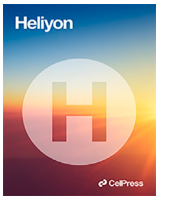
Heliyon 10 (2024) e26969

Contents lists available at ScienceDirect

Heliyon

journal homepage: www.cell.com/heliyon



Research article

LPMSAEF: Lightweight process mining-based software architecture evaluation framework for security and performance analysis

Mahdi Sahlabadi [a,*], Ravie Chandren Muniyandi [c], Zarina Shukur [c], Md Rezanur Islam [b], Morteza SaberiKamarposhti [c], Kangbin Yim [a]

[a] Department of Information Security Engineering, Soonchunhyang University, Chungnam, Asan-si, 31538, South Korea
[b] Software Convergence, Soonchunhyang University, Chungnam, Asan-si, 31538, South Korea
[c] Center For Cyber Security, Universiti Kebangsaan Malaysia, Bangi, Selangor, 43600, Malaysia

ARTICLE INFO

Keywords:
Lightweight early and late evaluation
Software architecture
Process mining
Petri nets complex and heterogeneous architecture

ABSTRACT

The article discusses the need for a lightweight software architecture evaluation framework that can address practitioners' concerns. Specifically, the proposed framework uses process mining and Petri nets to analyze security and performance in software development's early and late stages. Moreover, the framework has been implemented in six case studies, and the results show that it is a feasible and effective solution that can detect security and performance issues in complex and heterogeneous architecture with less time and effort. Furthermore, the article provides a detailed explanation of the framework's features, factors, and evaluation criteria. Additionally, this article discusses the challenges associated with traditional software architecture documentation methods using Unified Modeling Language diagrams and the limitations of code alone for creating comprehensive Software Architecture models. Various methods have been developed to extract implicit Software Architecture from code artifacts, but they tend to produce code-oriented diagrams instead of Software Architecture diagrams. Therefore, to bridge the model-code gap, the article proposes a framework that considers existing Software Architecture in the source code as architectural components and focuses on Software Architecture behaviors for analyzing performance and security. The proposed framework also suggests comparing Software Architecture extracted by different Process Mining algorithms to achieve consensus on architecture descriptions, using visualizations to understand differences and similarities. Finally, the article suggests that analyzing the previous version of a system's Software Architecture can lead to improvements and deviations from planned Software Architecture can be detected using traceability approaches to aid software architects in detecting inconsistencies.

* Corresponding author.
E-mail address: sahlabadi@ieee.org (M. Sahlabadi).

https://doi.org/10.1016/j.heliyon.2024.e26969
Received 22 June 2023; Received in revised form 17 February 2024; Accepted 22 February 2024

Available online 28 February 2024

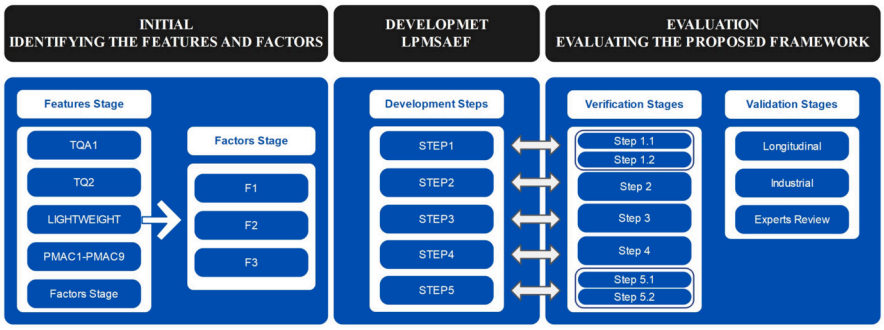


Fig. 1. The Methodology of LPMSAEF.

1. Introduction

Software systems are one of the most intricate systems created by humans today [33] and it is difficult to analyze Software Architectures (SA) because they have components that are developed by various programming languages and established on various networks [15], distinctive hardware, and operating systems. In spite of this heterogeneity [18], it is essential to organize (software architecting) the software's components properly [37]. This organization of components is an error-prone process since the processes of communication of software components are complex. To overcome this issue, different big up-front designs [3] are sketched based on software analysts' experiences in order to explore the design problems, particularly during the early stages of software design. SAs, which are poorly designed, lead to the major failure of a software project. Accordingly, it is vital to evaluate SAs for the early detection of architectural problems prior to software development.

Research communities have proposed miscellaneous SA evaluation methods to uncover SA problems systematically. Generally, SA evaluations are advantageous and inexpensive to detect the risks or problems in the early stage [27] to solve problems, while they are costly in the late or even maintenance phases. Scenario-based Software Architecture Analyses, like the Architecture-based Tradeoff Analysis Method (ATAM), are able to analyze SAs [30]. However, they are inadequate to cope with this complex technical topic [26] since they rely on limited scenarios. As a result, other techniques, such as simulation and mathematics, were used to promote the methods. Architectural evaluation methods are still not being used widely by practitioners [4]. To address these limitations, researchers have employed various methodologies, such as Design Science Research Methodology (DSRM), which is particularly suitable for developing new methods and frameworks to enhance the effectiveness of architectural evaluation methods, as demonstrated in this study.

This research adapts DSRM to conduct the study since DSRM is a suited method to build artifacts, such as prototype systems, and develop new methods and frameworks [20]. The research strategy included three phases: initial, development, and evaluation. The initial phase presents an overview of the framework using lightweight SA evaluations to improve its functionality. Secondly, the development phase involves creating the framework based on identified features and factors. Finally, the evaluation phase consists of verification and validation stages, which include longitudinal and industrial case studies and expert reviews to increase the validity of the proposed framework (LPMSAEF). The framework was verified and validated through case studies and expert review.

2. Methodology

In this study, DSRM is utilized to construct a new framework. Typically, DSRM comprises two distinct components, namely the building phase and the evaluation phase. The research methodology consists of three distinct phases. The initial phase introduced the overview and profile of the framework based on the features and factors encouraging the use of lightweight SA evaluations in the sense of definitions, procedures, and tools to improve the functionalities of the aimed framework in the industry [30,31]. The development phase intends to develop the lightweight SA evaluation framework based on the identified features and factors. Hence, the SA evaluation framework was implemented based on the identified features and factors. The evaluation phase consists of two stages: verification and validation. Subsequently, in the verification stage of the framework, three longitudinal cases were studied. In these three case studies, the applications' source code and the development teams were accessible, which are the main case studies that provided the evidence and data to support the development of the framework. In the validation stage of the framework, the results of longitudinal case studies, besides three industrial case studies, were used. Moreover, the experts' review was conducted to increase the validity of LPMSAEF. The proposing phase is verified by obtaining new information from the implementation of the framework in case studies, which may aid the next proposing cycle. The framework was verified and subsequently validated by the case studies and the expert's review. The Methodology is depicted in Fig. 1.

2.1. Initial phase

The profile for the software evaluation framework is designed based on our proposed software evaluation comparison framework, the criteria for comparing lightweight evaluation methods [30], and the tools and techniques [31], which are depicted in Fig. 2.

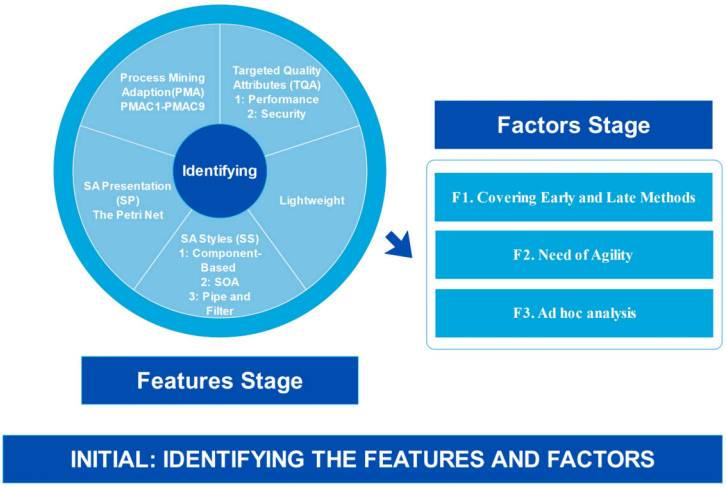


Fig. 2. Initial phase [30].

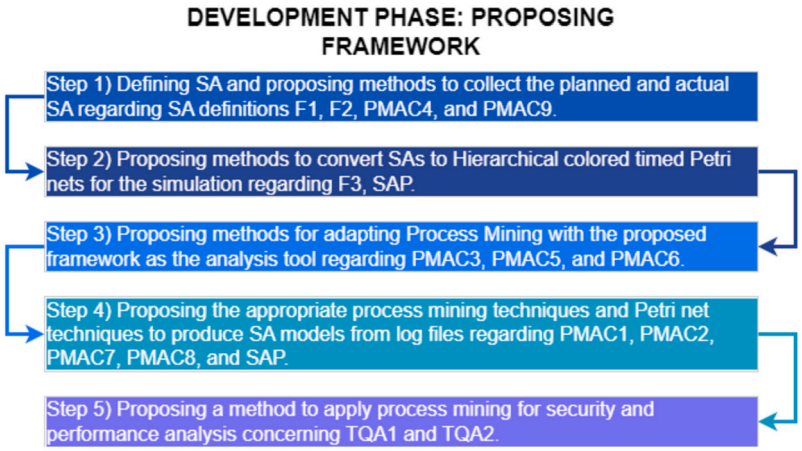


Fig. 3. Development stage.

2.2. Development phase

The SA evaluation framework was proposed based on the identified features and factors. The ultimate goal of the framework is to evaluate SA's performance and sustainability. To achieve this goal, firstly, based on the profile of the framework, the following steps have been taken:

- The input and the skeleton of the framework were identified. Consequently, SA was defined, and methods were proposed to collect the planned (the main requirements, related Unified Modeling Language (UML) diagrams) and actual SA (the implemented system).
- Methods were proposed to present SAs in hierarchical colored timed Petri Net format. This conversion had two benefits. First, the simulation was conducted and produced data for Process Mining (PM), and second, the comparison of SAs models in Petri Net so differences between the planned and implemented SA could be detected.
- Lastly, the PM-based methods were proposed to evaluate the security and performance of SA.

The Development phase is depicted in Fig. 3.

2.3. Evaluation phase

The aims of this phase are to verify and validate the framework and whether it can be an effective solution to the defined problem. In fact, the framework was constructed in line with the design science paradigm. The framework was first built and then evaluated by case studies. It is necessary to verify that the framework is correctly implemented by running case studies. Then, LPMSAEF must be validated with respect to the features and factors. The proposed SA evaluation framework is verified by using three longitudinal case

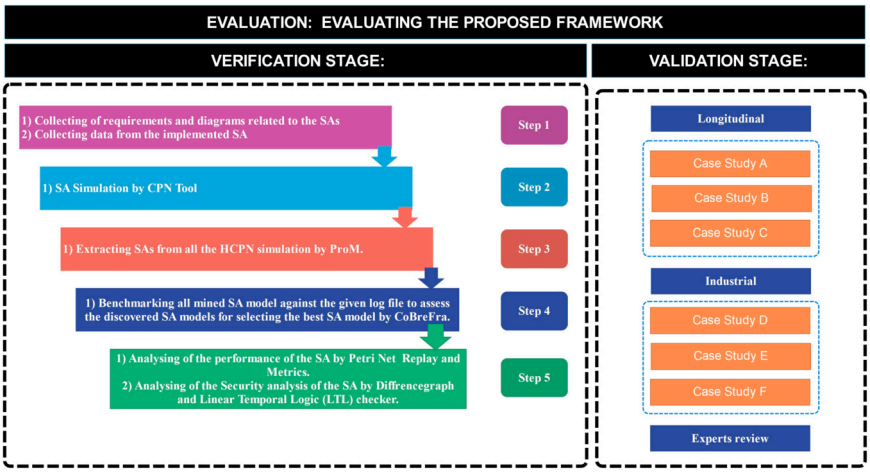


Fig. 4. Verified stage.

studies (A, B, and C). In the evaluation part, the results of the verification are presented in detail. Therefore, these results improve LPMSAEF. The data and evidence from the longitudinal case studies and three further industrial case studies (E, F, and G) were used to validate LPMSAEF. Moreover, the experts reviewed the framework and agreed with its functionalities. The evaluation phase is generally depicted in Fig. 4.

Case Study A The case has been developed to provide an online shopping system and has been studied for six months. The data are collected from the development team and architects of projects while conducting the case study.

Title Online Shopping.
SA Style SOA.
Description The Online Shopping System provides services in order to conduct shopping. The solution works based on the SOA, which contains multiple services such as inventory, catalog, delivery order, customer account, credit card authorization, and email services. SOA is a rampant pattern of the distributed SA [24].
Reason of selection The online shopping website is selected as the case study because it has been extensively used in many related researches in the realm of SA design and review by Petri Net. Moreover, it is an example of a highly distributed Web–based system providing services.

Case Study B The software mining academic group located in Singapore has developed the case for research purposes. The case has been studied for six months. The data are collected from the development team and architects of projects during the conduct of the case study.

Title The personalized bug prediction.
SA Style Pipe and Filter.
Description The system supports personalized bug prediction in open-source software by analyzing code patterns' changes referring to bugs and bug fixes made by developers over a period of time. The system aims to identify possible locations that may contain bugs in source code. The structure of the application can be seen as an Event-driven architectural style, which is strongly recommended for highly distributed applications as well as for applications engaging with the transmission of the events among software components.
Reason of selection This study was selected since Mining Software Repositories (MSR) is a growing area of Software Engineering [36] and the application consists of 12 java files containing 5134 Lines Of Code (LOC) of Fava, 9 bash files, and 1 awk (1000LOC). The application also uses Weka, Deckard, and Javancss. It also interacts with Github, Bugzilla, and Jira. It is heterogeneous software with a Data-Flow Architecture style with the pattern of Pipe and Filter and is a well-known distributed SA.

Case Study C The cybersecurity group of Universiti Kebangsaan Malaysia (UKM), located in Malaysia, has developed the case for research purposes. The case has been studied for two months. The data are collected from the developers and the architect of projects during the conduction of the case study.

Title Air conditioner controlling through a remote location.
SA Style Component-Based.

Description Description: The system is called an "Air conditioner controlling through a remote location." This system checks if the aircon system is ON and there is nobody in the room; then, it will trigger an alarm on the user's mobile phone. This distributed embedded system is composed of many different software/hardware components, human interfaces, actuators, and sensors [28].

Reason of selection This case study was selected since it is a heterogeneous project containing a microcontroller (Arduino) that should be granted by C++, a Global System for Mobiles (GSM) module, sensors, and a mobile application. It is difficult to design and evaluate such architecture.

2.3.1. Methods of implementation

The simulation of the SA was executed using the CPN tool, which is a Petri Net tool, whereas the ProM tool, which is a process mining tool, was utilized for security and performance analysis. To examine the degree of conformance between the actual SA and the planned SA, a comprehensive analysis of discrepancies between the two models is deemed essential. The Differencegraph plugin [12,11], which is accessible in ProM and capable of supporting heuristic or Petri Net formats, is employed to perform a comparative analysis of the two models and to isolate discrepancies that are then subjected to further analysis through visualization.

