Edusaintek: Jurnal Pendidikan, Sains dan Teknologi
Volume 12 Issue 1 2025 Pages 253 - 272
p-ISSN: 1858-005X e-ISSN: 2655-3392 DOI: https://doi.org/10.47668/edusaintek.v12i1.1489
website: https://journalstkippgrisitubondo.ac.id/index.php/EDUSAINTEK

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

Rifaldi Fakhrii[1*], Novi Setiani[2]
[1,2] Universitas Islam Indonesia, Indonesia
*Corresponding author: 20523023@students.uii.ac.id

Abstract: This Study aims to analyze and compare four unit test automation frameworks, namely EvoSuite, Randoop, Squaretest, and Github Copilot, using the Distance to Ideal Alternative (DIA) method. The four frameworks were evaluated based on three main parameters: Technical Economic View, Testing Process View, and Quality Standard View. The DIA method was chosen because it provides more consistent results by calculating the distance between positive and negative ideal alternative solutions. This research collects data through questionnaires distributed to developers with experience in the unit testing program. The result showed that Github Copilot occupies the best position in almost all aspects, especially in programming language flexibility and ease of use. However, it has a weakness in subscription fees. The results of this study are expected to guide software developers in choosing a test automation framework that suits their needs, both from a technical and economic perspective, as well as increase the literature related to the performance and effectiveness of unit testing tools in software development.

Keywords: Metode The Distance to Ideal Alternative, Unit Test.

Abstrak: Penelitian ini bertujuan untuk menganalisis perbandingan empat *framework* otomatisasi pengujian unit, yaitu *EvoSuite, Randoop, Squaretest,* dan *Github Copilot*, dengan menggunakan metode *The Distance to Ideal Alternative (DIA)*. Keempat *framework* dievaluasi berdasarkan tiga parameter utama: *Technical Economic View, Testing Process View, dan Quality Standard View*. Metode DIA dipilih karena keunggulannya dalam memberikan hasil yang lebih konsisten dengan perhitungan jarak terhadap alternatif solusi ideal positif dan negatif. Penelitian ini melibatkan 10 responden yang memiliki pengalaman dalam bidang pengujian unit (*unit testing*), sehingga memberikan perspektif yang relevan terhadap evaluasi *framework* yang diuji. Hasil penelitian menunjukkan bahwa *Github Copilot* menempati posisi terbaik dengan skor terendah (0.041) mengungguli *EvoSuite, Randoop,* dan *Squaretest* sebagai alternatif terbaik meskipun memiliki kelemahan pada biaya berlangganan sesuai informasi dari situs resminya yang menjadi salah satu pertimbangan bagi pengguna. Hasil pada penelitian ini diharapkan dapat memberikan panduan bagi pengembang perangkat lunak dalam memilih *framework* otomatisasi pengujian yang sesuai dengan kebutuhan, baik dari segi teknis maupun ekonomi, serta meningkatkan literatur terkait kinerja dan efektivitas alat pengujian unit dalam pengembangan perangkat lunak.

Kata kunci: Metode Jarak Menuju Alternatif Ideal, Uji Unit

PENDAHULUAN

Pengujian perangkat lunak adalah salah satu langkah penting dalam memastikan bahwa semua komponen sistem berfungsi dengan baik dan pengujian dilakukan untuk mendeteksi kesalahan (*error)* dan cacat (*bug*) selama eksekusi perangkat lunak. Kesalahan (*error*) dan cacat (*bug*) akan berdampak signifikan pada perangkat lunak yang

dapat menyebabkan perangkat lunak tidak beroperasi sesuai spesifikasi yang telah ditentukan (Hasibuan & Dirgahayu, 2021). Unit testing merupakan salah satu metode yang cukup populer yang digunakan dalam proses pengujian perangkat lunak yang berfokus pada pengujian bagian-bagian terkecil dari kode, seperti fungsi dan metode (Ariqah, 2023; Kua, 2019). Pengujian unit yang dirancang dengan baik tidak hanya membantu mendeteksi kesalahan pada tahap awal pengujian, tetapi juga berfungsi sebagai dokumentasi untuk memantau perkembangan perangkat lunak. Dokumentasi ini berperan dalam mengidentifikasi masalah selama proses pengembangan dan membantu mencegah munculnya kode-kode yang mengandung kesalahan (*error*) dan kecacatan (*bug*) (Dinella et al., 2022). Namun, dalam prosesnya yang masih dilakukan secara manual akan membutuhkan banyak waktu dan tenaga yang signifikan, terutama pada proyek perangkat lunak yang kompleks.

Otomatisasi unit *testing* hadir sebagai solusi untuk mengurangi beban kerja manual dalam proses pengujian. Dengan otomatisasi, pengujian dapat dilakukan lebih cepat dan lebih efisien, mengurangi biaya dan tenaga yang diperlukan dibandingkan pengujian manual (Fewster & Graham, 1999). Pengujian otomatis yang dirancang secara efektif dapat melengkapi pengujian manual serta meningkatkan kualitas keseluruhan dari rangkaian pengujian (Fontes et al., 2021). Selain itu, tingkat kesempurnaan dari suatu proses otomatisasi pengujian juga merupakan faktor penting dalam menentukan kualitas dan efektivitas keseluruhan pengujian (Wang et al., 2020).

