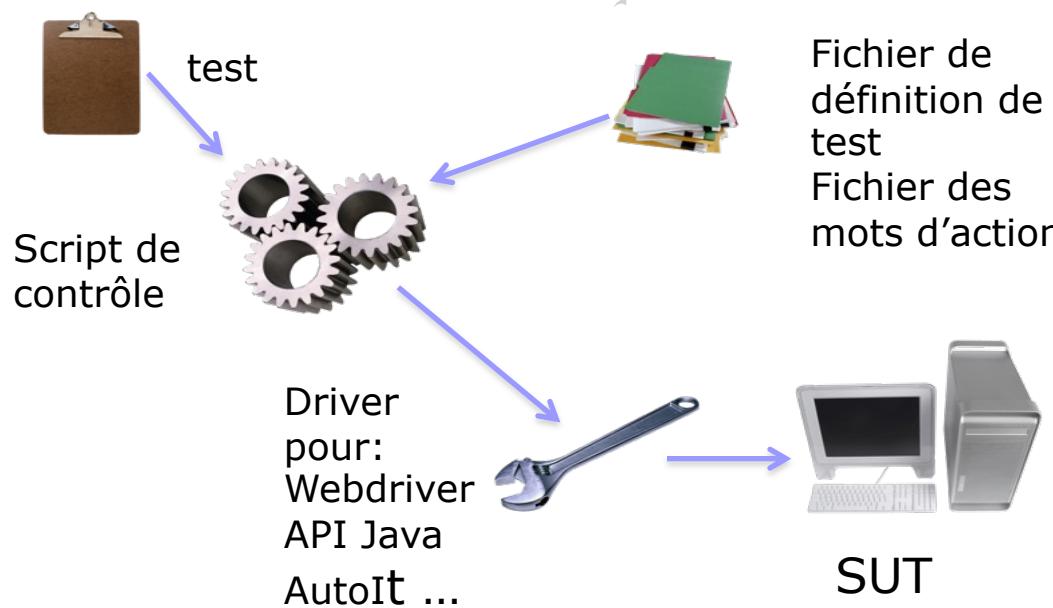
### inconvénients

Gestion des fichiers de données via le  
TAS

Oubli des cas de tests négatifs (cas  
d'erreur)

## ■ Tests pilotés par les mots clés

- Basé sur le test piloté par les données
- Les fichiers de données contiennent les définitions d'actions + 1 seul script de contrôle.



## ■ Tests pilotés par les mots clés

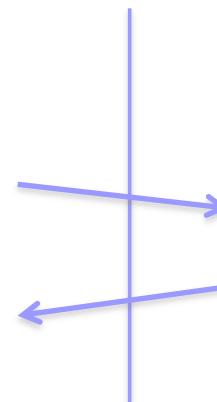
- Les mots-clés représentent des actions de haut niveau orientés métier
- Les analystes de test définissent ces mots-clés
- Un mot clé représente une suite d'actions basiques



1. Spécification des tests

2. Création d'une bibliothèque de mots-clés de haut niveau.

5. Mise au point des mots-clés de haut niveau avec le produit développé.  
7. Exécution des tests.



3. Implémentation via la création d'une bibliothèque de mots-clés de bas niveau.  
4. Mise au point de ces mots-clés avec le produit développé.



## ■ Test piloté par les mots clés

### Avantages

Ajout de nouveaux tests peu couteux  
(une fois le script de contrôle écrit +  
mots clés)  
Scripting possible par analyste de test  
Actions de haut niveau compréhensible  
par tous  
Facile à maintenir

### inconvénients

L'Implémentation des mots clés reste  
une tâche technique  
Faire les bons choix des mots clés

#### ■ Tests pilotés par les processus

- Basé sur le test piloté par les mots clés
- On implémente les cas d'utilisation métier
  
- Ex : passer la commande puis vérifier que la commande est passée.

## ■ Test piloté par les processus

### Avantages

Utilisation des scénarios de cas d'utilisation

Bibliothèques de test dédiées contenant les étapes détaillées

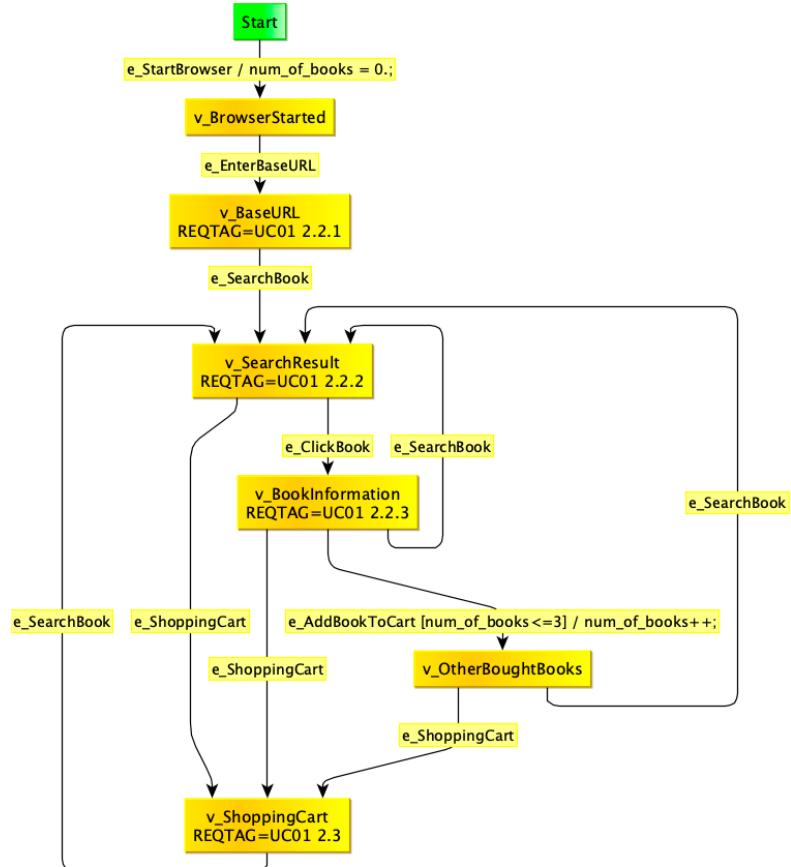
### inconvénients

Plus compliqué pour l'analyste de test technique

Vérifier que les processus et mots clés sont correctement implémentés.

## ■ Tests pilotés par les modèles

- Les scripts de test sont générés à partir de modèles
- Indépendants de la technologies de Scripting



## ■ Test piloté par les modèles

### Avantages

L'analyste se concentre sur le test en terme de logique métier, données, scenarios, configurations.

Les scripts sont générés automatiquement indépendamment de la technologie

En cas d'évolution on ne change que le modèle, la génération de script étant automatisée.

### inconvénients

Capacité d'abstraction pour concevoir le modèle (expertise).

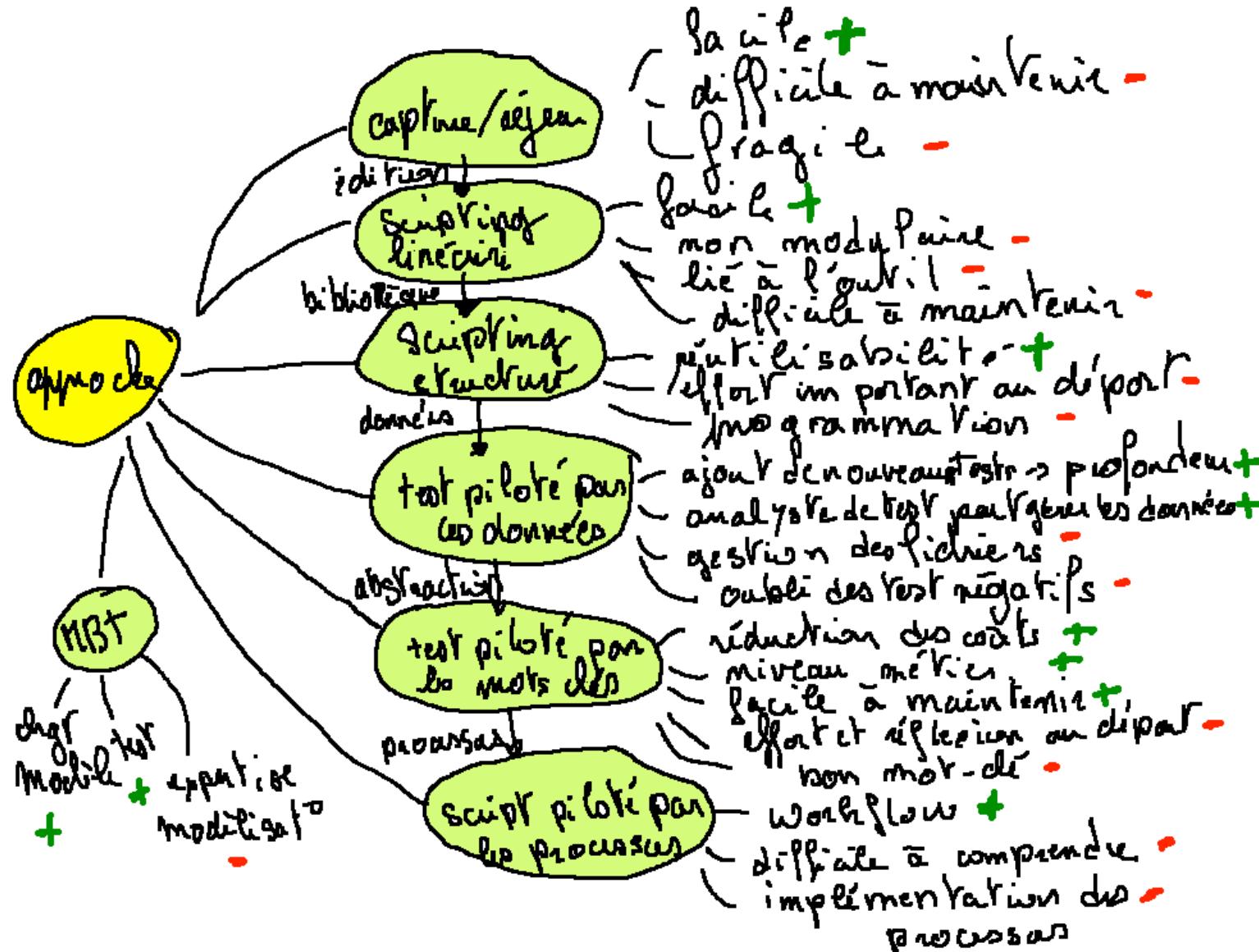
Peu d'outils sur le marché.

Les modèles doivent être vérifiés et consolidés.

# 3-Architecture générique d'automatisation de tests

## 3.2- Conception de la TAA

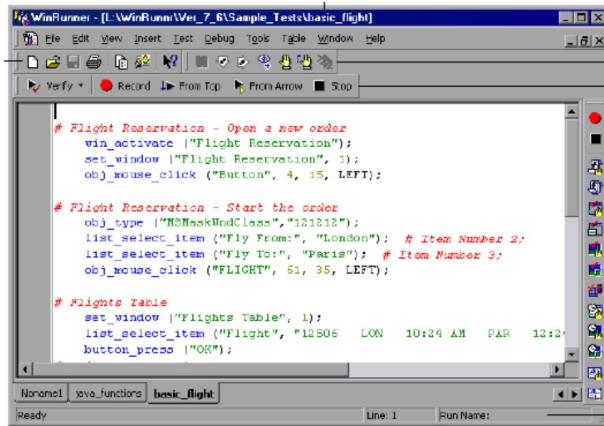
### 3.2.2 Approche pour l'automatisation des tests



### 3-Architecture générique d'automatisation de tests : EXEMPLE fil rouge

#### 3.2- Conception de la TAA

##### 3.2.2 Approche pour l'automatisation des tests



```

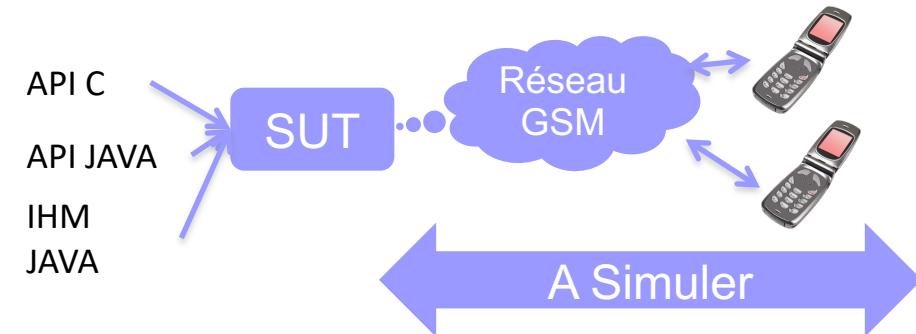
# Flight Reservation - Open a new order
win_activate ("Flight Reservation");
set_window ("Flight Reservation", 1);
obj_mouse_click ("Button", 4, 15, LEFT);

# Flight Reservation - Start the order
obj_type ("MSMaskWndClass", "121212");
list_select_item ("Fly From:", "London"); # Item Number 2
list_select_item ("Fly To:", "Paris"); # Item Number 3
obj_mouse_click ("FLIGHT", 61, 35, LEFT);

# Flights Table
set_window ("Flights Table", 1);
list_select_item ("Flight", "12506 LON 10:24 AM PAR 12:2");
button_press ("OK");

```

The screenshot shows the WinRunner test script editor interface. The script window displays a series of commands for automating a flight reservation process. The commands include opening a window, selecting items from dropdown menus, and interacting with a flight table. The bottom of the screen shows tabs for 'Noname1', 'java\_junctions', and 'basic\_flight', with 'basic\_flight' currently selected.



- Pour les IHMs et APIs, automatisation basée données + structurée.
- Etant donnée la durée de vie de ces applications et les évolutions prévues il était important d'assurer la maintenabilité.
- Par la suite un moteur de test maison, a été développé permettant de modéliser les tests au format XML → rajout d'un niveau d'abstraction.

- Vous devez implémenter l'automatisation pour un système avec les fonctionnalités suivantes:
  - Recevoir des messages du hardware
  - Gestion des messages d'erreurs par le moteur de règles
  - Envoyer des emails à l'administrateur (contient des recommandations pour les modifications du hardware ou de sa configurations)
- Vous avez cherché une solution d'automatisation et vous avez trouvé que vous pouvez automatiser la génération de message, la réception des messages et le traitement des messages. Etant donné la définition de couche de définition, quelle serait la meilleure approche pour la définition des tests:
  - a) Enregistrement/rejet
  - b) Script basé sur les données
  - c) Scripting linéaire
  - d) Scripting structuré

- Si le test basé sur les modèles a été sélectionné comme approche d'automatisation, comme cela influence les couches de la TAA:
  - a) Toutes les couches sont utilisées mais la génération de test est automatisée à partir du modèle
  - b) Il n'y a pas besoin de couche d'exécution par que la définition des tests est suffisante pour tester le software
  - c) Pas d'adaptation nécessaire car les interfaces sont définis par le modèle.
  - d) Il n'y a pas besoin de concevoir les tests API car ils sont couverts par le modèle

## ■ Interfaces

- ❖ Internes ou externes → il faut prévoir les interfaces pour les tests pendant la conception du SUT

## ■ Données

- ❖ Configurations, instantiations, administrations, données utilisateurs ou en provenance d'autres systèmes → Il faut prévoir de façon automatique la mise à jour ou l'insertion de ces données.

## ■ Configurations du SUT

- ❖ Un SUT peut être déployé dans différents environnements techniques, la TAA doit être en mesure de gérer ces différentes configurations (VM, simulateurs ou émulateurs)...

## ■ Normes et éléments juridiques

- ❖ Confidentialité des données → en prévoir la gestion dans le TAA

## ■ Outils et environnements outillés pour développer le SUT

- ❖ Exigences, modélisation, IDE, intégration, déploiement → le TAA devrait pouvoir s'interfacer pour la traçabilité, compatibilité, réutilisation d'artefact

## ■ Interfaces de test dans le produit logiciel

- ❖ Laisser des interfaces spécifiques aux tests peut être utiles aux activités de support/maintenance.
- ❖ Vérifier qu'elles ne sont pas accessibles par d'autres et qu'elles n'introduisent pas des failles de sécurité.

## Données: gestion du contenu initial de la base :

### ■ Recopie d'une base de production.

- ❖ Données réalistes.
- ❖ Tous les cas de test ne sont pas réalisables par manque de données—> compléter.
- ❖ Adapter les tests automatiques aux données, non maîtrise des données.
- ❖ Rendre anonyme les données pour des considérations légales

### ■ Import/export ou outil type dbUnit.

- ❖ Maîtrise des données de tests.
- ❖ Prévoir de la diversification.
- ❖ Sensibles aux évolutions de la structure de la base de donnée.

### ■ Alimentation par API ou IHM.

- ❖ Pas sensible aux évolutions de la base de données.
- ❖ Nécessite de tester en premier tout ce qui concerne les API d'alimentation.

→ Il est préférable de prévoir des procédures automatiques pour alimenter la base de donnée, les actions manuelles étant sujettes à erreur.

## ■ Données: fichier de configuration.

□ Je démarre le système avec un fichier de configuration x qui permet d'alerter via SMS.

1. Je stoppe le système et j'écrase le fichier de configuration avec le fichier x

2. Après redémarrage on exécute le ou les tests.

1. Je stoppe le système et j'écrase le fichier de configuration avec le fichier initial.



Rq 1. Le fichier de configuration doit être sauvegardé avec le test.

Rq 2. On peut également prévoir une procédure automatique

## ■ Exemple

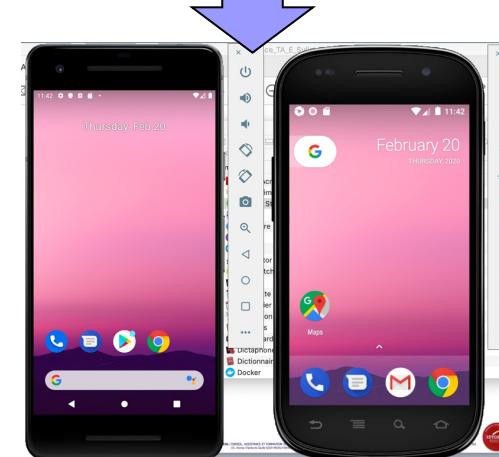
- Une application android peut-être testée sur différents émulateurs.
- Le choix de l'émulateur se fait au niveau du script, soit au moment de l'exécution (paramètre)

```
test_demo
[Tags] regression
lancer l'application
lancer le de
vérifier

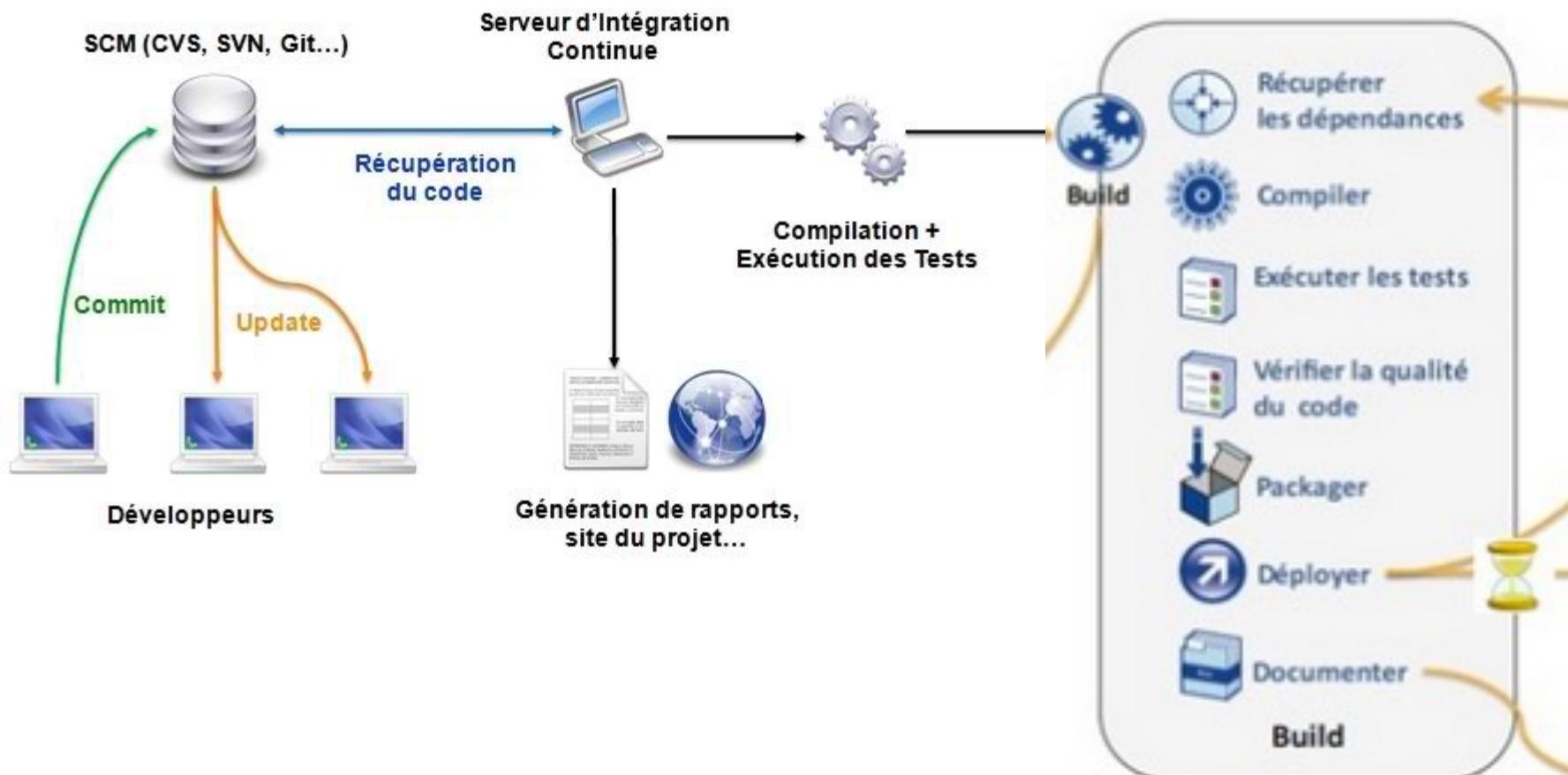
*** Keyword ***
lancer l'application
Open Application http://localhost:4723/wd/hub platformName=Android platformVersion=10 deviceName=titi

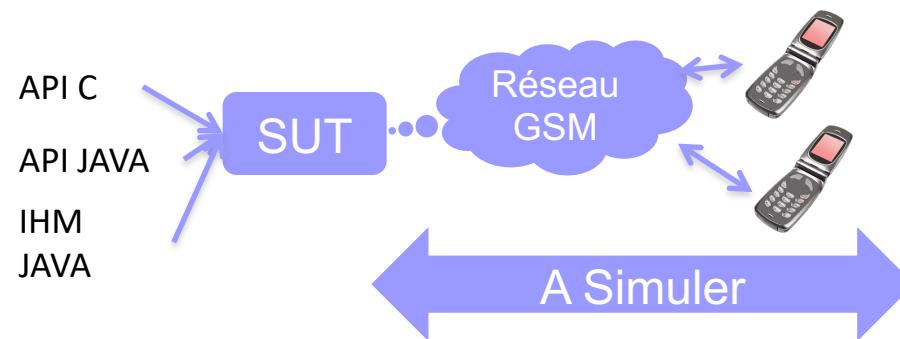
lancer le de
Click Element class=android.widget.Button

vérifier
${Var} Get Text id=com.example.dominiquemereaux.dice:id/textTitle
Should Be Equal ${Var} 6 sided dice
```



## ■ Les tests s'intègrent dans le gestionnaire d'intégration continue





- Données → Les données nécessaires aux tests seront alimentés via des API de provisioning du SUT → implique que ces API soit livrées et testées au plus tôt.
- Environnement: API Java ou C++ sous Windows ou linux → utilisation de scripts python/jython qui s'utilisent indifféremment dans divers environnements.

