

## Saé S1.01 : implémentation d'un besoin client

### Spécification et réalisation des tests

#### Introduction

Dans le processus de développement d'un logiciel, l'activité de **test** du programme est très importante. Elle peut prendre plus de la moitié du temps d'un développeur. Son objectif est de détecter les anomalies du programme, de limiter les erreurs (bugs), d'éviter que le programme plante, etc.

Cette activité est absolument indispensable pour garantir l'exactitude et la robustesse d'un programme.

Il existe plusieurs types de vérification de la qualité d'un logiciel (test de charge, test d'intrusion, revue de code, etc.) mais nous allons nous intéresser dans cette séance aux tests fonctionnels de type boîte noire : cela veut dire que nous allons regarder comment fonctionne le logiciel, comment il assure les services demandés (ça c'est le côté « fonctionnel »), mais sans regarder la manière dont il est codé (c'est le côté « boîte noire »). On va donc vérifier que le programme se comporte comme on le souhaite, c'est-à-dire qu'il donne les résultats attendus mais aussi qu'il réagisse correctement aux situations d'erreur.

Cette séance est divisée en deux parties.

1. La **spécification des tests** qui consiste, avant l'écriture d'un programme, à prévoir et à décrire les différents tests qui devront être réalisés.
2. La **réalisation des tests** qui, une fois que le programme est écrit, consiste à effectuer tous les tests qui ont été spécifiés préalablement et à vérifier si le comportement du programme est conforme au comportement attendu. Si c'est le cas on dit que le test est réussi.

## Partie 1 : la spécification des tests

La spécification des tests se fait avant même que le programme soit écrit. Elle consiste, pour chaque fonctionnalité (ou service) du logiciel :

- à **prévoir** les différentes situations qui pourront arriver lorsque la fonctionnalité sera utilisée, afin que le programme y soit préparé et apporte une réponse adéquate. Attention, il est inenvisageable de tester de manière exhaustive toutes les combinaisons possibles (voir plus loin). On se contentera d'envisager quelques cas « normaux », les cas particuliers, les cas limites, les cas d'erreur, etc. Tout l'art du testeur réside dans le choix de ces cas.
- à **décrire** (=spécifier) les vérifications qu'il faudra faire (plus tard, quand le programme sera écrit) pour chacune de ces situations.

### Test / Cas de test

Revenons sur du vocabulaire employé plus haut. On parle de **test** lorsqu'on évoque une fonctionnalité que l'on souhaite soumettre à des vérifications (une procédure ou une fonction du programme, une action utilisateur, un calcul, etc.).

On parle de **cas de test** pour évoquer une des situations possibles qui peuvent se produire lors de l'exécution de cette fonctionnalité.

Pour un **test**, il y aura donc plusieurs **cas de test**, c'est à dire plusieurs situations à vérifier (cas « normaux », cas d'erreur, etc.).

### Exhaustivité impossible

Un test même bien mené n'est pas une preuve. Le seul moyen de prouver qu'une fonctionnalité est correcte serait de tester toutes les combinaisons possibles (tous les cas possibles) ce qui bien sûr n'est pas réaliste, comme le montre l'exemple suivant.

#### *Exemple*

*Considérons une fonction qui doit calculer la moyenne de deux notes de DS.*

*En supposant que les deux notes sont des entiers, il y aurait 21 valeurs possibles pour la première note et 21 pour la seconde, ce qui donnerait 441 combinaisons possibles c'est-à-dire 441 cas de test (sans parler des cas d'erreur). Alors, ça fait déjà beaucoup, mais c'est encore jouable.*

*Mais si on considère maintenant des notes avec 2 décimales, ce qui est souvent le cas, il y aurait 2001 valeurs possibles pour la première note et 2001 pour la seconde, ce qui donnerait plus de 4 millions de combinaisons à tester. On le voit, ça devient vite problématique...*

Envisager tous les cas n'est donc tout simplement pas possible. On va donc se contenter de choisir de manière judicieuse :

- un ou plusieurs cas « normaux » (ou nominaux),
- des cas particuliers,
- des cas limite,
- des cas d'erreur.

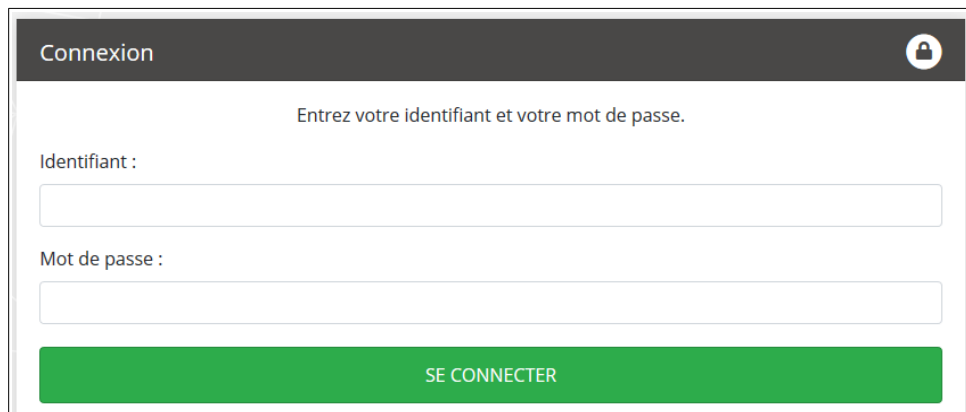
### **Exemple**

Revenons sur la fonction qui doit calculer la moyenne de deux notes de DS. Pour tester cette fonction, on pourra par exemple envisager les cas de test suivants :

- cas « normaux » : (12.5 et 9.3) ou encore (18.56 et 10.07)
- cas particulier : cas où les deux notes sont égales
- cas limite : si l'une des notes (ou les deux) est égale à 0 ; si l'une des notes (ou les deux) est égale à 20
- cas d'erreur : il manque une note ; l'une des notes n'est pas valide, etc.

