

IMPLEMENTASI CONTINUOUS INTEGRATION DAN CONTINUOUS DELIVERY PADA PROYEK APLIKASI CHART LYRIC EDITOR

(Implementation of Continuous Integration and Continuous Delivery on Chart Lyric Editor Application Project)

Ghifary Ahada Azra^[1], Sri Endang Anjarwani^[1], Moh. Ali Albar^[1]

^[1]Dept Informatics Engineering, Mataram University
Jl. Majapahit 62, Mataram, Lombok NTB, INDONESIA

Email: masterghifary6403@gmail.com, [endang, mohalialbar]@unram.ac.id

Abstract

Modern software development paradigms demand iterative code integration and delivery to ensure release speed and system stability. However, reliance on manual processes often hinders these standards due to error proneness. In the context of the Chart Lyric Editor application, development stages are still performed manually, leading to human error risks and release artifact inconsistencies. This research aims to address these issues by implementing Continuous Integration (CI) and Continuous Delivery (CD) using the Design Science Research Methodology (DSRM) approach. The solution is designed using GitHub Actions with a separated workflow strategy executed on a Windows environment. This strategy divides the process into two parts: an automated integration workflow (CI) triggered by code changes and a controlled release workflow (CD) via manual triggers. Evaluation was conducted by comparing release cycle efficiency and build consistency between the automated pipeline and the previous manual procedure. The results demonstrate that the CI/CD implementation significantly improves release cycle efficiency, reducing the total process duration from 17 minutes 51 seconds manually to an average of 1 minute 47 seconds. Furthermore, this automation establishes a more structured development workflow, minimizes potential human errors, and maintains the stability and consistency of application release artifacts.

Keywords: Continuous Integration, Continuous Delivery, GitHub Actions, Design Science Research Methodology, DevOps.

1. PENDAHULUAN

Perkembangan teknologi perangkat lunak mendorong proses pengembangan aplikasi agar dilakukan secara lebih cepat, terstruktur, dan konsisten. Dalam praktik pengembangan modern, pendekatan *DevOps* digunakan untuk mengintegrasikan aktivitas pengembangan dan operasional agar proses pembuatan, pengujian, serta distribusi perangkat lunak dapat berjalan lebih efisien. Pendekatan tersebut menjadi relevan karena proses pengembangan perangkat lunak tidak hanya berfokus pada penulisan kode, tetapi juga mencakup pengujian, pembuatan artefak aplikasi, dan distribusi hasil rilis kepada pengguna [1].

Proses pengembangan aplikasi yang masih dilakukan secara manual dapat menimbulkan permasalahan pada efektivitas dan efisiensi kerja. Pengembangan tradisional umumnya memisahkan proses perancangan, pengujian, dan *deployment*, sehingga setiap pembaruan aplikasi harus melalui

tahapan berulang yang memerlukan waktu lebih lama. Kondisi tersebut juga dapat meningkatkan risiko keterlambatan rilis, ketidaksesuaian lingkungan eksekusi, dan kesalahan manusia pada saat proses pengujian maupun distribusi aplikasi [2].

Permasalahan serupa ditemukan pada proyek aplikasi *desktop Chart Lyric Editor*. Aplikasi ini dikembangkan menggunakan bahasa pemrograman *Python* dengan *framework PyQt5* dan digunakan sebagai alat bantu untuk menyunting skrip lirik pada berkas lagu *custom* dalam komunitas *Clone Hero*. Sebelum penerapan otomatisasi, proses pengembangan aplikasi mulai dari pengujian, kompilasi (*build*), hingga rilis masih dilakukan secara manual di lingkungan lokal. Berdasarkan hasil observasi pada proyek ini, prosedur manual memicu terjadinya kesalahan manusia (*human error*), seperti kelalaian pengembang dalam membersihkan *cache* saat mengeksekusi *PyInstaller* secara berulang, yang menyebabkan berkas eksekusi hasil *build* menjadi korup atau tidak berfungsi. Selain itu, pengemasan dan

penamaan versi rilis yang dilakukan secara manual rentan mengalami kesalahan ketik (*typo*), yang berdampak pada inkonsistensi artefak rilis aplikasi.

Salah satu pendekatan yang dapat digunakan untuk mengatasi permasalahan tersebut adalah penerapan *Continuous Integration* dan *Continuous Delivery* (CI/CD). *Continuous Integration* memungkinkan setiap perubahan kode diverifikasi melalui proses integrasi dan pengujian otomatis, sehingga kesalahan dapat terdeteksi lebih awal sebelum perubahan digabungkan ke dalam basis kode utama. Sementara itu, *Continuous Delivery* mendukung proses penyediaan artefak aplikasi agar berada dalam kondisi siap rilis setelah melewati tahapan pengujian dan pemeriksaan kualitas [3].

GitHub Actions merupakan salah satu platform yang dapat digunakan untuk mengimplementasikan CI/CD karena terintegrasi langsung dengan repositori *GitHub* dan mendukung pembuatan alur kerja otomatis. Penggunaan *GitHub Actions* dalam proses CI/CD memungkinkan pengembang menjalankan pengujian, *build*, dan *deployment* secara otomatis ketika terjadi perubahan pada kode sumber [4]. Dengan demikian, *GitHub Actions* dapat digunakan untuk membangun alur kerja yang lebih terstandar dan mengurangi ketergantungan terhadap prosedur manual.

Berbeda dengan penelitian-penelitian terdahulu yang mayoritas berfokus pada implementasi *Continuous Integration* dan *Continuous Delivery* untuk aplikasi berbasis web (menggunakan kontainer seperti *Docker*), penelitian ini menawarkan kontribusi kebaruan dengan menerapkan otomatisasi *DevOps* secara spesifik pada lingkungan pengembangan aplikasi desktop berbasis antarmuka grafis (GUI) *PyQt5*. Fokus utama tidak hanya pada integrasi kode, melainkan juga pada bagaimana *GitHub Actions* dapat dikonfigurasi pada *environment Windows* untuk menangani kompilasi artefak desktop secara terisolasi dan mendistribusikannya secara otomatis.

