Perbandingan Pengujian Manual dan Otomatis *Playwright* dengan dan tanpa *Page Object Model*

Ramadhani Rizky Syahputra[1,*], Tito Pinandita[2], Dimara Kusuma Hakim[3], Maulida Ayu Fitriani[4]

[1,2,3,4] Fakultas Teknik dan Sains, Teknik Informatika, Universitas Muhammadiyah Purwokerto, Banyumas, Indonesia
Email: [1,*]r.rizky.sy@gmail.com, [2]titop@ump.ac.id, [3]dimarakusumahakim@gmail.com, [4]maulidaayuf@ump.ac.id
[*)] Email Penulis Utama

Abstrak−Pengujian perangkat lunak adalah tahapan kritis dalam siklus pengembangan sistem informasi. *Website* Skripsweet platform untuk manajemen skripsi mahasiswa saat ini masih mengandalkan pengujian *manual* yang terbukti tidak efisien dan rentan terhadap *human error*. Penelitian ini membandingkan tiga skenario pengujian: (1) pengujian *manual*, (2) *automation testing* tanpa *Page Object Model* (POM), dan (3) *automation testing* dengan POM menggunakan *framework* Playwright. Total 26 *test case* dikembangkan mencakup modul *landing page*, autentikasi, *forgot password*, *onboarding*, *dashboard*, *pricing* & *checkout*, *settings*, dan *AI checker*, setiap *test case* dieksekusi sebanyak lima kali untuk memperoleh rata-rata waktu eksekusi yang representatif. Metrik yang digunakan meliputi e*xecution time*, *lines of code* (LOC), *cyclomatic complexity,* dan *code reusability*. Hasil penelitian menunjukkan bahwa *automation testing* mencapai efisiensi waktu 91,0% (tanpa POM) dan 83,8% (dengan POM) dibandingkan pengujian *manual*. Menariknya, *automation* tanpa POM menunjukkan *throughput* eksekusi mentah yang lebih cepat (*mean* 49,82 detik) dibandingkan *automation* dengan POM (*mean* 89,75 detik) karena lapisan abstraksi *method call* tambahan, namun *automation* dengan POM menunjukkan konsistensi yang jauh lebih tinggi (SD = 1,12 detik) dibandingkan *automation* tanpa POM (SD = 2,77 detik) maupun pengujian *manual* (SD = 13,11 detik). Penerapan POM menurunkan LOC file spesifikasi pengujian sebesar 31,2% (dari 461 menjadi 317 baris), mempertahankan *cyclomatic complexity* rata-rata dari 1,37 menjadi 1,22, dan mencapai *reusability* dengan 9 *page object class* yang diinstansiasi di seluruh *test suite*, dengan *BasePage* diwarisi oleh seluruh 9 *subclass*. Kontribusi utama penelitian ini adalah kerangka evaluasi kuantitatif yang membandingkan tiga skenario pengujian secara bersamaan pada empat metrik, memberikan bukti nyata bagi praktisi pada praktik pengujian platform manajemen skripsi.

Kata Kunci: Pengujian otomatisasi, Playwright, *Page Object Model*, Kemudahan pemeliharaan, Penjaminan mutu perangkat lunak

Abstract−*Software testing is a critical stage in the information systems development cycle. The Skripsweet website platform for student thesis management currently relies on manual testing, which has proven to be inefficient and prone to human error. This study compares three testing scenarios: (1) manual testing, (2) automation testing without a Page Object Model (POM), and (3) automation testing with a POM using the Playwright framework. A total of 26 test cases were developed covering the landing page, authentication, forgot password, onboarding, dashboard, pricing & checkout, settings, and AI checker modules, each test case was executed five times to enable descriptive statistical analysis (mean and standard deviation). The metrics used included execution time, lines of code (LOC), cyclomatic complexity, and code reusability. The results show that automation testing achieves 91.0% (without POM) and 83.8% (with POM) time efficiency compared to manual testing. Interestingly, automation without POM exhibited faster raw execution (mean 49.82 seconds) than automation with POM (mean 89.75 seconds) due to the additional method call abstraction layer, yet automation with POM demonstrated substantially higher consistency (SD = 1.12 seconds) compared to both automation without POM (SD = 2.77 seconds) and manual testing (SD = 13.11 seconds). The implementation of POM decreased LOC in test specification files by 31,2% (from 461 to 317 lines), maintained the average cyclomatic complexity from 1.37 to 1.22, and achieved class reusability with 9 Page Object Classes instantiated across the test suite, with BasePage inherited by all 9 subclasses. The main contribution of this study is a quantitative evaluation framework comparing three testing scenarios across four metrics simultaneously, providing clear evidence for practitioners in student thesis management platform testing contexts.*

Keywords: *Automation testing, Playwright, Page Object Model, Maintainability, Software quality assurance*

1. PENDAHULUAN

Pengujian perangkat lunak (*software testing*) adalah komponen esensial dalam siklus *Software Development Life Cycle* (SDLC). Kegagalan dalam tahapan pengujian dapat mengakibatkan munculnya *defect* pada lingkungan produksi yang berdampak langsung pada kualitas layanan kepada pengguna akhir [1]. Studi studi di industri menunjukkan bahwa menyeimbangkan pengujian *manual* dan otomatis masih menjadi tantangan nyata bagi tim pengembang di berbagai bidang [2]. Dalam konteks *website* Skripsweet platform yang telah digunakan secara aktif oleh mahasiswa untuk mengelola skripsi, kesalahan fungsional pada modul modul inti seperti autentikasi atau proses *checkout* berpotensi langsung mengganggu proses pengerjaan pengguna yang sedang berjalan,

menjadikan keandalan sistem sebagai kebutuhan yang tidak dapat ditunda.

Pengujian *manual*, meskipun telah lama menjadi praktik standar, menghadapi keterbatasan mendasar dalam menghadapi kompleksitas aplikasi modern. Pengujian *manual* tidak dapat dieksekusi secara berulang dengan presisi yang konsisten sehingga rentan terhadap *human error* [2]. Skalabilitas pengujian *manual* juga sangat terbatas, bertambahnya jumlah fitur secara langsung meningkatkan beban kerja penguji secara proporsional [3]. Di samping itu, proses pengujian *manual* sulit disatukan ke dalam alur kerja pengembangan modern karena lambat dan masih sangat bergantung pada manusia [4]. Kompleksitas aplikasi web yang terus meningkat semakin mempersulit pengujian *manual* untuk mempertahankan cakupan yang memadai [5]. Keterbatasan ini mendorong adopsi *automation testing* sebagai solusi yang lebih *scalable* dan konsisten.

Di antara berbagai *framework automation testing* berbasis *web*, pemilihan *framework* yang tepat menjadi faktor penting dalam keberhasilan implementasi *automation testing* [6]. Playwright yang dikembangkan oleh Microsoft telah mulai diadopsi dalam praktik pengujian perangkat lunak di Indonesia, dengan studi kasus industri menunjukkan kemudahan penggunaannya khususnya bagi penguji yang baru mengenal *automation testing* [7]. Playwright menawarkan dukungan lintas browser (Chromium, Firefox, dan WebKit), mekanisme *auto waiting* yang meminimasi *flaky test*, serta kemampuan eksekusi paralel yang meningkatkan *throughput* pengujian [8]. Beberapa laporan industri mencatat peningkatan minat penggunaan Playwright di kalangan praktisi pengujian dalam beberapa tahun terakhir, menjadikannya salah satu *framework* dengan pertumbuhan adopsi tertinggi [9].

Namun, efektivitas *automation testing* tidak semata bergantung pada pemilihan *framework*, melainkan juga pada arsitektur dan pola desain kode uji. *Page Object Model* (POM) merupakan *design pattern* yang merepresentasikan antarmuka halaman *web* ke dalam kelas kelas terpisah, memisahkan logika uji dari detail implementasi UI [10]. Penerapan POM menghasilkan kode yang lebih modular, mudah dipelihara, dan dapat digunakan kembali atribut yang sangat bernilai dalam lingkungan pengembangan *agile* di mana antarmuka pengguna berubah secara reguler [5].

