ANALISIS EFEKTIVITAS METODE MANUAL DAN METODE BERBASIS ARTIFICIAL INTELLIGENCE DALAM PEMBUATAN SCRIPT AUTOMATION TESTING PADA APLIKASI BERBASIS WEBSITE MENGGUNAKAN KATALON STUDIO

(Skripsi)

Oleh

NADA BERLIANI PUTRI
2215061119

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK ELEKTRO
FAKULTAS TEKNIK
UNIVERSITAS LAMPUNG
2026

ABSTRAK

ANALISIS EFEKTIVITAS METODE MANUAL DAN METODE BERBASIS ARTIFICIAL INTELLIGENCE DALAM PEMBUATAN SCRIPT AUTOMATION TESTING PADA APLIKASI BERBASIS WEBSITE MENGGUNAKAN KATALON STUDIO

Oleh
Nada Berliani Putri

Penelitian ini bertujuan menganalisis perbandingan efektivitas pembuatan *script automation testing* secara manual dan berbasis *Artificial Intelligence* (AI) menggunakan Katalon Studio pada tiga aplikasi berbasis web. Pengujian dilakukan menggunakan pendekatan *black-box testing* terhadap 30 *test case* yang mencakup 15 fitur dengan tingkat kompleksitas *simple*, *average*, dan *complex*. Efektivitas metode diukur menggunakan tiga metrik, yaitu waktu pembuatan *script*, jumlah iterasi perbaikan, serta kompleksitas *script* yang dianalisis menggunakan *Lines of Code* (LOC) dan *Cyclomatic Complexity* (CC). Hasil penelitian menunjukkan bahwa metode berbasis AI lebih efektif dalam waktu pembuatan *script* dibandingkan metode manual. Total waktu pembuatan *script* pada metode manual sebesar 213 menit dengan rata-rata 7 menit per *test case*, sedangkan metode berbasis AI membutuhkan 101 menit dengan rata-rata 3 menit per *test case*. Hal ini menunjukkan bahwa AI mampu mempercepat proses pembuatan *script* sebesar 52,6%. Pada aspek iterasi perbaikan, metode manual memerlukan total 9 iterasi atau sebesar 30% dari keseluruhan *test case*, sedangkan metode berbasis AI memerlukan 5 iterasi atau sebesar 16,7%. Hasil tersebut menunjukkan bahwa *script* berbasis AI cenderung lebih stabil pada proses eksekusi awal, meskipun kualitas *prompt* berpengaruh terhadap hasil *script* yang dihasilkan. Pada aspek kompleksitas *script*, metode manual menghasilkan *script* yang lebih sederhana dengan total LOC sebesar 1528 dan total CC sebesar 76, sedangkan metode berbasis AI menghasilkan total LOC sebesar 1948 dan total CC sebesar 101. Hasil penelitian menunjukkan bahwa metode berbasis AI lebih efektif dari sisi efisiensi waktu, sedangkan metode manual lebih unggul dalam kesederhanaan dan kemudahan pemeliharaan *script automation testing*.

Kata Kunci: *Artificial Intelligence*, *Automation Testing*, Efektivitas, Katalon Studio, *Script*, *Website*.

ABSTRACT

EFFECTIVENESS ANALYSIS OF MANUAL AND ARTIFICIAL INTELLIGENCE-BASED METHODS IN AUTOMATION TESTING SCRIPT DEVELOPMENT FOR WEB-BASED APPLICATIONS USING KATALON STUDIO

By

Nada Berliani Putri

This study aims to analyze the effectiveness comparison between manual and Artificial Intelligence (AI)-based automation testing script generation using Katalon Studio on three web-based applications. Testing was conducted using a black-box testing approach on 30 test cases covering 15 features with simple, average, and complex levels of complexity. The effectiveness of each method was measured using three metrics: script creation time, number of script revision iterations, and script complexity measured using Lines of Code (LOC) and Cyclomatic Complexity (CC). The results show that the AI-based method was more effective in terms of script creation time compared to the manual method. The total script creation time using the manual method was 213 minutes with an average of 7 minutes per test case, while the AI-based method required 101 minutes with an average of 3 minutes per test case. This indicates that AI accelerated the script creation process by 52.6%. In terms of revision iterations, the manual method required a total of 9 iterations or 30% of all test cases, while the AI-based method required 5 iterations or 16.7%. These findings indicate that AI-generated scripts tended to be more stable during initial execution, although prompt quality significantly influenced the generated results. In terms of script complexity, the manual method produced simpler scripts with a total LOC of 1528 and total CC of 76, while the AI-based method produced a total LOC of 1948 and total CC of 101. The study concludes that the AI-based method is more effective in terms of time efficiency, while the manual method is superior in script simplicity and maintainability.

Keyword: Artificial Intelligence, Automation Testing, Effectiveness, Katalon Studio, Script, *Website.*

ANALISIS EFEKTIVITAS METODE MANUAL DAN METODE BERBASIS ARTIFICIAL INTELLIGENCE DALAM PEMBUATAN SCRIPT AUTOMATION TESTING PADA APLIKASI BERBASIS WEBSITE MENGGUNAKAN KATALON STUDIO

Oleh

NADA BERLIANI PUTRI

Skripsi

Sebagai Salah Satu Syarat untuk Mencapai Gelar
SARJANA TEKNIK

Pada

Jurusan Teknik Elektro
Fakultas Teknik

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK ELEKTRO
FAKULTAS TEKNIK
UNIVERSITAS LAMPUNG
2026

Judul Skripsi	Analisis Efektivitas Metode Manual Dan Metode Berbasis Artificial Intelligence dalam Pembuatan Script Automation Testing pada Aplikasi Berbasis Website Menggunakan Katalon Studio
Nama Mahasiswa	Nada Berliani Putri
Nomor Pokok Mahasiswa	2215061119
Program Studi	S1 Teknik Informatika
Fakultas	Teknik

MENYETUJUI

1. Komisi Pembimbing

Wahyu Eko S. S.T., M.Sc.	**Ir. Trisya Septiana, S.T., M.T., IPM.**
NIP 197412012001121001	NIP 199009212019032025

2. Mengetahui

Ketua Jurusan Teknik Elektro	Ketua Program Studi Teknik Elektro
Herlinawati, S.T., M.T.	**Yessi Mulyani, S.T., M.T.**
NIP 19710314 199903 2 001	NIP 19731226 200012 2001

MENGESAHKAN

1. Tim Penguji

Ketua : **Wahyu Eko Sulistiono, S.T., M.T.** .....

Sekretaris : **Ir. Trisya Septiana, S.T., M.T., IPM.** .....

Penguji : **Mona Arif Muda, S.T., M.T** .....

2. Dekan Fakultas Teknik

Dr. Ahmad Herison, S.T., M.T
NIP. 196910302000031001

Tanggal Lulus Ujian Skripsi: 26 Mei 2026

SURAT PERNYATAAN

Melalui ini saya menyatakan bahwa skripsi ini tidak terdapat karya yang pernah dilakukan orang lain dan sepanjang sepengetahuan saya tidak terdapat atau diterbitkan oleh orang lain, kecuali secara tertulis diacu dalam naskah ini sebagaimana yang disebutkan dalam daftar pustaka. Selain itu, saya menyatakan pula bahwa skripsi ini dibuat oleh saya sendiri. Apabila di kemudian hari terbukti bahwa skripsi saya ini merupakan salinan atau dibuat oleh orang lain, maka saya bersedia menerima sanksi sesuai dengan ketentuan hukum atau akademik yang berlaku.

Bandar Lampung, 26 Mei 2026
Pembuat Pernyataan

Nada Berliani Putri
NPM. 2215061119

RIWAYAT HIDUP

Penulis lahir di Pesawaran pada tanggal 15 Oktober 2003. Penulis merupakan anak kedua dari tiga bersaudara, anak dari pasangan Edian Feri dan Liza Mulya Heti. Penulis mengawali pendidikan di MIN 1 Pesawaran dari tahun 2010 hingga 2016, kemudian melanjutkan ke MTsN 1 Pesawaran pada tahun 2016 hingga 2019.
Pendidikan menengah atas ditempuh di MAN 1 Pesawaran, Jurusan Ilmu Pengetahuan Alam (MIPA), dari tahun 2019 hingga 2022.

Penulis menjadi mahasiswa di Universitas Lampung, Program Studi Teknik Elektro, sejak tahun 2022 melalui jalur SBMPTN. Selama masa studi, penulis aktif dalam berbagai kegiatan akademik dan organisasi. Penulis pernah menjadi anggota Himpunan Mahasiswa Teknik Informatika (HIMATRO) selama beberapa bulan pada tahun 2023 dan dipercaya sebagai penanggung jawab *Essay* dalam kegiatan *Himatro Research Class* (HRC). Selain itu, penulis juga aktif dalam Unit Kegiatan Mahasiswa (UKM) English Society serta terlibat dalam berbagai kepanitiaan kegiatan kemahasiswaan, salah satunya pada kegiatan Inauguration.

Di luar lingkungan kampus, penulis aktif dalam kegiatan sosial dan kerelawanan melalui komunitas Rayakan Asa sebagai volunteer. Penulis juga pernah berpartisipasi sebagai volunteer dalam komunitas Dengan Damai pada tahun 2025. Pada bidang pengembangan kompetensi dan pengalaman profesional, penulis mengikuti Program Studi Independen Bersertifikat di Bangkit Academy pada tahun 2024 dengan *learning path Android Development*. Penulis juga melaksanakan program magang di Dinas Komunikasi, Informatika, Statistik, dan Persandian Kabupaten Pesawaran pada tahun 2025 sebagai *Programmer*. Selain itu, pada tahun 2025 penulis mengikuti pelatihan dan memperoleh Sertifikasi Kompetensi Programmer yang diselenggarakan oleh Badan Nasional Sertifikasi Profesi (BNSP).

PERSEMBAHAN

Alhamdulillah, Atas Berkat Rahmat Allah yang Maha Kuasa

KUPERSEMBAHKAN KARYA INI UNTUK

Ayah dan Mama Tercinta

Edian Feri dan Liza Mulya Heti

Kakak dan Adikku Tersayang

Dicky Lian Pratama Dan Bima Rizky Andhika

Keluarga Besar yang selalu menjadi sumber semangat, Dosen yang telah membimbing dengan tulus, serta sahabat dan seluruh teman yang setia membersamai setiap langkah perjalanan ini.

Last but not least, Nada Berliani Putri, diri penulis sendiri, atas perjuangan yang mungkin tidak banyak terlihat oleh orang lain. Terima kasih untuk tetap bertahan, berjuang, dan terus melangkah hingga sampai pada titik ini.

MOTTO

“Perang telah usai, aku bisa pulang.
Kubaringkan panah dan berteriak MENANG!”
(Nadin Amizah)

“*Everything you lose is a step you take*.”
(Taylor Swift)

"*The important thing is not to stop questioning.*
Curiosity has its own reason for existing."
(Albert Einstein)

SANWACANA

Segala puji hanya milik Allah Subhanahu wa Ta'ala, atas rahmat, hidayah, dan karunia-Nya sehingga penulis dapat menyelesaikan skripsi ini dengan baik. Penulis menyadari bahwa terselesaikannya skripsi ini tidak lepas dari bantuan, bimbingan, doa, dan dukungan dari berbagai pihak yang telah hadir dalam perjalanan panjang ini.

Skripsi dengan judul "Analisis Efektivitas Metode Manual dan Metode Berbasis Artificial Intelligence dalam Pembuatan Script Automation Testing pada Aplikasi Berbasis Website Menggunakan Katalon Studio" disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana Teknik pada Jurusan Teknik Elektro, Fakultas Teknik, Universitas Lampung. Melalui kesempatan ini, penulis ingin menyampaikan rasa terima kasih yang sebesar-besarnya dan penghargaan yang setinggi-tingginya kepada seluruh pihak yang telah memberikan kontribusi nyata dalam proses penyelesaian skripsi ini.

1. Kedua orang tuaku, ayah dan mama serta kakak dan adikku, yang senantiasa hadir dengan doa yang tak pernah putus, dukungan yang tak tergoyahkan, dan kepercayaan penuh dalam setiap langkah perjalanan hidup penulis. Terima kasih atas segala cinta, pengorbanan, dan apresiasi yang selalu diberikan di setiap pencapaian penulis, sekecil apapun itu.
2. Bapak Dr. Ahmad Herison, S.T., M.T., selaku Dekan Fakultas Teknik Universitas Lampung.
3. Ibu Herlinawati, S.T., M.T selaku Ketua Jurusan Teknik Elektro Universitas Lampung.
4. Ibu Yessi Mulyani, S.T., M.T selaku Kepala Program Studi Teknik Informatika Universitas Lampung.
5. Bapak Wahyu Eko S., S.T., M. Sc. selaku dosen pembimbing utama, yang dengan penuh kesabaran telah memberikan bimbingan, arahan, masukan yang konstruktif, serta selalu meluangkan waktu untuk berdiskusi bersama penulis selama proses penyusunan skripsi ini berlangsung.

6. Ibu Ir. Trisya Septiana, S.T., M.T., IPM. selaku dosen pembimbing pendamping, yang telah memberikan saran, masukan, serta dukungan yang sangat berarti dalam proses penelitian dan penyusunan skripsi ini.
7. Mona Arif Muda, S.T., M.T. selaku dosen penguji, penelitian yang telah memberikan kritik dan saran yang membangun sehingga skripsi ini dapat menjadi lebih baik dan lebih sempurna.
8. Segenap Dosen di Jurusan Teknik Elektro Universitas Lampung, yang telah dengan ikhlas mendidik dan mentransfer ilmu pengetahuan, wawasan, serta pengalaman berharga yang menjadi bekal penulis dalam menjalani kehidupan akademik maupun profesional.
9. Segenap Staf di Jurusan Teknik Elektro dan Fakultas Teknik, yang telah sangat membantu penulis baik dalam hal administrasi dan hal-hal lainnya.
10. Agnes, Cindy, Sheina, Yosi, dan Almh. Fiska, sahabat-sahabat terbaik yang telah mewarnai hari-hari perkuliahan penulis dengan tawa, kebersamaan, dan kenangan yang tak terlupakan. Terima kasih telah menjadi penghibur di saat lelah, penyemangat di saat goyah, dan teman berbagi dalam suka maupun duka.
11. Dina, Dini, Jihan, Mardiana, dan Suci, sahabat setia sejak masa sekolah yang hingga kini masih hadir memberikan tawa, kehangatan, dan kebersamaan yang tulus. Terima kasih telah menjadi bagian tak terpisahkan dari perjalanan hidup penulis.
12. Seluruh teman-teman mahasiswa Teknik Informatika Kelas C angkatan 2022, terima kasih atas solidaritas dan kebersamaan yang tidak pernah terasa sendiri. Kita tumbuh bersama dalam proses yang panjang, saling menguatkan dalam keterbatasan, dan saling menjadi saksi bahwa setiap perjuangan selalu menemukan akhirnya.

Bandar Lampung, 26 Mei 2026

Nada Berliani Putri

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

I. PENDAHULUAN

1.1 Latar Belakang dan Masalah

Quality Assurance (QA) memiliki peran penting untuk memastikan bahwa sistem yang dikembangkan memenuhi standar kualitas yang ditetapkan serta berfungsi sesuai dengan kebutuhan pengguna. QA tidak hanya berfokus pada proses pengujian di tahap akhir pengembangan, tetapi juga berperan dalam menjamin kualitas perangkat lunak secara keseluruhan melalui proses verifikasi dan validasi yang terstruktur. Salah satu aktivitas utama dalam QA adalah pengujian perangkat lunak (*software testing*), yang bertujuan untuk mendeteksi kesalahan (*bug*), memverifikasi kesesuaian fungsi dengan spesifikasi, serta memastikan keandalan aplikasi sebelum digunakan oleh pengguna akhir. Pelaksanaan pengujian yang sistematis terbukti mampu meningkatkan kualitas perangkat lunak sekaligus menekan biaya perbaikan setelah sistem rilis[1].

Pengujian dapat dilakukan melalui dua pendekatan utama, yaitu pengujian manual dan pengujian otomatis. Pengujian manual dilakukan oleh *tester* dengan cara menjalankan aplikasi secara langsung tanpa bantuan *script* otomatis. Pendekatan ini memungkinkan QA untuk mengevaluasi sistem dari sudut pandang pengguna, namun memiliki sejumlah keterbatasan, seperti waktu pelaksanaan yang relatif lama, ketergantungan pada ketelitian *tester*, serta kurang efektif untuk pengujian berulang yang bersifat repetitif. Kondisi tersebut menyebabkan pengujian manual menjadi kurang optimal ketika diterapkan pada aplikasi berbasis web yang memiliki banyak fitur, alur proses yang kompleks, serta frekuensi perubahan sistem yang tinggi [2].

