

BAB II

TINJAUAN PUSTAKA

2.1 Studi Sebelumnya

Pada penelitian sebelumnya "*Automation Regression Testing* pada Aplikasi Teman Diabetes Dengan Menggunakan Metode *Black Box Testing*" oleh Yoanna Fransisca Putri. Penelitian ini membandingkan efektivitas dan efisiensi antara *manual operational acceptance testing* dan *automation regression testing* dalam proses pengujian aplikasi Teman Diabetes. Pada metode *manual operational acceptance testing*, dibutuhkan waktu selama dua minggu untuk menguji 72 *test case* dan berhasil menemukan 7 *bugs* baru. Sementara itu, *automation regression testing* dilakukan menggunakan Katalon Studio memerlukan waktu eksekusi hanya 1 jam 1 menit 44 detik untuk 89 *test case*, dan menghasilkan 4 temuan *bugs*. Hasil penelitian menunjukkan bahwa *automation regression testing* secara signifikan lebih efisien dari segi waktu pengujian, meskipun jumlah *bugs* yang ditemukan sedikit lebih rendah dibandingkan metode *manual* [5].

Penelitian selanjutnya berjudul "Analisis Perbandingan Pengujian *Manual* dan *Automation Testing* pada Website E-Commerce" oleh Dewandra Sapto Prasetyo dan Widya Silfianti. Studi ini memfokuskan pengujian pada halaman-halaman yang umum diakses oleh pengguna dalam proses pemesanan produk, seperti halaman *login*, pencarian produk, *detail* produk, keranjang belanja, hingga *logout*. Pengujian *manual* dilakukan dengan menjalankan 10 *test case* secara langsung menggunakan Microsoft Edge Chromium, mengikuti langkah-langkah yang telah ditentukan dalam skenario pengujian. Waktu eksekusi diukur mulai dari saat pengguna membuka situs web dan melakukan *login* hingga proses *logout* dan menutup situs selesai dilakukan. Sementara itu, pengujian otomatis dilakukan menggunakan *tools* Katalon Studio dengan memasukkan seluruh *test case* ke dalam *test suite*, kemudian dijalankan secara otomatis. Kedua metode pengujian diulang sebanyak lima kali (lima iterasi). Hasil pengujian menunjukkan bahwa

waktu eksekusi *automation testing* secara signifikan lebih cepat dibandingkan dengan pengujian *manual*. Untuk satu *test case*, waktu tercepat dengan pengujian otomatis adalah 28,2 detik dan waktu terlama 65,6 detik. Sedangkan pada pengujian *manual*, waktu tercepat adalah 68 detik dan waktu terlama mencapai 74,4 detik [8].

Theresia *et al.* melakukan studi mengenai penerapan *Automation Testing* menggunakan pendekatan *Black Box* dengan teknik *Boundary Value Analysis* dan *Equivalence partitioning* pada aplikasi mobile ACC Leads. Tujuan dari penelitian ini adalah untuk menggantikan pengujian *manual* yang memakan waktu dengan pengujian otomatis, serta menilai kemampuan ACC Leads dalam menampilkan peringatan guna mencegah pengguna menginput data dengan tipe atau panjang yang tidak sesuai. Hasil penelitian menunjukkan bahwa sistem masih belum sepenuhnya mampu memberikan peringatan terhadap panjang dan tipe data yang tidak *valid* [9].

Penelitian selanjutnya berjudul “Otomatisasi Pengujian Perangkat Lunak Menggunakan Metode *Data Driven* untuk Meningkatkan Efisiensi Waktu Pengujian” oleh Margaret Teacher Banjarnahor dan Lucia Sri Istiyowati. Penelitian ini bertujuan untuk mengevaluasi efektivitas proses otomatisasi pengujian perangkat lunak melalui penerapan *automation script* menggunakan Katalon Studio dengan pendekatan *data driven testing*. Fokus utama penelitian adalah untuk melihat sejauh mana metode ini mudah dipahami oleh tim *Software Quality assurance* (SQA). Penelitian dilakukan pada tim SQA Peer To Landing tanpa latar belakang pengalaman dalam *automation testing*. Tim diminta untuk mengonversi seluruh *test case* yang telah dibuat ke dalam bentuk *test suite*, kemudian mengeksekusinya menggunakan Katalon Studio. Hasil evaluasi berdasarkan penilaian responden menunjukkan skor pencapaian sebesar 80,21%, yang dikategorikan dalam rentang "sangat baik" (80%–100%). Temuan ini menunjukkan bahwa *automation testing* yang dikembangkan dengan Katalon Studio dan

metode data *driven testing* tergolong sangat baik dalam hal kemudahan penggunaan dan dapat dipelajari dengan cepat [10].