Penelitian ini menggunakan pendekatan *Design Science Research Methodology* (DSRM) sebagai kerangka kerja dalam merancang, mengimplementasikan, dan mengevaluasi artefak berupa *pipeline* CI/CD. DSRM sesuai digunakan dalam penelitian pengembangan sistem karena berfokus pada penciptaan artefak untuk menyelesaikan permasalahan tertentu melalui tahapan identifikasi masalah, definisi tujuan solusi, desain dan pengembangan, demonstrasi, evaluasi, serta komunikasi [5]. Dalam konteks penelitian ini, artefak yang dikembangkan berupa konfigurasi *workflow*

GitHub Actions untuk mengotomatisasi proses pengujian, *build*, dan distribusi aplikasi *Chart Lyric Editor*.

Berdasarkan permasalahan tersebut, penelitian ini bertujuan untuk merancang dan mengimplementasikan *pipeline* CI/CD berbasis *GitHub Actions* pada proyek aplikasi *Chart Lyric Editor*. Penelitian ini juga bertujuan untuk mengevaluasi perbandingan efisiensi waktu dan konsistensi proses rilis antara prosedur manual dan prosedur otomatis. Hasil penelitian diharapkan dapat menunjukkan bahwa penerapan CI/CD mampu menghasilkan alur pengembangan yang lebih terstruktur, mengurangi potensi *human error*, serta meningkatkan konsistensi artefak rilis aplikasi.

2. TINJAUAN PUSTAKA

2.1. Penelitian Terkait

Beberapa penelitian terdahulu telah mengkaji penerapan *DevOps* dan CI/CD dalam berbagai studi kasus pengembangan perangkat lunak. Untuk mengidentifikasi posisi dan kebaruan penelitian ini (*research gap*), rangkuman perbandingan antara penelitian terdahulu dengan penelitian ini disajikan pada Tabel 1. Parameter perbandingan difokuskan pada objek penelitian, platform/alat yang digunakan, metode, serta hasil yang dicapai.

Tabel 1. Perbandingan Penelitian Terkait

Penelitian	Objek Penelitian	Metode & Alat	Hasil Utama
Tohirin et al. [6]	Aplikasi Web e-Skrining Covid-19	<i>DevOps</i> , <i>GitLab</i>	Otomatisasi integrasi & kelayakan kode.
Paramudita et al. [1]	Aplikasi <i>Mobile Android</i> .	<i>DevOps</i> , CI/CD	Peningkatan stabilitas aplikasi <i>mobile</i> .
Jaeni et al. [2]	<i>Performance Testing</i> Infrastruktur	CI/CD, <i>Jenkins</i> , <i>Docker</i>	Mempercepat feedback antar tim.
Apriliyanto et al. [4]	Aplikasi Web <i>Landing Page</i> UMKM.	CI/CD, <i>GitHub Actions</i>	<i>Deployment</i> web berjalan otomatis.
Zulhaki m & Kurniawan [7]	Aplikasi Web berbasis Kontainer.	CI/CD, <i>GitHub Actions</i>	Proses <i>deployment</i> lebih cepat dari <i>Jenkins</i> .

Enda et al. [3]	Aplikasi Web Profil Karang Taruna	CI, <i>GitHub</i>	Deteksi kesalahan (<i>bug</i>) lebih awal.
Penelitian ini	Aplikasi <i>Desktop Chart Lyric Editor</i> .	CI/CD, <i>GitHub Actions</i>	Efisiensi siklus rilis dan konsistensi <i>build file executable</i> di <i>Windows</i> .

Berdasarkan Tabel 1, sebagian besar penelitian terdahulu mengenai implementasi *DevOps* dan CI/CD difokuskan pada ekosistem aplikasi berbasis web dan aplikasi perangkat *mobile*, serta sering kali bergantung pada lingkungan kontainer seperti *Docker*. Sangat minim penelitian yang membahas penerapan otomatisasi CI/CD pada aplikasi desktop berskala kecil yang menggunakan antarmuka grafis (GUI). Oleh karena itu, penelitian ini mengisi celah tersebut dengan mengimplementasikan *GitHub Actions* secara spesifik pada *environment runner Windows* untuk memfasilitasi pengujian antarmuka (*PyQt5*) dan kompilasi (*build*) berkas eksekusi tunggal (.exe), yang sebelumnya dilakukan secara manual oleh pengembang.

2.2. Dasar Teori

2.2.1. *DevOps*

DevOps merupakan pendekatan pengembangan perangkat lunak yang mengintegrasikan aktivitas pengembangan dan operasional agar proses pembuatan, pengujian, serta distribusi aplikasi dapat dilakukan secara lebih cepat dan andal [1]. Dalam konteks pengembangan perangkat lunak, *DevOps* tidak hanya dipahami sebagai penggunaan alat otomatisasi, tetapi juga sebagai pendekatan kerja yang menekankan integrasi proses, pengurangan pekerjaan manual, dan peningkatan konsistensi dalam siklus rilis aplikasi.

Sebagai contoh penerapannya pada proyek aplikasi *desktop*, *DevOps* diwujudkan melalui otomatisasi proses kompilasi kode sumber menjadi berkas eksekusi menggunakan modul kompilasi seperti *PyInstaller*, serta pengujian secara otomatis pada *environment* sistem operasi target (misalnya menggunakan *runner Windows*) setiap kali terjadi perubahan kode. Hal ini menjadi bentuk nyata penggantian praktik tradisional di mana pengembang biasanya harus melakukan *packaging* dan pengujian secara manual di komputer lokal.

Pada penelitian ini, konsep *DevOps* digunakan sebagai dasar untuk menyusun alur kerja otomatis yang menghubungkan proses perubahan kode, pengujian, pembuatan artefak aplikasi, dan distribusi hasil rilis. Penerapan konsep tersebut diharapkan dapat mengurangi ketergantungan terhadap prosedur manual yang sebelumnya digunakan pada proyek *Chart Lyric Editor*.

2.2.2. *Continuous Integration* dan *Continuous Delivery*

Continuous Integration merupakan praktik pengembangan perangkat lunak yang mendorong pengembang untuk mengintegrasikan perubahan kode secara berkala ke dalam repositori utama. Setiap perubahan kode kemudian diverifikasi melalui proses otomatis, seperti *build* dan pengujian, untuk mendeteksi kesalahan integrasi lebih awal [3]. Praktik ini penting karena kesalahan yang ditemukan lebih awal cenderung lebih mudah diperbaiki dibandingkan kesalahan yang baru diketahui setelah proses rilis.