Berbagai *framework* otomatisasi pengujian unit, seperti *JUnit, TestNG, Mocha, Jasmine, QUnit, Evosuite, SquareTest, Randoop,* dan *Github Copilot*, telah dikembangkan untuk mempermudah dan meningkatkan efisiensi proses pengujian unit. *EvoSuite* adalah alat yang secara otomatis membuat rangkaian pengujian untuk mencapai jangkauan kode yang luas (Fraser & Arcuri, 2011). Sementara *Randoop* menggunakan algoritma umpan balik untuk menghasilkan pengujian unit di *Java* (Pacheco & Ernst, 2007). *Squaretest* merupakan *plugin* di *Intellij* IDEA yang membantu dalam pembuatan unit tes secara otomatis untuk kelas *java*. *Github Copilot* adalah *plugin IDE (Integrate Development Environment)* yang menggunakan teknologi AI untuk membantu menulis kode. *Github copilot* diluncurkan pada bulan Juni 2021 oleh *Github*. Meskipun berbagai *framework* ini memiliki pendekatan yang beragam, belum banyak penelitian yang mengeksplorasi perbandingan efektivitas *framework* dengan menggunakan metode

pengambilan keputusan multi kriteria. Pada penelitian ini akan menggunakan keempat *framework* yaitu *Evosuite, Randoop, Github copilot,* dan *Squaretest.*

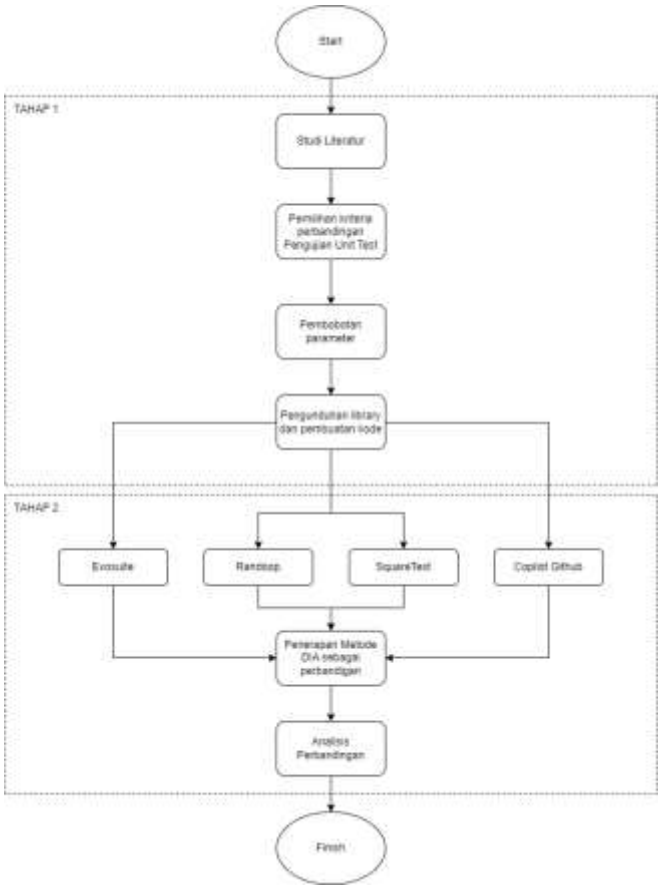
Keempat *framework* tersebut akan dibandingkan untuk mengukur kinerja setiap *framework* dalam pengujian unit berdasarkan parameter *Technical and Economic View, Testing process View, dan Quality Standards View* (Abdulwareth & Al-Shargabi, 2021). Metode yang digunakan adalah *Metode The Distance to Ideal Alternative (DIA)*, metode ini 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 DIA menggunakan jarak manhattan untuk menghitung jarak antara setiap alternatif dengan atribut ideal positif dan atribut ideal negatif (Lestari & Lubis, 2022; Tran & Boukhatem, 2008). Metode DIA dipilih karena memiliki keunggulan dalam mengukur kinerja relatif dari berbagai alternatif keputusan dengan cara matematis yang sederhana, mudah dimengerti, dan efisiensi dalam komputasi (Jun et al., n.d.,2019).

Penelitian ini diharapkan dapat memberikan panduan bagi pengembang perangkat lunak dalam memilih *framework* otomatis yang paling sesuai untuk pengujian unit serta penelitian ini dapat memperkaya literatur tentang kinerja dari setiap *framework* pengujian unit otomatis.

METODE

Pada Penelitian ini menggunakan pendekatan kuantitatif dengan mengumpulkan data dari responden melalui penyebaran kuesioner. Penyebaran kuesioner merupakan bagian dari penelitian survei, menurut Agung Widhi Kurniawan (2016) penelitian survei bertujuan untuk memahami dan menganalisis data yang dikumpulkan dari sebuah sampel representatif sehingga ditemukan kejadian-kejadian relatif, distribusi, dan interaksi antar variabel, penelitian survei dapat dilakukan dalam skala populasi yang luas maupun kecil. Dalam penelitian kuantitatif data penelitiannya berupa angka-angka dan analisis menggunakan statistik (Muharam et al., 2024). Oleh karena itu, data yang didapatkan dari kuesioner yang disebarkan digunakan untuk menentukan bobot pada parameter dan sub parameter yang telah ditentukan.

Pada Gambar 1 merupakan alur proses yang dilakukan pada penelitian ini. Tahap pertama pada alur proses adalah melakukan penentuan kriteria serta pembobotan parameter didasarkan pada studi literatur yang ditelaah, kemudian dilakukan pengunduhan *library* yang dibutuhkan dalam proyek dan pembuatan kode. Pada tahap kedua adalah keempat *framework* dijalankan untuk melakukan pengujian terhadap kode yang dibuat. Selanjutnya, dilakukan perbandingan kinerja setiap *framework* dengan menggunakan metode *The Distance to Ideal Alternatif*.



Gambar 1. Alur proses

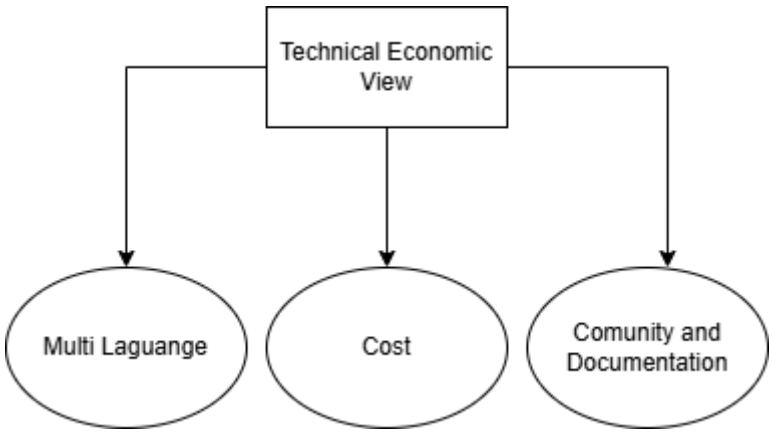
Studi Literatur

Studi literatur ini dilakukan dengan tujuan untuk mengumpulkan bahan, referensi, informasi, dan data untuk penelitian ini. Analisis komprehensif terhadap literatur relevan memungkinkan penentuan parameter dan metodologi yang sesuai dengan topik penelitian. Proses ini melibatkan evaluasi terhadap berbagai sumber, termasuk jurnal dan artikel yang berkaitan dengan topik penelitian ini.

