Dampak Maturitas CI/CD Terhadap Kualitas Kode dan Rilis Proyek *Open Source* Menggunakan MSR

Syahbarudin Abdillah[1,*], Donny Maulana[2], Annisa Maulana Majid[3]

Fakultas Teknik, Teknik Informatika, Universitas Pelita Bangsa, Kab.Bekasi, Indonesia
Email: [1,*]syahbarudin.312210656@mhs.pelitabangsa.ac.id, [2]donny.maulana@pelitabangsa.ac.id,
[3]annisa.maulanamajid@pelitabangsa.ac.id
[*)] Email Penulis Utama

Abstrak− Dalam era pengembangan perangkat lunak modern, *Continuous Integration* (CI) dan *Continuous Deployment* (CD) telah menjadi praktik kunci untuk menjaga efisiensi rilis dan kualitas kode. Namun, studi empiris yang secara kuantitatif mengukur dampak adopsi CI/CD pada proyek *open source* dibandingkan dengan metode manual masih terbatas. Penelitian ini bertujuan untuk menganalisis dampak adopsi CI/CD serta hubungan antara tingkat maturitas teknis (*Build Health* dan *Time to Fix*) terhadap kualitas kode (*Bug Density*) dan efisiensi rilis. Penelitian ini menggunakan pendekatan kuantitatif dengan metode *Mining Software Repositories* (MSR). Data dikumpulkan dari repositori publik di GitHub yang menggunakan bahasa pemrograman JavaScript dan PHP. Sampel penelitian dibagi menjadi dua kelompok, yaitu proyek yang menerapkan CI/CD dan proyek yang dikelola secara manual. Instrumen penelitian berupa skrip otomasi berbasis Python digunakan untuk mengekstraksi jejak digital historis, termasuk riwayat *commit*, log *workflow*, dan laporan *issue*. Selanjutnya, data yang terkumpul akan dianalisis menggunakan uji statistik inferensial, meliputi uji beda dan uji korelasi, untuk memvalidasi hipotesis penelitian. Hasil dari penelitian ini memberikan bukti empiris mengenai efektivitas CI/CD sebagai rujukan bagi pengembang dalam mengoptimalkan strategi otomatisasi.

Kata Kunci: *Continuous Integration*, *Continuous Deployment*, *Mining Software Repositories*, Kualitas Perangkat Lunak, GitHub.

Abstract− *In the era of modern software development, Continuous Integration (CI) and Continuous Deployment (CD) have become key practices for maintaining release efficiency and code quality. However, empirical studies that quantitatively measure the impact of CI/CD adoption on open source projects compared to manual methods are still limited. This study aims to analyze the impact of CI/CD adoption and the relationship between technical maturity levels (Build Health and Time to Fix) on code quality (Bug Density) and release efficiency. This study uses a quantitative approach with the Mining Software Repositories (MSR) method. Data was collected from public repositories on GitHub that use the JavaScript and PHP programming languages. The research sample was divided into two groups, namely projects that implemented CI/CD and projects that were managed manually. The research instrument, in the form of a Python-based automation script, was used to extract historical digital traces, including commit history, workflow logs, and issue reports. Furthermore, the collected data will be analyzed using inferential statistical tests, including difference tests and correlation tests, to validate the research hypothesis. The results of this study can provide empirical evidence regarding the effectiveness of CI/CD as a reference for developers in optimizing automation strategies.*

Keywords: *Continuous Integration*, *Continuous Deployment*, *Mining Software Repositories*, *Software Quality*, *GitHub.*

1. PENDAHULUAN

Pengembangan perangkat lunak di era digital menuntut kecepatan, efisiensi, dan keandalan tinggi untuk menjaga daya saing pasar yang bergerak cepat [1]. Metodologi DevOps menempatkan *Continuous Integration* (CI) dan *Continuous Deployment* (CD) sebagai inti otomatisasi rekayasa perangkat lunak modern guna memenuhi tuntutan tersebut [2]. CI mengharuskan penggabungan kode secara teratur untuk memicu pengujian otomatis dan mendeteksi masalah sedini mungkin [3], sementara CD melengkapi praktik ini dengan mengotomatisasi perilisan ke lingkungan produksi [4]. Keseluruhan proses ini diatur dalam model *pipeline*, di mana hasil satu pengujian menjadi masukan otomatis untuk tahap berikutnya demi memastikan alur kerja yang efisien [5][6].

Secara teoritis, pendekatan terstruktur ini menjanjikan peningkatan kualitas perangkat lunak dan pengiriman perubahan yang lebih cepat [1]. Organisasi yang mengimplementasikan praktik DevOps yang matang menunjukkan hasil kinerja superior, termasuk kemampuan melakukan *deployment* 46 kali lebih sering, 440 kali lebih cepat dalam *lead time*, dan 96 kali lebih cepat dalam pemulihan kegagalan dibandingkan metode tradisional [5]. Otomatisasi ini juga memangkas waktu perbaikan kesalahan (*fault recovery time*) hingga 90% [7], serta menghindari kelemahan *deployment* manual yang rentan kesalahan [7]. Oleh karena itu, CI/CD kini menjadi standar baru di platform seperti GitHub melalui adopsi layanan terkelola seperti GitHub Actions dan Travis CI [2].

Meskipun manfaatnya jelas, kualitas dan tingkat maturitas implementasi CI/CD pada proyek *open source* masih sangat bervariasi. Bukti menunjukkan bahwa banyak pengembang cenderung tidak menguji kode secara menyeluruh sebelum mengirim *pull request*, yang mengindikasikan pendekatan CI yang belum matang [8]. Selain itu, kompleksitas konfigurasi *pipeline* sering kali membebani pengembang dengan pemeliharaan intensif, di mana persentase besar perubahan konfigurasi dialokasikan hanya untuk memperbaiki isu *build* [9][10]. Kondisi ini

menunjukkan adanya ketimpangan antara potensi manfaat ideal dengan realitas implementasi di lapangan. Masalah ini diperburuk oleh minimnya literatur yang membahas topik ini, karena mayoritas penelitian sebelumnya cenderung berfokus pada proyek *closed source* atau lingkungan industri, sehingga dinamika dampak maturasi CI/CD di ekosistem *open source* belum terpetakan secara mendalam.

Penelitian ini bertujuan menganalisis dampak maturitas CI/CD terhadap kualitas kode dan rilis pada proyek *open source* melalui analisis korelasi berbasis MSR. Dengan mengukur tingkat maturitas secara terukur, penelitian ini akan memvalidasi apakah disiplin otomatisasi yang ketat berbanding lurus dengan peningkatan metrik kualitas perangkat lunak dan keandalan pengiriman [7]. Hasil analisis memberikan wawasan empiris bagi pengembang dalam mengoptimalkan strategi CI/CD demi mencapai siklus rilis yang lebih andal dan efisien.

2. METODE PENELITIAN

2.1. Tinjauan Umum

Penelitian ini dirancang sebagai studi kuantitatif yang menerapkan paradigma *Mining Software Repositories* (MSR) untuk mengeksplorasi fenomena pengembangan perangkat lunak secara empiris. Pendekatan ini dipilih karena memungkinkan untuk mengekstraksi wawasan objektif dari sejumlah besar data historis yang tersimpan dalam repositori pengontrol versi, tanpa bergantung pada persepsi subjektif pengembang yang biasa diperoleh melalui metode survei atau wawancara.

Secara umum, penelitian ini berfokus pada pengungkapan hubungan kausalitas antara tingkat maturitas praktik otomatisasi (*Continuous Integration/Continuous Deployment*) dengan kualitas produk perangkat lunak yang dihasilkan. Seluruh proses penelitian, mulai dari pengumpulan data hingga analisis, dilakukan secara non-intrusif menggunakan instrumen perangkat lunak berbasis Python untuk memastikan data yang diperoleh merepresentasikan kondisi riil aktivitas pengembangan di lapangan.

