JURNAL TEKNIK INFORMATIKA

Vol. 5 No. 1 Tahun. 2024
ISSN 2809-1353
DOI : 10.58794/jekin.v5i1.900

Penggunaan Software Metrics Dan Abstract Syntax Tree Untuk Mendeteksi Code Smell Pada Bahasa Pemrograman Python

Hamzan Wadi [1], Kasmawi [2], Muhammad Asep Subandri [3]
[1,2,3]Politeknik Negeri Bengkalis
e-mail: [1] hamzanwadisiregar@gmail.com, [2]kasmawi@polbeng.ac.id, [3]msubandri@polbeng.ac.id

Abstract* – *Code smell is a design flaw or bad practice in program code that is detrimental to software development projects. It can lead to decreased code quality, increased project complexity, obstacles in code maintenance, and risk of errors. This research aims to develop a code smell detection application in the Python programming language. The method used is software metrics implemented on the Abstract Syntax Tree (AST) structure. This system converts Python program into AST, develops code smell detection logic with software metrics approach, and tested using blackbox testing. The results show that the system is able to detect code smell types such as long method, lazy class, feature envy, and code complexity. Blackbox testing proves that the system functionality runs well and as expected.

Keywords - Abstract Syntax Tree (AST), Code Smell, Python, Software Metrics.

Abstrak - Code smell* adalah kelemahan desain atau praktik buruk dalam kode program yang merugikan proyek pengembangan perangkat lunak. Hal ini dapat menyebabkan penurunan kualitas kode, meningkatkan kompleksitas proyek, hambatan dalam pemeliharaan kode, dan risiko kesalahan. Penelitian ini bertujuan untuk mengembangkan aplikasi deteksi *code smell* pada bahasa pemrograman *Python*. Metode yang digunakan adalah *software metrics* yang diimplementasikan pada struktur *Abstract Syntax Tree* (AST). Sistem ini mengonversi program Python menjadi AST, mengembangkan logika deteksi *code smell* dengan pendekatan *software metrics*, dan diuji menggunakan *blackbox testing*. Hasil penelitian menunjukkan bahwa sistem mampu mendeteksi jenis *code smell* seperti *long method, lazy class, feature envy,* dan kompleksitas kode. Pengujian *blackbox* membuktikan bahwa fungsionalitas sistem berjalan dengan baik dan sesuai harapan.

Kata Kunci - Abstract Syntax Tree (AST), Code Smell, Python, Software Metrics

I. PENDAHULUAN

Pengembangan perangkat lunak merupakan bidang yang sangat penting dalam teknologi informasi saat ini. Kualitas kode program yang baik adalah faktor kunci untuk menciptakan perangkat lunak yang andal, mudah dipahami, dan mudah dirawat. Salah satu masalah umum dalam pengembangan perangkat lunak adalah adanya code smell. Code smell merujuk pada tanda-tanda atau karakteristik dalam kode program yang menunjukkan kelemahan desain atau praktik buruk yang dapat memengaruhi kualitas dan pemeliharaan perangkat lunak.

Code smell dapat menyebabkan berbagai masalah dalam perangkat lunak, termasuk kesulitan dalam memahami kode, yang dapat menghambat kolaborasi tim, memperlambat proses pengembangan, dan meningkatkan risiko kesalahan[1]. Code smell dapat berupa pola yang tidak efisien, pengulangan kode yang tidak perlu, atau struktur kode yang kompleks dan sulit dipahami. Hal ini menyebabkan berbagai masalah, termasuk kesulitan dalam memahami kode, yang dapat menghambat kolaborasi tim, memperlambat proses pengembangan, dan meningkatkan risiko kesalahan. Oleh karena itu, penting untuk mendeteksi dan mengatasi code smell sejak awal dalam siklus pengembangan perangkat lunak[1]. Salah satu pendekatan yang dapat digunakan adalah penggunaan software metrics pada struktur Abstract Syntax Tree (AST) dari kode program.

Lazy class adalah code smell yang mengacu pada kelas yang memiliki sedikit baris kode dan tidak memiliki metode yang mencukupi. Kelas semacam ini menunjukkan bahwa fungsionalitas dan tanggung jawab kelas tersebut tidak cukup terdefinisi dengan jelas, yang dapat mengakibatkan kompleksitas kode yang tidak perlu[2].

Feature envy adalah code smell yang terjadi ketika metode dalam sebuah kelas mengakses objek data dari kelas lain lebih banyak daripada data dalam kelas itu sendiri. Hal ini menunjukkan bahwa metode tersebut tampaknya lebih fokus pada atribut kelas lain daripada kelas itu sendiri, mengindikasikan adanya masalah dalam pemisahan tanggung jawab antara kelas[2].

Long method adalah code smell yang menunjukkan bahwa terdapat terlalu banyak pernyataan, variabel, serta kondisi pengambilan keputusan dan pengulangan dalam satu metode. Hal ini mengakibatkan metode menjadi terlalu panjang, sulit dipahami, dan sulit untuk digunakan kembali, yang pada akhirnya dapat mengurangi kualitas dan pemeliharaan kode[2].

