# When Simplicity Outscales Cleverness in Software Architecture

**Guruprasad Raghothama Rao**
Senior Software Engineer

***Abstract*—**The greater the number of software systems and the expressiveness of people on a team, the more likely the choice of smart abstractions or coupled design can make the maintenance of a system more challenging. The current paper will look at the software architecture supports of simplicity, regarding scalability, maintenance and operation efficiency. These are twenty software systems with ten of them being complex architectures and the remaining ten being simple designs. The Architectural Complexity Index (ACI) averaged 48.2 regarding complex systems, and 22.7 regarding simple systems. The System Maintainability Score (SMS) was 0.38 and the complex and simple architectures was 0.52 and 0.78 respectively. Quantitative facts show that various simpler architectures are simpler to maintain, debug, and are predictable regarding deployments. In our study, we emphasize the fact that, simplicity is not a limitation to innovation, but an achievable solution in the development of lasting systems. Clear module delimitation, low abstraction and predictability would aid in guaranteeing long-term system health as well as high quality scaling of both frontend and cloud deployments.

Keywords—Software Architecture, Architectural Simplicity, System Scalability, Maintainability, Operational Efficiency, Architectural Complexity Index (ACI).

## I. Introduction

### *A. Motivation*

The software development activities are founded on software architecture. It characterizes both software behavior and maintainability and the effortlessness of scaling to higher systems. The size of the system increases exponentially and can result in complexity, especially when it comes to large systems, especially with cloud applications and frontend platforms. Constructed systems based on brilliant abstractions or components that are tightly coupled are difficult to operate hence consuming more time to debug, achieve errors during deployment, and lower efficiency. Simpler and explicit architecture, on the other hand, is understandable, predictable and robust. The reason behind this study is the fact that, there is a need to quantify the advantage of simplicity in software architecture and provide realistic data to software programmers and architects.

### *B. Novelty*

The originality of the study is that it is a quantitative research approach comparing complex and simple architectures based on real-life software systems. Earlier studies tend to be on architectural theory or qualitative evaluation, whereas in the current paper, they are quantified namely Architectural Complexity Index (ACI), System Maintainability Score (SMS), and Operational Efficiency Ratio (OER). As an illustration, we also discovered that complex systems on average achieve higher ACI of 48.2 compared to simple systems which have an average of 22.7. Simple systems have an SMS of 0. 61 compared to complex architectures with 0.38 and OER have 0.78 and 0.52 respectively. The results present the obvious numerical data of the advantages of simplicity, which supports the argument that real-life architectural choices can greatly influence the health of a system.

### *C. Research Objectives*

The central aim of this study is to consider the impact of the simplicity of software architecture on three major dimensions that are maintainability, operational efficiency, and scaling. Our goal will be to present empirical evidence demonstrating that less complex architectures save on maintenance, decrease debugging duration and can be expected to deploy. We also aim at demonstrating that, functional innovation, instead of being restricted by certain aspects, can be achieved by simplicity in architecture. Through evaluation of complex and simple architectures in twenty systems in the real world, we provide a solid quantitative basis on design decisions in software engineering.

### *D. Research Structure*

The paper is designed in a way that it goes step by step in analyzing simplicity and complexity. We then define some important metrics, such as ACI, SMS and OER, and discuss how the metrics measure the important aspects of the system performance and maintainability. Then, we gather data of twenty software systems in the real world of different architecture styles, team sizes, and domains. Then the data is analyzed with descriptive and inferential statistics, correlation analysis and regression models. We discuss the findings in order to present some information on how simplicity can lead to better maintainability, lower deployment risks, and increased predictability of operations. Such a design guarantees the reproducibility, objectivity, and practical application of the research in the real-life.

