**IT-EXPLORE**
*Jurnal Penerapan Teknologi Informasi dan Komunikasi*
**ISSN 2828-7940, e-ISSN 2829-1727**
**Volume 02 Nomor 02 Tahun 2023**

# ANALISIS *QUALITY CODE* MENGGUNAKAN *SONARQUBE* DALAM SUATU APLIKASI BERBASIS LARAVEL

**Ajeng Febriana Rahmawati[1], Yeremia Alfa Susetyo[2]**

[1, 2] Program Studi Teknik Informatika, Fakultas Teknologi Informasi
Universitas Kristen Satya Wacana
Jl. Dr. O. Notohamidjodjo No. 1-10, Blotongan, Salatiga, Jawa Tengah 50715
Email : 672019323@student.uksw.edu[1], yeremia.alfa@uksw.edu[2]

**Riwayat artikel:**
*Submitted: 31-03-2023* *Revised: 14-04-2023* *Published: 15-06-2023*

***Abstrak –*** Analisis Kode Statis (*Static Code Analysis*) masuk ke dalam metode *white box testing* yang digunakan pada pengembangan aplikasi. Analisis ini dilakukan dengan cara mengevaluasi *source code* untuk mendeteksi pelanggaran kode apa saja yang terdapat dalam *project* aplikasi *open source To Do List* berbasis *website* menggunakan *Laravel* dan *Vue.js*. Proses analisis kode statis bertujuan memberikan pemahaman serta memastikan *basic code* pada *project* telah memenuhi standarisasi pengkodean yang telah ditetapkan. Teknik pengujian *source code* dilakukan melewati proses *scanning* pada *project* dengan standarisasi pengkodean yang telah ditetapkan dan diatur dalam *Quality Gate SonarQube* sebagai acuan untuk seberapa tinggi kualitas kode yang harus dilalui. Hasil analisis yang diperoleh melalui *tool SonarQube* menunjukkan pelanggaran kode yang ditemukan secara keseluruhan pada *project To Do List* dengan beberapa kategori *issues*, yaitu 4 *bug*, 2 *security hotspots*, 31 *code smell*, dan 117 *duplicate lines of code* dengan *density* sebesar 18,3% yang membutuhkan total waktu estimasi pengerjaan untuk perbaikan pelanggaran kode sebanyak 2 jam 35 menit.

***Kata Kunci –*** analisis kode, standarisasi pengkodean, pelanggaran kode, *SonarQube*

***Abstract –*** *Static Code Analysis is included in the white box testing method used in application development by evaluating the source code to detect any code violations contained in website-based open source To Do List application projects using Laravel and Vue.js. The static code analysis process aims to provide understanding and ensure that the basic code on the project meets the established coding standards. The source code testing technique is carried out through the scanning process on the project with predetermined coding standardization and is set in the SonarQube Quality Gate as a reference for how high the quality of the code must be passed. The results of the analysis obtained through the SonarQube tool showed code violations found in the To Do List project with several categories of issues, namely 4 bugs, 2 security hotspots, 31 code smells, and 117 duplicate lines of code with a density of 18,3% which requires a total estimated work time to repair code violations of 2 hours 35 minutes.*

***Keywords –*** *code analysis, coding standardization, code defect, SonarQube*

# I. PENDAHULUAN

Analisis kode statis (*Static Code Analysis*) atau bisa juga disebut dengan analisis statis merupakan metode *debugging* program dan pengevaluasian *source code* yang dapat dilakukan dengan cara memeriksa *source code* tanpa menjalankan program[1]. Analisis kode statis termasuk ke dalam metode *white box testing*. Proses analisis kode statis memberikan pemahaman tentang *basic code* yang terdapat di dalam suatu *project*, membantu untuk memastikan bahwa *basic code* di dalam sebuah *project* telah memenuhi standar pengkodean yang sudah ditetapkan, aman, dan terlindungi.

*Project* yang diuji merupakan aplikasi *website* dengan menggunakan *Laravel* dan *Vue.js*. Setelah kode dibuat akan dilakukan pemeriksaan *source code* dengan menjalankan *tool* untuk menganalisis kode program pada aplikasi. *Tool* analisis kode statis yang juga termasuk ke dalam *Static Application Security System* (SAST) menjadi semakin populer sebagai pendeteksi kemungkinan sumber kerentanan di proses *development*[2]. Penggunaan *tool* untuk menganalisa kode dapat membantu untuk mempermudah pengujian *source code*. Terlebih lagi jika pengujian dilakukan untuk *source code* pada aplikasi yang kompleks maka akan sulit untuk menganalisa pelanggaran kode secara manual. Analisis kode dilakukan oleh *developer* dengan mengacu pada operasi *static analysis tools* yang merupakan *tool* pengujian *source code* untuk mengotomatisasikan proses agar menjadi lebih efisien untuk menemukan *issues* pada kode program aplikasi[3]. Dengan mengetahui pelanggaran kode yang terdeteksi di dalam *project* digunakan sebagai acuan untuk melakukan peningkatan kualitas kode sesuai standarisasi pengkodean yang telah ditetapkan. Maka dari itu, *developer* dapat menyesuaikan *source code* yang belum sesuai dengan standar yang ada. Setelah ketidaksesuaian kode telah memenuhi standar, *source code* dapat beralih ke proses *deployment*.

Analisis kode dalam penelitian ini dilakukan bertujuan mempermudah *developer* menemukan *issues* yang terdeteksi pada *source code* saat proses pengembangan aplikasi. Oleh karena itu, proses analisis kode dilakukan dengan menggunakan *tool Static Application Security System* (SAST) untuk membantu *developer* memperbaiki pelanggaran kode yang ditemukan pada kode program aplikasi. Sehingga, hasil analisis pada kode aplikasi yang ditemukan dapat melewati standar yang telah diatur. Dengan demikian, *developer* dapat menghasilkan *source code* dengan kualitas dan keamanan yang memenuhi standarisasi pengkodean agar dapat bekerja secara efektif dan optimal. Selain itu, melalui *tool Static Application Security System* (SAST) juga dapat dilakukan pengujian *source code* secara berkala untuk *maintenance* kode program secara menyeluruh di setiap pembaruan kode baru yang dirilis.