Kompleksitas kode menjadi perhatian utama sepanjang siklus hidup perangkat lunak. Metrik kompleksitas kode digunakan untuk mengukur kualitas properti perangkat lunak. Kompleksitas perangkat lunak pada dasarnya memprediksi informasi analitis tentang keandalan, uji coba, dan kemampuan pemeliharaan dari pengukuran otomatis kode. Kode sumber yang kompleks akan sulit dikembangkan dan dipelihara oleh pengembang perangkat lunak[3].

Menurut indeks komunitas pemrograman TIOBE[4], Python telah menjadi salah satu bahasa pemrograman yang paling populer, menduduki peringkat pertama pada Juni 2023. Kesederhanaan sintaksis Python dan keluwesannya dalam berbagai bidang seperti analisis data, pembelajaran mesin, dan pengembangan web telah memberikan kontribusi signifikan pada popularitasnya. Namun, terdapat kekurangan dalam penelitian yang fokus pada deteksi code smell pada Python, meskipun beberapa penelitian sebelumnya telah berhasil dalam mendeteksi code smell pada bahasa pemrograman lain seperti Java[5]–[7].

Penelitian ini bertujuan untuk mengembangkan sistem yang mampu mendeteksi code smell pada Python menggunakan pendekatan software metrics pada struktur AST. Diharapkan bahwa hasil penelitian ini akan memberikan kontribusi positif dalam memperbaiki kualitas perangkat lunak Python, dengan mendeteksi dan mengatasi code smell secara efektif sehingga kode program menjadi lebih mudah dipahami, dirawat, dan dikembangkan.

II. PENELITIAN YANG TERKAIT

Beberapa penelitian yang pernah dilakukan mengenai deteksi code smell memberikan wawasan yang relevan dengan penelitian ini. Penelitian[8] membahas tentang pendekatan baru dalam mendeteksi temporary field code smell menggunakan metrik desain berorientasi objek, terutama TCC (tight class cohesion). Masalah yang diangkat dalam penelitian ini adalah kurangnya perhatian terhadap deteksi dan refaktorisasi temporary field code smell serta kurangnya pemahaman dan pedoman yang jelas dalam mengenali dan mengatasi code smell tersebut. Tujuan penelitian ini adalah mengembangkan pendekatan baru untuk deteksi temporary field code smell dan memberikan kontribusi dalam pemahaman dan peningkatan maintainability perangkat lunak. Metode penelitian melibatkan pengembangan metrik baru dan aturan deteksi berdasarkan definisi formal temporary field serta pengujian dan perbandingan dengan pendekatan yang sudah ada. Hasil penelitian menunjukkan keberhasilan pendekatan deteksi temporary field code smell yang diusulkan dengan menggunakan metrik-metrik yang baru dikembangkan. Perbedaan penelitian ini dengan penelitian yang dilakukan adalah pada jenis code smell yang difokuskan (temporary field code smell) dan metrik yang digunakan (metrik desain berorientasi objek).

Selain itu, Penelitian[9] membahas tentang pendekatan deteksi code smells menggunakan nanopatterns dan asosiasi rule mining. Masalah yang dihadapi adalah kurangnya penyelesaian terhadap ketidakpastian dalam proses deteksi code smells, perbedaan hasil dari berbagai alat deteksi, dan pengaruh refactoring pada model yang terkait. Tujuan penelitian ini adalah mengembangkan kerangka kerja untuk mendeteksi code smells dengan memanfaatkan nanopatterns sebagai dasar pembuatan aturan dan mengatasi masalah-masalah yang ada. Metode penelitian melibatkan analisis bytecode Java, penggunaan tool untuk mengidentifikasi nanopatterns, ekstraksi aturan asosiasi, dan pengoptimalan aturan menggunakan algoritma BCO. Hasil penelitian ini adalah pengembangan kerangka kerja yang dapat mengatasi ketidakpastian, membandingkan hasil alat deteksi, dan mempertimbangkan pengaruh refactoring. Perbedaan dengan penelitian yang dilakukan adalah pada pendekatan yang digunakan (nanopatterns dan asosiasi rule mining) serta bahasa pemrograman yang difokuskan (Java dalam penelitian tersebut, sedangkan penelitian yang akan dilakukan fokus pada Python.

Penelitian lainnya [6] berfokus pada pengembangan tool refactoring otomatis untuk mendeteksi dan mengatasi lazy class code smell. Penelitian ini menggunakan pendekatan software metrics untuk mendeteksi lazy class code smell dan menerapkan aturan-aturan dalam sistem untuk mengidentifikasi code smell. Refactoring dilakukan secara otomatis untuk menghilangkan code smell pada program. Penelitian ini relevan karena menggunakan software metrics untuk mendeteksi code smell, meskipun jenis code smell yang difokuskan berbeda. Sa'adah berfokus pada lazy class, sementara penelitian ini mencakup berbagai jenis code smell pada bahasa pemrograman Python yaitu long method, lazy class, feature envy dan memberikan informasi tentang kompleksitas kode.