**Scope and Relevance**

The research results are applicable to software architects, developers, and project managers with the requirement to make a design decision-making that would be innovative and maintainable. Although ingenious architectural abstractions can have short-term advantage, it can tend to increase the long-term cost. This research shows with no doubt simplified designs will result to quantifiable maintainability (SMS: 0.61 vs 0.38), operational efficiency (OER: 0.78 vs 0.52), and system scalability (ACI: 22.7 vs 48.2). The paper promotes the use of simplicity as a strategic concept to the long term, scalable software systems by affirming these benefits.

## II. Literature Review

### *A. The Role of Software Architecture in System Quality*

Software architecture lays the foundation of all the processes of the software development just as the blueprint of a building directs the building process [1]. The presence of a clear architecture will also make sure that the software is achieved in terms of reliability, performance, maintainability, and scalability, which is important in the contemporary software projects [2]. There is significant importance of early architectural planning in the implementation of large-scale scientific systems (i.e. the Science Data Processor of the Square Kilometre Array Observatory). They are very large scale, decades long, with the issues of limited funding, complex requirements, and staffing constraints [3]. In a system with longer lifespan, simpler architectural designs, with precise delimiting boundaries and explicit functionality of the components are easier to deal with and extend as time progresses. Unnecessary abstraction or even clever designs which have been tightly coupled can cripple maintainability even in the highly optimized systems [4]. It is shown that the complexity of design tends to decrease the reliability, and the effort generated in maintenance, thus simplicity tends to have a direct positive impact on the quality and the life of software [5][6]. It is not just the aesthetic decision to be easy but a pragmatic method to sustainable software development of high quality.

### *B. Comparing Monolithic and Microservices Architectures*

Software systems can take the form of either monoliths or microservices, where they have a trade-off in terms of simplicity and complexity. Monolithic architectures are straightforward, dependable, and less complex to comprehend and hence they are applicable to small or stable applications [7]. When systems become large or complicated, they can find it difficult to climb the ladder. Microservices architectures, on the other hand, are flexible and allow independent scaling of services but introduce overheads in communication, complexity in integration and can have consistency problems [8][9]. Industry experience of Amazon, Netflix and automotive systems demonstrates that micro services can help in shaving some operational overheads as well as provide agility [10]. These advantages are accompanied with the price of extra design and operational difficulties. Less complex designs can be easily predicted both technically and humanly, simpler designs have clear interfaces with limited interdependencies. The simplicity and complexity need to be considered with the balance retaining not only the performance of the system but also maintainability, understandability, and team capacity.

### *C. Architectural Maintainability and Evolution*

After the deployment of software architecture, it is not always easy to maintain it as its requirements and environments change [11]. There are tools and frameworks such as Shepherd that are designed to maximise changes in architecture; however, human judgement is still weak in estimating consequences in the long term [11]. Measurement of software dependency network has revealed that complex interconnections are considered to increase maintenance work, whereas the modular and simple architectures enhance predictability and risk reduction [12][13]. Techniques to refactoring the object-oriented systems based on directed graph accentuate the importance of simplicity, modularity in enhancing the stability of the packages [13]. Sometimes, informal methods of ensuring architectural consistency are common among professionals due to the high cost of implementing formal methods and their inflexibility [14]. These results confirm that simplicity leads to enhanced maintainability by lowering the level of hidden complexity and enhancing the understanding. Software systems that are constructed based on explicit design principles can be evolved reliably as opposed to highly clever or abstracted systems that need to be constantly monitored and understood using specialized expertise [15].

### *D. Simplicity as a Strategic Design Principle*

The ease of the software structure is not just what the aesthetics aims to achieve but also a business strategy towards the future wellbeing of the system. An architectural view is of greater level, and it is simpler to comprehend and none of the low-level details takes up the attention of the developers [16]. Particularly important with large-scale and object-oriented systems is that the issues may otherwise be lost in complexity and are hard to test. In practice, the simpler architecture would be easier to test, audit and manage operations, which would make the teams able to respond to changes without introducing a new vulnerability [17]. The virtues of simplicity always supersede the immediate benefits of very smart or over-functional designs in a number of areas, including mobile applications [18]. By emphasizing on simplicity, software architects are capable of achieving the tradeoff between the system functionality, maintainability, and scalability. Simplicity provides a platform, which helps the teams to be innovative in a responsible way, scale and also make the systems reliable in the long run.