Penelitian selanjutnya “Evaluasi Penggunaan *Manual* dan Automated Software Testing pada Pelaksanaan End-To-End Testing” oleh Min *et al.* Penelitian ini masih membandingkan efektifitas dan efisiensi dari *manual* dan *automation testing* namun menggunakan *tools* yang berbeda yaitu Protractor. Berbeda dengan Katalon, Protractor ditujukan bagi *tester* yang sudah familiar dengan teknologi seperti AngularJS dan JavaScript. Dalam penelitian ini, *tester* dibagi menjadi dua kelompok: satu melakukan end-to-end *testing* secara *manual*, dan yang lain menggunakan *automation testing* dengan Protractor. Masing-masing kelompok melakukan dua iterasi pengujian: iterasi pertama pada sistem sebelum *bug fixing*, dan iterasi kedua setelah *bug* diperbaiki. Pada iterasi pertama, *automation* membutuhkan waktu lebih lama karena harus membuat skrip pengujian terlebih dahulu. Namun, pada iterasi berikutnya, waktu pengujian dapat dipersingkat secara signifikan karena *tester* hanya perlu menjalankan kembali skrip yang sudah ada. Dengan demikian, *automation testing* memberikan efisiensi tinggi dalam proses *testing* pada sistem yang terus berkembang [11].

Tabel 2. 1 Tabel perbandingan studi sebelumnya

No	Judul	Peneliti	Tujuan	Metode	Hasil
1.	<i>Automation Regression Testing</i> pada Aplikasi Teman Diabetes Dengan Metode <i>Black Box Testing</i> [5]	Yoanna Fransisca Putri[5]	Membandingkan efektivitas dan efisiensi <i>manual operational acceptance testing</i> dengan <i>automation regression testing</i> pada aplikasi Teman Diabetes.	Metode yang digunakan adalah <i>Black Box Testing</i> dengan menggunakan <i>System Testing Life Cycle (STLC)</i> .	<i>Automation testing</i> jauh lebih cepat (1 jam vs 2 minggu) dengan jumlah <i>bugs</i> ditemukan sedikit lebih rendah.
2.	Analisis Perbandingan Pengujian <i>Manual</i> dan <i>Automation Testing</i> pada Website E-Commerce [8].	Dewandra Sapto Prasetyo & Widya Silfianti[8]	Membandingkan kecepatan eksekusi pengujian <i>manual</i> dan <i>automation</i> pada fitur umum e-commerce seperti halaman <i>login</i> , pencarian produk, <i>detail</i> produk,	Metode yang digunakan adalah <i>Black Box Testing</i> . Untuk <i>manual testing</i> menggunakan	Waktu <i>automation</i> jauh lebih cepat: 28,2–65,6 detik per TC dibanding <i>manual</i> 68–74,4 detik.

			keranjang belanja, hingga logout.	Microsoft Edge dan <i>Automation Testing</i> menggunakan Katalon Studio.	
3.	Otomatisasi Pengujian Perangkat Lunak Menggunakan Metode Data Driven untuk Meningkatkan Efisiensi Waktu Pengujian [10].	M.T. Banjarnahor & Lucia Sri Istiyowati[10]	Mengukur efektivitas dan kemudahan penggunaan metode <i>automation</i> untuk mempercepat proses pengujian software.	Metode yang digunakan adalah <i>Black Box Testing</i> dengan Teknik Data Driven.	<i>Automation</i> lebih efisien untuk regression (500+ TC per iterasi), mendeteksi <i>bug</i> lebih baik daripada <i>manual</i> (<i>manual</i> tidak temukan <i>bug</i>).
4.	<i>Automation Testing</i> Aplikasi Mobile ACC Leads Menggunakan Metode <i>Black Box</i>	M. Theresia[9]	Menerapkan <i>automation testing</i> sebagai alternatif pengujian <i>manual</i> yang memakan waktu lebih lama,	Metode yang digunakan adalah <i>Black Box Testing</i>	Penelitian menunjukkan bahwa aplikasi ACC Leads masih