2.4. Validation stage

This stage aims to validate the applicability, repeatability, efficiency, and other features and factors of LPMSAEF. The validation has been done by longitudinal case study results, applying the evaluation framework to three industrial case studies. Finally, the experts review the framework entirely by an expert survey. Single-case studies are mostly criticized for the case's uniqueness. Three case studies seem sufficient to reach the degree of certainty to support the research's claim. Case studies' artifacts (documentation, prototype, log files, etc.) of a software development project can be investigated to indicate what really happens in software projects [22]. As a consequence, in this research, six case studies with their artifacts are used to boost results. For business reasons, the industrial cases never shared source code and SA models, but they applied LPMSAEF and collected the results. These case studies are in the domains of online exchange, company workflow, and mobile phone applications. In the following paragraphs, these case studies are introduced with the reason for their selection. These case studies were applied by the companies and the results were collected. These case studies aimed to test the framework in the industry.

Case Study D The case has been monitored after applying the SA evaluation framework by the development team and architect. The online payment project is by Rizpardakht Company, located in Dubai. The case has been studied for two months. The data are collected from the development team and architect of the project while conducting the case study.

Title Online Exchange Company.
SA Style N/A.
Description Python and Mongo DB are used to develop the project.
Reason of selection It is a heterogeneous and distributed system and the company accepts to apply the framework.

Case Study E The case has been monitored after applying the framework by the development team and architect. The project of Workflow of Social Security Organization Research Institute is located in the Middle East. The case has been studied for two months. The data are collected from the development team and architect of the project while conducting the case study.

Title Workflow of Social Security Organization Research Institute.
SA Style N/A.
Description SharePoint is used to develop the project.
Reason of selection It is a heterogeneous and distributed system, and the institute accepts the application of the framework.

Case Study F The case has been monitored after applying the framework by the development team and architect. The mobile phone application project was done by an IT company located in Malaysia. The case has been studied for two months. The data are collected from the development team and architect of the project during the conduction of the case study.

Title Golf Mobile phone application
SA Style N/A.
Description Android programming technology is used to develop the project.
Reason of selection It is a heterogeneous and distributed system, and the company accepts to apply the framework.

2.4.1. Industrial and longitudinal case studies' results analysis

The validation of the framework means that LPMSAEF works well and delivers reliable results better than the methods in the hands of the practitioners. This comparison is based on lightweightness criteria that totally mean less time, effort, and formalism.

Thus, the functionality of LPMSAEF, which is the detection of flaws in the security and performance of SA, is compared with the current method of SA evaluation, which is used by the practitioners in the case studies.

2.4.2. Expert review

In addition, the acceptance of the proposed SA evaluation framework needs the cooperation of professionals interested in SAs evaluations. These experts have been selected as they have a general overview and background of this research. Table 9 shows the expert evaluation form for the framework. The expert evaluation survey was conducted in two sessions, and the questionnaire was handed out and replied to during sessions. Each session consists of the framework presentation with a demo and a replying questionnaire consisting of closed and open-ended questions. First, deal with the motivation, aim, and objectives of this research as well as the research outline, the scope, and the significance of the study. Firstly, the research carries out the methodology that was adapted based on the objectives and scopes outlined. The procedures, tools, and case studies are also introduced. Secondly, the research implements the framework that applies the performance and security analyses of SA, which are also discussed from the PM technique's views. Thirdly, the research illustrates the results obtained from the expert survey, three industrial case studies, and three main longitudinal case studies. In addition, LPMSAEF is verified and validated. Finally, the research summarizes the contributions of this research and the upcoming outlines for future works. The study conducts the research in three phases, and each phase is designed to achieve the objectives. So, the overall conduction of this research is based on these objectives in line with DSRM. The processes of validation and verification of the "software architecture evaluation/analysis/review methods" are fundamentally based on case studies [30]. Those solutions, which have been applied to six case studies or more, can be considered reliable solutions, so this research uses six case studies. In this study, the case studies are selected based on the domain and the features that suit LPMSAEF. They are heterogeneous and distributed, and their source code is available for the longitudinal case studies. The study upholds the validity of the result by an expert review and three industrial case studies. The results of the implementation of the evaluation framework on the industrial case studies are monitored and analyzed.

3. Initial phase

The existing SA evaluation methods were fully reviewed to identify LPMSAEF's main features in the first stage. In this stage, the SA evaluation framework's features were identified based on the text analysis of researchers' and practitioners' SA definitions and all published studies for the last three decades on SA's topics. This analysis concluded a lightweight SA evaluation solution was needed to uncover problems in distributed and heterogeneous software's security and performance. Consequently, the security and performance analysis of SA were reviewed and the proper SA presentation and styles for the distributed and heterogeneous software were identified [30]. In the second stage, the factors of lightweights are identified to improve the SA evaluation framework used in the industry. In that, lightweights were identified from the weaknesses of the current state of the art in the lightweight SA analysis methods. Indeed, the study tried to bridge the gap of less usage of systematic SA evaluations in the industry. First, it should be clear why the industry refrains from SA evaluation methods proposed by academics. As a result, this study followed two strands of academics and practitioner's concerns. The practitioners have considered the customers that their needs as the SA evaluation framework's features, and academics should also be considered. This mindset led to the text analyzing the online web repository of SA definitions. The main features were lightweight, heterogeneity, performance, and security. Moreover, the study conducted a systematic literature review on SA evaluation methods. As a result, the SA evaluation comparison framework was proposed as a basis for the SA evolutions comparison. Then, it narrowed down the literature to the lightweight SA evaluation methods. A total of six SA evaluation methods were studied deeply to identify the factors influencing the SA evaluation method [30].

4. Development

This phase depends on existing kernel theories and the way they are applied to solve the problems at hand. Throughout the framework development phase, the research engages in the process of iteration, refining the framework's design and implementation based on feedback from the stakeholders.

4.1. LPMSAEF

In this section, LPMSAEF is fully discussed. First of all, the existing methods are compared using a criteria table, which is fully described in our previous papers [31,30]. In that research, the comparison between ATAM as the standard heavyweight method and five fashionable lightweight methods is performed, and three main factors for lightweights are considered, including performance and security issues. Also, in our other research [10], PM tools are employed to overcome the challenges of recognition and adaption of tools and techniques for the lightweight software architecture evaluation framework to address the Targeted Quality Attributes (security and performance) and the lightweightness factors of the Hypothetical Software Architecture Evaluation Framework (HSAEF) [31]. In addition, previous research, as discussed above, is briefly explained in the next two sections.

4.1.1. Overview and profile of LPMSAEF

The profile for the software evaluation framework is designed based on the proposed software evaluation comparison framework and the criteria for comparing lightweight evaluation methods. The framework's profile is presented in Table 1 with respect to the above-mentioned explanation and identified features, factors, and tools. This is to remind you that the comparison framework

Table 1
Profile of LPMSAEF.

Component & Criteria	Element	LPMSAEF's Profile	Features and Lightweightness Factors
Context, C1 and C3	**Objectives**	To detect flaws and issues in the performance and security, the conformance of planned and implemented SA.	Targeted Quality
	Inputs	Informal description of requirements, UML diagrams for the planned architecture and source code for the implemented architecture	Lightweightness
	Outputs	Risks, issues, and thoroughness of the evaluated SA.	N/A
	Scope	A set of specific architecture models (SOA, Pipe and Filter, Component-based architecture) presented in Petri Net.	Presentation And Style of SA, Scope of Evaluation
Time C1, C3 and C5	**Schedule**	Early and late.	Covering Early and Late Methods
Contents, C2 and C5	**Evaluation Approaches**	Scenario-based, Simulation-based, PM, Experience-based.	N/A
	Tools for Automation	Prom, CPN Tools.	Tools And Techniques.
	Priority Setting	During the review.	Need of Agility
	Scope	A set of specific architecture models (SOA, Pipe and Filter, Component-based architecture) presented in Petri Net.	Presentation And Style of SA, Scope of Evaluation
Stakeholder, C4 and C6	**Knowledge of Evaluators**	General knowledge about SA.	Ad-Hoc Analysis
	Reviewers	Company-Internal or External Reviewers.	Ad Hoc Analysis.
	Social interaction	A face-to-face meeting between reviewers and the architect.	Lightweightness Reliability
C7	**Methods of validation**	6 case studies and experts review	N/A

contains the components and related elements of SA evaluation methods. This comparison framework provides the taxonomic responses to the criteria (C1 to C7) for comparing lightweight methods [30]. As it is discussed in the [30], the framework compares SA evaluation methods with some essential criteria. These criteria are listed below from C1 to C7:

C1 The main goal of the method.
C2 The evaluation technique(s).
C3 Covered Quality Attribute QAs.
C4 Stakeholder's Engagement.
C5 How applied techniques are arranged and performed to achieve the method goal.
C6 How user experience interferes with the method.
C7 Method Validation.

The objective of the framework is to check security and performance requirements in order to determine the satisfaction level of its non-functional requirements. In addition, a system can be evaluated to check if it obeys the functional requirements. The detected issues in the evaluation of SA are the output of this framework. The framework's inputs are scheduled in the early and late stages of software development. In the early stage, either UML diagrams describing SA or requirements stated in natural language are collected as a planned SA, and in the late stage, source code is collected as an actual SA. The scope of the framework focuses on distributed software. The evaluation approaches used in LPMSAEF are scenario, simulation, PM, and experience-based. UML and Natural Language Requirement (NLR) are the scenarios that are supposed to be converted into Petri Net for the simulation. Then, the results will be evaluated by PM techniques. The need for Ad Hoc Analysis shows that the knowledge of evaluators is important for the lightweight evaluation framework. The experiences of the architect and reviewers are used in the Priority setting of the decision stage in the framework. Prom and CPN Tools are the tools for automation for the analysis and simulation. The knowledge of evaluators, the reviewers, and their social interactions follow the minimal intersections of the lightweight methods [31,30]. In the methods of validation, as it was explained, the validation is credible enough when at least three case studies are investigated to support the research's claim. Moreover, the robust, lightweight methods [31,30] have been applied to three to six case studies. Additionally, the experts review the framework entirely to uphold the validity of the research.

4.1.2. Components of LPMSAEF

Upon reviewing the proposed SA evaluation framework, the present research endeavors to incorporate its distinctive features and factors towards the development of an effective and streamlined SA evaluation process. In order to evaluate the performance of the SA model, it must first be converted into a performance model, after which timing information can be integrated into the model.

Additionally, in accordance with the procedures in software evaluation methods, it is essential to obtain both the intended and actual architecture for conducting late SA evaluations. This enables a thorough comparison between the original plan and the implemented architecture, facilitating a comprehensive assessment of its quality and areas for potential improvement.

The current study seeks to acquire the components and their interrelationships of the proposed evaluation framework through an analysis of the structure of late SA evaluation methods and SA-based performance analyses. These components, along with their respective features and factors, are encapsulated in a component as depicted in Fig. 5, which also illustrates the relationships between them.

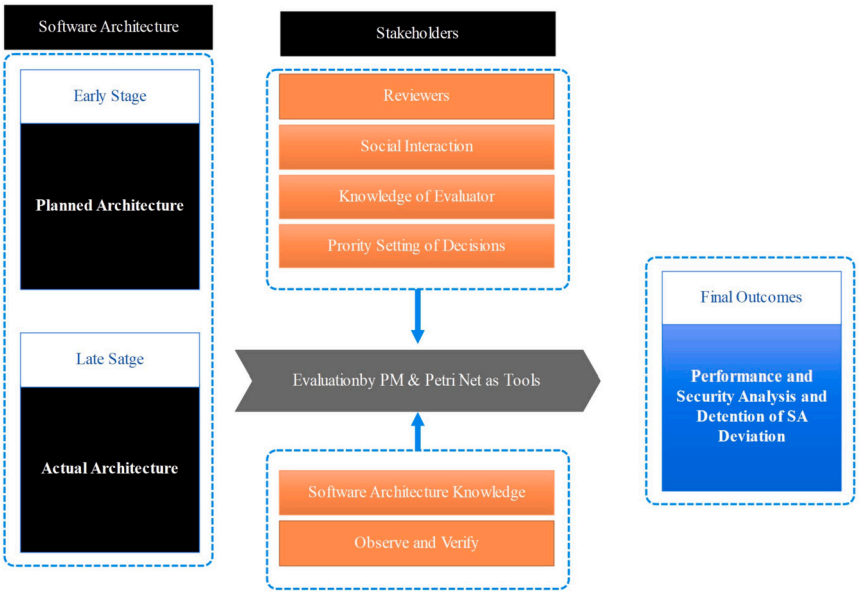


Fig. 5. Components and their relationships.

Table 2
Views of the proposed approach.

SA's Level	Views	Framework's view
Conceptual Architecture	Application domain, abstract software paradigms, design methods design using domain-specific components and connectors, performance estimation, safety and reliability analysis, understanding the static and dynamic configurability of the system	Normally, these architectures are stated in UMLs or natural languages. The conceptual architectures are converted to a Petri Net.
Module Architecture	Enabling software technologies, organization structure, design principles management and control of module interfaces, change impact analysis, consistency checking of interface constraints, configuration management.	This is less abstract rather than conceptual architecture. The Petrified conceptual SA to HCPN with regards to Module architecture.
Execution architecture	Hardware architecture, run-time environment performance criteria, communication mechanisms performance and schedulability analysis, static and dynamic configuration of the system, porting systems to different execution environments.	These architectures produce Timed-HCPN by adding some timing information to the HCPN.
Code architecture	Programming language, development tools and environment, external subsystems,	Software activities are recorded in order to have Log files for Prom. PM models these activities into Petri Net by means of Prom. The log file should be in MXML format.

4.1.3. Mechanism

PM measures the deviation of an actual SA from its planned architecture. The planned architecture is determined and mined by converting UML diagrams into Petri Net and simulating them to produce data for PM. The actual architecture is obtained by recovering the SA from the source code using various tools and methodologies. Deviations between the planned and actual architectures are identified, and recommendations for changes are provided to the development team for improvement. The process is repeated to confirm that the actual architecture conforms to the planned architecture, as indicated in Fig. 5.

4.2. LPMSAEF to analyze SA

The previous sections provide an overview of LPMSAEF, including its profile, components, and mechanism. This section elaborates on the framework's elements, specifically the three components of the SA evaluation framework: software architecture, collecting the planned architecture, and recovering the actual architecture. Each component is explained in its corresponding subsection. The section also introduces PM and Petri Net, which are used to evaluate the performance and security of the SA by comparing the planned and actual SAs. LPMSAEF is based on different levels of SAs, which are explained in Table 2 and depicted in Fig. 6 represents an expansion of Fig. 5, illustrating intricate specifics. The framework's lightweight definition and abstraction make it easy to distinguish and understand. The section concludes by outlining the three functionalities of LPMSAEF, which are further explained in the subsequent sections.

