Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

Implementation and Evaluation of Static Code Analysis to Identify Security and Code Quality Issues in Academic Information Systems

Cecep Muhamad Sidik Ramdani[*1], Rahmi Nur Shofa[2], Muhammad Adi Khairul Anshary[3], Acep Irham Gufroni[4], Aria Priawan Yahya[5], Wan Mohd Amir Fazamin Bin Wan Hamzah[6]

[1,2,4]Department of Information Systems, Siliwangi University, Indonesia
[3,5]Department of Informatics, Siliwangi University, Indonesia
[6]Faculty of Informatics and Computing, Sultan Zainal Abidin University, Malaysia

Email: [1]cecepmuhamad@unsil.ac.id

Received : Oct 14, 2025; Revised : Nov 19, 2025; Accepted : Nov 18, 2025; Published : Dec 23, 2025

Abstract

In today's digital era, websites have become a key component of various digital services, from government and education to business. However, many security incidents occur due to undetected *source code* vulnerabilities, such as *vulnerabilities*, *bugs*, and *code smells*, which can degrade system performance and reliability. Therefore, a systematic approach is needed to detect and prevent these issues as early as possible. This study aims to implement and evaluate the effectiveness of *the Static Code Analysis* (SCA) method in identifying security and code quality issues in web applications. The tool used was SonarQube, which was then implemented in the SIMAK Universitas Siliwangi. Evaluation and testing were conducted on the tool's ability to detect various types of problems, its level of accuracy, and its ease of integration into the software development process. In this study, the evaluated aspects were *bugs*, *code smells*, and *vulnerabilities*. The results of this study found 23,241 issues, consisting of 2,356 *bugs* and 20,885 *code smells*, without any *vulnerabilities* found. With a problem ratio of 3.84% of the total code lines of 605,130, and a severity classification dominated by issues at the Critical and Major levels, these results provide an overview of the technical condition of the code used in SIMAK Universitas Siliwangi. This research is expected to provide practical contributions for software developers and security teams in continuously improving the quality and security of web applications. The outcomes of this study are expected to offer substantial and actionable contributions toward advancing the overall quality, robustness, and security of software systems. By strengthening these foundational aspects, the research is projected to positively influence the reliability, continuity, and long-term sustainability of academic service delivery within higher-education environments.

Keywords : *Bugs, Code Smells, SonarQube, Static Code Analysis (SCA), Source code, Vulnerabilities.*

1. INTRODUCTION

Digitalisasi proses akademik di perguruan tinggi menuntut Sistem Informasi Akademik (SIMAK) yang andal, aman, dan berkualitas, karena sistem ini memproses data berisiko tinggi seperti identitas mahasiswa, nilai, keuangan, serta pelaporan terintegrasi ke Pangkalan Data Pendidikan Tinggi (PDDIKTI) [1], [2]. Dalam praktik, SIMAK umumnya berwujud aplikasi web yang mengotomasi proses KRS, penjadwalan, penilaian, hingga pelaporan dan integrasi sehingga kualitas kode dan keamanan aplikasi web menjadi faktor penentu keberhasilan layanan akademik [3], [4].

Di sisi lain, lanskap ancaman siber terhadap layanan publik Indonesia, termasuk sektor pendidikan semakin meningkat. Laporan beberapa tahun terakhir menunjukkan insiden kebocoran data berskala besar dan gangguan layanan, termasuk peristiwa terkait Pusat Data Nasional dan data institusi pemerintah [5], [6]. Secara global, Verizon DBIR 2025 menganalisis lebih dari 12 ribu pelanggaran data terkonfirmasi dan menyoroti eksploitasi kerentanan serta kesalahan aplikasi begitu meningkat sangat signifikan [7].

Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

Pada tahap pembangunan, permasalahan yang sering muncul pada SIMAK adalah kualitas kode yang kurang terstandar dan dokumentasi yang minim. Banyak pengembang hanya berfokus pada fungsionalitas utama tanpa memperhatikan praktik terbaik seperti *code review*, modularitas, dan standar pengembangan perangkat lunak. Kondisi ini membuat sistem sulit dipelihara dan dikembangkan di kemudian hari. Selain itu, aspek keamanan sering kali belum menjadi prioritas sejak awal. Akibatnya, kode yang dihasilkan rentan terhadap kerentanan umum seperti SQL Injection, *Cross-Site Scripting* (XSS), maupun *Broken Access Control*. Kurangnya integrasi praktik pengembangan aman, seperti penerapan Static Code Analysis (SCA) dan security testing otomatis dalam pipeline pengembangan, turut memperbesar risiko keamanan yang tersembunyi di dalam kode.

Permasalahan juga muncul pada tahap implementasi. Salah satu masalah utama adalah beban akses yang tinggi pada periode krusial, yang sering menyebabkan sistem melambat bahkan down. Selain itu, resistensi pengguna juga menjadi kendala, di mana dosen, staf, maupun mahasiswa kadang enggan beradaptasi dengan sistem baru sehingga penggunaan sistem menjadi tidak konsisten. Dari sisi data, input manual yang masih dominan menimbulkan kesalahan, duplikasi, dan inkonsistensi yang dapat berimplikasi pada ketidakakuratan laporan ke PDDIKTI. Aspek keamanan operasional pun menjadi masalah serius, mengingat banyak SIMAK yang tidak rutin diperbarui atau dipantau sehingga rawan kebocoran data.

Untuk menjawab kebutuhan tersebut, kerangka NIST *Secure Software Development Framework* (SSDF) SP 800-218 mendorong integrasi praktik keamanan dalam seluruh siklus pengembangan (DevSecOps), termasuk otomatisasi pengujian statik/dinamik [8], [9]. Dalam konteks DevSecOps untuk aplikasi web seperti SIMAK, penerapan SCA sebagai kontrol preventif pada kualitas kode diakui sebagai praktik efektif untuk menekan risiko sejak dini [10], [11]. SCA memungkinkan pendeteksian dini suatu *bugs*, *code smells*, serta *Vulnerability* tanpa mengeksekusi program, sehingga mengurangi biaya perbaikan dan menstandarkan kualitas kode secara menyeluruh [12].