Continuous Delivery merupakan praktik yang memastikan perangkat lunak selalu berada dalam kondisi siap rilis setelah melewati proses pengujian dan pemeriksaan kualitas [3]. Dalam penelitian ini, konsep *Continuous Delivery* diterapkan melalui *workflow build* yang dijalankan secara terkontrol untuk menghasilkan artefak aplikasi. Dengan demikian, proses rilis tidak lagi sepenuhnya bergantung pada tahapan manual yang berulang.

CI/CD dapat membantu mengatasi masalah efektivitas dan efisiensi pada proses pengembangan tradisional. Penerapan CI/CD memungkinkan proses pengujian dan rilis dilakukan secara otomatis, sehingga pengembang memperoleh umpan balik lebih cepat terhadap perubahan kode yang dilakukan [2]. Dalam penelitian ini, penerapan CI/CD difokuskan pada pemisahan alur kerja pengujian dan alur kerja *build* agar proses validasi kode dan distribusi aplikasi dapat berjalan lebih sistematis.

2.2.3. *GitHub Actions*

GitHub Actions merupakan platform otomatisasi yang terintegrasi dengan repositori *GitHub* dan dapat digunakan untuk membuat *workflow* berdasarkan peristiwa tertentu pada repositori. Dalam konteks CI/CD, *GitHub Actions* memungkinkan pengembang menjalankan pengujian, *build*, dan *deployment* secara

otomatis ketika terjadi perubahan kode sumber [4]. Karakteristik tersebut menjadikan *GitHub Actions* sesuai digunakan pada proyek yang telah memanfaatkan *GitHub* sebagai sistem pengelolaan kode sumber.

Pada penelitian ini, *GitHub Actions* digunakan untuk membangun dua alur kerja utama, yaitu *workflow test* dan *workflow build*. *Workflow test* berfungsi menjalankan pengujian otomatis ketika terjadi perubahan kode, sedangkan *workflow build* digunakan untuk menghasilkan artefak rilis aplikasi secara terkontrol. Pemisahan kedua alur kerja tersebut bertujuan agar proses validasi dan proses distribusi aplikasi dapat dijalankan sesuai kebutuhan pengembangan.

2.2.4. Design Science Research Methodology

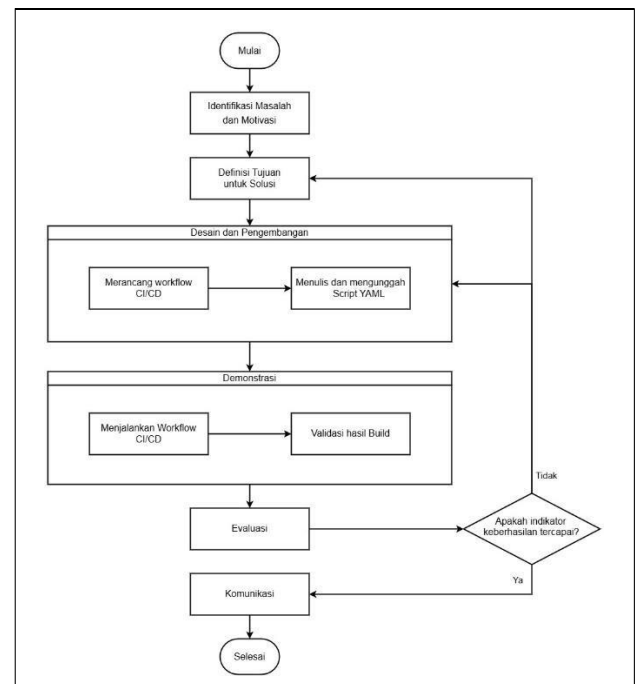
Design Science Research Methodology (DSRM) merupakan pendekatan penelitian yang berorientasi pada perancangan dan pengembangan artefak untuk menyelesaikan suatu permasalahan. DSRM mencakup enam kegiatan utama, yaitu identifikasi masalah dan motivasi, definisi tujuan solusi, desain dan pengembangan, demonstrasi, evaluasi, serta komunikasi [5]. Pendekatan ini sesuai digunakan pada penelitian yang menghasilkan artefak teknis, seperti model, metode, sistem, atau konfigurasi alur kerja.

Dalam penelitian ini, DSRM digunakan sebagai kerangka kerja untuk merancang dan mengevaluasi artefak berupa *pipeline CI/CD* berbasis *GitHub Actions*. Tahap identifikasi masalah digunakan untuk memahami kendala proses manual pada pengembangan *Chart Lyric Editor*. Tahap desain dan pengembangan digunakan untuk menyusun konfigurasi *workflow*, sedangkan tahap evaluasi digunakan untuk membandingkan efisiensi dan konsistensi proses rilis antara prosedur manual dan otomatis.

3. METODE PENELITIAN

Tahapan penelitian ini disusun berdasarkan *Design Science Research Methodology* (DSRM), yaitu metode yang berorientasi pada perancangan dan evaluasi artefak untuk menyelesaikan suatu permasalahan tertentu. Dalam penelitian ini, artefak yang dikembangkan berupa *pipeline CI/CD* berbasis *GitHub Actions* untuk mengotomatisasi proses pengujian,

build, dan rilis aplikasi *Chart Lyric Editor*. Alur tahapan penelitian dengan pendekatan DSRM ditunjukkan pada Gambar 3.1.



Gambar 1. Tahapan Penelitian Dengan Metode DSRM

Gambar 1 mengilustrasikan tahapan penelitian yang diadopsi dari kerangka kerja DSRM, yang dirancang secara sekuensial namun bersifat iteratif. Alur penelitian dimulai dari identifikasi masalah, definisi tujuan dan solusi, desain dan pengembangan, demonstrasi, evaluasi, hingga komunikasi. Di dalam siklus DSRM ini, terdapat mekanisme perulangan pada tahap evaluasi. Berdasarkan hasil pengujian artefak (*pipeline CI/CD*), jika indikator keberhasilan belum tercapai, maka dilakukan iterasi perbaikan sebanyak dua kali, meliputi penyesuaian parameter konfigurasi script YAML, pengaturan caching PyInstaller, dan perbaikan penanganan path modul pengujian hingga seluruh skrip dapat berjalan stabil tanpa kegagalan. Setelah seluruh kriteria terpenuhi, alur dilanjutkan ke tahap komunikasi.