# II. KAJIAN PUSTAKA

## 2.1. Peneliti Terdahulu

Penelitian yang membahas mengenai *Implementasi Static Application Security Testing Menggunakan Jenkins CI/CD Berbasis Docker Container Pada PT. Emporia Digital Raya* menjelaskan jika perkembangan teknologi memberikan dampak yang positif terutama dalam proses sehingga *deployment* kerentanan dalam sebuah aplikasi dapat diatasi. Dengan menggunakan *Static Application Security Testing* (SAST) yang merupakan salah satu proses di dalam *DevSecOps* bertujuan guna mempercepat

*development* dan mengurangi kerentanan suatu kode program pada sebuah aplikasi. Berdasarkan penelitian tersebut dapat disimpulkan bahwa analisis kode statis dengan menggunakan *static analysis tools* dapat mempercepat aplikasi yang dikembangkan dalam proses *development* dan juga meminimalisir kerentanan yang terdeteksi dikarenakan *Static Application Security Testing* (SAST) membantu untuk mengatasinya[4].

Dalam penelitian yang membahas *Mining fix Patterns for FindBugs Violations* dijelaskan bahwa analisis kode statis dilakukan dengan menggunakan *static analysis tools*. *Project* yang diuji memiliki *basic code* berskala besar, dimana *static analysis tools* berfungsi untuk mendeteksi kesalahan yang dilakukan di awal proses *development* yang kemudian melaporkan hasilnya kepada *developer*. Pada penelitian tersebut juga menyebutkan bahwa *static analysis tools* bekerja secara statis dimana program aplikasi tidak perlu dijalankan saat *tools* sedang melakukan eksekusi pada program yang diuji. Berdasarkan penelitian tersebut maka dapat disimpulkan bahwa teknologi yang digunakan di dalam *static code analysis* sangat berdampak besar terutama pada proses *development* guna mempercepat pekerjaan *developer* untuk meningkatkan kualitas kode agar dapat dilanjutkan ke tahap *deployment*[5].

Dalam sebuah penelitian *A Case Study Comparing Static Analysis Tools for Evaluating SwiftUI Projects* membahas tentang *static analysis tools* yang digunakan yaitu *SonarQube* dan *Codacy* untuk dibandingkan dalam mengevaluasi sebuah *project SwiftUI*. Pada penelitian tersebut melakukan pengujian untuk mengetahui apakah terdapat perbedaan terkait hasil yang dilaporkan pada kode program yang diuji. Berdasarkan penelitian tersebut dapat disimpulkan bahwa *tools SonarQube* lebih sukses daripada *Codacy* dalam mengidentifikasi kode duplikat. Namun, UI/UX *SonarQube* yang diperlihatkan tidak ramah bagi pemula. Penelitian dengan menggunakan *SonarQube* memiliki hasil metriks yang luas sehingga agak rumit untuk penggunaannya. Tidak seperti *Codacy* yang memiliki UI/UX lebih nyaman diperlihatkan untuk pemula. Selain itu, *Codacy* tidak seperti *SonarQube* yang memiliki *quality gate* untuk *maintenance* kode program secara berkala[6].

Pada penelitian terdahulu dijelaskan jika *tool Static Application Security Testing* (SAST) paling populer yang digunakan oleh *developer open source* adalah *Findbugs, Checkstyle*, dan *SonarQube*. Di antara *tools* yang disebutkan, *SonarQube* merupakan satu-satunya *tools* yang menyediakan indeks *Technical Debt* untuk mendapatkan estimasi waktu perbaikan pada pelanggaran kode[7]–[9].

*SonarQube* memuat metriks yang bermacam-macam untuk mengidentifikasi *quality code* seperti yang ditampilkan pada penelitian-penelitian terdahulu. Namun, setiap hasil dari penelitian terdahulu tidak menyebutkan secara menyeluruh metriks apa saja yang dimiliki *SonarQube* untuk fitur *quality code.* Oleh karena itu, pada penelitian ini akan dijelaskan apa saja metriks yang dimiliki oleh *SonarQube* secara mendetail untuk mendapatkan hasil analisis *quality code* pada *source code* aplikasi berbasis *website* yang menggunakan *Laravel.* Matriks yang dijelaskan terdapat *Bug, Vulnerability, Security Hotspots, Maintainability,* hingga *Duplicate code*. Dimana pada penelitian terdahulu tidak mencantumkan metriks *Duplicate code* secara detail.

## 2.2. Analisis Kode Statis

Analisis kode statis adalah sebuah metode *debugging* program yang dilakukan untuk memeriksa kode tanpa menjalankan program. Proses tersebut memberikan pemahaman tentang struktur kode dan juga dapat membantu memastikan jika kode tersebut telah memenuhi standarisasi. Analisis kode statis digunakan dalam *software engineering* oleh tim *software development* dan *quality assurance*. *Tools* otomatis dapat membantu *developer* dalam melakukan analisis kode statis dengan melakukan *scanning* semua kode dalam *project* guna memeriksa kerentanan saat memvalidasi kode[10].

### *2.3. SonarQube*

*SonarQube* adalah *platform open-source* yang dikembangkan oleh *SonarSource*, digunakan untuk inspeksi berkelanjutan pada kualitas kode aplikasi. Lebih dari itu, *SonarQube* juga dapat melakukan analisis kode secara statis, yang mana menyediakan laporan lebih detail terkait *bug, code smells, vulnerability* hingga *duplication code*. *SonarQube* didukung dengan 30+ bahasa pemrograman yang terbagi ke dalam beberapa versi, yaitu 17 bahasa pemrograman untuk *Community Edition*, 24 bahasa pemrograman *Developer Edition*, 29 bahasa pemrograman untuk *Enterprise* dan *Data Center Edition*. *SonarQube* mengklasifikasikan aturan ke dalam 5 tingkatan, yaitu *Blocker, Critical, Major, Minor*, dan *Info*. Setiap aturan pengkodean dalam *SonarQube* mendefinisikan sebuah *Quality Gate*. *Quality Gate* akan menentukan status *Passed* atau *Failed* sebuah *source code* dan menampilkan penilaian untuk masalah yang ditemukan[11], [12].

### *2.4. Laravel*

*Laravel* merupakan *framework* aplikasi *website* dengan *syntax* yang ekspresif dan elegan. Kerangka kerja pada web menyediakan struktur dan titik awal membuat aplikasi dengan detail yang merupakan bawaan dari *Laravel*. Fitur-fitur yang disediakan seperti *dependency injection* menyeluruh, layer abstraksi *database* ekspresif, antrian dan pekerjaan terjadwal, pengujian unit dan integrasi, dan banyak lagi[13].