## ■ Exigence de contrôle d'exécution de test

- ❖ Batch, exécution interactive (via IHM) doivent pouvoir être pris en compte

## ■ Exigences de reporting

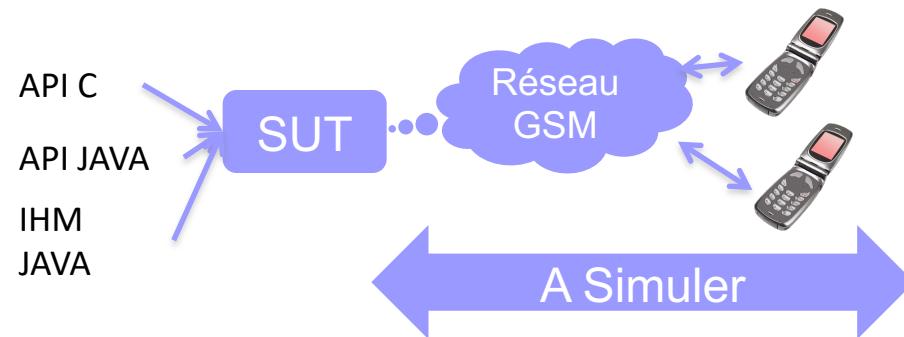
- ❖ Modèle et formats de rapports

## ■ Rôles et droits d'accès

- ❖ Pour des questions de sécurité la TAA pourrait avoir à prendre en charges des rôles et droits d'accès

## ■ Panorama des outils en place

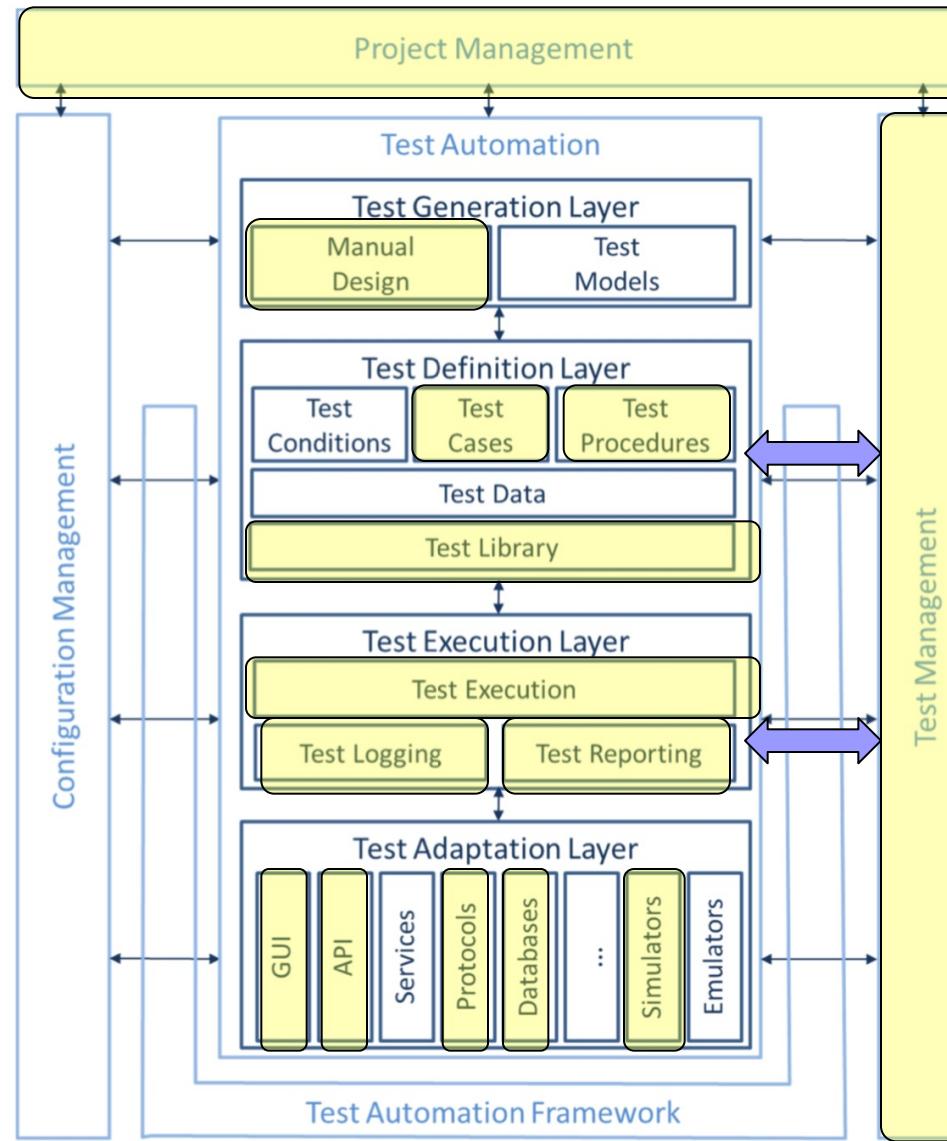
- ❖ Les outils utilisés par la TAA doivent pouvoir s'intégrer avec les autres outils
- ❖ Les scripts de tests doivent être versionnés.



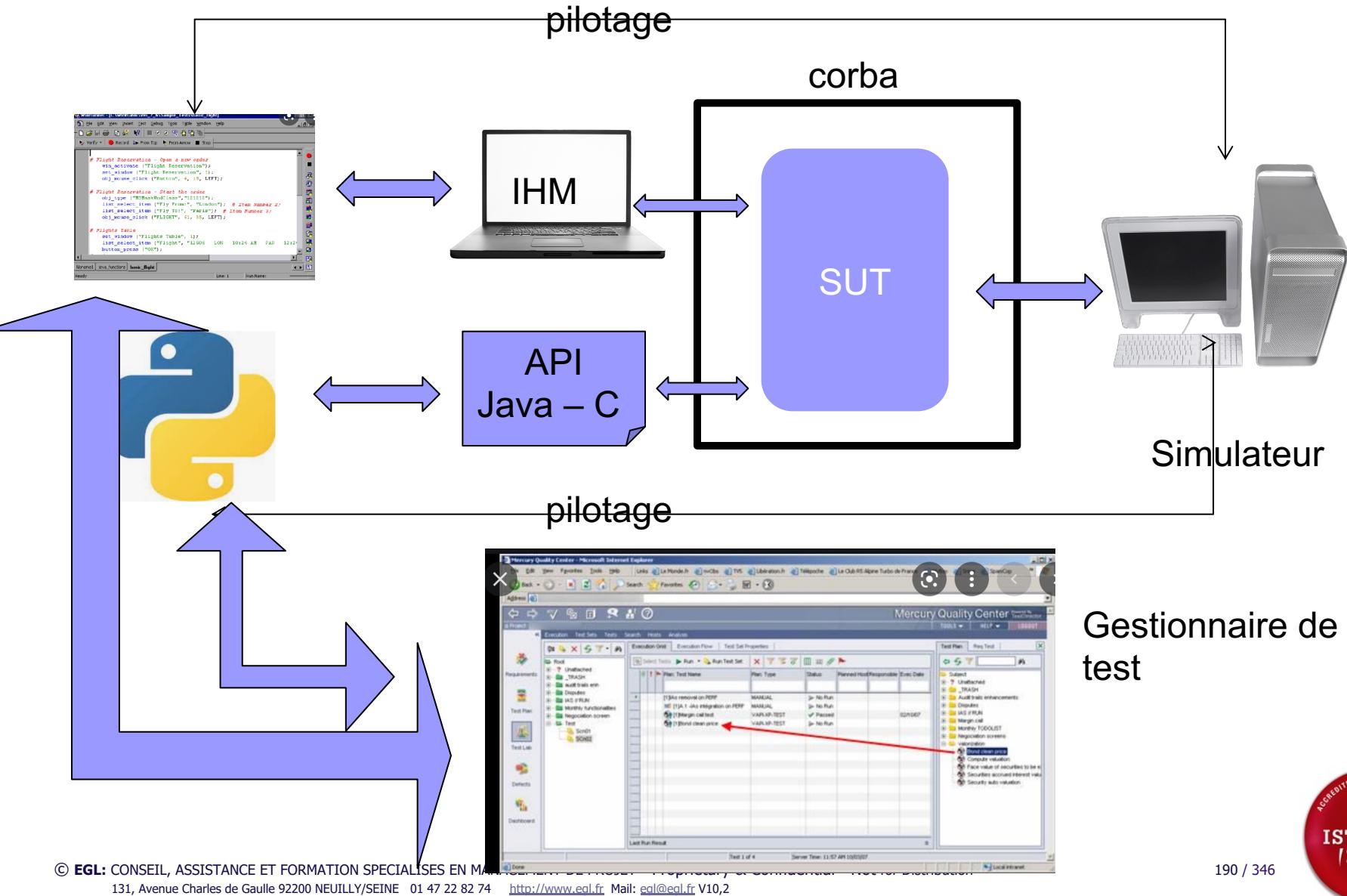
- Les rapports de tests manuels et automatiques sont consolidés. L'exécution des tests automatiques se fait à partir de l'outils de gestion de test (quality center).

### 3-Architecture générique d'automatisation de tests : EXEMPLE fil rouge

#### 3.2 Conception de la TAA



# 3-Architecture générique d'automatisation de tests : EXEMPLE fil rouge : Architecture



### ■ Exigences simulateur

- Le simulateur doit tenir la charge.
- Le simulateur doit être pilotable via des APIs
- Le simulateur doit posséder une interface de test

...

### ■ Exigences exécutions des tests

- Les tests d'IHM doivent être lancés via le gestionnaire de test
- Les tests d'API doivent être pilotables via le gestionnaire de test

...

...

- **Quel est l'avantage premier d'utiliser l'abstraction dans la TAA?**
  - a) Cela permet plus de flexibilité pour le futur
  - b) Cela demande plus de compétences pour l'implémenter
  - c) Cela permet de supporter n'importe quelle méthode de scripting
  - d) Cela améliore la performance du TAS
- **Quel est le désavantage premier de l'utilisation de l'abstraction de la TAA?**
  - a) Cela permet plus de flexibilité pour le futur
  - b) Cela demande plus de compétences pour l'implémenter
  - c) Cela permet de supporter n'importe quelle méthode de scripting
  - d) Cela améliore la performance du TAS

- Vous développez une TAS pour une application qui permet aux étudiants de sélectionner, s'inscrire à des cours et de payer en ligne. L'inscription aux cours est contrôlé par un ensemble de règles qui prennent en compte la spécialisation de l'étudiant, les pré-requis, le planning et la charge. Les données des cours changent fréquemment incluant les pré-requis et les dates. Une de vos préoccupations est de construire un framework d'automatisation qui va vous permettre de gérer les changements de données tout en fournissant un résultat correct. A quelle caractéristique doit répondre la TAS pour la maintenir malgré les données changeante:
  - a) Capacité de la TAS à déterminer la bonne règle en fonction des données en entrées.
  - b) Capacité de la TAS à contrôler la décision (inscription)
  - c) Capacité de la TAS à prendre les entrées utilisateur pour la sélection des cours
  - d) Capacité de la TAS à fournir des résultats montrant les règles sélectionnées et les données en sortie

## 3.2- Conception de la TAA Exercice

**Vous développez une TAS pour une application qui permet aux étudiants de sélectionner, s'inscrire à des cours et de payer en ligne. L'inscription aux cours est contrôlé par un ensemble de règles qui prennent en compte la spécialisation de l'étudiant, les pré-requis, le planning et la charge. Les données des cours changent fréquemment incluant les pré-requis et les dates.**

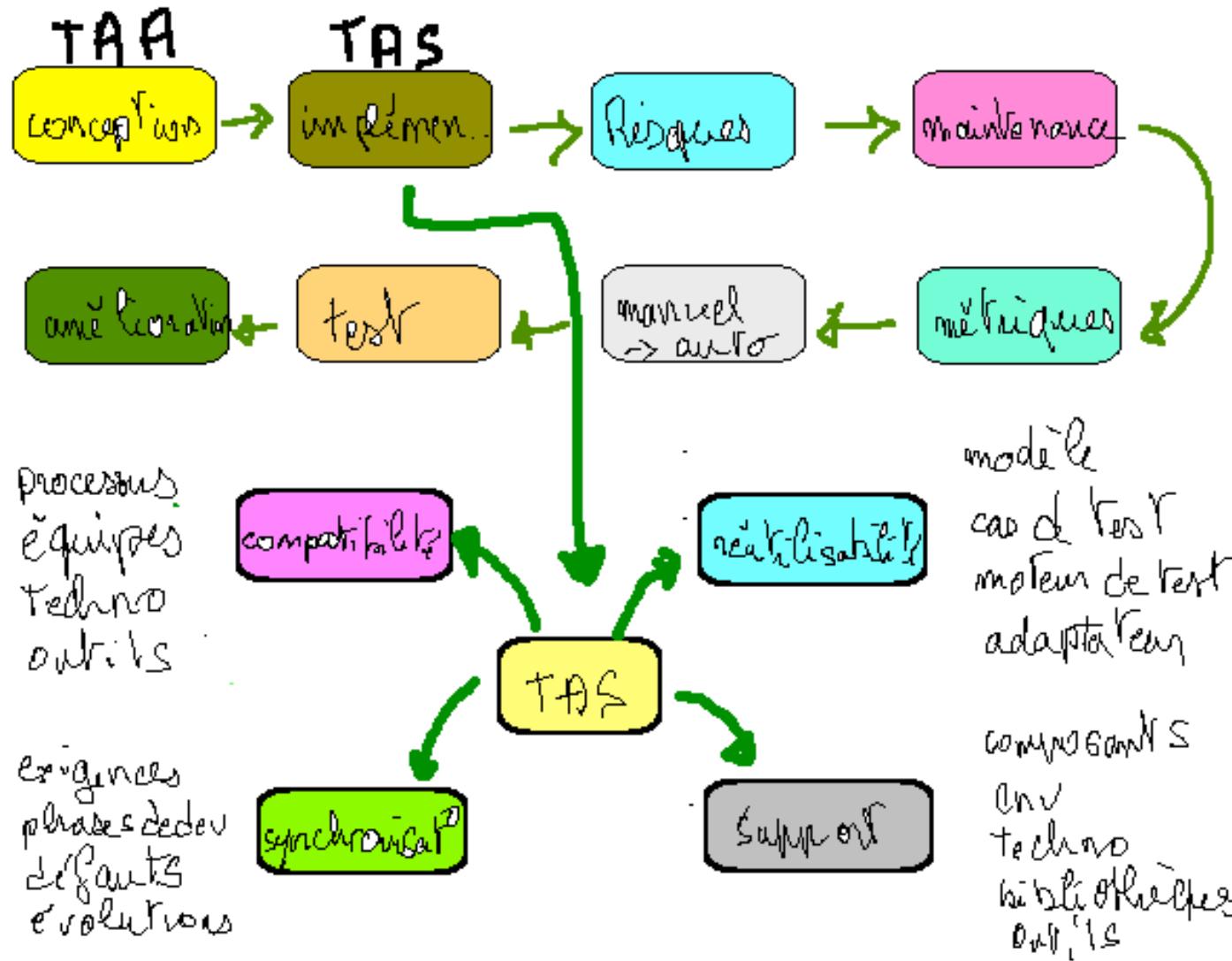
**Une de vos préoccupations est de construire un framework d'automatisation qui va vous permettre de gérer les changements de données tout en fournissant un résultat correct.**

**Les développeurs ont accepté de construire une interface de test qui vous permettra d'interroger l'entrée du moteur de règles et de récupérer les résultats de la décision. En utilisant cette interface de test, vous avez pu construire votre automatisation de test de manière à ce qu'elle puisse traiter différentes entrées et vous pouvez valider que les résultats sont corrects.**

**Le SUT est prêt à être mis en production. Que doit-on faire avec l'interface de test ?**

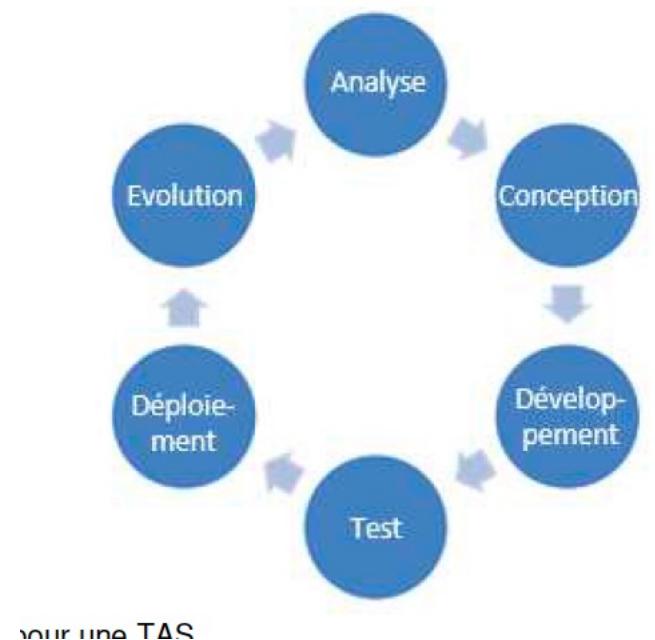
- 1. Il doit être supprimé du code pour éliminer les risques de sécurité.**
- 2. Il doit être laissé dans le code afin de minimiser les dysfonctionnements dans le code avant sa publication.**
- 3. Il doit être désactivé dans le code pour la production mais facilement activé pour les tests des futures versions**
- 4. Elle n'aurait pas dû être utilisée pour les tests en raison du risque "effet de sonde".**

- Si le SUT a une API que vous devez tester, quelle partie de la TAA va être utiliser pour créer ces tests
  - a) Générateur d'API
  - b) La couche d'interface
  - c) La couche d'adaptation
  - d) La couche de conception
- Vous implémentez l'automatisation pour un projet avec des contraintes de sûreté. Le reporting de l'automatisation est extrêmement important et doit être fiable. Vous voulez compiler les résultats des tests auto et manuel dans votre outil de gestion de test. Quelle couche de la TAA vous permet de gérer ce point:
  - a) Couche de reporting
  - b) Couche de logging
  - c) Couche d'exécution
  - d) Couche d'adaptation



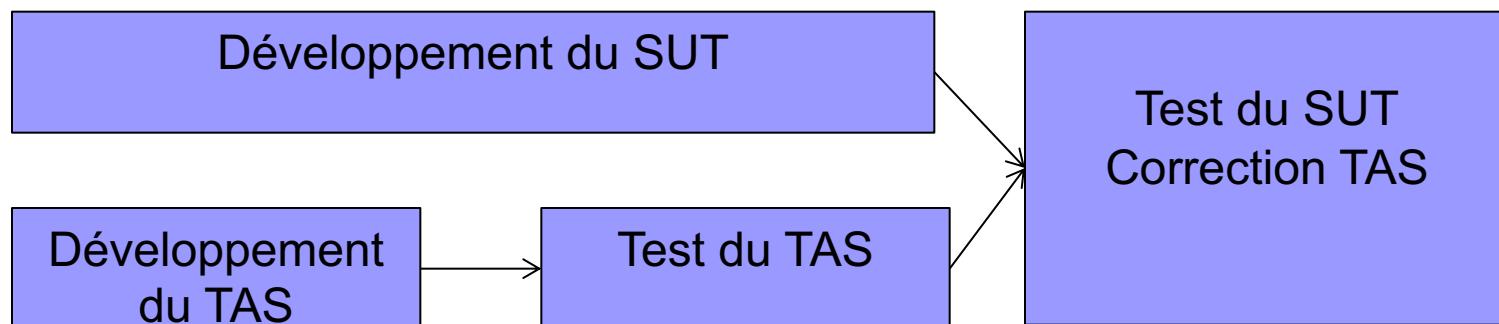
- Il s'agit d'un projet logiciel à part entière

- Gestion de projet, planning, coût ...
  - Exigences, architecture, conception, développement, test, déploiement, évolution.
  - Il faudra également prévoir la sauvegarde, l'archivage et la suppression selon les procédures établies pour tout artefact logiciel.



## ■ Compatibilité des processus

- L'ensemble des développements (TAS et SUT) devrait être synchronisés de façon à assurer le test du SUT en temps et en heure.
- Si le planning de test du SUT se base sur la disponibilité du TAS et des tests automatiques, il est essentiel de suivre l'avancement du TAS et les risques associés.



## ■ Compatibilité des équipes

- Bénéfices grâce à la collaboration des équipes
- Revue des exigences TAS et SUT
- Conception et développement
- ...

## ■ Compatibilité technologique

- Cela peut passer par l'utilisation des mêmes langages,  
**En quoi cela peut-il être intéressant?**
- En cas d'incompatibilité on peut utiliser des surcouches,  
adaptateur

## ■ Compatibilité des outils

- Même gestionnaire exigences, anomalies ...
- Suivi des mêmes processus qualité
- Facilitent la transparence, collaboration

## ■ Exigences

### □ Exigences de la TAS elle-même

- ❖ Conception de test
- ❖ Développement de test
- ❖ Analyse des résultats de test
- Ce sont des exigences génériques applicables à toutes les instances de TAS

### □ Exigences relatives au SUT

- ❖ Relatives aux propriétés et caractéristiques qui doivent être testées par la TAS
- Si le SUT est modifié, il faut s'assurer que la TAS reste compatible

## ■ Exemple d'exigences

### □ Exigences de la TAS elle-même

- ❖ Les résultats de test automatique pourront être publiés via le serveur d'intégration continue.
- ❖ Les Logs de test doivent être datés

### □ Exigences relatives au SUT

- ❖ La TAS doit pouvoir piloter le SUT via des appels REST
- ❖ L'état des variables internes du SUT doit être présent dans les logs de test.

## ■ Synchronisation des phases de développement

### □ Objectif TAS disponible en tant et en heure

❖ Meilleure efficacité.

## ■ Synchronisation de la gestion des défauts.

### □ TAS

### □ SUT

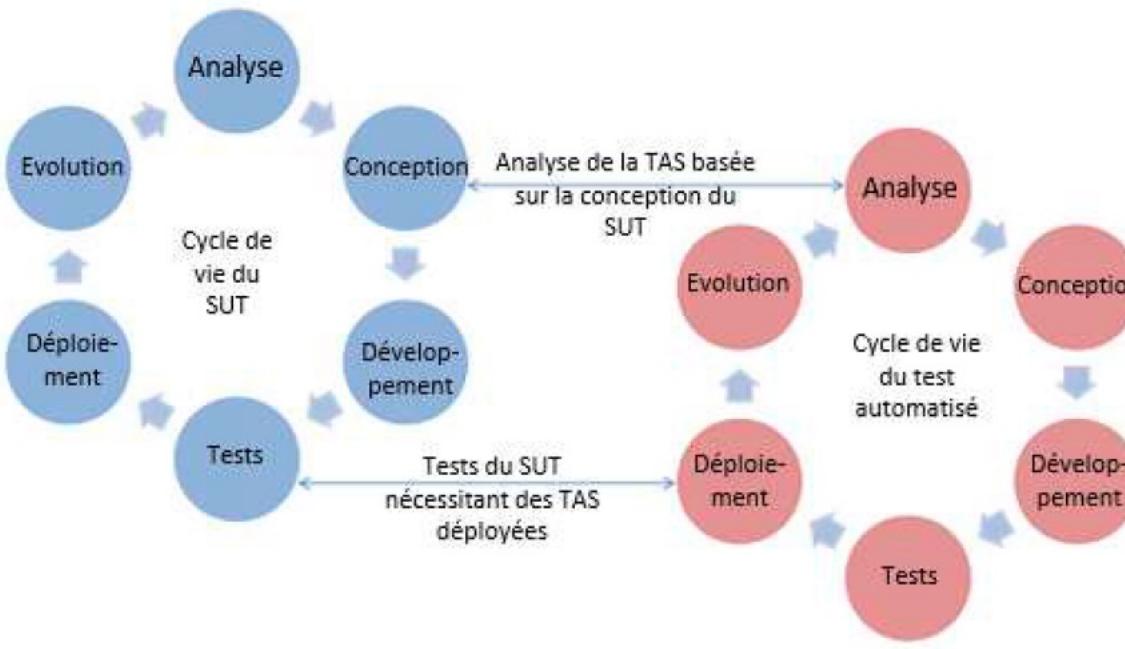
### □ Exigences/Conception/Spécification

→ La correction d'un défaut sur le SUT peut avoir un impact sur la TAS et vice-versa.