Penelitian terdahulu telah membandingkan *automation testing* dengan pengujian *manual* [2],[3], atau mengevaluasi efektivitas *design pattern* POM pada *test suite* berbasis web [10], namun masing masing dilakukan secara terpisah dengan metrik yang berbeda beda [11]. Namun, sepengetahuan penulis, belum terdapat studi yang secara menyeluruh membandingkan tiga skenario sekaligus *manual testing*, *automation* tanpa POM, dan automation dengan POM dengan menggunakan serangkaian metrik kuantitatif yang terstandar pada domain sistem *website* Skripsweet, *research gap* inilah yang menjadi motivasi utama penelitian ini.

Berdasarkan latar belakang tersebut, penelitian ini bertujuan untuk menjawab dua research questions, yaitu (1) tingkat efektivitas *automation testing* menggunakan Playwright dalam meningkatkan efisiensi waktu pengujian pada *website* Skripsweet dibandingkan pengujian manual, serta (2) pengaruh pendekatan POM terhadap *maintainability* dan struktur kode dalam *automation testing*.

1.1 *Automation Testing* pada Sistem Informasi

Balsam dan Mishra [5] mengidentifikasi bahwa pengujian aplikasi *web* menghadapi tantangan struktural yang unik, seperti antarmuka yang kompleks, banyak jalur interaksi, dan ketergantungan tinggi terhadap JavaScript dinamis. Tantangan-tantangan ini mendorong adopsi *automation testing* berbasis *framework* seperti Playwright yang mampu menangani kompleksitas tersebut secara terprogram, sekaligus menekankan pentingnya *design pattern* seperti *POM* untuk mengelola kompleksitas kode uji.

Haas et al. [2] mengkaji proses pengujian otomatis dan *manual* pada lima perusahaan industri dari domain yang berbeda, menemukan bahwa tantangan terbesar dalam optimasi pengujian otomatis dan *manual* meliputi *test flakiness*, biaya pemeliharaan tinggi, dan kesulitan mengintegrasikan pengujian ke dalam alur pengembangan. Teknik optimasi seperti *test impact analysis* terbukti dapat mengurangi waktu eksekusi pengujian secara signifikan. Studi ini juga mencatat bahwa penggunaan *design pattern* yang tepat seperti POM penting untuk mendukung penerapan *automation testing* dalam jangka panjang, karena minimnya standarisasi arsitektur kode uji menjadi penyebab utama rendahnya ROI *automation testing* di banyak organisasi. Sejalan dengan itu, Wang et al. [12] mengidentifikasi 26 *best practice test* automation dari 81 studi, di antaranya pengembangan *test case* berkualitas tinggi dan pemisahan tanggung jawab (*separation of concern*) antara logika uji dan detail UI yang menjadi prinsip dasar POM.

1.2 Perbandingan Framework: Playwright dan Selenium

Playwright dirancang dengan arsitektur protokol CDP (Chrome DevTools Protocol) yang memungkinkan komunikasi langsung antara test runner dan browser engine, berbeda dari Selenium yang menggunakan HTTP WebDriver API [8]. Arsitektur ini mendasari mekanisme *auto waiting* bawaan Playwright, yang secara otomatis menunggu kondisi elemen siap sebelum eksekusi aksi, mengeliminasi kebutuhan *explicit sleep*() yang menjadi sumber utama *test flakiness*. Survei SmartBear [9] mencatat peningkatan minat penggunaan Playwright di

kalangan developer, dengan stabilitas eksekusi dan dukungan *multi browser* sebagai keunggulan utama dibandingkan Selenium.

Karagöz et al. [13] menunjukkan bahwa *Page Object* masih menjadi pola desain yang paling banyak diadopsi untuk menjaga *maintainability* dan skalabilitas pengujian *end to end*, namun proses pembuatan dan pemeliharaannya masih sebagian besar dilakukan secara *manual* dan memakan banyak *effort*, sementara solusi otomatisasi yang ada masih terbatas penerapannya secara praktis mendorong eksplorasi pendekatan baru seperti automation berbasis *Large Language Model* untuk menghasilkan *Page Object*.

Ricca dan Stocco [14] dalam survei literatur abu abu tentang praktik web *test automation* mengidentifikasi bahwa mayoritas praktisi memprioritaskan *maintainability* dan *reusability* sebagai kriteria utama keberhasilan *test automation*. Mereka juga menemukan bahwa POM merupakan *design pattern* yang paling banyak direkomendasikan oleh praktisi. Temuan ini memperkuat relevansi penelitian ini yang mengukur *maintainability* secara kuantitatif melalui metrik LOC, *CC*, dan *reusability*.

1.3 *Page Object Model*: Desain dan Dampak

Leotta et al. [10] melakukan studi empiris komparatif tentang tiga pendekatan *web test automation NLP based*, programmable, dan *capture* & *replay* menemukan bahwa pendekatan berbasis kode terprogram (*programable*), yang merupakan dasar POM, memberikan *maintainability* tertinggi dan tingkat kegagalan terendah akibat perubahan UI. POM diidentifikasi sebagai pola desain yang paling banyak direkomendasikan karena kemampuannya memisahkan logika pengujian dari detail implementasi UI, sehingga perubahan UI hanya perlu diperbaiki di satu tempat (*Page Object class*) tanpa menyentuh file spesifikasi pengujian. Studi ini menjadi landasan teoritis bagi implementasi POM dalam penelitian kami, namun belum mengukur dampak POM secara simultan terhadap efisiensi waktu eksekusi dan *maintainability* dalam satu kerangka eksperimen celah yang menjadi salah satu kontribusi penelitian ini.

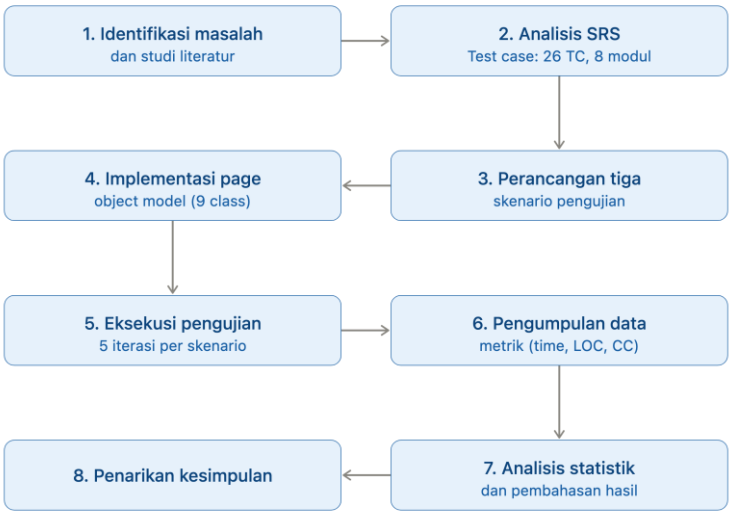
Wohlin [15] menegaskan bahwa studi kasus (*case study*) adalah metode penelitian yang sesuai untuk mengevaluasi fenomena rekayasa perangkat lunak dalam konteks nyata, khususnya ketika fenomena tersebut sulit dipisahkan dari konteksnya seperti perbandingan pendekatan pengujian pada satu sistem informasi yang aktif dikembangkan. Karakteristik *within subject comparative study* yang diadopsi pada penelitian ini sejalan dengan prinsip tersebut, di mana ketiga skenario pengujian dievaluasi pada objek dan kondisi sistem yang identik untuk menjaga validitas internal.

Temuan-temuan di atas sangat relevan bagi *website* Skripsweet yang masih aktif dikembangkan, mengingat sistem ini memiliki delapan modul fungsional yang berpotensi mengalami perubahan UI secara reguler. Berdasarkan sintesis literatur di atas dan peta praktik terbaik *automation testing* berbasis web [14], yang diperkuat oleh tinjauan sistematis tentang kematangan test automation [12], teridentifikasi tiga *research gap* : (1) studi yang ada membandingkan hanya dua skenario sekaligus (*manual* vs. *automation*, atau POM vs. non POM) [16],[17], sedangkan perbandingan tiga skenario secara bersamaan pada satu platform belum dilakukan, (2) metrik yang digunakan umumnya terbatas pada waktu atau LOC saja, tanpa analisis statistik deskriptif (*mean*, SD) per iterasi yang memungkinkan penilaian konsistensi, serta (3) domain sistem manajemen skripsi berbasis web belum pernah menjadi objek studi empiris pada ranah *automation testing* berbasis Playwright. Penelitian ini hadir untuk mengisi ketiga gap tersebut secara bersamaan, sebagai novelty utama penelitian ini.