### *2.5. Vue.js*

*Vue.js* merupakan sebuah *framework JavaScript* untuk membangun *user interface*. *Vue.js* dibangun di atas HTML standar, CSS, dan *JavaScript*. *Vue.js* menyediakan model pemrograman deklaratif dan berbasis komponen yang dapat mengembangkan *user interface* secara efisien, baik sederhana maupun kompleks[14].

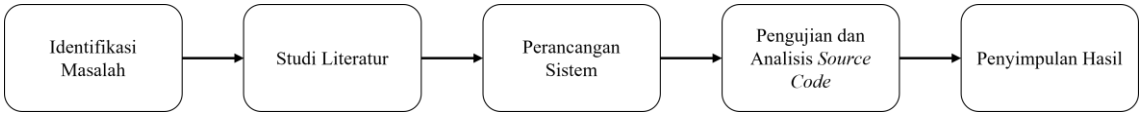
### *2.6. PostgreSQL*

*PostgreSQL* merupakan *Relational Database Management System* (RDBMS) yang bersifat *open source*. *PostgreSQL* menawarkan sebuah sistem basis data yang dilengkapi dengan fitur-fitur yang dapat membantu *developer* dalam membangun aplikasi seperti dapat membuat tipe data dan membuat *custom function*. Bahkan pada *PostgreSQL* kita dapat menuliskan kode dari bahasa pemrograman lain tanpa *recompiling* ulang *database*. Dengan adanya fitur-fitur tersebut, diharapkan RDBMS yang dikembangkan oleh *Berkeley Computer Science Department* ini dapat mempermudah para pemakai dalam mengimplementasikan sistem ini serta mencukupi kebutuhan proses aplikasi data masa di masa yang akan datang[15].

## III. METODE PENELITIAN

Ada beberapa tahapan yang akan dilakukan dalam penelitian Menganalisis *Code Quality* Aplikasi Berbasis *Laravel* Menggunakan *SonarQube*. Tahapan penelitian yang akan digunakan dapat dilihat pada Gambar 1.




Gambar 1. Tahapan Penelitian

Tahapan penelitian ini menggambarkan langkah-langkah untuk melakukan pengujian kualitas kode pada *project To Do List*, diantaranya :

1. Identifikasi Masalah

   Pada tahapan awal ini dilakukan identifikasi masalah pada proses pengujian *source code* dengan menggunakan *System Application Security Testing* (SAST). Masalah yang ditemukan merupakan hasil dari minimnya analisis kode yang ditemukan dengan menggunakan salah satu *tool* SAST.

2. Studi Literatur

   Tahap selanjutnya dilakukan studi literatur yang dibutuhkan dalam proses penelitian analisis *code quality* berdasarkan pada permasalahan pelanggaran kode yang ditemukan dari hasil pengujian *project* aplikasi *open source* berbasis *website* yang menggunakan *Laravel* dan *Vue.js*.

3. Perancangan Sistem

   Perancangan sistem dilakukan dengan membangun *server SonarQube* sebagai media pendukung pada penelitian. Dilakukan konfigurasi *requirements* pada *server SonarQube* yang terdapat pada *virtual machine* supaya *server* dapat dijalankan secara stabil. *Server SonarQube* memiliki data yang tersimpan pada *server database PostgreSQL.* Data hasil analisis dari *SonarQube* ditampilkan melalui *dashboard SonarQube* yang diakses melalui *browser website*.

4. Pengujian dan Analisis *Source Code*

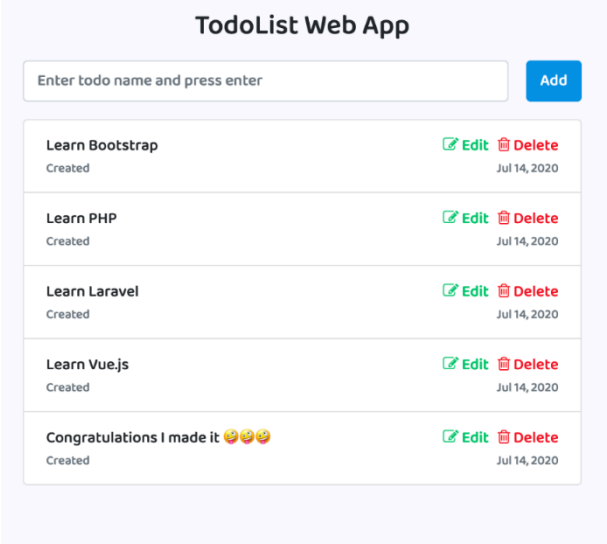
   Setelah sistem berhasil diimplementasikan, pada tahap ini dilakukan pengujian dan analisis melalui proses *scanning* pada *project* untuk menampilkan *report* berdasarkan pada standar pengkodean yang telah ditetapkan di dalam *Quality Gate* guna menguji lulus atau tidaknya kualitas kode program pada *project* aplikasi.

5. Penyimpulan Hasil

   Setelah proses pengujian berhasil dilakukan pada aplikasi dan mendapatkan hasil analisis *source code*, maka peneliti menyimpulkan hasil penelitian ini dalam bentuk laporan.

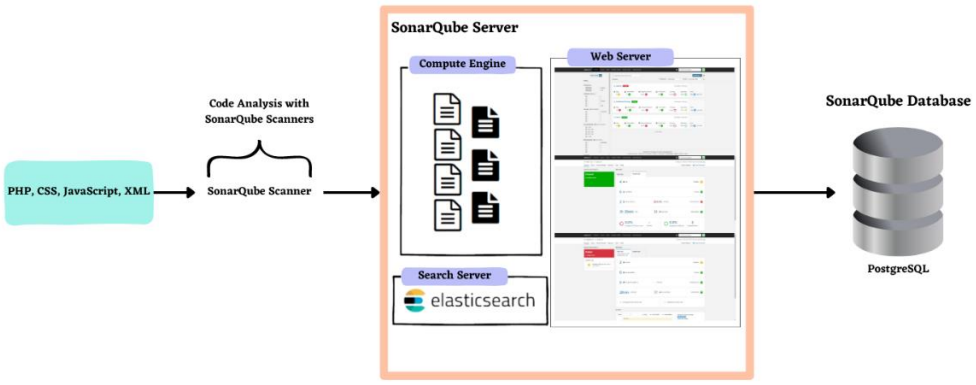
## IV. HASIL DAN PEMBAHASAN

Pada penelitian ini mengambil data *source code* pada sebuah aplikasi *To Do List.* Aplikasi *To Do List* dipublikasikan secara *open source* melalui GitHub sehingga peneliti dapat mengakses *source code* pada *repository* aplikasi *To Do List* secara bebas.