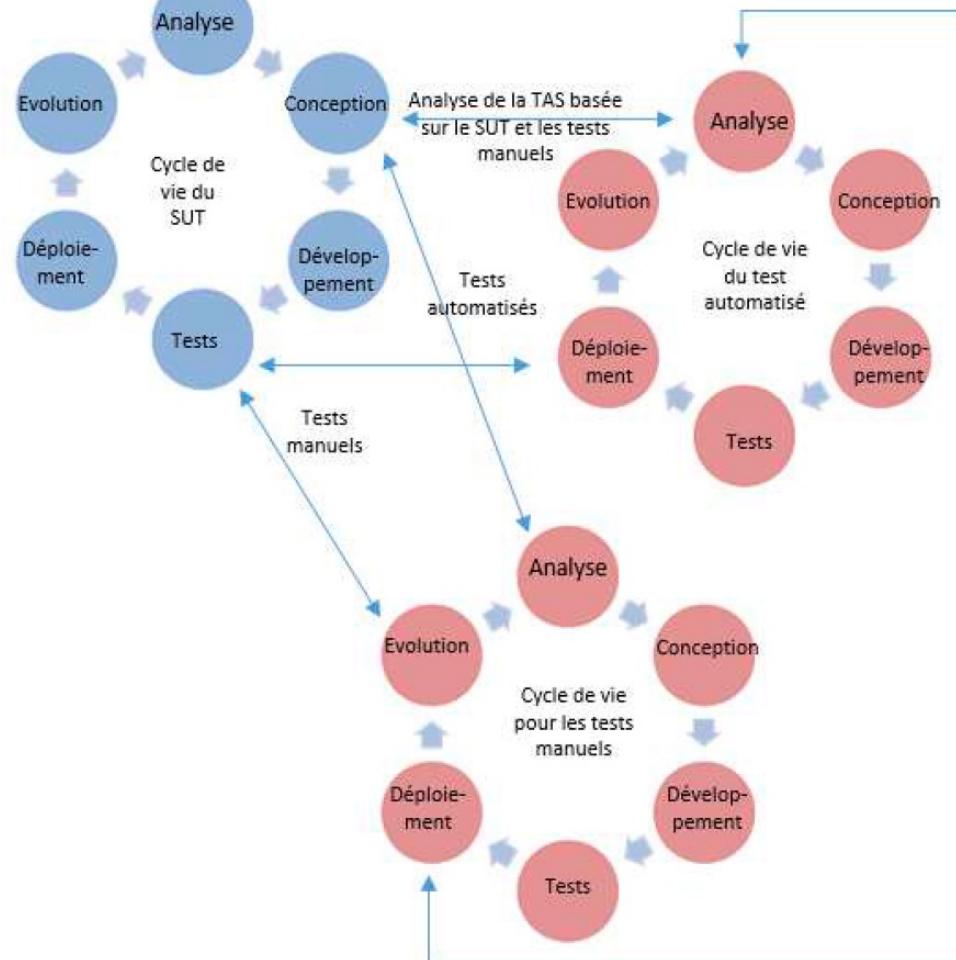
## ■ Synchronisation des évolutions du SUT et du TAS

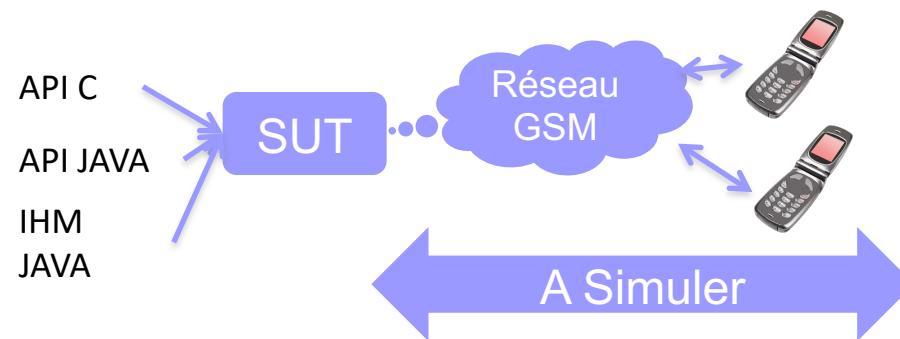
- Une évolution du SUT peut avoir des impacts sur le TAS
  - ❖ Perte de contrôlabilité ou observabilité
  - Synchronisation des évolutions (y compris le coût)
  - Ex: l'utilisation d'un framework pour générer des pages web avec des Id variables
- Une évolution dans le TAS peut également avoir un impact sur le SUT

## ■ Exemple de synchronisation



## ■ Exemple de synchronisation





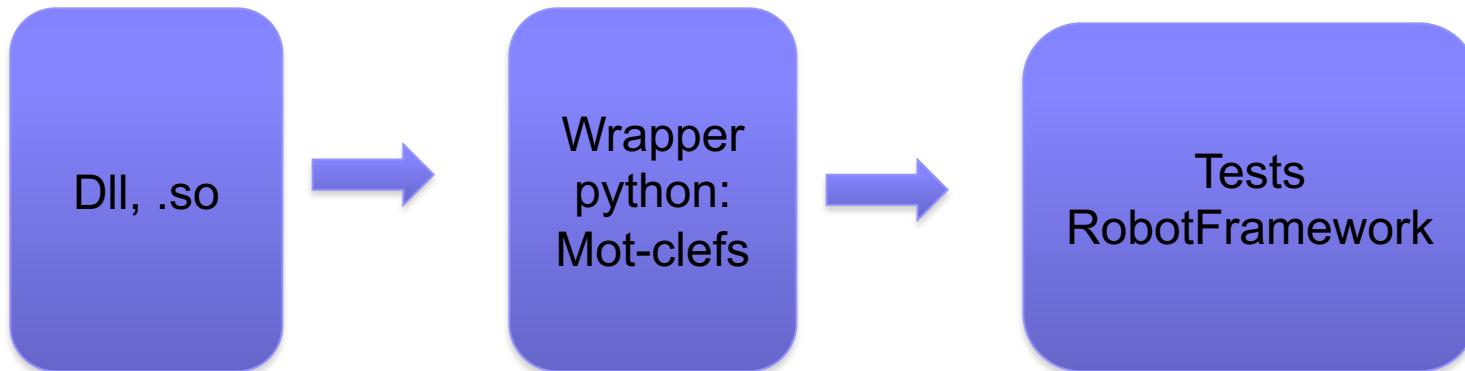
- Evolution IHM Swing → IHM Web
  - Migration vers QTP qui supporte le Web
- Evolution API CORBA → API SOAP
  - Mise en place d'outils pour tester ce nouveau type d'interface et évolution de la TAS pour intégrer ce nouveau type de test
  - Formation des testeurs aux nouvelles technologies

## ■ Que peut-on réutiliser sur différents projets?

- Parties de modèles d'objectif de test, scénarios de test, composant de test ou données
- Parties de cas de test, procédures de test, bibliothèques de test.
- Le moteur de test, le cadre (framework) de reporting
- Les adaptateurs
- Exemples:
  - Un fichier excel contenant des données de contenu (utilisateur) pourrait être réutilisées dans divers tests.
  - Un composant permettant de se connecter au SUT.
  - Une bibliothèque permettant d'envoyer et de recevoir des requêtes SOAP

## ■ Maintenabilité et réutilisabilité

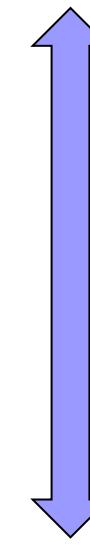
- Des interfaces génériques au lieu d'interfaces spécifiques
- Documentation/formation sur les artefacts de la TAS de façon à faciliter l'adoption par les différentes équipes
- Maintenance et suivi de TAS en prévision des évolutions possibles.



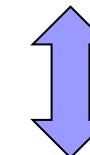
Exemple: Si la signature d'une fonction change, on modifie le wrapper et non le test lui-même → ainsi on anticipe une possible évolution de la bibliothèque à tester.

#### ■ Capacité à supporter:

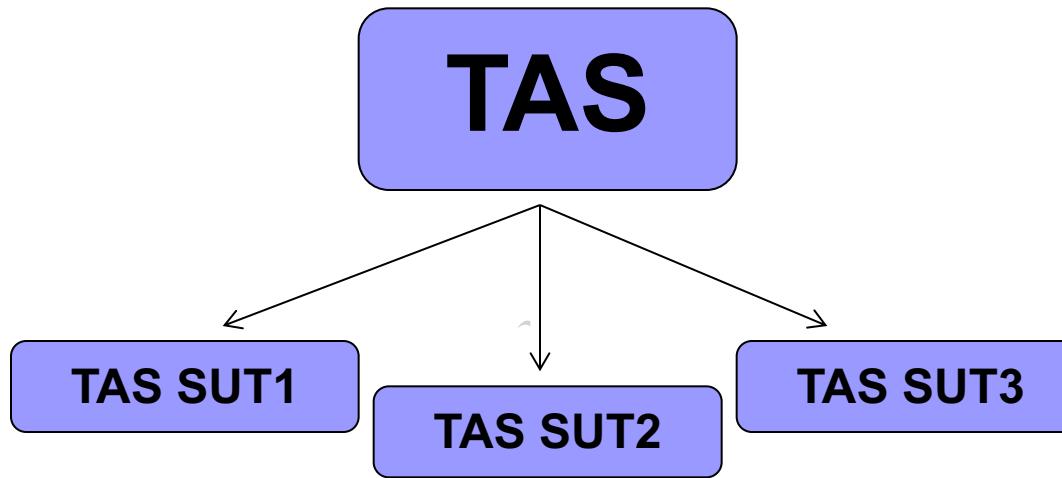
- Nombre et interconnexion des composants du SUT
- Environnements logiciels et matériels
- Technologies (langages, OS)
- Bibliothèques et package utilisées par le SUT
- Outils pour mettre en œuvre les composants du SUT



Tous niveaux de tests



Unitaire et intégration

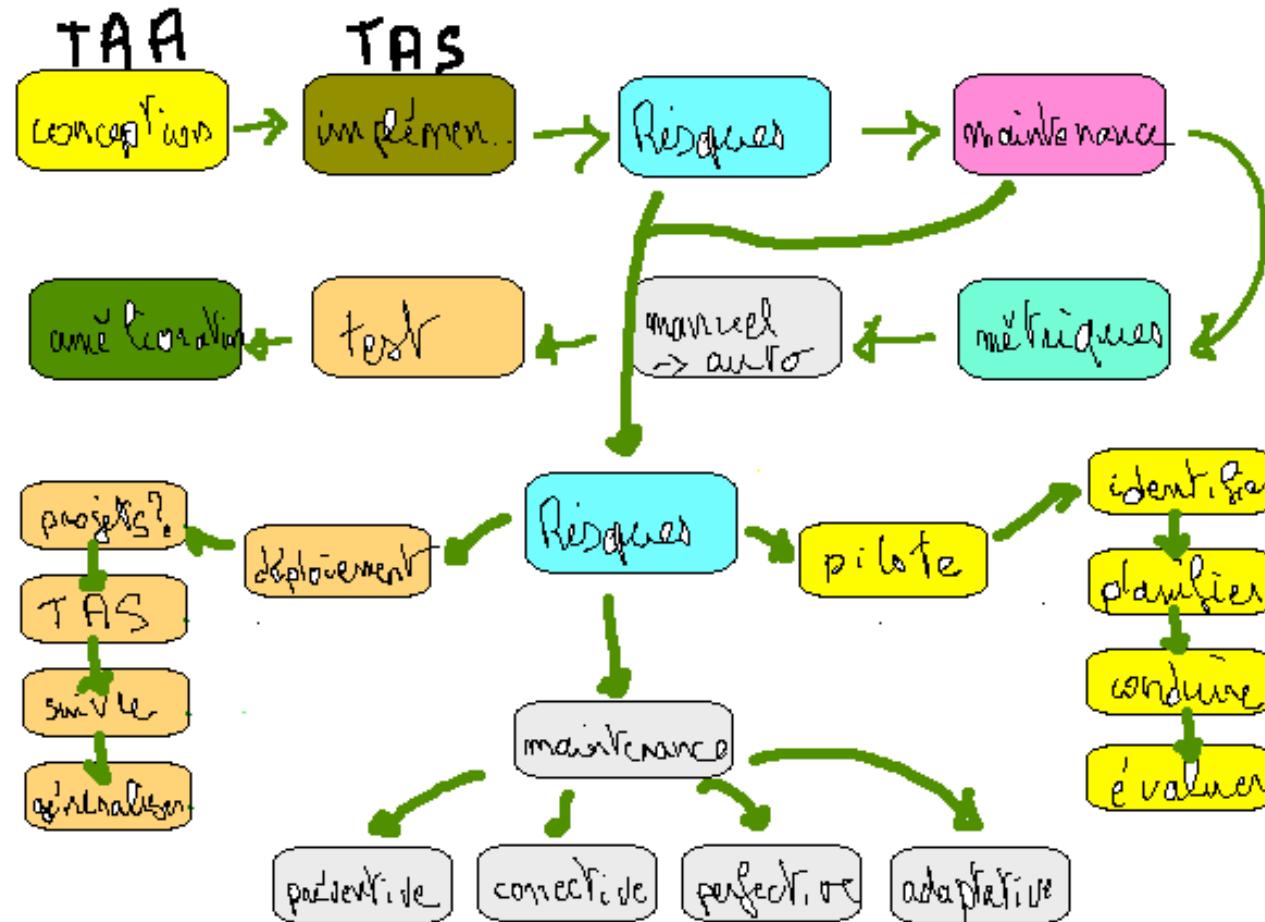


- Via la gestion de configuration**
- Via du paramétrage**

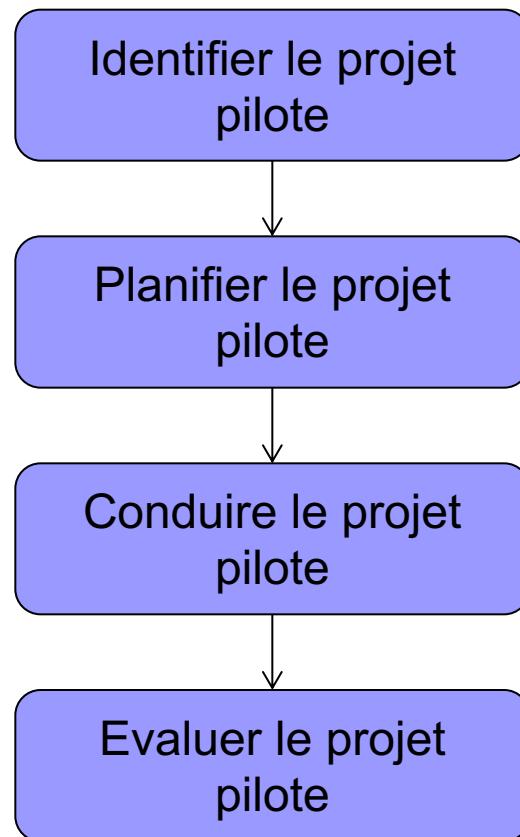
# Objectifs d'apprentissage du chapitre

- **4.1 Sélection de l'approche d'automatisation des tests et planification du déploiement**
  - ALTA-E-4.1.1 (K3) Appliquer les lignes directrices qui permettent un pilote de test et des activités de déploiement efficaces**
- **4.2 Évaluation des risques et stratégies d'atténuation (mitigation)**
  - ALTA-E-4.2.1 (K4) Analyser les risques de déploiement, identifier les problèmes techniques pouvant mener à l'échec du projet d'automatisation des tests et planifier les stratégies d'atténuation.**
- **4.3 Maintenance des tests automatisés**
  - ALTA-E-43.1 (K2) Comprendre quels facteurs supportent et affectent la maintenabilité de la TAS**

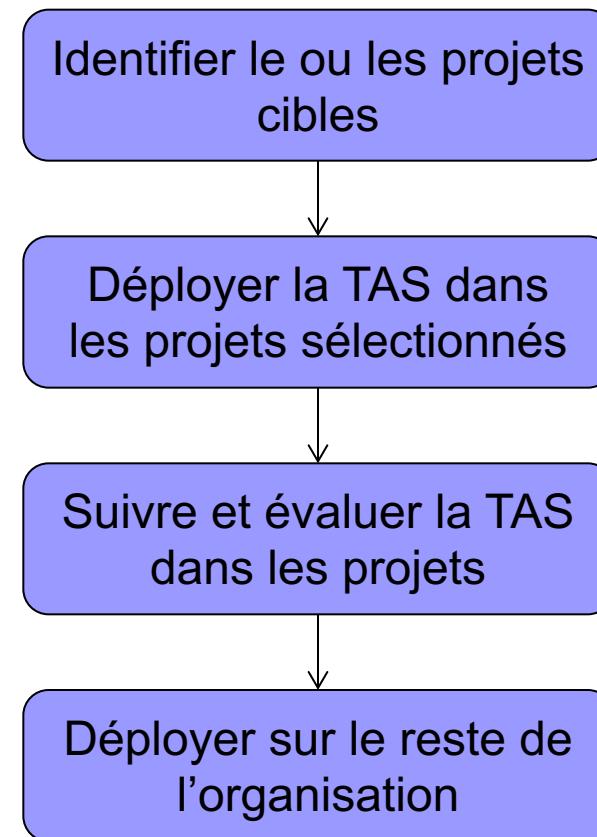
## 4-Risque et contingence lié au déploiement



### ■ Pilote:



### ■ Déploiement:



# 4-Risque et contingence lié au déploiement

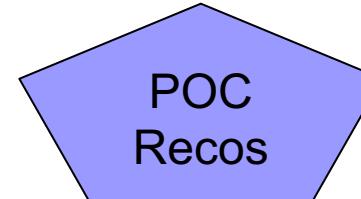


## 4.1 Sélection de l'approche d'automatisation de test et de la planification du déploiement

Que peut-on automatiser?  
Trouver un automate est-il suffisant?  
Quel retour sur investissement?  
Développement supplémentaires?  
Nouvelles ressources?  
Nouvelles compétences?



Expression de besoins



Quel projet pilote?

### ■ Le projet pilote permet de finaliser la TAS avant le déploiement

- Courbe d'apprentissage
- Adaptation de TAS aux outils, processus ou le contraire
- Interface du testeur
- Modèles, gestion de configuration, maintenance ...
- Métriques à mettre en place
- Evaluation des bénéfices
- Evaluer les compétences et prévoir les formations adéquates

### ■ Exemples:

- Sur le projet pilote, on récupère des données de la production, mais les dates des données ne sont pas adéquates →
  - ❖ prévoir dans la TAS un outil permettant de gérer le vieillissement des données.
- On veut récupérer les résultats de tests automatiques dans l'outil de gestion de test pour avoir la couverture fonctionnelle (test manuel et automatique). Les tests automatiques sont lancés via jenkins.
  - ❖ Rajouter dans l'outil d'intégration continue un plugin qui permettra de rapatrier les résultats de tests dans l'outil de gestion de test.

### ■ Critères sélection projet pilote:

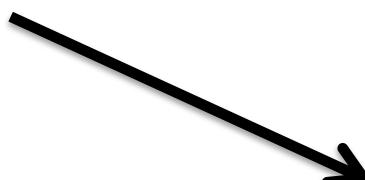
Critères	Oui/Non	Pourquoi?
Projet critique		
Projet trivial		
Projet représentatif		

## 4-Risque et contingence lié au déploiement



- 4.1 Sélection de l'approche d'automatisation de test et de la planification du déploiement
  - 4.1.1 Projet Pilote

Périmètre gérable  
Pas trop de contraintes délais  
budget  
Personnes motivées



Accord du management

En résumé ...

### ■ Planification:

- Plan, budget, ressource, reporting, jalons ...
- Temps alloué au champion sur la TAS, engagement du management
- Dans le cas d'une TAS développée en interne on devrait intégrer ses développeurs dans les activités de déploiement.

### ■ Conduire le pilote

- Evaluer les fonctionnalités du TAS attendues
- Doit-on aligner le TAS et le processus de test?

### ■ Evaluer le pilote

- Toutes les parties prenantes

## 4-Risque et contingence lié au déploiement



### 4.1 Sélection de l'approche d'automatisation de test et de la planification du déploiement

#### 4.1.1 Projet Pilote



A cartoon illustration of a conference room. A large wooden conference table is covered with various items: a laptop, a smartphone, a book, a small plant, and several colorful, crumpled pieces of paper. Several office chairs are arranged around the table, some occupied by people whose heads are obscured by large, fluffy pillows. In the background, there's a window showing a view of green trees and a small framed picture on the wall.

Quel retour sur investissement?  
Quels sont les projets  
susceptibles de bénéficier du  
meilleur retour sur  
investissement?  
Outil supplémentaire?

Liste de  
projets ...  
Ou pas

## 4-Risque et contingence lié au déploiement

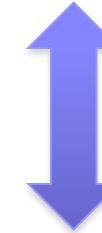
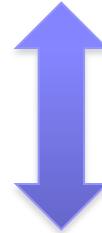
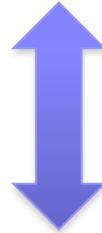
- 4.1 Sélection de l'approche d'automatisation de test et de la planification du déploiement
- 4.1.2 Déploiement

Un déploiement incrémental permettra d'identifier et résoudre au fur et à mesure de possibles goulets d'étranglement.

Projet 1

Projet 2

Projet n



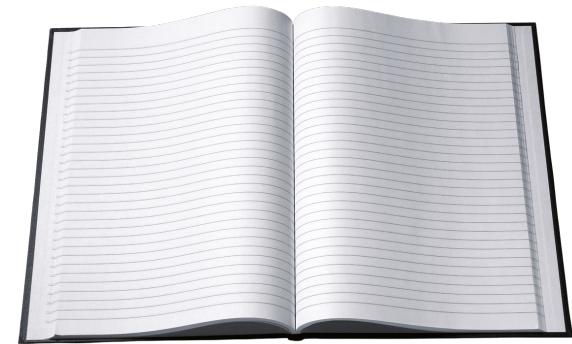
Suivi et amélioration des pratiques, alignement TAS et Process de test

## 4-Risque et contingence lié au déploiement



### 4.1 Sélection de l'approche d'automatisation de test et de la planification du déploiement

#### 4.1.2 Déploiement



Guides d'utilisation  
Objectifs  
d'automatisation  
Recommandations  
Formations

Implémentation

Mise à jour :  
• Guides d'utilisation  
• Objectifs d'automatisation  
• Recommandations  
• Formations

## ■ Les TAS sont déployées soit

- En début de projet
- Lors d'une nouvelle version (gel de code, fin d'itération)

→ Optimal en terme d'effort et ne perturbe pas le processus de test

→ Sauf en cas de composant défaillant, ou de problème avec la TAS.

Au bout de 2 ans vous avez convaincu votre manager de faire un pilote avec un outil d'automatisation en adéquation avec votre organisation. Vous devez maintenant sélectionner le projet pour le pilote:

Projet A: un projet de 2 ans en phase de recueil d'exigences, pour un nouveau site web e-commerce interne. Seul la partie panier est développer par un tiers et sera intégrer par les développeurs.

Projet B: Un projet concernant une chaîne d'approvisionnement critique, en retard de 6 mois et avec un feu rouge.

Projet C: Une mise à jour d'un projet RH de gestion du temps sur mobile et PC. Projet de 4 mois entièrement développé en interne

Projet D: le panier du projet A.

Quel est le projet qui conviendrait le mieux?

- a) Le A parce qu'il s'agit d'un gros projet et en début de développement.
- b) Le B parce que c'est une opportunité pour montrer les capacités de l'outil à remettre le projet dans les rails en terme de planning.
- c) Le C car projet court et non critique et non-trivial (mobile et PC)
- d) Le D parce que petit projet et qu'il permettra d'évaluer les capacités de l'outil sur un projet tier.

Après un pilote réussi vous avez déployé la TAS dans votre organisation. Vous menez des rétrospectives quand les projets se finissent, et vérifiez le ROI et fournissez des guides de mise en œuvre et des formations si besoin. De quel autre type d'information avez-vous besoin pour mesurer l'efficacité de la TAS?

- a) Nombre de modifications sur les processus pour utiliser la TAS
- b) Le nombre de modifications sur la TAS pour s'adapter aux processus
- c) Des informations sur comment est utilisée la TAS parmi tous les projets
- d) Le reporting fourni au management concernant les défauts trouvés dans la TAS.

- Il est nécessaire comme pour tout projet de faire un suivi des risques
- Problème Techniques Typiques
  - Difficultés d'analyse liée à trop d'abstraction
  - Tableaux de données trop grands/complexes/lourds
  - Dépendance à des composants liés à des environnements spécifiques comme des dll
- Ex: l'utilisation d'outil type MBT demande un niveau d'abstraction élevée et les testeurs rencontrent de grosses difficultés lors de la modélisation.

### ■ Risques typiques

- Ressources techniques difficiles à trouver
- Livrables du SUT qui pénalise la TAS
- Retard dans l'introduction de l'automatisation
- Retard de la mise à jour de la TAS/ Mise à jour du SUT
- Problème de capture des objets

- Une mise en place tardive de l'automatisation peut restreindre l'automatisation car certaines sorties ne sont pas observables (manque d'API).
- Les scripts de tests sont écrits en java → difficiles de trouver des testeurs fonctionnels avec des compétences en java.

### ■ Problèmes potentiels

- Migration vers un environnement différent
- Déploiement de l'environnement cible
- Nouvelle livraison du développement

→ Il faut prévoir de stratégies d'atténuation des risques

→ Avant la livraison pour test identifier les potentiels problèmes et vérifier avant livraison.

### ■ Rappel

- La TAS doit être sous contrôle de gestion**
- Ses fonctionnalités doivent être documentées**
- Une procédure de déploiement doit être documentée et versionnée**

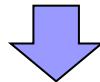
### ■ Cela facilite:

- Premier déploiement**
- Redéploiement lors de nouvelle version**
- Déploiement dans d'autres environnement**
- Maintenance**

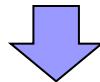
→ **l'idéal est de déployer de façon automatique, pour éviter au maximum les actions manuelles source d'erreur.**

### 1<sup>er</sup> déploiement

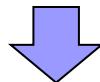
Définir l'infrastructure dans laquelle la TAS fonctionnera



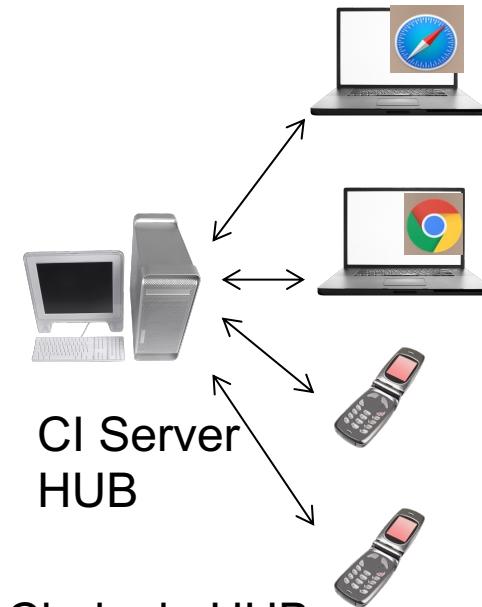
Créer l'infrastructure pour la TAS



Créer une procédure pour maintenir la TAS et son infrastructure



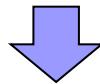
Créer une procédure pour maintenir la suite de test que la TAS exécutera



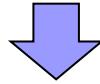
- Choix du HUB
- Mise en place de l'infra
- Procédure de choix des environnements (version des OS, version navigateurs )
- Gestion des suites de test en gestion de conf et déploiement via le serveur d'intégration continue

### Maintenance de la TAS

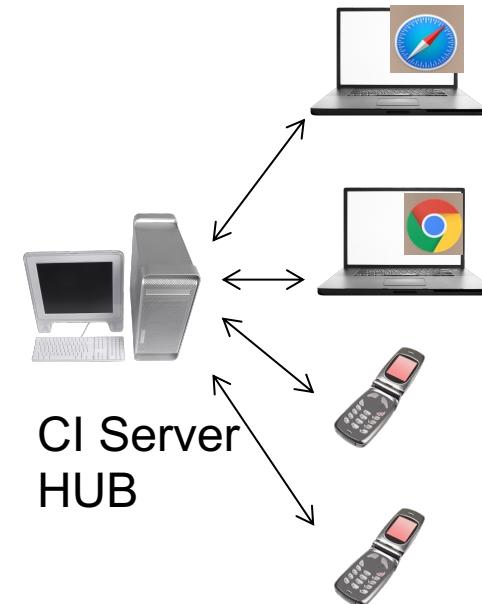
Evaluer les évolutions de la nouvelle version de la TAS



Tester les nouvelles fonctionnalités de la TAS et les régressions



Vérifier si la suite de test doit être adaptée pour la nouvelle version de la TAS



- La nouvelle TAS permet de gérer de nouvelles versions de mobiles android.
- Le type d'automatisation est modifié
- La suite de test peut-être potentiellement modifiée → vérifier dans la documentation android

### ■ Risques 1<sup>er</sup> déploiement

- Le temps d'exécution des suites de test est trop long par rapport au planning des cycles d'exécution de test.**
- Problèmes d'installation et configuration de l'environnement de test.**

### ■ Risques déploiements suivants

- La suite de test doit être modifiée**
- Les bouchons et pilotes doivent être modifiés**
- Infrastructure de test doit être modifiée**
- Mise à jour défectueuse**

**Vous avez pris en charge une TAS qui fonctionne correctement. Elle utilise le scripting basé sur les mot-clés et est correctement architecturée. L'architecte a changé d'entreprise. La TAS est déployée sur plusieurs projets et a plusieurs bibliothèques de mot-clés, organisées par projets. Elles sont maintenues par les équipes projets.**

**Etant donné ces informations, quel est le risque le plus significatif:**

- a) **Les scripts peuvent devenir obsolètes si non maintenus.**
- b) **Le niveau d'abstraction et le départ de l'architecte de test peut entraîner des difficultés à maintenir le système.**
- c) **Les nouveaux projets peuvent moins bien fonctionner que les projets courants**
- d) **Comme les bibliothèques sont maintenus par les équipes , il y a un risque sur le respect des standards de codage.**

**Vous avez pris en charge une TAS qui fonctionne correctement. Elle utilise le scripting basé sur les mot-clés et est correctement architecturée. L'architecte a changé d'entreprise. La TAS est déployée sur plusieurs projets et a plusieurs bibliothèques de mot-clés, organisées par projets. Elles sont maintenues par les équipes projets.**

**Etant donné ces informations, quel serait la meilleure stratégie de mitigation pour la TAS?**

- a) Demander à utiliser les scripting basé sur les données au lieu des mot-clés pour réduire le niveau d'abstraction
- b) Définir des pratiques de codage à suivre par tous les utilisateurs de la TAS.
- c) Documenter l'architecture et l'approche d'abstraction de la TAS pour faciliter la maintenance
- d) Demander aux équipes d'arrêter de maintenir les scripts et créer une équipe dédiée pour l'écriture des scripts

**La maintenance de la TAS et des tests automatiques doit être prise en compte dès le départ**

**→ c'est une des principales causes d'échec des projets d'automatisation**

**Une solution d'automatisation doit posséder les caractéristiques suivantes:**

- Modulaires**
- Evolutive**
- Fiable**
- Compréhensible**
- vérifiable**

## Maintenance préventive

- Evolution de la TAS en vu du support d'autres types de test, différentes interfaces, différentes versions ou nouveau SUT

## Maintenance corrective

- Correction de bug suite à des tests de maintenance.