	Testing Dengan Teknik Boundary Value Analysis Dan Equivalence Partitioning.		dengan tujuan memastikan aplikasi ACC Leads dapat memberikan peringatan ketika pengguna memasukkan tipe atau panjang data yang tidak sesuai.	dengan teknik Boundary Value Analysis (BVA) dan Equivalence Partitioning (EP).	menerima input tidak valid, baik dari segi format maupun panjang karakter, sehingga mekanisme validasinya perlu ditingkatkan untuk mencegah kesalahan input pengguna.
5	Evaluasi Penggunaan Manual dan Automated Software Testing pada Pelaksanaan End-To-End Testing [11].	Lian Min <i>et al.</i>	Mengevaluasi perbedaan efektivitas dan efisiensi antara manual dan automation testing dalam end-to-end testing.	Metode yang digunakan adalah Black Box Testing dengan tools automation Protactor.	Automation testing membutuhkan waktu lebih lama di tahap awal karena memerlukan proses setup skrip



				dan penyesuaian objek. Namun, setelah skrip tersusun, <i>automation</i> menjadi jauh lebih efisien dibanding <i>manual testing</i> , terutama saat dilakukan pengujian berulang.
--	--	--	--	--

2.2 Dasar Teori

2.2.1 Sistem Informasi Manajemen Aset

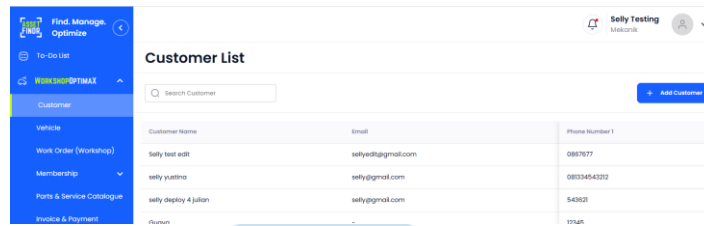
Aset merupakan segala sesuatu baik itu *tangible* atau *intangible* yang memiliki nilai jual, nilai ekonomi, nilai komersial atau nilai tukar yang dimiliki oleh instansi, organisasi, badan usaha ataupun individu [12]. Keberadaan aset memiliki nilai yang penting dalam kegiatan operasional perusahaan, sehingga jika tidak dikelola dengan baik dapat mengakibatkan terhambatnya kegiatan operasional perusahaan. Oleh karena itu perlu dilakukan manajemen aset .

Manajemen aset merupakan proses pengorganisasian, perencanaan dan pengawasan terhadap aset dimulai dari aset itu dibuat hingga dimusnahkan dengan tujuan untuk mengoptimalkan penggunaan aset dan meminimalisir resiko yang berkaitan dengan usia hidup aset [12]. Sistem informasi manajemen aset adalah sistem yang dapat memperlancar suatu proses pencatatan atau pendataan aset [13].

2.2.2 PT Solusi Berkat Inovasi (AssetFindr)

PT Solusi Berkat Inovasi, yang lebih dikenal dengan nama AssetFindr, adalah perusahaan teknologi yang fokus mengembangkan ekosistem manajemen pemeliharaan aset secara menyeluruh, dengan perhatian khusus pada aset kendaraan. Didirikan pada tahun 2024 oleh Sugih Sutjiono (CEO) dan Benediktus Sumarso (CPTO) di Jakarta, Indonesia, AssetFindr bertujuan untuk meningkatkan efisiensi operasional dan manajemen aset melalui solusi digital berbasis *cloud*, mencakup pelacakan kendaraan secara *real-time* menggunakan GPS, pemantauan kondisi kendaraan berbasis sensor, serta otomatisasi untuk pengelolaan pemeliharaan dan perbaikan, sistem ini memungkinkan pengguna untuk memantau, mencatat, dan mengoptimalkan aset kendaraan secara *real-time*, dari mana saja dan kapan saja, mengurangi risiko kerusakan mendadak, dan meningkatkan efisiensi penggunaan kendaraan [3].

