

**RANCANG BANGUN EKSTENSI VISUAL STUDIO CODE
UNTUK DETEKSI BUG JAVASCRIPT BERBASIS
RULE-BASED STATIC ANALYSIS**

TUGAS AKHIR



Disusun Oleh:

Rafi Alaudin Al Hafidz

22106050020

**STATE ISLAMIC UNIVERSITY
SUNAN KALIJAGA
YOGYAKARTA**

**PROGRAM STUDI S1 INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI SUNAN KALIJAGA
YOGYAKARTA**

2026

LEMBAR PENGESAHAN



KEMENTERIAN AGAMA
UNIVERSITAS ISLAM NEGERI SUNAN KALIJAGA
FAKULTAS SAINS DAN TEKNOLOGI
Jl. Marsda Adisucipto Telp. (0274) 540971 Fax. (0274) 519739 Yogyakarta 55281

PENGESAHAN TUGAS AKHIR

Nomor : B-1281/Un.02/DST/PP.00.9/06/2026

Tugas Akhir dengan judul : RANCANG BANGUN EKSTENSI VISUAL STUDIO CODE UNTUK DETEKSI BUG
JAVASCRIPT BERBASIS RULE-BASED STATIC ANALYSIS

yang dipersiapkan dan disusun oleh:

Nama : RAFI ALAUDIN AL HAFIDZ
Nomor Induk Mahasiswa : 22106050020
Telah diujikan pada : Jumat, 05 Juni 2026
Nilai ujian Tugas Akhir : A

dinyatakan telah diterima oleh Fakultas Sains dan Teknologi UIN Sunan Kalijaga Yogyakarta

TIM UJIAN TUGAS AKHIR



Ketua Sidang

Dr. Agus Mulyanto, S.Si., M.Kom., ASEAN Eng.
SIGNED

Valid ID: 6a22db192eef7



Penguji I

Muhammad Mustakim, S.T. M.T.
SIGNED

Valid ID: 6a25d7d4c0e9d



Penguji II

Ir. Muhammad Didik Rohmad Wahyudi, S.T.,
MT.
SIGNED

Valid ID: 6a27885be88b1



Yogyakarta, 05 Juni 2026

UIN Sunan Kalijaga
Dekan Fakultas Sains dan Teknologi

Prof. Dr. Dra. Hj. Khurul Wardati, M.Si.
SIGNED

Valid ID: 6a28dbc353ff6

LEMBAR PERNYATAAN KEASLIAN

LEMBAR PERNYATAAN KEASLIAN

Yang bertanda tangan di bawah ini:

Nama : Rafi Alaudin Al Hafidz

NIM : 22106050020

Program Studi : Informatika

Fakultas : Sains dan Teknologi

Menyatakan dengan sesungguhnya bahwa skripsi saya yang berjudul “RANCANG BANGUN EKSTENSI VISUAL STUDIO CODE UNTUK DETEKSI BUG JAVASCRIPT BERBASIS RULE-BASED STATIC ANALYSIS” merupakan penelitian saya sendiri, tidak terdapat karya yang pernah diajukan untuk memperoleh gelar kesarjanaan di suatu perguruan tinggi, dan bukan plagiasi karya orang lain kecuali secara tertulis diacu dalam naskah ini dan disebutkan dalam daftar pustaka. Pernyataan ini dibuat sebagai salah satu syarat untuk memperoleh gelar Sarjana Program Studi Informatika pada Fakultas Sains dan Teknologi, Universitas Islam Negeri Sunan Kalijaga.

STATE ISLAMIC UNIVERSITY
SUNAN KALIJAGA
YOGYAKARTA

Yogyakarta, 02 Februari 2026

Yang membuat pernyataan,



METERAI
TEMPEL
10000
1681AAOX034287708

Rafi Alaudin Al Hafidz

22106050020

LEMBAR PERSETUJUAN

LEMBAR PERSETUJUAN SKRIPSI/TUGAS AKHIR

Hal : Persetujuan Skripsi/Tugas Akhir

Lamp. : -

Kepada
Yth. Dekan Fakultas Sains dan Teknologi
UIN Sunan Kalijaga
DI Yogyakarta

Assalamu'alaikum wr. wb.

Setelah membaca, meneliti, memberikan petunjuk dan mengoreksi serta mengadakan perbaikan seperlunya, maka saya selaku pembimbing berpendapat bahwa skripsi Saudara:

Nama : Rafi Alaudin Al Hafidz

NIM : 22106050020

Judul Skripsi : Rancang Bangun Ekstensi Visual Studio Code untuk Deteksi Bug JavaScript Berbasis Rule-Based Static Analysis

Sudah dapat diajukan kepada Program Studi Informatika Fakultas Sains dan Teknologi UIN Sunan Kalijaga Yogyakarta sebagai salah satu syarat untuk memperoleh gelar Sarjana Strata Satu dalam Program Studi Informatika.

Dengan ini kami mengharap agar skripsi/tugas akhir Saudara dapat segera dimunaqosyahkan. Atas perhatiannya saya ucapkan terima kasih.

Wassalamu'alaikum wr. wb.

Yogyakarta, 02 Februari 2026
Pembimbing,



Dr. Ir. Agus Mulyanto, S.Si., M.Kom.
NIP. 19710823 199903 1 003

LEMBAR PEDOMAN PENGGUNAAN TUGAS AKHIR

Tugas Akhir ini tidak dipublikasikan, tetapi tersedia di perpustakaan dalam lingkungan UIN Sunan Kalijaga Yogyakarta, diperkenankan dipakai sebagai referensi kepustakaan, tetapi pengutipan harus seizin penyusun, dan harus menyebutkan sumbernya sesuai dengan kebiasaan ilmiah. Dokumen Tugas Akhir ini merupakan hak milik UIN Sunan Kalijaga Yogyakarta.