## Maintenance perfective

- Amélioration de la performance, fiabilité, optimisation.

## Maintenance adaptative

- Support de nouveaux système d'exploitation, navigateurs web, gestionnaire de base de données

## Exercices, quel type de maintenance:

- La suite de test prend trop de temps, la TAS est modifiée de façon à exécuter plus rapidement les suites de test.**
- La TAS supporte les API de type SOAP, la TAS est modifiée de façon à supporter les API de type Rest.**
- La TAS permet d'accéder à une base de données MySQL elle évolue pour supporter ORACLE également.**
- La TAS est modifiée suite à la détection d'un problème identifié pendant l'exécution des tests sur le SUT.**

#### ■ Il faut tenir compte:

- Taille et complexité de la TAS
- Importance du changement
- Risque

#### ■ Faire une analyse d'impact et en fonction de l'impact planifier les modifications de façon progressive et les tester au fur et à mesure.

## ■ les bonnes pratiques nécessaire à la maintenance de la TAS

1. Procédure de déploiement, utilisation
2. Gestion de configuration des composants
3. Dépendances avec les outils tiers
4. Modulaire
5. Composant ou env de test remplaçable
6. TAS s'exécuter isolément de l'env de dev
7. La TAS doit séparer les scripts du TAF

→ Dans votre organisation qu'avez vous mis en place pour améliorer la maintenance?

### ■ Considérations pour améliorer la maintenance

#### □ Maintenance des composants et bibliothèques tierces

- ❖ Documenté et en gestion de configuration
- ❖ Plan en cas de problème, support
- ❖ Licence, qui peut modifier et comment
- ❖ Support et documentation sur nouvelles versions

#### □ Standard de nommage et conventions

- ❖ Facile à lire et à comprendre
- ❖ Facile d'intégrer les nouveaux entrants
- ❖ Variables, mots-clés, scénarios, scripts
- ❖ Les conventions et standards doivent convenus et documentés

## ■ Considérations pour améliorer la maintenance

### □ Documentation

- ❖ Qui écrit, qui met à jour
- ❖ Le code peut être auto-documenté (ex java-doc) mais qqn doit documenter la conception, architecture, dépendances et procédure de déploiement.
- ❖ La documentation doit se faire lors du processus de développement

### □ Formation

- ❖ La documentation peut servir de matériel de formation
- ❖ Spec fonctionnelles + conception architecture, doc de maintenance et déploiement, manuel utilisateur, exemple et exercice, trucs et astuces
- ❖ Le matériel de formation doit être également maintenu.

## Pourquoi est-il important d'avoir des standards et convention de nommage pour les suites de test et la TAS:

- a) Cela facilite l'analyse et la maintenance
- b) Cela assure la non duplication des noms de suite de test
- c) Cela permet de faciliter le check in et check out pour la gestion de source
- d) Cela facilite la création de convention de nommage propre à chaque équipe.

## 5.1 Sélection des métriques de la TAS

## 5.2 Mise en œuvre des métriques

## 5.3 Journalisation de la TAS et du SUT

## 5.4 Automatisation du reporting des tests

# Objectifs d'apprentissage du chapitre

## 5. Métriques et reporting sur l'automatisation des tests

### 5.1 Sélection des Métriques sur la TAS

**ALTA-E-5.1.1 (K2) Classer les métriques qui peuvent être utilisées pour suivre la stratégie d'automatisation des tests et son efficacité**

### 5.2 Mise en œuvre des métriques

**ALTA-E-5.2.1 (K3) Implémenter des méthodes de collecte de métriques pour prendre en charge les exigences techniques et de gestion. Expliquez comment la mesure de l'automatisation des tests peut être implémentée**

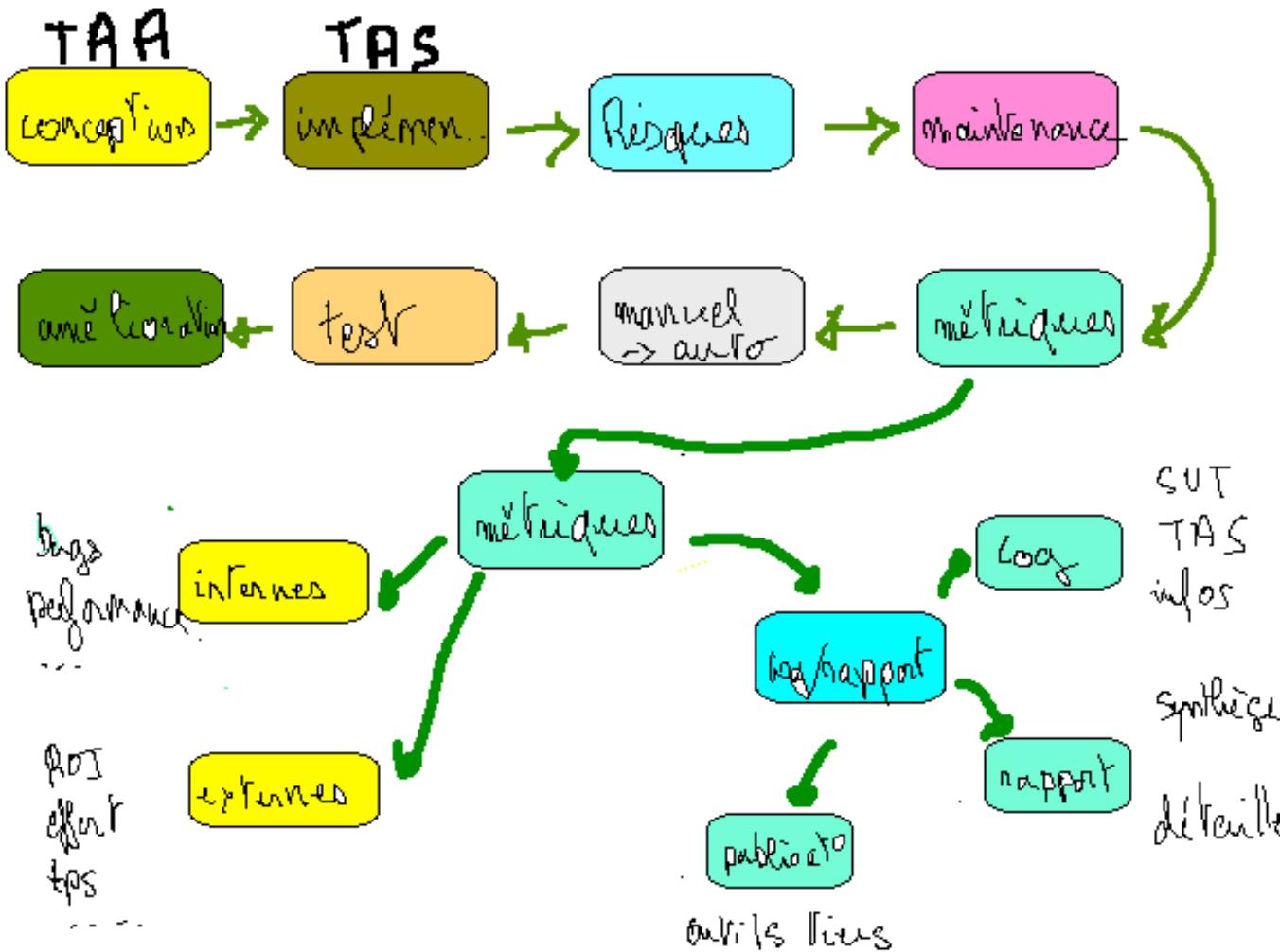
### 5.3 Journalisation de la TAS et du SUT

**ALTA-E-5.3.1 (K4) Analyser les données de journalisation des tests pour la TAS et le SUT**

### 5.4 Automatisation du reporting des tests

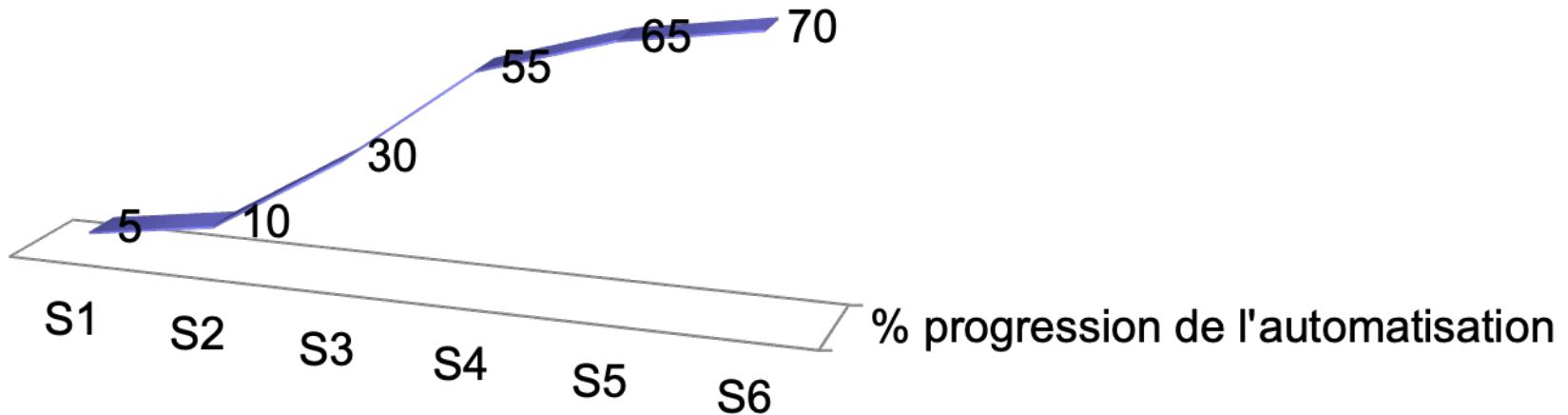
**ALTA-E-5.4.1 (K2) Expliquer comment un rapport d'exécution est construit et publié**

# 5-Métriques et reporting sur l'automatisation des tests



- Mesurer et suivre permet d'identifier le respect des plans (par exemple le % de tests réellement automatisés par rapport à l'objectif initial) mais surtout les points à améliorer:
  - Le ROI effectif.
  - % de tests automatisés.
    - Un objectif non atteint peut être significatif d'un manque de formation.
  - Le temps moyen d'écriture et de mise au point d'un test automatique.
    - Utile pour la planification.
  - L'effort d'analyse des incidents pendant les tests automatiques.
    - Les logs de test ne sont pas suffisants, les prérequis ne sont pas vérifiés.
  - L'effort de maintenance des tests automatiques.
    - De nouvelles pratiques sont peut-être à mettre en place pour améliorer la maintenabilité.
  - Le temps d'exécution des tests automatiques.
    - Des temps élevés peuvent montrer une utilisation abusive des temporisations.
  - Le nombre de tests automatiques comportant eux mêmes des défauts.
    - Les tests automatiques ne sont pas suffisamment testés.

### % progression de l'automatisation



## 5.1 Sélection des mesures

### ■ Interne

- Efficacité et efficience de la TAS

### ■ Externe

- Mesure de l'impact de la TAS sur les autres activités

### ■ Métriques internes de la TAS

- Métriques liées aux outils de scripts
- Densité de code défectueux d'automatisation
- Vitesse et efficacité des composants TAS

### ■ Externes

- Avantages de l'automatisation**
- Effort de construction des tests automatisés**
- Effort d'analyse des incidents de tests automatisés**
- Effort de maintenance des tests automatisés**
- Ratio de pannes/défaux**
- Temps d'exécution des tests automatisés**
- Nombre de cas de tests automatisés**
- Nombre de résultats réussis ou en échecs**
- Nombre de faux-échecs et de faux succès**
- Couverture de code**

## 5.1 Sélection des mesures

### ■ Avantages de l'automatisation

- Permet d'évaluer les avantages: réduction des temps d'exécution de test, fiabilité, répétabilité et fréquence des tests.
- Métriques possibles
  - ❖ Nombre d'heures d'effort de test manuel économisées
  - ❖ Réduction du temps d'exécution des tests de régression
  - ❖ Nombre de cycles supplémentaires d'exécution de test réalisés
  - ❖ Nombre ou pourcentage de tests supplémentaires exécutés
  - ❖ Pourcentage de cas de test automatisés par rapport à l'ensemble des cas de test
  - ❖ Augmentation de la couverture (exigences, fonctionnalités, structurelle)
  - ❖ Nombre de défauts trouvés plus tôt grâce à la TAS
  - ❖ Nombre de défauts constatés grâce à la TAS qui n'auraient pas été trouvés par des tests manuels
- Avec la réduction d'exécution il est possible de
  - ❖ Autres tests type exploratoire
  - ❖ De nouveaux bugs

## 5.1 Sélection des mesures

### ■ Effort de construction des tests automatisés

□ Coût non négligeable

□ Dépend

- ❖ Du test en lui-même (complexité)
- ❖ L'approche de script
- ❖ Courbe d'apprentissage de l'outil de script
- ❖ Compétence de l'ingénieur d'automatisation

□ Coût

→ Coût de fabrication automatisation moyen

→ Coût de fabrication moyen pour une série de test

→ En terme d'effort (effort de test manuel équivalent, EMTE)

→ Par exemple l'effort d'automatisation peut être 2 fois l'EMTE

## 5.1 Sélection des mesures

### ■ Effort pour analyser les incidents du SUT

Gérer les logs.



Pourquoi le test est-il KO?  
Je vais le rejouer en  
manuel.

#### → Expression de l'effort

- Moyenne par cas de test en échec
- Facteur EMTE

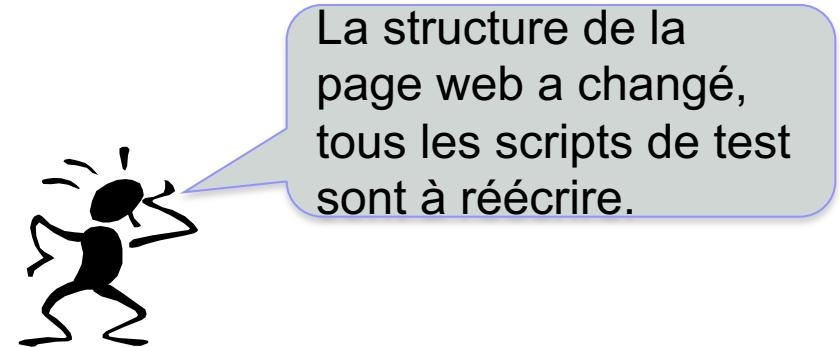
#### → Importance des logs et de la conception

- L'enregistrement du SUT et l'enregistrement de la TAS doivent être synchronisés
- Le TAS doit enregistrer le comportement attendu et réel
- Le TAS doit enregistrer les actions à effectuer

## 5.1 Sélection des mesures

### ■ Effort pour maintenir les tests automatisés

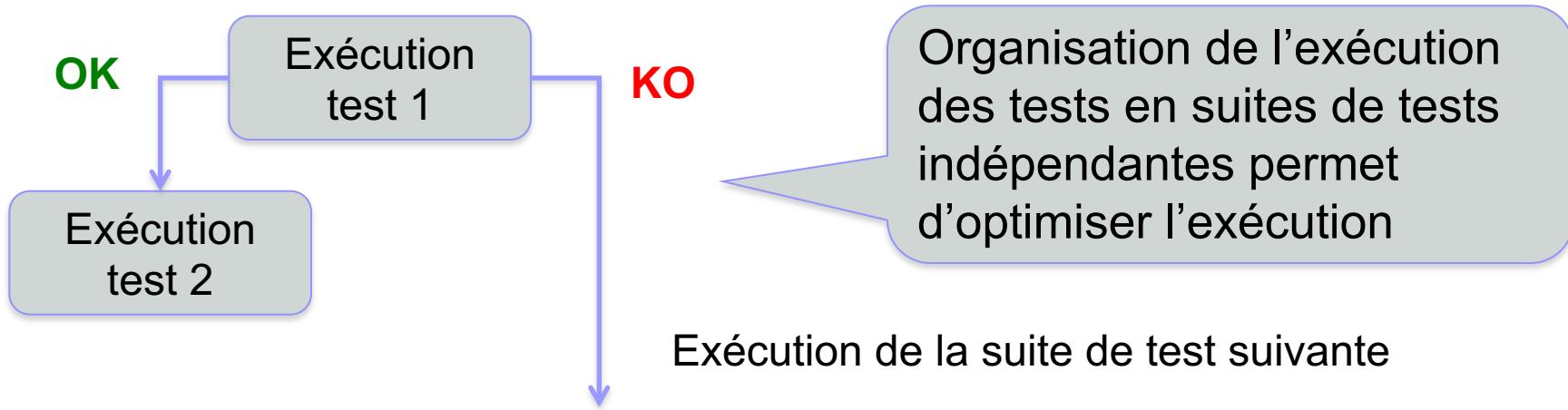
```
test_demo
[Tags] regression
Open Browser https://demo.prestashop.com/#/en/front gc
Set Selenium Implicit Wait 30
Select Frame framelive
Input Text name=s MUG
Press Keys name=s ENTER
Element Text Should Be xpath=//*[@id="js-product-list-top"]/div[1]/p
```



- Il faut veiller à ce que l'effort pour la maintenance soit moins important que les avantages apportés par l'automatisation
- Expression de l'effort
  - Ensemble des tests mis à jour
  - Facteur EMTE
  - Pourcentage de test nécessitant une maintenance
- Le coût Influence les évolutions
- Le coût est envisagé avec les évolution du SUT

## 5.1 Sélection des mesures

### ■ Ratio de pannes/défaux



- Mesurer le nombre de tests automatisés qui échouent pour un défaut donné indique qu'il y a un problème.
  - Modifier la conception des tests automatisés
  - Modifier leur sélection

## 5.1 Sélection des mesures

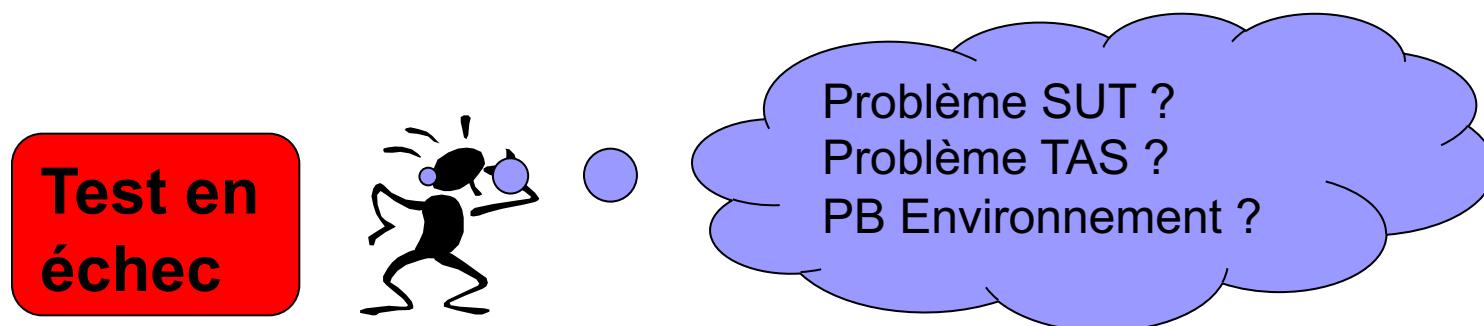
### ■ Temps d'exécution des tests automatisés

- Il peut augmenter au fur et à mesure du nombre de test automatisé.
- Mettre en place des mesures pour optimiser les temps d'exécution

### ■ Nombre de cas de tests automatisés

- Avancement du projet d'automatisation
- Attention n'indique pas la progression de la couverture

### ■ Nombre de résultats en échec ou en succès



## 5.1 Sélection des mesures

### ■ Nombre de résultats de faux-échecs et de faux succès

Gérer les logs.



Pourquoi le test est-il KO?  
Je vais le rejouer en  
manuel.

#### → Analyse de échecs

- Temps perdu en cas de problème sur TAS ou environnement
- Mesurer pour diminuer
- Peut-être liés à des instabilités (temps réel)

#### → Tendance à trop faire confiance aux tests automatiques

- Potentiels bugs non découverts
- Tester ses tests pour détecter des oracles invalides

## 5.1 Sélection des mesures

### Couverture de code

Coverage Report - All Packages

Package /	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	54	34 % <span style="background-color: red; color: green;">428/1255</span>	34 % <span style="background-color: red; color: green;">135/390</span>	2,398
org.masukomi.prefs	4	0 % <span style="background-color: red; color: green;">0/285</span>	0 % <span style="background-color: red; color: green;">0/78</span>	2,768
org.masukomi.todo	5	67 % <span style="background-color: red; color: green;">328/485</span>	56 % <span style="background-color: red; color: green;">123/216</span>	2,838
org.masukomi.todo.gui	45	20 % <span style="background-color: red; color: green;">100/485</span>	12 % <span style="background-color: red; color: green;">12/96</span>	1,778

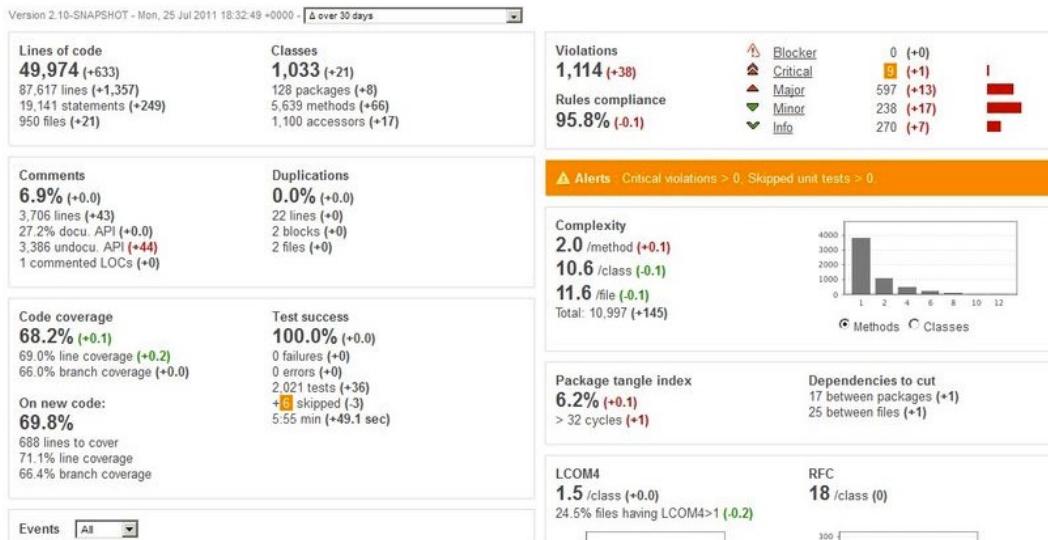
Report generated by [Cobertura](#) 1.9.4.1 on 01/02/12 10:13.

```
500                                }
501                                }
502    1      } else if (amount < 0) {
503    1      if (getCreativeFeasability() > 1) {
504    1          if ((getCreativeFeasability() + amount) < 1) {
505    0          int difference = getCreativeFeasability() - 1;
506    0          setCreativeFeasability(1);
507
508    0      return difference;
509
510    1  } else {
511
512
513    1      setCreativeFeasability(getCreativeFeasability()
514                                + amount);
515
516      return 0 - amount;
517
518    }
```

- Couverture de code 100% impossible
- Permet d'évaluer le risque de déploiement
- Permet également d'identifier des zones non testées

## 5.1 Sélection des mesures

### ■ Métriques d'outils de scripting



→ Semblable aux métriques utilisées pour le code source

- Complexité Cyclomatique
- Commentaires
- Non conformité ...