Sebagai bagian dari upaya peningkatan efektivitas QA, *automation testing* digunakan untuk mendukung proses pengujian dengan memanfaatkan *script* dan alat bantu (*tools*) yang mampu mengeksekusi skenario pengujian secara otomatis. Efektivitas *automation testing* tidak hanya ditentukan oleh hasil pengujiannya, tetapi juga oleh proses pembuatan *script automation testing* itu sendiri. *Script automation testing* yang efektif diharapkan dapat dibuat dalam waktu yang lebih singkat, membutuhkan sedikit iterasi perbaikan hingga mencapai kondisi stabil, serta memiliki tingkat kompleksitas yang dapat dikelola. Oleh karena itu, analisis terhadap efektivitas pembuatan *script automation testing* menjadi aspek penting dalam mendukung aktivitas QA pada aplikasi berbasis web. Berbagai penelitian menunjukkan bahwa penerapan *automation testing* pada aplikasi berbasis web dapat meningkatkan cakupan dan kualitas pengujian, dengan memanfaatkan *tools* seperti Katalon Studio, Selenium WebDriver, Cypress, dan Playwright [2].

Salah satu alat yang banyak digunakan untuk otomasi pengujian adalah Katalon Studio, yang mendukung pengujian lintas platform, termasuk aplikasi berbasis web. Fitur-fitur seperti *record & playback*, integrasi laporan otomatis, dan dukungan scripting berbasis *Groovy* menjadikan Katalon Studio sebagai pilihan ideal bagi tim *Quality Assurance (QA)*. Penelitian oleh [3] menunjukkan bahwa penggunaan Katalon Studio dapat meningkatkan efisiensi waktu pengujian hingga 42% tanpa menurunkan tingkat deteksi kesalahan (*defect detection rate*). Merujuk pada penelitian [4] menunjukkan bahwa *automation testing* dengan Katalon Studio pada aplikasi web mampu mengidentifikasi *error* pada kasus uji tertentu dan memberikan data kuantitatif terkait durasi respons dan status hasil pengujian (*passed/failed*).

Berbeda dengan *manual testing* yang dapat dilakukan langsung tanpa *script*, *automation testing* mengharuskan pengembang merancang dan mengembangkan *script* yang kompleks, sehingga sering kali memakan waktu dan rentan terhadap kesalahan pada tahap awal pengembangan *script*. Tantangan ini dapat menghambat efektivitas QA, terutama pada tim dengan keterbatasan sumber daya. Oleh karena itu, pemanfaatan *Artificial Intelligence* (AI) dalam pembuatan *script automation*

testing mulai dikembangkan untuk membantu QA menghasilkan *script* secara lebih cepat dan mengurangi kesalahan penulisan kode. Proses pembuatan *script automation testing* melalui bantuan AI seperti ChatGPT dapat dilakukan secara lebih efektif tanpa menghilangkan peran manusia dalam verifikasi dan penyesuaian. Sejumlah penelitian juga telah menunjukkan potensi AI dalam mendukung proses pembuatan *script automation testing*, seperti An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation [5], GenIA-E2ETest: A Generative AI-Based Approach for End-to-End Test Automation [6], dan A Review of Large Language Models for Automated Test Case Generation [7]. Penelitian yang secara khusus membandingkan efektivitas pembuatan *script automation testing* secara manual dan dengan bantuan AI dalam konteks aktivitas QA pada aplikasi berbasis web masih terbatas. Berdasarkan hal tersebut, penelitian ini diarahkan untuk menganalisis efektivitas pembuatan *script automation testing* di Katalon Studio dengan dua metode, yaitu secara manual dan dengan bantuan AI.

Objek pengujian yang digunakan dalam penelitian ini meliputi *website* Sistem Informasi Geografis Universitas Lampung, GitHub, dan OpenStreetMap. Ketiga *website* tersebut dipilih karena memiliki karakteristik fitur dan interaksi pengguna yang berbeda, mulai dari pengelolaan informasi spasial berbasis peta, pengelolaan repository dan *issue tracking*, hingga layanan navigasi dan *routing* berbasis geospasial. Variasi fitur tersebut digunakan untuk merepresentasikan kondisi pengujian *automation testing* pada aplikasi web dengan tingkat kompleksitas yang beragam. Fokus penelitian meliputi pengukuran waktu pembuatan *script* [8], [1], jumlah iterasi perbaikan hingga *script* mencapai kondisi stabil [9], serta kompleksitas *script* [10], [11]. Hasil penelitian diharapkan dapat memberikan gambaran empiris mengenai peran bantuan AI dalam meningkatkan efektivitas proses pembuatan *script automation testing* sebagai bagian dari aktivitas QA.

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah dipaparkan, maka rumusan masalah dalam penelitian ini adalah sebagai berikut.

1. Bagaimana efektivitas pembuatan *script automation testing* secara manual dan berbasis AI berdasarkan metrik waktu pembuatan *script*, jumlah iterasi perbaikan, dan kompleksitas *script*?
2. Bagaimana peran penggunaan AI dalam meningkatkan efektivitas dan kemudahan dalam proses pembuatan *script automation testing* di Katalon Studio pada aplikasi berbasis web?

1.3 Tujuan Penelitian

Tujuan penelitian ini adalah sebagai berikut.

1. Mengukur efektivitas pembuatan *script automation testing* secara manual dan berbasis AI pada Katalon Studio dengan menggunakan metrik waktu pembuatan *script*, jumlah iterasi perbaikan, dan kompleksitas *script*.
2. Menganalisis peran penggunaan AI dalam meningkatkan efektivitas dan kemudahan dalam proses pembuatan *script automation testing* di Katalon Studio pada aplikasi berbasis web, melalui perbandingan dengan pendekatan manual.

1.4 Manfaat Penelitian

Manfaat penelitian ini adalah sebagai berikut.

1. Menjadi referensi akademik bagi penelitian selanjutnya yang berkaitan dengan penerapan AI dalam pembuatan *script automation testing* dibidang *Software Quality Assurance* dan *Software Testing*.
2. Memberikan acuan bagi pengembang dan tim *Quality Assurance* dalam menentukan metode pembuatan *script automation testing* yang lebih efektif,

akurat, dan sesuai untuk meningkatkan efektivitas proses pengujian perangkat lunak.

1.5 Batasan Masalah

Batasan masalah dalam penelitian ini adalah sebagai berikut.

1. Penelitian ini difokuskan pada proses pembuatan dan analisis *script automation testing*, bukan pada evaluasi hasil atau keluaran fungsional dari eksekusi pengujian terhadap aplikasi secara menyeluruh.
2. Objek pengujian dalam penelitian ini dibatasi pada tiga aplikasi berbasis web, yaitu *Website* Sistem Informasi Geografis Universitas Lampung, GitHub, dan OpenStreetMap. *Website* Sistem Informasi Geografis Universitas Lampung merupakan *prototype* penelitian yang dikembangkan oleh mahasiswa untuk kepentingan akademis dan bukan *website* resmi institusi Universitas Lampung.
3. Lingkup pengujian dibatasi pada fitur utama yang mendukung interaksi pengguna pada masing-masing *website*, meliputi pencarian, filter, pengurutan, interaksi peta, rincian fasilitas, *issue tracking*, *routing*, *share location*, *layer* peta, pergantian moda transportasi, download repository ZIP, dan download *raw file*.
4. Analisis efektivitas pembuatan *script automation testing* dilakukan hanya berdasarkan tiga metrik, yaitu waktu pembuatan *script*, jumlah iterasi perbaikan, dan kompleksitas *script* yang diukur menggunakan *Lines of Code (LOC)* dan *Cyclomatic Complexity* (CC).
5. Proses pengujian *script* dalam penelitian ini hanya menggunakan Katalon Studio sebagai *tool automation testing*, tanpa melibatkan *tool* pengujian otomatis lainnya.
6. Pada metode berbasis *Artificial Intelligence (AI)*, AI berperan langsung sebagai pembuat *script* pengujian, di mana AI menghasilkan *script* awal berdasarkan skenario pengujian yang diberikan.

1.6 Sistematika Penulisan

Struktur penulisan pada penelitian ini adalah sebagai berikut.

BAB I: PENDAHULUAN

Bab ini berisi latar belakang masalah, rumusan masalah, tujuan penelitian, manfaat penelitian, batasan masalah, dan sistematika penulisan.

BAB II: TINJAUAN PUSTAKA

Bab ini berisi tentang penelitian-penelitian terdahulu dan teori-teori yang mendukung sebagai referensi dalam penelitian ini.

BAB III: METODOLOGI PENELITIAN

Bab ini berisi tentang waktu dan tempat penelitian, alat dan bahan yang digunakan dalam penelitian, dan prosedur serta tahapan penelitian.

BAB IV: HASIL DAN PEMBAHASAN

Bab ini berisi hasil dan pembahasan mengenai penelitian yang dilakukan.

BAB V: KESIMPULAN DAN SARAN

Bab ini memuat kesimpulan berdasarkan hasil penelitian dan saran yang diharapkan untuk pengembangan penelitian selanjutnya.

II. TINJAUAN PUSTAKA

2.1 Dasar Teori

2.1.1 Website

Website adalah sekumpulan halaman web yang saling terhubung dan dapat diakses melalui browser dengan memanfaatkan teknologi web standar. *Website* dibangun menggunakan tiga komponen utama, yaitu HTML sebagai struktur dasar halaman, CSS untuk mengatur tampilan dan tata letak antarmuka, serta JavaScript untuk memberikan perilaku dinamis dan interaktif pada halaman web. Kombinasi HTML, CSS, dan JavaScript menjadikan *website* mampu berfungsi tidak hanya sebagai media informasi statis, tetapi juga sebagai aplikasi berbasis web yang interaktif dan kompleks. *Website* dirancang agar dapat berjalan lintas browser dan mendukung berbagai perangkat, sehingga kualitas struktur kode dan konsistensi perilaku antarmuka menjadi aspek penting [12].

Aplikasi web dalam konteks kualitas perangkat lunak harus melalui pengujian menyeluruh. Pengujian mencakup performa (kecepatan respon dan penggunaan sumber daya), keamanan (perlindungan data pengguna), kompatibilitas (browser dan perangkat), dan konsistensi tampilan (UI/UX). Pengujian yang komprehensif dapat meningkatkan keandalan, mengurangi risiko kegagalan layanan, serta meningkatkan kepuasan pengguna secara signifikan [13]. Oleh karena itu, pengembangan aplikasi web tidak hanya menekankan aspek fungsional, tetapi juga kestabilan, responsivitas, dan kemampuan aplikasi dalam menangani beban pengguna yang tinggi.

2.1.2 Quality Assurance (QA)

Quality Assurance (QA) merupakan pendekatan sistematis dalam rekayasa perangkat lunak yang bertujuan untuk menjamin bahwa proses pengembangan dan produk perangkat lunak yang dihasilkan memenuhi standar kualitas yang telah ditetapkan. QA mencakup serangkaian aktivitas terencana dan terdokumentasi yang berfokus pada pencegahan kesalahan (*defect prevention*), pengendalian proses, serta evaluasi kualitas secara berkelanjutan sepanjang *software development lifecycle* (SDLC). Berbeda dengan pengujian perangkat lunak (*software testing*) yang berfokus pada pendeteksian kesalahan pada produk akhir, QA menekankan penjaminan kualitas melalui penerapan proses, metode, dan praktik yang terstruktur sejak tahap awal pengembangan [14]. QA juga dipandang sebagai bagian dari *software quality engineering* yang mencakup kegiatan perencanaan kualitas, pemantauan proses, serta penggunaan metrik untuk mengevaluasi dan meningkatkan kualitas perangkat lunak secara berkesinambungan. Pendekatan ini menempatkan QA tidak hanya sebagai aktivitas teknis, tetapi sebagai mekanisme manajerial dan teknis yang bertujuan untuk meningkatkan keandalan sistem, konsistensi kualitas, serta efektivitas proses pengembangan perangkat lunak, khususnya pada aplikasi berbasis web dengan tingkat kompleksitas yang tinggi [15].

2.1.3 Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan serangkaian aktivitas yang dilakukan untuk menemukan kesalahan dalam perangkat lunak serta memberikan jaminan bahwa perangkat lunak memiliki kualitas dan reliabilitas yang memadai. Aktivitas pengujian dapat dirancang dan dilaksanakan sepanjang tahapan pengembangan perangkat lunak, dengan tujuan memastikan kesesuaian perangkat lunak terhadap kebutuhan yang ditetapkan [16]. Pengujian yang dilakukan secara terstruktur dan sistematis terbukti mampu meningkatkan keandalan perangkat lunak serta mengurangi biaya pemeliharaan pasca-rilis[1]. Tanpa pengujian yang efektif, aplikasi dapat mengalami kegagalan, gangguan performa atau *bug* yang berdampak buruk terhadap pengalaman pengguna dan citra pengembang. Pengujian meliputi berbagai jenis, antara lain pengujian fungsional, regresi,

performa, serta pengujian antarmuka pengguna (UI). Masing-masing jenis pengujian memiliki peran penting untuk menjamin kualitas aplikasi secara menyeluruh.

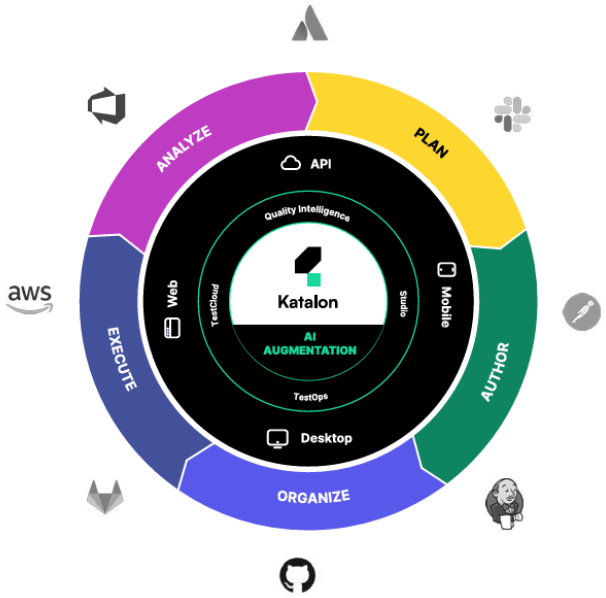
Aplikasi berbasis web selain pengujian fungsional juga memerlukan pengujian dari aspek keamanan untuk melindungi sistem dan data pengguna. Aplikasi web rentan terhadap berbagai kerentanan seperti *SQL injection*, *cross-site scripting (XSS)*, dan *file inclusion* yang dapat dieksploitasi oleh pihak tidak bertanggung jawab. Studi tersebut menekankan pentingnya *penetration testing* sebagai bagian dari pengujian perangkat lunak berbasis web, baik secara manual maupun otomatis, untuk mengidentifikasi celah keamanan sebelum aplikasi digunakan secara luas [13]. Temuan tersebut memperkuat urgensi penerapan *automation testing* pada aplikasi web secara menyeluruh, termasuk pengujian fungsional yang menjadi fokus utama dalam penelitian ini, meskipun penelitian ini berfokus pada pengujian keamanan.

2.1.4 Automation Testing

Automation testing merupakan pendekatan pengujian perangkat lunak yang memanfaatkan alat bantu otomatis untuk mengeksekusi serangkaian pengujian secara cepat dan berulang, khususnya ketika pengujian manual menjadi tidak efisien atau tidak memungkinkan dilakukan akibat keterbatasan waktu dan sumber daya [17]. Studi sistematis oleh [18] menyimpulkan bahwa *automation testing* terbukti lebih efektif dibanding manual, dengan peningkatan produktivitas dan akurasi pengujian hingga lebih dari 45%. Penelitian ini menunjukkan bahwa implementasi *automation testing* bukan hanya mempercepat waktu pengujian, tetapi juga mengurangi kesalahan manusia. Studi lain juga mengungkapkan bahwa penerapan otomasi menggunakan Selenium dan Katalon Studio mampu meningkatkan *test coverage* tanpa menambah biaya pengujian secara signifikan [4]. *Automation testing* dalam konteks aplikasi web sangat relevan karena aplikasi sering diperbarui, fitur bertambah, dan skenario uji menjadi kompleks, sehingga efektivitas dan efisiensi menjadi faktor penting. *Automation testing* memiliki tantangan tersendiri seperti kebutuhan pemeliharaan *script*, perubahan antarmuka aplikasi yang memengaruhi hasil uji, serta investasi awal dalam penyiapan alat dan

sumber daya manusia. Oleh karena itu, efektivitas *automation testing* sangat dipengaruhi oleh kemampuan tim dalam mengelola kompleksitas teknis dan pemeliharaan sistem.

