

AUDITING THE COVERAGE OF SOFTWARE QUALITY & SECURITY & COMPLIANCE REQUIREMENTS THROUGHOUT THE SOFTWARE DEVELOPMENT LIFE CYCLE: A HOLISTIC APPROACH

*Zhelyana Doneva Georgieva, Rositsa Doneva, Silvia Gaftandzhieva **

University of Plovdiv Paisii Hilendarski, Plovdiv
Bulgaria

* Corresponding Author, e-mail: sissiy88@uni-plovdiv.bg

Abstract: In recent years, due to the widespread adoption of Agile and DevSecOps approaches to software development, the increasing number of abuses and malicious actions targeting information technology-based resources and the accelerating need for software applications to adhere compliance requirements with certain standards, policies and regulations (e.g., the European Union Cyber Resilience Act or industry standards HIPAA and GDPR), the guarantee of all three – quality, the level of security and compliance for the needs of developing and delivering software solutions has become of significant importance. The paper presents a holistic approach to automated monitoring of software quality, security, and compliance, which has been researched and developed for the needs of one of the most growing industries, namely the FinTech industry. It introduces SDLC Auditor, a software tool designed to automatically monitor, control, and evaluate processes throughout the entire Software Development Life Cycle (SDLC) within Agile environments with regard to adherence to software quality, security, and compliance requirements. The study concludes by highlighting the advantages of the SDLC Auditor in developing stable, secure, and reliable software systems.

Key words: Software Development Life Cycle, agile approach, DevSecOps methodology, software quality, security and compliance, automated auditing.

1. INTRODUCTION

Various studies show that, with regard to software products, in today's constantly changing digital world, maintenance costs vary significantly and range between 40% and 90% of the total costs incurred over their lifecycle [1], depending on their complexity and operating environment. An unwritten rule here is that software maintenance costs can only increase and never decrease. Therefore, continuous monitoring and *quality management of the program code* [2, 3] throughout all stages of the software application development, delivery, and deployment processes is crucial

for ensuring their longevity, stability, security (i.e., vulnerability level), and effectiveness. In the context of the growing adoption of Agile [4] and DevOps [5] approaches to software development, such monitoring facilitates the maintenance of software products by reducing the effort required for their subsequent correction, protection, adaptation, or improvement.

In recent years, due to the growing number of abuses and malicious actions targeting computer systems and information technology-based resources, the guarantee of *software security* has become of significant importance in software development. This, in turn, added a new emphasis to the activities of continuous monitoring of the program code, namely, monitoring potential security issues and identifying vulnerabilities from unauthorized attacks at every stage of the development life cycle. [6]. Consequently, there arose a need to integrate security practices into the well-known DevOps software development methodology and to extend it to the so-called DevSecOps methodology [7, 8].

Last but not least, the software applications and development processes are increasingly subject to requirements for *compliance with specific regulations*, including both statutory and subordinate legislation, as well as internal organizational policies. In many cases, even mandatory adherence to certain policies and provisions (e.g., the European Union Cyber Resilience Act) or certification and licensing according to relevant standards maintained by standardization organizations such as IEEE and ISO [2] is required. Examples of standards in this field include: the ISO/IEC 25000 series on software quality attributes, the ISO/IEC 27000 series on information security management systems, ISO/IEC/IEEE 24774 on software lifecycle management, as well as the industry standards HIPAA and GDPR [9]. Compliance with various regulatory requirements regarding software, i.e., software compliance, is of critical importance to the software industry. For this reason, and due to their connection to the security of software products, compliance practices are also part of the DevSecOps methodology.

On the one hand, when it is necessary to implement a given DevSecOps practice, the use of a “manual” approach has a number of disadvantages. On the other hand, existing DevSecOps automation solutions (see 2) usually focus on isolated dimensions such as vulnerability scanning, CI/CD orchestration, management reporting or maturity assessment. Their analysis is not of interest here, as the approach proposed in the article is fundamentally different and integrates all these dimensions into a single framework for dynamic software quality, security and compliance (SQSC) auditing.