Penelitian ini memiliki kontribusi penting dalam memperkuat praktik *secure software development* di lingkungan pendidikan tinggi melalui penerapan SCA berbasis SonarQube pada SIMAK. Kebaruan penelitian ini terletak pada penerapan evaluasi kualitas dan keamanan kode secara menyeluruh terhadap sistem yang telah digunakan secara nyata oleh civitas akademika, bukan hanya pada skenario uji coba atau kode simulasi. Selain itu, penelitian ini mengintegrasikan hasil analisis SCA dengan klasifikasi tingkat keparahan (*severity*) untuk memberikan gambaran komprehensif mengenai kondisi teknis perangkat lunak akademik dan rekomendasi perbaikannya. Temuan ini diharapkan menjadi referensi bagi institusi pendidikan dalam membangun model pengujian keamanan dan kualitas kode yang berkelanjutan, sekaligus memperkuat tata kelola keamanan informasi di sektor akademik.

2. METHOD

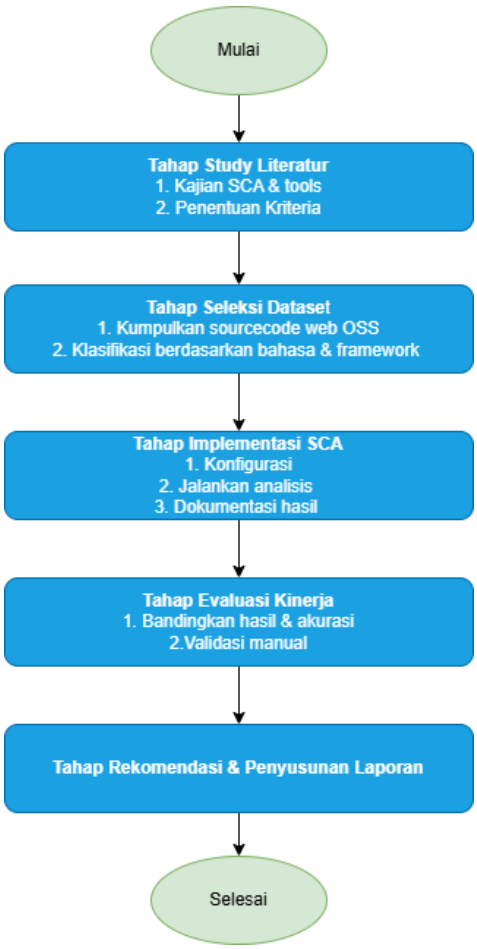
Pendekatan penelitian ini menggunakan metode kuantitatif-deskriptif dengan fokus pada evaluasi teknis kualitas kode. Analisis dilakukan melalui penerapan SCA menggunakan SonarQube yang dikonfigurasikan secara khusus untuk mendeteksi tiga kategori utama permasalahan, yaitu *bugs*, *code smells*, dan vulnerabilities. Setiap hasil temuan kemudian diklasifikasikan berdasarkan tingkat keparahan (*severity levels*): *Blocker, Critical, Major*, dan *Minor*. Proses ini memungkinkan identifikasi dan pengukuran tingkat kualitas perangkat lunak secara terukur, sekaligus memberikan dasar kuantitatif bagi penyusunan rekomendasi peningkatan mutu kode.

Selain itu, metode ini juga memanfaatkan pendekatan validasi berlapis, di mana sebagian hasil deteksi dianalisis ulang secara manual untuk memastikan akurasi dan mengurangi potensi false positive dari hasil pemindaian otomatis. Tahapan ini penting untuk menjamin validitas hasil serta memastikan bahwa rekomendasi perbaikan benar-benar merefleksikan kondisi aktual dari sistem yang dianalisis. Dengan desain metodologis ini, penelitian tidak hanya menghasilkan data statistik, tetapi juga

Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

memberikan panduan aplikatif bagi pengembang dalam menerapkan SCA secara efektif pada proyek perangkat lunak nyata.

Prosedur penelitian yang dilakukan terdiri dari beberapa tahapan seperti yang terlihat pada Gambar 1 berikut ini.



Gambar 1. Prosedur Penelitian.

2.1. Tahap Studi Literatur

Pada tahap ini dilakukan dua kegiatan utama:

a. Kajian SCA & *tools*: Kajian sistem dilakukan terhadap konsep SCA, metode, dan alat (*tools*) yang tersedia. Beberapa penelitian sebelumnya menunjukkan bahwa SCA dapat membantu mendeteksi kelemahan perangkat lunak lebih awal dalam siklus hidup pengembangan [13], [14].
b. Penentuan Kriteria: Setelah memahami karakteristik alat SCA, dilakukan penentuan kriteria seperti cakupan sistem pemrograman, dukungan framework, akurasi deteksi, serta integrasi dengan CI [15].

2.2. Tahap Seleksi Dataset

Pada tahap seleksi dataset, kegiatan yang dilakukan adalah sebagai berikut:

a. Kumpulkan source code web OSS: Dataset berupa kode sumber aplikasi web open source dipilih karena menyediakan variasi sistem dan kerangka kerja yang cukup luas serta mudah diakses [16].
b. Klasifikasi berdasarkan sistem & framework: Kode yang dikumpulkan dikelompokkan sesuai sistem pemrograman (misalnya PHP, Java, Python) serta framework (Laravel, Spring, Django). Hal ini penting karena akurasi alat SCA dapat bervariasi tergantung sistem yang dianalisis [17].

Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

2.3. Tahap Implementasi SCA

Pada tahap implementasi SCA, ada tiga kegiatan yang dilakukan, yaitu:

a. Konfigurasi: Peneliti melakukan konfigurasi pada alat SCA yang dipilih agar sesuai dengan dataset [18], [19].
b. Jalankan analisis: Analisis statis dijalankan untuk mendeteksi potensi kerentanan, *bugs*, dan *code smells* pada kode sumber tanpa mengeksekusi aplikasi [20].
c. Dokumentasi hasil: Hasil berupa laporan kelemahan, kualitas kode, dan rekomendasi perbaikan didokumentasikan untuk analisis selanjutnya [21].