2.1.5 Katalon Studio



Gambar 2.1 Katalon Studio [19]

Katalon Studio merupakan platform pengujian perangkat lunak *opensource* yang dikembangkan oleh Katalon LLC dan berfokus pada otomatisasi pengujian untuk aplikasi web, mobile, serta layanan API [20]. Platform ini kompatibel dengan berbagai sistem operasi, termasuk Windows, Linux, dan macOS, serta dilengkapi dengan antarmuka berbasis IDE yang memudahkan proses pengujian secara intuitif dan terstruktur.

Katalon menyediakan abstraksi berupa *built-in keywords* yang merepresentasikan aksi pengujian tertentu, seperti interaksi dengan elemen antarmuka, validasi kondisi sistem, serta pengelolaan alur pengujian, sehingga mempermudah proses penulisan *script automation testing*.

Platform ini dirancang untuk memberikan solusi pengujian yang efektif, baik bagi pengguna yang sudah berpengalaman maupun bagi pemula, karena antar mukanya yang ramah pengguna dan beragam fitur canggih. Salah satu keunggulan Katalon Studio adalah kemampuannya untuk mendukung pengujian lintas platform, termasuk pengujian aplikasi web melalui browser populer seperti Chrome, Firefox, Safari, dan Edge, pengujian aplikasi mobile pada perangkat Android dan iOS, serta pengujian API untuk layanan RESTful dan SOAP. Platform ini juga menyediakan fitur pelaporan dan dasbor bawaan untuk memantau serta menganalisis hasil pengujian secara lebih efektif. Hasil pengujian dapat diekspor ke dalam berbagai format seperti HTML atau PDF, memberikan informasi yang jelas mengenai keberhasilan maupun kegagalan tes, serta area yang memerlukan perbaikan.

Katalon Studio juga mendukung dua pendekatan utama dalam pembuatan *script automation testing*, yaitu penulisan *script* secara manual dan pemanfaatan fitur *record and playback*. Penulisan *script* manual memungkinkan pengujian dirancang dengan logika yang lebih terkontrol dan fleksibel, sedangkan fitur *record and playback* memudahkan pembuatan *script* secara cepat berdasarkan interaksi pengguna dengan sistem yang diuji.

Berdasarkan Gambar 2.1, Katalon Studio dirancang sebagai platform *automation testing* yang mendukung seluruh siklus pengujian perangkat lunak secara terintegrasi, mulai dari tahap *Analyze*, *Plan*, *Author*, *Organize*, hingga *Execute*. Tahap *Analyze* merepresentasikan proses analisis kebutuhan pengujian, seperti identifikasi fitur sistem yang akan diuji, jenis pengujian yang digunakan, serta skenario pengujian yang relevan. Tahap *Plan* berkaitan dengan perencanaan pengujian, meliputi penyusunan *test case* dan konfigurasi lingkungan pengujian. Selanjutnya, tahap *Author* merupakan tahap pembuatan *script automation testing*, baik melalui penulisan *script* secara manual maupun pemanfaatan fitur *record and playback*. Tahap *Organize* berfungsi untuk mengelola dan menyusun artefak pengujian, seperti pengelompokan *test case*, *test suite*, serta pengaturan struktur proyek pengujian agar lebih terorganisir. Tahap *Execute* merupakan proses menjalankan *script automation testing* pada platform yang didukung oleh Katalon

Studio, seperti *web*, *mobile*, *desktop*, dan *API*, serta memantau hasil eksekusi pengujian melalui laporan dan *dashboard* yang dihasilkan [19].

Tahapan tersebut menggambarkan alur kerja *Quality Assurance* dalam merancang, mengembangkan, mengelola, dan mengeksekusi *script automation testing* secara sistematis. Selain itu, Katalon Studio mendukung pengujian pada berbagai jenis aplikasi, meliputi aplikasi berbasis web, mobile, desktop, dan API. Integrasi fitur *AI augmentation* dan *quality intelligence* pada Katalon Studio bertujuan untuk meningkatkan stabilitas, efisiensi, serta kualitas proses pengujian, khususnya dalam pembuatan dan pelaksanaan *script automation testing*.

2.1.6 Artificial Intelligence (AI) dalam Automation Testing

Artificial Intelligence (AI) merupakan simulasi kecerdasan alami dan kecerdasan manusia yang diwujudkan melalui mesin atau sistem komputer, di mana sistem tersebut memiliki kemampuan untuk memperoleh dan menerapkan pengetahuan, melakukan penalaran, pembelajaran, serta pemecahan masalah [21]. Kemampuan AI untuk mempelajari pola dan mengekstraksi informasi dari data dalam konteks pengujian perangkat lunak menjadikannya teknologi yang sangat relevan dalam otomatisasi proses *testing* [22]. Peran AI dalam *automation testing* semakin terlihat signifikan seiring meningkatnya kompleksitas aplikasi modern. Penelitian oleh [23] menunjukkan bahwa teknologi AI dapat mengoptimalkan berbagai tahapan dalam *software testing life cycle*, terutama pada proses pembuatan *test case* dan validasi hasil pengujian.

Pemanfaatan *Large Language Models* (LLM) dalam *automation testing* semakin berkembang, khususnya pada proses *automated test generation*. LLM memiliki kemampuan untuk memahami struktur kode, deskripsi kebutuhan, serta konteks fungsional aplikasi, sehingga dapat digunakan untuk menghasilkan *test case* dan *test script* secara otomatis. Pendekatan ini memungkinkan pembuatan pengujian dilakukan secara lebih cepat dan adaptif, terutama pada tahap awal pengembangan perangkat lunak. Selain itu, LLM juga berperan sebagai alat *rapid test prototyping* yang membantu penguji atau pengembang dalam menyiapkan skenario uji awal

sebelum dilakukan penyempurnaan lebih lanjut secara manual. Kualitas test yang dihasilkan tetap dipengaruhi oleh kompleksitas kode, pemilihan model, serta strategi *prompting*, sehingga validasi dan penyesuaian oleh manusia tetap diperlukan untuk menjamin ketepatan dan relevansi pengujian [24].

Generative Artificial Intelligence (GAI) merupakan salah satu cabang dari kecerdasan buatan yang dirancang untuk menghasilkan data atau konten baru berdasarkan pola yang dipelajari dari data masukan. Berbeda dengan pendekatan AI konvensional yang berfokus pada analisis atau klasifikasi, *Generative AI* mampu menghasilkan keluaran yang bersifat baru namun tetap koheren, relevan secara konteks, serta memiliki kesamaan gaya dan struktur dengan data pelatihan [25]. Konsep *Generative AI* dalam perkembangannya memperluas peran AI dalam *automation testing* dari sekadar sistem diskriminatif menjadi sistem yang bersifat generatif, yaitu mampu menghasilkan artefak baru secara probabilistik berdasarkan distribusi data yang telah dipelajari. Berbeda dengan AI konvensional yang berfokus pada klasifikasi atau pengambilan keputusan, GAI memiliki karakteristik menghasilkan variasi keluaran yang tidak selalu identik untuk input yang sama, termasuk dalam bentuk *test case* dan *test script* [26]. Karakteristik ini memungkinkan proses pengujian menjadi lebih fleksibel dan responsif terhadap perubahan kebutuhan aplikasi, namun juga menuntut mekanisme evaluasi tambahan untuk memastikan konsistensi dan kualitas hasil pengujian.

Kualitas *source code* yang dihasilkan oleh AI juga menjadi perhatian penting dalam *automation testing*. Kualitas kode yang dihasilkan oleh beberapa AI *generative engines* seperti ChatGPT (GPT-3.5), GPT-4, dan Google Bard dengan menggunakan *test suite* dan metrik kualitas perangkat lunak, antara lain *Lines of Code (LOC), Cyclomatic Complexity, dan Cognitive Complexity* umumnya bersifat fungsional dan mampu lolos pengujian otomatis, kualitas struktur dan kompleksitas kode masih bervariasi antar model [27]. Temuan ini menegaskan bahwa *automation testing* berperan penting tidak hanya untuk memvalidasi fungsionalitas, tetapi juga sebagai alat evaluasi kualitas *script* yang dihasilkan oleh AI.

2.1.7 Efektivitas dan Metrik

Berdasarkan model *Quality in Use* pada ISO/IEC 9126-4, efektivitas didefinisikan sebagai kemampuan produk perangkat lunak untuk memungkinkan pengguna mencapai tujuan tertentu dengan tingkat akurasi dan kelengkapan yang sesuai dalam konteks penggunaan yang telah ditentukan [28]. Konsep efektivitas tersebut dalam penelitian ini diadaptasi ke dalam konteks pembuatan *script automation testing*, di mana efektivitas diartikan sebagai kemampuan metode yang digunakan untuk menghasilkan *script* pengujian yang memenuhi kebutuhan pengujian, stabil dalam pelaksanaan, serta meminimalkan kebutuhan perbaikan ulang. Metrik dalam konteks *Quality Assurance* merupakan ukuran kuantitatif yang digunakan untuk mengevaluasi dan membandingkan kinerja suatu proses atau hasil pengujian secara objektif. Oleh karena itu, dalam penelitian ini efektivitas diukur menggunakan tiga metrik utama yang merepresentasikan kualitas proses dan hasil pembuatan *script*, yaitu:

1. Waktu pembuatan *script*, yaitu durasi yang dibutuhkan sejak penulisan awal hingga *script* siap untuk dieksekusi. Satuan yang digunakan dalam metrik ini adalah menit. Metrik ini digunakan karena mencerminkan kemampuan suatu metode dalam mencapai tujuan pengujian dalam waktu yang optimal. Penelitian [1] menunjukkan bahwa waktu merupakan salah satu aspek utama dalam mengevaluasi efektivitas pengujian perangkat lunak. Oleh karena itu, metode yang mampu menghasilkan *script* siap pakai dalam waktu lebih singkat dapat dikatakan lebih efektif.

   Cara perhitungan waktu pembuatan script dilakukan dengan mencatat waktu mulai (*start time*) pada saat penulisan *script* dimulai dan waktu selesai (*end time*) ketika *script* siap dieksekusi pertama kali, kemudian menghitung selisih keduanya sebagai berikut:

   Waktu Pembuatan Script = Waktu Selesai – Waktu Mulai

   Contoh:

   Apabila penulisan script dimulai pada pukul 10.00 dan selesai pada pukul 10.07, maka waktu pembuatan script adalah 7 menit.

2. Jumlah iterasi perbaikan, yang menunjukkan banyaknya siklus *debugging* yang diperlukan hingga *script* dapat dijalankan tanpa *error*. Satuan metrik ini adalah jumlah iterasi (kali perbaikan). Metrik ini merepresentasikan stabilitas dan keberhasilan metode dalam menghasilkan *script* yang benar. Penelitian [9] menjelaskan bahwa efektivitas *debugging* menurun seiring bertambahnya jumlah iterasi perbaikan. Oleh karena itu, metode yang memerlukan lebih sedikit iterasi perbaikan dinilai lebih efektif dalam mencapai tujuan pengujian.

   Cara perhitungan jumlah iterasi perbaikan dilakukan dengan menghitung berapa kali *script* mengalami status *failed* dan harus diperbaiki hingga akhirnya mencapai status *passed*. Setiap satu kali perbaikan dihitung sebagai satu iterasi.

   Contoh: Apabila *script* mengalami kegagalan pada eksekusi pertama, diperbaiki satu kali, lalu berhasil dijalankan pada eksekusi kedua, maka jumlah iterasi perbaikan adalah 1 iterasi.

3. Kompleksitas *script* yang diukur menggunakan jumlah baris kode (*Lines of Code*/LOC) dan *Cyclomatic Complexity (CC)*. Satuan LOC adalah jumlah baris kode, sedangkan CC dinyatakan dalam jumlah jalur eksekusi independen pada *script*. LOC digunakan untuk merepresentasikan ukuran dan kepadatan kode, sedangkan CC digunakan untuk mengukur kompleksitas alur logika berdasarkan jumlah jalur eksekusi independen dalam script. Pemilihan metrik ini didukung oleh penelitian [11] yang menyatakan bahwa kompleksitas kode yang tinggi berdampak negatif terhadap kualitas perangkat lunak. Mengingat kualitas perangkat lunak berkaitan dengan kemampuan sistem dalam berkinerja secara efektif, *script automation testing* dengan kompleksitas yang lebih rendah dinilai lebih efektif karena lebih mudah dipahami, dipelihara, dan dijalankan. Penggunaan LOC dan CC sebagai metrik kompleksitas juga didukung oleh penelitian lain seperti penelitian [10] yang menggunakan *cyclomatic complexity* dan LOC sebagai metrik utama untuk mengukur kompleksitas, serta penelitian [29] yang memasukkan LOC dan *cyclomatic complexity*

sebagai metrik standar dalam analisis kualitas dan prediksi cacat perangkat lunak.

Secara formal, *cyclomatic complexity* merupakan metrik baku yang mengacu pada definisi McCabe dan dapat dihitung menggunakan *Control Flow Graph* (CFG) dengan rumus:

$$\mathbf{CC = E - N + 2}$$ [30]

Di mana *E* menyatakan jumlah *edges* (sisi) dan *N* menyatakan jumlah *nodes* (simpul) pada CFG. Praktik pengukuran pada *script automation testing* yang umumnya memiliki struktur logika sederhana, perhitungan CC dapat disederhanakan dengan menghitung jumlah titik keputusan (*decision point*) seperti struktur *if*, *else*, dan *loop*, kemudian ditambahkan satu jalur dasar. Cara perhitungan LOC dilakukan dengan menghitung jumlah seluruh baris kode yang menyusun *script*, tidak termasuk baris kosong atau komentar.

Contoh: Sebuah *script* dengan 25 baris kode memiliki nilai LOC = 25. Apabila *script* tersebut memiliki dua kondisi *if*, maka nilai *cyclomatic complexity* adalah CC = 3.

Metrik-metrik tersebut digunakan karena *automation testing* bertujuan untuk mengoptimalkan proses pengujian yang seringkali repetitif dan kompleks, terutama pada aplikasi web yang memerlukan kompatibilitas lintas perangkat[18]. Efektivitas pembuatan *script* dalam praktiknya dipengaruhi oleh metode yang digunakan. Metode manual, yang melibatkan penulisan *script* secara langsung oleh manusia, cenderung fleksibel namun rentan terhadap kesalahan manusia dan memakan waktu lebih lama, sehingga efektivitasnya lebih rendah dalam skala besar. Sebaliknya, metode berbasis *Artificial Intelligence* (AI), seperti menggunakan alat seperti ChatGPT untuk generasi *script* otomatis, dapat meningkatkan efektivitas dengan mempercepat waktu pembuatan dan mengurangi kesalahan awal, meskipun memerlukan verifikasi manusia untuk memastikan akurasi [31]. Penelitian [32] menunjukkan bahwa integrasi AI dalam *automation testing* dapat meningkatkan efektivitas hingga 35% dalam hal waktu dan akurasi, dengan catatan bahwa *tool* seperti Katalon Studio memfasilitasi eksekusi yang lebih efisien. Studi ini menekankan bahwa efektivitas tidak hanya bergantung pada

metode, tetapi juga pada kemampuan tool dalam menangani kompleksitas pengujian, seperti variasi perangkat dan antarmuka pengguna.

2.1.8 Kriteria Aplikasi Web yang Mudah Diuji dengan Automation Testing

Pengujian perangkat lunak merupakan aktivitas penting dalam *Software Development Life Cycle* untuk menjamin kualitas dan reliabilitas aplikasi web. Dalam konteks *automation testing*, aplikasi yang mudah diuji memiliki karakteristik tertentu yang mendukung eksekusi *script* secara stabil dan berulang [4]. Salah satu kriteria utama adalah stabilitas antarmuka (UI) serta konsistensi locator elemen. Elemen UI yang memiliki identifikasi unik dan tidak berubah-ubah memungkinkan *script* otomatis menemukan elemen secara akurat. Perubahan DOM atau elemen dinamis dapat menyebabkan *fragile test* dan meningkatkan kebutuhan pemeliharaan *script* [33].

Konsistensi perilaku halaman, termasuk waktu pemuatan yang terprediksi juga sangat berpengaruh terhadap keberhasilan otomatisasi. Aplikasi yang memuat konten secara deterministik akan lebih mudah disinkronkan dengan proses eksekusi *script*. Ketidakpastian dalam *loading* dan permintaan asinkron menjadi salah satu penyebab utama kegagalan otomatisasi (*timeout* atau elemen tidak ditemukan) [33]. Kriteria lainnya adalah alur kerja (*workflow*) yang stabil dan terdokumentasi. Aplikasi dengan pola navigasi yang konsisten misalnya prosedur login atau proses transaksi lebih mudah diotomasi karena langkah-langkahnya dapat direplikasi tanpa penyesuaian terus-menerus. Studi menunjukkan bahwa *workflow* yang terstruktur meminimalkan perubahan *script* serta meningkatkan efektivitas pengujian otomatis [4].

