

ANALISIS PERBANDINGAN UNIT TEST AUTOMATION FRAMEWORK DENGAN METODE THE DISTANCE TO THE IDEAL ALTERNATIF



Disusun Oleh:

N a m a : Rifaldi Fakhrii
NIM : 20523023

**PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA
FAKULTAS TEKNOLOGI INDUSTRI
UNIVERSITAS ISLAM INDONESIA
2020**

HALAMAN PENGESAHAN DOSEN PEMBIMBING

**ANALISIS PERBANDINGAN UNIT TEST AUTOMATION
FRAMEWORK DENGAN METODE THE DISTANCE TO THE
IDEAL ALTERNATIF**

TUGAS AKHIR



(Dr. Novi Setiani, S.T., M.T.)

HALAMAN PENGESAHAN DOSEN PENGUJI

ANALISIS PERBANDINGAN UNIT TEST AUTOMATION FRAMEWORK DENGAN METODE THE DISTANCE TO THE IDEAL ALTERNATIF

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta,

Tim Penguji

Dr. Novi Setiani, S.T, M.T.

Anggota 1

Andhika Giri Persada, S.Kom., M.Eng.

Anggota 2

Arrie Kurniawardhani S.SI., M.kom.

Mengetahui,
Ketua Program Studi Informatika – Program Sarjana

Fakultas Teknologi Industri
Universitas Islam Indonesia



(Dhomas Hatta Fudholi S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Rifaldi Fakhrii

NIM : 20523023

Tugas akhir dengan judul:

**ANALISIS PERBANDINGAN UNIT TEST AUTOMATION
FRAMEWORK DENGAN METODE THE DISTANCE TO THE
IDEAL ALTERNATIF**

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 29 Oktober 2024



(Rifaldi Fakhrii)

HALAMAN PERSEMBAHAN

Dengan mengucapkan rasa syukur yang sangat dalam, karya tugas akhir ini saya persembahkan kepada orang tua tercinta yang selalu bertanya “kapan sidang?”, dan “kapan selesainya?” karena itulah saya menyelesaikan tugas akhir ini. Semoga dengan selesainya tugas akhir ini bisa membuat orang tua yang tercinta bahagia.

HALAMAN MOTO

“Aku Sesuai dengan prasangka hambaku”

“Kalau Anda sedang punya persoalan yang membuat Anda kecewa, jangan larut dalam kekecewaan tersebut. Perangi perasaan yang tidak baik. Boleh jadi sesuatu yang Anda tidak sukai (sesuatu membuat Anda kecewa) ada kebaikan di dalamnya”

(Ustad Adi Hidayat)

“Tidak peduli betapa kuatnya dirimu, jangan pernah mengatasi semuanya sendirian, sebaliknya kau pasti akan gagal.”

(Itachi Uchiha)

“A little sadness will come before happiness comes to us”

(Bob ross)

“Belajarlah memaafkan jika kamu disakiti, belajarlah untuk meminta maaf ketika kamu menyakiti, belajar dari kesalahan yang kamu perbuat”

(Jiraiya)

KATA PENGANTAR

Assalamu 'alaikum Warahmatullahi Wabarakatuh

Alhamdulillah rabbil 'alamin

Tiada kalimat yang pantas penulis ucapkan kecuali rasa syukur kepada Tuhan Yang Mah Esa atas selesainya tugas akhir ini. Perasaan senang, terharu, dan sedih saya curahkan semuanya dalam penulisan tugas akhir ini. Tidak lupa pula dukungan baik dari berbagai pihak yang diberikan kepada penulis dalam penyusunan makalah ini. Oleh karena itu, izinkan penulis mengucapkan rasa terima kasih kepada :

1. Allah SWT yang selalu memberikan kemudahan bagi saya, yang selalu memberikan rahmat dan hidayahnya kepada saya, dan menemani perjalanan saya dalam menyelesaikan pengerjaan tugas akhir ini.
2. Kedua orang tua yang senantiasa selalu berdoa dan memberikan dukugan dalam bentun apapun kepada penulis.
3. Ibu Dr. Novi Setiani, S.T., M.T. selaku dosen pembimbing tugas akhir yang senantiasa mmeberikan kemudahan, arahan, ilmu, serta waktu untuk membimbing penulis dalam menyelesaikan tugas akhir ini.
4. Sahabat baik saya Alfito Yoga Pamungkas dan Aulia Rahman Hakim yang selalu mendukung saya dalam bentuk apapun dan selalu peduli atas segala permasalahan.
5. Rekan-Rekan perkuliahan saya khususnya Yuanda, Gilang, Ramadzikri, Thoriq, Rifqi Setya Iskandar, Ivan, Muhammad Sulthon, Hudha, Reza, dan yang tidak bisa saya sebutkan satu persatu yang menemani saya dalam pengerjaan tugas akhir ini.
6. Kerabat dekat saya Mba della yang membantu saya dalam pengerjaan tugas akhir ini.

Penulis menyadari bahwa dalam penelitian laporan tugas akhir ini masih terdapat banyak kekurangan. Oleh sebab itu, penulis menerima segala bentuk kritik dan saran yang membangun. Semoga Allah SWT memberikan keberkahan kepada kita semua, aamiin

Wassalamu 'alaikum warrahmatullahi wabarakatuh

Yogyakarta, 29 Oktober 2024



(Rifaldi Fakhrii)

SARI

Pengujian perangkat lunak memerlukan metode yang efektif untuk mengevaluasi kinerja *framework* pengujian otomatis. Dalam hal ini, otomatisasi pengujian unit semakin diperlukan dalam pengembangan perangkat lunak, namun pilihan *framework* untuk otomatisasi pengujian masih membutuhkan perbandingan yang mendalam untuk mengetahui kinerja *framework*. Dalam Penelitian ini bertujuan untuk menganalisis dan membandingkan kinerja dari empat *framework* pengujian otomatis unit test, yaitu EvoSuite, Randoop, Squaretest, dan Github Copilot dengan menggunakan metode *The Distance to The Ideal Alternative (DIA)* untuk menentukan alternatif terbaik berdasarkan parameter *Technical Economic View*, *Testing Process View*, dan *Quality Standard View*. Analisis dalam penelitian ini dilakukan melalui pengujian terhadap empat *framework* otomatisasi pengujian unit. Selain itu, tanggapan dari responden yang berpengalaman dalam unit testing juga dikumpulkan untuk memberikan perspektif tambahan. Hasil analisis dari penelitian ini menunjukkan bahwa Github Copilot merupakan *framework* terbaik dengan skor tertinggi dalam aspek kemudahan penggunaan, efisiensi, dan kualitas hasil pengujian. Diharapkan hasil penelitian ini dapat memberikan informasi mengenai efektivitas dan efisiensi dari setiap *framework* yang diuji, serta rekomendasi untuk memilih alat pengujian unit otomatis yang paling optimal dalam konteks pengembangan perangkat lunak.

Kata kunci: *framework*, *The Distance to The Ideal Alternative*, *unit test*.

GLOSARIUM

Glosarium memuat daftar kata tertentu yang digunakan dalam laporan dan membutuhkan penjelasan, misalnya kata serapan yang belum lazim digunakan. Urutkan sesuai abjad. Contoh penulisannya seperti di bawah ini:

Code Coverage	Pengukuran seberapa luas kode yang dieksekusi selama pengujian.
Distance to The Ideal Alternative (DIA)	Metode dalam pengambilan keputusan multi-kriteria (MADM) yang menilai alternatif berdasarkan jarak alternatif ideal yang diinginkan.
Evosuite	<i>Framework</i> otomatis untuk menghasilkan tes unit untuk Java guna meningkatkan cakupan kode.
Github Copilot	Plugin yang menggunakan AI untuk membantu dalam pembuatan kode secara otomatis di dalam IDE.
Randoop	<i>Framework</i> yang menggunakan algoritma umpan balik untuk menghasilkan tes unit secara otomatis pada kode Java.
Squaretest	Plugin IntelliJ IDEA yang membuat <i>unit test</i> secara otomatis untuk kelas Java,
Unit Test	Metode pengujian perangkat lunak pada komponen terkecil, seperti fungsi atau metode, untuk memastikan kesesuaiannya pada proyek.

DAFTAR ISI

HALAMAN JUDUL	i
HALAMAN PENGESAHAN DOSEN PEMBIMBING.....	ii
HALAMAN PENGESAHAN DOSEN PENGUJI.....	iii
HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR	iv
HALAMAN PERSEMBAHAN	v
HALAMAN MOTO	vi
KATA PENGANTAR	vii
SARI	viii
GLOSARIUM	ix
DAFTAR ISI	x
DAFTAR TABEL	xii
DAFTAR GAMBAR.....	xiv
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	4
1.3 Batasan Masalah.....	4
1.4 Tujuan Penelitian.....	4
1.5 Manfaat Penelitian.....	4
1.6 Metode Penelitian.....	5
1.7 Struktur Laporan.....	5
BAB II TINJAUAN PUSTAKA DAN LANDASAN TEORI	7
2.1 Landasan teori	7
2.1.1 Unit Test.....	7
2.1.2 Alat Pengujian Unit Otomatis	7
2.1.3 Metode Pengukuran coverage	8
2.1.4 Kriteria Perbandingan Alat pengujian.....	9
2.1.5 Metode Perbandingan The Distance To The Ideal Alternatif	10
2.2 Kajian pustaka	13
2.2.1 Penelitian terdahulu.....	13
2.2.2 Telaah kajian Pustaka.....	15
BAB III METODOLOGI PENELITIAN	17
3.1 Alur Proses	17
3.2 Studi Literatur.....	18
3.3 Pemilihan Parameter Perbandingan <i>Framework Unit Test</i>	18
3.4 Pembobotan parameter dan nilai	21
3.4.1 Demografi Responden.....	21
3.4.2 Pemberian bobot Parameter dan sub parameter	23
3.4.3 Parameter Technical Economic View.....	24
3.4.4 Parameter Testing Process View.....	25
3.4.5 Parameter Quality Standards View	28
3.4.6 Pembobotan seluruh parameter utama	29
3.5 Pengunduhan Library dan Pembuatan kode	29
BAB IV HASIL DAN PEMBAHASAN	35
4.1 Pengujian Unit dengan Evosuite	35
4.1.1 Persiapan dan Instalasi EvoSuite	35
4.1.2 Proses Menghasilkan Unit Test Otomatis	37
4.1.3 Hasil Pengujian Unit dan Analisis	45
4.2 Pengujian Unit dengan Randoop	45
4.2.1 Persiapan dan Instalasi Randoop.....	45
4.2.2 Proses Menghasilkan Unit Test Otomatis	47
4.2.3 Hasil pengujian Unit dan Analisis.....	51
4.3 Pengujian Unit dengan Github Copilot	52

4.3.1	Persiapan dan Instalasi Github Copilot	52
4.3.2	Proses Menghasilkan Unit Test Otomatis	53
4.3.3	Hasil Pengujian Unit dan Analisis	57
4.4	Pengujian Unit dengan Squaretest.....	59
4.4.1	Persiapan dan Instalasi Squaretest	59
4.4.2	Proses Menghasilkan Unit Test Otomatis	60
4.4.3	Hasil Pengujian Unit dan Analisis	63
4.5	Penerapan Metode The Distance to The Ideal Alternative.....	65
4.5.1	Pengumpulan Data	65
4.5.2	Penyusunan Matriks Keputusan.....	69
4.5.3	Normalisasi Matriks Keputusan.....	73
4.5.4	Pembobotan Matriks Normalisasi	74
4.5.5	Menentukan Solusi Ideal Positif dan Negatif Parameter Technical Economic View	77
4.5.6	Menentukan Solusi Ideal Positif dan Negatif Parameter Testing Process View	77
4.5.7	Menentukan Solusi Ideal Positif dan Negatif Parameter Quality Standard View..	78
4.5.8	Perhitungan jarak Manhattan	79
4.5.9	Menentukan <i>Positive Ideal Alternative (PIA)</i>	82
4.5.10	Mengidentifikasi Peringkat	82
4.6	Melakukan Analisis Perbandingan	84
4.6.1	Peringkat Untuk Masing-Masing Parameter	84
4.6.2	Menghitung Skor <i>Weighted Scoring</i>	85
4.6.3	Alternatif Terbaik untuk <i>Framework</i>	85
BAB V	KESIMPULAN DAN SARAN	87
5.1	Kesimpulan.....	87
5.2	Saran.....	87
DAFTAR	PUSTAKA.....	88
LAMPIRAN		91

DAFTAR TABEL

Tabel 2.1 Penelitian Terdahulu	13
Tabel 3.1 Bobot pada parameter Technical Economic View	24
Tabel 3.2 Pembobotan nilai sub parameter <i>Multi Language</i>	24
Tabel 3.3 Pembobotan nilai sub parameter <i>Cost</i>	25
Tabel 3.4 Pembobotan nilai sub parameter <i>Community Support and Documentation</i>	25
Tabel 3.5 Bobot pada parameter Testing Process view	25
Tabel 3.6 Pembobotan nilai sub parameter Sistem Pelacakan Bug	26
Tabel 3.7 Pembobotan nilai sub parameter Pengujian Otomatis	26
Tabel 3.8 Pembobotan nilai sub parameter <i>Execution Logging</i>	26
Tabel 3.9 Pembobotan nilai sub parameter <i>Script Creation</i>	27
Tabel 3.10 Pembobotan nilai sub parameter <i>Reporting</i>	27
Tabel 3.11 Pembobotan nilai sub parameter <i>Code Coverage</i>	27
Tabel 3.12 Bobot pada parameter <i>Quality Standard View</i>	28
Tabel 3.13 Pembobotan nilai sub parameter <i>Readability</i>	28
Tabel 3.14 Rentang nilai <i>Usability</i>	28
Tabel 3.15 Pembobotan parameter utama.....	29
Tabel 4.1 Hasil pengujian otomatis setiap <i>framework</i>	65
Tabel 4.2 Hasil pengujian otomatis setiap <i>framework</i>	66
Tabel 4.3 Hasil matriks keputusan parameter Technical Economic View	70
Tabel 4.4 Hasil matriks keputusan parameter Testing Process View	70
Tabel 4.5 Hasil matriks keputusan parameter Quality Standard View	72
Tabel 4.6 Hasil normalisasi matriks keputusan Technical Economic View	73
Tabel 4.7 Hasil normalisasi keputusan Testing Process view.....	73
Tabel 4.8 Hasil normalisasi matriks keputusan Quality Standard View.....	74
Tabel 4.9 Hasil pembobotan matriks normalisasi Technical Economic View	75
Tabel 4.10 Hasil pembobotan matriks normalisasi Testing Process View	76
Tabel 4.11 Hasil pembobotan matriks normalisasi Quality Standard View	76
Tabel 4.12 Hasil solusi ideal positif Technical Economic View	77
Tabel 4.13 Hasil solusi ideal negatif Technical Economic View	77
Tabel 4.14 Hasil solusi ideal positif Testing Process View	78
Tabel 4.15 Hasil solusi ideal negatif Testing Process View	78
Tabel 4.16 Hasil solusi ideal positif Quality Standard View	78
Tabel 4.17 Hasil solusi ideal negatif Quality Standard View	79
Tabel 4.18 Jarak manhattan max technical economic view	80
Tabel 4.19 Jarak manhattan min technical economic view.....	80

Tabel 4.20 Jarak manhattan max testing process view	81
Tabel 4.21 Jarak manhattan min testing process view	81
Tabel 4.22 Jarak manhattan max quality standard view	81
Tabel 4.23 Jarak manhattan min quality standard view	82
Tabel 4.24 Hasil peringkat pada <i>technical economic view</i>	83
Tabel 4.25 Hasil peringkat pada <i>testing process view</i>	83
Tabel 4.26 Hasil peringkat pada <i>quality standard view</i>	84
Tabel 4.27 Hasil peringkat dan bobot masing-masing <i>framework</i>	84
Tabel 4.28 Hasil Peringkat Akhir	85

DAFTAR GAMBAR

Gambar 3.1 Alur Proses	18
Gambar 3.2 Pendidikan terakhir responden	22
Gambar 3.3 Pengalaman kerja responden	22
Gambar 3.4 Grafik penggunaan framework pengujian otomatis	23
Gambar 3.5 Library Joda-Time	30
Gambar 3.6 Depedensi Joda Time	30
Gambar 3.7 Kode pada kelas Acara	31
Gambar 3.8 kode pada kelas Jadwal	33
Gambar 3.9 Kode pada kelas ManajemenJadwal	34
Gambar 4.1 Tampilan setting proyek IntelliJ IDEA	36
Gambar 4.2 Konfigurasi file pom.xml	36
Gambar 4.3 Depedensi dan Plugin EvoSuite	37
Gambar 4.4 Perintah maven untuk menghasilkan <i>unit test</i>	38
Gambar 4.5 laporan Eksekusi EvoSuite	38
Gambar 4.6 Perintah export untuk memindahkan direktori	38
Gambar 4.7 Hasil Test yang dihasilkan EvoSuite setelah di export	39
Gambar 4.8 Kelas Acara_ESTest EvoSuite	43
Gambar 4.9 Kelas Jadwal_ESTest EvoSuite	44
Gambar 4.10 Kelas ManajemenJadwal_ESTest Evosuite	45
Gambar 4.11 Perintah untuk menjalankan tes	45
Gambar 4.12 Hasil tes setelah dijalankan	45
Gambar 4.13 Situs resmi Randoop	46
Gambar 4.14 Lokasi file jar Randoop	46
Gambar 4.15 Depedensi JUnit	47
Gambar 4.16 Perintah menjalankan Randoop menghasilkan test	47
Gambar 4.17 kelas AcaraRegressionTest0 Randoop	49
Gambar 4.18 Kelas JadwalRegressionTest0 Randoop	50
Gambar 4.19 perintah untuk mengatur environment variable	51
Gambar 4.20 Perintah untuk mengkompilasi	51
Gambar 4.21 Perintah untuk menjalankan tes yang dihasilkan Randoop	51
Gambar 4.22 Hasil tes pada kelas JadwalRegressionTest0 setelah dijalankan	52
Gambar 4.23 Install Github Copilot pada IntelliJ IDEA	53
Gambar 4.24 Icon untuk membuka Github Copilot di IntelliJ IDEA	54
Gambar 4.25 Panel chat Github Copilot	54
Gambar 4.26 Kelas AcaraTest Github Copilot	56

Gambar 4.27 Kelas JadwalTest Github Copilot.....	57
Gambar 4.28 Kelas JadwalTest Github Copilot.....	57
Gambar 4.29 Hasil run kelas Acaratest Github Copilot.....	58
Gambar 4.30 Hasil run kelas JadwalTest Github Copilot.....	58
Gambar 4.31 Hasil ManajemenJadwalTest Github Copilot	59
Gambar 4.32 Instalasi plugin Squaretest pada IntelliJ IDEA	60
Gambar 4.33 Proses menghasilkan <i>unit test</i> dengan Squaretest	61
Gambar 4.34 Kelas AcaraTest Squaretest.....	62
Gambar 4.35 Kelas JadwalTest Squaretest	63
Gambar 4.36 Kelas ManajemenJadwalTest Squaretest	63
Gambar 4.37 Hasil run test pada kelas AcaraTest Squaretest.....	64
Gambar 4.38 Hasil run test pada kelas JadwalTest Squaretest	64
Gambar 4.39 Hasil run test pada kelas ManajemenJadwalTest Squaretest	65
Gambar 4.40 Pertanyaan pada <i>framework</i> EvoSuite.....	67

BAB I

PENDAHULUAN

1.1 Latar Belakang

Pengujian adalah salah satu langkah penting dalam pengembangan perangkat lunak yang bertujuan untuk memastikan bahwa setiap bagian dapat berfungsi dengan benar. Selain itu, pengujian dilakukan untuk mengidentifikasi kesalahan (*error*) dan kecacatan (*bug*) saat eksekusi perangkat lunak tersebut. Kesalahan dalam perangkat lunak dapat mengakibatkan perangkat lunak tersebut tidak berfungsi dengan benar, sementara kecacatan dapat mengakibatkan perangkat lunak tersebut tidak berjalan sesuai dengan spesifikasi yang telah ditentukan (Hasibuan & Dirgahayu, 2021). Dalam melakukan pengujian perangkat lunak perlu dilakukan dengan pengerjaan yang efektif dan efisien. Pengujian perangkat lunak yang efektif dan efisien melibatkan berbagai tahapan, salah satunya adalah *unit testing*.

Unit testing merupakan salah satu metode pengujian perangkat lunak yang fokus pada pengujian unit terkecil dalam kode, seperti fungsi atau metode (Kua, 1995). Pentingnya pengujian unit dalam pengembangan perangkat lunak tidak dapat diabaikan, pengujian unit yang dirancang dengan baik untuk komponen tertentu seperti metode, kelas, atau modul dapat berfungsi sebagai bentuk dokumentasi. Dokumentasi membantu mengidentifikasi masalah pada saat melakukan pengembangan perangkat lunak untuk membantu mencegah terjadinya kode-kode yang memiliki kesalahan (*error*) dan kecacatan (*bug*) (Dinella dkk., 2022). Manfaat *unit testing* tidak bisa diabaikan, terutama karena pengujian ini membantu memastikan kode program dapat berjalan secara efektif dan efisien. Dalam pengembangan perangkat lunak, menemukan dan mengatasi *error* atau *bug* merupakan tantangan besar, terutama seiring dengan semakin kompleksnya proyek yang dikembangkan. Semakin besar proyek perangkat lunak yang dikembangkan, semakin banyak unit testing yang diperlukan. Karena jumlah komponen yang harus diuji meningkat.

Test automation pengujian perangkat lunak dapat mengurangi secara signifikan upaya yang diperlukan untuk pengujian yang memadai. Selain itu, *automation* memungkinkan peningkatan jumlah pengujian yang dapat diselesaikan dalam waktu yang terbatas. Pengujian dapat dijalankan dalam hitungan menit, jika dilakukan secara manual maka akan memakan waktu berjam-jam dalam pengerjaannya (Fewster & Graham, 1999). Meskipun, *test automation* sangat efektif untuk menghemat waktu dan tenaga, pengujian unit tetap memainkan peran penting dalam memastikan kualitas perangkat lunak. Pengembang harus menulis tes

sebelum kode kelas sebenarnya dibuat. Dengan begitu pengembang dapat menjalankan tes tersebut untuk menganalisis dan mencari tanda-tanda kesalahan dalam fungsionalitas unit. Pengujian ini sederhana, berjalan cepat, dan efektif dalam menguji fungsionalitas sistem tingkat rendah. Namun, pembuatan pengujian unit adalah proses yang memakan banyak waktu, tenaga, dan memiliki banyak elemen manual yang berulang.

Jika proses pembuatan pengujian unit dapat diotomatisasi, upaya dan biaya pengujian unit dapat dikurangi secara signifikan. Pembuatan pengujian otomatis yang efektif juga dapat melengkapi kasus pengujian yang ditulis secara manual dan membantu memastikan kualitas rangkaian pengujian (Fontes dkk., 2021). Selanjutnya, tingkat kesempurnaan dari proses pengujian otomatisasi merupakan faktor penting dalam penilaian efektivitasnya (Wang dkk., 2020). Untuk mendukung pengembangan dan memastikan kualitas perangkat lunak, terdapat banyak *framework* otomatis yang tersedia seperti *Junit*, *XUnit*, *Randoop*, *Robot Framework*, *EvoSuite*, dan lain-lain.

Dalam penelitian ini menggunakan kode berbasis Java pada pengembangan aplikasi desktop. Java dipilih karena memiliki kemudahan untuk mengorganisir kode, dan mempermudah pengelolaan aplikasi berdasarkan struktur dasar yang dimiliki Java, seperti *package*, kelas, dan metode. Selain itu, Java memiliki polimorfisme yang sederhana dibandingkan bahasa yang lain, sehingga mempermudah penerapan prinsip berorientasi objek (OOP) (Cohen dkk., 2006; Demurjian Sr & Shin, t.t.). Penelitian ini juga menggunakan empat jenis *framework* yang telah dikembangkan untuk mendukung pengujian unit secara otomatis yaitu EvoSuite, Randoop, Squartest, dan Github Copilot. Tiga alat pertama yaitu Evosuite, Randoop, dan Squartest digunakan dalam penelitian ini karena *framework* tersebut berfokus pada pengujian kode berbasis Java, sedangkan Github Copilot, meskipun mendukung banyak bahasa pemrograman, digunakan dalam penelitian ini untuk menjaga konsistensi dalam pengujian kode Java. EvoSuite adalah alat yang mengotomatisasi tugas program dengan secara sistematis menghasilkan rangkaian pengujian yang mencapai jangkauan kode yang luas, tinggi, dan sekecil mungkin serta memberikan pernyataan (Fraser & Arcuri, 2011b). EvoSuite mengoptimalkan seluruh rangkaian pengujian dengan mempertimbangkan seluruh kriteria cakupan keseluruhan secara bersamaan (Fraser & Arcuri, 2011a). Randoop adalah alat yang mengeksekusi kode Java secara otomatis yang menggunakan algoritma berdasarkan umpan balik untuk menghasilkan pengujian unit untuk program-program java (Pacheco & Ernst, 2007). Squartest merupakan plugin di IntelliJ IDEA yang memungkinkan pembuatan unit test secara otomatis untuk kelas java. Plugin ini secara otomatis akan menginisialisasi objek

dependensi yang diperlukan, sehingga mempercepat proses pengujian. Github Copilot adalah plugin IDE (*Integrated Development Environment*) yang memanfaatkan teknologi AI dalam penulisan kode. Github copilot diluncurkan pada bulan Juni 2021 oleh Github(Siroš dkk., 2024).

Metode *The Distance to The Ideal Alternative* (DIA) adalah suatu metode yang dimiliki oleh MADM dan didasarkan pada prinsip yang sama dengan metode *Technique for Order Preference by Similarity to Ideal Solution* (Topsis). Tujuan utama pengembangan metode ini adalah untuk meningkatkan kinerja metode TOPSIS(Lestari & Lubis, 2022). Metode DIA memiliki sejumlah keunggulan, antara lain prinsip yang sederhana, mudah dipahami, komputasi yang efisien, dan kemampuan mengukur efektivitas relatif alternatif keputusan dalam istilah matematika sederhana(Jun dkk., t.t.-a).

Berbagai teknik, alat, dan metode yang telah mendukung otomatisasi pengujian unit dalam pengembangan perangkat lunak. Meskipun metode The Distance to Ideal Alternative (DIA) telah diterapkan berbagai penelitian, studi yang secara khusus membandingkan kinerja *framework* pengujian unit otomatis dengan metode pengambilan keputusan multi kriteria, seperti The Distance to The Ideal Alternative (DIA), masih terbatas. Mengingat pentingnya otomatisasi pengujian unit sebagai bagian integral dari pengembangan perangkat lunak, penelitian ini berfokus kepada perbandingan kinerja masing-masing *framework* pengujian unit otomatis untuk memberikan wawasan terkait efektivitas, efisiensi dan kualitas pengujian unit test.

Dalam penelitian ini metode *The Distance to the Ideal Alternatif* (DIA) akan digunakan untuk memahami karakteristik setiap pengujian unit yang dihasilkan oleh alat EvoSuite, Randoop, Squartest, dan Github Copilot. Lalu, parameter yang akan digunakan untuk menguji kinerja framework yaitu *Technical and Economic View*, *Testing process View*, dan *Quality Standards View*(Abdulwareth & Al-Shargabi, 2021). Dengan memahami karakteristik dan kinerja pengujian unit yang dihasilkan oleh alat pengujian unit otomatis, penelitian ini diharapkan dapat memberikan informasi dan rekomendasi *framework* pengujian yang paling efektif dan efisien dalam pengembangan perangkat lunak.

Di samping itu, pengujian dilakukan dengan membuat kode yang dibuat oleh penulis yaitu kelas Acara, Jadwal, dan ManajemenJadwal lalu menggunakan library Joda-Time yang diperoleh dari platform GitHub. Joda-Time dikembangkan oleh Stephen Colebourne, yang merupakan library yang dirancang untuk menggantikan kelas-kelas tanggal dan waktu standar di Java. Banyak sistem kalender seperti Julian, Gregorian, Buddha, Koptik, Etiopia, dan Islam

dapat digunakan di library ini sambil tetap menyediakan API yang sederhana dan mudah digunakan. Selain itu, Joda-Time adalah solusi yang sangat baik untuk menangani waktu dalam aplikasi java karena mendukung operasi terkait zona waktu, durasi, format, dan parsing tanggal dan waktu.

1.2 Rumusan Masalah

Bagaimana menganalisis perbandingan pengujian unit otomatis berdasarkan parameter *Technical Economic View*, *Testing Process View*, dan *Quality Standard View* menggunakan metode *The Distance to The Ideal Alternative (DIA)*. dan bagaimana hasil analisisnya?

1.3 Batasan Masalah

Batasan masalah dari penelitian ini yaitu:

- a. Pengujian yang dilakukan menggunakan bahasa pemrograman Java.
- b. Pengujian dilakukan dalam pengembangan aplikasi berbasis dekstop
- c. Pengujian yang dilakukan adalah pengujian unit test dengan menggunakan kode yang dibuat, kode yang diuji terdiri dari kelas Acara, Jadwal, dan ManajemenJadwal, serta menggunakan library Joda-Time.
- d. Pengujian dilakukan dengan empat *test automation framework*, yaitu EvoSuite, Randoop, Squaretest, dan Github Copilot. Tiga *framework* pertama yaitu EvoSuite, Randoop, Dan Squaretest dirancang secara khusus untuk mendukung pengujian kode berbasis Java. Sementara pada *framework* Github Copilot disesuaikan dengan ketiga alat sebelumnya untuk menjaga konsistensi dalam pengujian.
- e. Parameter perbandingan dari segi kinerja yaitu *Technical and Economic View*, *Testing process View*, dan *Quality Standards View*.

1.4 Tujuan Penelitian

Tujuan penelitian dalam penelitian ini yaitu:

- a. Melakukan pengujian langsung menggunakan empat *unit test automation framework* yaitu EvoSuite, Randoop, Squaretest, dan Github Copilot dengan kode yang dibuat oleh penulis. Dimulai dari setup framework hingga menghasilkan unit test.
- b. Melakukan perbandingan kinerja *unit test automation framework* berdasarkan parameter *Technical Economic View*, *Testing Process View*, dan *Quality Standard View* menggunakan Metode pengambilan keputusan multi kriteria yaitu *The Distance to Ideal Alternatie(DIA)*.

1.5 Manfaat Penelitian

Manfaat dari penelitian ini adalah sebagai berikut:

- a. Mengetahui kinerja pengujian unit otomatis dalam melakukan pengujian pada sebuah studi kasus.
- b. Pembaca dapat mengetahui *test automation framework* yang terbaik dari segi kinerja.