## 5.1 Sélection des mesures

### ■ Autre exemple de métriques d'outils de Scripting (C)

#### Code Complexity Report

Source file: *./chess.h*

Source file: *./chess.c*

Function name	Cyclomatic complexity (15)	LOC (1000000)	Token count	Parameter count (100)
pieceEchiquier	5	19	60	2
getPositionHorizontale	1	4	8	0
getPositionVerticale	1	4	8	0
deplacementEstValide	3	8	40	2

Source file: *./main.c*

Function name	Cyclomatic complexity (15)	LOC (1000000)	Token count	Parameter count (100)
testCreation1	1	4	13	1
main	4	20	95	0

LOC : Ligne of Code

## 5.1 Sélection des mesures

- **Densité de défaut d'automatisation**
  - Gérée comme le SUT
  - Nécessite gestion de configuration et gestionnaire d'anomalie
- **Vitesse et efficacité des composants de la TAS**
  - Si cette mesure se dégrade cela peut être du à un problème dans le SUT
  - Pour cela la TAS doit être aussi efficace que le SUT en terme de performance
- **Indicateurs de tendance**
  - Général aux mesures
  - Permet de détecter des problèmes de fond.

## 5.2 Mise en œuvre de la mesure

### ■ Abstraction = meilleure mise en œuvre des améliorations

- Ex: Si on améliore les logs (module commun) cela s'applique à tous les tests.

### ■ Caractéristiques permettant les mesures

- Les langages eux-mêmes (ex: outils de qualimétrie)
- Le reporting
  - ❖ Comparaison avec résultats précédents
  - ❖ Informations supplémentaires (copies écran)

- **KEYWORD** SeleniumLibrary. **Input Text** id=pwd, giuhi

**Documentation:** Types the given `text` into text field identified by `locator`.

**Start / End / Elapsed:** 20200307 14:06:41.411 / 20200307 14:06:41.836 / 00:00:00.425

- **KEYWORD** SeleniumLibrary. **Capture Page Screenshot**

**Documentation:** Takes screenshot of the current page and embeds it into log file.

**Start / End / Elapsed:** 20200307 14:06:41.420 / 20200307 14:06:41.832 / 00:00:00.412

14:06:41.832 INFO



## 5.2 Mise en œuvre de la mesure

### ■ Intégration avec d'autres outils tiers

#### □ Nécessite un format « universel »

- ❖ Xml
- ❖ Excel
- ❖ ...

### ■ Exemple

#### □ Les résultats des tests exécutés avec google test peuvent générés au format « xml junit » on peut alors

#### Test Result : (root)

0 failures ( $\pm 0$ )

8 tests ( $\pm 0$ )

Took 0 ms.

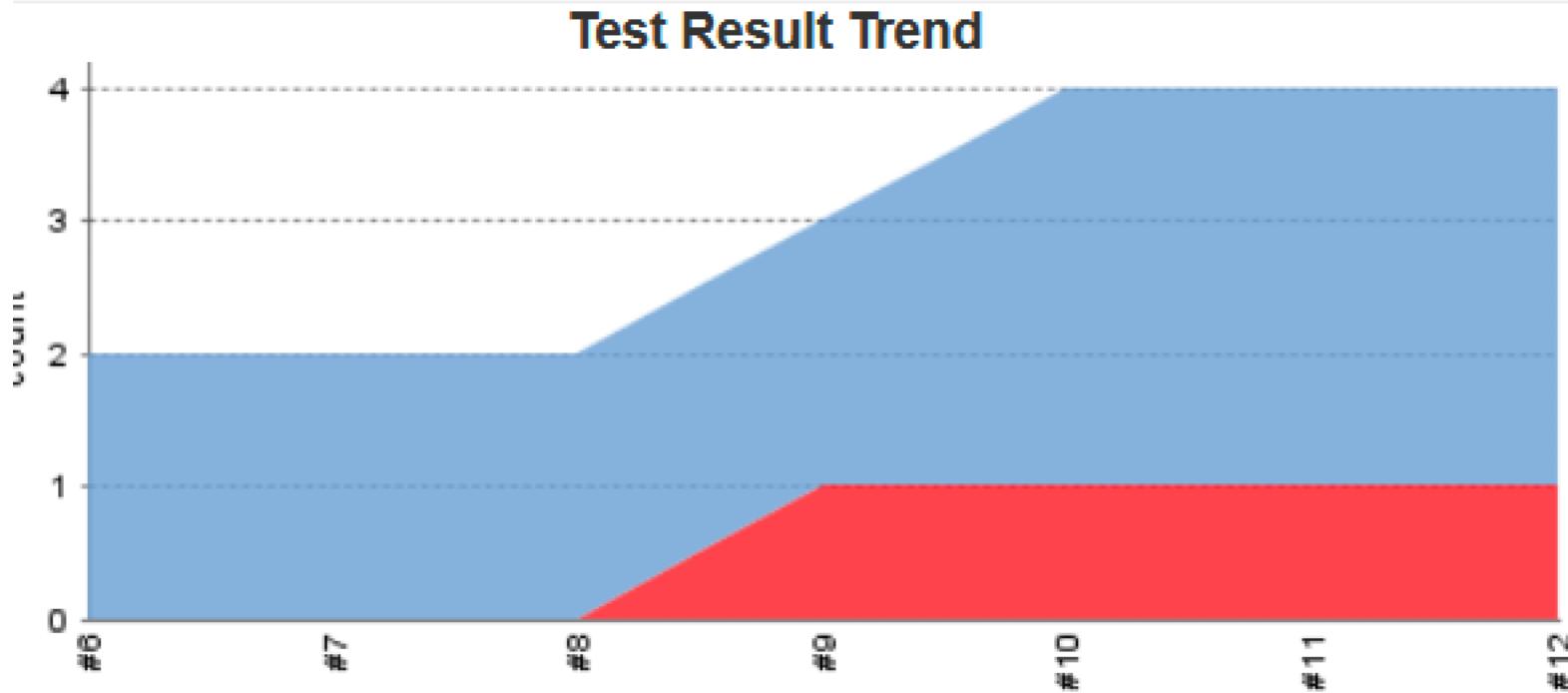
 [add description](#)

#### All Tests

Class	Duration	Fail (diff)	Skip (diff)	Pass (diff)	Total (diff)
<a href="#">AnimalVariations/AnimalTest</a>	1 ms	0	0	4	4
<a href="#">CalculTest</a>	0 ms	0	0	2	2
<a href="#">toto/isFraisParamTest</a>	0 ms	0	0	2	2

### ■ Visualisation des résultats

- Avancement sous forme de tableaux de bord



## 5.3 Enregistrement de la TAS et du SUT

### ■ Logs/enregistrements de test

- Servent à analyser les problèmes

#### Statut de la suite de test

- **SUITE** Test 00:00:03.349  
**Full Name:** Test  
**Documentation:** demo for appium library  
**Source:** /Users/dominiquemereaux/code/RobotFramework/Web/test.robot  
**Start / End / Elapsed:** 20200307 14:06:38.488 / 20200307 14:06:41.837 / 00:00:03.349  
**Status:** 1 critical test, 0 passed, 1 failed  
1 test total, 0 passed, 1 failed

#### Statut du test

Nom du test, date,  
début et fin

- **TEST** test\_demo 00:00:03.194  
**Full Name:** Test.test\_demo  
**Tags:** demo, regression  
**Start / End / Elapsed:** 20200307 14:06:38.642 / 20200307 14:06:41.836 / 00:00:03.194  
**Status:** FAIL (critical)  
**Message:** Element with locator 'id=pwd' not found.  
+ **KEYWORD** SeleniumLibrary.Open Browser http://www.qualifiez.fr/examples/Selenium/, gc 00:00:02.570  
+ **KEYWORD** SeleniumLibrary.Input Text id=login, czdce 00:00:00.196  
- **KEYWORD** SeleniumLibrary.Input Text id=pwd, giuhi 00:00:00.425  
**Documentation:** Types the given text into text field identified by locator.  
**Start / End / Elapsed:** 20200307 14:06:41.411 / 20200307 14:06:41.836 / 00:00:00.425  
+ **KEYWORD** SeleniumLibrary.Capture Page Screenshot 00:00:00.412  
14:06:41.411 INFO Typing text 'giuhi' into text field 'id=pwd'.  
14:06:41.836 FAIL Element with locator 'id=pwd' not found.

Etapes  
du test.

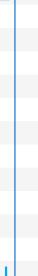
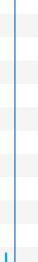
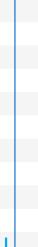
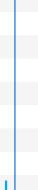
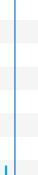
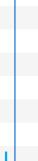
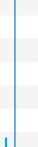
## 5.3 Enregistrement de la TAS et du SUT

### ■ Autres informations utiles

- Infos dynamiques → utilisation d'outil tiers
- Compteur de réussite de test (fiabilité/stress)
- Enregistrement des choix aléatoires
- Détails des actions pour rejet
- Capture écran, vidéo
- Informations supplémentaires utiles en cas d'échec
- Utilisation de la couleur

### ■ Côté SUT

- Date, message erreur, trace
- Log réseau, échanges d'informations
- Configuration du système, OS

Name	Status	Type	Initiator	Size	Time	Waterfall
search?q=toto&oq=toto&aqs=chrome.....	200	document	Other	300 KB	1.07 s	
prompt.js	200	script	prompt-injecter.js:22	9.5 KB	20 ms	
runScript.js	200	script	runScript-injecter.j...	1.4 KB	105 ms	
international-womens-day-2020-67536...	200	png	/search?q=toto&o...	3.0 KB	9 ms	
data:image/gif;base...	200	gif	/search?q=toto&o...	(memory ...)	0 ms	
photo.jpg	200	png	/search?q=toto&o...	(disk cache)	2 ms	
googlemeric_color_24dp.png	200	png	/search?q=toto&o...	(disk cache)	3 ms	
desktop_searchbox_sprites302_hr.webp	200	webp	/search?q=toto&o...	(disk cache)	3 ms	
data:image/svg+xml;...	200	svg+xml	VM15 search:128	(memory ...)	0 ms	
data:image/png;base...	200	png	/search?q=toto&o...	(memory ...)	0 ms	
data:image/png;base...	200	png	/search?q=toto&o...	(memory ...)	0 ms	
data:image/png;base...	200	png	/search?q=toto&o...	(memory ...)	0 ms	
data:image/ipeg;bas...	200	ipeg	VM16 search:133	(memory ...)	0 ms	

→ l'horodatage côté SUT et TAS permet de rapprocher les logs.

## 5.4 Reporting de l'automatisation des tests

→ Il est nécessaire de fournir un reporting synthétique et détaillé des tests.

Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
TP	33	25	8	00:01:58	
TP.Appium	3	0	3	00:00:57	
TP.Appium.Dice	1	0	1	00:00:19	
TP.Appium.Untitled Text	1	0	1	00:00:18	
TP.Appium.Web	1	0	1	00:00:20	
TP.CDIPython	1	0	1	00:00:00	
TP.CDIPython.calculFrais	1	0	1	00:00:00	
TP.Chaine	3	3	0	00:00:00	
TP.Chaine.suiteC	3	3	0	00:00:00	
TP.DB	1	0	1	00:00:00	
TP.de.TP BaseDonnee	1	0	1	00:00:00	
TP.Dialog	3	3	0	00:00:12	
TP.Dialog.Chap2	3	3	0	00:00:12	
TP.LibrairiePython	5	4	1	00:00:00	
TP.LibrairiePython.Boucle	4	4	0	00:00:00	
TP.LibrairiePython.TPPython	1	0	1	00:00:00	
TP.Prestashop	2	1	1	00:00:46	
TP.Prestashop.Recherche	2	1	1	00:00:46	
TP.REST	2	2	0	00:00:01	
TP.REST.TP REST	2	2	0	00:00:01	
TP.Screenshot	1	1	0	00:00:00	
TP.Screenshot.Suite	1	1	0	00:00:00	
TP.TPBase	12	11	1	00:00:00	
TP.TPBase.Chap1	10	9	1	00:00:00	
TP.TPBase.TeplateDeTEst	2	2	0	00:00:00	

## 5.4 Reporting de l'automatisation des tests

### ■ Reporting détaillé

#### Test Details

LOG

Totals Tags Suites Search

Name:	TP.Prestashop.Recherche						
Status:	2 critical test, 1 passed, <b>1 failed</b> 2 test total, 1 passed, <b>1 failed</b>						
Start / End Time:	20200308 11:37:32.963 / 20200308 11:38:19.152						
Elapsed Time:	00:00:46.189						
Log File:	<a href="#">log.html#s1-s7-s1</a>						
Name	Documentation	Tags	Crit.	Status	Message	Elapsed	Start / End
TP.Prestashop.Recherche.Faire une recherche MUG			yes	FAIL	The text of element 'xpath=//div[@id='js-product-list-top']//div/p' should have been 'Il y a 5 produit.' but it was 'Il y a 5 produits.'	00:00:34.917	20200308 11:37:33.229 20200308 11:38:08.146
TP.Prestashop.Recherche.lien femmes			yes	PASS		00:00:11.006	20200308 11:38:08.146 20200308 11:38:19.152

### ■ Contenu des rapports

- **Synthèse + possibilité d'avoir une vue détaillée (par ex via des liens)**
- **Liste des tests en échec avec les raisons de l'échec**
- **Des échecs peuvent être liés au TAS**

### ■ Publication des rapports

- **Publication via web, liste de diffusion ou outil de test (ex ALM)**
- **Historique des rapports nécessaire pour comparaison avec les versions précédentes de façon à pouvoir faire des analyses (pb régression, tendance ...)**

- % test auto faible
- Tps écriture élevé
- Analyse pb difficile
- Effort de maintenance important
- Tps élevé d'exécution
- Test auto avec défauts
- Utilisation de tempo
- Manque de formation
- Manque de tests
- Manque de simplicité
- Manque d'information dans les logs
- Nouvelles pratiques

On vous a demandé d'implémenter l'automatisation sur un projet qui dérape. Après analyse vous découvrez que les testeurs manuels n'arrivent pas à tester les nouvelles fonctionnalités parce que les tests de régression prennent 75% de leur temps. En final les nouvelles fonctionnalités sont livrées avec beaucoup de défauts.

Etant donné ces informations quelle métrique devriez vous tracer pour démontrer la valeur des tests automatiques?

- a) Le pourcentage du code couvert par l'automatisation.
- b) L'effort de test équivalent si test manuel
- c) Nombre de défauts trouvés par l'automatisation
- d) Pourcentage de builds acceptés/rejetés par les tests automatiques.

**Vous avez terminé l'automatisation des tests pour un projet financier. Au cours de l'effort d'automatisation, un temps important a été consacré à l'automatisation de la validation des résultats des rapports. Cela a été particulièrement difficile car les développeurs ont utilisé un nouvel outil pour créer les rapports et les tableaux de bord. Un codage important a été nécessaire**

**pour permettre à l'automatisation de reconnaître les champs dans les tableaux de rapport et de vérifier l'apparence du tableau de bord.**

**Les développeurs vous ont dit qu'ils prévoient de retravailler les modules de rapports afin de modifier la façon dont les rapports sont présentés et de changer l'affichage des tableaux de bord.**

**Votre direction veut savoir combien de temps cela prendra et si cela doit être fait ou si cette partie de l'application ne doit pas être testée par l'automatisation. Vous estimez que le temps nécessaire sera de trois jours par script de test et qu'il y a 20 scripts de test impliqués dans ce test.**

**De quelles autres informations avez-vous besoin pour savoir si vous devez effectuer ces changements ou non ?**

- a. Le ratio entre les échecs et les défauts**
- b. Le nombre de fois que les tests seront exécutés au cours de la durée de vie du logiciel de reporting.**
- c. La comparaison entre les 60 jours et l'EMTE**
- d. Le temps nécessaire à l'analyse des défaillances du SUT**

Si vous mesurez le nombre de lignes de code et la complexité cyclomatique du code d'automatisation des tests, quel type de mesure recueillez-vous ?

- a) Métriques des outils de scripting
- b) La densité de défaut sur le code de test
- c) Tendances
- d) Nombre de faux négatifs

Si vous mesurer la fréquence avec laquelle le code de test automatique rapporte des défauts qui ne sont pas réellement des défauts, que mesurez vous?

- a) Métriques des outils de scripting
- b) La densité de défaut sur le code de test
- c) Tendances
- d) Nombre de faux positifs

#### Quel est l'objectif du rapport d'exécution de test

- a) Fournir des détails de l'exécution des test au niveau cas de test pour la gestion de projet
- b) Fournir des informations sur les problèmes pour que le TAE corrige les problèmes
- c) Fournir un résumé et des informations sur les tendances pour les parties prenantes
- d) Fournir un retour au développeur de SUT sur les données utilisées dans les tests

## **6.1 Critères d'automatisation**

## **6.2 Identifier les étapes nécessaires pour automatiser les tests de régression**

## **6.3 Facteurs à considérer pour l'automatisation des tests de nouvelles fonctionnalités**

## **6.4 Facteurs à considérer pour l'automatisation des re-tests et des défauts**

# Objectifs d'apprentissage du chapitre

## 6. Transition du test manuel vers un environnement automatisé

### 6.1 Critères d'automatisation

ALTA-E-6.1.1 (K3) Analyser les facteurs pertinents pour déterminer les critères appropriés à l'automatisation des tests

ALTA-E-6.1.2 (K2) Comprendre les facteurs de transition entre le test manuel et l'automatisation

### 6.2 Identifier les étapes nécessaires pour automatiser les tests de régression

ALTA-E-6.2.1 (K2) Expliquer les étapes nécessaires pour automatiser les tests de régression

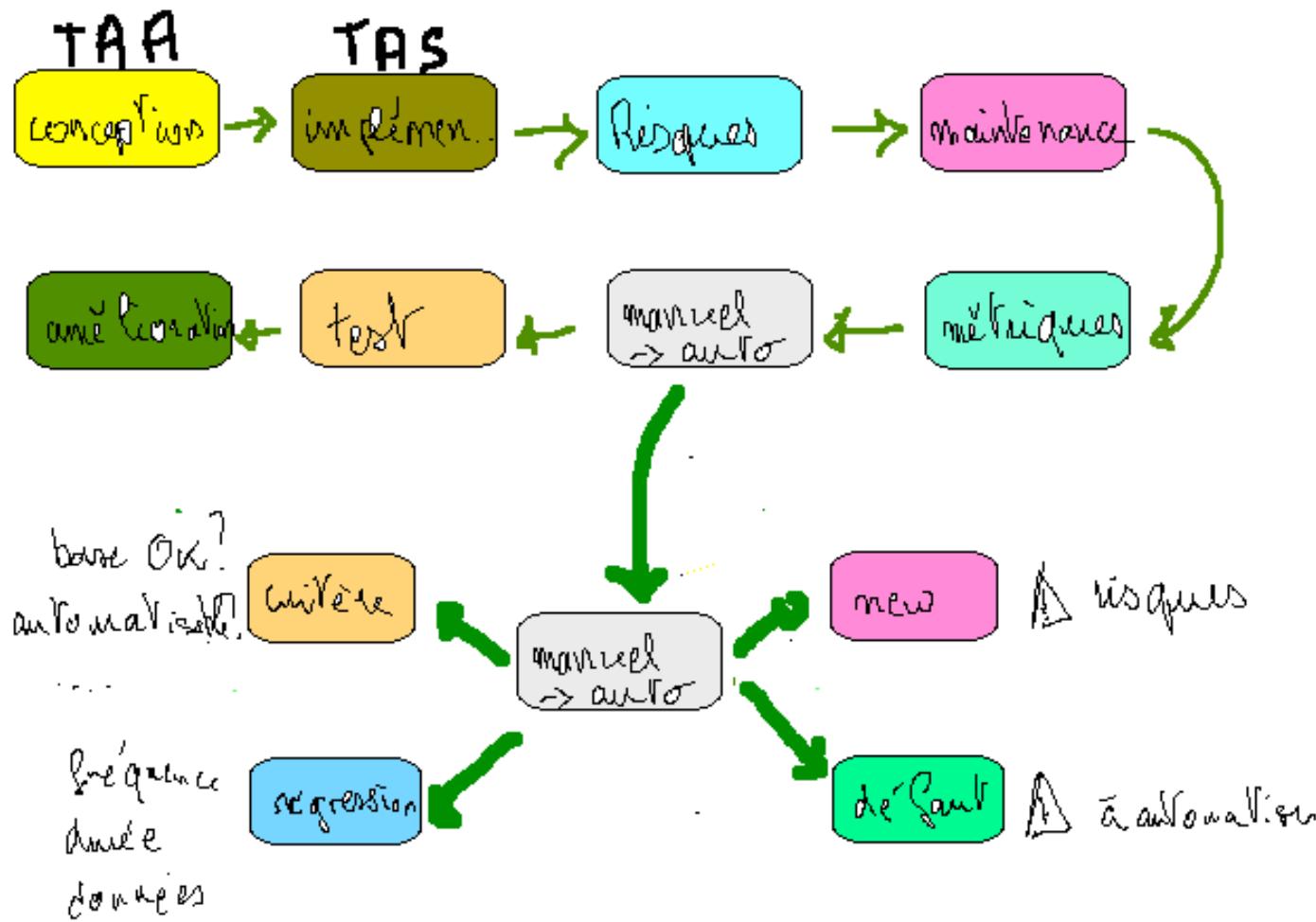
### 6.3 Facteurs à considérer pour l'automatisation des tests d'une nouvelle fonctionnalité

ALTA-E-6.3.1 (K2) Expliquer les facteurs à considérer pour automatiser les tests d'une nouvelle fonctionnalité

### 6.4 Facteurs à considérer pour l'automatisation des re-tests des défauts

ALTA-E-6.4.1 (K2) Expliquer les facteurs à considérer pour l'automatisation des re-tests des défauts

## 6- Transition du test manuel vers un environnement automatisé



### 6.1 Critères d'automatisation

**Il s'agit d'analyser l'existant et de déterminer au préalable si:**

**■ Les spécifications de test sont:**

- Complètes : prérequis, données, jeux d'essais, étapes avec contrôles.**
- Suffisamment élaborées pour servir de base à l'automatisation.**

**■ Le framework de test existant est:**

- Souple et compatible avec les évolutions du SUT.**
- Maintenable.**

**■ Les tests automatisés sont**

- Maintenables, évolutifs.**

- L'automatisation nécessite des pré-requis et en particulier l'existence de tests manuels.
  - Sont-ils bien écrits pour être automatisés?
    - ❖ Objectif, étapes, jeux de données, vérifications automatisables
    - ❖ Réécriture peut être nécessaire
  - L'idéal est de penser automatisation avant de spécifier les tests.
- Certains tests ne sont fiables qu'automatisés
  - Calcul, robustesse etc ...

## ■ Exercice: définir des points de vérification

- Objectif : vérifier la pertinence de la recherche
- Action : saisir le mot clef « MUG »
- Contrôle : vérifier la liste des résultats affichés (5)

The screenshot shows a search results page for 'MUG'. At the top, there's a navigation bar with 'Home / Search results' and a dropdown for 'Sort by: Relevance'. Below the navigation, it says 'SEARCH RESULTS' and 'There are 5 products.' On the left, there's a sidebar with language ('English'), currency ('EUR €'), sign-in options, and a 'Cart (1)' button. The main area has a search bar with 'MUG' typed in and a magnifying glass icon. Below the search bar, there are five product cards:

Product Image	Name	Price
	Customizable Mug	€11.91
	Mug The Best Is Yet To Come	€10.19
	Mug The Adventure Begins	€10.19
	Mug Today Is A Good Day	€10.19

## 6.1 Critères d'automatisation

**Tous les tests n'ont pas vocation à être automatisés → critères**

### ■ Fréquence d'utilisation du test automatique.

- Plus le test est joué souvent (non régression, cycle de type agile) meilleur est le retour sur investissement.

### ■ Complexité de l'automatisation.

- KISS, le script de test ne doit pas devenir un autre système à tester

### ■ La compatibilité des outils avec le SUT.

- Elle doit être évaluée pour s'assurer que même les cas complexes seront automatisables. Il faudra également se poser la question du support (outil Open Source ou Payant)

### ■ Maturité du processus de test.

- L'automatisation (développement) nécessite un processus de test maîtrisé, dans le cas contraire on va désorganiser le processus de test.

## 6.1 Critères d'automatisation

### ■ Stabilité/adéquation du SUT.



Maintenance due  
aux évolutions  
fréquente

Meilleure retour sur  
investissement en particulier  
sur mode séquentiel

Fin de projet

Mauvais ROI.

- En cas de re-conception (architecture vieillissante, non adaptée à de futures évolutions), il peut être utile de faire une étude pour voir ce qui peut être réutilisé.

### 6.1 Critères d'automatisation

#### ■ Durabilité de l'environnement automatisé

- Adaptation au changement, diagnostic, maintenabilité, ajout de nouvelle fonctionnalité.