The paper presents a study aimed to investigate an issue that is of crucial importance for one of the most rapidly developing industries – financial technology (FinTech). Namely, automation of activities for monitoring the quality of program code and the level of security and compliance for the needs of developing and delivering software solutions, in line with the Agile approach to software development. A holistic approach is applied, in which the three main components (security, quality, and compliance with various regulatory requirements) that ensure a high-value software product are considered together throughout all stages of the Software Development Life Cycle (SDLC). The study further aims to demonstrate how heterogeneous SDLC governance controls can be unified into a single operational

framework for dynamic monitoring and auditing. A unified software tool, ***SDLC Auditor***, is proposed for the automatic monitoring, control, and evaluation of processes throughout the entire software code lifecycle with regard to adherence to SQSC requirements. To this end, the specifics of Agile development, delivery, and implementation processes for software products in the FinTech industry have been analyzed based on a specific corporate FinTech ecosystem. The development of the ***SDLC Auditor*** is based on a specially created conceptual model for automatic dynamic monitoring and assessment of SQSC control coverage. In conclusion, the advantages of ***SDLC Auditor*** for the development of stable, secure, and reliable software systems are summarized.

2. ANALYSIS AND DESIGN

The specific **corporate FinTech ecosystem** for software application development is typical of any “software-based company,” with the specific goal of providing software products and services in the financial sector based on various software solutions and characteristic modern innovations such as cloud computing, Artificial Intelligence (AI) and Agile project development and management structures. It ensures the development and maintenance of the FinTech software applications needed by the company, both in the form of multi-module monolithic platforms and based on microservice architecture.

From the comprehensive ecosystem of interconnected development environments, technologies, platforms, cloud services, API platforms, security systems, regulations, developers, and more, only those components that are essential to the purpose of this expose will be briefly presented and analyzed here.

In general, the ecosystem supports a development process that follows the Agile DevSecOps methodology, ensuring the regular delivery of working software through its development in small increments – iterations or sprints – by cross-functional teams. Each sprint encompasses the processes of five iterative phases: planning, development, testing, code integration, and deployment.

The “*Continuous Integration/Continuous Deployment*” (CI/CD) approach is followed, so that all the steps performed by development teams in a given software development cycle are integrated into a CI/CD pipeline. This ensures the automation of the entire SDLC [10, 11]. This enables the implementation of core DevSecOps practices such as Source Code Management (SCM), Git-based version control [12], Continuous and Canary Deployments [13]. The implementation of these fundamental concepts from the DevSecOps culture relies on appropriate modern technologies, tools, and services from the corporate FinTech ecosystem under consideration, automating the SDLC (see Table 1), among which the GitLab platform plays a key role in building CI/CD pipelines and automating the processes of development, testing, merging, and continuous delivery/deployment of code.

Table 1. Main technologies/tools providing CI/CD pipeline automation

Providing technology/tool	URL
GitLab intelligent orchestration platform for <i>DevSecOps</i> [14]	https://about.gitlab.com/
Jira project management tool	https://www.atlassian.com/software/jira/guides/getting-started/introduction#what-is-jira-software
Bitbucket collaboration tool	https://bitbucket.org/product/guides/getting-started/overview#key-terms-to-know
Jenkins open-source automation server	https://www.jenkins.io/
Bamboo CI/CD server	https://www.atlassian.com/software/bamboo
Bitrise Mobile DevOps Platform	https://docs.bitrise.io/
ServiceNow AI Platform	https://www.servicenow.com/platform.html
Kubernetes open source system for automating the deployment, scaling, and management of containerized applications	https://kubernetes.io/

When creating CI/CD pipelines, the guiding principle is “Shift Left” – moving all necessary automated checks/controls/gates for software quality, security, and compliance as early as possible within the SDLC. Automated verification of whether the code meets the specified quality criteria (e.g., Code Quality Analysis; Code Inspection; Unit Test coverage; API tests; E2E, i.e., End-to-End tests) is performed by incorporating so-called Quality Gates into the CI/CD pipeline. In addition, practices such as Static Analysis, Policy-as-Code, Infrastructure as Code (IaC), and Dependency Checking are followed, with controls related to compliance and security requirements being implemented in the CI/CD pipelines [15] (e.g., SAST – Static Application Security Testing, DAST – Dynamic Application Security Testing, and SCA – Software Composition Analysis, i.e. Dependency Scanning). All of this enables the identification, prevention and correction, immediately and as early and cost-effectively as possible, of quality gaps, potential errors, incidents, vulnerabilities or misconfigurations, security issues, non-compliance with regulations and standards for information and system protection [16]. The implementation of these controls in CI/CD pipelines is achieved primarily through integration with **SonarQube** Server for quality-related ones or with the **Veracode** Platform and the **Terraform** IaC tool for compliance and security controls.