Aplikasi juga sebaiknya meminimalkan proteksi anti-otomatisasi, seperti CAPTCHA atau mekanisme anti-bot yang kompleks. Fitur tersebut biasanya mengharuskan interaksi manual sehingga menghambat proses otomatisasi. Terakhir, aplikasi web dikatakan mudah diuji jika mendukung metode pengujian otomatis, misalnya *black-box testing* berbasis GUI. Aplikasi yang menyediakan atribut aksesibilitas yang jelas seperti label, alt text, dan id element yang konsisten

sehingga memudahkan alat seperti Katalon Studio dan Selenium dalam mendeteksi serta berinteraksi dengan elemen antarmuka [4].

2.2 Penelitian Terdahulu

Penelitian terdahulu yang dilakukan oleh [34] dengan judul "Unit Test Generation using Generative AI: A Comparative Performance Analysis of Auto generation Tools" membandingkan efektivitas pembuatan *script* pengujian unit menggunakan *Large Language Model* (ChatGPT) dengan alat otomasi konvensional (Pynguin). Hasil penelitian menunjukkan bahwa ChatGPT mampu menghasilkan *code coverage* yang setara dengan Pynguin, meskipun tingkat kesalahan yang dihasilkan masih lebih tinggi. Penelitian ini menunjukkan potensi AI dalam membantu pembuatan *script* otomatis, namun belum diterapkan pada pengujian aplikasi menggunakan Katalon Studio.

Penelitian selanjutnya dilakukan oleh [31] dengan judul "AI-Generated Test Scripts for Web E2E Testing with ChatGPT and Copilot: A Preliminary Study." Berbeda dengan penelitian sebelumnya yang berfokus pada pengujian unit, penelitian ini mengkaji pembuatan *script* pengujian *end-to-end* (E2E) pada aplikasi web secara manual dan dengan bantuan AI menggunakan ChatGPT serta GitHub Copilot. Hasil penelitian menunjukkan bahwa penggunaan ChatGPT secara iteratif dapat menghemat waktu pembuatan *script* secara signifikan dibandingkan metode manual. Penelitian tersebut tetapi belum mengukur efektivitas berdasarkan jumlah iterasi perbaikan dan kompleksitas *script*, sehingga masih terdapat ruang untuk penelitian lanjutan yang menilai aspek-aspek tersebut dalam konteks pengujian otomatis menggunakan *tool* seperti Katalon Studio.

Penelitian oleh [32] dengan judul "AI-Powered Software Testing Tools: A Systematic Tool Review and Empirical Assessment of Their Features and Limitations" menilai efektivitas alat uji berbasis AI dalam meningkatkan efisiensi proses otomasi pengujian perangkat lunak. Hasil penelitian menunjukkan bahwa

penggunaan AI mampu mempercepat waktu pengujian dan meningkatkan akurasi deteksi kesalahan, namun tetap memerlukan pengawasan manusia untuk memastikan hasil pengujian yang valid. Studi ini memberikan landasan empiris tentang bagaimana AI dapat meningkatkan efektivitas pengujian perangkat lunak, namun belum melakukan analisis perbandingan secara langsung antara pembuatan *script* manual dan dengan bantuan AI.

Penelitian berikutnya dilakukan oleh [8] dengan judul "Generative AI in Automated Software Testing: A Comparative Study" yang meneliti perbandingan efektivitas alat pengujian otomatis konvensional seperti Selenium dan JUnit dengan alat berbasis AI seperti GPT-4 dan Codex. Penelitian ini menggunakan indikator waktu pembuatan *script*, akurasi, serta *test coverage*. Hasil penelitian menunjukkan bahwa penggunaan AI mampu mengurangi waktu pengujian hingga 40% tanpa mengurangi akurasi hasil uji. Penelitian tersebut namun belum mengukur jumlah iterasi perbaikan serta kompleksitas *script*, sehingga penelitian ini masih berpotensi dikembangkan lebih lanjut dalam konteks pengukuran efektivitas pembuatan *script* yang lebih komprehensif, khususnya pada aplikasi berbasis web.

Penelitian yang dilakukan oleh [35] dengan judul "LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities" meneliti penerapan *Large Language Models (LLM)* seperti ChatGPT dalam pembuatan serta migrasi *script* pengujian otomatis. Penelitian ini menguji kemampuan *LLM* dalam tiga aspek utama, yaitu pembuatan *script* berbasis skenario (*scenario-based test generation*), migrasi lintas platform (Android dan iOS), serta migrasi antar-aplikasi dengan fungsi serupa. Hasil penelitian menunjukkan bahwa *LLM* mampu menghasilkan *script* pengujian dengan struktur logis yang sesuai dan mampu memahami konteks bisnis aplikasi secara mendalam. Studi ini memperlihatkan potensi besar *LLM* dalam mendukung otomatisasi pengujian pada aplikasi *mobile*. Penelitian ini masih terbatas pada pengujian aplikasi *mobile* dan belum diterapkan pada aplikasi berbasis web, sehingga masih terdapat ruang untuk penelitian yang

memperluas konteks pengujian aplikasi web menggunakan *tool* seperti Katalon Studio.

Penelitian selanjutnya dilakukan oleh [24] dengan judul "Automated Test Generation Using Large Language Models" yang mengkaji kemampuan *Large Language Models* (LLM) dalam menghasilkan *test case* dan *test script* secara otomatis berdasarkan deskripsi kebutuhan pengujian. Penelitian ini menunjukkan bahwa LLM mampu memahami konteks fungsional aplikasi dan menghasilkan skenario pengujian yang relevan tanpa memerlukan penulisan script secara manual dari awal. Hasil penelitian juga mengindikasikan bahwa penggunaan LLM dapat mempercepat proses pembuatan artefak pengujian serta mengurangi beban kerja penguji. Penelitian tersebut masih berfokus pada proses generasi *test* secara umum dan belum mengevaluasi efektivitas pembuatan script berdasarkan metrik kuantitatif seperti waktu pembuatan, jumlah iterasi perbaikan, dan kompleksitas script, serta belum diterapkan pada *tool automation testing* tertentu seperti *Katalon Studio*. Oleh karena itu, penelitian ini membuka peluang untuk dilakukan pengembangan lebih lanjut dalam konteks pengujian aplikasi berbasis web menggunakan *tool* otomasi yang spesifik.

Penelitian selanjutnya dilakukan oleh [5] dengan judul "An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation". Penelitian ini memperkenalkan TESTPILOT, sebuah pendekatan adaptif berbasis *Large Language Model* untuk menghasilkan *unit test* secara otomatis tanpa memerlukan *fine-tuning* atau dataset pelatihan khusus. TESTPILOT memanfaatkan informasi kontekstual berupa *signature* fungsi, dokumentasi, contoh penggunaan, serta kode sumber untuk menghasilkan *unit test*, dan secara adaptif memperbaiki *test* yang gagal melalui proses *re-prompting*. Hasil evaluasi pada 25 paket npm menunjukkan bahwa TESTPILOT mampu mencapai *statement coverage* dan *branch coverage* yang lebih tinggi dibandingkan metode konvensional seperti Nessie. Penelitian ini belum membahas pembuatan *script automation testing* untuk pengujian aplikasi berbasis web maupun penerapannya pada *tool automation testing* seperti Katalon Studio meskipun menunjukkan performa yang unggul

dalam konteks pengujian unit, sehingga masih terdapat peluang penelitian lanjutan pada konteks tersebut.

Penelitian lain yang relevan dilakukan oleh [27] dengan judul "Studying the Quality of Source Code Generated by Different AI Generative Engines: An Empirical Evaluation". Penelitian ini mengevaluasi kualitas *source code* yang dihasilkan oleh berbagai AI generative engines berbasis *Large Language Model*, seperti ChatGPT (GPT-3.5), GPT-4, dan Google Bard. Evaluasi dilakukan menggunakan *automation testing* melalui *test suite* serta metrik kualitas kode, termasuk *Lines of Code (LOC)*, *Cyclomatic Complexity*, *dan Cognitive Complexity*. Hasil penelitian menunjukkan bahwa meskipun AI mampu menghasilkan kode yang fungsional dan lulus pengujian otomatis, kualitas struktur dan kompleksitas kode masih bervariasi dan memerlukan validasi lebih lanjut. GPT-4 cenderung menghasilkan kode yang lebih ringkas dan memiliki kompleksitas yang lebih rendah dibandingkan model lainnya. Penelitian tersebut berfokus pada evaluasi *source code* aplikasi secara umum dan belum secara spesifik membahas *script automation testing* dalam konteks aktivitas *Quality Assurance*, yang memiliki karakteristik berbeda karena merepresentasikan skenario pengujian dan interaksi antarmuka pengguna. Oleh karena itu, penelitian belum membandingkan efektivitas pembuatan *script automation testing* secara manual dan berbantuan AI pada *tool* tertentu seperti Katalon Studio, sehingga membuka peluang penelitian lanjutan.

Penelitian lain yang relevan dilakukan oleh [36] dengan judul "An empirical study of automated unit test generation for Python". Penelitian ini mengevaluasi efektivitas berbagai teknik *automated unit test generation* pada bahasa pemrograman Python dengan meninjau aspek kualitas *test case*, *code coverage*, serta stabilitas hasil pengujian. Hasil penelitian menunjukkan bahwa *automated test generation* mampu menghasilkan *unit test* dengan tingkat *coverage* yang tinggi, namun masih menghadapi tantangan dalam menghasilkan *test* yang mudah dipelihara dan sepenuhnya relevan dengan kebutuhan fungsional aplikasi. Studi ini menegaskan bahwa meskipun otomasi generasi *test* dapat meningkatkan efisiensi

pengujian, keterlibatan manusia masih diperlukan untuk validasi dan penyempurnaan hasil. Penelitian tersebut berfokus pada pengujian unit dan belum mengkaji pembuatan *test script* untuk pengujian aplikasi web secara *end-to-end* maupun implementasinya pada *tool automation testing* seperti Katalon Studio, sehingga membuka peluang penelitian lanjutan pada konteks tersebut.

Penelitian yang dilakukan oleh [37] dengan judul "Enhancing AI-Generated Code via Automated Evaluation and Validation" berfokus pada peningkatan kualitas *script* yang dihasilkan AI melalui mekanisme evaluasi dan validasi otomatis. Penelitian ini menemukan bahwa kombinasi AI dengan sistem validasi berbasis metrik mampu meningkatkan ketepatan hasil dan mengurangi kesalahan logika pada *script* pengujian. Penelitian ini masih bersifat konseptual dan belum menguji efektivitas pada platform tertentu seperti Katalon Studio. Penelitian ini dapat menjadi acuan dalam mengembangkan sistem pembuatan *script* yang menggabungkan kemampuan AI dan validasi manusia untuk meningkatkan efektivitas hasil pengujian.

Penelitian terakhir yang dilakukan oleh [38] dengan judul "Optimizing Case-Based Reasoning System for Functional Test Script Generation with Large Language Models" menjelaskan kemampuan dan keterbatasan AI dalam membantu proses otomasi pengujian perangkat lunak. Penelitian ini menunjukkan bahwa AI dapat mempercepat proses pembuatan *script* serta meningkatkan efisiensi kerja penguji, namun tetap membutuhkan pengawasan manusia dalam validasi hasil pengujian. Studi ini menegaskan bahwa kolaborasi antara AI dan manusia merupakan pendekatan yang efektif dalam menciptakan *script* pengujian yang akurat dan efisien, meskipun belum diterapkan dalam konteks tool spesifik seperti Katalon Studio.

Berdasarkan berbagai penelitian tersebut, dapat disimpulkan bahwa integrasi *Artificial Intelligence* dalam *automation testing* terbukti memiliki potensi besar dalam meningkatkan efisiensi waktu dan akurasi pembuatan *script*. Namun, sebagian besar penelitian terdahulu masih berfokus pada pengujian unit atau

pengujian berbasis web tertentu dan belum menyentuh konteks pengujian aplikasi web menggunakan Katalon Studio secara menyeluruh. Oleh karena itu, penelitian ini memiliki kontribusi baru dengan menganalisis efektivitas pembuatan *script automation testing* secara manual dan dengan bantuan *AI* dalam konteks pengujian aplikasi berbasis web menggunakan Katalon Studio, serta menilai hasilnya berdasarkan indikator waktu pembuatan, akurasi, dan keberhasilan eksekusi.

III. METODOLOGI PENELITIAN

3.1 Waktu dan Tempat Penelitian

3.1.1 Waktu Penelitian

Pembagian waktu penelitian dapat dilihat pada tabel berikut.

Tabel 3.1 Waktu Penelitian

No	Nama Kegiatan	Nov	Des	Jan	Feb	Mar	Apr
1	Studi Literatur						
2	Perancangan rencana pengujian						
3	Pembuatan *script*						
4	Pengujian						
5	Pengumpulan hasil dan Analisis						
6	Penulisan Laporan Ilmiah						

Berdasarkan Tabel 3.1, waktu penelitian dibagi ke dalam beberapa tahapan kegiatan yang meliputi studi literatur, perancangan rencana pengujian, pembuatan *script*, pengujian, pengumpulan dan analisis data, serta penulisan laporan ilmiah yang dilaksanakan secara bertahap.

3.1.2 Tempat Penelitian

Penelitian ini dilaksanakan di Program Studi Teknik Informatika, Universitas Lampung, Jl. Prof. Dr. Ir. Sumantri Brojonegoro No.1, Gedong Meneng, Kec. Rajabasa, Kota Bandar Lampung, Lampung.

3.2 Alat dan Bahan Penelitian

3.2.1 Alat Penelitian

Alat yang digunakan selama penelitian terdapat 2 jenis, yaitu perangkat keras (*Hardware*) dan perangkat lunak (*Software)* yang dapat dilihat di tabel berikut.

Tabel 3.2 Perangkat Keras yang Digunakan

No	Perangkat Keras	Spesifikasi	Kegunaan
1	*Laptop*	Merk Lenovo ideapad 3 14llL05, Intel Core i5-1035G1, CPU @ 1.00GHz, 12 GB, Windows 11	Digunakan untuk membuat, menjalankan, dan menguji *script automation testing* pada Katalon Studio serta penulisan laporan penelitian.
2	*Mouse*	-	Digunakan untuk menggerakkan kursor.

Berdasarkan Tabel 3.2, perangkat keras yang digunakan dalam penelitian ini berfungsi untuk mendukung proses pembuatan dan pengujian *script automation testing* serta penulisan laporan penelitian.

Tabel 3.3 Perangkat Lunak yang Digunakan

No	Perangkat Lunak	Versi	Kegunaan
1	Katalon Studio	10.0	Digunakan untuk melakukan pengujian/menjalankan *script automation testing*.
2	ChatGPT	5.2	Digunakan untuk membantu menghasilkan *script* pengujian sesuai skenario uji.
3	Microsoft Excel	2024	Digunakan untuk pencatatan dan analisis data hasil pengujian.
4	Draw.io	-	Digunakan untuk membuat diagram alur tahapan penelitian.

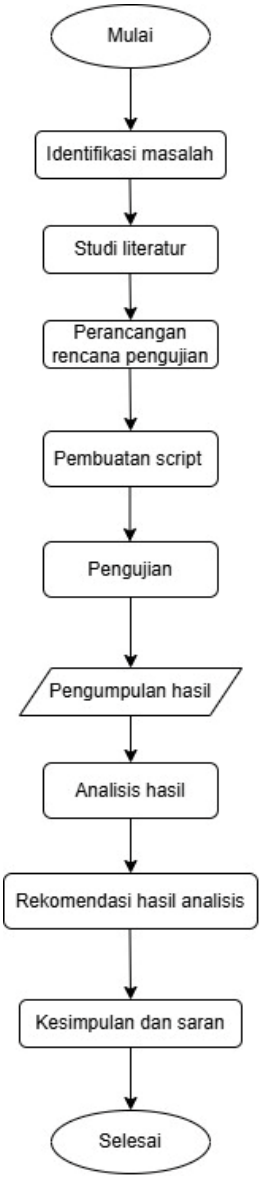
No	Perangkat Lunak	Versi	Kegunaan
5	Google Chrome	142.0.7444.176	Digunakan untuk membuka dan menjalankan website yang diuji.

Berdasarkan Tabel 3.3, perangkat lunak yang digunakan dalam penelitian ini mendukung pelaksanaan *automation testing*, pembuatan *script* berbasis *Artificial Intelligence* (AI), pencatatan dan analisis data, serta perancangan diagram tahapan penelitian.

