14. Refactoring dan Pengujian Perangkat Lunak

Ketika mengembangkan perangkat lunak, sering kali kita menulis kode yang berjalan dengan baik, tetapi seiring waktu, kode tersebut bisa menjadi sulit dipahami, diperbaiki, atau dikembangkan lebih lanjut. Oleh karena itu, diperlukan **refactoring**, yaitu proses merapikan dan meningkatkan struktur kode tanpa mengubah fungsionalitasnya. Selain itu, sebelum sebuah perangkat lunak dirilis, kita juga harus memastikan bahwa kode yang kita tulis benar-benar berjalan dengan baik melalui **pengujian perangkat lunak**.

Pada pertemuan ini, kita akan membahas **teknik refactoring untuk meningkatkan kualitas kode**, bagaimana cara melakukan **pengujian unit menggunakan Jest atau Mocha**, serta menerapkan **studi kasus pengujian perangkat lunak**.

---

1. Teknik Refactoring Kode untuk Meningkatkan Kualitas Perangkat Lunak

Apa itu refactoring?
Refactoring adalah **proses mengubah struktur internal kode tanpa mengubah fungsionalitas eksternalnya**. Tujuan refactoring adalah **menjadikan kode lebih bersih, mudah dipahami, dan mudah diperbaiki**.

Kenapa refactoring penting?

- **Meningkatkan keterbacaan kode** → Kode menjadi lebih rapi dan mudah dipahami oleh developer lain.
- **Mempermudah pemeliharaan** → Kode yang bersih lebih mudah diperbaiki dan dikembangkan.
- **Meningkatkan performa aplikasi** → Dengan mengoptimalkan logika kode, aplikasi bisa berjalan lebih cepat dan efisien.

Contoh Refactoring Kode:
Kode sebelum refactoring:

```
function hitungTotalHarga(barang) {
    let total = 0;
    for (let i = 0; i < barang.length; i++) {
        total += barang[i].harga * barang[i].jumlah;
    }
    return total;
}
```

Kode setelah refactoring dengan **higher-order function (`reduce`)**:

```
function hitungTotalHarga(barang) {
    return barang.reduce((total, item) => total + item.harga * item.jumlah, 0);
}
```

Perbedaan utama:

- **Kode lebih ringkas** → Menggunakan `reduce()` menggantikan perulangan manual.
- **Lebih mudah dibaca dan dipahami** → Struktur kode lebih bersih dan ekspresif.

---

2. Pengujian Unit Menggunakan Jest/Mocha

Apa itu pengujian unit?
Pengujian unit adalah **pengujian pada bagian kecil dari kode (biasanya fungsi atau kelas) untuk memastikan ia berfungsi dengan benar**.

Kenapa kita butuh pengujian unit?

- **Mencegah bug sejak dini** → Sebelum kode dirilis, kita bisa menemukan kesalahan lebih cepat.
- **Membantu refactoring dengan aman** → Kita bisa merapikan kode tanpa takut mengubah fungsionalitasnya.
- **Meningkatkan kepercayaan terhadap kode** → Dengan pengujian otomatis, kita lebih yakin bahwa kode berjalan sesuai harapan.

Menggunakan Jest untuk Pengujian Unit
Jest adalah framework pengujian yang populer untuk JavaScript. Contoh pengujian sederhana menggunakan Jest:

Install Jest terlebih dahulu:

```
npm install --save-dev jest
```

Buat file `math.js` dengan fungsi sederhana:

```
function tambah(a, b) {
    return a + b;
}

module.exports = tambah;
```

Buat file `math.test.js` untuk pengujian:

```
const tambah = require("./math");

test("Menjumlahkan 2 + 3 harus menghasilkan 5", () => {
   expect(tambah(2, 3)).toBe(5);
});
```

Jalankan pengujian dengan Jest:

```
npx jest
```

Hasil yang diharapkan:
Jika semua pengujian lulus, berarti fungsi bekerja dengan benar.
Jika ada kesalahan, kita harus memperbaiki kode sebelum lanjut ke tahap berikutnya.

---

3. Studi Kasus Pengujian Perangkat Lunak

Kasus: Validasi Input Formulir
Misalkan kita punya fungsi JavaScript yang memvalidasi apakah email yang dimasukkan pengguna valid atau tidak:

```
function validasiEmail(email) {
   return /^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(email);
}
```

Kita bisa membuat pengujian untuk memastikan fungsi ini bekerja dengan benar:

```
const validasiEmail = require("./validasiEmail");

test("Email valid harus dikembalikan sebagai true", () => {
   expect(validasiEmail("user@example.com")).toBe(true);
});

test("Email tanpa @ harus dikembalikan sebagai false", () => {
   expect(validasiEmail("userexample.com")).toBe(false);
});
```

Manfaat dari pengujian ini:

- Memastikan **hanya email yang valid yang diterima**.
- Mencegah **bug** yang bisa terjadi jika pengguna memasukkan email tidak sesuai format.

Kesimpulan

Di pertemuan ini, kita telah membahas **pentingnya refactoring untuk meningkatkan kualitas kode**, bagaimana melakukan **pengujian unit menggunakan Jest**, serta **contoh studi kasus pengujian perangkat lunak**.

Takeaways:
Refactoring membuat kode lebih bersih dan efisien.
Pengujian unit membantu mendeteksi bug sebelum kode dirilis.
Menggunakan Jest/Mocha bisa membuat proses pengujian lebih cepat dan otomatis.