1.6 Metode Penelitian

- a. Proses Studi Literatur : Penelitian ini menggunakan studi literatur melalui jurnal, buku, internet, dan sumber lain. Studi literatur dijadikan sebagai bahan referensi untuk penelitian ini.
- b. Proses Eksplorasi Data : Data yang digunakan adalah beberapa kode program yang diambil dari platform github.
- c. Proses Pengujian: Pengujian dilakukan dengan *unit testing* dengan alat EvoSuite, Randoop, Squaretest, dan Github Copilot sebagai perbandingan.
- d. Proses analisis hasil : Setelah dilakukannya pengujian unit akan dilakukan analisis keempat alat dengan parameter yang telah ditentukan sebagai perbandingan untuk menentukan ideal atau tidaknya alat yang digunakan dengan menggunakan metode *The Distance to The Ideal Alternatif*.

1.7 Struktur Laporan

Struktur penulisan laporan dari penelitian dijelaskan pada pembahasan di bawah ini:

a. BAB I PENDAHULUAN

Berisi tentang gambaran umum tentang topik yang dibahas. Mencakup tentang pembahasan latar belakang, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metode penelitian, dan struktur laporan.

b. BAB II LANDASAN TEORI

Berisi tentang teori-teori yang digunakan dalam penelitian ini untuk menyelesaikan permasalahan pada penelitian ini, teori-teori yang diuraikan dalam bab ini menggunakan jurnal, buku, dan artikel sebagai bahan referensi dengan topik yang meliputi masalah pengujian *unit testing* dengan metode *The Distance to The Ideal Alternatif* Mencakup tentang dasar teori pengujian perangkat lunak, *Unit Testing*, *symbolic execution*, *EvoSuite*, *Randoop*, dan kriteria-kriteria untuk perbandingan dari ketiga *tools* tersebut.

c. BAB III METODOLOGI PENELITIAN

Berisi tentang proses pengerjaan dari kode program yang sudah didapatkan dengan menggunakan tiga alat yaitu *EvoSuite*, *Randoop*, dan *Symbolic Execution*. Mencakup tentang alur proses, perbandingan alat dan rekomendasi.

d. **BAB IV HASIL DAN PEMBAHASAN**

Berisi tentang hasil pengujian unit yang telah dilakukan dan pembahasan tentang evaluasi dari pengujian dua alat yang telah digunakan dalam pengujian unit.

e. **BAB V KESIMPULAN DAN SARAN**

Berisi tentang rangkuman dan kesimpulan dari hasil penelitian serta saran-saran yang perlu diperhatikan dalam pengujian unit.

BAB II

TINJAUAN PUSTAKA DAN LANDASAN TEORI

2.1 Landasan teori

2.1.1 Unit Test

Unit testing adalah metode pengujian perangkat lunak yang berfokus pada komponen terkecil dari suatu aplikasi. *Unit testing* memberikan kemampuan untuk menguji perangkat lunak secara terpisah dengan melibatkan komponen unit terkecil dan menggunakan beberapa potongan kode seperti fungsi(Rahayu, 2020). Langkah umum yang dilakukan dalam pengujian unit adalah dengan membuat kode *unit testing* dan *test case* secara manual. *Test case* sendiri akan digunakan dalam proses integrasi dan berfokus pada *input* dan *output*(Pressman, 2010).

Pengujian unit berfokus pada validasi bahwa unit terkecil dalam suatu sistem memenuhi kontrak yang ditentukan oleh *API* dan dokumentasinya(Kua, 2019; Pressman, 2010). Proses ini menjadi kunci dalam memastikan bahwa setiap komponen perangkat lunak beroperasi sesuai dengan spesifikasi yang telah ditetapkan. Setelah semua unit teruji dan lulus, maka proses selanjutnya adalah menggabungkan atau mengintegrasikan komponen-komponen tersebut untuk membentuk aplikasi secara menyeluruh(Pressman, 2010).

2.1.2 Alat Pengujian Unit Otomatis

a. EvoSuite

EvoSuite adalah alat yang dapat menghasilkan rangkaian tes secara otomatis pada program *Java*. Alat ini dievaluasi di banyak proyek *open source* guna meningkatkan cakupan kode. Secara default, EvoSuite menghasilkan rangkaian *JUnit* yang dioptimalkan untuk memaksimalkan cakupan kode untuk setiap kelas. Alat ini juga dapat disesuaikan untuk mengoptimalkan skor mutasi dari rangkaian pengujian yang dihasilkan dengan *fitness function*(Roslan dkk., 2022). Pengoperasian EvoSuite melibatkan batas waktu tertentu untuk menghasilkan tes setiap kelas yang ditargetkan. Jika dalam batas waktu yang telah ditentukan EvoSuite gagal mencapai cakupan tes yang lengkap, maka program akan berhenti dan beralih ke tugas berikutnya. Oleh Karena itu, efisiensi dalam menelusuri dan mengevaluasi tes yang dihasilkan sangat penting. Dengan mempercepat proses, EvoSuite dapat meningkatkan jumlah tes yang dihasilkan dan dievaluasi dalam waktu yang diberikan, sehingga memaksimalkan efektivitas pengujian(Callan & Petke, 2022).

b. Randoop

Randoop adalah alat yang dikenal di bidang pengujian acak yang menggunakan algoritma berbasis umpan balik untuk secara otomatis menghasilkan tes unit pada program

Java. Alat ini menggunakan kelas *Java* sebagai kelas yang diuji (*CUT*) dan membuat urutan panggilan metode secara acak pada objek kelas tersebut. Selanjutnya, dengan mengeksekusi setiap urutan metode yang dihasilkan, *Randoop* menentukan apakah urutan tersebut cocok untuk diperluas guna menghasilkan urutan yang lebih panjang. Selama pembuatan urutan, *Randoop* menjalankan urutan tersebut untuk memverifikasi *CUT* berdasarkan sekumpulan kontrak yang sudah ditetapkan yang harus kompatibel dengan setiap kelas *Java*. Jika ada pelanggaran kontrak, *Randoop* akan menyimpan urutan metode tersebut sebagai tes yang menunjukkan adanya kesalahan. Jika tidak ada pelanggaran kontrak, maka urutan tersebut disimpan sebagai tes regresi yang menangkap perilaku sistem saat ini (Paydar & Azamnouri, 2019).

c. *Squaretest*

Squaretest adalah plugin yang terintegrasi dengan *IntelliJ IDEA (IDE)* yang dirancang membuat tes unit secara otomatis untuk kelas *Java*. Plugin ini menghasilkan tes dengan membangun urutan metode pemanggilan berdasarkan kelas sumber. *Squaretest* menghasilkan tes dengan menerapkan analisis alur data dan analisis alur kontrol.

d. *Github Copilot*

Github Copilot adalah alat atau plugin *IDE (Integrated Development Environment)* yang memanfaatkan teknologi AI dalam penulisan kode. *Github copilot* diluncurkan pada bulan Juni 2021 oleh *Github* (Siroš dkk., 2024). *Copilot* dapat menghasilkan kode dalam berbagai bahasa pemrograman seperti *Java*, *Python*, *C#*, *C++*, dan lainnya. Dalam penggunaannya, *Copilot* akan memberikan beberapa solusi dalam penyelesaian kode yang dikerjakan. Selain itu, *Copilot* juga dapat menghasilkan komentar yang menjelaskan tujuan dan fungsionalitas kode tersebut sesuai dengan proyek yang dikerjakan (Chatterjee dkk., 2024). Hal ini memungkinkan pengembang untuk memilih alternatif yang disarankan, sesuai dengan kebutuhan.

2.1.3 Metode Pengukuran *coverage*

Cakupan kode mengukur seberapa luas kode sumber telah dieksekusi selama kasus pengujian dilakukan dan dalam praktiknya, persentase cakupan kode sering digunakan sebagai pengukuran kecukupan (Tauriainen, 2023; Zhu dkk., 1997). Laporan *code coverage* menunjukkan persentase dari kode yang telah dijalankan selama pengujian, serta memberikan informasi bagian mana yang telah dieksekusi (tercakup) dan bagian mana yang belum (tidak tercakup). Presentasi ini dihitung berdasarkan berbagai jenis metrik cakupan, seperti *line coverage*, *branch coverage*, *statement coverage*, *function coverage*, *condition coverage*, *loop coverage*, dan sebagainya (Tauriainen, 2023). Rentang *code coverage* dimulai dari 0% (tidak

diuji sama sekali) sampai dengan 100% (sepenuhnya diuji). Meskipun beragam metodologi pengujian digunakan untuk menghasilkan *code coverage*, unit test umumnya menjadi metode yang dipilih untuk membuat laporan *coverage*. Karena sifat pengujian integrasi atau pengujian dari *end-to-end* biasanya dilakukan secara paralel, dengan proses yang berjalan asinkron, terkadang melalui utas paralel atau di berbagai mesin atau sistem, membuat *code coverage* untuk metodologi pengujian semacam ini akan lebih rumit (Myers dkk., 2011). Coverage dapat dihitung dengan membandingkan jumlah elemen dari jenis cakupan yang diinginkan dihitung dengan jumlah total kode sumber. Sebagai contoh line coverage dapat dihitung dengan cara membagi jumlah baris yang telah dieksekusi selama pengujian oleh total baris yang ada dalam kode sumber dapat dilihat pada persamaan (2.1) (Tauriainen, 2023).

$$\text{Line Coverage} = \frac{\text{line Executed in a test case}}{\text{Total number of line}} \quad (2.1)$$

Rumus ini bisa digunakan kepada jenis coverage yang lain, sebagai contoh, branch coverage dihitung dari jumlah branch yang dieksekusi dibagi dengan jumlah branch yang ada pada kode sumber.

2.1.4 Kriteria Perbandingan Alat pengujian

Untuk memperoleh pemahaman yang mendalam tentang efektivitas dan efisiensi kinerja *test automation framework* yang digunakan dalam pengujian unit. Terdapat tiga parameter utama untuk membandingkan kinerja *test automation framework* adalah sebagai berikut (Abdulwareth & Al-Shargabi, 2021):

a. Technical and Economic View

Kriteria pertama yang dipertimbangkan oleh pengembang/penguji untuk mengklasifikasi alat untuk *test automation framework* berdasarkan aspek teknis dan ekonomi. Kriteria ini penting karena menentukan apakah alat tersebut secara teknis sesuai untuk perangkat lunak yang diuji dan seberapa mudah alat tersebut digunakan.

b. Testing Process View

Testing process view merupakan *framework* komprehensif untuk memahami dan mengklasifikasikan aktivitas pengujian perangkat lunak. Ini digunakan sebagai dasar untuk memilih alat pengujian yang sesuai, merancang rencana pengujian, dan mengevaluasi hasilnya. *Framework* ini membantu memastikan kualitas perangkat lunak dengan mengidentifikasi aspek-aspek yang perlu diuji dan memberikan panduan untuk pelaksanaannya.

c. Quality Standards View

Quality standards view merupakan kriteria yang penting pada *test automation framework* untuk mengukur kualitas perangkat lunak mencakup beberapa aspek umum seperti efisiensi, maintainabilitas, keamanan, dan Fungsionalitas. Aspek ini penting untuk memastikan perangkat lunak berfungsi dengan baik, dapat dipelihara, dan aman. Berikut adalah parameter yang akan dijelaskan:

- a. Technical and Economic View, yang mencakup beberapa sub parameter sebagai berikut
 1. *Multi Language*
 2. *Cost*
 3. *Community support and Documentation*
- b. Testing Process View, yang mencakup beberapa sub parameter sebagai berikut
 1. Sistem pelacakan bug
 2. Pengujian Otomatis
 3. Pencatatan eksekusi
 4. Pembuatan Skrip
 5. *Reporting*
 6. *Code Coverage*
- c. Quality Standards View, yang mencakup beberapa sub parameter sebagai berikut
 1. *Readability*
 2. *Usability*

2.1.5 Metode Perbandingan The Distance To The Ideal Alternatif

Metode *The Distance to The Ideal Alternative* (DIA) adalah suatu metode yang dimiliki oleh MADM dan didasarkan pada prinsip yang sama dengan metode *Technique for Order Preference by Similarity to Ideal Solution* (Topsis). Metode ini didesain untuk mengatasi beberapa keterbatasan yang dimiliki oleh metode TOPSIS. Dasar Algoritma DIA memiliki prinsip yang mirip dengan metode TOPSIS, DIA menentukan nilai atribut alternatif pada ideal positif dan negatif dari setiap atribut. Perbedaannya terletak pada perhitungan jarak, metode DIA menggunakan *manhattan distance* untuk mengukur jarak antara nilai atribut dan nilai ideal positif dan negatif dari setiap atribut. Penetapan Alternatif Ideal Positif (PIA) dengan nilai minimum D_j^+ , dan nilai maksimal untuk D_j^- serta rumus yang diurutkan berdasarkan nilai R_i sebagai penentu peringkat alternatif (Tran & Boukhatem, 2008). Langkah-langkah yang dilakukan dalam penyelesaian masalah menggunakan metode DIA adalah sebagai berikut (Jun dkk., t.t.-b; Rijoly dkk., 2023):

a. Menentukan matriks keputusan

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \quad (2.2)$$

Dimana:

X adalah matriks keputusan

x_{ij} adalah setiap elemen pada matriks setiap elemen pada matriks $X, i=1,2,3,...,m; j=1,2,3,...,n$.

b. Normalisasi matriks keputusan

Matriks normalisasi diperoleh dari setiap elemen pada matriks keputusan dibagi dengan akar dari jumlah setiap elemen kolom matriks keputusan yang dikuadratkan. Persamaan ini dapat dilihat sebagai berikut:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^m x_{ij}^2}} \quad (2.3)$$

Dimana: r_{ij} adalah setiap elemen matriks normalisasi. x_{ij}^2 adalah nilai kuadrat setiap elemen pada matriks keputusan sehingga diperoleh matriks normalisasi (R):

$$R = \begin{bmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ r_{21} & r_{22} & \cdots & r_{2n} \\ \vdots & \vdots & & \vdots \\ r_{m1} & r_{m2} & \cdots & r_{mn} \end{bmatrix} \quad (2.4)$$

c. Pembobotan pada matriks yang telah dinormalisasi

Setelah proses normalisasi matriks keputusan, kemudian menentukan matriks pembobotan. Dimana setiap elemen matriks pembobotan diperoleh dari hasil setiap elemen bobot dikali dengan setiap elemen pada matriks normalisasi. Persamaan ini dapat dilihat sebagai berikut:

$$v_{ij} = r_{ij} w_i \quad (2.5)$$

Dimana: v_{ij} adalah setiap elemen pada matriks pembobotan. w_i adalah setiap elemen pada matriks(W).

$$V = \begin{bmatrix} r_{11}w_1 & r_{12}w_2 & \cdots & r_{1n}w_n \\ r_{21}w_1 & r_{22}w_2 & \cdots & r_{2n}w_n \\ \vdots & \vdots & \vdots & \vdots \\ r_{m1}w_1 & r_{m2}w_2 & \cdots & r_{mn}w_n \end{bmatrix} \quad (2.6)$$

d. Menentukan solusi ideal positif dan ideal negatif

Solusi ideal positif dinotasikan dengan A^+ dan solusi ideal negatif dinotasikan dengan A^- . Persamaan ini dapat dilihat sebagai berikut:

$$A^+ = \max v_{ij} = [a_1^+, a_2^+, \dots, a_n^+] \quad (2.7)$$

Dimana, $\max v_{ij}$ adalah nilai terbesar dari setiap kolom matrik V.

$$A^- = \min v_{ij} = [a_1^-, a_2^-, \dots, a_n^-] \quad (2.8)$$

Dimana, $\min v_{ij}$ adalah nilai terkecil dari setiap kolom matrik V.

e. Hitung jarak manhattan untuk atribut positif dan negatif

Jarak manhattan atribut positif dinotasikan dengan D_i^+ dan jarak manhattan atribut negatif dinotasikan dengan D_i^- . Persamaan ini dapat dilihat sebagai berikut:

$$D_i^+ = \sum_{j=1}^m [v_{ij} - a_i^+] \quad (2.9)$$

Dimana, a_i^+ adalah nilai dari setiap solusi ideal positif.

$$D_i^- = \sum_{j=1}^m [v_{ij} - a_i^-] \quad (2.10)$$

Dimana, a_i^- adalah nilai dari setiap solusi ideal negatif.

f. Menentukan positif ideal Alternatif (PIA)

PIA minimal dinotasikan dengan $\min (D_i^+)$, nilai ini diperoleh dari pencarian nilai terkecil pada setiap baris D_i^+ . Untuk PIA maksimal dinotasikan dengan $\max (D_i^-)$, dan nilai ini diperoleh dari pencarian nilai terbesar pada setiap baris D_i^- . Persamaan ini dapat dilihat sebagai berikut:

$$PIA = (\min(D_i^+), \max(D_i^-)) \quad (2.11)$$

g. Melakukan identifikasi peringkat

Peringkat dapat ditentukan dengan membandingkan nilai identifikasi peringkat (R_i) yang diperoleh dari persamaan sebagai berikut:

$$R_i = \sqrt{\left(D_i^+ - \min(D_i^+)\right)^2 + \left(D_i^- - \max(D_i^-)\right)^2} \quad (2.12)$$

Nilai R_i yang minimum mengindikasikan bahwa alternatif tersebut lebih terpilih sebagai peringkat yang lebih utama.

2.2 Kajian pustaka

Dalam penelitian ini, kajian pustaka digunakan untuk menganalisis objek yang digunakan dalam penelitian

2.2.1 Penelitian terdahulu

Bagian ini akan membahas penelitian-penelitian sebelumnya yang telah dilakukan melalui berbagai referensi terkait dengan topik penelitian ini. Tujuannya adalah untuk mengidentifikasi perbedaan pada setiap topik yang dipilih oleh penulis. Tabel Penelitian terdapat pada Tabel 2.1

Tabel 2.1 Penelitian Terdahulu

No	Judul	Peneliti dan Tahun	Tujuan	Metode	Hasil
1	A comparative Analysis of Test Automation Frameworks Performance for Functional Testing in Android-Based Applications using the Distance to the Ideal Alternative Method	Calysta Merina, Nenny Anggraini, Nashrul Hakiem. 2018	Untuk membandingkan efektivitas <i>automation testing framework</i> dalam menguji fungsi aplikasi berbasis android.	Distance to the Ideal Alternative (DIA).	Calabash merupakan framework pengujian otomatisasi yang terbaik dan terbukti berdasarkan evaluasi menyeluruh terhadap semua parameter yang diuji(Merina dkk., 2018).
2	A Large Scale Study On the Effectiveness of Manual and Automatic Unit Test Generation	Beatriz Souza, Patricia Machado 2020	Membandingkan efektivitas test suites yang ditulis secara manual (MTSs) dan test suites yang dihasilkan secara otomatis(ATSS) dalam mendeteksi bug.	Empirical Study	penelitian ini menemukan bahwa alat pengujian otomatis (Randoop dan EvoSuite) kurang efektif dibandingkan dengan tes manual dalam hal cakupan baris dan mutasi, dan ukuran suite tes tidak menunjukkan efektivitas, terutama untuk ATS Randoop.
3	A Study of Automated Software Testing: Automation Tools and Frameworks	Mubarak Albarka Umar, Chen Zhanfang 2019	Memberikan pemahaman yang komprehensif terkait alat dan kerangka kerja otomatisasi perangkat lunak.	Comprehensive study, STLC	pengujian adalah tantangan utama untuk mencapai "Kualitas dengan Kecepatan," karena menghemat waktu, biaya, serta meningkatkan efisiensi dan akurasi berdasarkan laporan kualitas dunia 2018-2019
4	Empirical Comparison Between Conventional and AI-based Automated Unit Test Generation Tools in Java	Gkikopouli Marios dan bataa Batjigdel. 2023	memahami kelebihan dan kelemahan dari alat-alat pengujian perangkat lunak berbasis AI, serta membuat keputusan yang tepat dalam mengadopsi dan mengintegrasikan alat-alat tersebut ke dalam proyek-proyek yang dikembangkan.	-	Hasil pada penelitian ini adalah alat generasi tes unit otomatis konvensional seperti Evosuite, umumnya menunjukkan kinerja yang lebih baik daripada alat berbasis kecerdasan buatan dalam hal deteksi kesalahan, cakupan kode, dan skor mutasi.

5	Toward a Multi-Criteria Framework for Selecting Software Testing Tools	Asma J.Abdulwareth , Asma A. Al-shargabi 2021	Menyusun taksonomi yang lengkap untuk mengklasifikasikan alat pengujian perangkat lunak berdasarkan berbagai kriteria yang signifikan	Analytic Hierarchy Process (AHP), TOPSIS.	Menggunakan <i>framework</i> yang diusulkan akan mempermudah pemilihan alat pengujian yang mencakup taksonomi komprehensif dan metode seleksi.
---	--	--	---	---	--

2.2.2 Telaah kajian Pustaka

Pada bagian ini, dilakukan telaah terhadap penelitian terdahulu yang relevan untuk memperkuat landasan teori dalam penelitian ini. Telaah difokuskan pada penelitian yang dilakukan oleh Calysta Merina, Nenny Anggraini, dan Nashrul Hakiem pada tahun 2018, yang berjudul “A comparative Analysis of Test Automation Frameworks Performance for Functional Testing in Android-Based Applications using the Distance to the Ideal Alternative Method”. Dalam jurnal ini berfokus pada perbandingan efektivitas *framework* pengujian otomatis dalam menguji fungsi aplikasi berbasis android, terdapat tiga *framework* pengujian otomatis yang dijadikan perbandingan dalam penelitian ini yaitu Espresso, Calabash, dan Appium. Metode yang digunakan adalah *Distance to The Ideal Alternative (DIA)* untuk menilai dan membandingkan performa setiap *framework* berdasarkan parameter yang telah ditentukan.

Pada penelitian yang dijalankan oleh penulis memiliki perbedaan yang signifikan dari jurnal yang disebutkan sebelumnya. Pertama, fokus utama pada jurnal tersebut adalah aplikasi web, sedangkan dalam penelitian yang dilakukan oleh penulis lebih difokuskan pada pengujian unit (*unit testing*). Kedua, *framework* pengujian otomatis yang digunakan penulis pada penelitian ini lebih sesuai dengan *unit testing*, berbeda dengan jurnal yang disebutkan sebelumnya. Ketiga, penulis melakukan penilaian dengan menggunakan parameter dan sub-parameter yang lebih rinci untuk mengukur dan membandingkan efektivitas setiap *framework* pengujian otomatis yang berfokus pada *unit testing*. Parameter digunakan sebagai bahan untuk evaluasi yang lebih terukur dan objektif terhadap performa dari setiap *framework*.

Untuk mendapatkan data yang relevan, penelitian ini menggunakan pendekatan empiris, yang melibatkan programmer sebagai responden utama dalam survei. Pendekatan ini dipilih karena memungkinkan pengumpulan informasi langsung dari para ahli yang memiliki pengalaman dalam penggunaan berbagai *framework* pengujian uni. Data yang dikumpulkan dalam survei ini akan dianalisis untuk mengidentifikasi preferensi dan efektivitas setiap *framework* dari perspektif yang berbeda.

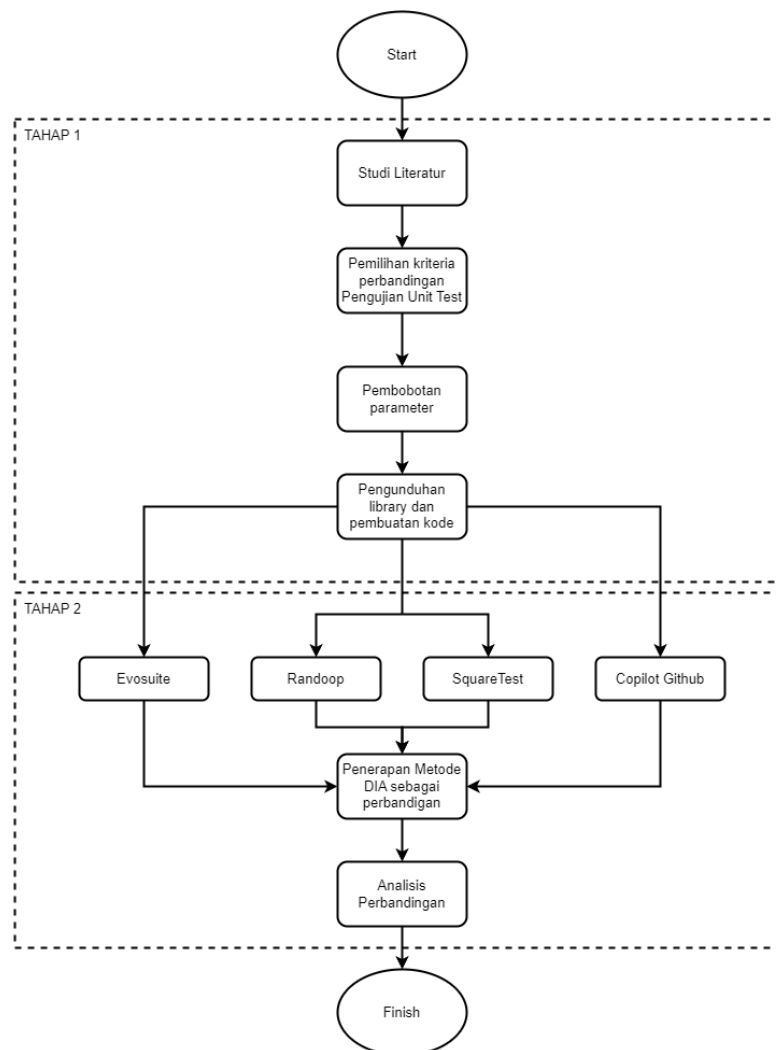
Oleh karena itu, telaah kajian pustaka ini diharapkan memberikan pemahaman yang lebih komprehensif dan berbasis bukti tentang efektivitas dan penggunaan pengujian otomatis dalam lingkup yang berbeda.

BAB III

METODOLOGI PENELITIAN

3.1 Alur Proses

Seperti yang terdapat Gambar 3.1 Alur Proses, penelitian ini dilakukan melalui dua tahapan utama. Tahap pertama mencakup analisis studi literatur untuk mengumpulkan informasi terkait dengan *framework* pengujian unit otomatis yang digunakan dan parameter perbandingan. Selanjutnya, melakukan pemilihan parameter perbandingan yang didasarkan pada hasil literatur dan melakukan pembobotan nilai, dan terakhir melakukan pembuatan kode dan pengunduhan *library*. eksplorasi terhadap data yang telah didapatkan. Aktivitas-aktivitas pada tahap ini akan dijelaskan lebih rinci pada sub bab berikutnya. Tahap kedua adalah melakukan pengujian unit menggunakan keempat alat dan melakukan perbandingan dengan metode yang sudah ditentukan. Detail mengenai aktivitas pada tahap kedua akan diuraikan pada Bab IV.



Gambar 3.1 Alur Proses

3.2 Studi Literatur

Dalam penelitian ini, diperlukan bahan, referensi, dan data untuk mendukung pembahasan yang sedang dilakukan. Pengumpulan data dan informasi yang relevan dengan topik penelitian diharapkan dapat memperkuat validitas dan kebenaran hasil penelitian. Melalui studi literatur, yang mencakup penelaahan dan analisis penelitian serupa yang telah dilakukan sebelumnya, diharapkan dapat menyediakan referensi, informasi, dan landasan teoritis yang kuat. Literatur yang diacu meliputi jurnal-jurnal baik nasional maupun internasional.

3.3 Pemilihan Parameter Perbandingan *Framework Unit Test*

Pemilihan parameter alat *unit test* merupakan langkah yang sangat penting dan diprioritaskan dalam penelitian ini untuk memastikan kualitas dan keandalan kode. Setiap alat memiliki fitur dan kemampuan yang berbeda, yang dapat mempengaruhi efektivitas proses pengujian. Oleh karena itu, menetapkan parameter yang berasal dari studi literatur yang

ditelaah dapat memberikan panduan yang komprehensif dan berbasis bukti pada penelitian ini. Dengan merujuk pada penelitian terdahulu terdapat tiga parameter perbandingan yang masing-masing mencakup kriteria-kriteria spesifik.

Parameter yang dijadikan sebagai acuan dalam penelitian ini diambil dari penelitian terdahulu adalah *Technical Economic View*, *Testing process View*, dan *Quality standards View* (Abdulwareth & Al-Shargabi, 2021). Ketiga parameter itu dipilih karena mencakup berbagai aspek penting dalam menilai kinerja *framework* pengujian otomatis terutama dalam konteks *unit test*. Mulai dari aspek teknis, proses pengujian, hingga standar kualitas yang dihasilkan. Dengan mempertimbangkan parameter ini, penelitian dapat memberikan penilaian menyeluruh tentang efektivitas dan efisiensi *framework* pengujian otomatis. Parameter pertama adalah parameter *Technical Economic View* yaitu Kriteria pertama yang dipertimbangkan oleh pengembang/penguji untuk mengklasifikasi alat untuk *test automation framework* berdasarkan aspek teknis dan ekonomi. Adapun sub-parameter *Technical Economic View* adalah sebagai berikut:

a. *Multi Language*

Kemampuan *framework* pengujian untuk mendukung berbagai bahasa pemrograman. Ini memungkinkan *framework* dapat menulis dan menjalankan tanpa batasan satu bahasa tertentu.

b. *Cost*

Biaya yang diperlukan pada *framework*. Apakah biaya tersebut gratis atau bayar

c. *Community Support and Documentation*

Mempunyai dukungan komunitas dan dokumentasi, garansi, instalasi, pembaruan, serta layanan bantuan .

Parameter kedua adalah *Testing Process View* yaitu merupakan *framework* komprehensif untuk memahami dan mengklasifikasikan aktivitas pengujian perangkat lunak. Ini digunakan sebagai dasar untuk memilih alat pengujian yang sesuai, merancang rencana pengujian, dan mengevaluasi hasilnya. Adapun sub-parameter *Testing Process View* adalah sebagai berikut:

a. *Sistem Pelacakan Bug*

Framework terdapat mekanisme untuk mencatat, melacak, dan mengelola *bug*.

b. *Pengujian Otomatis*

Framework dapat menjalankan test secara otomatis tanpa memerlukan intervensi manual.

c. *Execution Logging*

Apakah terdapat proses pencatatan untuk semua aktivitas dan hasil selama eksekusi pengujian.

d. *Script Creation*

Seberapa lama waktu yang dibutuhkan untuk menulis skrip pada *framework*.

e. *Reporting*

Framework menyediakan laporan yang merangkum hasil dari pengujian yang dilakukan.

f. *Code Coverage*

Apakah *framework* memberikan rincian *code coverage* untuk setiap blok kode atau metode dalam kode.