3.1. Tahapan Penelitian

1. Identifikasi Masalah

Tahap identifikasi masalah dilakukan dengan meninjau proses pengembangan aplikasi *Chart Lyric Editor* yang sebelumnya masih dijalankan secara manual. Proses tersebut mencakup pengujian aplikasi, pembuatan berkas eksekusi, pengemasan hasil *build*, serta publikasi artefak rilis. Berdasarkan hasil observasi, prosedur manual berpotensi menimbulkan

human error, ketidakkonsistenan hasil *build*, serta keterlambatan dalam siklus rilis aplikasi. Permasalahan tersebut menjadi dasar kebutuhan terhadap sistem otomatisasi yang mampu menjaga konsistensi alur pengembangan.

2. Definisi Tujuan

Tahap definisi tujuan dilakukan dengan merumuskan solusi teknis berdasarkan permasalahan yang telah diidentifikasi. Tujuan utama dari penelitian ini adalah merancang dan mengimplementasikan pipeline CI/CD menggunakan *GitHub Actions* untuk mengotomatisasi proses pengujian, *build*, dan publikasi artefak aplikasi. Selain itu, tahap ini juga menetapkan indikator keberhasilan artefak, yaitu keberhasilan eksekusi *workflow*, efisiensi waktu proses, kestabilan hasil pengujian, dan konsistensi artefak rilis yang dihasilkan.

3. Desain dan Pengembangan

Tahap desain dan pengembangan merupakan proses perancangan artefak CI/CD serta implementasinya ke dalam konfigurasi teknis. Pada tahap desain, sistem dirancang dengan membagi *pipeline* menjadi dua mekanisme utama, yaitu *Workflow Test* sebagai proses CI dan *Workflow Build* sebagai proses CD. Pada tahap pengembangan, rancangan tersebut direalisasikan dalam bentuk konfigurasi YAML pada repositori *GitHub* agar dapat dijalankan melalui *GitHub Actions*.

4. Demonstrasi

Tahap demonstrasi dilakukan dengan menjalankan artefak *pipeline* pada lingkungan pengembangan aplikasi *Chart Lyric Editor*. Demonstrasi *Workflow Test* dilakukan melalui pemicu otomatis saat terjadi perubahan kode pada repositori, sedangkan demonstrasi *Workflow Build* dilakukan melalui pemicu manual. Tahap ini bertujuan untuk memastikan bahwa rancangan *pipeline* dapat menjalankan pengujian otomatis, membangun aplikasi, dan menghasilkan artefak rilis sesuai dengan alur yang telah dirancang.

5. Evaluasi

Tahap evaluasi dilakukan untuk menilai efektivitas artefak CI/CD dalam menyelesaikan permasalahan proses manual. Evaluasi dilakukan dengan membandingkan proses manual dan proses otomatis berdasarkan dua kategori metrik utama, yaitu *time-based metrics* dan *quality-based metrics*.

a. *Time-based Metrics*: Digunakan untuk mengukur durasi, kecepatan, dan waktu siklus eksekusi dari setiap tahapan di dalam pipeline (pengujian, *build*, dan rilis aplikasi) [2]. Adapun batas keberhasilan (*success criteria*) yang ditetapkan pada metrik ini adalah reduksi durasi total siklus rilis otomatis harus mampu mencapai efisiensi yang signifikan.

b. *Quality-based Metrics*: Digunakan untuk menilai kualitas, stabilitas, dan konsistensi proses yang dihasilkan, mencakup tingkat keberhasilan eksekusi *workflow*, status pengujian otomatis, ketersediaan artefak hasil *build*, serta stabilitas hasil rilis aplikasi. Merujuk pada *konsep quality-based metrics* dari Apriliyanto et al. [4], penelitian ini merumuskan secara mandiri batas keberhasilan kuantitatif yang disesuaikan dengan kebutuhan kompilasi aplikasi *desktop*, yang meliputi:

- *Build Success Rate* mencapai 100% dari total iterasi pengujian (berhasil mengompilasi file secara utuh tanpa korup).
- Tingkat kelulusan 100% pada rangkaian pengujian otomatis.
- Eliminasi total kesalahan penulisan pada penomoran versi rilis berkat penggunaan variabel dinamis sistem.

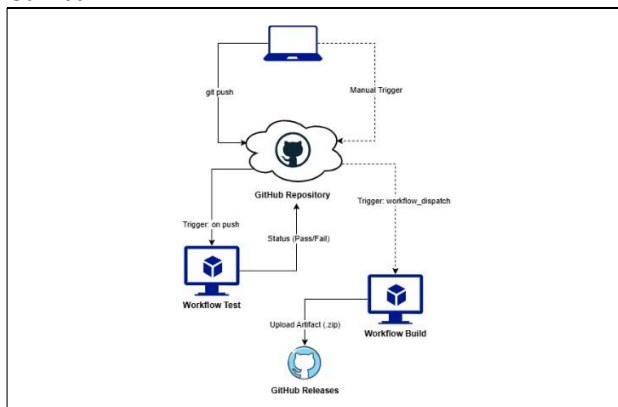
Dengan menggunakan indikator dan batas keberhasilan terukur tersebut, evaluasi tidak hanya menilai kecepatan proses, tetapi juga memastikan apakah otomatisasi mampu menghasilkan alur kerja yang stabil dan konsisten dibandingkan prosedur manual.

6. Komunikasi

Tahap komunikasi dilakukan dengan menyajikan hasil perancangan, implementasi, dan evaluasi artefak ke dalam bentuk artikel ilmiah. Pada tahap ini, hasil penelitian dikomunikasikan melalui uraian mengenai rancangan arsitektur CI/CD, implementasi *workflow*, hasil demonstrasi, serta pembahasan evaluasi. Tahap komunikasi juga digunakan untuk menjelaskan kontribusi penelitian, yaitu penerapan CI/CD berbasis *GitHub Actions* pada aplikasi desktop *Chart Lyric Editor* sebagai upaya meningkatkan efisiensi dan konsistensi proses pengembangan aplikasi.

