

Performance and CI/CD Compatibility Analysis of Selenium, Cypress, and Playwright UI Testing Frameworks Using the Distance to Ideal Alternative Method

M Dirga Purnomo Putra ^{1*}, Rakhmat Prasetyo Agung Nugroho ^{2*}, Ridho Surya Kusuma ^{3*}, Latifah Iriani ^{4*}

* PJJ Informatika, Universitas Siber Muhammadiyah

dirga20220100042@sibermu.ac.id ¹, agungprasetyo@sibermu.ac.id ², ridhosuryakusuma@sibermu.ac.id ³, latifahiriani@sibermu.ac.id ⁴

Article Info

Article history:

Received 2026-07-24

Revised 2026-09-11

Accepted 2026-10-17

Keyword:

CI/CD,
Cypress,
Playwright,
Selenium,
UI Test Automation.

ABSTRACT

Prior comparative studies have evaluated Selenium, Cypress, and Playwright in terms of functional features or local execution speed, whereas their integration efficiency within a Continuous Integration/Continuous Deployment (CI/CD) pipeline, a decisive factor in modern DevOps practice, has received limited empirical attention. Selecting the optimal framework for User Interface (UI) test automation is nevertheless a critical challenge in modern software development, especially in the context of deep integration into CI/CD. This study aims to analyze the comparative performance and CI/CD compatibility of three leading frameworks: Selenium, Cypress, and Playwright. Through a quantitative experimental design, three real-world test scenarios covering user authentication, dynamic UI interactions, and end-to-end transactions were executed on the GitHub Actions platform. Performance evaluation was based on three weighted criteria: Average Execution Time (Runtime) (C_1), Execution Validity (C_2), and CI/CD Compatibility (C_3). The test results were analyzed using the Multi-Criteria Decision Making (MCDM) method, specifically Distance to Ideal Alternative (DIA), to generate an objective preference ranking. The DIA analysis results ranked Playwright ($R_i = 0,000$) as the top-performing framework, followed by Cypress ($R_i = 0,066$), and Selenium ($R_i = 0,406$), with the underlying runtime differences confirmed statistically significant (Kruskal-Wallis test, $p < 0,001$). These findings confirm that the modern architectures of Playwright and Cypress are capable of delivering significantly higher execution speeds and better CI/CD compatibility compared to traditional WebDriver-based architectures. As a practical implication, the adoption of modern frameworks, particularly Playwright, is highly recommended for software developers, software testers, and DevOps teams to optimize the feedback loop and improve the efficiency of the application release process.



This is an open access article under the [CC-BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.

I. PENDAHULUAN

Dalam siklus pengembangan perangkat lunak *Software Development Life Cycle* (SDLC), fase pengujian merupakan komponen krusial yang dapat memakan hingga 40% dari total alokasi waktu proyek [1]. Secara historis, pengujian yang dilakukan secara manual memiliki kerentanan tinggi terhadap kesalahan manusia (*human error*) dan ketidakefisienan waktu. Sebagai respons terhadap tantangan tersebut, industri perangkat lunak telah beralih secara masif ke pengujian

otomatis. Pergeseran menuju pengujian otomatis ini juga tercermin pada praktik pengembangan perangkat lunak berskala besar, khususnya pada proyek *open source*. Studi empiris oleh Di Meglio et al. [2] menunjukkan bahwa pengujian antarmuka grafis atau *Graphical User Interface* (GUI) otomatis telah diadopsi secara luas dan dipelihara secara berkelanjutan sepanjang siklus hidup aplikasi, serta menjadi bagian integral dari alur kerja DevOps modern. Otomasi memungkinkan validasi fungsionalitas, kinerja, dan

keamanan aplikasi dilakukan secara berulang dan konsisten, yang merupakan prasyarat utama dalam penerapan metodologi DevOps modern, khususnya praktik *Continuous Integration* (CI) yakni integrasi perubahan kode secara berkala, dan *Continuous Deployment* (CD) yaitu proses peluncuran rilis ke lingkungan produksi secara otomatis. Integrasi pengujian otomatis dalam ekosistem *CI/CD* terbukti secara empiris mampu mereduksi waktu *deployment* dan tingkat kegagalan rilis secara signifikan, sebagaimana dibuktikan dalam studi infrastruktur berbasis kontainer oleh Hyun et al. [3]

Namun, dalam implementasi nyata di lapangan, tim pengembang sering menghadapi kendala signifikan berupa *flaky tests* (pengujian yang tidak stabil) dan antrian *CI/CD* yang panjang. Ketidakstabilan tersebut seringkali bukan disebabkan oleh *bug* pada aplikasi, melainkan ketidakmampuan alat uji tradisional dalam menangani elemen web yang dinamis secara *real-time*. Seringkali tes gagal karena skrip berjalan lebih cepat daripada proses *rendering* halaman, menyebabkan elemen belum siap saat diperintah. Dampak dari ketidakstabilan ini meluas hingga mempengaruhi psikologi pengembang. Studi empiris berskala besar oleh Di Meglio et al. [2] terhadap 472 proyek *open-source* mengungkapkan bahwa *flaky tests* merupakan tantangan struktural yang persisten dalam pengujian antarmuka grafis berbasis web. Sebagian besar isu yang dihasilkan secara otomatis oleh sistem *CI/CD* diidentifikasi bersumber dari ketidakstabilan eksekusi dan kelambatan *test suite*, sehingga cenderung ditutup tanpa penanganan aktif oleh pengembang dan mengindikasikan erosi kepercayaan terhadap *test suite* dalam ekosistem *CI/CD*. Akibatnya, waktu tim terbuang untuk *debugging* skrip tes yang sebenarnya tidak bermasalah, dan siklus rilis fitur menjadi terhambat. Temuan Dobslaw et al. [4] memperkuat argumentasi tersebut dengan menyatakan bahwa *Return on Investment* (ROI) dari otomatis *GUI* sangat bergantung pada pengurangan biaya pemeliharaan (*maintenance cost*) jangka panjang, di mana ketidakstabilan alat uji dapat membatalkan keuntungan efisiensi yang diharapkan.

Diversitas *framework* pengujian otomatis yang tersedia di pasar menghadirkan tantangan tersendiri dalam proses pemilihan alat yang tepat. Pemilihan *framework* otomatisasi pengujian tidak hanya didasarkan pada popularitas, namun harus mempertimbangkan keselarasan antara kebutuhan proyek dan kemampuan teknis alat tersebut. Abdulwareth et al. [5] menegaskan bahwa pemilihan alat pengujian otomatis harus dilakukan melalui evaluasi multikriteria yang komprehensif, dengan faktor krusial yang meliputi kompatibilitas lintas peramban, kemudahan integrasi ke dalam ekosistem *CI/CD*, serta biaya pemeliharaan skrip dalam jangka panjang. Faktor-faktor tersebut terbukti secara empiris menjadi penentu utama keputusan adopsi maupun migrasi kerangka kerja pengujian berbasis web, sebagaimana dikonfirmasi oleh Di Meglio et al. [2] melalui analisis pola migrasi pada 472 proyek *open-source*. Merujuk pada parameter kualitas tersebut, ekosistem pengembangan

perangkat lunak saat ini didominasi oleh beberapa kerangka kerja utama yang kerap dijadikan tolok ukur komparatif. Sebagai salah satu pionir yang paling banyak diadopsi, Selenium, yang diluncurkan pada tahun 2004, telah lama menjadi standar industri (*de facto*) berkat dukungan multibahasa dan kompatibilitas lintas peramban (*cross-browser*). Kendati demikian, arsitektur Selenium yang berbasis WebDriver menciptakan latensi komunikasi inheren karena beroperasi sebagai proksi antara skrip pengujian dan peramban. Sebagai alternatif, *framework* modern seperti Cypress (2015) dan Playwright (2020) menawarkan arsitektur berbeda masing-masing dengan pendekatan *in-browser execution* dan protokol WebSocket yang diklaim mampu meminimalisir latensi tersebut. Studi komparatif oleh Bhimanapati et al. [6] mengonfirmasi bahwa pendekatan arsitektur Cypress menawarkan kecepatan eksekusi yang lebih baik untuk pengujian lokal dibandingkan Selenium, meskipun Selenium masih unggul dalam cakupan kompatibilitas browser. Analisis arsitektural oleh García et al. [7] mempertegas hal ini dengan menyoroti bahwa penggunaan *Chrome DevTools Protocol* (CDP) pada Playwright memungkinkan komunikasi dua arah dengan latensi rendah (*low-latency*) yang tidak dimiliki oleh arsitektur *Representational State Transfer* (REST) berbasis pertukaran data HTTP satu arah pada Selenium, sehingga secara inheren meningkatkan akurasi penanganan elemen dinamis tanpa perlu intervensi manual yang berlebihan. Keunggulan ini diperkuat oleh riset Polkhovskaya et al. [8] yang menyoroti efektivitas Playwright dalam menangani konten dinamis serta fitur isolasi konteks yang mempercepat eksekusi tes paralel.