Parameter ketiga adalah *Quality Standards View* merupakan kriteria yang penting pada *test automation framework* untuk mengukur kualitas perangkat lunak mencakup beberapa aspek umum seperti efisiensi, maintainabilitas, keamanan, dan Fungsionalitas. Adapun sub-parameter *Quality Standards View* sebagai berikut:

a. *Readability*

Seberapa mudah kode dapat dibaca dan dipahami oleh manusia, khususnya oleh pengembang.

b. *Usability*

Usability terkait apakah *framework* mudah digunakan, waktu pelatihannya singkat, dan kurva pembelajarannya rendah. Pada bagian ini untuk mempermudah mencari penilaian akan digunakan metode System Usability Scale (SUS)(Brooke, 2013). Berikut merupakan pertanyaan yang digunakan

1. Saya merasa tertarik untuk menggunakan sistem ini setelah melihat demo
2. Saya merasa bingung menggunakan sistem ini
3. Saya merasa dapat dengan cepat memahami cara kerja sistem ini
4. Saya merasa membutuhkan banyak bantuan untuk mempelajari cara menggunakan sistem ini
5. Saya merasa fitur-fitur dalam sistem ini berjalan dengan baik
6. Saya merasa akan mengalami kesulitan saat menggunakan sistem ini
7. Saya merasa sistem ini mudah dipahami
8. Saya merasa Membutuhkan waktu yang lama untuk mempelajari dan menggunakan sistem ini
9. Saya merasa instruksi atau panduan yang diberikan cukup untuk saya memahami sistem ini

10. Saya merasa bahwa orang lain membutuhkan waktu lebih lama untuk memahami cara menggunakan sistem ini.

Pertanyaan tersebut disesuaikan dengan cara kerja *System Usability Scale* dengan membuat pernyataan positif pada pertanyaan ganjil dan membuat pernyataan negatif pada pertanyaan genap agar memudahkan dalam perhitungan (Brooke, 2013; Sauro & Lewis, 2011).

3.4 Pembobotan parameter dan nilai

Pada tahap ini, pembobotan parameter dilakukan dengan menentukan bobot untuk parameter dan sub parameter yang digunakan sebagai pembanding untuk menilai kinerja setiap *framework* menggunakan metode *The Distance to Ideal Alternative (DIA)*. Sub parameter dalam penelitian ini dikategorikan menjadi dua tipe. Untuk sub parameter dengan tipe *benefit*, penentuan bobot didasarkan pada nilai yang semakin besar semakin baik, karena nilai tersebut mencerminkan relevansi dari sub parameter yang diukur, seperti pada sub parameter *code coverage* untuk mengukur seberapa besar cakupan kode. Sedangkan untuk sub parameter dengan tipe *cost*, penentuan bobot didasarkan pada nilai yang semakin kecil semakin baik, karena nilai kecil mencerminkan efisiensi, nilai tersebut juga mencerminkan relevansi dari sub parameter yang diukur, seperti pada sub parameter *script creation* yang mengukur waktu yang diperlukan untuk membuat skrip pengujian. Prinsip-prinsip *Multiple Attribute Decision Making (MADM)* akan digunakan dalam penelitian ini. Bobot yang lebih besar menunjukkan pengaruh yang signifikan terhadap hasil akhir, dengan total bobot yang harus berjumlah 1 sesuai dengan aturan metode DIA. Proses pengambilan data untuk pembobotan dilakukan dengan cara menyebar kuesioner kepada programmer yang memiliki pengalaman di bidang *unit testing*. Kuesioner ini disebarkan kepada responden dengan tujuan untuk mengetahui prioritas yang diperlukan dalam *automation framework*, sehingga dari data yang diperoleh akan diidentifikasi parameter-parameter yang paling berpengaruh dalam meningkatkan efektivitas dan efisiensi pengujian otomatis.

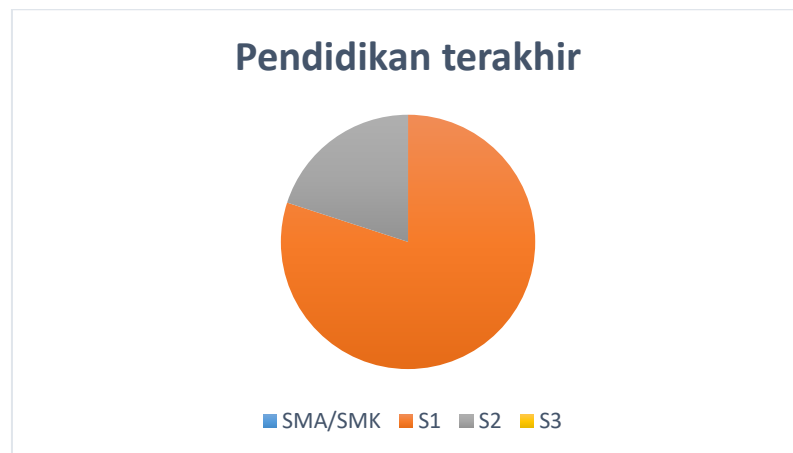
3.4.1 Demografi Responden

Penelitian ini melibatkan responden berjumlah 10 orang yang memiliki pengalaman di bidang *unit testing*. Setiap responden memiliki latar belakang pendidikan dan pengalaman kerja yang variatif. Variasi dalam lamanya pengalaman kerja dan latar belakang yang dimiliki oleh responden memberikan wawasan yang mendalam terhadap dinamika kerja dalam industri pengembangan perangkat lunak. Data ini menunjukkan bahwa sebagian besar responden telah memiliki pengalaman yang cukup lama, sementara sebagian lainnya baru memulai bekerja

dalam bidang ini, namun telah terlibat dalam beberapa tugas pengembangan dan juga pengelolaan aplikasi.

a. Pendidikan terakhir

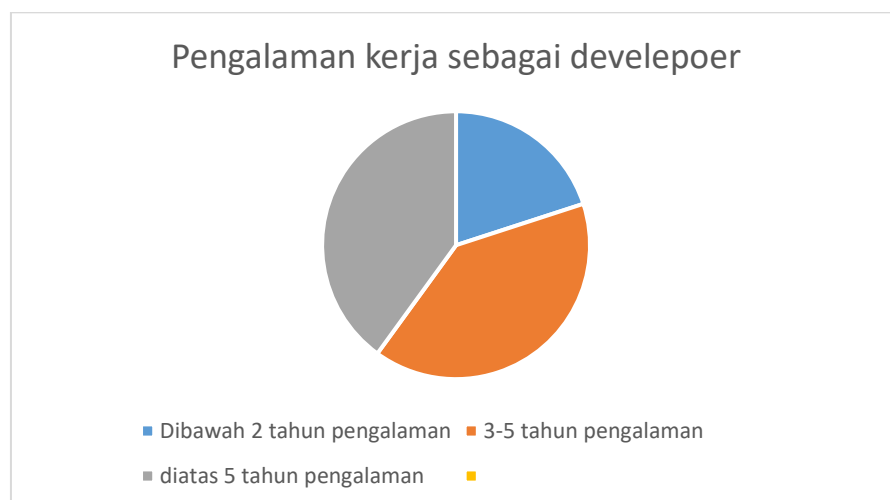
Pada Gambar 3.2 menunjukkan bahwa mayoritas responden memiliki latar belakang pendidikan terakhir sebagai sarjana S1 yaitu sebanyak 80% dan 20% responden memiliki latar belakang pendidikan terakhir sebagai sarjana s2.



Gambar 3.2 Pendidikan terakhir responden

b. Pengalaman kerja

Berdasarkan data yang diperoleh, 40% responden memiliki pengalaman kerja di atas 5 tahun, 40% memiliki pengalaman kerja antara 3 hingga 5 tahun, dan 20% dari responden memiliki pengalaman kerja di bawah dua tahun pengalaman.



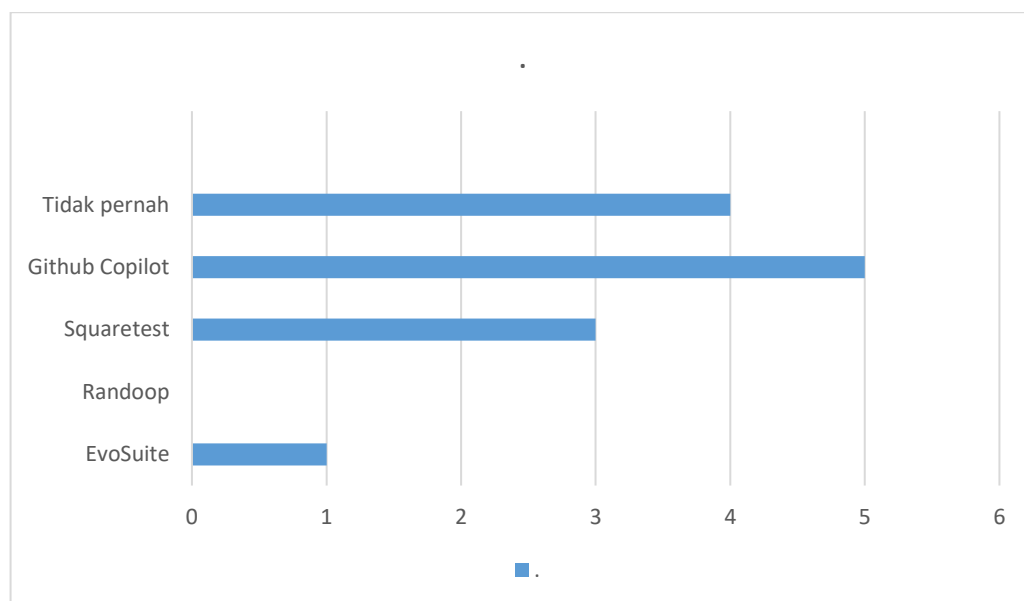
Gambar 3.3 Pengalaman kerja responden

c. Bahasa yang dikuasai dan *framework* yang digunakan

Berdasarkan hasil kuesoner, sebanyak 80% responden menguasai bahasa pemrograman Java, dan 60% responden menguasai bahasa JavaScript. Sementara itu, bahasa-bahasa lain seperti Dart, C#, C++, Ruby, Kotlin, Go, dan PHP dikuasai oleh 10% hingga 20% responden. Sebanyak 10% responden menguasai bahasa python. Selanjutnya, data berikutnya adalah pengalaman responden dalam menggunakan *framework unit test*. *Framework* JUnit menjadi yang paling banyak digunakan oleh responden dengan persentase 20%, diikuti oleh *framework* seperti Cypress, Jest, JUnit Testify, PHPUnit, Pest, dan Squaretest yang masing-masing digunakan oleh 10% responden. Sebanyak 10% responden menyatakan tidak menggunakan *framework* pengujian *unit test*.

d. Preferensi Responden terhadap framework pengujian otomatis

Berdasarkan pada Gambar 3.4, sebanyak 50% responden menyatakan pernah menggunakan Github Copilot sebagai framework untuk pengujian *unit test*. 30% responden juga pernah menggunakan Squaretest dalam pengujian *unit test*. Namun, hanya 10% responden menyatakan pernah menggunakan EvoSuite, dan tidak ada responden yang pernah menggunakan Randoop. Sebanyak 40% responden menyatakan bahwa tidak menggunakan framework pengujian otomatis dari daftar yang diberikan.



Gambar 3.4 Grafik penggunaan framework pengujian otomatis

3.4.2 Pemberian bobot Parameter dan sub parameter

Bobot untuk parameter dan sub parameter dihitung berdasarkan jawaban responden untuk mengukur tingkat kepentingan relatif parameter dan sub parameter. Untuk parameter yang terdiri dari tiga sub-parameter atau kurang, skema yang digunakan memberikan tiga poin

untuk pilihan yang dianggap paling penting, dua poin untuk yang kedua, dan satu poin untuk yang ketiga. Sementara itu, Dalam parameter '*Testing Process View*' yang memiliki lebih dari tiga sub-parameter, skema penilaian diperbesar hingga keenam, memberikan poin dari enam untuk yang paling penting hingga satu untuk yang paling tidak penting. Bobot untuk setiap parameter dan sub-parameter kemudian dinormalisasikan menggunakan persamaan (3.1) untuk parameter dan persamaan (3.2) untuk parameter, sehingga dari persamaan tersebut totalnya menjadi satu untuk setiap parameter dan sub parameter.

$$\text{Bobot Sub parameter} = \frac{\text{Total poin sub parameter}}{\text{Total poin dari semua sub parameter}} \quad (3.1)$$

$$\text{Bobot Parameter} = \frac{\text{Total poin Parameter}}{\text{Total poin dari semua Parameter}} \quad (3.2)$$

3.4.3 Parameter Technical Economic View

Dari hasil kuesoner yang telah dikumpulkan langkah selanjutnya adalah menghitung nilai bobot untuk setiap sub parameter. Bobot tersebut ditentukan dengan menggunakan skema yang telah ditentukan sebelumnya dan dinormalisasikan menggunakan persamaan (3.1) . Hasilnya sebagai berikut :

Tabel 3.1 Bobot pada parameter Technical Economic View

Sub Parameter	Jumlah Responden (pilihan)	Total Skor	Tipe	Bobot
Multi Language	2 (pertama), 3 (kedua), dan 5 (ketiga)	17	<i>Benefit</i>	0.27
Cost	4 (pertama), 3 (kedua), dan 3 (ketiga)	21	<i>Cost</i>	0.34
Community Support and Documentation	5 (pertama), 4 (kedua), dan 1 (ketiga)	24	<i>Benefit</i>	0.39

Pembobotan nilai sub parameter *Scripting Language*

Pada Tabel 3.2 ditampilkan bobot untuk nilai sub parameter *Multi Language* yang akan memudahkan dalam penyusunan matriks keputusan sesuai dengan ketentuan metode DIA.

Tabel 3.2 Pembobotan nilai sub parameter *Multi Language*

Jumlah bahasa	Rentang nilai
$5 \leq x$	1
$4 \leq x < 5$	0.75

$3 \leq x < 3$	0.5
$2 \leq x < 3$	0.25
$1 \leq x < 2$	0

Pembobotan nilai sub parameter *Cost*

Tabel 3.3 Pembobotan nilai sub parameter *Cost*

Sub parameter	Rentang nilai	
<i>Cost</i>	<i>Framework</i> gratis = 1	<i>Framework</i> berbayar = 0

Pada sub parameter *Cost*, *framework* yang gratis diberikan bobot nilai 1, sedangkan *framework* berbayar diberikan bobot nilai 0, sebagaimana ditampilkan pada tabel Tabel 3.3.

Pembobotan nilai sub parameter *Community Support and Documentation*

Tabel 3.4 Pembobotan nilai sub parameter *Community Support and Documentation*

Sub parameter	Rentang nilai	
<i>Community Support and Documentation</i>	<i>Framework</i> yang memiliki dukungan komunitas dan dokumentasi lengkap = 1	<i>Framework</i> yang tidak memiliki dukungan komunitas dan dokumentasi lengkap = 0

Pada sub parameter *Community Support and Documentation*, *framework* yang memiliki dukungan komunitas dan dokumentasi lengkap diberi bobot nilai 1, sedangkan *framework* yang tidak memiliki dukungan komunitas dan dokumentasi lengkap diberi bobot nilai 0.

3.4.4 Parameter Testing Process View

Tabel 3.5 Bobot pada parameter Testing Process view

Sub Parameter	Jumlah Responden (pilihan)	Total Skor	Tipe	Bobot
Sistem Pelacakan Bug	1 (pertama), 4 (kedua), 2 (ketiga), 2 (keempat), dan 1 (kelima)	42	<i>Benefit</i>	0.19
Pengujian Otomatis	5 (pertama), 2 (kedua), 1 (ketiga), 1 (keempat), dan 1 (keenam)	48	<i>Benefit</i>	0.22
<i>Execution Logging</i>	1 (kedua), 2 (ketiga), 2 (keempat), 2 (kelima), dan 3 (keenam)	26	<i>Benefit</i>	0.12

<i>Script Creation</i>	1 (orang), 2 (kedua), 2 (ketiga), 4 (kelima), 1 (keenam)	33	<i>Cost</i>	0.15
<i>Reporting</i>	1 (pertama), 1 (kedua), 2 (ketiga), 3 (keempat), 1 (kelima), dan 2 (keenam)	32	<i>Benefit</i>	0.15
<i>Code Coverage</i>	3 (pertama), 1 (kedua), 2 (ketiga), 1 (keempat), 1 (kelima), dan 2 (keenam)	38	<i>Benefit</i>	0.17

Pembobotan nilai sub parameter Sistem Pelacakan Bug

Tabel 3.6 Pembobotan nilai sub parameter Sistem Pelacakan Bug

Sub parameter	Rentang nilai	
Sistem Pelacakan Bug	<i>Framework</i> yang memiliki sistem pelacakan bug = 1	<i>Framework</i> yang tidak memiliki sistem pelacakan bug = 0

Pada sub parameter Sistem Pelacakan Bug, *framework* yang memiliki fitur pelacakan bug diberi bobot nilai 1, sedangkan *framework* yang tidak memiliki fitur pelacakan bug diberi bobot nilai 0.

Pembobotan nilai sub parameter Pengujian Otomatis

Tabel 3.7 Pembobotan nilai sub parameter Pengujian Otomatis

Sub parameter	Rentang nilai	
Pengujian Otomatis	<i>Framework</i> yang dapat melakukan pengujian otomatis = 1	<i>Framework</i> yang tidak dapat melakukan pengujian otomatis = 0

Pada sub parameter Pengujian Otomatis, *framework* yang dapat melakukan pengujian otomatis diberi bobot nilai 1, sedangkan *framework* yang tidak dapat melakukan pengujian otomatis diberi bobot nilai 0.

Pembobotan nilai sub parameter *Execution Logging*

Tabel 3.8 Pembobotan nilai sub parameter *Execution Logging*

Sub parameter	Rentang nilai
---------------	---------------

<i>Execution Logging</i>	<i>Framework</i> yang memiliki pencacatan selama program berjalan = 1	<i>Framework</i> yang tidak memiliki pencacatan selama program berjalan = 0
--------------------------	---	---

Pada sub parameter *Execution Logging*, *framework* yang memiliki fitur pencacatan selama program berjalan diberi bobot nilai 1, sedangkan *framework* yang tidak memiliki fitur pencacatan selama program berjalan diberi bobot nilai 0.

Pembobotan nilai sub parameter *Script Creation*

Pada Tabel 3.9 ditampilkan bobot untuk nilai sub parameter *Script Creation*.

Tabel 3.9 Pembobotan nilai sub parameter *Script Creation*

Rentang waktu (menit)	Rentang nilai
$x < 1$	1
$1 \leq x < 2$	0.75
$2 \leq x < 3$	0.50
$3 \leq x < 4$	0.25
$4 \leq x < 5$	0

Pembobotan nilai sub parameter *Reporting*

Tabel 3.10 Pembobotan nilai sub parameter *Reporting*

Sub parameter	Rentang nilai	
<i>Reporting</i>	<i>Framework</i> memiliki laporan hasil eksekusi program = 1	<i>Framework</i> tidak memiliki hasil eksekusi program = 0

Pada sub parameter *Reporting*, *framework* yang memiliki fitur laporan hasil eksekusi diberi bobot nilai 1, sedangkan *framework* yang tidak memiliki fitur laporan hasil eksekusi diberi bobot nilai 0.

Pembobotan nilai sub parameter *Code Coverage*

Pada Tabel 3.11 ditampilkan bobot untuk nilai sub parameter *Code Coverage*.

Tabel 3.11 Pembobotan nilai sub parameter *Code Coverage*

Code Coverage	Rentang nilai
$x > 80$	1
$70 \leq x < 80$	0.75
$60 \leq x < 70$	0.5

$50 \leq x < 60$	0.25
$50 < x$	0

3.4.5 Parameter Quality Standards View

Tabel 3.12 Bobot pada parameter *Quality Standard View*

Sub Parameter	Jumlah responden (pilihan)	Total skor	Tipe	Bobot
<i>Readability</i>	5 (pertama), 5 (kedua)	15	<i>Benefit</i>	0.48
<i>Usability</i>	6 (pertama), 4 (kedua)	16	<i>Benefit</i>	0.52

Pembobotan nilai sub parameter *Readability*

Pada Tabel 3.13 ditampilkan bobot untuk nilai sub parameter *Readability*.

Tabel 3.13 Pembobotan nilai sub parameter *Readability*

Deskripsi	Rentang nilai
$x \geq 4.5$	1
$3.5 \leq x < 4.5$	0.75
$2.5 \leq x < 3.5$	0.50
$1.5 \leq x < 2.5$	0.25
$x < 1.5$	0

Pembobotan nilai sub parameter *Usability*

Pada Tabel 3.14 ditampilkan bobot untuk nilai sub parameter *Usability*, yang ditentukan berdasarkan skor System Usability Scale (SUS). Standar rata-rata skor SUS adalah 68 (Sauro & Lewis, 2011).

Tabel 3.14 Rentang nilai *Usability*

Rentang SUS score	Rentang nilai
$x \geq 80$	1
$70 \leq x < 80$	0.75
$60 \leq x < 70$	0.5
$50 \leq x < 60$	0.25
$x < 50$	0

3.4.6 Pembobotan seluruh parameter utama

Tabel 3.15 Pembobotan parameter utama

Sub Parameter	Jumlah responden (pilihan)	Total skor	Bobot
Technical Economic View	1 (pertama), 4 (kedua), dan 5 (ketiga)	16	0.26
Testing Process View	6 (pertama), 2 (kedua), dan 2 (ketiga)	24	0.39
Quality Standards View	4 (pertama), 4 (kedua), dan 2 (ketiga)	22	0.35

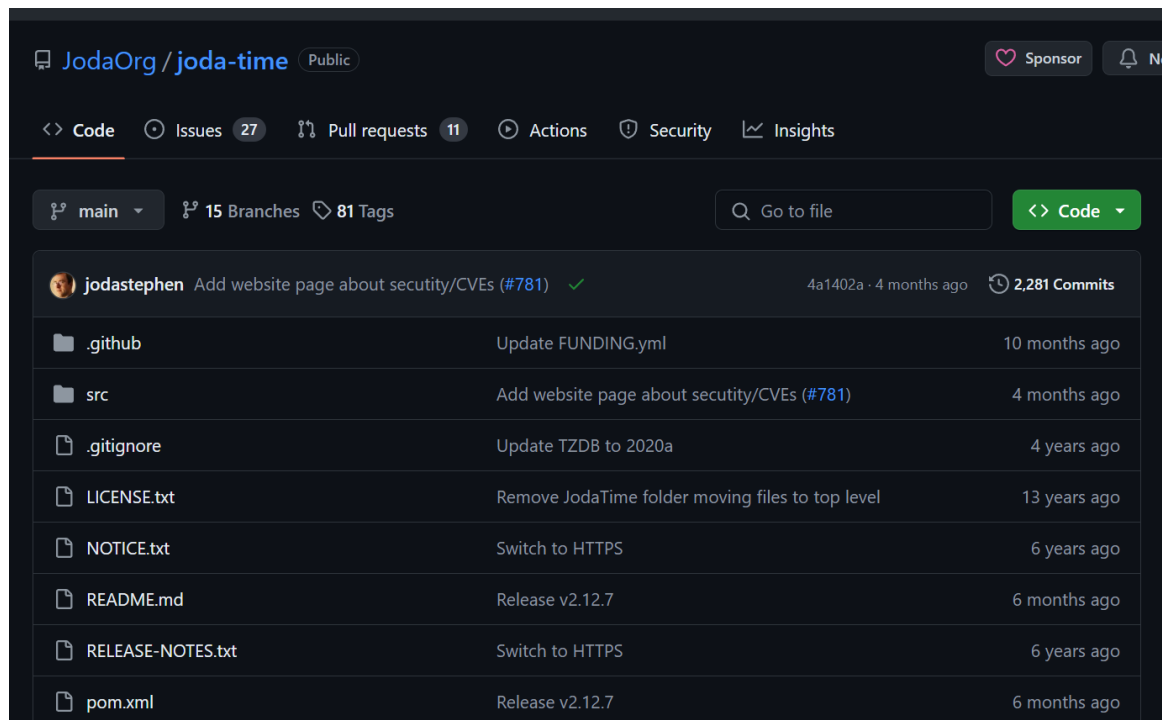
Pada parameter *Technical Economic View* bobot nilai adalah 0.26, kemudian untuk parameter *Testing Process View* adalah 0.39, dan parameter terakhir yaitu *Quality Standard View* adalah 0.35.

3.5 Pengunduhan Library dan Pembuatan kode

Pada tahap ini, dilakukan pengunduhan library Joda-Time dari repository GitHub dan pembuatan kode dilakukan dengan memanfaatkan library yang telah diunduh, memastikan proyek dijalankan dengan spesifikasi yang diinginkan. Berikut ini langkah-langkah dalam pembuatan kode.

a. Pencarian library dan integrasi ke dalam kode

Pada Gambar 3.5 ditunjukkan tampilan repository Joda-Time, library ini diunduh dari GitHub dan diintegrasikan ke dalam proyek yang telah dibuat.



Gambar 3.5 Library Joda-Time

Integrasi library ke dalam pom.xml

```
<dependency>
  <groupId>joda-time</groupId>
  <artifactId>joda-time</artifactId>
  <version>2.12.5</version>
</dependency>
```

Gambar 3.6 Depedensi Joda Time

Pembuata Kode

Pada tahap ini penulis membuatkan kode sebagai bahan untuk diuji menggunakan keempat framework, di sini penulis hanya akan menampilkan kode-kode yang penting termasuk bug yang dimasukan oleh penulis untuk mengetahui apakah framework dapat melacak bug atau tidak.

b. Pembuatan kelas dengan nama ‘Acara’

```
1 package org.example;
2
3 import org.joda.time.DateTime;
4 import org.joda.time.DateTimeZone;
5 import org.joda.time.format.DateTimeFormat;
6 import org.joda.time.format.DateTimeFormatter;
7
8 import java.io.Serializable;
9
```

```

10 public class Acara implements Serializable {
11     private static final long serialVersionUID = 1L;
12
13     private String judul;
14     private String deskripsi;
15     private DateTime waktuMulai;
16     private DateTime waktuSelesai;
17     private static final DateTimeFormatter format =
18     DateTimeFormat.forPattern("dd/MM/yyyy HH:mm")
19         .withZone(DateTimeZone.forID("Asia/Jakarta"));
20
21     public Acara(String judul, String deskripsi, DateTime waktuMulai,
22     DateTime waktuSelesai){
23         if (waktuMulai == null){
24             throw new IllegalArgumentException("waktu mulai tidak boleh
25 null");
26         }
27         if (waktuSelesai == null){
28             throw new IllegalArgumentException("waktu selesai tidak boleh
29 null");
30         }
31         if(judul == null || deskripsi == null){
32             throw new IllegalArgumentException("Judul dan deskripsi tidak
33 boleh kosong");
34         }
35         if (waktuMulai.isAfter(waktuSelesai)){ // Cek apakah waktu mulai
36 lebih awal dari waktu selesai //jika ingin menggunakan bug maka mengubah
37 isAfter menjadi isBefore
38             throw new IllegalArgumentException("Waktu mulai harus diisi
39 sebelum waktu selesai");
40         }
41         this.judul = judul;
42         this.deskripsi = deskripsi;
43         this.waktuMulai = waktuMulai;
44         this.waktuSelesai = waktuSelesai;
45
46         ... ..
47
48         // Cek apakah event ini bertabrakan dengan event lain
49         public boolean isOverlapping(Acara other) {
50             return !(this.waktuSelesai.isBefore(other.waktuMulai) ||
51 this.waktuMulai.isAfter(other.waktuSelesai));
52         }
53
54         @Override
55         public String toString() {
56             // DateTimeFormatter format = DateTimeFormat.forPattern("dd/MM/yyyy
57 HH:mm").withLocale(new Locale("in", "ID"));
58             return "Acara{" +
59                 "Judul='" + judul + '\'' +
60                 ", Deskripsi='" + deskripsi + '\'' +
61                 ", waktuMulai=" + format.print(waktuMulai) +
62                 ", waktuSelesai=" + format.print(waktuSelesai) +
63                 '}';
64         }
65     }

```

Gambar 3.7 Kode pada kelas Acara

c. Pembuatan kelas dengan nama ‘Jadwal’

Pada kelas Jadwal, penulis sengaja menambahkan bug pada metode ‘tambahAcara’ yang terletak di baris ke-19 dan pada metode ‘cariEvents’ yang terletak di baris ke-31. Bug ini dimaksudkan sebagai bahan uji untuk mengetahui apakah framework yang digunakan dapat

mendeteksi bug atau tidak. Selain itu, kelas ini juga memiliki beberapa metode lainnya yang diuji.

```

1  public void tambahAcara(Acara acara){
2      DateTimeZone zone = DateTimeZone.forID("Asia/Jakarta");
3      DateTime waktuMulaiJakarta = new
4      DateTime(acara.getWaktuMulai(), zone);
5      DateTime waktuSelesaiJakarta = new
6      DateTime(acara.getWaktuSelesai(), zone);
7      Interval newEventInterval = new Interval(waktuMulaiJakarta,
8      waktuSelesaiJakarta);
9      for (Acara existingEvent : acaraS) {
10         DateTime existingWaktuMulaiJakarta = new
11         DateTime(existingEvent.getWaktuMulai(), zone);
12         DateTime existingWaktuSelesaiJakarta = new
13         DateTime(existingEvent.getWaktuSelesai(), zone);
14         Interval existingEventInterval = new
15         Interval(existingWaktuMulaiJakarta,
16         existingWaktuSelesaiJakarta);
17         // Bug ini akan selalu melempar exception walau acara
18         yang dibuat tidak tumpang tindih
19         if (!newEventInterval.overlaps(existingEventInterval)){
20             throw new IllegalArgumentException("Acara baru
21             bertepatan dengan acara yang sudah ada");
22         }
23     }
24     acaraS.add(acara);
25 }
26
27 public List<Acara> cariEvents(String kataKunci){
28     return acaraS.stream()
29         // Bug: Mengembalikan acara yang tidak sesuai dengan
30         kata kunci
31         .filter(e -> !e.getJudul().contains(kataKunci) &&
32         !e.getDeskripsi().contains(kataKunci))
33         .collect(Collectors.toList());
34 }
35
36 public boolean hapusAcaraBerdasarkanJudul(String judul) {
37     for (Iterator<Acara> iterator = acaraS.iterator();
38     iterator.hasNext();) {
39         Acara acara = iterator.next();
40         if (acara.getJudul().equalsIgnoreCase(judul)) {
41             iterator.remove();
42             return true; // Event ditemukan dan dihapus
43         }
44     }
45     return false; // Tidak ada event dengan judul tersebut
46 }
47 public void lihatJadwal(){
48     DateTimeZone zone = DateTimeZone.forID("Asia/Jakarta");
49     DateTimeFormatter formatter =
50     DateTimeFormat.forPattern("dd/MM/yyyy HH:mm");
51     if (acaraS.isEmpty()){
52         System.out.println("Tidak ada event di jadwal");
53         return;
54     }
55     acaraS.forEach(event -> System.out.println(event));
56 }
57
58 @Test
59 void hapusEventShouldRemoveEvent() {
60     Jadwal jadwal = new Jadwal();
61     DateTime waktuMulai = new DateTime(2023, 10, 10, 9,
62     0);
63 }

```

```

64     DateTime waktuSelesai = new DateTime(2023, 10, 10, 10,
65     0);
66     Acara acara = new Acara("Judul", "Deskripsi",
67     waktuMulai, waktuSelesai);
68     jadwal.tambahAcara(acara);
69     assertTrue(jadwal.hapusEvent(acara));
70     assertEquals(0, jadwal.getEvents().size());
71 }
72
73
74

