Kelompok 1

Test-Driven Development (TTD)

Anggota Kelompok

- ROFIFAH GINA TAMALA
- NUR AZIZAH
- AISKA SUCI RAHMADANI SAMIN
- SITTI HADIJAH

KAJIAN TEORITIS

- ***Definisi dan Sejarah Test Driven Development (TDD)***

  Test Driven Development (TDD) adalah metode pengembangan perangkat lunak dengan prinsip test-first approach, yaitu menulis tes sebelum membuat kode sehingga kualitas terjaga sejak awal. Konsep ini diperkenalkan Kent Beck (2002) dalam bukunya Test Driven Development: By Example dan menjadi bagian dari Extreme Programming (XP) dalam kerangka Agile. Penelitian terbaru, seperti Rafiadly dkk. (2023), membuktikan TDD efektif diterapkan di aplikasi nyata dengan hasil uji 100%.

Penerapan Model (Aktivitas/Fase TDD)

Penerapan TDD biasanya mengikuti siklus Red – Green – Refactor:

1. Red (Test Fails): Pengembang menulis tes terlebih dahulu, hasil awal pasti gagal karena kode belum dibuat.
2. Green (Test Passes): Menulis kode minimum agar tes lulus. Tujuannya hanya membuat tes berjalan, tanpa memperhatikan efisiensi.
3. Refactor: Setelah tes berhasil, kode disempurnakan agar lebih rapi, efisien, dan mudah dipelihara.

Siklus ini berulang secara iteratif untuk setiap fitur kecil hingga membentuk sistem lengkap. Pada penelitian Siraj & Setiani (2025), TDD diterapkan pada aplikasi Sedonor untuk meningkatkan kualitas kode. Hasilnya menunjukkan penurunan signifikan pada metrik kompleksitas seperti Lines of Code (LOC) dan Response for Class (RFC), membuktikan bahwa siklus TDD mampu menghasilkan kode yang modular dan mudah diuji

Karakteristik Model TDD

Beberapa karakteristik utama TDD antara lain:

- Iteratif dan Incremental: pengembangan dilakukan dalam potongan kecil yang diuji secara bertahap.
- Berorientasi pada Unit Test: setiap fungsi diuji secara individual.
- Refactoring Berkelanjutan: kode terus diperbaiki agar tetap bersih tanpa mengubah fungsionalitas.
- Dokumentasi Otomatis: unit test berfungsi sebagai dokumentasi perilaku sistem.
- Meningkatkan Modularitas: kode lebih mudah dipelihara dan diuji kembali ketika ada perubahan.

Karakteristik ini terbukti dalam penelitian Sedonor, di mana penerapan TDD membuat kode lebih modular dan efisien sehingga lebih mudah dikembangkan ke tahap selanjutnya

Kelebihan dan Kelemahan TDD

Kelebihan TDD:

1. Mengurangi Bug Sejak Awal: Karena pengujian ditulis lebih dulu, kesalahan dapat terdeteksi sebelum kode berkembang lebih jauh
2. Kode Lebih Modular: Penelitian Sedonor menunjukkan modularitas meningkat dengan turunnya metrik LOC hingga 91,9% [3]
3. Mendukung Refactoring Aman: Perubahan kode dapat dilakukan tanpa khawatir merusak fungsionalitas.
4. Dokumentasi Otomatis: Unit test menjadi dokumentasi yang jelas tentang cara kerja kode.
5. Meningkatkan Kepercayaan Diri Programmer: Setiap perubahan langsung bisa diuji dengan cepat.

Kelemahan TDD:

1. Membutuhkan Waktu di Awal: Penulisan tes sebelum kode menambah waktu pengerjaan awal proyek
2. Tidak Mudah pada Legacy Code: Sulit diterapkan pada kode lama yang tidak modular.
3. Butuh Keahlian Tinggi: Pengembang harus paham prinsip pengujian dengan baik.
4. Tidak Selalu Cocok untuk Deadline Ketat: Karena waktu awal pengerjaan bisa lebih lama.

Perbandingan dengan Model atau Metodologi Lain

1. TDD vs Waterfall
    - Waterfall bersifat linear dan kaku, pengujian dilakukan di akhir.
    - TDD bersifat iteratif, pengujian dilakukan sejak awal.
    - TDD lebih fleksibel terhadap perubahan kebutuhan dibanding Waterfall.
2. TDD vs BDD (Behavior Driven Development)
    - TDD berfokus pada implementasi kode dan unit test yang ditulis oleh developer.
    - BDD menekankan perilaku sistem dalam bahasa alami, mudah dipahami oleh stakeholder.
    - BDD meningkatkan komunikasi tim, sementara TDD lebih teknis
3. TDD vs ATDD (Acceptance Test Driven Development)
    - TDD fokus pada level unit test.
    - ATDD menekankan kolaborasi tim (developer, tester, user) untuk mendefinisikan tes penerimaan.
    - ATDD lebih berorientasi pada kebutuhan pengguna

Dari perbandingan ini, jelas bahwa TDD unggul dalam deteksi bug sejak awal, tetapi BDD dan ATDD unggul dalam aspek kolaborasi dan komunikasi. [1]

Alat Bantu Model atau Metode TDD

Dalam praktiknya, TDD memerlukan dukungan alat bantu untuk menulis dan menjalankan tes otomatis. Beberapa framework populer antara lain:

- JUnit (Java) – digunakan luas pada pengembangan berbasis Java dan Android
- Mockito & MockK – untuk membuat objek tiruan dalam pengujian.
- PyTest, Unittest – untuk Python.
- Jest, Mocha – untuk JavaScript/Node.js.
- NUnit – untuk C#.

Selain itu, integrasi dengan Continuous Integration (CI) tools seperti Jenkins, GitHub Actions, atau GitLab CI semakin memperkuat praktik TDD karena setiap perubahan kode langsung diuji secara otomatis.

Penutup

Berdasarkan pembahasan yang telah dilakukan, dapat disimpulkan bahwa Test Driven Development (TDD) merupakan pendekatan pengembangan perangkat lunak yang efektif dalam meningkatkan kualitas, modularitas, serta reliabilitas sistem sejak tahap awal. Prinsip test-first approach menjadikan proses pengujian sebagai fondasi utama sehingga bug dapat diminimalkan, kode lebih ringkas, dan proses refactoring dapat dilakukan dengan aman.

Data empiris dari penelitian Naviku dan Sedonor membuktikan bahwa TDD mampu memberikan hasil nyata, baik dari sisi keberhasilan pengujian maupun penurunan kompleksitas kode. Selain itu, perbandingan dengan metodologi lain menunjukkan bahwa TDD unggul dalam aspek teknis, sementara BDD dan ATDD lebih menekankan kolaborasi serta komunikasi dengan stakeholder.

Dengan demikian, TDD tidak hanya dipandang sebagai teknik pengujian, tetapi juga sebagai paradigma baru dalam pengembangan perangkat lunak modern yang menekankan kualitas, keberlanjutan, dan efisiensi.