Evaluasi *framework* pengujian otomatis pada umumnya masih dilakukan secara deskriptif tanpa pendekatan pengambilan keputusan yang sistematis, padahal pemilihan *framework* melibatkan beberapa kriteria yang saling bertentangan, seperti kecepatan eksekusi yang tinggi namun kompatibilitas integrasi yang terbatas. Kesenjangan inilah yang mendasari penggunaan metode *Multi Criteria Decision Making* berupa *Distance to Ideal Alternative* pada penelitian ini, karena metode tersebut memungkinkan penentuan alternatif optimal secara objektif dari beberapa kriteria yang bersifat kuantitatif maupun kualitatif secara bersamaan.

Meskipun berbagai studi komparatif telah dilakukan sebelumnya, seperti penelitian oleh Prasetyo [9], masih terdapat kesenjangan penelitian (*research gap*) yang signifikan. Penelitian terdahulu umumnya berfokus pada perbandingan fitur fungsional semata, tanpa mengevaluasi secara mendalam efisiensi integrasi *framework* tersebut ke dalam ekosistem DevOps modern (*CI/CD*). Lambatnya waktu eksekusi (*runtime*) dalam *pipeline CI/CD* dapat berdampak langsung pada tertundanya rilis fitur dan penurunan produktivitas tim pengembang. Hubungan antara kematangan otomatisasi pengujian dan efektivitas proses rilis juga telah dibuktikan secara empiris oleh Wang et al. [10] yang menemukan bahwa proyek dengan tingkat kematangan otomatisasi pengujian yang lebih tinggi cenderung

menghasilkan kualitas perangkat lunak yang lebih baik serta siklus rilis yang lebih singkat, sehingga mendukung stabilitas dan efisiensi *pipeline CI/CD*. Oleh karena itu, diperlukan analisis komprehensif yang tidak hanya membandingkan kecepatan, tetapi juga kemudahan integrasi menggunakan metode pengambilan keputusan objektif seperti *Distance to Ideal Alternative* (DIA). Fleksibilitas metode DIA dalam menangani kriteria yang saling bertentangan telah divalidasi dalam berbagai studi sistem pendukung keputusan terbaru, termasuk oleh Putra et al. [11] yang menegaskan akurasi metode ini dalam menentukan peringkat preferensi berbasis jarak ideal.

Penerapan metode DIA untuk seleksi alat teknis telah divalidasi oleh Adiatma et al. [12] serta Fakhrii et al. [13] yang membuktikan efektivitas metode ini dalam memberikan peringkat preferensi yang objektif pada kasus pemilihan perangkat lunak. Berbeda dari Prasetyo [9] yang menerapkan DIA pada lingkungan eksekusi lokal dengan fokus komparasi fitur UI, penelitian ini mengintegrasikan pengujian ke dalam *pipeline CI/CD* berbasis GitHub Actions pada aplikasi *Swag Labs*. Penambahan kriteria Kompatibilitas *CI/CD* sebagai dimensi evaluasi menjadi pembeda utama yang menjadikan hasil perankingan lebih relevan dengan kebutuhan praktik DevOps modern. Selain itu, perbedaan performa antar *framework* turut divalidasi secara statistik melalui uji signifikansi (*Kruskal-Wallis* dan uji lanjut *Dunn*), sehingga hasil perankingan DIA yang dihasilkan tidak hanya bersifat deskriptif, melainkan didukung oleh bukti statistik yang teruji secara metodologis. Berangkat dari celah penelitian dan justifikasi metode tersebut, penelitian ini diharapkan dapat memberikan rekomendasi yang objektif dan terukur bagi praktisi pengembang perangkat lunak, penguji perangkat lunak serta tim DevOps dalam menentukan *framework* otomasi antarmuka pengguna yang paling optimal. Melalui evaluasi kinerja komparatif dan efisiensi integrasi ke dalam *pipeline CI/CD*, luaran dari penelitian ini diharapkan mampu menjadi panduan strategis untuk meminimalisir kendala *flaky tests*, meningkatkan efisiensi pemeliharaan skrip, dan mempercepat siklus rilis dalam ekosistem pengembangan perangkat lunak modern.

II. METODE

Penelitian ini menganalisis kinerja teknis dari tiga kerangka kerja pengujian otomatis yang diajukan sebagai alternatif (A).

A. Objek Penelitian

Penelitian ini berfokus pada evaluasi tiga kerangka kerja (*framework*) otomasi *User Interface* (UI), yaitu Selenium, Cypress, dan Playwright. Pemilihan ketiga *framework* tersebut didasarkan pada perbedaan karakteristik arsitektur yang mendasarinya: Selenium merepresentasikan pendekatan berbasis *WebDriver*, Cypress mengadopsi eksekusi *in-browser*, dan Playwright menggunakan protokol *Chrome DevTools Protocol* (CDP). Diversitas arsitektur ini secara

metodologis diperlukan untuk mengukur dampak desain internal *framework* terhadap efisiensi waktu eksekusi dan kompatibilitas *pipeline CI/CD* secara empiris. Spesifikasi detail dari ketiga kerangka kerja tersebut dirangkum pada Tabel I.

TABEL I
DAFTAR ALTERNATIF KERANGKA KERJA PENGUJIAN

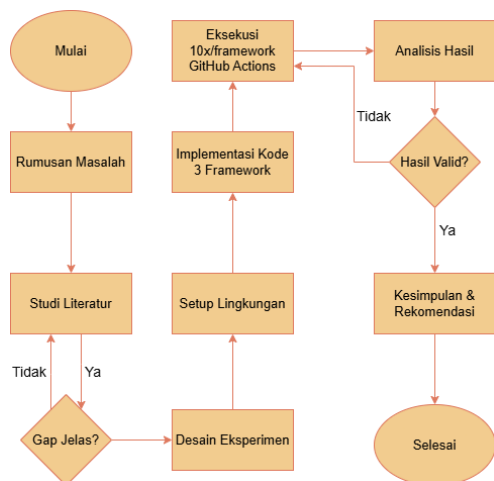
Alternatif	Kerangka Kerja	Deskripsi Arsitektur
A_1	Selenium	Representasi arsitektur tradisional berbasis <i>WebDriver</i> .
A_2	Cypress	Representasi arsitektur <i>all-in-one</i> JavaScript.
A_3	Playwright	Representasi arsitektur modern berbasis CDP/ <i>WebSocket</i> .

Subjek pengujian yang digunakan dalam penelitian ini adalah aplikasi web *Sauce Demo* (*Swag Labs*). Secara fungsional, aplikasi ini merepresentasikan alur kerja standar perdagangan elektronik modern, yang mencakup modul otentikasi pengguna, navigasi katalog produk dinamis, manajemen keranjang belanja, hingga simulasi penyelesaian transaksi (*checkout*).

Pemilihan *Sauce Demo* sebagai subjek eksperimen didasarkan pada dua alasan teknis utama. Pertama, aplikasi ini menyediakan lingkungan yang stabil dan terkontrol (statis secara fungsionalitas utama), sehingga menjamin reproduktibilitas skenario pengujian. Kondisi statis ini krusial untuk memastikan bahwa setiap perbedaan waktu eksekusi (*runtime*) murni disebabkan oleh arsitektur internal *framework* yang diuji, bukan oleh fluktuasi latensi atau perubahan kode pada sisi server aplikasi. Kedua, kompleksitas elemen pada situs ini cukup representatif untuk menguji ketahanan alat otomasi terhadap isu *flaky tests*. Aplikasi ini merupakan *prototype e-commerce* yang dikembangkan dan didedikasikan secara resmi oleh *Sauce Labs* khusus untuk keperluan pengujian perangkat lunak, sehingga secara inheren valid digunakan sebagai subjek evaluasi kinerja alat pengujian otomatis.

