SJCS Scientific Journal of Computer Science

futuristech PT. Teknologi Futuristik Indonesia

Date of publication April 24, 2026, date of current version April 24, 2026.
Digital Object Identifier 10.64539/sjcs.v2i2.2026.456

Review

Software Ecosystem Architectural Challenges and Mitigation Strategies: A Systematic Literature Review

Inayat Ur Rahman[1], Atta Ur Rahman[1,*], Sara Shazad[2], Sajid Ur Rahman[3]

[1] School of Computer & Software, Nanjing University of Information Science & Technology, Nanjing, 210044, China; e-mail: 202551200013@nuist.edu.cn, inayatsoft@gmail.com (I. Ur Rahman), 202451200014@nuist.edu.cn (A. Ur Rahman).
[2] Department of Computer Science, University of Peshawar, Peshawar, Pakistan; e-mail: sara@uop.edu.pk (S. Shazad).
[3] School of software engineering, University of Malakand, Khymber Phwa, 18800, Pakistan; e-mail: Sajidkvk753@gmail.com (S. Ur Rahman).
* Correspondence

The authors received no financial support for the research, authorship, and/or publication of this article.

Abstract: Software ecosystems (SECO) play a crucial role in modern software development by enabling accelerated innovation, collaboration among multiple stakeholders, and efficient utilization of shared resources and technologies. However, achieving these benefits requires robust, adaptable, and well-structured architectural design and management. Despite their importance, SECO architectures face several critical challenges, including interface instability, security vulnerabilities, scalability limitations, governance complexity, sustainability concerns, and evolving ecosystem dynamics. Although prior studies have explored individual aspects of SECO, there is a clear research gap in providing a comprehensive and systematic synthesis of architectural challenges and their corresponding mitigation strategies. In particular, no systematic literature review (SLR) has thoroughly examined these issues in an integrated manner. To address this gap, this study aims to systematically identify, categorize, and analyze architectural challenges in SECO and evaluate existing mitigation techniques. A structured SLR methodology is employed to collect, assess, and synthesize relevant literature, leading to the development of a conceptual framework that organizes both challenges and solutions. The findings reveal that key mitigation strategies—such as modularization, variability management, custom design approaches, and sandboxing—can significantly improve architectural stability, scalability, and sustainability. These results provide valuable insights for both researchers and practitioners by offering a consolidated understanding of SECO architectural issues and practical guidance for designing more resilient and sustainable software ecosystems.

Keywords: SECO; SLR; Mitigation Strategies; Software architecture; Software ecosystem architecture.

1. Introduction

Software ecosystem (also referred to as SECO) indicates a fairly new trend in the field of software development. It is a technique of determining how multiple software development companies are associated with each other when a large portion of various independent variables are present [1]. Alterations in the magnitude of the software ecosystems definition also bring in varying definitions. Several definitions of software ecosystems have since been applied, including: According to Soureya [2], a set of software products and the firms that make them that support, mediate, and automate the activities as well as transactions of the players on the interconnected social or economic ecosystems. A software ecosystem consists of actors who have come together create software as a system and simultaneously interact in a joint software and service market, and the interrelation between them as defined by E. Onagh and M. Nayebi [3]. Such interactions tend to be facilitated by an analogous techno-logical infrastructure or operation by the employee of the exchange of information, ideas, and objects. The interdependence and network of connections are getting more emphasis. Software ecosystems are usually under the management of one or a few coordinating parties with a financial interest in the success of the ecosystem. These coordinators are often also responsible for the supporting infrastructure that underlies the

ecosystem, including a software platform company. Nevertheless, fewer traditional organizers of the ecosystem may have a bearing on the ecosystem, including open-source platform consortia or even individual developers. Such individuals who have been recipients of the trend to expand the software ecosystem and are in a position to control the development of the platform or the environments around it are said to be software ecosystem coordinators.

They are also often taking a partial responsibility in creating the underlying technology most of the time as by E. Onagh and M. Nayebi [3]. Nowadays, such co-innovations are all the more widespread than either single-source or solo innovations, as they involve many actors. Due to an innovation, businesses collaborate and compete to help in supporting new products, and as a result, co-evolve their capabilities, necessitating businesses to satisfy consumer demand and eventually embrace the new evolution of advances [4]. These open alliances with vendors, clients, outsourced companies, and related product or service creators, technology organizations, and a multiplicity of other organizations affect and are affected by the development and delivery of the company's own offerings. Due to these perspectives, scholars created software ecosystems as an attempt to articulate a different perspective on the software business. The emergence of Google Android, Apple iOS, Microsoft, and Salesforce.com ecosystems has helped to attract attention to the growth of SECO. Software innovation takes place when different parties, including the outside and software firms, collaborate to achieve a certain goal. Bearing in mind that not much research on SECO had been conducted till recently, both scholars and business professionals.

A close analysis of software ecosystems was done to talk about the features, merits, and demerits of SECO. The paradigm of software development is rapidly evolving [5]. The number of people who use Software as a Service (SaaS) nowadays heavily depends on computing everywhere. In this case, and with the advent of issues such as Distributed Software Development (DSD), retaining a central Architecture in most scenarios is not enough amongst the business developers. To cooperate, they should open up their architectures to external developers. This alliance between a wide variety of software solutions, companies, and developers has created a new idea in development. This type of situation was termed Software Ecosystem (SECO) [5].

A software platform that is open is called a SECO and typically consists of a keystone, a platform, and several specialized agents. The centralizer also has an effect on the development of the platform, as well as the management of the relations with third parties. Nevertheless, niche agents refer to the ones that impact the development of the ecosystem [5]. Application evaluation and the iOS ecosystem are offered by Apple at a monthly fee and a share of the monthly earnings from app sales [6]. Other examples of a (smartphone) software ecosystem, in addition to iOS, will be the Android ecosystem, a popular system of Google [6]. The prediction was made earlier in 2012 that a greater number of smartphones would be sold as compared to laptops [6]. They contrast with the early years of software engineering when a software product was the result of the effort of a single software vendor to create a monolithic product; instead, the software packages nowadays typically depend on the efforts of third-party vendors or open-source sources [3]. Ecosystem extensions are the products of partnerships between software development firms and service providers, where both the software products, components, and technologies suppliers and buyers engage in designing competitive value. The well-being of such a product software company is no longer merely reliant on the excellence of its own creation but on how it handles its relationships [3].

There are various ways in which software and tangible things are different. There are no physical limitations with regard to software; therefore, the primary limitations of the software are intellectual, social, and financial [3]. It is unheard of to see a gross profit margin of 99 per cent on the goods sold in other businesses [3]. That is, there are no costs that can be linked to the reproduction of software products. SECO has the following advantages, which are enumerated [4]: 1) The presence of appeal to new players and the co-evolution and development of software in the concerned company. 2) Decreases the amount of resources it takes to create and share software. 3) Assist with the evaluation and cognition of software Architecture. 4) Phase communication and co-operation between different autonomous software growth economists. A software ecosystem has many advantages, but there are also many risks and challenges associated with it. These challenges or risks should be paid more attention to. Software ecosystem challenges have further groups with further description of software ecosystem challenges, which are Architecture, Governance, Sustainability, Data Analytics, Dynamics and Evolution, as well as Domain-Specific Ecosystem [7].

In several studies, SECO has been found to have architectural problems as the most important ones [8], [9] has indicated that one cannot develop a software ecosystem design without considering social issues when considering management (technical) processes and business legislation. There must be communication between the actors and the platform upon which the ecosystem is built. Researchers have come up with numerous architectural problems, as well as additional methods of mitigating such problems [4], [10], [11]. Nonetheless, no study has investigated the whole issue and the impact of the various possible mitigation methods on the architectural concerns in the wake of the SLR. Therefore, it may be difficult to trace the connection between mitigation strategies and architectural issues. This is necessary to identify every Ar-

chitecture issue as an outcome. The paper is trying to fill this knowledge lack of knowledge on in the literature by thoroughly classifying all architectural barriers and remedies. The objective of the current study will be achieved by formulating the following objectives:

- SLR has been conducted to determine the full list of the SECO possible concerns and mitigation plans.
- We were also able to determine all of the threats to the SECO architecture, which is important to the research.

The paper was organized as follows: Section 2 contains the literature review and the related work that has been utilized in previous literature and research. It gives concise literature on SECO, SECO Architecture, and mitigation strategy or practices issues. Research design/ methodology is section 3, which is involved in this research study, as the method of the research is set to be carried out to answer the research questions and proposed conceptual framework. The Section 4 is results and findings, where the results of the research that have been addressed are defined. Section 5 is the conclusion section of the research work, and this section has outlined the proposed work for the future.

2. Literature Review

The literature review is discussed in order to be better aware of the terms used in the investigation. The review paper on software ecosystem, the challenges with software ecosystem Architecture, factors affecting software ecosystem Architecture problems, mitigation, and strategies have been discussed in detail. The gap in research of the current studies is also mentioned in this section.

2.1. Software Ecosystem

Software ecosystem (sometimes referred to as SECO) is a relatively recent designation used to refer to a relatively recent development in the software development business. It is a way of forging connections among the limited software development companies with each other when there are many independent variables [1]. The definition of software ecosystem has also had diverse definitions due to variation in the definition itself. The term software ecosystem is nowadays defined in various fashions, and a software ecosystem has been defined [2] as the set of software solutions that facilitate, enable, and automate the activity and the interplay between the members of the connected social or commercial ecosystems and the firms offering those solutions [3] coined the term software ecosystem, which is described as a group of actors that work together to form software solutions which are run using a common software and services market, with interactions among players shaping the overall structure of the ecosystem. Such interactions operate based on the exchange of knowledge, ideas, and artefacts, and usually follow up with an identical technology environment or market. More attention is being given to connectivity and dependence in business connections. Software ecosystems are normally spearheaded and organized by one or more coordinating parties who have a vested interest in the success of the ecosystem.

These coordinators, as a rule, are also responsible for the underlying technology, which acts as the foundation of the ecosystem, like a software platform creating a business. Nevertheless, the number of traditional ecosystem organizers is also lower, including open-source platform consortia or even a lone developer who can influence the ecosystem. Software Ecosystem coordinators have been described as the parties that have enjoyed the software ecosystem growth and are capable of influencing how the platform or its ecosystems evolve. As is generally the case, they are also (in some measure) in charge of the further development of the underlying technology [3]. Multiple players now become sources of innovations, and they are co-innovations. The corporations coevolve their potential in the context of innovation, collaborate and engage in competition to assist the new goods, satisfy the consumer needs, and ultimately follow another series of innovations [4].

