BAB II
TINJAUAN PUSTAKA

2.1 Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah proses penting yang dilakukan dalam siklus pengembangan perangkat lunak. Tujuan utama dari proses pengujian adalah untuk menjamin bahwa perangkat lunak yang telah dikembangkan memenuhi kebutuhan pengguna serta beroperasi sesuai dengan fungsi yang telah ditetapkan. Pengujian perangkat lunak tidak hanya bertujuan untuk menemukan kesalahan, tetapi juga untuk memberikan tingkat kepercayaan terhadap kualitas sistem. Dengan kata lain, pengujian berfungsi sebagai mekanisme validasi dan verifikasi bahwa produk perangkat lunak telah memenuhi spesifikasi dan standar yang ditentukan[7]. Verifikasi dilakukan agar dapat memastikan bahwa proses pembuatan perangkat lunak telah dilakukan dengan benar, sedangkan validasi bertujuan untuk memastikan bahwa perangkat lunak yang dibuat benar-benar sesuai dengan kebutuhan yang diminta oleh pengguna.

Secara umum, pengujian perangkat lunak dapat dilakukan secara manual maupun otomatis[8]. Pengujian manual dilakukan oleh seoramg tester yang secara langsung melakukan pemeriksaan fungsi sistem sesuai dengan skenario yang sudah ditentukan. Sementara itu, pengujian otomatis dilakukan dengan menggunakan skrip atau alat bantu yang mampu mengeksekusi pengujian secara otomatis dan berulang. Kedua metode tersebut memiliki keunggulan dan kelemahan masing-masing, dan seringkali digunakan secara bersamaan dalam proyek pengembangan perangkat lunak.

Dengan melakukan pengujian secara sistematis dan menyeluruh, pengembang dapat meminimalkan risiko terjadinya kesalahan yang dapat berdampak pada kerugian finansial, merusak reputasi, atau membahayakan pengguna. Oleh karena itu, pengujian perangkat lunak merupakan salah satu aktivitas yang penting dan tidak boleh diabaikan dalam proses pengembangan sistem yang handal, efisien, dan berkualitas tinggi[7].

2.2 Pengujian Otomatis

Pengujian otomatis merupakan salah satu metode dalam pengujian perangkat lunak yang dilakukan dengan bantuan skrip atau alat bantu untuk menjalankan pengujian secara otomatis dan berulang. Menurut penelitian yang dilakukan oleh Mubarak et al., pengujian perangkat lunak dapat dilaksanakan baik secara manual maupun otomatis[8]. Namun, dalam konteks sistem yang kompleks dan terus mengalami pembaruan, pengujian otomatis semakin menjadi hal yang penting karena dapat meningkatkan efisiensi, konsistensi, dan kecepatan dalam proses memverifikasi fungsionalitas aplikasi.

Secara umum, pengujian otomatis dirancang untuk mengotomatiskan skenario pengujian yang sebelumnya dilakukan secara manual. Proses ini mencakup penulisan skrip pengujian yang dapat dijalankan oleh alat bantu, sehingga alat bantu tersebut dapat menjalankan berbagai tindakan di dalam aplikasi dan membandingkan hasil yang diperoleh dengan hasil yang ditentukan sebelumnya. Dengan begitu, pengujian dapat dilakukan secara cepat, konsisten, dan dapat diulang berkali-kali tanpa memerlukan intervensi manusia secara langsung[8].

Selain itu, pengujian otomatis dapat dijadwalkan untuk berjalan secara rutin, misalnya setiap kali terdapat pembaruan pada kode sumber, sehingga integrasi pengujian dalam proses pengembangan perangkat lunak menjadi lebih seamless dan mendukung pendekatan Continuous Integration/Continuous Deployment (CI/CD). Dalam lingkungan pengembangan modern yang bersifat iteratif dan cepat, kemampuan ini sangat krusial untuk menjaga stabilitas dan kualitas perangkat lunak.