2.2.3 Fitur-fitur AssetFindr



Gambar 2. 1 Contoh fitur AssetFindr

Sistem AssetFindr merupakan *platform* manajemen aset kendaraan berbasis web yang menyediakan berbagai layanan sesuai kebutuhan pengguna [3]. Sistem ini mengorganisasi fitur-fitur utamanya ke dalam beberapa *package*. Setiap package dikembangkan untuk memenuhi kebutuhan pengguna yang berbeda, tergantung pada jenis aset yang dikelola dan skenario operasional di lapangan. Berikut merupakan package yang disediakan oleh AssetFindr beserta fitur-fitur di dalamnya.

a. Workshop Optimax Package

Workshop optimax package dirancang khusus untuk mendukung operasional bengkel. Adapun fitur-fitur yang termasuk dalam paket ini adalah sebagai berikut

- *Customer*

Fitur ini digunakan untuk mengelola data pelanggan seperti nama, kontak, alamat, daftar kendaraan yang dimiliki serta riwayat layanan yang pernah dilakukan.

- *Vehicle*

Fitur ini berfungsi untuk mencatat dan mengelola data kendaraan milik pelanggan.

- *Work order*

Fitur ini digunakan untuk mencatat dan mengelola proses layanan perbaikan dan pemeliharaan kendaraan. Proses ini meliputi pencatatan keluhan, diagnosa, pengerjaan servis, pembelian suku cadang, serta biaya layanan.

- *Membership*

Pengelolaan keanggotaan pelanggan, termasuk jenis keanggotaan, masa berlaku, serta hak atau manfaat yang diperoleh anggota, seperti potongan harga.

b. General AssetFindr Package

General AssetFindr Package ditujukan untuk perusahaan yang membutuhkan sistem manajemen aset secara umum. *Package* ini cocok digunakan untuk perusahaan dengan banyak aset fisik seperti di bidang logistik, manufaktur, atau konstruksi.

Fitur-fitur dalam paket ini meliputi

- *Dashboard*
Fitur ini akan menampilkan informasi seperti statistik performa aset.
- *Work order*
Sama seperti *work order* pada *workshop package*, fitur *work order* pada *general assetfindr package* adalah sebagai layanan servis dan perawatan kendaraan, dari pembuatan hingga penyelesaian. Namun terdapat sedikit perbedaan pada *flow* dan tampilannya.
- *Asset*
Fitur ini digunakan untuk pencatatan dan pengelolaan data aset perusahaan. Baik berupa aset kendaraan ataupun bukan.
- *Report*
Penyusunan laporan-laporan terkait *asset* dan *work order*.

c. Fleet Optimax Package

Fleet optimax package merupakan *package* yang digunakan untuk mengelola aset kendaraan yang telah ditingkatkan ke layanan *Optimax*.

- *Fleet Dashboard*
Menampilkan statistik data aset kendaraan secara menyeluruh.
- *Fleet Asset List*
Menyajikan daftar kendaraan yang termasuk dalam layanan *fleet optimax*, lengkap dengan *detail* teknis dan status terkini.

- *Fleet Report*

Menyediakan laporan-laporan terkait penggunaan, kondisi, dan perawatan kendaraan dalam lingkup *fleet*.

d. *Tyre Optimax Package*

Tyre optimax package merupakan *package* khusus untuk pengelolaan data ban kendaraan. Fitur-fitur dalam *package* ini meliputi

- *Tyre Dashboard*

Melihat statistik ban seperti jumlah ban pada merek atau *size* dan kondisi tertentu.

- *Tyre List*

Daftar seluruh ban yang tercatat dalam sistem, termasuk informasi teknis, posisi pemasangan, dan statusnya.

- *Linked Tyre*

Fitur untuk mengaitkan ban dengan kendaraan tertentu, sehingga riwayat penggunaan dan posisi ban dapat dilacak secara akurat.

- *Tyre Inspection*

Fitur untuk pencatatan hasil inspeksi kondisi ban secara *manual*. Pada *tyre inspection user* dapat menginputkan hasil inspeksi ban pada kendaraan (*linked tyre*) atau ban saja (*unlinked tyre*).