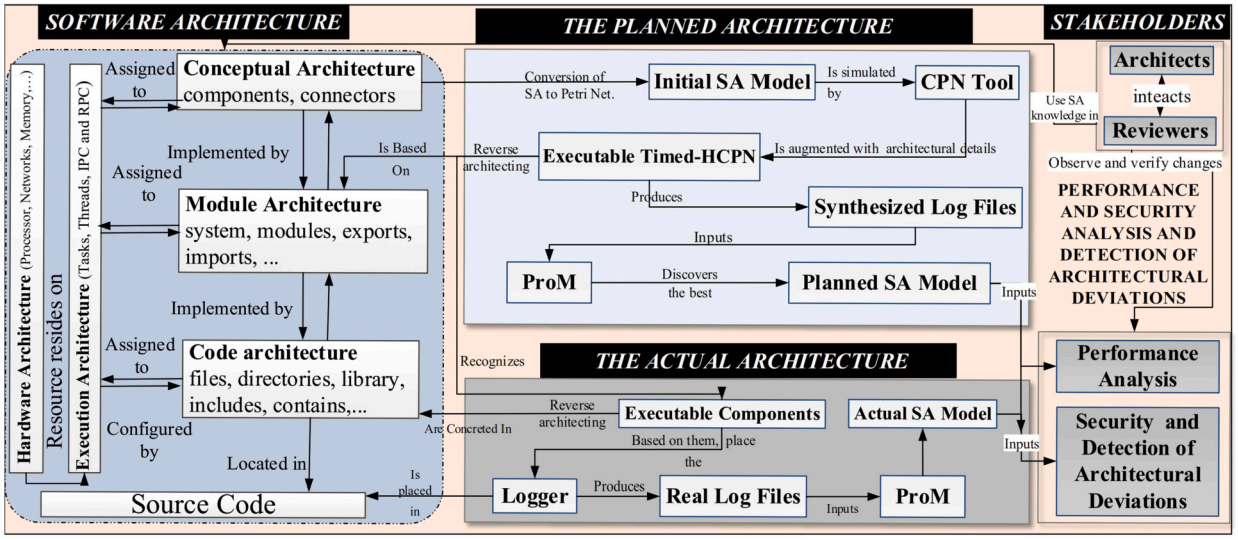


Fig. 6. The lightweight SA evaluation framework using PM and Petri Net.

4.2.1. Elements of LPMSAEF

LPMSAEF comprises four key components, depicted as black rectangles in Fig. 5. Each component can encompass four essential elements. Data, SA model, application techniques, procedures, and algorithms, as shown in Fig. 6. The research details each component's elements within their respective section.

The data predominantly comprises log files that are utilized by the PM tool. These log files should be in the MXML format to be processed by the Prom tool. LPMSAEF includes two types of log files: synthesized log files generated through simulation using CPN 4.0 and real log files extracted from source code by loggers based on software activities. Models are primarily in Petri Nets, and LPMSAEF produces two primary models: the architectural model, which encompasses conceptual, module, code, execution, and hardware architecture, and the initial SA model representing the SA in a Petri Net. Additionally, the framework generates an executable timed HCPN model, which enriches the initial simulated SA by incorporating properties to generate reliable log files, as well as planned and actual/implemented SA models obtained from synthesized and real log files, respectively.

Application and techniques encompass the PM tool, Prom, which generates SA models in Petri Net format, with the techniques and algorithms being applied through Prom plug-ins. Additionally, the CPN tool is utilized for simulation in Petri Net format, while loggers are utilized to record SA activities.

Procedures and algorithms play a significant role across multiple facets of LPMSAEF, including the conversion of SA to Petri Net, reverse architecting, simulation, production and analysis of log files, security analysis, performance analysis, detection of SA deviation, and stakeholder interaction.

4.3. Stakeholders

The framework for the analysis SA utilizes the stakeholder component, which is comprised of evaluators, priority setting, reviewers, and social interaction. This research highlights the importance of defining SA and its abstraction levels. Hofmeister's framework [14] for SA of distributed systems, which delves SA at four levels of granularity, is presented in detail. The SA definition is based on the three elements of data, connecting elements, and processing elements, which help architects to abstract SA. The research also further delves into the different types of components, connectors, data, and configurations that makeup SA. Finally, the research notes that there are various views on SA and provides a table explaining the different architectural views. Consider this kind of component belongs to SA and should not be taken as the framework's components. LPMSAEF covers four levels of SA abstraction (see Table 2), notably, conceptual architecture is regarded as the input to LPMSAEF, a result of the lightweight SA analysis method's minimalistic approach [30].

4.4. The planned architecture

The component discussed in the paper aims to produce a high-quality planned SA, accomplished through three steps: converting SA into a Petri Net, adapting PM to produce SA models, and assessing SA models to identify the optimal one. The primary emphasis) of the paper lies on the initial step, which involves abstracting SA into three layers (conceptual, module, and code architecture) and combining them into a single HCPN representing SA. UML diagrams are commonly used to describe SA and non-functional requirements, and an algorithm is used to transform UML diagrams into executable models based on various Petri Net extensions. LPMSAEF aims to help architects design SA and understand the true SA through recommended algorithms. Importantly, the paper does not cover the validity of the SA itself.

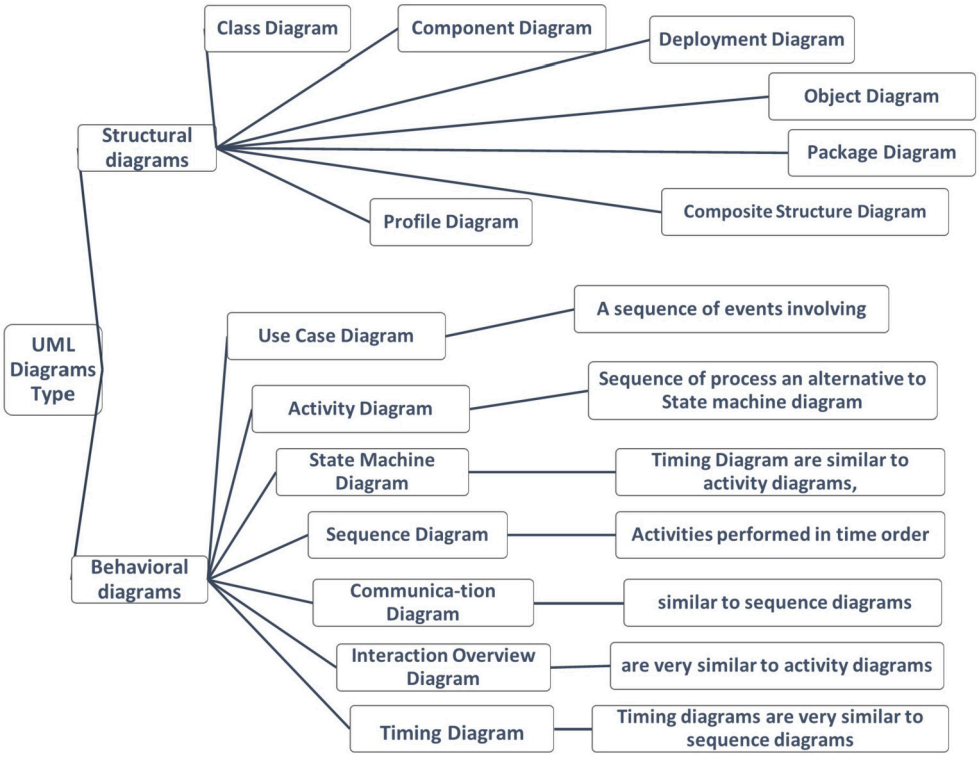


Fig. 7. UML diagrams' categories.

4.5. Identifying events in SAs

SA is defined differently. The related definition is provided to understand the conceptual architecture layer. UML defines SA as Architecture, which is the organizational structure and system's associated behavior.

Architecture can be decomposed into parts that interact recursively through interfaces, relationships that connect parts, and restraints assembling parts. Parts that interact through interfaces consist of components, classes, and subsystems". Referring to this definition, system behavior and decomposability are two main concerns of SA from UML's point of view. UML has been divided into two categories: behavioral and structure diagrams. Fig. 7 indicates 14 types of UML diagrams and their categorizations [35]. The prerequisite for applying PM into an area is an event log. An architect should be able to exploit existing diagrams in order to identify the events and activities and adapt them to PM.

Behavioral diagrams, including use case, sequence, and communication diagrams, and structural diagrams, with a focus on component diagrams, have been carefully selected. These chosen diagrams primarily depict the key components of the SA and associated activities, while other behavioral diagrams, like sequence diagrams, share similarities; the additional structural diagrams are more inclined towards representing software structure rather than SA. In Fig. 8, the UML diagrams are linked to PM. As a result, in LPMSAEF, UML diagrams are related to the PM definition, instantiation, and recording. Use cases identify tasks, communication diagrams, and sequence diagrams and provide a detailed elaboration of the behavior depicted in the use case diagram. Therefore, an architect should approach this diagram from the PM's perspective.

PM produces SAs from the log file, which is the list of the sequence of activities. These activities are associated with an event. Therefore, the events should be identified prior to starting with PM in a system.

4.6. Conversion of SA to Petri Net

To sum up, the intended SA can be transformed into Petri Net based on the suggested and proposed algorithm [28] and the proposed diagram (see Fig. 8), according to the architect's domain knowledge. An architect should collect the planned SA, which is stated in either UML diagrams or NL requirements. This should be done to identify events since it is a prerequisite for the collection of log files.

4.7. The actual architecture

This section of LPMSAEF focuses on identifying the components of a software SA to place loggers to log SA activities, with the ultimate goal of producing the actual SA. The framework employs forward engineering in each case study, where documents and

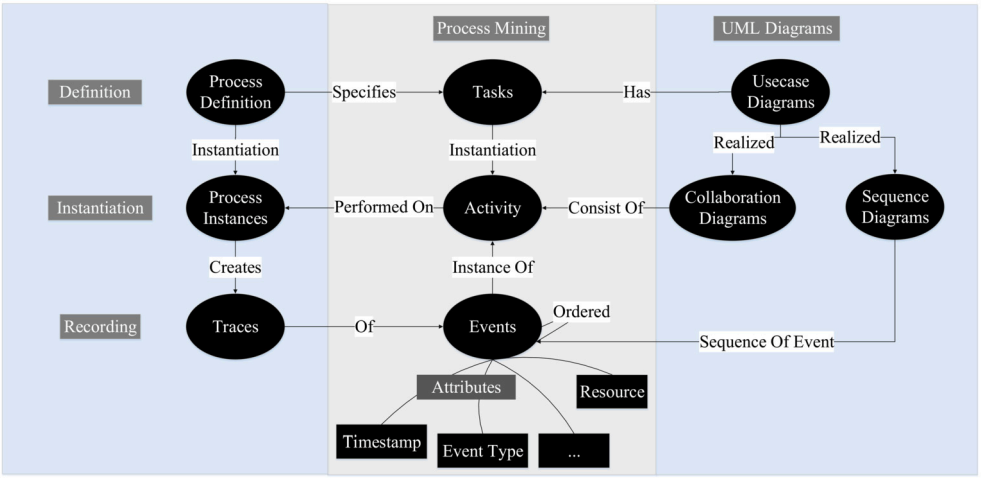


Fig. 8. Relationship between UML diagrams and Log files.

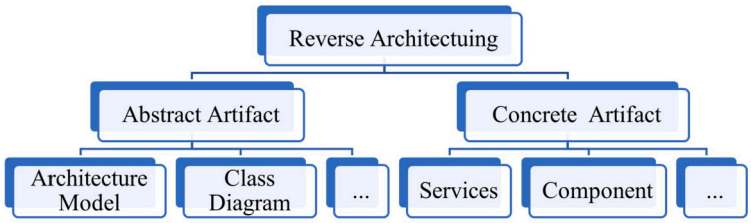


Fig. 9. Classification of reverse engineering results.

source codes go through reverse engineering paradigms. Reverse engineering is used to mine software artifacts from legacy source code, which can be high-level abstractions or concrete software artifacts. Behavioral UML diagrams are used to identify events, while structural grams are used to identify system components. Component diagrams were addressed in the previous subsection.

4.8. Reverse architecting

Reverse architecting is considered a particular kind of reverse engineering, which follows three significant steps of mining, abstraction, and presentation, which are depicted in Fig. 9. This research enters the case studies' artifacts into LPMSAEF through these three steps:

Mining In this initial step, information should be mined from available documentation and sources from the documented history of the system (Software Repositories). The identification processes' input can be source codes, documentation, historical information, or human expert knowledge:

DOCUMENTATION Software documentation is categorized into two broad categories of text or model-based documents, which represent software artifacts at various levels of abstraction. LPMSAEF relies on minimal SA documents.

HUMAN EXPERT KNOWLEDGE Mostly, software architects are considered human experts who have highly relevant knowledge about the design and implementation of software systems. There are a few methods that prioritize human expert views as the primary perspective on architecture [25] Human experts play a crucial role in identifying SA elements and their dependencies based on their background knowledge and experience derived from previous projects. They can contribute valuable architectural information, which is not evident from source code and requirements without considering the level of analysis.

SOURCE CODE The loggers are placed into source code to log components' activities.

First, a mixture of the above-mentioned input resources is applied with the purpose of recognizing SA according to the dynamic analysis requisite of at least the source code and another input that explains the execution scenarios, relying on use cases [8]. LPMSAEF effectively reduces code complexities through the use of human experts and documentation. Second, a combination is made by the investment of the human experts' knowledge to modify and analyze the resulting architecture. Third, the final architectural information can serve as the identification approaches' guide. The classification results of the identification approaches that are presented about software artifacts essential as inputs.

Abstraction In this step, obtained information from the previous step should be abstracted in line with the reverse architecting objective. Data should be reduced into a manageable quantity. The abstraction uses the three elements of the SA definition to abstract the SA.

Presentation This step transforms abstracted data into an understandable model for the user. Petri Net is powerful and has the advantage of modeling languages. In addition, Petri Net has been used to analyze SA. Petri Nets are a common tool for representing the result in PM. LPMSAEF uses SAs resented in Timed HCPN which is a version of Petri Net [31]. The simulated SA in basic Petri Net is converted to Timed HCPN supported by CPN Tools. The initial petrified SA models are augmented with architectural details. The new model identifies the executable components of the software based on the module architecture. The executable components are the concert components in the code architecture

4.9. Performance and security analysis and detection of architectural deviation

This is the analyzing part of the framework, which aims to analyze the performance and security of SA; besides, it detects the architectural deviations. In this framework, the performance of the planned SA and the security of the planned and actual SA are analyzed. To identify architectural deviations, it is necessary to compare the planned and actual SA. As a result, this component runs after the planned and actual architecture components, which need to adapt PM and select the best model of SA [31]. This section discusses performance and security analysis and the detection of architectural deviations. The Performances and security are QA discernible during the system execution. Therefore, by observing the SA behavior, the performance can be measured, and the security flaws can be detected. The quantitative analysis of an SA enables the architect to detect potential performance problems in the early stages of software development. Detecting potential software performance issues in the early stages of development can result in more cost-effective solution development.) In other words, designing a software system and analyzing its performance have occurred before the system's implementation.

4.10. Performance analysis, the metrics and simulation approach

This section explains how the proposed approach can analyze the performance of SA by PM. The primary idea of simulation-based performance analysis is leading a series of long simulations of a CPN model. During this modeling, the data is collected from the observed binding elements for calculating estimations of the system's performance measures. The PM techniques collect and transform the log files to process the model with Petri Net presentation. Petri Net can assist in evaluating the system performance and also validate the security. It has an enhanced capability to represent models that incorporate both functional and non-functional details. The idea of using PM in SA design is introduced for the first time in LPMSAEF. The performance analysis with the Petri Net plug-in assesses the performance of SA processes by replaying traces of the log in the SA model. The performance analysis plug-in obtains the Performance Indicators' value. It provides important performance indicators that are summoned. These key performance indicators are understandable intuitively. This plug-in needs the log file and the relevant SA model which is presented in Petri Net [41].

The method produces the performance information from the log file and SA model. The replay method executes the traces in the Petri Net. The events of traces must be associated with transitions in the given Petri Net. When an event occurs, the corresponding transmission (s) in the Petri Net is fired, and measurements are taken. The following performance metrics are:

Cases the total number of process instances.

Cases the total number of process instances.

Perfectly-Fitting Cases The number of process instances that complete properly and successfully, i.e. the number of instances that can be replayed in the Petri Net without any problems.