2. METODE PENELITIAN

2.1 Desain Penelitian

Penelitian ini menggunakan pendekatan kuantitatif eksperimental dengan desain *within subject comparative study*. Tiga kondisi pengujian dibandingkan secara langsung pada objek yang sama: (1) *manual testing* sebagai baseline, (2) *automation testing* tanpa POM, dan (3) *automation testing* dengan POM menggunakan *Playwright*. Setiap *test case* dieksekusi sebanyak lima kali pada masing-masing kondisi oleh peneliti sebagai eksekutor tunggal, untuk menghasilkan data yang cukup bagi analisis statistik deskriptif (*mean* dan *standard deviation*). Pendekatan ini memastikan bahwa variasi hasil yang diperoleh mencerminkan perbedaan metode pengujian, bukan variasi antar objek. Desain *within subject* dipilih karena ketiga skenario diuji pada objek dan kondisi sistem yang sama persis, sehingga faktor luar yang berpotensi memengaruhi hasil (variabel pengganggu) dapat dikendalikan lebih ketat dibandingkan jika pengujian dilakukan pada objek yang berbeda beda [18]. Pendekatan ini mengurangi ancaman terhadap validitas internal akibat perbedaan karakteristik aplikasi antar kondisi.
Tahapan pelaksanaan penelitian secara keseluruhan, mulai dari identifikasi masalah hingga penarikan kesimpulan, disajikan pada Gambar 1.



Gambar 1. Diagram Alur Penelitian

2.2 Objek Penelitian

Objek penelitian adalah *website* Skripsweet, yang dikembangkan menggunakan Supabase (*backend*) dan React (*frontend*), dijalankan secara lokal (*localhost*). Sistem ini mencakup delapan modul fungsional yang menjadi cakupan pengujian: *landing page*, *autentikasi*, *forgot password*, *onboarding*, *dashboard*, *pricing* & *checkout*, *settings*, dan *ai checker*. Arsitektur Supabase sebagai *backend as a service* (*BaaS*) dikombinasikan dengan React sebagai *frontend*, memberikan karakteristik pengujian yang representatif untuk sistem berbasis web modern [5].

2.3 Lingkungan dan Tools Pengujian

Spesifikasi lengkap lingkungan pengujian disajikan pada Tabel 1.

Tabel 1. Spesifikasi Lingkungan Pengujian

Komponen	Spesifikasi
Sistem Operasi	macOS
Prosesor	Apple M4
RAM	16 GB
Framework Uji	Playwright dan TypeScript
Browser Target	Chromium (*default*), Firefox, WebKit
Node.js	v20.12.0
IDE	Visual Studio Code dan Playwright *Extension* (macOS)
Metrik LOC & CC	ESLint v8.57 dengan *plugin complexity*, cloc v1.98
Server Aplikasi	*Localhost (*macOS*)*

2.4 Implementasi Page Object Model

Dalam skenario ketiga (*automation* dengan POM), setiap halaman *web* direpresentasikan oleh sebuah kelas TypeScript terpisah yang disimpan dalam direktori /*pages*/, mengikuti struktur dasar *page object pattern* [10]. Setiap *page object class* menyimpan dua elemen: (1) *locator* elements yang mendefinisikan selektor CSS/XPath untuk elemen UI, dan (2) *action methods* yang merepresentasikan interaksi pengguna [10]. Arsitektur direktori proyek disusun sebagai berikut: /*pages*/ untuk *page object classes*, /*tests*/ untuk file spesifikasi pengujian, dan /*fixtures*/ untuk data uji. Sebanyak sembilan *page object classes* dikembangkan, satu untuk setiap modul dalam cakupan 26 *test case*: *AuthPage*, *LandingPage*, *DashboardPage*, *OnboardinPage*, *PricingPage*, *CheckoutPage*, *SettingsPage*, *AiCheckerPage*, dan *ForgotPasswordPage*, dengan *BasePage* sebagai kelas induk yang diwarisi oleh seluruh 9 kelas tersebut (100%) [10]. Pendekatan berbasis hierarki *page object* ini didukung oleh riset yang

menunjukkan peningkatan *maintainability* secara signifikan pada *test suite* berbasis web [10]. Modularitas hierarki pewarisan kelas ini mengikuti prinsip DRY (*don't repeat yourself*) yang direkomendasikan dalam rekayasa kode pengujian yang berkelanjutan [10]. Pemisahan *locator* dari logika uji terbukti mengurangi duplikasi kode dan meningkatkan keterbacaan skrip pengujian secara keseluruhan [10].

2.5 Skenario dan *Test Case* Pengujian

Total 26 *test case* dikembangkan berdasarkan dokumen *Software Requirements Specification* (SRS) *website* Skripsweet. Pembuatan *test case* dari SRS adalah praktik standar dalam pengujian berbasis spesifikasi (*specification based testing*) untuk memastikan cakupan fungsional yang sistematis, sesuai dengan definisi dan panduan yang tercantum dalam standar internasional pengujian perangkat lunak [11]. Teknik *equivalence partitioning* diterapkan dalam menentukan representasi kelas nilai input pada setiap *test case* [11]. *Equivalence partitioning* adalah teknik pengujian *black box* yang membagi domain input menjadi beberapa "kelas ekuivalen" kelompok nilai input yang diasumsikan akan diproses sistem dengan cara yang sama, sehingga cukup diuji satu nilai representatif per kelas tanpa perlu menguji seluruh kemungkinan nilai. Distribusi *test case* disajikan pada Tabel 2, sedangkan deskripsi lengkap setiap *test case* disajikan pada Tabel 3.

Tabel 2. Spesifikasi Lingkungan Pengujian

Modul	Jumlah *Test Case* (TC)	Prioritas	Tipe Pengujian
Landing Page	3	Tinggi	*UI, Navigation, Responsive*
Autentikasi (*Sign Up* / *Sign In* / *Sign Out*)	7	Tinggi	*Functional, Negative*
Forgot Password	2	Sedang	*Functional, Validation*
Onboarding	2	Tinggi	*Functional*
Dashboard	2	Tinggi	*Functional, Navigation*
Pricing & Checkout	5	Tinggi	*Functional, Integration*
Settings	4	Sedang	*Functional, UI*
AI Checker	1	Tinggi	*Functional, Access Control*
Total	26	-	-

Tabel 3. Deskripsi Lengkap *Test Case* dan Waktu Eksekusi Rata rata (5 Iterasi)

