BAB III
LANDASAN TEORI

3.1. Pengujian Perangkat Lunak

Pengujian perangkat lunak atau dikenal dengan istilah *software testing* merupakan suatu pengujian yang dilakukan dalam menguji kualitas perangkat lunak atau *software* untuk memenuhi ekspetasi penggunaan dari sisi *customer* / *end-user* [18]. Terdapat monografi pertama yang membahas mengenai pengujian perangkat lunak dan menyatakan bahwa pengujian merupakan proses mencari kesalahan atau proses sistematis saat program dijalankan [19]. Secara teknis berdasarkan standar ANSI/IEEE 1059, pengujian perangkat lunak adalah proses untuk menganalisis suatu *item* pada perangkat lunak dan mendeteksi perbedaan antara kondisi yang ada dengan kondisi yang diharapkan [20].

Bahasa pemrograman semakin bervariasi, namun pengujian perangkat lunak dapat dikatakan lebih mudah dilakukan karena karakteristik *software* dan *operation system* memiliki teknologi yang lebih mutakhir. Berdasarkan definisi yang sudah dijabarkan, pengujian perangkat lunak merupakan fase penting dalam *Software Development Life Cycle* (SDLC). Tujuannya untuk menemukan kesalahan saat pengembangan perangkat lunak semaksimal mungkin, sehingga memenuhi standar kualitas produk sebelum tahap peredaran. Pengujian dikatakan berhasil apabila dapat memvalidasi dan memverifikasi antara spesifikasi dengan kebutuhan dari *user*. Aspek tersebut memastikan kualitas perangkat lunak sudah terpenuhi dengan baik selama pengembangan produk [21].

Berdasarkan kategorinya, pengujian perangkat lunak dibagi menjadi dua bagian, yaitu pengujian statis dan pengujian dinamis. Pengujian statis mereferensikan pengujian yang dilakukan secara manual tanpa harus mengeksekusi *code* dan dikenal juga dengan istilah *verification testing*. Penerapan metode ini dilakukan pada dokumentasi seperti *Software Requirement Specification* (SRS), berkas desain, *source code*, test suites, dan halaman konten *website*. Dengan kata lain, pengujian statis melakukan evaluasi terhadap pengkodingan yang sudah

terdokumentasikan sebelum *code* diterapkan. Pengujian dinamis mereferensikan pengujian dengan validasi terhadap sistem yang dipertimbangkan dengan menyasar pada perilaku dinamis suatu pengkodingan. Metode ini berdasarkan analisis nilai masukan dan nilai keluaran pada perangkat lunak. Secara progresif, pengujian dibagi menjadi *black box testing*, *white box testing*, dan *gray box testing* [22].

3.1.1. *Black Box Testing*

Gambar 3. 1. Representasi *Black Box Testing*

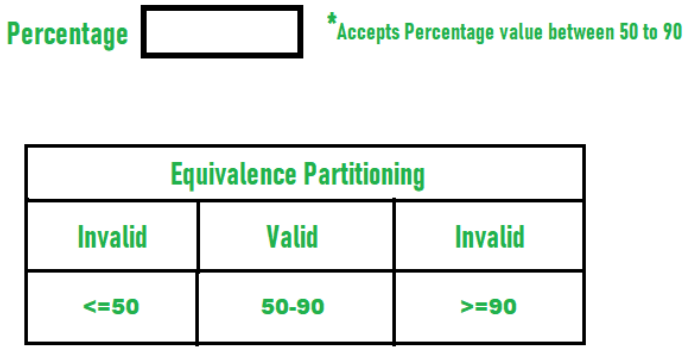
Black box testing merupakan suatu metode pengujian yang berorientasi pada fungsi-fungsi yang ada pada program. Dengan kata lain, *black box testing* merupakan suatu metodologi yang didasarkan untuk mengamati hasil dari berbagai nilai masukan tanpa melakukan analisis pengkodingan sama sekali. Pengujian ini memastikan analisis nilai masukan dan nilai keluaran sudah tepat. Gambar 3.1 merupakan ilustrasi dari cara kerja *black box testing*. Metode ini bertujuan untuk memenuhi kriteria berdasarkan kebutuhan dengan fungsionalitas dari perangkat lunak. Oleh karena itu, *black box testing* dapat disebut juga sebagai *functional testing* [23]. *Black box testing* memiliki keunggulan terkait pengujian yang mengambil sudut pandang kebutuhan dari *user* dan efisien jika diterapkan pada sistem besar. Selain itu, pengujian dapat dilakukan oleh pihak ketiga dan non-teknis sekalipun karena kemampuan pemrograman tidak diperlukan dalam metode ini [24].