Throughput Time The throughput time of the process instances

Frequency The number of visits of tokens to the place during replay of the process instances in the Petri Net.

Waiting Time The time that passes from the (full) enabling of a transition until its firing, i.e. time that a token spends in the place waiting for a transition (to which the place is an input place) to fire and consume the token.

Synchronization Time The time that passes from the partial enabling of a transition (i.e. at least one input place marked) until full enabling (i.e. all input places are marked). The time that a token spends in a place, waiting for the transition (to which this place is the input place) to be fully enabled.

Sojourn Time The total time a token spends in a place during a visit (Waiting time + Synchronization time).

The performance analysis with the Petri Net plug-in assesses the performance of SA processes by replaying traces of the log in the SA model. The performance analysis plug-in obtains the Performance Indicators' value. It provides important Performance Indicators that are summoned. These key performance indicators are understandable intuitively. This plug-in needs the log file and the relevant SA model which is presented in Petri Net [20].

4.11. Security analysis

LPMSAEF introduces an innovative PM approach in order to detect abnormal behaviors of software components from the architectural aspect. The approach identifies security threats by SA running. This approach draws inspiration from the research presented

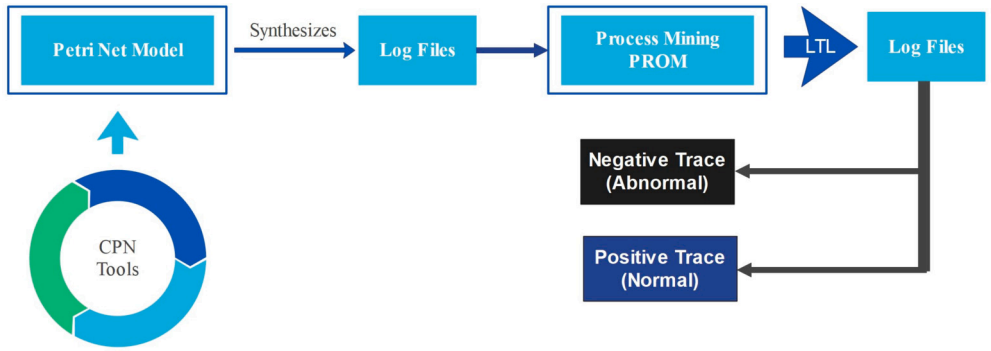


Fig. 10. Security analysis by LTL.

in reference [29]. It comprises two key steps: simulation using Colored Petri Nets (CPN) and thread discovery employing Process Mining (PM) techniques. One of our cases has undergone implementation using this method and has been subsequently published [32]].

As shown in Fig. 10, in the simulation step, the use case diagrams are simulated to synthesize log files in order to be inputted to the PM tool. These use cases represent the normal behavior of the system. In the discovery step, we attempted to identify possible threats from log files by PM techniques via the following techniques:

- The inconsistency is investigated by the PM tool in order to detect abnormal traces.
- Linear Temporal Logic (LTL), which is a kind of temporal logic, is used to find abnormal traces. System policy, security strategy, and rules are the constraints represented by LTL.

Deviation of the implemented SA from the planned SA is a recurrent problem that may cause security problems, so the deviation should be detected. Utilizing a Difference graph facilitates the task. Architectural Deviations and Code Analysis from Security's View Analyzing the consistency of the design code is one of the significant parts of the evaluation of late SA. Although effort has been put into this area, there is still no formal framework and taxonomy for analyzing design-code discrepancies and prioritizing the interventions to bring the design up-to-date are still lacking.

Moreover, design-level vulnerabilities are a chief resource of security risks in software [39]. The basic differentiation between early and late SA evaluation is that implementation of an SA is not necessary for the early SA evaluation, while late SA evaluation. Nevertheless, most mathematical model-based early SA evaluation approaches need architectural components' implementation-oriented data. This research introduces the late SA evaluation method which works based on simulation data and systems prototype. Security is a challenging quality attribute to attain, as small coding slips can lead to gaping security holes. During securing a system, developers generally follow checklists, searching for identified types of attack vectors and observing their code to make sure it is not vulnerable. It is possible to validate and automate such a method by modeling the system as a Data Flow Diagram (DFD) and developing a program analyzer that would excerpt a similar one from the source code. The DFD was analyzed, and the source code was ensured not to have any deviations that could cause problems. Design challenges are also provided due to the capability to apply more security measures. Although each additional measure comes at a cost, it is still hard for stakeholders to know the most optimal basket of measures for them [9]. Traditional security testing has deficiencies in providing flexible architecture–level tests with regard to the code. Security testing basically differs from traditional testing due to its emphasis on what an application should do rather than what it should not do. Therefore, positive requirements are distinguished. Unlike functional testing, security testing concentrates more on negative behaviors, such as undesirable behavior that influences the system's security or breaches the confidentiality of data.

In addition, it is not always promising to plan a requirement for a particular software artifact when the requirement is not implemented in a particular place. Functional security testing examines the mechanisms of security implemented in the system in order to confirm it behaves as it is expected. This is mostly based on security software requirements. Nevertheless, the functional security testing system does not guarantee that an attacker cannot penetrate the system. Ensuring the system's correct behavior does not guarantee the lack of undocumented behaviors. It just confirms that the system works according to the specification. In this testing, the requirements are related to the security properties of the system, such as integrity and confidentiality of data. Functional testing is derived from specifications automatically. In addition, the purpose of code-based testing is to uncover code-based weaknesses. A general problem related to functional testing is that it considers what the program should do. It means functional testing does not purposely examine what the application may still do further than the requirements or specifications. It is important, in security testing, to examine situations that are absent in the specifications. We believe that functional testing is considerably different from security testing, because the reason is security is not an externally observable property, unlike functionality, and in the future, its results or consequences are not easily predictable [2].

Hidden functionality in the software poses a significant problem. One reason is even after that, the software does not have malicious codes. This problem allows users to get unauthorized access to the system. Back doors are good examples of such hidden functionality typically employed as undocumented, extra software constructs that lead to bypass of security mechanisms. Therefore,

it is hard to identify such hidden functionality. There are some reasons for the difficulty of detecting hidden functionality. First, the SA of a system may not be clear. The used architecture patterns, styles, and other constructs are usually not visible in the code, and the original developers and architects are not often available in order to explain why there is a certain structure for the software or why certain functions are inserted into the code. Moreover, the documentation may not be up to date or available. In LPMSAEF the conformance of the planned SA with the implemented SA is considered as the realization flaws and placed under security problems.

4.12. Deviation of SA

Deviation of the implemented SA from the planned SA is the recurrent problem. This problem occurs due to application changes, some new requirements adaptation, developer replacement, or mistakes [34]. As it is discussed, late SA evaluation methods identify the differences between the planned SA and implemented SAs. Normally, these methods rely on the techniques and tools that reconstruct the actual architecture to compare it with the planned architecture. They are applied at the testing phase to check the actual SA's conformance with the planned design. Software's execution data can be used by the software dynamic analysis in order to understand the software's behavior. Some Tools and techniques extract the information during the software's run-time. However, these techniques are not able to discover the concurrency very well. For the complex models, they encounter the state explosion problem. Some techniques portray software execution's traces into sequence diagrams, but these sequence diagrams still lack concurrency descriptions. Besides, a sequence diagram merely depicts the behavior of a single execution trace. These approaches produce many behavioral models using software execution data containing multiple traces, while there is a need for a comprehensive and concise model reflecting all data. Moreover, as the nature of the software is hierarchical, the flat sequence diagrams or automation-based models do not indicate the actual behavior of the software accurately [19].

Several approaches and commercial tools have been proposed to uncover architectural violations. These tools are very dependent on the platform and programming style. On the other hand, they may detect hundreds of violations with no evidence and weak support [34,43]. A typical UML model is either a platform-specific model (PSM) or a platform-independent model (PIM). The PIM is an accurate model that depicts SA. PIM can be transformed into any specific platform. Therefore, PIM is an untestable model for all platforms. It is notably useful since the PIM version of SA can be mapped to various platforms, such as NET COM, J2EE, CORBA, Web Services, or other Web platforms. In LPMSAEF, the conceptual architecture is used, which is expressed as a UML platform–independent model [13]. In the PM realm, normally, conformance can be looked at from two perspectives: Firstly, models do not represent real behavior, and secondly, the reality actually deviates from the expected model. The first one is for the descriptive models to predict or capture the reality, while the second one is for normative models to control or influence the reality [1]. In LPMSAEF, the second perspective is highlighted to capture deviations of the implemented SA from the planned SA. In order to check the conformance of the SA, it is essential to analyze the differences between SA models. The difference graph is a plug-in of ProM that provides the differences and commonalities between process models that are presented in heuristic or Petri Net format. In LPMSAEF, the Difference graph compares the conformance of the actual and planned SA before being visualized. The result of the difference graph indicates the deviations between the models [12,11].

4.13. Significance of LPMSAEF

This research provides valuable contributions to knowledge, practice, and community. It presents a lightweight framework for identifying security and performance issues with software architectures. The framework uses performance modeling to detect conformance, bottlenecks, and security flaws and reduces the cost of development and analysis. It relies on factual and detailed system processes rather than abstract models. The study may also help identify potential issues with SA maintenance and heterogeneous and distributed SAs. The study provides sufficient details and information for it to be replicated and implemented.

The perception of SA is crucial for stakeholders in the development and maintenance of a system. Traditionally, SA is documented manually using UML diagrams without guarantee of conforming to the implemented architecture, and the source code does not provide reliable data on the actual architecture. Implicit reflection of SA in code structure and dependencies poses a challenge for extracting the SA model from source code artifacts. Methods such as clustering algorithms and manual system decomposition have been developed to extract the implicit SA from source code artifacts. However, due to the diverse nature of software decompositions and dependency types, there are multiple descriptions of SA.

Tools such as NDepend, Structure101, Sonargraph, and Lattix [10,5] scan codebases to identify elements and their dependencies but mostly produce code-oriented diagrams instead of SA diagrams. This "model-code gap" [9] results from the difference between the mental models of a software system and those used in code. Mapping language constructs to SA concepts, such as layers and components, is an alternative, but the code does not contain enough information about SA to extract a comprehensive model. SA diagrams should indicate the big picture of SA at both the higher level (context and containers) and lower level (components). Although Architecture Description Languages (ADLs) can bridge this gap, they are rarely used in real-world projects. SA diagrams generated automatically by static analysis and modeling tools analyzing existing code bases can reflect code changes in diagrams easily. However, they are polluted with details.

Understanding the dependencies of code entities at a high level of abstraction can convert implicit SA to explicit information. Consensus on architecture descriptions can result in factual architecture with tips for development, design, and process. By comparing SA extracted by different PM algorithms, a comprehensive SA description can be identified. Visualizations are effective in

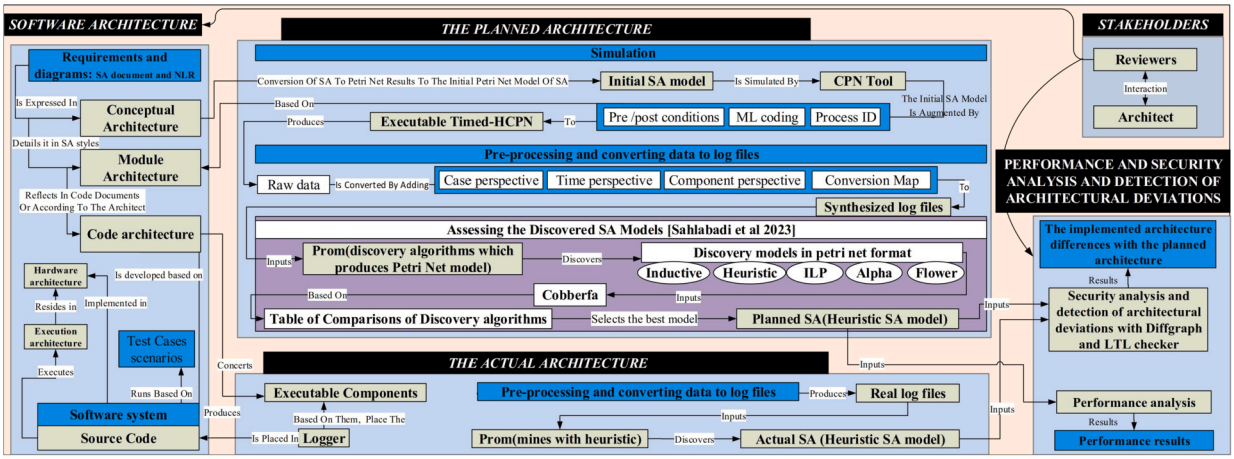


Fig. 11. Verified SA evaluation framework.

understanding the differences and similarities of SA descriptions. There is potential for improvement in conformance checking and implementation based on SA.

In LPMSAEF, existing SA in the source code is considered to be the architectural components that are the major building blocks of the system, and the loggers trace their activities. This approach is similar to ArchJava, which reflects architectural concepts on the implementation with the aid of ADL [21]. The focus of LPMSAEF has shifted from the structural properties of SA to SA behaviors for analyzing performance and security.

Analyzing the previous version of a system's actual SA may lead to a better investigation of the system to improve SA. Comparing the current actual SA with both the planned SA and the previous actual SA allows for project progress measurement and the identification of inconsistencies between design and implementation. The research identifies deviations from the planned SA by the traceability approach to aid the software architect in detecting inconsistencies.

5. Verification

This section presents a detailed explanation of conducting longitudinal case studies during the verification stage of the research. Fig. 11 demonstrates the sequential steps involved in extracting log files from the distributed software system's SA and conducting analysis. The PM is employed for SA model identification, enabling the framework to effectively evaluate the performance and security of SAs. Furthermore, the framework's analyses facilitate bottleneck detection, code conformance, auditing of SA using LTL, and identification of abnormal usage of SA by users. Furthermore, this article focuses on one of the three cases presented, with the online Shop case being the primary example discussed in the subsequent section.

5.1. Online Shop

In the subsequent section, the online Shop case is selected for presentation. The exclusion of other cases is intended to limit the paper's volume.

5.1.1. Overview

The Service-Oriented Architecture (SOA) is an appropriate approach for maintaining large, heterogeneous, and long-lasting distributed systems that are susceptible to changes for maintainability purposes. These systems encounter bottlenecks and performance issues that impede their scalability, and these concerns are taken into account by different stakeholders with diverse interests. Imperfection stands out as a key feature of these systems. Some industrial reference architectures are standards and commonly known [38], and they can be implemented directly with minor modifications. The web-based shopping solution developed with the realization of SOA is considered a standard reference architecture, and it denotes the correct ways that subsystems and components should work.

5.1.2. Requirement and diagrams

The Web-based Online Shopping System allows customers to order items by storing personal information in their account and verifying the validity of their credit card. The supplier checks availability and shipping details, notifies the customer when the order is shipped, and charges their account. The use case model involves customers and suppliers, with the former browsing, selecting, ordering, and viewing items and the latter creating catalogs, processing orders, confirming shipments, billing customers, and servicing orders. Activity diagrams depict these use cases and can be used for business process analysis and SOA model design; they were created based on the analysis of 426 case studies [16].