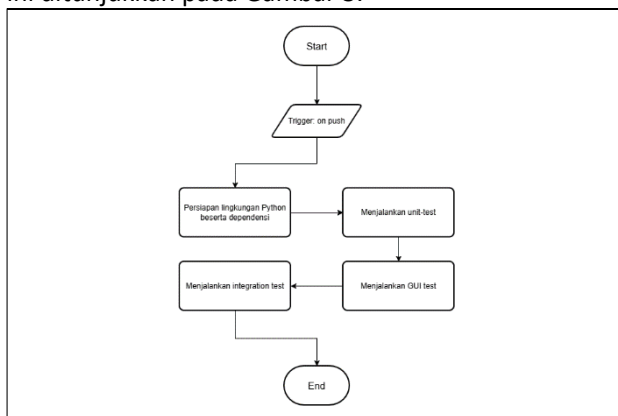
3.2. Desain Arsitektur CI/CD

Perancangan arsitektur CI/CD pada proyek ini diawali dengan mendefinisikan aliran data dari lingkungan lokal pengembang menuju server eksekusi *GitHub Actions*. Arsitektur ini memisahkan proses operasional menjadi dua mekanisme utama yang berjalan pada lingkungan eksekusi (*runner*) untuk menjamin kompatibilitas penuh aplikasi *desktop* berbasis *Windows*. Kedua mekanisme tersebut adalah mekanisme integrasi otomatis (*Workflow Test*) dan mekanisme rilis terkontrol (*Workflow Build*). Desain arsitektur sistem secara keseluruhan dapat dilihat pada Gambar 2.



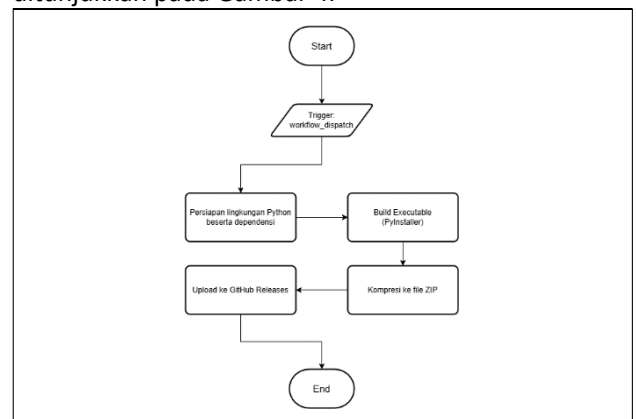
Gambar 2. Diagram Arsitektur Sistem CI/CD

Mekanisme integrasi otomatis atau *Workflow Test* bertindak sebagai proses Continuous Integration (CI). Proses ini dirancang untuk mendeteksi perubahan kode di repositori utama melalui pemicu otomatis, yang menginstruksikan sistem untuk menyiapkan lingkungan *Python* dan menjalankan serangkaian pengujian. Pengujian dilakukan secara bertahap meliputi *Unit Test*, *GUI Test*, dan *Integration Test*. Apabila seluruh pengujian berhasil dieksekusi tanpa kesalahan, umpan balik status kesuksesan akan dikirimkan kembali ke repositori. Alur logika pengujian ini ditunjukkan pada Gambar 3.



Gambar 3. Flowchart Workflow Test

Mekanisme rilis terkontrol atau *Workflow Build* bertindak sebagai proses *Continuous Delivery* (CD). Berbeda dengan proses pengujian, alur kerja ini tidak dijalankan secara otomatis, melainkan menggunakan pemicu manual guna memberikan kendali penuh kepada pengembang dalam menentukan kelayakan rilis suatu versi aplikasi. Setelah pemicu diaktifkan, sistem akan melakukan kompilasi kode sumber menggunakan *PyInstaller* untuk menghasilkan file eksekusi (*executable*). File hasil *build* tersebut kemudian dikompresi ke dalam format ZIP dan diunggah secara otomatis ke halaman *GitHub Releases* agar dapat diunduh oleh pengguna. Alur rilis aplikasi ini ditunjukkan pada Gambar 4.



Gambar 4. Flowchart Workflow Build

Seluruh rancangan alur kerja tersebut kemudian ditranslasikan ke dalam parameter teknis berupa *jobs* dan *steps* yang dapat diproses oleh *GitHub Actions*. Meskipun kedua *workflow* berjalan pada lingkungan *Windows* yang serupa, spesifikasi langkahnya dibedakan berdasarkan tujuan utamanya; validasi logika kode untuk *Workflow Test* dan manajemen artefak untuk *Workflow Build*. Rincian spesifikasi teknis dari rancangan arsitektur ini dirangkum pada Tabel 2.

Tabel 2. Rancangan Job dan Langkah (Steps) pada Pipeline CI/CD

Workflow	Langkah (Steps)	Deskripsi
Test	Set up Python	Mempersiapkan lingkungan <i>Python</i> .
	Install dependencies	Menginstal <i>library</i> dari "requirements.txt"
	Run unit-test	Menjalankan pengujian <i>unit</i> .
	Run GUI test	Menjalankan pengujian <i>GUI</i> .

	<i>Run Integration test</i>	Menjalankan pengujian integrasi.
<i>Build</i>	<i>Set up Python</i>	Mempersiapkan lingkungan <i>Python</i> .
	<i>Install dependencies</i>	Menginstal <i>library</i> dari " <i>requirements.txt</i> ".
	<i>Build executable</i>	Menjalankan <i>build executable</i> .
	<i>Compress to ZIP</i>	Mengompres hasil <i>build</i> ke file ZIP.
	<i>Upload Release</i>	Mengupload <i>file</i> ke halaman rilis.