Pemilihan Parameter Perbandingan Pengujian Unit Test

Pemilihan parameter perbandingan menjadi prioritas dalam penelitian ini. Studi literatur yang ditelaah digunakan untuk menetapkan parameter yang relevan. 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 Penelitian terdahulu berfokus pada pemilihan parameter pengujian dengan menyajikan tiga parameter utama yang masing-masing mencakup beberapa sub parameter.

Merujuk pada studi literatur yang ditulis oleh Abdulwareth & Al-Shargabi (2021) Menyatakan bahwa terdapat tiga parameter utama dalam pemilihan *test automation framework*. Parameter *Technical Economic View* yaitu Kriteria pertama yang dipertimbangkan oleh pengembang/penguji untuk mengklasifikasi alat untuk *test automation framework* berdasarkan aspek teknis dan ekonomi.

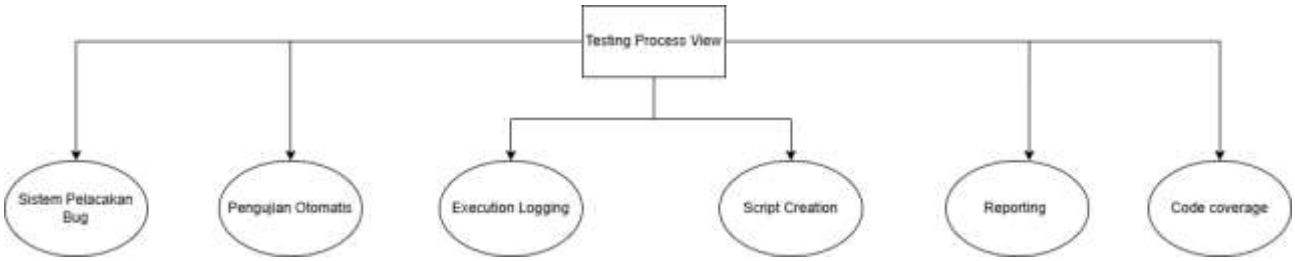


Gambar 2. Parameter *Technical Economic View*

Adapun sub parameter *Technical Economic View* adalah sebagai berikut: (1) *Multi Laguange* merupakan *Framework* pengujian untuk mendukung berbagai bahasa pemrograman. Ini memungkinkan alat atau *framework* dapat menulis dan menjalankan tanpa batasan satu bahasa tertentu (Pelivani & Cico, 2021); (2) *Cost* merupakan Biaya yang diperlukan pada *framework*. Apakah biaya tersebut gratis atau berbayar (Wang et al., 2022); (3) *Community Support and Documentation* mengacu pada bantuan yang tersedia dari komunitas pengguna serta ada panduan resmi yang disediakan (Wang et al., 2022).

Parameter kedua adalah *Testing Process View* yaitu merupakan *framework* komprehensif untuk memahami dan mengklasifikasikan aktivitas pengujian perangkat

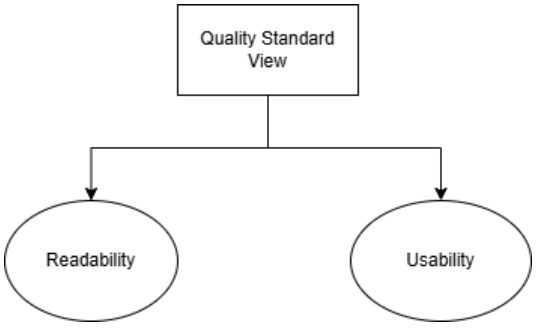
lunak. Ini digunakan sebagai dasar untuk memilih alat pengujian yang sesuai, merancang rencana pengujian, dan mengevaluasi hasilnya.



Gambar 3. *Parameter Testing Process View*

Adapun sub-parameter *Testing Process View* adalah sebagai berikut: (1) Sistem Pelacakan *Bug*: Apakah terdapat mekanisme untuk mencatat, melacak, dan mengelola *bug* pada alat (Wang et al., 2022); (2) Pengujian Otomatis: *Framework* dapat menjalankan test secara otomatis tanpa memerlukan intervensi manual (Wang et al., 2022); (3) *Execution Logging*: Apakah terdapat proses pencatatan untuk semua aktivitas dan hasil selama eksekusi pengujian otomatis (Wang et al., 2022); (4) *Script Creation* : seberapa lama waktu yang dibutuhkan untuk menulis skrip pada *framework* (Pelivani & Cico, 2021); (5) *Reporting* : *Framework* menyedikan laporan yang merangkum hasil dari pengujian yang dilakukan (Wang et al., 2022); (6) *Code Coverage* : *Framework* memberikan rincian *code coverage* untuk setiap blok kode atau metode dalam kode (Daka et al., 2015).

Parameter ketiga Parameter ketiga adalah *Quality Standard View* merupakan kriteria yang penting pada *test automation framework* untuk mengukur kualitas perangkat lunak mencakup beberapa aspek umum seperti efisiensi, maintainabilitas, keamanan, dan Fungsionalitas.



Gambar 4. Parameter *Quality Standard View*

Adapun sub-parameter *Quality Standards View* sebagai berikut: (1) *Readability* : Seberapa mudah kode dapat dibaca dan dipahami oleh manusia, khususnya oleh pengembang (Daka et al., 2015); (2) *Usability* : Seberapa mudah dan efisien dalam

menggunakan framework untuk membuat, menjalankan, dan memelihara pengujian unit. (Daka et al., 2015).