Penelitian selanjutnya[10] membahas pendekatan baru dalam mendeteksi code smells dengan memanfaatkan fitur gabungan dari metrik perangkat lunak dan fitur teks dalam kode sumber. Penelitian ini menggunakan pendekatan berbasis pembelajaran mesin dengan jaringan saraf berbasis pembelajaran mendalam, khususnya Convolutional Neural Network (CNN). Hasil penelitian menunjukkan bahwa pendekatan yang dikembangkan mampu menggabungkan fitur-fitur metrik perangkat lunak dan fitur teks menggunakan jaringan saraf berbasis CNN untuk mendeteksi code smells. Perbedaan penelitian ini dengan penelitian yang dilakukan adalah pada pendekatan yang digunakan pendekatan berbasis pembelajaran mesin dengan jaringan saraf berbasis CNN.

Penelitian selanjutnya[11] membahas tentang keberadaan code smells dalam produk-produk perangkat lunak dalam Software Product Line (SPL), terutama ketika menggunakan strategi bottom-up dalam pembangunan SPL. Code smells dapat mempengaruhi kualitas SPL dan produk turunannya. Tujuan penelitian ini adalah mengurangi code smells dalam SPL dengan menganalisis, mendeteksi, dan memperbaiki code smells dalam kode sumber produk, serta membangun model fitur/variabilitas dari SPL. Metode penelitian melibatkan analisis kode sumber, deteksi code smells, refactoring, dan reverse engineering. Pendekatan ini bertujuan untuk meningkatkan kualitas SPL dengan menghilangkan code smells dan membangun model fitur/variabilitas yang akurat. Perbedaan penelitian ini dengan penelitian yang dilakukan adalah jenis

pendekatan yang digunakan yaitu analisis kode sumber, deteksi code smells, refactoring, dan reverse engineering sementara penelitian yang dilakukan menggunakan software metrics dan abstract syntax tree.

Kelima penelitian ini menunjukkan pendekatan yang berbeda dalam mendeteksi code smell, baik dari segi jenis code smell yang dideteksi maupun metode yang digunakan, namun semuanya memberikan kontribusi penting dalam bidang deteksi code smell yang relevan dengan penelitian yang dilakukan.

.

III. METODE PENELITIAN

A. Requirement

Tahap awal dalam penelitian ini adalah mengidentifikasi dan merumuskan masalah yang akan dijadikan sebagai objek penelitian. Pada tahap ini dilakukan tinjauan literatur terkait code smell, pendekatan software metrics, dan struktur Abstract Syntax Tree (AST) pada bahasa pemrograman Python. Selain itu, Pengumpulan rule atau aturan code smell dari jurnal penelitian terdahulu juga dilakukan untuk dijadikan acuan dalam melakukan analisis sistem. Rule atau aturan code smell dari jurnal penelitian terdahulu juga dikumpulkan sebagai acuan dalam melakukan analisis sistem [2]- [3]

TABEL I
RULE ATURAN DETEKSI CODE SMELL

No	Code Smell	Aturan
1	Long Method	a. NOP > 7 b. NLOC > 20 c. VG > 5
2	Lazy Class	a. If NOM = 0 b. If LOC < 100 & WMC <= 2 *If there is 1 rule above then this code code is detected
3	Feature Envy	a. If CBO > 5 b. If LCOM > 2 *If there is a rule above then this code smell is detected
4	Kompleksitas Kode	CC 1 – 10 = Simpel CC 11 – 20 = Cukup kompleks CC 21 – 50 = Kompleks CC 50+ = Sangat Kompleks

Penjelasan:

NOP : Number of Parameters (jumlah parameter)
NLOC : Metrics Non-Comment Lines of Code
VG : Metrics Variables Global
LOC : Metrics Lines of Code
CBO : Metrics Coupling Between Objects
NOM : Number of Methods (jumlah method)
WMC : Metrics Weighted Method Count
LCOM : Metrics Lack of Cohesion in Methods
CC : Cyclomatic Complexity

B. Design

Pada tahap ini, dilakukan analisis sistem dengan mengidentifikasi metode efektif dari penelitian terdahulu untuk menganalisis code smell pada kode Python. Sistem yang dirancang mencakup penerapan metrik perangkat lunak pada struktur AST kode Python. Evaluasi metrik dilakukan untuk mendeteksi code smell seperti Long Method, Lazy Class, Feature Envy, dan kompleksitas kode. Pembuatan use case dan activity diagram digunakan untuk memvisualisasikan interaksi komponen sistem dan fungsionalitasnya.

C. Implementation

Pada tahap ini, sistem yang telah dirancang diimplementasikan berdasarkan pendekatan software metrics pada struktur AST. Logika deteksi code smell dan penghitungan metrik diterapkan ke dalam sistem. Implementasi dilakukan dengan menggunakan bahasa pemrograman yang sesuai untuk memastikan desain dapat diterjemahkan ke dalam bentuk yang dapat dipahami oleh mesin.