4. HASIL DAN PEMBAHASAN

Bagian ini memaparkan hasil dari implementasi dan evaluasi *pipeline Continuous Integration* (CI) dan *Continuous Delivery* (CD) berbasis *GitHub Actions* yang telah diterapkan pada lingkungan pengembangan proyek aplikasi *Chart Lyric Editor*. Sesuai dengan tahapan *Design Science Research Methodology* (DSRM), pemaparan pada bab ini dibagi menjadi tiga fokus utama, yaitu realisasi teknis konfigurasi *workflow* (implementasi), pembuktian operasional sistem (demonstrasi), serta pengukuran tingkat keberhasilan artefak (evaluasi). Evaluasi hasil penelitian dititikberatkan pada analisis komparatif antara prosedur pengembangan manual yang berjalan sebelumnya dengan infrastruktur otomasi yang telah dibangun. Analisis efektivitas solusi tersebut diukur menggunakan dua indikator metrik utama, yakni *time-based metrics* untuk membuktikan peningkatan efisiensi durasi siklus rilis, dan *quality-based metrics* untuk menilai peningkatan stabilitas serta konsistensi *output* dari pengembangan aplikasi.

4.1. Implementasi Pipeline CI/CD

Tahap implementasi merupakan perwujudan dari rancangan konseptual arsitektur sistem ke dalam infrastruktur otomasi berbasis *GitHub Actions*. Konfigurasi awal pada pipeline difokuskan pada persiapan lingkungan eksekusi (*runner*), di mana sistem dikonfigurasi menggunakan lingkungan *Windows*. Pemilihan lingkungan ini krusial untuk menjamin kompatibilitas penuh dari proses *build* dan pengujian aplikasi *Chart Lyric Editor* yang secara spesifik ditargetkan beroperasi pada sistem operasi *Windows*. Pada tahap inisialisasi ini, sistem diinstruksikan untuk menggunakan *Python 3.10* dan melakukan instalasi seluruh dependensi aplikasi secara otomatis dengan merujuk pada file "*requirements.txt*" yang

berada di repositori. Selain dependensi utama, *pipeline* juga menginstal pustaka tambahan berupa *pytest* dan *PyInstaller* yang akan digunakan pada tahap pengujian dan kompilasi *file* eksekusi.

Infrastruktur *Continuous Integration* (CI) diimplementasikan melalui mekanisme *Workflow Test* yang berfungsi untuk memvalidasi integritas kode sumber. Alur kerja ini dikonfigurasi menggunakan pemicu (*event trigger*) yang berjalan secara otomatis setiap kali terdapat aktivitas pembaruan kode, baik melalui instruksi *push* maupun *pull request* yang mengarah ke *branch* utama. Setelah terpicu, *pipeline* akan mengeksekusi pengujian otomatis secara sekuensial yang terbagi ke dalam tiga tingkatan: *Unit Test* untuk memvalidasi fungsi fundamental aplikasi, *GUI Test* untuk memverifikasi responsivitas elemen antarmuka pengguna berbasis *PyQt5*, dan *Integration Test* untuk menguji stabilitas interaksi antar-modul. Mekanisme ini memastikan bahwa setiap perubahan tervalidasi secara komprehensif sebelum diintegrasikan secara permanen ke dalam sistem.

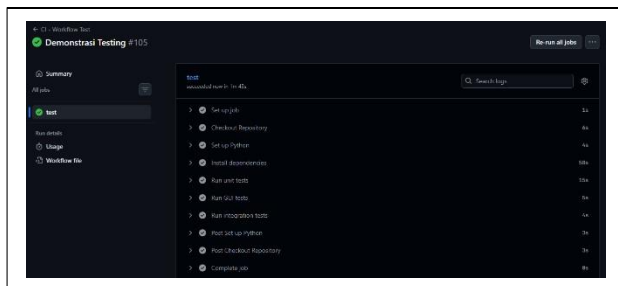
Sementara itu, mekanisme *Continuous Delivery* (CD) diimplementasikan melalui *Workflow Build* untuk menangani proses kompilasi dan distribusi rilis. Berbeda dengan *Workflow Test*, alur kerja ini menggunakan pemicu manual. Pendekatan ini diimplementasikan guna memberikan kendali penuh kepada pengembang dalam menentukan kelayakan sebuah versi sebelum didistribusikan kepada pengguna. Saat *workflow* ini dijalankan, modul *PyInstaller* akan mengompilasi kode sumber menjadi *file* eksekusi secara otomatis dan menimpa *build* lama tanpa memerlukan konfirmasi manual. Aplikasi yang telah berhasil dibangun kemudian dikompresi ke dalam format arsip ZIP. Proses diakhiri dengan pendistribusian arsip tersebut secara otomatis ke halaman *GitHub Releases* menggunakan penamaan *tag* rilis yang dikombinasikan dengan variabel *run number*, sehingga menghasilkan penomoran versi yang unik, terstruktur, dan konsisten.

Selain itu, mitigasi dan penanganan kegagalan (*error handling*), *pipeline* menerapkan parameter "*-clean*" pada *PyInstaller* untuk membersihkan *cache* secara otomatis guna mencegah rusaknya artefak *build*, serta menggunakan argumen "*--tb=short*" pada *pytest* untuk meringkas *log traceback* galat.

Sistem menerapkan mekanisme *fail-fast* yang menghentikan eksekusi secara otomatis dan mengirimkan notifikasi kepada pengembang apabila terjadi kegagalan pada tahap *workflow*, sehingga proses *debugging* dapat segera dilakukan.

4.2. Demonstrasi Pipeline CI/CD

Tahap demonstrasi dilakukan untuk membuktikan bahwa arsitektur infrastruktur otomatis yang telah diimplementasikan mampu beroperasi dengan stabil dan berhasil mentransformasi prosedur manual dalam skenario pengembangan nyata. Pengujian terhadap mekanisme *Continuous Integration* (CI) disimulasikan melalui pengiriman pembaruan kode sumber (*commit* dan *push*) ke dalam *branch* utama pada repositori. Sesuai dengan konfigurasi pemicu otomatis, sistem *GitHub Actions* langsung merespons aktivitas tersebut dengan mengeksekusi *Workflow Test*. Hasil simulasi menunjukkan bahwa pipeline berhasil memvalidasi integritas kode secara komprehensif, ditandai dengan keberhasilan penyelesaian seluruh tahapan pengujian tanpa adanya kesalahan atau *error*. Keberhasilan ini membuktikan bahwa pembaruan kode tidak memicu regresi pada fungsionalitas aplikasi. Bukti dari *log* eksekusi pengujian ini disajikan pada Gambar 5.



