BAB II

TINJAUAN LITERATUR

Bab ini menguraikan berbagai hasil penelitian sebelumnya yang berkaitan dengan topik yang dibahas, serta menyajikan sumber-sumber rujukan yang mendukung penelitian ini. Dengan demikian, tinjauan literatur berfungsi memberikan gambaran perbandingan yang jelas antara penelitian terdahulu dan penelitian saat ini.

2.1 Landasan Teori

2.1.1 Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan proses penting dalam pengembangan sistem yang bertujuan untuk menjamin kualitas, kinerja, serta keamanan produk yang dihasilkan. Proses ini mencakup evaluasi terhadap sistem atau komponen perangkat lunak untuk mengidentifikasi kesalahan serta memastikan bahwa sistem berfungsi sesuai dengan spesifikasi yang telah ditetapkan. Oleh karena itu, pengujian menjadi tahap krusial karena berperan dalam memastikan kualitas perangkat lunak, baik dari sisi perancangan maupun struktur kontrol pemrogramannya [9].

2.1.2 Jenis-Jenis Pengujian Perangkat Lunak

a) *White Box Testing* dan *Black Box Testing*

Secara umum, terdapat beberapa pendekatan dalam pengujian perangkat lunak, di antaranya *White Box Testing* dan *Black Box Testing*. *White Box Testing*

dilakukan dengan menganalisis struktur internal dan kode program untuk mendeteksi kesalahan logika maupun implementasi pada level modul. Sementara itu, *Black Box Testing* berfokus pada pengujian dari sisi eksternal, yaitu dengan mengevaluasi fungsi-fungsi aplikasi, tampilan antarmuka, serta kesesuaian alur sistem dengan spesifikasi yang telah dirancang, tanpa memperhatikan detail kode program di dalamnya [10].

b) *Unit testing* dan *Integration Testing*

Selain itu, terdapat *Unit Testing* dan *Integration Testing* dalam praktik pengujian perangkat lunak. *Unit Testing* dilakukan untuk menguji unit terkecil kode agar berfungsi dengan benar dan konsisten. Selanjutnya, *Integration Testing* bertujuan menguji gabungan beberapa unit untuk memastikan interaksi antar komponen berjalan sesuai harapan [11].

2.1.3 Pengujian Manual dan Otomatis

Pengujian manual merupakan metode pengujian yang dilakukan secara langsung oleh manusia dengan mengikuti skenario yang telah ditentukan untuk mengevaluasi fungsionalitas sistem. Metode ini memiliki keterbatasan, terutama dari segi waktu dan ketergantungan pada kemampuan penguji dalam menemukan serta melaporkan kesalahan. Sebaliknya, pengujian otomatis memanfaatkan perangkat lunak untuk mengeksekusi skenario dan kasus uji secara sistematis. Pendekatan ini lebih efisien karena mampu menghemat waktu dan tenaga, serta efektif dalam menangani pengujian dengan tingkat kompleksitas tinggi dan volume data yang besar [12].

2.1.4 Pengujian Otomatis Perangkat Lunak

a) Pengujian Otomatis / *Automation testing*

Pengujian otomatis merupakan pengembangan dari pengujian manual yang memanfaatkan skrip dan tools untuk menjalankan pengujian secara berulang dan terstruktur. Pendekatan ini umumnya digunakan dalam pengembangan perangkat lunak serta meningkatkan efisiensi pengujian.

b) Testing Framework

Untuk mendukung pelaksanaan pengujian otomatis, diperlukan suatu *testing framework* sebagai kerangka kerja yang menyediakan aturan, struktur, serta alat bantu dalam merancang, menjalankan, dan mengevaluasi proses pengujian perangkat lunak. *Framework* ini umumnya dilengkapi dengan fitur pengelolaan *test case*, mekanisme *assertion*, serta pembuatan laporan hasil pengujian. Dengan demikian, penggunaan *testing framework* dapat meningkatkan keteraturan, konsistensi, dan kemudahan pemeliharaan dalam proses pengujian perangkat lunak [13].

c) Jest

Jest merupakan *framework* pengujian yang dikembangkan untuk mendukung pengujian unit dan integrasi dengan performa yang cepat serta konfigurasi yang sederhana. *Jest* memiliki berbagai fitur unggulan seperti *zero configuration*, *snapshot testing*, *automatic mocking*, *code coverage reporting*, dan kemampuan menjalankan pengujian secara paralel. Selain itu, *Jest* juga dikenal mampu menangani pengujian secara paralel serta mendukung deteksi permasalahan seperti *flaky test* yang dapat memengaruhi konsistensi hasil pengujian [14].

d) *Test case* dan *Test coverage*

Dalam pengujian otomatis, *test case* digunakan sebagai acuan untuk menguji fungsi atau fitur sistem. *Test case* merupakan kumpulan skenario yang dirancang untuk memverifikasi apakah sistem menghasilkan output sesuai dengan yang diharapkan [15]. Umumnya, *test case* mencakup identitas pengujian, deskripsi, input, langkah pengujian, serta hasil yang diharapkan.