3.2.2 Bahan Penelitian

Bahan penelitian dalam studi ini berupa aplikasi yang dijadikan objek pengujian, yaitu *website* Sistem Informasi Geografis Universitas Lampung, GitHub, dan OpenStreetMap. *Website* Sistem Informasi Geografis Universitas Lampung dapat diakses melalui https://sig-unila.vercel.app/. *Website* ini merupakan aplikasi berbasis Web GIS yang dikembangkan dalam penelitian sebelumnya oleh mahasiswa Universitas Lampung sehingga berstatus sebagai *prototype* untuk kepentingan akademis dan bukan sebagai sistem resmi institusi. *Website* GitHub dapat diakses melalui https://github.com/. *Website* OpenStreetMap dapat diakses melalui https://www.openstreetmap.org/. Ketiga *website* tersebut digunakan sebagai media uji dalam pelaksanaan *automation testing* untuk mengevaluasi perbedaan karakteristik *script* yang dibuat secara manual dan menggunakan pendekatan *Artificial Intelligence* (AI). Penggunaan beberapa *website* dengan variasi kompleksitas fitur bertujuan agar hasil penelitian dapat merepresentasikan kondisi pengujian *automation testing* pada berbagai jenis aplikasi web secara lebih luas dan beragam.

3.3 Tahapan Penelitian



Gambar 3.1 Alur Tahapan Penelitian

Gambar 3.1 merupakan alur atau tahapan yang dilakukan selama penelitian berlangsung. Tahapan tersebut terdiri dari perancangan rencana pengujian, pembuatan *script*, pengujian dan eksekusi *script*, pengumpulan data, dan analisis.

3.3.1 Perancangan Rencana Pengujian

Perancangan rencana pengujian merupakan tahap awal dalam penelitian ini yang bertujuan untuk menyusun panduan pelaksanaan pengujian perangkat lunak secara sistematis. Tahap ini ditentukan objek pengujian, pendekatan pengujian, lingkup

pengujian, serta fitur-fitur yang akan diuji. Objek pengujian pada penelitian ini terdiri atas tiga *website* berbasis web, yaitu *website* Sistem Informasi Geografis Universitas Lampung, GitHub, dan OpenStreetMap.

Website Sistem Informasi Geografis Universitas Lampung merupakan aplikasi berbasis Web GIS yang menyediakan informasi spasial fasilitas kampus melalui peta interaktif dan antarmuka dinamis. *Website* ini memungkinkan pengguna melakukan pencarian, penyaringan, dan pengurutan fasilitas, berinteraksi dengan marker peta, mengakses informasi rinci fasilitas, serta mengirimkan pesan melalui formulir kontak.

GitHub digunakan sebagai objek pengujian untuk merepresentasikan aplikasi berbasis repository management dengan fitur pencarian repository, pengelolaan issue, dan pengaksesan *file* repository secara daring. *Website* ini memiliki karakteristik antarmuka dinamis dengan interaksi pengguna yang cukup kompleks. OpenStreetMap digunakan sebagai objek pengujian untuk merepresentasikan aplikasi berbasis pemetaan dan navigasi. *Website* ini memungkinkan pengguna melakukan pencarian lokasi, perhitungan rute perjalanan, pengaturan tampilan peta, berbagi lokasi, serta pergantian moda transportasi. Penggunaan ketiga website tersebut bertujuan agar penelitian dapat mencakup variasi fitur, tingkat kompleksitas, serta karakteristik interaksi pengguna yang lebih beragam dalam proses *automation testing*.

3.3.1.a Pendekatan Pengujian

Pengujian dilakukan dengan metode *black-box testing*, yang berfokus pada pengujian fungsionalitas fitur berdasarkan masukan (*input*) dan keluaran (*output*) tanpa memeriksa kode sumbernya.

3.3.1.b Lingkup Pengujian

Lingkup pengujian pada penelitian ini mencakup fitur-fitur utama pada *website* Sistem Informasi Geografis Universitas Lampung, GitHub, dan OpenStreetMap yang digunakan oleh pengguna umum dalam melakukan interaksi dengan sistem.

Pengujian difokuskan pada fitur yang merepresentasikan proses pencarian data, navigasi halaman, validasi informasi, interaksi dengan elemen dinamis, serta integrasi layanan berbasis web dan geospasial. Fitur-fitur yang diuji dipilih berdasarkan tingkat frekuensi penggunaan, kompleksitas logika aplikasi, serta keterlibatannya dalam proses utama sistem. Penggunaan beberapa *website* dengan karakteristik yang berbeda bertujuan agar pengujian mampu merepresentasikan berbagai kondisi automation testing pada aplikasi web modern.

Fitur-fitur yang akan diuji adalah sebagai berikut.

Tabel 3.4 Fitur-Fitur yang akan Diuji

No.	Fitur	Keterangan
1	Pencarian Fasilitas	Fitur yang memungkinkan pengguna mencari fasilitas Universitas Lampung berdasarkan kata kunci tertentu. Sistem menampilkan daftar fasilitas yang sesuai secara dinamis berdasarkan input pengguna.
2	Filter Fasilitas Berdasarkan Fakultas	Fitur yang memungkinkan pengguna memfilter fasilitas berdasarkan fakultas yang tersedia di Universitas Lampung, sehingga hanya fasilitas yang relevan yang ditampilkan pada antarmuka.
3	Pengurutan Fasilitas (A–Z dan Z–A)	Fitur yang memungkinkan pengguna mengurutkan daftar fasilitas berdasarkan abjad dari A–Z atau Z–A untuk memudahkan pencarian informasi.
4	Interaksi Marker pada Peta	Fitur yang memungkinkan pengguna berinteraksi dengan marker fasilitas pada peta, termasuk memilih aksi untuk melihat rincian fasilitas atau melakukan navigasi ke lokasi.
5	Rincian Fasilitas	Fitur yang menampilkan informasi detail fasilitas, meliputi deskripsi, jam operasional, akses pengguna, serta tautan navigasi ke Google Maps.
6	Formulir Kontak Kami	Fitur yang memungkinkan pengguna mengirimkan pesan kepada pengelola sistem dengan mengisi nama, email, subjek,

No.	Fitur	Keterangan
		dan isi pesan, termasuk proses validasi dan pengiriman data.
7	Tentang *Website*	Fitur yang memungkinkan pengguna untuk melihat deskripsi, manfaat, dan tujuan dari *Website* Sistem Informasi Geografis Unila.
8	*GitHub Issue Tracker*	Fitur pengelolaan dan pencarian issue pada repository GitHub.
9	*OpenStreetMap Routing*	Fitur pencarian dan perhitungan rute perjalanan pada OpenStreetMap.
10	*Share OpenStreetMap*	Fitur berbagi lokasi atau tampilan peta melalui tautan.
11	*Layer OpenStreetMap*	Fitur pergantian layer atau tampilan peta pada OpenStreetMap.
12	*Copy Share Link*	Fitur menyalin tautan *share* lokasi atau peta.
13	Pergantian Moda Transportasi	Fitur pergantian moda transportasi pada navigasi OpenStreetMap.
14	Download Repository ZIP	Fitur mengunduh repository GitHub dalam format ZIP.
15	Download Raw File GitHub	Fitur mengakses atau mengunduh file mentah (*raw file*) pada GitHub.

Berdasarkan Tabel 3.4, fitur-fitur yang diuji dalam penelitian ini mencakup fitur utama dari tiga *website* yang memiliki karakteristik berbeda, yaitu *website* Sistem Informasi Geografis Universitas Lampung, GitHub, dan OpenStreetMap. Fitur-fitur tersebut merepresentasikan berbagai jenis interaksi pengguna, mulai dari pencarian data, navigasi halaman, interaksi peta, pengelolaan repository, hingga layanan navigasi berbasis geospasial. Pemilihan fitur dilakukan untuk memastikan bahwa pengujian mencakup fungsi dengan tingkat interaksi pengguna dan kompleksitas sistem yang beragam, sehingga dapat digunakan untuk mengevaluasi perbedaan karakteristik *script automation testing* yang dibuat secara manual maupun menggunakan pendekatan *Artificial Intelligence* (AI).

3.3.1.c Kompleksitas Fitur Menggunakan Function Point Analysis

Kompleksitas fitur dalam penelitian ini ditentukan menggunakan *Function Point Analysis* (FPA). FPA merupakan metode pengukuran perangkat lunak yang berfokus pada ukuran dan kompleksitas fungsional sistem berdasarkan fungsi yang diterima oleh pengguna, tanpa bergantung pada bahasa pemrograman maupun jumlah baris kode. Pendekatan ini banyak digunakan dalam penelitian untuk mengukur ukuran fungsional sistem secara objektif, termasuk dalam konteks analisis dan evaluasi perangkat lunak berbasis web[39].

Penentuan kompleksitas fitur dilakukan untuk memberikan konteks terhadap perbedaan hasil pengukuran efektivitas pembuatan *script automation testing* antara metode manual dan metode berbasis *Artificial Intelligence* (AI). Meskipun objek penelitian hanya terdiri dari satu sistem, yaitu Sistem Informasi Geografis Unila, setiap fitur yang diuji memiliki karakteristik fungsional yang berbeda, baik dari sisi alur proses, validasi data, maupun interaksi dengan basis data. Perbedaan karakteristik tersebut berpotensi memengaruhi tingkat kesulitan pembuatan *script automation testing*.

Kompleksitas fungsional sistem dalam FPA ditentukan dengan mengidentifikasi lima jenis fungsi utama, yaitu *External Input* (EI), *External Output* (EO), *External Inquiry* (EQ), *Internal Logical File* (ILF), dan *External Interface File* (EIF). Identifikasi kelima jenis fungsi tersebut pada penelitian ini dilakukan pada masing-masing fitur yang diuji. Jumlah dan kombinasi fungsi EI, EO, EQ, ILF, dan EIF yang terdapat pada suatu fitur digunakan untuk menghitung nilai *Crude Function Point* (CFP) fitur tersebut. Nilai CFP kemudian digunakan untuk mengelompokkan fitur ke dalam tiga tingkat kompleksitas, yaitu rendah (*simple*), sedang (*average*), dan tinggi (*complex*). Klasifikasi kompleksitas fitur dalam penelitian ini ditentukan berdasarkan rentang nilai CFP, yaitu fitur dengan nilai CFP ≤ 3 dikategorikan sebagai kompleksitas rendah (*simple*), fitur dengan nilai CFP antara 4 hingga 5 dikategorikan sebagai kompleksitas sedang (*average*), dan fitur dengan nilai CFP $\geq$ 6 dikategorikan sebagai kompleksitas tinggi (*complex*). Pengelompokan tingkat

kompleksitas ini mengacu pada prinsip klasifikasi kompleksitas fungsional sebagaimana diterapkan dalam penelitian FPA pada sistem informasi sejenis[40].

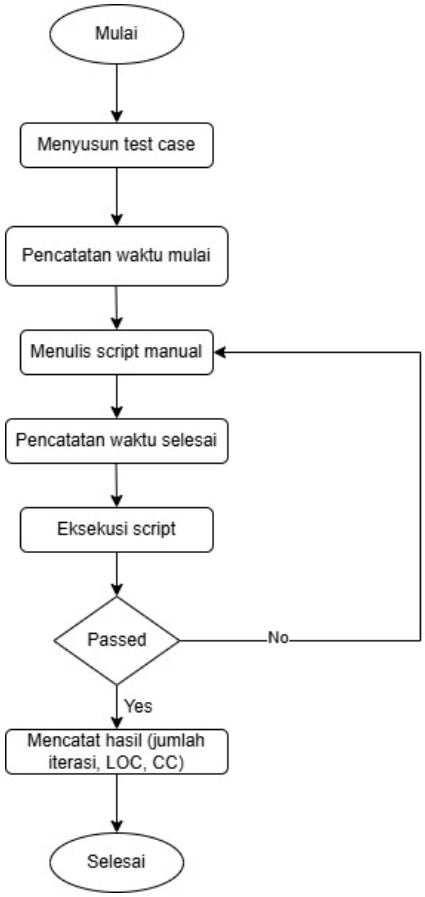
Pengelompokan kompleksitas fitur dalam penelitian ini tidak digunakan untuk membandingkan tingkat kesulitan antar fitur, melainkan digunakan sebagai konteks analisis dalam membandingkan efektivitas metode pembuatan *script automation testing* secara manual dan berbasis AI pada fitur yang sama. Oleh karena itu, perbedaan nilai waktu pembuatan *script*, jumlah iterasi perbaikan, serta kompleksitas *script* yang diukur menggunakan *Lines of Code* (LOC) dan *Cyclomatic Complexity* (CC) dapat dianalisis secara lebih objektif dan terkontrol.

3.3.2 Pembuatan *Script*

Tahapan pembuatan *script* merupakan inti implementasi penelitian, di mana *script automation testing* dikembangkan berdasarkan skenario pengujian yang telah dirancang. Proses ini dilakukan dengan dua metode berbeda yaitu manual dan berbasis AI untuk memungkinkan perbandingan efektivitas.

3.3.2.a Pembuatan Script Manual

Script dengan metode manual ditulis langsung di Katalon Studio menggunakan bahasa *Groovy* tanpa bantuan alat eksternal. Peneliti menerjemahkan setiap langkah *test case* ke dalam perintah *script*, termasuk penentuan objek uji (misalnya, elemen UI seperti tombol atau *field input*), aksi (seperti klik atau *input* teks), verifikasi hasil, dan penanganan kondisi *error*.

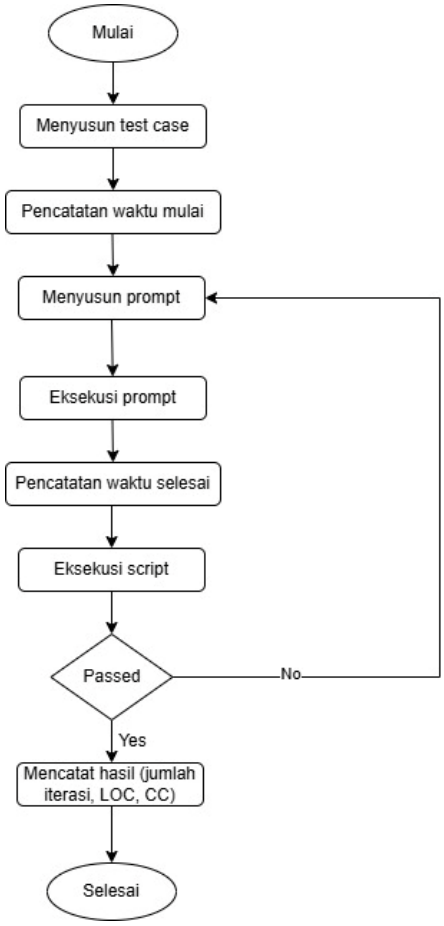


Gambar 3III.2 Diagram Alir Pembuatan Script Manual

Berdasarkan Gambar 3.2 Diagram Alir Pembuatan *Script* Manual, proses pembuatan *script* dimulai dengan penyusunan *test case* untuk setiap fitur yang akan diuji, kemudian dilakukan pencatatan waktu mulai sebagai dasar pengukuran durasi pembuatan *script*. Selanjutnya, peneliti menuliskan *script automation testing* secara manual di Katalon Studio menggunakan bahasa pemrograman *Groovy* dengan menerjemahkan setiap langkah pada *test case* ke dalam perintah pengujian. Setelah proses penulisan *script* selesai, dilakukan pencatatan waktu selesai untuk mengetahui total waktu yang dibutuhkan dalam pembuatan *script* manual. *Script* yang telah dibuat kemudian dieksekusi untuk memverifikasi kesesuaian dengan skenario pengujian. Apabila hasil eksekusi menunjukkan status *failed*, dilakukan analisis untuk mengidentifikasi penyebab kegagalan. Kegagalan yang terjadi pada tahap eksekusi langkah-langkah *test case*, seperti kegagalan menemukan objek atau kesalahan logika *script*, dikategorikan sebagai kegagalan *script* dan menjadi dasar dilakukannya perbaikan serta iterasi ulang hingga memperoleh status *passed*.