Gambar 5. Log Eksekusi *Workflow Test*

Setelah kode dinyatakan valid dan aman, demonstrasi dilanjutkan pada pengujian arsitektur *Continuous Delivery* (CD) melalui eksekusi *Workflow Build*. Sesuai dengan desain sistem yang membutuhkan kontrol pengembang. Sistem berhasil merespons instruksi tersebut dengan mengompilasi kode sumber menjadi berkas eksekusi (*executable*) menggunakan *PyInstaller*, mengompresinya ke dalam format arsip ZIP, dan secara otomatis mempublikasikan berkas distribusi tersebut ke halaman *GitHub Releases*.

Validasi akhir dilakukan untuk memastikan ketersediaan dan integritas fungsional dari artefak

yang dihasilkan. Peninjauan pada halaman rilis repositori yang ditunjukkan pada Gambar 6 membuktikan bahwa sistem secara presisi berhasil mempublikasikan catatan rilis baru dengan penamaan tag versi yang terstruktur. Untuk mengonfirmasi integritas hasil kompilasi sistem, berkas arsip tersebut diunduh, diekstrak, dan dieksekusi secara langsung pada lingkungan *Windows* lokal. Aplikasi terbukti dapat beroperasi secara normal dengan antarmuka grafis yang ditampilkan secara utuh tanpa terjadinya *crash* atau *startup error*. Hasil ini memberikan validasi empiris bahwa pipeline yang dikembangkan tidak menghasilkan berkas korup dan mampu menjaga stabilitas operasional perangkat lunak pada setiap siklus distribusi. Namun, validasi fungsionalitas artefak pada tahap demonstrasi ini masih terbatas dilakukan pada *environment* sistem operasi *Windows* lokal pengembang. Pengujian langsung oleh pengguna eksternal dalam skala luas belum dicakup dalam evaluasi ini, sehingga menjadi salah satu keterbatasan penelitian yang dapat disempurnakan pada pengembangan selanjutnya.



Gambar 6. Hasil Publikasi Artefak *Build* pada Halaman *GitHub Releases*

4.3. Analisis dan Evaluasi Hasil

Evaluasi hasil bertujuan untuk menilai tingkat efektivitas arsitektur CI/CD yang telah diimplementasikan berdasarkan dua indikator metrik utama, yaitu *time-based metrics* untuk mengukur efisiensi waktu siklus rilis dan *quality-based metrics* untuk menilai konsistensi luaran *pipeline* [4]. Pengukuran efisiensi waktu dilakukan untuk membuktikan dampak teknis dari transisi prosedur manual menuju otomatisasi infrastruktur. Data *baseline* prosedur manual diperoleh dari satu sampel pencatatan waktu operasional lokal pengembang secara *real-time*. Sementara itu, untuk membuktikan keabsahan data otomatisasi serta meminimalisir

dampak fluktuasi beban server *cloud runner*, *pipeline* CI/CD dieksekusi dan diuji sebanyak 10 kali pengulangan (*runs*) secara independen [8].

Keaslian dan rekam *log* dari 10 sampel pengujian tersebut terekam secara otomatis pada riwayat eksekusi *GitHub Actions* di repositori proyek. Data yang disajikan pada Tabel 3 merupakan nilai rerata (*mean*) dari 10 kali percobaan tersebut, di mana durasi murni proses pengujian (*testing*) berkisar antara 18–42 detik, kompilasi (*build*) berkas .exe berdurasi 50–75 detik, dan publikasi rilis memakan waktu 11–14 detik. Rincian perbandingan efisiensi waktu siklus rilis disajikan pada Tabel 3.

Tabel 3. Perbandingan Efisiensi Waktu Siklus Rilis

Tahapan Pengembangan	Durasi Manual	Durasi Otomatis (Rata-rata)
Pengujian (<i>Testing</i>)	12 menit 2 detik	28 detik
Kompilasi (<i>Build</i>)	3 menit 30 detik	1 menit 7 detik
Kompresi dan Publikasi	2 menit 19 detik	12 detik
Total Waktu Siklus	17 menit 51 detik	1 menit 47 detik

Selain peningkatan kecepatan siklus rilis, evaluasi menggunakan *quality-based metrics* menunjukkan bahwa solusi otomasi berhasil memitigasi potensi *human error* secara kuantitatif. Berdasarkan pengujian sebanyak 10 kali iterasi eksekusi *pipeline*, tercatat *Build Success Rate* mencapai 100% (10 dari 10 eksekusi berhasil mengompilasi kode sumber menjadi berkas .exe yang valid tanpa adanya kegagalan *pipeline*). Selain itu, *Defect Rate* terkait kesalahan penomoran versi atau kegagalan rilis bernilai 0%, karena seluruh penamaan *tag* dikelola secara otomatis oleh variabel sistem *GitHub Actions*.

. Pada aspek integritas pengujian, penerapan *pipeline* memastikan bahwa seluruh rangkaian uji coba berjalan secara sekuensial dan menyeluruh sebelum proses rilis dapat dipicu, sehingga meniadakan risiko adanya fitur yang terlewat untuk dievaluasi oleh pengembang.

Konsistensi tahap kompilasi dan distribusi rilis juga mengalami peningkatan pesat. Sistem menggunakan parameter pembersihan *cache* otomatis dan berjalan di lingkungan runner yang steril, sehingga

menghilangkan risiko berkas korup akibat residu kompilasi sebelumnya. Selain itu, standarisasi penomoran versi diatur secara dinamis memanfaatkan variabel sistem *GitHub*, menutup sepenuhnya peluang terjadinya kesalahan pengetikan (*typo*) atau duplikasi rilis. Secara keseluruhan, *pipeline* CI/CD terbukti menciptakan siklus distribusi yang stabil, konsisten, dan mengurangi secara signifikan ketergantungan pada prosedur manual. Perbandingan komprehensif terkait parameter stabilitas antara kedua prosedur disajikan pada Tabel 4.