```

Gambar 3.8 kode pada kelas Jadwal

d. Pembuatan kelas dengan nama ‘ManajemenJadwal’

Pada kelas ManajemenJadwal, penulis sengaja menambahkan bug pada metode ‘tambahAcara’ yang terletak di baris ke-8. Bug ini dimaksudkan sebagai bahan uji untuk mengetahui apakah framework yang digunakan dapat mendeteksi bug atau tidak. Selain itu, kelas ini juga memiliki beberapa metode lainnya yang diuji.

```

1  public void tambahEvent() {
2      System.out.print("Masukan judul: ");
3      String judul = scanner.nextLine();
4
5      System.out.print("Masukan deskripsi: ");
6      String deskripsi = scanner.nextLine();
7      //bug format tanggal yang berbeda dengan format yang diinginkan
8      DateTimeFormatter format = DateTimeFormat.forPattern("dd-MM-yyyy
9      HH:mm").withZone(DateTimeZone.forID("Asia/Jakarta"));
10     DateTime waktuMulai = null;
11     DateTime waktuSelesai = null;
12
13     while (true) {
14         try {
15             System.out.print("Masukan waktu mulai dd/MM/yyyy: ");
16             waktuMulai = format.parseDateTime(scanner.nextLine());
17             System.out.print("Masukan waktu selesai dd/MM/yyyy: ");
18             waktuSelesai = format.parseDateTime(scanner.nextLine());
19
20
21             Acara acara = new Acara(judul, deskripsi, waktuMulai,
22             waktuSelesai);
23
24             for (Acara existingEvent : jadwal.getEvents()) {
25                 if (acara.isOverlapping(existingEvent)) {
26                     System.out.println("Acara bertabrakan dengan acara yang
27                     sudah ada.");
28                     return;
29                 }
30             }
31
32             jadwal.tambahAcara(acara);
33             System.out.println("Acara berhasil ditambahkan.");
34             simpanJadwal();
35             break;
36         } catch (IllegalArgumentException e) {
37             System.out.println("Error: " + e.getMessage() + " Silahkan
38             masukan waktu yang benar");
39         }
40     }
41 }
42
43 private void cariEvent() {
44     System.out.println("Cari acara: ");

```

```

45     System.out.println("1. Berdasarkan judul");
46     System.out.println("2. Berdasarkan tanggal");
47     System.out.println("3. Kembali ke menu utama");
48     System.out.println("pilih: ");
49     int pilihan = scanner.nextInt();
50     scanner.nextLine();
51
52     switch (pilihan) {
53         case 1:
54             System.out.print("Cari acara berdasarkan judul: ");
55             String judul = scanner.nextLine();
56             List<Acara> acaras = jadwal.cariEvents(judul);
57             if (acaras.isEmpty()) {
58                 System.out.println("Tidak ada acara dengan judul
59 tersebut.");
60             } else {
61                 for (Acara acara : acaras) {
62                     System.out.println(acara);
63                 }
64             }
65             break;
66         case 2:
67             System.out.print("Cari acara untuk tanggal: ");
68             String tanggal = scanner.nextLine();
69             DateTimeFormatter formatter =
70 DateTimeFormat.forPattern("dd/MM/yyyy");
71
72             try {
73                 DateTime date =
74 formatter.parseDateTime(tanggal).withZone(DateTimeZone.forID("Asia/Jakarta"
75 ));
76                 List<Acara> acarasTanggal = jadwal.acaraUntukTanggal(date);
77                 if (acarasTanggal.isEmpty()) {
78                     System.out.println("Tidak ada acara pada tanggal
79 tersebut.");
80                 } else {
81                     for (Acara acara : acarasTanggal) {
82                         System.out.println(acara);
83                     }
84                 }
85             } catch (IllegalArgumentException e) {
86                 System.out.println("Format tanggal salah");
87             }
88             break;
89         case 3:
90             System.out.println("Kembali ke menu utama");
91             return;
92         default:
93             System.out.println("Pilihan tidak valid");
94             break;
95     }
96 }
97
98
99 }

```

Gambar 3.9 Kode pada kelas ManajemenJadwal

BAB IV

HASIL DAN PEMBAHASAN

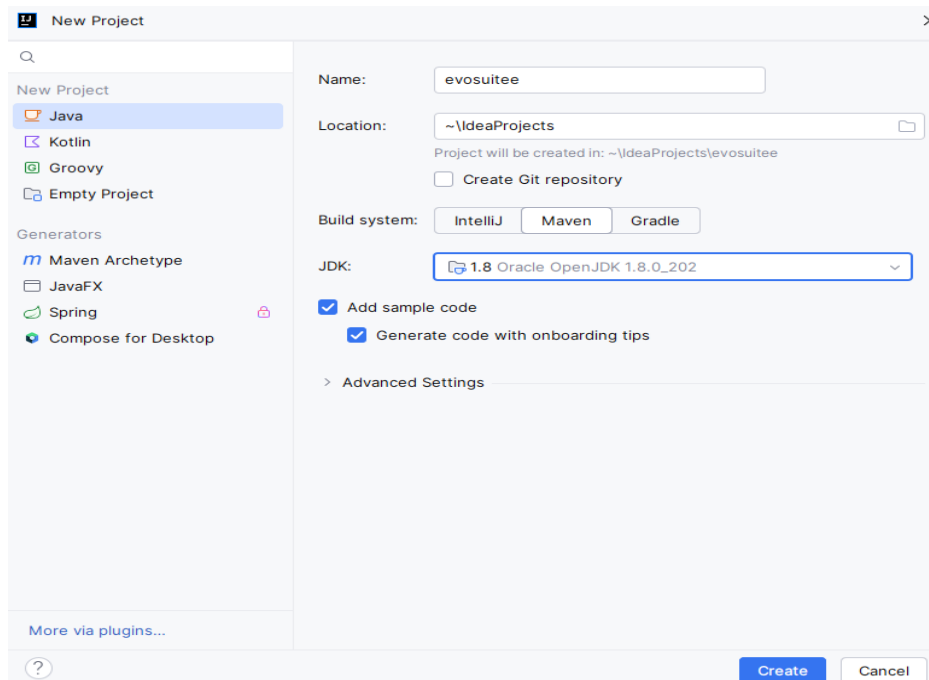
Pada tahap ini akan dilakukan pengujian keempat *framework* otomatis sesuai dengan alur proses yang telah ditentukan dan melakukan perbandingan menggunakan metode DIA. Dalam pengujian keempat *framework* penulis menggunakan IntelliJ IDEA. Pada tahap ini penulis berfokus pada *setup* masing-masing *framework*, menghasilkan *unit test*, menunjukkan hasil kode yang dihasilkan, dan menjalankan hasil tes yang dihasilkan dari masing-masing *framework*.

4.1 Pengujian Unit dengan Evosuite

Sebelum proses pengujian unit menggunakan EvoSuite perlu melakukan *setup* terlebih dahulu. EvoSuite dirancang khusus untuk proyek-proyek yang menggunakan bahasa pemrograman Java. Oleh karena itu, proyek yang digunakan penulis menggunakan Java. Terdapat tiga kelas yang telah dijelaskan sebelumnya, namun, penulis akan menampilkan kode-kode yang penting yang digunakan. kelas-kelas ini digunakan sebagai bahan untuk pengujian.

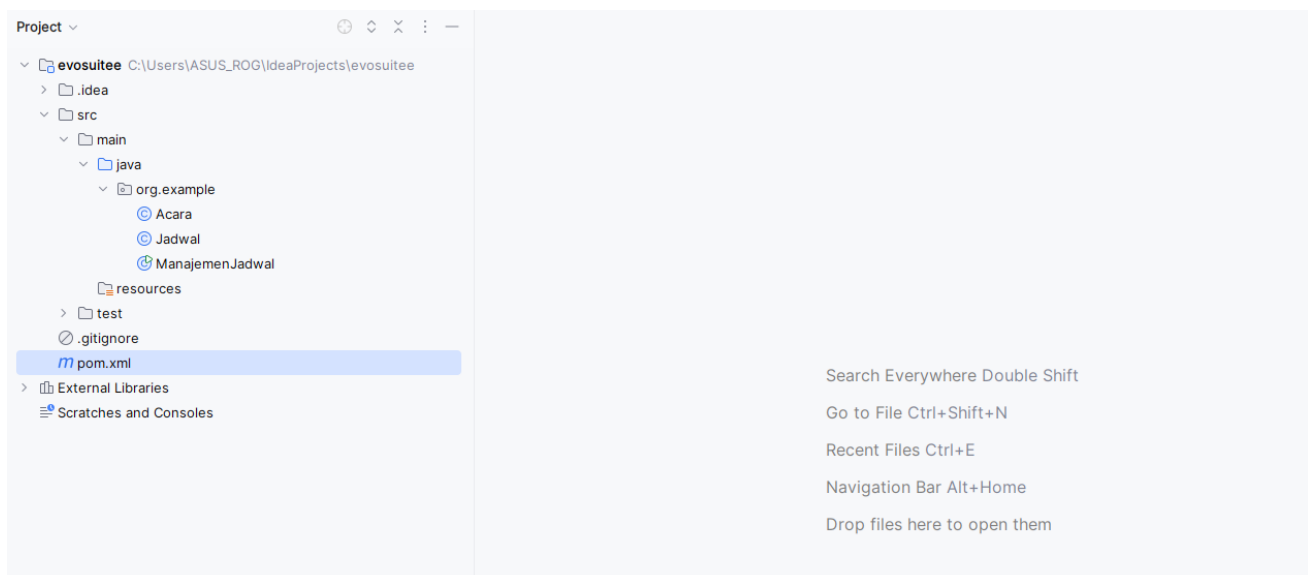
4.1.1 Persiapan dan Instalasi EvoSuite

Hal pertama yang perlu dilakukan adalah melakukan setting proyek di IntelliJ IDEA dengan menggunakan maven sebagai *build* proyek. EvoSuite hanya kompatibel dengan Java 8, untuk memenuhi persyaratan kompatibilitas dengan EvoSuite, penulis mengonfigurasi proyek menggunakan versi Java tersebut.



Gambar 4.1 Tampilan setting proyek IntelliJ IDEA

Setelah melakukan setting proyek di IntelliJ IDEA, halaman utama IntelliJ IDEA akan tampil. Selanjutnya, penulis membuat tiga kelas yang akan digunakan sebagai objek pengujian dengan EvoSuite, yang bertujuan untuk menghasilkan skrip *unit test* secara otomatis.



Gambar 4.2 Konfigurasi file pom.xml

Kemudian, dalam konfigurasi file pom.xml, penulis menambahkan depedensi EvoSuite, Joda Time, dan JUnit untuk mendukung proses pengujian unit dan penambahan plugin EvoSuite agar *framework* dapat berjalan sesuai dengan fungsinya.

```

<dependencies>
  <dependency>
    <groupId>joda-time</groupId>
    <artifactId>joda-time</artifactId>
    <version>2.12.5</version>
  </dependency>
  <dependency>
    <groupId>org.evosuite</groupId>
    <artifactId>evosuite-standalone-runtime</artifactId>
    <version>1.0.6</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
  </dependency>
</dependencies>
<build>
  <plugins>
    <plugin>
      <groupId>org.evosuite.plugins</groupId>
      <artifactId>evosuite-maven-plugin</artifactId>
      <version>1.0.6</version>
      <executions>
        <execution>
          <goals>
            <goal>generate</goal>
            <goal>export</goal>
          </goals>
        </execution>
      </executions>
      <configuration>
        <extraArgs>-Duse_separate_classloader=false</extraArgs>
      </configuration>
    </plugin>
  </plugins>
</build>

```

Gambar 4.3 Depedensi dan Plugin EvoSuite

4.1.2 Proses Menghasilkan Unit Test Otomatis

a. Menghasilkan Unit Test Otomatis

Langkah berikutnya adalah menghasilkan *unit test* secara otomatis menggunakan EvoSuite, dengan perintah maven berikut:

```
mvn compile evosuite:generate
```

Gambar 4.4 Perintah maven untuk menghasilkan *unit test*

Perintah ini akan mengkompilasi kode sumber proyek dan kemudian EvoSuite akan mengesekusi setiap kelas di proyek secara otomatis untuk menghasilkan *unit test*. Pada Gambar 4.5 EvoSuite mulai mengeksekusi tiga kelas yang telah dibuat.

```
--- evosuite:1.0.6:generate (default-cli) @ evosuite ---
Going to generate tests with EvoSuite
Total memory: 800mb
Time per class: 2 minutes
Number of used cores: 1
Target: C:\Users\ASUS_R06\IdeaProjects\evosuite\target\classes
Basedir: C:\Users\ASUS_R06\IdeaProjects\evosuite
Started spawn process manager on port 49753
SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/C:/Users/ASUS_R06/.m2/repository/org/evosuite/evosuite-master/1.0.6/evosuite-master-1.0.6.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: Found binding in [jar:file:/C:/Users/ASUS_R06/.m2/repository/ch/qos/logback/logback-classic/1.1.3/logback-classic-1.1.3.jar!/org/slf4j/impl/StaticLoggerBinder.class]
SLF4J: See http://www.slf4j.org/codes.html#multiple\_bindings for an explanation.
SLF4J: Actual binding is of type [ch.qos.logback.classic.util.ContextSelectorStaticBinder]
* EvoSuite 1.0.6
Registered remote process from /127.0.0.1:49755
Going to execute 3 jobs
```

Gambar 4.5 laporan Eksekusi EvoSuite

Setelah menjalankan perintah tersebut, EvoSuite akan menghasilkan *unit test* di direktori *.evosuite/best-test/org/example*. Namun, *Unit test* tersebut belum masuk ke direktori *src/test/java*, yang merupakan lokasi standar untuk menjalankan unit test.

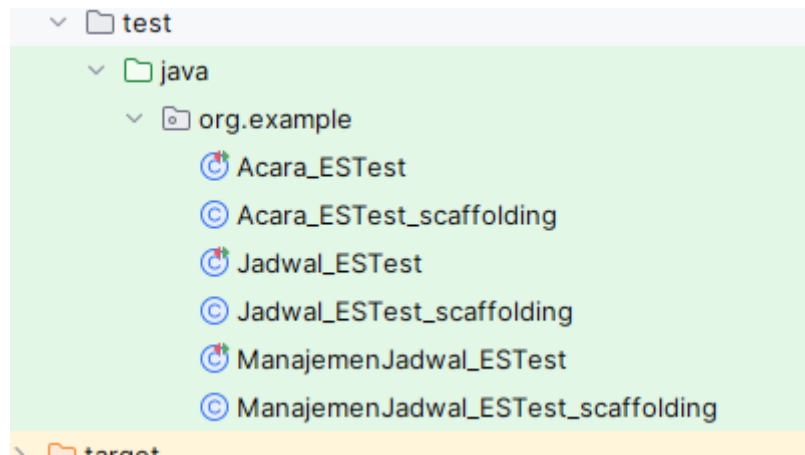
b. Export Hasil Unit Test

File *unit test* yang dihasilkan oleh EvoSuite awalnya ditempatkan di direktori *evosuite/best-test/org/example*. Untuk memindahkan file tersebut ke direktori standar *src/test/java* agar dapat dijalankan sebagai bagian dari proses pengujian, dapat menggunakan perintah maven berikut:

```
mvn evosuite:export
```

Gambar 4.6 Perintah export untuk memindahkan direktori

Setelah menjalankan perintah tersebut hasil unit test dapat dilihat pada Gambar 4.7 bahwa file unit test sudah berada di dalam direktori *src/test/java*.



Gambar 4.7 Hasil Test yang dihasilkan EvoSuite setelah di export

Berikut ini adalah hasil unit test yang dihasilkan secara otomatis oleh EvoSuite untuk masing-masing kelas yaitu kelas Acara, Jadwal, dan ManajemenJadwal.

1. Hasil Test dari kelas Acara_ESTest

```

/*
 * This file was automatically generated by EvoSuite
 * Sun Oct 20 09:51:19 GMT 2024
 */

package org.example;

import org.junit.Test;
import static org.junit.Assert.*;
import static org.evosuite.runtime.EvoAssertions.*;
import org.evosuite.runtime.EvoRunner;
import org.evosuite.runtime.EvoRunnerParameters;
import org.evosuite.runtime.System;
import org.example.Acara;
import org.joda.time.Chronology;
import org.joda.time.DateTime;
import org.joda.time.DateTimeZone;
import org.joda.time.LocalDate;
import org.joda.time.LocalDateTime;
import org.joda.time.MutableDateTime;
import org.joda.time.chrono.CopticChronology;
import org.joda.time.chrono.GJChronology;
import org.joda.time.chrono.GregorianChronology;
import org.joda.time.chrono.JulianChronology;
import org.junit.runner.RunWith;

@RunWith(EvoRunner.class) @EvoRunnerParameters(mockJVMNonDeterminism = true,
useVFS = true, useVNET = true, resetStaticState = true, separateClassLoader =
true, useJEE = true)
public class Acara_ESTest extends Acara_ESTest_scaffolding {

    @Test(timeout = 4000)
    public void test00() throws Throwable {
        CopticChronology copticChronology0 = CopticChronology.getInstanceUTC();
        DateTime dateTime0 = new DateTime((Object) null, copticChronology0);
        Acara acara0 = new Acara("No valid ISO8601 format for fields: ", "",
dateTime0, (DateTime) null);
        DateTime dateTime1 = acara0.getWaktuSelesai();
        assertEquals("No valid ISO8601 format for fields: ", acara0.getJudul());
        assertNull(dateTime1);
        assertEquals("", acara0.getDeskripsi());
    }
}

```

```

@Test(timeout = 4000)
public void test01() throws Throwable {
    DateTime dateTime0 = new DateTime(0L);
    Acara acara0 = new Acara("TQdv+S\"L", "", dateTime0, dateTime0);
    acara0.getWaktuSelesai();
    assertEquals("TQdv+S\"L", acara0.getJudul());
    assertEquals("", acara0.getDeskripsi());
}

@Test(timeout = 4000)
public void test02() throws Throwable {
    System.setCurrentTimeMillis((-1L));
    DateTime dateTime0 = DateTime.now();
    Acara acara0 = new Acara("v(<a(*S`o6`zGR'_((", (String) null, dateTime0,
dateTime0);
    DateTime dateTime1 = acara0.getWaktuSelesai();
    assertEquals(dateTime0, dateTime1);
}

@Test(timeout = 4000)
public void test03() throws Throwable {
    DateTime dateTime0 = new DateTime((Chronology) null);
    Acara acara0 = new Acara("Acara{Judul=''", "", dateTime0, dateTime0);
    acara0.setWaktuMulai((DateTime) null);
    acara0.getWaktuMulai();
    assertEquals("Acara{Judul=''", acara0.getJudul());
    assertEquals("", acara0.getDeskripsi());
}

@Test(timeout = 4000)
public void test04() throws Throwable {
    DateTime dateTime0 = new DateTime(0L);
    Acara acara0 = new Acara("TQdv+S\"L", "", dateTime0, dateTime0);
    acara0.getWaktuMulai();
    assertEquals("TQdv+S\"L", acara0.getJudul());
    assertEquals("", acara0.getDeskripsi());
}

@Test(timeout = 4000)
public void test05() throws Throwable {
    System.setCurrentTimeMillis((-1L));
    DateTime dateTime0 = DateTime.now();
    Acara acara0 = new Acara("v(<a(*S`o6`zGR'_((", (String) null, dateTime0,
dateTime0);
    DateTime dateTime1 = acara0.getWaktuMulai();
    assertEquals(dateTime1, dateTime0);
}

@Test(timeout = 4000)
public void test06() throws Throwable {
    DateTime dateTime0 = DateTime.now();
    Acara acara0 = new Acara((String) null, "L;Loq9$Qr`D", dateTime0,
dateTime0);
    String string0 = acara0.getJudul();
    assertNull(string0);
}

@Test(timeout = 4000)
public void test07() throws Throwable {
    DateTimeZone dateTimeZone0 = DateTimeZone.getDefault();
    GregorianCalendar gregorianCalendar0 =
GregorianCalendar.getInstance(dateTimeZone0);
    MutableDateTime mutableDateTime0 = new MutableDateTime((-623L),
(Chronology) gregorianCalendar0);
    DateTime dateTime0 = mutableDateTime0.toDateTimeISO();
    Acara acara0 = new Acara("", "", waktuMulai=", dateTime0, dateTime0);

```

```

        String string0 = acara0.getJudul();
        assertEquals(" waktuMulai=", acara0.getDeskripsi());
        assertEquals("", string0);
    }

    @Test(timeout = 4000)
    public void test08() throws Throwable {
        JulianChronology julianChronology0 = JulianChronology.getInstance();
        LocalDate localDate0 = new LocalDate((Chronology) julianChronology0);
        LocalTime localTime0 = LocalTime.fromMillisOfDay(3520L, (Chronology)
julianChronology0);
        DateTimeZone dateTimeZone0 = DateTimeZone.getDefault();
        DateTime dateTime0 = localDate0.toDateTime(localTime0, dateTimeZone0);
        Acara acara0 = new Acara((String) null, (String) null, dateTime0,
dateTime0);
        String string0 = acara0.getDeskripsi();
        assertNull(string0);
    }

    @Test(timeout = 4000)
    public void test09() throws Throwable {
        CopticChronology copticChronology0 = CopticChronology.getInstanceUTC();
        DateTime dateTime0 = new DateTime((Object) null, copticChronology0);
        Acara acara0 = new Acara("h\RmbI", "", dateTime0, dateTime0);
        String string0 = acara0.getDeskripsi();
        assertEquals("", string0);
        assertEquals("h\RmbI", acara0.getJudul());
    }

    @Test(timeout = 4000)
    public void test10() throws Throwable {
        GJChronology gJChronology0 = GJChronology.getInstanceUTC();
        DateTimeZone dateTimeZone0 = gJChronology0.getZone();
        DateTime dateTime0 = new DateTime(dateTimeZone0);
        Acara acara0 = new Acara("{}2", " waktuSelesai=", dateTime0, dateTime0);
        // Undeclared exception!
        try {
            acara0.isOverlapping((Acara) null);
            fail("Expecting exception: NullPointerException");

        } catch (NullPointerException e) {
            //
            // no message in exception (getMessage() returned null)
            //
            verifyException("org.example.Acara", e);
        }
    }

    @Test(timeout = 4000)
    public void test11() throws Throwable {
        Acara acara0 = null;
        try {
            acara0 = new Acara("Q[]z{", "Q[]z{", (DateTime) null, (DateTime) null);
            fail("Expecting exception: NullPointerException");

        } catch (NullPointerException e) {
            //
            // no message in exception (getMessage() returned null)
            //
            verifyException("org.example.Acara", e);
        }
    }

    @Test(timeout = 4000)
    public void test12() throws Throwable {
        DateTime dateTime0 = DateTime.now();
        DateTime dateTime1 = dateTime0.withCenturyOfEra(44);
    }

```

```

        Acara acara0 = new Acara((String) null, (String) null, dateTime1,
dateTime1);
        Acara acara1 = new Acara((String) null, (String) null, dateTime0,
dateTime0);
        boolean boolean0 = acara0.isOverlapping(acara1);
        assertFalse(boolean0);
    }

    @Test(timeout = 4000)
    public void test13() throws Throwable {
        DateTime dateTime0 = new DateTime();
        Acara acara0 = new Acara("dd9tMM/yyyy5HHKm", "dd9tMM/yyyy5HHKm", dateTime0,
dateTime0);
        boolean boolean0 = acara0.isOverlapping(acara0);
        assertTrue(boolean0);
    }

    @Test(timeout = 4000)
    public void test14() throws Throwable {
        DateTime dateTime0 = DateTime.now();
        DateTime dateTime1 = dateTime0.withCenturyOfEra(44);
        Acara acara0 = new Acara((String) null, (String) null, dateTime1,
dateTime1);
        acara0.setWaktuSelesai(dateTime0);
        boolean boolean0 = acara0.isOverlapping(acara0);
        assertFalse(boolean0);
    }

    @Test(timeout = 4000)
    public void test15() throws Throwable {
        DateTime dateTime0 = new DateTime();
        DateTime dateTime1 = dateTime0.withMillisOfSecond(16);
        Acara acara0 = null;
        try {
            acara0 = new Acara("Wc", "Wc", dateTime0, dateTime1);
            fail("Expecting exception: IllegalArgumentException");

        } catch (IllegalArgumentException e) {
            //
            // Waktu mulai harus diisi sebelum waktu selesai
            //
            verifyException("org.example.Acara", e);
        }
    }

    @Test(timeout = 4000)
    public void test16() throws Throwable {
        DateTime dateTime0 = new DateTime();
        Acara acara0 = new Acara("dd9tMM/yyyy5HHKm", "dd9tMM/yyyy5HHKm", dateTime0,
dateTime0);
        String string0 = acara0.getJudul();
        assertEquals("dd9tMM/yyyy5HHKm", string0);
    }

    @Test(timeout = 4000)
    public void test17() throws Throwable {
        DateTime dateTime0 = DateTime.now();
        Acara acara0 = new Acara((String) null, (String) null, dateTime0,
dateTime0);
        DateTime dateTime1 = acara0.getWaktuMulai();
        assertEquals(dateTime0, dateTime1);
    }

    @Test(timeout = 4000)
    public void test18() throws Throwable {
        DateTime dateTime0 = new DateTime();
        Acara acara0 = new Acara("dd9tMM/yyyy5HHKm", "dd9tMM/yyyy5HHKm", dateTime0,

```

```

dateTime0);
acara0.setJudul("dd9tMM/yyyy5HHKm");
assertEquals("dd9tMM/yyyy5HHKm", acara0.getDeskripsi());
}

@Test(timeout = 4000)
public void test19() throws Throwable {
    DateTime dateTime0 = DateTime.now();
    Acara acara0 = new Acara((String) null, (String) null, dateTime0,
dateTime0);
    String string0 = acara0.toString();
    assertEquals("Acara{Judul='null', Deskripsi='null', waktuMulai=15/02/2014
03:21, waktuSelesai=15/02/2014 03:21}", string0);
}

@Test(timeout = 4000)
public void test20() throws Throwable {
    DateTime dateTime0 = new DateTime();
    Acara acara0 = new Acara("dd9tMM/yyyy5HHKm", "dd9tMM/yyyy5HHKm", dateTime0,
dateTime0);
    DateTime dateTime1 = acara0.getWaktuSelesai();
    assertEquals(dateTime1, dateTime0);
}

@Test(timeout = 4000)
public void test21() throws Throwable {
    DateTime dateTime0 = new DateTime();
    Acara acara0 = new Acara("dd9tMM/yyyy5HHKm", "dd9tMM/yyyy5HHKm", dateTime0,
dateTime0);
    acara0.setDeskripsi("dd9tMM/yyyy5HHKm");
    assertEquals("dd9tMM/yyyy5HHKm", acara0.getDeskripsi());
}

@Test(timeout = 4000)
public void test22() throws Throwable {
    DateTime dateTime0 = new DateTime();
    Acara acara0 = new Acara("dd9tMM/yyyy5HHKm", "dd9tMM/yyyy5HHKm", dateTime0,
dateTime0);
    String string0 = acara0.getDeskripsi();
    assertEquals("dd9tMM/yyyy5HHKm", string0);
}
}

```

Gambar 4.8 Kelas Acara_ESTest EvoSuite

2. Hasil Test dari kelas Jadwal_ESTest

```

//method tambahAcara
@Test(timeout = 4000)
public void test3() throws Throwable {
    Jadwal jadwal0 = new Jadwal();
    // Undeclared exception!
    try {
        jadwal0.tambahAcara((Acara) null);
        fail("Expecting exception: IllegalArgumentException");

    } catch (IllegalArgumentException e) {
        //
        // Acara tidak boleh null
        //
        verifyException("org.example.Jadwal", e);
    }
}

//method cariEvents
@Test(timeout = 4000)
public void test5() throws Throwable {
    Jadwal jadwal0 = new Jadwal();

```

```

        List<Acara> list0 = jadwal0.cariEvents("UvU09K'AV5yx^L%VhA");
        assertTrue(list0.isEmpty());
    }

    @Test(timeout = 4000)
    public void test1() throws Throwable {
        Jadwal jadwal0 = new Jadwal();
        // Undeclared exception!
        try {
            jadwal0.hapusAcaraBerdasarkanJudul((String) null);
            fail("Expecting exception: IllegalArgumentException");

        } catch (IllegalArgumentException e) {
            //
            // Judul tidak boleh null
            //
            verifyException("org.example.Jadwal", e);
        }
    }

    @Test(timeout = 4000)
    public void test2() throws Throwable {
        Jadwal jadwal0 = new Jadwal();
        boolean boolean0 = jadwal0.hapusAcaraBerdasarkanJudul("UvU09K'AV5yx^L%VhA");
        assertFalse(boolean0);
    }

    @Test(timeout = 4000)
    public void test0() throws Throwable {
        Jadwal jadwal0 = new Jadwal();
        jadwal0.lihatJadwal();
    }

```

Gambar 4.9 Kelas Jadwal_ESTest EvoSuite

3. Hasil Test dari kelas ManajemenJadwal_ESTest

```

    @Test(timeout = 4000)
    public void test03() throws Throwable {
        String[] stringArray0 = new String[7];
        stringArray0[0] = "yuUR3m0?Sm1'";
        stringArray0[1] = "m/g%.ogJIE iCl";
        DatagramChannel datagramChannel0 = DatagramChannel.open();
        Scanner scanner0 = new Scanner(datagramChannel0);
        Jadwal jadwal0 = new Jadwal();
        ManajemenJadwal manajemenJadwal0 = new ManajemenJadwal(scanner0,
        jadwal0);
        // Undeclared exception!
        try {
            manajemenJadwal0.tambahEvent();
            fail("Expecting exception: NotYetConnectedException");

        } catch (NotYetConnectedException e) {
        }
    }

```

Gambar 4.10 Kelas ManajemenJadwal_ESTest Evosuite

4.1.3 Hasil Pengujian Unit dan Analisis

Langkah selanjutnya adalah menjalankan unit test yang telah dihasilkan oleh EvoSuite dengan menggunakan perintah Maven berikut:

```
mvn test
```

Gambar 4.11 Perintah untuk menjalankan tes

Gambar 4.11 menunjukkan perintah untuk menjalankan unit test. Berdasarkan hasil yang ditampilkan pada Gambar 4.12, sebanyak 48 unit tes telah dijalankan tanpa ada kegagalan, kesalahan, atau tes yang dilewati. Namun, Seperti yang dijelaskan sebelumnya penulis memasukan bug pada metode tambahAcara dan cariEvents pada kelas Jadwal, lalu penambahan bug pada kelas ManajemenJadwal. Meskipun bug tersebut telah disisipkan dalam kode, EvoSuite tidak menghasilkan pengujian pada bug yang telah penulis buat. Oleh karena itu, kode yang diuji berhasil melewati seluruh tes yang dihasilkan oleh EvoSuite.