2.2. Objek dan Sumber Data

Untuk memastikan validitas dan reliabilitas hasil penelitian, objek dan sumber data ditentukan dengan kriteria spesifik yang merepresentasikan ekosistem pengembangan perangkat lunak modern.

2.2.1. Objek Penelitian

Objek utama dalam penelitian ini adalah artefak digital yang dihasilkan selama siklus hidup pengembangan perangkat lunak (*Software Development Life Cycle*). Artefak ini meliputi:

a. File Konfigurasi CI/CD
   Merupakan Metadata yang terdapat dalam file YAML (seperti .github/workflows) yang mendefinisikan aturan, tahapan, dan kompleksitas pipeline otomatisasi.
b. Riwayat Commit dan Build
   Merupakan catatan perubahan kode sumber serta status eksekusi build (sukses atau gagal) yang mencerminkan stabilitas teknis proyek.
c. Laporan Isu (*Issue Tracking*)
   Merupakan Data pelaporan *bug* dan diskusi teknis yang menjadi indikator utama kualitas kode pasca-rilis.

2.2.2. Sumber Data

Data yang dipakai berupa data sekunder yang diperoleh dari GitHub, yang merupakan platform hosting repositori terbesar di dunia dan menyediakan metadata yang kaya melalui REST API publik. Pemilihan sumber data difokuskan pada dua ekosistem bahasa pemrograman utama, yaitu Javascript dan PHP. Kedua bahasa ini dipilih karena karakteristiknya yang bertipe dinamis (*dynamically-typed*), yang secara teoritis sangat membutuhkan implementasi CI/CD yang ketat untuk mencegah kesalahan *runtime*.

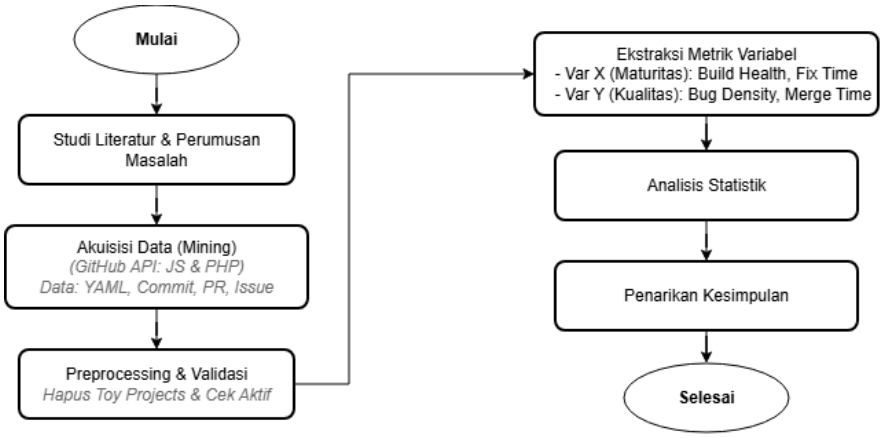
2.2.3. Populasi dan Sampel

Populasi penelitian mencakup seluruh repositori publik aktif berbahasa JavaScript dan PHP di GitHub. Dari populasi tersebut, pengambilan sampel dilakukan menggunakan teknik *Purposive Sampling* dengan total target sampel sebanyak 600 repositori. Sampel ini diklasifikasikan ke dalam dua kelompok studi untuk keperluan analisis komparatif:

a. Kelompok Eksperimen
   Merupakan Proyek yang telah menerapkan dan aktif menggunakan layanan CI/CD (seperti GitHub Actions) dalam kurun waktu 5 tahun terakhir.
b. Kelompok Kontrol
   Merupakan Proyek yang dikelola secara manual (tanpa CI/CD) namun tetap memiliki aktivitas pengembangan dan pelaporan isu yang terstruktur.

2.3. Alur Penelitian

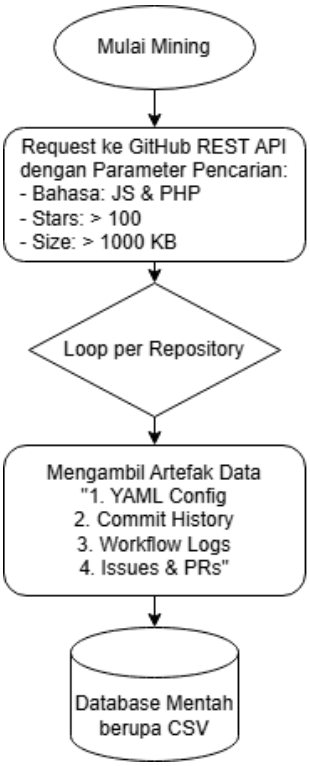
Proses penelitian dilaksanakan mengikuti alur kerja sistematis untuk mengubah data mentah repositori menjadi wawasan analitis, sebagaimana diadaptasi dari tahapan penelitian *MSR* [11][12][13]. Seluruh rangkaian kegiatan, mulai dari akuisisi data hingga penarikan kesimpulan, digambarkan secara visual dalam gambar 1 berikut:



Gambar 1. Alur Penelitian

Berdasarkan gambar di atas, rincian dari setiap tahapan penelitian adalah sebagai berikut:

a. Studi Literatur dan Perumusan Masalah
   Tahap awal dimulai dengan mengidentifikasi kesenjangan (*gap*) pada penelitian terdahulu mengenai dampak teknis penerapan CI/CD. Berdasarkan studi literatur, dirumuskan masalah penelitian yang berfokus pada analisis korelasi antara tingkat maturitas otomatisasi dengan kualitas perangkat lunak.
b. Akuisisi Data (*Data Mining*)



Gambar 2. Alur Mining Data

Gambar diatas merupakan proses pengumpulan data mentah dari repositori GitHub yang dilakukan tanpa interaksi manusia (*non-intrusif*) menggunakan teknik *mining* data sekunder. Instrumen utama yang digunakan adalah perangkat lunak (skrip Python) yang memanfaatkan GitHub REST API dan pustaka PyDriller. Sesuai dengan batasan masalah, proses *mining* difokuskan pada:

1. Bahasa Pemrograman

Bahasa yang digunakan sebagai objek penelitian difokuskan pada proyek yang menggunakan bahasa JavaScript dan PHP.

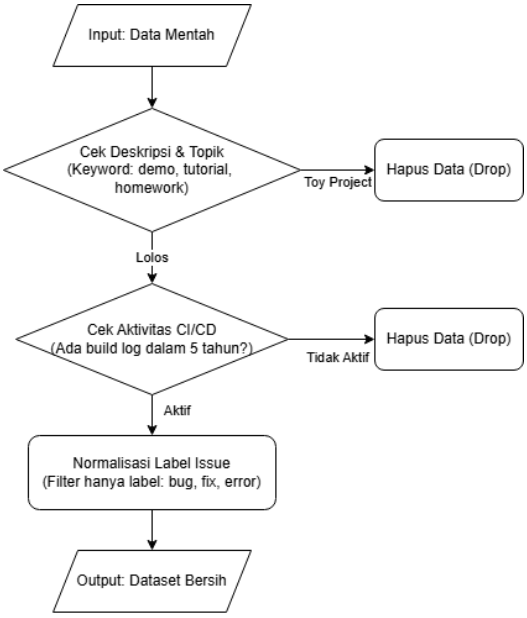
2. Artefak Data
Data yang dikumpulkan dikategorikan menjadi tiga jenis utama:

1. Data Konfigurasi CI/CD:
Data ini berisikan data keberadaan dan isi file konfigurasi (misalnya `.github/workflows/*.yml` atau `travis.yml`) yang digunakan untuk mengidentifikasi adopsi praktik CI/CD pada repositori.