Selain itu, digunakan *test coverage* sebagai metrik untuk mengukur sejauh mana pengujian mencakup bagian kode program. *Test coverage* menunjukkan persentase kode yang telah diuji, dengan beberapa jenis seperti statement coverage, branch coverage, dan function coverage [16]. Semakin tinggi nilai coverage, semakin luas cakupan pengujian, meskipun tidak menjamin sistem sepenuhnya bebas dari kesalahan.

e) *Software Testing Life Cycle* (STLC)

Agar proses pengujian berjalan secara sistematis dan terstruktur, digunakan pendekatan *Software Testing Life Cycle* (STLC). STLC merupakan serangkaian tahapan yang dilakukan dalam proses pengujian perangkat lunak, mulai dari analisis kebutuhan hingga evaluasi hasil pengujian [17]. Dengan menerapkan STLC, proses pengujian dapat dilakukan secara terarah, terdokumentasi dengan baik, serta mampu meningkatkan kualitas dan keandalan sistem secara keseluruhan.

Berdasarkan uraian tersebut, kombinasi pengujian otomatis dengan *framework* seperti *Jest*, *test case* terstruktur, *test coverage*, dan penerapan STLC efektif meningkatkan kualitas perangkat lunak. Pendekatan ini membuat pengujian

lebih efisien, konsisten, dan andal dalam mendukung pengembangan sistem web seperti SiKosa.

2.2 Literatur Review

Berdasarkan telaah terhadap jurnal-jurnal sebelumnya, diperoleh hasil perbandingan yang disajikan pada tabel berikut.

Tabel 2. 1 Review Jurnal Terdahulu

No	Tinjauan Pustaka	Kesimpulan
(1)	(2)	(3)
1	Ahmad Jailani, Muhammad Ainul yaqin (2024) Pengujian Aplikasi Sistem Informasi Akademik menggunakan Metode Blackbox dengan Teknik *Boundary Value Analysis*	Artikel ini menekankan pentingnya pengujian perangkat lunak untuk menjaga kualitas aplikasi akademik dengan menggunakan metode *Black Box Testing* dan teknik *Boundary Value Analysis* (BVA). Pengujian ini efektif mendeteksi error pada batas input dan menghasilkan temuan bahwa hanya 60% pengujian yang berhasil. Namun, penelitian memiliki keterbatasan, seperti tidak dianalisisnya penyebab kegagalan secara mendalam, tidak adanya otomatisasi pengujian, serta cakupan pengujian yang belum menyeluruh.
2	Ibnu Adha Shaleh, Juma Prayogi, Perdi Pirdaus, Riky Syawal, Aries Saifudin (2021) Pengujian *Black Box* pada Sistem Informasi Penjualan Buku Berbasis Web dengan Teknik *Equivalent Partitions*	Artikel ini menyoroti pentingnya kualitas perangkat lunak pada sistem informasi berbasis web dan menggunakan *Black Box Testing* dengan teknik *Equivalent Partitions* untuk menguji validitas input pada form aplikasi penjualan buku. Teknik ini memberikan struktur pengujian yang lebih baik dan relevan untuk sistem berbasis web. Namun, penelitian ini memiliki keterbatasan karena seluruh pengujian dilakukan secara manual dan tidak menyertakan data kuantitatif seperti jumlah keberhasilan atau kegagalan *test case*.
3	Dahlia Widhyaestoeti, Saidul Iqram, Siti Nur Mutiyah, Yasmin Khairunnisa (2021) *Black Box Testing* Equivalence Partitions Untuk Pengujian Front-End Pada Sistem Akademik Sitoda	Penelitian ini menguji front-end sistem akademik SITODA berbasis web menggunakan metode *Black Box Testing* dengan teknik Equivalence Partitioning. Pengujian disusun melalui tabel perencanaan dan tabel fungsional, serta melibatkan end-user untuk memastikan relevansi kebutuhan. Teknik ini membuat *test case* lebih terstruktur dan meningkatkan keandalan antarmuka. Namun, pengujian hanya mencakup front-end, dilakukan secara manual, dan hasilnya bersifat deskriptif tanpa data kuantitatif.