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 48, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 4.650 s
[INFO] Finished at: 2024-10-20T17:22:10+07:00
[INFO] -----
```

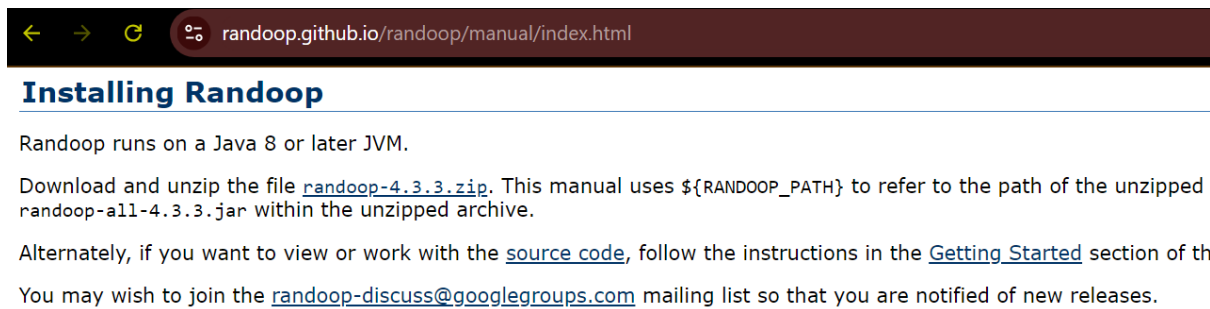
Gambar 4.12 Hasil tes setelah dijalankan

4.2 Pengujian Unit dengan Randoop

Sebelum proses pengujian unit menggunakan *framework* Randoop perlu melakukan *setup* terlebih dahulu. Seperti EvoSuite, Randoop berfokus untuk proyek yang menggunakan bahasa pemrograman Java.

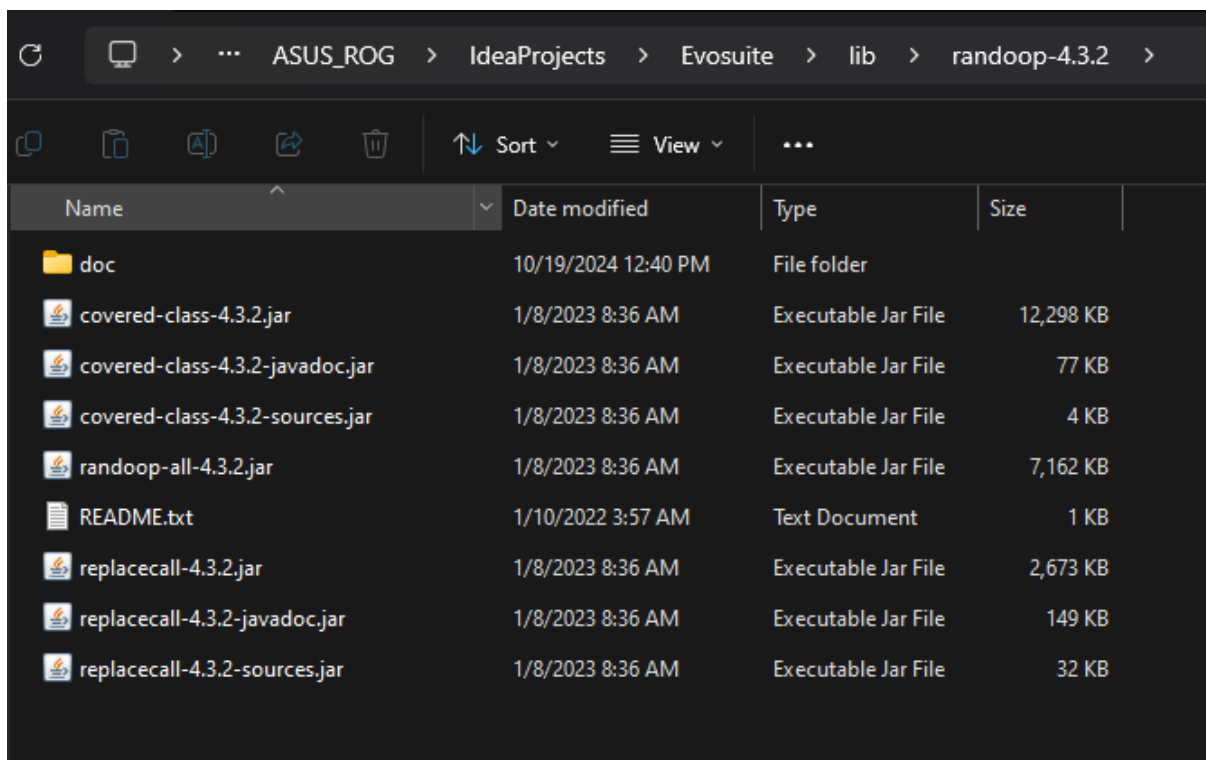
4.2.1 Persiapan dan Instalasi Randoop

Langkah pertama dalam menggunakan Randoop adalah mengunduh dan menginstal dari situs resminya. File yang diunduh berupa file *zip* yang berisi *jar* Randoop.



Gambar 4.13 Situs resmi Randoop

Setelah mengunduh berkas *zip*, penulis melakukan ekstrak isi berkas tersebut. Penulis kemudian membuat folder *lib* di dalam direktori proyek untuk menyimpan file *jar* Randoop. File tersebut nantinya dapat dijalankan saat menggunakan perintah Randoop.



Gambar 4.14 Lokasi file jar Randoop

Langkah selanjutnya adalah melakukan setup pada file pom.xml, tambahkan depedensi JUnit seperti pada Gambar 4.15. Depedensi ini diperlukan untuk menghasilkan *unit test* menggunakan framework JUnit.

```
<dependency>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId>
  <version>4.12</version>
  <scope>test</scope>
</dependency>
```

Gambar 4.15 Depedensi JUnit

4.2.2 Proses Menghasilkan Unit Test Otomatis

a. Menjalankan Randoop dan Menghasilkan Unit Test Otomatis

Untuk menjalankan Randoop dan menghasilkan unit test otomatis, gunakan perintah Maven seperti pada Gambar 4.16. Sebagai contoh perintah ini akan mengesekusi kelas Acara.

```
java -classpath
"C:\Users\ASUS_ROG\IdeaProjects\Evosuite\target\classes;C:\Users\ASUS_ROG.m2\repo
sitory\joda-time\joda-time\2.12.5\joda-time-
2.12.5.jar;C:\Users\ASUS_ROG\IdeaProjects\Evosuite\lib\randoop-4.3.2\randoop-all-
4.3.2.jar" randoop.main.Main gentests --testclass=org.example.Acara --regression-
test-basename=AcaraRegressionTest --output-limit=60 --junit-output-
dir=src/test/java/org/example
```

Gambar 4.16 Perintah menjalankan Randoop menghasilkan test

classpath berisi direktori proyek, library yang dibutuhkan, dan file *jar* Randoop agar dapat menjalankan tes dengan benar. *--testclass* digunakan untuk menentukan kelas yang ingin diuji dalam hal ini adalah kelas Acara, *--regression-test-basename* digunakan untuk memberi nama dasar pada kelas regrei yang dihasilkan. *--Output-limit* digunakan untuk membatasi jumlah tes yang dihasilkan Randoop hingga 60 tes. Perintah *--junit-output-dir* menentukan lokasi penyimpanan hasil pengujian yang dihasilkan, yaitu di direktori *src/test/java/orang/example*. Perintah tersebut digunakan dengan cara yang sama untuk kedua kelas utama.

Setelah menjalankan perintah tersebut Randoop akan menghasilkan dua kelas regression test. Kedua kelas ini berfungsi sebagai regression tests, yaitu tes yang lulus saat dijalankan. Berikut merupakan unit test yang dihasilkan Randoop.

1. Hasil test dari kelas AcaraRegressionTest0

```
package org.example;

import org.junit.FixMethodOrder;
import org.junit.Test;
import org.junit.runners.MethodSorters;
```

```

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class AcaraRegressionTest0 {

    public static boolean debug = false;

    @Test
    public void test1() throws Throwable {
        if (debug)
            System.out.format("%n%s%n", "AcaraRegressionTest0.test1");
        java.lang.Object obj0 = new java.lang.Object();
        java.lang.Class<?> wildcardClass1 = obj0.getClass();
        org.junit.Assert.assertNotNull(wildcardClass1);
    }

    @Test
    public void test2() throws Throwable {
        if (debug)
            System.out.format("%n%s%n", "AcaraRegressionTest0.test2");
        org.joda.time.DateTime dateTime2 = null;
        org.joda.time.DateTime dateTime3 = null;
        // The following exception was thrown during execution in test generation
        try {
            org.example.Acara acara4 = new org.example.Acara("", "", dateTime2,
dateTime3);
            org.junit.Assert.fail("Expected exception of type
java.lang.IllegalArgumentException; message: waktu mulai tidak boleh null");
        } catch (java.lang.IllegalArgumentException e) {
            // Expected exception.
        }
    }

    @Test
    public void test3() throws Throwable {
        if (debug)
            System.out.format("%n%s%n", "AcaraRegressionTest0.test3");
        org.joda.time.DateTime dateTime2 = null;
        org.joda.time.DateTime dateTime3 = null;
        // The following exception was thrown during execution in test generation
        try {
            org.example.Acara acara4 = new org.example.Acara("hi!", "",
dateTime2, dateTime3);
            org.junit.Assert.fail("Expected exception of type
java.lang.IllegalArgumentException; message: waktu mulai tidak boleh null");
        } catch (java.lang.IllegalArgumentException e) {
            // Expected exception.
        }
    }

    @Test
    public void test4() throws Throwable {
        if (debug)
            System.out.format("%n%s%n", "AcaraRegressionTest0.test4");
        org.joda.time.DateTime dateTime2 = null;
        org.joda.time.DateTime dateTime3 = null;
        // The following exception was thrown during execution in test generation
        try {
            org.example.Acara acara4 = new org.example.Acara("", "hi!",
dateTime2, dateTime3);
            org.junit.Assert.fail("Expected exception of type
java.lang.IllegalArgumentException; message: waktu mulai tidak boleh null");
        } catch (java.lang.IllegalArgumentException e) {
            // Expected exception.
        }
    }

    @Test
    public void test5() throws Throwable {

```

```

        if (debug)
            System.out.format("%n%s%n", "AcaraRegressionTest0.test5");
        org.joda.time.DateTime dateTime2 = null;
        org.joda.time.DateTime dateTime3 = null;
        // The following exception was thrown during execution in test generation
        try {
            org.example.Acara acara4 = new org.example.Acara("hi!", "hi!",
dateTime2, dateTime3);
            org.junit.Assert.fail("Expected exception of type
java.lang.IllegalArgumentException; message: waktu mulai tidak boleh null");
        } catch (java.lang.IllegalArgumentException e) {
            // Expected exception.
        }
    }
}

```

Gambar 4.17 kelas AcaraRegressionTest0 Randoop

2. Hasil test dari Kelas JadwalRegressionTest0

```

@Test
public void test01() throws Throwable {
    if (debug)
        System.out.format("%n%s%n", "JadwalRegressionTest0.test01");
    org.example.Jadwal jadwal0 = new org.example.Jadwal();
    org.example.Acara acaral = null;
    // The following exception was thrown during execution in test generation
    try {
        jadwal0.tambahAcara(acaral);
        org.junit.Assert.fail("Expected exception of type
java.lang.IllegalArgumentException; message: Acara tidak boleh null");
    } catch (java.lang.IllegalArgumentException e) {
        // Expected exception.
    }
}

@Test
public void test04() throws Throwable {
    if (debug)
        System.out.format("%n%s%n", "JadwalRegressionTest0.test04");
    org.example.Jadwal jadwal0 = new org.example.Jadwal();
    boolean boolean2 = jadwal0.hapusAcaraBerdasarkanJudul("");
    org.example.Acara acara3 = null;
    // The following exception was thrown during execution in test generation
    try {
        jadwal0.tambahAcara(acara3);
        org.junit.Assert.fail("Expected exception of type
java.lang.IllegalArgumentException; message: Acara tidak boleh null");
    } catch (java.lang.IllegalArgumentException e) {
        // Expected exception.
    }
    org.junit.Assert.assertTrue("'" + boolean2 + "' != '" + false + "'", boolean2
== false);
}

@Test
public void test15() throws Throwable {
    if (debug)
        System.out.format("%n%s%n", "JadwalRegressionTest0.test15");
    org.example.Jadwal jadwal0 = new org.example.Jadwal();
    java.util.List<org.example.Acara> acaraList2 = jadwal0.cariEvents("hi!");
    java.lang.Class<?> wildcardClass3 = acaraList2.getClass();
    org.junit.Assert.assertNotNull(acaraList2);
    org.junit.Assert.assertNotNull(wildcardClass3);
}

```

```

@Test
public void test20() throws Throwable {
    if (debug)
        System.out.format("%n%s%n", "JadwalRegressionTest0.test20");
    org.example.Jadwal jadwal0 = new org.example.Jadwal();
    boolean boolean2 = jadwal0.hapusAcaraBerdasarkanJudul("");
    jadwal0.lihatJadwal();
    jadwal0.lihatJadwal();
    java.util.List<org.example.Acara> acaraList5 = jadwal0.getEvents();
    java.util.List<org.example.Acara> acaraList6 = jadwal0.getEvents();
    boolean boolean8 = jadwal0.hapusAcaraBerdasarkanJudul("");
    boolean boolean10 = jadwal0.hapusAcaraBerdasarkanJudul("");
    java.util.List<org.example.Acara> acaraList12 = jadwal0.cariEvents("hi!");
    org.junit.Assert.assertTrue("'" + boolean2 + "' != '" + false + "'", boolean2
== false);
    org.junit.Assert.assertNotNull(acaraList5);
    org.junit.Assert.assertNotNull(acaraList6);
    org.junit.Assert.assertTrue("'" + boolean8 + "' != '" + false + "'", boolean8
== false);
    org.junit.Assert.assertTrue("'" + boolean10 + "' != '" + false + "'",
boolean10 == false);
    org.junit.Assert.assertNotNull(acaraList12);
}

```

Gambar 4.18 Kelas JadwalRegressionTest0 Randoop

3. Hasil test dari kelas ManajemenJadwalRegressionTest0

```

@FixMethodOrder(MethodSorters.NAME_ASCENDING)
public class ManajemenJadwalRegressionTest0 {

    public static boolean debug = false;

    @Test
    public void test1() throws Throwable {
        if (debug)
            System.out.format("%n%s%n", "ManajemenJadwalRegressionTest0.test1");
        org.example.ManajemenJadwal manajemenJadwal0 = new
org.example.ManajemenJadwal();
        java.lang.Class<?> wildcardClass1 = manajemenJadwal0.getClass();
        org.junit.Assert.assertNotNull(wildcardClass1);
    }

    @Test
    public void test2() throws Throwable {
        if (debug)
            System.out.format("%n%s%n", "ManajemenJadwalRegressionTest0.test2");
        org.example.ManajemenJadwal manajemenJadwal0 = new
org.example.ManajemenJadwal();
        // flaky: manajemenJadwal0.tambahEvent();
        org.example.Jadwal jadwal2 = manajemenJadwal0.getJadwal();
        org.junit.Assert.assertNotNull(jadwal2);
    }
}

```

b. Mengatur Environment Variable

Langkah selanjutnya adalah mengatur *environment variable* untuk memastikan bahwa *library* yang dibutuhkan oleh proyek dapat ditemukan dan digunakan saat menjalankan

pengujian unit. Penulis menambahkan *library* yang diperlukan yaitu JUnit, Hamcrest, dan Joda Time dengan menggunakan perintah berikut:

```
$env:JUNITPATH =
"target\classes;C:\Users\ASUS_ROG\.m2\repository\junit\junit\4.12\junit-
4.12.jar;C:\Users\ASUS_ROG\.m2\repository\org\hamcrest\hamcrest-
core\1.3\hamcrest-core-1.3.jar;C:\Users\ASUS_ROG\.m2\repository\joda-time\joda-
time\2.12.5\joda-time-2.12.5.jar"
```

Gambar 4.19 perintah untuk mengatur environment variable

Gambar 4.19 menunjukkan perintah untuk mengatur *environment variable*. Setelah *environment* telah ditetapkan, langkah selanjutnya adalah melakukan kompilasi terhadap file Java yang dihasilkan oleh Randoop agar dapat dijalankan dalam pengujian menggunakan JUnit. Berikut adalah perintah yang digunakan untuk mengkompilasi:

```
javac -classpath ".;$env:JUNITPATH" -d
"C:\Users\ASUS_ROG\IdeaProjects\Evosuite\target\test-classes" -sourcepath
"C:\Users\ASUS_ROG\IdeaProjects\Evosuite\src\test\java\org\example"
"C:\Users\ASUS_ROG\IdeaProjects\Evosuite\src\test\java\org\example\AcaraRegression
nTest.java"
"C:\Users\ASUS_ROG\IdeaProjects\Evosuite\src\test\java\org\example\AcaraRegression
nTest0.java"
```

Gambar 4.20 Perintah untuk mengkompilasi

Gambar 4.20 merupakan perintah kompilasi. Perintah ini menggunakan *classpath* yang sudah diatur sebelumnya dengan *environment variable* JUNITPATH untuk menyertakan *library* JUnit, Hamcrest, dan Joda-Time yang diperlukan dalam proses kompilasi.

4.2.3 Hasil pengujian Unit dan Analisis

Untuk menjalankan *unit test* dengan JUnit menggunakan perintah dibawah ini:

```
java -classpath "C:\Users\ASUS_ROG\IdeaProjects\Evosuite\target\test-
classes;$env:JUNITPATH" org.junit.runner.JUnitCore
org.example.AcaraRegressionTest org.example.AcaraRegressionTest0
```

Gambar 4.21 Perintah untuk menjalankan tes yang dihasilkan Randoop

Gambar 4.21 menunjukkan perintah untuk menjalankan tes yang dihasilkan oleh Randoop menggunakan Junit. Berdasarkan hasil yang ditampilkan pada **Error! Reference source not found.**, sebagai contoh disini kelas JadwalRegressionTest sebanyak 66 *unit test* telah dijalankan tanpa ada kegagalan, kesalahan, atau tes yang dilewati. Namun, Seperti yang dijelaskan sebelumnya penulis memasukan bug pada metode tambahAcara dan cariEvents pada kelas Jadwal, lalu penambahan bug pada kelas ManajemenJadwal. Meskipun bug tersebut telah disisipkan dalam kode, Randoop tidak menghasilkan pengujian pada bug yang telah penulis buat.

```

Tidak ada event di jadwal
.Tidak ada event di jadwal
Tidak ada event di jadwal
..Tidak ada event di jadwal
.Tidak ada event di jadwal
Tidak ada event di jadwal
.Tidak ada event di jadwal
Tidak ada event di jadwal
Tidak ada event di jadwal

Time: 0.086

OK (66 tests)

PS C:\Users\ASUS_R06\IdeaProjects\randooptest>

```

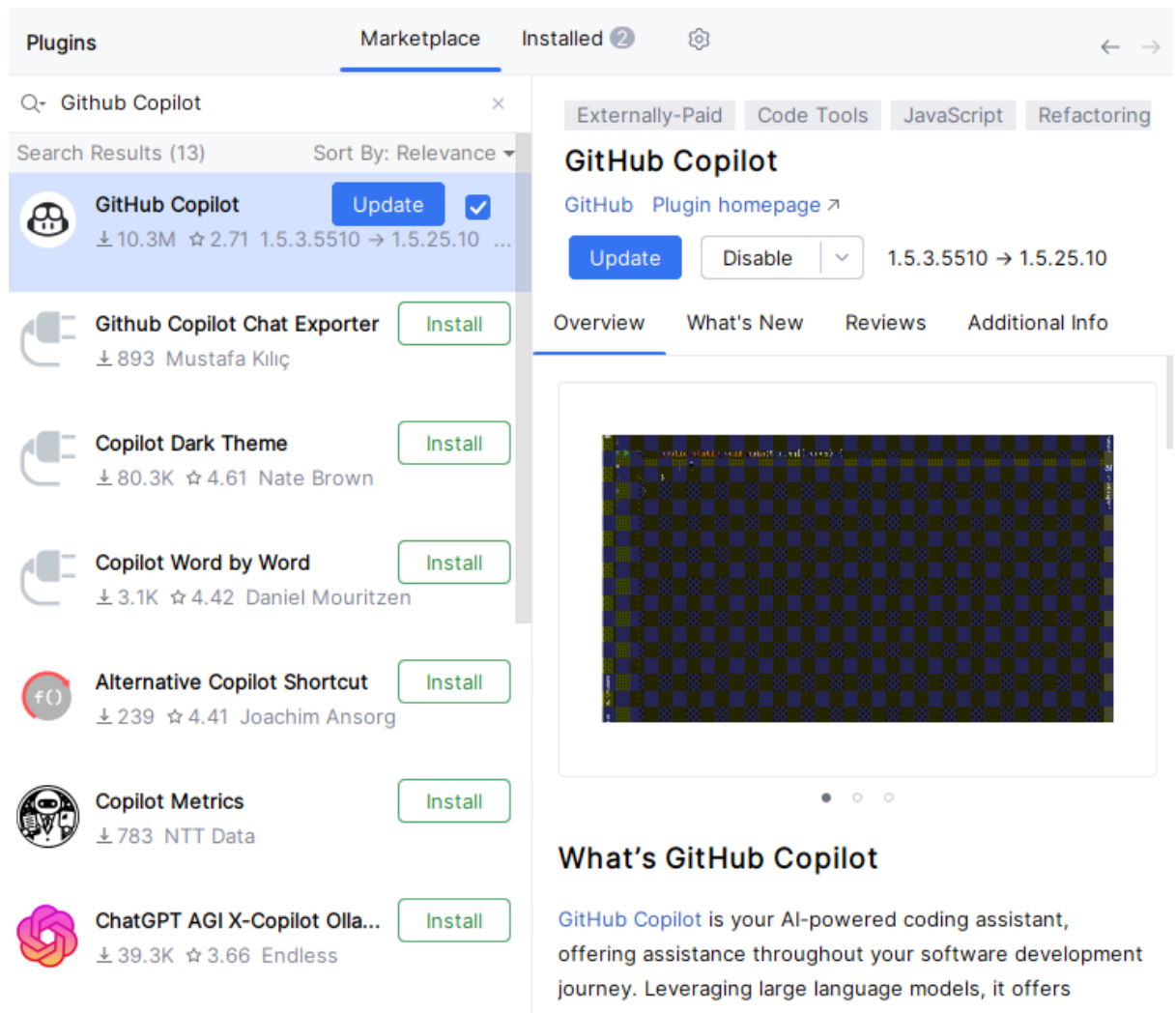
Gambar 4.22 Hasil tes pada kelas JadwalRegressionTest0 setelah dijalankan

4.3 Pengujian Unit dengan Github Copilot

Sebelum proses pengujian unit menggunakan *framework* Github Copilot perlu menginstal plugin di dalam IntelliJ IDEA. Github Copilot bisa menggunakan banyak bahasa pemrograman. Namun, penulis menyesuaikan seperti *framework* sebelumnya yaitu menggunakan bahasa Java untuk memudahkan pengujian.

4.3.1 Persiapan dan Instalasi Github Copilot

Langkah pertama dalam menggunakan Github Copilot pada IntelliJ IDEA adalah menginstal plugin Github Copilot melalui marketplace yang terdapat pada IntelliJ IDEA. Setelah instalasi selesai, plugin dapat langsung digunakan untuk melakukan pengujian.



Gambar 4.23 Install Github Copilot pada IntelliJ IDEA

4.3.2 Proses Menghasilkan Unit Test Otomatis

Pada bagian *sidebar* di sisi kanan halaman IntelliJ IDEA, terdapat ikon Github Copilot yang dapat digunakan untuk membuka panel *chat* seperti yang ditunjukkan pada Gambar 4.23. Untuk menghasilkan *unit test* secara otomatis, penulis mengetik perintah `/tests` di dalam panel *chat* Github Copilot. Github Copilot akan menjalankan perintah tersebut dan menampilkan hasil unit tests secara otomatis seperti yang ditampilkan pada Gambar 4.25.



Gambar 4.24 Icon untuk membuka Github Copilot di IntelliJ IDEA



Gambar 4.25 Panel chat Github Copilot

Berikut merupakan *unit test* yang dihasilkan oleh Github Copilot dari kelas Acara, Jadwal, dan ManajemenJadwal.

1. Test dihasilkan kelas AcaraTest

```
package org.example;

import org.example.Acara;
import org.joda.time.DateTime;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class AcaraTest {

    @Test
    void constructorShouldThrowExceptionIfWaktuMulaiIsAfterWaktuSelesai() {
        DateTime waktuMulai = new DateTime(2023, 10, 10, 10, 0);
        DateTime waktuSelesai = new DateTime(2023, 10, 10, 9, 0);
        assertThrows(IllegalArgumentException.class, () -> new Acara("Judul",
"Deskripsi", waktuMulai, waktuSelesai));
    }

    @Test
    void constructorShouldCreateAcaraIfWaktuMulaiIsBeforeWaktuSelesai() {
        DateTime waktuMulai = new DateTime(2023, 10, 10, 9, 0);
        DateTime waktuSelesai = new DateTime(2023, 10, 10, 10, 0);
        Acara acara = new Acara("Judul", "Deskripsi", waktuMulai, waktuSelesai);
        assertNotNull(acara);
    }

    @Test
    void isOverlappingShouldReturnTrueIfEventsOverlap() {
        DateTime waktuMulai1 = new DateTime(2023, 10, 10, 9, 0);
        DateTime waktuSelesai1 = new DateTime(2023, 10, 10, 10, 0);
        Acara acara1 = new Acara("Judul1", "Deskripsi1", waktuMulai1,
waktuSelesai1);

        DateTime waktuMulai2 = new DateTime(2023, 10, 10, 9, 30);
        DateTime waktuSelesai2 = new DateTime(2023, 10, 10, 10, 30);
        Acara acara2 = new Acara("Judul2", "Deskripsi2", waktuMulai2,
waktuSelesai2);

        assertTrue(acara1.isOverlapping(acara2));
    }

    @Test
    void isOverlappingShouldReturnFalseIfEventsDoNotOverlap() {
        DateTime waktuMulai1 = new DateTime(2023, 10, 10, 9, 0);
        DateTime waktuSelesai1 = new DateTime(2023, 10, 10, 10, 0);
        Acara acara1 = new Acara("Judul1", "Deskripsi1", waktuMulai1,
waktuSelesai1);

        DateTime waktuMulai2 = new DateTime(2023, 10, 10, 10, 30);
        DateTime waktuSelesai2 = new DateTime(2023, 10, 10, 11, 30);
        Acara acara2 = new Acara("Judul2", "Deskripsi2", waktuMulai2,
waktuSelesai2);

        assertFalse(acara1.isOverlapping(acara2));
    }

    @Test
    void toStringShouldReturnFormattedString() {
        DateTime waktuMulai = new DateTime(2023, 10, 10, 9, 0);
        DateTime waktuSelesai = new DateTime(2023, 10, 10, 10, 0);
        Acara acara = new Acara("Judul", "Deskripsi", waktuMulai, waktuSelesai);
    }
}
```

```

        String expected = "Acara{Judul='Judul', Deskripsi='Deskripsi',
waktuMulai=10/10/2023 09:00, waktuSelesai=10/10/2023 10:00}";
        assertEquals(expected, acara.toString());
    }
}

```

Gambar 4.26 Kelas AcaraTest Github Copilot

2. Test yang dihasilkan JadwalTest

```

@Test
void tambahAcaraShouldThrowExceptionIfOverlap() {
    Jadwal jadwal = new Jadwal();
    DateTime waktuMulai1 = new DateTime(2023, 10, 10, 9, 0);
    DateTime waktuSelesai1 = new DateTime(2023, 10, 10, 10, 0);
    Acara acara1 = new Acara("Judul1", "Deskripsi1", waktuMulai1, waktuSelesai1);
    jadwal.tambahAcara(acara1);

    DateTime waktuMulai2 = new DateTime(2023, 10, 10, 9, 30);
    DateTime waktuSelesai2 = new DateTime(2023, 10, 10, 10, 30);
    Acara acara2 = new Acara("Judul2", "Deskripsi2", waktuMulai2, waktuSelesai2);

    assertThrows(IllegalArgumentException.class, () ->
jadwal.tambahAcara(acara2));
}

@Test
void cariEventsShouldReturnEventsMatchingKeyword() {
    Jadwal jadwal = new Jadwal();
    DateTime waktuMulai = new DateTime(2023, 10, 10, 9, 0);
    DateTime waktuSelesai = new DateTime(2023, 10, 10, 10, 0);
    Acara acara = new Acara("Judul", "Deskripsi", waktuMulai, waktuSelesai);
    jadwal.tambahAcara(acara);
    List<Acara> events = jadwal.cariEvents("Judul");
    assertEquals(1, events.size());
}

@Test
void cariEventsShouldReturnEmptyListIfNoEventsMatchKeyword() {
    Jadwal jadwal = new Jadwal();
    DateTime waktuMulai = new DateTime(2023, 10, 10, 9, 0);
    DateTime waktuSelesai = new DateTime(2023, 10, 10, 10, 0);
    Acara acara = new Acara("Judul", "Deskripsi", waktuMulai, waktuSelesai);
    jadwal.tambahAcara(acara);
    List<Acara> events = jadwal.cariEvents("TidakAdaKataKunci");
    assertTrue(events.isEmpty());
}

@Test
void tambahAcaraShouldAddSequentialEventsWithoutException() {
    Jadwal jadwal = new Jadwal();
    DateTime waktuMulai1 = new DateTime(2023, 10, 10, 9, 0);
    DateTime waktuSelesai1 = new DateTime(2023, 10, 10, 10, 0);
    Acara acara1 = new Acara("Judul1", "Deskripsi1", waktuMulai1, waktuSelesai1);
    jadwal.tambahAcara(acara1);

    DateTime waktuMulai2 = new DateTime(2023, 10, 10, 10, 0);
    DateTime waktuSelesai2 = new DateTime(2023, 10, 10, 11, 0);
    Acara acara2 = new Acara("Judul2", "Deskripsi2", waktuMulai2, waktuSelesai2);

    assertDoesNotThrow(() -> jadwal.tambahAcara(acara2));
    assertEquals(2, jadwal.getEvents().size());
}

```

```

@Test
void hapusEventShouldRemoveEvent() {
    Jadwal jadwal = new Jadwal();
    DateTime waktuMulai = new DateTime(2023, 10, 10, 9, 0);
    DateTime waktuSelesai = new DateTime(2023, 10, 10, 10, 0);
    Acara acara = new Acara("Judul", "Deskripsi", waktuMulai, waktuSelesai);
    jadwal.tambahAcara(acara);
    assertTrue(jadwal.hapusEvent(acara));
    assertEquals(0, jadwal.getEvents().size());
}

```

Gambar 4.27 Kelas JadwalTest Github Copilot

3. Test dihasilkan ManajemenJadwalTest