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 framewrok mengunakan metode *The Distance to Ideal Alternative (DIA)*. Prinsip-prinsip *Multiple Atribute 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.

Demografi Responden

Penelitian ini melibatkan responden dengan jumlah 10 responden, Kuesioner disebarkan kepada programmer yang memiliki pengalaman dibidang unit testing. Setiap responden memiliki latar belakang pendidikan dan pengalaman kerja yang variatif. Mayoritas responden memiliki latar belakang pendidikan terakhir sebagai sarjana S1 yaitu sebanyak 80% dan 20% responden memiliki latar belakang pendidikan terakhir sebagai sarjana s2. 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.

Bobot untuk sub-parameter dihitung berdasarkan jawaban terhadap kuesioner yang telah dibuat untuk mengukur tingkat kepentingan relatif sub-parameter dalam tiga parameter utama yaitu *Technical Economic View, Testing Process View, dan Quality View*. Responden diminta memberikan peringkat pada parameter utama dan sub-parameter dari yang paling penting hingga yang paling tidak penting. Parameter utama terdiri dari *Tehcnical Economic View*, *Testing Process View*, dan *Quality Standards View* dengan sub-parameter yang berbeda. Berdasarkan jawaban tersebut, bobot ditentukan dengan mengadaptasi skema penilaian yang sesuai dengan jumlah sub-parameter masing-masing 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

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 1 untuk parameter dan persamaan 2 untuk parameter, sehingga dari persamaan tersebut totalnya menjadi satu untuk setiap parameter dan sub parameter.

$$Bobot\ Sub\ parameter = \frac{Total\ poin\ sub\ parameter}{Total\ poin\ dari\ semua\ sub\ parameter} \quad (1)$$

$$Bobot\ Parameter = \frac{Total\ poin\ Parameter}{Total\ poin\ dari\ semua\ Parameter} \quad (2)$$

Bobot Nilai Parameter dan Sub Parameter

Tabel 1. Bobot nilai parameter dan sub parameter

Parameter	Bobot	Sub Parameter	Bobot Sub parameter
Technical Economic View	0.26	Multi laguange	0.27
		Cost	0.34
		Community Support and Documentation	0.39
Testing Process View	0.39	Sistem Pelacakan Bug	0.19
		Pengujian Otomatis	0.22
		Execuition Logging	0.12
		Script Creation	0.15
		Reporting	0.15
		Code Coverage	0.17
Quality Standard View	0.35	Readability	0.48
		Usability	0.52

Pada Tabel 1 menampilkan hasil kuesioner yang disebarkan kepada responden, yang mencakup bobot untuk parameter dan sub parameter. Perhitungan bobot untuk setiap parameter dan sub parameter menggunakan skema penilaian bobot yang telah ditentukan, kemudian dinormalisasikan. Proses normalisasi pada sub parameter dengan menghitung total poin yang diperolah sub parameter dibagi dengan total poin yang diperoleh semua sub parameter di masing-masing parameter utama. Sementara itu, bobot untuk setiap parameter utama dihitung dengan membagi total poin yang diperoleh oleh parameter tersebut dengan total poin dari semua parameter.

Metode *The Distance To Ideal Alternatif*

Metode *The Distance to The Ideal Alternatif* (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 mahhatan distance untuk mengukur jarak antara nilai atribut dan nilai ideal positif dan negatif dari setiap atribut. Penetapan *Alternatif Ideal Positif* (PIA) dengan nilai minimum Dj+, dan nilai maksimal untuk Dj- serta rumus yang diurutkan berdasarkan nilai Ri sebagai penentu peringkat alternatif (Tran & Boukhatem, 2008b). Langkah-langkah yang dilakukan dalam penyelesaian masalah menggunakan metode DIA adalah sebagai berikut (Jun et al., n.d.,2019):

Menentukan matriks keputusan

Bobot yang telah ditetapkan

$$X = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} \quad (3)$$

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:

$$r_{ij} = \frac{x_{ij}}{\sqrt{\sum_{i=1}^{m} x^2{}_{ij}}} \quad (4)$$

Pembobotan pada matriks yang telah dinormalisasi

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

$$v_{ij} = r_{ij} w_i \quad (5)$$

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

$$A^+ = \max v_{ij} = [a_1^+, a_2^+, \dots, a_n^+] \quad (6)$$

$$A^- = \min v_{ii} = [a_1^-, a_2^-, \dots, a_n^-] \quad (7)$$

Hitung jarak mahattan untuk atribut positif dan negatif

Jarak manhattan atribut positif dinotasikan dengan D_i^+ dan jarak manhattan atribut negatif dinotasikan dengan D_i^-. Lalu a_i^+adalah nilai dari setiap solusi ideal positif, dan a_i^- adalah nilai dari setiap solusi ideal negatif. Persamaan ini dapat dilihat sebagai berikut :

$$D_i^+ = \sum_{i=1}^{m} \left[v_{ij} - a_i^+ \right] \quad (8)$$

$$D_i^- = \sum_{i=1}^{m} \left[v_{ij} - a_i^- \right] \quad (9)$$

Menentukan positif ideal alternatif (PIA)