Namun demikian, pengujian otomatis juga memiliki tantangan tersendiri. Pembuatan dan pemeliharaan skrip pengujian membutuhkan waktu dan keterampilan teknis tertentu. Skrip yang tidak dirancang dengan baik dapat menyebabkan false positive atau false negative, sehingga hasil pengujian menjadi kurang akurat[2]. Oleh karena itu, pengujian otomatis sebaiknya diterapkan pada

skenario yang stabil dan berulang, seperti pengujian fungsionalitas inti, validasi form, atau pengujian API.

Secara keseluruhan, pengujian otomatis merupakan strategi penting dalam pengembangan perangkat lunak modern yang dapat meningkatkan efisiensi pengujian, mengurangi human error, dan mendukung proses pengembangan yang lebih cepat.

2.3 Framework Pengujian Otomatis

Framework pengujian otomatis merupakan komponen penting dalam proses pengujian perangkat lunak yang dilakukan secara otomatis. Menurut penelitian, automation testing framework adalah kombinasi dari alat, praktik terbaik, dan pedoman teknis yang dirancang untuk memfasilitasi proses pengujian perangkat lunak secara terstruktur dan efisien[9]. Framework ini menyediakan fondasi teknis yang memungkinkan pengembang dan penguji untuk menulis, menjalankan, serta memelihara skrip pengujian secara sistematis.

Sebuah framework pengujian terdiri dari dua bagian utama, yaitu struktur fisik dan interaksi logis. Struktur fisik merujuk pada susunan folder, file, skrip, dan konfigurasi yang digunakan dalam membangun dan menjalankan pengujian. Struktur ini biasanya dirancang secara modular agar skrip dapat digunakan kembali dan mudah dipelihara[10]. Di sisi lain, interaksi logis mengacu pada cara skrip pengujian berkomunikasi dengan aplikasi yang diuji, serta bagaimana framework menangani proses seperti setup, teardown, logging, reporting, dan penanganan exception.

Keberadaan framework pengujian otomatis memungkinkan proses pengujian menjadi lebih konsisten, terorganisir, dan mudah dikendalikan. Dengan adanya struktur dan standar yang ditetapkan dalam framework, tim pengujian dapat mengurangi duplikasi kode, meningkatkan keterbacaan, serta mempercepat pengembangan skrip pengujian[11]. Secara keseluruhan, framework pengujian otomatis berfungsi sebagai fondasi teknis dan prosedural dalam pengembangan skrip uji otomatis yang andal. Dengan menggunakan framework yang baik, tim

pengembang dan penguji dapat memastikan bahwa proses pengujian tidak hanya berjalan secara otomatis, tetapi juga dilakukan dengan cara yang konsisten, dapat diulang, dan dapat diskalakan sesuai kebutuhan proyek perangkat lunak.

2.4 Pengujian End to End

Pengujian end-to-end merupakan salah satu bentuk pengujian perangkat lunak yang bertujuan untuk memverifikasi seluruh alur sistem dari awal hingga akhir secara menyeluruh[5]. Dalam pengujian ini, aplikasi diuji sebagai satu kesatuan untuk memastikan bahwa semua komponen berfungsi dengan baik ketika digunakan bersama-sama dalam kondisi yang menyerupai lingkungan nyata. Pengujian end-to-end mengevaluasi integrasi dan interaksi antara berbagai bagian sistem, termasuk antarmuka pengguna, basis data, API, serta komponen eksternal lainnya yang terlibat dalam proses bisnis.

Tujuan utama dari pengujian end-to-end adalah untuk memastikan bahwa aplikasi mampu menjalankan proses bisnis sebagaimana mestinya dari perspektif pengguna akhir[12]. Oleh karena itu, Pengujian end-to-end sering kali mencerminkan pengalaman pengguna secara langsung, sehingga sangat penting dalam menjamin kualitas dan keandalan aplikasi sebelum dirilis ke lingkungan produksi. Pengujian jenis ini memiliki cakupan yang lebih luas dibandingkan pengujian unit atau pengujian integrasi. Jika pengujian unit fokus pada fungsi atau komponen kecil secara terpisah, maka pengujian end-to-end memastikan bahwa seluruh sistem dapat berfungsi secara harmonis saat digunakan secara utuh. Hal ini juga mencakup validasi terhadap dependensi eksternal seperti layanan pihak ketiga, sistem otentikasi, atau gateway pembayaran, yang tidak selalu diuji secara menyeluruh pada level pengujian lain.