Listing 1. States of the services and users and their interactions.

```
Variables
    var actname: STRING;
    var request: STRING;
    var res: RESOURCE;
    var service: SERVICE;
    var id,id_r: INT;
    var anonymous:STRING ;
    var x:Command;
    var s:Commands;
    var y:Type;
    var z:TL;
ColorSets
    colset UNIT = unit;
    colset INT = int;
    colset BOOL = bool;
    colset STRING = string;
    colset RESOURCE = subset STRING with ["user","anonymous"];
    colset SERVICE =subset STRING with ["~Cancel_CartAll",
    "user_verification", "Add_RowClient", "Delete_Client",
    "Add_RowItem", "Delete_Item", Category_Name", "Cart_Number",
    "Cancel_CartAll", "Cancel_CartRow", "Unit_Price","Total_Price",
    "AddToCart", "Category_Type", "ItemQuantity", "ClientDeposit",
    "Cart", "ItemNP", "ClientName", "Invoice_Transfer", "UpdateClient",
    "Update_RowItem", "InvoiceBill", "admin_verification", "Client",
    "Item", "UpdateQuantity", "UpdateDeposit", "UpdateCartQuantity",
    "ItemPic"];
    colset TIMEDINT = int timed;
    colset USERS = product TIMEDINT * RESOURCE ;
    colset Command = string;
    colset Commands = list Command;
    colset Type = string;
    colset CT = product Commands* Type;
    colset TL = list Type;
    colset SCL = product Command *TL;
```

5.1.3. Process ID generator

The CaseID function produces case ID.

Algorithm 1: Process ID Generator.

```
Function CaseID(ID):
    if ID < 10 then
        return True;
    else
        return False;
```

The function adds a random time to the token for the purpose of simulating, which is shown in Algorithm 2.

Algorithm 2: Random Function.

```
Function IAT():
    return [exponential(1.0/3.0)];
```

5.1.4. Math Language (ML) code

The variables and color sets are used by places, transitions, and tokens in order to enrich the simulation. In Listing 1, the services and users and their interactions are stated.

5.2. Simulation

Each activity from the activity diagrams starts a chain of services. The activities and services are mapped to the transition in an HCPN. The activity belonging to the activity diagrams is white, while it is black when it belongs to the services. The model indicates how the services should be orchestrated in order to perform an activity of the activity diagram. As indicated in Fig. 12, the white transitions, which are the heads of the chain of services, state the purpose of service orchestration. For instance, for “Select Items From Catalog”, five services can occur. As shown in Fig. 12, after the Unit Price transition, place C15 can opt for either Continue Shopping or Cancel CartRow. Actually, these places are considered in order to simulate various services’ orchestrations.

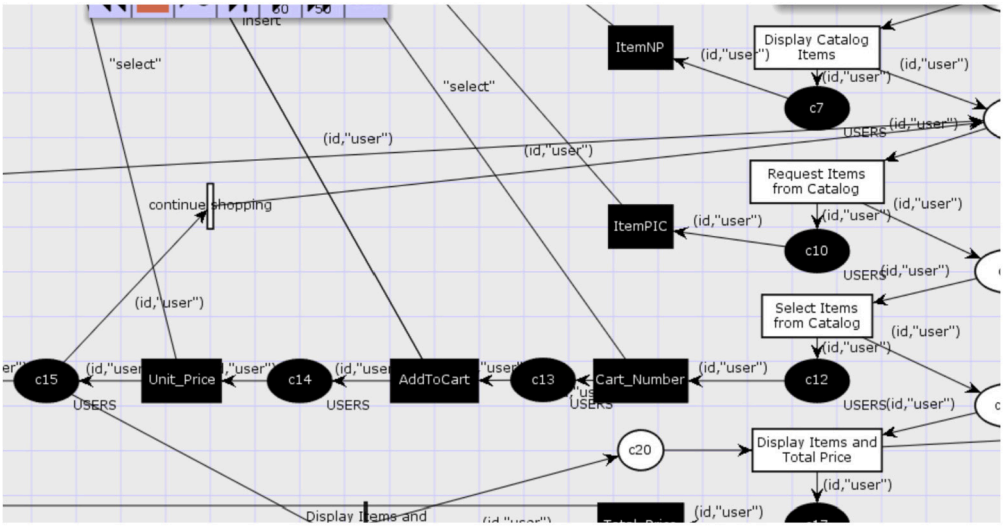


Fig. 12. The view of the simulation.

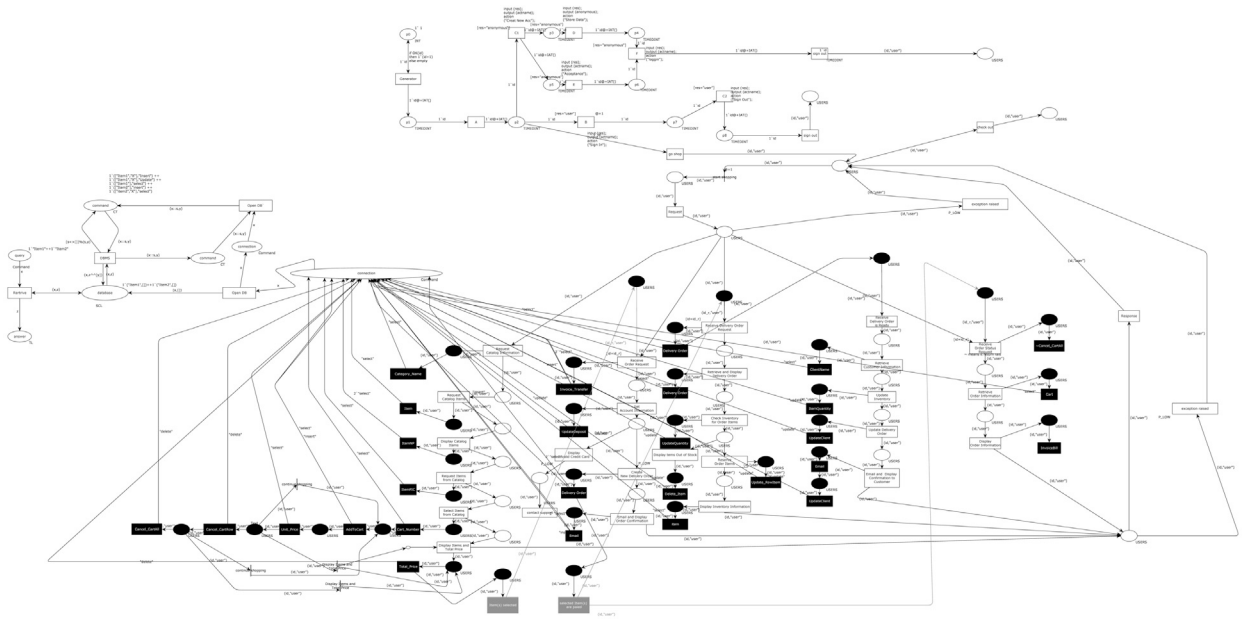


Fig. 13. Holostic view of the simulation.

There are 28 activities and 31 services, plus some transitions in between for database, logging, exceptions, and case generations. As indicated in Fig. 13, this amount of places and transitions is confusing for the software architect. In order to ease stakeholders' understanding of SA, with regard to the property of hierarchical view in HCPN, the transitions are paged under the category of activity diagrams plus two more categories of DB and logging. As indicated in Fig. 14, logging is placed after Case Generator, and DB represents the database of the website.

Web-based servers and transaction processing systems are combined to support online shopping's business processes. The logic of these processes can be either explicitly stated in process models or implicitly implemented during programming time. The resources of these systems might share common databases. So, database management systems (DBMSs) should be looked at seriously [13]. As shown in Fig. 15, there is a place named "connection," which holds the request for services from the database. This place links the services to the database page. As it is indicated, for one purchase, 38 transactions should happen in the database. A guard is a CPN ML Boolean expression that evaluates the condition. As CPN fires the tokens simultaneously, the guard is used to check the tokens with the same ID trigger transactions. As indicated in Fig. 16, the guard [id r = id] checks that the tokens have exactly the same ID, which means these tokens belong to the same case. The preconditions for the activities are held. For example, as indicated in Fig. 17,

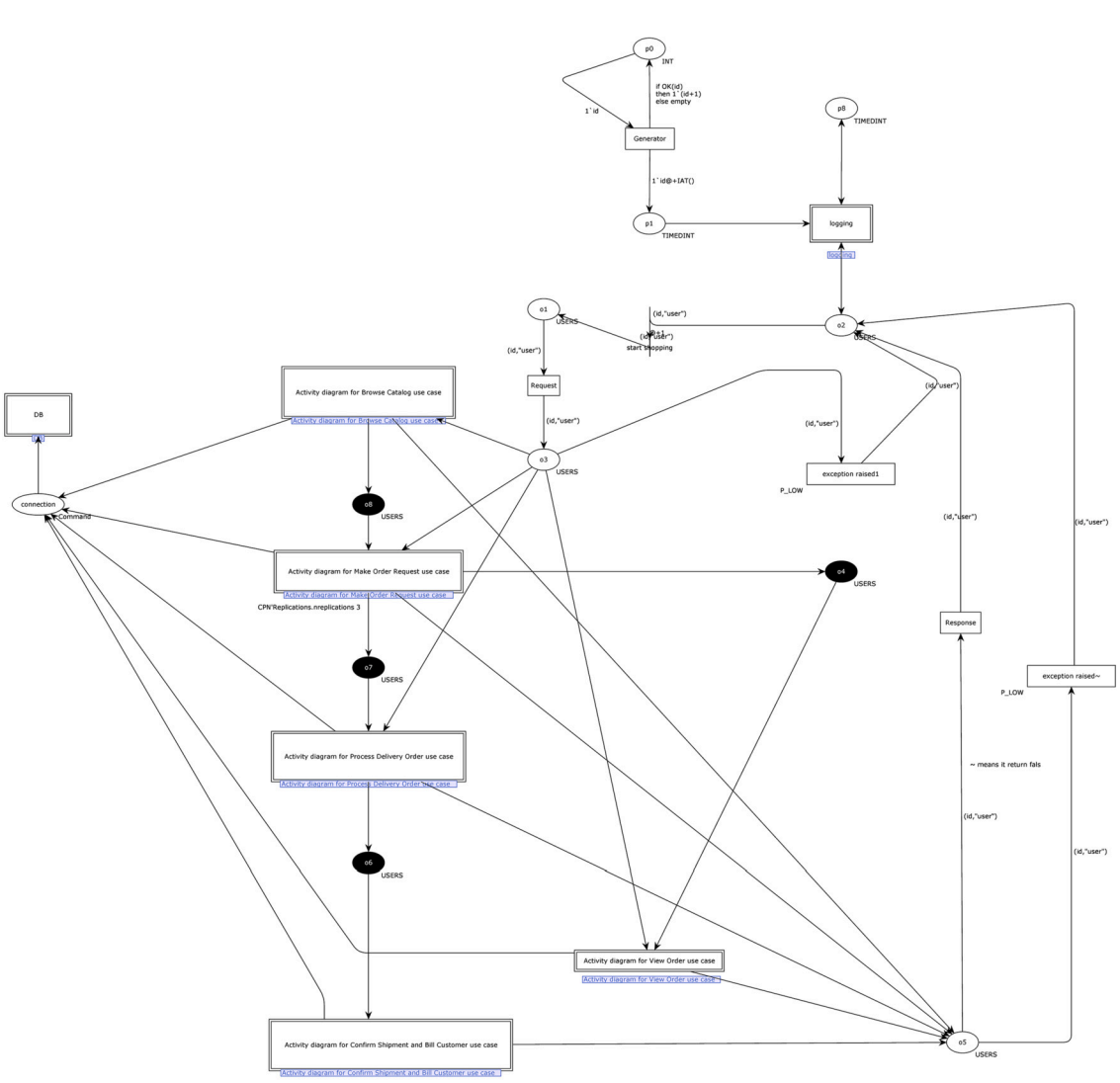


Fig. 14. HCPN model of the online Bebshop.

for the "Display Item And Total Price" activity, at least one item should be selected. Place c20 will receive the token whenever an item is selected. As the transition needs c20 and c11 to run, this precondition is held by place c20.

5.2.1. *Pre/post condition*

There is a constraint that does not allow customers to see the total price whenever they cancel the cart. The non-existence constraint of the CPN tool is used in order to apply this limitation to the simulation. The constraint is shown in the Fig. 18.

5.2.2. *The log file structure*

Having a proper log file is the aim of the simulation so that the mining process can be applied. CPN tools produce a text file containing multiple rows. Each row includes step, time, TransitionName, Instance: PageName, and variables. Each row also contains the attributes of an event. Data attributes should be preprocessed in order to convert them to a log file, which is understandable by Prom. Table 3 shows an example row of the output of our simulation.

As it is discussed in [31], the log files should contain some basic information. Meanwhile, as 21 Indicates, three perspectives should be added in order to make the log file. Case Perspective: The variable "id" represents the case ID of processes, and it is generated by the case ID generator. "TransionName" actually represents the events. Instance: PageName represents the model's subpages, which are actually paged under the category of activity diagrams plus two more categories of DB and logging. "Step" represents the times that the CPN model runs. Time Perspective: Generates a timed model, including information on case generation times, execution times, etc. The timed token was added (colset TIMEDINT = int timed;). As it is shown in Fig. 19, whenever a token fires, the time attributes will increase.

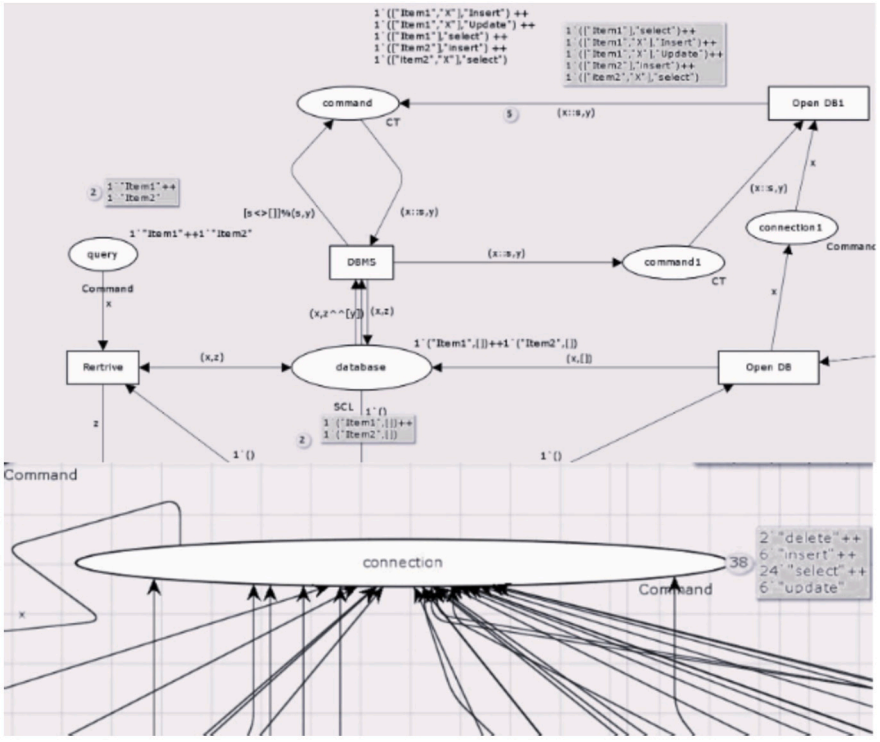


Fig. 15. Database Bottle Neck.