For planning, designing, developing, and delivering the relevant high-quality product or product improvements, *SDLC controls for quality, compliance, and security* regarding the five iterative phases of each sprint are clearly defined in the corporate ecosystem policies. All mandatory controls are marked out, for example for: Architecture Review, Code Review (e.g., based on Merge Requests in the Source Code Management Tool), the CI/CD process (incl. the application of Quality & Security Gates), Testing (including post-deployment tests such as Performance Testing, Resiliency Testing and Security Tests), Defect Management, Release Management, Transition and Operational Support (including Segregation of Duties – SoD), and Decommissioning. Of particular interest are the general Input Criteria and Output Criteria for each iteration – namely, the Definition of Readiness (DoR) and the

Definition of Done (DoD). DoR ensures that teams have sufficient information to deliver a product that meets all requirements, while DoD mandatory qualifies an item for release when all conditions or acceptance criteria are met, where a critical requirement is the absence of high or critical security vulnerabilities. SDLC controls are provided by the various technologies and tools of the Fintech ecosystem. For example, **SonarQube** provides Quality Gates, Code Coverage, and Technical Debt; **GitLab** manages Merge Requests, Code Review, and SAST/SCA; and **Jira** manages Defect Management and requirements traceability.

In addition to the SDLC controls discussed, the strategies of the ecosystem regarded also define a set of appropriate *Key Performance Indicators (KPIs)* along with their threshold values (e.g., Code Test Coverage – Backend > 80% or Maintainability/Technical Debt of new code < 2%) and Critical Security Metrics (based on the principle of Zero Tolerance for Vulnerabilities), observation with which is a necessary condition for achieving DoD.

The culture of collaboration and communication among cross-functional teams is fostered by the appropriate environments included in the ecosystem, such as the **Confluence** team workspace.

The detailed analysis of the corporate FinTech ecosystem for software development demonstrates a clear need for comprehensive automation of CI/CD verification processes, ensuring that software products comply with all corporate and regulatory standards before reaching the end user.