Langkah awal dalam analisis statis melibatkan transformasi kode sumber Python yang dimasukkan oleh pengguna menjadi struktur data yang disebut Abstract Syntax Tree (AST) menggunakan modul ast bawaan Python. AST adalah representasi struktural dari kode yang memodelkan hubungan hierarkis antar elemen sintaksis. Fungsi index() mengaktifkan proses transformasi dengan **ast.parse(code)**, menghasilkan AST yang disimpan dalam variabel tree. Modul ast memfasilitasi pemrosesan pohon dari tata bahasa abstrak Python, yang memungkinkan analisis mendalam terhadap struktur sintaksis kode[12]. Selain itu, AST juga dapat digunakan untuk mewakili sebagian besar bahasa pemrograman procedural[13].

Deteksi code smell long method diterapkan untuk meningkatkan pemahaman terhadap kualitas perangkat lunak. Implementasi deteksi ini bertujuan untuk mengidentifikasi metode yang terlalu panjang dan kompleks, yang dapat menyulitkan pemeliharaan dan evolusi perangkat lunak.

- NOP (Number of Parameters): Mengukur jumlah parameter dalam metode. **len(node.args.args)** digunakan untuk menghitung NOP.
- NLOC (Number of Lines of Code): Mengukur jumlah baris kode dalam metode dengan menghitung pernyataan menggunakan **ast.walk(node).**
- VG (Variables Global): Mengukur jumlah variabel global dalam metode dengan iterasi melalui simpul AST dan menghitung variabel global menggunakan **ast.Global.**

Dalam penelitian ini, deteksi code smell lazy class diterapkan untuk memahami dan meningkatkan kualitas perangkat lunak. Implementasi deteksi ini bertujuan untuk mengidentifikasi kelas-kelas yang tidak memberikan kontribusi signifikan terhadap fungsionalitas sistem, yang dapat menjadi beban tambahan dalam pemeliharaan perangkat lunak. Deteksi lazy class didasarkan pada beberapa metrik perangkat lunak.

- NOM (Number of Methods): Mengukur jumlah metode dalam kelas dengan menghitung **ast.FunctionDef** dalam tubuh kelas.
- LOC (Lines Of Code): Mengukur jumlah baris kode dalam kelas dengan menghitung pernyataan menggunakan **ast.walk(node).**
- WMC (Weighted Methods per Class): Mengukur kompleksitas metode dalam kelas dengan menghitung CC dari metode menggunakan fungsi **calculate_cc** dan menjumlahkannya.

Dalam penelitian ini, deteksi code smell Feature Envy diterapkan untuk meningkatkan pemahaman terhadap kualitas perangkat lunak. Implementasi deteksi ini bertujuan untuk mengidentifikasi kelas yang mungkin terlalu bergantung pada fitur-fitur dari kelas lain, menunjukkan gejala kurangnya kohesi dalam desain perangkat lunak.

- CBO (Coupling Between Object classes): Mengukur ketergantungan kelas terhadap kelas lain dengan memeriksa pemanggilan metode dan referensi kelas lain dalam AST.
- LCOM (Lack of Cohesion in Methods): Mengukur kohesi metode dalam kelas dengan menghitung pasangan metode yang berbagi variabel instance.

Dalam analisis kompleksitas kode menggunakan rumus Cyclomatic Complexity **M = E - N + 2** yang dihitung pada struktur ast kode python yang diinput oleh pengguna. Berikut adalah potongan kode penerapan perhitungan Cyclomatic Complexity

- Cyclomatic Complexity: Mengukur kompleksitas kode dengan menghitung jumlah simpul, sambungan, dan percabangan dalam AST. Kompleksitas dihitung menggunakan rumus **edges - nodes + 2 + branches.**

D. Verification

Pada tahap ini dilakukan dua pengujian, yaitu uji website menggunakan metode black-box testing dan uji hasil deteksi code smell. Pengujian website dilakukan untuk mengevaluasi fungsi-fungsi dan interaksi sistem, sementara uji hasil deteksi code smell difokuskan pada evaluasi ketepatan sistem dalam mengidentifikasi code smell pada contoh-contoh kode yang ada.

E. Maintenance

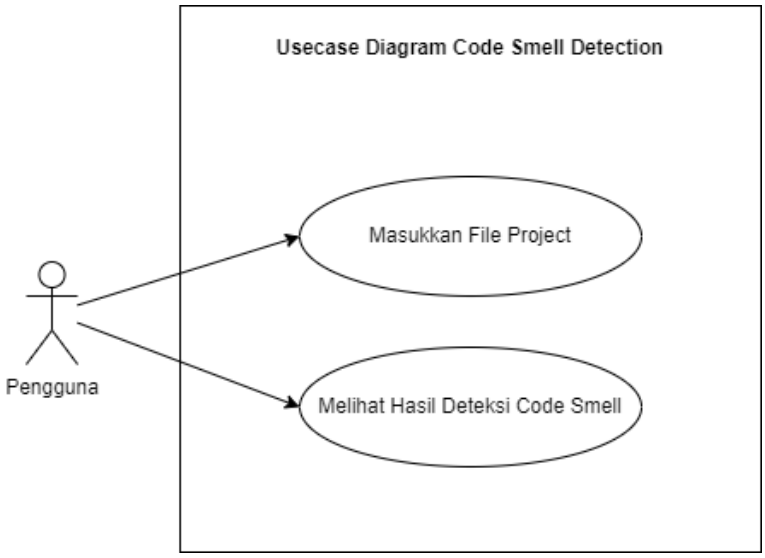
Dalam tahap ini dilakukan penanganan kesalahan (error handling) berdasarkan hasil dari pengujian. Setiap kesalahan yang diidentifikasi selama pengujian sistem akan dianalisis secara mendalam untuk memahami akar

penyebabnya. Proses ini melibatkan perbaikan bug, pembaruan kode, dan implementasi solusi untuk memastikan bahwa kesalahan tersebut tidak muncul kembali di masa depan.