2.2. Software Ecosystem challenge

Some of the benefits that SECO has are as follows [4]:

1)facilitates the prosperity of software co-evolution and innovation of the organizing organization, and strengthens the appeal of new players. Cuts the production and distribution cost of software. Create a software architecture study and knowledge. 4) Favors sharing information and cooperation among competitors in the development of various software separately [7]. The software ecosystem is quite risky and challenging, and beneficial. More likely, it is such struggles or threats that would be taken into account. Software ecosystem issues are also established by a variety of fields, which encompass Architecture Governance, Sustainability, Data, Analytics, Dynamics, and Evolution Domain-Specific Ecosystem [7].

2.3. Software Ecosystem Architecture challenges

In other words, researchers have developed the SECO architectural problems as the most significant ones [8]. Manias mentions that building a software ecosystem design can never take place without looking at social aspects and management (technical) processes, and business laws [9]. There is a need for actors to interact with each other and with the platform on which the ecosystem is underpinned. Some of the architectural issues identified by the researchers include interface stability, security, reliability, scalability, longevity, solidity, and integration application [12]-[17]. There are also other mitigation measures, such as Variable design, Custom design, Assessment of ecosystem health, Metrics, Framework, and Sandbox [18]-[20]. It has

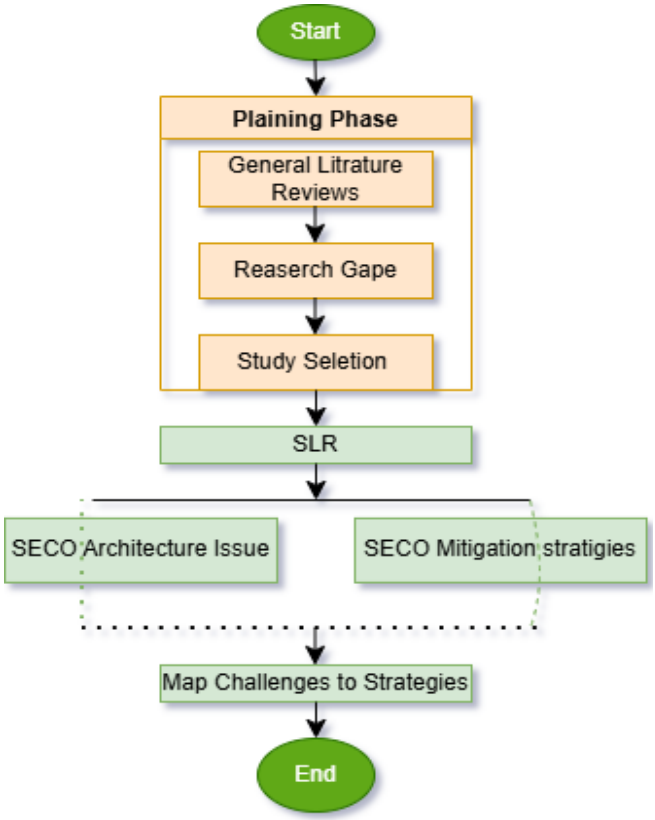

Figure 1. Research Design.

also been proposed. Nonetheless, all the studies have not identified that or enquired how the differing feasible mitigation plans could influence the architectural issues after the SLR. This may lead to the inability to identify the specific relationship between mitigation measures and architectural issues. Due to this fact, it is imperative to detect all the architectural issues. SLR has been carried out by systematically grouping all the challenges to Architecture and solutions.

2.4. Software ecosystem Architecture challenges & mitigation strategies

Software Ecosystem Architecture was regarded as a big issue in the software ecosystem. Several architectural issues, such as interface stability, security, dependability, scalability, durability, solidity, and application integration. The other issues of the software ecosystem include architecture as a critical issue. These problems and hurdles have been reduced through several mitigation measures. The alternative mitigation strategies include: Variable Design, Custom Design, Ecosystem Health Assessment, Metrics, Framework, and Sandbox [18]-[20].

3. Research Methodology

Research design gives the method of research that has been adopted with the aim of answering the research questions of this study. There are three stages of a research approach. At the beginning, it fulfills a Systematic Literature Review (SLR) to identify the issues of Software Ecosystem in Architecture and mitigation strategies and practices of SECO.

3.1. Research Design

In this section, the steps for this study or research that have been taken are discussed in detail. In Figure 1 the flow of methodology for the research Design.

3.2. Systematic Literature Review

SLR is an objective and approach to gathering data about the existing research findings concerning the research topics in an objective and consistent fashion. Besides, the SLR systematically incorporates research work. The SLR will be seeking to have credible, manageable and precise methods to be used in the objective assessment of research subjects. The main goal of this study is to identify the most suitable study by using inclusion and exclusion criteria from previous studies, and also to identify the quality standards to be used to assess the findings and information drawn out of the previous studies.

An SLR approach was developed to find the software ecosystem Architecture issues, affecting factors to software architecture, and its mitigation strategies. Using [21] guidelines in this study were used. Three stages of SLR applied in this SLR, i.e. preparing the review, conducting the review and reporting the review. Figure 2 demonstrates the stages of SLR, as well as their sub-steps.

Furthermore, these research steps have been divided into subsections.

- Planning the review phase is employed to delimit the plan that we have devised to carry out the SLR.
- The second is conducting review phase is a significant stage of SLR. This stage adopts the search strategy identified in the initial phase of SLR in the process of acquiring data contained in the literature.
- The third and final stage of SLR is the reporting of the review. It is applied to document the findings of the two previous stages. Each of the steps will be discussed below.

3.2.1. Planning The Review

This stage entails 6 stages. i.e.

- Research Question
- Electronic Databases
- Search String
- Inclusion Criteria
- Exclusion Criteria
- Quality Criteria for Study Selection

All the steps are discussed in detail below.

3.2.1.1. Research Questions

In this section we discuss research question for our research are in Table 1 is given.

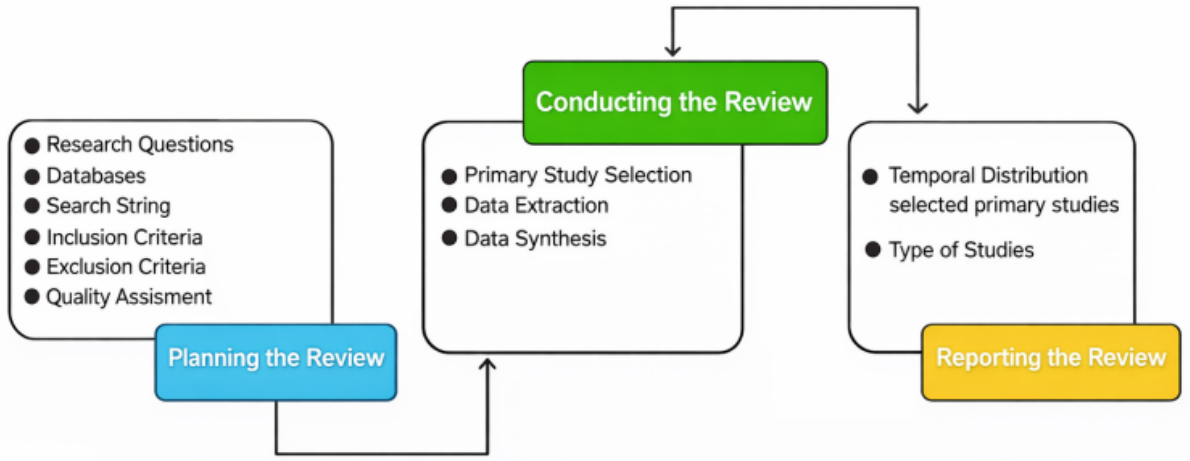

Figure 2. SLR Protocol.

Table 1. Research Questions.

S. No.	Research Question	Answer
RQ1	What are the SECO Architecture factors listed in the literature that affect software ecosystem architecture?	The response to this research question would be to offer a comprehensive list of Architecture Issues in Software Ecosystem that have been elicited through the undertaking of SLR.
RQ2	What are the mitigation strategies given in the literature for SECO Architecture issues in the software ecosystem?	The answer to this research question is to provide a detailed list of mitigation strategies for SECO Architecture issues in SECO that have been extracted by performing SLR.

Table 2. Quality Assessment Questions.

S. No.	Quality Assessment Question	Score
Q1	Is the study talking of the Architecture problems in SECO?	Yes = 1, Partial = 0.5, No = 0
Q2	Does the study propose any mitigation strategies, practices, tools, and techniques to overcome Architecture issues in SECO?	Yes = 1, Partial = 0.5, No = 0
Q3	Does the literature discuss and evaluate framework to mitigate challenges in SECO?	Yes = 1, Partial = 0.5, No = 0
Q4	Does the Study related to SECO?	Yes = 1, Partial = 0.5, No = 0

3.2.1.2. Electronic Data Base

Based on the recommendations, the 6 data sources have been used.

- Google Scholar
- IEEE Explore
- Science Direct
- Research Gate
- Semantic Scholar
- Online Wiley Library

3.2.1.3. Search String

The following search string used to extract the nominated studies is the following:

(software ecosystem OR ecosystem OR SECO) AND (architecture issues OR Architecture challenges OR factor affecting Architecture OR Architecture barriers OR Architecture problem OR Architecture risk OR Architecture thread) AND (Mitigation strategies OR practices OR solution OR recommendation OR guideline OR Safeguard)

3.2.1.4. Inclusion Criteria

For Inclusion criteria the following instruction.

- Studies must be published in Journal or Conference.
- Studies must be written in English language.
- The links of the studies are accessible.
- Studies having full text.
- Studies that answering questions of the research.
- Studies which is published from 2000 to 2026.
- Studies that refers mitigation practices for Architecture in SECO.

3.2.1.5. Exclusion Criteria

Using Following criteria some studies were excluded.

- Books and various blogs are excluded.
- Slides, tutorial, editorials etc. Are removed.
- studies excluded which language not English.
- Duplicated or repeated studies.
- Technical reports and white papers.
- Studies were not available in full-text.