Gambar 2. Aplikasi *To Do List*

Sampel pengujian pada penelitian ini merupakan *source code* aplikasi *website* berbasis *framework Laravel* dengan *front end* menggunakan *Vue.js* untuk menghasilkan *Single Page Application* (SPA) serpeti pada Gambar 2. Bahasa pemrograman yang terdeteksi digunakan dalam aplikasi ini yaitu PHP, CSS, *JavaScript*, dan XML. *Source code* pada aplikasi mendapatkan perlakuan proses pengujian melalui *tool SonarQube* sesuai dengan ketentuan dari bahasa pemrogramannya[16], [17].



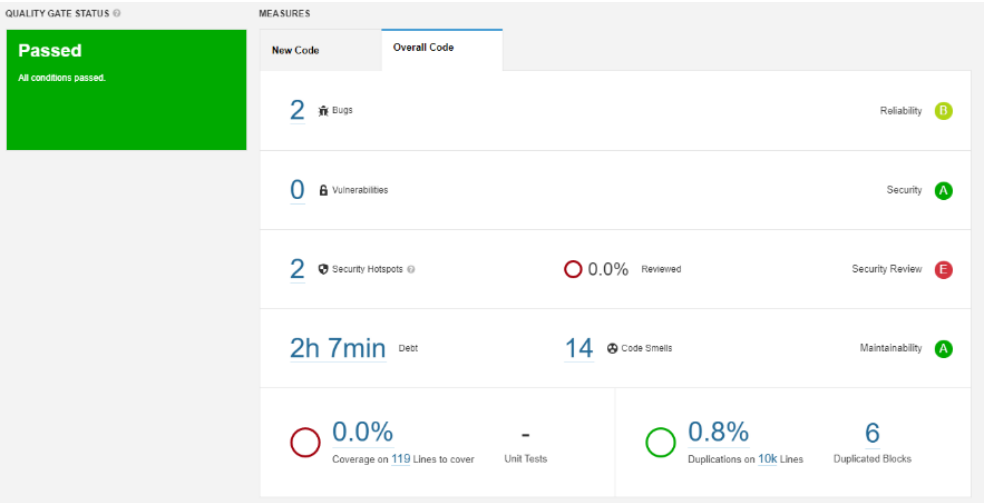

Gambar 3. Arsitektur *SonarQube*

Sesuai dengan aplikasi *To Do List,* maka visualisasi arsitektur *SonarQube* dapat dilihat pada Gambar 3 dengan komponen yang terdiri dari Bahasa Pemrograman, *SonarQube Scanner*, *SonarQube Server*, serta *SonarQube Database*. Arsitektur *SonarQube* dirancang guna membuat *server* yang dijalankan di dalam *virtual machine*. Terdapat *requirements* untuk merancang sistem yang digunakan agar penelitian dapat dilakukan. *Requirements* pada *virtual machine* yang digunakan dalam penelitian ini menggunakan sistem operasi *Linux* versi *Red Hat* 8.6 yang memiliki minimum 2GB

RAM dan 2 prosesor CPU agar *server* berjalan secara efisien. Selain itu, *requirements* yang dibutuhkan di dalam *server* antara lain[18] :

1. *Java Library*. Pada penelitian ini untuk mendukung *platform* pada *SonarQube* dibutuhkan OpenJDK 11 atau JRE 11.
2. *Database*. Dalam penelitian ini menggunakan *PostgreSQL* versi 13 yang digunakan sebagai menampung data laporan *scanning* yang telah dieksekusi.
3. *Browser website*. Dalam penelitian ini menggunakan *Google Chrome*, namun *browser* lain yang dapat digunakan seperti *Microsoft Edge, Safari, Mozilla Firefox*, atau *Opera*. *Browser* ini digunakan sebagai tampilan visualisasi laporan analisis *scanning* yang telah selesai dieksekusi.
4. *SonarQube* merupakan *tools* utama yang dibutuhkan pada penelitian ini. *Server SonarQube* yang digunakan adalah *Community Edition* versi 8.9.9 dan *SonarScanner* versi 4.7.0 sebagai *scanner* untuk *source code* yang akan diuji[19].

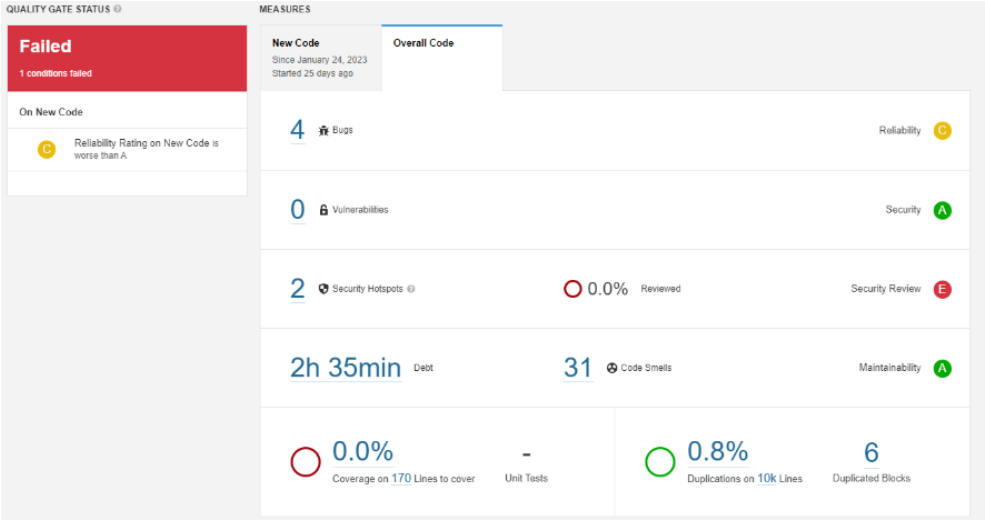
Setelah *requirements* berhasil terkonfigurasi dengan baik, aplikasi akan dieksekusi melalui proses scanning yang dilakukan oleh *SonarScanner* di dalam *server SonarQube*. Hasil eksekusi proses *scanning* akan ditampilkan dalam *dashboard* web *SonarQube* dengan *Elasticsearch* untuk mengeksekusi permintaan *User Interface* (UI) yang diakses melalui *domain ip address* dengan *default port* 9000 dimana data akan tersimpan di dalam *database PostgreSQL*[20]. *Report* analisis yang dihasilkan pada pengujian *scanning project* pertama kali dapat terlihat seperti Gambar 4.