ABSTRAK

Penelitian ini membahas rancang bangun ekstensi Visual Studio Code untuk mendeteksi potensi bug pada kode JavaScript berbasis Rule-Based Static Analysis. Permasalahan yang diangkat adalah keterlambatan deteksi bug saat proses penulisan kode, terutama pada penggunaan operator yang kurang tepat, debugging statement yang tertinggal, fungsi berisiko, dan praktik penulisan kode yang kurang baik.

Metode pengembangan yang digunakan adalah Design and Development (D&D), meliputi tahap analisis, desain, pengembangan, dan evaluasi. Sistem bekerja dengan membaca kode JavaScript, melakukan parsing menjadi Abstract Syntax Tree (AST), memeriksa kode melalui rule engine, lalu menampilkan hasil deteksi menggunakan Diagnostic API pada editor Visual Studio Code.

Pengujian dilakukan melalui black box testing, rule testing terhadap 15 rule deteksi bug JavaScript, pengujian tambahan pada 10 file JavaScript dari contoh kode publik yang mengacu pada dokumentasi ESLint Rules, serta User Acceptance Testing (UAT) terhadap 10 responden. Rule yang diuji meliputi no-assignment-in-if, use-eq-eq-eq, use-isnan, no-console-log, no-debugger, no-self-assign, no-empty-block, no-constant-condition, no-math-object-call, no-alert, no-template-curly-in-string, no-throw-literal, no-nested-ternary, no-eval, dan no-implied-eval. Pemilihan rule diadaptasi dari pola kesalahan pada tools analisis kode JavaScript seperti ESLint dan JSHint, kemudian disesuaikan dengan ruang lingkup penelitian. Hasil pengujian menunjukkan bahwa seluruh rule berhasil mendeteksi pola kesalahan sesuai skenario dengan tingkat keberhasilan 100%. Hasil UAT memperoleh nilai rata-rata 4,53 dari skala 5 dengan kategori sangat setuju. Dengan demikian, ekstensi yang dikembangkan dapat digunakan sebagai alat bantu deteksi dini potensi bug JavaScript pada editor Visual Studio Code.

Kata kunci: JavaScript, Visual Studio Code, ekstensi, *Rule-Based Static Analysis*, AST, deteksi bug.

ABSTRACT

This research discusses the design and development of a Visual Studio Code extension for detecting potential bugs in JavaScript code using Rule-Based Static Analysis. The main problem addressed is the delayed detection of bugs during the coding process, particularly issues related to improper operator usage, remaining debugging statements, risky function usage, and poor coding practices.

The development method used in this research is Design and Development (D&D), which consists of analysis, design, development, and evaluation stages. The system works by reading JavaScript code, parsing it into an Abstract Syntax Tree (AST), analyzing it through a rule engine, and displaying the detection results using the Diagnostic API in the Visual Studio Code editor.

Testing was conducted through black box testing, rule testing on 15 JavaScript bug detection rules, additional testing on 10 JavaScript files from public code examples referring to ESLint Rules documentation, and User Acceptance Testing (UAT) involving 10 respondents. The tested rules include no-assignment-in-if, use-eq-eq-eq, use-isnan, no-console-log, no-debugger, no-self-assign, no-empty-block, no-constant-condition, no-math-object-call, no-alert, no-template-curly-in-string, no-throw-literal, no-nested-ternary, no-eval, and no-implied-eval. The rule selection was adapted from error patterns documented in JavaScript code analysis tools such as ESLint and JSHint, then adjusted to the scope of this research. The results show that all rules successfully detected the expected error patterns, achieving a rule detection success rate of 100%. The UAT result obtained an average score of 4.53 out of 5, categorized as strongly agree. Therefore, the developed extension can be used as an early detection tool for potential JavaScript bugs in the Visual Studio Code editor.

Keywords: JavaScript, Visual Studio Code, extension, *Rule-Based Static Analysis*, AST, bug detection.

MOTTO

“Hidup yang tidak dipertaruhkan tidak akan pernah dimenangkan.”
(*Sutan Sjahrir*)

“Terbentur, terbentur, terbentur, terbentuk.”
(*Tan Malaka*)

“Barang siapa ingin mutiara, harus berani terjun di lautan yang dalam.”
(*Soekarno*)

“Jangan takut jatuh, karena yang tidak pernah memanjatlah yang tidak pernah
jatuh.”
(*Buya Hamka*)

“Orang boleh pandai setinggi langit, tapi selama ia tidak menulis, ia akan hilang
di dalam masyarakat dan dari sejarah.”
(*Pramoedya Ananta Toer*)

“Tidak ada jalan pintas untuk menjadi Hokage.”
(*Naruto Uzumaki*)

“Kalau kau tidak mau mengambil risiko, kau tidak dapat menciptakan masa
depan.”
(*Monkey D. Luffy*)

“Kau gagal, tetapi masih bisa mampu bangkit kembali. Karena itu menurutku arti
dari kuat yang sebenarnya.”
(*Hinata Hyuga*)

KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِينَ، وَالصَّلَاةُ وَالسَّلَامُ عَلَى سَيِّدِنَا مُحَمَّدٍ، وَعَلَى آلِهِ وَصَحْبِهِ أَجْمَعِينَ

Puji syukur penulis panjatkan ke hadirat Allah SWT, Tuhan semesta alam, atas segala rahmat, taufik, hidayah, serta inayah-Nya sehingga penulis dapat menyelesaikan skripsi yang berjudul **“Rancang Bangun Ekstensi Visual Studio Code untuk Deteksi Bug JavaScript Berbasis Rule-Based Static Analysis”**. Shalawat serta salam semoga senantiasa tercurahkan kepada Nabi Muhammad SAW, keluarga, sahabat, serta seluruh umatnya hingga akhir zaman.