2. Data Aktivitas Pengembangan:
Data ini mencakup data riwayat *commit*, frekuensi *build*, serta status keberhasilan *build* (*pass/fail*) untuk mengukur stabilitas teknis.

3. Data Kualitas dan Efisiensi
Data ini Meliputi data pelaporan *issue* (khususnya *bug report*) untuk mengukur kualitas, serta waktu penyelesaian *pull request* (*merge time*) dan riwayat *deployment* untuk mengukur efisiensi rilis.

c. *Preprocessing* dan Validasi Data
Data mentah yang diperoleh dibersihkan untuk menjamin validitas analisis statistik. Langkah-langkah penyaringan data ditunjukkan pada gambar 3 berikut:

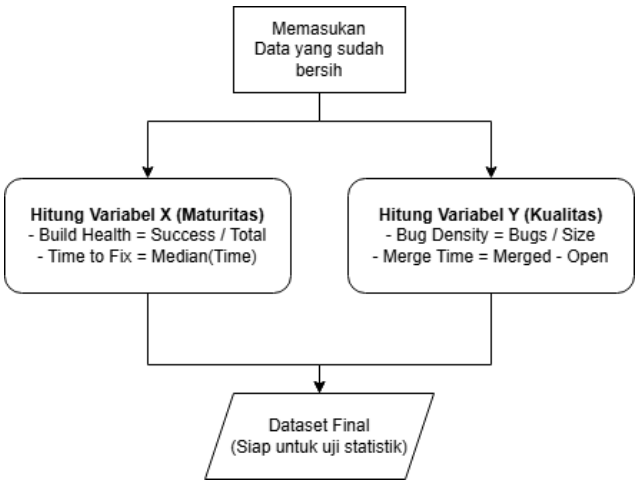


Gambar 3. Alur *Pre-processing*

Langkah pada gambar diatas meliputi:

1. Menginput data mentah yang sudah didapatkan dari hasil *mining*
2. Penyaringan *Toy Projects*:
   Menghapus repositori yang terindikasi sebagai tugas kuliah atau demo.
3. Validasi Aktivitas:
   Repositori diperiksa berdasarkan riwayat aktivitas CI/CD dalam 5 tahun terakhir yang di ikut sertakan.
4. Normalisasi Label Isu (*Issue Normalization*)
   Tahap ini bertujuan untuk menyeragamkan kategorisasi label pada data *Issues.*

d. Ekstraksi Metrik Variabel
Setelah data bersih diperoleh, tahap selanjutnya adalah menghitung nilai variabel berdasarkan data log historis. Proses transformasi data log menjadi metrik penelitian ditunjukkan pada gambar 4:



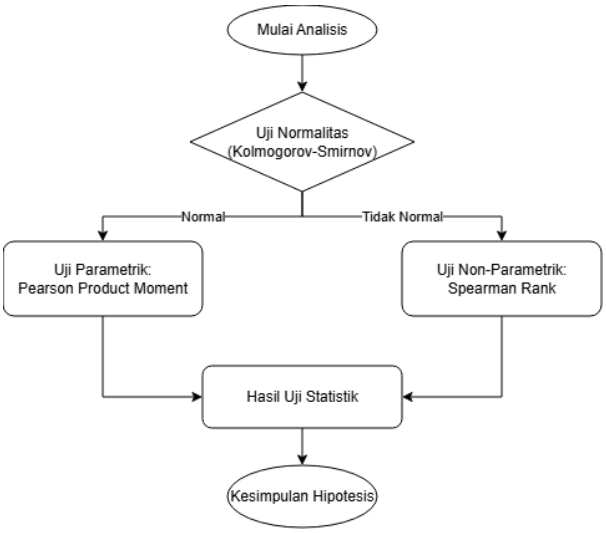
Gambar 4. Ekstraksi Variabel

Langkah pada gambar di atas meliputi:

1. Input Data Bersih:
   Menggunakan dataset yang telah lolos validasi sebagai basis perhitungan.
2. Perhitungan Variabel Independen (X):
   Menghitung metrik Maturitas CI/CD, yaitu *Build Health* (Rasio keberhasilan *build*) dan *Time to Fix* (Median waktu perbaikan).
3. Perhitungan Variabel Dependen (Y):
   Menghitung metrik luaran, yaitu *Bug Density* (Rasio *bug* per ukuran kode) dan *Merge Time* (Kecepatan rilis fitur).

e. Analisis Statistik

Tahap ini merupakan inti dari analisis kuantitatif, di mana uji statistik inferensial diterapkan untuk memverifikasi hipotesis penelitian secara empiris. Alur logika pengambilan keputusan terkait pemilihan metode statistik yang tepat diilustrasikan pada gambar 5:



Gambar 5. Uji Statistik

Langkah pada gambar di atas meliputi:

1. Uji Normalitas:
   Menguji distribusi data menggunakan metode Kolmogorov-Smirnov untuk menentukan apakah data berdistribusi normal atau tidak.
2. Penentuan Metode Uji:
   Memilih jalur pengujian berdasarkan hasil normalitas. Jalur kiri (*Pearson*) jika data normal, atau jalur kanan (*Spearman*) jika data tidak normal.

3. Uji Hipotesis (Korelasi):
   Melakukan perhitungan koefisien korelasi dan nilai signifikansi untuk menarik kesimpulan hubungan antar variabel.

f. Penarikan Kesimpulan
   Tahap akhir adalah merumuskan kesimpulan berdasarkan hasil uji statistik, serta memberikan saran praktis bagi pengembang perangkat lunak terkait strategi penerapan CI/CD yang efektif.

2.4. Alat yang digunakan

Untuk menunjang kelancaran dalam proses penelitian, digunakan seperangkat instrumen yang terdiri dari perangkat keras dan perangkat lunak yang mumpuni. Dari sisi perangkat keras, penelitian ini dilaksanakan menggunakan laptop dengan spesifikasi prosesor Intel Core i5 Generasi ke-8 yang didukung oleh memori (RAM) sebesar 16 GB. Kapasitas memori ini sangat krusial untuk mencegah kendala *out-of-memory* saat memproses *dataframe* dalam jumlah besar. Selain itu, media penyimpanan berjenis SSD berkapasitas 256 GB dipilih untuk menjamin kecepatan operasi baca-tulis (*I/O operations*) yang tinggi selama proses ekstraksi ribuan data repositori, serta didukung oleh koneksi internet yang stabil untuk memastikan komunikasi dengan GitHub REST API berjalan tanpa hambatan.

Di sisi perangkat lunak, lingkungan kerja utama beroperasi pada sistem operasi Windows 10 Pro 64-bit dengan bahasa pemrograman Python versi 3.10 atau yang lebih baru sebagai instrumen komputasi utama. Visual Studio Code (VS Code) digunakan sebagai *Integrated Development Environment* (IDE) terpusat untuk seluruh tahapan teknis. Secara spesifik, proses akuisisi data (*mining*) dieksekusi menggunakan skrip Python standar (.py) di dalam VS Code, sedangkan tahap eksplorasi dan analisis statistik dilakukan menggunakan format Jupyter Notebook (.ipynb) yang juga dijalankan melalui ekstensi VS Code untuk interaktivitas data yang lebih baik. Analisis data difasilitasi oleh ekosistem pustaka Python yang lengkap, meliputi Requests untuk akuisisi data, Pandas untuk manipulasi data tabel, serta SciPy dan Statsmodels untuk pengujian statistik inferensial (*Mann-Whitney U Test* dan *Spearman Rank*). Proses visualisasi data dibantu oleh pustaka Matplotlib dan Seaborn. Selain itu, digunakan juga aplikasi pendukung seperti Microsoft Word untuk penulisan laporan seminar proposal, Microsoft Excel untuk pengecekan data awal. Kombinasi alat-alat ini dipilih karena kemudahan akses, sifatnya yang gratis/open-source, serta kemampuannya dalam memfasilitasi seluruh tahapan proses data mining.