B. Alur Penelitian

Penelitian dilaksanakan mengikuti tahapan sistematis yang ditunjukkan pada Gambar 1.



Gambar 1. Alur Penelitian

Gambar 1 menunjukkan alur penelitian yang dilaksanakan secara sistematis mulai dari perumusan masalah hingga analisis hasil. Fase awal dimulai dengan merumuskan masalah berdasarkan tinjauan literatur terhadap penelitian-penelitian terdahulu yang berfokus pada komparasi alat otomasi UI seperti kajian metrik kecepatan oleh Bhimanapati et al. [6] dan analisis struktural oleh Prasetyo [9]. Melalui studi pustaka tersebut, divalidasi sebuah *research gap* atau celah kebaruan (*novelty*), di mana evaluasi performa *framework* belum dikaitkan secara empiris dengan efisiensi integrasinya di dalam *pipeline CI/CD* menggunakan metode objektif *Distance to Ideal Alternative* (DIA).

Pada tahap desain eksperimen, penelitian merancang tiga skenario pengujian *end-to-end* (E2E) yang diimplementasikan pada tiga *framework* (Selenium, Cypress, Playwright) dengan aplikasi objek pengujian Swag Labs, sebagaimana dirincikan pada Tabel II.

TABEL II
RINCIAN SKENARIO PENGUJIAN

Skenario	Jumlah Test Case	Kompleksitas	Deskripsi
S1: Autentikasi Pengguna (<i>Login</i>)	3	Rendah	<i>Login</i> dengan kredensial valid dan tidak valid (2 test), serta <i>logout</i> (1 test)
S2: Interaksi Antarmuka Dinamis	7	Sedang	Pengurutan produk berdasarkan nama dan harga (4 test), manajemen keranjang: tambah, hapus, lihat (3 test)
S3: Transaksi <i>End to End</i> (<i>Checkout</i>)	4	Tinggi	Alur bertingkat: tambah ke keranjang → <i>checkout</i> → isi data → konfirmasi pesanan (4 test, bersifat kumulatif)

Adapun alur serta tingkat kompleksitas dari masing-masing skenario pengujian tersebut diilustrasikan secara visual pada Gambar 2.



Gambar 2. Alur dan Tingkat Kompleksitas Skenario Pengujian

Selanjutnya, *setup* lingkungan penelitian dilakukan pada dua lingkungan paralel: pengembangan lokal dan eksekusi otomatis *CI/CD*.

Tahap implementasi kode menghasilkan skrip pengujian untuk ketiga *framework*, diikuti eksekusi pengujian sebanyak 10 kali per *framework* pada platform GitHub Actions untuk pengumpulan data empiris. Pengulangan sebanyak 10 kali ini dilakukan untuk mendapatkan nilai rata-rata waktu eksekusi yang lebih stabil dan akurat. Mengingat lingkungan eksekusi berbasis *cloud* (*CI/CD*) rentan terhadap perubahan kecepatan jaringan, jumlah pengulangan ini dinilai memadai untuk meminimalkan bias pada hasil pengujian tanpa memperlama proses secara keseluruhan.

Selanjutnya, data hasil pengujian dianalisis menggunakan metode *Distance to Ideal Alternative* (DIA). Metode ini dipilih karena kemampuannya dalam memberikan peringkat secara objektif berdasarkan kriteria yang memiliki satuan ukur berbeda. Penggunaan metode DIA untuk menyeleksi dan membandingkan kinerja *framework* otomasi pengujian juga telah didukung dan tervalidasi oleh literatur terdahulu, sebagaimana diterapkan pada penelitian Adiatma et al. [12] serta Fakhrii et al. [13]. Perhitungan menggunakan metode DIA ini terdiri dari empat tahapan matematis (normalisasi, pembobotan, solusi ideal positif, jarak Euclidean), yang kemudian diikuti dengan analisis hasil berdasarkan tiga kriteria berbobot: rata-rata *runtime* (C_1), validitas eksekusi (C_2), dan skor kompatibilitas *CI/CD* (C_3).

C. Alat dan Bahan

1. Perangkat Keras (*Hardware*): Penelitian ini menggunakan dua lingkungan kerja yang dipisahkan secara fungsional untuk menjamin validitas data. Lingkungan Pengembangan digunakan secara khusus untuk penulisan kode dan analisis awal. Sementara itu, Lingkungan Pengujian yang berbasis server *CI/CD* digunakan untuk pengambilan data *benchmark* agar hasil stabil dan terisolasi dari beban kerja lokal. Spesifikasi perangkat keras untuk kedua lingkungan tersebut dirincikan pada Tabel III.

TABEL III
SPESIFIKASI LINGKUNGAN PERANGKAT KERAS

Komponen	Lingkungan Pengembang	Lingkungan Pengujian
Platform	PC Lokal	GitHub Hosted Runner
Sistem Operasi	Windows 11	Linux (ubuntu-latest)
CPU	AMD Ryzen 5 5600G	2 vCPU
RAM	16 GB	8 GB

2. Perangkat Lunak (*Software*): Penelitian ini memanfaatkan serangkaian perangkat lunak untuk menunjang tahap pengembangan, manajemen versi, eksekusi otomatis, hingga analisis data. Terdapat tiga kerangka kerja (*framework*) otomatisasi yang digunakan sebagai objek pembandingan utama dalam studi ini. Rincian lengkap perangkat lunak beserta fungsinya disajikan dalam Tabel IV.

TABEL IV
DAFTAR PERANGKAT LUNAK PENELITIAN

Kategori	Perangkat Lunak	Keterangan
<i>Development</i>	Visual Studio Code	Lingkungan penulisan kode
<i>Runtime</i>	Node.js (LTS)	Eksekusi program JavaScript
<i>Version Control</i>	Git	Manajemen versi & repositori
<i>Framework Otomasi</i>	selenium-webdriver cypress @playwright/test	Versi 4.x Versi 13.x Versi 1.x
Analisis Data	Ms. Excel / Google Sheets	Perhitungan metode DIA
Analisis Statistik	Python (SciPy, Pandas)	Uji normalitas, homogenitas varians, dan signifikansi (<i>Shapiro-Wilk, Levene, Kruskal-Wallis, Dunn's test</i>)

Implementasi ketiga framework mempertimbangkan kompatibilitas bahasa pemrograman dan pengalaman teknis. Selenium dikonfigurasi menggunakan paket *selenium-webdriver* berbasis Node.js guna menyeragamkan ekosistem eksekusi dengan Cypress dan Playwright, sehingga variasi waktu eksekusi yang terukur dapat diatribusikan pada perbedaan arsitektur *framework* dan meminimalkan pengaruh perbedaan bahasa pemrograman sebagai variabel pengganggu (*confounding variable*). Cypress menggunakan JavaScript sebagai satu-satunya bahasa yang didukung platform, selaras dengan kecepatannya dalam pengujian UI. Sementara itu, Playwright dikonfigurasi dengan TypeScript setelah evaluasi mendalam untuk mengatasi hambatan implementasi dengan bahasa lain [14].

Pipeline CI/CD pada aplikasi web modern umumnya memanfaatkan *runner* Linux berbasis *ubuntu-latest* dengan lingkungan eksekusi Node.js versi LTS. Tahap *build* dan

pengujian otomatisasi menggunakan perintah seperti *npm ci* untuk instalasi dependensi bersih dan *npm run build* untuk proses kompilasi aplikasi. Seluruh proses eksekusi tersebut diorkestrasi menggunakan platform GitHub Actions. Platform ini dipilih karena secara khusus dirancang untuk mengintegrasikan pengujian ke dalam *workflow* deklaratif, yang memungkinkan eksekusi proses secara paralel sehingga mampu mereduksi total waktu *pipeline* serta meminimalisir potensi kesalahan manual [15]. Lebih lanjut, studi komparatif oleh Moñ et al. [16] membuktikan bahwa Playwright secara konsisten menunjukkan efisiensi konsumsi sumber daya komputasi yang lebih baik dibandingkan Cypress maupun Selenium, dengan rata-rata utilisasi CPU yang lebih rendah pada lebih dari separuh skenario pengujian. Karakteristik efisiensi sumber daya ini menjadikan Playwright sebagai pilihan kerangka kerja yang strategis dalam lingkungan eksekusi berbasis *cloud* seperti GitHub Actions, di mana alokasi vCPU dan memori bersifat terbatas. Rincian konfigurasi tambahan lingkungan *CI/CD* untuk masing-masing *framework* yang diuji disajikan pada Tabel V.