Namun demikian, pengujian end-to-end baisanya lebih rumit dan membutuhkan waktu eksekusi yang lebih lama. Hal ini terjadi karena proses pengujian mencakup banyak tahapan dan interaksi antar subsistem. Oleh karena itu, perencanaan skenario pengujian end-to-end harus dilakukan secara hati-hati agar pengujian tetap relevan, efisien, dan tidak redundan. Meski demikian, manfaat dari pengujian end-to-end sangat signifikan karena dapat mendeteksi kesalahan yang

mungkin luput dalam pengujian skala kecil, serta memberikan jaminan bahwa aplikasi siap digunakan oleh pengguna akhir[5].

2.5 Kasus Uji

Kasus uji merupakan elemen fundamental dalam proses pengujian perangkat lunak. Secara umum, kasus uji adalah seperangkat kondisi atau variabel yang digunakan oleh penguji untuk menentukan apakah suatu fitur atau fungsi dari perangkat lunak bekerja sesuai dengan yang diharapkan[13]. Dengan kata lain, kasus uji berfungsi sebagai pedoman sistematis untuk menguji dan memverifikasi kebenaran dari hasil eksekusi sistem terhadap spesifikasi yang telah ditetapkan.

Setiap kasus uji umumnya mencakup beberapa komponen penting, antara lain: identifikasi kasus uji, deskripsi singkat pengujian, langkah-langkah uji, data uji, hasil yang diharapkan, serta hasil aktual. Dengan mencantumkan informasi tersebut, kasus uji dapat menjadi dokumentasi yang jelas dan terukur dalam proses pengujian. Dokumentasi yang baik memungkinkan tim pengujian untuk mengevaluasi fungsionalitas perangkat lunak dengan lebih akurat dan objektif.

Pada tabel 2.1 dibawah ini merupakan contoh isi yang ada dalam document kasus uji pada umumnya.

Tabel 2. 1 Contoh Kasus Uji

ID Kasus uji	Fitur	Kondisi Sebelumnya	Kasus uji	Langkah Uji	Hasil yang Diharapkan
TC-1	Fitur A	-	-	-	-
TC-2	Fitur B	-	-	-	-

Selain itu, kasus uji juga dapat bersifat positif maupun negatif. Kasus uji positif dirancang untuk menguji fungsi sistem dengan input yang valid dan sesuai, sedangkan kasus uji negatif menguji bagaimana sistem merespon terhadap input yang tidak valid, kondisi error, atau skenario yang tidak terduga. Keduanya sama pentingnya untuk memastikan sistem tidak hanya berfungsi dengan benar, tetapi juga tangguh dalam menghadapi kemungkinan kesalahan.

Setiap test case seharusnya terdokumentasi dengan jelas, termasuk langkah-langkah yang harus diikuti, data uji yang diperlukan, dan hasil yang diharapkan. Proses pengujian menggunakan test case membantu identifikasi dan perbaikan kesalahan sejak awal pengembangan, meningkatkan keandalan, keamanan, dan kinerja perangkat lunak. Dengan merinci test case dengan baik, tim pengembangan dapat memastikan bahwa produk akhir memenuhi standar kualitas dan kebutuhan pengguna dengan sebaik mungkin[6].

