

**RANCANG BANGUN ALAT ELIMINASI DEAD CODE DAN UNUSED DEPENDENSI
SEMI-OTOMATIS UNTUK PROYEK JAVASCRIPT / TYPESCRIPT**

TUGAS AKHIR

Untuk memenuhi sebagian persyaratan

Mencapai derajat sarjana S-1

Program Studi Informatika



Disusun oleh:

Dwi Rif'an Kurniawan

22106050073

STATE ISLAMIC UNIVERSITY
SUNAN KALIJAGA
YOGYAKARTA
PROGRAM STUDI 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-2091/U.n.02/DST/PP.00.9/08/2026

Tugas Akhir dengan judul : Rancang Bangun Alat Eliminasi Dead Code dan Unused Dependensi Semi-otomatis untuk Proyek JavaScript / TypeScript

yang dipersiapkan dan disusun oleh:

Nama : DWI RIFAN KURNIAWAN
Nomor Induk Mahasiswa : 22106050073
Telah diujikan pada : Rabu, 19 Agustus 2026
Nilai ujian Tugas Akhir : A

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

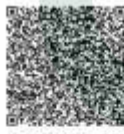
TIM UJIAN TUGAS AKHIR



Khusni Sidang

Dr. Agung Fatmanto, S.Si., M.Kom.
SIGNED

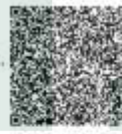
Valid ID: fad8a5ad1122a



Penguji I

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

Valid ID: fad8a5ad1122a



Penguji II

Muhammad Galih Wicaksono, M.T.
SIGNED

Valid ID: fad8a5ad1122a



Yogyakarta, 19 Agustus 2026

UIN Sunan Kalijaga

Dekan Fakultas Sains dan Teknologi

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

Valid ID: fad8a5ad1122a

LEMBAR PERNYATAAN

LEMBAR PERNYATAAN KEASLIAN

Yang bertanda tangan di bawah ini:

Nama : Dwi Rif'an Kurniawan
NIM : 22106050073
Program Studi : Informatika
Fakultas : Sains dan Teknologi

Menyatakan dengan sesungguhnya, bahwa skripsi saya yang berjudul "**Rancang Bangun Alat Eliminasi Dead Code dan Unused Dependensi Semi-otomatis untuk Proyek JavaScript / TypeScript**" 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. Sebagai salah satu syarat untuk memperoleh gelar Sarjana Program Studi Informatika pada Fakultas Sains dan Teknologi, Universitas Islam Negeri Sunan Kalijaga.

Yogyakarta, 14 Agustus 2026
Yang membuat pernyataan,



Dwi Rif'an Kurniawan
22106050073

LEMBAR PERSETUJUAN TUGAS AKHIR



Universitas Islam Negeri Sunan Kalijaga



FM-UINSK-BM-05-03/R0

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 : Dwi Rif'an Kurniawan
NIM : 22106050073
Judul Skripsi : Rancang Bangun Alat Eliminasi Dead Code dan Unused
Dependensi Semi-otomatis untuk Proyek JavaScript /
TypeScript

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 berharap agar skripsi / tugas akhir Saudara dapat segera dimunaqsyahkan. Atas perhatiannya saya ucapkan terima kasih.

Wassalamu 'alaikum wr. wb.

Yogyakarta, 14 Agustus 2026

Pembimbing,

Dr. Agung Fatwanto, S.Si., M.Kom.
NIP. 19770103 200501 1 003

ABSTRAK

Perkembangan proyek JavaScript dan TypeScript dapat meninggalkan *dead code* dan *unused dependency* yang meningkatkan kompleksitas pemeliharaan serta risiko kesalahan. Penelitian ini bertujuan membangun alat berbasis *Command Line Interface* (CLI) untuk mendeteksi dan mengeliminasi kode serta dependensi yang tidak digunakan secara semi-otomatis. Sistem menerapkan analisis statis berbasis *Abstract Syntax Tree* (AST) untuk memetakan deklarasi dan referensi kode serta graf dependensi untuk menelusuri hubungan antarberkas. Proses eliminasi dilengkapi pemilihan kandidat, konfirmasi pengguna, pencadangan, dan mekanisme pemulihan. Sistem dikembangkan menggunakan metode *Extreme Programming* secara iteratif. Evaluasi dilakukan pada empat repositori sumber terbuka menggunakan anomali alami dan spesimen injeksi terkontrol, kemudian hasilnya dibandingkan dengan *ground truth* hasil pemeriksaan manual. Dari 106 kasus pengujian, sistem menghasilkan 100 *True Positive*, 1 *False Positive*, dan 5 *False Negative*, dengan *precision* 99,01%, *recall* 95,24%, dan *F1-score* 97,09%. Seluruh rangkaian validasi setelah remediasi berhasil dijalankan tanpa kegagalan atau *crash*. Hasil ini menunjukkan bahwa pendekatan berbasis AST memiliki kinerja yang baik, meskipun peninjauan pengguna tetap diperlukan untuk pola dinamis dan perilaku yang bergantung pada konteks *runtime*.

Kata kunci: *Abstract Syntax Tree*, *dead code*, *unused dependency*, analisis statis, *Command Line Interface*.

STATE ISLAMIC UNIVERSITY
SUNAN KALIJAGA
YOGYAKARTA

ABSTRACT

The development of JavaScript and TypeScript projects can leave behind dead code and unused dependencies, increasing maintenance complexity and the risk of errors. This study aims to develop a Command Line Interface (CLI)-based tool for semi-automatically detecting and eliminating unused code and dependencies. The system applies Abstract Syntax Tree (AST)-based static analysis to map code declarations and references, along with a dependency graph to trace relationships between files. The elimination process includes candidate selection, user confirmation, backup, and recovery mechanisms. The system was developed iteratively using the Extreme Programming method. Evaluation was conducted on four open-source repositories using naturally occurring anomalies and controlled injected specimens, with the results compared against ground truth established through manual inspection. From 106 test cases, the system produced 100 True Positives, 1 False Positive, and 5 False Negatives, achieving a precision of 99.01%, recall of 95.24%, and F1-score of 97.09%. All post-remediation validation procedures were completed without failures or crashes. These results demonstrate that the AST-based approach performs well, although user review remains necessary for dynamic patterns and behavior that depend on runtime context.