Penerapan metode *black box testing* tidak lepas dari teknik-teknik pengujian yang dilakukan pada suatu perangkat lunak. Dilansir dari studi perbandingan, teknik-teknik *black box testing* yang umum digunakan adalah *pairwise testing* (PWT), *equivalence class partitioning* (ECP) atau *equivalence partitioning* (EP), *boundary value analysis* (BVA), dan *decision table testing* (DTT). Teknik-teknik yang dianggap paling sederhana, efisien, dan murah dalam penggunaannya adalah

EP dan BVA. Hal ini terkait teknik-teknik pengujian tersebut menerapkan analisis pada nilai masukan dan nilai keluaran untuk memastikan fungsi sudah berjalan dengan semestinya [25].

3.1.2. *Equivalence Partitioning*

Equivalence partitioning merupakan suatu teknik pengujian yang berorientasi pada nilai masukan dan nilai keluaran yang dihasilkan pada suatu fungsi. Istilah *equivalence partitioning* mulai dikenal dari gagasan bahwa masukan dari suatu program dapat dilakukan partisi pada kelas-kelas setara (*equivalent*). Hal ini dilakukan untuk melakukan pengujian perangkat lunak berdasarkan segala nilai masukan akan menguji kelas secara keseluruhan. Dalam penerapannya, beberapa penelitian menjelaskan bahwa dengan segala nilai masukan dapat menemukan kesalahan fungsionalitas yang ada pada domain. Sebagai contoh, nilai masukan dapat berupa nilai negatif atau memiliki nilai lebih dari sama dengan nol [26]. Nilai masukan tersebut nantinya akan menghasilkan nilai keluaran dengan kategori *valid* atau *invalid* berdasarkan spesifikasi fungsionalitas yang sudah ditentukan dari kebutuhan *user*. Gambar 3.2 merupakan representasi cara kerja *equivalence partitioning*.



Gambar 3. 2. Representasi *Equivalence Partitioning*

Implementasi teknik *equivalence partitioning* dilakukan dengan membagi kelas-kelas dari nilai masukan ke dalam rancangan *test case*. Kategori *valid* atau *invalid* akan didapatkan berdasarkan kendala masukan yang terjadi pada fungsi.

Pengujian yang dilakukan cukup dilakukan satu kondisi dari setiap kelas [27]. Hal ini disebut sebagai *Weak Equivalence Class Testing* (WECT). Selain WECT, terdapat *Strong Equivalence Class Testing* (SECT) yang mana *test case* berisi pengujian terhadap seluruh interaksi dari semua kelas. Keunggulan *equivalence partitioning* adalah memberikan sensasi pengujian yang lengkap, mencakup nilai masukan dan nilai keluaran secara menyeluruh dengan kelas yang lebih kecil, serta terhindar dari redundansi pengujian [28].

3.1.3. Pengujian Manual

Pengujian manual atau *manual testing* adalah pengujian perangkat lunak yang dilakukan tanpa *tool* khusus. Definisi lainnya, pengujian manual merupakan suatu proses yang digunakan untuk memeriksa desain dari sistem dan identifikasi kesalahan-kesalahan yang ada berdasarkan *test case* [29]. Dengan kata lain, pengujian manual dijalankan berdasarkan skenario pengujian yang sudah dibuat pada *test case*, kemudian membandingkan nilai keluaran sesungguhnya dengan ekspetasi nilai keluaran sesuai rancangan kebutuhan *user*. Pengujian secara manual memiliki durasi pengerjaan yang cukup lama dan biaya lebih besar dibandingkan metode pengujian yang lain. Hal ini wajar mengingat pengujian manual dilakukan secara langsung oleh sumber daya manusia. Keterbatasan waktu dan kemampuan dari sumber daya manusia dapat mempengaruhi proses berlangsung hingga hasil dari pengujian. Namun, pengujian manual sangat efektif untuk mendeteksi kesalahan logika yang ada pada perangkat lunak [30].

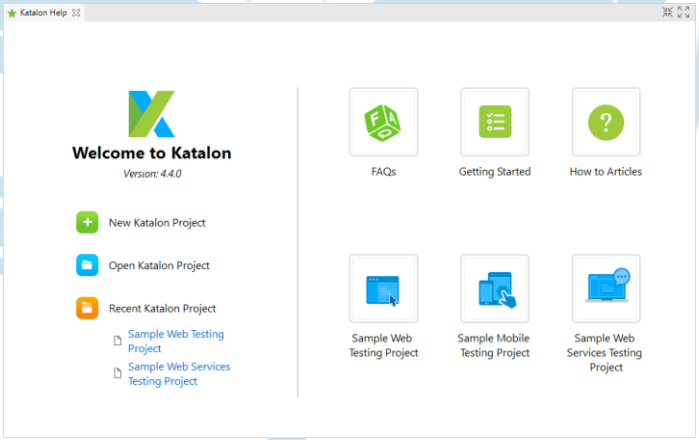
3.1.4. Pengujian Otomatis

Pengujian otomatis atau *automation testing* adalah pengujian perangkat lunak yang dilakukan dengan *automation tool*. Metode ini merupakan solusi dalam meminimalkan pekerjaan yang repetitif [31]. Pengujian otomatis berperan penting dalam SDLC dengan pendekatan *agile development* yang serba cepat. Tujuannya untuk mempercepat siklus pengujian dengan mengurangi upaya dalam menjalankan pengujian manual [32]. Hal ini menjelaskan pengujian otomatis dapat disebut pengujian dinamis. Pada saat eksekusi *test case* akan ada laporan *bug* dari *tool*. Jika

