Evaluating Software Architecture: A Systematic Mapping Study on Design Metrics and Their Applications

Vira Liubchenko [1]*†

[1]*Odesa Polytechnic National University, 1 Shevchenka av., Odesa, 65044, Ukraine*

Abstract
Architectural design is pivotal in software development, shaping system functionality and behaviour. This study consolidates experiences with architectural design metrics to enhance software development processes and products. Metrics, while not definitive of quality, highlight potential issues and areas for improvement. Essential properties of effective metrics include simplicity, empirical credibility, consistency, uniform dimensionality, language independence, and actionable feedback. A systematic mapping study analysed publications on software architectural design metrics, focusing on quality models and metamodels. The study identified vital metrics such as coupling, cohesion, complexity, and modularity and explored their applications in various contexts, including microservices and security. Results indicate a lack of substantial experience in applying these metrics. The metrics taxonomy includes structural integrity, adaptability, complexity, maintainability, and traceability, providing a comprehensive framework for evaluating software architecture. The study emphasises the need for further research to develop predictive models using architectural metrics, which could improve software quality by proactively addressing architectural issues. Future efforts should delve into data accumulation and investigate models for using these metrics for predictive purposes, ultimately enhancing software quality and development processes.

Keywords
Software architecture, design metrics, quality models, empirical software engineering, component cohesion, architectural stability, metric taxonomy, predictive models.

1. Introduction

Architectural design is a cornerstone of software development, as it shapes the entire system and influences its functionality and behaviour. It involves creating a high-level abstraction of system topology, functionality, and behaviour, which serves as the basis for detailed design and implementation. While various interpretations of software architecture exist, they all centre on components, modules, and their interconnections.

Guided by established design patterns, best practices, and stakeholder considerations, architectural design should prioritise adherence to proven methodologies. This process often entails navigating trade-offs to reconcile conflicting objectives. A meticulously planned design is indispensable throughout the system's creation and evolution, especially when accommodating changes and adaptations.

Quality architecture serves as the foundation for ongoing software development endeavours. It's reasonable to infer that the attributes of an architectural design shape the qualities of individual development artefacts and the software as a whole. If we could quantitatively describe these dependencies and influences, we could anticipate the characteristics of these artefacts and the entire software system.

Datasets commonly employed in empirical software engineering research, such as NASA's repositories, predominantly encompass code characteristics and defects metrics. Hence, it's

14th International Scientific and Practical Conference from Programming UkrPROGP'2024, May 14-15, 2024, Kyiv, Ukraine
* Corresponding author.
† These authors contributed equally.
lvv@op.edu.ua (V. Liubchenko)
0000-0002-4611-7832 (V. Liubchenko)

pertinent to investigate the utilisation and purposes of metrics describing architectural design in published studies.

The metrics themselves do not determine the quality of architecture as good or bad. However, they serve as valuable aids for architects, highlighting potential problematic elements and areas for improvement in their designs. A software architecture analysis tool computes various metrics, adhering to specific guidelines for software metrics. These metrics must possess the following properties:

- Simplicity and computability: metrics should be straightforward to understand and apply.
- Empirical credibility: metrics should align with engineers' expectations and practical experiences.
- Consistency and objectivity: metrics should yield unambiguous results.
- Uniform dimensionality: metrics should maintain mathematical coherence.
- Language independence: metrics should be applicable across different programming languages.
- Facilitation of feedback: metrics should offer actionable insights to enhance software quality and performance.

The primary objective of this paper is to consolidate the existing experiences in employing architectural design metrics. Additionally, we aim to identify promising areas for their application, aiming to enhance both the software development process and the resultant product.

The rest of the paper is structured as follows. First, in section 2, we explain our study design and methodology. Next, section 3 presents architectural metrics and their properties identified during our review. In section 4, we propose the taxonomy of architectural metrics. Section 5 discusses two ways to use metrics to enhance software development. Finally, we close the study by summarising the conclusions in section 6.

2. Research Method

We undertook a systematic mapping study to analyse existing publications about using software architectural design metrics. This study is a valuable tool for examining the evolution of research in this domain over time. Below, we outline the methodology employed in conducting this research.

The primary objective of this research is to systemise metrics that characterise the architecture design and methods of its application in software engineering processes, with a focus on quality models and metamodels. Towards that goal, we explored the space of software design metrics and their support for software engineering processes. This study aims to provide an overview of the state-of-the-art research on software architecture evaluation, focusing on metrics defined for architectural diagrams. The research questions are:

RQ1. What are the metrics that are applied to architectural design? This question is crucial as it forms the basis of our understanding of how architecture design is measured and evaluated.