Keywords: *Abstract Syntax Tree, dead code, unused dependency, static analysis, Command Line Interface.*

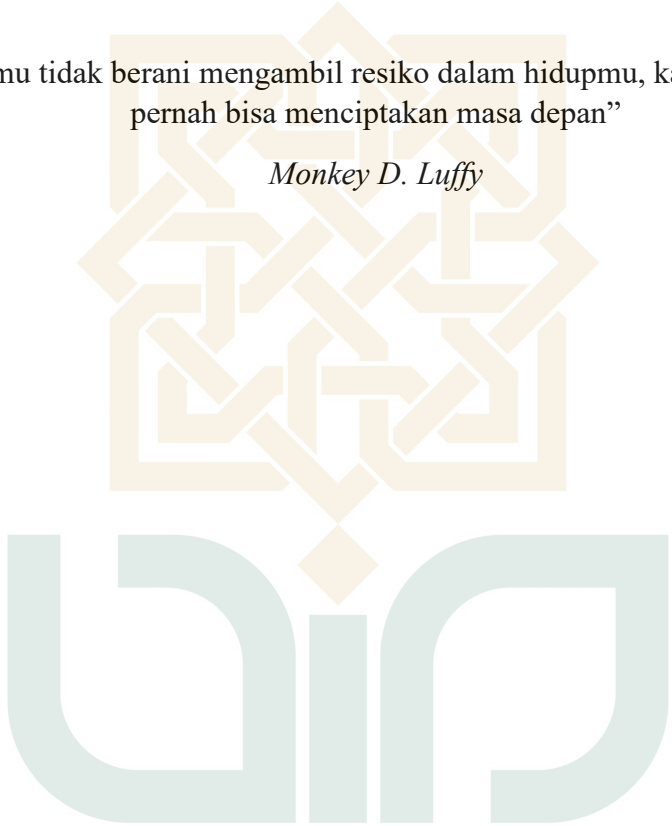
MOTTO

“Sirno dalane pati nur sifat, luber tanpo kebek”

Hilangkan rasa iri dengki jadilah orang bermanfaat, sederhana dan tetap rendah hati

“Jika kamu tidak berani mengambil resiko dalam hidupmu, kamu tidak akan pernah bisa menciptakan masa depan”

Monkey D. Luffy



STATE ISLAMIC UNIVERSITY
SUNAN KALIJAGA
YOGYAKARTA

HALAMAN PERSEMBAHAN

Alhamdulillah Rabbil 'Alamin, dengan penuh rasa syukur ke hadirat Allah SWT atas rahmat, pertolongan, dan kemudahan-Nya, Tugas Akhir ini dapat diselesaikan. Semoga karya sederhana ini memberikan manfaat bagi pengembangan ilmu pengetahuan serta bernilai sebagai amal kebaikan. Dengan segenap kerendahan hati, Tugas Akhir ini saya persembahkan kepada:

1. Kedua orang tua tercinta atas kasih sayang, doa, dukungan, nasihat, dan pengorbanan yang menjadi kekuatan bagi saya dalam menyelesaikan pendidikan ini.
2. Seluruh keluarga yang senantiasa memberikan perhatian, semangat, dan dukungan selama perkuliahan serta penyusunan Tugas Akhir.
3. Cintina Dewi Masitah, *partner* yang senantiasa menemani, mendengarkan, dan memberikan semangat hingga Tugas Akhir ini dapat diselesaikan.
4. Diri saya sendiri yang telah berusaha, bertahan, dan terus belajar hingga mampu menyelesaikan Tugas Akhir ini.

Terima kasih atas setiap doa dan dukungan yang telah diberikan. Semoga Allah SWT membalas seluruh kebaikan dengan rahmat dan keberkahan-Nya.

STATE ISLAMIC UNIVERSITY
SUNAN KALIJAGA
YOGYAKARTA

KATA PENGANTAR

Alhamdulillah Rabbil 'Alamin, segala puji dan syukur penulis panjatkan ke hadirat Allah SWT atas rahmat-Nya sehingga Tugas Akhir berjudul “Rancang Bangun Alat Eliminasi Dead Code dan Unused Dependensi Semi-Otomatis untuk Proyek JavaScript/TypeScript” dapat diselesaikan. Selawat serta salam semoga senantiasa tercurah kepada Nabi Muhammad saw., beserta keluarga, sahabat, dan umatnya.

Tugas Akhir ini disusun sebagai salah satu syarat memperoleh gelar Sarjana pada Program Studi Informatika, Fakultas Sains dan Teknologi, Universitas Islam Negeri Sunan Kalijaga Yogyakarta. Penelitian ini mengembangkan DeadKiller CLI untuk mendeteksi dan membantu mengeliminasi dead code serta unused dependency. Penyelesaian Tugas Akhir ini tidak terlepas dari bantuan dan dukungan berbagai pihak. Oleh karena itu, penulis menyampaikan terima kasih kepada:

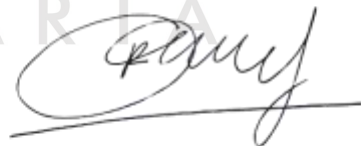
1. Prof. Noorhaidi Hasan, S.Ag., M.A., M.Phil., Ph.D., selaku Rektor Universitas Islam Negeri Sunan Kalijaga Yogyakarta.
2. Prof. Dr. Dra. Hj. Khurul Wardati, M.Si., selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Sunan Kalijaga Yogyakarta.
3. Dr. Muhammad Mustakim, S.T., M.T., selaku Ketua Program Studi Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Sunan Kalijaga Yogyakarta.
4. Dr. Agung Fatwanto, S.Si., M.Kom., selaku dosen pembimbing Tugas Akhir atas waktu, bimbingan, arahan, dan masukan selama penelitian dan penyusunan Tugas Akhir.
5. Seluruh dosen Program Studi Informatika yang telah memberikan ilmu, pengalaman, dan wawasan selama masa perkuliahan.
6. Seluruh tenaga kependidikan dan staf Fakultas Sains dan Teknologi yang telah membantu keperluan akademik dan administrasi penulis.

7. Kedua orang tua tercinta atas kasih sayang, doa, nasihat, pengorbanan, dan dukungan yang tidak pernah terputus.
8. Seluruh keluarga yang senantiasa memberikan perhatian, semangat, dan dukungan selama perkuliahan serta penyusunan Tugas Akhir.
9. Teman-teman Program Studi Informatika angkatan 2022 atas bantuan, kebersamaan, dan semangat selama masa perkuliahan.
10. Seluruh teman-teman Matdis, Jual Beli Musang, dan Kos Griya Rizky yang telah memberikan dukungan, semangat, kebersamaan, serta bantuan selama proses penyelesaian Tugas Akhir ini.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki keterbatasan. Oleh karena itu, kritik dan saran yang membangun sangat diharapkan sebagai bahan perbaikan dan pengembangan. Semoga Tugas Akhir ini bermanfaat bagi pembaca, pengembang perangkat lunak, dan penelitian selanjutnya dalam bidang analisis statis, dead code, serta pengelolaan dependensi pada proyek JavaScript dan TypeScript.