2.5. Definisi Operasional Variabel

Definisi operasional variabel merupakan pedoman teknis yang digunakan untuk menjembatani konsep-konsep teoritis dalam penelitian ini dengan indikator empiris yang dapat diobservasi dan diukur secara kuantitatif. Karena penelitian ini menggunakan pendekatan *Mining Software Repositories* (MSR), pengukuran variabel tidak dilakukan melalui instrumen subjektif seperti kuesioner, melainkan melalui ekstraksi jejak digital historis (*historical digital footprints*) yang terekam secara otomatis dalam metadata repositori.

Seluruh data variabel diperoleh dari log aktivitas pengembangan yang tersimpan di GitHub, termasuk riwayat *workflow* CI/CD, stempel waktu (*timestamp*) commit, serta label pelaporan isu. Pendekatan ini menjamin objektivitas data karena merefleksikan perilaku nyata pengembang dan kinerja sistem yang sesungguhnya di lapangan. Variabel-variabel tersebut diklasifikasikan ke dalam dua kategori utama sebagai berikut:

a. Variabel Independen (Maturitas CI/CD):

1. *Build Health*
   Variabel ini mengukur rasio keberhasilan integrasi kode. Nilai yang mendekati 1 (atau 100%) mengindikasikan tingkat maturitas yang tinggi, menandakan bahwa proses integrasi berjalan stabil. *Build Reliability:* Persentase rasio build yang sukses dibandingkan total build.

$$\text{Build Health} = \left(\frac{\text{Build}_{Success}}{\text{Build}_{Total}}\right) \times 100\% \quad (1)$$

Dimana:
1. $\sum Build_{Succes}$ merupakan jumlah eksekusi *pipeline* berstatus *passed*.
2. $\sum Build_{Total}$ merupakan total eksekusi *pipeline*.
3. 100% merupakan faktor pengali untuk merubah data menjadi bentuk persentasi.

2. *Time to Fix*
   Variabel ini dihitung menggunakan median waktu yang dibutuhkan tim untuk memulihkan *build* yang statusnya gagal (*failed*) menjadi sukses kembali (*passed*).

$$\text{Time to Fix} = \text{Median}(T_{Success} - T_{Failure}) \quad (2)$$

Dimana:
1. $T_{Success}$ merupakan waktu pencatatan saat *build* berikutnya berhasil diperbaiki
2. $T_{Failure}$ merupakan waktu pencatatan saat sebuah *build* terdeteksi gagal
3. $Median$ merupakan nilai tengah dari seluruh selisih waktu perbaikan yang dihitung.

3. *Build Activity*
Menggambarkan intensitas penggunaan *Continuous Integration* (CI) dalam proyek, diukur berdasarkan rata-rata jumlah eksekusi *build* per bulan.

b. Variabel Dependen:
1. Kualitas Kode
Kualitas kode dapat diukur dari densitas *bug* (*Bug Density*), yaitu rasio jumlah issue bertipe bug yang dilaporkan pasca-rilis dibagi dengan ukuran kode (*Lines of Code*) atau jumlah commit.

$$\text{Bug Density} = \frac{\sum \text{Issue}_{Bug}}{\text{Total LOC}} \quad (3)$$

Dimana :
1. $\sum(IssueBug)$: Merupakan jumlah total *issue* yang memiliki label kategori *bug*, *defect*, atau *error*.
2. $total(LOC)$: Merupakan *Total Lines of Code*, yaitu jumlah baris kode sumber dalam repositori (tidak termasuk baris kosong atau komentar) saat data diambil.

2. Efisiensi Rilis
Efisiensi rilis dapat diukur menggunakan *Merge Time* (waktu rata-rata penerimaan *Pull Request*) dan *Deployment Frequency* (jika terdeteksi adanya *job deployment*).

$$\text{Efisiensi Rilis} = \text{Median}\left(T_{deploy} - T_{commit}\right) \quad (4)$$

Dimana :
1. T_{deploy} : Merupakan waktu saat perubahan kode berhasil digabungkan (*merge*) ke *main branch* atau selesai di-*deploy*.
2. T_{commit} : Merupakan waktu saat pengembang pertama kali melakukan *commit* pada kode tersebut.
3. Median : Merupakan nilai tengah yang digunakan untuk merepresentasikan durasi tipikal agar data tidak bias oleh *pull request* yang terbengkalai lama (*stale*).

2.6. Teknik Analisis Data

Teknik analisis data dalam penelitian ini diarahkan untuk mengolah data sekunder yang telah melalui tahap *preprocessing* menjadi informasi yang valid guna menjawab rumusan masalah dan menguji hipotesis penelitian. Dikarenakan data yang dikumpulkan merupakan data numerik berskala rasio (seperti jumlah *bug*, durasi waktu, dan frekuensi), maka metode analisis yang diterapkan adalah statistik inferensial. Pendekatan ini dipilih untuk menarik kesimpulan umum (generalisasi) mengenai populasi proyek *open source* berdasarkan sampel data yang telah diambil.

Seluruh proses pengolahan dan perhitungan statistik dilakukan secara komputasional menggunakan perangkat lunak analisis data berbasis Python (memanfaatkan pustaka *SciPy* dan *Statsmodels*). Penggunaan alat bantu komputasi ini bertujuan untuk meminimalisir kesalahan perhitungan manusia (*human error*) serta menangani volume data repositori yang besar secara efisien. Secara sistematis, alur analisis data dibagi menjadi tiga tahapan utama, dimulai dari pemaparan karakteristik data, pengujian prasyarat distribusi, hingga pengujian hipotesis untuk melihat hubungan antar variabel. Rincian tahapan tersebut adalah sebagai berikut:

a. Statistik Deskriptif
Tahap awal analisis dilakukan dengan menghitung statistik deskriptif untuk memberikan gambaran umum mengenai karakteristik data yang telah dikumpulkan. Parameter yang diukur meliputi nilai rata-rata (*mean*), nilai tengah (*median*), standar deviasi, serta nilai minimum dan maksimum dari variabel maturitas CI/CD dan kualitas kode. Analisis ini bertujuan untuk mendeteksi adanya anomali data (*outlier*) serta memberikan pemahaman awal mengenai profil distribusi data sebelum dilakukan pengujian inferensial lebih lanjut.

b. Uji Prasyarat Analisis (Uji Normalitas)
Sebelum dilakukan uji korelasi, wajib dilakukan uji normalitas untuk menentukan apakah data sampel berasal dari populasi yang berdistribusi normal atau tidak. Mengingat data yang dipakai berjumlah 600 data maka metode yang digunakan adalah Uji *Kolmogorov-Smirnov* .

$$D = \sup_x |F_n(x) - F(x)| \quad (5)$$

Dimana :
1. D : Merupakan nilai statistik uji Kolmogorov-Smirnov (selisih maksimum).
2. sup_x : Merupakan *Supremum*, yaitu jarak vertikal terjauh antara dua kurva distribusi.
3. $F_n(x)$: Merupakan fungsi distribusi kumulatif empiris (berdasarkan data observasi).
4. $F(x)$: Merupakan fungsi distribusi kumulatif teoritis (distribusi normal ideal).