IV. HASIL DAN PEMBAHASAN

A. Use Case Diagram

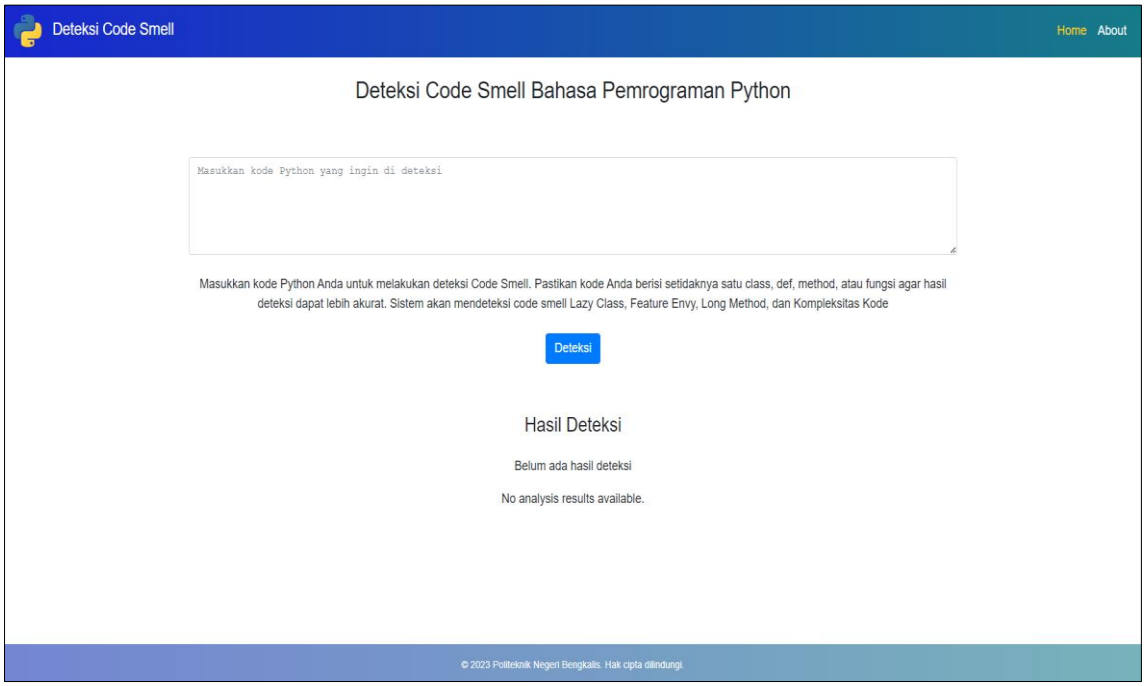
Use Case Diagram yang disajikan menggambarkan berbagai aktivitas yang dapat dilakukan oleh pengguna sistem deteksi code smell pada kode Python. Diagram ini memvisualisasikan interaksi antara pengguna dan sistem, termasuk input kode Python dan hasil yang diperoleh.



Gambar 1. Use Case Diagram

B. Halaman Home

Halaman Home adalah antarmuka utama tempat pengguna dapat memasukkan kode Python yang ingin dianalisis. Pengguna diharapkan untuk memasukkan kode ke dalam form yang disediakan dan mengklik tombol deteksi untuk memulai proses analisis.