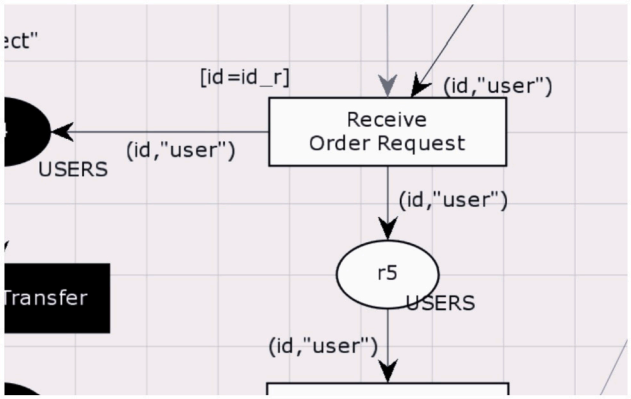


Fig. 16. Guards of the online Shop simulation.

Component Perspective: The TransionName, which represents events, can be considered a component. Also, the services are grouped in sets called components:

- Billing (`Cancel CartAll`, `Total Price`)
- Customer Account (`UpdateDeposit`, `UpdateClient`, `ClientName`)
- CreditCard (`Invoice Transfer`)
- Email (`Email`)
- Catalog (`Category Name`, `Item`, `ItemNP`, `Item- PIC`, `ItemQuantity`, `AddToCart`, `Cart`, `Cart Number`, `Cancel CartRow`, `Unit Price`)
- DeliveryOrder (`Update RowItem`, `Delivery Order`)
- Inventory (`InvoiceBill`, `UpdateQuantity`)
- CustomerInteraction (`start shopping`, `go shop`, `Request`, `Response`, `OrderedIsclicked`, `Did Not`)

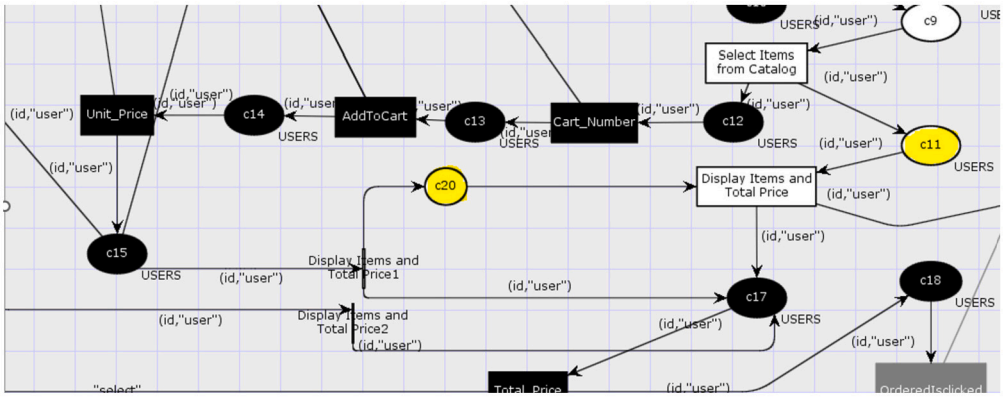


Fig. 17. Dependency of components.

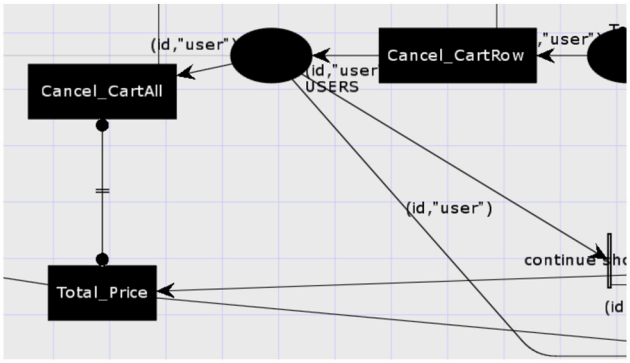


Fig. 18. Pre/post condition of online Shop.

Table 3
Raw data of online Shop simulation.

Step	Time	Transition Name	Instance:PageName	Variables
74	5.0	Unit_Price	Activity_diagram_for_Browse_Catalog_Use_Case	ID = 3

Table 4
Processed event data of online Shop.

Step	Component	Service	Usecase	Case ID	Timestamp
74	Catalog	Unit_Price	Activity_diagram_for_Browse_Catalog_Use_Case	3	00:01:01

TransionName contains the activities of diagrams; these activities should also be ignored (intentions). The CPN tool's output is a txt file, which should be converted to a CSV file. So it becomes portable to prom. The preprocessor application, which is written in the bash script, cleans and removes the duplicated task, adds the component perspective to the txt file, and then converts it to a CSV file (see Table 4).

The data should be mapped to a log file. The log file format has been discussed in [10]. The data are transformed into a Prom's Log file with the aid of the conversion map: (CASEID-> Name (it groups events into traces), TIMESTAMP-> Completion Time, Component-> Resource, USE CASE -> Role). As shown in Fig. 20, the CPN Tool produces the raw data, then it is preprocessed, and finally, with the aid of the conversion map, it becomes a log file, which is understandable for process mining.

Table 5 indicates the typical information stored for the Unit Price event. It is an example that shows how data are stored in a log file. This event belongs to a trace. Fig. 21 shows the Unit Price event is placed in a sequence of events.

5.2.3. Assessing the discovered SA models' results

Fig. 22 states that CPN Tools produces Logs for discovery algorithms of PM in a totally controlled environment, which are suitable for testing Process Mining Algorithms. SA Models' Results To improve the quality of mined SAs, the present study proposes the incorporation of PM within the HSAEF.

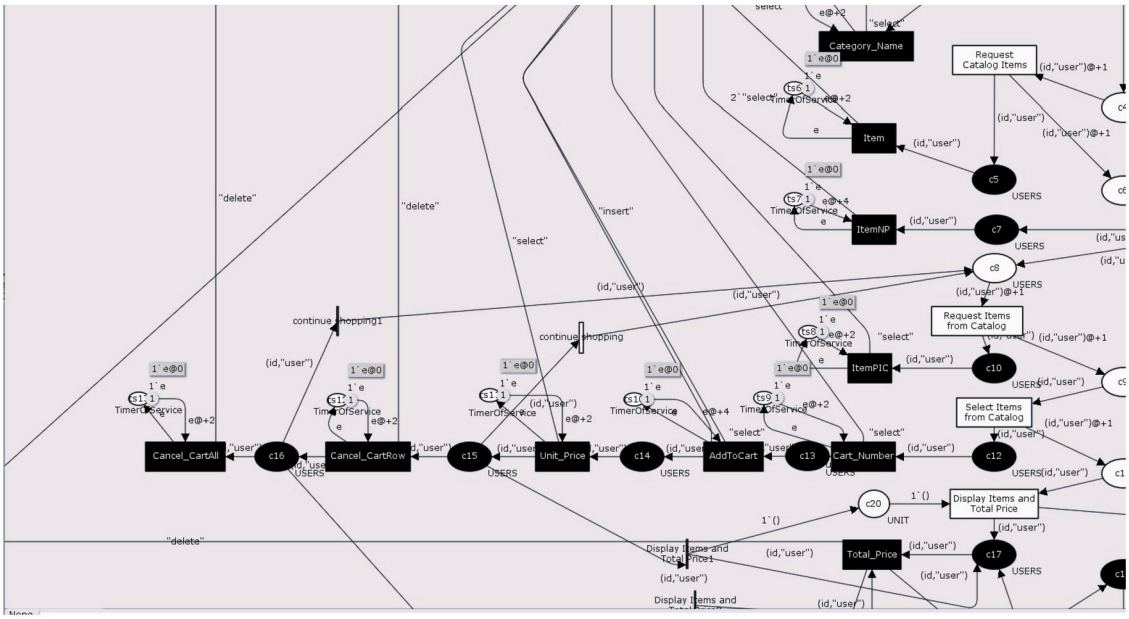


Fig. 19. Timed-tokens of simulation of online Shop.

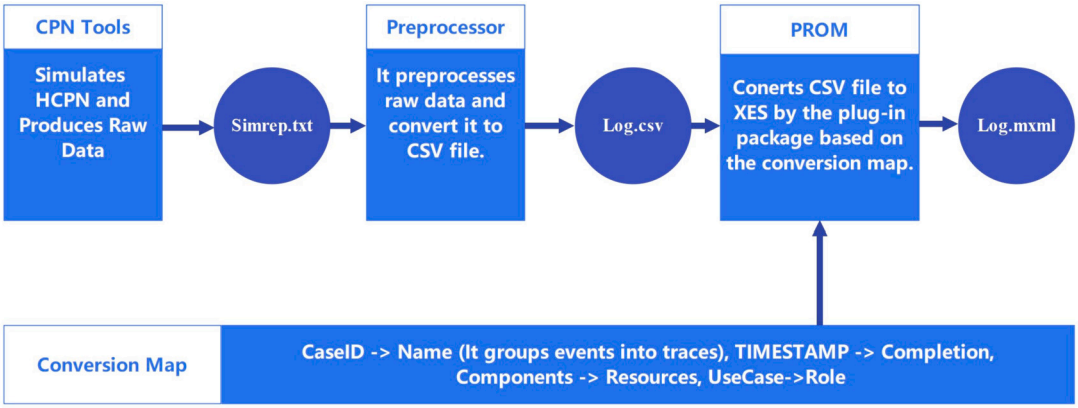


Fig. 20. From raw data to log file for online Shop.

Table 5
A Row of online Shop log file.

Name	Type	Value
concept:name	LITERAL	Unit_Price
lifecycle:transition	LITERAL	Complete
org:resource	LITERAL	Catalog
org:role	LITERAL	Activity_diagram_for_Browse_Catalog_use_case
step	LITERAL	74
time:timestamp	TIMESTAMP	1970-01-01T00:01:01Z

Fig. 21. The event in a trace in online Shop's event log.

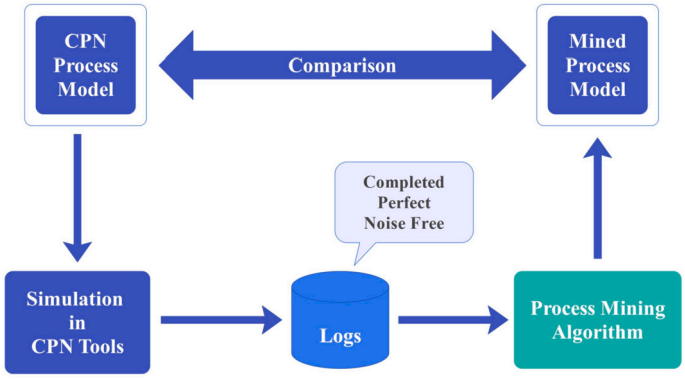


Fig. 22. The event in a trace in online Shop's event log.

Table 6
The services of online Shop; CA = Customer Account.

Service No	Service	Parameters	Purpose	Function
1	User_Verification	(string username, string password)	Sign in	CA
2	Add_Rowclient	(string UserName, string Password, string FName, string LName, string PhoneNum, string Address, string Email, float Deposit)	Sign up	CA
3	Admin_Verification	(string admin name, string password)	Allows Admin to Login	CA
4	Client	Void	Shows All Clients	CA
5	Delete_Client	(int ID)	Deactivates account	CA
6	Add_Rowitem	(string Category, string Name, float Price, int Quantity, string PicURL)	Adds item	Catalog
7	Delete_Item	(int ID)	Deactivates item	Catalog
8	Category_Name	(string cat)	categorizes items	Catalog
9	Cart_Number	void Produce cart number Catalog		
10	Cancel_Cartall	(int ID)	Cancel the cart	Billing
11	Cancel_Cartrow	(int OrderID)	Drops selected item	Catalog
12	Unit_Price	(string name)	Cost the selected item	Catalog
13	Total_Price	(string name, int num)	Sum the selected times' prices	Billing
14	AddToCart	(int cartNum, string itemName, int quantity)	Adds to item order	Catalog
15	Category_Type	void	Shows how many categories are used	Catalog
16	ItemQuantity	(string name)	Shows current item quantity	Inventory
17	ClientDeposit	(int ID)	Checks customer credit	CA
18	Cart	(int cartNum)	Presents order information	Catalog
19	ItemNp	(int cartID)	Returns item information	Catalog
20	ClientName	(int ID)	Returns client's information	CA
21	Invoice_Transfer	(int clientID, int cartNum)	Records invoice and issues it	CreditCard
22	UpdateClient	(int ID)	Records client history	CA
23	Update_Rowitem	(int ID, string Category, string Name, float Price, int Quantity, string PicURL, bool Ar)	Updates item's information	Catalog
24	InvoiceBill	(int orderID)	Issues invoice	Inventory
25	Item	Void	Shows All Items	Catalog
26	UpdateQuantity	(string name, int quantity)	Updates quantity of items	Inventory
27	UpdateDeposit	(int ID, float deposit)	Updates deposit of client A	CA
28	UpdateCartQuantity	(int id, int quantity)	Updates Inventory	Catalog
29	ItemPic	(string name)	Saves item picture	Catalog
30	Email	(address, subject, body)	Sends confirmation Email	Email
31	Delivery Order	(int clientID, int cartNum, date)	Records Delivery Status	DeliveryOrder

5.2.4. Code architecture

The technical design follows the case study that has been stated in the book [23]. The standard B2C pattern is for an online Shop. The primary B2C pattern is implemented: account management, user login, category of search and view, shopping cart, etc. The system is designed, developed, and deployed according to the NET technologies. Common technologies such as ASP NET, Windows XP server, Server controls, Web-based Windows, ADO NET, C# programming language, and Internet information service are used to build this system. In this case study, we do not introduce and focus on these technologies. Alternately, we present a big picture of all services of the system in order to give an overview of the SA used in this system shown in Table 6, the information or five modules:

- `Category overview`: Enables users to search the items by their category.
- `Item search`: Enables users to search the items through the text boxes or dropdown lists.

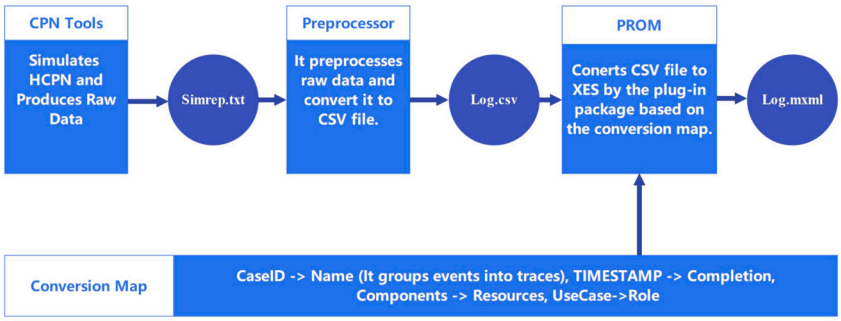


Fig. 23. Conversion map for online Shop.

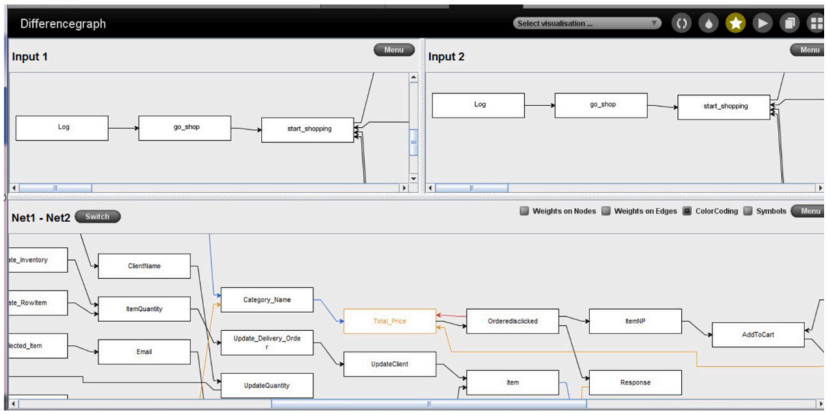


Fig. 24. The result of conformance checking of online Shop.