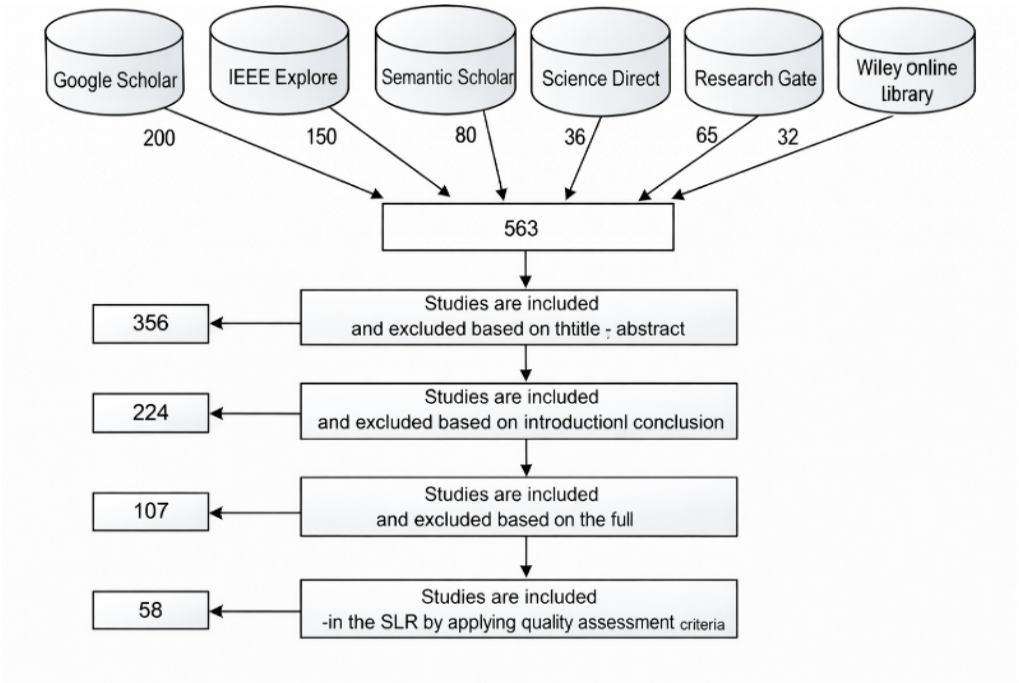

Figure 3. Tollgate Approach Distribution.

Table 3. Tollgate Approach for Primary Study Selection.

Electronic Databases	L1	L 2	L3	L4	L 5
Google Scholar	200	130	90	30	25
IEEE Explore	150	95	60	40	20
Semantic Scholar	80	50	30	15	5
Science Direct	36	21	11	8	4
Research Gate	65	40	20	9	3
Willey online library	32	20	13	5	1
Total	563	356	224	107	58

- Studies that are not able to access internet are not included.
- Do not discuss mitigation practices for Architecture in SECO.

3.2.1.6. Quality Criteria to study selection

The selection of the study was rigorous by implementing a quality assessment scoring mechanism. The assessment of each of the studies was conducted according to the preset criteria, and the scores were given as follows: Yes = 1, Partial = 0.5, and No = 0.

There was an inclusion criterion that would include the final analysis that is 2.0 or higher (out of 4). The inclusion threshold was set at a minimum quality 2 out of 4. Study Scoring below this threshold were excluded to ensure methodological rigor and reliability. Table 2 establishes the criteria question of quality assessment.

3.2.2. Conducting the Review

The review is conducted of three steps i.e.

- Study Selection
- Data extraction
- Data synthesis

Each of the steps is discussed below.

3.2.2.1. Primary Study Selection

Tollgate approach [22] is adapted for the study. This phase has 5 steps.

- **Level 1:** Search the relevant studies based on the search string and inclusion criteria.
- **Level 2:** Inclusion and exclusion of studies are based on the title and abstract.
- **Level 3:** The introduction and conclusion are used to include and exclude the studies.
- **Level 4:** The inclusion and exclusion of studies are done depending on the entire text.
- **Level 5:** The SLR entails final primary studies through the application of quality assessment criteria.

Used to select the primary studies and thoroughly to purify all the discovered studies. Tollgate approach has 5 levels as illustrated in Table 3. The search strings used in the above electronic database found 563 studies. Besides, 58 large studies were nominated to SLR by means of five-level tollgate approaches Figure 3 represent the tollgate data representation.

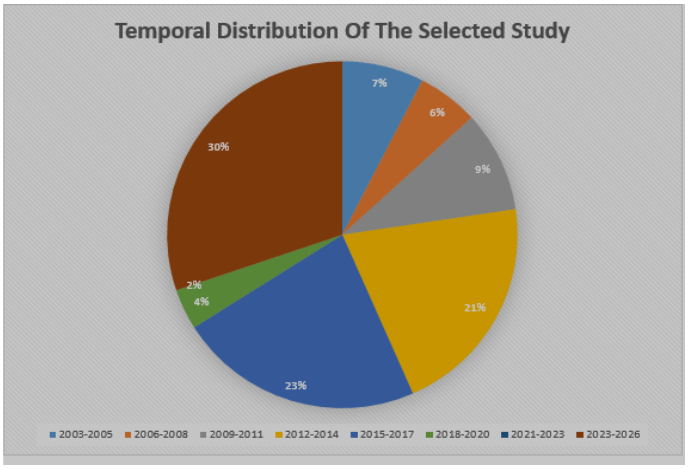

Figure 1. Temporal Distribution of the Selected Studies.

Table 4. Quality Evaluations of the Studies.

Ref No	Year	Q1	Q2	Q3	Q4	Total Quality Score
[1]	2025	1	1	1	1	4
[2]	2025	1	0	0	1	2
[3]	2025	1	0.5	0	1	2.5
[4]	2025	1	1	1	1	4
[5]	2025	1	0	0	1	2
[6]	2025	1	0	0	1	2
[7]	2025	1	0	0	1	2
[8]	2025	1	0	0	1	2
[9]	2026	1	1	1	1	4
[10]	2025	1	1	1	1	4
[11]	2024	1	0	0	1	2
[23]	2024	1	0	0	1	2
[12]	2013	1	0	0	1	2
[24]	2014	1	0	0.5	1	2.5
[25]	2017	1	0	0	1	2
[26]	2020	1	0	0	1	2
[27]	2020	1	0	.5	1	2.5
[28]	2013	1	0	1	1	3
[29]	2014	1	0	0	1	1.5
[30]	2017	1	1	1	1	4
[31]	2016	1	0	0	1	2
[32]	2024	1	1	0	1	3
[33]	2016	1	0	0	1	2
[34]	2004	1	1	1	1	4
[35]	2010	1	0	0	1	2
[36]	2010	1	1	1	1	4
[18]	2017	1	1	1	1	4
[37]	2012	1	1	1	1	4
[38]	2014	1	1	0	1	3
[13]	2013	1	1	1	1	4
[39]	2013	1	1	0	1	3
[40]	2010	1	.5	1	1	3.5
[41]	2016	1	1	0	1	3
[42]	2020	1	.5	1	1	3.5
[43]	2017	1	1	1	1	4
[44]	2012	1	0	0	1	2
[45]	2013	1	1	1	1	4
[14]	2024	1	0	0	1	2
[19]	2015	1	0	1	1	2
[46]	2024	1	1	1	1	4
[47]	2015	1	0	0	1	2
[15]	2007	1	0	0	1	2
[16]	2015	1	1	0	1	3
[17]	2016	1	0	1	1	3
[48]	2016	1	1	0	1	3
[49]	2025	1	.5	0	1	2.5
[20]	2011	1	0	0	1	2
[50]	2011	1	0	0	1	2
[51]	2005	1	1	0	1	3
[52]	2008	1	0	0	1	2
[53]	2009	1	0	.5	1	2.5
[54]	2007	1	0	0	1	2
[55]	2012	1	0	0	1	2
[56]	2004	1	0	0	1	2
[57]	2011	1	1	0	1	3
[58]	2009	1	.5	0	1	2.5
[59]	1998	1	1	1	1	4
[60]	2005	1	1	1	1	4

Quality assessment of the mentation studies has been done with the aim of measuring the quality of the chosen studies. In studies answering the questions fully, the studies are registered 1. The studies that include data that solely investigate the issues and include no mitigation method in terms of quality evaluation queries are registered as 0.5. Although the study. Table 4 represent the Quality Evaluation of study.

The studies that are rejected are those with scores of 1 or lower. The research that failed to answer the questions of the quality assessment and the score is 1 or set below1. Those works have incorporated which cover the problem and remedial strategies regarding Architecture in SECO. The number is 58 included studies. Final data extraction is selected based on the contents of such studies that scored =< 2.

3.2.2.2. Data Extraction

In this portion, we responded to the Question of the research. We Study title, abstract, conclusion and issue and mitigation strategies of Architecture in SECO were identified and classified under various classes such as interface stability, security, reliability etc., that the data are the data that fulfilled the purpose of our research question. Read the title, abstract, etc., which were discussed.

3.2.2.3. Data Synthesis

The Architecture issues have been formed from existing 58 primary studies whose existence has been determined through SLR challenges and its mitigation. The data obtained in the primary studies were compared to the questions of the research.

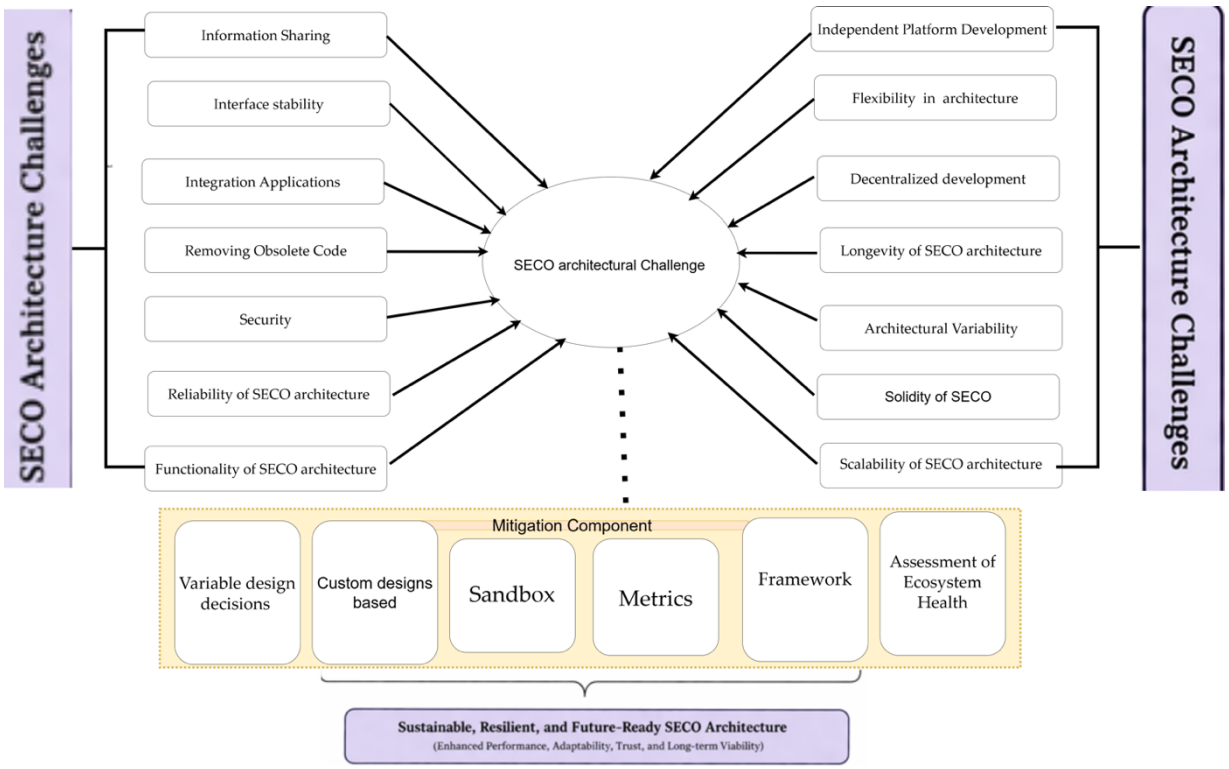

Figure 5. Conceptual framework diagram.