2.4. Tahap Evaluasi Kinerja

Pada tahap ini, kegiatan yang dilakukan adalah:

a. Bandingkan hasil & akurasi: Hasil dari alat SCA dibandingkan berdasarkan jumlah temuan, cakupan kelemahan (misalnya OWASP Top 10, CWE Top 25), dan akurasi deteksi [22], [23].
b. Validasi manual: Beberapa temuan diverifikasi secara manual dengan membaca kode dan melakukan uji reproduksi untuk membedakan *true positive* dan *false positive* [24], [25].

2.5. Tahap Rekomendasi dan Penyusunan Laporan

Pada tahap rekomendasi dan penyusunan laporan, kegiatan yang dilakukan yaitu sistem rekomendasi terkait efektivitas SCA dalam mendeteksi kelemahan dan meningkatkan kualitas kode pada SIMAK. Hasil penelitian kemudian disusun dalam laporan akademik yang memuat metodologi, hasil, evaluasi, serta rekomendasi praktis untuk pengembangan perangkat lunak yang lebih aman dan berkualitas [26].

3. RESULT

3.1. Studi Literatur

Dari kajian yang dilakukan, pada penelitian ini digunakan tools SCA dengan Sonarqube ntuk mengevaluasi kualitas kode. SonarQube yang digunakan dalam pengujian adalah versi 25.1.0.102122, yang dilisensikan menggunakan LGPL v3. Dalam implementasinya, SonarQube dikonfigurasi dengan menggunakan database PostgreSQL versi 8.14. Pemilihan PostgreSQL sebagai database dilakukan karena SonarQube secara resmi merekomendasikan PostgreSQL untuk penyimpanan data dan pengelolaan proyek. Parameter yang digunakan dalam menganalisis kualitas kode, yaitu mengevaluasi *bugs*, *code smells*, dan *Vulnerability.*

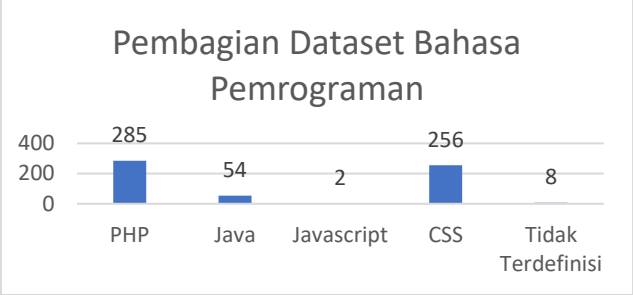
3.2. Seleksi Dataset

Pada tahap ini ditemukan sebanyak 605 ribu baris kode yang berasal dari beberapa sistem pemgrograman, diantaranya adalah PHP, Java, Javascript, dan CSS, seperti terlihat pada Tabel 1 dan Gambar 2 berikut ini.

Tabel 1. Pembagian Dataset.

Bahasa Pemrograman	Jumlah Baris Code
PHP	285.000
Java	54.000
Javascript	2.000
CSS	256.000
Tidak Terdefinisi	8.000

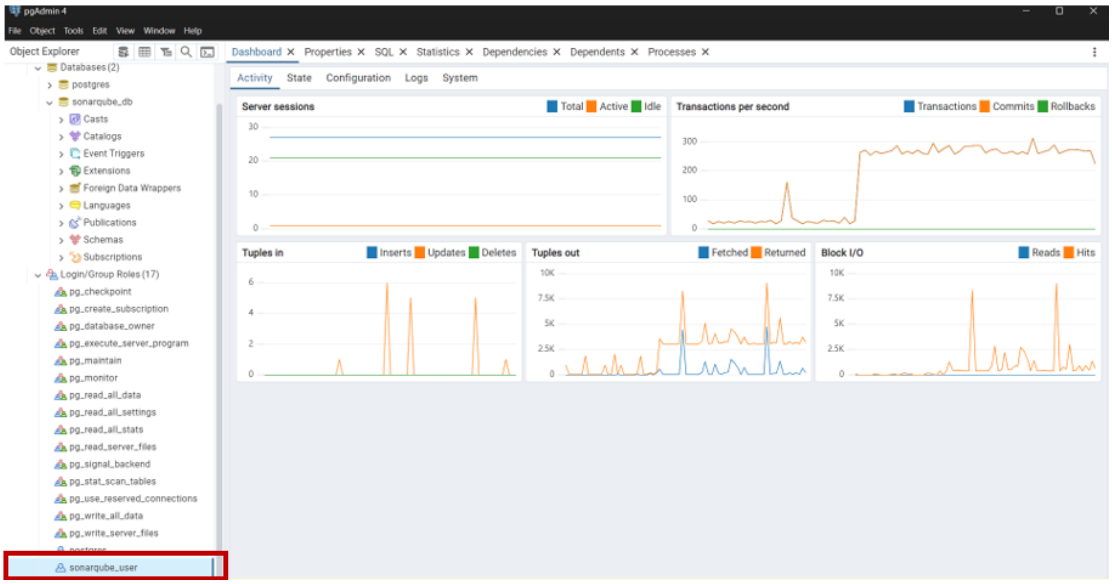
Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371



Gambar 2. Diagram Pembagian Dataset Bahasa Pemrograman.