c. Uji Korelasi

Pengujian ini merupakan inti analisis yang bertujuan untuk mengukur kekuatan dan arah hubungan antara variabel independen (Maturitas CI/CD: *Build Health, Time to Fix*) dengan variabel dependen (Kualitas Kode: *Bug Density* dan Efisiensi Rilis: *Merge Time*). Pemilihan teknik korelasi didasarkan pada hasil uji normalitas sebelumnya:

1. *Pearson Product Moment Correlation*

Digunakan apabila data terbukti berdistribusi secara normal dan memiliki hubungan linier.

$$r_{xy} = \frac{n\sum(xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \quad (6)$$

Dimana :
1. r_{xy} : Merupakan koefisien korelasi Pearson.
2. n : Merupakan jumlah sampel repositori.
3. x : Merupakan nilai skor variabel independen (Maturitas CI/CD).
4. y : Merupakan nilai skor variabel dependen (Kualitas Kode/Efisiensi).
5. $\sum xy$: Merupakan jumlah hasil perkalian skor X dan Y.
6. $\sum x$: Merupakan jumlah kuadrat skor X.
7. $\sum y$: Merupakan jumlah kuadrat skor Y.

2. *Spearman Rank Correlation*

Digunakan apabila data tidak berdistribusi normal. Teknik ini lebih direkomendasikan dalam studi repositori perangkat lunak (*Software Engineering*) mengingat data metrik kode sering kali memiliki distribusi yang miring (*skewed*) dan mengandung *outlier*.

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)} \quad (7)$$

Dimana :
1. ρ (rho) : Koefisien korelasi Spearman.
2. d_i : Selisih peringkat (*rank*) antara variabel X dan variabel Y untuk setiap data.
3. n : Jumlah sampel repositori.
4. 1 dan 6 : Konstanta baku rumus Spearman.

d. Interpretasi Hasil

Untuk menjawab rumusan masalah secara objektif, analisis diawali dengan merumuskan hipotesis statistik yang terdiri dari Hipotesis Nol (H_0), yang mengasumsikan tidak adanya hubungan signifikan antara tingkat maturitas CI/CD dengan kualitas kode (*Bug Density*) maupun efisiensi rilis (*Lead Time*), dan Hipotesis Alternatif (H_a) yang menyatakan adanya hubungan tersebut. Keputusan pengujian didasarkan pada perbandingan nilai probabilitas ($P - value$) terhadap taraf signifikansi ($\alpha = 0.05$) apabila ($P - value < 0.05$), maka (H_0) ditolak yang berarti terbukti secara statistik terdapat hubungan yang signifikan antar variabel, sedangkan jika ($P - value > 0.05$) maka (H_0) diterima. Jika hubungan terbukti signifikan, analisis dilanjutkan dengan menginterpretasikan nilai koefisien korelasi (r atau ρ) untuk menentukan kekuatan hubungan (berkisar dari sangat lemah hingga sangat kuat) serta arah hubungannya, di mana korelasi bernilai negatif (-) pada variabel *Bug Density* dan *Lead Time* diharapkan muncul sebagai indikasi bahwa peningkatan maturitas CI/CD berdampak pada penurunan tingkat *error* dan durasi tunggu rilis.

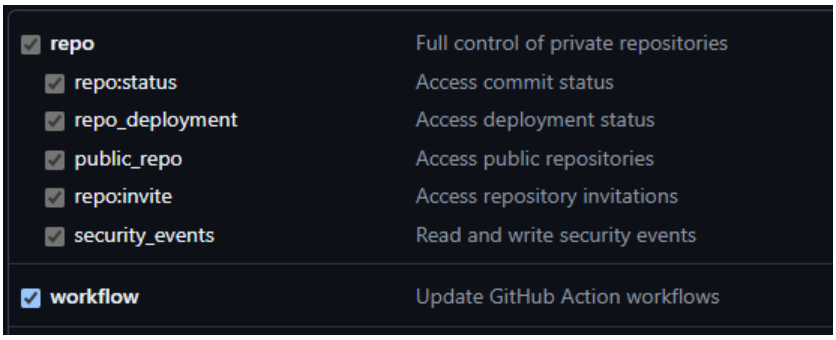
3. HASIL DAN PEMBAHASAN

3.1. Implementasi

Proses pengumpulan data dalam penelitian ini dilakukan melalui teknik *Mining Software Repositories* (MSR) dengan memanfaatkan antarmuka GitHub REST API. Pengumpulan data tidak dilakukan secara manual, melainkan diimplementasikan menggunakan skrip pemrograman Python yang dirancang khusus untuk

mengekstraksi metrik maturitas *Continuous Integration* dan *Continuous Deployment* (CI/CD), riwayat *issue*, serta data *pull request* secara otomatis.

Penelitian ini memfokuskan ekstraksi pada repositori yang menggunakan bahasa pemrograman JavaScript dan PHP. Pemilihan kedua bahasa ini didasarkan pada popularitasnya yang sangat tinggi di ekosistem pengembangan web *open-source*, sehingga dapat merepresentasikan populasi proyek yang masif. Untuk memastikan bahwa proyek yang dianalisis memiliki skala, kualitas, dan tingkat keaktifan yang memadai, pencarian dibatasi dengan parameter khusus, yaitu repositori harus memiliki lebih dari 20 *stars* dan ukuran repositori di atas 500 KB.



Gambar 6. Konfigurasi Token Akses pada API GitHub

Dalam pelaksanaannya, proses penarikan data berskala besar ini dihadapkan pada regulasi batasan permintaan (*rate limit*) dari peladen GitHub. Guna mencegah terhentinya program di tengah jalan, algoritma skrip Python telah dilengkapi dengan mekanisme penanganan galat (*error handling*). Ketika sistem mendeteksi respons batas limit (status kode HTTP 403 atau 429), program akan secara otomatis melakukan jeda eksekusi (*sleep*), lalu melanjutkan proses ekstraksi kembali setelah limit diatur ulang oleh peladen.

Gambar 7. Proses mengambil data

3.2. Hasil *Processing*

Data mentah yang telah ditarik dari GitHub tidak dapat langsung digunakan untuk pengujian statistik karena berpotensi mengandung derau (*noise*). Oleh karena itu, data tersebut melewati serangkaian tahapan *preprocessing* yang ketat guna menjamin homogenitas dataset.

Tahap *preprocessing* dibagi menjadi dua filter utama:

a. Penyaringan Proyek Mainan (*Toy Projects*)
   Mengeliminasi repositori yang dibuat hanya untuk tujuan edukasi sementara, seperti tugas kuliah, tutorial, atau repositori demo.
b. Validasi Aktivitas CI/CD
   Mengeliminasi repositori yang dikelola secara manual. Repositori yang tidak memiliki rekam jejak aktivitas *workflow* CI/CD (seperti GitHub Actions) dalam kurun waktu 5 tahun terakhir langsung dihapus (*drop*) dari dataset.

Melalui kedua tahap penyaringan di atas, didapatkan total sampel akhir sebanyak **600 repositori** proyek *open-source* ber-CI/CD aktif.