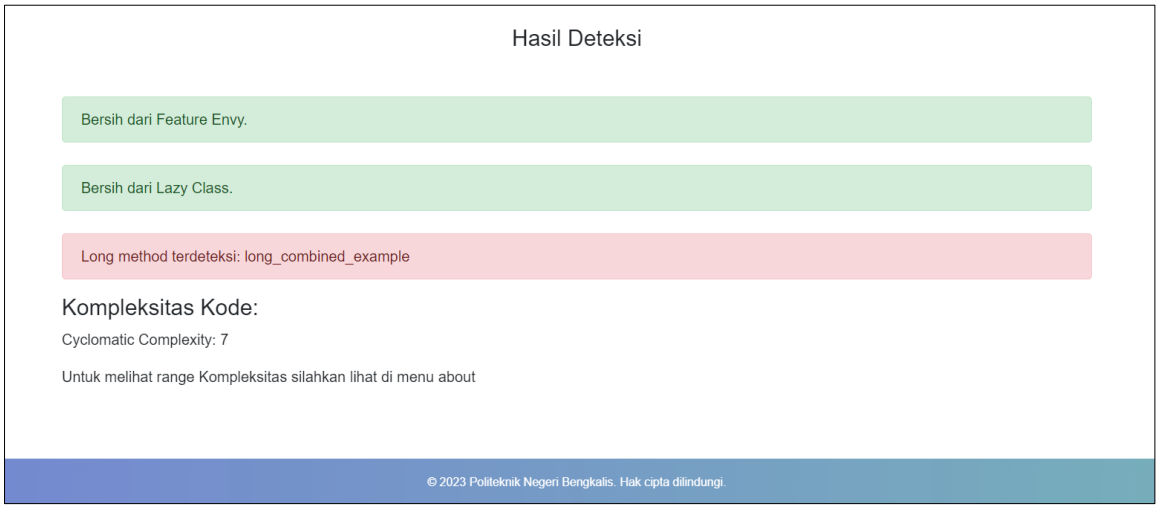


Gambar 2. Tampilan Halaman Home

Halaman ini menyediakan antarmuka yang intuitif untuk pengguna yang memungkinkan mereka untuk memasukkan kode dengan mudah. Proses penginputan ini penting karena kualitas deteksi code smell sangat bergantung pada kode yang diinputkan. Pada tahap ini, sistem menyusun kode yang dimasukkan dalam bentuk Abstract Syntax Tree (AST) untuk dianalisis lebih lanjut.

C. Halaman Hasil Deteksi

Setelah kode Python dimasukkan dan tombol deteksi diklik, sistem akan memproses kode tersebut untuk mendeteksi berbagai jenis code smell seperti Feature Envy, Lazy Class, Long Method, serta melakukan analisis kompleksitas kode. Hasil dari deteksi ini akan ditampilkan di halaman hasil deteksi.

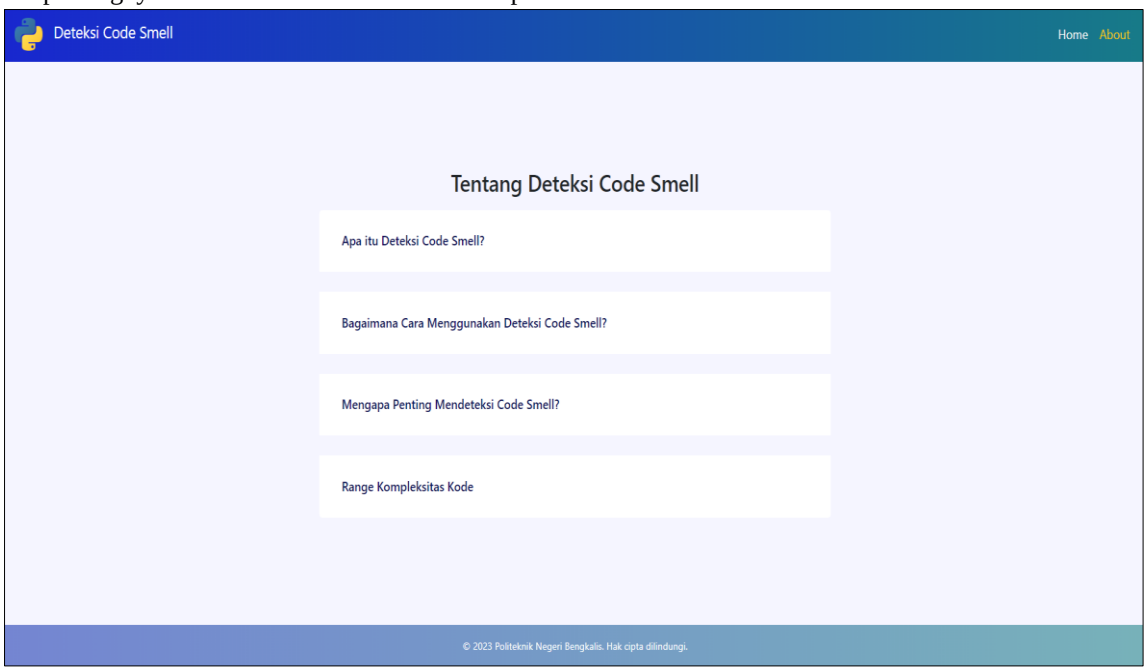


Gambar 3. Tampilan Halaman Hasil Deteksi

Pada halaman ini, pengguna dapat melihat apakah kode mereka mengandung code smell tertentu dan seberapa kompleks kode tersebut. Misalnya, jika sistem mendeteksi adanya code smell seperti Feature Envy, Lazy Class, atau Long Method, hasil tersebut akan ditampilkan beserta penjelasan singkat mengenai deteksi tersebut. Hasil analisis kompleksitas kode akan menunjukkan tingkat kompleksitas berdasarkan nilai Cyclomatic Complexity. Halaman ini juga memberikan umpan balik langsung kepada pengguna tentang kualitas kode mereka, yang dapat membantu dalam proses perbaikan dan refactoring.

D. Halaman About

Halaman About menyajikan informasi tambahan mengenai sistem, termasuk pengertian code smell, cara menggunakan sistem, serta pentingnya mendeteksi code smell dan kompleksitas kode.



Gambar 4. Tampilan Halaman About

Halaman ini berfungsi sebagai panduan dan sumber informasi bagi pengguna. Dengan menyediakan informasi tentang code smell dan cara kerja sistem, halaman ini mendukung pemahaman pengguna mengenai pentingnya deteksi code smell dan bagaimana interpretasi hasil dapat membantu dalam meningkatkan kualitas perangkat lunak.