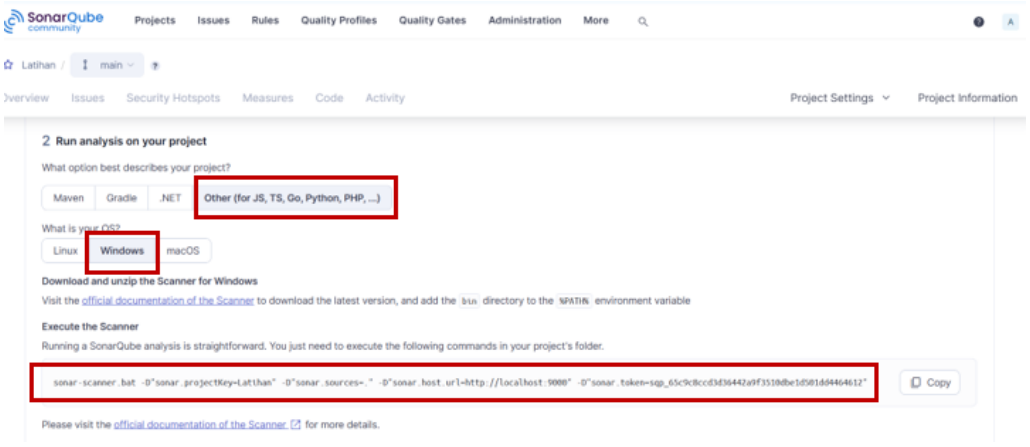
3.3. Implementasi SCA

Untuk seluruh kode yang dihasilkan dari dataset pada tahapan sebelumnya, dilakukan pengujian terhadap kualitas kode, dengan menggunakan alat bantu analisis statis, yaitu SonarQube. Pengujian dilakukan untuk memperoleh informasi terkait kualitas kode dari aspek *bugs, code smells*, dan *vulnerabilities* atau potensi kerentanan keamanan, seperti terlihat pada Gambar 3 berikut ini.



Gambar 3. Postgree SQL.

Proses integrasi database yang dilakukan mencakup pembuatan *user* dan *password* yang digunakan untuk login ke dalam sistem SonarQube sebagai admin, seperti yang terlihat pada Gambar 4 berikut ini.



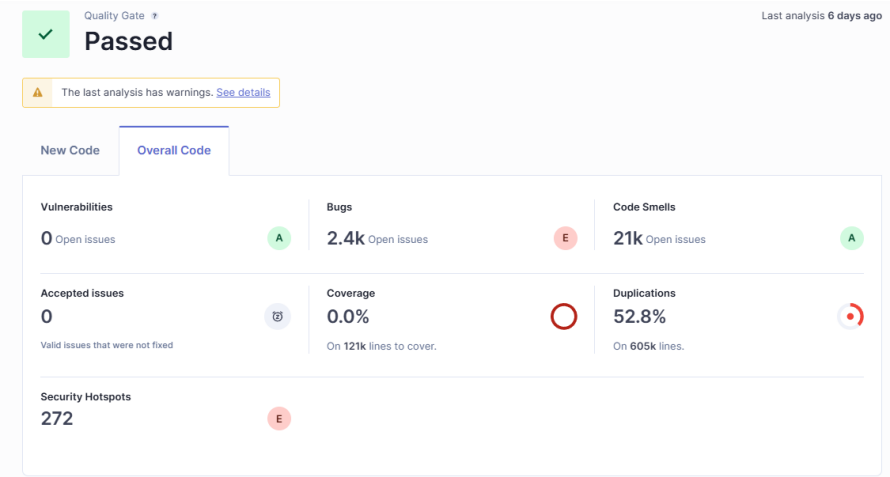
Gambar 4. Generate Token SonarQube.

Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

Sebelum proses analisis dilakukan, langkah awal adalah membuat sebuah proyek baru di SonarQube yang disesuaikan seperti pada Gambar 4. Setelah konfigurasi proyek selesai, sistem akan memberikan opsi untuk melakukan *generate* token yang digunakan sebagai autentikasi selama proses analisis berjalan. Karena sistem operasi yang digunakan adalah Windows, maka token disesuaikan dengan environment tersebut, seperti terlihat pada Gambar 5 berikut.

Gambar 5. Integrasi dengan VS Code.

Token yang telah di-*generate* SonarQube kemudian dimasukkan ke dalam terminal Visual Studio Code pada proyek hasil generate dari plugin Dualite. Pada gambar 5 diatas, terlihat bahwa proses dilakukan melalui integrasi penghubung antara SonarQube dan VS Code agar proses analisis dapat berjalan secara otomatis melalui perintah pada terminal. Selanjutnya SonarQube akan melakukan pemindaian menyeluruh terhadap semua file, yang dapat dilihat pada Gambar 6 berikut ini.



Gambar 6. Hasil analisi SonarQube.

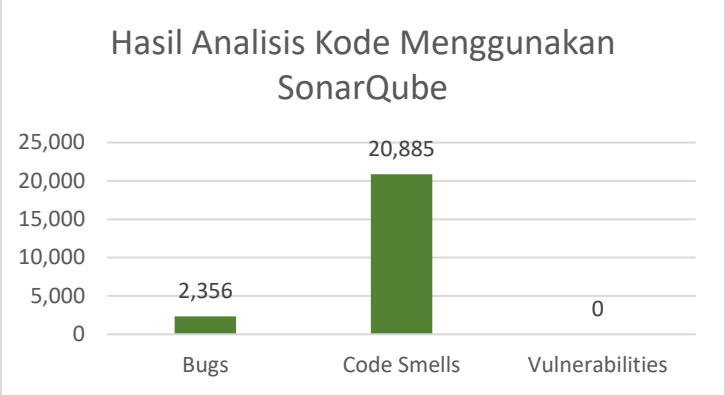
Hasil analisis yang ditampilkan pada Gambar 6 diatas, menunjukkan hasil pemeriksaan kualitas kode menggunakan SonarQube. Dari seluruh kode yang dianalisis, ditemukan sebanyak 2.400 *bugs*, 21.000 *code smells*, serta 0 *vulnerabilities*. Selain itu, SonarQube juga mendeteksi adanya 272 *security hotspots* yang berpotensi menjadi celah keamanan. Hasil analisis yang telah dilakukan turut mencatat bahwa tingkat duplikasi kode mencapai 52,8% pada sekitar 605 ribu baris kode, sementara nilai *coverage* tercatat 0%. Data kuantitatif ini memberikan gambaran bahwa permasalahan utama pada proyek terletak pada tingginya jumlah *code smells* dan *duplications*, yang kemudian akan dianalisis lebih lanjut pada Tabel 2, dengan hasil analisis kode yang dapat dilihat pada Gambar 7.