Table 5. Presented the SECO Architecture Issues and their Mitigation Strategies.

Type	Issue	References
Literature Review	Information Sharing	[4], [7], [8], [11], [14], [29], [30], [44]-[46]
Journal	Security, Reliability of SECO architecture,	[1]-[3], [5], [6]
	Functionality of SECO Architecture	[13], [18], [19], [26], [28], [35], [44], [46]
Case Study	Independent Platform Development	[8], [12], [14]-[17], [23], [33], [35], [40], [47]
Review	Flexibility in SECO Architecture	[7], [13], [16], [17], [26], [35], [41], [47]-[49]
Case Study	SECO Architectural Variability	[9], [12], [13], [24], [26], [27], [48]
journal	Removing Obsolete Code	[13], [42]
Case Study	Decentralized development	[25], [27]
Literature Review	Solidity of SECO Architecture	[18], [20], [25], [32]
Journal	Longevity of SECO architecture	[12], [13], [18], [34], [36]-[39], [50]-[54]
Case study	Interface stability	[18], [26], [36], [50]
Literature Review	Niche Creation	[7], [25], [55]
Literature Review	Integration Application	[4], [5], [7], [8], [19], [29], [43]-[46], [59]
Journal	Reliability of SECO architecture	[6], [31], [57], [58]
	Functionality of SECO Architecture	[13], [18], [26], [28], [35], [44], [56], [60]
Case Study	Platform Openness Dilemma, Independent	[8], [11]
	Platform Development	[12], [14]-[17], [23], [33], [35], [47]

3.2.3 Reporting the Review

Figure 4 illustrate the temporal distribution of the selected study from 2000 to 2026, showing a significant increase in publication after 2015, which indicating growing research interest in SECO Architecture. Apportionment of primary literature is represented in Figure 4.

3.3. Proposing conceptual framework

The proposed conceptual model has fourteen exogenous variables and six mitigation strategies. These variables have indirect effect on the SECO architecture, which is it is definitely a dependent variable. Mitigation strategies are as demonstrated in the conceptual diagram below, architectural issues define the independent variables and the dependent variables are marked by the architectural issues. The proposed conceptual framework is shown in Figure 5.

3.4. Answering the Question

SECO Architecture issue and their mitigation strategies ((RQ1, RQ2): This section offers the mitigation strategies as well as the challenges concerning the SECO Archi-

tecture that were elicited after undertaking a structured literature review. The SECO Architecture Issues have been displayed in Table 5.

3.5. Answer of the study Question through Data Extraction

3.5.1. (RQ3): SECO challenge

There are various benefits of SECO, like [4]: 1) Facilitates software coevolution and innovation success in the participating organization and enhances attraction to new participants. 2) Reduce the redesigning and relating costs. 3) support analysis and comprehension of software Architecture 4). Encourages collaboration and exchange of information between various independent software vendors.

The software ecosystem is also associated with numerous risks and challenges in addition to numerous benefits. These challenges or threats should be put more into consideration. Software ecosystem issues are further categorized using a number of areas, such as Architecture, Governance, Sustainability, Data Analytics, Dynamics and Evolution, and Domain-Specific Ecosystem [7].

3.5.2. (RQ4): SECO architecture challenges

In various studies, researchers have established that the architectural challenges facing SECO are the most important in its structural set-up [8]. According to R. Feitosa [9], software ecosystem design cannot be developed without factoring in social factors besides management (technical) processes and business laws. There should be communication between actors and the platform that sustains the ecosystem. A variety of software security architectural issues, such as the stability of the interface, security, reliability, scalability, longevity, solidity, integration application and other mitigation measures, such as Variable, Custom, Assessment, Metrics, Framework, Sandbox and others, have also been suggested.

Nevertheless, the studies have not discovered or examined how the different mitigation strategies could affect the architectural issues that will occur after the SLR. Because of this, it may prove challenging to establish the exact relation between the mitigation plans and architectural issues. Due to that, it is of extreme importance to trace all architectural issues. The proposed study will help seal this gap in the literature by systematically classifying all Architecture challenges and solutions and experimentally estimating the impact of these difficulties and remedies on the software ecosystem that SECO organizations offer.

3.6. Examining Recognized Obstacles and Methods of Industry and IT Experts.

The system of identified SECO (Security, interface stability, solidity etc.) SECO Architecture Difficulties and techniques when analyzed with the help of industry professionals and IT professionals may serve as a valuable source of information about how the organizations cope with serious problems in modern IT infrastructure. SECO is an integrated methodology which takes into account the following factors in designing and deploying IT systems: security, efficiency, cost-effectiveness, and operational factors.

4. Result and Findings

The findings indicate that SECO Architecture challenges are highly interrelated rather than independent. For example, interface instability directly affects integration efficiency, while security vulnerability often reduces system reliability. So it highlights the interdependencies SECO architecture challenge form a complex rather than isolated problem as well as mitigation strategies also demonstrating a many to many relationships between challenges and solution.

4.1. Results from SLR

This part provides the problems associated with software ecosystem Architecture and parameters influencing the problems, and the remedy approaches to software ecosystem architecture as identified with the use of SLR. The SECO Architecture challenges and their solution measures have been elaborated to answer Research Q1 and Research Q2. The issues of SECO Architecture, along with their challenges and solution strategies, were discovered in the study of literature review and SLR. These problems, together with their pertinent causes and their abatement measures, were used after reviewing the aforementioned references. Among 58 reused primary studies considered in the SLR, issues of SECO Architecture and their solution measures were find out in the mentioned this study.

The discussion of the chosen studies states that SECO arrows in architecture are very interdependent and not independent. As an example, integration problems have close connections with interface stability because unstable interfaces have a direct influence on component interoperability. On the same note, the security related problems and the reliability problems frequently accompany each other due to the similarity of dependency in the distributed environment. Moreover, mitigation techniques, including design in modules, API standardization, and sandboxing, deal with several issues at the same time. This means that the mitigation measures are systemic and not one-dimensional problems. In such a way, it is possible to note that the results reveal that there are many-to-many relationships between challenges and mitigation strategies and complexity of SECO architecture. The SECO Architecture challenges has been shown along with their relative frequency (how much each individual was given score in the survey) in Table 6.

4.1.1. Information Sharing

Means that a software Ecosystem architecture can support growing performance, capacity and resource requirements within the system. Software ecosystem archi-

Table 6. SECO Architecture Challenges.

Factor	Frequency
Information Sharing	26%
Interface stability	43%
Integration Applications	29%
Removing Obsolete Code	25%
Security	57%
Reliability of SECO architecture	53%
Functionality of SECO architecture	32%
Independent Platform Development	31%
Flexibility in architecture	49%
Decentralized development	34%
Longevity of SECO architecture	43%
Architectural Variability	36%
Solidity of SECO architecture	36%
Scalability of SECO architecture	66%

tecture implies that a system is able to accommodate growing performance, capacity, and resource demand. On the one hand, sharing of the required information is an inherent feature of software ecosystem design since it facilitates the communication and coordination of the software elements and interests to be instigated respectively.

Nevertheless, there are many pitfalls which can be related with the accomplishment of successful exchange of information. The interoperability of various software applications and systems is one of the challenges. Various platforms and programming languages and data formats may also be contained within a software ecosystem therefore complicating the maintenance of the communications in a flowing format. This challenge can be reduced by standardization of protocols, APIs and data formats.

Another important issue is the data security and privacy. The fact that most of the elements might have access to sensitive information results in the possibility of tampering or gaining access to information without the required authorization. Thus, information protection in terms of strong authentication, encryption and authorization is required to meet data protection rules. The policies on data governance also come in handy with regard to regulating access and usage of data in the ecosystem.

Other than this there is the heightened complexity and volume of information with the increase in size of the ecosystem. It is possible to use distributed databases, data lakes, and data warehouses to process giant data. The more sophisticated functions of the system could be implemented by the even higher existing advanced technologies such as artificial intelligence and machine learning because they could process and analyze data as well as make decisions automatically.

4.1.2. Interface Stability

Any alteration in the current software ecosystem structure has no impact on the capability of the other component to interface with it. One of the major issues of software ecosystem architecture is the stability of the interface. An interface is a connection between different software components and includes the way in which they will interact with each other and how they will exchange data. But with the evolving ecosystem, where the elements of the ecosystem are upgraded and revised, the stability of the interfaces will be harder to sustain. Backward compatibility is important for the introduction of new features or making changes in order to avoid delays and ensure seamless integration. Enhancing interdependence among components requires scalability, thoughtful control, and effective communication to the extended degree needed to maintain interoperability of the related versions to avoid disrupting existing integrations. Dependence is one issue that influences interface stability.

Components often rely on external libraries, frameworks, or services, and this often creates a complex web of dependencies that results in a software ecosystem. Alterations to a component may affect the stability of the interfaces on which a dependent component relies. In order to avoid compatibility issues caused by problems and ensure the interface stability, it becomes necessary to handle dependencies and ensure that various versions are compatible. The solution to this issue is provided by the help of effective dependency management methods such as package managers, version pinning, and automated testing. Furthermore, a separate set of governance and communication systems within the software ecosystem should be developed to maintain stability in the interface. The high-quality change management procedures, versioning rules, and feedback can ensure the creation of a high-quality interface that is stable across the software ecosystem architecture through effective coordination and decision-making.

4.1.3. Integration Applications

In order to accomplish a successful software environment, software architectures provide the relationship between different applications and systems with the SECO. The process that determines interactions between systems and applications is called software ecosystem architecture. Another valuable component is integration that enables the data sharing, communication and coordinated operations of the various software components:

1) Interoperability: Interoperability is one of the objectives of software ecosystem architecture. It assures that various components are able to communicate and meet with the rest. This is by including common interfaces, APIs and common data formats.
2) Data Integration: Service-Oriented architecture (SOA) aids in the integration by highly isolating functionality in easy to reuse autonomous services

across applications. The mobile API integration is also another significant variable because it enables the units to communicate, and API gateways enable it to be managed, tracked, and secured. In micro services architecture, the applications are divided into small discrete services and they interact with the help of APIs. This makes it easy to incorporate, scale, integrate and update without impacting on the entire system.

