Rakamin Academy

Reading Material

Memahami Testing Principles - Test Driven Development (TDD)

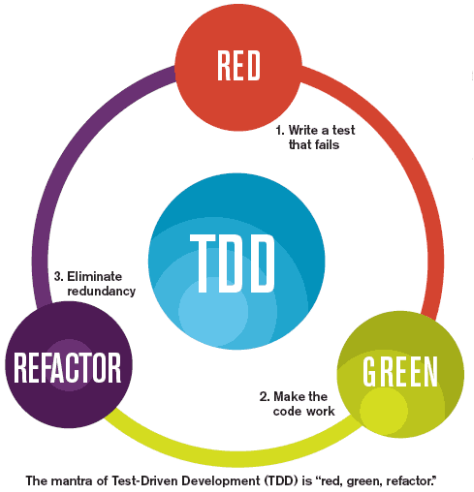
Rakamin Academy

READING

Test Driven Development

Definisi

Teknik pengembangan perangkat lunak di mana test cases dikembangkan, otomatisasi (automated), dan kemudian aplikasi akan dikembangkan secara bertahap untuk melewati test cases tersebut.



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

TDD Cycle

TDD memiliki 3 cycle utama yang saling berhubungan satu sama lain yaitu Red, Green dan Refactor. Berikut penjelasan lebih lengkap untuk 3 cycle:

1. RED, proses ketika kita menulis semua test cases ke dalam sebuah script automation kemudian menjalankan test cases tersebut dengan ekspektasi gagal.
2. GREEN, proses membuat code dari aplikasi dan kemudian menjalankan test cases dengan ekspektasi berhasil.
3. REFACTOR, proses ketika developer melakukan refactor code ketika test cases yang berkaitan sudah passed.

Gambaran proses yang dijalankan cukup sederhana antara lain:

1. Tulis semua test cases dalam bentuk script automation
2. Kemudian jalankan test cases tersebut dan pastikan mereka gagal
3. Tulis kode aplikasi untuk membuat test cases tersebut berhasil
4. Jalankan lagi test cases
5. Kemudian refactor kode di step ke 3
6. Ulangi step 1 sampai 5 dengan penambahan kode aplikasi dan test cases.

Bagaimana? Sederhana kan? Tapi terlihat lambat jika diimplementasikan penuh ke dalam sebuah development app dengan jangka waktu yang pendek. Sebaliknya, proses tersebut akan sangat membantu untuk sebuah project development application dengan jangka waktu panjang, mengingat sumber daya manusia masa kini sering silih berganti kurang dari 3 bulan maka membuat test cases dan TDD bisa mendisiplinkan proses development aplikasi.

Kenapa Menggunakan TDD?

Dengan menggunakan TDD maka kita akan memecah kode menjadi potongan-potongan logic yang mudah dipahami, hal nii membantu kita untuk memastikan ketepatan kode yang kita buat. Memecah kode menjadi potongan logic akan menjadi penting karena ada beberapa hal yang menjadi sulit ketika mengembangkan sebuah aplikasi:

1. Menyelesaikan bug atau suatu masalah yang komplek dalam sekaligus dalam satu waktu
2. Mengetahui kapan dan dimana harus mulai memperbaiki sebuah bug atau masalah
3. Menambah tingkat kompleksitas dari sebuah code script tanpa menambah error atau bug
4. Menyadari kapan sebuah code akan mengalami issue

Dari ke 4 hal tersebut sebetulnya sudah sangat cukup untuk mengambil keputusan perlu tidaknya menggunakan TDD. Perlu digaris bawahi bahwa TDD tidak 100% menjadi jaminan bahwa kode aplikasi akan terbebas dari bug atau error, seperti prinsip test bahwa tidak ada aplikasi yang 100% bebas bug. Namun TDD akan membantu kita untuk menulis kode dengan lebih baik, logic yang terstruktur sehingga membuat kita paham dengan apa yang sedang kita kerjakan.

Keuntungan dan Tantangan Menggunakan TDD

TDD mendorong penulisan kode yang dapat diuji, membuat loosely-coupled code yang berpotensi menjadi kode yang lebih modular. Karena terstruktur dengan baik, kode modular lebih mudah untuk ditulis, di-debug, dipahami, dipelihara, dan digunakan kembali. TDD membantu:

1. Mengurangi biaya produksi.
2. Membuat refaktor atau refactoring dan penulisan ulang lebih mudah dan cepat karena ada prinsip. Red dan Green flags memastikan kode aplikasi berjalan dengan baik, Refactor memastikan kode aplikasi ditulis dengan benar.
3. Mencegah bug, karena penulisan kode sudah diingiri dengan test.
4. Meningkatkan kolaborasi team.
5. Menambah tingkat percaya diri bahwa kode akan berjalan sesuai ekspektasi.
6. Meningkatkan kualitas code patterns.
7. Mengurangi rasa cemas atau takut terhadap perubahan kode sewaktu-waktu.

TDD juga mendorong peningkatan terus menerus. Ini akan memperlihatkan area dari kode yang perlu ditingkatkan sehingga membantu meningkatkan kualitas dan desain aplikasi secara keseluruhan.

Selain kelebihan dari penerapan TDD, ada beberapa tantangan yang mungkin harus dihadapi ketika menjalankan TDD antara lain:

1. Untuk individual mungkin akan ada suatu waktu dimana kita lupa untuk menjalankan test secara teratur, menulis test yang banyak dalam satu waktu, menulis test yang terlalu besar.
2. Sebagai team mungkin ada suatu waktu dimana team harus menghasilkan aplikasi dalam waktu singkat dan untuk memotong waktu maka TDD ditiadakan terlebih dahulu, sebetulnya hal ini tidak ideal karena akan merusak alur TDD yang sudah dibangun sebelumnya.

Tools TDD

TDD merupakan sebuah konsep dalam pengembangan aplikasi, konsep ini kemudian diadaptasi oleh berbagai tool yang akan membantu pengembang antara lain:

1. JUnit

   Digunakan dalam bahasa pemrograman Java dan berjalan di atas Java Virtual Machine

(JVM). JUnit menawarkan API TestEngine yang bisa digunakan untuk mengembangkan framework test pada JVM. Selain itu JUnit memiliki fitur Console yang memungkinkan untuk menjalankan test dari CLI.

2. Mockito

   Membuat framework mocking (tiruan) sangat penting untuk TDD project. Hal ini memungkinkan kita dan tim DevOps membuat objek tiruan untuk pengujian. Contohnya kita dapat membuat mock atau tiruan database, web service yang kemudian dipakai sebagai bahan test.

3. Pytest

   Pytest merupakan kerangka kerja pengujian di bahasa pemrograman Python. Kita dapat menulis test cases, menguji database dan API. Salah satu fiturnya adalah python memiliki 1000+ built in function yang akan sangat membantu dalam proses penulisan kode dan testing.