BAB 8

Test Driven Development

8.1 Tujuan

1. Mahasiswa dapat menjelaskan konsep dasar Test Driven Development (TDD).
2. Mahasiswa dapat mengembangkan kode program menggunakan prinsip TDD.
3. Mahasiswa dapat menulis unit test menggunakan unittest pada Python.

8.2 Pengantar

8.2.1 Kenapa Kita Membutuhkan Test Driven Development?

Bayangkan kalian baru saja membuat sebuah program. Program ini tampaknya berjalan baik. Namun, beberapa hari kemudian, kalian diminta menambahkan fitur baru.

Saat menambahkan fitur ini, tiba-tiba ada bagian lain dari program yang rusak padahal kalian merasa tidak mengubahnya!

Apa yang salah?

Situasi ini sangat umum dalam dunia pengembangan perangkat lunak. Makin besar program, makin banyak bagian yang saling terkait, dan makin sulit untuk mengetahui apakah perubahan kecil merusak bagian lain.Tanpa alat bantu yang sistematis, program menjadi rapuh, tidak terprediksi, dan penuh bug.

Di sinilah ide Test Driven Development (TDD) menjadi relevan.

TDD bukan sekadar teknik testing.

TDD adalah strategi berpikir: kita mengubah cara kita membangun program agar lebih aman, lebih terstruktur, dan lebih siap berkembang.

Tetapi, bagaimana caranya?

Bagaimana mungkin kita menulis kode sebelum menulis program itu sendiri?

8.2.2 Apa Itu Test Driven Development?

Test Driven Development (TDD) adalah sebuah pendekatan pengembangan perangkat lunak di mana kita menulis tes terlebih dahulu, baru kemudian menulis kode yang membuat tes tersebut lulus.

Dengan kata lain:

Kita menentukan lebih dulu apa yang program harus lakukan.

Kita menulis tes untuk menguji apakah program melakukan itu.

Baru kita menulis programnya.

TDD pertama kali dipopulerkan oleh Kent Beck, seorang tokoh utama dalam gerakan Extreme Programming (XP). Dalam bukunya Test-Driven Development: By Example, ia menjelaskan bahwa TDD bukan hanya tentang menguji program, tapi merancang program melalui tes.

Definisi sederhana TDD:

"Tulis tes kecil, lihat tes gagal, buat kode minimum supaya tes lulus, lalu perbaiki kode."

Prinsip TDD terdengar sederhana, tetapi ketika diterapkan secara konsisten, ia mengubah total cara kita berpikir tentang pemrograman.

8.2.3 Prinsip Dasar TDD

Ada tiga aturan sederhana dalam TDD:

1. Tidak boleh menulis kode produksi (kode nyata) kecuali untuk membuat tes lulus.
2. Tidak boleh menulis tes lebih banyak daripada yang diperlukan untuk membuat tes pertama gagal.
3. Tidak boleh menulis kode lebih banyak daripada yang diperlukan untuk membuat tes lulus.

Artinya, kita selalu bergerak dalam siklus berikut:

Tes → Kode → Refaktor → Tes lagi.

TDD mengajarkan kita untuk menulis hanya kode yang dibutuhkan.

Hal ini membuat program menjadi ramping, terstruktur, dan mudah dipelihara.

8.2.4 Alur Kerja TDD: Red → Green → Refactor

TDD biasanya digambarkan dalam siklus tiga tahap, yaitu:

a. Red: Tulis Tes, Lihat Gagal

Pertama, kita menulis tes yang mendeskripsikan fungsionalitas yang diinginkan.

Tes ini tentu saja gagal karena kode untuk membuatnya lulus belum ada.

Contoh:

```
1. def test_penjumlahan():
2.     assert penjumlahan(2, 3) == 5
```

Tetapi fungsi penjumlahan() belum ada → tes gagal.

b. Green: Buat Kode Minimum untuk Membuat Tes Lulus

Kita menulis kode sekecil mungkin supaya tes lulus.