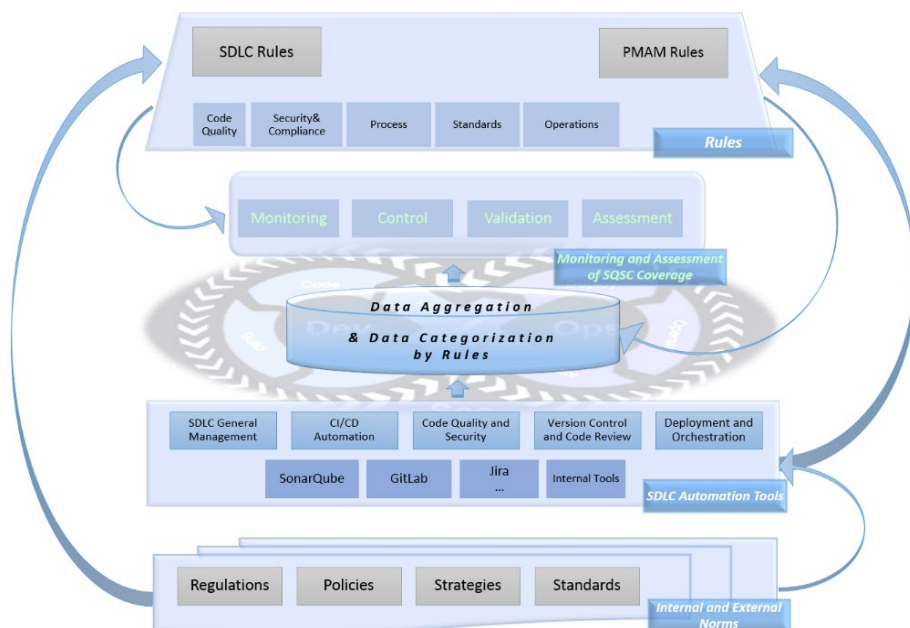


Figure 1. SQSC Coverage Audit Conceptual Model

Based on this analysis, a **conceptual model for automatic dynamic monitoring and assessment of SQSC control coverage** has been developed. According to the model (see Figure 1), the control and assessment process is based on DevSecOps-

related SDLC *internal and external norms* based on applicable regulations, policies, and strategies, as well as on the appropriately selected *SDLC automation tools*, which are generally third-party cloud services (see above). Together, the applicable norms and the selected tools define the so-called *Rules* that govern this process. *Rules* are conditionally divided into two groups, based on their purpose in the dynamic monitoring process—SDLC Rules and Platform Maturity Assessment Model (PMAM) Rules. These rules define which controls, key indicators, and critical metrics need to be tracked for the purposes of controlling and validating the SQSC.

SDLC Rules is an open set of rules that defines mandatory requirements for SDLC quality, security, and compliance governance. These rules establish the controls (along with the thresholds for the relevant KPIs) for software applications developed across the various SDLC phases, covering five SDLC categories: Code Quality, Security & Compliance, Process, Standards, and Operations. SDLC Rules currently includes 20 rules that ensure every software application approved for deployment in each iteration meets the highest internal and external standards, regulations, policies, and strategies. The goal is to prevent and address all possible errors and threats (e.g., build breakers).

PMAM Rules define the broader operational maturity of FinTech software product development – maturity across product areas and lines of business. Based on the 12-Factor App methodology [17], 13 rules have been defined for this group.

The data used to verify the controls, indicators, and metrics according to these defined rules is generated by various external/third-party (e.g., the aforementioned SonarQube, GitLab, Jira) and internal (some custom-built tools) SDLC automation tools. Therefore, it is first necessary to aggregate all this data and then categorize it according to the rules from the 5 categories of SDLC Rules and PMAM Rules – *Data Aggregation & Data Categorization by Rules*.

This allows, for each specific software application, after appropriate data processing according to the rules applicable to it, to obtain final verification results and to calculate a final assessment of quality and reliability monitoring – *Monitoring and Assessment of SQSC Coverage*.

3. SDLC AUDITOR

Following the model presented in Figure 1, the *SDLC Auditor* software tool was designed, implemented, and tested for automatic dynamic monitoring and assessment of SQSC control coverage.

The main functional requirements met by the *SDLC Auditor* are:

- It allows for seamless integration into Agile and DevSecOps application development workflows, regardless of the programming language or their architecture – monolithic or microservices;
- It automatically monitors, ensures, and validates adherence to the specified SDLC Rules and PMAM Rules (see 2) that apply to the relevant software application throughout the entire SDLC, as:
 - automates the adherence to quality code development practices, i.e., monitors the presence and adherence to quality gates, profiles, and reviews,

and allows or blocks the progression of a given software application further in the CI/CD pipeline to compilation, merging, etc.;

- automates the enforcement of software security and compliance practices, i.e., monitors the presence and adherence to relevant key scans, to the Threat Model for applications in production, etc.;
- automates the continuous delivery/deployment process by allowing or blocking the inclusion of a given application in the CD pipeline, depending on whether or not all controls and requirements defined by Rules have been met;
- determines eligibility for Canary Deployment, for both manual and automatic Canary Deployment, by validating the application's readiness for such advanced practices;

- Calculates the so-called SDLC Score and PMAM Score for a specific software application, based on the validation of adherence to the applicable SDLC Rules and PMAM Rules;

- Maintains a complete set of historical data on the status of adherence to the Rules for all software applications;

- Provides useful information on: the current state of the SDLC and the quality, security, and compliance of the applications under development; deviations in adherence to Rules (e.g., incorrect configurations and missing integrations with third-party services) that could hinder the monitoring of the SQSC; previous statuses or trends in SQSC control coverage, etc.;

- Offers a feature-rich, responsive, and interactive interface with extensive tools for visualizing data and results, providing access and control at various levels, across different parameters and dimensions, through control panels, chart types, pattern detection, anomaly highlighting, real-time dashboards, interactive storytelling, etc.;

- Enables the generation of reports and the ability to export data in real time or print to paper;

- Provides immediate feedback on code quality, vulnerabilities, and compliance.