```

@Test
void tambahEventShouldNotAddEventIfOverlap() {
    // Arrange
    DateTime waktuMulai1 = new DateTime(2023, 10, 1, 10, 0);
    DateTime waktuSelesai1 = new DateTime(2023, 10, 1, 12, 0);
    Acara acara1 = new Acara("Meeting", "Project discussion",
    waktuMulai1, waktuSelesai1);
    manajemenJadwal.getJadwal().tambahAcara(acara1);

    DateTime waktuMulai2 = new DateTime(2023, 10, 1, 11, 0);
    DateTime waktuSelesai2 = new DateTime(2023, 10, 1, 13, 0);
    Acara acara2 = new Acara("Workshop", "Skill development",
    waktuMulai2, waktuSelesai2);

    // Act & Assert
    assertThrows(IllegalArgumentException.class, () ->
    manajemenJadwal.getJadwal().tambahAcara(acara2));
    assertEquals(1, manajemenJadwal.getJadwal().getEvents().size());
    assertEquals(acara1, manajemenJadwal.getJadwal().getEvents().get(0));
}

@Test
void tambahEventShouldAddEventIfNoOverlap() {
    // Arrange
    DateTime waktuMulai = new DateTime(2023, 10, 1, 10, 0);
    DateTime waktuSelesai = new DateTime(2023, 10, 1, 12, 0);
    Acara acara = new Acara("Meeting", "Project discussion", waktuMulai,
    waktuSelesai);

    // Act
    manajemenJadwal.getJadwal().tambahAcara(acara);

    // Assert
    assertEquals(1, manajemenJadwal.getJadwal().getEvents().size());
    assertEquals(acara, manajemenJadwal.getJadwal().getEvents().get(0));
}

```

Gambar 4.28 Kelas JadwalTest Github Copilot

4.3.3 Hasil Pengujian Unit dan Analisis

Penulis menjalankan pengujian dengan melakukan klik pada tombol *run* pada setiap kelas *unit test*. Hasil pengujian dapat langsung dilihat melalui tampilan IntelliJ IDEA.

a. Hasil test dari AcaraTest dan Analisis

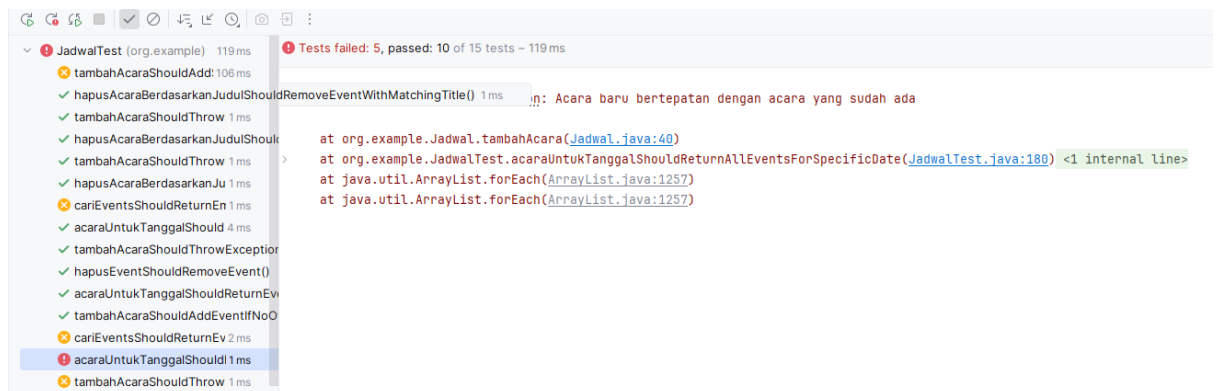
Sebanyak 5 *unit test* telah dijalankan tanpa ada kegagalan, kesalahan, atau tes yang dilewati. Hasil pengujian menunjukkan bahwa semua metode dalam kelas AcaraTest berhasil melewati pengujian dengan sukses seperti yang ditampilkan pada Gambar 4.29.



Gambar 4.29 Hasil run kelas Acaratest Github Copilot

b. Hasil test dari JadwalTest dan Analisis

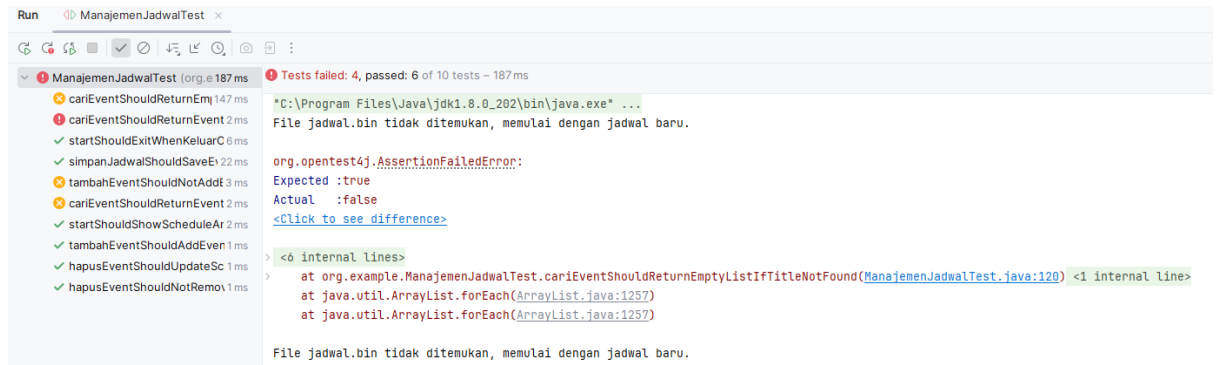
Pada kelas JadwalTest, sebagian pengujian tidak dapat dieksekusi karena terdapat bug yang sengaja disisipkan oleh penulis pada metode cariEvents dan tambahAcara.



Gambar 4.30 Hasil run kelas JadwalTest Github Copilot

c. Hasil test dari ManajemenJadwalTest dan Analisis

Pada kelas ManajemenJadwalTest terdapat sebagian pengujian tidak dapat dieksekusi karena terdapat bug yang sengaja disisipkan oleh penulis pada metode tambahAcara. Namun, bug yang dimasukan dalam metode tambahAcara di kelas ManajemenJadwal adalah bug format tanggal. Hasil tes yang dibuat oleh Github Copilot tidak membuat tes terkait format tanggal.



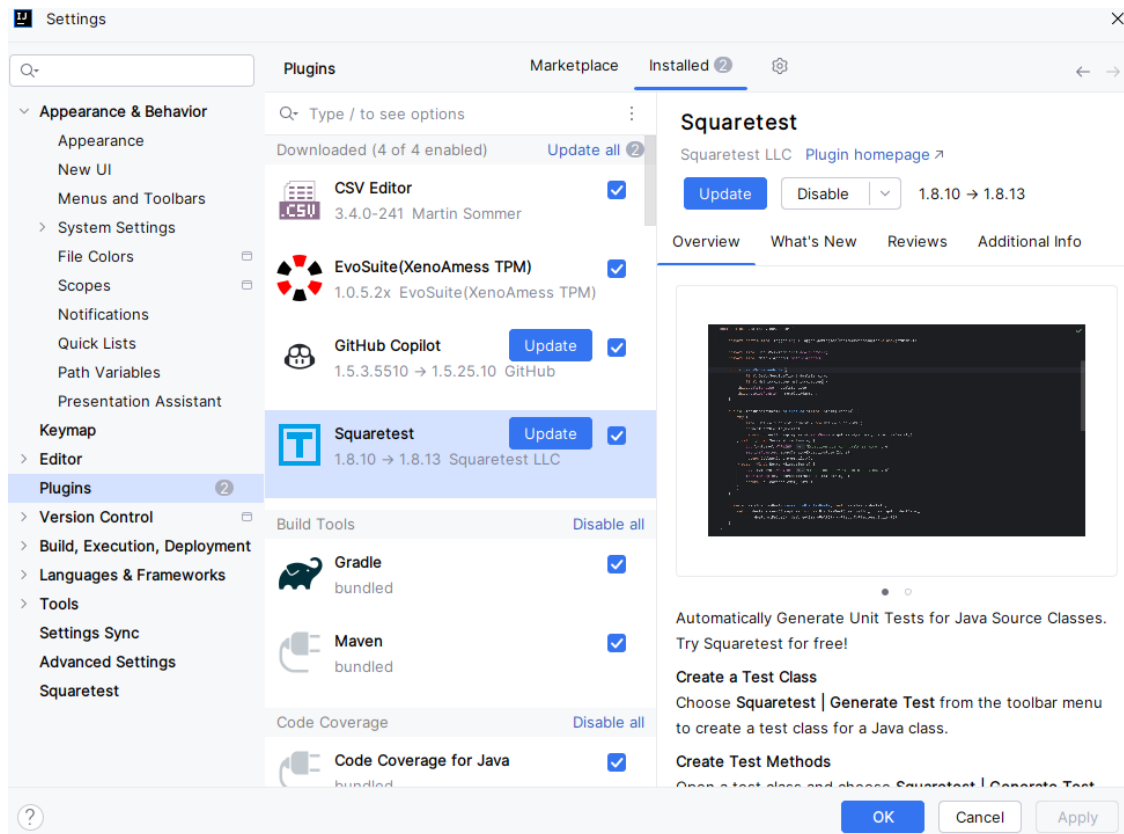
Gambar 4.31 Hasil ManajemenJadwalTest Github Copilot

4.4 Pengujian Unit dengan Squaretest

Sebelum proses pengujian unit menggunakan *framework* Squaretest perlu menginstal plugin di dalam IntelliJ IDEA. Seperti EvoSuite dan Randoop, Squaretest berfokus pada proyek yang menggunakan bahasa pemrograman Java.

4.4.1 Persiapan dan Instalasi Squaretest

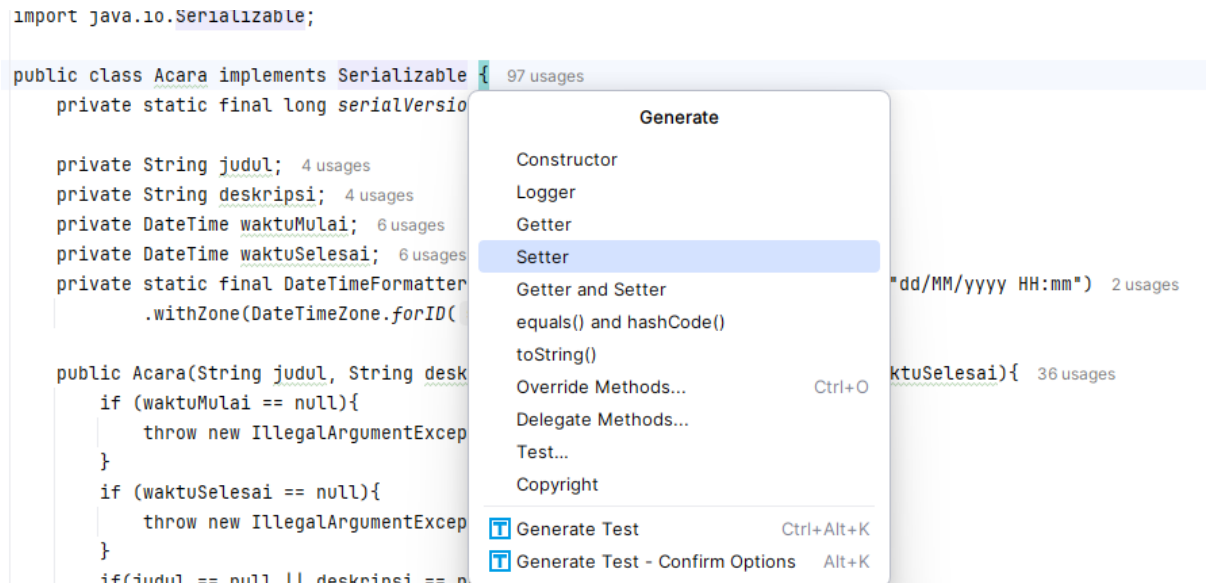
Untuk menjalankan *framework* Squaretest, langkah pertama adalah menginstal plugin Squaretest melalui marketplace yang terdapat pada IntelliJ IDEA. Pada Gambar 4.32 ditampilkan bahwa plugin Squaretest telah berhasil ditambahkan ke daftar plugin yang terinstal di IntelliJ IDEA.



Gambar 4.32 Instalasi plugin Squaretest pada IntelliJ IDEA

4.4.2 Proses Menghasilkan Unit Test Otomatis

Untuk menghasilkan *unit test* secara otomatis menggunakan Squaretest penulis cukup menggunakan tombol pintasan 'Alt + Ins' di IntelliJ IDEA, kemudian pilih opsi 'Generate Test', seperti yang ditampilkan pada Gambar 4.33, Squaretest akan memproses kelas yang dipilih dan langsung menghasilkan kode *unit test* berdasarkan metode yang ada di dalam kelas tersebut.



Gambar 4.33 Proses menghasilkan *unit test* dengan Squaretest

Berikut merupakan unit test yang dihasilkan oleh Squaretest dari kelas Acara, Jadwal, dan ManajemenJadwal

a. Hasil test dari kelas AcaraSquaretest

```
package org.example;

import org.joda.time.DateTime;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.junit.jupiter.api.Assertions.assertFalse;

class AcaraTest {

    private Acara acaraUnderTest;

    @BeforeEach
    void setUp() {
        acaraUnderTest = new Acara("judul", "deskripsi", new DateTime(2020, 1, 1,
1, 0, 0, 0),
        new DateTime(2020, 1, 1, 1, 0, 0, 0));
    }

    @Test
    void testJudulGetterAndSetter() {
        final String judul = "judul";
        acaraUnderTest.setJudul(judul);
        assertEquals(judul, acaraUnderTest.getJudul());
    }

    @Test
    void testDeskripsiGetterAndSetter() {
        final String deskripsi = "deskripsi";
        acaraUnderTest.setDeskripsi(deskripsi);
        assertEquals(deskripsi, acaraUnderTest.getDeskripsi());
    }

    @Test
    void testWaktuMulaiGetterAndSetter() {
        final DateTime waktuMulai = new DateTime(2020, 1, 1, 1, 0, 0, 0);
```

```

        acaraUnderTest.setWaktuMulai(waktuMulai);
        assertEquals(waktuMulai, acaraUnderTest.getWaktuMulai());
    }

    @Test
    void testWaktuSelesaiGetterAndSetter() {
        final DateTime waktuSelesai = new DateTime(2020, 1, 1, 1, 0, 0, 0);
        acaraUnderTest.setWaktuSelesai(waktuSelesai);
        assertEquals(waktuSelesai, acaraUnderTest.getWaktuSelesai());
    }

    @Test
    void testIsOverlapping() {
        // Setup
        final Acara other = new Acara("judul", "deskripsi", new DateTime(2020, 1,
1, 10, 0), new DateTime(2020, 1, 1, 12, 0));

        final Acara acaraUnderTest = new Acara("judul2", "deskripsi2", new
DateTime(2020, 1, 2, 10, 0), new DateTime(2020, 1, 2, 12, 0));

        // Run the test
        final boolean result = acaraUnderTest.isOverlapping(other);

        // Verify the results
        assertFalse(result); // Ini harus false karena event tidak bertabrakan
    }

    @Test
    void testToString() {
        Acara acaraUnderTest = new Acara("judul", "deskripsi", new DateTime(2020,
1, 1, 1, 0), new DateTime(2020, 1, 1, 1, 0));
        String expected = "Acara{Judul='judul', Deskripsi='deskripsi',
waktuMulai=01/01/2020 01:00, waktuSelesai=01/01/2020 01:00}";
        assertEquals(expected, acaraUnderTest.toString());
    }
}

```

Gambar 4.34 Kelas AcaraTest Squaretest

b. Hasil tes dari kelas JadwalSquaretest

```

@Test
void testTambahAcara() {
    // Setup
    final Acara acara = new Acara("judul", "deskripsi", new DateTime(2020, 1, 1,
1, 0, 0, 0),
        new DateTime(2020, 1, 1, 1, 0, 0, 0));

    // Run the test
    jadwalUnderTest.tambahAcara(acara);

    // Verify the results
}

@Test
void testCariEvents() {
    // Setup
    // Run the test
    final List<Acara> result = jadwalUnderTest.cariEvents("kataKunci");

    // Verify the results
}

```

```

@Test
void testHapusAcaraBerdasarkanJudul() {
    // Setup
    // Run the test
    final boolean result = jadwalUnderTest.hapusAcaraBerdasarkanJudul("judul");

    // Verify the results
    assertFalse(result);
}

@Test
void testLihatJadwal() {
    // Setup
    // Run the test
    jadwalUnderTest.lihatJadwal();

    // Verify the results
}

```

Gambar 4.35 Kelas JadwalTest Squaretest

c. Hasil tes dari kelas ManajemenJadwalSquaretest

```

@Test
void testTambahEvent() {
    // Setup
    String simulatedInput = "Judul Acara\nDeskripsi Acara\n01/10/2023\n10:00\n01/10/2023 12:00\n";
    InputStream inputStream = new
    ByteArrayInputStream(simulatedInput.getBytes());
    Scanner scanner = new Scanner(inputStream);
    ManajemenJadwal manajemenJadwal = new ManajemenJadwal(scanner, new
    Jadwal());