### **Exercice 1**

Dans le cadre d'un projet de développement d'un logiciel, il est prévu une phase de connexion dont l'interface pourrait ressembler à ceci :



The image shows a login form with a dark header bar containing the title 'Connexion' and a lock icon. Below the header, the instruction 'Entrez votre identifiant et votre mot de passe.' is centered. There are two input fields: 'Identifiant :' and 'Mot de passe :'. At the bottom, there is a green button labeled 'SE CONNECTER'.

On s'intéresse ici à la validation de ce formulaire de connexion, c'est cette fonctionnalité qu'il faudra tester. Quelles sont d'après vous les différentes situations à envisager lorsque l'utilisateur cliquera sur SE CONNECTER ? Autrement dit, quels sont d'après vous les **cas de test** à prévoir ?

## Format de spécification

Il existe plusieurs manières de spécifier c'est-à-dire de décrire les tests. Le format proposé ici a l'avantage d'être simple car il consiste à indiquer, pour chaque test, les informations suivantes :

- *Identification du test*
- *Contexte d'exécution*
- *Étapes de test à effectuer*
- *Pour chaque cas du test :*
  - *Numéro du cas*
  - *Données de test*
  - *Résultats attendus*
  - *Résultats obtenus*
  - *Résultat du test (réussi ou non).*

Le « **contexte d'exécution** » permet de préciser l'état dans lequel se trouvera le programme au début du test.

Les « **étapes de test à effectuer** » indiquent la marche à suivre pour réaliser le test.

Les deux dernières informations « **Résultats obtenus** » et « **Résultat du test** » ne seront connues que lors de la réalisation des tests (voir Partie 2).

### Un exemple de spécification

**Identification du test**      calcul de moyenne de deux notes de DS

**Contexte d'exécution**      exécution complète du programme

**Étapes de test à effectuer**    saisir 2 valeurs réelles

Numéro du cas	Données de test	Résultats attendus	Résultats obtenus	Résultat du test (réussi ou non).
Cas 1	12.54 et 9.35	10.95		
Cas 2	18.0 et 12.0	15.00		
Cas 3	7.5 et 7.5	7.50		
Cas 4	14 et 7	10.50		
Cas 5	0 et 12.2	6.10		
Cas 6	5.4 et 20	12.70		
Cas 7	21.2 et 12.0	message d'erreur		
Cas 8	13.45 et -6.5	message d'erreur		

### **Exercice 2**

On prévoit d'écrire une fonction qui doit calculer et afficher le plus grand de deux entiers saisis par l'utilisateur. Trois tests sont envisagés :

- 1 - test de validité de la première valeur saisie
- 2 - test de validité de la deuxième valeur saisie
- 3 - test du calcul du max

En utilisant le format vu précédemment, spécifiez pour chacun des trois tests les cas de test à prévoir.

### **Cahier de tests**

On appelle **cahier de tests** le document qui rassemble tous les tests concernant un programme.

## Partie 2 : la réalisation des tests

Cette étape intervient lorsque le programme (ou la fonction concernée) est écrit et qu'il est exécutable. Réaliser un test c'est vérifier les cas de tests qui ont été spécifiés lors de l'étape précédente. Il s'agit alors d'exécuter le programme (ou la fonction) autant de fois qu'il y a de cas de test en fournissant à chaque fois les données prévues et en comparant le résultat obtenu (fourni par le programme) avec le résultat attendu. S'il y a concordance, le test a réussi, sinon il a échoué.

Les tests peuvent être réalisés :

- à la main,
- par des fonctions de test à programmer (cf. TP6 d'Algo/Prog sur les fonctions),
- ou à l'aide d'outils spécialisés (Cunit par exemple).

On se contentera dans cette séance d'effectuer les tests « à la main », c'est-à-dire que les données à fournir au programme seront saisies au clavier à chaque exécution.

### Exercice 3

Téléchargez depuis Moodle le programme **moyenne.c**. Compilez-le et réalisez le test prévu dans la partie 1 (8 cas de test sont prévus).

Dans le fichier **TestMoyenne.ods** (disponible sur Moodle), indiquez à chaque fois si le test a réussi ou non.

Que concluez-vous pour ce programme ?

Remarque : ne regardez pas comment le programme est écrit, contentez-vous de le tester (cf. « boîte noire » !).

### Exercice 4

Même exercice avec le programme **variation.c** et le cahier de test **TestVariation.ods**.

### Exercice 5

On souhaite écrire un programme qui calcule la moyenne de  $n$  entiers positifs ou nuls et qui affiche la moyenne de ces  $n$  valeurs. La saisie des valeurs se terminera par -1.

1 – Spécifiez les tests et les cas de test à prévoir pour ce programme.

2 – Écrivez le programme et compilez-le.

3 – Réalisez les tests prévus en 1. Que concluez-vous sur votre programme ?