Penulisan skripsi ini disusun sebagai salah satu syarat untuk memperoleh gelar Sarjana pada Program Studi Informatika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Sunan Kalijaga Yogyakarta. Penulis menyadari bahwa skripsi ini masih memiliki kekurangan, baik dari segi penulisan, penyusunan, maupun pembahasan. Oleh karena itu, kritik dan saran yang membangun sangat penulis harapkan demi perbaikan dan penyempurnaan karya ini.

Penyusunan skripsi ini tidak terlepas dari bantuan, doa, dukungan, dan bimbingan dari berbagai pihak. Oleh karena itu, dengan segala kerendahan hati, penulis menyampaikan terima kasih kepada:

1. Bapak Prof. Noorhaidi Hasan, S.Ag., M.A., M.Phil., Ph.D. selaku Rektor Universitas Islam Negeri Sunan Kalijaga Yogyakarta.
2. Ibu Prof. Dr. Dra. Hj. Khurul Wardati, M.Si. selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Sunan Kalijaga Yogyakarta.
3. Bapak Dr. Muhammad Mustakim, S.T., M.T. selaku Ketua Program Studi Informatika yang telah memberikan dukungan selama masa perkuliahan.
4. Bapak Dr. Agus Mulyanto, S.Si., M.Kom., ASEAN Eng. selaku dosen pembimbing akademik sekaligus dosen pembimbing skripsi yang telah

memberikan bimbingan, arahan, dan masukan selama proses perkuliahan hingga penyusunan skripsi ini.

5. Seluruh dosen Program Studi Informatika Fakultas Sains dan Teknologi UIN Sunan Kalijaga Yogyakarta yang telah memberikan ilmu, pengalaman, dan pembelajaran selama masa studi.
6. Kedua orang tua tercinta, yang doanya tidak pernah putus dan kasih sayangnya selalu menjadi kekuatan terbesar bagi penulis. Terima kasih atas segala pengorbanan, kesabaran, dukungan, dan kepercayaan yang diberikan sehingga penulis mampu melewati proses panjang dalam menyelesaikan studi ini di UIN Sunan Kalijaga Yogyakarta. Sudah selayaknya tulisan sederhana ini penulis persembahkan kepada mereka.
7. Keluarga besar penulis yang senantiasa memberikan dukungan, motivasi, dan doa dalam proses penyusunan skripsi ini.
8. Putri Ayu Ah, yang telah memberikan dukungan, semangat, perhatian, dan motivasi kepada penulis selama proses penyusunan skripsi ini. Terima kasih atas kesabaran dan dukungan yang telah diberikan sehingga penulis dapat terus berusaha menyelesaikan karya ini.
9. Keluarga besar Kos Griya Rizky yang telah menjadi tempat tinggal, tempat berproses, dan ruang kebersamaan selama masa studi. Terima kasih atas suasana kekeluargaan, dukungan, serta kebersamaan yang telah memberikan warna dalam perjalanan penulis selama menempuh pendidikan di Yogyakarta.
10. Rekan-rekan mahasiswa Informatika yang telah menjadi teman seperjuangan selama masa studi, berbagi ilmu, pengalaman, serta semangat dalam menghadapi berbagai tantangan akademik.
11. Rekan-rekan seperjuangan yang telah membantu, mendukung, dan memberikan semangat kepada penulis dalam menyelesaikan skripsi ini.

12. Seluruh pihak yang tidak dapat penulis sebutkan satu per satu, terima kasih atas segala bentuk bantuan, doa, dan dukungan yang telah diberikan.

13. Terima kasih kepada Rafi Alaudin Al Hafidz, diri penulis sendiri, karena telah berusaha, bertahan, dan tidak menyerah meski harus melewati berbagai tantangan yang menguras energi dan emosi.

Penulis berharap semoga Allah SWT senantiasa melimpahkan rahmat, hidayah, dan balasan kebaikan kepada semua pihak yang telah membantu. Semoga skripsi ini dapat memberikan manfaat bagi pembaca serta menjadi kontribusi dalam bidang pengembangan perangkat lunak, khususnya pada penerapan *static analysis* dan pengembangan ekstensi Visual Studio Code.



DAFTAR ISI

LEMBAR PENGESAHAN	ii
LEMBAR PERNYATAAN KEASLIAN	iii
LEMBAR PERSETUJUAN.....	iv
LEMBAR PEDOMAN PENGGUNAAN TUGAS AKHIR.....	v
ABSTRAK	vi
ABSTRACT.....	vii
MOTTO.....	viii
KATA PENGANTAR.....	ix
DAFTAR ISI	xii
DAFTAR TABEL.....	xvi
DAFTAR GAMBAR	xvii
BAB I	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah	4
1.3. Batasan Masalah.....	4
1.4. Tujuan Penelitian.....	5
1.5. Manfaat Penelitian	5
1.5.1 Manfaat Teoritis	5
1.5.2 Manfaat Praktis	6
BAB II.....	7
2.1 Tinjauan Pustaka	7
2.2 Landasan Teori	9
2.2.1 <i>JavaScript</i>	9
2.2.2 <i>Static Code Analysis</i>	10