TABEL V
KONFIGURASI TAMBAHAN LINGKUNGAN *CI/CD*

Aspek	Selenium	Cypress	Playwright
Runs-on	ubuntu-latest	ubuntu-latest	ubuntu-latest
Trigger	push, pull_request	push, pull_request, workflow_dispatch	push, pull_request
Actions/ checkout	actions/checkout@v4	actions/checkout@v4	actions/checkout@v4
Actions/ setup-node	setup-node@v4	setup-node@v4	setup-node@v4

Sebagaimana telah dijelaskan pada bagian Alur Penelitian, pengulangan sebanyak sepuluh kali per skenario menghasilkan total tiga puluh eksekusi data untuk ketiga framework pada setiap skenario. Rata-rata dari sepuluh pengulangan tersebut selanjutnya digunakan sebagai nilai runtime representatif pada matriks keputusan yang dijabarkan pada kriteria evaluasi berikut.

D. Kriteria Evaluasi

Evaluasi kinerja pada penelitian ini didasarkan pada tiga kriteria terukur. Kriteria pertama adalah rata-rata *runtime* (C_1), yaitu atribut biaya (*cost*) yang diperoleh dari rata-rata waktu yang dibutuhkan setiap *framework* untuk menyelesaikan skenario pengujian secara keseluruhan; semakin kecil nilai waktu eksekusi dalam satuan detik, semakin baik kinerja *framework* yang bersangkutan.

Kriteria kedua adalah status validitas eksekusi (C_2), yaitu variabel kontrol (*control variable*) kualitatif untuk mengukur kelayakan operasional, dengan status "Valid" diberikan apabila pengujian berjalan sukses tanpa kegagalan sistem (*crash*) dan status "Non-Valid" diberikan apabila terjadi

kegagalan eksekusi. Status ini diverifikasi secara otomatis melalui *exit code* pada setiap *job* GitHub Actions, dengan ketentuan *exit code* bernilai nol dikategorikan valid dan bernilai bukan nol dikategorikan tidak valid, yang selanjutnya didukung dengan pemeriksaan manual terhadap berkas *test report* untuk memastikan tidak terdapat kegagalan tersembunyi yang luput dari deteksi *exit code*.

Kriteria ketiga adalah skor kompatibilitas *CI/CD* (C_3), yaitu atribut keuntungan (*benefit*) yang mengukur kemudahan dan stabilitas integrasi kerangka kerja ke dalam ekosistem *pipeline CI/CD*. Karena bersifat kualitatif, nilainya diquantifikasi menggunakan skala ordinal 5 poin yang diadaptasi dari prinsip evaluasi heuristik perangkat lunak, sebagaimana lazim diterapkan dalam pemodelan MCDM untuk mengonversi parameter kualitatif menjadi matriks matematis yang terukur. Rubrik penilaian tersusun secara gradual: skor 5 (Sangat Baik) diberikan apabila integrasi bersifat *plug-and-play* dengan dukungan penuh (*native*) melalui *official actions/plugins* tanpa memerlukan skrip tambahan; skor 4 (Baik) diberikan apabila hanya membutuhkan konfigurasi parameter minimal, namun tetap didukung penuh oleh dokumentasi resmi tanpa modifikasi lingkungan server; skor 3 (Moderat) diberikan apabila menuntut instalasi dan konfigurasi dependensi secara manual, seperti pengaturan *browser driver* atau *display server* tambahan pada Linux, namun sistem tetap berjalan fungsional dan stabil; skor 2 (Buruk) diberikan apabila memerlukan skrip modifikasi pihak ketiga (*workarounds*) yang kompleks dan rentan mengalami ketidakstabilan eksekusi (*flaky*); dan skor 1 (Sangat Buruk) diberikan apabila kerangka kerja tidak kompatibel atau secara konsisten mengalami kegagalan sistem saat dieksekusi di dalam *pipeline CI/CD* terotomatisasi.

E. Uji Signifikansi Statistik

Perbedaan rata-rata *runtime* antar *framework* diuji signifikansinya secara statistik menggunakan data sepuluh pengulangan tiap *framework*. Sebelum pengujian utama, data terlebih dahulu diperiksa asumsi kenormalan dan kesetaraan variansnya untuk menentukan jenis uji yang sesuai.

Uji kenormalan data menggunakan *Shapiro-Wilk test*, sedangkan uji kesetaraan varians antarkelompok menggunakan *Levene's test*. Uji signifikansi utama menggunakan *Kruskal-Wallis test* (nonparametrik) sebagai acuan karena asumsi kenormalan dan kesetaraan varians tidak sepenuhnya terpenuhi, dengan *one-way ANOVA* sebagai uji pembandingan (parametrik). Untuk mengetahui pasangan *framework* yang berbeda signifikan, dilakukan uji lanjut (*post-hoc*) menggunakan *Dunn's test* dengan koreksi *Bonferroni*. Seluruh pengujian statistik tersebut dihitung menggunakan Python dengan pustaka SciPy dan Pandas, dengan taraf signifikansi $\alpha = 0,05$ sebagai acuan dalam menentukan signifikansi hasil uji.

F. Metode Distance to Ideal Alternative (DIA)

Metode *Distance to Ideal Alternative* dipilih dibandingkan metode *Multi Criteria Decision Making* lain dengan pertimbangan berikut. Metode *Analytical Hierarchy Process* memerlukan serangkaian perbandingan berpasangan antarkriteria yang umumnya melibatkan penilaian pakar, sehingga kurang sesuai untuk penelitian dengan kriteria yang telah terukur secara kuantitatif seperti pada penelitian ini. Metode *Simple Additive Weighting* menjumlahkan skor terbobot secara linear tanpa memperhitungkan jarak terhadap solusi ideal, sehingga kurang sensitif terhadap perbedaan performa antaralternatif yang relatif kecil. Metode *Distance to Ideal Alternative* memiliki prinsip perhitungan yang serupa dengan *TOPSIS*, yaitu mengukur jarak setiap alternatif terhadap solusi ideal positif, namun dengan tahapan komputasi yang lebih ringkas karena tidak memerlukan perhitungan jarak terhadap solusi ideal negatif. Karakteristik tersebut menjadikan metode ini sesuai untuk penelitian dengan jumlah kriteria terbatas seperti pada penelitian ini.

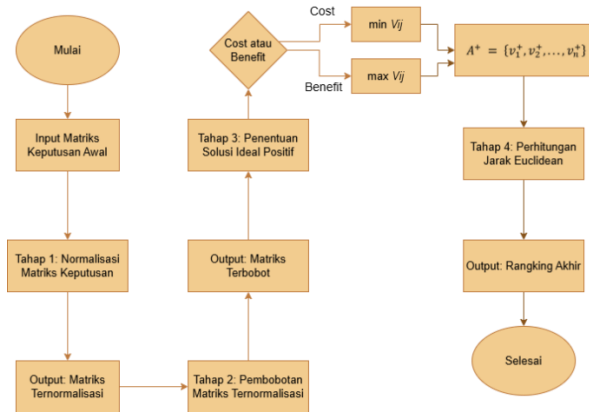
Analisis data untuk menentukan peringkat *framework* dilakukan menggunakan metode Pengambilan Keputusan Multikriteria atau *Multi-Criteria Decision Making* (MCDM), yakni *Distance to Ideal Alternative* (DIA). Secara prinsipil dalam sebuah Sistem Pendukung Keputusan (SPK), setiap kriteria evaluasi harus memiliki daya pembeda antar alternatif. Apabila terdapat kriteria yang menghasilkan nilai seragam atau tidak memiliki variasi data sama sekali pada seluruh *framework* yang diuji, maka kriteria tersebut hanya difungsikan sebagai prasyarat kelayakan awal.