For efficient data management and application performance, ***SDLC Auditor*** follows a **two-tier architecture** (see Figure 2) – Presentation Tier (the front-end) and Application & Data Tier (the back-end).

The front-end of the solution is implemented on the TypeScript-based Angular framework, served via the NGINX web server and communicating with the back-end via REST web services and the JSON data interchange format, while the back-end comprises a Spring Boot-based REST API with embedded business logic, external API clients for integrating with external SDLC automation tools, and a PostgreSQL database, enhanced by the Liquibase migration approach.

The tool itself complies with major application security practices and standards, utilizing an authentication and authorization model with role-based access control and an OAuth2 authentication flow with Azure Active Directory.

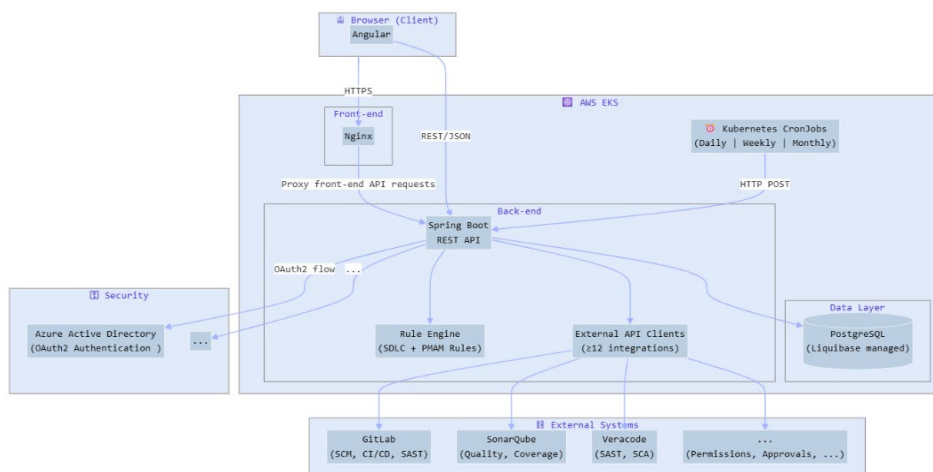


Figure 2. SDLC Auditor architecture

SDLC Auditor is deployed on one of the most popular cloud service platforms, Amazon Web Services (AWS). It is containerized and deployed on AWS Elastic Kubernetes Service (EKS) with Terraform-managed infrastructure. Automated **SDLC Auditor** checks are managed by Kubernetes CronJobs. Thus the adherence to cloud-native best practices, guarantees SDLC Auditor's scalability, availability, and security.

The back-end's **Rule Engine** is the core component of **SDLC Auditor** that implements its business logic. It monitors, enforces, and validates the rules applicable to a given software application across the two rule categories (SDLC and PMAM). The Rule Engine uses a pluggable architecture, as the set of rules is dynamically configurable (e.g. customization of KPIs thresholds) and rules can be added or removed. Furthermore, in specific cases, rules can be enabled or disabled, the thresholds of the relevant KPIs can be modified, and specific SDLC automation tools can be excluded from certain rules.

Mandatory checks monitored for all applications (most popular programming languages are supported) include SonarQube Quality Gates, Profiles, and Code Coverage; Build Breakers and Merge Breakers (based on Bamboo, GitLab, Jenkins); E2E tests and Auto Deploy Approval (via ServiceNow); SAST and SCA/Dependency Scan (via GitLab and Veracode); DAST and Fuzz Testing (via GitLab), and more.