Yogyakarta, 14 Agustus 2026

Penulis,



Dwi Rif'an Kurniawan
22106050073

DAFTAR ISI

LEMBAR PENGESAHAN	ii
LEMBAR PERNYATAAN	iii
LEMBAR PERSETUJUAN TUGAS AKHIR.....	iv
ABSTRAK	v
ABSTRACT	vi
MOTTO	vii
HALAMAN PERSEMBAHAN	viii
KATA PENGANTAR.....	ix
DAFTAR ISI.....	xi
DAFTAR TABEL	xiv
DAFTAR GAMBAR.....	xvi
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	5
1.3 Batasan Masalah.....	6
1.4 Tujuan Penelitian	7
1.5 Manfaat Penelitian	8
1.5.1 Manfaat Teoritis	8
1.5.2 Manfaat Praktis	8
BAB II KAJIAN PUSTAKA.....	10
2.1 Tinjauan Pustaka	10
2.2 Landasan Teori.....	14
2.2.1 Ekosistem Node.js, Manajemen Dependensi, dan <i>Software bloat</i>	14
2.2.2 JavaScript dan ECMAScript	15
2.2.3 TypeScript.....	15
2.2.4 Dead Code dan Unused Dependencies	16
2.2.5 Analisis Kode Statis (<i>Static Code Analysis</i>)	16
2.2.6 Abstract Syntax Tree (AST).....	17
2.2.7 <i>Command Line Interface (CLI)</i>	17
2.2.8 Algoritma Breadth-First Search (BFS)	18

2.2.9	Peta Ketergantungan (<i>Dependency Graph</i>)	19
2.2.10	Visualisasi Kode (<i>Code Visualization</i>)	19
2.2.11	Prinsip Clean Code.....	19
2.2.12	Software Development Life Cycle (SDLC).....	20
2.2.13	Extreme Programming (XP)	21
2.2.14	Evaluasi Akurasi dengan Confusion Matrix	22
BAB III METODE PENGEMBANGAN SISTEM		23
3.1	Metode Pengembangan Sistem	23
3.2	Tahapan Pelaksanaan Rekayasa Sistem	24
3.2.1	Tahap Perencanaan (<i>planning</i>).....	24
3.2.2	Tahap Perancangan (<i>Design</i>)	25
3.2.3	Tahap Pengkodean (<i>coding</i>).....	25
3.2.4	Tahap Pengujian (<i>Testing</i>)	26
3.3	Metode Evaluasi Kinerja Deteksi.....	27
BAB IV PERANCANGAN DAN IMPLEMENTASI SISTEM		29
4.1	Spesifikasi Kebutuhan Sistem.....	29
4.1.1	Kebutuhan Fungsional	29
4.1.2	Kebutuhan Non Fungsional.....	30
4.1.3	User Story dan Kriteria Penerimaan (<i>acceptance test criteria</i>)	30
4.1.4	Rencana Iterasi Pengembangan.....	31
4.2	Lingkungan Pengembangan dan Pemilihan Teknologi	32
4.2.1	Spesifikasi Perangkat Keras (<i>Hardware</i>).....	33
4.2.2	Lingkungan Eksekusi Dasar.....	33
4.2.3	Core Engine & AST Processing.....	34
4.2.4	Dashboard Pelaporan Visual.....	34
4.2.5	Manajemen Berkas & Lapisan Antarmuka Terminal	34
4.2.6	Alat Pendukung Pengembangan	35
4.3	Perancangan Sistem	35
4.3.1	Perancangan Arsitektur Sistem	36
4.3.2	Perancangan Struktur Data.....	37
4.3.3	Perancangan Algoritma Inti	42
4.3.4	Perancangan Alur Sistem	44
4.4	Implementasi Iterasi	45
4.4.1	Iterasi 1 : Pembangunan Core Parser	45

4.4.2 Iterasi 2 : Pengembangan Mesin Pemetaan & Analisis (Graph Builder & Analyzer)	51
4.4.3 Iterasi 3 : Pembangunan Eliminator dan Mekanisme Penghapusan Aman 69	
4.4.4 Iterasi 4 : Pengembangan Modul Antarmuka dan Pelaporan (CLI & Reporter)	76
4.5 Pengujian dan Evaluasi Sistem	85
4.5.1 Rancangan dan Tahapan Pengujian	85
4.5.2 Hasil Pengujian	89
BAB V KESIMPULAN DAN SARAN	96
5.1 Kesimpulan	96
5.2 Saran.....	97
DAFTAR PUSTAKA.....	100
LAMPIRAN.....	103

DAFTAR TABEL

Tabel 2.1 Ringkasan Tinjauan Pustaka	12
Tabel 4.1 User Story dan Kriteria Penerimaan	31
Tabel 4.2 Rencana Iterasi Pengembangan	32
Tabel 4.3 Spesifikasi Perangkat Keras.....	33
Tabel 4.4 Lingkungan Eksekusi Dasar.....	33
Tabel 4.5 Komponen Representasi Abstract Syntax Tree	38
Tabel 4.6 Atribut DeclarationInfo pada Lexical Scope Record.....	39
Tabel 4.7 Komponen Utama Project Dependency Graph.....	40
Tabel 4.8 Atribut Utama Dead Node Object.....	41
Tabel 4.9 Daftar Prioritas Tugas Iterasi 1	46
Tabel 4.10 Cakupan Pengujian Iterasi 1	47
Tabel 4.11 Perbandingan Benchmark Parser Tanpa dan dengan ParseCache	50
Tabel 4.12 Hasil Penyelesaian Tugas Iterasi 1.....	51
Tabel 4.13 Daftar Prioritas Tugas Iterasi 2	52
Tabel 4.14 Cakupan Pengujian Iterasi 2	53
Tabel 4.15 Hasil Penyelesaian Tugas Iterasi 2.....	68
Tabel 4.16 Daftar Prioritas Tugas Iterasi 3	69
Tabel 4.17 Cakupan Pengujian Iterasi 3	70
Tabel 4.18 Hasil Penyelesaian Tugas Iterasi 3.....	76
Tabel 4.19 Daftar Prioritas Tugas Iterasi 4	77
Tabel 4.20 Cakupan Pengujian Iterasi 4	77
Tabel 4.21 Perintah dan Fungsi DeadKiller CLI	78
Tabel 4.22 Hasil Penyelesaian Tugas Iterasi 4.....	84
Tabel 4.23 Repositori Pengujian	85
Tabel 4.24 Kategori Spesimen Injeksi	86
Tabel 4.25 Definisi Klasifikasi Hasil Deteksi.....	88
Tabel 4.26 Makna Status Rekomendasi Tindakan.....	89
Tabel 4.27 Rekapitulasi Hasil Pengujian Proyek Nyata	89
Tabel 4.28 Rincian Rekomendasi dan Status Eksekusi Proyek Nyata	90
Tabel 4.29 Rekapitulasi Hasil Pengujian Spesimen Injeksi.....	92
Tabel 4.30 Rincian Rekomendasi dan Status Eksekusi Spesimen Injeksi.....	92
Tabel 4.31 Rekapitulasi Keseluruhan Pengujian Sistem	94

Tabel A.1 Skenario Pengujian Iterasi 1.....	103
Tabel B.1 Skenario Pengujian Iterasi 2.....	103
Tabel C.1 Skenario Pengujian Iterasi 3.....	104
Tabel D.1 Skenario Pengujian Iterasi 4.....	105
Tabel E.1 Ground Truth Proyek Nyata vanilla-node-rest-api.....	105
Tabel E.2 Hasil Pemindaian Proyek Nyata vanilla-node-rest-api.....	107
Tabel E.3 Ground Truth Spesimen Injeksi vanilla-node-rest-api	107
Tabel E.4 Hasil Pemindaian Spesimen Injeksi vanilla-node-rest-api	108
Tabel E.5 Ground Truth Proyek Nyata CAC	110
Tabel E.6 Hasil Pemindaian Proyek Nyata CAC.....	110
Tabel E.7 Ground Truth Spesimen Injeksi CAC	110
Tabel E.8 Hasil Pemindaian Spesimen Injeksi CAC	112
Tabel E.9 Ground Truth Proyek Nyata dotenv	113
Tabel E.10 Hasil Pemindaian Proyek Nyata dotenv	114
Tabel E.11 Ground Truth Rencana Injeksi dotenv.....	114
Tabel E.12 Hasil Pemindaian Spesimen Injeksi dotenv.....	115
Tabel E.13 Ground Truth Proyek Nyata node-express-boilerplate.....	116
Tabel E.14 Hasil Pemindaian Proyek Nyata node-express-boilerplate	118
Tabel E.15 Ground Truth Spesimen Injeksi node-express-boilerplate	119
Tabel E.16 Hasil Pemindaian Spesimen Injeksi node-express-boilerplate.....	120