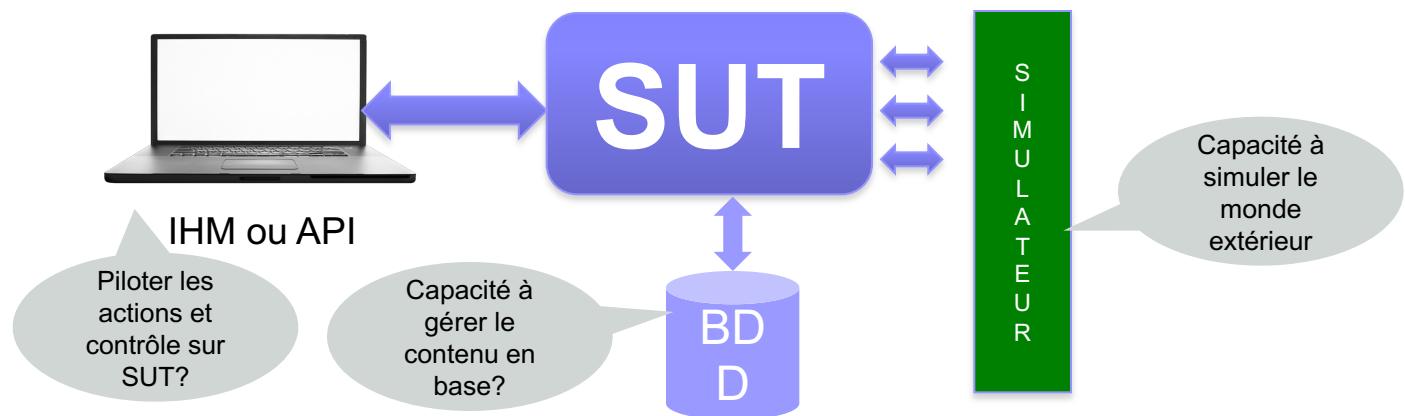
#### Exemple fil rouge:

Existant: un simulateur de SMS-C existe, mais il ne permet pas d'automatiser tous les cas, en particulier les cas d'erreurs. Il faut le faire évoluer également pour les nouvelles fonctionnalités, mais en l'état il apparaît que ces évolutions vont coûter cher.

Coût du nouveau simulateur versus le ROI des nouveaux tests automatisés.

## 6.1 Critères d'automatisation

- La possibilité de contrôler le SUT / la testabilité du SUT.



- Planification technique en support à l'analyse du ROI

- Etude pour vérifier le ROI
- Intégrer les coûts liés à l'automatisation
- Automatisation et chemin critique

## 6.1 Critères d'automatisation

### ■ Éléments pour préparer la transition vers l'automatisation

#### □ Disponibilité d'outils d'automatisation

- ❖ Les outils doivent être installés et mis à jour et testés dans la TAS.

#### □ Données de test et cas de test corrects (spécification)

- ❖ Procédures et résultats attendus

#### □ Coopération entre développeurs et ingénieurs automatisation des tests

- ❖ Les testeurs automaticiens ont besoin de connaître les aspects techniques du SUT, les développeurs par leur choix peuvent faciliter l'automatisation (contrôlabilité et observabilité)

#### □ Effort parallèle

- ❖ Lors de la transition il peut être judicieux d'avoir une équipe parallèle qui gère l'automatisation.

#### □ Reporting sur automatisation

- ❖ Statuts des tests, Statistiques générales, Performances de la TAS

## 6.1 Critères d'automatisation

### Périmètre de l'automatisation

- ❖ Est-ce un test de non-régression qui doit être joué à chaque livraison?
- ❖ **Criticité de la fonction associée?**
- ❖ Est-ce un cas de test nominal ou un cas d'erreur?
- ❖ **Le cas de test est-il simple à automatiser?**
- ❖ Le cas de test est-il sujet à erreur si manuel?
- ❖ Quelle charge pour automatiser?
- ❖ Est-il joué fréquemment?
- ❖ L'automatisation du test permet-il de le rendre plus fiable?

→ Ces critères sont définis en amont de projet et sont appliqués au moment de la spécification des tests.

## 6- Transition du test manuel vers un environnement automatisé

### 6.1 Critères d'automatisation



Quels objectifs, quel périmètre? critères  
Environnement de test:

Quels outils?  
Des simulateurs?  
Données?

Formations, expertises?  
Métriques?

→ Stratégie d'automatisation

## 6.1 Critères d'automatisation

Exemple de modèle de stratégie:

### ■ Périmètre/objectifs:

- Les objectifs principaux / les fonctionnalités à automatiser par priorité.
- La liste des critères éventuellement pondérés pour choisir les tests de régression et les tests à automatiser.

### ■ Approche

- Solution choisie parmi les pistes proposées:
  - ❖ Framework de test
  - ❖ Guides de réalisation des scripts de test

### ■ Ressources/rôles et responsabilités

- Description de l'équipe, expertise et responsabilité

### ■ Outils

- Version, License, formation

### ■ Planification : développement simulateur et outil de test, mise en place du framework de test, création des jeux d'essais, définition des environnements, installation des logiciels nécessaires

### ■ Environnement

- Plateforme, base de données, volumétrie

### ■ Livrables

- Rapport de test, script de test, métriques

### ■ Risques

- Risques liés à l'automation: environnement de test non disponible, outil de test non disponible ou défaillant, simulateur non disponible, livraison incomplète ...

### ■ Données de test

### ■ Rapports/Résultats

- Comment les résultats sont rapportés dans l'outil de gestion de test, où sont stockés les logs (gestion de configuration)

id	Différents Jeux de données	navigateurs	Plusieurs environnements?	Logique métier complexe	Differentiels jeux d'utilisateurs	Implique un gros volume de données	Dépendances?	Candidat à automation	Commentaires
TC01	Oui	Oui	Oui	Non	Non	Non	Non	Oui	Même cas de test pour différents jeux de données
TC02	Oui	Oui	Oui	Non	Oui	Non	Oui	Oui	
TC03	Non	Non	Non	Oui	Oui	Oui	Oui	Oui	Logique métier complexe, probabilité d'erreur importante si manuel
TC04	Non	Non	Non	Non	Non	Non	Non	Non	
TC05	Oui	Non	Non	Non	Oui	Non	Non	Non	2 jeux de données
TC06	Oui	Oui	Non	Oui	Non	Oui	Non	Oui	

## 6.1 Critères d'automatisation

### Connaissance de l'équipe sur le changement de concept

- Testeurs de différents horizons (métier, fonctionnel ...)
- T shape skill (hors test):

- Test API auto
- Test IHM auto
- Base de données
- API REST, SOAP
- Outils
- ...

Connaissances basiques sur les différents aspects techniques

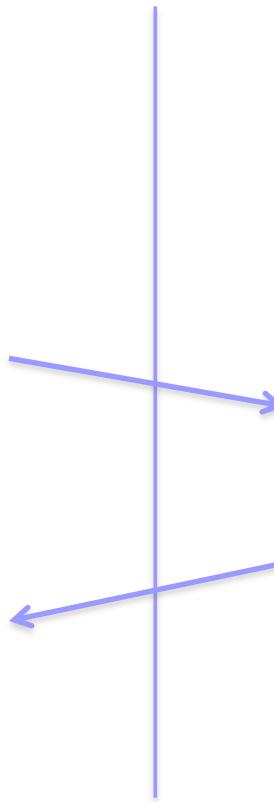
E  
X  
P  
E  
R  
T  
I  
S  
E

## 6.1 Critères d'automatisation

### Rôles et responsabilités : exemple

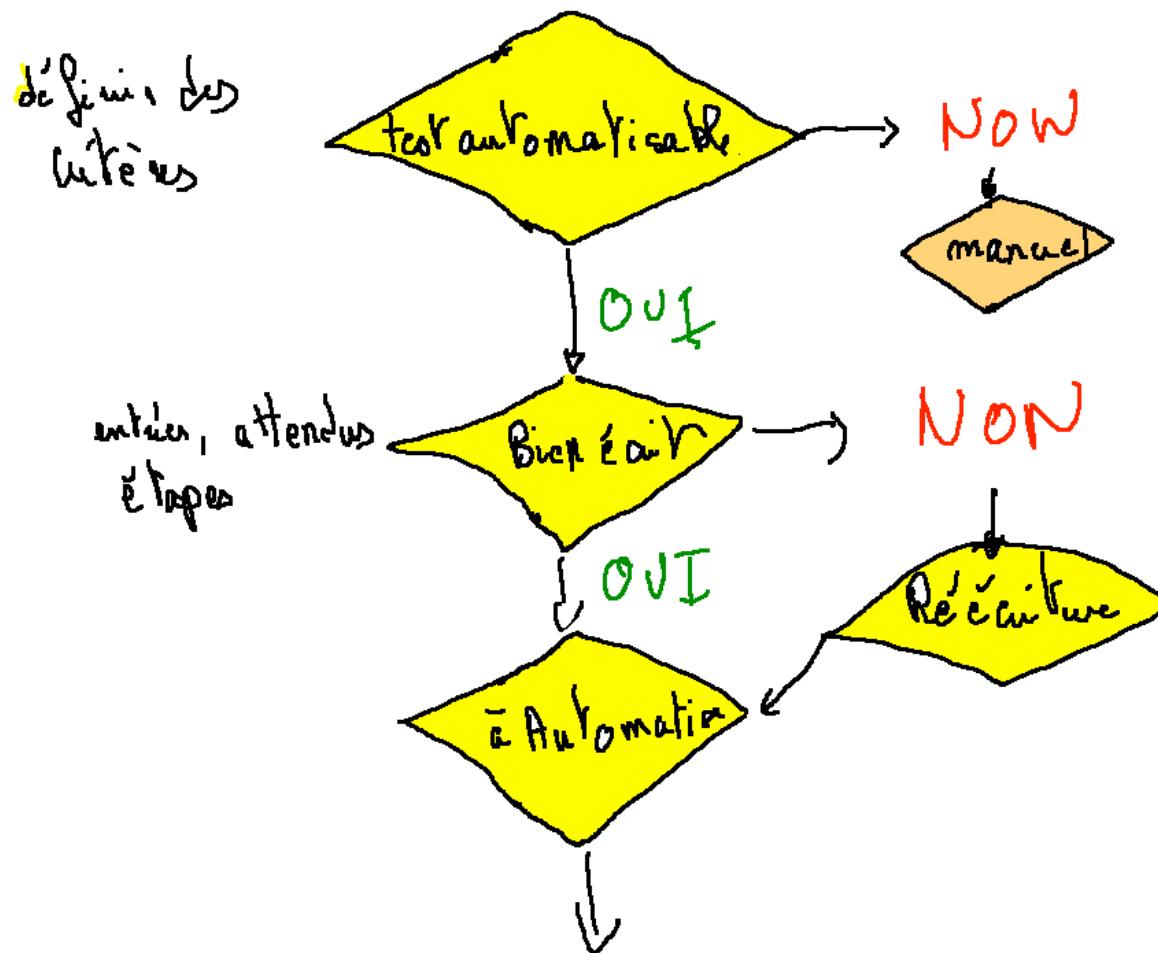


1. Spécification des tests et sélection des tests à automatiser.
2. Création d'une bibliothèque de mots-clés de haut niveau.
5. Mise au point des mots-clés de haut niveau avec le produit développé.
7. Exécution des tests.



3. Implémentation via la création d'une bibliothèque de mots-clés de bas niveau.
4. Mise au point de ces mots-clés avec le produit développé.

### ■ En résumé



## Test de régression = cible automatisation

- Fréquence d'exécution → ROI important
- Durée d'exécution: automatiser les tests les plus coûteux en temps permet d'optimiser le ROI, peuvent être joués plus souvent
- Recouvrement fonctionnel: identifier les composants fonctionnels réutilisables → diminution du coût de fabrication
- Partage de données: identifier les données communes à plusieurs tests (BDD, paramétrage ...) → diminution du coût de fabrication
- Couverture du SUT:
  - ❖ Large = maximum de fonctionnalité
  - ❖ Profondeur = fiabilité

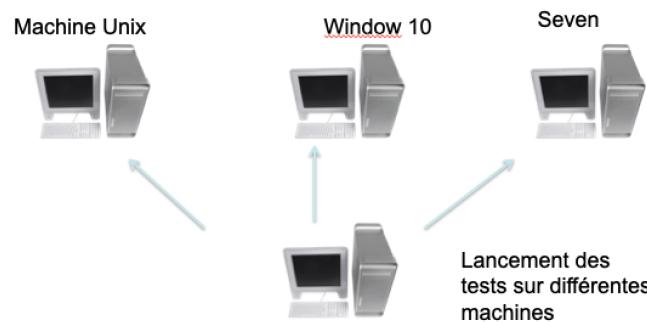
## Test de régression = cible automatisation

### □ Tests exécutables

- ❖ Vérifier le test manuellement permet de le corriger et ensuite de l'automatiser et d'identifier les potentiels point de blocage ou synchronisation.

### □ Les grands ensembles de test de régression

- ❖ A cause du nombre et de la durée d'exécution il peut être nécessaire de paralléliser les tests.
- ❖ Dans certains cas on peut devoir ne faire qu'un sous ensemble (coût/disponibilité de l'environnement de test)



# 6- Transition du test manuel vers un environnement automatisé



## 6.2 Identifier les étapes nécessaires pour implémenter les tests de régression

Suite de test

Test 1:  
Création d'un utilisateur

Test 2:  
Modification d'un utilisateur

Test 3:  
Suppression d'un utilisateur

Test

Pré-requis: être connecté en tant qu'administrateur

A1: rechercher un utilisateur

C1: Utilisateur accessible

A2: Ajouter un projet

A2: Vérifier projet affecté

Actions communes

Données partagées

Pool d'utilisateurs

Pool de Projets

## 6.2 Identifier les étapes nécessaires pour implémenter les tests de régression

Quelles étapes pourraient être mise en commun?

Quelles données?

Pour la connexion.

Contact us

English ▾

Currency: USD \$ ▾

Sign in

Cart (0)

my store

CLOTHES ACCESSORIES ART

Search our catalog



Log in to your account

Email

Password  SHOW

[Forgot your password?](#)

SIGN IN

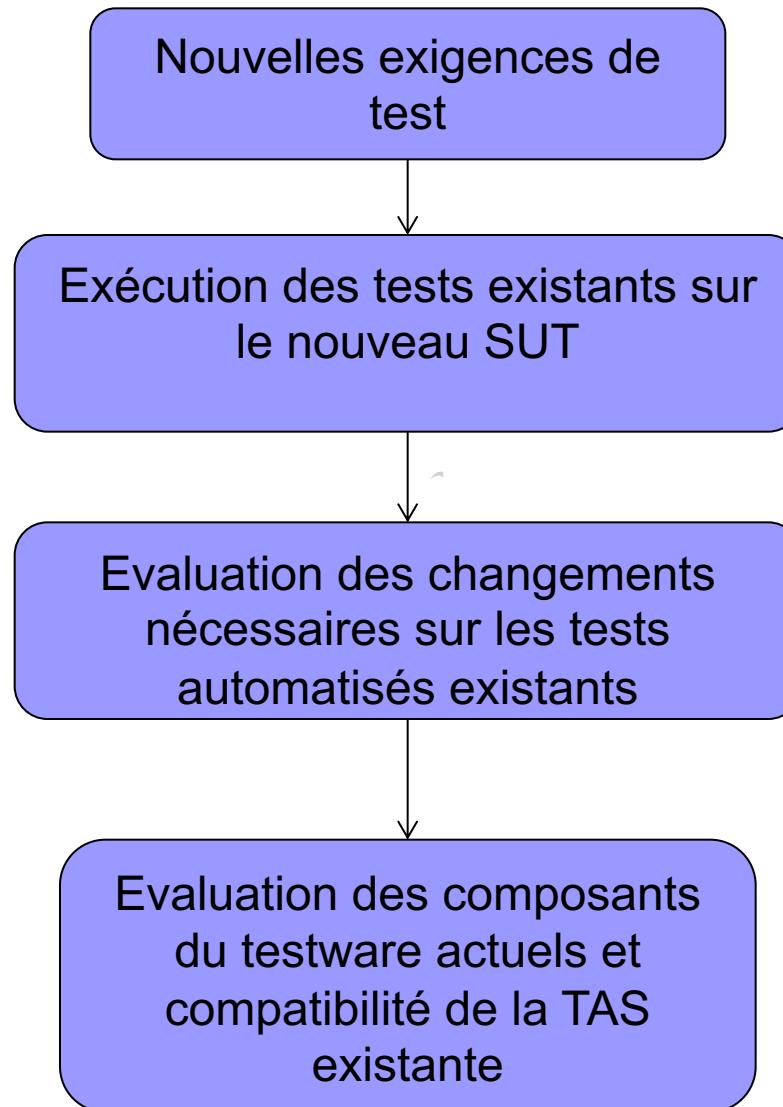
---

[No account? Create one here](#)

### 6.3 Facteurs à considérer lors de l'automatisation de tests de nouvelles fonctionnalités

- En faisant l'étude en amont il est possible de travailler de concert développeur/architecte pour avoir une SUT testable.
- Nécessité une analyse pour savoir si la TAS est adéquate pour le test des nouvelles fonctionnalités.
- Etude d'impact:
  - ajout, modification, évolution des composants de la TAS.
  - Outil de test supplémentaire
- Exemple: implémentation de nouvelles interfaces Rest pour accéder aux SUT → nouvel outil pour test pour ces interfaces.

### 6.3 Facteurs à considérer lors de l'automatisation de tests de nouvelles fonctionnalités

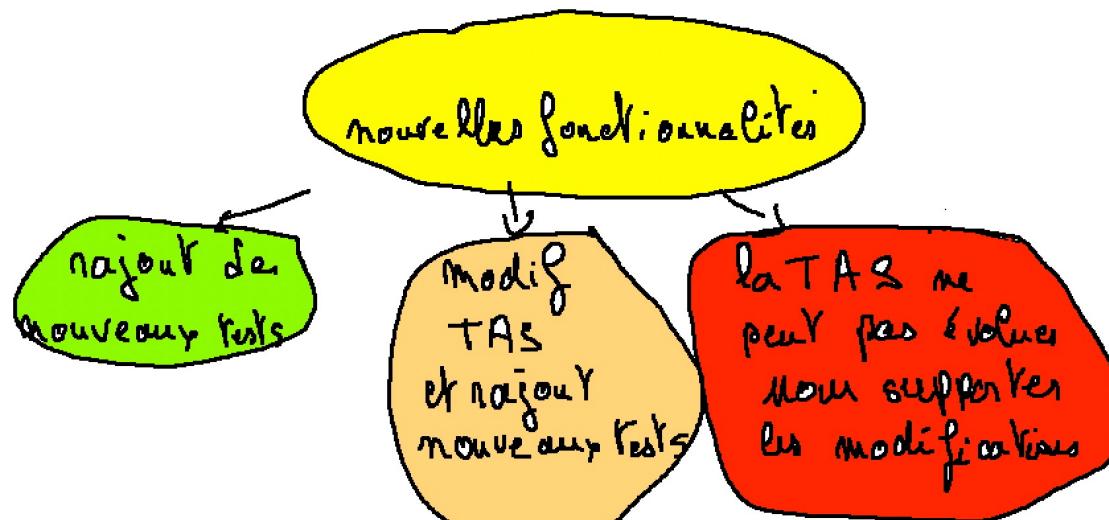


## ■ Mise à jour de la planification

- Une traçabilité exigence et cas de test permet de déterminer les cas de test à mettre à jour

## ■ Pour finir la TAS actuelle peut ne pas être en adéquation avec les évolutions du SUT.

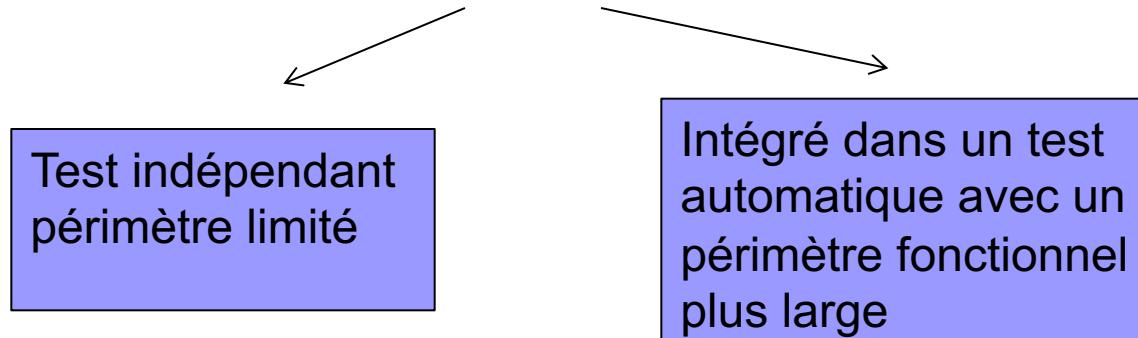
→ Nouvelle architecture si évolution de la TAS impossible ou trop coûteuse.



## 6.4 Facteurs à considérer lors de l'automatisation des tests de confirmation

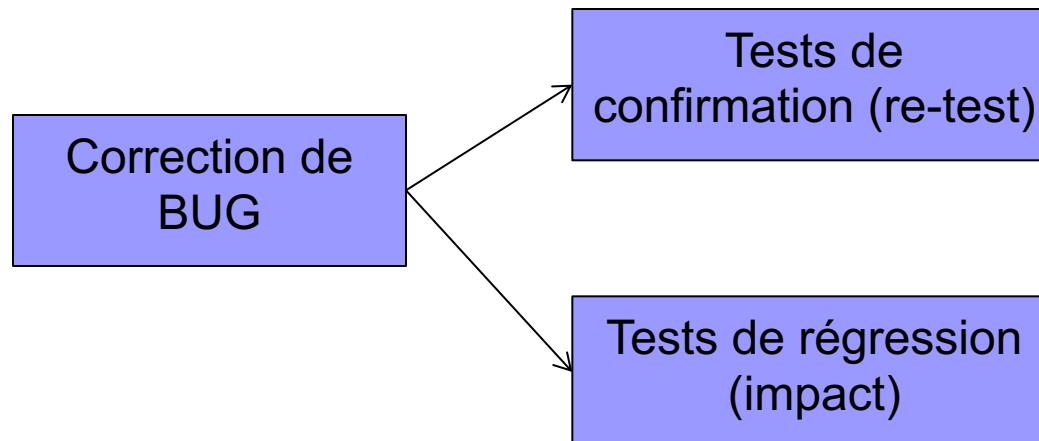
### ■ Rappel : test de confirmation = test vérifiant la correction d'anomalie

- Bon candidat à l'automatisation, car possible réapparition de ces problèmes suite à une mauvaise gestion de conf.

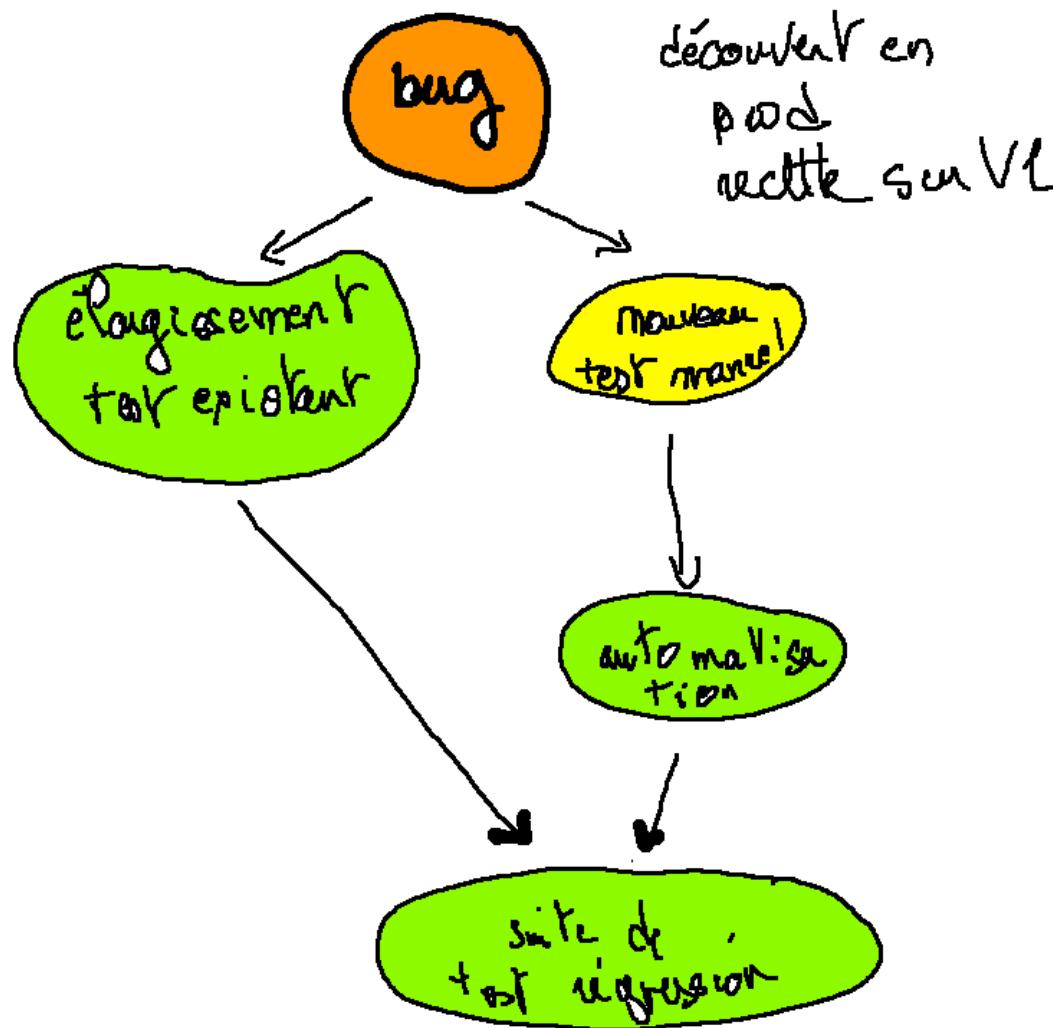


- Exemple un contrôle d'erreur (saisie d'une chaîne vide) que l'on intègre dans un test piloté par les données qui contrôle un ensemble plus large d'erreurs (caractères non autorisés, chaîne trop courte ...)