Gambar 4. *Report* Analisis *Project To Do List*

Laporan hasil analisis dari proses *scanning SonarQube* merupakan temuan keseluruhan dari pengujian pada *project To Do List*. Menurut hasil pada Gambar 4, status *Quality Gate* pada *Overall Code* dinyatakan *Passed* dan berwarna hijau. Hal ini disebabkan penilaian pada hasil analisis di berbagai metriks yang telah dijalankan oleh *SonarScanner*. Terdapat 2 *Bugs* yang ditemukan pada *code base*, sehingga mendapatkan status B pada *Reliability*. Terdapat 2 *Security Hotspots* yang ditemukan sehingga menghasilkan status E pada *Security Review*. Dan kemudian yang terakhir, ditemukan 14 *Code Smells* dengan perkiraan waktu pengerjaan untuk perbaikan selama 2 jam 7 menit

dengan menghasilkan status A pada *Maintainability*. Namun, terdapat *warnings* pada hasil *scanning* pertama ini sehingga perlu dilakukan pengujian kembali. *Report* analisis yang dihasilkan dari pengujian kedua dapat dilihat pada Gambar 5.




Gambar 5. *Report* Analisis *Project To Do List*

Menurut dari *report* yang dihasilkan pada Gambar 5, status *Quality Gate* berubah menjadi *Failed* dan berwarna merah pada *On New Code*. Ditemukan status C pada *Reliability Rating* pada *On New Code* dikarenakan *Quality Gate* yang ditetapkan adalah A. Terdapat beberapa perbedaan yang dihasilkan setelah melakukan pengujian kedua. Dalam kondisi *Overall Code* ditemukan 2 *Bugs* pada pengujian pertama, namun pada pengujian kedua menjadi 4 *Bugs* yang menghasilkan status C pada *Reliability*. Juga ditemukan penambahan jumlah *Code Smells* yang semula berjumlah 14 *issues* dengan estimasi waktu 2 jam 7 menit menjadi berjumlah 31 *issues Code Smell* dengan perkiraan pengerjaan untuk perbaikan yaitu 2 jam 35 menit.

Dari perbedaan analisis yang dihasilkan pada pengujian pertama dan kedua dapat disimpulkan jika perubahan yang terjadi di dalam *project* dapat mempengaruhi kualitas kode yang didapatkan. Menurut Tollin dkk. *directory* yang terkena lebih banyak pelanggaran kode memiliki kerentanan dengan upaya perbaikan yang lebih besar. Namun, Tollin dkk. tidak memprioritaskan atau mengklasifikasikan pelanggaran kode yang paling berubah[21]. Selama kualitas kode belum memenuhi target standar kualitas yang ditentukan, maka perbaikan dan perubahan kode harus tetap dilakukan sampai kualitas kode dapat melewati standar pengkodean yang telah ditetapkan. Pada Tabel 1 ditampilkan hasil analisis berdasarkan pengujian tiap *directory* pada *project To Do List.*

**Tabel 1** Informasi Setiap *Directory* dalam *To Do List*

| Project Name | Bug | Vulnerability | Security Hotspots | Maintainability | | Duplicate |
|---|---|---|---|---|---|---|
| | | | | Code Smell | Technical Debt | |
| app | 0 (A) | 0 (A) | 0 (A) | 6 (A) | 13min (A) | 0% |
| bootstrap | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0% |
| config | 0 (A) | 0 (A) | 1 (E) | 2 (A) | 18min (A) | 0% |

| | | | | | | |
|---|---|---|---|---|---|---|
| database | 0 (A) | 0 (A) | 0 (A) | 2 (A) | 5min(A) | 0% |
| node_mod | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0% |
| public | 2 (C) | 0 (A) | 0 (A) | 7 (A) | 7min(A) | 0% |
| resources | 2 (B) | 0 (A) | 1 (E) | 13 (A) | 1h 47min (A) | 18,3% |
| routes | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0% |
| screenshots | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0% |
| storage | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0% |
| tests | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0 (A) | 0% |
| phpunit.xml | 0 (A) | 0 (A) | 0 (A) | 1 (A) | 5min (A) | 0% |

Penemuan *issues* pada Tabel 1 menunjukkan jika pada metriks *Bug* terdeteksi *issues* di dalam *directory public* dengan jumlah 2 *Bug* berjenis *Major* yang mendapatkan status C dan *resources* berjumlah 2 *Bug* berjenis *Minor* dengan status B. Pada metriks *Vulnerability* tidak ditemukan *issues*. Metriks *Security Hotspots* terdeteksi 1 *issue* pada masing-masing *directory config* dan *resources* dengan status E. Pada metriks *Maintainability* terdeteksi *issues* di dalam *package app* dengan jumlah 6 *Code Smell* yang diestimasikan untuk diperbaiki dalam waktu 13 menit, *config* dengan 2 *Code Smell* mendapatkan estimasi waktu 18 menit, *database* dengan 5 *Code Smell* mendapatkan estimasi waktu 5 menit, *public* dengan 7 *Code Smell* mendapatkan estimasi waktu 7 menit, dan *resources* dengan 13 *Code Smell* mendapatkan estimasi waktu 1 jam 47 menit. Selain itu, terdapat *issues* metriks *Maintainability* pada *file phpunit.xml* dengan 1 *Code Smell* yang diestimasikan dalam waktu 5 menit. Dari penemuan *issues* pada setiap *directory* diketahui jika *directory resources* mendapatkan penemuan pelanggaran kode paling banyak dan didukung dengan terdeteksi sebanyak 18,3% kepadatan duplikasi kode dari keseluruhan *directory resources*.