DAFTAR GAMBAR

Gambar 3.1 Alur Pengembangan Sistem Extreme Programming [22].	24
Gambar 4.1 Diagram Arsitektur Pipeline Sistem	37
Gambar 4.2 Class Diagram Struktur Data Sistem	38
Gambar 4.3 Alur Transformasi Struktur Data	42
Gambar 4.4 Diagram Alur Sistem	45
Gambar 4.5 Implementasi Fungsi <code>parseCode()</code>	49
Gambar 4.6 Hasil Pengujian Modul Core Parser	50
Gambar 4.7 Activity Diagram Penelusuran AST dan Resolusi Lingkup	56
Gambar 4.8 Activity Diagram Analisis Alur Data dan Kontrol	57
Gambar 4.9 Activity Diagram Keterlacakan Berkas	59
Gambar 4.10 Implementasi AST Analyzer	62
Gambar 4.11 Implementasi Entry Point Finder	63
Gambar 4.12 Implementasi Graph Builder Berbasis BFS	64
Gambar 4.13 Implementasi Dependency Analyzer	65
Gambar 4.14 Implementasi Rule Engine	66
Gambar 4.15 Implementasi Integration Pipeline	67
Gambar 4.16 Hasil Pengujian Graph Builder dan Analyzer	68
Gambar 4.17 Implementasi Code Cleaner	73
Gambar 4.18 Implementasi Dependency Cleaner	74
Gambar 4.19 Implementasi Eliminator Pipeline	75
Gambar 4.20 Hasil Pengujian Modul Eliminator	75
Gambar 4.21 Activity Diagram Routing dan Eksekusi Perintah CLI	79
Gambar 4.22 Rancangan Dashboard Visualisasi HTML	80
Gambar 4.23 Implementasi Ekosistem Perintah CLI	81
Gambar 4.24 Antarmuka Interactive Wizard	82
Gambar 4.25 Dashboard Visualisasi HTML DeadKiller	83
Gambar 4.26 Hasil Pengujian Modul CLI dan Reporter	84
Gambar F.1 Hasil Benchmark Parser Tanpa dan dengan ParseCache	121

BAB I

PENDAHULUAN

1.1 Latar Belakang

Ekosistem pengembangan perangkat lunak modern, khususnya yang berbasis JavaScript berbasis spesifikasi ECMAScript dan superset TypeScript pada lingkungan runtime Node.js, telah mengalami transformasi signifikan berkat adopsi luas manajer paket seperti NPM (*Node Package Manager*) [1]. Ketersediaan ribuan pustaka pihak ketiga (*third-party libraries*) memberikan efisiensi bagi pengembang untuk mempercepat siklus produksi aplikasi tanpa harus membangun fungsi dari nol. Namun, kemudahan integrasi ini sering kali menjadi pedang bermata dua, di mana praktik penambahan dependensi yang tidak teregulasi dengan ketat kerap memicu fenomena *software bloat*, yaitu kondisi ketika aplikasi terbebani oleh kode-kode yang sebenarnya tidak diperlukan untuk fungsionalitas intinya. Penumpukan tersebut dapat berasal dari dependensi langsung maupun transitif yang tidak lagi digunakan (pustaka yang ditarik secara otomatis), yang memperbesar struktur pohon dependensi secara signifikan [2]. Akibatnya, proyek perangkat lunak modern cenderung memiliki ukuran yang jauh lebih besar daripada yang seharusnya dan pada akhirnya menciptakan kompleksitas berlebih serta tumpukan kode yang sulit dikelola secara efektif.

Dampak dari inefisiensi struktur kode ini sangat terasa pada lingkungan *Continuous Integration* (CI), yang menjadi tulang punggung pengembangan modern. Penelitian oleh Weeraddana et al. mengungkapkan fakta empiris yang mengkhawatirkan, yaitu sekitar 55,88% waktu komputasi pada proses *build* terbuang sia-sia hanya untuk mengunduh, menginstal, dan memproses dependensi yang sebenarnya tidak digunakan oleh aplikasi [3]. Pemborosan ini tidak hanya merugikan dari segi waktu, tetapi juga berdampak pada membengkaknya penggunaan *bandwidth* jaringan untuk mentransmisikan paket yang redundan [4]. Selain itu, penumpukan komponen yang tidak diperlukan dapat menyebabkan

ukuran artefak aplikasi menjadi lebih besar, sehingga membebani kapasitas penyimpanan (*storage*) [5], serta meningkatkan konsumsi energi pada infrastruktur server [4]. Selain memperlambat waktu rilis aplikasi ke pasar (*time-to-market*), hal ini juga secara drastis meningkatkan biaya operasional infrastruktur *cloud* yang harus ditanggung oleh organisasi.

Di sisi lain, komunitas pengembang sebenarnya memiliki kesadaran tinggi akan bahaya penumpukan ini. Studi oleh Jaisri et al. (2025) terhadap ekosistem NPM menemukan bahwa tren pengembang saat ini mulai bergerak menuju pembuatan pustaka mandiri (*self-contained*), di mana Studi tersebut menemukan bahwa 40,42% pustaka mandiri mencapai kondisi tersebut setelah pengembang menghapus dependensi yang dianggap membebani. Langkah ini didorong oleh motivasi utama untuk meningkatkan performa dan memangkas ukuran aplikasi [1].

Selain berdampak pada performa dan biaya operasional, permasalahan ini juga memiliki implikasi serius terhadap aspek keamanan yang kritikal. Koishybayev dan Kapravelos menyoroti bahwa setiap dependensi tambahan yang dimasukkan ke dalam proyek secara otomatis memperluas *attack surface* (permukaan serangan) aplikasi [6]. Berdasarkan analisis mereka, ancaman ini sangat nyata, di mana ditemukan 119.433 paket Node.js yang secara tidak sadar membawa dependensi dengan kerentanan (*vulnerability*) aktif ke dalam aplikasi [6]. Kode mati (*dead code*) atau dependensi tak terpakai yang tertinggal di dalam paket produksi sering kali luput dari pemantauan pembaruan keamanan, namun tetap mengandung potensi kerentanan yang dapat dieksploitasi oleh pihak tidak bertanggung jawab. Bahkan, keberadaan dependensi transitif sering kali menjadi titik lemah yang mengancam rantai pasok perangkat lunak (*software supply chain*) secara diam-diam [2]. Lebih jauh lagi, peringatan kerentanan pada dependensi yang sebenarnya tidak digunakan ini sering memicu *notification fatigue* (kelelahan notifikasi) dan menyita waktu pengembang saat proses audit keamanan [2]. Oleh karena itu, identifikasi dependensi yang tidak digunakan dapat menjadi salah satu langkah pendukung dalam mengurangi komponen yang tidak diperlukan pada suatu proyek.