TC-ID	Modul	Deskripsi Skenario	Data Pengujian	*Manual* (detik)	Auto tanpa POM (detik)	Auto dengan POM (detik)
TC-01	*Landing Page*	Hero, CTA, dan link nav utama tampil	cek tampilan judul utama, tombol, dan menu navigasi muncul	6,7	0,874	2,460
TC-02	*Landing Page*	CTA *primary* mengarah ke /*auth* atau /*onboarding*	Tekan tombol utama di halaman depan (misalnya "Mulai Sekarang" / "Daftar")	7,9	0,823	2,740
TC-03	*Landing Page*	Tampilan responsif di viewport mobile (390px)	Dibuka lewat layar ukuran HP (390 x 844 piksel)	11,9	0,705	2,480
TC-04	Autentikasi	*Sign up* sukses dengan *password* kuat	Nama: Mahasiswa Baru / Email: (dibuat otomatis,	25,1	11,580	13,060

			beda tiap kali dicoba) / Password: (password yang sudah ditentukan kuat)			
TC-05	Autentikasi	*Reject sign up* dengan *password* lemah	Nama: Tester / Email: (dibuat otomatis) / Password: 123 (sengaja lemah)	24,8	11,460	13,120
TC-06	Autentikasi	*Sign in* sukses *redirect* ke *dashboard*	Email: rikiriou44@gmail.com / Password: riki1234	34,9	2,160	3,500
TC-ID	**Modul**	**Deskripsi Skenario**	**Data Pengujian**	***Manual* (detik)**	**Auto tanpa POM (detik)**	**Auto dengan POM (detik)**
TC-07	Autentikasi	Link “Lupa *password*” mengarah ke */forgot-password*	Tekan *link* "Lupa *password*"	8,5	1,660	3,100
TC-08	Autentikasi	*Sign in* dengan kredensial salah menampilkan *error*	Email: wrong@example.com / Password: testing2password3abc	18,0	2,000	3,920
TC-09	Autentikasi	*Sign out* menghapus *session* dan *redirect* /auth	Tekan tombol "Keluar"	17,8	0,815	2,420
TC-10	Autentikasi	*Session* tidak aktif *redirect* ke /auth	Buka halaman *Dashboard* tanpa login dulu	13,7	0,857	0,711
TC-11	*Forgot Password*	Kirim reset link sukses pesan konfirmasi muncul	Email: user@example.com	22,1	0,897	2,720
TC-12	*Forgot Password*	Email kosong tombol *disable* atau *error* muncul	Kolom email dibiarkan kosong, langsung tekan kirim.	9,6	0,774	2,460
TC-13	*Onboarding*	Alur topik baru sampai selesai *redirect* ke *dashboard*	Isi minat, jurusan, dan lama.	71,0	0,762	3,560
TC-14	*Onboarding*	Tombol Lanjut *disabled* bila step belum diisi	Semua kolom dibiarkan kosong (tidak diisi)	37,8	0,574	2,520
TC-15	*Dashboard*	*Render welcome banner & quick actions*	cek tampilan: ucapan selamat datang & *menu* cepat muncul	8,2	2,100	3,920
TC-16	*Dashboard*	*Quick action Roadmap*	Tekan menu "*Roadmap*"	7,3	1,171	4,000

		navigasi ke /*roadmap*				
TC-17	*Pricing*	Menampilkan kartu *Basic* dan *Plus*	cek tampilan: dua paket harga, *Basic* dan Plus, muncul	15,5	1,700	2,700
TC-18	*Pricing*	CTA Plus mengarah ke /*checkout*	Tekan tombol pilih paket Plus	22,6	1,560	3,200
TC-19	*Checkout*	*Render summary* plan dan tombol Bayar	Buka halaman *checkout* untuk paket Plus	24,9	1,181	1,200
TC-ID	**Modul**	**Deskripsi Skenario**	**Data Pengujian**	***Manual* (detik)**	**Auto tanpa POM (detik)**	**Auto dengan POM (detik)**
TC-20	*Checkout*	Klik Bayar memanggil API *invoice*	Tekan tombol "Bayar"	26,5	1,316	1,250
TC-21	*Checkout*	*Apply* promo *code* menampilkan *toast* konfirmasi	Masukkan kode promo: PROMO50	27,7	0,534	0,453
TC-22	*Settings*	*Update* nama lengkap tersimpan ke server	Ganti nama jadi: Nama Updated	18,6	0,944	3,740
TC-23	*Settings*	*Toggle dark mode* mengubah *class* HTML	Tekan tombol ganti tampilan gelap/terang	6,0	0,762	2,940
TC-24	*Settings*	*Export data*	Tekan tombol "Ekspor Data	36,4	0,750	2,580
TC-25	*Settings*	*Delete account* membuka dialog konfirmasi	Tekan tombol "Hapus Akun"	26,6	0,728	2,640
TC-26	*AI Checker*	*Hard-cap* input di 24.000 karakter (*maxlength*)	Ketik teks sepanjang 25.000 huruf (lebih dari batas maksimal 24.000)	25,8	1,136	2,360
-	-	Total / Mean	-	555,6 / 21,37	49,8 / 1,92	89,8 / 3,45

2.6 Metrik Pengukuran

Empat metrik kuantitatif digunakan dalam penelitian ini:
(1) *Execution Time*: Total waktu eksekusi seluruh *test suite* dalam detik. Metrik ini menjadi indikator efisiensi yang paling langsung terukur dalam studi komparatif pengujian perangkat lunak [3]. Seluruh pengujian dieksekusi secara satu per satu pada ketiga skenario untuk memastikan keadilan perbandingan, fitur eksekusi paralel *Playwright* tidak diaktifkan pada fase pengukuran ini.Efisiensi dihitung menggunakan rumus (1):

$$Efficiency(\%) = \frac{T_{manual} - T_{automation}}{T_{manual}} \times 100\% \quad (1)$$

(2) *Lines of Code* (LOC): Dihitung menggunakan *tool* cloc v1.98, mencakup seluruh *file* TypeScript dalam direktori /*tests*/ dan /*pages*/. LOC digunakan sebagai proksi awal untuk mengukur ukuran dan pertumbuhan kode

pengujian [10]. Analisis LOC dalam konteks *maintainability* dilengkapi dengan metrik CC karena LOC saja tidak mencerminkan kompleksitas struktural kode [3].
(3) *Cyclomatic Complexity* (CC): Dihitung menggunakan *rule complexity* bawaan ESLint v8.57 (parser @typescript-eslint) pada *level* fungsi, dengan threshold 0 agar seluruh fungsi termasuk yang berkompleksitas 1 turut tercatat dalam perhitungan rata-rata. Nilai *cyclomatic complexity* dihitung menggunakan rumus (2):

$$CC = E - N + 2P. \quad (2)$$

dengan:
E = jumlah *edge* pada *Control Flow Graph* (CFG),
N = jumlah node pada CFG,
P = jumlah komponen terhubung (*connected components*).
Metrik CC digunakan secara luas dalam studi empiris pengujian perangkat lunak sebagai indikator kompleksitas struktural kode [10]. CC adalah metrik struktural yang berkorelasi dengan tingkat kesalahan dan kemudahan pemeliharaan kode [5], sehingga relevan untuk membandingkan kompleksitas antara implementasi POM dan non POM.
(4) *Code Reusability*: Dihitung sebagai total jumlah instansiasi *page object class* di seluruh *test file*, mencerminkan berapa kali kelas yang sama digunakan kembali tanpa duplikasi kode. *Reusability* menjadi indikator kunci keberhasilan penerapan *design pattern* modular seperti POM dalam *test suite* berbasis objek [10].
Analisis statistik deskriptif (*mean* dan *standard deviation*) dihitung berdasarkan data dari lima iterasi eksekusi. Nilai *standard deviation* populasi dihitung menggunakan rumus (3):

$$SD = \sqrt{\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{n}} \quad (3)$$

dengan:
x_i = nilai pada iterasi ke-i,
μ = nilai rata-rata (*mean*),
$n = 5$ = jumlah iterasi.
Pendekatan statistik deskriptif ini mengacu pada praktik yang direkomendasikan dalam desain eksperimen perangkat lunak untuk studi komparatif berskala kecil [18].