PIA minimal dinotasikan dengan min (D_i^+), nilai ini diperoleh dari pencarian nilai terkecil pada setiap baris $Di+$. 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:

$$\text{PIA} = \left(min(D_i^+),\ max(D_i^-) \right) \quad (10)$$

Melakukan identifikasi peringkat

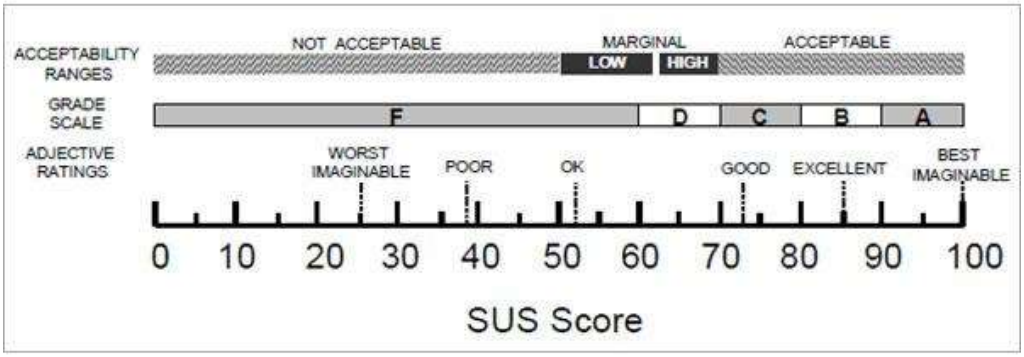
Peringkat dapat ditentukan dengan membandingkan nilai identifikasi peringkat (R_i) yang diperoleh dari persamaan sebagai berikut:

$$R_i = \sqrt{\left(D_i^+ - min\ (D_i^+)\right)^2 + \left(D_i^- - max\ (D_i^-)\right)^2} \quad (11)$$

System Usability Scale (SUS)

System Usability Scale (SUS) digunakan untuk mengukur *usability* (Nugroho & Cahyono, 2024). Metode SUS adalah metode pengujian yang cepat, terjangkau, dan

efisien untuk mengumpulkan data yang valid serta menghasilkan skor yang jelas dan akurat, dengan respons yang diukur menggunakan skala 1 hingga 5. Namun untuk mempermudah perhitungan skor, sebuah penyesuaian dilakukan. Menurut Brooke (2013) untuk pertanyaan yang diformulasikan secara positif skor yang diperoleh adalah nilai dari respons dikurangi 1. Sementara, untuk pertanyaan yang diformulasikan secara negatif skor dihitung dengan mengurangi nilai respons dari 5, lalu dijumlahkan dan hasilnya dikalikan dengan 2.5 untuk mendapatkan skor total SUS. Pada Gambar 7 merupakan rentang SUS skor, Skor SUS sebesar 68 menandakan skor yang baik karena skor ini berada di persentil ke-50 (Sauro & Lewis, 2011).



Gambar 5. Rentang SUS *score*

Pembuatan Kode dan Penambahan *Library*

Pada bagian ini, Sebuah kode aplikasi bernama *scheduleapp* dibuat yang dijadikan sebagai bahan untuk melakukan pengujian terhadap *framework* otomatis. *Scheduleapp* berisi tiga kelas utama yaitu Acara, Jadwal, dan manajemen jadwal. Selain itu, Pengunduhan *library Joda-Time* digunakan untuk pengelolaan waktu yang kompleks. Kode ini akan diuji menggunakan *framework* pengujian unit yang telah ditentukan, untuk mengukur kinerja setiap *faramework* dan memastikan bahwa semua *framework* berjalan sesuai dengan spesifikasi yang ditentukan.

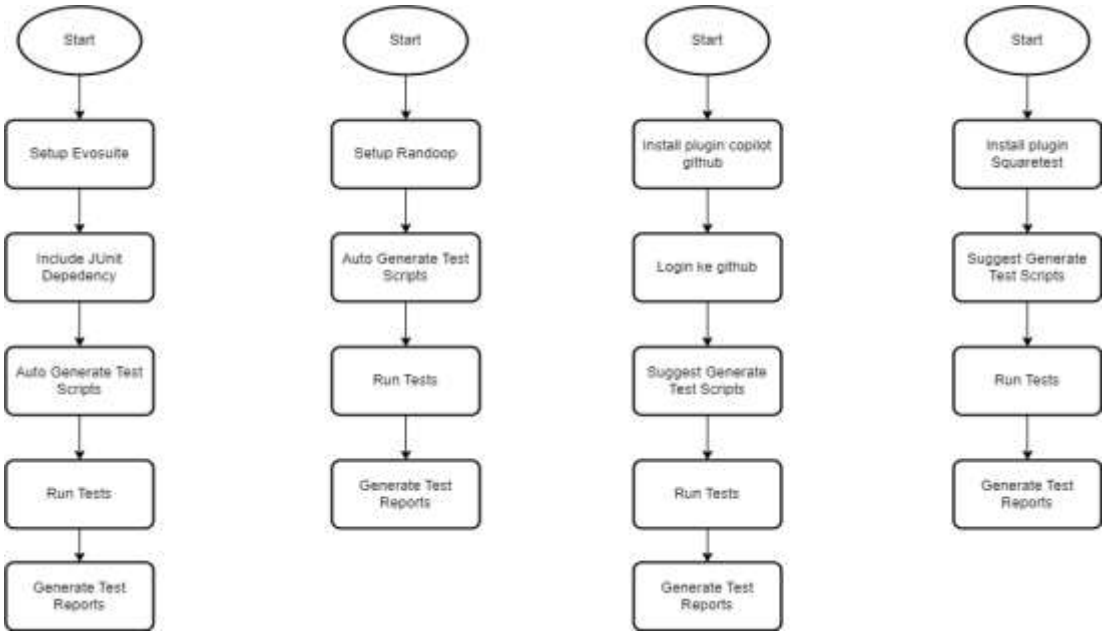
HASIL DAN PEMBAHASAN

Hasil

Framework Otomatis Unit test

Pada penelitian ini melakukann pengujian terhadap empat *framework* pengujian otomatis untuk unit testing yaitu *Evosuite, Randoop, Squaretest*, dan *Github Copilot*. Keempat *framework* tersebut menghasilkan skrip pengujian otomatis. Untuk *Evosuite, Randoop*, dan *Squaretest* secara khusus berfokus untuk melakukan pengujian otomatis

pada *test case* berbasis *Java*. Sementara itu, *Github copilot* mendukung berbagai bahasa pemrograman, namun dalam penelitian ini penggunaanya disesuaikan untuk bahasa *Java* guna memudahkan pengujia. Alur dalam menggunakan keempat *framework* berbeda-beda, dapat dilihat pada Gambar 8.