No	repo_name	language	x_reliability	x_mttr_hours	y1_bug_density	y2_release_interval_days
1	nix-community/nur-search	JavaScript	0.57	0	0	0.433414352
2	justoneapi/crawl-data-api	JavaScript	0.98	1.815833333	0	0
3	eadpucv/pix	JavaScript	0.903225806	5.631944444	0	0.000127315
4	night/betterttv	JavaScript	0.7	0.039444444	14.59059233	0.017210648
5	jonobr1/force-directed-graph	JavaScript	0.684931507	0.054722222	0.386996904	6.94E-05
6	skylartaylor/cros-updates	JavaScript	0.98	0.876527778	0	0.002939815
7	zarazhangrui/follow-builders	JavaScript	0.984375	20.23805556	0	0
8	KilledByAPixel/LittleJS	JavaScript	0.94	85.20305556	0	0.408668981
9	Baskerville42/outage-data-ua	JavaScript	0.99	0.863333333	0.000166497	0.001423611
10	wazum/sluggi	PHP	0.7	0	3.362648733	9.585775463
11	mikopbx/Core	PHP	0.68	0.110833333	0.765250531	0.000162037
12	eliashaeussler/typo3-solver	PHP	0.89	0.000833333	0	0.419618056
13	WordPress/wordpress.org	PHP	0.56	0	0	0
14	vakhov/fresh-proxy-list	PHP	0.99	1.71	0	0
15	Roave/behat-psr11extension	PHP	0.21	9.372083333	0	0.194988426

Gambar 8. Cuplikan Dataset Hasil Ekstraksi Variabel X dan Y

3.3. Analisa Statistik deskriptif

Analisis statistik deskriptif merupakan tahapan awal yang sangat krusial dalam rangkaian pengolahan data penelitian kuantitatif. Fungsi utama dari analisis ini adalah untuk mereduksi dan meringkas kumpulan data mentah (*raw data*) yang bervolume besar menjadi bentuk ringkasan informasi yang lebih terstruktur, sistematis, dan mudah dipahami, tanpa melakukan penarikan kesimpulan atau generalisasi secara langsung terhadap populasi luas. Dalam konteks penelitian rekayasa perangkat lunak (*software engineering*) yang memanfaatkan jejak digital historis, statistik deskriptif memberikan fondasi dasar untuk mengamati kecenderungan pemusatan data (*central tendency*) serta derajat penyebaran data (*dispersion*) dari setiap dimensi metrik teknis yang telah diekstraksi.

Pada penelitian ini, analisis statistik deskriptif diterapkan secara menyeluruh terhadap dataset final yang mencakup 600 repositori proyek *open-source* aktif di platform GitHub. Pengamatan dilakukan secara granular pada empat metrik operasional utama yang merepresentasikan variabel penelitian, yaitu metrik keandalan alur kerja (*x_reliability*), kecepatan pemulihan kegagalan sistem (*x_mttr_hours*), kepadatan cacat kode (*y1_bug_density*), dan interval waktu antar rilis fitur (*y2_release_interval_days*).

Melalui pendekatan deskriptif kuantitatif ini, peneliti dapat melakukan identifikasi awal terhadap karakteristik unik dari data repositori, mendeteksi keberadaan nilai-nilai ekstrem atau pencilan (*outlier*) yang sering muncul pada data aktivitas pengembang, serta melihat gambaran variabilitas metrik antara proyek berbasis bahasa pemrograman JavaScript dan PHP. Pemahaman mendalam mengenai profil data mentah ini sangat diperlukan sebagai landasan dan syarat sebelum melangkah pada tahapan pengujian prasyarat distribusi dan pengujian hipotesis inferensial.

Secara komprehensif, ringkasan hasil kalkulasi statistik deskriptif yang memuat indikator nilai rata-rata (*mean*), standar deviasi (*std*), nilai minimum (*min*), nilai tengah (*median* atau kuantil 50%), serta nilai maksimum (*max*) untuk masing-masing variabel penelitian disajikan secara sistematis pada Tabel 1 berikut:

Tabel 1. Statistik Deskriptif

Statistik	x_reliability	x_mttr_hours	y1_bug_density	y2_release_interval_days
mean	0.779908	35.125293	1.506645	1.623657
std	0.257217	254.631372	8.751848	9.73116
min	0	0	0	0
50% (median)	0.87	0.068333	0	0.099832
max	1	5083.648889	193.69895	203.573403

Berdasarkan Tabel 1 di atas, ringkasan statistik deskriptif telah memetakan ukuran pemusatan data (seperti rata-rata dan median) sekaligus ukuran penyebaran data (standar deviasi, nilai minimum, dan maksimum) dari keempat variabel utama pada keseluruhan 600 repositori sampel. Melalui distribusi agregat angka-angka tersebut, tren performa teknis dan perilaku metrik rekayasa perangkat lunak dari ekosistem proyek *open-source* yang diteliti dapat diinterpretasikan secara lebih spesifik melalui beberapa poin pemahaman berikut:

a. Keandalan Build (X1)
   Dengan nilai rata-rata 77.9% dan median 87%, mayoritas proyek *open-source* sampel telah memiliki ekosistem CI/CD yang andal.

b. Kecepatan Perbaikan (X2)
   Nilai median MTTR berada di angka 0.068 jam (sekitar 4 menit), mengindikasikan perbaikan sistem otomasi umumnya dilakukan dengan sangat cepat. Namun, keberadaan nilai rata-rata yang melambung hingga 35 jam membuktikan adanya pencilan (*outlier*) ekstrem di mana kerusakan sistem gagal diperbaiki selama berhari-hari.
c. Indikasi Sebaran Data
   Adanya perbedaan jauh antara *Mean* dan *Median* diiringi nilai standar deviasi yang besar merupakan indikasi awal bahwa data tidak berdistribusi secara normal, yang selanjutnya akan dibuktikan melalui uji prasyarat analisis.

3.4. Uji normalitas

Sebelum melakukan pengujian hipotesis, perlu dilakukan Uji Normalitas untuk mengetahui apakah populasi data pada masing-masing variabel berdistribusi normal atau tidak. Hasil dari uji ini akan menentukan teknik korelasi yang digunakan (Pearson atau Spearman).

Penelitian ini menggunakan metode Kolmogorov-Smirnov (K-S) dengan taraf signifikansi (α) sebesar 0.05. Hasil komputasi uji normalitas disajikan pada Tabel 2.

Tabel 2. Uji Normalitas

Variabel	P-Value	Keterangan	y1_bug_density	y2_release_interval_days
X1 (Reliability)	0	TIDAK Normal	1.506645	1.623657
X2 (MTTR)	0	TIDAK Normal	8.751848	9.73116
Y1 (Bug Density)	0	TIDAK Normal	0	0
Y2 (Release Interval)	0	TIDAK Normal	0	0.099832
max	1	5083.648889	193.69895	203.573403

Berdasarkan Tabel 2, nilai probabilitas (*p-value*) dari keempat variabel menunjukkan angka 0.00000. Karena nilai tersebut lebih kecil dari 0.05, maka sebaran data pada seluruh variabel penelitan terbukti secara statistik tidak berdistribusi normal. Ketidaknormalan ini sangat lazim dalam data MSR, di mana metrik perangkat lunak rentan terhadap nilai ekstrem. Karena data tidak memenuhi asumsi parametrik, maka pengujian hipotesis ditetapkan menggunakan Korelasi Rank Spearman.

3.5. Uji korelasi

Pengujian hipotesis dilakukan untuk mengukur signifikansi dampak maturitas CI/CD menggunakan korelasi Spearman. Rangkuman hasil pengujian disajikan pada Tabel 3.