3. HASIL DAN PEMBAHASAN

3.1 Analisis Statistik Waktu Eksekusi

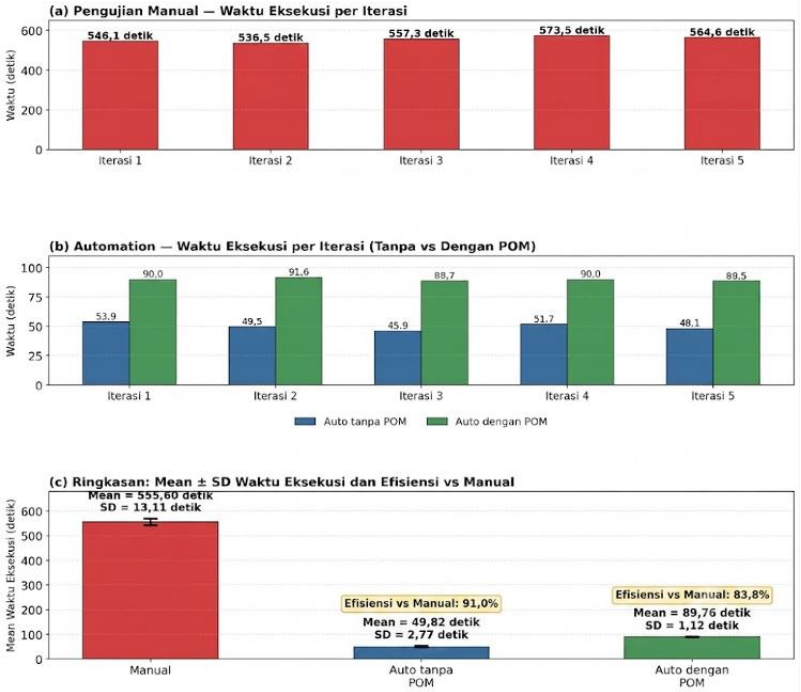
Data waktu eksekusi dari 5 iterasi pengujian untuk seluruh *test suite* (26 TC) disajikan pada Gambar 2. Analisis statistik deskriptif mencakup *mean* (μ) dan *standard deviation* (SD) untuk setiap skenario.

Berdasarkan Ganbar 1, nilai *mean* waktu eksekusi pengujian *manual* sebesar 555,6 detik, automation tanpa POM sebesar 49,82 detik, dan *automation* dengan POM sebesar 89,75 detik. Dengan menggunakan rumus efisiensi yang telah ditetapkan, *automation* tanpa POM mencapai efisiensi 91,0% dibandingkan *manual*, sedangkan *automation* dengan POM mencapai efisiensi 83,8% dibandingkan *manual*. Namun, *automation* tanpa POM justru menunjukkan *throughput* eksekusi mentah yang lebih cepat dibandingkan *automation* dengan POM selisih 39,93 detik atau *automation* dengan POM 80,1% lebih lambat secara *raw execution time*. Temuan ini akan dibahas lebih lanjut pada sub bab 3.5.

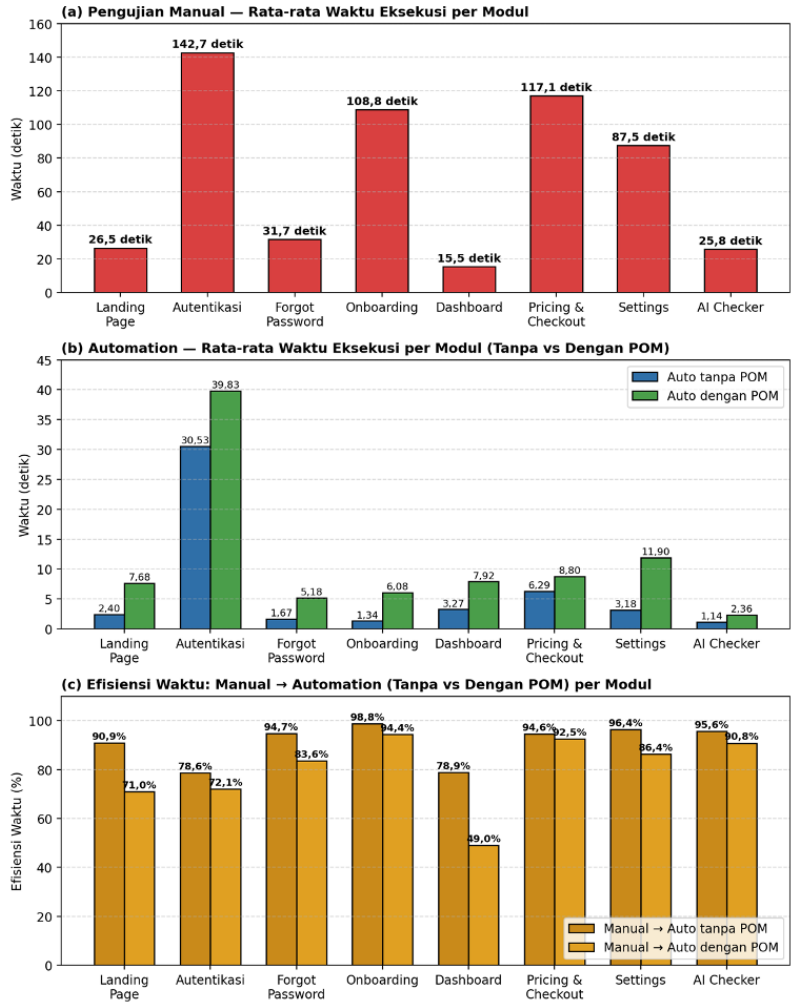
Nilai *standard deviation* memberikan gambaran konsistensi pengujian. Pengujian *manual* memiliki SD tertinggi (σ = 13,11 detik), mencerminkan variabilitas yang tinggi akibat faktor manusiawi. *Automation* tanpa POM memiliki SD (σ = 2,77 detik), sedangkan *automation* dengan POM menunjukkan SD terendah (σ = 1,12 detik) meskipun *mean* waktu eksekusinya lebih tinggi, *automation* dengan POM justru paling konsisten antar iterasi.

3.2 Perbandingan Tiga Skenario per Kelompok Modul

Gambar 3 menyajikan rata-rata waktu eksekusi berdasarkan pengelompokan modul, disertai nilai efisiensi komparatif antar skenario



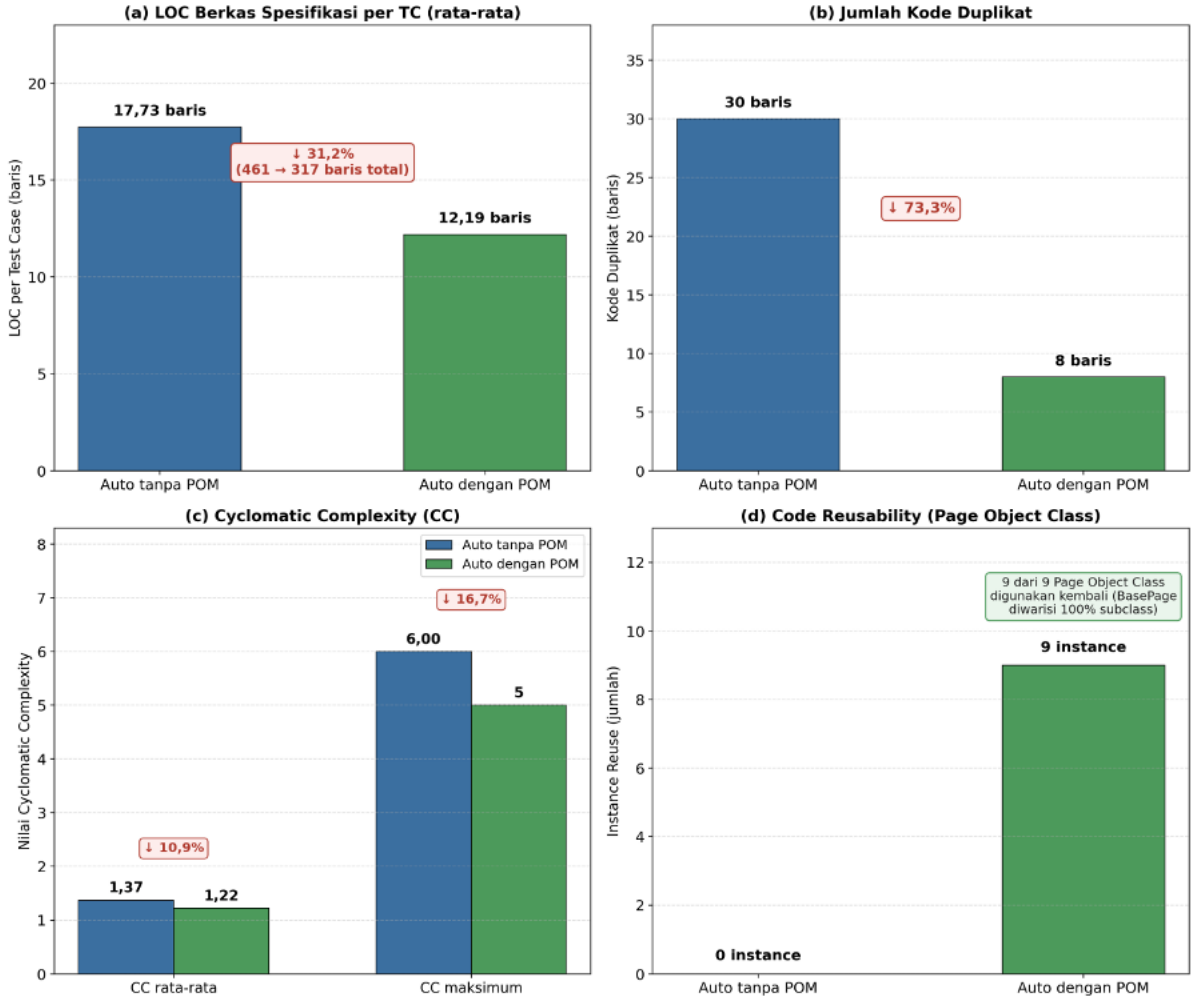
Gambar 2. Waktu Eksekusi per Iterasi dan Analisis Statistik (Total 26 Test Case, dalam detik)