3) Event-Driven Architecture (EDA): EDA offers real time interaction of components. It allows systems to respond dynamically to events and the demarcation barriers that exist between the different components of the ecosystem to be ambiguous.
4) Middleware: Middleware is made up of integration platforms and message brokers which support communication between various systems. They facilitate sharing of data and achieve coordination of activities in the ecosystem.

4.1.4. Removing Obsolete Code

An obsolete code is software code that is no longer required or of use in the present software ecosystem. An essential part of maintaining a modern and functioning ecosystem is the elimination of problems regarding the Architecture of old software ecosystems. The deployment of old parts, technology or processes may slow down the development, increase maintenance expenses and lead to compatibility issues. One of the challenges of disposing of the old parts is legacy system integration. Most software ecosystems already have long-running setups of legacy systems that are hard to upgrade. Such systems often have complicated dependencies and are not compatible with newer technologies. A rigorous migration strategy involving partial decommissioning and compatibility layers/middleware can solve this problem and phase out old components as disruptively as possible.

Precautions of backward compatibility and the minimization of the effect on existing users are other challenges. The interfaces, APIs, and data formats need to be modified to eliminate outdated software ecosystem elements. To ensure that customers are not left waiting, it is thus important that current users are adequately transferred to avoid delays in the process. It is easy to accommodate the change and decrease the possible problems through proper documentation, effective communication and support throughout the migration process. The organizational opposition and inertia may also serve as a hindrance to the removal of outdated software ecosystem design. The people who are accustomed to the existing system and practices will not be open to change.

To address this resistance, it will need effective change management plans such as benefits that are well communicated, participation by key stakeholders in decision process, and training procedures that help the users become familiar with new technology and practices. To overcome organizational resistance and facilitate the removal of the outdated elements as the elements of the software ecosystem architecture, encouraging the culture of innovation and agility might also prove helpful.

4.1.5. Security of SECO Architecture

This includes the protection of the hardware, software, networks, data and the individuals who are operating in the SECO. Since many of the elements are related, and the potential risks of data breaches, unauthorized access, and malicious attacks may be high, security is one of the important concerns when developing software ecosystem architecture. To be able to deliver credible security in the SECO, several issues shall be resolved. It is a problem in itself, the complexity of the ecosystem. The numerous interdependent elements within the ecosystem mean that weaknesses in a given element can influence the other system. Strict security testing, vulnerability testing, and penetration testing are essential to detect and resolve any defects and security vulnerabilities. SECO, the software ecosystem, may not be able to ensure security in most of the technologies, platforms, and programming languages. Every technology may possess distinct security weaknesses and mitigation. Ensuring that security measures are applied uniformly and keeping up with updates on security as they arise can be challenging and time-consuming to operate within the system.

The solution to this issue can be provided through the introduction of security frameworks, rules, and regulations that ensure a range of technologies and the enforcement of compliance. Moreover, it is important to ensure that data security and communication are guaranteed in the software ecosystem. Components often handle sensitive data and information, and it is based on this consideration that questions arise of privacy, confidentiality and integrity. Sensitive data should be safeguarded by means of access control, secure communication channels, and powerful encryption algorithms used throughout the transfer or storage of the same.

4.1.6. Architecture Reliability

Means the capability of software Ecosystem architecture to be reliable and dependable in performance over time. Reliability is another significant issue in the architecture of a software ecosystem since it directly affects its user experience and overall functionality. A good software ecosystem must surmount several challenges. To begin with, it may not be easy to maintain compatibility between different components, as well as control dependencies. Ensuring that the new parts adequately interface with the old ones is getting more and more essential as the ecosystem grows and additional components are introduced. In this regard, version control, comprehensive compatibility testing, and dependency documentation are important to

maintain reliability and prevent compatibility issues which may lead to system failures. A software ecosystem architecture, as well as SECO, must be fault-tolerant and resilient.

The elements that make up the ecosystem can either malfunction or face downtimes due to various reasons, such as the crash of hardware, network issues, or bugs in the software. Redundancy strategies, which include backup systems, load balancing and failover processes, are implemented to help ensure constant availability as well as mitigate the impact of failures. Moreover, it is possible to establish effective monitoring and alerting mechanisms that will help to detect issues at a very fast rate and rectify them, which leads to a higher level of reliability. Besides, the software ecosystem Architecture encounters problems when it comes to data consistency and integrity.

The importance of integrity and consistency of data increases with its transfer to different components. Inconsistent data may lead to data corruption, errors, and inaccurate outcomes. Progressing holistic data validation strategies, the use of transactional processing, and the use of data synchronization measures would assist in guaranteeing data integrity and uniformity throughout the ecosystem.

4.1.7. Software Ecosystem Architecture Functionality

Software ecosystem architecture functionality is the set of features, capabilities, and behaviors that a software system has been programmed to handle. Software ecosystem Architecture should pay attention to functionality since it influences the abilities and performance of the ecosystem. Nonetheless, several challenges could be encountered when deploying and maintaining a common optimum functionality. The first one is that the integration of numerous elements with many functionalities may be challenging. All components may have a different set of functionality, APIs and data formats. In order to provide simple communication and exchange of data, smoothly connecting and integrating components with varying functionality requires proper planning, threshold standards, and high volumes of testing.

It is challenging to coping with the functional complexity of the ecosystem, SECO. As the size of the ecosystem increases, so does the number of components included in the ecosystem and their functionality, which forms a more complex system. The interactions, dependence, and behavior of various components may not be easy to understand and control. The concepts of abstraction, encapsulation and modular design can help in lessening the system complexity and hence make it easier to manage and maintain. Efficient record keeping and knowledge sharing process also enable developers and users to understand the features and abilities of different components of the ecosystem.

Also, a software ecosystem design should ensure the scalability and adaptability of the functionality. The functionality should have the potential to ensure its expansion and adaptation in case of a change in the ecosystem and the changing needs of the user. It is possible to support future upgrades and changes with the help of designing components and APIs that are flexible, scalable, and easily changeable. Effective resource allocation can be achieved by the implementation of scalable infrastructure, including cloud computing and containerization, which enables feature requirements to adapt to the changing nature.

4.1.8. Independent Platform Development

It is a programming feature meaning whether a platform is reliant on any certain operating system or hardware platform or not. Platform independence is a major challenge because software ecosystem Architecture attempts to provide the ability to have software components work well in a large number of platforms, operating systems and devices. Platform independence involves the challenges that are broken. To begin with, managing platform-specific dependencies may be difficult. Components might have to comply with special specifications, APIs or platform software frameworks. Abstraction layers, platform-neutral APIs, and platform-neutral components can be used to reduce platform dependencies and enhance between-platform information portability.

SECO, and sometimes it is hard to ensure consistent performance and behavior on different platforms. The platforms can consist of different hardware constraints, resource constraints, or performance properties. This is essential in order to optimize components to cross platform performance and also adapt components behavior based on the capabilities of the underlying platform. This issue can be addressed and coherent conduct and execution in numerous platforms through measures such as performance profiling, resource management and platform-specific optimizations. Also, having platform compatibility and upgrades is fundamental to platform independence. Platforms change with time; new releases might introduce changes or the outgrowth of certain functionalities. Platform independence is ensured by making sure that software components are made compatible with new platform versions, by conducting consistency testing on a regular basis, and by tracking platform upgrades. Best methodologies to address this issue are to have version control, adopt agile methodologies in development and remain up to date with the upgrades of a platform by developer networks and forums.

4.1.9. Flexibility of SECO Architecture

Means the capability of a software Ecosystem architecture to change and evolve in response to external requirements of the SECO. With SECO Architecture attempting to make sure that software components can work

efficiently, regardless of numerous platforms, operating systems, and devices, platform independence is one of the challenges. Platform independence comes with breaking an array of challenges. To begin to handle platform-specific dependencies may be a hard task. The components might be required to conform to distinct platform requirements, APIs, or platform software systems. Platform Non-portable application components can be minimized and cross-context portability enhanced through abstraction layers, platform-neutral APIs and platform-independent components.

SECO, it may be challenging to ensure that there is uniform behavior and performance across platforms. Platforms can be known to have different hardware or resource constraints, or performance properties. It is important to make the components as cross-platform as possible and change the behavior depending on the features provided by the underlying platform. The solution to this issue may be the strategies such as performance profiling, resource management, and platform-specific optimizations, which can resolve this issue and ensure that the behavior and performance of various platforms are similar. Also, platform independence requires platform maintenance and updates.

Platforms change with time, and releases can carry with them changes or stagnate certain features. Platform Independence is ensured by keeping software components compatible with new versions of the platform, carrying out frequent compatibility tests, and keeping track of platform upgrades. Some of the best strategies to overcome this issue are the utilization of version control and the application of agile development techniques and staying at par with platforms by joining developer networks and forums.

4.1.10. Decentralized Development SECO Architecture

Where various development teams are engaged in working independently on various sections of the SECO Architecture. The Structure of the SECO Architecture has a barrier with regard to the decentralized development since it requires the integration of the activities of many teams or contributors who can be geographically dispersed or work individually. To ensure effective cooperation and coordination, several concerns that this strategy has brought about should be addressed. First, it is vital to make sure that there are similar standards and directives in the decentralized teams. A team can have development procedures, coding fashions and quality control procedures of its own. Unity can be ensured, and the interaction between different elements in the ecosystem can also be ensured through the creation of clear principles, conducting regular code reviews, and implementing continuous integration processes.

Cooperation and communication are complicated by a decentralized development paradigm of SECO. The communication way, work hours, or location can be different and in various time zones in cases of teams. Overcoming communication problems requires proper collaboration tools, frequent meetings or video conferences, and formulated communication norms. To ensure the effective coordination of the work of decentralized teams, the culture of openness, promotion of knowledge exchange, and fostering the feeling of teamwork are also essential. Moreover, it is important to have the efficient version control and change management within a decentralized development architecture.

When the various teams are handling different parts or modules concurrently, there is need to have versions of the code, conflicts to be resolved and changes appropriately traced. To address this problem and administer and introduce changes in the Architecture of the software ecosystem, auto-testing and deployment processes, branching, and merging methods, and version management systems may be proposed.