- **Suivi des coûts de correction des anomalies**
- **Attention à faire également l'analyse d'impact et à inclure les tests de régression nécessaires.**



## 6.4 Facteurs à considérer lors de l'automatisation des tests de confirmation



- Lors de la transition vers le test automatique que fait-on des testeurs manuels?
  - a) Ils sont mis sur d'autres projets
  - b) On les forme à la programmation pour écrire les scripts
  - c) Ils seront formés à la programmation tout en les assurant de leur expertise dans le test.
  - d) On leur dit que s'ils n'ont pas de compétences en programmation ils vont perdre leur boulot.
- Comment peut-on gérer l'inter dépendance entre test
  - a) Stocker les data utilisées par de multiples tests à l'extérieur.
  - b) Passer les donner d'un test à l'autre via le script
  - c) Lier les tests dans un test plus large
  - d) Faire en sorte que les tests soient toujours dans le même ordre grâce à un harnais de test.

- Si vous avez utilisé une approche axée sur les mots-clés pour les scripts d'automatisation des tests et que vous devez maintenant tester de nouvelles fonctionnalités qui ont été ajoutées au SUT, que devez-vous faire ?
  - a. Si possible, ajoutez de nouveaux mots-clés pour la nouvelle fonctionnalité.
  - b. Si possible, ajoutez de nouveaux scripts basés sur des données pour la nouvelle fonctionnalité.
  - c. Examinez l'approche axée sur les mots-clés pour voir si elle est toujours la bonne solution.
  - d. Envisagez d'utiliser un nouvel outil de test qui vous permettra de tester à partir de l'API plutôt que de l'interface.

- **Quelle est la raison principale de l'automatisation des tests de confirmation ?**
  - a. Il fournit des tests de régression pour les domaines qui ont été affectés par un changement.
  - b. Il vérifie que le développeur a corrigé correctement le problème signalé
  - c. Il vérifie que le défaut ne s'est pas réintroduit dans une version ultérieure.
  - d. Il couvre une zone de fonctionnalité plus large et est susceptible de capter tout changement

### 7.1 Vérification des composants de l'environnement de test automatisé

### 7.2 Vérification de la suite de test automatisés

# Objectifs d'apprentissage du chapitre

## 7. Vérification de la TAS

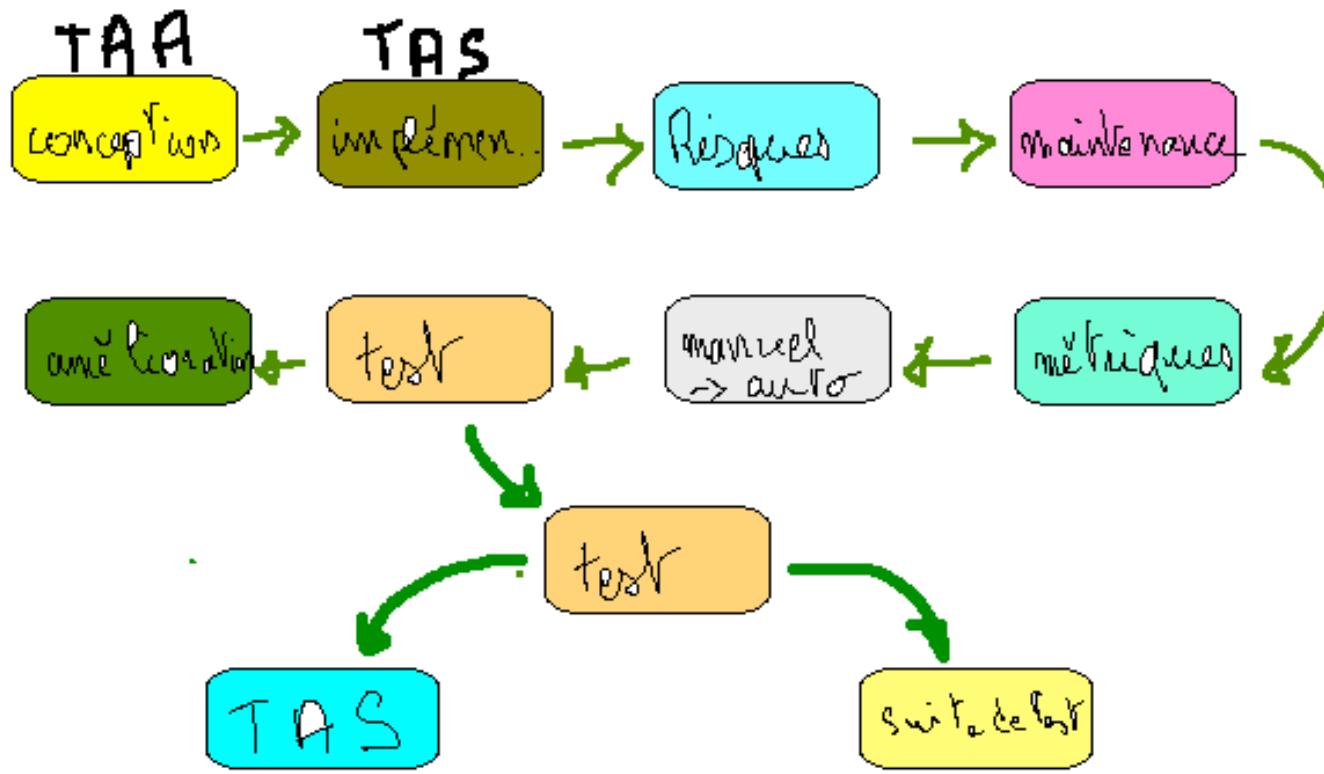
### 7.1 Vérification des composants de l'environnement de test automatisé

ALTA-E-7.1.1 (K3) Appliquer les contrôles de validité à l'environnement de test automatisé et à la configuration de l'outil de test

### 7.2 Vérification de la suite de tests automatisés

ALTA-E-7.2.1 (K3) Vérifier le comportement correct d'un script de test automatisé et/ou d'une suite de test donnée

## 7- Vérification de la TAS



méthodologie config  
comparaison ...

compteur fiable

## 7.1 Vérification des composants de l'environnement de test automatisé

Avant de lancer l'exécution des tests automatisés il est nécessaire de vérifier un certain nombre de points:

Installation, initialisation, configuration et customisation de l'outil de test.

Exécution de scripts avec succès et échec connus

Répétabilité dans la mise en place/destruction de l'environnement de test

Configuration de l'environnement de test et des composants

Connectivité sur les systèmes/interfaces internes et externes

Tester les caractéristiques de la structure.

Intrusivité des outils de tests automatisés ds l'environnement de test

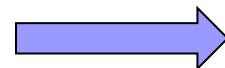
## ■ Installation, initialisation, configuration et customisation de l'outil

- L'idéal est de pouvoir installer à partir d'un repository central (dépôt) de façon automatisée de façon à éviter:
  - ❖ Des problèmes de versions des outils
  - ❖ Des configurations et versions différentes sur différents SUT
  - ❖ Ne pas bénéficier des dernières mises à jour/corrections
- Nécessite des composants identifiés et versionnés
- Les installations/configurations manuelles sont source d'erreurs
  - ❖ Copie de fichiers
  - ❖ Configuration manuelle

## ■ Exécution de scripts de tests avec des succès et échec connus.



Tests OK en échec → pb composant



Test KO en succès → pb composant

## ■ Que contrôle-t-on?

- Les enregistrements/résultats de test sont-ils correctement générés?**
- Les performances (temps d'exécution des tests).**
- L'installation et le démontage (setUp et TearDown) des tests se font-ils correctement?**

- Répétabilité de la mise en place/destruction de l'environnement de test.

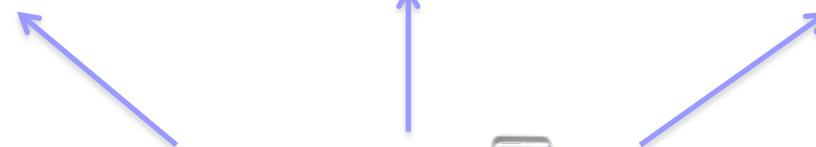
Machine Unix



XP



Windows 10



Déploiement de la TAS  
dans différents  
environnements de test

- Comportement identique
- Objectif atteignable grâce à la gestion de configuration.

#### ■ Gestion de configuration de l'environnement de test et des composants

- Documentation de la TAS
- Formation

#### ■ Connectivité interface

- Vérifier (à l'aide de tests et pré-test) avant de lancer les tests automatiques que les interfaces externes et internes (hook) sont disponibles

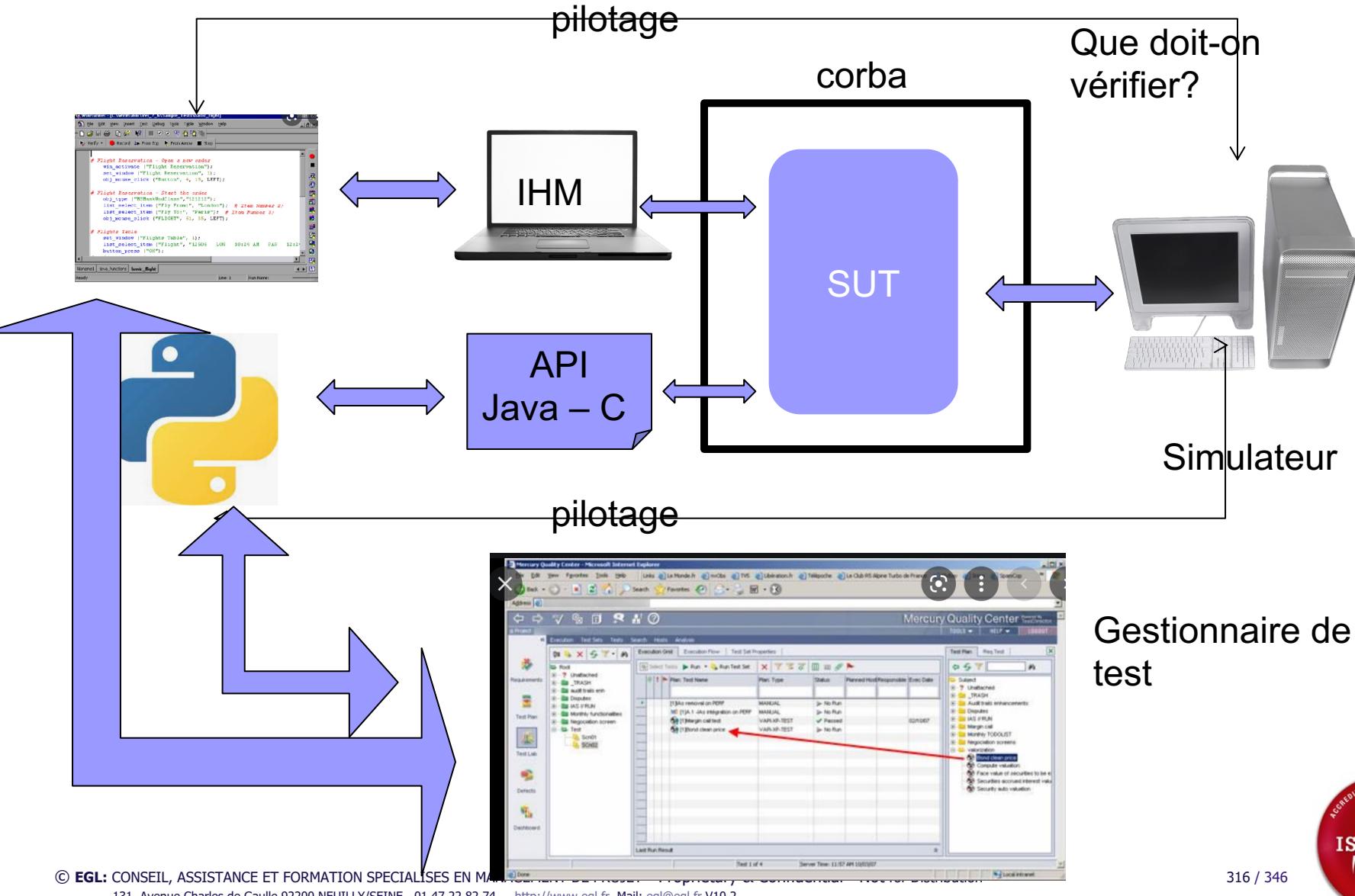
#### ■ Tester les caractéristiques (techniques) de la structure

- Détection des fuites mémoires, erreurs inattendues, dégradation performance, comparaisons complexes ...

- **Intrusivité → peut affecter le comportement du SUT**
  - **Niveau bas si utilisation des interfaces externes**
    - ❖ Signaux électroniques, USB etc...
  - **Niveau plus élevé si injection de commandes**
    - ❖ Test IHM (ex envoie de commandes javascript sur un navigateur), classique sur les outils de tests.
  - **Niveau important si utilisation de ressources internes du SUT**
    - ❖ API internes
- **Un niveau d'intrusion élevé peut générer des erreurs qui n'arriveraient pas dans le monde réel. C'est pour cela qu'il est nécessaire d'évaluer l'impact du niveau d'intrusion.**

# 7- Vérification de la TAS

## 7.1 Vérification des composants de l'environnement de test automatisé



### ■ Points à vérifier

- Exécuter les scripts de test avec les réussites et les échecs connus
  - ❖ Tests OK ou KO → Tester les tests , les faire évoluer
- Vérifier la suite de tests
  - ❖ Complète, version, framework et SUT compatible
- Vérifier les nouveaux tests sur le SUT
  - ❖ Attention spécifique → erreurs probables
- Tests implémentant des nouvelles fonctionnalités de la TAS
  - ❖ Pb dans la TAS
- Prendre en compte la répétabilité des tests
  - ❖ Instabilité, pb intermittents → si le test ne peut être fiabilisé mieux vaut le retirer de la suite de test.
- Points de vérification suffisants/incomplets
  - ❖ Preuves de réalisation des tests via les logs

**Vous avez créé une TAS qui sera utilisée pour plusieurs projets dans votre organisation. Jusqu'à présent, vous avez configuré manuellement la TAS pour chaque projet. Ces configurations n'ont posé aucun problème et vous avez pu vérifier manuellement que la configuration était correcte.**

**Votre responsable vient de vous informer que le succès du TAS est ressenti dans toute l'entreprise. Vous allez installer le TAS pour d'autres SUT dans les semaines à venir. Laquelle des propositions suivantes est la meilleure approche pour gérer cette installation ?**

- a. Continuez l'installation manuelle car elle fonctionne et vous permet de vérifier manuellement les résultats.**
- b. Continuer avec l'installation manuelle mais automatiser la vérification des résultats en créant un test d'acceptation automatisé pour le TAS.**
- c. Automatiser l'installation en créant des scripts qui copieront les fichiers d'un TAS fonctionnel à un autre.**
- d. Automatiser l'installation en créant des scripts d'installation qui installeront le TAS à partir d'un référentiel central.**

**Vous venez de lancer un TAS qui utilise des scripts SQL pour accéder à la base de données SUT afin d'acquérir les données nécessaires aux tests. Les équipes de développement expérimentent différentes bases de données et ne vous indiquent pas toujours celle qu'elles utilisent. Vous avez construit une matrice de gestion de configuration complexe qui charge les scripts SQL appropriés pour la base de données utilisée par un SUT particulier. Malheureusement, les développeurs ont continué à apporter des modifications et vous constatez que les scripts SQL incorrects sont parfois chargés, ce qui entraîne un échec complet de l'exécution de l'automatisation.**

**Compte tenu de ces informations, quelle serait la meilleure approche pour s'assurer que les bons scripts sont chargés ?**

- a. Demander aux développeurs de vous dire quelle base de données est utilisée.
- b. Charger un ensemble de scripts SQL par défaut, et lorsqu'ils échouent à cause d'un type de base de données incorrect, le signaler comme un défaut.
- c. Exécuter une courte série de tests qui utilise un ensemble de scripts SQL et, en fonction de ceux qui fonctionnent, charger l'ensemble approprié de scripts à utiliser pendant l'exécution de l'automatisation.
- d. Retirer le composant base de données des scripts et utiliser à la place des données codées en dur.

- Vous avez pris en charge une TAS et créé une version de référence. Cette version servira de base à toutes les TAS pour les projets. Vous voulez vérifier si cette TAS fonctionne correctement. Quel est le meilleur moyen?
  - a) Exécuter un script en erreur et vérifier qu'il est passant.
  - b) Exécuter une suite de test et vérifier qu'elle se finit sans erreur
  - c) Exécuter une suite de tests et vérifier que les résultats sont consistant avec les précédentes exécutions.
  - d) Exécuter un test des tests qui active une nouvelle fonctionnalité du framework et vérifier l'exécution avec succès.

- Vous êtes en train de déployer une suite d'automatisation pour un nouveau produit. Le développement se poursuit et l'automatisation doit s'adapter à la nouvelle fonctionnalité et doit aussi continuer à fonctionner pour l'existant. Compte tenu de ces informations, quel est le domaine le plus susceptible d'échouer dans l'automatisation des tests et que devez-vous surveiller pour vous assurer qu'il n'échoue pas ?
  - a) Échecs connus ; s'assurer que les tests qui devraient échouer continuent de le faire.
  - b) Réussites connues ; s'assurer que les tests qui devraient réussir continuent à réussir
  - c) Nouvelles fonctionnalités ; surveillez les nouveaux tests pour vous assurer qu'ils fonctionnent correctement.
  - d) Fonctionnalité stable ; contrôlez les tests existants pour vous assurer qu'ils fonctionnent correctement.

### 8.1 Options pour améliorer l'automatisation des tests

### 8.2 Adaptation de l'automatisation des tests à l'environnement et aux changements du SUT

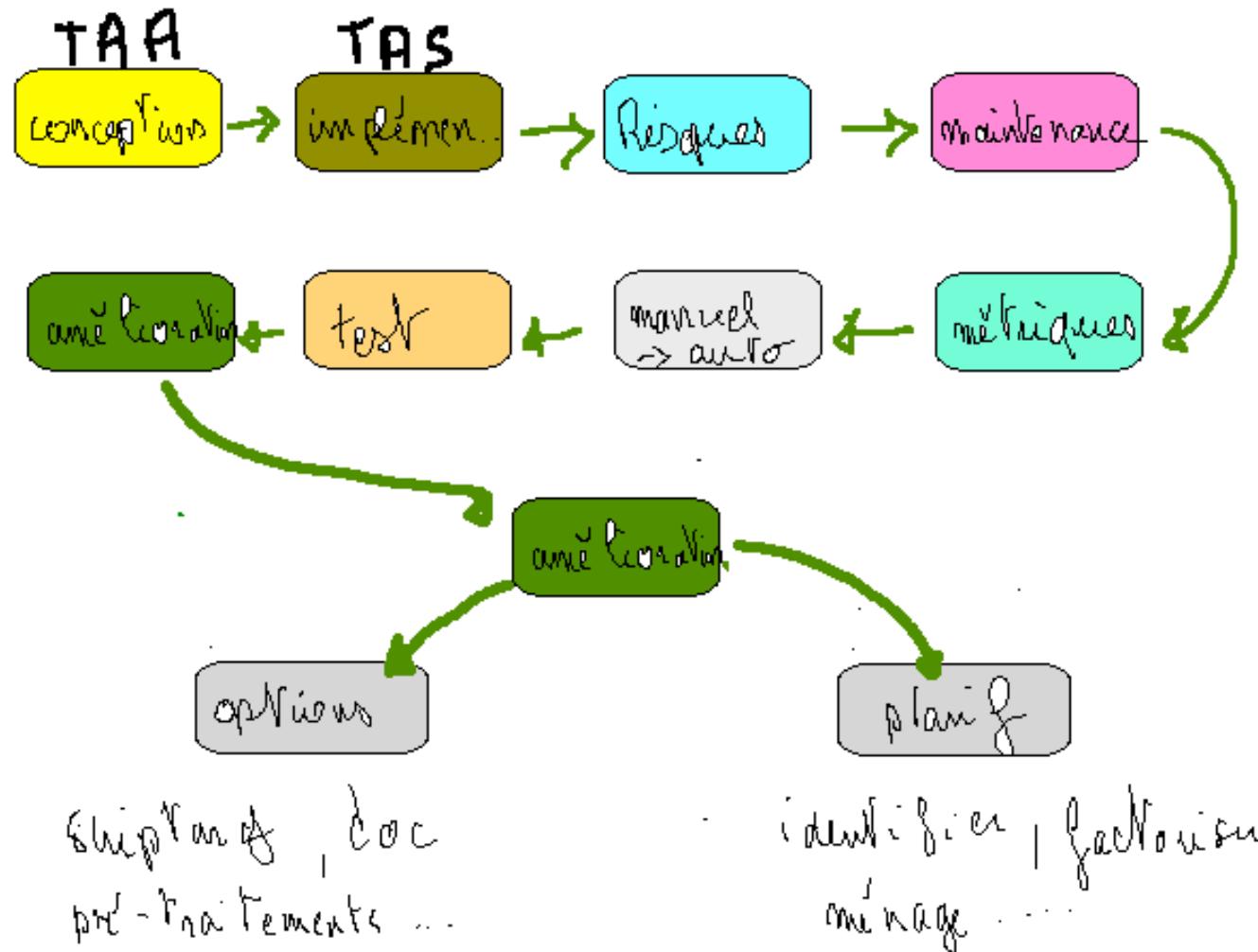
### 8.1 Options pour améliorer l'automatisation de test

**ALTA-E-8.1.1 (K4) Analysez les aspects techniques d'une solution d'automatisation de test déployée et fournissez des recommandations pour son amélioration**

### 8.2 Adaptation de l'automatisation des tests à l'environnement et aux changements du SUT

**ALTA-E-8.2.1 (K4) Analyser le testware automatisé, y compris les composants de l'environnement de test, les outils et les bibliothèques de fonctions, de façon à comprendre où la consolidation et les mises à jour doivent être faites en fonction d'un ensemble donné de changements d'environnement de test ou de modifications du SUT**

## 8- Amélioration continue



## ■ Amélioration continue

### □ Efficacité

- ❖ Réduire les tâches manuelles restantes comme la mise en place du framework d'automatisation, des configurations du SUT, du provisionning de données etc ...

### □ Facilité utilisation

- ❖ Passer à un scripting basé sur les mots-clés

### □ Compétences

- ❖ Former les testeurs à de nouvelles techniques d'automatisations

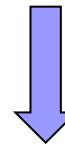
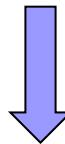
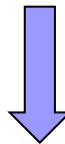
### □ Etc ...

→ Objectif trouver l'axe d'amélioration qui apporte le plus de Retour Sur Investissement.