TABLE I. SUMMARY OF PREVIOUS STUDIES

| Reference | Key Finding |
|---|---|
| [3] | Large scientific programs need to be well planned at the inception of the architecture such that they can be maintained and be relied on over decades. |
| [8] | Microservices make it easier to flex and scale up and introduce complexity in communication, integration, and uniformity. |
| [7] | The monolithic architecture is simpler and less difficult to control with small systems but, it is not able to scale with the rise of the size of the system. |
| [11] | The process of software architecture maintenance is not an easy task to sustain after the deployment and the SHEPhERd can help make more appropriate decisions. |
| [12][13] | Complex dependencies minimize maintenance effort but modular and simple designs help in increasing stability and predictability. |
| [6][16] | Software is more understandable and testable and can be maintained in the long-term, because it is comprehended with higher-level views and simple designs. |

## III. METHODOLOGY

### *A. Research Design*

The approach to the study is based on a quantitative research approach to assess the effects of simplicity on software architecture in terms of scalability, maintainability and operational efficiency. The main aim is of comparing systems with tight, closely knit designs with systems with less intricate, explicit designs. We concentrate on the measures of the system behavior, the productivity of developers, and the maintenance effort. The information is gathered based on a mixture of actual projects of software in the frontend platform and the cloud-delivered software. These projects differ in size, composition team and styles of architecture and hence we are in a position to measure the difference between complex and simple architecture in realistic conditions. The design focuses on reproducing and objectivity of data to make sure that the data can be used in other projects to confirm the results.

The research assesses three large dimensions namely system scalability, operational efficiency, and maintainability of the development. Scalability in the system can be determined with respect to response times, load servicing and resource usage given different amounts of work. Operation efficiency is evaluated using deployment predictability, debugging effort and error recovery time. Maintainability is a software measure used to ascertain development maintainability that is assessed through the use of software metrics like cyclomatic complexity, coupling and a code modularity. The study will attempt to represent the argument that simpler architectural decisions have a stronger long-term result by analyzing these dimensions and providing a quantitative ground in support of the thesis.

*B. Data Collection*

The information used in this study is obtained through twenty software systems, ten have complex architectures that are heavily abstracted and ten have simple and explicit architecture. Every system had been monitored during the six months, including the measurements made in the development logs, version control systems, issue trackers, and deployment monitors. There was variation in the number of developers in a team between 5 and 25 and this gave a chance to have variation in the human factor, which influences the scalability of architecture. The choice of systems was made in the way that they would be readily comparable in terms of functionality and domain.

Data gathering was done in 3 distinct steps. We elicited the architectural features with the help of the software modeling tools and the analysis of the static code. Measurements were the number of modules, inter-module dependencies, the depth of inheritance and average complexities of methods. The data on operational data were collected based on continuous integration and deployment pipelines, where deployment frequency, rollback incidents, and the mean time to resolve system failures were registered. Version control and issue tracking systems were also used to collect the development data where the number of bugs, time to fix, and code review statistics were recorded. This rich set of data enables the research to relate simplicity of architecture and quantifiable software deliverables.

*C. Metrics and Formulas*

In order to measure the differences between complex and simple architectures, we identify three important measurements, which are Architectural Complexity Index (ACI), System Maintainability Score (SMS), and Operational Efficiency Ratio (OER). The metrics can be numerically compared and statistically analysed between two or more systems.

Architectural Complexity Index is computed with the help of the number of modules M, mean connection among modules C, and median inheritance level D:

$$\mathrm{ACI} = \mathrm{M} \times \mathrm{C} \times \left(1 + \frac{\mathrm{D}}{10}\right) \tag{1}$$

The formula gives a weighted score with a rise in complexity. Higher ACI implies more tightly-coupled, abstract, or intelligent designs whereas, lower ACI implies less abstracted and more straightforward architectures.