Tabel 2.1 Review Jurnal Terdahulu (Lanjutan)

(1)	(2)	(3)
4	Mintarsih (2023) Pengujian *Black Box* Dengan Teknik Transition Pada Sistem Informasi Perpustakaan Berbasis Web Dengan Metode Waterfall Pada SMC Foundation	Penelitian ini menguji Sistem Informasi Perpustakaan SMC Foundation berbasis web menggunakan metode *Black Box Testing* dengan teknik *State Transition Testing*. Teknik ini memastikan setiap perpindahan state dalam sistem berjalan sesuai ekspektasi. Pendekatan tersebut relevan untuk sistem dengan banyak perubahan status dan mudah diterapkan karena berfokus pada input–output. Namun, pengujian masih dilakukan secara manual dan tidak disertai data kuantitatif mengenai jumlah transisi atau tingkat keberhasilan pengujian.
5	Muhammad Jibril, Zulrahmadi, Muhammad Amin (2024) Pengujian Sistem Informasi E-Modul Pada Smpn 1 Tempuling Menggunakan *Black Box Testing*	Penelitian ini menguji fungsionalitas sistem informasi e-modul di SMPN 1 Tempuling menggunakan metode *Black Box Testing*, yang berfokus pada perilaku aplikasi tanpa melihat kode sumber. Hasil pengujian menunjukkan bahwa sistem telah sesuai spesifikasi dan mudah diuji karena metodenya sederhana. Namun, pengujian dilakukan secara manual dan tidak disertai data kuantitatif seperti jumlah *test case* atau tingkat keberhasilan.
6	Dona Yuliawati, Anggi Andriyadi, Nursiyanto (2022) Pengujian Sistem Informasi E-Monitoring Pengelolaan Pembangunan Desa Dengan Menggunakan Metode *Black Box Testing*	Penelitian ini menguji sistem informasi e-monitoring pembangunan desa menggunakan metode *Black Box Testing* dengan total 42 skenario uji untuk menilai kesesuaian fungsi sistem. Hasil menunjukkan 95% skenario berjalan valid, sementara beberapa lainnya masih memerlukan perbaikan. Pengujian dinilai objektif karena menyertakan data kuantitatif, meskipun seluruh proses masih dilakukan secara manual.
7	Jafar Shadiq, Ahmad Safei, Rayhan Wahyudin Ratu Loly (2021) Pengujian Aplikasi Peminjaman Kendaraan Operasional Kantor Menggunakan *Black Box Testing*	Penelitian ini menguji aplikasi sistem informasi peminjaman kendaraan operasional menggunakan metode *Black Box Testing* dengan teknik *Equivalence Partitioning*. Teknik ini mengelompokkan input ke dalam kelas valid dan tidak valid untuk memastikan sistem memberikan output sesuai harapan. Pendekatan ini membuat pengujian lebih terstruktur dan sistematis, meskipun masih dilakukan secara manual dan tidak disertai data kuantitatif mengenai hasil *test case*.

Tabel 2.1 Review Jurnal Terdahulu (Lanjutan)

(1)	(2)	(3)
8	Arif Samdono, Anggraini Puspita Sari, Firza Prima Aditiawan (2024) Pengujian *Black Box* Pada Sistem Informasi Stok dan Penjualan Berbasis Website Menggunakan Metode *Equivalence Partitioning* (Studi Kasus: Cv. Algani Karya Mandiri)	Artikel ini menguji Sistem Informasi Stok dan Penjualan berbasis web di CV. Algani Karya Mandiri menggunakan metode *Black Box Testing* dengan teknik *Equivalence Partitioning* untuk memastikan fungsionalitas sistem dan mendeteksi potensi kesalahan. Hasil menunjukkan efektivitas sistem sebesar 91,87%, menandakan tingkat keandalan yang cukup baik. Namun, beberapa masalah masih ditemukan dan pengujian dilakukan sepenuhnya secara manual tanpa pembahasan mengenai otomatisasi
9	Amanda Amalia, Salva Wanda Putri Hamidah, Titus Kristanto (2021) Pengujian *Black Box* Menggunakan Teknik *Equivalence Partitions* Pada Aplikasi E-Learning Berbasis Web	Artikel ini menguji kualitas aplikasi E-Learning berbasis web di Institut Teknologi Telkom Surabaya menggunakan metode *Black Box Testing* dengan teknik *Equivalence Partitioning*. Pengujian dilakukan melalui identifikasi fungsi, perancangan *test case*, eksekusi uji, dan penarikan kesimpulan. Teknik ini efektif karena membagi data uji ke dalam kelas valid dan tidak valid. Namun, hasil pengujian hanya menyatakan bahwa aplikasi layak digunakan tanpa data kuantitatif, dan proses pengujian masih sepenuhnya manual tanpa pembahasan otomasi.
10	Fitriana Sekar Kinasih, Habib Nur Gian, Hanifah Permatasari (2023) Pengujian Sistem Informasi Inventaris Barang Berbasis Website Menggunakan Metode *Whitebox Testing*	Artikel ini menguji sistem informasi inventaris barang berbasis web pada halaman distributor dengan menggunakan metode *White Box Testing* melalui analisis *Cyclomatic Complexity*. Pengujian ini mengevaluasi alur logika internal sistem untuk mengukur kompleksitas dan mengidentifikasi jalur independen yang berpotensi menimbulkan error. Namun, objek uji hanya terbatas pada satu halaman, tidak membahas aspek fungsional dari sisi pengguna, dan belum mengombinasikan metode lain untuk hasil yang lebih menyeluruh.
11	Philipp Straubinger, Tommaso Fulcini, dkk (2024) *IntelliGame in Action: An Experience Report on Gamifying JavaScript Unit Tests*	Penelitian ini mengkaji integrasi IntelliGame, yaitu plugin berbasis gamifikasi, dalam pengujian unit *JavaScript* menggunakan *Jest* melalui eksperimen terkontrol terhadap 152 partisipan. Hasil penelitian menunjukkan bahwa penerapan gamifikasi dalam proses pengujian mampu meningkatkan keterlibatan serta memengaruhi perilaku developer dalam menulis dan menjalankan pengujian.