2.2.3 Rule-Based Static Analysis.....	11
2.2.4 Abstract Syntax Tree (AST)	13
2.2.5 Visual Studio Code Extension	15
2.2.6 Software Testing	16
2.3 Research Gap (Celah Penelitian)	18
2.4 Kerangka Konseptual Penelitian.....	19
2.5 Posisi Penelitian (State of The Art).....	22
BAB III	24
3.1 Metode Pengembangan	24
3.2 Tahapan Penelitian	25
3.2.1 Analysis.....	26
3.2.2 Design	26
3.2.3 Development	27
3.2.4 Evaluation	27
3.3 Analisis Kebutuhan Sistem	28
3.3.1 Kebutuhan Perangkat Lunak.....	29
3.3.2 Kebutuhan Perangkat Keras.....	29
3.4 Perancangan Sistem	30
3.4.1 Diagram Arsitektur Sistem.....	30
3.4.2 Use Case Diagram.....	32
3.4.3 Flowchart Sistem.....	33
3.4.4 Activity Diagram.....	35
3.4.5 Perancangan <i>Rule Engine</i>	37
3.4.6 Perancangan Alur <i>AST</i>	38
3.5 Implementasi Sistem	38

3.5.1 Implementasi <i>Visual Studio Code</i> Extension	39
3.5.2 Implementasi Parser <i>AST</i>	39
3.5.3 Implementasi <i>Rule Engine</i>	40
3.5.4 Implementasi <i>Diagnostic API</i>	40
3.6 Rencana Pengujian	41
3.6.1 Metode Pengujian (<i>Black Box Testing</i>)	41
3.6.2 Lingkungan Pengujian	41
3.6.3 Skenario Pengujian.....	42
3.6.4 Pengujian File JavaScript dari Sumber Publik.....	43
3.6.5 User Acceptance Testing (UAT).....	44
BAB IV	46
4.1 Hasil Analisis	46
4.1.1 Analisis Masalah Deteksi Bug <i>JavaScript</i>	47
4.1.2 Analisis Kebutuhan Sistem	48
4.1.3 Pemetaan Masalah dan Solusi Sistem.....	50
4.1.4 Hasil Analisis Rule Deteksi Bug	52
4.1.5 Ringkasan Hasil Analisis	53
4.2 Hasil Pengembangan Sistem.....	54
4.2.1 Hasil Pengembangan Ekstensi	55
4.2.2 Struktur Proyek Ekstensi.....	56
4.2.3 Implementasi Pemindaian Kode	57
4.2.4 Implementasi Parser <i>AST</i>	58
4.2.5 Implementasi <i>Rule Engine</i>	60
4.2.6 Implementasi <i>Diagnostic API</i>	62
4.2.7 Ringkasan Hasil Pengembangan Sistem	64

4.3 Hasil Pengujian	64
4.3.1 Hasil Pengujian Black Box	65
4.3.2 Hasil Pengujian Rule Detection	66
4.3.3 Hasil Pengujian File JavaScript dari Sumber Publik	69
4.3.4 Rekapitulasi Cakupan Pengujian 15 Rule	71
4.3.5 User Acceptance Testing (UAT).....	72
4.4 Pembahasan Sistem.....	75
4.4.1 Kesesuaian Sistem dengan Tujuan Penelitian	75
4.4.2 Kelebihan Sistem	77
4.4.3 Keterbatasan Sistem.....	78
4.4.4 Perbandingan dengan Tools Sejenis.....	79
4.4.5 Implikasi Hasil Penelitian	80
4.4.6 Ringkasan Pembahasan Sistem.....	81
4.5 Ringkasan Hasil	82
BAB V.....	85
5.1 Kesimpulan	85
5.2 Saran.....	86
DAFTAR PUSTAKA.....	88

DAFTAR TABEL

Tabel 2. 1 Penelitian Terkait <i>Static Code Analysis</i> dan Bug Detection.....	8
Tabel 2. 2 Jenis Bug <i>JavaScript</i> yang Umum Terjadi	10
Tabel 2. 3 Perbandingan <i>Static Analysis</i> dan <i>Dynamic Analysis</i>	11
Tabel 2. 4 Karakteristik <i>Rule-Based Static Analysis</i>	12
Tabel 2. 5 Contoh Rule Deteksi Bug <i>JavaScript</i>	13
Tabel 2. 6 Node <i>AST JavaScript</i>	14
Tabel 2. 7 Komponen <i>Visual Studio Code</i> Extension.....	16
Tabel 2. 8 Skenario Pengujian Rule Deteksi Bug <i>JavaScript</i>	17
Tabel 2. 9 Research Gap Penelitian.....	19
Tabel 2. 10 Komponen Framework Penelitian.....	21
Tabel 2. 11 Posisi Penelitian terhadap Penelitian Sebelumnya	23
Tabel 3. 1 Kebutuhan Perangkat Lunak	29
Tabel 3. 2 Kebutuhan Perangkat Keras	29
Tabel 3. 3 Skenario Pengujian 15 Rules.....	42
Tabel 4. 1 Analisis Kebutuhan Sistem.....	49
Tabel 4. 2 Pemetaan Masalah dan Solusi Sistem	51
Tabel 4. 3 Hasil Analisis Rule Deteksi Bug	53
Tabel 4. 4 Struktur Proyek Ekstensi	57
Tabel 4. 5 Node <i>AST</i> yang Digunakan dalam Sistem.....	60
Tabel 4. 6 Rule yang Diimplementasikan pada Sistem.....	62
Tabel 4. 7 Hasil Pengujian Rule Detection	66
Tabel 4. 8 Hasil Pengujian File <i>JavaScript</i> dari Sumber Publik	70
Tabel 4. 9 Rekapitulasi Pengujian 10 File <i>JavaScript</i> Publik.....	70
Tabel 4. 10 Rekapitulasi Cakupan Pengujian 15 Rule	72
Tabel 4. 11 Skala Likert	73
Tabel 4. 12 Instrumen Kuesioner UAT	73
Tabel 4. 13 Kategori Skor Penilaian UAT	74
Tabel 4. 14 Perbandingan Sistem dengan Tools Sejenis	79