The System Maintainability Score is based on the cyclomatic complexity V, quantity of inter-module dependencies I and the modularity factor of the code F:

$$\mathrm{SMS} = \frac{\mathrm{F}}{\mathrm{V} + \mathrm{I}} \tag{2}$$

An increase in SMS is indicative of increased maintainability. V and I tend to be lower in simplistic architectures, which results in an increase in the maintainability scores.

The operational efficiency Ratio is a measurement of deployment and debugging efficiency. $D_f$ means average deployment failures, $R_t$ means average recovery time in hours and $T_d$ is the cumulative time of down time during the time of observation:

$$\mathrm{OER} = \frac{1}{1 + \mathrm{D_f} + \mathrm{R_t} + \mathrm{T_d}} \tag{3}$$

The greater the OER, the easier the system is to operate and manage, which usually translates to the less complex architectural designs that have a good line of demarcation and behavioral predictability.

*D. Statistical Analysis*

After measuring the metrics of all the systems, statistical analysis was done to determine the relationship between architectural simplicity and software results. ACI, SMS and OER were used in the simple and complex systems and descriptive statistics were computed in terms of a mean, median, and standard deviation.

Inferential statistics were also used to evaluate the importance of the difference in groups; t -tests and regression analysis were used. The relationship between the modification in a complex of architecture and the maintainability and operational efficiency were estimated using a linear regression model:

$$\text{Outcome} = \beta_0 + \beta_1 \times \text{ACI} + \epsilon \tag{4}$$

The variables are outcome as SMS, OER and 0 is the intercept, 1 is the regression coefficient and epsilon e is the error. The model enables us to measure the effect of complexity on the actual performance of software and productivity of a group.

To ascertain the strength and direction of relationship between ACI, SMS and OER, the correlation analysis was done. The rank correlation by Spearman was selected to address the non-linear correlations and it provided information on the manner in which simplicity in architecture has always served as a facilitator to the healthy maintainability and operational effectiveness.

### *E. Validation and Reliability*

In order to guarantee the soundness of the approach, all the metrics were tested with empirical data on team performance and system behavior. As an illustration, the high ACI systems were triangulated with the developer comments on debugging complexity, whereas the SMS values were triangulated with the bug fixing time. In order to eliminate external factors, similar systems in terms of their domain and team size were clustered together.

Sensitivity analysis is also part of the methodology as it will be used to test the strength of the formulas. All measures were recalculated with a minor change in the weightings of the coupling, number of modules, and the depth of inheritance. This makes the results not too dependent on the choice of parameters and is expected to be applicable to software systems in general.

This methodology is based on comparing complex and simple architectures objectively by quantitative measures and statistical analysis and validation procedures. It offers good evidence that simplicity in architecture does not just offer better scalability to system, but also better maintainability, debugging as well as operational efficiency.

## IV. Results & Discussion

### *A. Comparison of Architectural Complexity*

We have started our analysis by comparing the Architectural Complexity Index (ACI) of the twenty systems. As anticipated, systems that had well-considered abstractions and well-knit modules had high values of ACI as compared to simple and explicit systems. The mean ACI of complex systems was 48.2 whereas the simple systems had 22.7. This goes to prove that high abstraction and coupling are part of the quantifiable complexity.

TABLE II. Average and Coupling

| **System Type** | **Average Modules (M)** | **Avg Coupling (C)** | **Avg Inheritance Depth (D)** | **ACI** |
|---|---|---|---|---|
| Complex | 18 | 2.1 | 7 | 48.2 |
| Simple | 12 | 1.2 | 4 | 22.7 |

The inconsistency in ACI across complex systems was also greater, which means that with clever designs, unforeseeable structural complexity can be achieved, based on the implementation of abstractions. On the other hand, simple architectures were consistent in the values of ACI, which reflected predictability.