Metriks *Bug* merupakan kerusakan teknis dalam aplikasi yang menyebabkan resiko program aplikasi menjadi macet atau *output* tidak valid yang memicu *error*[22]. Gambar 6 menunjukkan rincian analisis *Bug* yang ditemukan.




Gambar 6. Rincian *Issues Bug*

Pada Gambar 6 telah dirincikan apa saja *issues* yang terdeteksi dalam metriks *Bug*. Dimana 2 *Bug* masuk ke dalam kategori *Major* dan 2 *Bug* lainnya masuk ke dalam kategori *Minor*. Untuk kategori *Major* mengacu pada adanya *duplication* dan hilangnya *function* di dalam kode program. *Major* dapat menyebabkan kemungkinan kesalahan atau kegagalan ketika *function* masing-masing dipanggil, namun, untuk fungsionalitas tidak akan berfungsi seperti yang diharapkan atau tidak akurat. Untuk kategori *Minor* mengacu pada *Bug* HTML yang kemungkinan dapat menyebabkan sedikit atau bahkan tidak ada

kesalahan fungsional, tetapi kesalahan *cosmetic* yang mempengaruhi *User Interface* (UI) atau tampilan *front end* aplikasi.

Kemudian terdapat *issues* pada metriks *Security Hotspots* yang meninjau bagian kode yang peka terhadap keamanan untuk mengetahui ada atau tidaknya ancaman pada keamanan kode. Semakin kecil ancaman yang ditemukan maka semakin aman sebuah kode jika terjadi serangan seperti yang ditampilkan pada Tabel 2.

**Tabel 2** Rincian Analisis *Security Hotspots*

| Lokasi Sampel | Kode Sampel | Jenis | Tingkatan |
|---|---|---|---|
| config/cors.php | 'allowed_origins' => ['*'], | *Insecure Configuration* | *Low* |
| resources/views/todos.blade.php | <script src=https://use.fontawesome.com/186d66fd50.js> </script? | *Others* | *Low* |

Hasil analisis Tabel 2 terdapat dua *review priority* level *low*. Salah satu penemuan *Security Hotspots* masuk ke dalam kategori *insecure configuration* dimana terdeteksi untuk melakukan peninjauan kemanan pada kebijakan permissive CORS (*Cross-Origin Resource Sharing*) yang menyebabkan kerentanan CVE-2018-0296 dan CVE-2017-14460. Kebijakan yang sama pada *browser* mencegah secara *default* untuk keamanannya, *front end JavaScript* untuk melakukan *request* HTTP dari *resources* asal ke *resources* yang berbeda (*domain, protocol*, atau *port*). Target yang diminta dapat menambahkan HTTP sebagai *response* yang disebut CORS, yang bertindak seperti arahan untuk *browser* dan mengubah kebijakan *access control*. Penemuan *Security Hotspots* lainnya masuk ke dalam kategori *others* dimana terdeteksi pengambilan *resources* eksternal dari CDN. Penggunaan fitur *resource integrity* yang tidak terverifikasi dapat memengaruhi keamanan aplikasi. *Resources* akan terblokir oleh fitur *resource integrity* jika tidak cocok sehingga perlu ditinjau untuk tidak menggunakannya.

Kemudian, pada metriks *Maintainability* ditemukan *issues* yang berisi *Code Smell* dan *Technical Debt. Code smell* terdeteksi karena terdapat suatu permasalahan dalam pola struktur bagian dari kode program[23]. Namun, Tufano dkk. menunjukkan bahwa hampir 80% dari *Code Smell* tidak pernah dihapus dari kode program. Jika *Code Smell* ingin dihapus, maka dengan cara menghilangkan modifikasi *Code Smell*[24]. Palomba dkk. menunjukkan bahwa *programmer* harus meninjau kembali *Code Smell* yang ditemukan dan melakukan *refactoring* guna meningkatkan *maintenance* kode secara keseluruhan[25]. Hasil analisis *Maintainability* ditunjukkan pada Tabel 3.

**Tabel 3** Rincian Analisis *Maintainablity*

| Deskripsi | Nomor Baris | Jenis | Debt |
|---|---|---|---|
| app/Console/Kernel.php | | | |
| *Remove this commented out code.* | L27 | *Major* | 5min |
| app/Exceptions/Handler.php | | | |
| *Remove this method "report" to simply inherit it.* | L37 | *Minor* | 4min |
| *Remove this method "render" to simply inherit it.* | L51 | *Minor* | |
| app/Http/Controllers/TodoItemController.php | | | |
| *Complete the task associated to this "TODO" comment.* | L56 | *Info* | - |

| | | | |
|---|---|---|---|
| app/Providers/EventServiceProvider.php | | | |
| *Remove this method "boot" to simply inherit it.* | L28 | *Minor* | 2min |
| app/Providers/RouteServiceProvider.php | | | |
| *Remove this method "boot" to simply inherit it.* | L31 | *Minor* | 2min |
| config/database.php | | | |
| *Define a constant instead of duplicating this literal "127.0.0.1" 4 times.* | L49 | *Critical* | 10min |
| config/logging.php | | | |
| *Define a constant instead of duplicating this literal "logs/laravel.log" 4 times.* | L46 | *Critical* | 8min |
| database.migrations/2020_07_14_073502_create_todo_items_table.php | | | |
| *Complete task associated to this "TODO" comment.* | L27 | *Info* | - |
| database/seeds/DatabaseSeeder.php | | | |
| *Remove this commented out code.* | L14 | *Major* | 5min |
| phpunit.xml | | | |
| *Remove this commented out code.* | L25 | *Major* | 5min |
| public/css/app.css | | | |
| *Unexpected duplicate selector "hr", first used at line 82* | L449 | *Major* | |
| *Unexpected duplicate selector "pre", first used at line 201* | L558 | *Major* | |
| *Unexpected duplicate selector ".dropright.dropdown-toggle::after", first used at line 3541* | L3556 | *Major* | |
| *Unexpected duplicate selector ".dropleft.dropdown-toggle::after", first used at line 3568* | L3575 | *Major* | |
| *Unexpected duplicate selector ".dropleft.dropdown-toggle::before", first used at line 3579* | L3593 | *Major* | 7min |
| *Unexpected duplicate selector ".navbar-expand > .container, .navbar-expand > .container-fluid, .navbar-expand > .container-sm, .navbar-expand > .container-md, .navbar-expand > .container-lg, .navbar-expand > .container-xl", first used at line 4826 (no-duplicate-selectors)* | L4849 | *Major* | |
| *Unexpected duplicate selector ".breadcrumb-item + .breadcrumb-item::before", first used at line 5231* | L5235 | *Major* | |
| resources/js/app.js | | | |
| *Remove this commented out code.* | L20 | *Major* | 5min |
| resources/js/bootstrap.js | | | |
| *Remove this commented out code.* | L32 | *Major* | 15min |
| *Remove this commented out code.* | L34 | *Major* | |
| *Remove this commented out code.* | L41 | *Major* | |
| resources/js/components/TodoItem.vue | | | |
| *Unexpected empty source* | L12 | *Major* | 1min |
| resources/js/components/TodoList.vue | | | |
| *Complete task associated to this "TODO" comment.* | L163 | *Info* | |
| *Complete task associated to this "TODO" comment.* | L174 | *Info* | |
| *Complete task associated to this "TODO" comment.* | L208 | *Info* | - |
| *Complete task associated to this "TODO" comment.* | L243 | *Info* | |
| *Complete task associated to this "TODO" comment.* | L265 | *Info* | |
| resources/lang/en/validation.php | | | |
| *6 duplicated blocks of code must be removed.* | - | *Major* | |
| *Define a constant instead of duplicating this literal "The selected :attribute is invalid." 3 times.* | L44 | *Critical* | 1h 26min |
| *Define a constant instead of duplicating this literal "The :attribute format is invalid." 3 times.* | L94 | *Critical* | |