The main idea behind the Rule Engine implementation is to integrate so-called Monitoring Gates into GitLab's CI/CD pipelines for individual software applications, which monitor adherence to the quality, compliance, and security controls defined by the Rules. It helps automate the calculation of the SDLC Score and PMAM Score on the fly (during pipeline execution). The scores are values in the range [0, 100] based on the percentage of applicable rules met relative to the total number of rules in the category.

In general, the Rule Engine processes SQSC data collected from internal and external SDLC automation tools in the following steps (see Figure 3):

- Filter Rules: Check each rule's applicability;

- **Evaluate Rules:** Evaluate applicable rules, e.g. the data provided by SonarQube for Technical Debt percentage to one of the SDLC Rules, after comparison with the already mentioned appropriate KPI ($< 2\%$) allows producing of evaluation data (met/not met) for that rule;
- **Collect Results:** Store evaluation data (met/not met) for each rule;
- **Calculate Score:** Count the number of met rules, divide by the number of applicable rules, and multiply by 100;
- **Store Data:** Save scores and rule results to the database.

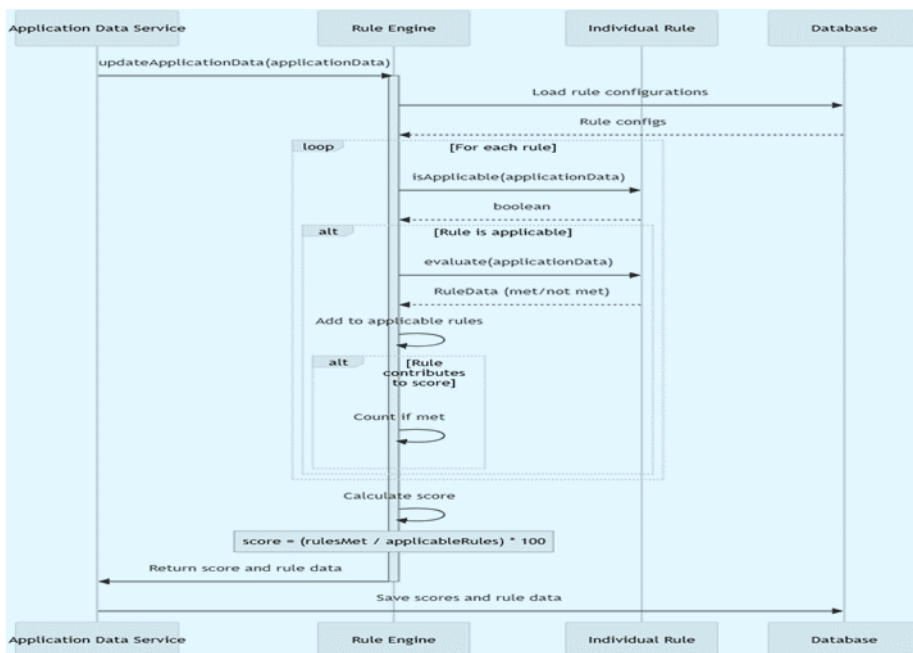


Figure 3. Rule Engine Evaluation Flow

The Rule Engine also performs some important post-processing actions, such as blocking deployments unless all quality, security, and compliance controls defined by the Rules are met (SDLC Score $< 100\%$), or generating trend data for historical analysis and dashboards, as well as sending email notifications about applications with SQSC violations.

The process for monitoring and controlling SQSC coverage, managed by the Rule Engine, is triggered on a scheduled basis (daily via a Kubernetes CronJob), manually (triggered by an administrator via the API), or automatically (from GitLab CI/CD pipelines). The results of this process (referred to as Application Health for short) are visualized by the **SDLC Auditor front-end** in customizable, feature-rich User interface (UI) components (using the PrimeNG Angular UI Component Library. Access and control are provided for different levels (admin and non-admin users) and purposes (Administration, Help, Application Health, Reporting & Analysis, etc.) and based on various parameters (SDLC Health, PMAM Health, SCM & CI/CD Tools,

Dependencies, Deployments, Rollbacks). Figure 4 presents the Overall SDLC Health Dashboard of the *SDLC Auditor*.