E. Black Box Testing

Blackbox testing dilakukan untuk menguji fungsionalitas sistem sesuai dengan yang diharapkan. Berikut adalah hasil pengujian yang dilakukan.

TABEL II
PENGUJIAN BLACK BOX TESTING

No	Pengujian	Test Case	Hasil yang diharapkan	Hasil pengujian	Kesimpulan
1	Input kode python	Pengguna mengklik form input kode python	Bisa mengetik kode python yang ingin diinputkan	Sesuai harapan	Valid
2	Pengguna melakukan deteksi dengan inputan berupa kode python	Pengguna mengklik tombol deteksi	Menampilkan hasil deteksi code smell	Sesuai harapan	Valid
3	Pengguna melakukan deteksi dengan inputan kosong	Pengguna mengklik tombol deteksi	Menampilkan peringatan untuk mengisi form	Sesuai harapan	Valid
4	Pengguna melakukan deteksi dengan inputan selain kode python	Pengguna mengklik tombol deteksi	Menampilkan peringatan untuk mengisi kode bahasa pemrograman python	Sesuai harapan	Valid
5	Pengguna ingin berpindah ke menu About	Pengguna mengklik menu About di tampilan navbar	Berpindah halaman ke menu About	Sesuai harapan	Valid
6	Pengguna ingin berpindah ke menu	Pengguna mengklik menu Home di	Berpindah halaman ke menu Home	Sesuai harapan	Valid

	Home	tampilan navbar			
7	Pengguna ingin melihat informasi di menu About	Pengguna mengklik judul infomasi yang ingin dilihat	Menampilkan informasi secara lengkap	Sesuai harapan	Valid

F. Analisis Hasil Deteksi

Tabel berikut merangkum hasil deteksi code smell pada berbagai contoh kode Python yang diuji, termasuk detail deteksi dan kompleksitas kode.

TABEL III
HASIL DETEKSI CODE SMELL

Kode	Code Smell Yang Diuji	Kompleksitas	Detail Deteksi
1	Lazy Class	1	Kode terdeteksi sebagai Lazy Class, menandakan bahwa kelas MyClass memiliki inisialisasi anggota yang tertunda dan tidak efisien. Kompleksitas rendah (1) menunjukkan bahwa meskipun struktur kelas sederhana, efisiensinya rendah akibat inisialisasi yang tidak optimal.
2	Non-Lazy Class	1	Kode kedua tidak terdeteksi memiliki Lazy Class, Feature Envy, atau Long Method. Kompleksitas rendah (1) menunjukkan bahwa kode ini efisien, terstruktur dengan baik, dan tidak mengandung code smell yang signifikan.
3	Feature Envy	7	Kode terdeteksi sebagai Feature Envy, di mana kelas terlalu bergantung pada kelas lain untuk mendapatkan informasi atau fungsi. Kompleksitas rendah (7) menunjukkan bahwa meskipun ada ketergantungan yang signifikan, struktur kode tetap relatif sederhana dan tidak terlalu rumit.
4	Non-Feature Envy	7	Kode perbaikan berhasil menghilangkan Feature Envy dan Lazy Class. Kompleksitas tetap rendah (7), menunjukkan bahwa perbaikan membuat kode lebih bersih dan lebih efisien, tanpa menambah kerumitan.
5	Long Method	7	Kode ini mengandung Long Method, di mana fungsi long_combined_example terlalu panjang dan sulit dipahami. Kompleksitas rendah (7) menunjukkan bahwa meskipun fungsi panjang, strukturnya masih relatif sederhana dan tidak terlalu kompleks.
6	Non-Long Method	6	Kode perbaikan tidak mengandung Long Method dan dinyatakan bersih dari code smell lainnya. Kompleksitas menurun sedikit (6), menunjukkan bahwa perbaikan berhasil mengurangi panjang metode dan meningkatkan efisiensi tanpa menambah kompleksitas.
7	Kompleks	17	Kode ini menunjukkan kompleksitas tinggi (17), menandakan bahwa kode sangat kompleks dan mungkin memerlukan refactoring. Kompleksitas yang tinggi menunjukkan bahwa kode ini memiliki banyak elemen yang berkontribusi pada kerumitannya.

Pengujian ini menunjukkan efektivitas sistem dalam mendeteksi berbagai code smell dan memberikan informasi tentang kompleksitas kode, menjadikannya alat yang berguna untuk meningkatkan kualitas perangkat lunak.

V. KESIMPULAN

Penelitian ini berhasil mengembangkan sebuah sistem deteksi code smell yang mampu mengidentifikasi berbagai jenis code smell pada kode Python, termasuk long method, lazy class, dan feature envy. Sistem ini memanfaatkan software metrics pada struktur Abstract Syntax Tree (AST) untuk analisis yang lebih mendalam dan terstruktur. Implementasi logika deteksi berdasarkan aturan yang ditetapkan, serta pengukuran kompleksitas kode, telah berhasil dilakukan. Pengujian black-box menunjukkan bahwa sistem berfungsi sesuai harapan, dengan hasil yang akurat dalam mendeteksi code smell dan mengukur kompleksitas kode. Sistem ini memberikan wawasan berharga mengenai kualitas kode dan dapat menjadi alat bantu efektif dalam perbaikan kualitas perangkat lunak.