Gambar 6. Alur Pengujian

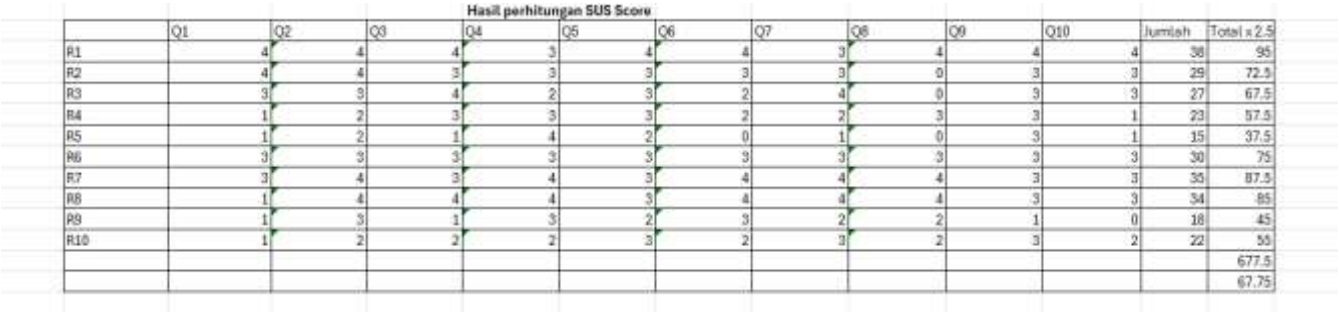
Hasil pengumpulan Sub Parameter *Usability*

Pengukuran SUS dilakukan berdasarkan jawaban responden dari 10 pertanyaan mengenai pengalaman penggunaan *framework*. Data jawaban pada gambar 7 kemudian diolah untuk mendapatkan skor SUS pada gambar 8. Sebagai contoh hasil perhitungan pada *framework EvoSuite* menunjukkan rata-rata skor SUS adalah 61,13.

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

Gambar 7. Data jawaban responden pada *framework EvoSuite*



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

Gambar 8. Hasil perhitungan skor SUS pada *framework EvoSuite*

Penerapan *Metode The Distance to Ideal Alternatif*

Setelah melakukan pengujian kepada empat *framework* otomatis untuk unit testing langkah berikutnya adalah mengumpulkan data, menganalisis hasil pengujian, dan menerapkan metode *The Distance to The Ideal Alternative* (DIA). Metode ini digunakan untuk mengevaluasi kinerja serta membandingkan keempat *framework* otomatis yang telah diuji. Hasil dari evaluasi akan memberikan gambaran *framework* yang paling optimal dari segi kinerja untuk pengujian unit testing.

Pengumpulan Data

Pada Tabel 2 merupakan data hasil pengujian otomatis yang telah dilakukan pada keempat *test automation framework.*

Tabel 2. Data hasil dari setiap *framework*

Nama *Framework*	***Multi Languange***	***Scrip Creation***	***Cost***	***Community Support and Documentation***
EvoSuite	1 bahasa	03:55 menit	Gratis	ada
Randoop	1 bahasa	01:40 menit	Gratis	ada
Squaretest	1 bahasa	00:05 detik	Berbayar	ada
Github Copilot	8 bahasa	02:00 menit	Berbayar	ada

Menentukan matriks keputusan

Matriks keputusan dihitung untuk tiga parameter utama yaitu *Technical Economic View, Testin Process View, dan Quality Standard View*. Sebagai contoh tabel 3 menunjukkan matriks keputusan untuk parameter *Technical Economic View*. Metode yang sama diterapkan untuk *Testing Process View* dan *Quality Standard View*.

Tabel 3. Matriks keputusan *technical economic view*

Alternatif	**Bobot Sub Parameter**		
	Multi-Language	***Cost***	***Community Support and Documentation***
EvoSuite	0	1	1
Randoop	0	1	1
Squaretest	1	0	1
Github Copilot	1	0	1

Normalisasi Matriks Keputusan

Proses normalisasi dilakukan menggunakan persamaan (4), dengan nilai yang digunakan berasal dari matriks keputusan. Sebagai contoh pada tabel 6 merupakan hasil normalisasi matriks keputusan pada sub parameter *Technical Economic View*. Metode yang sama diterapkan untuk *Testing Process View* dan *Quality Standard View*.

Tabel 6. Normalisasi matriks keputusan *Technical economic view*

Alternatif	**Bobot Sub Parameter**		
	Multi Language	***Cost***	***Community Support and Documentation***
EvoSuite	0	0.71	0.50
Randoop	0	0.71	0.50
Squaretest	0	0	0.50
Github Copilot	1	0	0.50

Pembobotan matriks yang telah dinormalisasi

Berdasarkan hasil normalisasi matriks keputusan maka nilai tersebut dimasukkan ke dalam pembobotan matriks dengan menggunakan persamaan (5). Sebagai contoh pada tabel 7 merupakan hasil dari sub parameter *technical economic view* yang telah dinormalisasi.

Tabel 7. Pembobotan matriks keputusan *technical economic view*

Alternatif	***Technical Economic View***		
	Multi-Language	***Cost***	***Community Support and Documentation***
EvoSuite	0	0.241	0.195
Randoop	0	0.241	0.195
Squaretest	0	0	0.195
Github Copilot	0.270	0	0.195

Menentukan solusi ideal positif dan negatif

Pada tahap ini, hasil dari pembobotan pada matriks yang telah di normalisasi, selanjutnya menentukan solusi ideal positif dan negatif. Pada tabel 8 dan tabel 9 merupakan contoh solusi ideal positif dari parameter *technical economic view*.