Gambar 3. Rata-rata Waktu Eksekusi dan Efisiensi per Modul (mean 5 iterasi, dalam detik)

3.3 Metrik *Maintainability*: LOC, *Cyclomatic Complexity*, dan *Reusability*

Gambar 4 menyajikan perbandingan metrik *maintainability* pada ketiga skenario pengujian.



Gambar 4. Perbandingan Metrik *Maintainability* Kode Pengujian

3.4 Ringkasan Perbandingan Tiga Skenario

Tabel 4 menyajikan rekapitulasi seluruh metrik secara komparatif untuk memudahkan pengambilan keputusan.

Tabel 4. Ringkasan Seluruh Metrik Pengujian

Metrik	*Manual Testing*	Auto tanpa POM	Auto dengan POM (Playwright)
Mean Exec. Time (detik)	555,6	49,82	89,75
SD *Exec. Time* (detik)	13,11	2,77	1,12
Efisiensi vs. *Manual* (%)	-	91,0%	83,8%
Total LOC	-	461	667
CC Rata-rata	-	1,37	1,22
Code *Reusability* (*instance*)	-	0	9 *instance* / 9 *class*
Kode Duplikat (baris)	-	30	8

3.5 Efektivitas Playwright dalam Efisiensi Pengujian

Temuan penelitian ini mengkonfirmasi efektivitas *automation testing* berbasis Playwright dibandingkan pengujian *manual* dalam aspek efisiensi waktu [19]. Efisiensi *automation* tanpa loc mencapai 91,0%, sementara *automation* dengan POM mencapai 83,8% dibandingkan pengujian *manual*, keduanya jauh melampaui rata-rata yang dilaporkan dalam studi historis Haas et al. [2], yang mengidentifikasi bahwa tantangan utama *automation testing* seperti *test flakiness* dan biaya pemeliharaan seringkali mengurangi efisiensi aktual di bawah ekspektasi teoritis. Di sini, *automation* tanpa POM justru menunjukkan efisiensi yang lebih tinggi dibandingkan *automation* dengan POM, suatu temuan yang berbeda dari ekspektasi awal, namun dapat dijelaskan secara teknis pada paragraf berikutnya.

Automation tanpa POM lebih cepat karena *locator* langsung dipanggil di dalam *file test*, tanpa harus melalui kelas *page object* terlebih dahulu. Sebaliknya, *automation* dengan POM harus membuat objek dan memanggil *method* secara berlapis untuk setiap interaksi dengan tampilan, sehingga menambah sedikit waktu proses meskipun jumlahnya kecil dan konsisten di setiap eksekusi. Penjelasan ini didasarkan pada analisis struktur kode, bukan dari hasil perekaman jejak eksekusi secara langsung terhadap durasi setiap pemanggilan fungsi. *Auto waiting* Playwright [8] tetap aktif pada kedua skenario sehingga tidak menjadi faktor pembeda. *Standard deviation* yang jauh lebih rendah pada automation dengan POM (σ = 1,12 detik) dibandingkan *automation* tanpa POM (σ = 2,77 detik) dan terutama dibandingkan pengujian *manual* (σ = 13,11 detik) menunjukkan bahwa pemusatan *locator* dan standardisasi strategi wait pada kelas *page object* justru menghasilkan perilaku eksekusi yang lebih seragam antar iterasi atribut yang sangat krusial untuk integrasi CI/CD *pipeline* [4], di mana kepastian durasi waktu sering kali lebih bernilai daripada kecepatan mentah semata.

Menarik untuk dicatat bahwa selisih antara *automation* dengan POM dan tanpa POM bervariasi cukup besar antar-modul, berkisar dari 30,5% (Autentikasi) hingga 353,7% (*Onboarding*) lebih lambat untuk POM (Gambar 3). Variasi ini berkaitan dengan kompleksitas relatif setiap modul: pada modul dengan logika sederhana dan *locator* minim (seperti *Onboarding*), *overhead* pemanggilan *method* POM menjadi proporsi yang lebih besar terhadap total waktu eksekusi yang singkat. Sebaliknya, pada modul dengan operasi I/O signifikan (seperti Autentikasi yang melibatkan komunikasi dengan Supabase), waktu eksekusi didominasi oleh latensi jaringan/database sehingga *overhead* arsitektur POM menjadi proporsi yang lebih kecil. Dengan demikian, *trade off* antara kecepatan eksekusi mentah dan konsistensi/*maintainability* perlu dipertimbangkan secara kontekstual sesuai karakteristik modul yang diuji, yang akan dibahas lebih lanjut pada sub bab berikutnya terkait dampak POM terhadap *maintainability*.

3.6 Dampak POM terhadap *Code Maintainability*

Penurunan LOC dari 461 menjadi 317 baris pada berkas spesifikasi pengujian (↓31,2%) dan pengurangan kode duplikat dari 30 menjadi 8 baris (↓73,3%) menjadi bukti nyata yang kuat untuk efektivitas POM dalam meningkatkan *maintainability*. Seluruh nilai diukur menggunakan cloc v1.98 dan jscpd v4 terhadap basis kode final yaitu 26 *test case* (TC-01-TC-26). Meskipun total LOC keseluruhan proyek POM (667 baris, termasuk 350 baris pada berkas *page object*) lebih besar dibandingkan proyek non POM, kompleksitas pada berkas spesifikasi pengujian justru berkurang karena *locator* dan logika interaksi UI dipindahkan ke kelas *page object* yang reusable. Pola ini konsisten dengan temuan Leotta et al. [10] yang melaporkan bahwa pendekatan terprogram berbasis *page object* secara konsisten mengurangi upaya pemeliharaan *test suite*, meskipun besaran penghematan bergantung pada ukuran dan kompleksitas aplikasi yang diuji.

Penurunan *cyclomatic complexity* rata-rata dari 1,37 menjadi 1,22 (↓10,9%) memang tidak sebesar penurunan LOC dan duplikasi, namun tetap bermakna secara praktis. Hal ini menunjukkan bahwa sebagian besar fungsi pada kedua skenario sudah tergolong sederhana (nilai CC mendekati 1), karena *assertion* di Playwright ditulis secara deklaratif dan jarang memiliki percabangan logika yang rumit. Berdasarkan konvensi *cyclomatic complexity* yang dirujuk dalam studi terkini [20], nilai CC hingga 10 dianggap sederhana dengan risiko kesalahan rendah, konvensi yang masih dirujuk dalam studi kompleksitas kode terkini [20]. Pada skenario non POM, CC maksimum mencapai 6 yang terjadi pada *test case onboarding* karena alur percabangan (*loop* dan kondisi *isVisible*) ditulis langsung di dalam berkas spesifikasi pengujian. Pada skenario POM, CC maksimum turun menjadi 5 dan berpindah ke *OnboardingPage*.ts, karena logika percabangan tersebut didistribusikan ke metode metode Page Object yang lebih granular, sehingga setiap unit kode memiliki tanggung jawab yang lebih sempit dan mudah diuji secara terisolasi. Kedua nilai CC maksimum tetap berada jauh di bawah ambang batas tersebut, menandakan bahwa basis kode pengujian secara keseluruhan tergolong sederhana.