    // Run the test
    assertDoesNotThrow(() -> manajemenJadwal.tambahEvent());
}

```

Gambar 4.36 Kelas ManajemenJadwalTest Squaretest

4.4.3 Hasil Pengujian Unit dan Analisis

Penulis menjalankan test dengan mengklik tombol run pada setiap kelas unit test dan dari situ juga dapat dilihat hasil test.

a. Hasil test dari AcaraTest Squaretest dan Analisis

Sebanyak 6 *unit test* telah dijalankan tanpa ada kegagalan, kesalahan, atau tes yang dilewati. Hasil pengujian menunjukkan bahwa semua metode dalam kelas AcaraTest berhasil melewati pengujian dengan sukses seperti yang ditampilkan pada Gambar 4.37.



Gambar 4.37 Hasil run test pada kelas AcaraTest Squaretest

b. Hasil test dari JadwalTest Squaretest dan Analisis

Sebanyak 7 *unit test* telah dijalankan tanpa ada kegagalan, kesalahan, atau tes yang dilewati. Namun, Seperti yang dijelaskan sebelumnya penulis sengaja memasukan bug pada metode tambahAcara dan cariEvents pada kelas Jadwal. Meskipun bug tersebut telah disisipkan dalam kode, Squaretest tidak menghasilkan pengujian pada bug yang telah penulis buat.



Gambar 4.38 Hasil run test pada kelas JadwalTest Squaretest

c. Hasil test dari ManajemenJadwalTest Squaretest dan Analisis

Sebanyak 3 *unit test* telah dijalankan tanpa ada kegagalan, kesalahan, atau tes yang dilewati. Namun, terdapat 1 unit tes tidak berhasil dijalankan seperti yang ditampilkan pada Gambar 4.39 karena penulis menambahkan bug pada metode tambahAcara di kelas ManajemenJadwal.



Gambar 4.39 Hasil run test pada kelas ManajemenJadwalTest Squaretest

4.5 Penerapan Metode The Distance to The Ideal Alternative

Setelah menyelesaikan pengujian *unit test* secara otomatis pada keempat *framework* otomatis yang dipilih, langkah selanjutnya adalah mengumpulkan data yang diperlukan, baik yang telah dikumpulkan maupun yang akan dikumpulkan, dan menganalisis hasil pengujian tersebut dengan menerapkan metode *The Distance to The Ideal Alternatie* (DIA) sebagai pembanding. Metode DIA digunakan untuk mengavaluasi dan membandingkan kinerja dari setiap *framework* berdasarkan parameter-parameter yang telah ditentukan sebelumnya. Dengan menggunakan metode DIA, penulis dapat menentukan *framework* mana yang paling mendekati kondisi ideal berdasarkan parameter yang telah ditetapkan, dengan demikian, memungkinkan pemilihan *framework* yang paling optimal dari segi kinerja untuk pengujian *unit test*.

4.5.1 Pengumpulan Data

Proses pengumpulan data untuk keempat *framework* yaitu EvoSuite, Randoop, Github Copilot, dan Squaretest dilakukan dua pendekatan. Pertama, data dikumpulkan melalui pengujian langsung kepada keempat *framework* yang dilakukan oleh penulis. Kedua, data tambahan diperoleh melalui kuesioner yang diberikan kepada responden, dengan fokus pada sub parameter *usability* dan *readability*.

Tabel 4.1 Hasil pengujian otomatis setiap *framework*

Nama <i>Framework</i>	Multi Language	Script Creation	Cost	Pengujian Otomatis	Community Support and Documentation
EvoSuite	1 bahasa	03:55 menit	Gratis	Ada	Ada
Randoop	1 bahasa	01:40 menit	Gratis	Ada	Ada
Squaretest	1 bahasa	00:05 detik	Berbayar	Ada	Ada
Github Copilot	8 bahasa	02:00 menit	Berbayar	Ada	Ada

Tabel 4.2 Hasil pengujian otomatis setiap *framework*

Nama <i>Framework</i>	Sistem Pelacakan Bug	Execution Logging	Reporting	Code Coverage	Readability	Usability
EvoSuite	Tidak ada	Ada	Ada	62%	3.75	67.75
Randoop	Tidak Ada	Ada	Ada	22%	3.15	59.75
Squaretest	Tidak Ada	Tidak Ada	Ada	63%	3.9	79.25
Github Copilot	Tidak Ada	Tidak Ada	Ada	64%	3.75	81.25

a. Hasil pengumpulan data Readability

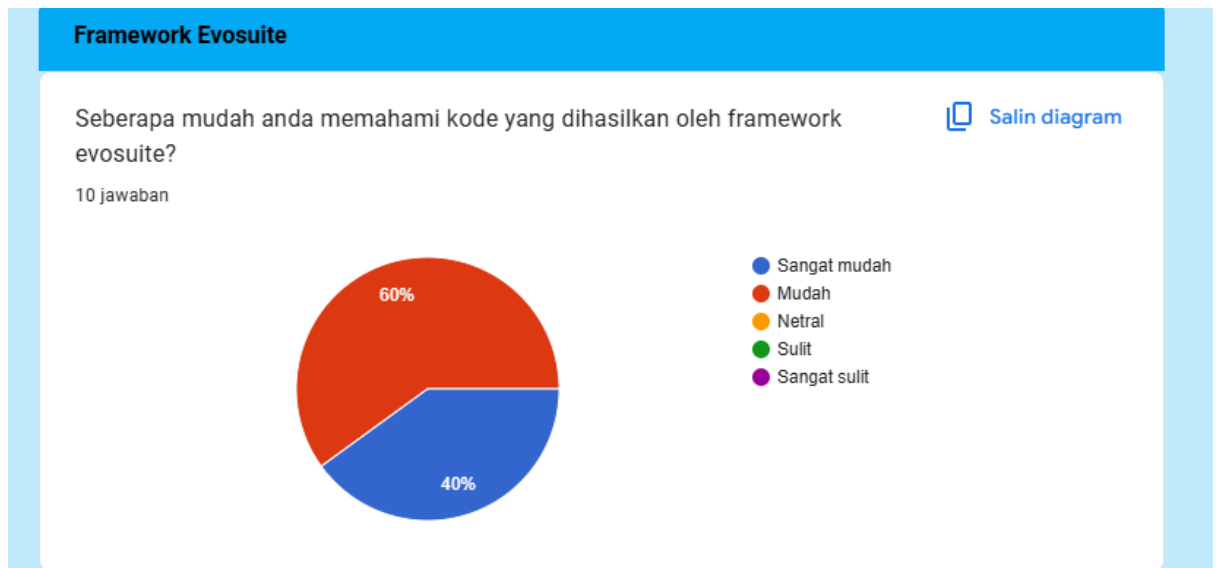
Dalam mengumpulkan data untuk sub parameter *Readability*, penulis menggunakan *skala likert* untuk menghitung bobot nilai *Readability* berdasarkan kuesoner yang disebarkan. *Skala linkert* dikembangkan pertama kali oleh Rensis Linkert pada tahun 1932 dalam mengukur sikap dan presepsi masyarakat (Mawardi, 2019). Oleh karena itu, skala ini digunakan untuk mengukur presepsi responden terhadap keterbacaan suatu kode. Pada kuesioner yang disusun oleh penulis, responden diminta untuk memberikan jawaban berdasarkan skala tertentu antara lain, ‘sangat mudah’, ‘mudah’, ‘netral’, ‘cukup sulit’, dan ‘sangat sulit’ dengan poin 5 untuk ‘sangat mudah’ hingga poin 1 untuk ‘sangat sulit’. Berdasarkan jawaban responden, poin tersebut akan dikalikan dengan jumlah responden yang memilih skala tersebut. Berikut merupakan contoh pengambilan bobot nilai untuk *Readability* pada *framework* EvoSuite.

a. Menghitung total skor setiap pertanyaan

Rumus yang digunakan adalah menggunakan persamaan (4.1)

$$\begin{aligned}
 \text{Total Skor} = & (\text{Nilai skala} * \text{jumlah responden yang memilih}) \\
 & + (\text{ilai skala} * \text{jumlah responden yang memilih}) \quad (4.1)
 \end{aligned}$$

Sebagai contoh, untuk pertanyaan pertama pada *framework* EvoSuite yang ditunjukkan pada Gambar 4.40, 6 responden memilih ‘sangat mudah’, dan 4 responden memilih ‘mudah’.



Gambar 4.40 Pertanyaan pada *framework* EvoSuite

Dengan demikian, perhitungan total skornya sebagai berikut :

$$Total\ skor = (4 * 5) + (6 * 4) = 20 + 24 = 44$$

b. Rata-rata skor

Setelah memperoleh total skor pada setiap pertanyaan, langkah selanjutnya adalah menghitung rata-rata dengan menggunakan persamaan (4.2)

$$rata - rata = \frac{Total\ Skor}{Jumlah\ Responden} \quad (4.2)$$

Dengan demikian, perhitungan rata-rata skor sebagai berikut :

$$Rata - rata = \frac{44}{10} = 4.4$$

c. Total keseluruhan skor

Untuk mendapatkan nilai keseluruhan, Rata-rata dari setiap pertanyaan dihitung menggunakan menggunakan persamaan (4.3). Dalam data yang diperoleh untuk setiap pertanyaan adalah 4.4, 4.4, 3.2, dan 3.0.

$$Total\ seluruh\ skor = \frac{Rata - rata\ setiap\ skor\ pertanyaann}{Jumlah\ pertanyaan} \quad (4.3)$$

Dengan demikian, nilai keseluruhan diperoleh hasilnya sebagai berikut :

$$Total\ seluruh\ skor\ pertanyaan = \frac{4.4 + 4.4 + 3.2 + 3.0}{4} = \frac{15.9}{4} = 3.75$$

Hasil pengumpulan data Usability

Dalam mengumpulkan data untuk sub parameter *Usability*, penulis menggunakan *SUS Score* untuk menghitung bobot nilai *Usability* berdasarkan kuesoner yang disebar. Proses perhitungan *SUS Score* terdapat 10 pertanyaan yang diberikan kepada responden dengan setiap pertanyaan menggunakan skala 1-5. Dalam perhitungan SUS, pertanyaan dibagi menjadi dua kategori yaitu, item positif berada pada pertanyaan ganjil, dan item negatif pada pertanyaan genap. Untuk setiap jawaban pada pertanyaan ganjil (1, 3, 5, 7, 9), nilai yang diberikan responden dikurangi dengan 1, sedangkan pada pertanyaan genap (2, 4, 6, 8, 10), nilai 5 dikurangi dengan nilai yang diberikan responden. Hasil dari setiap perhitungan ini kemudian dijumlahkan dan dikalikan dengan 2,5 untuk mendapatkan skor akhir (Sauro & Lewis, 2011). Berikut merupakan contoh pengambilan *SUS Score* untuk *Usability* pada *framework* EvoSuite.

Tabel 4.3. Jawaban Responden pada Framework EvoSuite

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10
R1	5	1	5	2	5	1	4	1	5	1
R2	5	1	4	2	4	2	4	5	4	2
R3	4	2	5	3	4	3	5	5	4	2
R4	2	3	4	2	4	3	3	2	4	4
R5	2	3	2	1	3	5	2	5	4	4
R6	4	2	4	2	4	2	4	2	4	2
R7	4	1	4	1	4	1	5	1	4	2
R8	2	1	5	1	4	1	5	1	4	2
R9	2	2	2	2	3	2	3	3	2	5
R10	2	3	3	3	4	3	4	3	4	3

Pada Tabel 4.3 menunjukkan jawaban tiap responden terhadap 10 pertanyaan SUS untuk *framework* EvoSuite. Setiap angka pada tabel mewakili jawaban responden berdasarkan skala 1-5, di mana '1' menunjukkan tidak setuju dan '5' menunjukkan sangat setuju.

Tabel 4.4 Hasil perhitungan SUS score

	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Jumlah	Total x 2.5
R1	4	4	4	3	4	4	3	4	4	4	38	95
R2	4	4	3	3	3	3	3	0	3	3	29	72.5
R3	3	3	4	2	3	2	4	0	3	3	27	67.5
R4	1	2	3	3	3	2	2	3	3	1	23	57.5
R5	1	2	1	4	2	0	1	0	3	1	15	37.5
R6	3	3	3	3	3	3	3	3	3	3	30	75
R7	3	4	3	4	3	4	4	4	3	3	35	87.5
R8	1	4	4	4	3	4	4	4	3	3	34	85
R9	1	3	1	3	2	3	2	2	1	0	18	45
R10	1	2	2	2	3	2	3	2	3	2	22	55
												677.5
												67.75

Pada Tabel 4.4 menunjukkan hasil perhitungan SUS Score untuk *framework* EvoSuite berdasarkan jawaban dari tiap responden. Setiap baris pada tabel mewakili jawaban dari satu responden, dengan total skor dihitung berdasarkan penjumlahan nilai setiap pertanyaan. Total skor dari setiap responden dikalikan dengan 2,5 untuk mendapatkan skor SUS per responden. Skor akhir untuk *framework* EvoSuite dihitung sebagai rata-rata dari seluruh total skor responden, dengan hasil akhir sebesar 67,75.

4.5.2 Penyusunan Matriks Keputusan

Berdasarkan data yang dikumpulkan pada bab sebelumnya mengenai pemilihan parameter dan sub parameter, penulis menyusun matriks keputusan untuk parameter dan sub parameter yang telah ditetapkan.

Matriks keputusan Technical Economic View

Penetapan nilai pada matriks keputusan yang ditampilkan pada Tabel 4.5 ditentukan berdasarkan hasil pengumpulan data yang ditampilkan pada Tabel 4.1 dan nilai dari setiap sub parameter pada matriks keputusan *Technical Economic View* diberikan dengan mengacu bobot yang telah ditentukan sebelumnya dalam Tabel 3.2, Tabel 3.3, dan Tabel 3.4.

Tabel 4.5 Hasil matriks keputusan parameter Technical Economic View

Alternatif	Sub Parameter		
	<i>Multi Language</i>	<i>Cost</i>	<i>Community Support and Documentation</i>
EvoSuite	0	1	1
Randoop	0	1	1
Squaretest	0	0	1
Github Copilot	1	0	1

Proses penentuan bobot dilakukan dengan langkah berikut:

a. Sub parameter Multi Laguange

Nilai pada sub parameter *multi laguange* ditentukan berdasarkan jumlah bahasa pemrograman yang didukung oleh *framework* sebagaimana ditampilkan pada Tabel 4.1 . Jika mendukung lebih dari lima bahasa, diberi nilai 1. Jika hanya mendukung kurang dari dua bahasa diberi nilai 0, bobot kriteria tersebut ditetapkan pada Tabel 3.2.

b. Sub parameter Cost

Penilaian sub parameter *cost* ditentukan berdasarkan status *framework* apakah gratis atau berbayar, sesuai hasil pengujian pada Tabel 4.1. Jika status *framework* gratis diberi bobot 1, sedangkan *framework* yang berbayar diberi bobot 0, bobot kriteria tesebut ditetapkan pada Tabel 3.3.

c. Sub parameter Community Support and Documentation

Nilai pada sub parameter ini diberikan berdasarkan dukungan komunitas dan dokumentasi yang tersedia pada *framework*, sebagaimana ditunjukan pada Tabel 3.4. Jika *framework* memiliki dukungan komunitas dan dokumentasi yang lengkap diberi bobot 1, sedangkan yang tidak memiliki dukungan memadai diberi bobot 0. Bobot kriteria tersebut ditetapkan pada Tabel 3.4.

Matriks keputusan Testing Process View

Penetapan nilai pada matriks keputusan yang ditampilkan pada Tabel 4.6 ditentukan berdasarkan hasil pengumpulan data yang ditampilkan pada Tabel 4.1 dan Tabel 4.2. Nilai dari setiap sub parameter pada matriks keputusan *Testing Process View* diberikan dengan mengacu bobot yang telah ditentukan sebelumnya dalam Tabel 3.6, sampai dengan Tabel 3.11.

Tabel 4.6 Hasil matriks keputusan parameter Testing Process View

Alternatif	Sub Parameter
------------	---------------

	Sistem Pelacakan Bug	Pengujian Otomatis	<i>Execution Logging</i>	<i>Script Creation</i>	<i>Reporting</i>	<i>Code Coverage</i>
EvoSuite	0	1	1	0.25	1	0.5
Randoop	0	1	1	0.75	1	0.25
Squaretest	0	1	0	1	0	0.5
Github Copilot	0	1	0	0.50	1	0.5

Proses penentuan bobot dilakukan dengan langkah berikut:

a. Sub parameter Sistem Pelacakan Bug

Nilai pada sub parameter sistem pelacakan bug ditentukan berdasarkan fitur pelacakan bug yang terdapat pada *framework* sebagaimana ditampilkan pada Tabel 4.2. Jika *framework* memiliki fitur pelacakan bug, diberi bobot 1. Jika tidak memiliki fitur pelacakan bug, diberi bobot 0, bobot kriteria tersebut ditetapkan pada Tabel 3.6.

b. Sub parameter Pengujian Otomatis

Nilai pada sub parameter pengujian otomatis ditentukan berdasarkan *framework* dapat melakukan pengujian otomatis sebagaimana ditampilkan pada Tabel 4.1. Jika *framework* dapat melakukan pengujian otomatis, diberi nilai 1. Jika tidak dapat melakukan pengujian otomatis, diberi bobot 0, bobot kriteria tersebut ditetapkan pada Tabel 3.7.

c. Sub parameter Execution Logging

Nilai pada sub parameter *execution logging* ditentukan berdasarkan *framework* memiliki laporan selama pengujian berlangsung, sebagaimana ditampilkan pada Tabel 4.2. Jika *framework* memiliki laporan selama pengujian berlangsung, diberi bobot 1. Jika tidak memiliki laporan selama pengujian berlangsung, diberi bobot 0. Bobot kriteria tersebut ditetapkan pada Tabel 3.8.

d. Sub parameter Script Creation

Nilai pada sub parameter *script creation* ditentukan berdasarkan waktu yang dibutuhkan *framework* untuk menghasilkan *unit test*, sebagaimana ditampilkan pada Tabel 4.1. Jika *framework* dapat menghasilkan tes kurang dari 1 menit diberi bobot 1. Jika *framework* menghasilkan tes lebih dari 5 menit diberi bobot 0. Bobot kriteria tersebut ditetapkan pada Tabel 3.9.

e. Sub parameter Reporting

Nilai pada sub parameter *reporting* ditentukan berdasarkan *framework* memiliki laporan setelah pengujian selesai, sebagaimana ditampilkan pada Tabel 4.2. Jika *framework* terdapat

laporan setelah pengujian selesai, diberi bobot 1. Jika *framework* tidak terdapat laporan setelah pengujian selesai, diberi bobot 0. Bobot kriteria tersebut ditetapkan pada Tabel 3.10.

f. Sub parameter Code Coverage

Nilai pada sub parameter *code coverage* ditentukan berdasarkan tingkat cakupan kode yang dicapai oleh setiap *framework*, sebagaimana ditampilkan pada Tabel 4.2. *Framework* yang mencapai *code coverage* lebih dari 80%, diberi bobot 1. Jika *framework* kurang dari 50%, diberi bobot 0. Bobot kriteria tersebut ditetapkan pada Tabel 3.11.

Matriks keputusan Quality Standard View

Penetapan nilai pada matriks keputusan yang ditampilkan pada Tabel 4.7 ditentukan berdasarkan hasil pengumpulan data yang ditampilkan pada Tabel 4.2. Nilai dari setiap sub parameter pada matriks keputusan *Quality Standard View* diberikan dengan mengacu bobot yang telah ditentukan sebelumnya dalam Tabel 3.13 dan Tabel 3.14.

Tabel 4.7 Hasil matriks keputusan parameter Quality Standard View

Alternatif	Sub Parameter	
	Readability	Usability
EvoSuite	0.75	0.50
Randooop	0.50	0.25
Squaretest	0.75	0.75
Github Copilot	0.75	1

Proses penentuan bobot dilakukan dengan langkah berikut

a. Sub Parameter Readability

Nilai pada sub parameter *readability* ditentukan berdasarkan seberapa mudah kode dapat dipahami oleh pengguna. *Readability* diukur berdasarkan skala rentang nilai yang ditampilkan pada Tabel 3.13. *Framework* dengan nilai *readability* 4,5 atau lebih diberi bobot 1, sedangkan *framework* dengan nilai *readability* kurang dari 1,5 diberi bobot 0.

b. Sub Parameter Usability

Nilai pada sub parameter *usability* ditentukan berdasarkan skor SUS yang menggambarkan seberapa mudah *framework* tersebut digunakan oleh pengguna. Rentang skor SUS ini, sebagaimana ditampilkan pada Tabel 3.14 yang digunakan untuk menetapkan bobot nilai. *Framework* dengan skor SUS 80 atau lebih diberi bobot 1, sedangkan *framework* dengan skor kurang dari 50 diberi 0.

4.5.3 Normalisasi Matriks Keputusan

Langkah normalisasi matriks keputusan dilakukan dengan menggunakan persamaan (2.3). Normalisasi dilakukan untuk menghasilkan data normalisasi dari setiap parameter. Proses normalisasi matriks keputusan diterapkan pada parameter *Testing Process View* dan *Quality Standard View*.

Normalisasi Matriks keputusan Technical Economic View

Dengan menggunakan persamaan (2.3). Normalisasi matriks keputusan dilakukan berdasarkan data matriks keputusan yang ditampilkan pada Tabel 4.5 yaitu parameter *Technical Economic View*. Sebagai contoh perhitungan pada *framework* EvoSuite pada sub parameter *Multi Language* berdasarkan Tabel 4.5, hasil normalisasi yang diperoleh sebagai berikut :

$$\frac{0}{\sqrt{0^2 + 0^2 + 0^2 + 1^2}} = 0$$

Untuk melihat hasil perhitungan lengkap dari perhitungan normalisasi matriks keputusan pada setiap sub parameter *Technical Economic View*, data tersebut disajikan pada Tabel 4.8.

Tabel 4.8 Hasil normalisasi matriks keputusan Technical Economic View

Alternatif	Sub Parameter		
	<i>Multi Language</i>	<i>Cost</i>	<i>Community Support and Documentation</i>
EvoSuite	0	0.71	0.50
Randoop	0	0.71	0.50
Squaretest	0	0	0.50
Github Copilot	1	0	0.50

Dari hasil normalisasi pada Tabel 4.8 untuk parameter *Testing Process View*, yang mencakup sub parameter *multi laguange*, *cost*, dan *community support and documentation*. Nilai yang lebih tinggi menunjukkan kinerja yang lebih baik pada sub parameter terkait.

Normalisasi Matriks keputusan Testing Process View

Dengan menggunakan persamaan (2.3). Normalisasi matriks keputusan dilakukan berdasarkan data matriks keputusan yang ditampilkan pada Tabel 4.6 untuk parameter *Testing Process View*. Pada Tabel 4.9 ditampilkan hasil normalisasi keputusan untuk setiap sub parameter *Testing Process View*.

Tabel 4.9 Hasil normalisasi keputusan Testing Process view

Alternatif	Sub Parameter
------------	---------------

	Sistem Pelacakan Bug	Pengujian Otomatis	<i>Execution Logging</i>	<i>Script Creation</i>	<i>Reporting</i>	<i>Code Coverage</i>
EvoSuite	0	0.50	0.71	0.18	0.58	0.55
Randoop	0	0.50	0.71	0.55	0.58	0.27
Squaretest	0	0.50	0	0.73	0	0.55
Github Copilot	0	0.50	0	0.36	0.58	0.55

Dari hasil normalisasi pada Tabel 4.9, untuk parameter *Testing Process View*, yang mencakup sub parameter sistem pelacakan bug, pengujian otomatis, *execution logging*, *script creation*, *reporting*, dan *code coverage*. Nilai yang lebih tinggi menunjukkan kinerja yang lebih baik pada sub parameter terkait.

Normalisasi Matriks keputusan Quality Standard View

Dengan menggunakan persamaan (2.3). Normalisasi matriks keputusan dilakukan berdasarkan data matriks keputusan yang ditampilkan pada Tabel 4.7 untuk parameter *Quality Standard View*. Pada Tabel 4.10 ditampilkan hasil normalisasi keputusan untuk setiap sub parameter *Quality Standard View*.

Tabel 4.10 Hasil normalisasi matriks keputusan Quality Standard View

Alternatif	Sub Parameter	
	Readability	Usability
EvoSuite	0.54	0.36
Randoop	0.36	0.18
Squaretest	0.54	0.55
Github Copilot	0.54	0.73

Dari hasil normalisasi pada Tabel 4.10, untuk parameter *Quality Standard View*, yang mencakup sub parameter *readability* dan *usability*. Nilai yang lebih tinggi menunjukkan kinerja yang lebih baik pada sub parameter terkait.

4.5.4 Pembobotan Matriks Normalisasi

Pada tahap ini, hasil dari normalisasi matriks keputusan dikalkulasikan dengan bobot parameter atau sub parameter sesuai dengan persamaan (2.5) dan persamaan (2.6). Proses pembobotan matriks normalisasi diterapkan pada setiap parameter *Technical Economic View*, *Testing Process View*, dan *Quality Standard View*.

Pembobotan Matriks Normalisasi Parameter Technical Economic View

Dengan menggunakan persamaan (2.5) dan persamaan (2.6). pembobotan matriks normalisasi dilakukan dengan menggunakan data normalisasi matriks keputusan pada Tabel 4.8 dan dikalkulasikan menggunakan bobot yang telah ditentukan untuk setiap sub parameter pada Tabel 3.1. Untuk perhitungan pada *framework* EvoSuite pada sub parameter *multi language*, hasil pembobotan matriks normalisasi yang diperoleh sebagai berikut :

$$(0 \times 0.27) = 0$$

Untuk melihat hasil perhitungan lengkap dari perhitungan pembobotan matriks normalisasi keputusan pada setiap sub parameter *Technical Economic View*, data tersebut disajikan pada Tabel 4.11.

Tabel 4.11 Hasil pembobotan matriks normalisasi Technical Economic View

Alternatif	Sub Parameter		
	<i>Multi Language</i>	<i>Cost</i>	<i>Community Support and Documentation</i>
EvoSuite	0	0.241	0.195
Randoop	0	0.241	0.195
Squaretest	0	0	0.195
Github Copilot	0.270	0	0.195

Dari hasil pembobotan matriks normalisasi yang ditampilkan pada Tabel 4.11, skor yang diperoleh menunjukkan seberapa baik setiap *framework* dalam masing-masing sub parameter pada parameter *Technical Economic View*. *Framework* dengan skor yang lebih tinggi menunjukkan kinerja yang lebih baik pada sub parameter yang memiliki bobot penilaian yang lebih tinggi.

Pembobotan Matriks Normalisasi Testing Process View

Dengan menggunakan persamaan (2.5) dan persamaan (2.6). pembobotan matriks normalisasi dilakukan dengan menggunakan data normalisasi matriks keputusan pada Tabel 4.9 dan dikalkulasikan menggunakan bobot yang telah ditentukan untuk setiap sub parameter pada Tabel 3.5. Untuk perhitungan pada *framework* EvoSuite pada sub parameter sistem pelacakan bug, hasil pembobotan matriks normalisasi yang diperoleh sebagai berikut :

$$(0 \times 0.19) = 0$$

Untuk melihat hasil perhitungan lengkap dari perhitungan pembobotan matriks normalisasi keputusan pada setiap sub parameter *Testing Process View*, data tersebut disajikan pada Tabel 4.12.

Tabel 4.12 Hasil pembobotan matriks normalisasi Testing Process View

Alternatif	Sub Parameter					
	Sistem Pelacakan Bug	Pengujian Otomatis	<i>Execution Logging</i>	<i>Script Creation</i>	<i>Reporting</i>	<i>Code Coverage</i>
EvoSuite	0	0.110	0.085	0.027	0.087	0.094
Randoop	0	0.110	0.085	0.082	0.087	0.046
Squaretest	0	0.110	0	0.109	0	0.094
Github Copilot	0	0.110	0	0.054	0.087	0.094

Dari hasil pembobotan matriks normalisasi yang ditampilkan pada Tabel 4.12, skor yang diperoleh menunjukkan seberapa baik setiap *framework* dalam masing-masing sub parameter pada parameter *Testing Process View*. *Framework* dengan skor yang lebih tinggi menunjukkan kinerja yang lebih baik pada sub parameter yang memiliki bobot penilaian yang lebih tinggi.

Pembobotan Matriks Normalisasi Quality Standard View

Dengan menggunakan persamaan (2.5) dan persamaan (2.6). pembobotan matriks normalisasi dilakukan dengan menggunakan data normalisasi matriks keputusan pada Tabel 4.10 dan dikalkulasikan menggunakan bobot yang telah ditentukan untuk setiap sub parameter pada Tabel 3.12. Untuk perhitungan pada *framework* EvoSuite pada sub parameter sistem pelacakan bug, hasil pembobotan matriks normalisasi yang diperoleh sebagai berikut :

$$(0.54 \times 0.48) = 0.259$$

Untuk melihat hasil perhitungan lengkap dari perhitungan pembobotan matriks normalisasi keputusan pada setiap sub parameter *Quality Standard View*, data tersebut disajikan pada Tabel 4.13.

Tabel 4.13 Hasil pembobotan matriks normalisasi Quality Standard View

Alternatif	Sub Parameter	
	Readability	Usability
EvoSuite	0.259	0.187
Randoop	0.172	0.094
Squaretest	0.259	0.286
Github Copilot	0.259	0.379

Dari hasil pembobotan matriks normalisasi yang ditampilkan pada Tabel 4.13, skor yang diperoleh menunjukkan seberapa baik setiap *framework* dalam masing-masing sub parameter pada parameter *Quality Standard View*. *Framework* dengan skor yang lebih tinggi

menunjukkan kinerja yang lebih baik pada sub parameter yang memiliki bobot penilaian yang lebih tinggi.

4.5.5 Menentukan Solusi Ideal Positif dan Negatif Parameter Technical Economic View

Pada langkah ini, penulis menetapkan alternatif positif dan negatif untuk setiap sub parameter berdasarkan nilai pada Tabel 4.11 menggunakan persamaan persamaan (2.7) dan persamaan (2.8). Solusi ideal positif ditetapkan berdasarkan nilai terbesar, sedangkan Solusi ideal negatif ditetapkan dari nilai terkecil dalam tabel Tabel 4.11 untuk masing-masing sub parameter.

Tabel 4.14 Hasil solusi ideal positif Technical Economic View

Solusi Ideal Positif	Sub parameter	$A^+ = \max V_{ii} = [V_1^+ V_2^+ \dots V_3^+]$
$V1^+$	<i>Multi Language</i>	0.270
$V2^+$	<i>Cost</i>	0.241
$V3^+$	<i>Community support and documentation</i>	0.195

Dari Tabel 4.14 yang ditampilkan bahwa solusi ideal positif pada sub parameter *multi language* memiliki nilai tertinggi sebesar 0.270, pada sub parameter *cost* sebesar 0.241, dan pada sub parameter *community support and documentation* sebesar 0.195.

Tabel 4.15 Hasil solusi ideal negatif Technical Economic View

Solusi Ideal negatif	Sub parameter	$A^- = \max V_{ii} = [V_1^- V_2^- \dots V_3^-]$
$V1^-$	<i>Multi Language</i>	0
$V2^-$	<i>Cost</i>	0
$V3^-$	<i>Community support and documentation</i>	0.195

Dari Tabel 4.15 yang ditampilkan bahwa solusi ideal negatif pada sub parameter *multi language* dan *cost* memiliki nilai terendah yaitu 0. Sementara itu, pada sub parameter *community support and documentation* nilai adalah 0.195.

4.5.6 Menentukan Solusi Ideal Positif dan Negatif Parameter Testing Process View

Pada langkah ini, penulis menetapkan alternatif positif dan negatif untuk setiap sub parameter berdasarkan nilai pada Tabel 4.12 menggunakan persamaan persamaan (2.7) dan persamaan (2.8). Solusi ideal positif ditetapkan berdasarkan nilai terbesar, sedangkan Solusi ideal negatif ditetapkan dari nilai terkecil dalam tabel Tabel 4.12 untuk masing-masing sub parameter.

Tabel 4.16 Hasil solusi ideal positif Testing Process View

Solusi Ideal Positif	Sub parameter	$A^+ = \max V_{ii} = [V_1^+ V_2^+ \dots V_3^+]$
V_1^+	Sistem Pelacakan Bug	0
V_2^+	Pengujian Otomatis	0.110
V_3^+	<i>Execution Logging</i>	0.085
V_4^+	<i>Script Creation</i>	0.109
V_5^+	<i>Reporting</i>	0.087
V_6^+	<i>Code Coverage</i>	0.094

Dari Tabel 4.16 yang ditampilkan bahwa solusi ideal positif pada sub parameter sistem pelacakan bug nilai adalah 0, pada sub parameter pengujian otomatis nilai tertinggi adalah 0.110, pada sub parameter *execuition logging* nilai tertinggi adalah 0.085, pada sub parameter *script creation* nilai tertinggi adalah 0.109, pada sub parameter *reporting* nilai tertinggi adalah 0.087, dan pada sub parameter *code coverage* nilai tertinggi adalah 0.094.

Tabel 4.17 Hasil solusi ideal negatif Testing Process View

Solusi Ideal negatif	Sub parameter	$A^- = \max V_{ii} = [V_1^- V_2^- \dots V_3^-]$
V_1^-	Sistem Pelacakan Bug	0
V_2^-	Pengujian Otomatis	0.110
V_3^-	<i>Execution Logging</i>	0
V_4^-	<i>Script Creation</i>	0.027
V_5^-	<i>Reporting</i>	0
V_6^-	<i>Code Coverage</i>	0.046

Dari Tabel 4.17 yang ditampilkan bahwa solusi ideal negatif pada sub parameter sistem pelacakan bug nilai terendah adalah 0, pada sub parameter pengujian otomatis nilai terendah adalah 0.110, pada sub parameter *execuition logging* nilai terendah adalah 0, pada sub parameter *script creation* nilai terendah adalah 0.027, pada sub parameter *reporting* nilai terendah adalah 0, dan pada sub parameter *code coverage* nilai terendah adalah 0.046.

4.5.7 Menentukan Solusi Ideal Positif dan Negatif Parameter Quality Standard View

Pada langkah ini, penulis menetapkan alternatif positif dan negatif untuk setiap sub parameter berdasarkan nilai pada Tabel 4.13 menggunakan persamaan persamaan (2.7) dan persamaan (2.8). Solusi ideal positif ditetapkan berdasarkan nilai terbesar, sedangkan Solusi ideal negatif ditetapkan dari nilai terkecil dalam tabel Tabel 4.13 untuk masing-masing sub parameter.

Tabel 4.18 Hasil solusi ideal positif Quality Standard View

Solusi Ideal Positif	Sub parameter	$A^+ = \max V_{ii} = [V_1^+ V_2^+ \dots V_3^+]$
----------------------	---------------	---

V_1^+	<i>Readability</i>	0.259
V_2^+	<i>Usability</i>	0.379

Dari Tabel 4.18 yang ditampilkan bahwa solusi ideal positif pada sub parameter *readability* nilai tertinggi adalah 0.259 dan pada sub parameter *usability* nilai tertinggi adalah 0.379.

Tabel 4.19 Hasil solusi ideal negatif Quality Standard View

Solusi Ideal Positif	Sub parameter	$A^- = \max V_{ii} = [V_1^- V_2^- \dots V_3^-]$
V_1^-	<i>Readability</i>	0.172
V_2^-	<i>Usability</i>	0.094

Dari Tabel 4.19Tabel 4.18 yang ditampilkan bahwa solusi ideal negatif pada sub parameter *readability* nilai terendah adalah 0.172 dan pada sub parameter *usability* nilai terendah adalah 0.094.

4.5.8 Perhitungan jarak Manhattan

Pada tahap ini, penulis menentukan hasil perhitungan jarak Manhattan maksimum dan minimum dengan menggunakan persamaan (2.9) dan persamaan (2.10). Proses perhitungan jarak manhattan diterapkan pada setiap parameter *Technical EconomicView*, *Testing Process View*, dan *Quality Standard View*.

Jarak Manhattan Technical Economic view

Dengan menggunakan persamaan (2.9) dan persamaan (2.10) untuk mendapatkan hasil perhitungan jarak manhattan maksimum dan minimum dilakukan berdasarkan hasil pembobotan matriks normalisasi pada Tabel 4.11, serta solusi ideal positif dan solusi ideal negatif yang ditentukan pada Tabel 4.14 dan Tabel 4.15. Untuk menghitung jarak manhattan pada framework EvoSuite untuk sub parameter *multi language*, diperoleh hasil sebagai berikut.

$$\begin{aligned}
 D_1^+ &= [V_{1,1} - a_1^+] + [V_{2,1} - a_2^+] + [V_{3,1} - a_3^+] \\
 &= (0 - 0.270) + (0.241 - 0.241) + (0.195 - 0.195) \\
 &= 0.270 \text{ (mahattan maksimum)}
 \end{aligned}$$

$$\begin{aligned}
 D_1^- &= [V_{1,1} - a_1^-] + [V_{2,1} - a_2^-] + [V_{3,1} - a_3^-] \\
 &= (0 - 0) + (0 - 0.241) + (0.195 - 0.195) \\
 &= 0.241 \text{ (mahattan minimum)}
 \end{aligned}$$

Untuk melihat hasil perhitungan lengkap dari jarak Manhattan maximum dan minimum pada setiap Alternatif, data tersebut disajikan pada Tabel 4.20 dan Tabel 4.21.

Tabel 4.20 Jarak manhattan max technical economic view

Alternatif	Manhattan maximum	nilai
EvoSuite	D_1^+	0.270
Randoop	D_2^+	0.270
Squaretest	D_3^+	0.511
Github Copilot	D_4^+	0.241

Dari hasil perhitungan jarak manhattan maximum pada Tabel 4.20, dapat dilihat bahwa Github Copilot memiliki nilai terendah (0.241), diikuti oleh Evosuite dan Randoop dengan nilai yang sama (0.270). Squaretest memiliki nilai tertinggi (0.511). Nilai jarak maksimum yang lebih rendah menunjukkan kinerja *framework* yang lebih dekat dengan solusi ideal positif. Oleh karena itu, Github Copilot memiliki kinerja terbaik yang baik pada parameter *Technical Econmic View*.

Tabel 4.21 Jarak manhattan min technical economic view

Alternatif	Manhattan minimum	nilai
EvoSuite	D_1^-	0.241
Randoop	D_2^-	0.241
Squaretest	D_3^-	0
Github Copilot	D_4^-	0.270

Dari hasil perhitungan jarak manhattan minimum pada Tabel 4.21, bahwa Squaretest memiliki nilai terendah adalah nilai 0, menandakan *framework* tersebut mendekati solusi ideal negatif. Sedangkan Github Copilot memiliki nilai tertinggi (0.270) yang menunjukkan jaraknya paling jauh dari solusi ideal negatif. Jarak minimum yang lebih besar menunjukkan kinerja *framework* yang lebih baik karena semakin jauh dari kondisi yang tidak diinginkan.

Jarak Manhattan Testing Process View

Dengan menggunakan persamaan (2.9) dan persamaan (2.10) untuk mendapatkan hasil perhitungan jarak manhattan maksimum dan minimum dilakukan berdasarkan hasil pembobotan matriks normalisasi pada Tabel 4.12, serta solusi ideal positif dan solusi ideal negatif yang ditentukan pada Tabel 4.16 dan Tabel 4.17. Berikut hasil lengkap dari perhitungan jarak manhattan yang disajikan pada Tabel 4.22 dan Tabel 4.23.

Tabel 4.22 Jarak manhattan max testing process view

Alternatif	Manhattan maximum	nilai
EvoSuite	D_1^+	0.082
Randoop	D_2^+	0.075
Squaretest	D_3^+	0.171
Github Copilot	D_4^+	0.140

Dari hasil perhitungan jarak manhattan maximum pada Tabel 4.22, dapat dilihat bahwa Randoop memiliki nilai terendah (0.075), diikuti oleh EvoSuite (0.082). Github Copilot memiliki nilai yang tinggi (0.140), sedangkan Squaretest memiliki nilai yang lebih tinggi (0.171). Nilai jarak maksimum yang lebih rendah menunjukkan kinerja framework yang lebih dekat dengan solusi ideal positif. Oleh karena itu, Randoop menunjukkan kinerja terbaik pada parameter *Testing Process View*.

Tabel 4.23 Jarak manhattan min testing process view

Alternatif	Manhattan minimum	nilai
EvoSuite	D_1^-	0.219
Randoop	D_2^-	0.226
Squaretest	D_3^-	0.129
Github Copilot	D_4^-	0.161

Dari hasil perhitungan jarak manhattan minimum pada Tabel 4.23, Squaretest memiliki nilai terendah (0.129), disusul oleh EvoSuite memiliki nilai (0.161), yang menandakan bahwa *framework* tersebut mendekati solusi ideal negatif. Sedangkan Randoop memiliki nilai tertinggi (0.226), menunjukkan jarak yang paling jauh dari solusi ideal negatif. Jarak minimum yang lebih besar menunjukkan kinerja *framework* yang lebih baik karena semakin jauh dari kondisi yang tidak diinginkan.

Jarak Manhattan Quality Standard View

Dengan menggunakan persamaan (2.9) dan persamaan (2.10) untuk mendapatkan hasil perhitungan jarak manhattan maksimum dan minimum dilakukan berdasarkan hasil pembobotan matriks normalisasi pada Tabel 4.13, serta solusi ideal positif dan solusi ideal negatif yang ditentukan pada Tabel 4.18 dan Tabel 4.19. Berikut hasil lengkap dari perhitungan jarak manhattan yang disajikan pada Tabel 4.24 dan Tabel 4.25.

Tabel 4.24 Jarak manhattan max quality standard view

Alternatif	Manhattan maximum	nilai
EvoSuite	D_1^+	0.192
Randoop	D_2^+	0.372

Squaretest	D_3^+	0.093
Github Copilot	D_4^+	0

Dari hasil perhitungan jarak manhattan maximum pada Tabel 4.24, dapat dilihat bahwa Github Copilot memiliki nilai terendah (0), diikuti oleh Squaretest (0.093) lalu EvoSuite dengan nilai (0.192). Randoop memiliki nilai yang tinggi (0.372). Nilai jarak maksimum yang lebih rendah menunjukkan kinerja framework yang lebih dekat dengan solusi ideal positif. Oleh karena itu, Randoop menunjukkan kinerja terbaik pada parameter *Quality standard view*.

Tabel 4.25 Jarak manhattan min quality standard view

Alternatif	Manhattan minimum	nilai
EvoSuite	D_1^-	0.180
Randoop	D_2^-	0
Squaretest	D_3^-	0.279
Github Copilot	D_4^-	0.372

Dari hasil perhitungan jarak manhattan minimum pada Tabel 4.25, Randoop memiliki nilai terendah (0), disusul oleh EvoSuite memiliki nilai (0.180), yang menandakan bahwa *framework* tersebut mendekati solusi ideal negatif. Sedangkan Github Copilot memiliki nilai tertinggi (0.372), menunjukkan jarak yang paling jauh dari solusi ideal negatif. Jarak minimum yang lebih besar menunjukkan kinerja *framework* yang lebih baik karena semakin jauh dari kondisi yang tidak diinginkan.

4.5.9 Menentukan *Positive Ideal Alternative (PIA)*

Berdasarkan persamaan (2.11) untuk masing-masing parameter dapat ditentukan dengan mencari nilai terendah pada jarak Manhattan maksimum dan nilai tertinggi pada jarak Manhattan minimum. Nilai PIA untuk masing-masing parameter adalah sebagai berikut.

Nilai PIA pada parameter *Technical Economic View* adalah :

$$PIA = 0.241; 0.368$$

Nilai PIA pada parameter *Testing Process View* adalah :

$$PIA = 0.075; 0.226$$

Nilai PIA pada parameter *Quality Standard View* adalah :

$$PIA = 0; 0.372$$

4.5.10 Mengidentifikasi Peringkat

Pada tahap ini mengidentifikasi peringkat merupakan langkah terakhir untuk menentukan peringkat berdasarkan nilai terendah, yang menunjukkan jarak terpendek dari alternatif ideal dengan menggunakan persamaan (2.12), proses ini menghasilkan nilai R_i untuk setiap

parameter. Proses identifikasi peringkat diterapkan pada setiap parameter *Technical Economic View*, *Testing Process View*, dan *Quality Standard View*.

Pemilihan *Framework* Terbaik untuk parameter *Technical Economic View*

Berdasarkan persamaan (2.12), digunakan untuk menerapkan hasil perhitungan jarak manhattan maksimum dan minimum, dimana nilai R_i dihitung sebagai jarak gabungan dari kedua hasil tersebut, maka diperoleh hasil perhitungan nilai R_i untuk *framework* EvoSuite sebagai berikut.

$$\sqrt{(0.270 - 0.241)^2 + (0.241 - 0.270)^2} = \sqrt{0.001682} = 0.041$$

Untuk melihat hasil lengkap dan perhitungan pada masing-masing *framework*, data tersebut disajikan dalam **Error! Reference source not found..**

Tabel 4.26 Hasil peringkat pada *technical economic view*

<i>Framework</i>	R_i	Peringkat
Github Copilot	0	1
Randoop	0.041	2
EvoSuite	0.041	2
Squaretest	0.382	3

Dari hasil perhitungan nilai R_i yang ditampilkan pada Tabel 4.26, dapat dilihat bahwa Github Copilot memiliki nilai R_i yang rendah yaitu (0), yang menunjukkan kedekatan dengan solusi ideal positif dan jarak terjauh dari solusi ideal negatif. Oleh karena itu, Github Copilot menempati peringkat pertama pada parameter *Technical Economic View*. Sementara, Randoop dan EvoSuite memiliki nilai R_i yang sama yaitu (0.041), sehingga keduanya berada di peringkat kedua. Squaretest, dengan nilai R_i tertinggi (0.382) berada di peringkat ketiga.

Pemilihan *Framework* Terbaik untuk parameter *Testing Process View*

Hasil peringkat *framework* pada parameter *Testing Process View* sebagai berikut.

Tabel 4.27 Hasil peringkat pada *testing process view*

<i>Framework</i>	R_i	Peringkat
Randoop	0	1
EvoSuite	0.009	2
Github Copilot	0.092	3
Squaretest	0.136	4

Dari hasil perhitungan nilai R_i yang ditampilkan pada Tabel 4.27, dapat dilihat bahwa Randoop memiliki nilai R_i yang rendah yaitu (0), yang menunjukkan kedekatan dengan solusi ideal positif dan jarak terjauh dari solusi ideal negatif. Oleh karena itu, Randoop menempati

peringkat pertama pada parameter *Testing Process View*. Selanjutnya, EvoSuite berada di peringkat kedua dengan nilai R_i yaitu (0.009), diikuti oleh Github Copilot di peringkat ketiga dengan nilai R_i yaitu (0.092). Squaretest berada di peringkat keempat dengan nilai R_i tertinggi (0.136) berada di peringkat keempat.

Pemilihan *Framework* Terbaik untuk parameter *Quality Standard View*

Hasil peringkat *framework* pada parameter *Quality Standard View* sebagai berikut.

Tabel 4.28 Hasil peringkat pada *quality standard view*

<i>Framework</i>	R_i	Peringkat
Github Copilot	0	1
Squaretest	0.131	2
EvoSuite	0.272	3
Randoop	0.526	4

Dari hasil perhitungan nilai R_i yang ditampilkan pada Tabel 4.28, dapat dilihat bahwa Github Copilot kembali menempati peringkat pertama dengan nilai R_i yang rendah yaitu (0), yang menunjukkan kedekatan dengan solusi ideal positif dan jarak terjauh dari solusi ideal negatif. Oleh karena itu, Github Copilot menempati peringkat pertama pada parameter *Quality Standard View*. Selanjutnya, Squaretest berada di peringkat kedua dengan nilai R_i yaitu (0.131), diikuti oleh EvoSuite berada di peringkat ketiga dengan nilai R_i yaitu (0.272). Terakhir Randoop berada di peringkat keempat dengan nilai R_i tertinggi yaitu (0.526).

4.6 Melakukan Analisis Perbandingan

Berdasarkan hasil nilai R_i dari ketiga parameter yang telah ditentukan, dilakukan analisis kembali untuk membandingkan nilai dari ketiga parameter. Analisis ini menggunakan Weighted Scoring Model, menurut (Ayan dkk., 2023) metode ini dapat diintegrasikan dengan metode lain dalam proses pembobotan atau mencari alternatif pemeringkatan. Dengan demikian, akan mempermudah untuk mendapatkan hasil akhir yang valid agar perbandingan *unit test automation framework* bisa terkakulasi secara keseluruhan.

4.6.1 Peringkat Untuk Masing-Masing Parameter

Berdasarkan hasil peringkat yang diperoleh pada Tabel 4.26 hingga Tabel 4.28 dan bobot yang sudah ditentukan pada Tabel 3.15, berikut ini disajikan tabel peringkat beserta bobot pada Tabel 4.29 untuk masing-masing *framework*.

Tabel 4.29 Hasil peringkat dan bobot masing-masing *framework*

Parameter	EvoSuite	Randoop	Squaretest	Github Copilot	Bobot
-----------	----------	---------	------------	----------------	-------

Technical Economic View	2	2	3	1	0.26
Testing Process View	2	1	4	3	0.39
Quality Standard View	3	4	2	1	0.35

4.6.2 Menghitung Skor *Weighted Scoring*

Pada tahap ini dilakukan perhitungan skor total dengan menggunakan metode *weigted scoring*. Peringkat setiap *framework* pada masing-masing parameter dikalkulasikan dengan bobot yang telah ditentukan, yang tertampil pada Tabel 4.29. Berikut adalah contoh perhitungan untuk *Weighted Scoring* pada *framework* EvoSuite

$$\text{Skor} = (2 \times 0.26) + (2 \times 0.39) + (3 \times 0.35) = 2.35$$

Untuk melihat hasil lengkap dan perhitungan pada masing-masing *framework*, data tersebut disajikan dalam tabel Tabel 4.30.

Tabel 4.30 Hasil Peringkat Akhir

Nama <i>Framework</i>	Total Skor	Peringkat
Github Copilot	1.78	1
Randoop	2.31	2
EvoSuite	2.35	3
Squaretest	3.04	4

4.6.3 Alternatif Terbaik untuk *Framework*

Dari hasil perhitungan di atas, Github copilot menjadi alternatif terbaik dengan total skor terendah (1.78) dan menempati peringkat pertama. Randoop berada di peringkat kedua dengan skor (2.31), diikuti oleh EvoSuite di peringkat ketiga dengan skor (2.35), dan Squaretest menempati peringkat keempat dengan skor (3.04). Pada EvoSuite dan Randoop memerlukan setup yang cukup kompleks dan memakan waktu, terutama pada Randoop yang memerlukan banyak dependensi. Sebaliknya, Github Copilot dan Squeretest menunjukkan kemudahan dalam setup, hanya memerlukan sedikit setup awal. Dalam hal eksekusi pengujian, Squaretest memiliki waktu eksekusi yang relatif cepat, meskipun hasil pengujiannya seringkali memerlukan modifikasi manual. Sebaliknya, Github Copilot, EvoSuite, dan Randoop membutuhkan waktu eksekusi yang lebih lama, terutama pada EvoSuite yang memiliki proses eksekusi yang cukup memakan waktu. Dalam hal laporan pengujian, EvoSuite dan Randoop

menghasilkan laporan atau log saat pengujian dan laporan saat menghasilkan unit test. Sedangkan, Github Copilot dan Squaretest tidak menyediakan laporan atau log otomatis, sehingga pengguna tidak memiliki rincian dari proses pengujian. Untuk semua framework dilengkapi dengan dokumentasi dan dukungan komunitas yang memadai, sehingga memudahkan pengguna dalam mengatasi kendala teknis. Namun, tidak ada dari keempat framework ini yang memiliki fitur pelacakan bug secara resmi, sehingga untuk mengidentifikasi bug penguji harus memeriksa baris kode yang menyebabkan kesalahan secara manual.

BAB V

KESIMPULAN DAN SARAN

1.1 Kesimpulan

Kesimpulan pada penelitian ini adalah hasil analisis perbandingan untuk keempat framework adalah sebagai berikut: Pada parameter Technical Economic View, berdasarkan analisis menggunakan metode DIA, Github Copilot memiliki skor terbaik dalam aspek dukungan multi bahasa, biaya, dan dukungan komunitas dan dokumentasi. Sedangkan pada parameter Testing Process View, Randoop memiliki skor terbaik karena dalam keunggulannya dalam pengujian otomatis, terdapat laporan selama eksekusi tes dan laporan hasil eksekusi. Sementara itu, pada parameter Quality Standard View Github Copilot kembali menunjukkan performa terbaik dalam aspek keterbacaan (Readability) dan kemudahan penggunaan (Usability) memiliki skor terbaik dalam aspek hasil pengujian.

Hasil penelitian ini memperjelas bahwa keempat framework memiliki kelebihan dan keterbatasan yang perlu dipertimbangkan dalam memilih alat yang tepat untuk proyek pengembangan perangkat lunak.

1.2 Saran

Dari tahapan yang sudah dilakukan dalam penelitian ini, terdapat beberapa kendala dan batasan yang dihadapi. Berikut beberapa saran yang diberikan untuk penelitian selanjutnya.

1. Kompatibilitas Framework : Saat menggunakan framework EvoSuite terdapat kendala dengan kompatibilitas framework terhadap versi java yang digunakan penulis. Disarankan untuk penelitian berikutnya dapat mempertimbangkan penggunaan framework yang lebih kompatibel dengan versi java yang lebih baru.
2. Pengujian lebih lanjut : Mengingat bahwa penelitian ini menggunakan kode yang dibuat oleh penulis yaitu kelas Acara, Jadwal, dan ManajemenJadwal sebagai objek pengujian. Disarankan untuk penelitian berikutnya dapat menggunakan kode yang lebih kompleks untuk menguji kinerja framework secara lebih mendalam.
3. Menggunakan metode lain : Selain metode The Distance to The Ideal Alternative (DIA) yang digunakan dalam penelitian ini. Metode evaluasi lain dapat dipertimbangkan untuk penelitian selanjutnya, sebagai bahan untuk memberikan perspektif tambahan, sehingga hasil yang diperoleh lebih komprehensif.

DAFTAR PUSTAKA

- Abdulwareth, A. J., & Al-Shargabi, A. A. (2021). Toward a multi-criteria framework for selecting software testing tools. *IEEE Access*, 9, 158872–158891.
- Ayan, B., Abacıoğlu, S., & Basilio, M. P. (2023). A comprehensive review of the novel weighting methods for multi-criteria decision-making. *Information*, 14(5), 285.
- Brooke, J. (2013). SUS: a retrospective. *Journal of usability studies*, 8(2).
- Callan, J., & Petke, J. (2022). Multi-objective genetic improvement: A case study with evosuite. *International Symposium on Search Based Software Engineering*, 111–117.
- Chatterjee, S., Liu, C. L., Rowland, G., & Hogarth, T. (2024). The Impact of AI Tool on Engineering at ANZ Bank An Emperical Study on GitHub Copilot within Coporate Environment. *arXiv preprint arXiv:2402.05636*.
- Cohen, T., Gil, J., & Maman, I. (2006). JTL: the Java tools language. *ACM SIGPLAN Notices*, 41(10), 89–108.
- Demurjian Sr, S. A., & Shin, D. G. (t.t.). *The Java Programming Language/Environment and Risks/Benefits of Software Engineering with Java*.
- Dinella, E., Ryan, G., Mytkowicz, T., & Lahiri, S. K. (2022). Toga: A neural method for test oracle generation. *Proceedings of the 44th International Conference on Software Engineering*, 2130–2141.
- Fewster, M., & Graham, D. (1999). *Software test automation*. Addison-Wesley Reading.
- Fontes, A., Gay, G., Neto, F. G. de O., & Feldt, R. (2021). Automated support for unit test generation: a tutorial book chapter. *arXiv preprint arXiv:2110.13575*.
- Fraser, G., & Arcuri, A. (2011a). Evolutionary generation of whole test suites. *2011 11th International Conference on Quality Software*, 31–40.
- Fraser, G., & Arcuri, A. (2011b). Evosuite: automatic test suite generation for object-oriented software. *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 416–419.
- Hasibuan, A. N., & Dirgahayu, T. (2021). Pengujian dengan Unit Testing dan Test case pada Proyek Pengembangan Modul Manajemen Pengguna. *AUTOMATA*, 2(1).
- Jun, L. N., Suhery, C., & Ristian, U. (t.t.-a). SISTEM SELEKSI PENERIMAAN SISWA BARU SMA TARUNA BUMI KHATULISTIWA MENGGUNAKAN METODE THE DISTANCE TO THE IDEAL ALTERNATIVE. *Coding Jurnal Komputer dan Aplikasi*, 7(02).

- Jun, L. N., Suhery, C., & Ristian, U. (t.t.-b). SISTEM SELEKSI PENERIMAAN SISWA BARU SMA TARUNA BUMI KHATULISTIWA MENGGUNAKAN METODE THE DISTANCE TO THE IDEAL ALTERNATIVE. *Coding Jurnal Komputer dan Aplikasi*, 7(02).
- Kua, P. (1995). Unit Testing. URL: <https://www.thekua.com/publications/AppsUnitTesting.pdf>.
- Kua, P. (2019). Unit Testing. URL: <https://www.thekua.com/publications/AppsUnitTesting.pdf>.
- Lestari, Y. D., & Lubis, Y. F. A. (2022). Sistem Pendukung Keputusan Pemilihan Tempat Kegiatan Olahraga di Medan dengan Metode Distance To The Ideal Alternative (DIA). *Jurnal Komputer Teknologi Informasi dan Sistem Informasi (JUKTISI)*, 1(2), 56–64.
- Mawardi, M. (2019). Rambu-rambu penyusunan skala sikap model Likert untuk mengukur sikap siswa. *Scholaria: Jurnal Pendidikan Dan Kebudayaan*, 9(3), 292–304.
- Merina, C., Anggraini, N., & Hakiem, N. (2018). A Comparative analysis of test automation frameworks performance for functional testing in android-based applications using the distance to the ideal alternative method. *2018 Third International Conference on Informatics and Computing (ICIC)*, 1–6.
- Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing*. John Wiley & Sons.
- Pacheco, C., & Ernst, M. D. (2007). Randoop: feedback-directed random testing for Java. *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, 815–816.
- Paydar, S., & Azamnouri, A. (2019). An experimental study on flakiness and fragility of randoop regression test suites. *Fundamentals of Software Engineering: 8th International Conference, FSEN 2019, Tehran, Iran, May 1-3, 2019, Revised Selected Papers 8*, 111–126.
- Pressman, R. S. (2010). *Software Engineering: A Practitioner's Approach* (7 ed.). McGraw-Hill.
- Rahayu, D. K. P. (2020). *Unit Testing Pada Aplikasi Web Mobile (Studi Kasus Bisnis Jasa Laundry)*.
- Rijoly, M., Sapulette, N. T., Tomasouw, B. P., & Patty, D. (2023). Penerapan Metode The Distance To The Ideal Alternative (DIA) Untuk Menyelesaikan Pegawai Di PT. Fast Food Indonesia (KFC Indonesia) Kakialy Tanah Tinggi, Ambon. *Tensor: Pure and Applied Mathematics Journal*, 4(1), 13–20.
- Roslan, M. F., Rojas, J. M., & McMinn, P. (2022). An empirical comparison of EvoSuite and DSpot for improving developer-written test suites with respect to mutation score. *International Symposium on Search Based Software Engineering*, 19–34.
- Sauro, J., & Lewis, J. R. (2011). When designing usability questionnaires, does it hurt to be positive? *Proceedings of the SIGCHI conference on human factors in computing systems*, 2215–2224.
- Siroš, I., Singelée, D., & Preneel, B. (2024). GitHub Copilot: the perfect Code compLeeter? *arXiv preprint arXiv:2406.11326*.
- Tauriainen, J. (2023). *Correlation of Unit Test Code Coverage with Software Quality*.

- Tran, P. N., & Boukhatem, N. (2008). The distance to the ideal alternative (DiA) algorithm for interface selection in heterogeneous wireless networks. *Proceedings of the 6th ACM international symposium on Mobility management and wireless access*, 61–68.
- Wang, Y., Mäntylä, M. V, Demeyer, S., Wiklund, K., Eldh, S., & Kairi, T. (2020). Software Test Automation Maturity--A Survey of the State of the Practice. *arXiv preprint arXiv:2004.09210*.
- Zhu, H., Hall, P. A. V, & May, J. H. R. (1997). Software unit test coverage and adequacy. *Acm computing surveys (csur)*, 29(4), 366–427.

3. Full kode Kelas Acara