Tabel 3. Uji Korelasi

Variabel yang Diuji	Metode Terpilih	Nilai Korelasi (r)	P-Value	Kesimpulan
Reliability* vs *Bug Density	Spearman (Rank)	-0.0966	0.0179	SIGNIFIKAN (H1 Diterima, Arah Negatif)
MTTR* vs *Release Interval	Spearman (Rank)	0.1832	0.00001	SIGNIFIKAN (H1 Diterima, Arah Positif)

Berdasarkan data yang disajikan pada Tabel 3, seluruh pengujian hipotesis dilakukan menggunakan metode statistik non-parametrik yaitu Korelasi *Rank Spearman*. Pemilihan metode ini didasarkan pada hasil uji prasyarat sebelumnya yang menyatakan bahwa seluruh variabel tidak berdistribusi normal, sehingga analisis linear

konvensional (Pearson) tidak dapat diterapkan karena akan menghasilkan kesimpulan yang bias. Dalam uji korelasi Spearman, kekuatan hubungan antar variabel diukur melalui nilai koefisien korelasi (r atau ρ) yang memiliki rentang nilai antara -1 hingga +1. Nilai koefisien yang mendekati angka 0 menandakan hubungan yang semakin lemah, sedangkan nilai yang mendekati angka -1 atau +1 menunjukkan hubungan monotonik yang semakin kuat dan sempurna.

Selain melihat nilai koefisien korelasi (r), indikator paling krusial untuk menentukan sah atau tidaknya suatu hipotesis dalam penelitian kuantitatif ini adalah dengan meninjau nilai probabilitas atau $P - Value$. Aturan pengambilan keputusan statistik yang diterapkan secara ketat mengacu pada batas taraf signifikansi ($\alpha = 0.05$). Apabila nilai $P - Value$ yang dihasilkan dari perhitungan komputasi menunjukkan angka yang lebih kecil atau sama dengan taraf signifikansi ($p \leq 0.05$), maka Hipotesis Nol (H_0) secara otomatis ditolak dan Hipotesis Alternatif (H_a) diterima, yang berarti hubungan antar variabel terbukti signifikan secara statistik. Sebaliknya, jika $P - Value > 0.5$, maka (H_0) diterima yang mengindikasikan bahwa tidak terdapat hubungan yang nyata antar variabel.

Tabel 3 secara jelas memperlihatkan bahwa kedua skenario pengujian hipotesis baik pengujian dampak keandalan terhadap kepadatan bug maupun kecepatan perbaikan terhadap interval rilis berhasil memproleh nilai P-Value yang berada di bawah ambang batas 0.05. Hal ini menunjukkan bahwa seluruh hipotesis alternatif H_a yang diajukan dalam penelitian ini diterima secara sah. Untuk membedah secara rinci mengenai mekanisme hubungan, nilai koefisien korelasi, arah hubungan, serta interpretasi teknis dari masing-masing hasil pengujian statistik tersebut, analisis mendalam dijabarkan pada sub-bab berikut ini.

3.5.1 Dampak keandalan terhadap *bug*

Pada pengujian pertama, dirumuskan hipotesis H_1 bahwa terdapat hubungan yang signifikan antara tingkat keandalan CI/CD dengan kepadatan *bug*. Berdasarkan Tabel 4.3, diperoleh *p-value* sebesar 0.01790. Karena $p - value \leq 0.05$, maka H_0 ditolak dan H_1 diterima. Nilai koefisien korelasi (r) menunjukkan angka negatif -0.0966. Hal ini bermakna semakin matang/tinggi tingkat keandalan suatu *pipeline* CI/CD, maka kepadatan kecacatan perangkat lunak (*bug density*) pada repositori tersebut akan semakin menurun.

3.5.2 Dampak kecepatan terhapad efisiensi rilis

Pada pengujian kedua, dirumuskan hipotesis H_1 bahwa terdapat hubungan yang signifikan antara kecepatan perbaikan (MTTR) dengan interval rilis. Berdasarkan komputasi data, diperoleh $p - value$ sebesar 0.00001. Karena $p - value \leq 0.05$ maka H_0 ditolak dan H_1 diterima. Koefisien korelasi (r) menunjukkan nilai positif sebesar 0.1832. Hal ini membuktikan bahwa semakin besar waktu yang dibutuhkan tim untuk memulihkan kegagalan *build* (MTTR tinggi/lambat), maka interval waktu antar rilis fitur baru akan semakin melebar (proses rilis melambat).

3.6. Pembahasan

Hasil analisis data kuantitatif dari 600 repositori GitHub memberikan pembuktian empiris yang berharga bagi pengembangan teori DevOps dan rekayasa perangkat lunak praktis. Temuan ini menegaskan bahwa sekadar mengadopsi alat bantu otomatisasi CI/CD tidak secara otomatis menjamin kesuksesan proyek; manfaat nyata dari teknologi tersebut sangat bergantung pada tingkat maturitas (kematangan) implementasinya di lapangan.

Korelasi negatif yang terbukti pada hipotesis pertama (Keandalan terhadap Kepadatan Bug) mendukung kuat premis teoritis bahwa mekanisme *Continuous Integration* yang matang bertindak sebagai "gerbang kualitas" (*safety net*) yang tangguh. Proyek pengembangan yang berhasil mempertahankan tingkat *build reliability* yang tinggi mencerminkan adanya kedisiplinan tim dalam menerapkan arsitektur pengujian otomatis yang komprehensif (*unit testing, linting, integration testing*).

Di sisi lain, hasil pengujian hipotesis kedua (MTTR terhadap Interval Rilis) menyoroti aspek perilaku sumber daya manusia (*human factors*) dalam budaya kerja DevOps. Hubungan searah yang signifikan ini membuktikan bahwa durasi perbaikan kerusakan *build* otomatis yang berlarut-larut secara statistik menjadi hambatan teknis (*bottleneck*) utama yang melumpuhkan efisiensi alur *Continuous Deployment*.

Sebagai pendalaman analisis, rata-rata *Mean Time to Recovery* (MTTR) yang melambung hingga 35 jam (dengan *median* hanya 4 menit) terjadi akibat pencilan (*outlier*) ekstrem pada proyek *open source*. Kegagalan *pipeline* sering terbengkalai berhari-hari karena sifat kontribusi yang mengandalkan tenaga sukarela (*volunteer-driven*), sehingga tumpukan *build* yang tidak terurus ini menarik nilai rata-rata secara drastis ke atas.

Selain itu, terdapat perbedaan dinamika yang signifikan antara proyek JavaScript dan PHP. Proyek JavaScript memiliki rantai dependensi pustaka (*NPM*) yang masif dan berlapis, sehingga *pipeline* CI/CD sangat rentan mengalami kegagalan akibat konflik pembaruan pihak ketiga (*dependency hell*). Sebaliknya, proyek PHP umumnya memiliki arsitektur *server-side* yang lebih monolitik dan stabil, di mana kegagalan *pipeline* lebih banyak dipicu oleh kesalahan logika internal kode dibandingkan masalah integrasi eksternal.

3.6.1 Ancaman Terhadap Validitas (*Threats of Validity*)

Dalam setiap penelitian empiris, khususnya yang berfokus pada rekayasa perangkat lunak dan memanfaatkan pendekatan *Mining Software Repositories* (MSR), tidak dapat dihindari adanya potensi bias dan batasan metodologis yang dapat memengaruhi hasil akhir. Bagian "Ancaman Terhadap Validitas" ini disusun bukan untuk melemahkan signifikansi temuan yang telah dijabarkan pada sub-bab sebelumnya, melainkan sebagai bentuk transparansi akademis untuk memberikan batasan yang jelas mengenai sejauh mana hasil pengujian ini dapat diinterpretasikan dan digeneralisasikan.