- *Tyre Report*

Fitur ini melihat laporan terkait kondisi, rotasi, dan histori penggantian ban.

e. *Core Platform*

Merupakan kumpulan fitur pendukung yang digunakan secara lintas-package. Fitur-fitur dalam *Core Platform* meliputi:

- *To-do List*

Berfungsi sebagai pengingat tugas bagi pengguna. Meski termasuk dalam *Core Platform* dan bukan bagian dari *Workshop Package*, saat ini fitur ini hanya digunakan oleh akun dengan akses *Workshop Package*.

- *Approval Membership*

Fitur persetujuan untuk pendaftaran atau aktivasi keanggotaan pelanggan.

- *Settings*

Fitur ini terkait informasi perusahaan, manajemen *user*, manajemen lokasi dan *vendor* (biasanya digunakan pada *fleet* dan *tyre optimax package*).

- *Library*

Fitur ini digunakan untuk pengelolaan data aset konfigurasi, seperti tipe aset, tipe ban, dan kategori komponen.

Berikut ini adalah tabel yang merangkum nama *package*, jenis pengguna utama, fitur yang tersedia, serta catatan terkait pengaturan *role* pengguna.

Tabel 2. 2 Package AssetFindr

Package	Deskripsi Singkat	Pengguna Utama	Fitur	Role
<i>Workshop Optimax</i>	Merupakan <i>package</i> yang dirancang dengan fitur-fitur khusus untuk mendukung operasional bengkel.	Bengkel.	<i>Customer</i> <i>Vehicle</i> <i>Work order</i> <i>Membership</i>	Admin Mekanik
<i>General AssetFindr</i>	Merupakan <i>package</i> yang ditujukan untuk perusahaan yang membutuhkan sistem manajemen aset secara umum.	Perusahaan logistik, manufaktur, konstruksi dan perusahaan lain yang memerlukan manajemen aset.	<i>Dashboard</i> <i>Asset</i> <i>Work order</i> <i>Report</i>	<i>Role</i> pengguna disesuaikan dengan struktur organisasi perusahaan.
<i>Fleet Optimax</i>	Merupakan <i>package</i> yang digunakan untuk manajemen aset kendaraan yang telah di- <i>upgrade</i> ke layanan <i>Optimax</i> , atau untuk kendaraan yang membutuhkan	Perusahaan logistik, manufaktur, konstruksi dan perusahaan lain yang memerlukan manajemen aset yang	<i>Fleet Dashboard</i> <i>Fleet Asset List</i> <i>Fleet Report</i>	<i>Role</i> pengguna disesuaikan dengan struktur organisasi perusahaan.

	pengelolaan secara lebih khusus dan terstruktur.	lebih terstruktur dan lengkap.		
<i>Tyre Optimax</i>	<i>Tyre Optimax</i> merupakan package khusus untuk pengelolaan data ban kendaraan.	Perusahaan pengelola aset ban kendaraan.	<i>Tyre Dashboard</i> <i>Tyre List</i> <i>Tyre Linked</i> <i>Tyre Inspection</i> <i>Tyre Bonus & Penalty</i> <i>Tyre Report.</i>	<i>Role</i> pengguna disesuaikan dengan struktur organisasi perusahaan.
<i>Core Platform</i>	Merupakan kumpulan fitur pendukung yang digunakan secara lintas-package.	Bengkel maupun perusahaan yang memiliki AssetFindr package.	<i>To Do List</i> <i>Approval Membership</i> <i>Settings</i> <i>Library</i> <i>Notification</i> <i>Manage Profile</i>	<i>Role</i> pengguna disesuaikan dengan struktur organisasi perusahaan.

2.2.4 Testing

2.2.4.1 Software Testing

Software testing adalah salah satu tahapan atau proses penting yang dilakukan dalam pengembangan *software* yang bertujuan untuk mengetahui informasi tentang kualitas *software* dan menemukan kemungkinan terjadinya *bug* (kesalahan atau cacat) dalam *software* yang sedang dikembangkan [5]. *Software* harus diuji untuk mengetahui kualitas *software* sebelum akhirnya dirilis dan digunakan oleh pengguna sesungguhnya. *Software* dikatakan sudah memenuhi *requirement* jika *software* tersebut sudah lolos tahap uji di mana fitur- fitur yang dikembangkan berjalan sesuai dengan yang diharapkan dan tidak ditemukan kesalahan yang signifikan [14].