tidak menghasilkan *bug*, maka iterasi akan selesai dan *backlog* pengujian berakhir [33]. Namun, efektivitas penerapan pengujian otomatis berdasarkan penentuan *tool* dan pemilihan *test case* [34].

Pengujian otomatis dapat digunakan untuk pengujian fungsionalitas yang menguji fungsi-fungsi sesuai dengan kebutuhan *user*. Hal ini terkait juga dengan keunggulan pengujian otomatis, yaitu dapat dilakukan *regression testing*, kecepatan eksekusi, hemat waktu, dan dapat dibuat oleh seluruh anggota tim. Namun, dengan mengandalkan *tool* saja, pengujian otomatis belum dapat dikatakan sempurna. Artinya, peran sumber daya manusia secara langsung saat pengujian masih sangat dibutuhkan dalam memenuhi kualitas perangkat lunak yang beredar. Saat menerapkan pengujian otomatis, perlu adanya *support* & *maintenance* supaya tetap terus menjaga kualitas pengujian. *Automation tool* yang sering digunakan adalah Selenium, Katalon Studio, Unified Functional Testing, Testcomplete, dan Watir [35].

3.2. Katalon Studio



Gambar 3. 3. Tampilan Katalon Studio

Gambar 3.3 merupakan tampilan aplikasi Katalon Studio yang merupakan salah satu *automation tool* untuk pengujian perangkat lunak. *Automation tool* ini diciptakan pada tahun 2015 oleh Katalon Inc. dan dibangun dengan *frameworks open-source* Selenium dan Appium, sehingga didukung berbagai sistem operasi. Meskipun baru rilis dibandingkan *automation tool* lain, Katalon Studio

menawarkan fitur lengkap dalam implementasi pengujian otomatis untuk aplikasi *web*, API, *desktop*, dan *mobile*. Katalon Studio memungkinkan *team project* dengan mengurangi upaya dan keahlian yang diperlukan mempelajari dan integrasi untuk kebutuhan pengujian perangkat lunak secara otomatis. Keunggulan dalam penggunaan Katalon Studio adalah tidak adanya biaya terkait lisensi dan pemeliharaan aplikasi, integrasi fitur-fitur yang diperlukan untuk pembuatan dan eksekusi *test case*, serta mudah untuk digunakan oleh pemula sekali pun [36]. Pembuatan test case pada Katalon Studi dapat dilakukan dengan beberapa cara, yaitu *record and replay*, *manual mode*, dan *script mode* [37].

3.3. *Techno Expertise Academy* (TEA)

Gambar 3. 4. Logo Aplikasi '*Quest Master* & *Techno Expertise Academy*'

Techno Expertise Academy (TEA) merupakan sebuah sistem yang dibangun oleh perusahaan Astra Credit Companies dan merupakan bagian dari '*Quest Master* & *Techno Expertise Academy*' (QMTEA). Gambar 3.4 merupakan gambar logo aplikasi QMTEA yang saat ini sedang dikembangkan. TEA adalah sistem yang digunakan sebagai media pembelajaran *hard skill* terkait bahasa pemrograman, *software*, dan *tools* yang digunakan oleh Astra Credit Companies dalam pengembangan perangkat lunak. Dilansir dari pihak *Technocenter* selaku *product owner* dan *user* dari aplikasi QMTEA, konsep aplikasi ini terinspirasi dari *platform*

SoloLearn yang merupakan *e-learning* untuk mempelajari pemrograman dari tingkat pemula hingga mahir [38]. Aplikasi ini nantinya akan digunakan oleh para peserta *bootcamp*, *internship*, *freelancer*, dan berbagai pihak lain yang terafiliasi dengan *Technocenter* dalam bidang teknologi informasi. Dalam pengembangannya, aplikasi QMTEA menerapkan konsep gamifikasi. Gamifikasi adalah suatu implementasi konsep-konsep umum yang ada di dalam *game* yang diintegrasikan kedalam pekerjaan dan interaksi sosial manusia [39]. Dengan menerapkan konsep gamifikasi pada aplikasi QMTEA, para *user* dapat berkompetisi dan berinteraksi sehingga betah dan terpacu dalam meningkatkan *hard skill*.