Kriteria yang seragam ini tidak akan diikutsertakan dalam perhitungan jarak Euclidean guna mencegah terjadinya bias pada hasil akhir. Sebagai penyesuaian, alokasi bobot pada kriteria-kriteria yang tersisa akan dihitung dan dinormalisasi ulang secara proporsional. Penentuan bobot 60 persen untuk kriteria *runtime* dan 40 persen untuk kriteria kompatibilitas *CI/CD* didasarkan pada dampak operasional masing-masing kriteria terhadap efisiensi *pipeline CI/CD*. Kriteria *runtime* diberi bobot dominan karena secara langsung menentukan lama siklus *feedback loop* pengujian otomatis, sehingga berdampak signifikan terhadap kecepatan siklus rilis perangkat lunak. Kriteria kompatibilitas *CI/CD* tetap dipertimbangkan sebagai kriteria penting, namun bersifat sekunder karena kendala pada aspek ini umumnya dapat diatasi melalui penyesuaian konfigurasi *workflow*, berbeda dengan keterbatasan *runtime* yang bersifat intrinsik terhadap arsitektur masing-masing *framework*. Rasio pembobotan ini juga sejalan dengan rentang variasi data pada matriks keputusan, di mana rentang nilai *runtime* antar-*framework*, yaitu 11,396 hingga 41,533 detik, jauh lebih lebar dibandingkan rentang skor kompatibilitas *CI/CD*, yaitu skor 3 hingga 5, sehingga kriteria *runtime* secara proporsional layak diberikan bobot yang lebih besar.

Untuk menguji kestabilan (*robustness*) hasil perankingan terhadap subjektivitas penentuan bobot tersebut, dilakukan pula analisis sensitivitas dengan menghitung ulang nilai R_i

pada beberapa skenario kombinasi bobot w_1 (*runtime*) dan w_3 (kompatibilitas *CI/CD*) di sekitar bobot utama yang digunakan (60:40), yaitu 50:50, 70:30, dan 40:60. Apabila peringkat akhir tetap konsisten pada seluruh skenario tersebut, maka kesimpulan penelitian dapat dianggap tidak sensitif terhadap pilihan bobot subjektif.

Adapun tahapan perhitungan metode DIA yang diterapkan dalam penelitian ini dapat dilihat secara sistematis pada Gambar 3.



Gambar 3. Alur Proses Perhitungan Metode Dia

III. HASIL DAN PEMBAHASAN

Penelitian ini melakukan analisis komparatif performa dan integrasi *framework* Selenium, Cypress, dan Playwright pada lingkungan *CI/CD* menggunakan metode *Distance to Ideal Alternative* (DIA). Data empiris dikumpulkan melalui eksekusi otomatis di GitHub Actions untuk memastikan objektivitas hasil pengujian. Berikut adalah temuan dari penelitian ini:

A. Hasil Pengumpulan Data

Data mentah (*raw data*) hasil perekaman waktu eksekusi (*runtime*) dari seluruh iterasi pengujian disajikan secara terperinci pada Tabel VI guna mendukung prinsip transparansi dan reproduktibilitas penelitian, seluruh log eksekusi, detail artefak, dan kode sumber (*source code*) pengujian dapat diakses secara publik melalui repositori GitHub pada tautan berikut:

<https://github.com/katanyaaman/test-reports-benchmark>

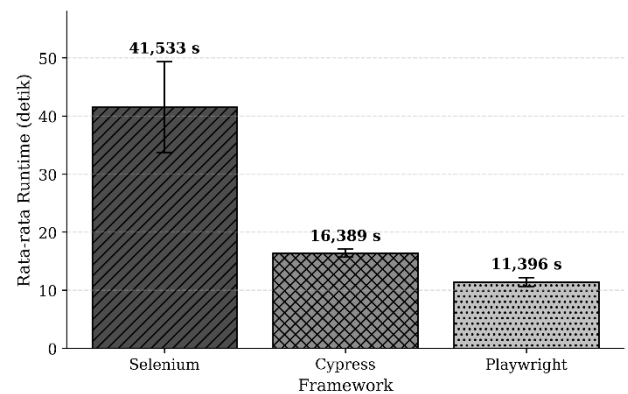
TABEL VI
DATA MENTAH WAKTU EKSEKUSI PENGUJIAN (DETIK)

Iterasi Eksekusi	Selenium (A_1)	Cypress (A_2)	Playwright (A_3)
Percobaan 1	61,813	16,379	12,161
Percobaan 2	38,312	15,740	11,086
Percobaan 3	41,755	16,793	12,045
Percobaan 4	36,889	15,395	10,276

Percobaan 5	36,196	16,958	11,020
Percobaan 6	45,086	17,303	12,775
Percobaan 7	43,221	15,445	10,600
Percobaan 8	35,343	16,154	11,492
Percobaan 9	36,334	16,469	10,974
Percobaan 10	40,376	17,257	11,529
Rata-rata C_1	41,533	16,389	11,396

Data empiris hasil pengujian direkapitulasi ke dalam tiga kriteria evaluasi yang telah ditetapkan pada bagian metodologi, yakni rata-rata *runtime* (C_1), status validitas eksekusi (C_2), dan skor kompatibilitas *CI/CD* (C_3).

Pada kriteria C_1 , Playwright mencatat *runtime* tercepat dengan rata-rata 11,396 detik, diikuti Cypress 16,389 detik, sedangkan Selenium memiliki rata-rata tertinggi 41,533 detik. Perbedaan signifikan ini menunjukkan efisiensi waktu eksekusi Playwright 3,6 kali lebih baik dibandingkan Selenium, yang memberikan dampak langsung pada *feedback loop* dalam *pipeline CI/CD*, sebagaimana divisualisasikan pada Gambar 4.



Gambar 4. Grafik Perbandingan Rata-rata Runtime

Sebagai pelengkap, Playwright juga tercatat 43,8 persen lebih cepat dibandingkan Cypress (11,396 detik berbanding 16,389 detik), menunjukkan bahwa keunggulan Playwright konsisten terhadap kedua *framework* pembandingan, bukan hanya terhadap Selenium. Pada kriteria C_2 , ketiga *framework* mencatatkan keberhasilan eksekusi 100% pada seluruh skenario pengujian. Tidak ditemukan kegagalan sistem (*crash*), sehingga ketiga *framework* dinyatakan Valid. Mengingat ketiga *framework* menunjukkan status eksekusi yang seragam pada C_2 , kriteria tersebut berfungsi sebagai prasyarat kelayakan awal (*eligibility threshold*) dan tidak diikutsertakan dalam perhitungan jarak Euclidean.

Pada kriteria C_3 , Cypress dan Playwright berhasil memperoleh skor maksimal yaitu 5 (Sangat Baik). Hal ini dikarenakan keduanya menyediakan dukungan integrasi *plug-and-play* yang mulus melalui ketersediaan *official GitHub Action*. Sebaliknya, Selenium dinilai dengan skor 3 (Moderat). Penurunan skor ini terjadi karena Selenium menuntut intervensi konfigurasi manual di dalam *pipeline* lingkungan Linux, yang mengharuskan instalasi peramban Chrome, *ChromeDriver*, serta *display server* Xvfb secara terpisah.

B. Hasil Uji Signifikansi Statistik

Hasil pemeriksaan asumsi dan uji signifikansi disajikan pada Tabel VII. Karena data tidak sepenuhnya memenuhi asumsi kenormalan dan kesetaraan varians, uji nonparametrik *Kruskal-Wallis* digunakan sebagai acuan utama. Hasil pengujian menunjukkan perbedaan *runtime* antartetiga *framework* bersifat signifikan (nilai p di bawah 0,001). Untuk mengetahui pasangan *framework* mana saja yang berbeda secara signifikan, dilakukan uji lanjut *Dunn* (dengan koreksi *Bonferroni*) yang hasilnya disajikan pada Tabel VIII.

