Test driven development (TDD)

Programmation séquentielle en C, 2020-2021

Orestis Malaspinas (A401), ISC, HEPIA

2021-02-24

Inspirés des slides de F. Glück

Qu'est-ce que le TDD

En deux phrases

- Écrire les tests en premier;
- Le code de l'application ensuite.

Ou encore

- Les spécifications de l'application sont transformées en tests;
- Le code est amélioré pour passer les tests.

D'habitude on fait l'inverse: on code et on écrit les tests après.

Cycle de développement

Une façon de voir le TDD est via la boucle suivante:

1. Écrire un test qui échoue.
2. Écrire juste assez de code pour que le test réussisse.
3. Refactorer.

Et ainsi de suite jusqu'à ce que l'application soit écrite.

Pourquoi utiliser le TDD

Il y a plusieurs avantages au TDD, notamment:

1. Les tests sont complètement intégrés au cycle de développement:
   - plutôt qu'ajoutés (ou pas) à la fin,
   - les tests sont une garantie future.
2. L'assurance que les tests testent les bonnes choses:
   - les fonctionalités au coeur de votre projet sont testées,
   - aide à la documentation.
3. Toutes les fonctionalités sont couvertes:
   - chaque fonctionalité a son/ses tests avant d'être implémentée,
   - ... "en théorie".

Utilisation du TDD

Quand utiliser le TDD

- Quand votre projet est clairement défini (vous savez ce que vous voulez faire).
- Savez assez clairement ce que sont les entrées/sorties.
- Corrigez des bugs.

Quand éviter le TDD

- Quand vous expérimentez (language, algorithme, ...).

Exemple

Librarie de fractions

1. Creation de fraction,
2. Simplification d'une fraction,
3. Addition de deux fraction, ...

TDD: création de fraction

1. Écrire un test:

```
frac a;
frac_init(&a, 4, 5);
assert(a.num == 4 && a.denom == 5);
```

2. Écrire le code

```
typedef struct {
    int num, denom;
}

void frac_init(frac *a, int num, int denom) {
    a->num = num;
    a->denom = denom;
}
```

TDD: simplification de fraction (1/N)

1. Écrire un test:

```
frac a;
frac_init(&a, -4, -5);
assert(a.num == 4 && a.denom == 5);
```

2. Écrire le code

```
void simplify(frac *a) {
    // write the code to simplify for two negative numbers
}

void frac_init(frac *a, int num, int denom) {
    a->num = num;
    a->denom = denom;
    simplify(a)
}
```

TDD: simplification de fraction (2/N)

1. Écrire un test:

```
frac a;
frac_init(&a, 4, -5);
assert(a.num == -4 && a.denom == 5);
```

2. Écrire le code

```
void simplify(frac *a) {
    // refactor for one positive and one negative number
}

void frac_init(frac *a, int num, int denom) {
    a->num = num;
    a->denom = denom;
    simplify(a)
}
```