Selain penyusunan test case secara tradisional, salah satu teknik yang dapat digunakan untuk meningkatkan efektivitas proses pengujian adalah *Equivalence Class Partitioning* (ECP). ECP merupakan metode perancangan kasus uji dengan membagi himpunan input ke dalam beberapa kelas ekuivalen, di mana setiap kelas dianggap mewakili perilaku sistem yang sama[22]. Dengan membagi input menjadi valid equivalence class dan invalid equivalence class, penguji dapat memilih satu atau beberapa nilai representatif dari setiap kelas tanpa harus menguji seluruh kemungkinan input. Pendekatan ini membuat proses pengujian menjadi lebih efisien karena mengurangi jumlah test case, tetapi tetap menjaga cakupan pengujian dan kemampuan deteksi kesalahan. ECP sangat berguna ketika sistem memiliki banyak kemungkinan nilai input, sehingga teknik ini membantu penguji memfokuskan upaya pada data yang paling kritis dan representatif dalam memastikan kualitas perangkat lunak.

2.6 Multi Attribute Rating Technique (SMART)

Simple Multi Attribute Rating Technique (SMART) merupakan salah satu metode pengambilan keputusan yang digunakan untuk mengevaluasi dan membandingkan sejumlah alternatif berdasarkan beberapa kriteria atau atribut[6]. Metode ini termasuk bagian dari kelompok multi-criteria decision-making (MCDM), yang sangat bermanfaat ketika pengambilan keputusan tidak dapat didasarkan hanya pada satu faktor, melainkan memerlukan pertimbangan terhadap berbagai aspek secara bersamaan.

SMART dikembangkan untuk memberikan pendekatan yang sederhana, intuitif, dan fleksibel dalam menilai alternatif-alternatif berdasarkan bobot dan nilai

preferensi dari masing-masing kriteria[6]. Proses utama dalam metode ini melibatkan penetapan kriteria yang relevan, pemberian bobot pada tiap kriteria berdasarkan tingkat kepentingannya, serta pemberian skor terhadap masing-masing alternatif pada setiap kriteria yang telah ditentukan. Hasil akhir berupa skor agregat untuk tiap alternatif, yang dihitung dengan menjumlahkan hasil perkalian antara bobot dan nilai setiap kriteria.

Kelebihan metode SMART terletak pada kesederhanaannya serta kemampuannya untuk diaplikasikan dalam berbagai konteks pengambilan keputusan, termasuk dalam bidang rekayasa perangkat lunak, manajemen proyek, dan evaluasi sistem. Metode ini juga memudahkan pelibatan pemangku kepentingan karena tidak memerlukan perhitungan matematis yang kompleks, sehingga transparan dan mudah dipahami.

2.7 Penelitian Terdahulu

Penelitian jelas mempertimbangkan dan menganalisis penelitian sebelumnya. Berdasarkan temuan penelitian sebelumnya, penulis berharap penelitian ini akan memberikan informasi tambahan yang berharga dan berfungsi sebagai referensi yang dapat digunakan oleh peneliti lain di masa depan. Beberapa penelitian relevan dari berbagai literatur yang telah dicari.

Penelitian pertama yang dijadikan acuan adalah "Perbandingan Kinerja *Framework Automation* UI *Testing* Menggunakan Metode *The Distance To The Ideal Alternative*" karya G. A. Prasetyo dan N. Setiani. Dalam penelitian tersebut, dibahas membandingkan kinerja tiga *framework automation testing*, yaitu Selenium, Cypress, dan Playwright, dalam pengujian UI pada platform website, dengan menerapkan metode decesion making *Distance to Ideal Alternative* (DIA) untuk menilai performa *framework*[14]. Penelitian tersebut berkaitan dengan diperlukannya perbandingan kinerja *framework automation testing* menggunakan *Multi Criteria Decesion Making* (MCDM) agar dapat menentukan *framework* mana yang dapat digunakan oleh penguji dengan efektif dan efisien. Oleh karena itu, penulis merujuk pada penelitian G. A. Prasetyo dan N. Setiani sebagai penelitian

terdahulu yang relevan, dengan dasar ada kaitannya dengan perbandingan kinerja *framework automation testing* menggunakan *Multi Criteria Decesion Making* (MCDM). Sementara dalam penelitian ini berfokus pada membandingkan tiga macam *framework* dengan satu framework yang sama, yaitu Cypress, dan dua *framework* yang berbeda, yaitu Serenity, dan Robot, menerapkan metode *Multi Criteria Decesion Making* (MCDM) *Simple Multi Attribute Rating Technique* (SMART).