Dalam konteks yang lebih fundamental, keberadaan kode sampah ini bertentangan dengan prinsip kualitas perangkat lunak. Boomsma mendefinisikan *dead code* sebagai bagian dari kode sumber yang dapat dieksekusi namun tidak pernah dipanggil selama operasi sistem berjalan [7]. Penumpukan *dead code* dapat menambah kompleksitas yang tidak diperlukan, memicu munculnya *code smells*, dan menyulitkan proses pemahaman serta pemeliharaan kode dalam jangka Panjang [8]. Oleh karena itu, eliminasi *dead code* menjadi penting untuk mengurangi komponen yang tidak lagi memberikan kontribusi terhadap fungsi program serta menjaga basis kode tetap lebih sederhana dan mudah dipelihara.

Meskipun urgensi penyelesaian masalah ini sangat tinggi, metode deteksi dan eliminasi yang tersedia saat ini masih memiliki keterbatasan signifikan. Pendekatan deteksi secara manual dinilai tidak praktis untuk proyek berskala besar [9] dan sangat berisiko menimbulkan *human error* di mana satu kesalahan keputusan penghapusan dapat mematahkan fungsionalitas sistem di tahap produksi akibat kompleksitas ketergantungan antar-modul [10]. Di sisi lain, metode otomatis seperti perangkat Lacuna yang dikembangkan oleh Malavolta et al. mampu memberikan hasil deteksi kode mati yang komprehensif dengan memanfaatkan pendekatan analisis dinamis [4]. Namun, metode yang mengandalkan analisis dinamis pada dasarnya memiliki keterbatasan karena membebani sistem dengan menimbulkan *overhead* eksekusi *runtime* yang signifikan, terutama pada proyek dengan cakupan test suite yang luas dan dependensi kompleks [2]. Sementara itu, terobosan terbaru menggunakan *Large Language Models* (LLM) untuk eliminasi kode mati memerlukan sumber daya komputasi yang relatif tinggi (misalnya GPU/akses API berbayar) serta pipeline integrasi yang tidak sederhana, sehingga kurang efisien untuk dijadikan utilitas harian pengembang [11]. JavaScript dan TypeScript dipilih sebagai objek penelitian karena penelitian ini berfokus pada ekosistem Node.js dan NPM yang memiliki penggunaan modul dan dependensi pihak ketiga secara luas [2]. JavaScript modern melalui *ECMAScript Modules* menyediakan deklarasi *import* dan *export* yang membentuk relasi antar-modul dan dapat dianalisis secara statis [12], sedangkan TypeScript memperluas JavaScript

dengan sistem tipe serta konstruksi seperti *interface*, *type alias*, dan *type-only import/export* [13], [14]. Selain itu, penelitian Liu et al. masih berfokus pada paket berbasis CommonJS [2], sehingga terdapat ruang untuk mengevaluasi pendekatan analisis statis pada JavaScript modern dan TypeScript. Oleh karena itu, pendekatan analisis statis menggunakan parser Abstract Syntax Tree (AST), yaitu representasi struktur sintaksis kode sumber dalam bentuk pohon hierarkis yang memungkinkan mesin memahami logika program tanpa perlu mengeksekusinya, menjadi solusi yang sangat relevan [15]. Parser AST terbukti mampu mengekstraksi dan membedah struktur semantik dari varian TypeScript dan JavaScript secara langsung, memungkinkan pemetaan grafik ketergantungan (dependency graph) yang presisi tanpa memerlukan proses kompilasi awal maupun eksekusi program [6].

Untuk menjembatani kesenjangan tersebut, penelitian ini mengusulkan pengembangan alat berbasis *Command Line Interface* (CLI) sebuah antarmuka interaktif berbasis teks yang memungkinkan pengembang mengeksekusi perintah optimasi secara langsung dari terminal [16] yang memanfaatkan analisis statis berbasis Abstract Syntax Tree (AST) serta dilengkapi dengan fitur visualisasi relasi antar-modul. AST digunakan untuk mengubah kode sumber menjadi representasi pohon sintaksis yang dapat ditelusuri guna mengidentifikasi deklarasi, referensi, dan struktur penggunaan elemen kode [15]. Selanjutnya, hubungan impor dan ekspor antarberkas dipetakan dalam bentuk dependency graph agar keterhubungan setiap modul di dalam proyek dapat dianalisis secara terstruktur tanpa mengeksekusi program target pada saat runtime [4].

Fitur visualisasi interaktif digunakan untuk menyajikan struktur proyek, relasi antarberkas, serta daftar temuan dead code dan unused dependencies agar lebih mudah dipahami oleh pengguna. Informasi tersebut diharapkan dapat membantu pengguna melakukan peninjauan sebelum mengambil keputusan penghapusan, mengingat kesalahan dalam menghapus kode atau dependensi yang sebenarnya masih digunakan dapat menyebabkan perubahan perilaku maupun gangguan terhadap fungsi sistem [7], [9], [10].

Alat yang dikembangkan dirancang untuk mengotomatisasi proses deteksi dan memberikan rekomendasi eliminasi dead code serta unused dependencies pada proyek JavaScript ES6+ dan TypeScript. Proses eliminasi dilakukan secara semi-otomatis melalui penyajian hasil analisis dan konfirmasi pengguna sebelum perubahan diterapkan. Kinerja metode deteksi selanjutnya dievaluasi menggunakan Confusion Matrix melalui pengukuran Precision, Recall, dan F1-Score untuk mengetahui tingkat ketepatan sistem dalam mengidentifikasi elemen kode yang tidak digunakan.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah yang telah diuraikan, dapat diidentifikasi beberapa permasalahan yang cukup krusial dalam pengembangan proyek JavaScript/TypeScript saat ini, yaitu:

1. Masih terbatasnya alat bantu (*tool*) yang praktis dan ringan di lingkungan pengembangan lokal yang mampu mendeteksi *dead code* tanpa memerlukan eksekusi program (*runtime*) yang berat.
2. Efektivitas yang rendah dari metode pemeriksaan manual yang memakan waktu lama dan sangat rentan terhadap *human error*, akibat ketiadaan representasi visual struktur kode serta metode dinamis yang membebani kinerja sistem.
3. Sistem keamanan pada manajemen dependensi proyek yang belum dirancang secara optimal untuk meminimalisir *attack surface*, sehingga kode sampah sering kali menjadi celah masuknya ancaman siber.

Berdasarkan identifikasi masalah tersebut, pertanyaan penelitian yang diajukan adalah:

1. Bagaimana merancang dan membangun *tool* berbasis *Command Line Interface* (CLI) dengan menggunakan metode analisis statis *Abstract Syntax Tree* (AST) beserta antarmuka pelaporan visualisasi peta relasi antar-modul

untuk mengotomatisasi deteksi *dead code* dan *unused dependencies* pada proyek JavaScript/TypeScript pada lingkungan Node.js secara efisien?