Tabel 2.1 Review Jurnal Terdahulu (Lanjutan)

(1)	(2)	(3)
12	Negar Hashemi, dkk (2025) Detecting and Evaluating Order-Dependent *Flaky tests* in *JavaScript*	Penelitian ini mengkaji fenomena *flaky test* pada pengujian *JavaScript* menggunakan *Jest*, khususnya yang disebabkan oleh ketergantungan urutan eksekusi (test order dependency). Melalui eksperimen terhadap 81 proyek, dilakukan pengacakan urutan test dan eksekusi ulang untuk mendeteksi perubahan hasil pengujian. Hasil menunjukkan terdapat 55 kasus order-dependent tests pada 10 proyek, yang sebagian besar terjadi antar test. Penyebab utama yang ditemukan adalah penggunaan file bersama (shared files) dan kondisi mocking yang tidak terisolasi (shared mocking state). Temuan ini menegaskan bahwa selain faktor umum, terdapat penyebab baru pada *Jest* yang dapat memengaruhi stabilitas pengujian, sehingga perancangan test yang baik menjadi penting untuk menjaga keandalan hasil pengujian.

Berdasarkan tabel 2.1, ringkasan dari berbagai penelitian tersebut menunjukkan bahwa, Ahmad Jailani dan Muhammad Ainul Yaqin menekankan pentingnya pengujian perangkat lunak menggunakan *Black Box Testing* dan teknik *Boundary Value Analysis*, namun masih memiliki kelemahan berupa cakupan pengujian yang terbatas dan belum adanya otomatisasi [18]. Selanjutnya, Ibnu Adha Shaleh dkk menunjukkan bahwa teknik *Equivalent Partitions* dapat meningkatkan struktur pengujian input pada sistem berbasis web, meskipun seluruh proses masih dilakukan secara manual dan tanpa data kuantitatif [19]. Menurut Dahlia Widhyaestoeti dkk teknik *Equivalence Partitioning* juga efektif untuk menyusun *test case* front-end secara terstruktur, tetapi pengujian mereka masih manual dan hanya mencakup sisi antarmuka tanpa hasil kuantitatif [20]. Penelitian Mintarsih menegaskan bahwa teknik *State Transition Testing* relevan untuk sistem dengan banyak perubahan status, namun kembali ditemukan bahwa pengujian masih manual dan tanpa data kuantitatif pendukung [21].

Kemudian, Muhammad Jibril dkk membuktikan bahwa *Black Box Testing* dapat memastikan kesesuaian fungsionalitas sistem e-modul, tetapi pengujiannya masih manual dan tidak menyertakan jumlah *test case* maupun tingkat keberhasilan [22]. Sementara itu, Dona Yuliawati dkk memberikan temuan yang lebih objektif dengan mencatat 95% skenario pengujian berjalan valid dari total 42 skenario, meski seluruh proses tetap dilakukan secara manual [23]. Penelitian oleh Jafar Shadiq dkk menunjukkan bahwa teknik *Equivalence Partitioning* dapat membuat pengujian lebih terstruktur dan sistematis, namun kelemahannya serupa, masih manual dan tanpa data kuantitatif [24]. Menurut Arif Samdono dkk efektivitas sistem mencapai 91,87% melalui pengujian *Equivalence Partitioning*, menandakan reliabilitas yang baik, tetapi penelitian belum membahas otomatisasi pengujian [25]. Penelitian Amanda Amalia dkk juga menegaskan efektivitas teknik *Equivalence Partitioning* pada aplikasi e-learning, tetapi hasil pengujian bersifat deskriptif tanpa data kuantitatif dan tetap manual [26].