DAFTAR GAMBAR

Gambar 3. 1 Prosedur dengan metode pengembangan <i>D&D</i>	25
Gambar 3. 2 Diagram arsitektur	31
Gambar 3. 3 Use Case Diagram	32
Gambar 3. 4 flowchart sistem ekstensi <i>Visual Studio Code</i>	34
Gambar 3. 5 Activity Diagram	36
Gambar 4. 1 Tampilan Ekstensi pada <i>Visual Studio Code</i>	55
Gambar 4. 2 Struktur Proyek Ekstensi	56
Gambar 4. 3 Implementasi Pemindaian Kode <i>JavaScript</i>	58
Gambar 4. 4 Implementasi Parser <i>AST</i>	59
Gambar 4. 5 Implementasi <i>Rule Engine</i>	61
Gambar 4. 6 Tampilan Diagnostic Warning	63
Gambar 4. 7 Tampilan Diagnostic Error	63
Gambar 4. 8 Hasil Deteksi Rule <i>no-console-log</i>	67
Gambar 4. 9 Hasil Deteksi Rule <i>no-debugger</i>	68
Gambar 4. 10 Hasil Deteksi Rule <i>no-eval</i>	68
Gambar 4. 11 Hasil Deteksi Rule <i>no-assignment-in-if</i>	69
Gambar 4. 12 Hasil Deteksi Rule <i>no-implied-eval</i>	69

STATE ISLAMIC UNIVERSITY
SUNAN KALIJAGA
YOGYAKARTA

BAB I

PENDAHULUAN

1.1. Latar Belakang

Perkembangan teknologi perangkat lunak yang pesat mendorong kebutuhan akan kualitas kode program yang tinggi untuk menjamin keandalan, keamanan, dan kemudahan pemeliharaan sistem [1]. Dalam pengembangan aplikasi web modern, *JavaScript* telah menjadi salah satu bahasa pemrograman yang paling dominan digunakan karena fleksibilitasnya serta kemampuannya dalam membangun aplikasi berbasis *client-side* maupun *server-side* [2]. *JavaScript* juga menjadi komponen utama dalam berbagai *framework* modern seperti *React*, *Next.js*, dan *Node.js* yang banyak digunakan dalam industri perangkat lunak saat ini [3], [4], [5].

Namun demikian, karakteristik *JavaScript* yang bersifat dynamic typing dan loosely typed sering kali menimbulkan berbagai potensi kesalahan dalam penulisan kode. Kesalahan tersebut dapat berupa penggunaan variabel yang tidak terdefinisi, variabel yang tidak digunakan, kesalahan logika program, maupun penggunaan praktik penulisan kode yang kurang baik (*bad practice*). Permasalahan tersebut dapat berdampak pada meningkatnya jumlah bug dalam perangkat lunak, menurunkan kualitas kode, serta meningkatkan biaya pemeliharaan sistem apabila kesalahan baru ditemukan pada tahap akhir pengembangan [2], [6].

Dalam praktik pengembangan perangkat lunak, salah satu permasalahan yang sering terjadi adalah keterlambatan deteksi bug. Bug sering kali baru ditemukan pada saat tahap pengujian atau bahkan setelah aplikasi dijalankan pada lingkungan produksi. Kondisi ini menyebabkan proses perbaikan menjadi lebih kompleks karena *developer* harus melakukan penelusuran ulang terhadap kode yang telah dibuat. Oleh karena itu, diperlukan suatu pendekatan yang mampu mendeteksi potensi kesalahan kode sejak tahap awal pengembangan [7], [1].

Salah satu pendekatan yang dapat digunakan untuk mengatasi permasalahan tersebut adalah *Static Code Analysis*, yaitu teknik analisis kode sumber tanpa perlu menjalankan program. Static code analysis memungkinkan identifikasi kesalahan

sintaks, kesalahan logika sederhana, serta pelanggaran standar penulisan kode secara otomatis. Dengan menerapkan pendekatan ini, potensi bug dapat dideteksi lebih awal sehingga developer dapat segera melakukan perbaikan sebelum kode dijalankan [8], [9].

Salah satu metode yang umum digunakan dalam static code analysis adalah *Rule-Based Static Analysis*. Pendekatan ini bekerja dengan menggunakan sekumpulan aturan (*rules*) yang dirancang untuk mendeteksi pola kode tertentu yang berpotensi mengandung kesalahan. Model ini memiliki keunggulan karena bersifat deterministik, mudah diimplementasikan, serta tidak membutuhkan komputasi yang kompleks seperti pendekatan berbasis *machine learning*. Selain itu, *rule-based analysis* juga memungkinkan *developer* untuk menambahkan aturan baru sesuai dengan kebutuhan proyek [10].