2. Bagaimana kinerja metode analisis statis berbasis Abstract Syntax Tree (AST) dalam mendeteksi *dead code* dan *unused dependencies* serta meminimalkan risiko kesalahan identifikasi dan penghapusan pada struktur proyek JavaScript/TypeScript yang dinamis?

1.3 Batasan Masalah

Agar pembahasan dalam penelitian ini lebih terarah dan tidak menyimpang dari tujuan utama yang telah ditetapkan, maka penulis membatasi ruang lingkup masalah sebagai berikut:

1. Perangkat lunak dikembangkan sebagai utilitas berbasis *Command Line Interface* (CLI) yang berjalan pada lingkungan Node.js. Evaluasi utama difokuskan pada proyek yang menggunakan NPM.
2. Analisis dilakukan terhadap kode sumber JavaScript modern (ES6+) dan TypeScript.
3. Pembahasan utama mencakup pelacakan cakupan leksikal serta relasi modul menggunakan ES Modules melalui deklarasi `import` dan `export`. Dukungan terhadap sistem modul lain dan perilaku khusus *framework* tidak menjadi objek evaluasi utama.
4. Sistem menerapkan analisis statis berbasis *Abstract Syntax Tree* (AST) tanpa mengeksekusi program target sehingga perilaku yang hanya muncul saat *runtime* tidak dapat dianalisis secara menyeluruh.
5. Proses eliminasi dilakukan secara semi-otomatis dengan menampilkan laporan dan meminta konfirmasi pengguna sebelum perubahan diterapkan untuk mengurangi risiko *false positive*.
6. Analisis intra-file mencakup deteksi variabel, fungsi, parameter, dan impor yang tidak digunakan; *write-only assignment*; *dead store*; *unreachable*

code; kondisi duplikat atau kontradiktif; serta fungsi internal yang tidak memiliki jalur pemanggilan.

7. Analisis lintas berkas dilakukan melalui *dependency graph* untuk mendeteksi *dead file*, *unused export*, dan *unused dependency*.
8. Data pengujian diperoleh dari repositori sumber terbuka di GitHub yang dipilih menggunakan teknik *purposive sampling* berdasarkan kesesuaian dengan ruang lingkup penelitian.
9. Evaluasi difokuskan pada pengujian fungsional serta pengukuran *precision*, *recall*, dan *F1-score*. Pengukuran waktu dan memori digunakan sebagai data pendukung, sedangkan audit keamanan dan *benchmark* komparatif terhadap waktu *build* tidak dilakukan.
10. Deteksi *unused class method* bersifat heuristik dan tidak mencakup secara menyeluruh penggunaan melalui *inheritance*, *reflection*, *dependency injection*, akses properti terkomputasi, atau pemanggilan dinamis. Temuan tersebut hanya direkomendasikan untuk ditinjau dan tidak dihapus secara otomatis, sedangkan *unused object properties* tidak termasuk dalam cakupan deteksi.

1.4 Tujuan Penelitian

Berpijak pada rumusan masalah yang telah dikemukakan, tujuan utama dari penelitian ini adalah:

1. Merancang dan Membangun Artifak Perangkat Lunak : Menghasilkan sebuah perangkat lunak utilitas berbasis *Command Line Interface* (CLI) di lingkungan Node.js yang mampu mengotomatisasi proses deteksi dan eliminasi *dead code* serta *unused dependencies* menggunakan pendekatan analisis statis *Abstract Syntax Tree* (AST).
2. Mengevaluasi kinerja deteksi dari metode AST dalam mengidentifikasi *dead code* dan *unused dependencies* berdasarkan instrumen *Confusion Matrix* (*Precision*, *Recall*, dan *F1-Score*) guna meminimalisir risiko kesalahan identifikasi pada proyek berbasis JavaScript dan TypeScript.

1.5 Manfaat Penelitian

Penelitian ini diharapkan dapat memberikan kontribusi yang signifikan, baik dari segi pengembangan ilmu pengetahuan maupun penerapannya di industri, yang dijabarkan sebagai berikut:

1.5.1 Manfaat Teoritis

Manfaat teoritis berkaitan dengan kontribusi penelitian terhadap khazanah keilmuan Rekayasa Perangkat Lunak (Software Engineering), khususnya dalam domain analisis kode dan kualitas perangkat lunak:

1. Pengayaan Literatur Analisis Statis : Penelitian ini memberikan bukti empiris mengenai efektivitas metode Analisis Statis berbasis *Abstract Syntax Tree* (AST) dalam mendeteksi kode dan dependensi yang tidak digunakan. Hasil penelitian ini menunjukkan bahwa analisis struktur kode tanpa eksekusi (*no runtime*) dapat digunakan sebagai pendekatan alternatif dalam proses identifikasi *dead code* dan *unused dependency* pada proyek JavaScript dan TypeScript.
2. Kontribusi terhadap Evaluasi Metode Deteksi Kode : Penelitian ini memberikan hasil pengukuran kinerja metode deteksi berbasis AST menggunakan *Confusion Matrix* melalui nilai *Precision*, *Recall*, dan *F1-Score*. Hasil tersebut diharapkan dapat menjadi referensi bagi penelitian selanjutnya dalam pengembangan metode analisis statis untuk mendeteksi elemen kode yang tidak digunakan.

1.5.2 Manfaat Praktis

Manfaat praktis berkaitan dengan dampak langsung penggunaan alat (*tool*) yang dihasilkan bagi para pemangku kepentingan (*stakeholders*):

1. Bagi Pengembang Perangkat Lunak (*Software Developers*) : Alat ini membantu pengembang menerapkan prinsip *Clean Code* dengan mengotomatisasi deteksi "sampah kode" yang sering terlewatkan secara

manual. Hal ini meminimalisir *human error* dan membuat basis kode (*codebase*) lebih mudah dipahami dan dikembangkan lebih lanjut. Selain itu, laporan visualisasi interaktif berupa peta struktur proyek akan membantu pengembang memahami topologi ketergantungan internal, yang secara langsung mempermudah proses audit kode dan mempercepat pemahaman alur sistem bagi pengembang baru (*onboarding*).

2. Bagi Efisiensi Lingkungan CI/CD (*DevOps*) : Dengan membantu mengidentifikasi dependensi yang tidak digunakan, alat ini berpotensi mendukung pengurangan komponen yang perlu diproses dalam lingkungan CI/CD. Dampak terhadap waktu build tidak diukur secara langsung dalam penelitian ini.
3. Bagi Keamanan Aplikasi : Alat ini berfungsi sebagai mekanisme preventif untuk membantu mengurangi *attack surface* (permukaan serangan) aplikasi Node.js. Dengan membantu mengidentifikasi pustaka pihak ketiga yang tidak digunakan, alat ini dapat mendukung pengurangan komponen yang membentuk *attack surface* aplikasi. Dampak keamanan tidak dievaluasi secara langsung dalam penelitian ini.

BAB V

KESIMPULAN DAN SARAN

Bab ini memuat kesimpulan yang diperoleh berdasarkan proses perancangan, implementasi, dan pengujian sistem. Selain itu, bab ini menyajikan saran pengembangan yang disusun berdasarkan keterbatasan sistem dan hasil evaluasi penelitian.