- `Shopping cart`: Represents the bought items of the user.
- `User login`: Allows users to log in to the system and to create or modify an account for a customer.
- `Order process`: Permits users to order after picking out the items and then end the transactions. The data is returned from the database by SQL commands Medeiros 2005.

5.2.5. Conformance checking

System test cases are manually written and maintained. The test scripts are a sequence of actions to be performed by testers to test various scenarios and flow in the software systems. The system is exposed to test the scenario by testers. Subsequently, the components' activities are logged. The activities of the system services are mapped to a log file based on the conversion map. This research is inspired by [13], so each tester performs all the test cases; whenever a test case is applied in the system, services' activities are logged. The sequences of activities will be added to the rest of the test cases' log files under one case ID, which is a tester case ID. The similar processes described for raw data in order to have a log file should be repeated for the test case scenarios as well, which is depicted in Fig. 23. As shown in Fig. 24, OrderIsClicked is supposed to call the total price according to the planned architecture, while the implemented architecture does not follow the plan. This deviation is detected and solved.

5.2.6. Performance

As shown in Table 7, selecting items from the catalog, displaying items and Total Price, and requesting items from the Catalog are the bottlenecks of the SA. These services are engaged with DB more than other services. As Fig. 25 indicates, the DB is SQL-based. The suggestion is to use a Non-SQL database instead of the current DB to make the application faster.

5.2.7. Assessing the discovered SA models' results

To improve the quality of mined SAs, the present study proposes the incorporation of PM within the HSAEF. The challenges posed by event correlation, timestamp, data cleaning, scoping, and granularity in PM are addressed, and the Play-in, Play-out, and Replay strategy is employed to establish links between events and SA activities. The CoBefra benchmarking method is utilized to evaluate the quality of SAs generated by diverse PM discovery algorithms. The outcomes of this evaluation suggest that heuristic algorithms can proficiently manage low-frequency behavior and noise encountered in the log [31]. CPN simulations are used to opt for the fittest PM algorithm, and the process is depicted in Fig. 22.

Table 7
Performance result of online Shop.

Case Property			Value		
#Cases			10		
#Perfectly-Fitting Cases			9		
#Non-fitting Cases			1		
#Properly started cases			10		
Case Throughput time (Avg)			36.00 s		
Case Throughput time (Min)			0.00 s		
Case Throughput time (Max)			60.00 s		
Case Throughput time (Std. Dev.)			30.98 s		
Observation period			8.00 m		
Select_Item_from_catalog					
Property	Min	Max	Avg	Std Dev	Freq
Throughput Time	0.00 ms	0.00 ms	0.00 ms	0.00 ms	9
Waiting Time	0.00 ms	60.00 s	33.33 s	31.62 s	9
Sojourn Time	0.00 ms	60.00 s	33.33 s	31.62 s	9
#Unique cases (throughput)			9		
Display_Items_and_Total_Price					
Property	Min	Max	Avg	Std. Dev.	Freq
Throughput Time	0.00 ms	0.00 ms	0.00 ms	0.00 ms	5
Waiting Time	0.00 ms	60.00 s	48.00 s	26.83 s	5
Sojourn Time	0.00 ms	60.00 s	24.00 s	30.98 s	10
#Unique cases (throughput)			9		

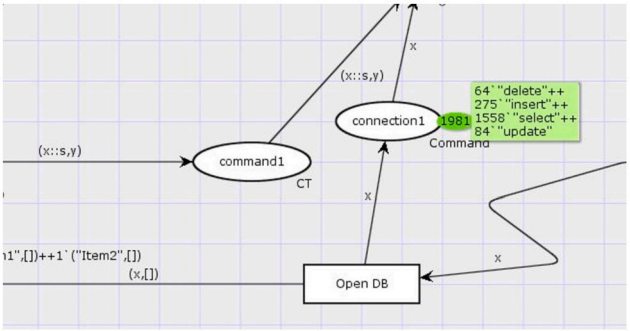


Fig. 25. DB of online Shop.

6. Validation

This section presents how case studies and expert reviews evaluate LPMSAEF. First, the proposed SA evaluation framework has been applied to three industrial case studies for validation. Then, the results of these longitudinal case studies, along with the industrial case studies, are discussed. In the end, experts review the framework. To formulate the theory outlined, we undertook the following steps:

- During the participation in the cases, we discovered the bottlenecks, deviations, and security breaches in the case studies.
- We have taken two points in time per case study and described the differences at each. We made the problems we discovered explicit and explained them.
- Triggered by the acquired experience while working on the cases, we conducted a thorough literature search. By generalizing the research and experiences, we created the axes that are the core of LPMSAEF.
- We validated the proposed method with the case study material.
- Finally, we identified several problems that occurred in the case studies.

In the case studies, we used comparative multi-case analysis. The initial theory building and measurement are done as an iterative process during the full period of all the case studies. We used longitudinal case studies for a period of time ranging from 2 to 6 months, where the quantitative data obtained from the software system and qualitative by the participant observer complemented, in some cases, interviews with key participants in the project or product development team. In other cases, the qualitative data was discussed with project participants to validate the findings. For each case, the research started with a discussion about what happened during the case, resulting in the descriptions mentioned here. Some phases were identified. These phases connected case studies to

Table 8
Result of applying proposed framework on case studies.

Case	Case Study Description	Period (month)	Technologies	No of Detected Issues	No of Major Issue	Effort (staff/hour)	Experience vs Proposed
A	Online Shop	6	N/A	7	5	2/20	+
B	Bug Perdition System	6	N/A	6	3	3/10	+
C	Mobile App for Air-Condition	2	N/A	0	0	2/5	N/A
D	Online Exchange	2	Python, Mongo DB	4	3	2/20	+
E	Workflow of Company	2	SharePoint	3	1	1/10	-
F	Golf Mobile Phone App	2	Android Technology	6	3	3/30	+

find problems and validate LPMSAEF. The quality assessment in this research is conducted through discussions with architects and project team members. It's worth noting that the selection of cases is not random. From our experience, other cases could have been chosen. The cases are similar in that they all involved relatively small, collocated teams facing complex, real-life problems, but they involve a variety of situations from a small product company to small projects at large companies, to moderate-size projects at large customer sites, to the large company changing its way of working.

6.1. How to describe the experiments

SA evaluation methods should recognize potential problems for SAs. The mentioned methods in [30] mostly describe the evaluation process for early evaluation, which means analyzing the designed SA before its implementation. Nevertheless, the late architecture evaluation methods analyze the implemented SAs. The late architectural methods evaluate SAs in the form of steps (e.g., the nine steps of the ATAM and roles (e.g., evaluation team, architect, stakeholders)) [42]. Although these processes explain the basics of the evaluation methods, they ignore attributes of systems that should be studied. The qualities and attributes of software need to be studied in order to evaluate the implemented SAs. This research uses [30] to determine the qualities attributes. The late SA evaluation techniques basically concentrate on extracting the high-level view of systems, which are components of their dependency relations. This particular view is compared with the designed SA. However, sometimes, the SA documentation is not available or updated. Besides, the SAs need to be in situ by using multiple views as an overview and the architects [7,17]. So, in this research, the architects expect the SAs.

- The proposed software architectural evaluation has been conducted on SA of three distributed case studies (A, B, and C in Table 8). The initial framework was built and tested on these cases, as we fully accessed the source code and SA documents.
- LPMSAEF also has been tested by three companies (D, E, and F in Table 8).

6.2. How to conduct an evaluation

SA review methods evaluate the QAs of SAs in depth. However, SA evaluations mainly consume a huge amount of staff. Generally, the SA review methods are expensive to be applied to projects. There are many SA evaluation methods that work based on SA documentation, but the documentation is a problem in every project. The SA reviews need the requirements and specifications of the system in detail. Typically, it takes two to six weeks to prepare this information [42]. However, a lightweight review process that addresses these income possibilities still gives projects some of the benefits of architecture review. Some SA patterns are focused on a lower level of implementation, like object-oriented design patterns. LPMSAEF concentrates on the entire software system's design with a higher level of implementations and modular decomposition.

- LPMSAEF works in a shorter time with less effort.
- LPMSAEF is compatible with small projects as well.
- LPMSAEF does not require heavy architecture documentation. Instead, it exploits any existing documentation and software architect's inferences about QAs of SA and SA's implementation.
- LPMSAEF can be prepared shortly, and with a short review, it can give feedback to a project within some hours. This conduct enables projects to proceed faster in terms of changing requirements.

The main components of the proposed SA evaluation framework are the same as other SA evaluation methods, but they are more focused and simpler. The reviewer should be knowledgeable in SA and the domain of SA. The reviewers cannot belong to the team, or they should not engage in the project seriously so they can come up with a fresh idea to audit and review SAs. Key developers and key interested stakeholders conduct a review in the early stage of development whenever the SAs structure is drawn. There is no need for preparation, but the reviewer should be prepared by studying SAs, requirements, and available documentation such as use cases or user stories. Industry predominantly uses experience methods for software architecture evaluation, while in academics, the preferred method is ATAM [30]. This study aims to compare these two methods by conducting case studies. As a result, the reviewer used their experience-based method in conjunction with the five-step method obtained from ATAM. During the review meeting, these five steps were mentioned that should be followed based on the minimal procedure:

- Reviewers will either walk through scenarios or go through the user stories that are relevant quality attributes.
- They discuss the SA's sketch.
- They determine the architectural style(s) which should be used.
- They examine SA against QAs by reviewing scenarios, implementations, and where in the architecture the implementation occurs.
- They find potential problems with quality or attribute issues.

An SA review should run in a short time because participants are not obligated to prepare. Lightweight documentation is based on comprehensive and conceptual architecture documentation, but it is not easy to produce it for a project. The case studies used a walking skeleton, which is a kind of early end-to-end implementation of the SAs. It is considered as a prototyping to study the concepts of SA. The best time for an SA review is when the walking skeleton is finished. Since there is an implemented SA, the contrary heavyweight, a traditional SA review consumes considerable up-front efforts. It is very unpleasant for the project. For experiences with LPMSAEF, it has been used in multiple projects as case studies. The result of five projects was highly successful, while just one project was partially unsuccessful.

Table 8 demonstrates that the proposed framework outperforms the five-step ATAM and experience methods in handling case studies. It summarizes the projects' results. The "Major issues" column consists of important incompatibilities between the implemented SA and the planned SA and performance issues. Projects are heterogeneous and distributed. For all, architects and developers have had high informal communication with the reviewers. Mostly, they had no proper SA documentation and did not follow a specific architecture pattern. Mostly, they were exposed to changes by coming new user needs. Participants have positive ideas about LPMSAEF and its results. Observing the results and receiving feedback from participants yields several advantages for the framework. The reviews, in some cases (D, F), were halted or prolonged as novice architects have delayed establishing SA requirements, and it made reviewing impossible. Unsuccessful results (case E) notice the review should run in every stage of software development. The miscommunication and mismanagement in the project affected the review, which is why some researchers focus on improving communication among team members during the development process [6]. For case F, as they newly became a mobile development and the Android software development kit was under constant change, this affected feature development and implementation and imposed some overwork.

6.3. Validating LPMSAEF by expert review

LPMSAEF has been validated entirely by five experts in order to determine its positive features, suggest enhancements, and identify its obstacles and weaknesses. The target experts are specialized in the evaluation of SAs with at least ten years of experience. They are computer scientists selected from academics and practitioners from the industry. The sessions were held in Japan Advanced Institute of Science and BriteSoft Solutions (M) Sdn Bhd. The questionnaire was handed out and replied to. These experts have been selected as they have a general overview and background of this research. Hence, they were clarified about the related methods/techniques for the reviewing of the framework. The expert evaluation consists of three parts. Firstly, LPMSAEF was briefly presented to the experts with a demo. Secondly, the demographic information of the expert has been collected. Then, the evaluation sections were designed as a structured questionnaire consisting of both closed and open-ended questions. The closed-ended questions were measured on a 5-point Likert scale ranging from 1 (totally agree) to 5 (totally agree). The open-ended questions are also posed to allow the participants to freely promote their views about LPMSAEF. The expert evaluation questionnaire was designed by two research studies, where [40] systematically evaluated SA's features by a number of experts.

Experts play a crucial role in ensuring that scientific knowledge is reliable, relevant and contributes effectively to the collective understanding of the world. They achieve this by identifying the limitations and implications of research. Additionally, they evaluate the framework's flaws, advantages, and limitations. By doing so, they help to produce robust knowledge that can be applied in real-world scenarios and contribute to the cumulative body of scientific knowledge.

6.4. Closed-ended section

The attained results of the closed-ended questions from the surveyed experts illustrate a reasonable acceptance as all the experts' ratings were different from 2 to 5 out of a scale of 5, which is the maximum. The overall average rate of the experts' evaluation was 3.93 out of 5. The average rates of expert 1, expert 2, expert 3, expert 4, and expert 5 were 3.93, 3.8, 4.6, 3.53, and 3.8, respectively. The expert assessments were summarized in Table 9, which illustrates that all surveyed experts agreed with the achievement of LPMSAEF to reach the main objective of this research. Furthermore, all experts agreed with the context of LPMSAEF, including the goals, outputs, scope, and repeatability for stakeholders (refer to questions 1 and 2 in Table 9). They also agreed that the detected problems in the QAs are very beneficial for the architects (refer to question 3 in Table 9). In evaluating the expected contribution of LPMSAEF, all experts agreed that the lightweightness improves the applicability of this framework (see questions 4 to 9 in Table 9). The majority of surveyed experts moderately agreed on all the features of minimal lightweightness: Stakeholder engagement, minimal documentation, minimal time, and minimal formalism. Additionally, they agree with the life-cycle phase application domain, SA documentation, Mechanisms, stakeholder involvement, forming of analysis tool support, ability to be used in agile methods, and the maturity of the method (refer to questions 10 to 15 in Table 9). Although some experts state data mining can be an alternative tool, all agree that PM performs very well and effectively. In the expected contribution to the knowledge, five experts stated that they are satisfied with the effective proposed framework and consider it significant.

Table 9
Questionnaire.

Q Set1	The main objectives of the framework, the Scope and outputs
Q 1	Is LPMSAEF effective enough to find the potential flaws and problems in the architecture?
Q 2	Is LPMSAEF repeatable in the same scope?
Q Set2	Covered QAs: performance, security, and conformance.
Q 3	Does LPMSAEF scrutinize the quality attributes in a clear way to help architects make better decisions?
Q Set3	Lightweightness features: Stakeholders engagement, minimal doc, minimal time, minimal formalism.
Q 4	Are the concerns covered by the framework well-aligned with the concerns of the stakeholders?
Q 5	Have the stakeholders and their concerns been clearly defined?
Q 6	Are the SA document and understood SA requirements minimal enough?
Q 7	Can the framework be constructed with the available tools, techniques, and people within the minimal time?
Q 8	Is the formalism of SA, which is reflected in Petri Net, minimal?
Q 9	Is the information included in the SA model minimal?
Q Set4	Time of evaluation in the SA lifecycle: Early and late.
Q 10	Does LPMSAEF beneficially cover to evaluate all stages of the SA development?
Q Set5	The evaluation techniques and Tools.
Q 11	Are there any alternative techniques that can be used in the framework to achieve the same objective?
Q 12	Are the Experience-based, Scenario-based, and Simulation-based methods arranged minimally in LPMSAEF?
Q 13	Are the tools supporting Analysis in the best way in the framework?
Q Set6	Ad-hoc and Agility.
Q 14	Does the framework exploit the experience, expertise, and argumentation of software architects effectively in a minimal way?
Q 15	Can the framework match the agile development methodology?
OPEN-ENDED QS	
What are the potential pros (i.e., the positive features) of LPMSAEF? What are the cons of LPMSAEF i.e.,; what are the obstacles in the applying of the framework? What are the weaknesses that are hard to solve while Adopting/Applying LPMSAEF?	