RQ2. What are the purposes for which they are applied?

A search process has been developed to answer these questions. The overall selection process is depicted in Figure 1.

We searched four reference databases for architecture design metrics papers: IEEE Xplore, ACM Digital Library, ScienceDirect, and Elsevier. The primary keywords were "software architecture design" and "metrics."

The search encompassed the title, abstract, and keywords fields, and the formulated search string was:

("software architecture design") AND (("metrics" OR "measuring")).

This string was encoded according to the syntax format of each respective database.

The search was initially confined to papers published until 2018. However, this time limit was subsequently removed due to the limited number of results obtained.

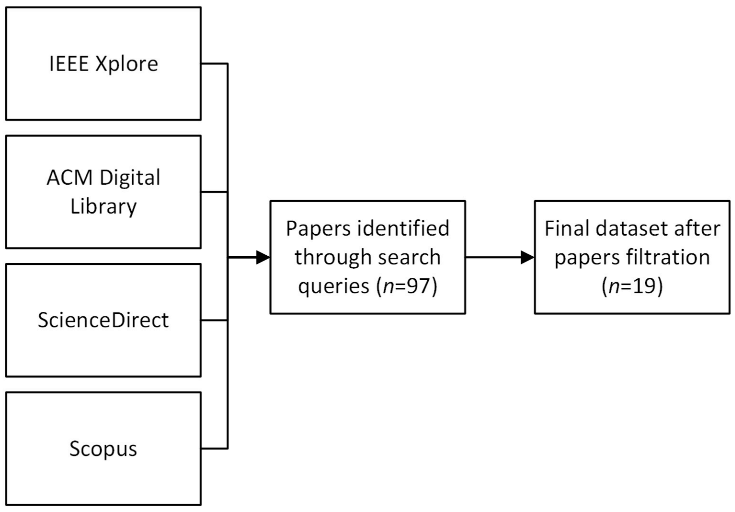
IEEE Xplore
ACM Digital Library
ScienceDirect
Scopus
Papers identified through search queries (*n*=97)
Final dataset after papers filtration (*n*=19)

Figure 1: Selection process.

We used inclusion and exclusion criteria to review and select the papers.
The inclusion criteria are:

- Articles written in English language.
- Published in Journals or Conference Proceedings.
- The paper's topic is related to software architecture design and measurements.
- Content: the complete text must be available.

The exclusion criteria are:

- Duplicates.
- Articles written as systematic mapping studies.
- Metrics are for a specific quality characteristic such as security.

The domain of the paper is non-software, such as hardware infrastructure architecture.
As illustrated in Figure 1, the search on the databases utilising the specified search string yielded 97 papers. These papers underwent filtration in two phases based on predefined selection criteria:
Phase 1: Elimination of duplicates and review papers.
Phase 2: Comprehensive reading of the full-text papers.
Phase 2 has filtered out, for example, the works on the similarity metrics, detailed design and program code metrics, dependencies between code properties and architecture smell, and architecture knowledge base.
Following the selection procedure, only 19 primary papers were retrieved for the systematic mapping study, representing merely 19.6% of the initial dataset. This limited number underscores the apparent lack of substantial experience in successfully applying such metrics. Figure 2 illustrates the trend in publication over the designated period.

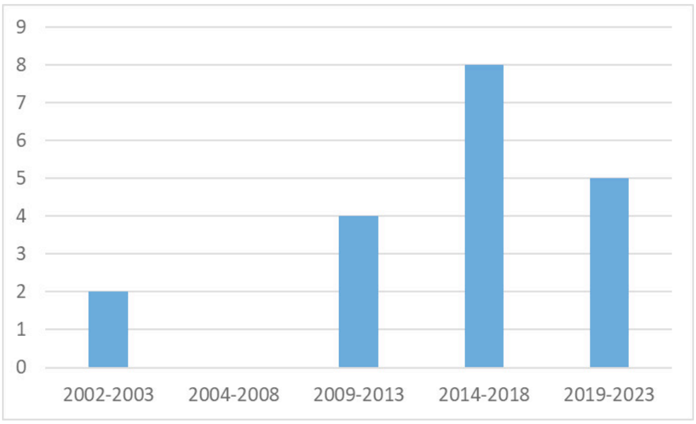

Figure 2: Paper Distribution per Year.