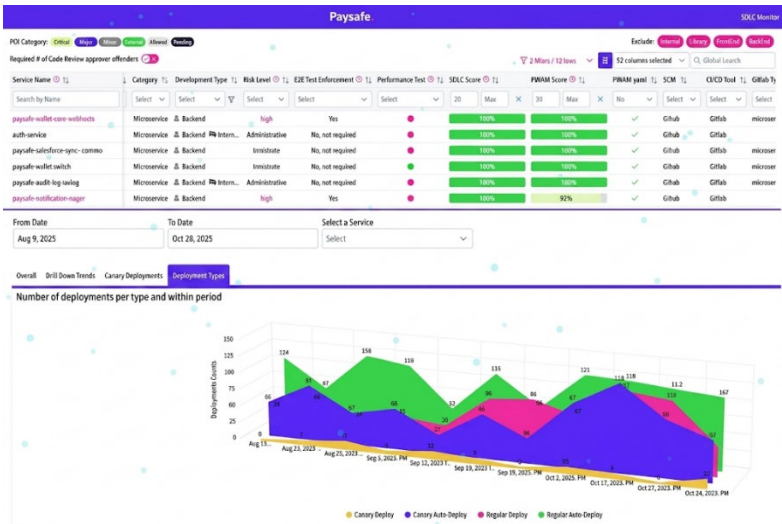


Figure 4. SDLC Auditor: Overall SDLC Health Dashboard

4. CONCLUSION

The results of this study are of significant importance for managing the quality, security, and compliance of software products. They not only enable the implementation of one of the key DevSecOps principles – automation – but also further advance its paradigm. They offer a holistic approach to the process of automated SQSC assurance by automating its monitoring, control, and evaluation in a data-informed manner where the data is collected dynamically from SDLC automation tools. The conducted practical validation of the *SDLC Auditor* tool within a large-scale FinTech ecosystem on over 2500 applications shows high appreciation from all teams, due to acceleration, immediate visibility and traceability of all operations.

The *SDLC Auditor* tool, which objectifies SQSC assurance activities and enhances their effectiveness, offers a number of advantages, such as:

- Transforms SQSC requirements, scattered across numerous regulations, policies, and strategies, into a unified, insurmountable barrier against low-quality or insecure code;
- Ensures the delivery of secure, high-quality software by Agile and DevSecOps teams that meets the highest corporate and regulatory standards;
- Provides the necessary level of applications maturity in the continuous deployment process, which is a testimonial that they satisfy the DoD and also to their logical consistency, reusability, stability, and security;

- Improves communication, strengthens transparency, and supports data-driven decision-making to address problem areas or to implement improvements regarding the SQSC;

- Reflects the current status of the applications under development in real time and maintains mechanisms for timely warnings or alerts regarding non-compliance;

- Serves as a single source of truth for audits of SDLC and SQSC;

- Last but not least, it promotes practices within teams that encourage accountability and adherence to the best standards for DevSecOps software development.

And while the **SDLC Auditor** tool has one natural limitation – because it was developed for a specific corporate ecosystem – this does not apply to the conceptual model for automatic dynamic monitoring and assessment of SQSC control coverage (see Figure 1), which would find application in any modern software application development ecosystem. In the future, one could explore the development of the model in a dynamic, AI-based environment and the integration of proactive functionality into a corresponding AI-powered tool.

ACKNOWLEDGEMENT

The paper is financed by the Scientific Research Fund at the University of Plovdiv Paisii Hilendarski, project № MUPD25-FMI-007.