Penggunaan data sekunder yang bersumber dari jejak digital platform pihak ketiga (GitHub) membawa sejumlah tantangan bawaan, terutama terkait akurasi pelaporan metrik oleh pengembang dan representasi populasi perangkat lunak. Oleh karena itu, meskipun hasil penelitian terbukti signifikan secara statistik, terdapat tiga dimensi batasan utama yang perlu dipertimbangkan secara kritis untuk menjaga objektivitas temuan riset ini, yaitu:

a. Magnitudo Hubungan
   Nilai koefisien korelasi Spearman yang didapatkan dalam penelitian ini berada pada cakupan magnitudo yang tergolong lemah ($r < 0.3$. Namun, dalam ranah empiris Rekayasa Perangkat Lunak, nilai korelasi yang kecil ini sangat wajar karena kualitas sistem secara praktis dipengaruhi oleh puluhan variabel eksternal secara simultan (seperti keahlian individu kontributor, beban kerja, dan kompleksitas arsitektur perangkat lunak). Nilai *p-value* yang sangat kecil membuktikan bahwa meskipun dampaknya tidak dominan secara absolut, pengaruh maturitas CI/CD tetap konsisten dan terukur di tengah variabel pengganggu tersebut.
b. Validitas Konstruk (*Construct Validity*)
   Penelitian ini menggunakan kepadatan *issue* dengan label "bug" dari GitHub sebagai metrik proksi untuk mengukur kualitas kode (*Bug Density*). Namun, praktik pelabelan *issue* dapat bervariasi secara signifikan antar repositori. Terdapat kemungkinan bahwa beberapa kecacatan perangkat lunak diselesaikan melalui *pull request* tanpa pernah dicatat sebagai *issue* berlabel "*bug*", sehingga terdapat potensi *bug* yang tidak masuk ke dalam perhitungan skrip *mining*.
c. Validitas Eksternal (*External Validity*)
   Sampel dalam penelitian ini dibatasi secara spesifik pada repositori yang menggunakan bahasa pemrograman JavaScript dan PHP. Oleh karena itu, kemampuan generalisasi temuan ini pada ekosistem bahasa pemrograman *compiled* yang memiliki arsitektur *build* lebih berat (seperti Java, C++, atau Go) mungkin memerlukan penyesuaian asumsi metodologis lebih lanjut.

3.6.2 Implikasi Teoritis dan Praktis

Temuan dalam penelitian ini memiliki beberapa implikasi penting yang dapat diterapkan:

a. Implikasi Teoretis
   Penelitian ini memperkuat literatur akademik di bidang *Software Engineering* dan DevOps dengan membuktikan secara empiris bahwa prinsip-prinsip DevOps memiliki korelasi linear terhadap kualitas produk akhir di ekosistem *open-source*.
b. Implikasi Praktis bagi *Maintainer* Proyek
   Memberikan panduan berbasis data bahwa tim tidak boleh membiarkan *pipeline* integrasi sering mengalami kegagalan guna menekan angka *bug density*.
c. Implikasi Praktis bagi Pengembang (*Developer*)
   Temuan ini menegaskan bahwa otomatisasi penuh sekalipun tidak akan memberikan efisiensi rilis jika tidak diimbangi dengan budaya responsivitas yang tinggi terhadap kegagalan *build*.

4. KESIMPULAN

Berdasarkan pengolahan data terhadap 600 repositori menggunakan teknik MSR, dapat disimpulkan bahwa maturitas implementasi CI/CD memberikan dampak yang nyata dan terukur terhadap kualitas perangkat lunak. Pertama, terbukti adanya korelasi signifikan dengan arah berbanding terbalik antara keandalan *pipeline* CI/CD dengan kepadatan *bug* (*bug density*). *Pipeline* integrasi otomatis yang andal dan stabil berfungsi efektif sebagai gerbang kualitas yang mendeteksi kesalahan sintaksis secara dini. Kedua, ditemukan hubungan signifikan yang berbanding lurus antara *Mean Time to Recovery* (MTTR) terhadap interval rilis. Waktu pemulihan kerusakan *pipeline* yang lambat terbukti membekukan alur *Continuous Deployment*, sehingga menghambat distribusi pembaruan fitur ke pengguna. Penelitian ini menegaskan kepada praktisi *open-source* bahwa kelincahan pengembangan tidak hanya ditentukan oleh otomatisasi alat, melainkan dari kedisiplinan dan responsivitas tim dalam menjaga stabilitas infrastruktur DevOps tersebut.

UCAPAN TERIMAKASIH

Terima kasih disampaikan kepada seluruh pihak yang terlibat dalam penelitian ini, serta kepada program studi Teknik Informatika Universitas Pelita Bangsa yang telah memfasilitasi kelancaran penelitian ini.

DAFTAR PUSTAKA

[1] Z. Zulhakim and A. Kurniawan, "Implementasi Continuous Integration Dan Continuous Deployment Pada Pengembangan Aplikasi Website Menggunakan Docker Dan Github Actions," pp. 1–11, 2024.

[2] F. Zampetti, S. Geremia, G. Bavota, S. Italiana, and M. Di Penta, "CI / CD Pipelines Evolution and Restructuring : A Qualitative and Quantitative Study" *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Luxembourg, 2021, pp. 471-482, doi: 10.1109/ICSME52107.2021.00048.

[3] Hapnes *et al.*, "Evaluasi Metodologi CI / CD untuk Pengembangan," vol. 8, no. 2, pp. 227–234, 2022. 3

[4] M. Syauqi, A. Amrullah, and G. I. Marthasari, "Otomatisasi Proses Deployment dengan Metode CI / CD Menggunakan Jenkins dan Docker Pada Web Service i-Lab," vol. 6, no. 4, pp. 313–322, 2024. 4

[5] N. Ruseno, S. Kurniawan, and G. Santoso, "Penerapan Devops (Development Operations) Dalam Pengembangan Aplikasi Untuk Meningkatkan Kecepatan Delivery Software," vol. 01, no. 01, pp. 44–53, 2025. 5

[6] J. Santos and D. Alencar, "*Investigating the Impact of Continuous Integration Practices on the Productivity and Quality of Open-Source Projects*," vol. 1, no. 1. Association for Computing Machinery, 2022. doi: 10.1145/3544902.3546244.

[7] Jaeni, N. A. S, and A. D. L,"IMPLEMENTASI CONTINUOUS INTEGRATION / CONTINUOUS DELIVERY (CI / CD) PADA PERFORMANCE TESTING DEVOPS" vol. 2, pp. 1–5, 2021. doi: https://doi.org/10.24076/joism.2022v4i1.887.

[8] M. R. Wróbel, J. Szymukowicz, and P. Weichbroth, "Using Continuous Integration Techniques in Open Source Projects — An Exploratory Study," vol. 11, pp. 113848–113863 no. October, 2023, doi: 10.1109/ACCESS.2023.3324536.

[9] D. Spadini and A. Bacchelli, "PyDriller : Python framework for mining software repositories PyDriller : Python Framework for Mining Software Repositories," no. December, pp. 908–911, 2021, doi: 10.1145/3236024.3264598

[10] M. Ccallo-luque and A. Quispe-quispe, "Adoption and Adaptation of CI / CD Practices in Very Small Software Development Entities : A Systematic Literature Review," pp. 1–7, 2024, doi:10.48550/arXiv.2410.00623

[11] E. C. Silva, R. Rodrigues, J. Oliveira, D. Boechat, and C. Tavares, "Evaluating Test Quality in GitHub Repositories : A Comparative Analysis of CI / CD Practices Using GitHub Actions"2024, doi: https://doi.org/10.5753/vem.2024.3842.

[12] M. Soliman, M. Albonico, and I. Malavolta, "Mining software repositories for software architecture — A systematic mapping study," *Inf. Softw. Technol.*, vol. 181, no. January, p. 107677, 2025, doi: 10.1016/j.infsof.2025.107677.

[13] Z. Codabux, F. Fard, R. Verdecchia, F. Palomba, and S. E. Jan, "Teaching Mining Software Repositories," pp. 1–41, 2025, doi: 10.48550/arXiv.2501.01903.