Sebaliknya, apabila seluruh langkah *test case* berhasil dieksekusi namun hasil verifikasi akhir tidak sesuai dengan *output* yang diharapkan, maka kegagalan tersebut dikategorikan sebagai *defect* pada aplikasi yang diuji, bukan sebagai kesalahan pada *script automation testing*. *Script* dalam kondisi ini tidak dilakukan perbaikan lebih lanjut, dan *script* dianggap *passed*. Setelah *script* dinyatakan berhasil, peneliti mencatat hasil pengujian berupa jumlah iterasi perbaikan serta kompleksitas *script.*

3.3.2.b Pembuatan Script Berbasis AI



Gambar 3.3 Diagram Alir Pembuatan Script Berbasis AI

Berdasarkan Gambar 3.3 Diagram Alir Pembuatan *Script* Berbasis AI, proses pembuatan *script* diawali dengan penyusunan *test case* untuk setiap fitur yang akan diuji dan pencatatan waktu mulai. Tahapan selanjutnya adalah penyusunan *prompt* yang berisi deskripsi skenario pengujian serta langkah-langkah *test case*. *Prompt*

tersebut kemudian dieksekusi menggunakan ChatGPT untuk menghasilkan *script automation testing* awal. Dilakukan pencatatan waktu selesai setelah *script* dihasilkan, sebagai dasar pengukuran durasi pembuatan *script* berbasis AI. *Script* kemudian dieksekusi menggunakan Katalon Studio untuk memverifikasi keberhasilan pengujian. Apabila hasil eksekusi menunjukkan status *failed*, dilakukan analisis untuk mengidentifikasi penyebab kegagalan. Kegagalan yang terjadi pada tahap eksekusi langkah-langkah *test case*, seperti kegagalan menemukan objek atau kesalahan logika *script*, dikategorikan sebagai kegagalan *script* dan menjadi dasar dilakukannya perbaikan serta iterasi ulang hingga memperoleh status *passed*. Sebaliknya, apabila seluruh langkah *test case* berhasil dieksekusi namun hasil verifikasi akhir tidak sesuai dengan *output* yang diharapkan, maka kegagalan tersebut dikategorikan sebagai *defect* pada aplikasi yang diuji, bukan sebagai kesalahan pada *script automation testing*. *Script* dalam kondisi ini tidak dilakukan perbaikan lebih lanjut, dan *script* dianggap *passed*. Setelah *script* dinyatakan berhasil, peneliti mencatat hasil pengujian berupa jumlah iterasi perbaikan serta kompleksitas *script.*

Peneliti sebagai bagian dari instrumen penelitian pada metode pembuatan *script* berbasis AI, menyusun sebuah *prompt* tetap (*fixed prompt*) yang digunakan secara konsisten untuk menghasilkan *script automation testing* menggunakan ChatGPT. *Prompt* ini dirancang untuk merepresentasikan skenario pengujian secara konseptual tanpa mendefinisikan detail teknis implementasi secara spesifik.

Prompt yang digunakan dalam penelitian ini disusun berdasarkan prinsip *prompt engineering* dengan pendekatan *structured instruction prompt*. *Prompt* diawali dengan penetapan peran (*role assignment*) sebagai *automation testing engineer*, diikuti dengan penyajian konteks aplikasi dan fitur yang diuji, instruksi tugas yang eksplisit, skenario pengujian yang terstruktur, serta pembatasan keluaran berupa bahasa pemrograman dan perangkat bantu yang digunakan. Struktur *prompt* tersebut dirancang bersifat konseptual dan tidak terlalu spesifik terhadap detail implementasi teknis tertentu guna menghindari *over-specification* dan menjaga kemampuan generalisasi model. Perancangan *prompt* ini mengacu pada *best*

practices prompt engineering sebagaimana dijelaskan oleh [41] yang menekankan pentingnya kejelasan peran, konteks, tujuan, dan batasan keluaran dalam menghasilkan respons model yang konsisten dan relevan. Oleh karena itu, *prompt* dalam penelitian ini diperlakukan sebagai instrumen penelitian yang digunakan secara terkontrol dan konsisten pada seluruh proses pembuatan *script automation testing* berbantuan AI. *Prompt* yang digunakan dituliskan dalam bentuk teks utuh sebagaimana digunakan pada saat interaksi dengan ChatGPT, dan disajikan sebagai berikut.

Anda bertindak sebagai automation testing engineer dengan tingkat kemampuan entry-level hingga intermediate.
Buatlah script automation testing menggunakan Katalon Studio untuk aplikasi berbasis web dengan pendekatan black-box testing. Aplikasi yang diuji adalah Website Sistem Informasi Geografis (SIG) Universitas Lampung, yaitu aplikasi Web GIS yang menyediakan peta interaktif dan informasi fasilitas di lingkungan Universitas Lampung. Fitur yang diuji adalah fitur pencarian fasilitas, yang memungkinkan pengguna mencari fasilitas berdasarkan kata kunci tertentu. Skenario pengujian adalah pencarian fasilitas dengan kata kunci valid, dengan langkah-langkah sebagai berikut.

1. *Pengguna membuka halaman utama website Sistem Informasi Geografis Universitas Lampung.*
2. *Pengguna mengakses fitur pencarian fasilitas.*
3. *Pengguna memasukkan kata kunci fasilitas pada kolom pencarian.*
4. *Sistem menampilkan daftar fasilitas yang sesuai dengan kata kunci yang dimasukkan.*

Data uji yang digunakan merupakan kata kunci valid yang sesuai dengan data fasilitas yang tersedia pada sistem. Hasil yang diharapkan adalah sistem berhasil menampilkan daftar fasilitas yang relevan sesuai dengan input pengguna. Buatlah script automation testing yang dapat dijalankan pada Katalon Studio menggunakan bahasa Groovy, memanfaatkan built-in keywords Katalon, serta menggunakan Object Repository untuk pengelolaan elemen uji. Script harus

bersifat umum, stabil, dan tidak terlalu spesifik terhadap implementasi teknis tertentu, serta mampu dijalankan tanpa error dalam sekali eksekusi.
Dalam pembuatan script, gunakan struktur Object Repository dan locator XPath yang telah ditentukan berikut ini dan jangan memodifikasi locator yang diberikan:

Prompt tersebut digunakan sebagai *fixed prompt* dan diterapkan secara konsisten pada seluruh *test case* dalam penelitian ini dengan penyesuaian pada konteks fitur dan skenario pengujian.

Berikut disajikan contoh *test case* dan *script automation testing*. Contoh ini digunakan sebagai ilustrasi proses penelitian dan tidak mewakili keseluruhan data pengujian.

Tabel 3.5 Contoh *Test Case*

Elemen	Deskripsi
TC ID	TC-RG-01
Fitur	Register
Skenario	Registrasi pengguna baru dengan data valid
Langkah Uji	1. Buka halaman registrasi 2. Isi nama lengkap pada form registrasi 3. Isi alamat email valid 4. Isi password 5. Isi konfirmasi password 6. Klik tombol “Daftar”
Input	Data Valid
Output yang diharapkan	Sistem menampilkan notifikasi “Registrasi berhasil”
Status	*Passes/Failed*

Berdasarkan Tabel 3.5, *test case* TC-RG-01 disusun untuk merepresentasikan skenario registrasi pengguna baru dengan data valid. *Test case* ini digunakan sebagai acuan dalam pembuatan *script automation testing* baik secara manual

maupun berbasis AI, sehingga memastikan kesesuaian antara skenario pengujian, input yang diberikan, serta output yang diharapkan.

```
import static com.kms.katalon.core.testobject.ObjectRepository.findTestObject
import com.kms.katalon.core.webui.keyword.WebUiBuiltInKeywords as WebUI
WebUI.openBrowser('')
WebUI.navigateToUrl('https://monitoring-skripsi-perikanan.ac.id/register')

// isi nama lengkap
WebUI.setText(findTestObject('Register/input_namaLengkap'), 'Mahasiswa Uji')

// isi email
WebUI.setText(findTestObject('Register/input_email'), 'mahasiswa.uji@univ.ac.id')

// isi password
WebUI.setText(findTestObject('Register/input_password'), 'Password123')

// isi konfirmasi password
WebUI.setText(findTestObject('Register/input_konfirmasiPassword'), 'Password123')

// klik tombol daftar
WebUI.click(findTestObject('Register/button_daftar'))

// verifikasi registrasi berhasil
WebUI.verifyTextPresent('Registrasi berhasil', false)
WebUI.closeBrowser()
```

Gambar 3.4 Contoh Script Manual

Berdasarkan Gambar 3.4, *script automation testing* manual ditulis secara langsung oleh peneliti dengan menerjemahkan setiap langkah pada *test case* ke dalam perintah pengujian di Katalon Studio.

```
import static com.kms.katalon.core.testobject.ObjectRepository.findTestObject
import com.kms.katalon.core.webui.keyword.WebUiBuiltInKeywords as WebUI
import com.kms.katalon.core.model.FailureHandling

// Open browser
WebUI.openBrowser('')
WebUI.maximizeWindow()

// Navigate to register page
WebUI.navigateToUrl('https://monitoring-skripsi-perikanan.ac.id/register')

// Verify register page loaded
if (WebUI.verifyElementPresent(findTestObject('Register/input_namaLengkap'), 10, FailureHandling.OPTIONAL)) {

    WebUI.setText(findTestObject('Register/input_namaLengkap'), 'Mahasiswa Uji')
    WebUI.setText(findTestObject('Register/input_email'), 'mahasiswa.uji@univ.ac.id')
    WebUI.setText(findTestObject('Register/input_password'), 'Password123')
    WebUI.setText(findTestObject('Register/input_konfirmasiPassword'), 'Password123')

    WebUI.click(findTestObject('Register/button_daftar'))

} else {
    WebUI.comment('Halaman registrasi tidak berhasil dimuat.')
    WebUI.closeBrowser()
    assert false : 'Register page not found'
}

// Wait for response
WebUI.delay(2)

// Validate registration result
boolean isSuccess = WebUI.verifyTextPresent('Registrasi berhasil', false, FailureHandling.OPTIONAL)

if (!isSuccess) {
    WebUI.comment('Registrasi gagal atau notifikasi tidak ditemukan.')
}

// Close browser
WebUI.closeBrowser()
```

Gambar 3.5 Contoh Script AI

Berdasarkan Gambar 3.5, *script automation testing* berbasis AI dihasilkan melalui pemanfaatan *prompt* yang berisi deskripsi skenario pengujian. *Script* yang dihasilkan memiliki struktur yang lebih generik dan dilengkapi dengan penanganan kondisi tertentu.

3.3.2.c Normalisasi dan Pengendalian Variabel Pembuatan Script

Penelitian ini menerapkan normalisasi dan pengendalian variabel pada proses pembuatan *script automation testing* untuk menjaga validitas internal penelitian dan meminimalkan bias yang dapat memengaruhi hasil perbandingan. Normalisasi dilakukan pada tiga aspek utama, yaitu objek pengujian (*website*), pembuatan *script* manual, dan pembuatan *script* berbasis AI.

Objek pengujian pada penelitian ini terdiri atas tiga aplikasi berbasis web, yaitu Sistem Informasi Geografis Universitas Lampung, GitHub, dan OpenStreetMap. *Website* Sistem Informasi Geografis Universitas Lampung merupakan aplikasi berbasis Web GIS yang menyediakan fitur pencarian fasilitas, filter data, interaksi *marker* peta, serta formulir kontak berbasis web. Aplikasi Web GIS dalam konteks rekayasa perangkat lunak seperti ini termasuk ke dalam aplikasi web dengan kompleksitas menengah (*medium complexity*) karena menggabungkan penyajian peta digital yang interaktif dan integrasi antarmuka yang responsif, yang jauh lebih kompleks dibandingkan web statis sederhana yang hanya menampilkan konten tidak dinamis[42]. GitHub digunakan untuk merepresentasikan aplikasi web dengan fitur repository management dan pengelolaan file berbasis cloud. OpenStreetMap digunakan untuk merepresentasikan aplikasi web berbasis peta interaktif dan navigasi rute perjalanan.

Penggunaan beberapa *website* dilakukan untuk meningkatkan variasi skenario pengujian, namun lingkungan pengujian tetap dikendalikan dengan menggunakan browser, *tool automation testing*, serta pendekatan pengujian yang sama pada seluruh objek penelitian, sehingga variasi hasil yang diperoleh tetap merefleksikan perbedaan metode pembuatan *script automation testing*.

Seluruh *script automation testing* pada metode pembuatan *script* manual dibuat oleh peneliti dengan tingkat kemampuan *intermediate* dalam bidang pengujian perangkat lunak. Tingkat kemampuan *programmer* dalam penelitian ini mengacu pada klasifikasi umum dalam rekayasa perangkat lunak yang membedakan antara *entry-level (novice), intermediate*, dan *expert*, yang masing-masing memiliki karakteristik berbeda dalam memahami dan menyelesaikan permasalahan pemrograman.

Tingkat *entry-level (novice)* didefinisikan sebagai individu yang telah memahami dasar-dasar *automation testing*, seperti penggunaan *keyword* dasar pada Katalon Studio, pembuatan *test case* sederhana, serta pemahaman konsep *black-box testing*, namun masih memiliki keterbatasan dalam menangani kasus kompleks dan *debugging* lanjutan. Secara kognitif, *programmer* pada level ini cenderung memahami kode secara lokal dan berfokus pada detail sintaks atau baris per baris, serta belum mampu melihat struktur program secara menyeluruh atau menggunakan pola penyelesaian masalah secara efektif, sebagaimana dijelaskan dalam penelitian mengenai perbedaan pemahaman antara *novice* dan *expert programmer*.

Tingkat *intermediate* didefinisikan sebagai individu yang telah mampu mengembangkan *script* dengan struktur yang lebih kompleks, memahami penggunaan *locator* secara lebih fleksibel, serta mampu melakukan *debugging* terhadap *error* yang muncul selama eksekusi. Hal ini sejalan dengan temuan studi *longitudinal* yang menunjukkan bahwa peningkatan keahlian ditandai dengan kemampuan mengenali pola kode dan peningkatan efisiensi dalam memahami program. Kemampuan pemrograman juga mencakup aspek pemahaman konsep, efisiensi solusi, serta kemampuan *debugging* yang meningkat seiring pengalaman. Level *intermediate* dalam konteks eksperimen perangkat lunak juga merepresentasikan kategori *programmer* yang berada di antara junior dan senior, dengan kemampuan yang cukup untuk menangani tugas dengan kompleksitas menengah namun belum mencapai tingkat optimal dalam efisiensi dan abstraksi seperti pada *expert*.

Tingkat *expert* merepresentasikan individu yang memiliki kemampuan tinggi dalam memahami sistem secara konseptual serta mampu menyelesaikan permasalahan kompleks secara efisien dengan memanfaatkan pengalaman dan *pattern recognition*. *Programmer expert* tidak hanya memahami detail kode, tetapi juga mampu melihat keseluruhan arsitektur sistem dan mengoptimalkan solusi yang dihasilkan.

Berdasarkan klasifikasi tersebut, peneliti dikategorikan berada pada tingkat *intermediate* karena telah memiliki kemampuan dalam menyusun *script automation testing* dengan struktur yang lebih kompleks, memahami penggunaan *locator* secara fleksibel, serta mampu melakukan *debugging* terhadap *error* yang muncul selama proses eksekusi. Peneliti belum memiliki pengalaman industri yang luas maupun kemampuan optimasi dan abstraksi tingkat lanjut yang umumnya dimiliki oleh *programmer expert*, sehingga posisi ini secara konseptual berada pada level menengah sesuai dengan karakteristik yang dijelaskan dalam literatur.

Pembatasan tingkat kemampuan ini bertujuan untuk merepresentasikan kondisi realistis pengguna awal hingga menengah dalam penggunaan automation testing, sehingga hasil penelitian dapat mencerminkan efektivitas metode dalam konteks penggunaan praktis. Peneliti memiliki pemahaman dasar mengenai konsep pengujian perangkat lunak, pengujian *black-box*, serta penggunaan Katalon Studio sebagai alat *automation testing*, namun tidak dimaksudkan untuk merepresentasikan keterampilan *QA engineer* profesional dengan pengalaman industri tingkat lanjut. Peran peneliti dalam penelitian ini adalah berperan sebagai *executor of testing activities*, yaitu pihak yang bertanggung jawab secara langsung dalam menyusun dan mengeksekusi *script* pengujian manual sesuai dengan skenario uji yang telah ditetapka. Peran tersebut diterapkan bukan untuk menggambarkan kemampuan umum penguji perangkat lunak, melainkan untuk menjalankan eksperimen secara terkontrol dan konsisten.

Penggunaan satu pelaksana eksperimen merupakan pendekatan yang lazim dalam penelitian eksperimental di bidang *software engineering*, khususnya pada studi

yang membandingkan metode atau teknik pembuatan *test script*, dengan tujuan mengendalikan variabel *human factor* seperti perbedaan tingkat keterampilan dan pengalaman antar individu. Pembatasan pembuatan *script* manual pada satu *executor*, membuat variasi hasil yang disebabkan oleh perbedaan kompetensi personal dapat diminimalkan, sehingga perbedaan yang diamati lebih merefleksikan karakteristik metode pembuatan *script* itu sendiri. Penggunaan satu pelaksana eksperimen juga membatasi variasi sudut pandang dalam proses pembuatan script, sehingga hasil penelitian ini tidak dimaksudkan untuk merepresentasikan seluruh variasi kemampuan penguji.