Hasil analisis Tabel 3 telah dirincikan temuan pelanggaran kode pada *Maintainability.* Terdapat 4 *issues* masuk ke kategori *Critical* yang dapat menyebabkan kelemahan keamanan aplikasi sehingga harus segera ditinjau untuk memperbaiki kode program. Kemudian, 16 *issues* masuk ke kategori *Major* sehingga mengakibatkan penurunan kualitas dan ketidakakuratan *source code* yang disebabkan oleh kesalahan pada saat pembuatan kode program sehingga memasukkan kode yang sebenarnya tidak diperlukan. Untuk 4 *issues* lainnya masuk ke kategori *Minor* yang menyebabkan sedikit *error* yang *redundant* atau berulang. Sehingga bukan termasuk ke dalam perbaikan yang segera ditangani, namun tetap perlu dilakukan peninjauan untuk mengurangi *error* yang ditemukan. Dan tersisa 7 *issues* yang masuk ke kategori *Info* dimana temuan ini tidak didefinisikan sebagai pelanggaran kode sehingga tidak mempengaruhi kualitas kode pada *source code*. Selain itu, *Maintainability* terdapat *Technical Debt* yang digunakan sebagai acuan estimasi waktu untuk memperbaiki *Code Smell* yang ditemukan[26]. Pada penelitian ini, *Technical Debt* yang tertera dalam Tabel 3 memiliki perbedaan jumlah waktu dalam satu kategori dikarenakan *Technical Debt* dipengaruhi oleh cara pengerjaan yang berbeda untuk setiap *issues* yang ditemukan[27].

Kemudian, penemuan terakhir yang terdeteksi yaitu *Duplication.* Duplikasi kode merupakan masalah umum pada kualitas kode yang dapat mempersulit *maintenance* kode program, meningkatkan risiko kemunculan *bug,* hingga berdampak pada penurunan performa. Untuk menemukan duplikasi, kode dibandingkan setiap blok sehingga ditemukan 6 blok dengan total 117 baris kode dari total 10.239 baris kode yang mendapatkan *density* sebesar 0,8%. Duplikasi kode yang ditemukan memungkinkan risiko keamanan pada aplikasi sehingga perlu untuk peninjauan lebih lanjut oleh *developer.*

## V. SIMPULAN DAN SARAN

Berdasarkan pada penelitian ini didapatkan kesimpulan jika *tool SonarQube* yang digunakan untuk menguji *code quality* pada *project To Do List* menunjukkan bahwa analisis *SonarQube* dengan metriks yang luas dapat menjadi acuan pada standarisasi pengkodean yang telah diatur sehingga dapat membantu *developer* untuk meningkatkan kualitas kode program. Penggunaan *SonarQube* dapat menguntungkan jika dikonfigurasi dengan benar. Dalam *project To Do List* didapatkan hasil analisis dengan berbagai matriks, yaitu temuan di dalam matriks *bug* dengan 2 *issues* kategori *Major* dan 2 *bug* kategori *Minor*, kemudian ditemukan 2 *issues* level *low* pada matriks *security hotspots*, dan pada matriks *code smell* ditemukan 31 *issues* dengan 4 *issues* kategori *critical*, 16 *issues* kategori *major*, 4 *issues* kategori *minor*, 7 *issues* merupakan *info*.

Penelitian ini dapat berguna untuk memahami bagaimana hasil penemuan *code quality* dari pengujian kode program dengan mempertimbangkan pelanggaran yang ditemukan untuk diperbaiki sesuai aturan standarisasi pengkodean yang relevan. Pada penelitian ini menggunakan *SonarQube server* versi 8.9.9 sehingga direkomendasikan untuk menggunakan *SonarQube server* versi LTS terbaru agar *server* dapat berjalan dengan stabil. Ke depannya, eksperimen pasti dilakukan kembali untuk memperbaiki pelanggaran yang terdeteksi pada kode program sehingga akan mengalami inspeksi kode secara berkelanjutan. Penelitian ini dapat diterapkan pada aplikasi lainnya untuk mendapatkan pelanggaran kode yang ditemukan guna meningkatkan kualitas kode program.

## DAFTAR PUSTAKA

[1] D. Marcilio, R. Bonifacio, E. Monteiro, E. Canedo, W. Luz, and G. Pinto, "Are static analysis violations really fixed? a closer look at realistic usage of sonarqube," in *IEEE International Conference on Program Comprehension*, May 2019, vol. 2019-May, pp. 209–219. doi: 10.1109/ICPC.2019.00040.

[2] D. Marcilio, C. A. Furia, R. Bonifácio, and G. Pinto, "SpongeBugs: Automatically generating fix suggestions in response to static code analysis warnings," *Journal of Systems and Software*, vol. 168, Oct. 2020, doi: 10.1016/j.jss.2020.110671.