Maintenance



Améliorations



scripting vérification Architecture Pré-traitement documentation Support  
Post-  
traitement

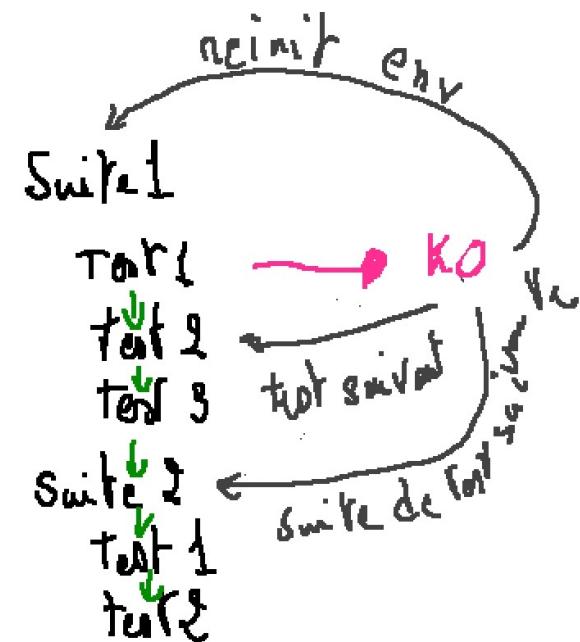
## ■ Scripting: améliorer l'approche

### □ Evaluer le chevauchement

- ❖ Repérer les actions et contrôles communs (mots-clés)
- ❖ Bibliothèques d'actions et vérifications (éventuellement paramétrables)

### □ Gestion erreur

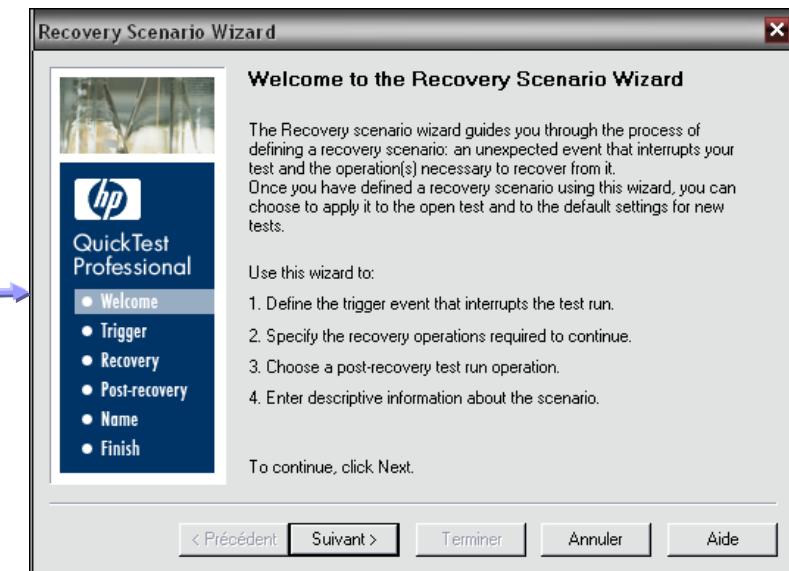
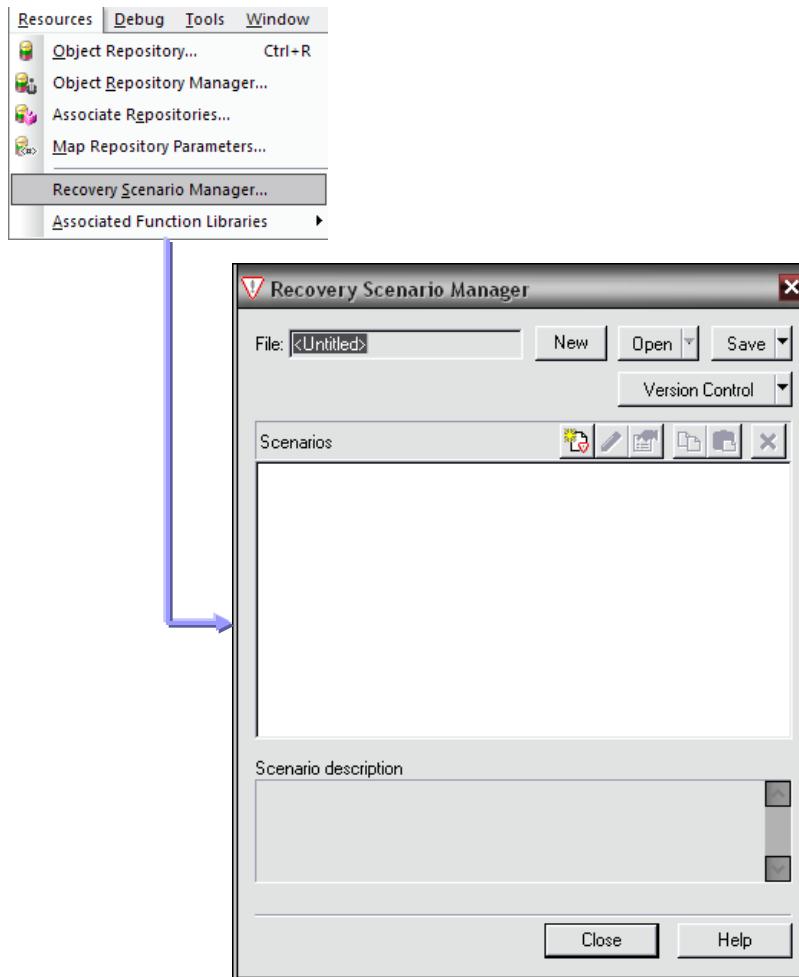
- ❖ Action de récupération (redémarrer le SUT ex)
- ❖ Passer à la suite de test suivante



## 8- Amélioration continue

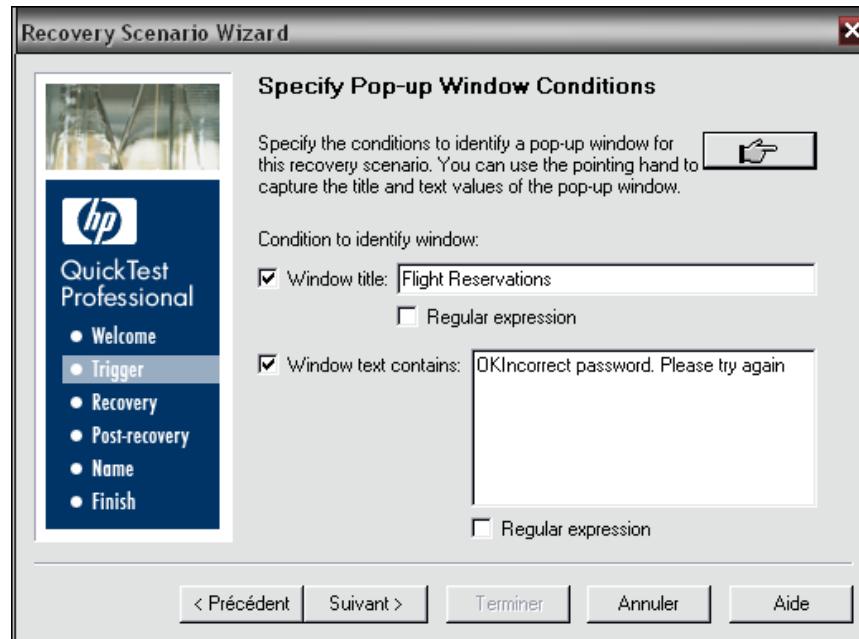
### 8.1 Options pour améliorer l'automatisation de test

#### Exemple de récupération d'erreur



# 8- Amélioration continue

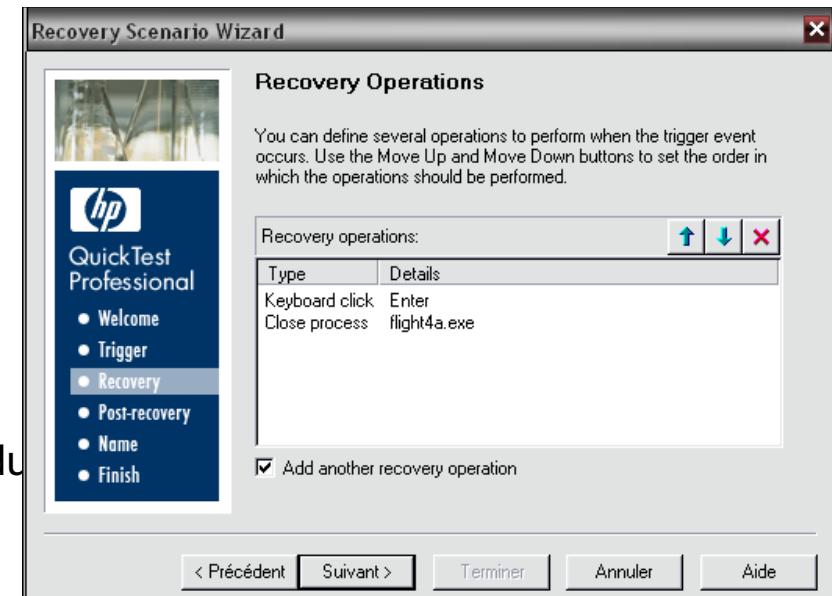
## 8.1 Options pour améliorer l'automatisation de test



**Recovery** : opération qu' il convient d' effectuer pour reprendre l' exécution du test : cliquer sur le bouton ok d' une fenêtre contextuelle, redémarrer windows...

## Exemple de récupération d'erreur

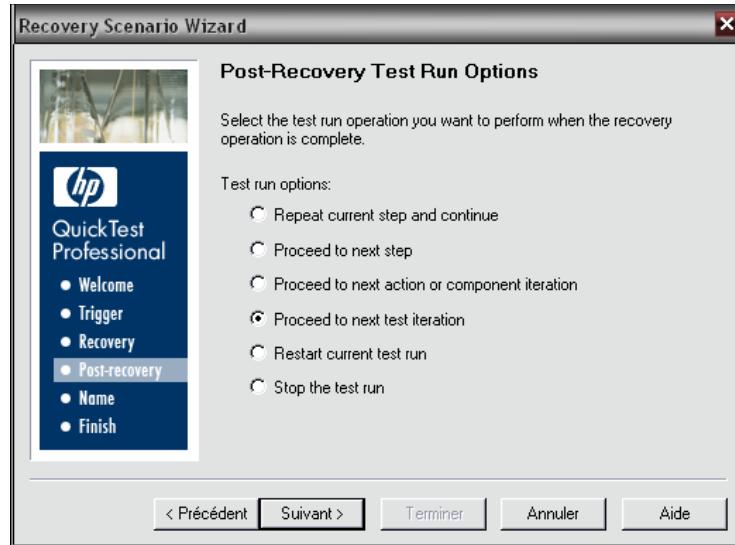
**Trigger:** événement déclenchant l' interruption du test : ouverture d' une fenêtre contextuelle, erreur d' exécution...



## 8- Amélioration continue

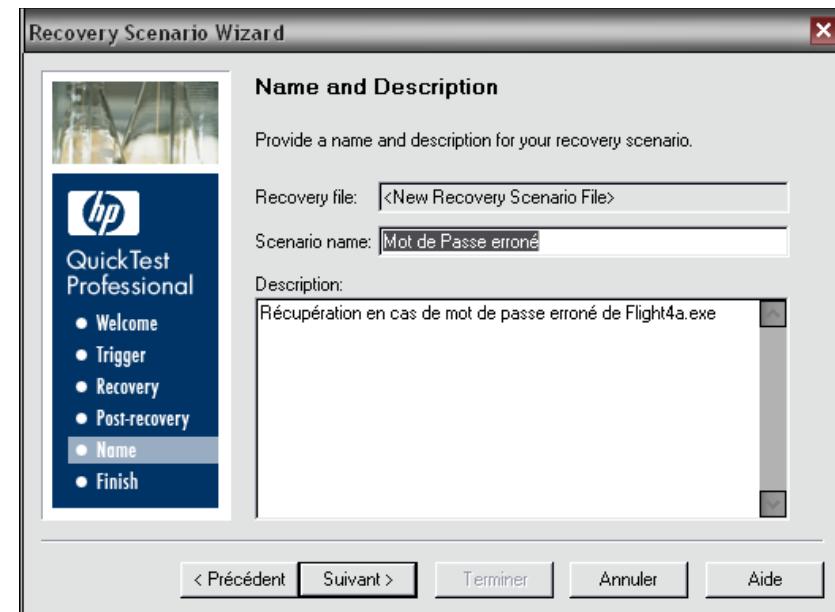
### 8.1 Options pour améliorer l'automatisation de test

#### Exemple de récupération d'erreur



Ce scénario sera associé au test en cours ou à l'ensemble des tests QTP et sera modifiable à partir du gestionnaire des scénarios de reprise.

**Post recovery** : instruction qui indique à QTP comment se comporter, après avoir surmonté l'évènement interrompant le test, en indiquant le mode de reprise souhaité : redémarrer le test depuis le début, passer directement à l'étape suivante,...



## 8- Amélioration continue

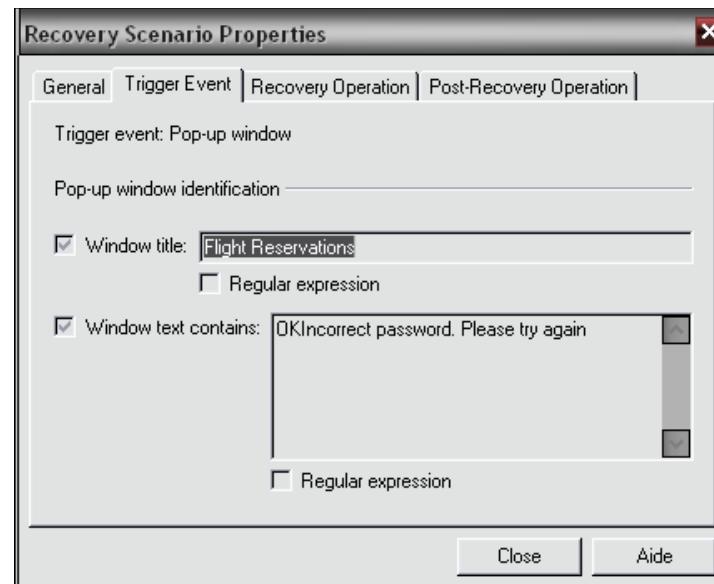
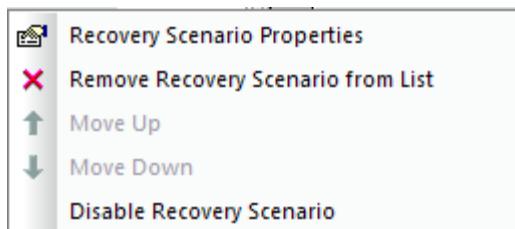
### 8.1 Options pour améliorer l'automatisation de test

The screenshot shows a software interface for test automation. On the left, there's a tree view of resources: 'Associated Function Libraries', 'Associated Recovery Scenarios' (with a red warning icon for 'Mot de Passe erroné'), 'Associated Repositories per Action' (with 'Internal Actions' and 'External Actions' branches), and 'Action1'. The main panel shows a table for 'Action1' under 'Login'. The table has two columns: 'Item' and 'Operation'. The items listed are 'Agent Name' (Set), 'Agent Name' (Type), 'Password' (SetSecure), and 'OK' (Click). Each item has a small icon next to it.

### Exemple de récupération d'erreur

Un scénario de reprise peut lors de sa création être associé au test en cours ou par défaut à tout nouveau test. Il apparaîtra alors dans les ressources du test.

Ses caractéristiques sont facilement accessibles, il pourra être désactivé ou supprimé du test en cours.



## ■ Scripting: améliorer l'approche

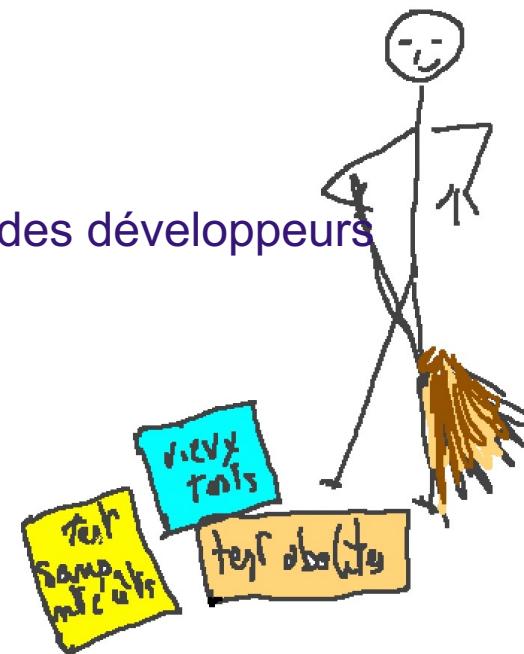
### □ Mécanisme d'attente

- ❖ Suppression des attentes en dur (gain de temps)
- ❖ Attente par polling avec un timeout (pas de perte de temps)
- ❖ Souscription à des evts avec le SUT (techno adaptée)

### □ Testware=logiciel

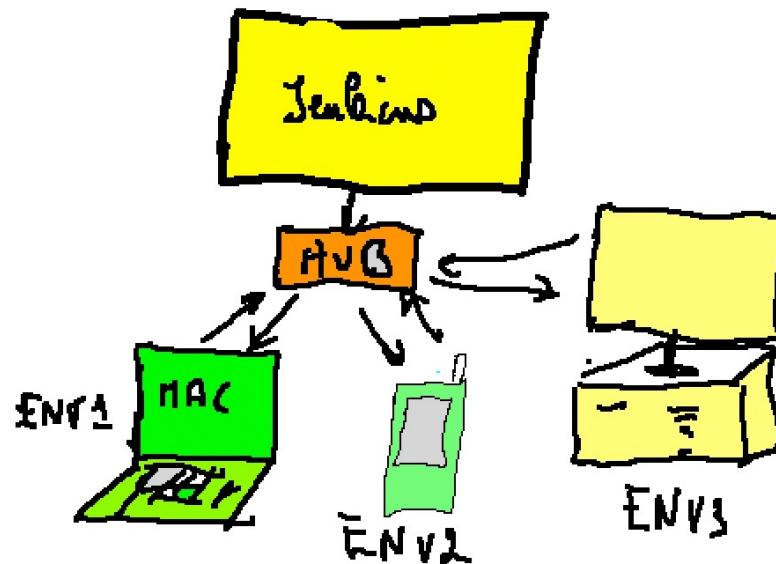
- ❖ Normes de codage
- ❖ Analyse statique
- ❖ faire développer des composants par des développeurs

### □ Faire le tri (suppression et maj)



## ■ Exécution des tests

- Durée de la suite
- Dispatching des tests sur différents environnement de test
- Création de suite de test en fonction de la durée
- Suppression de doublons



#### ■ Vérification

- Mise en commun de vérifications standards**
- Vérification paramétrées**

#### ■ Architecture SUT ou TAS

- Amélioration de la testabilité → peut-être coûteux si fait tardivement**
- Importance du test suffisamment tôt**

#### ■ Pré-traitement(installation) et Post-Traitemen(suppression)

- Standardiser des tâches avant et après des tests automatisés**

### ■ Exemple : Un localisateur XPath générique

```
def find_by_xpath(driver, path_string):  
    element = driver.find_element_by_xpath('path_string')  
    return element
```

### ■ Peut être appelé pour différents attributs en changeant la chaîne qui est passée

```
path_string = "//*[@id = '%s']" % 'id_to_find'  
path_string = "//*[@class = '%s']" % 'class_to_find'
```

### ■ Appel de la fonction générique

```
element_found = find_by_xpath(driver, path_string)
```

## 8.1 Options pour améliorer l'automatisation de test

Que peut-on faire pour améliorer le code?

```
@Test
// exemple javascript
public void testPrestashop5() throws Exception {
    // Aller sur la page prestashop
    driver.get("http://prestashop.qualifiez.fr/en/");
    // saisir le champ de recherche
    driver.findElement(By.name("s")).sendKeys("MUG");
    // cliquer sur le bouton de recherche
    driver.findElement(By.cssSelector("button > .search")).click();
    // contrôler le résultat
    assertEquals(driver.findElement(By.cssSelector(".total-products > p")).getText(),"There are 5 products.");
}

@Test
// exemple javascript
public void testPrestashop6() throws Exception {
    // Aller sur la page prestashop
    driver.get("http://prestashop.qualifiez.fr/en/");
    // saisir le champ de recherche
    driver.findElement(By.name("s")).sendKeys("Mug");
    // cliquer sur le bouton de recherche
    driver.findElement(By.cssSelector("button > .search")).click();
    // contrôler le résultat
    assertEquals(driver.findElement(By.cssSelector(".total-products > p")).getText(),"There are 5 products.");
}
```

Décrivez les mécanismes d'attentes dans les 2 tests?

```
@Test
// exemple javascript
public void testPrestashop5() throws Exception {
    // Aller sur la page prestashop
    driver.get("http://prestashop.qualifiez.fr/en/");
    // saisir le champ de recherche
    driver.findElement(By.name("s")).sendKeys("MUG");
    // cliquer sur le bouton de recherche
    driver.findElement(By.cssSelector("button > .search")).click();
    // attendre et contrôler le résultat
    Thread.sleep(5000);
    assertEquals(driver.findElement(By.cssSelector(".total-products > p")).getText(),"There are 5 products.");
}

@Test
// exemple javascript
public void testPrestashop7() throws Exception {
    // Aller sur la page prestashop
    driver.get("http://prestashop.qualifiez.fr/en/");
    // saisir le champ de recherche
    driver.findElement(By.name("s")).sendKeys("MUG");
    // cliquer sur le bouton de recherche
    driver.findElement(By.cssSelector("button > .search")).click();
    // attendre et contrôler le résultat
    WebDriverWait wait = new WebDriverWait(driver, 30);
    wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.cssSelector(".total-products > p"))));
    assertEquals(driver.findElement(By.cssSelector(".total-products > p")).getText(),"There are 5 products.");
}
```

#### ■ Documentation

- Toutes formes de documentations peuvent être améliorées et standardisées**

#### ■ Fonctionnalités de la TAS

- Nouveaux reporting**
- Fonction d'enregistrement pour par ex faciliter la création de nouveaux scripts**
- Intégration avec d'autres systèmes**

#### ■ Mise à Jour/améliorations de la TAS

- Les nouvelles versions de la TAS doivent être testées avant d'être utilisée avec des tests représentatifs**

## 8- Amélioration continue

### 8.2 Planification de la mise en œuvre des améliorations de l'automatisation de test.

- **Comme tout logiciel toute modification doit être planifiée avec attention pour éviter toute régression en particulier sur:**
  - **La fiabilité**
  - **La performance**
- **Identifier les changements**
  - **Nécessite une analyse d'impact**
- **Augmenter l'efficacité et la productivité des fonctions de la TAS**
  - **Nouveaux moyens plus performants à intégrer**
    - ❖ (ex: techniques de dev, dll)
- **Cibler les diverses fonctions qui agissent sur le même type de contrôle**
  - **Consolider en généralisant des contrôles spécifiques**
    - ❖ Contrôles graphique

#### ■ Refactoriser la TAA

- Certaines évolutions du SUT vont nécessiter une évolution de l'architecture de test.
- Toujours prévoir une architecture évolutive

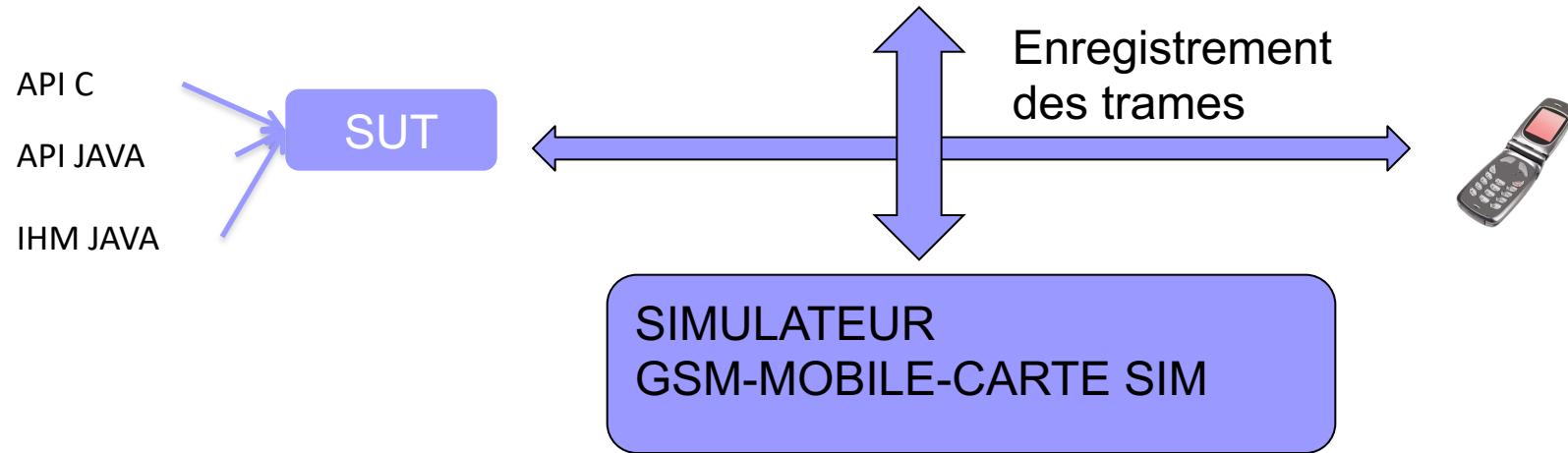
#### ■ Convention de nommage

- Respecter les conventions de nommage, cohérente sur les changements.

#### ■ Evaluation des scripts existants pour la révision/retrait du SUT

- Cette évaluation fait partie du processus de changement et d'amélioration
- On peut par exemple éliminer les tests peu fiables, coûteux en analyse de résultats

## 8.2 Amélioration continue EXEMPLE fil rouge



Les scripts de tests côté simulateur sont générés à partir d'enregistrement fait sur le vrai réseau avec un téléphone mobile.

→ PB: ces enregistrements nécessitent un abonnement + un vrai téléphone et peuvent être laborieux.

→ L'outil a été amélioré en connectant au simulateur un lecteur de carte SIM. Les enregistrements sont facilités, plus d'abonnement, ni téléphone.

→ Cette modification n'a pas d'impact sur la solution.

- **Et vous quelles améliorations avez-vous mis en place?**

- Vous travaillez avec une solution d'automatisation des tests qui est utilisée depuis un an. Vous avez constaté qu'il y a souvent des faux positifs, ce qui demande beaucoup de temps de recherche et de debug.

Bien que chacun de ces faux positifs soit corrigé lorsqu'il est découvert, il semble toujours y en avoir de nouveaux lorsque de nouveaux scripts sont écrits.

Quel est le problème le plus probable avec la TAS qui permet à ces faux-positifs de continuer à être introduits ?

- a. Les mécanismes d'attente ne sont pas corrects et le logiciel poursuit son exécution au lieu d'attendre une réponse de l'action précédente.
- b. Il y a un chevauchement dans les cas de test et les étapes de test qui fait qu'un changement dans un domaine casse le même code qui est utilisé ailleurs.
- c. Les mécanismes de récupération d'erreur du TAS et du SUT ne sont pas compatibles.
- d. Les fonctions de vérification ne sont pas normalisées et sont codées pour chaque script plutôt que d'utiliser une méthode de vérification éprouvée.

- Vous avez implémenté une TAS pour l'automatisation de WEB services. Il arrive que certains de ces services sont parfois indisponibles causant une erreur dans le script de test. Cela provoque des erreurs en cascade et vous fait perdre du temps. Il peut-être également difficile de trouver la cause racine du problème.

Etant donné ces informations quel serait le meilleur process à implémenter quand les scripts détectent un service indisponible

- a) Arrêter le test au moment qui permet d'éviter les erreurs en cascade
- b) Redémarrer le test quand la panne est découverte
- c) Redémarrer le service à partir du script, attendre qu'il soit prêt et continuer l'exécution.
- d) Redémarrer le système à partir du script, attendre qu'il soit prêt et continuer l'exécution.

- Vous travaillez avec une équipe d'automatisation qui a travaillé sur plusieurs projets. Les différentes fonctions développées par l'équipe ont été sauvegardées dans la bibliothèque de fonctions et sont disponibles pour l'équipe d'automatisation. Vous avez cherché dans la bibliothèque de fonctions une fonction qui gère les tableaux GUI (IHM) et vous avez découvert qu'il existe cinq fonctions différentes créées pour manipuler les données dans les tableaux.

En les regardant, vous constatez qu'elles se ressemblent toutes, même si certaines sont plus élégantes que d'autres.

Que devez-vous faire ?

- Développez votre propre contrôle de table car vous serez sûr qu'il fonctionne pour votre application.
- Prenez l'un des contrôles de table existants et réutilisez-le ou modifiez-le pour l'adapter à votre application.
- Consolider les contrôles de tableau en un seul en veillant à ne pas casser les caractéristiques de chaque contrôle individuel.
- Travaillez avec les autres automaticiens pour déterminer si la consolidation peut être faite et revoir les conventions de nommage pour vous assurer que les contrôles individuels sont nommés de manière appropriée pour refléter leur fonctionnalité

## MERCI

**pour votre participation !**

## BON EXAMEN