TABEL VII
HASIL UJI ASUMSI DAN UJI SIGNIFIKANSI

Pengujian	Hasil	Kesimpulan
Uji kenormalan (Shapiro-Wilk) data Selenium	nilai $p = 0,003$	Tidak normal
Uji kenormalan (Shapiro-Wilk) data Cypress	nilai $p = 0,469$	Normal
Uji kenormalan (Shapiro-Wilk) data Playwright	nilai $p = 0,902$	Normal
Uji kesetaraan varians (Levene) antarkelompok	nilai $p = 0,018$	Varians tidak setara
Uji signifikansi utama (Kruskal-Wallis)	$H = 25,806$; $df = 2$; nilai $p = 0,0000025$	Signifikan
Uji signifikansi pembandingan (One-way ANOVA)	$F = 124,99$; $df = 2$ dan 27 ; nilai $p < 0,001$	Signifikan (hasil konsisten)

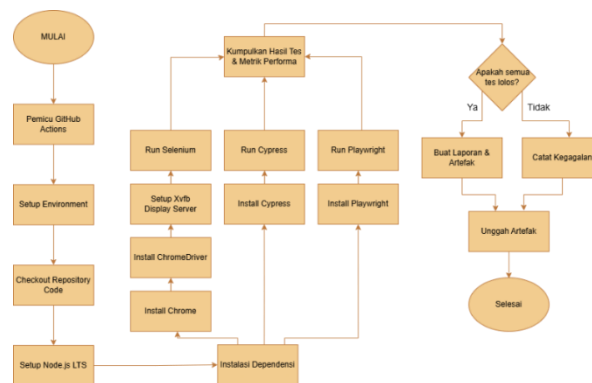
TABEL VIII
HASIL UJI LANJUT (POST-HOC) PAIRWISE PERBEDAAN RUNTIME

Pasangan Framework	Nilai p	Kesimpulan
Selenium dan Cypress	0,033	Berbeda signifikan
Selenium dan Playwright	$< 0,001$	Berbeda signifikan
Cypress dan Playwright	0,033	Berbeda signifikan

C. Implementasi Pipeline CI/CD

Data waktu eksekusi (*runtime*) yang digunakan untuk perhitungan murni diambil dari durasi pengujian inti saja. Waktu tambahan yang dihabiskan untuk persiapan sistem (seperti proses instalasi awal di lingkungan *CI/CD*) tidak dimasukkan agar perbandingan performa antar *framework* tetap adil dan objektif.

Untuk memastikan bahwa setiap iterasi pengujian berjalan pada lingkungan yang konsisten, tahapan eksekusi di dalam *pipeline* dirancang secara terstruktur sebagaimana diilustrasikan pada Gambar 5.



Gambar 5. Alur Kerja Pipeline CI/CD

Gambar 5 menggambarkan alur kerja *pipeline CI/CD* yang diimplementasikan pada platform GitHub Actions untuk eksekusi pengujian otomatis ketiga *framework*. *Pipeline* dimulai dari *trigger git push* ke *branch* utama yang mengaktifkan GitHub Actions dengan *runner ubuntu-latest*. Tahap *setup environment* mencakup instalasi Node.js LTS dan *npm ci* untuk dependensi bersih.

Hasil dari ketiga *branch* diintegrasikan melalui proses "*Merge Result From 3 Branch*" yang mengumpulkan metrik performa: rata-rata *runtime* (C_1), validitas eksekusi (C_2), dan skor kompatibilitas *CI/CD* (C_3). Titik keputusan memvalidasi keberhasilan pengujian sebelum generasi laporan dan *upload artifacts* ke GitHub. *Pipeline* ini memastikan reproduktibilitas dan konsistensi eksekusi yang mencerminkan praktik DevOps modern.

D. Analisis Perhitungan Metode DIA

Meskipun C_2 disertakan dalam matriks keputusan sebagai representasi data pengujian secara utuh, kriteria tersebut dikecualikan dari perhitungan jarak Euclidean karena tidak memiliki variasi nilai antar alternatif, sehingga tidak memberikan daya pembeda dalam proses pemeringkatan. Oleh karena itu, perhitungan matematis difokuskan pada kriteria C_1 dan C_3 , di mana alokasi bobot dinormalisasi ulang menjadi 60% untuk rata-rata *runtime* C_1 dan 40% untuk skor kompatibilitas *CI/CD* C_3 . Penerapan metode DIA diawali dengan penyusunan matriks keputusan awal X_{ij} sebagaimana ditunjukkan pada Tabel IX.

TABEL IX
MATRIKS KEPUTUSAN AWAL (X_{ij})

Alternatif (A)	C_1 (Runtime Cost)	C_2 (Validitas)	C_3 (CI/CD - Benefit)
Selenium (A_1)	41,533	Valid	3
Cypress (A_2)	16,389	Valid	5
Playwright (A_3)	11,396	Valid	5

Berikut adalah tahapan perhitungan rinci untuk mendapatkan peringkat akhir.

Tahap pertama adalah normalisasi matriks (r_{ij}), yang dilakukan untuk menyamakan satuan antar kriteria, khususnya pada kriteria kuantitatif C_1 dan C_3 . Pada kriteria *runtime* (C_1), nilai pembagi dihitung sebagai berikut:

$$\begin{aligned}\text{Cost Pembagi} &= \sqrt{(41,533^2 + 16,389^2 + 11,396^2)} \\ &= \sqrt{(1724,990 + 268,600 + 129,869)} \\ &= \sqrt{2123,459} \approx 46,081\end{aligned}$$

Nilai normalisasi untuk masing-masing alternatif adalah Selenium ($\frac{41,533}{46,081} = 0,901$), Cypress ($\frac{16,389}{46,081} = 0,356$), dan Playwright ($\frac{11,396}{46,081} = 0,247$). Adapun pada kriteria kompatibilitas CI/CD (C_3), nilai pembagi dihitung sebagai berikut:

$$\text{Benefit Pembagi} = \sqrt{(3^2 + 5^2 + 5^2)} = \sqrt{(59)} \approx 7,681$$

Nilai normalisasi untuk masing-masing alternatif adalah Selenium ($\frac{3}{7,681} = 0,391$), Cypress ($\frac{5}{7,681} = 0,651$), dan Playwright ($\frac{5}{7,681} = 0,651$).

Tahap kedua adalah pembobotan matriks (v_{ij}). Setelah normalisasi, nilai dikalikan dengan bobot preferensi yang telah disesuaikan, yaitu $w_1 = 0,60$ dan $w_3 = 0,40$. Hasil pembobotan dapat dilihat pada Tabel X.

TABEL X
Matriks Terbobot (v_{ij})

Alternatif (A_i)	C_1 (0,60 x r_{1i})	C_3 (0,40 x r_{3i})
Selenium (A_1)	0,541	0,156
Cypress (A_2)	0,214	0,260
Playwright (A_3)	0,148	0,260

Tahap ketiga adalah penentuan solusi ideal positif (A^+) berdasarkan atribut masing-masing kriteria. Pada kriteria C_1 (*Runtime – Cost*), nilai terkecil dianggap terbaik sehingga solusi ideal ditetapkan sebesar 0,148, sedangkan pada kriteria C_3 (*CI/CD – Benefit*), nilai terbesar dianggap terbaik sehingga solusi ideal ditetapkan sebesar 0,260. Dengan demikian, solusi ideal positif adalah $A^+ = \{0,148; 0,260\}$.

Tahap terakhir adalah perhitungan jarak Euclidean (R_i) setiap alternatif terhadap solusi ideal positif. Untuk alternatif Selenium (A_1):

$$\begin{aligned}R_1 &= \sqrt{(0,541 - 0,148)^2 + (0,156 - 0,260)^2} \\ R_1 &= \sqrt{0,1544 + 0,0108} \approx 0,406\end{aligned}$$

Untuk alternatif Cypress (A_2):

$$\begin{aligned}R_2 &= \sqrt{(0,214 - 0,148)^2 + (0,260 - 0,260)^2} \\ R_2 &= \sqrt{0,0044 + 0} \approx 0,066\end{aligned}$$

Untuk alternatif Playwright (A_3):

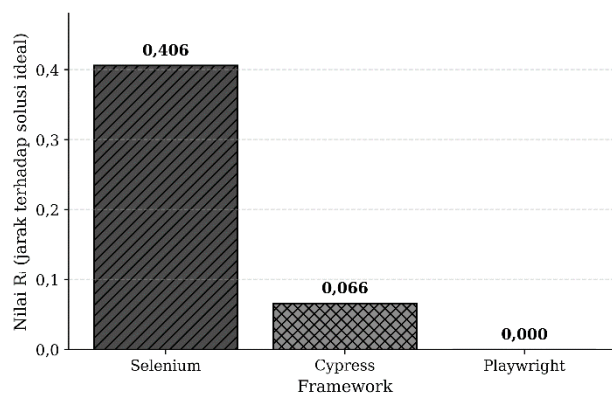
$$\begin{aligned}R_3 &= \sqrt{(0,148 - 0,148)^2 + (0,260 - 0,260)^2} \\ R_3 &= \sqrt{0 + 0} = 0,000\end{aligned}$$

Berikut adalah visualisasi hasil akhir metode DIA, di mana nilai R_i yang mendekati 0 menunjukkan performa yang semakin ideal. Hasil lengkap pemeringkatan disajikan dalam Tabel XI.