Penurunan CC yang relatif kecil ini wajar terjadi karena kode pengujian *end to end* di Playwright memang umumnya berbentuk alur linear (rangkaian aksi dan pengecekan/*assertion*), bukan logika bisnis bercabang seperti pada kode aplikasi produksi. Implikasi praktisnya, manfaat utama POM terhadap *maintainability* pada konteks penelitian ini terutama bersumber dari pengurangan LOC dan duplikasi yang berkaitan langsung dengan upaya pemeliharaan saat terjadi perubahan UI bukan dari penyederhanaan struktur

kontrol program. Temuan ini melengkapi pemahaman bahwa kontribusi POM terhadap *maintainability* bersifat multidimensi: sebagian besar keuntungan terletak pada sentralisasi *locator* dan *code reusability*, sementara penurunan kompleksitas struktural hanya terlihat jelas pada modul dengan logika percabangan yang memang kompleks sejak awal, seperti *Onboarding*. Bagi praktisi, temuan ini menyarankan bahwa keputusan mengadopsi POM sebaiknya tidak semata didasarkan pada ekspektasi penurunan CC yang besar, melainkan pada potensi pengurangan duplikasi dan upaya pemeliharaan jangka panjang.

Nilai *reusability* sebesar 9 *instance* dari 9 *page object class* menunjukkan setiap kelas diinstansiasi tepat satu kali melalui *custom fixture* Playwright (*fixtures/index.ts),* yang kemudian dapat dipanggil ulang oleh seluruh *test case* yang membutuhkan kelas tersebut tanpa perlu menulis ulang *locator*. *BasePage* sebagai kelas induk diwarisi (*extends*) oleh seluruh 9 dari 9 *subclass* (100%), menjadikannya komponen yang paling banyak memberi kontribusi *reuse* karena *method* umum (misal navigasi, penantian elemen) hanya didefinisikan sekali. *AuthPage* menjadi kelas dengan pemakaian langsung terbanyak pada skenario yang memerlukan autentikasi sebagai prasyarat. Pola *reuse* ini konsisten dengan temuan nyata yang menekankan nilai strategis POM dalam mengelola ketergantungan antar halaman [10]. Penentuan nilai uji pada setiap *test case* mengacu pada teknik *equivalence partitioning* [11] agar kelas input yang representatif tetap tercakup meskipun *locator* telah dipindahkan ke kelas *page object.*

3.7 Internal Validity

Ancaman utama terhadap validitas internal adalah potensi *learning effect* pada pengujian *manual*, yaitu penguji yang semakin familiar dengan *test case* dapat menyelesaikan pengujian lebih cepat pada iterasi berikutnya. Penelitian ini dilaksanakan oleh peneliti sebagai penguji tunggal (*single tester*), yang menjadi keterbatasan yang diakui secara eksplisit. Untuk mengatasi efek ini, diterapkan tiga strategi: (1) jeda istirahat minimal 30 menit antar iterasi untuk mengurangi efek memorisasi, (2) urutan eksekusi *test case* diacak pada setiap iterasi untuk mengurangi bias urutan (*order effect*), dan (3) setiap langkah pengujian *manual* mengacu pada panduan uji tertulis (*test script*) yang telah ditetapkan sebelumnya agar prosedur eksekusi konsisten antar iterasi [18]. Ancaman kedua adalah *instrumentation bias*: penggunaan *Playwright test runner* untuk mencatat waktu automation secara otomatis, dibandingkan stopwatch *manual* pada pengujian, meminimasi *human error* dalam pencatatan waktu, namun pengukuran waktu pengujian *manual* tetap mengandung margin error estimasi sebesar ±2–3 detik per *test case*. Mitigasi *instrumentation bias* melalui otomatisasi pencatatan waktu ini sejalan dengan rekomendasi dalam panduan eksperimentasi perangkat lunak [18]. Penggunaan alat pencatat waktu otomatis juga direkomendasikan secara konsisten dalam panduan desain eksperimen perangkat lunak untuk meminimalkan *measurement bias* [18].

3.8 External Validity

Generalisabilitas temuan penelitian ini dibatasi oleh beberapa faktor, sejalan dengan catatan dalam literatur bahwa efektivitas *automation testing* bersifat kontekstual terhadap arsitektur dan lingkungan aplikasi yang diuji [3]. Objek penelitian adalah satu sistem spesifik (*website* Skripsweet) dengan arsitektur Supabase dan React, hasil mungkin berbeda pada aplikasi dengan arsitektur berbeda (misalnya, aplikasi berbasis React Native atau sistem *legacy* berbasis monolitik) [3]. Leotta et al. [10] mencatat bahwa hasil eksperimen berbasis POM pada satu aplikasi tidak selalu dapat digeneralisasikan langsung ke aplikasi lain karena perbedaan arsitektur UI dan kepadatan skenario uji. Pada aspek lingkungan, pengujian dilakukan pada jaringan lokal dengan latensi minimal, performa *automation testing* dapat berbeda pada lingkungan *cloud* atau jaringan dengan latensi tinggi. Faktor pembatas lainnya, pengujian *manual* dilaksanakan oleh peneliti sebagai penguji tunggal dengan latar belakang pengembang web, penguji dengan profil yang berbeda (tester berpengalaman atau penguji awam) berpotensi menghasilkan data waktu yang berbeda. Hal ini menjadi ancaman eksternal yang perlu dipertimbangkan dalam replikasi studi ini pada situasi yang lebih beragam [18].

3.9 Construct validity

Construct validity berkaitan dengan kesesuaian antara metrik yang digunakan dan konsep yang ingin diukur. LOC yang digunakan sebagai ukuran *maintainability* memiliki keterbatasan: jumlah baris kode yang lebih sedikit tidak selalu berarti kode tersebut lebih mudah dipahami, misalnya jika kode dipadatkan (*minified*). Interpretasi LOC idealnya juga mempertimbangkan bagaimana kode berubah seiring waktu, karena pertambahan LOC tidak selalu sejalan dengan pertambahan kompleksitas fungsional [10]. Penelitian ini melengkapi LOC dengan *cyclomatic complexity* untuk memberikan gambaran yang lebih komprehensif. Selain itu, penggunaan 5 iterasi per *test case*, meskipun cukup untuk analisis statistik deskriptif dasar, belum memenuhi asumsi *central limit theorem* secara penuh. Perlu ditekankan bahwa *mean* dan SD pada penelitian ini berfungsi sebagai statistik deskriptif untuk

menggambarkan lima iterasi yang diamati, bukan sebagai dasar inferensi ke populasi eksekusi yang lebih luas, nilai SD yang jauh lebih rendah pada skenario POM tetap bermakna sebagai indikator konsistensi relatif antar skenario pada kondisi eksperimen yang identik. Penelitian selanjutnya disarankan menggunakan minimal 30 iterasi jika ingin menerapkan uji statistik inferensial [18]. Peningkatan jumlah iterasi juga membuka peluang penerapan uji hipotesis statistik seperti *t-test* berpasangan untuk membandingkan skenario secara lebih ketat [18].

4. KESIMPULAN

Penelitian ini menjawab dua *research questions* melalui eksperimen komparatif terkontrol dengan desain *within subject*, melibatkan 26 *test case* yang dieksekusi dalam 5 iterasi pada tiga skenario pengujian: *manual testing*, automation tanpa POM, dan automation dengan POM menggunakan Playwright. Berkaitan dengan (efisiensi waktu), *automation testing* menggunakan Playwright mencapai efisiensi 91,0% (tanpa POM) dan 83,8% (dengan POM) dibandingkan pengujian *manual*, dengan *mean* waktu eksekusi berkurang dari 555,6 detik menjadi 49,82 detik (tanpa POM) dan 89,75 detik (dengan POM). Temuan menarik menunjukkan bahwa automation dengan POM justru memiliki *throughput* eksekusi mentah lebih rendah dibandingkan tanpa POM, akibat *overhead* lapisan abstraksi *method call* pada *page object class*. Namun, *automation* dengan POM menunjukkan *standard deviation* jauh lebih rendah (σ = 1,12 detik) dibandingkan *automation* tanpa POM (σ = 2,77 detik) dan pengujian *manual* (σ = 13,11 detik), membuktikan konsistensi tinggi yang menjadi prasyarat integrasi ke dalam *pipeline* CI/CD[4].Temuan ini mengindikasikan adanya *trade off* antara kecepatan eksekusi mentah dan konsistensi, di mana POM mengorbankan sedikit kecepatan untuk keuntungan stabilitas yang signifikan.