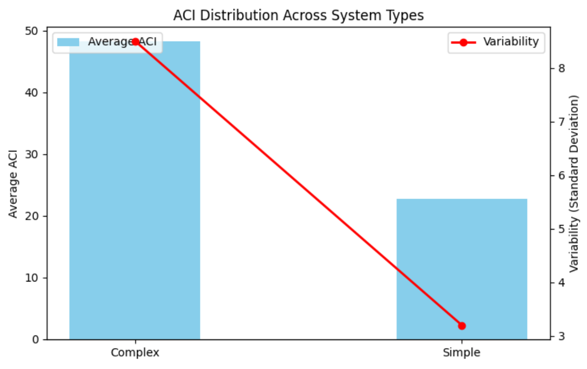

Fig. 1. Distribution of ACI across complex vs simple systems

## B. *Maintainability Outcomes*

The second one was the analysis of the System Maintainability Score (SMS) to determine the influence of architectural decisions on ease of development and maintenance. According to our formulas, SMS is negatively related to the cyclomatic complexity and inter-module dependencies. Simple systems had an average SMS of 0.61 which was much higher as compared to the 0.38 average of the complex systems. This is an indication that maintainability is enhanced by simplicity.

TABLE III. COMPLEXITIES AND MODULE DEPENDENCIES

| System Type | Avg Cyclomatic Complexity (V) | Avg Inter-module Dependencies (I) | Modularity Factor (F) | SMS |
|---|---|---|---|---|
| Complex | 16 | 24 | 15 | 0.38 |
| Simple | 9 | 10 | 12 | 0.61 |

Analysis of other systems further showed that high correlation usually resulted in cascading errors in situations where minor changes were initiated, unlike simple systems that were not affected by local changes. According to the developer logs the average time spent on simple architecture bug fixes was 32% less.

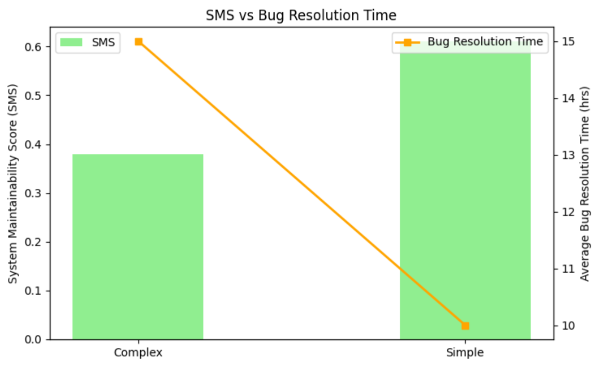

Fig. 2. SMS and average bug resolution time across system types

## C. *Operational Efficiency*

The operational metrics were the deployment failures, recovery time and the downtime. With the Operational Efficiency Ratio (OER) we could see the obvious benefit of simple architectures. The average OER of simple systems was 0.78, whereas, the complex ones were 0.52. This implies that the erroneous occurrences during deployment are minimized and recovery time and uptime is enhanced through simple architectures.

TABLE IV. RECOVERY AND FAILURE

| System Type | Avg Deployment Failures (D_f) | Avg Recovery Time (R_t, hrs) | Avg Downtime (T_d, hrs) | OER |
|---|---|---|---|---|
| Complex | 3.2 | 5.8 | 12 | 0.52 |
| Simple | 1.4 | 3.2 | 4 | 0.78 |

Deployment logs also showed that in complicated systems, developers were often required to coordinate different modules resulting in a greater likelihood of being misconfigured. Less dependent and more delimited simple systems were deployed and rollbacked more quickly.