Di sisi lain, penggunaan alat bantu pengembangan (*development tools*) yang terpisah dari editor kode sering kali menyebabkan *developer* harus melakukan perpindahan konteks (*context switching*) antara proses penulisan kode dan proses analisis kualitas kode. Hal ini dapat menurunkan produktivitas *developer* karena harus menggunakan berbagai *tools* yang berbeda untuk melakukan pengecekan kualitas kode [11].

Visual Studio Code merupakan salah satu *code editor* yang paling banyak digunakan oleh *developer* perangkat lunak karena sifatnya yang ringan, fleksibel, serta memiliki ekosistem ekstensi yang luas. *Visual Studio Code* menyediakan *Extension API* yang memungkinkan *developer* untuk menambahkan berbagai fitur tambahan sesuai kebutuhan, termasuk fitur analisis kode secara otomatis. Dengan adanya dukungan ekstensi tersebut, proses analisis kode dapat dilakukan langsung di dalam editor tanpa memerlukan aplikasi tambahan [12], [11].

Meskipun telah tersedia beberapa *tools* seperti *ESLint* dan *JSHint* untuk melakukan analisis kode *JavaScript*, *tools* tersebut umumnya berfokus pada standar penulisan kode (*linting*) dan belum secara spesifik dirancang sebagai alat pembelajaran implementasi *rule-based static analysis* dalam bentuk ekstensi

mandiri. Selain itu, pengembangan ekstensi khusus dengan *rule* yang dirancang sendiri dapat memberikan fleksibilitas lebih dalam menyesuaikan kebutuhan deteksi bug tertentu [13], [14].

Pemilihan 15 rule pada penelitian ini tidak dibuat secara subjektif oleh penulis, tetapi diadaptasi dari pola kesalahan yang telah terdokumentasi pada tools analisis kode JavaScript, terutama ESLint Rules Reference dan JSHint. ESLint menyediakan daftar rules untuk mengidentifikasi dan melaporkan pola masalah pada kode JavaScript, sedangkan JSHint digunakan untuk mendeteksi error dan potensi masalah pada kode JavaScript. Beberapa rule pada sistem ini memiliki padanan konsep dengan rules pada ESLint, seperti *no-cond-assign* untuk assignment pada kondisi, *eqeqeq* untuk penggunaan operator perbandingan ketat, *use-isnan*, *no-console*, *no-debugger*, *no-self-assign*, *no-empty*, *no-constant-condition*, *no-obj-calls*, *no-alert*, *no-template-curly-in-string*, *no-throw-literal*, *no-nested-ternary*, *no-eval*, dan *no-implied-eval*. Nama rule pada ekstensi disesuaikan dengan kebutuhan implementasi, namun dasar pola deteksinya mengacu pada praktik static analysis JavaScript yang telah digunakan pada tools tersebut [10], [13], [14].

Berdasarkan permasalahan tersebut, penelitian ini mengusulkan rancang bangun sebuah ekstensi *Visual Studio Code* yang mampu mendeteksi potensi bug pada kode *JavaScript* menggunakan pendekatan *Rule-Based Static Analysis*. Ekstensi ini diharapkan mampu membantu *developer* dalam mengidentifikasi potensi kesalahan kode secara otomatis, memberikan peringatan secara langsung pada editor, serta meningkatkan kualitas kode sejak tahap awal pengembangan.

Melalui penelitian ini diharapkan dapat dihasilkan sebuah produk perangkat lunak berupa ekstensi *Visual Studio Code* yang mampu melakukan deteksi bug *JavaScript* secara otomatis berdasarkan aturan yang telah dirancang, sehingga dapat membantu meningkatkan efisiensi proses pengembangan serta kualitas perangkat lunak yang dihasilkan [7].

1.2. Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, maka rumusan masalah dalam penelitian ini adalah sebagai berikut:

Bagaimana merancang dan membangun ekstensi *Visual Studio Code* berbasis *Rule-Based Static Analysis* untuk mendeteksi potensi bug pada kode *JavaScript* secara otomatis?

1.3. Batasan Masalah

Agar penelitian ini memiliki ruang lingkup yang jelas dan terarah, maka ditetapkan beberapa batasan masalah sebagai berikut:

1. Sistem hanya menganalisis kode program berbahasa JavaScript.
2. Analisis hanya dilakukan pada file JavaScript berekstensi .js.
3. Ekstensi hanya berjalan pada editor Visual Studio Code.
4. Analisis kode dilakukan menggunakan pendekatan static analysis tanpa menjalankan program.
5. Sistem menggunakan pendekatan Rule-Based Static Analysis.
6. Analisis struktur kode dilakukan menggunakan Abstract Syntax Tree (AST).
7. Parser yang digunakan dalam sistem adalah Acorn.
8. Sistem hanya mendeteksi bug berdasarkan rule yang telah ditentukan sebelumnya.
9. Rule yang digunakan difokuskan pada 15 pola kesalahan JavaScript yang diadaptasi dari pola rule pada ESLint Rules Reference dan JSHint, yaitu no-assignment-in-if, use-eq-eq-eq, use-isnan, no-console-log, no-debugger, no-self-assign, no-empty-block, no-constant-condition, no-math-object-call, no-alert, no-template-curly-in-string, no-throw-literal, no-nested-ternary, no-eval, dan no-implied-eval.

10. Sistem tidak melakukan runtime analysis, dynamic analysis, maupun automatic bug fixing.
11. Sistem hanya menampilkan hasil deteksi dalam bentuk warning dan error melalui Diagnostic API.
12. Pengujian sistem dilakukan menggunakan black box testing, rule testing terhadap rule yang telah diimplementasikan, pengujian terhadap 10 file JavaScript dari sumber publik, serta User Acceptance Testing (UAT) sebagai evaluasi penerimaan pengguna.