Jurnal Teknik Informatika (JUTIF)
P-ISSN: 2723-3863
E-ISSN: 2723-3871
Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

Tabel 2. Hasil Analisis Kode Menggunakan SonarQube.

Jenis Masalah	Jumlah Temuan
Bugs	2.356
Code Smells	20.885
Vulnerabilities	0
Lines of Code	605.130

`



Gambar 7. Diagram Hasil Analisis Kode.

Pada Gambar 7 diatas, SonarQube mengelompokkan hasil temuannya ke dalam beberapa kategori berdasarkan jenis masalah, yaitu *bugs*, *code smells*, dan *vulnerabilities*. Dari hasil analisis, jumlah *bugs* yang terdeteksi mencapai 2.356 isu, sedangkan pada kategori *code smells* ditemukan sebanyak 20.885 isu. Adapun pada kategori *vulnerabilities* tidak ditemukan adanya permasalahan. Secara keseluruhan, kode yang dianalisis berjumlah 605.130 baris, sehingga data ini memberikan gambaran awal mengenai tingkat kualitas perangkat lunak yang diuji sebelum masuk ke klasifikasi lebih lanjut berdasarkan tingkat keparahan (*severity*) seperti terlihat pada Tabel 3.

Tabel 3. Klasifikasi *Severity* Hasil Analisis Kode.

Jenis Masalah	*Blocker*	*Critical*	*Major*	*Minor*	Total
Bugs	28	90	2.100	130	2.348
Code Smells	24	11.509	6.028	3.211	20.772
Vulnerabilities	0	0	0	0	0
Total Jenis Masalah	52	11.599	8.128	3.341	23.120

Dari Tabel 3 diatas, terlihat bahwa SonarQube mengklasifikasikan setiap temuan berdasarkan tingkat keparahan (*severity*), yang terdiri dari empat kategori, yaitu *Blocker, Critical, Major,* dan *Minor*. Berdasarkan hasil analisis, temuan pada kategori *bugs* didominasi oleh tingkat Major sebanyak 2.100 isu, sedangkan pada kategori *code smells* sebagian besar berada pada tingkat *Critical* dengan 11.509 isu., sementara itu tidak ditemukan adanya *vulnerabilities* pada sistem yang diuji.

3.4. Evaluasi

Studi kasus pada tahap ini dilakukan untuk mendokumentasikan hasil analisis kualitas kode dari SIMAK Universitas Siliwangi yang digunakan secara nyata oleh mahasiswa dan civitas akademika. Analisis dilakukan dengan memanfaatkan SonarQube sebagai alat bantu *static code analysis*, dengan fokus pada tiga jenis temuan utama, yaitu: *bugs*, *code smells*, dan *vulnerabilities*. Data kuantitatif yang

Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

diperoleh kemudian diolah lebih lanjut untuk menggambarkan kondisi kualitas perangkat lunak pada SIMAK, serta dikelompokkan berdasarkan tingkat keparahan (*severity*), meliputi *Blocker, Critical, Major*, dan *Minor,* seperti terlihat pada Tabel 4.

Tabel 4. Kriteria Kelayakan Kode.

Kategori	Persentase Masalah (%)	Kelayakan
A	0 - 5%	Layak
B	6 - 10%	Cukup Layak
C	11 - 20%	Kurang Layak
D	21 - 50%	Tidak Layak

Kelayakan kualitas kode dalam penelitian ini mengacu pada standar kategori yang telah ditentukan oleh SonarQube. Data kuantitatif yang diperoleh dari hasil pemindaian SonarQube kemudian diolah lebih lanjut guna memberikan gambaran menyeluruh mengenai kualitas kode yang mendasari SIMAK. Selain itu, setiap temuan juga dikelompokkan berdasarkan tingkat keparahan (*severity*), meliputi *Blocker, Critical, Major,* dan *Minor*, sehingga memudahkan proses evaluasi serta penyusunan rekomendasi perbaikan.

a. Total Masalah yang Ditemukan

Tabel 5. Total Masalah yang Ditemukan.

Jenis Masalah	Jumlah
Bugs	2.356
Code Smells	20.885
Vulnerabilities	0
Total Masalah	**23.241**
Lines of Code (LoC)	605.130

Pada Tabel 5 diatas dapat dilihat bahwa berdasarkan hasil pemindaian menggunakan SonarQube, ditemukan bahwa terdapat sejumlah:

- 2.356 *bugs*
- 20.885 *code smells*
- 0 *vulnerabilities*

dengan Total masalah:

$$Total\ masalah = 2.356 + 20.885\ + 0 = 23.241 \quad (1)$$

Total keseluruhan masalah dalam proyek kode SIMAK Universitas Siliwangi adalah sebanyak 23.241 isu seperti terlihat pada Tabel 5 diatas. Untuk mengetahui sejauh mana jumlah masalah ini mempengaruhi kualitas keseluruhan kode, maka dilakukan perhitungan rasio masalah terhadap total baris kode (*lines of code*/LOC). Perhitungan ini mengacu pada rumus:

$$Rasio\ Masalah = \left(\frac{Total\ masalah}{Total\ baris\ kode}\right) x\ 100 \quad (2)$$

Dengan total 23.241 masalah dari 605.130 baris kode, maka:

$$Rasio\ Masalah = \left(\frac{23.241}{605.130}\right) x\ 100 = 3{,}84\% \quad (3)$$

Jurnal Teknik Informatika (JUTIF) Vol. 6, No. 6, December 2025, Page. 5791-5804
P-ISSN: 2723-3863 https://jutif.if.unsoed.ac.id
E-ISSN: 2723-3871 DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

Berdasarkan kriteria kelayakan kode yang digunakan dalam penelitian ini, dapat dilihat pada Tabel 4 sebelumnya, nilai 3,84% berada dalam kategori A (0–5%). Dengan demikian, kualitas kode SIMAK Universitas Siliwangi dinilai masuk kategori Layak.