REFERENCES

- [1] Ventionteams. *Software Maintenance Costs - 2024 Benchmark Overview*. URL: <https://ventionteams.com/enterprise/software-maintenance-costs> (Visited on 29 April 2026).
- [2] Gobov, D., Zuieva, O. Software Quality Attributes in Requirements Engineering, *International Journal of Information Technology and Computer Science*, vol. 17, no. 4, 2025, 38–48. DOI:10.5815/ijitcs.2025.04.04
- [3] Basheleishvili, I., Tsirumua, S., Bardavelidze, A. Algorithmization and realization of the software tool for the software code quality assessment. *International Journal on Information Technologies & Security*, vol. 14, no. 2, 2022, 27-38.
- [4] Doneva, Z., Gaftandzhieva, S. Agile Methodology in Software Development: Code Quality and Security Compliance Benefits and Challenges. In: Yang, X.S., Sherratt, S., Dey, N., Joshi, A. (eds) *Proceedings of Ninth International Congress on Information and Communication Technology. ICICT 2024*. *Lecture Notes in Networks and Systems*, vol 1004. 541–553, 2024, Springer, Singapore. https://doi.org/10.1007/978-981-97-3305-7_43.
- [5] Narang, P., Mittal, P. Performance analysis of DevOps based hybrid models integrated with different automation tool chains for quality software development. *International Journal on Information Technologies and Security*, vol. 14, no. 4, 2022, 25-32.
- [6] Janisar, A. A., Meidan, A., Kalid, K. S. b., Gilal, A. R., Sarlan, A. B. Security Requirements Engineering: A Review and Analysis. *Computers*, vol. 14, no. 10, 2025, 429. DOI: 0.3390/computers14100429

- [7] Haber, M.J. Secured DevOps (SecDevOps). In: Privileged Attack Vectors. 2nd ed.; Publisher: Apress, Berkeley, CA, 2020, pp. 251-255. DOI: 10.1007/978-1-4842-5914-6_19
- [8] Bernardino, N. A., Sequeira, B., Piza, E., Henriques, F., Neves, F., Reis, C. I. Enhancing DevSecOps: Three Custom Tools for Continuous Security. In *2024 IEEE 11th International Conference on Cyber Security and Cloud Computing*, 2024, 53–58. DOI: 10.1109/CSCloud-EdgeCom62360.2024.10605157
- [9] Sandu, A. K. DevSecOps: Integrating Security into the DevOps Lifecycle for Enhanced Resilience. *Technology & Management Review*, vol. 6, 2021, 1–19.
- [10] Stahl, D., Martensson, T., Bosch, J. Continuous practices and DevOps: beyond the buzz, what does it all mean?. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications*, 2017, 440–448. DOI: 10.1109/SEAA.2017.8114695
- [11] Rajapakse, R. N., Zahedi, M., Babar, M. A., Shen, H. Challenges and solutions when adopting DevSecOps: A systematic review. *Information and Software Technology*, vol. 141, 2022, 106700. DOI: 10.1016/j.infsof.2021.106700
- [12] Ponuthorai, P. K., Loeliger, J. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*, 3rd ed., Publisher: O'Reilly Media, 2022 (546 p.)
- [13] Gonzalez D., Varghese D. Canary deployment, in the book *Implementing Modern DevOps*, 2017, Packt Publishing (326 p.)
- [14] Jones, A. *Advanced GitLab CI/CD Pipelines: An In-Depth Guide for Continuous Integration and Deployment*, Publisher: Walzone Press, 2024 (302 p.)
- [15] Tatineni, S., Allam, K. DevOps Security: Integrating Security into the DevOps Workflow. *EPH-International Journal of Science and Engineering*, vol. 10, no. 1, 2024, 13–21.
- [16] Islavath, N. Shifting Security Left: Integrating DevOps with Secure Development Practices. *Journal of Artificial Intelligence, Machine Learning and Data Science*, vol. 1, no. 1, 2020, 1368–1371.
- [17] Hofmann, M., Schnabel, E., Stanley, K. *Microservices Best Practices for Java*, Publisher: IBM Redbooks, 2017 (134 p.)

Information about the authors:

Zhelyana Doneva Georgieva – PhD student at Paisii Hilendarski University of Plovdiv. Until now she is an author of four scientific publications (incl. two indexed in Web of Science, SCOPUS) with total 5 noticed citations.

Rositsa Doneva, PhD – Professor at Paisii Hilendarski University of Plovdiv. She is an author of over 220 publications (46 indexed in Web of Science and 50 indexed in Scopus) with over 1000 citations.

Silvia Gaftandzhieva, PhD – Associate Professor at Paisii Hilendarski University of Plovdiv. She is an author of over 160 publications (70 indexed in Web of Science and 80 indexed in Scopus) with over 900 citations.

Manuscript received on 05 May 2026