1.4. Tujuan Penelitian

Tujuan dari penelitian ini adalah sebagai berikut:

1. Merancang dan membangun ekstensi *Visual Studio Code* berbasis *Rule-Based Static Analysis* untuk mendeteksi potensi bug pada kode *JavaScript*.
2. Mengembangkan fitur peringatan (*warning*) pada editor *Visual Studio Code* untuk membantu *developer* mengidentifikasi kesalahan kode.
3. Menghasilkan produk perangkat lunak berupa ekstensi *Visual Studio Code* yang dapat membantu *developer* dalam mendeteksi bug secara dini.

1.5. Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan manfaat baik secara teoritis maupun praktis.

1.5.1 Manfaat Teoritis

Secara teoritis, penelitian ini diharapkan dapat memberikan kontribusi dalam pengembangan ilmu di bidang *Software Engineering*, khususnya dalam penerapan static code analysis dan pengembangan ekstensi pada *Integrated Development Environment (IDE)*. Selain itu, penelitian ini juga diharapkan dapat menjadi referensi bagi penelitian selanjutnya yang berkaitan dengan deteksi bug perangkat lunak berbasis *rule-based analysis*.

1.5.2 Manfaat Praktis

Secara praktis, penelitian ini diharapkan dapat memberikan manfaat sebagai berikut:

1. Bagi *developer* perangkat lunak, ekstensi yang dikembangkan dapat membantu dalam mendeteksi potensi bug sejak tahap awal penulisan kode sehingga dapat meningkatkan kualitas perangkat lunak.
2. Bagi mahasiswa atau peneliti, penelitian ini dapat menjadi referensi dalam pengembangan *tools* berbasis static analysis.
3. Bagi dunia akademik, penelitian ini dapat menjadi contoh implementasi pengembangan perangkat lunak berbasis proyek dalam bidang *Software Engineering*.



BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil penelitian yang telah dilakukan, dapat disimpulkan bahwa ekstensi Visual Studio Code untuk deteksi bug JavaScript berbasis *Rule-Based Static Analysis* berhasil dirancang dan dibangun sesuai dengan tujuan penelitian. Sistem yang dikembangkan mampu melakukan pemindaian kode JavaScript, membentuk struktur *Abstract Syntax Tree* atau AST, menjalankan pemeriksaan rule melalui *rule engine*, serta menampilkan hasil deteksi dalam bentuk *warning* dan *error* pada editor Visual Studio Code.

Ekstensi yang dikembangkan mampu membantu pengguna dalam mendeteksi potensi bug JavaScript secara dini tanpa menjalankan program. Proses deteksi dilakukan dengan pendekatan analisis statis, yaitu membaca kode sumber, melakukan parsing menjadi AST, memeriksa struktur kode berdasarkan rule yang telah ditentukan, kemudian menampilkan hasil deteksi melalui Diagnostic API. Dengan demikian, sistem dapat memberikan umpan balik langsung kepada pengguna ketika ditemukan pola kode yang berpotensi bermasalah.

Hasil pengujian menunjukkan bahwa sistem dapat menjalankan fungsi utama dengan baik. Pengujian dilakukan menggunakan metode black box testing, rule testing terhadap 15 rule deteksi bug JavaScript, pengujian tambahan pada 10 file JavaScript dari sumber publik, serta UAT terhadap 10 responden. Berdasarkan hasil pengujian, seluruh rule berhasil mendeteksi pola kesalahan sesuai dengan skenario yang telah ditentukan. Dari total 15 rule yang diuji, seluruhnya menghasilkan output sesuai harapan, sehingga tingkat keberhasilan pengujian rule detection mencapai 100%. Hasil UAT memperoleh nilai rata-rata 4,53 dari skala 5 dengan kategori sangat setuju.

Rule yang berhasil diuji mencakup beberapa pola kesalahan umum pada JavaScript, seperti penggunaan assignment di dalam kondisi if, penggunaan operator `==`, perbandingan langsung terhadap NaN, penggunaan `console.log()`,

penggunaan debugger, penggunaan `eval()`, blok kode kosong, kondisi konstan, serta penggunaan string pada fungsi `setTimeout()` atau `setInterval()`. Pemilihan rule tersebut diadaptasi dari pola rule pada ESLint Rules Reference dan karakteristik deteksi pada JSHint, kemudian disesuaikan dengan ruang lingkup sistem yang dikembangkan.

Secara keseluruhan, sistem yang dikembangkan telah memenuhi kebutuhan fungsional utama, yaitu mendeteksi potensi bug JavaScript berdasarkan rule yang telah ditentukan. Pengujian pada 10 file JavaScript dari sumber publik menunjukkan bahwa seluruh rule memiliki cakupan pengujian pada contoh kode dari sumber luar, sedangkan hasil UAT menunjukkan bahwa ekstensi memperoleh penerimaan positif dari responden. Namun, sistem belum ditujukan untuk menggantikan tools profesional seperti ESLint, SonarQube, Semgrep, atau CodeQL karena ruang lingkupnya masih terbatas pada rule yang diimplementasikan dalam penelitian.

5.2 Saran

Berdasarkan hasil penelitian dan keterbatasan sistem yang telah dijelaskan, terdapat beberapa saran untuk pengembangan penelitian selanjutnya.