2.2.4.2 Software Testing Life Cycle (STLC)

STLC merupakan serangkaian langkah sistematis dalam proses pengujian perangkat lunak yang menggunakan metode *Black Box Testing*. Adapun tahapan dari STLC yaitu *Requirement Analysis*, *Test Planning*, *Test case Development*, *Environment Set Up*, *Test Execution*, dan *Test Closure* [15].

- *Requirement Analysis*

Tahap *Requirement Analysis* merupakan fase awal dalam *Software Testing Life Cycle* (STLC) yang berfungsi untuk memahami dan mengevaluasi kebutuhan sistem yang akan diuji. Pada tahap ini, tim *Quality assurance* (QA) meninjau dokumen kebutuhan seperti *Product Requirement Document* (PRD) atau *Software Requirement Specification* (SRS) untuk mengidentifikasi bagian-bagian sistem yang dapat diuji (*testable requirements*). Tahap ini juga bertujuan untuk mengklarifikasi kebutuhan yang belum lengkap, ambigu, atau berisiko menimbulkan kesalahan saat proses pengujian berlangsung.

- *Test Planning*

Pada fase *test planning*, QA mulai menyusun rencana pengujian berdasarkan hasil *requirement analysis*. Tahap ini mencakup penentuan cakupan pengujian, sumber daya yang dibutuhkan, penjadwalan aktivitas pengujian,

penetapan peran dan tanggung jawab anggota tim, serta pemilihan alat bantu (*tools*) yang akan digunakan.

- *Test case Development*

Fase ketiga pada STLC ini, QA membuat acuan dalam pengujian, seperti *Test case*, *Test Skenario*, dan *Test Script Automation* berdasarkan pada *Test case* yang sudah dibuat.

- Environment Setup

Tahap environment setup dilakukan untuk memastikan bahwa seluruh lingkungan (*environment*) pengujian, baik dari sisi perangkat keras (*hardware*) seperti komputer, jaringan, dan server, maupun dari sisi perangkat lunak (*software*) seperti sistem operasi, browser, database, dan *tools testing*, telah dikonfigurasi sesuai dengan kebutuhan pengujian yang direncanakan. Tahap ini dapat berjalan secara paralel dengan *tets case development* dan bersifat independen, artinya dapat dilakukan terpisah dari tahapan lain selama tidak mengganggu kelancaran pelaksanaan pengujian.

- *Test Execution*

Setelah *test case* disusun, tahap selanjutnya adalah *test execution*. Tim QA akan melakukan eksekusi pengujian baik secara *manual* maupun *automation* sesuai metode yang dipilih, dan mencatat hasil aktual dari setiap langkah pengujian. Hasil tersebut kemudian akan dibandingkan dengan *expected result* yang telah ditentukan pada *test case*. Jika hasil aktual sesuai, maka *test case* dinyatakan *passed*. Jika tidak sesuai, maka dinyatakan *failed* dan dilanjutkan dengan pencatatan *bug* atau defect untuk dilaporkan kepada developer. *Test execution* bertujuan untuk memastikan bahwa sistem telah berfungsi sesuai dengan kebutuhan yang ditentukan pada tahap *requirement analysis*.

- *Test Closure*

Tahap *test closure* merupakan fase akhir dalam siklus pengujian software. Pada tahap ini, tim QA melakukan evaluasi terhadap seluruh aktivitas pengujian yang telah dilakukan. Aktivitas dalam *test closure* meliputi pembuatan laporan akhir pengujian, analisis pencapaian tujuan pengujian,

dokumentasi defect yang ditemukan dan status penyelesaiannya, serta review terhadap efektivitas proses pengujian.