[3] A. Habib and M. Pradel, "How many of all bugs do we find? A study of static bug detectors," in *ASE 2018 - Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, Sep. 2018, pp. 317–328. doi: 10.1145/3238147.3238213.

[4] K. Liu *et al.*, "Mining Fix Patterns for FindBugs Violations," *IEEE Transactions on Software Engineering*, vol. 47, no. 1, pp. 1–24, 2020, [Online]. Available: https://github.com/mojohaus/nbm-maven-plugin

[5] A. Mahandis Shama and D. W. Chandra, "Implementasi Static Application Security Implementasi Static Application Security Testing Menggunakan Jenkins CI/CD Berbasis Docker Container Pada PT. Emporia Digital Raya," *Jurnal Ilmiah Informatika (JIF)*, vol. 9, no. 2, pp. 1–5, 2021.

[6] G. B. Imbugwa, L. J. P. De Araújo, M. Khazeev, E. Enombe, H. Saliu, and M. Mazzara, "A case study comparing static analysis tools for evaluating SwiftUI projects," in *Journal of Physics: Conference Series*, Dec. 2021, vol. 2134, no. 1. doi: 10.1088/1742-6596/2134/1/012022.

[7] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, "How Developers Engage with Static Analysis Tools in Different Contexts," 2019.

[8] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, A. Zaidman, and H. C. Gall, "Context Is King: The Developer Perspective on the Usage of Static Analysis Tools," 2018. [Online]. Available: https://gsuite.google.com/products/forms/

[9] V. Lenarduzzi, N. Saarimäki, and D. Taibi, "Some SonarQube Issues have a Significant but SmallEffect on Faults and Changes. A large-scale empirical study," *Journal of Systems and Software*, pp. 1–28, Aug. 2019, doi: 10.1016/j.jss.2020.110750.

[10] M. I. Shiddiq, "IMPLEMENTASI WHITE BOX TESTING BERBASIS PATH PADA FORM LOGIN APLIKASI BERBASIS WEB," *Jurnal Siliwangi*, vol. 8, no. 1, pp. 1–6, 2022.

[11] "SonarQube 8.9." https://docs.sonarqube.org/latest/ (accessed Nov. 24, 2023).

[12] "SonarSource – Self-managed | SonarQube | Sonar.", Accessed: Nov. 24, 2022. [Online]. Available: https://www.sonarsource.com/products/sonarqube/

[13] "Laravel – The PHP Framework for Web Artisans.", Accessed: Mar. 21, 2023. [Online]. Available: https:// https://laravel.com/

[14] "Vue.js – The Progressive JavaScript Framework.", Accessed: Nov. 24, 2022.

[Online]. Available: https://vuejs.org/guide/introduction.html/

[15] "PostgreSQL.", Accessed: Nov. 21, 2022. [Online]. Available: https://www.postgresql.org/about/

[16] S. Kawuma, E. Nabaasa, D. B. Sabiiti, and V. Mabirizi, "An Empirical Study of Bugs in Eclipse Stable Internal Interfaces," *Indian Journal of Software Engineering and Project Management (IJSEPM) 1* , pp. 1–7, 2021, [Online]. Available: www.ijsepm.latticescipub.com

[17] K. Filus, P. Boryszko, J. Domańska, M. Siavvas, and E. Gelenbe, "Efficient feature selection for static analysis vulnerability prediction," *Sensors (Switzerland)*, vol. 21, no. 4, pp. 1–25, Feb. 2021, doi: 10.3390/s21041133.

[18] I. R. Onyenweaku, M. S. Brown, M. Pelosi, and M. H. Shahine, "A SonarQube Static Analysis of the Spectral Workbench," *International Journal of Natural Science and Reviews*, vol. 6, no. 16, pp. 1–15, 2021.

[19] B. Barta, G. Manz, I. Siket, and R. Ferenc, "Challenges of SonarQube Plug-In Maintenance," in *SANER '19 : proceedings of the 2019 IEEE 26th International Conference on Software Analysis, Evolution, and Reengineering*, 2019, pp. 574–578.

[20] M. J. Salamea and C. Farre, "Influence of Developer Factors on Code Quality: A Data Study," in *Proceedings - Companion of the 19th IEEE International Conference on Software Quality, Reliability and Security, QRS-C 2019*, Jul. 2019, pp. 120–125. doi: 10.1109/QRS-C.2019.00035.

[21] I. Tollin, F. A. Fontana, M. Zanoni, and R. Roveda, "Change prediction through coding rules violations," in *ACM International Conference Proceeding Series*, Jun. 2017, vol. Part F128635, pp. 61–64. doi: 10.1145/3084226.3084282.

[22] V. Lenarduzzi, F. Lomio, H. Huttunen, and D. Taibi, "Are SonarQube Rules Inducing Bugs?," in *SANER '20 : proceedings of the 2020 IEEE 27th International Conference on Software Analysis, Evolution, and Reengineering*, 2021, pp. 501–511.

[23] F. Lomio, S. Moreschini, and V. Lenarduzzi, "A machine and deep learning analysis among SonarQube rules, product, and process metrics for fault prediction," *Empir Softw Eng*, vol. 27, no. 7, Dec. 2022, doi: 10.1007/s10664-022-10164-z.

[24] M. Tufano *et al.*, "When and Why Your Code Starts to Smell Bad (and Whether the Smells Go Away)," *IEEE Transactions on Software Engineering*, 2018, [Online]. Available: https://projects.apache.org/indexes/quick.html

[25] F. Palomba, G. Bavota, M. Di Penta, F. Fasano, R. Oliveto, and A. De Lucia, "A large-scale empirical study on the lifecycle of code smell co-occurrences," *Inf Softw Technol*, vol. 99, pp. 1–10, Jul. 2018, doi: 10.1016/j.infsof.2018.02.004.

[26] M. T. Baldassarre, V. Lenarduzzi, S. Romano, and N. Saarimäki, "On the diffuseness of technical debt items and accuracy of remediation time when using SonarQube," *Inf Softw Technol*, vol. 128, Dec. 2020, doi: 10.1016/j.infsof.2020.106377.

[27] N. Saarimaki, M. T. Baldassarre, V. Lenarduzzi, and S. Romano, “On the Accuracy of SonarQube Technical Debt Remediation Time,” in *Proceedings - 45th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2019*, Aug. 2019, pp. 317–324. doi: 10.1109/SEAA.2019.00055.