Tabel 4. Perbandingan Stabilitas dan Konsistensi Siklus Rilis

Rilis		
Parameter Konsistensi	Prosedur Manual (Lokal)	Otomatisasi CI/CD (<i>Github Actions</i>)
Integritas Pengujian	Rentan ada pengujian fitur yang terlewat karena bergantung sepenuhnya pada ketelitian pengembang.	Rangkaian pengujian (<i>Unit</i> , <i>GUI</i> , <i>Integration</i>) dieksekusi secara sekuensial dan menyeluruh sebelum rilis dipicu.
Kebersihan Hasil Build	Berisiko menyebabkan <i>error</i> pada berkas eksekusi jika pengembang lupa membersihkan <i>cache</i> lama.	Selalu konsisten karena instruksi <i>build</i> berjalan di server yang bersih dan menggunakan parameter otomatis.
Penomoran Versi Rilis	Rentan terjadi duplikasi versi atau kesalahan pengetikan (<i>typo</i>).	Terstandarisasi secara otomatis dan memiliki identitas unik menggunakan variabel sistem.
Lingkungan & Distribusi	Konfigurasi lokal bisa berubah-ubah; proses kompresi dan upload manual	Lingkungan spesifik terjamin lewat skrip YAML, proses kompresi dan

5. KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan hasil perancangan, implementasi, dan evaluasi, penerapan *pipeline* CI/CD berbasis *GitHub Actions* pada proyek aplikasi *Chart Lyric Editor* telah

berhasil membentuk infrastruktur otomatis yang terstruktur dan andal. Pemisahan alur kerja menjadi *Continuous Integration* untuk validasi kode dan *Continuous Delivery* untuk kompilasi serta distribusi rilis terbukti efektif menggantikan prosedur pengembangan konvensional. Secara empiris, implementasi sistem ini memberikan peningkatan efisiensi yang sangat signifikan, di mana total durasi siklus rilis berhasil dipangkas dari 17 menit 51 detik pada prosedur manual menjadi rata-rata hanya 1 menit 47 detik melalui otomatisasi.

Selain meningkatkan kecepatan, infrastruktur CI/CD ini juga berhasil memitigasi permasalahan utama terkait potensi *human error* dan ketidakkonsistenan output build. Hal ini dicapai melalui standarisasi lingkungan eksekusi, otomatisasi pengujian (*Unit*, GUI, *Integration test*), serta penomoran versi rilis yang terstruktur secara otomatis pada halaman *GitHub Releases*. Meskipun demikian, penelitian ini memiliki keterbatasan ruang lingkup, di antaranya evaluasi efisiensi waktu yang dibatasi pada jumlah sampel sebanyak 10 kali iterasi pengujian, serta lingkungan operasional *pipeline* dan validasi artefak yang difokuskan sepenuhnya pada sistem operasi *Windows*.

5.2. Saran

Terdapat beberapa saran yang dapat dipertimbangkan untuk pengembangan perangkat lunak dan penelitian di masa mendatang:

1. Peningkatan Cakupan Pengujian: Cakupan pengujian (*test coverage*) dapat diperluas dengan menambahkan skenario uji coba tambahan pada fitur-fitur aplikasi yang belum dievaluasi secara menyeluruh, sehingga jaminan kualitas kode saat proses integrasi menjadi lebih maksimal.
2. Validasi Integritas Artefak Rilis: Sistem distribusi rilis (*Workflow Build*) dapat disempurnakan dengan menambahkan mekanisme validasi otomatis terhadap ukuran dan struktur direktori artefak sebelum dipublikasikan, guna memastikan tidak ada komponen *file* yang kurang atau tertinggal.
3. Eksplorasi Lingkungan *Multi-Platform*: Penelitian selanjutnya dapat mengeksplorasi penerapan *pipeline* pada *runner* lintas sistem operasi (seperti *Linux* dan *macOS*) guna

menguji kompatibilitas distribusi apabila aplikasi *Chart Lyric Editor* dikembangkan menjadi aplikasi *multi-platform*.

DAFTAR PUSTAKA

- [1] F. Paramudita, M. I. Zulfa, and A. Taryana, "Implementasi DevOps pada Pengembangan Aplikasi Android Pendeteksi Kualitas Beras Berbasis Machine Learning," *Transmisi: Jurnal Ilmiah Teknik Elektro*, vol. 26, no. 3, pp. 105–113, Jul. 2024, doi: 10.14710/transmisi.26.3.105-113
- [2] J. Jaeni, N. A. S., and A. D. Laksito, "Implementasi Continuous Integration/Continuous Delivery (CI/CD) pada Performance Testing DevOps," *Journal of Information System Management (JOISM)*, vol. 4, no. 1, pp. 62–66, Aug. 2022, doi: 10.24076/joism.2022v4i1.887.
- [3] D. Enda, S. Supria, I. Yulia, M. F. Amirul, and M. F. Asyrofi, "Penerapan Continuous Integration (CI) pada Aplikasi Web Profil Karang Taruna," in *Seminar Nasional Industri dan Teknologi*, 2022, pp. 56–63.
- [4] F. P. Apriliyanto, Y. Prihati, and C. A. Bakti, "Implementasi CI/CD Menggunakan Github Actions untuk Pengembangan Web Landing Page UMKM XYZ," *INTECOMS: Journal of Information Technology and Computer Science*, vol. 7, no. 4, pp. 1239–1249, Jul. 2024, doi: 10.31539/intecomsv7i4.10961.
- [5] E. Fernando, S. Surjandy, M. Meyliana, and P. Siagian, "Desain Sistem Pengenalan Varietas Bibit Tanaman Kelapa Sawit dengan Pendekatan Design Science Research Methodology (DSRM)," *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 7, no. 2, p. 249, Feb. 2020, doi: 10.25126/jtiik.2020721456.
- [6] T. Tohirin, S. F. Utami, S. R. Widiyanto, and W. Al Mauludyansah, "Implementasi DevOps pada Pengembangan Aplikasi e-Skrining Covid-19," *MULTINETICS*, vol. 6, no. 1, pp. 15–20, May 2020, doi: 10.32722/multinetics.v6i1.2764.
- [7] Z. Zuhakim and A. Kurniawan, "Implementasi Continuous Integration dan Continuous Deployment pada Pengembangan Aplikasi Website Menggunakan Docker dan Github Actions," *Universitas Negeri Surabaya*, 2024. Measurement, Simulation, and Modeling. New York: John Wiley & Sons, 1991