2.2.4.3 Metode Software Testing

2.2.4.3.1 Black Box Testing

Black box testing merupakan salah satu metode pengujian tanpa perlu melihat dan menguji *source code* program suatu *software*. Metode *black box testing* hanya berfokus pada *interface* atau *input* dan *output* pada *software* tanpa memperhatikan struktural dan komponen internalnya [16]. Tujuan pengujian ini adalah untuk memastikan bahwa program beroperasi sesuai dengan kebutuhan yang diinginkan oleh pengguna [17]. *Black box testing* juga disebut sebagai *functional testing* karena hanya menguji fungsional dari suatu *software* [18].

2.2.4.4 Jenis Pengujian Black Box Testing

2.2.4.4.1 Manual Testing

Manual testing atau pengujian *manual* merupakan pengujian yang dilakukan secara *manual* oleh penguji tanpa bantuan alat (*tools*) khusus [5]. Pada *manual testing*, penguji akan memposisikan diri sebagai *end user* dari *software* tersebut dan mulai mengeksplor seluruh fitur yang ada [19]. *Manual testing* bertujuan untuk mengidentifikasi *bug*, *defect*, masalah, atau cacat pada *software*. Konsep utama dari *manual testing* adalah untuk memastikan bahwa *software* bebas dari kesalahan dan dapat digunakan dengan baik sesuai dengan persyaratan fungsional dan *requirement* yang ditentukan. *Manual testing* memiliki beberapa level pengujian, yaitu *Black Box Testing*, *White Box Testing*, *Unit Testing*, *System Testing*, *Integration Testing*, dan *Acceptance Testing* [20].

2.2.4.4.2 Automation Testing

Automation testing atau pengujian otomatis merupakan pengujian yang dilakukan dengan *tools* pengujian khusus untuk menjalankan *test script* secara otomatis [5]. Ketika *software*, sistem atau aplikasi sedang dalam siklus pengembangan maka akan diperlukan pengujian berulang kali. Penggunaan *automation testing tools* memungkinkan rangkaian pengujian untuk direkam sehingga dapat dijalankan berulang kali sesuai kebutuhan [20]. *Automation testing* digunakan untuk mengurangi interaksi manusia dalam proses pengujian, sehingga

meminimalkan kebutuhan akan pengujian *manual*. Penggunaan alat ini membantu meningkatkan efisiensi dengan menghemat waktu, biaya, dan sumber daya manusia [19].

2.2.4.4.3 Manual dan Automation Testing (Hybrid)

Hybrid testing adalah pendekatan pengujian perangkat lunak yang menggabungkan metode *manual testing* dan *automation testing* untuk memaksimalkan efektivitas dan efisiensi pengujian. Pendekatan ini diambil karena masing-masing metode memiliki kelebihan dan keterbatasan, sehingga dengan mengombinasikannya, proses pengujian dapat lebih adaptif dan komprehensif [21].

2.2.4.5 Teknik Pengujian Black Box Testing

2.2.4.5.1 Equivalence partitioning

Equivalence partitioning merupakan teknik dalam pengujian *Black Box* yang bertujuan menyederhanakan proses pengujian dengan cara mengelompokkan data *input* ke dalam kelas-kelas yang dianggap setara. Setiap kelas mencerminkan kondisi *input* tertentu yang menghasilkan respons serupa dari sistem, sehingga satu *test case* dapat mewakili seluruh anggota dalam kelas tersebut [22].

Pengelompokan kelas ekuivalen dilakukan berdasarkan jenis *input* yang diuji, dengan ketentuan sebagai berikut:

1. Jika *input* berbentuk rentang nilai, maka ditetapkan satu kelompok data yang berada dalam rentang tersebut sebagai kelas *valid*, dan satu kelompok di luar rentang sebagai kelas *invalid*.
2. Jika *input* berasal dari himpunan nilai tertentu, maka nilai-nilai yang termasuk dalam himpunan tersebut dikategorikan sebagai kelas *valid*, sementara nilai di luar himpunan masuk ke dalam kelas *invalid*.
3. Jika *input* hanya menerima nilai spesifik, maka nilai tersebut dianggap sebagai kelas *valid*, sedangkan nilai selain itu menjadi bagian dari kelas *invalid*.
4. Jika *input* berupa pilihan boolean, maka sistem mengelompokkan satu nilai sebagai *valid* (misalnya *true*) dan nilai kebalikannya sebagai *invalid* (misalnya *false*).