Berkaitan dengan (*maintainability*), penerapan POM menurunkan LOC berkas spesifikasi pengujian sebesar 31,2%, mengurangi kode duplikat hingga 73,3%, dan menurunkan *cyclomatic complexity* rata-rata dari 1,37 menjadi 1,22. *Code reusability* mencapai 9 *instance reuse* dari 9 *page object class*, dengan *BasePage* diwarisi oleh seluruh *subclass* (100%), membuktikan efektivitas arsitektur POM dalam membangun *test suite* yang modular dan berkelanjutan [10].

Untuk penelitian selanjutnya, disarankan: (1) mengintegrasikan *framework* Playwright POM ke dalam *pipeline* CI/CD (GitHub Actions/Jenkins) dan mengukur dampaknya terhadap *deployment frequency*, (2) memperluas cakupan ke pengujian performa menggunakan Playwright dengan k6 untuk mengukur *response time* di bawah beban, serta (3) mereplikasi studi ini pada sistem platfrom manajemen skripsi dengan skala lebih besar untuk meningkatkan generalisabilitas temuan [3]. Terdapat beberapa keterbatasan penelitian yang perlu diperhatikan. Objek penelitian terbatas pada satu sistem dengan arsitektur Supabase dan React sehingga generalisasi ke arsitektur lain memerlukan studi replikasi. Dari sisi statistik, jumlah iterasi eksekusi sebanyak 5 kali belum memenuhi asumsi *central limit theorem* secara penuh sehingga analisis statistik inferensial belum dapat diterapkan. Di samping itu, pengujian *manual* dilaksanakan oleh peneliti sebagai penguji tunggal sehingga data waktu pengujian *manual* tidak mencerminkan variasi yang mungkin terjadi dengan penguji berlatar belakang berbeda. Keterbatasan keterbatasan ini telah diuraikan secara lebih rinci pada sub bab 3.7–3.9 (Ancaman terhadap Validitas).

DAFTAR PUSTAKA

[1] G. Murazvu, S. Parkinson, S. Khan, N. Liu, and G. Allen, "A Survey on Factors Preventing the Adoption of Automated Software Testing: A Principal Component Analysis Approach," *Software*, vol. 3, no. 1, pp. 1–27, 2024, doi: 10.3390/software3010001.

[2] R. Haas, R. Nömmer, E. Juergens, and S. Apel, "Optimization of Automated and Manual Software Tests in Industrial Practice: A Survey and Historical Analysis," *IEEE Transactions on Software Engineering*, vol. 50, no. 8, pp. 2005–2020, 2024, doi: 10.1109/TSE.2024.3418191.

[3] M. Leotta, B. García, F. Ricca, and J. Whitehead, "Challenges of End-to-End Testing with Selenium WebDriver and How to Face Them: A Survey," in *Proceedings of the 16th IEEE International Conference on Software Testing, Verification and Validation (ICST)*, 2023, pp. 339–350. doi: 10.1109/ICST57152.2023.00039.

[4] C. Ragkhitwetsagul, J. Krinke, M. Choetkiertikul, T. Sunetnanta, and F. Sarro, "Adoption of Automated Software Engineering Tools and Techniques in Thailand," *Empir. Softw. Eng.*, vol. 29, no. 4, p. 90, 2024, doi: 10.1007/s10664-024-10472-6.

[5] S. Balsam and D. Mishra, "Web Application Testing—Challenges and Opportunities," *Journal of Systems and Software*, vol. 219, p. 112186, 2025, doi: 10.1016/j.jss.2024.112186.

[6] U. Sa'adah *et al.*, "Framework Testing Otomatis Berbasis Serenity dan Jenkins," *Jurnal Ilmiah Teknologi Informasi (JUTI)*, vol. 19, no. 2, pp. 102–110, 2021, doi: 10.12962/j24068535.v19i2.a1017.

[7] A. Amalia and A. B. Cahyono, "Analisis Pemanfaatan Playwright untuk Automasi Pengujian Aplikasi Berbasis Web (Studi Kasus: Sistem Manajemen Jaringan)," *AUTOMATA*, vol. 3, no. 1, pp. 1–8, 2022, [Online]. Available: https://journal.uii.ac.id/AUTOMATA/article/view/21896

[8] M. Corporation, "Playwright Documentation: Core Concepts." [Online]. Available: https://playwright.dev/docs/intro

[9] S. Software, "State of Software Quality | Testing 2024." [Online]. Available: https://smartbear.com/state-of-software-quality/testing/2024/

[10] M. Leotta, F. Ricca, A. Marchetto, and D. Olianas, "An empirical study to compare three web test automation approaches: NLP-based, programmable, and capture&replay," *Journal of Software: Evolution and Process*, vol. 36, no. 5, May 2024, doi: 10.1002/smr.2606.

[11] "ISO/IEC/IEEE International Standard - Software and systems engineering --Software testing --Part 1:General concepts," Dec. 2021, doi: 10.1109/IEEESTD.2022.9698145.

[12] Y. Wang, M. V Mäntylä, Z. Liu, J. Markkula, and P. Raulamo-Jurvanen, "Improving Test Automation Maturity: A Multivocal Literature Review," *Software Testing, Verification and Reliability*, vol. 32, no. 3, p. e1804, 2022, doi: 10.1002/stvr.1804.

[13] B. Karagöz, F. Ricca, M. Biagiola, and A. Stocco, "Towards Automated Page Object Generation for Web Testing using Large Language Models," in *2026 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, IEEE, May 2026, pp. 352–363. doi: 10.1109/ICST69053.2026.00055.

[14] F. Ricca and A. Stocco, "Web Test Automation: Insights from the Grey Literature," in *Proceedings of the 47th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2021), Lecture Notes in Computer Science*, vol. 12607, Springer, 2021, pp. 472–485. doi: 10.1007/978-3-030-67731-2_35.

[15] C. Wohlin, "Case Study Research in Software Engineering—It is a Case, and it is a Study, but is it a Case Study?," *Inf. Softw. Technol.*, vol. 133, p. 106514, May 2021, doi: 10.1016/j.infsof.2021.106514.

[16] Rahmat Fauzan, Ferina Putri Soedjono, Annisa Ayu Permadani, and Muhammad Ainul Yakin, "Perbandingan Pengujian Manual dan Terotomasi pada Software Enterprise Resource Planning," *Journal of Advances in Information and Industrial Technology*, vol. 5, no. 1, pp. 23–30, May 2023, doi: 10.52435/jaiit.v5i1.318.

[17] A. U. Afidah and D. G. P. Putri, "Studi Komparasi Teknik Pengujian End-to-End pada E-payment: Studi Kasus Travelink (Sistem Web E-ticketing)," *Journal of Internet and Software Engineering*, vol. 5, no. 2, pp. 50–58, Nov. 2024, doi: 10.22146/jise.v5i2.11741.

[18] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2024. doi: 10.1007/978-3-662-69306-3.

[19] Dewandra Sapto Prasetyo and W. Silfianti, "ANALISIS PERBANDINGAN PENGUJIAN MANUAL DAN AUTOMATION TESTING PADA WEBSITE E-COMMERCE," *Jurnal Ilmiah Teknik*, vol. 2, no. 2, pp. 127–131, May 2023, doi: 10.56127/juit.v2i2.516.

[20] M. Stavtsev and Y. Bugayenko, "Evaluating the Dependency Between Cyclomatic Complexity and Response For Class," pp. 1–5, Oct. 2024.