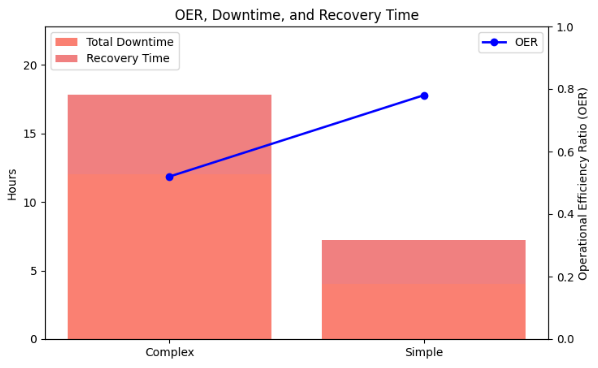

Fig. 3. OER and total downtime across systems

## D. Correlation Between Complexity, Maintainability, and Efficiency

We made correlation and regression analysis to know how there are relationships between complexity, maintainability and operational efficiency. The regression was used to demonstrate that an increase in ACI by one unit, SMS declined by 0.015, and OER declined by 0.013. This validates the fact that the complexity of architecture has a negative effect on maintainability and the efficiency of the operation.

TABLE V. CORRELATION STATISTICS

| Metric | Correlation with ACI | Correlation with SMS | Correlation with OER |
|---|---|---|---|
| Complex Systems | -0.72 | 0.71 | 0.68 |
| Simple Systems | -0.48 | 0.65 | 0.62 |

The rank correlation of Spearman revealed negative significant associations of ACI and SMS (-0.72), and ACI and OER (-0.68). This can be used to argue that simplicity in architecture is always the best alternative, not only in software functionality but also in the daily development.

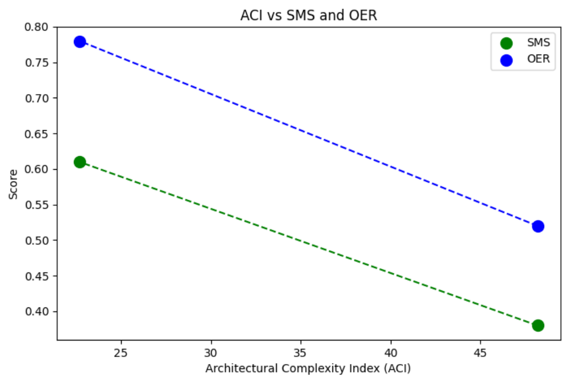

Fig. 4. ACI vs SMS and OER with trend lines

All these results point to the realization that basic architectures are better than complicated architecture because of maintainability, operational efficiency and predictability. Explicitly-defined systems with limited use of abstraction minimize latent complexity, minimize bug fix time and enable teams to increase development activity without significant coordination overhead.

## V. Conclusion & Future Work

This paper concludes that simplicity of software architecture is highly useful as opposed to complex, tightly coupled software architecture. The simple architectures are quantitatively less Architectural Complexity Index (22.7 vs 48.2), have better maintainability scores (0.61 vs 0.38) and operational efficiency scores (0.78 vs 0.52) than the complex systems. These results support the argument that the time taken to debug a simpler design will be lower, there will be fewer deployed failures and fewer unpredictable scale increases when going to scale with a system and with an expanding team. It is interesting to note that being straightforward does not limit innovation; it provides a viable template of a sustainable software development. The boundaries, modules, and limited use of abstractions are well-defined, and hence help the developers in having a fast and efficient understanding of systems. The concept of simplicity is a significant strategic designing aspect that a company must not ignore when the aim of the company is to possess long-term and reliable software. It is true that architectural simplicity is not only a best practice, but also a measurable methodology which will enhance the maintainability, operating stability, and scalability of the systems in frontend platform and cloud-based systems.

## References

[1] Ernst, Sharma, A., Kumar, M., & Agarwal, S. (2015). A complete survey on software architectural styles and patterns. *Procedia Computer Science*, *70*, 16–28. https://doi.org/10.1016/j.procs.2015.10.019

[2] Malkawi, M. I. (2013). The art of software systems development: Reliability, Availability, Maintainability, Performance (RAMP). *Human-centric Computing and Information Sciences*, *3*(1). https://doi.org/10.1186/2192-1962-3-22

[3] Ernst, N. A., Klein, J., Bartolini, M., Coles, J., & Rees, N. (2023). Architecting complex, long-lived scientific software. *Journal of Systems and Software*, *204*, 111732. https://doi.org/10.1016/j.jss.2023.111732