b. Persentase Setiap Jenis Masalah

Persentase setiap jenis masalah dihitung untuk mengetahui proporsi masing-masing jenis masalah (*Bugs*, *Code Smell*, dan *Vulnerability*) terhadap total jumlah masalah yang ditemukan. Meskipun dokumentasi resmi SonarQube tidak secara eksplisit mencantumkan rumus perhitungan ini, pendekatan yang dilakukan merupakan metode statistik umum yang digunakan untuk menganalisis distribusi data berdasarkan kategorinya. Rumus yang digunakan adalah sebagai berikut:

$$\text{Persentase Jenis Masalah} = \left(\frac{Jumlah\ Jenis\ Masalah}{Total\ masalah}\right) x\ 10 \quad (4)$$

- *Bug*

$$Presentase\ Bug = \left(\frac{2.356}{23.241}\right) x\ 100 = 10{,}13\% \quad (5)$$

- *Code Smells*

$$Presentase\ Code\ Smells = \left(\frac{20.885}{23.241}\right) x\ 100 = 89{,}87\% \quad (6)$$

- *Vulnerabilities*

$$Presentase\ Vulnerabilities = \left(\frac{0}{23.241}\right) x\ 100 = 0\% \quad (7)$$

Maka, hasil perhitungannya dapat dilihat pada tabel 6 berikut ini.

Tabel 6. Hasil Persentase Setiap Masalah.

Jenis Masalah	Jumlah	Persentase
Bugs	2.356	10,13%
Code Smells	20.885	89,87%
Vulnerabilities	0	0%
Total	**23.241**	100%

Dari Tabel 6 diatas, dapat dilihat bahwa mayoritas masalah berada pada kategori *Code Smells* sebesar 89,87%, sedangkan masalah pada kategori *Bugs* hanya sebesar 10,13%, dan tidak ditemukan adanya masalah pada kategori *Vulnerabilities*.

c. Persentase Kategori *Severity* (*Critical, Major, Minor*)

Setelah presentase jenis masalah di hitung, tahapan selanjutnya adalah menghitung persentase untuk setiap level *severity* berdasarkan jumlah temuan dalam kategori *Bugs, Code Smells*, dan *Vulnerabilities*. Rumus yang digunakan pada tahap ini merupakan penyesuaian dari persentase jenis masalah, yaitu:

$$Persentase\ Severity = \left(\frac{Jumlah\ Severity}{Total\ masalah}\right) x\ 100 \quad (8)$$

1. *Bug*

Jumlah total masalah *Bugs*: **2.356**

- *Blocker* :

$$\text{Persentase Critical} = \left(\frac{28}{2.356}\right) x\ 100 = 1{,}19\% \quad (9)$$

- *Critical*:

$$\text{Persentase Critical} = \left(\frac{90}{2.356}\right) x\ 100 = 3{,}82\% \quad (10)$$

Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

- *Major*

$$\text{Persentase Major} = \left(\frac{2.100}{2.356}\right) x\ 100 = 89{,}14\% \quad (11)$$

- *Minor*

$$\text{Persentase Minor} = \left(\frac{130}{2.356}\right) x\ 100 = 5{,}52\% \quad (12)$$

2. *Code Smells*

Jumlah total masalah *Code Smells*: **20.885**

- Blocker:

$$\text{Persentase Blocker} = \left(\frac{24}{20.885}\right) x\ 100 = 0{,}11\% \quad (13)$$

- Critical:

$$\text{Persentase Critical} = \left(\frac{11.509}{20.885}\right) x\ 100 = 55{,}13\% \quad (14)$$

- Major:

$$\text{Persentase Critical} = \left(\frac{6.028}{20.885}\right) x\ 100 = 28{,}87\% \quad (15)$$

- Minor:

$$\text{Persentase Minor} = \left(\frac{3.211}{20.885}\right) x\ 100 = 15{,}38\% \quad (16)$$

3. *Vulnerabilities*

Karena tidak ditemukan kerentanan (*vulnerability*), maka persentase untuk semua kategori severity adalah:

- *Blocker: 0%*
- *Critical: 0%*
  Major: 0%
- *Minor: 0%*

Dari seluruh rangkaian hasil diatas, maka perhitungannya dapat dilihat pada Tabel 7 berikut ini.

Tabel 7. Hasil persentase Kategori *Severity*

Jenis Masalah	*Blocker*	*Critical*	*Major*	*Minor*
Bugs	1,19%	3,82%	89,14%	5,52%
Code Smells	0,11%	55,13%	28,87%	15,38%
Vulnerabilities	0%	0%	0%	0%

Dari Tabel 7 diatas, dapat diketahui bahwa pada kategori *Bugs*, mayoritas isu berada pada tingkat *severity Major* sebesar 89,14%, sedangkan sisanya terdiri dari *Blocker* sebesar 1,19%, *Critical* sebesar 3,82%, dan *Minor* sebesar 5,52%. Pada kategori *Code Smells*, sebagian besar masalah berada pada tingkat *severity Critical* sebesar 55,13%, diikuti oleh *Major* sebesar 28,87%, serta *Minor* sebesar 15,38%, sementara *Blocker* hanya sebesar 0,11%. Adapun pada kategori *Vulnerabilities* tidak ditemukan masalah, sehingga tidak ada distribusi *severity* yang dapat dianalisis lebih lanjut.

Berdasarkan hasil analisis menggunakan SonarQube yang telah dilakukan, secara keseluruhan ditemukan sejumlah 23.241 isu, yang terdiri dari 2.356 *bugs* dan 20.885 *code smells*, tanpa adanya temuan *vulnerabilities*. Dengan rasio masalah sebesar 3,84% terhadap total baris kode sebanyak 605.130, serta klasifikasi *severity* yang didominasi oleh isu pada tingkat *Critical* dan *Major*, hasil ini memberikan gambaran mengenai kondisi teknis dari kode yang digunakan dalam SIMAK Universitas Siliwangi.

Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

3.5. Rekomendasi

a. Integrasi Static Code Analysis (SCA) ke dalam Proses DevOps

Dari hasil penelitian yang menunjukkan adanya temuan signifikan pada *code smells* (±89,87%) dan *bugs* (±10,13%) dengan total 23.241 isu, disarankan agar institusi pengembang SIMAK mengintegrasikan SCA seperti SonarQube ke dalam *pipeline* pengembangan perangkat lunak. Dengan integrasi otomatis, deteksi dini dapat dilakukan sebelum kode masuk ke tahap produksi, sehingga risiko keamanan dan biaya perbaikan dapat ditekan.

b. Prioritasi Perbaikan pada *Code Smells* Kritis

Mayoritas masalah berada pada kategori *Critical* (55,13%) yang berpotensi memengaruhi *maintainability* dan keamanan sistem di masa depan. Oleh karena itu, pengembang perlu menyusun roadmap perbaikan bertahap, dimulai dari temuan *Critical* dan *Major*, sebelum menindaklanjuti masalah *Minor* dan *Blocker*, hal ini akan meningkatkan kualitas perangkat lunak secara signifikan.

c. Standarisasi Kualitas Kode melalui Panduan Teknis

Hasil penelitian menunjukkan pentingnya adanya *coding standard* dan panduan teknis yang mengacu pada OWASP Top 10 dan *best practices* pemrograman aman. Dengan adanya pedoman internal, pengembang dapat lebih konsisten dalam menulis kode yang aman dan berkualitas.

d. Pelatihan dan *Capacity Building* bagi Tim Pengembang

Ditemukan bahwa sebagian besar masalah bersifat berulang (duplikasi kode hingga 52,8%). Hal ini menunjukkan perlunya peningkatan kapasitas SDM, baik melalui pelatihan penggunaan tool SCA maupun workshop tentang *secure coding*. Dengan demikian, kompetensi tim dalam pencegahan terjadinya *bugs* dan kerentanan system dapat meningkat.

e. Ekspansi Penelitian ke Tools dan Teknologi Lain

Penelitian yang dilakukan berfokus pada SonarQube, untuk kedepannya perlu dilakukan evaluasi komparatif dengan tools lain seperti Semgrep atau Bandit agar dapat memberikan rekomendasi yang lebih komprehensif terkait kelebihan dan kekurangan setiap *tools* sesuai jenis proyek (PHP, JS, Python).

f. Pengembangan Dashboard Monitoring Kualitas Kode

Untuk mendukung keberlanjutan, disarankan dibuat dashboard internal yang menyajikan metrik kualitas kode (*bugs*, *vulnerabilities, code smells, severity*) secara *real-time*. Dashboard ini dapat menjadi media monitoring dan evaluasi rutin bagi manajemen serta pengembang.

g. Publikasi Ilmiah dan Diseminasi Praktik Baik

Mengingat penelitian ini menghasilkan temuan yang signifikan dan komprehensif, maka publikasi di jurnal nasional maupun internasional perlu terus ditingkatkan. Selain itu, hasil penelitian dapat didiseminasikan kepada komunitas akademik dan pengembang lokal agar memberikan kontribusi luas terhadap praktik secure software development di Indonesia.

4. DISCUSSIONS

Hasil penelitian ini menunjukkan bahwa penerapan SCA menggunakan SonarQube mampu memberikan gambaran yang komprehensif terhadap kualitas dan keamanan kode pada SIMAK Universitas Siliwangi. Temuan sebanyak 23.241 isu yang terdiri dari *bugs* dan *code smells* menunjukkan adanya kebutuhan peningkatan praktik pengembangan perangkat lunak yang lebih terstandar dan berorientasi pada keamanan. Meskipun rasio masalah terhadap total baris kode hanya sebesar 3,84% dan dikategorikan "layak", tingginya jumlah *code smells* dan duplikasi kode menandakan potensi risiko terhadap *maintainability* dan efisiensi sistem di masa mendatang. Dengan demikian, penelitian ini memberikan kontribusi nyata dalam membuktikan efektivitas SCA sebagai instrumen deteksi dini untuk menjaga mutu perangkat lunak di lingkungan akademik. Selain memberikan evaluasi teknis terhadap kondisi kode yang ada, hasil penelitian ini juga menghasilkan rekomendasi strategis berupa integrasi SCA ke dalam proses DevOps, pengembangan dashboard monitoring kualitas kode, serta peningkatan kapasitas pengembang melalui pelatihan *secure coding*. Secara keseluruhan, penelitian ini berdampak pada penguatan tata kelola keamanan perangkat lunak di perguruan tinggi dan dapat menjadi acuan dalam membangun model evaluasi kualitas kode yang berkelanjutan dan terukur.

Beberapa penelitian sebelumnya juga telah menyoroti efektivitas penerapan SCA dalam mendeteksi kelemahan perangkat lunak, terutama di sektor publik dan pendidikan. Berbeda dengan penelitian-penelitian sebelumnya yang umumnya dilakukan pada sistem uji coba atau proyek eksperimental, penelitian ini memiliki kebaruan dalam penerapannya langsung pada sistem yang telah digunakan secara aktif oleh civitas akademika, yakni SIMAK Universitas Siliwangi. Pendekatan ini memungkinkan analisis kualitas dan keamanan kode dilakukan dalam konteks operasional nyata, sehingga hasil yang diperoleh merefleksikan kondisi teknis aktual dan tantangan implementatif di lingkungan pendidikan tinggi. Selain itu, penelitian ini tidak hanya berfokus pada jumlah temuan atau tingkat keparahan, tetapi juga mengintegrasikan hasil analisis dengan klasifikasi kelayakan kode dan rekomendasi strategis yang dapat diterapkan secara langsung untuk memperkuat *secure software governance*. Dengan demikian, penelitian ini memperluas perspektif SCA dari hanya sekedar alat deteksi teknis menjadi instrumen evaluasi berkelanjutan yang mendukung peningkatan kualitas tata kelola dan keamanan perangkat lunak akademik.