Tabel 8. Solusi ideal positif technical economic view

Solusi Ideal negative	**Sub parameter**	$A^+ = maxV_{ii} = [V_1^+ V_2^+ \cdots V_3^+]$
V1+	*Multi Laguange*	0.270
V2+	*Cost*	0.241
*V*3+	*Community support and documentation*	0.195

Tabel 9. Solusi ideal negatif *technical economic view*

Solusi Ideal negatif	Sub parameter	$A^- = maxV_{ii} = [V_1^- V_2^- \cdots V_3^-]$
V_1^-	Multi-Language	0
V_2^-	Cost	0
V_3^-	Community support and documentation	0.195

Menghitung jarak Manhattan

Berdasarkan hasil solusi ideal positif dan solusi ideal negatif dari setiap sub parameter. Selanjutnya adalah menghitung jarak manhattan max dan manhattan min pada setiap sub parameter menggunakan persamaan (8), dan persamaan (9). Pada tabel 10 dan tabel 11 merupakan contoh hasil dari parameter *technical economic view*.

Tabel 10. Jarak Manhattan *max technical economic view*

Alternatif	Jarak Manhattan Maximum	Nilai
EvoSuite	D_1^+	0.270
Randoop	D_2^+	0.270
Square test	D_3^+	0.511
Github Copilot	D_4^+	0.241

Tabel 11. Jarak manhattan min technical economic view

Alternatif	Jarak Manhattan Minimum	Nilai
EvoSuite	D_1^-	0.241
Randoop	D_2^-	0.241
Square test	D_3^-	0
Github Copilot	D_4^-	0.270

Menentukan Positif Ideal Alternatif (PIA)

Untuk Parameter *Technical Econoic View* nilai PIA adalah

$$0.241; 0.270$$

Untuk Parameter *Testing Process View* nilai PIA adalah

$$0.075; 0.226$$

Untuk Parameter *Quality View* nilai PIA adalah

$$0; 0.372$$

Identifikasi Peringkat

Nilai R_i yang lebih rendah menunjukkan bahwa alternatif tersebut lebih diutamakan dalam pemeringkatan. Berdasarkan persamaan yang telah dijelaskan, penulis akan menentukan peringkat dengan menganalisis setiap parameter. Hasil akhirnya adalah

penentuan peringkat berdasarkan parameter, yang disajikan dalam Tabel 12, Tabel 13, dan Tabel 14.

Tabel 12. Hasil peringkat *technical economic view*

Framework	R_i	**Peringkat**
EvoSuite	0.041	2
Randoop	0.041	2
Square test	0.382	3
Github Copilot	0	1

Tabel 13. Hasil peringkat *testing process view*

Framework	R_i	**Peringkat**
EvoSuite	0.009	2
Randoop	0	1
Square test	0.136	4
Github Copilot	0.092	3

Tabel 14. Hasil peringkat *quality standard view*

Framework	R_i	**Peringkat**
EvoSuite	0.272	3
Randoop	0.526	4
Squaretest	0.131	2
Github Copilot	0	1

Analisis Perbandingan

Pada tahap ini, penulis akan melakukan analisis perbandingan. Setelah mendapatkan nilai *Ri* dari tiga parameter yang telah ditentukan, penulis akan melanjutkan dengan analisis lebih lanjut untuk membandingkan nilai-nilai dari parameter tersebut.

Menurut Ayan et al. (2023), penggunaan *Weighted Scoring* Model menyederhanakan dan menguantifikasi proses pengambilan keputusan yang kompleks dengan mempertimbangkan berbagai faktor penting. Hal ini sangat berguna untuk mencapai hasil akhir yang lebih valid, sehingga memungkinkan perbandingan kinerja keseluruhan dari kerangka kerja otomatisasi pengujian dapat diukur. Berikut ini adalah langkah-langkah dalam *Weighted Scoring* Model.

Peringkat untuk setiap parameter

Tabel 15. Hasil peringkat dari setiap parameter dan bobot

Parameter	***Evosuite***	***Randoop***	***Squaretest***	***Github Copilot***	**Bobot**
Technical Economic View	0.041	0.041	0.382	0	0.26

Testing Process View	0.009	0	0.136	0.092	0.39
Quality Standards View	0.272	0.526	0.131	0	0.35

Menghitung skor tertimbang (*weighted Scoring*)

Tabel 16. Hasil perhitungan skor tertimbang

Framework	**Total Skor**	**Peringkat**
EvoSuite	0.109	2
Randoop	0.194	3
Squaretest	0.198	4
Github Copilot	0.036	1

Berdasarkan perhitungan di atas, alternatif terbaik adalah *Github Copilot* dengan total skor terkecil (0.036). *Github Copilot* menempati peringkat pertama, diikuti oleh *Randoop* di peringkat kedua, *EvoSuite* di peringkat ketiga, dan diikuti oleh *Square test* pada peringkat keempat.

Pembahasan

Dalam penelitian ini, keempat *framework* diuji berdasarkan parameter yang telah ditentukan yaitu parameter *Technical Economic View, Testing Process View,* dan *Quality Standard View*. Dari sisi *Technical Economic View EvoSuite* hanya mendukung satu bahasa pemrograman saja yaitu java, yang membuatnya fokus dan spesifik pada proyek yang berbasis java, Selain itu *EvoSuite* juga memiliki keunggulan dalam hal kemudahan setup dan tidak memerlukan biaya untuk digunakan. Lalu, Dukungan Dokumentasi dan komunitasnya lengkap sehingga memudahkan pengguna dalam mencari solusi jika menemui masalah. *Randoop* memiliki hal yang serupa dengan *EvoSuite* namun proses setup *Randoop* lebih rumit dibandingkan *EvoSuite,* yang membuat penulis perlu memakan waktu dalam instalasi dan konfigurasi. Pada *Squaretest*, memiliki kemudahan karena *framework* ini tidak memerlukan setup, *Squaretest* hanya perlu diinstal pluginnya sedangkan untuk komunitas dan dokumentasi sangat lengkap pada situs resminya namun dalam aspek ekonomi *framework* ini berbayar. *Github Copilot* unggul dalam aspek kemudahan penggunaan, dokumentasi dan comunity yang lengkap, dan dapat menggunakan banyak bahasa pemrograman hanya saja *Github Copilot* perlu membayar untuk mendapatkan fitur tambahan. Dari sisi *Testing Process* View Keempat *framework* mampu menghasilkan pengujian otomatis yang baik dan memudahkan, namun keempat