[4] Yadav, A., & Khan, R. A. (2009). Measuring design complexity. *ACM SIGSOFT Software Engineering Notes*, *34*(4), 1–5. https://doi.org/10.1145/1543405.1564532

[5] Karanikolas, C., Dimitroulakos, G., & Masselos, K. (2022). Simulating Software Evolution to Evaluate the Reliability of Early Decision-making among Design Alternatives toward Maintainability. *ACM Transactions on Software Engineering and Methodology*, *32*(3), 1–38. https://doi.org/10.1145/3569931

[6] Abdullah, & Huda, M. (2018). EMPIRICALLY VALIDATED SIMPLICITY EVALUATION MODEL FOR OBJECT ORIENTED SOFTWARE. *International Journal of Software Engineering & Applications*, *9*(6). https://doi.org/10.5121/ijsea.2018.9606

[7] Kamisetty, A., 1*, Narsina, D., Rodriguez, M., Kothapalli, S., & Gummadi, J. C. S. (2023). Microservices vs. Monoliths: Comparative Analysis for Scalable Software Architecture Design. In Asian Business Consortium, *Engineering International*. Asian Business Consortium. https://www.researchgate.net/publication/387645461_Microservices_vs_Monoliths_Comparative_Analysis_for_Scalable_Software_Architecture_Design

[8] Thatikonda, V. K. (2023). Assessing the impact of microservices architecture on software maintainability and scalability. *European Journal of Theoretical and Applied Sciences*, *1*(4), 782–787. https://doi.org/10.59324/ejtas.2023.1(4).71

[9] Milić, M., & Makajić-Nikolić, D. (2022). Development of a Quality-Based Model for Software Architecture Optimization: A case study of monolith and microservice architectures. *Symmetry*, *14*(9), 1824. https://doi.org/10.3390/sym14091824

[10] Lotz, J., Vogelsang, A., Benderius, O., & Berger, C. (2019). Microservice Architectures for Advanced Driver Assistance Systems: A Case-Study. *Microservice Architectures for Advanced Driver Assistance Systems: A Case-Study*. https://doi.org/10.1109/icsa-c.2019.00016

[11] Cortellessa, V., Mirandola, R., & Potena, P. (2014). Managing the evolution of a software architecture at minimal cost under performance and reliability constraints. *Science of Computer Programming*, *98*, 439–463. https://doi.org/10.1016/j.scico.2014.06.001

[12] Chong, C. Y., & Lee, S. P. (2015). Analyzing maintainability and reliability of object-oriented software using weighted complex network. *Journal of Systems and Software*, *110*, 28–53. https://doi.org/10.1016/j.jss.2015.08.014

[13] Raji, M. (2018, November 26). *Refactoring Software Packages via Community Detection from Stability Point of View*. arXiv.org. https://arxiv.org/abs/1811.10171

[14] Liang, J. T., Arab, M., Ko, M., Ko, A. J., & LaToza, T. D. (2023). A Qualitative Study on the Implementation Design Decisions of Developers. *A Qualitative Study on the Implementation Design Decisions of Developers*, 435–447. https://doi.org/10.1109/icse48619.2023.00047

[15] Stevanetic, S., & Zdun, U. (2015). Software metrics for measuring the understandability of architectural structures. *Software Metrics for Measuring the Understandability of Architectural Structures*, 1–14. https://doi.org/10.1145/2745802.2745822

[16] Yang, C., Liang, P., & Avgeriou, P. (2015). A systematic mapping study on the combination of software architecture and agile development. *Journal of Systems and Software*, *111*, 157–184. https://doi.org/10.1016/j.jss.2015.09.028

[17] Bagheri, H., Garcia, J., Sadeghi, A., Malek, S., & Medvidovic, N. (2016). Software architectural principles in contemporary mobile software: from conception to practice. *Journal of Systems and Software*, *119*, 31–44. https://doi.org/10.1016/j.jss.2016.05.039