6.5. Open-ended section

First of all, the five experts answered two open-ended Qs in order to recognize the potential advantages and disadvantages when applying the opposed framework. Moreover, the two last Qs were on the specific contexts in which the framework would be or would not be effective.

Expert Number 1 This expert has been working on a formal method, Petri Net and PM, for 32 years. He stated that the framework is largely beneficial for those who act in the industry. The recognized disadvantages were summarized in more detail below:

1. The framework needs to be tested more and more to become mature in many cases.
2. The framework should not have relied too much on the SA's expertise. It should be more automatized for more repeatability.

According to the expert, if the existing method is simpler than LPMSAEF, it will not be useful.

Expert Number 2 The expert has worked especially on SA for more than 20 years. The expert agrees that the low-cost solution of LPMSAEF might be a potential/merit advantage. Nevertheless, some challenges were also posed:

1. The framework requires to be tested with the covering of all the potential scenarios.
2. The Framework should be integrated with the other SA analysis methods.

Expert Number 3 This expert works as a senior software architect and has 18 years of experience. The advantage mentioned by the expert is that LPMSAEF is lightweight and can be implemented with minimal effort. The expert believes using PM as the best tool empowers the framework to achieve all its objectives.

Expert Number 4 This expert works as a software architect and has 13 years of experience. According to the experts, platform independence is the main advantage of LPMSAEF. On the other hand, since Petri Net and PM are not commonly known as software architects, they might not be showing any interest in picking the framework.

Expert Number 5 This expert works as a software architect and has 11 years of experience. The advantage that was mentioned by the expert is that the QA flaws can be detected by LPMSAEF. Meanwhile, the expert identified that the framework is only effective for distributed applications.

6.6. Aggregated findings

The outcomes from analyzing the conducted experts' evaluation were placed into seven main findings. While some of these findings showed the positive features of the proposed artifacts and suggested enhancements, others address obstacles and weaknesses:

Positive Features 1. The framework is highly advantageous.
2. The framework is lightweight and may need lower efforts for implementation.
3. The framework is platform-independent.

Suggested Enhancements 1. More automated.
2. Integrating LPMSAEF with the existing SA evaluation method.

Obstacles and Weaknesses 1. Testing many cases with different scenarios.
2. The absence of familiarity with Petri Net and PM.

7. Conclusion

The case studies were heterogeneous and distributed, indicating the feasibility and power of LPMSAEF. The potential issues in performance and security were discovered in the early and late phases of SA development. These case studies were elaborated on in detail to clarify the implementation of the framework verification. Moreover, the results of three industrial case studies contain testing of LPMSAEFs' results and comparison results of the framework with the companies' method. These results enrich (enhance) the evaluation of LPM-SAEF. Moreover, results attained from expert reviews show their satisfaction, and they considered LPMSAEF useful. In fact, LPMSAEF can be placed within other methods that evaluate SA. It would not be realistic to anticipate a particular proposal to detect all types of SA problems in every phase of SA development. Overall, LPMSAEF can be constantly monitored, as can the performance, security, and conformance of SA. In this research, a novel framework was proposed to evaluate SA performance and security. There are three main steps for proposing this framework. In the first step, the conceptual architecture is achieved by conversion of UML diagrams describing SA and requirements stated in natural language into Petri Net. In fact, the planned architecture is the ideal model of SA, which should be determined. Then, the actual architecture was recognized. After that, the process of SA recovery was implemented to extract SA from the existing source code. In this research, the source codes produce log files by the loggers, which were placed into the code in order to record components' activities. Typically, a large and complex system consists of multiple levels of abstraction which has its own architecture accordingly. LPMSAEF is connected between these layers of abstractions. The elements, components, connector, data, and configuration are identified to adapt PM into a software system for SA analysis. In the last step, the selected SA model should be analyzed from the security and performance views of SA by PM. In this research, when the HCPN model was done, the time tokens were added to the simulation. Then, the data was converted into a log file in order to analyze the performance of SA. The aim is to give an event log and some properties, whether the property holds or not. For this goal, a new language based on LTL has been developed and combined with a standard XML format in order to store the event logs. Giving an event log and an LTL property, the LTL Checker confirms whether the observed behavior matches the (un)desirable/(un)expected behavior or not. The conformance of the planned SA with the implemented SA is considered as the realization flaw and is placed under security problems. In order to check the conformance of the SA, it is essential to analyze the differences between SA models. The Differencegraph is a plugin of ProM that provides the differences and commonalities between process models, which are presented either in heuristic or Petri Net format. In this research, the conformance of the actual and planned SA is compared by a difference- graph. The result from the difference graph indicates the deviations between the models. Three heterogeneous and distributed longitudinal case studies are targeted for the verification of the SA evaluation framework. The framework was implemented within these cases with details. These implementations improved LPM-SAEF. Moreover, there are three black box industrial case studies that tested LPMSAEF. Ultimately, experts reviewed LPMSAEF entirely. The knowledge contributions of this research contain a lightweight framework analysis of the security and performance of the SAs. The framework detects SA-uncovered problems. In fact, the study demonstrates how PM can be utilized for the detection of SA conformance, bottlenecks, and security flaws. It is the first time that PM has been applied in a new context of SA analysis. It has been demonstrated that LPMSAEF is feasible and useful. The framework, with the aid of PM techniques and simulations, mitigates the problem of limited scenarios in the scenario-based technique. The combination of scenario-based, simulation-based, and PM techniques shaped an early-late, lightweight, and platform-independent framework that has proved to be a useful and significant arrangement. The findings of this study consist of the contribution to practice, which developed a framework that could be used (with other SA evaluation methods, if needed) to reduce the costs of SA development and SA analysis. Moreover, this study would enable the developers to be able to check SAs constantly by using PM. The framework was implemented in the case studies, and it showed how the framework can be applied and worked in practice and what its limitations are. A new novel framework was provided as a ubiquitous solution to an industrial problem, and the framework's efficacy was demonstrated. In addition, this research may contribute to indicating the new trends in the industry for SA analysis.

CRediT authorship contribution statement

Mahdi Sahlabadi: Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Software, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Ravie Chandren Muniyandi:** Writing – original draft, Supervision. **Zarina Shukur:** Writing – original draft, Supervision. **Md Rezanur Islam:** Writing – review & editing. **Morteza SaberiKamarposhti:** Writing – review & editing, Visualization. **Kangbin Yim:** Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that support the findings of this study are available from the corresponding author, Mahdi Sahlabadi, upon reasonable request.

Acknowledgements

This work was supported in part by the National Research Foundation of Korea (NRF) Grant funded by the Korean Government [Ministry of Science and ICT (MSIT)] under Grant 2021R1A4A2001810; in part by the Institute for Information and Communications Technology Planning and Evaluation (IITP) Grant funded by the Korean Government (MSIT), Convergence Security Core Talent Training Business (Soonchunhyang University), under Grant 2022-0-01197; and in part by Soonchunhyang University Research Fund.

References

[1] W. van der Aalst, W. van der Aalst, Advanced process discovery techniques, in: Process Mining: Data Science in Action, 2016, pp. 195–240.
[2] S. Al-Azzani, Architecture-centric testing for security, Ph.D. thesis, University of Birmingham, 2014.
[3] M.A. Babar, Making software architecture and agile approaches work together: foundations and approaches, in: Agile Software Architecture, Elsevier, 2014, pp. 1–22.
[4] M.A. Babar, I. Gorton, Software architecture review: the state of practice, Computer 42 (2009) 26–32.
[5] S. Brown, Visualise, Document and Explore Your Software Architecture–Software Architecture for Developers, Leanpub, 2016.
[6] M. De Luca, A.R. Fasolino, A. Ferraro, V. Moscato, G. Sperlí, P. Tramontana, A community detection approach based on network representation learning for repository mining, Expert Syst. Appl. (2023) 120597.
[7] J.B. De Vasconcelos, C. Kimble, P. Carreteiro, Á. Rocha, The application of knowledge management to software evolution, Int. J. Inf. Manag. 37 (2017) 1499–1506.
[8] P. Dugerdil, D. Sennhauser, Dynamic decision tree for legacy use-case recovery, in: Proceedings of the 28th Annual ACM Symposium on Applied Computing, 2013, pp. 1284–1291.
[9] G. Fairbanks, Just Enough Software Architecture: a Risk-Driven Approach, Marshall & Brainerd, 2010.
[10] F.A. Fontana, R. Roveda, M. Zanoni, C. Raibulet, R. Capilla, An experience report on detecting and repairing software architecture erosion, in: 2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), IEEE, 2016, pp. 21–30.
[11] M. Gall, G. Wallner, S. Kriglstein, S. Rinderle-Ma, Differencegraph-a prom plugin for calculating and visualizing differences between processes, 2015.
[12] Gall, S.W.D. Rass, Diffgraph, http://gruppe.wst.univie.ac.at/projects/diffgraph/, 2015, retrieved from University of Vienna website.
[13] H. Gomaa, Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures, Cambridge University Press, 2011.
[14] C. Hofmeister, R.L. Nord, D. Soni, Describing software architecture with UML, in: Software Architecture: TC2 First Working IFIP Conference on Software Architecture (WICSA1), 22–24 February 1999, San Antonio, Texas, USA, Springer, 1999, pp. 145–159.
[15] M.M. Islam, M.T.R. Khan, M.M. Saad, D. Kim, Software-defined vehicular network (SDVN): a survey on architecture and routing, J. Syst. Archit. 114 (2021) 101961.
[16] N.M. Josuttis, SOA in Practice: the Art of Distributed System Design, O'Reilly Media, Inc., 2007.
[17] S. Kudaravalli, S. Faraj, S.L. Johnson, A configural approach to coordinating expertise in software development teams, MIS Q. 41 (2017) 43–64.
[18] W. Li, T. Liu, Z. Xiao, H. Qi, W. Zhu, J. Wang, Tcader: a tightly coupled accelerator design framework for heterogeneous system with hardware/software co-design, J. Syst. Archit. 136 (2023) 102822.
[19] C. Liu, B. van Dongen, N. Assy, W.M. van der Aalst, Component behavior discovery from software execution data, in: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2016, pp. 1–8.
[20] S.T. March, G.F. Smith, Design and natural science research on information technology, Decis. Support Syst. 15 (1995) 251–266.
[21] M. Mirakhorli, J. Cleland-Huang, Detecting, tracing, and monitoring architectural tactics in code, IEEE Trans. Softw. Eng. 42 (2015) 205–220.
[22] A. Mockus, R.T. Fielding, J.D. Herbsleb, Two case studies of open source software development: Apache and Mozilla, ACM Trans. Softw. Eng. Methodol. 11 (2002) 309–346.
[23] S. Pashazadeh, E.A. Niyari, Modeling enterprise architecture using timed colored Petri net: single processor scheduling, arXiv preprint, arXiv:1404.2939, 2014.
[24] E. Patti, A.L.A. Syrri, M. Jahn, P. Mancarella, A. Acquaviva, E. Macii, Distributed software infrastructure for general purpose services in smart grid, IEEE Trans. Smart Grid 7 (2014) 1156–1163.
[25] M. Pinzger, H. Gall, M. Fischer, Towards an integrated view on architecture and its evolution, Electron. Notes Theor. Comput. Sci. 127 (2005) 183–196.
[26] N.S. Rosa, G.M. Campos, D.J. Cavalcanti, Lightweight formalisation of adaptive middleware, J. Syst. Archit. 97 (2019) 54–64.
[27] Q. Rouland, B. Hamid, J. Jaskolka, Specification, detection, and treatment of stride threats for software components: modeling, formal methods, and tool support, J. Syst. Archit. 117 (2021) 102073.
[28] M. Sahlabadi, R.C. Muniyandi, Z. Shukor, A. Sahlabadi, Heterogeneous hierarchical coloured Petri net software/hardware architectural view of embedded system based on system behaviours, Proc. Technol. 11 (2013) 925–932.
[29] M. Sahlabadi, R.C. Muniyandi, Z. Shukur, Detecting abnormal behavior in social network websites by using a process mining technique, J. Comput. Sci. 10 (2014) 393.
[30] M. Sahlabadi, R.C. Muniyandi, Z. Shukur, F. Qamar, Lightweight software architecture evaluation for industry: a comprehensive review, Sensors 22 (2022) 1252.
[31] M. Sahlabadi, R.C. Muniyandi, Z. Shukur, F. Qamar, S.H. Ali Kazmi, Process mining discovery techniques for software architecture lightweight evaluation framework, Comput. Mater. Continua 74 (2023).
[32] M. Sahlabadi, A. Sahlabadi, R.C. Muniyandi, Z. Shukur, Evaluation and extracting factual software architecture of distributed system by process mining techniques, Asia-Pac. J. Inf. Technol. Multimed. 6 (2017) 77–90.
[33] F. Sattler, S. Böhm, P.D. Schubert, N. Siegmund, S. Apel, Seal: integrating program analysis and repository mining, ACM Trans. Softw. Eng. Methodol. (2023).
[34] R. Terra, M.T. Valente, K. Czarnecki, R.S. Bigonha, A recommendation system for repairing violations detected by static architecture conformance checking, Softw. Pract. Exp. 45 (2015) 315–342.
[35] D. Torre, Y. Labiche, M. Genero, M.T. Baldassarre, M. Elaasar, UML diagram synthesis techniques: a systematic mapping study, in: Proceedings of the 10th International Workshop on Modelling in Software Engineering, 2018, pp. 33–40.
[36] M. Vidoni, A systematic process for mining software repositories: results from a systematic literature review, Inf. Softw. Technol. 144 (2022) 106791.
[37] M. Wahler, T. Gamer, A. Kumar, M. Oriol, Fasa: a software architecture and runtime framework for flexible distributed automation systems, J. Syst. Archit. 61 (2015) 82–111.

[38] R. Wang, C. Gu, S. He, Z. Shi, W. Meng, An interoperable and flat industrial Internet of things architecture for low latency data collection in manufacturing systems, J. Syst. Archit. 129 (2022) 102631.
[39] D. Xu, K.E. Nygard, Threat-driven modeling and verification of secure software using aspect-oriented Petri nets, IEEE Trans. Softw. Eng. 32 (2006) 265–278.
[40] C. Yang, P. Liang, P. Avgeriou, A survey on software architectural assumptions, J. Syst. Softw. 113 (2016) 362–380.
[41] N.M. Zaki, A. Awad, E. Ezat, Extracting accurate performance indicators from execution logs using process models, in: 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA), IEEE, 2015, pp. 1–8.
[42] A. Zalewski, S. Kijas, Beyond ATAM: early architecture evaluation method for large-scale distributed systems, J. Syst. Softw. 86 (2013) 683–697.
[43] B. Zuhaira, N. Ahmad, T. Saba, J. Haseeb, S.U.R. Malik, U. Manzoor, M.A. Balubaid, A. Anjum, Identifying deviations in software processes, IEEE Access 5 (2017) 20319–20332.