Seluruh *script* manual dibuat berdasarkan *test case* yang sama, menggunakan *tool* dan lingkungan pengujian yang identik, sehingga fokus perbandingan diarahkan secara langsung pada metode pembuatan *script.* Studi empiris menunjukkan bahwa dalam konteks industri, organisasi pengembangan perangkat lunak menuntut *tester* dengan tingkat keterampilan teknis dan pemahaman domain yang tinggi untuk menangani sistem yang kompleks [44]. Penelitian ini secara sengaja membatasi konteks pembuatan *script* manual pada tingkat kemampuan *intermediate* sebagai bagian dari desain eksperimen yang terkontrol.

Peneliti pada metode pembuatan *script* berbasis AI menggunakan *prompt* yang disusun oleh peneliti secara manual dan terstruktur. *Prompt* tersebut mencakup konteks aplikasi, deskripsi fitur yang diuji, langkah-langkah pengujian, serta ketentuan teknis penggunaan Katalon Studio. *Prompt* yang digunakan bersifat tetap (*fixed prompt*) dan diterapkan secara konsisten pada seluruh *test case*, tanpa dilakukan perubahan selama proses penelitian. *Prompt* diperlakukan sebagai variabel terkontrol, bukan variabel bebas, sehingga variasi hasil *script* yang dihasilkan AI tidak dipengaruhi oleh perbedaan strategi *prompting*[41].

Terdapat beberapa faktor yang berpotensi memengaruhi hasil penelitian, selain variabel yang dikendalikan. Faktor tersebut meliputi kompleksitas fitur sistem yang diuji, stabilitas elemen antarmuka (UI) seperti perubahan DOM dan waktu loading halaman, kualitas serta kejelasan prompt pada metode berbasis AI, serta pengalaman dan pemahaman peneliti dalam menggunakan tools automation testing.

Penggunaan beberapa website selain meningkatkan variasi fitur pengujian juga bertujuan untuk mengurangi bias penelitian yang mungkin muncul apabila pengujian hanya dilakukan pada satu jenis aplikasi web. Hasil penelitian diharapkan dapat memberikan gambaran yang lebih representatif mengenai karakteristik pembuatan *script automation testing* pada berbagai konteks aplikasi berbasis web.

Penelitian perbandingan metode pembuatan *script automation testing* secara konseptual idealnya melibatkan lebih dari satu pelaksana pengujian manual dengan tingkat kemampuan yang bervariasi, serta diterapkan pada beberapa aplikasi web dengan karakteristik yang berbeda, guna meningkatkan validitas eksternal dan potensi generalisasi hasil. Penelitian ini dilaksanakan dalam konteks skripsi dengan keterbatasan waktu dan sumber daya, sehingga penerapan desain ideal tersebut tidak memungkinkan untuk dilakukan secara optimal. Oleh karena itu, dilakukan penyederhanaan desain eksperimen dengan tetap mempertahankan prinsip penelitian ilmiah yang terkontrol, yaitu dengan membatasi pembuatan *script* manual pada satu pelaksana eksperimen dan menggunakan tiga aplikasi web dengan ruang lingkup fitur yang telah ditentukan sebagai objek pengujian. Penelitian melalui pendekatan ini diharapkan tetap mampu menghasilkan temuan yang valid secara internal dan sesuai dengan tujuan penelitian, meskipun berada dalam batasan ruang lingkup yang terdefinisi.

Penerapan normalisasi dan pengendalian variabel pada penelitian ini bertujuan untuk memastikan bahwa perbandingan efektivitas antara metode pembuatan *script* manual dan metode berbantuan AI dilakukan secara adil, terkontrol, dan dapat dipertanggungjawabkan secara metodologis. Penelitian ini tidak dimaksudkan untuk menggeneralisasi hasil ke seluruh jenis aplikasi web, seluruh tingkat keahlian pembuat *script*, maupun seluruh variasi *prompt* AI, melainkan terbatas pada konteks penelitian yang menggunakan *website* Sistem Informasi Geografis Universitas Lampung, GitHub, dan OpenStreetMap sebagai objek pengujian dengan ruang lingkup fitur yang telah ditetapkan.

3.3.3 Pengujian

Tahapan pengujian bertujuan untuk mengeksekusi *script* yang telah dibuat pada aplikasi berbasis web target menggunakan Katalon Studio, dengan fokus pada validasi fungsionalitas dan pengukuran keberhasilan. Eksekusi dilakukan secara berurutan, yaitu *script* manual dijalankan terlebih dahulu, diikuti *script* berbantuan AI, menggunakan perangkat uji yang sama (*browser*) dan lingkungan Katalon Studio yang terkonfigurasi secara identik untuk memastikan konsistensi.

Proses eksekusi melibatkan *setup* awal, seperti konfigurasi *browser* di Katalon Studio, pengaturan parameter pengujian, dan pemantauan *real-time* melalui *interface* Katalon. Sistem selama eksekusi menghasilkan laporan otomatis yang mencatat status keberhasilan (*passed/failed*), waktu eksekusi, pesan *error*, dan *screenshot* jika terjadi kegagalan. Jika ditemukan *error*, dilakukan *debugging* singkat untuk memastikan *script* dapat direproduksi.

Pengujian dilakukan untuk setiap *test case* pada fitur yang diuji, sehingga satu fitur yang memiliki beberapa skenario (misalnya berhasil dan gagal) akan menghasilkan beberapa *script* dan beberapa hasil eksekusi. Pengulangan eksekusi dilakukan untuk memastikan reliabilitas hasil. Penelitian [3] dalam *Journal of Software Engineering and Applications* menegaskan bahwa fitur eksekusi otomatis Katalon meningkatkan efisiensi deteksi kesalahan hingga 42% dan menyediakan data eksekusi yang akurat untuk penelitian berbasis automation testing.

3.3.4 Bug Report

Kesalahan yang ditemukan dalam proses pengujian didokumentasikan dalam bentuk *bug report* yang disusun secara sistematis dan rinci. Laporan ini memuat beberapa komponen utama, seperti *test case ID* yang digunakan sebagai acuan terhadap skenario pengujian tertentu, deskripsi *bug* untuk menjelaskan permasalahan yang terjadi, serta waktu pelaksanaan pengujian sebagai informasi kronologis.

Laporan juga mencantumkan informasi terkait lingkungan pengujian, meliputi platform, sistem operasi, dan *browser* yang digunakan. Informasi ini diperlukan agar pengembang dapat merekonstruksi kondisi saat bug ditemukan secara akurat. Setiap *bug* juga diklasifikasikan berdasarkan tingkat *severity* dan *priority* untuk menunjukkan tingkat keparahan serta urgensi penanganannya.

Langkah-langkah untuk mereproduksi *bug* turut disertakan agar pengembang dapat memahami proses terjadinya kesalahan dan melakukan perbaikan secara tepat. Dokumentasi *bug report* tidak hanya berfungsi sebagai panduan dalam proses perbaikan, tetapi juga sebagai arsip historis yang dapat digunakan sebagai bahan evaluasi dalam pengembangan sistem selanjutnya. Adanya dokumentasi yang terstruktur, membuat tim pengembang dapat mengantisipasi dan meminimalkan kemungkinan terjadinya kesalahan serupa di masa mendatang.

Klasifikasi dari tingkat *severity* atau keparahan tercantum dalam tabel berikut.

Tabel 3.6 Klasifikasi *Bug Severity*

Severity	Deskripsi	Dampak pada Sistem
Low	Kesalahan berskala kecil yang tidak memberikan pengaruh berarti terhadap sistem	Tidak memengaruhi fungsi maupun performa sistem secara keseluruhan
Minor	Kesalahan ringan yang dapat menimbulkan gangguan, tetapi tidak berdampak pada fungsi utama	Sistem tetap berjalan normal tanpa gangguan pada fungsi inti
Major	Kesalahan yang memengaruhi fungsi penting dalam sistem	Sebagian fungsi utama terganggu, namun sistem masih dapat dioperasikan
Critical	Kesalahan yang menyebabkan kegagalan sistem secara menyeluruh	Sistem tidak dapat digunakan dan fungsi utama tidak berjalan

Berdasarkan tabel 3.6, *bug severity* digunakan untuk mengidentifikasi tingkat keparahan suatu bug berdasarkan dampaknya terhadap kinerja dan fungsionalitas sistem. Tingkat *severity* menggambarkan seberapa besar pengaruh bug terhadap operasional sistem, mulai dari kesalahan yang tidak berdampak signifikan hingga kesalahan yang menyebabkan sistem tidak dapat digunakan.

Klasifikasi *bug priority* atau tingkat prioritas penanganan tecantum dalam tabel berikut.

Tabel 3.7 Klasifikasi *Bug Priority*

Priority	Deskripsi	Tindakan yang Diperlukan
Blocker	*Bug* yang menghambat seluruh proses sistem	Harus segera diperbaiki sebelum sistem dapat digunakan atau dirilis
Critical	*Bug* yang sangat penting dan berdampak pada fungsi utama sistem	Perlu segera diperbaiki dalam waktu dekat
Major	*Bug* yang cukup penting namun tidak bersifat mendesak	Dapat diperbaiki pada tahap pengembangan berikutnya
Minor	*Bug* dengan tingkat kepentingan rendah	Penanganan dapat dilakukan setelah *bug* dengan prioritas lebih tinggi
Trivial	*Bug* yang sangat kecil dan tidak berdampak signifikan	Perbaikan dapat ditunda atau dilakukan jika terdapat waktu luang

Berdasarkan tabel 3.7, *bug priority* digunakan untuk menentukan tingkat prioritas penanganan *bug* berdasarkan urgensi perbaikannya. Berbeda dengan *severity* yang berfokus pada dampak, *priority* lebih menitikberatkan pada waktu dan urutan penanganan *bug* dalam proses pengembangan. Penerapan klasifikasi *priority* ini bertujuan untuk membantu pengembang dalam mengalokasikan sumber daya

secara efektif sehingga perbaikan *bug* dapat dilakukan secara optimal sesuai dengan tingkat kepentingannya.

3.3.5 Pengumpulan Hasil

Pengumpulan hasil dalam penelitian ini dilakukan melalui pendekatan eksperimental, yaitu dengan mengamati secara langsung proses pembuatan dan penyempurnaan *script* pada Katalon Studio menggunakan dua metode berbeda, yaitu metode manual dan metode berbasis AI. Data yang dikumpulkan bersifat kuantitatif dan berfungsi untuk menilai efektivitas masing-masing metode berdasarkan tiga metrik, yaitu waktu pembuatan *script*, jumlah iterasi perbaikan, dan kompleksitas *script*.

1. Waktu Pembuatan *Script*

   Waktu pembuatan dicatat sejak peneliti mulai menulis *script* hingga versi awal *script* (*initial script*) selesai dibuat dan siap untuk diuji. Pengukuran waktu dilakukan menggunakan *stopwatch digital* untuk menjaga ketepatan durasi yang direkam.

2. Jumlah Iterasi Perbaikan

   Iterasi perbaikan didefinisikan sebagai jumlah siklus *debugging* yang diperlukan hingga *script* dapat berjalan tanpa *error*. Satu iterasi mencakup proses identifikasi kesalahan, revisi *script*, dan pengujian ulang. Data iterasi diperoleh melalui *log* hasil eksekusi Katalon Studio, dan pencatatan manual selama proses *debugging*.

3. Kompleksitas *Script*

   Kompleksitas diukur dari jumlah baris kode (Lines of Code/LOC) dan nilai *Cyclomatic Complexity (CC)* pada *script* final yang berhasil dijalankan tanpa *error*. LOC merepresentasikan ukuran kode secara kuantitatif, sedangkan CC digunakan untuk menggambarkan kompleksitas struktur logika dan jumlah jalur eksekusi independen dalam *script automation testing*. Perhitungan LOC dilakukan menggunakan fitur penghitung baris pada Katalon Studio atau secara manual. Kompleksitas dalam penelitian ini

tidak hanya dinilai berdasarkan banyaknya baris kode, tetapi juga mempertimbangkan kualitas struktur dan kegunaan bagian-bagian *script* tersebut. Oleh karena itu, *script* yang memiliki LOC lebih tinggi tetap dapat dianggap efektif apabila baris tambahan tersebut berfungsi untuk meningkatkan kejelasan alur, menyediakan mekanisme penanganan *error* (*error handling*), *logging*, atau langkah validasi yang berdampak positif pada kestabilan pengujian.

Seluruh data direkap dalam tabel untuk memastikan keteraturan dan konsistensi. Pendekatan pengumpulan hasil ini sesuai dengan rekomendasi penelitian eksperimental pada automation testing sebagaimana dijelaskan oleh [18], yang menyatakan bahwa data berbasis hasil eksekusi alat otomatis memberikan reliabilitas tinggi dalam evaluasi performa *script*.

3.3.6 Analisis Hasil

Tahap analisis merupakan proses untuk mengidentifikasi data kuantitatif yang telah dikumpulkan, dengan tujuan melihat efektivitas metode pembuatan *script* manual dan berbasis AI. Analisis pada metrik waktu pembuatan *script*, berfokus pada perbedaan durasi antara metode manual dan metode berbasis AI dalam menghasilkan *initial script*. Nilai waktu yang lebih singkat diinterpretasikan sebagai efektivitas yang lebih tinggi dalam konteks efisiensi proses pembuatan *script.*

Analisis pada metrik jumlah iterasi perbaikan dilakukan dengan membandingkan jumlah siklus *debugging* yang diperlukan oleh masing-masing metode hingga *script* dapat dijalankan tanpa *error*. Metode yang membutuhkan iterasi lebih sedikit dianggap memiliki kualitas *initial script* yang lebih baik, sehingga lebih efektif dalam menghasilkan *script* yang stabil. Analisis pada metrik kompleksitas *script* tidak hanya meninjau jumlah baris kode (LOC) pada *final script*, tetapi juga mempertimbangkan nilai fungsional dari struktur *script* tersebut. Walaupun LOC yang lebih kecil menunjukkan *script* yang lebih ringkas dan mudah dipelihara, *script* dengan LOC lebih besar dapat dinilai efektif apabila baris tambahan tersebut

memberikan manfaat, seperti *error handling* yang lebih jelas, proses validasi yang lebih aman, atau *logging* yang memudahkan pelacakan *error*. Selain jumlah baris kode, nilai *Cyclomatic Complexity (CC)* digunakan untuk menilai tingkat kompleksitas logika *script*. *Script* dengan nilai CC yang lebih rendah menunjukkan struktur kontrol yang lebih sederhana dan lebih mudah diuji, sedangkan nilai CC yang lebih tinggi mengindikasikan meningkatnya jumlah keputusan dan jalur eksekusi dalam *script*.

Seluruh hasil analisis disajikan dalam bentuk tabel untuk memudahkan visualisasi perbedaan antar-metode. Analisis dilakukan secara terpisah pada tiap metrik agar dapat diketahui pada aspek mana metode manual lebih unggul, dan pada aspek mana metode berbantuan AI menunjukkan efektivitas yang lebih baik. Pendekatan ini memastikan hasil penelitian bersifat objektif, terukur, dan relevan dalam konteks *automation testing*.

3.3.7 Rekomendasi Berdasarkan Hasil Analisis

Rekomendasi disusun pada tahap akhir penelitian sebagai keluaran penelitian yang bertujuan memberikan acuan dalam pemilihan metode pembuatan *script automation testing* sesuai dengan kebutuhan dan karakteristik pengujian. Rekomendasi ini disusun berdasarkan kerangka analisis yang telah ditetapkan, tanpa menetapkan keunggulan suatu metode secara langsung. Penyusunan rekomendasi didasarkan pada hasil pengukuran metrik efektivitas yang digunakan, yaitu waktu pembuatan *script*, jumlah iterasi perbaikan hingga *script* mencapai kondisi stabil, serta kompleksitas *script*. Metrik-metrik tersebut digunakan untuk menggambarkan karakteristik proses pembuatan *script automation testing* dengan metode manual maupun metode berbantuan *Artificial Intelligence* (AI).

Selain metrik efektivitas, rekomendasi juga disusun dengan mempertimbangkan karakteristik skenario pengujian, seperti tingkat kompleksitas alur sistem, kebutuhan kontrol logika pengujian, serta sifat pengujian yang bersifat umum, berulang, atau memerlukan penyesuaian teknis tertentu. Pertimbangan ini bertujuan agar rekomendasi yang dihasilkan bersifat kontekstual dan dapat digunakan sebagai

pedoman dalam menentukan pendekatan pembuatan *script* yang sesuai. Oleh karena itu, rekomendasi yang disusun pada tahap ini berfungsi sebagai panduan metodologis dalam pemilihan metode pembuatan *script automation testing* berdasarkan hasil analisis yang dilakukan, tanpa mengurangi peran *Quality Assurance* (QA) dalam melakukan evaluasi dan penyesuaian sesuai kebutuhan pengujian.