Penelitian kedua 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* " karya C. Merina, membahas tentang perbandingan kinerja tiga *test automation framework*, yaitu Espresso, Calabash, dan Appium dengan *covered test case, time complexity, execution speed,* dan *element inspection* untuk parameter *automated testing progress*, kemudian membandingkannya dengan matriks yang mendefinisikan parameter *tool usability* menerapkan metode *The Distance To The Ideal Alternative* (DIA)[3]. Penelitian tersebut dijadikan penelitian terdahulu oleh penulis karena memiliki kesamaan dalam menentukan tiga kriteria dari segi kinerja, yaitu *execution speed, time complexity,* dan *covered test case* yang akan dijadikan parameter dalam analisis perbandingan kinerja *framework automation testing*.

Penelitian ketiga karya H. V. Gamido dan M. V. Gamido dengan judul "*Comparative Review of The* Fitur*s of Automated Software Testing Tools*" membahas Analisis dan perbandingan *automated software testing tools* Selenium, QTP/UFT, TestComplete, dan Ranorex berdasarkan kemudahan penggunaan (*usability*), dukungan bahasa pemrograman, kompatibilitas platform, kemampuan *automation testing*, fitur *reporting*. Penelitian tersebut dijadikan acuan penelitian terdahulu karena memiliki kesamaan dalam mengevaluasi *tools* atau *framework testing* untuk menentukan *tools* atau *framework* mana yang terbaik.

Penelitian keempat, yang dilakukan oleh Hanif Ur Rahman, Asaad Alzayed, Muhammad Ismail Mohmand, Abdullah M. Albarrak, dan Sultan Noman Qasem berjudul "*Application Maintenance Offshoring Using HCI-Based Framework and*

Simple Multi Attribute Rating Technique (SMART)" membahas penerapan metode SMART dalam mendukung pengambilan keputusan pada pemilihan strategi *application maintenance offshoring* dengan mempertimbangkan berbagai kriteria berbasis *Human-Computer Interaction* (HCI). Penelitian ini menunjukkan bagaimana metode SMART digunakan untuk mengevaluasi berbagai alternatif secara sistematis dan membuat keputusan yang objektif berdasarkan pembobotan dan nilai *utility*. Metode ini memberikan landasan metodologis yang kuat untuk penelitian ini, meskipun subjek yang dikaji berbeda, yaitu pada evaluasi kinerja framework automation testing.

Penelitian terakhir yang dijadikan penelitian terdahulu adalah Jurnal karya Hamed Taherdoost dan Atefeh Mohebi berjudul "*Using SMART Method for Multi-Criteria Decision Making: Applications, Advantages and Limitations*" membahas konsep, tahapan, kelebihan, serta keterbatasan metode SMART dalam pengambilan keputusan multikriteria. Dalam penelitian tersebut, SMART dibandingkan dengan beberapa metode MCDM lain seperti AHP (Analytical Hierarchy Process), TOPSIS (Technique for Order Preference by Similarity to Ideal Solution), dan ELECTRE, yang umumnya memiliki proses perhitungan lebih kompleks seperti perbandingan berpasangan atau analisis jarak ke solusi ideal. Hasil pembahasan menunjukkan bahwa SMART lebih unggul dari segi kesederhanaan, kemudahan implementasi, serta efisiensi perhitungan, namun tetap mampu menghasilkan keputusan yang objektif melalui pembobotan dan perhitungan nilai utilitas [6]. Penelitian ini relevan dijadikan acuan karena menggunakan metode yang sama, yaitu SMART, sekaligus memberikan justifikasi ilmiah mengapa metode tersebut dipilih dibandingkan metode lain, sehingga dapat memperkuat dasar metodologi dalam penelitian ini, khususnya dalam proses evaluasi dan penentuan ranking framework automation testing secara terukur dan sistematis.