5.1 Kesimpulan

Berdasarkan rumusan masalah, tujuan penelitian, serta hasil perancangan, implementasi, dan pengujian yang telah dipaparkan pada bab-bab sebelumnya, diperoleh kesimpulan sebagai berikut:

1. Penelitian ini berhasil merancang dan membangun sistem berbasis Command Line Interface (CLI) pada lingkungan Node.js untuk mendeteksi dead code, berkas yang tidak terjangkau, dan dependensi yang tidak digunakan pada proyek JavaScript dan TypeScript. Sistem menerapkan analisis statis berbasis Abstract Syntax Tree (AST) dan dikembangkan menggunakan metode Extreme Programming (XP). Arsitektur sistem disusun dalam bentuk pipeline modular yang mencakup penguraian kode sumber, pemetaan graf dependensi, analisis cakupan dan alur program, klasifikasi keamanan temuan, remediasi, serta pelaporan. Hasil analisis disajikan melalui terminal, format JSON, dan dasbor HTML interaktif. Proses remediasi dilakukan secara semi-otomatis melalui pemilihan kandidat, peninjauan perubahan, konfirmasi pengguna, pencadangan berbasis checkpoint, dan mekanisme pemulihan. Dengan demikian, tujuan penelitian untuk membangun sistem yang membantu otomatisasi deteksi dan remediasi kode yang tidak digunakan telah tercapai sesuai dengan ruang lingkup analisis statis yang ditetapkan.
2. Evaluasi terhadap empat repositori sumber terbuka dan spesimen anomali yang diinjeksi secara terkontrol menghasilkan 100 True Positive (TP), 1

False Positive (FP), dan 5 False Negative (FN). Berdasarkan hasil tersebut, sistem memperoleh nilai precision sebesar 99,01%, recall sebesar 95,24%, dan F1-score sebesar 97,09%. Nilai precision menunjukkan bahwa sebagian besar temuan yang dilaporkan pada sampel pengujian sesuai dengan ground truth, sedangkan nilai recall menunjukkan bahwa masih terdapat pola tertentu yang belum terdeteksi. Keterbatasan tersebut terutama muncul pada penggunaan dependensi melalui konfigurasi tidak langsung, semantik API yang tidak dapat disimpulkan hanya dari AST, anotasi tipe berbasis JSDoc, serta analisis alur data atau kondisi yang melintasi blok bersarang. Klasifikasi rekomendasi Safe, Review, dan Risky digunakan secara terpisah dari perhitungan akurasi untuk membatasi tindakan otomatis berdasarkan tingkat risiko. Pada repositori, konfigurasi, versi, dan perintah validasi yang digunakan dalam eksperimen, tidak ditemukan kegagalan pada rangkaian validasi setelah remediasi terhadap kandidat yang dipilih. Hasil tersebut menunjukkan bahwa pendekatan yang diterapkan memiliki kinerja yang baik pada sampel penelitian, tetapi tidak menghilangkan kebutuhan terhadap peninjauan pengguna, khususnya untuk kode dinamis dan pola yang bergantung pada konteks runtime.

5.2 Saran

Berdasarkan keterbatasan yang ditemukan selama pengembangan dan pengujian, beberapa saran untuk penelitian dan pengembangan selanjutnya adalah sebagai berikut:

1. Penelitian selanjutnya dapat mengembangkan pendekatan hibrida yang menggabungkan analisis statis dengan informasi runtime secara terbatas. Informasi tersebut dapat membantu sistem memahami pemanggilan dinamis, reflection, dependency injection, serta semantik API atau callback yang tidak dapat disimpulkan hanya melalui struktur AST. Penerapan

pendekatan ini tetap perlu mempertimbangkan biaya eksekusi dan menjaga analisis statis sebagai mekanisme utama.

2. Kemampuan analisis tipe pada proyek JavaScript dapat diperluas dengan menambahkan pengurai JSDoc, khususnya untuk mengenali deklarasi seperti `@typedef`, `@property`, dan relasi penggunaan tipe. Pengembangan tersebut perlu membedakan dokumentasi biasa dari deklarasi tipe agar tidak menghasilkan temuan yang keliru.
3. Analisis alur data dan alur kontrol dapat ditingkatkan menjadi analisis antarblok dan antarprosedur yang lebih mendalam. Pengembangan ini diperlukan untuk mendeteksi write-only assignment, dead store, kontradiksi logika, dan kondisi mustahil yang melibatkan beberapa blok atau fungsi tanpa mengurangi perlindungan terhadap efek samping.
4. Analisis dependensi tidak langsung dapat diperluas melalui dukungan parser konfigurasi dan profil framework yang lebih banyak. Arsitektur berbasis plugin dapat digunakan untuk mengenali dependensi yang dipanggil melalui konfigurasi ESLint, Babel, bundler, test runner, atau mekanisme khusus framework, sehingga potensi false positive pada dependensi dapat dikurangi.
5. Mekanisme konsolidasi temuan perlu dikembangkan agar beberapa detektor yang melaporkan entitas sama dapat dipetakan ke satu kejadian kanonik tanpa menghilangkan informasi diagnostik setiap aturan. Normalisasi tersebut perlu diterapkan secara konsisten pada keluaran sistem, ground truth, dan evaluator agar perhitungan TP, FP, dan FN dapat direproduksi secara langsung.
6. Evaluasi berikutnya disarankan menggunakan jumlah dan variasi repositori yang lebih besar, mencakup aplikasi, pustaka, proyek monorepo, serta proyek dengan karakteristik framework yang berbeda. Setiap eksperimen sebaiknya menyimpan commit repositori, versi sistem, konfigurasi, keluaran mentah, hasil validasi, dan hash artefak agar hasil pengujian dapat direplikasi dan dibandingkan oleh peneliti lain.

7. Validasi setelah remediasi dapat diperluas dengan menjalankan kombinasi pemeriksaan sintaks, linting, kompilasi tipe, unit test, integration test, dan pengujian perilaku. Pengujian tersebut akan memberikan bukti yang lebih kuat mengenai dampak perubahan dibandingkan hanya mengandalkan keberhasilan satu perintah validasi proyek.
8. Penelitian selanjutnya dapat melakukan evaluasi komparatif terhadap alat deteksi *dead code* atau *unused dependency* yang memiliki ruang lingkup serupa. Perbandingan sebaiknya dilakukan menggunakan repositori, *ground truth*, unit evaluasi, dan aturan pencocokan yang sama sehingga nilai *precision*, *recall*, dan *F1-score* dari masing-masing alat dapat dibandingkan secara langsung. Pendekatan ini dapat memberikan gambaran yang lebih objektif mengenai kelebihan dan keterbatasan metode analisis statis berbasis AST yang diterapkan oleh DeadKiller dibandingkan pendekatan lain, seperti analisis dinamis atau hibrida [2], [4].