TABEL XI
HASIL AKHIR PERANGKINAN

Peringkat	Framework	Nilai R_i	Keterangan
1	Playwright	0,000	Sangat Direkomendasikan (Ideal)
2	Cypress	0,066	Direkomendasikan
3	Selenium	0,406	Cukup

Signifikansi statistik yang telah dibuktikan pada bagian sebelumnya turut memperkuat keandalan nilai R_i yang dihasilkan metode *Distance to Ideal Alternative*, karena perbedaan performa yang mendasari peringkat tersebut telah terbukti bukan sekadar variasi acak, melainkan perbedaan yang signifikan secara statistik. Nilai R_i sebesar 0,000 pada Playwright menunjukkan bahwa *framework* tersebut secara simultan mencapai performa terbaik pada kedua kriteria yang diperhitungkan, yaitu *runtime* tercepat dan skor kompatibilitas CI/CD maksimal, sehingga posisinya berimpit dengan solusi ideal positif. Nilai R_i sebesar 0,406 pada Selenium mencerminkan defisit pada kedua dimensi evaluasi sekaligus, bukan hanya pada satu kriteria. Cypress menempati posisi antara dengan nilai R_i sebesar 0,066, karena meskipun skor kompatibilitas CI/CD-nya setara dengan Playwright, waktu eksekusinya masih 43,8 persen lebih lambat dibandingkan Playwright.



Gambar 6. Grafik Perbandingan Jarak Euclidean (R_i)

Visualisasi peringkat akhir berdasarkan nilai jarak Euclidean (R_i) disajikan pada Gambar 6. Grafik tersebut mempertegas signifikansi keunggulan Playwright yang memiliki jarak 0,000 (berimpit dengan solusi ideal positif),

kontras dengan Selenium yang memiliki rentang jarak terjauh (0,406).

E. Analisis Sensitivitas

Untuk menguji kestabilan hasil pemeringkatan terhadap perubahan bobot kriteria, dilakukan perhitungan ulang nilai R_i menggunakan metode *Distance to Ideal Alternative* yang sama pada empat kombinasi bobot yang berbeda. Hasil pada Tabel XII menunjukkan bahwa peringkat Playwright sebagai alternatif terbaik bersifat konsisten pada seluruh kombinasi bobot yang diuji, sehingga kesimpulan penelitian ini dapat dipertanggungjawabkan secara metodologis.

TABEL XII
ANALISIS SENSITIVITAS

Skenario Bobot Runtime (w_1) : CI/CD (w_3)	R_i Selenium	R_i Cypress	R_i Playwright	Peringkat
50:50	0,352	0,054	0,000	Playwright, Cypress, Selenium
60:40 (digunakan)	0,406	0,066	0,000	Playwright, Cypress, Selenium
70:30	0,464	0,076	0,000	Playwright, Cypress, Selenium
40 banding 60	0,305	0,043	0,000	Playwright, Cypress, Selenium

F. Pembahasan

Pemeringkatan yang dihasilkan melalui metode DIA mencerminkan secara langsung kombinasi performa kecepatan eksekusi dan efisiensi integrasi pada masing-masing *framework*. Keunggulan mutlak Playwright (11,396 detik) yang 3,6 kali lebih cepat dibandingkan Selenium sangat dipengaruhi oleh arsitektur internalnya. Berbeda dengan Selenium yang masih mengandalkan protokol komunikasi tradisional dengan membuka dan menutup koneksi pada setiap perintah, Playwright mampu mempertahankan satu koneksi yang terus terbuka. Koneksi persisten tersebut memungkinkan pengiriman perintah secara serentak sehingga respons yang dihasilkan instan [17]. Perbedaan arsitektural tersebut tercermin secara langsung pada selisih *runtime* empiris sebesar 30,137 detik antara Selenium dan Playwright, sebuah selisih yang sejalan dengan karakteristik protokol komunikasi masing-masing *framework*. Evaluasi Tymoshchuk [18] memperkuat posisi tersebut dengan

merekomendasikan Playwright sebagai pilihan utama untuk skenario pengujian yang kompleks dan dinamis. Selain itu, ketersediaan dukungan resmi (*official action*) di GitHub Actions menjadikan proses integrasi Playwright berjalan mulus tanpa hambatan.

Cypress menempati posisi kedua dengan sedikit selisih jarak ideal akibat rata-rata waktu eksekusinya yang berada di bawah Playwright (16,389 detik). Walaupun demikian, Cypress memiliki keunggulan yang setara dengan Playwright dalam hal kemudahan integrasi *plug-and-play* di lingkungan CI/CD. Penelitian independen oleh Rizky et al. [19] juga mencatat bahwa Cypress tetap berdaya saing tinggi karena menawarkan kemudahan penulisan kode (*scripting simplicity*), yang sangat membantu pengembang saat menguji alur aplikasi perdagangan elektronik (*e-commerce*) secara dinamis.

Di sisi lain, Selenium menempati posisi terbawah karena kelemahan ganda pada parameter pengujian. Selain waktu eksekusinya yang paling lambat (41,533 detik) akibat besarnya latensi pada arsitektur *WebDriver*, kerangka kerja ini juga menuntut konfigurasi manual yang rumit di dalam *pipeline* CI/CD. Pengembang diharuskan untuk memasang peramban Chrome, komponen *ChromeDriver*, serta peladen tampilan virtual secara terpisah pada lingkungan Linux, yang pada akhirnya dapat meningkatkan kompleksitas dan potensi kegagalan eksekusi.

Meskipun Playwright unggul dalam kecepatan eksekusi dan kompatibilitas CI/CD, Selenium tetap relevan untuk kasus penggunaan yang menuntut kompatibilitas lintas peramban yang lebih luas, mengingat dukungan *WebDriver* Selenium terhadap peramban lama yang belum sepenuhnya tersedia pada Playwright maupun Cypress. Cypress menawarkan keseimbangan antara kecepatan eksekusi dan kemudahan penulisan skrip, sehingga menjadi pilihan yang sesuai bagi tim dengan sumber daya pengembangan *JavaScript* yang terbatas namun tetap membutuhkan integrasi CI/CD yang mulus. Dengan demikian, pemilihan *framework* yang optimal tetap perlu mempertimbangkan konteks proyek yang spesifik, bukan semata mengikuti hasil pemeringkatan tunggal.

Temuan bahwa Playwright unggul secara konsisten sejalan dengan hasil penelitian sebelumnya yang menggunakan metode *Distance to Ideal Alternative* pada evaluasi *framework* berbasis arsitektur modern. Berbeda dari penelitian tersebut yang mengevaluasi *framework* murni dari sisi kecepatan eksekusi lokal, penelitian ini menunjukkan bahwa keunggulan Playwright semakin diperkuat ketika kriteria kompatibilitas CI/CD turut disertakan, sebuah dimensi yang belum dievaluasi pada penelitian-penelitian sebelumnya.

IV. KESIMPULAN

Berdasarkan hasil analisis *Distance to Ideal Alternative* (DIA) terhadap tiga kerangka kerja otomatisasi antarmuka pengguna pada lingkungan *pipeline* CI/CD, penelitian ini menyimpulkan bahwa Playwright menempati peringkat

preferensi tertinggi dengan nilai jarak ideal $R_i = 0,000$. Posisi tersebut disusul oleh Cypress ($R_i = 0,066$) dan Selenium ($R_i = 0,406$). Perbedaan rata-rata *runtime* yang mendasari peringkat tersebut telah terbukti signifikan secara statistik melalui uji *Kruskal-Wallis* (nilai p di bawah 0,001), dengan uji lanjut *Dunn* yang mengonfirmasi bahwa seluruh pasangan *framework* berbeda secara signifikan, sehingga memperkuat validitas hasil pemeringkatan DIA. Perbedaan peringkat tersebut dipengaruhi oleh dua faktor utama: efisiensi waktu eksekusi Playwright yang 3,6 kali lebih tinggi dibandingkan Selenium, serta ketersediaan dukungan integrasi resmi pada platform GitHub Actions. Hal ini mengindikasikan bahwa pemanfaatan arsitektur berbasis protokol komunikasi dua arah (*WebSocket*) pada Playwright berkontribusi positif terhadap reduksi latensi waktu umpan balik, setidaknya pada konteks pengujian aplikasi *e-commerce* berskala menengah.