Pertama, sistem dapat dikembangkan dengan menambahkan jumlah rule deteksi bug JavaScript. Penambahan rule diperlukan agar sistem mampu mendeteksi lebih banyak pola kesalahan, baik yang berkaitan dengan kesalahan logika, praktik penulisan kode yang kurang baik, maupun potensi kerentanan keamanan sederhana.

Kedua, sistem dapat dikembangkan agar mendukung bahasa atau format lain seperti TypeScript, JSX, dan TSX. Pada penelitian ini, sistem masih difokuskan pada file JavaScript berekstensi `.js`, sehingga dukungan terhadap format lain dapat memperluas penggunaan ekstensi pada proyek modern yang menggunakan framework seperti React atau Next.js.

Ketiga, sistem dapat ditambahkan fitur perbaikan otomatis atau *automatic bug fixing*. Fitur ini dapat membantu pengguna tidak hanya mengetahui letak kesalahan, tetapi juga memperoleh opsi perbaikan langsung dari ekstensi sehingga proses perbaikan kode dapat dilakukan lebih cepat.



DAFTAR PUSTAKA

- [1] I. Sommerville, *Software Engineering*, 10th ed. Boston: Pearson, 2016.
- [2] Mozilla Developer Network, “JavaScript,” MDN Web Docs. Available: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- [3] Meta Open Source, “React Documentation.” Available: <https://react.dev/>
- [4] Vercel, “Next.js Documentation.” Available: <https://nextjs.org/docs>
- [5] Node.js, “Introduction to Node.js,” Node.js Learn. Available: <https://nodejs.org/learn/getting-started/introduction-to-nodejs>
- [6] Ecma International, “ECMA-262: ECMAScript Language Specification,” 2025. Available: <https://ecma-international.org/publications-and-standards/standards/ecma-262/>
- [7] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner’s Approach*, 8th ed. New York: McGraw-Hill, 2015.
- [8] B. Chess and J. West, *Secure Programming with Static Analysis*. Upper Saddle River, NJ: Addison-Wesley, 2007.
- [9] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, “Why Don’t Software Developers Use *Static Analysis* Tools to Find Bugs?” in Proc. 35th International Conference on *Software Engineering* (ICSE), 2013.
- [10] ESLint, “Rules Reference,” ESLint Documentation. Available: <https://eslint.org/docs/latest/rules/>
- [11] Microsoft, “Extension Capabilities Overview,” Visual Studio Code Extension API Documentation. Available: <https://code.visualstudio.com/api/extension-capabilities/overview>
- [12] Microsoft, “VS Code API,” Visual Studio Code Extension API Documentation. Available: <https://code.visualstudio.com/api/references/vscode-api>

- [13] *ESLint*, “Find and fix problems in your *JavaScript* code,” *ESLint* Documentation. Available: <https://eslint.org/>
- [14] *JSHint*, “*JSHint*, a *JavaScript* Code Quality Tool.” Available: <https://jshint.com/>
- [15] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix, and Y. Zhou, “Evaluating *Static Analysis* Defect Warnings on Production Software,” in Proc. 7th ACM SIGPLAN-SIGSOFT Workshop on *Program Analysis* for Software Tools and Engineering, 2007.
- [16] A. Bessey et al., “A Few Billion Lines of Code Later: Using *Static Analysis* to Find Bugs in the Real World,” *Communications of the ACM*, vol. 53, no. 2, pp. 66–75, 2010.
- [17] *Microsoft*, “Programmatic Language Features,” *Visual Studio Code Extension API* Documentation. Available: <https://code.visualstudio.com/api/language-extensions/programmatic-language-features>
- [18] M. Haverbeke, *Eloquent JavaScript*, 3rd ed. San Francisco: No Starch Press, 2018. Available: https://eloquentjavascript.net/3rd_edition/
- [19] V. Kashyap et al., “JSAI: A *Static Analysis* Platform for *JavaScript*,” in Proc. ACM SIGSOFT International Symposium on Foundations of *Software Engineering*, 2014.
- [20] *Acorn*, “*Acorn*: A small, fast, *JavaScript*-based *JavaScript* parser.” Available: <https://github.com/acornjs/acorn>
- [21] ESTree, “The ESTree Spec.” Available: <https://github.com/estree/estree>
- [22] *Esprima*, “*ECMAScript* parsing infrastructure for multipurpose analysis.” Available: <https://esprima.org/>

- [23] Microsoft, “DiagnosticCollection,” *Visual Studio Code API* Reference. Available: <https://code.visualstudio.com/api/references/vscode-api#DiagnosticCollection>
- [24] Microsoft, “Language Server Extension Guide,” *Visual Studio Code* Documentation. Available: <https://code.visualstudio.com/api/language-extensions/language-server-extension-guide>
- [25] G. J. Myers, C. Sandler, and T. Badgett, *The Art of Software Testing*, 3rd ed. Hoboken, NJ: Wiley, 2011.
- [26] SonarSource, “*SonarQube* Server Documentation.” Available: <https://docs.sonarsource.com/sonarqube-server>; *Semgrep*, “*Semgrep* Documentation.” Available: <https://semgrep.dev/docs/>
- [27] R. C. Richey and J. D. Klein, *Design and Development Research: Methods, Strategies, and Issues*. New York: Routledge, 2007.
- [28] J. Johnson, K. Boucher, Y. Connors, and J. Robinson, “Collaborative Requirements Development Methodology,” *Communications of the ACM*, vol. 44, no. 10, pp. 72–78, 2001.
- [29] npm, “npm Docs.” Available: <https://docs.npmjs.com/>
- [30] *Git*, “*Git* Documentation.” Available: <https://git-scm.com/docs/git>