Dengan menetapkan kelas-kelas ini, pengujian dapat dilakukan lebih efisien karena tidak perlu menguji setiap kemungkinan *input* secara individu, melainkan cukup perwakilannya dari masing-masing kelas [18].

2.2.5 Test case

Test case merupakan kumpulan *input* data dan kondisi operasional yang diperlukan untuk menguji suatu komponen, disertai dengan hasil yang diharapkan dari pengujian tersebut [14]. *Test case* disusun agar sistem yang akan diuji dapat memenuhi ketentuan, standar tertentu serta dapat berfungsi dengan baik.

Test case merupakan panduan yang digunakan *tester* untuk melakukan pengujian. Sebuah *test case* umumnya mencakup deskripsi pengujian, langkah-langkah pengujian (*test steps*), data uji yang digunakan (*test data*) dan hasil yang diharapkan (*expected result*) [23]. Jika hasil pengujian sesuai dengan ekspektasi yang tercantum dalam *test case*, maka sistem dinyatakan lulus pengujian (*passed*). Namun, jika hasil aktual tidak sesuai dengan hasil yang diharapkan, maka pengujian dinyatakan gagal (*failed*) dan dapat mengindikasikan adanya *bug* atau ketidaksesuaian fungsi pada sistem yang perlu diperbaiki [14].

2.2.6 Test Suite

Test suite merupakan kumpulan *test case* yang dapat diuji bersamaan secara paralel [24]. Keunggulan dari penggunaan *test suite* adalah kemampuannya untuk menggabungkan sejumlah *test case* sehingga dapat membentuk skenario pengujian yang beragam dan lebih kompleks dibandingkan hanya menggunakan *test case* secara terpisah. Melalui *test suite*, pengujian dapat dilakukan secara lebih menyeluruh yang tidak dapat dicapai hanya dengan satu *test case* tunggal [25].

2.2.7 Efektivitas Pengujian

Efektivitas pengujian merupakan indikator penting dalam mengukur sejauh mana *test case* mampu mendeteksi cacat (*bug*) pada perangkat lunak. Semakin banyak *bug* yang berhasil ditemukan dari sejumlah *test case* yang dijalankan, maka semakin tinggi tingkat efektivitasnya. Salah satu metrik yang umum digunakan adalah *Test case Effectiveness* (TCE) [26] .

TCE mengukur perbandingan antara jumlah *bug* yang ditemukan dengan jumlah *test case* yang dieksekusi. Metrik ini memberikan gambaran tentang kualitas

dan kemampuan *test case* dalam mengidentifikasi kesalahan yang mungkin terjadi pada sistem. Semakin tinggi nilai TCE, maka semakin efektif proses pengujian dalam mengungkapkan kesalahan sistem.

Rumus untuk menghitung metrik efektivitas adalah: jumlah *bug* yang ditemukan dibagi oleh jumlah total *test case* dikalikan dengan 100% yang dapat dilihat pada persamaan (1) [26].

$$TCE = \frac{DD}{TTC} \times 100\%$$

TCE = Test case Effectiveness

DD = Defect Detected

TTC = Total Test case

2.2.8 Efisiensi Pengujian

Efisiensi pengujian dapat dilihat dari berbagai aspek seperti biaya yang dikeluarkan, waktu, *effort*, *maintenance* dan tingkat keterulangan (*reusability*) [27]. Namun, pada penelitian ini, efisiensi dinilai berdasarkan dua indikator utama, yaitu waktu pembuatan *test case* dan waktu eksekusi *test case*, sebagai representasi dari aspek *effort* dan kecepatan proses pengujian. Efisiensi pengujian pada penelitian kali ini bertujuan untuk seberapa cepat dan optimal proses pembuatan dan eksekusi *test case*. Salah satu cara mengukurnya adalah dengan menghitung rata-rata durasi eksekusi per *test case*, yang merupakan indikator penting dalam evaluasi metode pengujian, baik *manual* maupun otomatis. Rumus yang digunakan dapat dilihat pada persamaan (2) [26].

$$tn = \frac{\sum_{i=1}^n ti}{n}$$

tn = Rata-rata durasi *test case*

ti = Durasi eksekusi *test case* ke-*i*

n = Total *test case*