```
package org.example;

import org.joda.time.DateTime;
import org.joda.time.DateTimeZone;
import org.joda.time.format.DateTimeFormat;
import org.joda.time.format.DateTimeFormatter;

import java.io.Serializable;

public class Acara implements Serializable {
    private static final long serialVersionUID = 1L;

    private String judul;
    private String deskripsi;
    private DateTime waktuMulai;
    private DateTime waktuSelesai;
    private static final DateTimeFormatter format =
        DateTimeFormat.forPattern("dd/MM/yyyy HH:mm")
            .withZone(DateTimeZone.forID("Asia/Jakarta"));

    public Acara(String judul, String deskripsi, DateTime waktuMulai,
        DateTime waktuSelesai){
        if (waktuMulai == null){
            throw new IllegalArgumentException("waktu mulai tidak boleh
null");
        }
        if (waktuSelesai == null){
            throw new IllegalArgumentException("waktu selesai tidak boleh
null");
        }
        if(judul == null || deskripsi == null){
            throw new IllegalArgumentException("Judul dan deskripsi tidak
boleh kosong");
        }
        if (waktuMulai.isAfter(waktuSelesai)){ // Cek apakah waktu mulai
lebih awal dari waktu selesai //jika ingin menggunakan bug maka mengubah
isAfter menjadi isBefore
            throw new IllegalArgumentException("Waktu mulai harus diisi
sebelum waktu selesai");
        }
        this.judul = judul;
        this.deskripsi = deskripsi;
        this.waktuMulai = waktuMulai;
        this.waktuSelesai = waktuSelesai;
    }

    public String getJudul(){

        return judul;
    }

    public void setJudul(String judul){

        this.judul = judul;
    }

    public String getDeskripsi() {
```

```

        return deskripsi;
    }

    public void setDeskripsi(String deskripsi) {
        this.deskripsi = deskripsi;
    }

    public DateTime getWaktuMulai() {
        return waktuMulai;
    }

    public void setWaktuMulai(DateTime waktuMulai) {
        this.waktuMulai = waktuMulai;
    }

    public DateTime getWaktuSelesai() {
        return waktuSelesai;
    }

    public void setWaktuSelesai(DateTime waktuSelesai) {
        this.waktuSelesai = waktuSelesai;
    }

    // Cek apakah event ini bertabrakan dengan event lain
    public boolean isOverlapping(Acara other) {
        return !(this.waktuSelesai.isBefore(other.waktuMulai) ||
this.waktuMulai.isAfter(other.waktuSelesai));
    }

    @Override
    public String toString() {
        //      DateTimeFormatter format =
DateTimeFormat.forPattern("dd/MM/yyyy HH:mm").withLocale(new Locale("in",
"ID"));
        return "Acara{" +
            "Judul='" + judul + '\'' +
            ", Deskripsi='" + deskripsi + '\'' +
            ", waktuMulai=" + format.print(waktuMulai) +
            ", waktuSelesai=" + format.print(waktuSelesai) +
            '\'';
    }
}

```

4. Full kode kelas Jadwal

```

package org.example;

import org.joda.time.DateTime;
import org.joda.time.DateTimeZone;
import org.joda.time.Interval;
import org.joda.time.format.DateTimeFormat;
import org.joda.time.format.DateTimeFormatter;

import java.io.Serializable;

```

```

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.stream.Collectors;

public class Jadwal implements Serializable {
    private static final long serialVersionUID = 1L;
    private List<Acara> acaraS;
    public Jadwal(){
        this.acaraS = new ArrayList<>();
    }

    public void tambahAcara(Acara acara){

        if (acara == null){
            throw new IllegalArgumentException("Acara tidak boleh null");
        }
        if (acara.getWaktuMulai() == null || acara.getWaktuSelesai() ==
null){
            throw new IllegalArgumentException("waktu mulai atau selesai
tidak boleh null");
        }
        DateTimeZone zone = DateTimeZone.forID("Asia/Jakarta");
        DateTime waktuMulaiJakarta = new DateTime(acara.getWaktuMulai(),
zone);
        DateTime waktuSelesaiJakarta = new
DateTime(acara.getWaktuSelesai(), zone);
        Interval newEventInterval = new Interval(waktuMulaiJakarta,
waktuSelesaiJakarta);
        for (Acara existingEvent : acaraS) {
            DateTime existingWaktuMulaiJakarta = new
DateTime(existingEvent.getWaktuMulai(), zone);
            DateTime existingWaktuSelesaiJakarta = new
DateTime(existingEvent.getWaktuSelesai(), zone);
            Interval existingEventInterval = new
Interval(existingWaktuMulaiJakarta, existingWaktuSelesaiJakarta);

            //          Bug ini akan selalu melempar exception walau acara yang
dibuat tidak tumpang tindih
            if (!newEventInterval.overlaps(existingEventInterval)){
                throw new IllegalArgumentException("Acara baru bertepatan
dengan acara yang sudah ada");
            }

        }

        acaraS.add(acara);
    }

    public boolean hapusEvent(Acara acara) {
        return acaraS.remove(acara);
    }
    public List<Acara> acaraUntukTanggal(DateTime date) {
        return acaraS.stream()
            .filter(e ->
e.getWaktuMulai().toLocalDate().equals(date.toLocalDate()))
            .collect(Collectors.toList());
    }
}

```

```

        public List<Acara> cariEvents(String kataKunci){
            return acaraS.stream()
                // Bug: Mengembalikan acara yang tidak sesuai dengan
kata kunci
                .filter(e -> !e.getJudul().contains(kataKunci) &&
!e.getDeskripsi().contains(kataKunci))
                .collect(Collectors.toList());

//                .filter(e -> e.getJudul().contains(kataKunci) ||
e.getDeskripsi().contains(kataKunci))
//                .collect(Collectors.toList());
        }
        public boolean hapusAcaraBerdasarkanJudul(String judul) {
            if(judul == null){
                throw new IllegalArgumentException("Judul tidak boleh null");
            }
            for (Iterator<Acara> iterator = acaraS.iterator();
iterator.hasNext();) {
                Acara acara = iterator.next();
                if (acara.getJudul().equalsIgnoreCase(judul)) {
                    iterator.remove();
                    return true; // Event ditemukan dan dihapus
                }
            }
            return false; // Tidak ada event dengan judul tersebut
        }
        public void lihatJadwal(){
            if (acaraS.isEmpty()){
                System.out.println("Tidak ada event di jadwal");
                return;
            }
            acaraS.forEach(event -> System.out.println(event));
        }

        public List<Acara> getEvents() {
            return new ArrayList<>(acaraS);
        }
    }

```

5. Full kode kelas Jadwal

```

package org.example;

import org.joda.time.DateTime;
import org.joda.time.DateTimeZone;
import org.joda.time.format.DateTimeFormat;
import org.joda.time.format.DateTimeFormatter;

import java.io.*;
import java.util.List;
import java.util.Scanner;

//TIP To <b>Run</b> code, press <shortcut actionId="Run"/> or
// click the <icon src="AllIcons.Actions.Execute"/> icon in the gutter.
public class ManajemenJadwal {
    private Jadwal jadwal;
    private Scanner scanner;

```

```

        private static final DateTimeFormatter format =
DateTimeFormat.forPattern("dd/MM/yyyy HH:mm")
                .withZone(DateTimeZone.forID("Asia/Jakarta"));

    public ManajemenJadwal() {
        this(new Scanner(System.in), new Jadwal());
    }
    public ManajemenJadwal(Scanner scanner, Jadwal jadwal) {
        this.scanner = scanner;
        this.jadwal = new Jadwal();
        memuatJadwal();
        if (this.jadwal == null) {
            this.jadwal = new Jadwal();
        }
    }

    public void start() {
        boolean running = true;
        while (running) {
            System.out.println("\nManajemen Jadwal");
            System.out.println("1. Tambah Acara");
            System.out.println("2. Cari Acara");
            System.out.println("3. Hapus Acara");
            System.out.println("4. Lihat Acara");
            System.out.println("5. Keluar");
            System.out.print("Pilih: ");

            int pilihan = scanner.nextInt();
            scanner.nextLine();

            switch (pilihan) {
                case 1:
                    tambahEvent();
                    break;
                case 2:
                    cariEvent();
                    break;
                case 3:
                    hapusEvent();
                    break;
                case 4:
                    lihatEvent();
                    break;
                case 5:
                    running = false;
                    break;
                default:
                    System.out.println("Tidak valid, silahkan cobalagi");
            }
        }
    }

    private void lihatEvent() {
        List<Acara> acaras = jadwal.getEvents();
        if (acaras.isEmpty()) {
            System.out.println("Tidak ada acara di jadwal.");
        } else {

```

```

        for (Acara acara : acaras) {
            System.out.println(acara);
        }
    }

private void cariEvent() {
    System.out.println("Cari acara: ");
    System.out.println("1. Berdasarkan judul");
    System.out.println("2. Berdasarkan tanggal");
    System.out.println("3. Kembali ke menu utama");
    System.out.println("pilih: ");
    int pilihan = scanner.nextInt();
    scanner.nextLine();

    switch (pilihan) {
        case 1:
            System.out.print("Cari acara berdasarkan judul: ");
            String judul = scanner.nextLine();
            List<Acara> acaras = jadwal.cariEvents(judul);
            if (acaras.isEmpty()) {
                System.out.println("Tidak ada acara dengan judul
tersebut.");
            } else {
                for (Acara acara : acaras) {
                    System.out.println(acara);
                }
            }
            break;
        case 2:
            System.out.print("Cari acara untuk tanggal: ");
            String tanggal = scanner.nextLine();
            DateTimeFormatter formatter =
DateTimeFormat.forPattern("dd/MM/yyyy");

            try {
                DateTime date =
formatter.parseDateTime(tanggal).withZone(DateTimeZone.forID("Asia/Jakart
a"));
                List<Acara> acarasTanggal =
jadwal.acaraUntukTanggal(date);
                if (acarasTanggal.isEmpty()) {
                    System.out.println("Tidak ada acara pada tanggal
tersebut.");
                } else {
                    for (Acara acara : acarasTanggal) {
                        System.out.println(acara);
                    }
                }
            } catch (IllegalArgumentException e) {
                System.out.println("Format tanggal salah");
            }
            break;
        case 3:
            System.out.println("Kembali ke menu utama");
            return;
        default:
            System.out.println("Pilihan tidak valid");
    }
}

```

```

        break;
    }

}

private void hapusEvent() {
    System.out.println("Pilih metode penghapusan:");
    System.out.println("1. Berdasarkan judul");
    System.out.println("2. Berdasarkan tanggal");
    System.out.println("3. Kembali ke menu utama");
    System.out.print("Pilih: ");
    int pilihan = scanner.nextInt();
    scanner.nextLine();

    switch (pilihan) {
        case 1:
            System.out.print("Hapus acara: ");
            String judul = scanner.nextLine();

            boolean removed =
jadwal.hapusAcaraBerdasarkanJudul(judul);

            if (removed) {
                System.out.println("Acara '" + judul + "' berhasil
dihapus.");
            } else {
                System.out.println("Acara tidak ditemukan.");
            }
            break;
        case 2:
            System.out.println("Hapus acara untuk tanggal:");
            String tanggal = scanner.nextLine();
            DateTimeFormatter formatter =
DateTimeFormat.forPattern("dd/MM/yyyy");

            try {
                DateTime date = formatter.parseDateTime(tanggal);
                List<Acara> acaras = jadwal.acaraUntukTanggal(date);
                if (acaras.isEmpty()) {
                    System.out.println("Tidak ada acara pada tanggal
tersebut.");
                } else {
                    for (Acara acara : acaras) {
                        jadwal.hapusEvent(acara);
                        System.out.println("Acara" + acara.getJudul()
+ " berhasil dihapus");
                    }
                }
            } catch (IllegalArgumentException e) {
                System.out.println("Format tanggal salah");
            }
            break;
        case 3:
            System.out.println("Kembali ke menu utama");
            return;
        default:
            System.out.println("Pilihan tidak valid");
            break;
    }
}

```

```

    }
    simpanJadwal();
}

public void tambahEvent() {
    System.out.print("Masukan judul: ");
    String judul = scanner.nextLine();

    System.out.print("Masukan deskripsi: ");
    String deskripsi = scanner.nextLine();
    //bug format tanggal yang berbeda dengan format yang diinginkan
    DateTimeFormatter format = DateTimeFormat.forPattern("dd-MM-yyyy
HH:mm").withZone(DateTimeZone.forID("Asia/Jakarta"));
    DateTime waktuMulai = null;
    DateTime waktuSelesai = null;

    while (true) {
        try {
            System.out.print("Masukan waktu mulai dd/MM/yyyy: ");
            waktuMulai = format.parseDateTime(scanner.nextLine());
            System.out.print("Masukan waktu selesai dd/MM/yyyy: ");
            waktuSelesai = format.parseDateTime(scanner.nextLine());

            Acara acara = new Acara(judul, deskripsi, waktuMulai,
waktuSelesai);

            for (Acara existingEvent : jadwal.getEvents()) {
                if (acara.isOverlapping(existingEvent)) {
                    System.out.println("Acara bertabrakan dengan
acara yang sudah ada.");
                    return;
                }
            }

            jadwal.tambahAcara(acara);
            System.out.println("Acara berhasil ditambahkan.");
            simpanJadwal();
            break;
        } catch (IllegalArgumentException e) {
            System.out.println("Error: " + e.getMessage() + "
Silahkan masukan waktu yang benar");
        }
    }
}

private void simpanJadwal() {
    try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream("jadwal.bin"))) {
        oos.writeObject(jadwal);
    } catch (IOException e) {
        System.err.println("Gagal menyimpan jadwal" +
e.getMessage());
    }
}

private void memuatJadwal() {
    try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream("jadwal.bin"))) {

```

```

        jadwal = (Jadwal) ois.readObject();

    } catch (IOException | ClassNotFoundException e) {
        System.err.println("Gagal memuat jadwal" + e.getMessage());
    }
}

public static void main(String[] args) {
    ManajemenJadwal manager = new ManajemenJadwal();
    manager.start();
}

public Jadwal getJadwal() {
    return this.jadwal;
}
}

```

6. Perhitungan coverage

3. Kelas ManajemenJadwal:

- **Total Line : 154**

$$\text{Line Coverage} = \frac{84}{154} = 0.54 \times 100 = 54\%$$

- **Total method : 16**

$$\text{Method Coverage} = \frac{12}{16} = 0.75 \times 100 = 75\%$$

- **Total Branch : 48**

$$\text{branch Coverage} = \frac{22}{84} = 0.26 \times 100 = 26\%$$

Menghitung rata rata setiap jenis coverage

$$\text{Line Coverage} = \frac{100\% + 52\% + 54\%}{3} = \frac{206}{3} = 68\%$$

$$\text{Method Coverage} = \frac{81\% + 62\% + 75\%}{3} = \frac{281}{3} = 72\%$$

$$\text{Branch Coverage} = \frac{100\% + 15\% + 26\%}{3} = \frac{141}{3} = 47\%$$

Menghitung rata-rata keseluruhan Coverage

$$\frac{\text{Line coverage} + \text{Method coverage} + \text{branch coverage}}{3}$$

$$\text{Total Keseluruhan Coverage} = \frac{68 + 72 + 47}{3} = 62\%$$