4.1.11. Longevity Software Ecosystem Architecture

An ecosystem that remains pertinent and practical over an extended duration is called longevity of the SECO. Longevity is a major setback to the Architecture of the software ecosystem since it is expected that the software systems should last long, yet technology is rapidly changing. One of the challenges is to be compatible with new platforms and technologies. Systems software, the software ecosystem may grow out of date or out of support as underlying technologies, programming languages, and platforms become outdated. To ensure that the ecosystem is compatible with the new technologies and meets the market evolution, periodical upgrades, restructuring, and migration techniques are needed. Backward compatibility is yet another challenge, and new features and upgrades must be added to the current product. More parts or versions can be added during the development of the ecosystem to enhance functionality or fix security vulnerabilities. Nonetheless, sometimes, it may be challenging to ensure that the upgrades do not interfere with integrations that are already in place or create compatibility issues.

This obstacle can be minimized and the continuity of user can be provided on the foundation of the current ecosystem that is being supported by version systems, recording the changes, and offering the migration paths or compatibility layers. The technological debt and legacy system is the long term challenge of SECO Architecture. Technical debt can accumulate in older systems, and it is harder to maintain and update them. The legacy systems may be either not well documented, use an old codebase, or use outdated technology. To ensure that the old systems are variable and consistent with emerging demands and technologies, very often, handling them requires some planning, refactoring, and embracing modernization efforts.

4.1.12. Architectural Variability

The capability of software ecosystem to offer several architectural solutions to a specific problem or group of problems. Software ecosystem architecture is challenging because it has to fit a range of architectural ideas, design paradigms, and implementation strategies. These challenges involved in managing the compatibility and the complexity of different architectural variations. Ensuring compatibility and the smooth integration may be tricky since every single component of the ecosystem may possess its own architectural needs and constraints. The solution to this issue is possible through ensuring uniformity and interoperability across the ecosystem through the application of architectural standards, all-around-the-clock tests of compatibility, and specifying clear and focal rules of architecture choices.

Another challenge is the continuation of documentation and sharing of knowledge concerning the different architectural variations. In cases where the architectural variations are numerous, there is need to have the right documentation that defines the justification, design principles and best practices that are associated to each architectural variation. Moreover, the ecosystem can also better understand and accommodate a wide variety of architectural variants through promoting the exchange of knowledge among developers, running frequent training sessions, and establishing communities of practice. Also, software ecosystem Architecture can be hard to ensure its extensibility and scalability due to architectural diversity. The surrounding conditions must be adaptable to accept the change of existing buildings or the outbreak of various architectural types in future. This can be addressed through extensible architectural designs, scalable infrastructures and modular design principles. The ecosystem can also expand and adapt to the changing demands and technology by fostering a culture of change and continuous improvement of the ecosystem. This helps in the investigation of new architectural strategies.

4.1.13. Solidity of SECO Architecture

Means that a Software Ecosystem Architecture or structure is strong, stable, and durable. This is a computer code that is specifically designed to be used to write smart contracts on block chain platforms. Although Solidity offers major problems in terms of the Architecture of software ecosystems, it has its own peculiar opportunities to construct decentralized apps. To begin with, the issue of security is of utmost priority when working with Solidity. The need for writing secure smart contracts requires an understanding of block chain security attacks, such as integer overflow/underflow attacks and reentrancy attacks. Another requirement is to audit and identify any possible weaknesses in the Solidity code that might create the risk of contract hacking or losing money. To successfully cope with these challenges, it is essential to perform critical security audits, utilize security analysis tools and follow best practices in secure coding.

SECO, which collaborates with Solidity, can enhance interoperability. The gap between different platforms needs to be closed with the help of cross-chain technology, interoperability standards, and clearly defined APIs to allow free integration. It is a problem that can be addressed, and the integration of Solidity-based systems with other ecosystems can be enhanced with the help of interoperability frameworks, such as Polka dot or Cosmos and following the established contract interfaces. Besides, Solidity is dynamic, which complicates versioning and backward compatibility problems.

The modifications, new features, and discontinuities to Solidity are implemented with new versions. This may complicate the operations of already operational smart contracts, particularly with new Solidity versions. The need to ensure backward compatibility and provide straightforward migration paths for the existing contracts emerges to ensure that the existing systems do not interrupt and the ecosystem continues to work as intended.

4.1.14. Scalability of SECO Architecture

Meaning Refers to the capacity of a software Ecosystem architecture to withstand growing requirements in terms of the performance, ability, and resources of the system. Software ecosystem Architecture has to consider scalability because it cannot withstand high workloads, grow with more users, and meet new demands. The issue of acquiring scalability, however, poses several problems which need to be addressed. To begin with, it may be difficult to keep the scalability of individual ecosystem components under control. Some of these elements can become bottlenecks as the system scales, forcing the overall functionality of the ecosystem to scale. These components can be identified and optimized, caching approaches can be implemented and load balancing or distributed computing can be employed to eliminate component level scalability problems.

SECO and inter-component connections and interactions should be attended to ensure that the system is scaled in general. The intricacy of information transmission and communication among the components increases with the maturing of the ecosystem. It will be crucial to ensure that the relations scale effectively. In order to eliminate this problem and enhance the overall scalability of the ecosystem, one can use loosely linked components, asynchronous messaging systems, as well as scalable data storage and retrieval mechanisms.

Another challenge towards scalability is the good management of resources. More commonly, additional computational resources such as servers, storage, and network infrastructure are packed in order to scale up the eco-

Table 7. SECO Architecture Issues Solution Strategies.

Factor	Frequency
Variable design decisions	41%
Custom designs based	37%
Assessment of ecosystem Health	40%
Metrics	33%
Framework	46%
Sandbox	32%

system. Nevertheless, to avoid wastage and ensure optimal performance, resource allocation, monitoring and utilization are highly required.

4.2. Mitigation Results from SLR

While globally developing, SECO development is a very important and basic factor. The software ecosystem issue has a sub factor, SECO Architecture issue, which influences the development of SECO across the world. This has been overcome by different mitigation strategies [18]-[20]. Factor and their frequency (How much score was given by the expert in the survey that is suitable solution or not) as given. Table 7 provides mitigation strategies that reduce the issues of the software ecosystem Architecture.

4.2.1. SECO Architecture Mitigation Strategies.

There are numerous architecture challenges mitigation strategies well which are discussed in detailed bellow.

4.2.1.1. Variable Design Decisions

The variability and flexibility of the design choices undertaken regarding different parts of the ecosystem are what are defined as variable design choices in the architecture of the software ecosystem. Even though variability has two positive characteristics in the form of adaptability and customization, there exist disadvantages. These challenges necessitate mitigation strategies to adequately deal with them. Originally, flexible design decisions are manageable by being simplified through standardization and the division of design into modules. The uniformity and compatibility of the ecosystem is encouraged by the composition of common design patterns, interfaces and coding methods between the components. This ensures that integration is made easier, reduces complexity and simplifies maintenance. Moreover, by adopting a modular approach, functionality can be encapsulated, and concerns can be separated, which simplifies their management and the impetus of the particular components without disturbing the entire ecosystem.

The key to reducing issues associated with the design selection of variables lies in SECO and in-depth documentation and information sharing. In the case of various components, design decision documentation, Architecture decision documentation and dependencies are provided to provide context and clarity to the development team of the various components. Data and design justification can also be shared effectively, less uncertainty created, and group communication encouraged by means of collaborative technologies or by building a central knowledge base. It is also possible to use regular code reviews and technical presentations as the means of striking the right balance in terms of design choices and reducing inconsistencies. Also, to deal with the problems caused by wavering design decisions, it is critical to design effective channels of communication and collaboration.

Communication simplicity between stakeholders, architects and development teams allows communication and consultation decision making. Through frequent meetings, arguments, contestations and feedback, design decisions can be elucidated, potential conflicts can be identified, and issues tackled. Good collaboration systems and tools offer real-time communication, version control, and the ability to trace design decisions, which can be used to mitigate the challenges caused by an assortment of design choices.

4.2.1.2. Custom Designs Based

Mitigation technique-based custom designs are vital in addressing the challenges in the architecture of the software ecosystem. Custom designs may involve development of standardized protocols, APIs and data format that are particularly tailored to the needs of the ecosystem to resolve interoperability problems. The problems of interoperability can be easily addressed through the creation of a specific design that can foster simplified communication and integration. In line with this, it is possible to enhance security through the application of unusual designs in the context of the software ecosystem. This could involve developing special access privileges, encryption procedures, and authentication procedures that comply with the specific security requirements of the ecosystem. Weaknesses may be addressed with the aid of custom security designs to help protect sensitive information, defend against dangers of unauthorized access, or data breaches. Scalability issues in the software ecosystem architecture can also be addressed using custom designs. This can include coming up with distinctive architectural designs that aid allocation of resources that are horizontal and can scale up and down without much difficulty, like with micro services or containerization. In order to make sure that the system will be able to cope with the growing workloads and accommodate growing user numbers, one can also consider such custom design solutions as caching strategies, load balance strategies, and scalable data storage solutions.

4.2.1.3. Assessment of Ecosystem Health

To critically detect and overcome the hurdles facing the software ecosystem architecture, there is need to take into consideration the wellbeing of a software ecosystem. Tactics can be employed to test and sustain the well-being

of a software ecosystem by taking several approaches. To begin with, periodical examinations and evaluations of the components of the ecosystem, its dependencies and interfaces may help find possible weak points, snags in performance or can be used to find outdated technology. By periodically assessing the health of the ecosystem, the stakeholders will be able to correct any shortcomings proactively, equip themselves with the necessary changes and replacements and ensure the ecosystem as a whole is healthy.

Secondly, proper monitoring and logging mechanisms should also be established to ensure an order of keeping track of the functionality and performance of the ecosystem. Monitoring tools may provide insight into the use of resources, response times, the frequency of errors, and other significant variables, leading to identifying issues and providing their resolution within a short time. Logging helps collect the relevant data about incidents, errors, and user interaction that makes troubleshooting and analysis easier. By implementing large-scale monitoring and logging methods, the stakeholders can gain access to the health of the ecosystem and take corrective measures to maintain optimal performance and manage any potential concerns.

Moreover, the culture of cooperation and constant evolution should be developed in order to overcome the challenges during the design of the software ecosystem. Opportunities to enhance the situation and address any emerging concerns can be identified more easily if developers, system architects, and end users are made to communicate freely and give feedback. Regular meetings, retrospectives, and cooperative problem-solving sessions should improve the health of the ecosystem because these activities will enable stakeholders to exchange ideas, discuss possible solutions, and collaborate. Software ecosystems can be effective in dealing with the issues, neo-adjusting to changing needs, and staying long-term healthy and sustainable by developing a culture that emphasizes learning, creativity, and collaboration.