DAFTAR PUSTAKA

- [1] P. Jaisri, B. Reid, and R. G. Kula, "A Preliminary Study on Self-contained Libraries in the NPM Ecosystem," pp. 53–65, 2025, doi: 10.1007/978-3-031-82610-8_4.
- [2] Y. Liu, D. Tiwari, C. Bogdan, and B. Baudry, "Detecting and removing bloated dependencies in CommonJS packages," *J. Syst. Softw.*, vol. 230, pp. 1–47, 2025, doi: 10.1016/j.jss.2025.112509.
- [3] N. R. Weeraddana, M. Alfadel, and S. McIntosh, "Dependency-Induced Waste in Continuous Integration," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 1–23, 2024, doi: 10.1145/3660823.
- [4] I. Malavolta *et al.*, "JavaScript Dead Code Identification, Elimination, and Empirical Assessment," *IEEE Trans. Softw. Eng.*, vol. 49, no. 7, pp. 3692–3714, 2023, doi: 10.1109/TSE.2023.3267848.
- [5] T. Kupoluyi, M. Chaqfeh, M. Varvello, W. Hashmi, L. Subramanian, and Y. Zaki, "Muzeel : A Dynamic JavaScript Analyzer for Dead Code Elimination in Today's Web," *arXiv Prepr.*, 2021.
- [6] I. Koishybayev and A. Kapravelos, "Mininode: Reducing the attack surface of node.js applications," *RAID 2020 Proc. - 23rd Int. Symp. Res. Attacks, Intrusions Defenses*, pp. 121–134, 2020.
- [7] H. Boomsma, "Dead code elimination for web applications written in dynamic languages".
- [8] M. Irfan, W. Widhiarso, and M. Rachmadi, "Penerapan Clean Code Pada Aplikasi Siizin Berdasarkan Smells And Heuristics," *3rd MDP Student Conf. 2024*, pp. 95–102, 2024.
- [9] C. Soto-Valero, N. Harrand, M. Monperrus, and B. Baudry, "A comprehensive study of bloated dependencies in the Maven ecosystem," *Empir. Softw. Eng.*, vol. 26, no. 3, 2021, doi: 10.1007/s10664-020-09914-8.
- [10] C. C. Chuang, L. Cruz, R. Van Dalen, V. Mikovski, and A. Van Deursen, "Removing dependencies from large software projects: Are you really sure?," *Proc. - 2022 IEEE 22nd Int. Work. Conf. Source Code Anal. Manip. SCAM 2022*, pp. 105–115, 2022, doi: 10.1109/SCAM55253.2022.00017.
- [11] M. Chen, G. Li, L. Wu, and R. Liu, "DCE-LLM : Dead Code Elimination with Large Language Models".
- [12] E. S. Modules, "Automated Refactoring of Legacy JavaScript Code to ES6 Modules," 2021.

- [13] E. International, "ECMA-262 ECMAScript 2026 Language Specification," no. June, 2026.
- [14] T. Z. Unisinos and C. André, "How TypeScript Compiler Options Influence Code Quality : A Correlation Study with Git Repositories," 2026, doi: 10.5753/jserd.2026.5985.
- [15] M. Mustamiin, E. Ismantohadi, T. Informatika, and P. N. Indramayu, "IMPLEMENTASI ABSTRACT SYNTAX TREE PADA PENILAIAN KODE," vol. 3, no. 1, pp. 109–116, 2022.
- [16] M. Schr, "An empirical investigation of command-line customization," 2022.
- [17] P. Cholke, T. Wasekar, S. Tambe, S. Sood, A. Wagaskar, and A. Thenge, "Lacuna-Based JavaScript Code Visualization and Dead Code Optimization: A Cross-Platform Performance Evaluation.," *SSRN Electron. J.*, pp. 1–11, 2025, doi: 10.2139/ssrn.5231348.
- [18] Ł. Kurant, "Static components dependency graph detection with evaluation metrics in React . js projects," vol. 45, pp. 127–134, 2025, doi: 10.15439/2025F5264.
- [19] Microsoft, "TypeScript for the New Programmer," Microsoft. Accessed: Aug. 13, 2026. [Online]. Available: <https://www.typescriptlang.org/docs/handbook/typescript-from-scratch.html>
- [20] Microsoft, "Modules," Microsoft. Accessed: Aug. 13, 2026. [Online]. Available: <https://www.typescriptlang.org/docs/handbook/2/modules.html>
- [21] R. C. Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*. Upper Saddle River: Prentice Hall, 2009.
- [22] A. A. I. I. Paramitha, E. Grana, A. Dewi, S. Informasi, U. Primakara, and S. Informasi, "IMPLEMENTASI METODE EXTREME PROGRAMMING UNTUK PENGEMBANGAN KARIER DAN ALUMNI," vol. 8, no. 5, pp. 10014–10022, 2024.
- [23] A. B. Paksi, N. Hafidhoh, and S. K. Bimonugroho, "Perbandingan Model Pengembangan Perangkat Lunak Untuk Proyek Tugas Akhir Program Vokasi," vol. 14, no. 1, pp. 70–79, 2023.
- [24] A. Shrivastava, I. Jaggi, N. Katoch, and D. Gupta, "A Systematic Review on Extreme Programming," 2021, doi: 10.1088/1742-6596/1969/1/012046.
- [25] P. Explained and K. Beck, *Praise for Extreme Programming Explained , Second Edition*.
- [26] F. Maurer and S. Martel, "Extreme Programming," no. February, 2002.
- [27] A. Tharwat, "Classification assessment methods," vol. 17, no. 1, pp. 168–

192, 2026, doi: 10.1016/j.aci.2018.08.003.

- [28] T. Gumelar, R. Astuti, and A. T. Sunarni, "SISTEM PENJUALAN ONLINE DENGAN METODE EXTREME PROGRAMMING," vol. 9, no. 2, pp. 87–90, 2017.
- [29] D. I. Putri, J. R. Pradana, and M. Baydhowi, "Penerapan Agile Extreme Programming untuk Optimasi Layanan Pemesanan Menu Digital pada Mogano Coffee and Space," vol. 12, no. 1, pp. 58–68, 2025.
- [30] M. Farhan, F. Rahman, K. Darussalam, R. C. Saphira, and F. Purwani, "IMPLEMENTASI EXTREME PROGRAMMING DALAM PENGEMBANGAN APLIKASI MOBILE PENGENALAN ORGANISASI PADA MASA ORIENTASI MAHASISWA Fakultas Sains dan Teknologi Universitas Islam Negeri Raden Fatah Palembang," vol. 14, no. 2, pp. 128–132, 2024.
- [31] Nursahiba and Depandi Enda, "Pengukuran Kualitas Perangkat Lunak Dengan Standar ISO/IEC 25010 Pada Website Jurusan Teknik Informatika Politeknik Negeri Bengkalis," *JEKIN - J. Tek. Inform.*, vol. 5, no. 1, pp. 13–23, 2025, doi: 10.58794/jekin.v5i1.839.
- [32] R. Agarwal, D. Hall, D. Umphress, and D. Hall, "Extreme Programming for a Single Person Team," pp. 82–87.
- [33] J. H. Andrews, L. C. Briand, and S. Member, "Using Mutation Analysis for Assessing and Comparing Testing Coverage Criteria," vol. 32, no. 8, pp. 608–624, 2006.

STATE ISLAMIC UNIVERSITY
SUNAN KALIJAGA
YOGYAKARTA