5. CONCLUSION

Penelitian ini menegaskan bahwa SCA merupakan pendekatan efektif untuk deteksi dini masalah keamanan dan kualitas perangkat lunak. Implementasi berkelanjutan melalui integrasi ke dalam *pipeline* DevOps, penyusunan pedoman *secure coding*, serta peningkatan kapasitas tim pengembang menjadi langkah penting untuk meningkatkan kualitas dan keamanan sistem informasi akademik secara berkesinambungan. Secara umum, penelitian ini memberikan kontribusi penting terhadap pengembangan ilmu pengetahuan di bidang Informatika dan Ilmu Komputer, khususnya pada ranah *software quality assurance* dan *secure software engineering*. Penelitian ini memperluas pemahaman tentang bagaimana metode analisis statis dapat diimplementasikan secara efektif di lingkungan akademik nyata, bukan hanya dalam konteks eksperimental atau laboratorium.

Namun demikian, penelitian ini masih memiliki keterbatasan pada penggunaan satu alat analisis, yaitu SonarQube, dan satu studi kasus utama, sehingga terbuka peluang bagi penelitian selanjutnya untuk memperluas cakupan objek, melakukan validasi lintas platform, serta mengkaji integrasi SCA dengan pendekatan keamanan berbasis kecerdasan buatan untuk hasil yang lebih adaptif dan komprehensif.

Beberapa arah pengembangan dapat dilakukan untuk penelitian selanjutnya. Pertama, perlu dilakukan studi komparatif terhadap berbagai alat *Static Analysis* seperti Semgrep, Bandit, atau SonarLint untuk mengukur efektivitas relatif dan tingkat akurasi antar platform. Kedua, integrasi hasil

Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

analisis ke dalam dashboard pemantauan kualitas kode berbasis *real-time analytics* dapat dikembangkan guna mendukung pengambilan keputusan manajerial dan teknis dalam tata kelola perangkat lunak. Ketiga, penelitian selanjutnya dapat menggabungkan metode *Static Analysis* dengan *Dynamic Analysis* untuk memperoleh hasil evaluasi yang lebih menyeluruh terhadap performa dan keamanan sistem. Terakhir, secara praktis, hasil penelitian ini dapat dijadikan dasar bagi penyusunan kebijakan *secure coding standard* dan panduan implementasi *Green and Secure Software Development* di lingkungan pendidikan tinggi maupun lembaga pemerintah yang mengelola sistem informasi publik.

CONFLICT OF INTEREST

Dalam penelitian ini penulis menyatakan tidak ada konflik kepentingan terkait hasil penelitian yang dibahas dalam artikel ini.

ACKNOWLEDGEMENT

Ucapan terima kasih yang sebanyak-banyaknya kepada LPPM Uiversitas Siliwangi yang memberikan dana terhadap penelitian ini.

REFERENCES

[1] LLDIKTI Wilayah XVII, Buku Pedoman Penilaian Maturitas Pengelola PDDIKTI 2024, Jan. 2025.
[2] LLDIKTI3, “Pengelolaan pelaporan PDDIKTI di ITPLN,” Materi Sosialisasi, Aug. 2024.
[3] SEVIMA, “Apa itu Sistem Informasi Akademik (SIAKAD)?,” Jun. 2023.
[4] Quipper, “Sistem Informasi Akademik (SIAKAD) - definisi dan fitur,” May. 2024.
[5] Katadata, “Pemerintahan, sektor paling rentan insiden siber,” Jul. 2024.
[6] Naval-CSIRT, “5,6 juta data Kemendikbudristek dibobol,” Oct. 2024.
[7] Verizon, “2025 Data Breach Investigations Report,” May. 2025.
[8] K. Souppaya and K. Scarfone, “NIST SP 800-218: Secure Software Development Framework (SSDF) v1.1,” *Gaithersburg: NIST*, 2022.
[9] K. Rokis and M. Kirikova, “Exploring Low-Code Development: A Comprehensive Literature Review,” *Complex Systems Informatics and Modeling Quarterly*, vol. 2023, no. 36, pp. 68–86, 2023.
[10] Practical DevSecOps, “Comprehensive guide to SAST implementation,” 2023.
[11] D. Patten, “Application security testing in CI/CD pipelines,” 2025.
[12] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, “How developers engage with static analysis tools in different contexts,” *Empir Softw Eng*, vol. 25, no. 2, pp. 1419–1457, Mar. 2020.
[13] W. Charoenwet, S. Charoenwet, and N. Yoshida, “An empirical study of static analysis tools for secure code review,” *arXiv*, 2024.
[14] A. Murali et al., “FuzzSlice: Pruning false positives in static analysis warnings,” *ICSE*, 2024.
[15] Practical DevSecOps, “Comprehensive guide to SAST implementation,” 2023.
[16] M. F. Santoso, “Implementation Of UI/UX Concepts And Techniques In Web Layout Design With Figma,” *Jurnal Teknologi Dan Sistem Informasi Bisnis*, vol. 6, no. 2, pp. 279–285, Apr. 2024.
[17] J. Park, J. Kim, and H. Choi, “Reduction of false positives for runtime errors in C/C++ static analysis,” *Electronics*, vol. 12, no. 16, p. 3518, 2023.
[18] SonarSource Docs, “Quality gates (SonarQube 10.6),” Aug. 2025.
[19] GitHub Docs, “About code scanning with CodeQL,” Mar. 2023.
[20] Oligo Security Academy, “Static code analysis: Methods, pros/cons,” Jul. 2025.

Vol. 6, No. 6, December 2025, Page. 5791-5804
https://jutif.if.unsoed.ac.id
DOI: https://doi.org/10.52436/1.jutif.2025.6.6.5371

[21] ISO, "ISO/IEC 25010:2011—System and software quality models," *Geneva: ISO*, 2011.
[22] OWASP, *OWASP Top 10: 2021*, 2021.
[23] MITRE, "CWE Top 25 Most Dangerous Software Weaknesses," 2024.
[24] Z. Wadhams, "Barriers to using SAST tools: A literature review," Montana State Univ., 2024.
[25] G. Liargkovas, M. Papadakis, and A. Zeller, "A study of static analysis alert suppressions," *arXiv*, 2023.
[26] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, "How developers engage with static analysis tools in different contexts," *Empir Softw Eng*, vol. 25, no. 2, pp. 1419–1457, Mar. 2020.