Fokus utama: membuat tes lulus, bukan membuat kode sempurna.

```
1. def penjumlahan(a, b):
2.     return a + b
3.
```

Jalankan tes → tes lulus!

c. Refactor: Perbaiki Kode Tanpa Mengubah Perilaku
Setelah tes lulus, kita boleh memperbaiki struktur kode (refactor).
Misalnya: memperbaiki nama variabel, menghilangkan duplikasi, menyederhanakan logika.
Selama refactor, kita terus jalankan tes untuk memastikan program tetap benar.
Ini adalah siklus TDD:
Red → Green → Refactor → Ulangi.

8.3.4 Tools untuk TDD di Python

Python menyediakan beberapa tools untuk mendukung TDD:

a. unittest

Framework testing bawaan di Python.
Terinspirasi dari JUnit di Java.

Contoh:

```
1. import unittest
2. class TestPenjumlahan(unittest.TestCase):
3.     def test_tambah_dua_angka(self):
4.         self.assertEqual(penjumlahan(2, 3), 5)
5.
6. if __name__ == "__main__":
7.     unittest.main()
```

b. pytest

Framework testing yang lebih modern, lebih sederhana syntax-nya daripada unittest.
Banyak digunakan di industri.
Contoh:

```
1. def test_penjumlahan():
2.     assert penjumlahan(2, 3) == 5
```

c. Mocking dengan unittest.mock

Untuk membuat tes unit tetap sederhana dengan menggantikan objek-objek eksternal.

8.4 Kegiatan Praktikum

8.4.4 Kegiatan 1 : Fungsi Penjumlahan dengan TDD

1. Buat sebuah file **test_add.py** kemudian tulis kode program berikut ini

```
1. import unittest
2. from add_func import add
3.
4. class TestAddFunction(unittest.TestCase):
5.     def test_add_two_numbers(self):
6.         self.assertEqual(add(3, 5), 8)
7.
8. if __name__ == "__main__":
9.     unittest.main()
10.
```

2. Buat sebuah file **add_func.py** kemudian tulis kode program berikut ini

```
1. def add(a, b):
2.     return a + b
3.
```

3. Jalankan file **test_add.py**, Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

8.4.5 Kegiatan 2 : Fungsi Cek Bilangan Ganjil dengan TDD

1. Buat sebuah file **test_odd.py** kemudian tulis kode program berikut ini

```
1. import unittest
2. from odd_func import is_odd
3.
4. class TestOddFunction(unittest.TestCase):
5.     def test_odd_number(self):
6.         self.assertTrue(is_odd(7))
7.
8.     def test_even_number(self):
9.         self.assertFalse(is_odd(8))
10.
11.if __name__ == "__main__":
12.    unittest.main()
```

2. Buat sebuah file **odd_func.py** kemudian tulis kode program berikut ini

```
1. def is_odd(number):
2.     return number % 2 == 1
```

3. Jalankan file **test_odd.py**, Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

8.4.6 Kegiatan 3 : Fungsi Validasi Password dengan TDD

1. Buat sebuah file **test_passwd.py** kemudian tulis kode program berikut ini

```
1. import unittest
2. from passwd_func import is_valid_password
3.
4. class TestPasswordValidation(unittest.TestCase):
5.     def test_short_password(self):
6.         self.assertFalse(is_valid_password("Ab3"))
7.
8.     def test_no_uppercase(self):
9.         self.assertFalse(is_valid_password("password123"))
10.
11.    def test_no_number(self):
12.        self.assertFalse(is_valid_password("Password"))
13.
14.    def test_valid_password(self):
15.        self.assertTrue(is_valid_password("Password123"))
16.
17.if __name__ == "__main__":
18.    unittest.main()
19.
```

2. Buat sebuah file **passwd_func.py** kemudian tulis kode program berikut ini

```
1. def is_valid_password(password):
2.     if len(password) < 8:
3.         return False
4.     if not any(c.isupper() for c in password):
5.         return False
6.     if not any(c.isdigit() for c in password):
7.         return False
8.     return True
```

```
9.
```

3. Jalankan file **test_passwd.py**, Amati hasilnya kemudian tulis analisis singkat tentang kegiatan ini

8.5 Tugas

Gunakan prinsip TDD untuk mengembangkan fungsi celsius_to_fahrenheit(c) yang mengkonversi suhu dari Celsius ke Fahrenheit. Gunakan rumus: F = C * 9/5 + 32. Pastikan fungsi tersebut diuji menggunakan unittest.