3.4 Teknik Pengukuran Efektivitas

Teknik pengukuran efektivitas digunakan untuk menentukan sejauh mana metode *manual* dan metode berbantuan *AI* memberikan hasil yang lebih optimal berdasarkan tiga metrik, yaitu waktu pembuatan *script*, jumlah iterasi perbaikan, dan kompleksitas *script*. Pembuatan dan eksekusi *script* dalam penelitian ini dilakukan pada level *test case,* bukan pada level fitur. Pendekatan ini dipilih karena satu fitur aplikasi umumnya terdiri dari beberapa skenario pengujian. Sebagai ilustrasi, fitur *register* dapat mencakup *test case* "berhasil *register*" dan "gagal *register*", sehingga menghasilkan lebih dari satu *script* yang harus diuji.

Setiap *test case* menghasilkan dua *script*, satu disusun menggunakan metode manual, dan satu disusun dengan bantuan AI. Seluruh *script* tersebut dikumpulkan datanya berdasarkan metrik-metrik penelitian yang telah ditetapkan. Pengukuran efektivitas dilakukan secara agregat, yaitu seluruh data dari setiap *test case* dihitung dan dianalisis secara keseluruhan untuk masing-masing metode sehingga hasil evaluasi lebih merefleksikan performa umum metode tersebut.

1. Efektivitas Waktu Pembuatan *Script*

   Durasi pembuatan *script* dicatat pada setiap *test case*. Selanjutnya, seluruh durasi untuk masing-masing metode manual dan AI diakumulasi dan dihitung nilai rata-ratanya. Metode yang memiliki rata-rata waktu pembuatan lebih rendah dianggap lebih efektif dalam hal efisiensi proses pengembangan *script*.

2. Efektivitas Iterasi Perbaikan

   Setiap *test case* dapat menghasilkan jumlah iterasi revisi yang berbeda, bergantung pada stabilitas *initial script*. Metode yang menunjukkan iterasi perbaikan paling sedikit dinilai lebih efektif karena lebih mampu menghasilkan *script* yang stabil dan meminimalkan kebutuhan perbaikan.

3. Efektivitas Kompleksitas *Script*

   Kompleksitas *script* diukur menggunakan dua metrik kuantitatif, yaitu *Lines of Code* (LOC) dan *Cyclomatic Complexity* (CC), yang dihitung pada *script* final dari setiap *test case* setelah berhasil dieksekusi tanpa *error*. Pengukuran diawali dengan perhitungan nilai LOC untuk merepresentasikan ukuran dan tingkat keringkasan *script*, kemudian dilanjutkan dengan pengukuran nilai CC untuk mengevaluasi kompleksitas logika berdasarkan jumlah percabangan dan jalur eksekusi yang terbentuk. Nilai LOC dan CC dari masing-masing *script* selanjutnya dirata-ratakan untuk setiap metode (manual dan berbasis AI) sebagai dasar perbandingan efektivitas. LOC yang lebih rendah menunjukkan *script* yang lebih ringkas, sedangkan nilai CC yang lebih rendah menunjukkan struktur logika yang lebih sederhana dan lebih mudah dipelihara. Selain pengukuran numerik tersebut, dilakukan pencatatan deskriptif terhadap baris kode tambahan yang memiliki fungsi teknis yang jelas, seperti *error handling*, validasi kondisi, dan *logging*, sebagai penjelasan pendukung tanpa memengaruhi nilai LOC dan CC. Dengan demikian, penilaian efektivitas kompleksitas tetap berlandaskan pada metrik objektif LOC dan CC, sementara interpretasi fungsional digunakan untuk memberikan konteks terhadap struktur *script* yang dihasilkan.

3.4.1 Metrik Pengukuran

Tabel berikut menyajikan metrik pengukuran yang digunakan dalam penelitian.

Tabel 3.8 Metrik Pengukuran Efektivitas *Script*

Metrik	Data yang Diambil	Cara Pengukuran	Output
Waktu pembuatan *script*	Waktu mulai dan selesai pembuatan *initial script*	*Stopwatch* atau pencatatan manual	Menit
Jumlah error/iterasi perbaikan *script*	Jumlah iterasi debugging hingga *script* berhasil dieksekusi tanpa *error*	Log eksekusi Katalon Studio & pencatatan iterasi	Jumlah iterasi
Kompleksitas *script*	Jumlah baris kode (LOC) dan nilai *Cyclomatic Complexity* (CC) dari *script* final dan nilai fungsional baris tambahan	Penghitungan melalui Katalon Studio atau manual	Jumlah baris kode dan catatan kegunaan, serta jumlah keputusan/percabangan

Berdasarkan Tabel 3.8, metrik pengukuran yang digunakan dalam penelitian ini bertujuan untuk mengevaluasi efektivitas pembuatan *script automation testing*. Metrik tersebut mencakup waktu pembuatan *script*, jumlah iterasi perbaikan, serta kompleksitas *script.*

V. KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian dan pengujian, dapat disimpulkan sebagai berikut.

1. Berdasarkan hasil penelitian, efektivitas pembuatan *script automation testing* pada Katalon Studio menunjukkan bahwa metode berbasis AI lebih unggul pada metrik waktu pembuatan *script* dan jumlah iterasi perbaikan, dengan rata-rata waktu 3 menit per *test case* dan 5 total iterasi perbaikan, dibandingkan metode manual yang membutuhkan rata-rata 7 menit per *test case* dan 9 iterasi perbaikan. Pada sisi kompleksitas *script* yang diukur melalui *Lines of Code* (LOC) *dan Cyclomatic Complexity* (CC), metode manual menghasilkan *script* yang lebih sederhana.
2. Penggunaan AI dalam proses pembuatan *script automation testing* pada Katalon Studio berperan sebagai akselerator yang meningkatkan efisiensi waktu dan stabilitas *script* awal, namun tidak sepenuhnya menggantikan peran manusia. Keterlibatan manusia tetap diperlukan dalam penyusunan *test case*, penentuan *object repository*, identifikasi *locator* elemen, dan perumusan *prompt* yang tepat. Metode berbasis AI lebih direkomendasikan ketika efisiensi waktu menjadi prioritas, sedangkan metode manual lebih sesuai apabila kesederhanaan dan kemudahan pemeliharaan *script* yang diutamakan.

5.2 Saran

Berdasarkan hasil penelitian yang telah dilakukan, terdapat beberapa saran yang dapat diberikan untuk pengembangan penelitian maupun implementasi di masa mendatang:

1. Melibatkan lebih dari satu penguji dengan tingkat kemampuan yang bervariasi dalam proses pembuatan *script automation testing* secara manual, sehingga hasil yang diperoleh lebih objektif dan tingkat generalisasi temuan penelitian dapat ditingkatkan.
2. Menggunakan variasi *tools automation testing* lain selain Katalon Studio, seperti Selenium atau Playwright, untuk membandingkan apakah pola efektivitas yang ditemukan dalam penelitian ini konsisten pada *tools* yang berbeda.

DAFTAR PUSTAKA

[1] K. S. Thant and H. H. K. Tin, “The Impact of Manual and Automatic Testing on Software Testing Efficiency and Effectiveness,” *Indian Journal of Science and Research*, vol. 3, no. 3, May 2023.

[2] S. Adiatma, A. Darmayantie, and Gunadarma University, “Implementation and Comparative Analysis of Test Automation Framework Performance for Functional Testing of Web-Based Applications using the Distance to the Ideal Alternative (DIA) Method,” *Scientific Journal Widya Teknik*, vol. 22, no. 1, pp. 36–41, May 2023.

[3] S. E. Anggrainy and A. H. Muhammad, “Efficiency Development Analysis Use of Automation Testing using Katalon on Mobile-Based Applications (Case Insurance Company),” *International Journal of Information System & Technology*, vol. 8, no. 1, pp. 54–59, 2024.

[4] S. S. Rahayu, D. A. Firmansyah, S. Susanti, and U. A. R. Sanjaya, “Analisis Penggunaan Tools Automation Testing pada Aplikasi: Systematic Literature Review,” *Riset dan E-Jurnal Manajemen Informatika Komputer*, vol. 8, no.1. 2024.

[5] M. Schäfer, S. Nadi, A. Eghbali, and F. Tip, “An empirical evaluation of using large language models for automated unit test generation,” *arXiv:2302.06527*, Dec. 2023.

[6] E. Júnior, A. Valejo, J. Valverde-Rebaza, and V. de O. Neves, “GenIA-E2ETest: A Generative AI-Based Approach for End-to-End Test Automation,” *Anais do XXXIX Simpósio Brasileiro de Engenharia de Software (SBES 2025)*, pp. 282–292, Sept 2025.

[7] A. Celik and Q. H. Mahmoud, “A Review of Large Language Models for Automated Test Case Generation,” *Mach. Learn. Knowl. Extr.*, vol. 7, no. 3, p. 97, Sept 2025.

[8] A. Mishra, “Generative AI in Automated Software Testing: A Comparative Study,” *International Journal of Science and Technology*, vol. 2, no. 1, 2024.

[9] M. Adnan and C. C. N. Kuhn, “Measuring and mitigating debugging effectiveness decay in code language models,” *Scientific Reports*, vol. 15, art. no. 44120, 2025.

[10] S. Omri, P. Montag, and C. Sinz, “Static Analysis and Code Complexity Metrics as Early Indicators of Software Defects,” *J. Softw. Eng. Appl.*, vol. 11, no. 04, pp. 153–166, 2022.

[11] D. I. De Silva, B. L. O. Sachethana, S. M. D. T. H. Dias, P. Y. C. Perera, M. E. Katipearachchi, and T. D. D. H. Jayasuriya, “The relationship between code complexity and software quality: An empirical study,” *in Proc. Empirical Software Engineering*, Sri Lanka Institute of Information Technology, Sri Lanka, 2022.

[12] J. Dean, *Web programming with HTML5, CSS, and JavaScript*. Burlington, MA: Jones & Bartlett Learning, 2021.

[13] E. A. Altulaihan, A. Alismail, and M. Frikha, “A Survey on Web Application Penetration Testing,” *Electronics*, vol. 12, no. 5, p. 1229, Mar. 2023.

[14] S. Goericke, Ed., *The Future of Software Quality Assurance*. Cham, Switzerland: Springer Nature Switzerland AG, 2021.

[15] J. Tian, *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. USA: Wiley & IEEE Computer Society Press, 2021.

[16] P. Ammann and J. Offutt, *Introduction to Software Testing*, 1st ed. USA: Cambridge University Press, 2022.

[17] E. Dustin, J. Rashka, and J. Paul, *Automated Software Testing (Introduction, management, and Performance)*, 9th ed. Canada: Addison Wesley, 2021.

[18] N. G. Berihun, C. Dongmo, and J. A. Van Der Poll, “The Applicability of Automated Testing Frameworks for Mobile Application Testing: A Systematic Literature Review,” *Computers*, vol. 12, no. 5, p. 97, May 2023.

[19] Katalon, “Katalon Studio Official Web Page.” [Online]. Available: www.katalon.com. [Accessed: 17-Nov-2025].

[20] Y. Kosasih dan A. B. Cahyono, “Automation Testing Tool dalam Pengujian Aplikasi The Point of Sale (Studi Kasus TPOS PT. Javasigna Intermedia),” *Jurnal Teknologi dan Sistem Komputer*, vol. 9, no. 1, pp. 18, 2021.

[21] S. Mohaghegh, *Artificial Intelligence for Science and Engineering Application*, 1st ed. Boca Raton FL: CRC Press, 2024.

[22] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*, 4th Edition. in Pearson Series in Artificial Intelligence. Hoboken, NJ: Pearson, 2021.

[23] K. M. Y. Ali and S. Akter, “AI-Driven Test Case Optimization: Enhancing Efficiency in Software Testing Life Cycle,” *Int. J. Comput. Sci. Inf. Technol.*, vol. 16, no. 6, pp. 65–79, Dec. 2024.

[24] M. Andrzejewski, N. Dubicka, J. Podolak, M. Kowal, and J. Siłka, “Automated Test Generation Using Large Language Models,” *Data*, vol. 10, no. 10, p. 156, Sept. 2025.

[25] D. Pallai, *Large Language Models for Developers (A Prompt-based Exploration)*. Boston: Mercury Learning and Information, 2023.

[26] L. Banh and G. Strobel, “Generative artificial intelligence,” *Electron. Mark.*, vol. 33, no. 1, p. 63, Dec. 2023.

[27] D. Tosi, “Studying the Quality of Source Code Generated by Different AI Generative Engines: An Empirical Evaluation,” *Future Internet*, vol. 16, no. 6, p. 188, May 2024.

[28] V. A. K. Siren, N. Y. Setiawan, dan R. I. Rokhmawati, “Evaluasi Kualitas Perangkat Lunak Menggunakan ISO/IEC 9126-4 Quality In Use (Studi Kasus: FILKOM Apps),” *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, vol. 3, no. 2, pp. 1625–1632, Feb. 2021.

[29] A. A. P. Ramadhani, R. A. Nugroho, M. R. Faisal, F. Abadi, and R. Herteno, “The impact of software metrics in NASA metric data program dataset modules for software defect prediction,” *TELKOMNIKA Telecommun. Comput. Electron. Control*, vol. 22, no. 4, p. 846, Aug. 2024.

[30] A. H. Odeh, M. Odeh, H. Odeh, and N. Odeh, “Measuring cyclomatic complexity of source code using machine learning,” *Revue d’Intelligence Artificielle*, vol. 38, no. 1, pp. 183–191, Feb. 2024.

[31] M. Leotta, H. Z. Yousaf, F. Ricca, and B. Garcia, “AI-Generated Test Scripts for Web E2E Testing with ChatGPT and Copilot: A Preliminary Study,” in *Proc. of the 28th International Conference on Evaluation and Assessment in Software Engineering*, Salerno Italy: ACM, June 2024.

[32] V. Garousi *et al.*, “AI-powered software testing tools: A systematic review and empirical assessment of their features and limitations” *Software Testing Journal*, 2024.

[33] I. Kertusha, G. Assress, O. Duman, and A. Arcuri, “A Survey on Web Testing: On the Rise of AI and Applications in Industry,” Aug. 12, 2025, arXiv:2503.05378. doi: 10.48550/arXiv.2503.05378.

[34] S. Bhatia, T. Gandhi, D. Kumar, and P. Jalote, “Unit Test Generation using Generative AI: A Comparative Performance Analysis of Autogeneration Tools,” in *Proc. ICSE 2024*, Lisbon, Portugal, Apr 2024.

[35] S. Yu, C. Fang, Y. Ling, C. Wu, and Z. Chen, “LLM for Test Script Generation and Migration: Challenges, Capabilities, and Opportunities,” *arXiv:2309.13574*, Sep 2023.

[36] S. Lukasczyk, F. Kroiß, and G. Fraser, “An empirical study of automated unit test generation for Python,” *Empir. Softw. Eng.*, vol. 28, no. 2, p. 36, Mar. 2023.

[37] E. Chen *et al.*, “Enhancing AI-generated Code via Automated Evaluation and Validation,” *Technical Disclosure Commons*, Jun. 2025.

[38] S. Guo *et al.*, “Optimizing Case-Based Reasoning System for Functional Test Script Generation with Large Language Models,” in *Proc. of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining vol.2*, pp. 4487–4498, Aug 2025.

[39] A. Sidqi, A. Yuswanto, and M. A. Yaqin, “Analisis Pengukuran Kompleksitas Website SOC Menggunakan Function Point,” *J. Autom. Comput. Inf. Syst.*, vol. 4, no. 2, pp. 67–75, Jul. 2024.

[40] F. F. Septiana and H. N. F. Fikrillah, “Penerapan Metode Indirect Approach Terhadap Software Monitoring Stunting Untuk Menghitung Cost Estimation,” *J. Algoritma*, vol. 22, no. 1, pp. 70–78, May 2025.

[41] S. Schulhoff *et al.*, “The Prompt Report: A Systematic Survey of Prompt Engineering Techniques,”, *arXiv*, Feb. 26, 2025.

[42] W. Yang *et al.*, “An MLLM-Assisted Web Crawler Approach for Web Application Fuzzing,” *Applied Sciences*, vol. 15, no. 2, p. 962, Jan. 2025.

[43] S. Shimizu and Y. Higo, “Coverage Isn’t Enough: SBFL-Driven Insights into Manually Created vs. Automatically Generated Tests,” *arXiv*: arXiv:2512.11223. Dec. 12, 2025.

[44] R. Florea, V. Stray, and D. I. K. Sjøberg, “On the roles of software testers: An exploratory study,” *The Journal of Systems & Software*, vol. 204, p. 111742, Oct. 2023.