Hasil analisis sensitivitas terhadap empat kombinasi bobot kriteria turut menunjukkan bahwa peringkat Playwright sebagai alternatif terbaik bersifat stabil dan tidak dipengaruhi oleh perubahan proporsi bobot, sehingga memperkuat validitas metodologis hasil penelitian ini.

Lebih lanjut, ketiga kerangka kerja menunjukkan tingkat validitas eksekusi sebesar 100% tanpa adanya kegagalan sistem (*crash*), yang menegaskan bahwa ketiganya secara fundamental layak diimplementasikan pada lingkungan produksi. Meskipun demikian, pendekatan arsitektur tradisional pada Selenium tetap menuntut konfigurasi manual untuk penyiapan *driver* dan *display server*, berbeda dengan kepraktisan *plug-and-play* yang ditawarkan Playwright dan Cypress, sehingga rentan terhadap kompleksitas pemeliharaan *pipeline*.

Penelitian ini memiliki beberapa keterbatasan yang perlu diperhatikan dalam menginterpretasikan hasil. Evaluasi hanya mencakup tiga skenario pengujian pada satu aplikasi, sehingga generalisasi hasil terhadap aplikasi berskala lebih besar perlu dilakukan secara hati-hati. Seluruh pengukuran performa dilakukan secara eksklusif pada platform GitHub Actions, sehingga hasil dapat bervariasi pada platform *CI/CD* lain yang memiliki karakteristik sumber daya berbeda. Metode *Distance to Ideal Alternative* yang digunakan juga hanya mempertimbangkan tiga kriteria evaluasi, sedangkan aspek lain seperti konsumsi memori, utilisasi CPU, kemudahan pemeliharaan skrip, dan stabilitas pengujian jangka panjang belum dievaluasi secara kuantitatif dalam penelitian ini.

Berdasarkan temuan tersebut, Playwright direkomendasikan sebagai opsi prioritas bagi tim pengembang yang ingin mengoptimalkan kecepatan *feedback loop* dalam praktik DevOps modern. Mengingat penelitian ini berfokus pada waktu eksekusi aplikasi web berbasis *prototype*, sebagai arah penelitian masa depan, disarankan untuk melakukan evaluasi yang mencakup metrik konsumsi komputasi, seperti utilisasi CPU dan memori selama *pipeline* berjalan. Selain itu, studi lanjutan perlu mengakomodasi pengujian pada aplikasi berskala *enterprise* dengan arsitektur *microservices*, serta membandingkan stabilitas *framework*

tersebut pada berbagai platform *CI/CD* lain seperti GitLab CI atau Jenkins untuk menghasilkan kesimpulan yang lebih komprehensif.

DAFTAR PUSTAKA

- [1] R. S. Pressman, *Software engineering: A practitioner's approach*. 2015.
- [2] S. Di Meglio, L. L. Starace, V. Pontillo, R. Opdebeeck, C. De Roover, and S. Di Martino, "Investigating the adoption and maintenance of web GUI testing: Insights from GitHub repositories," *Inf. Softw. Technol.*, vol. 189, p. 107928, Jan. 2026, doi: 10.1016/J.INFSOF.2025.107928.
- [3] G. Hyun, J. Oak, D. Kim, and K. Kim, "The impact of an automation system built with Jenkins on the efficiency of container-based system deployment," *Sensors* 2024, Vol. 24, Page 6002, vol. 24, no. 18, p. 6002, Sep. 2024, doi: 10.3390/S24186002.
- [4] F. Dobslaw, R. Feldt, D. Michaelsson, P. Haar, F. G. De Oliveira Neto, and R. Torkar, "Estimating return on investment for GUI test automation frameworks," *Proceedings - International Symposium on Software Reliability Engineering, ISSRE*, vol. 2019-October, pp. 271-282, Oct. 2019, doi: 10.1109/ISSRE.2019.00035.
- [5] A. J. Abdulwareth and A. A. Al-Shargabi, "Toward a multi-criteria framework for selecting software testing tools," *IEEE Access*, vol. 9, pp. 158872-158891, 2021, doi: 10.1109/ACCESS.2021.3128071.
- [6] V. Bhimanapati, P. Goel, and U. Jain, "Leveraging Selenium and Cypress for comprehensive web application testing," *Journal of Quantum Science and Technology*, vol. 1, no. 1, pp. 66-79, Mar. 2024, doi: 10.36676/JQST.V1.I1.10.
- [7] B. García, J. M. del Alamo, M. Leotta, and F. Ricca, "Exploring browser automation: A comparative study of Selenium, Cypress, Puppeteer, and Playwright," *Communications in Computer and Information Science*, vol. 2178 CCIS, pp. 142-149, 2024, doi: 10.1007/978-3-031-70245-7_10.
- [8] A. Polkhovskaya and J. Salonen, "Improving QA processes and implementing Playwright for API and E2E testing automation," 2025, Accessed: May 17, 2025. [Online]. Available: <http://www.theseus.fi/handle/10024/882858>
- [9] G. A. Prasetyo, "Analisis perbandingan UI test automation framework dengan metode The Distance to the Ideal Alternative," 2024, Accessed: May 17, 2025. [Online]. Available: <https://dspace.uii.ac.id/handle/123456789/51452>
- [10] Y. Wang, M. V. Mäntylä, Z. Liu, and J. Markkula, "Test automation maturity improves product quality—Quantitative study of open source projects using continuous integration," *Journal of Systems and Software*, vol. 188, p. 111259, Jun. 2022, doi: 10.1016/J.JSS.2022.111259.
- [11] A. P. Putra, D. Sartika, and D. Lianda, "Decision support system for the best junior high school using the Distance to the Ideal Alternative (DIA) method at the Education and Culture Office of the City of Bengkulu," *Jurnal Media Computer Science*, vol. 3, no. 2, Jul. 2024, doi: 10.37676/JMCS.V3I2.6618.
- [12] S. Adiatma and A. Darmayantie, "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," *Widya Teknik*, vol. 22, no. 2, pp. 36-41, May 2023, doi: 10.33508/WT.V22I1.5027.
- [13] R. Fakhri and N. Setiani, "Analisis perbandingan unit test automation framework dengan metode The Distance to the Ideal Alternatif," *EDUSAINTEK: Jurnal Pendidikan, Sains dan Teknologi*, vol. 12, no. 1, pp. 253-272, 2025, doi: 10.47668/EDUSAINTEK.V12I1.1489.
- [14] G. A. Prasetyo and N. Setiani, "Perbandingan kinerja framework automation UI testing menggunakan The Distance to the Ideal Alternative," *JUPI (Jurnal Ilmiah Penelitian dan Pembelajaran Informatika)*, vol. 10, no. 1, pp. 224-237, Jan. 2025, doi: 10.29100/JUPI.V10I1.5760.

- [15] F. P. Apriliyanto, Y. Prihati, and C. A. Bakti, "Implementasi CI/CD menggunakan Github Actions untuk pengembangan web landing page UMKM XYZ," *INTECOMS: Journal of Information Technology and Computer Science*, vol. 7, no. 4, pp. 1239–1249, Jul. 2024, doi: 10.31539/INTECOMS.V7I4.10961.
- [16] M. Moń and B. Pańczyk, "A comparative analysis of web application test automation tools," *Journal of Computer Sciences Institute*, vol. 35, pp. 159–165, Jun. 2025, doi: 10.35784/JCSI.7119.
- [17] B. Bylina and A. Antończak, "Analysis of end-to-end test automation tools based on the examples of Selenium WebDriver and Playwright," *Position Papers of the 19th Conference on Computer Science and Intelligence Systems*, vol. 40, pp. 1–8, Nov. 2024, doi: 10.15439/2024F3747.
- [18] A. Tymoshchuk, "The evolution of test automation: From Selenium to Playwright. A comparison of automation tools: Selenium vs. Playwright vs. Cypress," *International Journal of Computer (IJC)*, vol. 54, no. 1, pp. 37–45, Apr. 2025, Accessed: Jan. 16, 2026. [Online]. Available: <https://ijcjournal.org/InternationalJournalOfComputer/article/view/2355>
- [19] M. A. Rizky and A. Wibowo, "Analysis of automation testing on e-commerce websites and user responses," *Society*, vol. 13, no. 1, pp. 788–802, Apr. 2025, doi: 10.33019/society.v13i1.767.