4.2.1.4. Metrics

Measures, which provide impartial measurements and insights that can be used to make decisions and help monitor progress, are important in lessening the problems of software ecosystem design. The following are some of the ways metrics can be used to address these problems:

a) Compatibility and interoperability: It is possible to measure the compatibility and interoperability between different components of an ecosystem using metrics. By monitoring the number of successful integrations, compatibility issues, or failed interactions, teams can see where they can develop and set priorities on enhancing the level of interoperability. Also, it is possible to observe compatibility measures during component update or change over to ensure backward compatibility and minimize interrupts.
b) Security and Dependability: The security and stability of the ecosystem can be assessed with the help of metrics. Tracking measures on security breaches, systems malfunctions, and vulnerabilities could help identify possible weaknesses and areas to work on. The uptime, reaction times, or error rates can be monitored and information given regarding the reliability and performance of the ecosystem. These metrics can be applied to guide the use of security measures, fault tolerance methods, and performance improvement to enhance security and dependability. Scalability and flexibility.
c) Adaptability and Scalability: Measures can be used to determine the adaptability and scalability of the ecosystem. Information on the possibility of scaling and areas that require scaling efforts can be provided in response times, re-source usage, or capacity measurements. One can measure the adaptability of the ecosystem by monitoring metrics that reflect the ease with which the different components can be revised to include new functionality or be reconfigured according to the demands. These measurements can be used to make an informed decision about scaling strategies, architecture change, or technological change to achieve greater flexibility and scalability.

4.2.1.5. Framework

Frameworks are also important in reducing the ills of the software ecosystem architecture by providing a systematic and standardized way of development and integration. The following are some of the strategies structures can apply in dealing with these problems: Interoperability and standard-based practices: Structures often provide a set of pre-defined interfaces, protocols, and standards that facilitate communication between different parts of the ecosystem. By meeting these standards, developers can guarantee compatibility and smooth integration, as well as reduce the complexity of integrating dissimilar platforms and technologies.

Frameworks often include practices, standards, as well as design concepts that facilitate solutions to the common problems and promote ethical practices in SECO Architecture. The recommendations will help programmers to escape the hassle that could arise on a regular basis, ensure good code, and improve the reliability and maintainability of the software ecosystem architecture. They provide options of load balancing, horizontal scaling, and variable configuration techniques and thus are easier to adapt the ecosystem to the evolving demands and to handle increasing workloads.

Structures that promote extensiveness and modularity render it easy to add new features or components. Community and support: Structures often have lively communities with support forums where the developers can ask questions, educate themselves and cooperate with other developers to resolve issues. They provide troubleshooting forums, documentation and helpful tools, and these communities have shown to be of assistance to the developer upon breaking barriers and fixing architecture problems.

4.2.1.6. Sandbox

Sandboxing is an isolation technique which enables software components to execute in a restricted setting without damaging the rest of the software ecosystem. It is commonly applied as a part of mitigation measures to overcome the complexity in architecture, security threats, and compatibility issues. With the implementation of a new or changed part within a sandbox, the developers have an opportunity to measure its behavior, functionality, performance, and how it will integrate with other modules prior to deployment. This isolation is beneficial in the scan of defects, conflicts, and unforeseen interactions in the early stage, thus safeguarding the health of the live ecosystem. Security-wise, sandboxing restricts the effect of the potentially dangerous or vulnerable programs since any type of malicious programming can be kept in the isolated system, and thus it cannot extend throughout the system. Also, the sandbox environments allow version control and dependency management because multiple component versions can be tested separately.

This makes sure that the updates, changes, or new release will not interfere with the currently available functions in the ecosystem. Sandboxing enhances the architectural robustness via measured experimentation, risk management, and compatibility testing, and promotes the sustainable development of ecosystems of software applications.

5. Conclusion and Future Work

Most software organizations have been carrying out their development practices in the SECO environment in recent years. The existing literature has shown that the problem of Architecture is among the critical problems in SECO, and it is growing increasingly complex as time progresses. The current imminent growth in SECO motivated us to define the architecture issues that are having a great influence on the SECO environment. Hence, the SLR has been conducted in a bid to determine the factors that influence Architecture in SECO and mitigation and the conceptual framework has been proposed.

The identified factors and mitigation based on the results of SLR fall under respective categories, which include Interface stability, Integration Applications Removing Obsolete Code, Security, Reliability of SECO architecture, Functionality of SECO architecture, Independent Platform Development, Flexibility in architecture, Longevity of SECO architecture, Architectural Variability, Solidity of SECO architecture, and mitigation of the factors. The future direction of work will be to rank these challenges and alleviate them to the Numpy or ANP algorithm to find which challenge is the most critical and which challenge is the most effective mitigation of the challenges.

6. Declarations

6.1. Author Contributions

Inayat Ur Rahman: Conceptualization, Research Design, Methodology Development, Systematic Literature Search, Study Selection, Data Extraction, Data Synthesis, Visualization, and Writing – Original Draft Preparation; **Atta Ur Rahman:** Formal Analysis, Quality Assessment of Included Studies, Validation, Thematic Classification, Critical Review; **Sara Shazad:** Supervision, Project Administration, Strategic Guidance, Critical Feedback, and Final Approval of the Manuscript; **Sajid Ur Rahman:** Review & Editing, Proofreading, and Final Manuscript Refinement.

6.2. Institutional Review Board Statement

Not applicable.

6.3. Informed Consent Statement

Not applicable.

6.4. Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request. This study is based on a systematic review of previously published literature, and no new experimental or primary data were generated.

6.5. Acknowledgment

Not applicable.

6.6. Conflicts of Interest

The author declares no conflicts of interest.

7. References

[1] R. Rajab and M. Alnoukari, "DevOps integration with capability model maturity integration: A systematic mapping review," *IEEE Access*, vol. 13, pp. 31829- 31841, 2025. https://doi.org/10.1109/ACCESS.2025.3542630.

[2] Y. G. Soureya *et al*, "Adaptive software development: A comprehensive framework integrating artificial intelligence for sustainable evolution," *International Arab Journal of Information Technology*, vol. 22, no. 2, pp. 248–262, 2025. https://doi.org/10.34028/iajit/22/2/4.

[3] E. Onagh and M. Nayebi, "Extension decisions in open source software ecosystem," *Journal of Systems and Software*, vol. 212, p. 112552, 2025. https://doi.org/10.1016/j.jss.2025.112552.

[4] K. Saarni, M. Raatikainen, and S. Hyrynsalmi, "Failures in software ecosystems: A systematic literature review," in *Proceedings of the 2025 IEEE International Conference on Engineering, Technology, and Innovation*, 2025. https://doi.org/10.1109/ICE/ITMC65658.2025.11106539.

[5] A. Raj and R. Deora, "AI and ML powered feature prioritization in software product development," *International Journal of Data Mining & Knowledge Management Process*, vol. 15, no. 1, pp. 23–30, 2025. https://doi.org/10.5121/ijdkp.2025.15102.

[6] S. Hegazy *et al*, "Overcoming experimentation challenges in software ecosystems of large product and service organizations: A participatory action research study," *Journal of Systems and Software*, vol. 230, p. 112550, 2025, https://doi.org/10.1016/j.jss.2025.112550.

[7] J. Sun *et al*, "Collaboration challenges and opportunities in developing scientific open-source software ecosystem: A case study on Astropy," *Proceedings of the ACM on Human-Computer Interaction*, vol. 9, no. 7, pp. 1–33, 2025. https://doi.org/10.1145/3757462.

[8] Z. Yang, J. Shi, P. Devanbu, and David Lo, "Ecosystem of large language models for code," *ACM Transactions on Software Engineering and Methodology*, vol. 35, no. 1, pp. 1–30, 2025. https://doi.org/10.1145/1234567.1234569.

[9] R. Feitosa *et al*, "Actionable Framework for Understanding and Improving Social and Human Factors that Influence the Requirements Management in Software Ecosystems," *ACM Transactions on Software Engineering and Methodology*, 2026. https://doi.org/10.1145/3795773.

[10] A. Katal, P. Prasanna, R. Birla, and Kunal, "Evolution from monolithic to microservices architecture: A new era in software architecture," in *Advancements in Optimization and Nature-Inspired Computing for Solutions in Contemporary Engineering Challenges*, Springer, 2025, pp. 235–279. https://doi.org/10.1007/978-981-96-0706-8_12.

[11] O. M. Albada and N. Salman, "Software architecture in practice: Challenges and opportunities in the age of digital transformation: Literature review," 2024. https://www.researchgate.net/publication/381291461.

[12] S. S. Amorim, E. S. De Almeida, and J. D. McGregor, "Extensibility in ecosystem architectures: An initial study," in *Proceedings of the 8th International Workshop on Software Ecosystems*, 2013, pp. 11–15. https://doi.org/10.1145/2501585.2501588.

[13] A. Ghazawneh and O. Henfridsson, "Balancing platform control and external contribution in third-party development: the boundary resources model," *Information Systems Journal*, vol. 23, no. 2, pp. 173–192, 2013. https://doi.org/10.1111/j.1365-2575.2012.00406.x.

[14] B. Ghimire, Z. S. Li, and D. Damian, "Understanding user feedback in software ecosystems: A study on challenges and mitigation strategies," in *International Conference on Software Business,* pp 132–147, 2024. https://doi.org/10.1007/978-3-031-53227-6_10.

[15] K. Henttonen, M. Matinlassi, E. Niemelä, and T. Kanstren, "Integrability and extensibility evaluation from software architectural models: A case study," *Open Software Engineering Journal*, vol. 1, no. 1, pp. 1–20, 2007. https://doi.org/10.2174/1874107X00701010001.

[16] E. Constantinou and I. Stamelos, "Architectural stability and evolution measurement for software reuse," in *Proceedings of the ACM Symposium on Applied Computing*, 2015, pp. 1580–1585. https://doi.org/10.1145/2695664.2695895.

[17] M. Alenezi, "Software architecture quality measurement: Stability and understandability," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 7, pp. 292–299, 2016. https://doi.org/10.14569/ijacsa.2016.070775.

[18] C. F. Alves, J. A. Oliveira, and S. Jansen, "Understanding governance mechanisms and health in software ecosystems: A systematic literature review," in *Proceedings of the International Conference on Enterprise Information Systems*, 2017. https://doi.org/10.1007/978-3-319-93375-7_24.

[19] V. M. Melnuk, K. V. Melnuk, and N. A. Khrystynets, "Key performance indicators based software ecosystem (SECO) research using publications systematic mapping," *Komp'iuterno-intehrovani tekhnolohii: osvita, nauka, vyrobnytstvo*, no. 20, pp. 57–66, 2015.

[20] M. Galster, P. Avgeriou, D. Weyns, and T. Männistö, "Variability in software architecture: current practice and challenges," *ACM SIGSOFT Software Engineering Notes*, vol. 36, no. 5, pp. 30–32, 2011. https://doi.org/10.1145/2020976.2020978.