Beberapa saran pengembangan lanjutan dapat meningkatkan efektivitas sistem deteksi code smell untuk bahasa pemrograman Python. Pertama, melibatkan lebih banyak jenis code smell dalam proses deteksi akan memperluas cakupan analisis dan memberikan pandangan yang lebih lengkap terhadap potensi perbaikan dalam kode program. Penambahan fitur refactoring otomatis juga merupakan langkah penting; fitur ini tidak hanya akan memberikan saran perbaikan tetapi juga melakukan perubahan pada kode yang terdeteksi memiliki code smell secara otomatis. Dengan demikian, pengembang dapat secara efisien meningkatkan kualitas kode tanpa harus melakukan perubahan manual. Selain itu, integrasi fitur-fitur ini akan membawa dampak positif dalam pengembangan perangkat lunak, memperkaya fungsionalitas sistem, dan memberikan nilai tambah yang signifikan bagi para pengguna dalam mengelola dan meningkatkan kualitas kode program mereka.

UCAPAN TERIMA KASIH

Penulis mengucapkan rasa terima kasih yang sebesar-besarnya kepada Politeknik Negeri Bengkalis serta dosen pembimbing yang telah memberikan bantuan dan dukungan dalam penelitian ini. Selain itu, penulis juga menghaturkan terima kasih kepada seluruh anggota keluarga yang telah memberikan dukungan penuh dalam penyelesaian penelitian ini.

DAFTAR PUSTAKA

[1] M. Fowler, *Refactoring Improving the Design of Existing Code*, 2nd ed. Addison–Wesley, 2019.

[2] S. F. Sujadi, "Evaluasi Deteksi Smell Code dan Anti Pattern pada Aplikasi Berbasis Java," *J. Tek. Inform. dan Sist. Inf.*, vol. 5, no. 3, 2020, doi: 10.28932/jutisi.v5i3.1981.

[3] B. A. Sanusi, S. O. Olabiyisi, A. O. Afolabi, and ..., "Development of an Enhanced Automated Software Complexity Measurement System," *J. Adv. …*, vol. 1, no. 3, pp. 1–11, 2020, [Online]. Available: https://www.researchgate.net/profile/Adeolu-Afolabi/publication/338716707_Development_of_an_Enhanced_Automated_Software_Complexity_Measurement_Sys/links/5e4290ee458515072d91bbdf/Development-of-an-Enhanced-Automated-Software-Complexity-Measurement-Sys.pdf

[4] P. Jansen, R. Goud, and M. Wener, "The Python Programming Language," *TIOBE The Software Quality Company*, 2021. https://www.tiobe.com/tiobe-index/python/ (accessed Jul. 22, 2024).

[5] A. Kovačević *et al.*, "Automatic detection of Long Method and God Class code smells through neural source code embeddings," *Expert Syst. Appl.*, vol. 204, no. July 2021, 2022, doi: 10.1016/j.eswa.2022.117607.

[6] U. Sa'adah *et al.*, "Tool Refactoring Otomatis untuk Menangani Lazy Class Code Smell dengan Pendekatan Software Metrics," *J. Teknol. Inf. dan Ilmu Komput.*, vol. 9, no. 4, p. 743, 2022, doi: 10.25126/jtiik.2022934646.

[7] K. Azwega, A. Hendra Brata, E. Muhammad, and A. Jonemaro, "Pengembangan Sistem Deteksi God Class dan Brain Class Code Smell," 2020. [Online]. Available: http://j-ptiik.ub.ac.id

[8] R. Gupta and S. Kumar Singh, "A Novel Metric based Detection of Temporary Field Code Smell and its Empirical Analysis," *J. King Saud Univ. - Comput. Inf. Sci.*, vol. 34, no. 10, pp. 9478–9500, Nov. 2022, doi: 10.1016/j.jksuci.2021.11.005.

[9] D. Juliet Thessalonica, H. Khanna Nehemiah, S. Sreejith, and A. Kannan, "Intelligent Mining of Association Rules Based on Nanopatterns for Code Smells Detection," *Sci. Program.*, vol. 2023, 2023, doi: 10.1155/2023/2973250.

[10] A. Hamdy and M. Tazy, "Deep hybrid features for code smells detection," *J. Theor. Appl. Inf. Technol.*, vol. 98, no. 14, pp. 2684–2696, 2020.

[11] S. Ouali, "Generating Software Product Line Model by Resolving Code Smells in the Products' Source Code," *Int. J. Softw. Eng. Appl.*, vol. 12, no. 1, pp. 1–10, 2021, doi: 10.5121/ijsea.2021.12101.

[12] "Abstract Syntax Trees," *Python Software Foundation*, 2024. https://docs.python.org/3/library/ast.html (accessed Jul. 25, 2024).

[13] D. Thain, *Introduction to compilers and language design*, 2nd ed. notre dame: Independently published (June 18, 2020), 2022.