Fitriana Sekar Kinasih dkk menggunakan *White Box Testing* untuk mengukur kompleksitas logika sistem, namun pengujiannya terbatas pada satu halaman dan tidak mencakup aspek fungsional dari sudut pandang pengguna[27]. Philipp Straubinger dkk menunjukkan bahwa integrasi IntelliGame dengan *Jest* dalam pengujian unit *JavaScript* mampu meningkatkan keterlibatan dan perilaku developer dalam proses *testing*. Selain itu, pendekatan ini memberikan pengalaman pengujian yang lebih interaktif dan berpotensi meningkatkan kualitas serta efisiensi pengujian perangkat lunak [28]. Negar Hashemi dkk menunjukkan bahwa *flaky test* pada *Jest* banyak disebabkan oleh ketergantungan urutan eksekusi, dengan 55 kasus ditemukan pada 81 proyek. Penyebab utamanya adalah penggunaan *shared files* dan

shared mocking state, sehingga diperlukan perancangan test yang baik untuk menjaga stabilitas pengujian [29].

2.3 Kesimpulan Literatur

Berdasarkan hasil tinjauan dari berbagai penelitian sebelumnya, dapat disimpulkan bahwa mayoritas penelitian terkait pengujian perangkat lunak masih berfokus pada *Black Box Testing* yang dilakukan secara manual dengan menerapkan berbagai teknik seperti *Equivalence Partitioning*, *Boundary Value Analysis*, dan *State Transition*. Beberapa penelitian lain menerapkan pendekatan *White Box Testing*, seperti analisis *Cyclomatic Complexity* untuk mengukur kompleksitas kode dan jalur eksekusi.

Secara umum, hasil pengujian pada penelitian-penelitian tersebut menunjukkan bahwa sistem yang diuji telah berjalan sesuai kebutuhan. Namun, terdapat perbedaan antara penelitian terdahulu dengan penelitian ini dijelaskan sebagai berikut:

1. Penelitian terdahulu umumnya masih melakukan pengujian secara manual sehingga membutuhkan waktu lebih lama, kurang efisien, dan berpotensi menimbulkan *human error*. Sebaliknya, penelitian ini menerapkan otomatisasi menggunakan *Jest* sehingga pengujian unit dan integrasi dapat dijalankan secara berulang, lebih cepat, konsisten, dan mudah dipelihara.
2. Penelitian terdahulu umumnya belum memiliki dokumentasi, standar, dan cakupan pengujian yang terukur sehingga hasil pengujian sulit dilacak maupun direplikasi. Sebaliknya, penelitian ini menerapkan *Software Testing Life Cycle* (STLC) yang dilengkapi *test scenario*, *test case*, *Requirements Traceability Matrix* (RTM), serta pengukuran *test coverage*, sehingga setiap

kebutuhan sistem dapat ditelusuri dan tingkat cakupan pengujiannya dapat dievaluasi secara lebih sistematis.

3. Penelitian terdahulu cenderung hanya menguji modul atau fitur tertentu, seperti halaman admin, form input, atau fungsi tunggal, sehingga cakupan pengujian belum mewakili keseluruhan alur kerja sistem. Sebaliknya, penelitian ini melakukan pengujian pada modul-modul utama SiKosa melalui pengujian unit dan integrasi sehingga cakupan pengujian menjadi lebih komprehensif.
4. Penelitian terdahulu lebih berfokus pada validasi fungsionalitas sistem, sedangkan pengujian terhadap kualitas kode, seperti pengujian unit berbasis modular dan *mocking*, masih belum banyak diterapkan. Sebaliknya, penelitian ini mengadopsi pendekatan modular dengan dukungan *mocking* untuk memastikan stabilitas, kualitas, dan *maintainability* aplikasi dalam jangka panjang.
5. Penelitian terdahulu belum mendukung pendeteksian kesalahan secara berkelanjutan ketika terjadi perubahan kode. Sebaliknya, penelitian ini memanfaatkan fitur *watch mode* pada *Jest* untuk mendeteksi *regression bug* lebih cepat, sehingga perubahan yang berpotensi memengaruhi fungsi yang telah berjalan stabil dapat segera diidentifikasi.