[21] B. Kitchenham and P. Brereton, "A systematic review of systematic review process research in software engineering," *Information and Software Technology*, vol. 55, no. 12, pp. 2049–2075, 2013. https://doi.org/10.1016/j.infsof.2013.07.010.

[22] D. Namiot, M. Sneps-Sneppe, "On micro-services architecture," *International Journal of Open Information Technologies*, vol. 2, no. 9, pp. 24–29, 2014. http://injoit.ru/index.php/j1/article/view/139/104.

[23] V. Dixit and D. Kaur, "A systematic review for sustainable software development practice and paradigm," *Journal of Computational Analysis & Applications*, vol. 33, no. 6, pp. 170–185, 2024. https://eudoxuspress.com/index.php/pub/article/view/730.

[24] S. S. Amorim *et al*, "Flexibility in ecosystem architectures," in *Proceedings of the 2014 European Conference on Software Architecture Workshops*, 2014. https://doi.org/10.1145/2642803.2642817.

[25] I. Carvalho *et al*, "HEAL ME: An architecture for health software ecosystem evaluation," in *Proceedings of the 2017 IEEE/ACM Joint 5th International Workshop on Software Engineering for Systems-of-Systems and 11th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (JSOS)*, 2017. https://doi.org/10.1109/JSOS.2017.13.

[26] B. Schwichtenberg and G. Engels, "SECOarc: A framework for architecting healthy software ecosystems," *Communications in Computer and Information Science*, vol. 1269, pp. 95–106, 2020. https://doi.org/10.1007/978-3-030-59155-7_8.

[27] A. Abdalla, V. Ströele, F. Campos, R. Braga, J. M. N. David, "A software ecosystem platform for the development of recommender systems," *Research Square preprint*, 2020. https://doi.org/10.21203/rs.3.rs-34335/v1.

[28] E. Manalif, L. F. Capretz, and D. Ho, "Software ecosystems risks," in *Proceedings of the 8th International Joint Conference on Software Technologies (ICSOFT)*, 2013, pp. 417–422. https://www.academia.edu/download/32528954/Ekananta-Iceland-v2.pdf.

[29] M. Che and D. E. Perry, "Architectural design decisions in open software development: A transition to software ecosystems," in *Proceedings of the 23rd Australian Software Engineering Conference*, 2014. https://doi.org/10.1109/ASWEC.2014.37.

[30] S. S. Amorim *et al*, "Software ecosystems' architectural health: Another view," in *Proceedings of the IEEE/ACM Joint 6th International Workshop on Software Engineering for Systems-of-Systems (JSOS)*, 2017. https://doi.org/10.1109/JSOS.2017.15.

[31] K. Telschig *et al*, "SECO patterns: Architectural decision support in software ecosystems," in *Proceedings of the 1st International Workshop on Decision Making in Software Architecture (MARCH)*, 2016. https://doi.org/10.1109/MARCH.2016.10.

[32] W. Schueller and J. Wachs, "Modeling interconnected social and technical risks in open source software ecosystems," *Collective Intelligence*, vol. 3, no. 1, p. 26339137241231912, 2024. https://doi.org/10.1177/26339137241231912.

[33] K. Manikas, "Revisiting software ecosystems research: A longitudinal literature study," *Journal of Systems and Software*, vol. 117, pp. 84–103, 2016. https://doi.org/10.1016/j.jss.2016.02.003.

[34] M. Iansiti and R. Levien, "Strategy as ecology," *Harvard Business Review*, vol. 82, no. 3, pp. 68–81, 2004. https://pubmed.ncbi.nlm.nih.gov/15029791/.

[35] J. Bosch and P. Bosch-Sijtsema, "From integration to composition: On the impact of software product lines, global development and ecosystems," *Journal of Systems and Software*, vol. 83, no. 1, pp. 67–76, 2010. https://doi.org/10.1016/j.jss.2009.06.051.

[36] D. Dhungana, I. Groher, E. Schludermann, and S. Biffl, "Software ecosystems vs. natural ecosystems: learning from the ingenious mind of nature," in *Proceedings of the 4th European Conference on Software Architecture*, 2010, pp. 96-102. https://doi.org/10.1145/1842752.1842777.

[37] R. Kazman, M. Gagliardi, and W. Wood. "Scaling up software architecture analysis." *Journal of Systems and Software* 85, no. 7, pp. 1511-1519, 2012. https://doi.org/10.1016/j.jss.2011.03.050.

[38] A. Tiwana, *Platform ecosystems: Aligning architecture, governance, and strategy*. San Francisco, CA, USA: Morgan Kaufmann, 2014. https://books.google.co.id/books?id=IYDhAAAAQBAJ.

[39] S. Jansen, "How quality attributes of software platform architectures influence software ecosystems," in *Proceedings of the 2013 International Workshop on Ecosystem Architectures (WEA)*, Saint Petersburg, Russia, 2013, pp. 6–10. https://doi.org/10.1145/2501585.2501587.

[40] J. Bosch, "Architecture challenges for software ecosystems," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, Copenhagen, Denmark, pp. 93-95, 2010. https://doi.org/10.1145/1842752.1842776.

[41] K. Lyytinen, Y. Yoo, and R. J. Boland, "Digital product innovation within four classes of innovation networks," *Information Systems Journal*, vol. 26, no. 1, pp. 47–75, 2016. https://doi.org/10.1111/isj.12093.

[42] M. De Stefano, F. Pecorelli, Da. A. Tamburri, F. Palomba, A. De Lucia, "Refactoring Recommendations Based on the Optimization of Socio-Technical Congruence," in *Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2020, pp. 794–796. https://doi.org/10.1109/ICSME46990.2020.00094.

[43] R. T. da Silva, F. L. Gustavo, E. D. Audacio, and E. C. Genvigir, "Identifying actors to support software ecosystem health," in *Proceedings of the 2017 IEEE/ACM Joint 5th International Workshop on Software Engineering for Systems-of-Systems (JSOS)*, 2017, pp. 76–77. IEEE. https://doi.org/10.1109/JSOS.2017.8.

[44] R. Santos, C. Werner, O. Barbosa, and C. Alves, "Software Ecosystems: Trends and Impacts on Software Engineering," in *Proceedings of the 2012 26th Brazilian Symposium on Software Engineering (SBES)*, Natal, Brazil, 2012, pp. 206–210. https://doi.org/10.1109/SBES.2012.24.

[45] D. A. Tamburri, P. Lago, and H. van Vliet, "Organizational social structures for software engineering," *ACM Computing Surveys*, vol. 46, no. 1, art. no. 3, 2013. https://doi.org/10.1145/2522968.2522971.

[46] F. V. Pinheiro, E. Coutinho, M. E. Silva, C. Bezerra, "A systematic mapping of health, quality, evolution, simulation and modeling in software ecosystems," in *Proceedings of the 20th Brazilian Symposium on Information Systems*, 2024, pp. 1–10. https://doi.org/10.1145/3658271.3658297.

[47] A. Zimmermann, R. Schmidt, D. Jugel, and M. Möhring, "Evolving enterprise architectures for digital transformations," in *Proceedings of Gesellschaft für Informatik*, 2015, pp. 183–194. https://publikationen.reutlingen-university.de/frontdoor/deliver/index/docId/609/file/609.pdf.

[48] R. Weinreich and I. Groher, "Software architecture knowledge management approaches and their support for knowledge management activities: A systematic literature review," *Information and Software Technology*, vol. 80, pp. 265–286, 2016. https://doi.org/10.1016/j.infsof.2016.09.007.

[49] J. Schmidt and N. J. Foss, "Modularity, adaptation problems, and the governance and problem-solving capabilities of core firms in ecosystems," *Journal of Management*, vol. 51, no. 4, pp. 1484–1513, 2025. https://doi.org/10.1177/01492063231215023.

[50] H. Koziolek, "Sustainability evaluation of software architectures: A systematic review," in *Proceedings of the 2011 Federated Events on Component-Based Software Engineering, Software Architecture, QoSA+ISARCS*, 2011, pp. 3–12. https://doi.org/10.1145/2000259.2000263.

[51] M. E. Fayad, H. S. Hamza, and H. A. Sanchez, "Towards scalable and adaptable software architectures," in *Proceedings of the 2005 IEEE International Conference on Information Reuse and Integration*, 2005, pp. 102–107. https://doi.org/10.1109/IRI-05.2005.1506457.

[52] G. Brataas, P. Hughes, "Exploring architectural scalability," in *Proceedings of the 4th international workshop on Software and performance*, 2004, pp. 125–129. https://doi.org/10.1145/974044.974064.

[53] A. B. Bondi, "The software architect as the guardian of system performance and scalability," in *Proceedings of the 2009 ICSE Workshop on Leadership and Management in Software Architecture*, 2009. https://doi.org/10.1109/LMSA.2009.5074861.

[54] L. Duboc, D. S. Rosenblum, and T. Wicks, "A framework for characterization and analysis of software system scalability," in Proceedings of *the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2007, pp. 375–384. https://doi.org/10.1145/1254391.1254410.

[55] R. Kazman, M. Gagliardi, and W. Wood, "Scaling up software architecture analysis," *Journal of Systems and Software*, vol. 85, no. 7, pp. 1511–1519, 2012. https://doi.org/10.1016/j.jss.2011.03.050.

[56] J. Bosch, "Software architecture: The next step," in *Lecture Notes in Computer Science*, vol. 3047, pp. 194–199, 2004. https://doi.org/10.1007/978-3-540-24769-2_14.

[57] S. S. Jalali, H. Rashidi, and E. Nazemi, "A new approach to evaluate performance of component-based software architecture," in *Proceedings of the 2011 UKSim 5th European Symposium on Computer Modeling and Simulation*, 2011, pp. 451–456. https://doi.org/10.1109/EMS.2011.77.

[58] M. H. Valipour, B. Amirzafari, K. N. Maleki, and N. Daneshpour, "A brief survey of software architecture concepts and service-oriented architecture," in *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, 2009, pp. 34–38. https://doi.org/10.1109/ICCSIT.2009.5235004.

[59] P. Oreizy, N. Medvidovic, R. N. Taylor, and D. S. Rosenblum, "Software architecture and component technologies: Bridging the gap," 1998. [Online]. Available: http://www.objs.com/workshops/ws9801/papers/paper007.pdf.

[60] A. Jansen and J. Bosch, "Software architecture as a set of architectural design decisions," in *Proceedings of the 5th IEEE/IFIP Conference on Software Architecture (WICSA)*, 2005. https://doi.org/10.1109/WICSA.2005.61.