framework memang tidak memiliki fitur pelacakan bug. Untuk waktu yang dibutuhkan dalam membuat tes otomatis *Squaretest* dan *Github Copilot* lebih cepat dalam membuat tes otomatis dibandingkan *EvoSuite, Randoop,* dan *Github Copilot*. Namun, *Squaretest* dan *Github Copilot* tidak memiliki log pencacatan selama pengujian berlangsung serta *reporting* hasil pengujian. *Randoop* dan *Evosuite* memiliki log pencacatan selama pengujian berlangsung dan reporting setelah pengujian berhasil dijalankan. *Code coverage* pada ketiga *framework* yaitu *EvoSuite, SquareTest,* dan *Github Copilot* memiliki rata-rata *coverage* yang cukup dibandingkan *Randoop* yang masih kurang dari skala *coverage* yang baik. Dari sisi *Quality View, EvoSuite* dan *Randoop* yang menghasilkan skrip pengujian secara otomatis, cenderung menghasilkan kode pengujian yang tidak sepenuhnya *readable*. *Squaretest* dan *Github Copilot* cenderung menghasilkan kode yang sederhana dan struktur yang baik sehingga dapat dipahami dengan baik. *Evosuite* dan *Randoop* juga menawarkan kemudahan dalam otomatisasi pengujian, namun setup awal dan proses konfigurasi bisa lebih menantang, khususnya untuk *Randoop* yang memerlukan lebih banyak konfigurasi dibandingkan dengan *Evosuite*. *Squaretest* menonjol dalam hal kemudahan penggunaan karena hanya memerlukan penginstalan *plugin t*anpa setup rumit. *Github Copilot* menonjol di segala aspek.

Hasil dari penelitian ini menunjukkan bahwa Github Copilot unggul dalam segala aspek hanya saja kekurangannya hanya pada cost yang perlu dikeluarkan. *Randoop* menawarkan keseimbangan yang baik tapi memerlukan setup yang lebih kompleks dan rumit. Evosuite memiliki kemudahan dalam setup, dan pengujian otomatis namun dalam membuat tes scrip masih membutuhkan waktu yang cukup lama. Squaretest menawarkan banyak kemudahan namun masih perlu ditingkatkan lagi dalam membuat tes dan dalam aspek ekonomi bisa dikurangi biaya untuk

SIMPULAN

Berdasarkan penelitian yang telah dilakukan, empat *framework* otomatisasi pengujian unit, yaitu *Evosuite, Randoop, Squaretest,* dan *Github Copilot*, dibandingkan menggunakan metode *The Distance to Ideal Alternative* (DIA) berdasarkan tiga parameter utama yaitu *Technical Economic View, Testing Process View,* dan *Quality Standard View.*

Github Copilot menunjukkan performa terbaik dalam hampir semua aspek, terutama dalam hal fleksibilitas bahasa pemrograman dan kemudahan penggunaan,

meskipun memiliki kelemahan dari sisi biaya berlangganan untuk fitur-fitur tambahan. *Squaretest* unggul dalam kemudahan setup dan waktu pembuatan skrip pengujian yang sangat cepat, namun aspek ekonominya menjadi kelemahan karena *framework* ini berbayar. *EvoSuite* dan *Randoop* menawarkan solusi gratis dengan dukungan dokumentasi dan komunitas yang baik, namun kedua *framework* ini memerlukan waktu lebih lama dalam setup dan proses pengujian otomatis, khususnya *Randoop* yang memiliki proses konfigurasi lebih kompleks dibandingkan *EvoSuite*.

Secara keseluruhan, hasil penelitian ini menunjukkan kelebihan dan kekurangan dari setiap *framework*. Penelitian ini dapat menjadi panduan bagi pengembang dalam memilih *framework* otomatisasi pengujian unit yang paling sesuai dengan kebutuhan dan prioritas serta ekonomis dari proyek yang dikerjakan.

DAFTAR RUJUKAN

Abdulwareth, A. J., & Al-Shargabi, A. A. (2021). Toward a multi-criteria framework for selecting software testing tools. *IEEE Access*, *9*, 158872–158891.

Agung Widhi Kurniawan, Z. P. (2016). Metode Penelitian Kuantitatif. *Pandiva Buku*.

Ariqah, A. (2023). *Unit Testing atau Pengujian Unit Aplikasi Inventaris pada Badan Pusat Statistik Kota Malang*.

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).

Daka, E., Campos, J., Fraser, G., Dorn, J., & Weimer, W. (2015). Modeling readability to improve unit tests. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 107–118.

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. (2011). 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. (n.d.). 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. (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.

Muharam, C. S., Kusnendi, N., Fauzi, M. I., Yoseptry, R., & Wasliman, E. D. (2024). Efektivitas Model Pembelajaran Teaching Factory (Tefa) Dalam Meningkatkan Kualitas Lulusan Pada Smk Negeri Pertanian Pembangunan Cianjur. *Edusaintek: Jurnal Pendidikan, Sains Dan Teknologi*, *11*(4), 1839–1853.

Nugroho, M. O., & Cahyono, A. B. (2024). Perancangan Ui/Ux Digital Signage Untuk Peningkatan Akses Informasi Di Fakultas Teknologi Industri Uii. *Edusaintek: Jurnal Pendidikan, Sains Dan Teknologi*, *11*(2), 445–462.

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.

Pelivani, E., & Cico, B. (2021). A comparative study of automation testing tools for web applications. *2021 10th Mediterranean Conference on Embedded Computing (MECO)*, 1–6.

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.

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*.

Wang, Y., Mäntylä, M. V, Liu, Z., Markkula, J., & Raulamo-jurvanen, P. (2022). Improving test automation maturity: A multivocal literature review. *Software Testing, Verification and Reliability*, *32*(3), e1804.