3. Mapping Results

In this section, we describe the results of our review.

The first metric of interest is the cost associated with the evolution of architectural projects [1]. In software architectures, evolution involves a series of changes to components and connectors. To quantify the evolution of software architectures, authors proposed metrics based on rewriting operations applied to components and connectors: addition, removal, and modification. The authors defined the cost of a set of changes as the weighted sum of the costs of each operation. The metric is quite complicated; its application is very specific.

In the same year, the work [2] described a bunch of architectural metrics in particular:

Coupling counts the number of distinct components that a service interacts with.

Cohesion assesses internal consistency by counting the use cases involving a service being called by or calling another service within a specific component.

Complexity of Services calculates each component service's mean number of state transitions.

Number of Services per Component enumerates a particular component's services.

Fanin measures how often a component's services are invoked across all scenarios.

Fanout counts the external service calls a specific component makes in all scenarios.

Depth of Scenario assesses the nesting level of service calls within a scenario.

This study is interesting because most of the metrics listed have been used in many later studies. We also see an attempt to link to the previous development phase—requirements analysis. It's crucial to understand that the Depth of Scenario metric primarily sheds light on the complexity of the use case scenarios depicting the architecture rather than the inherent complexity of the architecture itself.

Let's focus on Cohesion metrics, which have remained in demand over the years. For instance, a study [3] proposed defining cohesion based on factors such as the number of requirements associated with a software component, the number of functions per software component, and the average interaction among functions within components. Another study [4] introduced the Lack of Cohesion Metric, which gauges the coherence or, in simpler terms, the resemblance among operations within specific services and their interrelatedness. Furthermore, the paper [5] empirically investigated the practical viability of service level cohesion metrics within an open-source Microservices Architecture application context.

Sethi et al. [6] introduced a collection of metrics focused on design stability and modularity, which uniquely incorporates considerations of environmental factors and the evaluation of alternatives. Specifically, they developed the Decision Volatility metric to evaluate the stability of decisions based on the number of environmental conditions affecting them and the breadth of their impact. For modularity, they presented the Independence Level metric, aimed at measuring a design's capacity to facilitate independent searching and replacement at the module level, essentially assessing its capability to produce option values.

A study [7] introduced a comprehensive metric named Component Balance (CB). CB is a composite metric derived from two distinct measures: System Breakdown, which evaluates the degree to which a system is decomposed into a sensible number of components, and Component Size Uniformity, which assesses the consistency of component sizes within the system. In a subsequent study by the authors [8], they examined two architecture-level metrics: Component Balance and Dependency Profiles. The analysis focused on understanding the practical challenges of implementing these metrics within an industrial context.

In work by Perez-Palacin et al. [9], a dedicated metric was introduced to comprehensively assess the system's size and the effort required to manage its overall architecture. The Absolute Software Units Index (ASUI) quantifies the total number of software units utilised within the system.

The investigation outlined in [10] focused on the interaction aspects of architectural design, distinguishing between two types of interactions. In-interactions denote the edges incoming to a component originating from other components, while Out-interactions refer to edges outgoing from one component to other components. Two metrics were introduced based on these interaction characteristics. The Interaction Ratio of a component is calculated by dividing the total number of Out-interactions by the total number of In-interactions associated with that component. Additionally, the Percentage of Interaction between Components is determined as the ratio of the sum of In-interactions and Out-interactions to the maximum potential interactions among the components.

Unlike prior research, the study [11] advocates for employing fundamental metrics: the Number of Components, the Number of Connectors (regardless of their directionality), and the Number of Elements (totalling the components and connectors). The authors named these metrics understandability ones, offering a straightforward approach to evaluating system comprehensibility.

The research [12] assembled and structured a preliminary compilation of architectural metrics according to viewpoints and domains. Nevertheless, the study concluded that evaluating the overall usefulness of architectural metrics remains an open research question.

Fascinating is the study [13], which focused on quantifying architectural decay. In the work, the authors relied on three existing metrics based on component interdependencies. These are the Ratio of Cohesive Interactions, Instability and Modularization Quality. The authors also defined three additional new metrics that explicitly detect architectural smells. These are Bidirectional Component Coupling, Architectural Smell Density, and Architectural Smell Coverage. The set of metrics should provide a set of predictive models that help engineers predict bugs by using both implicit and explicit system features.

In the work [14], the authors undertook a comprehensive evaluation of architecture by categorising established metrics into a hierarchical structure and introducing novel aggregated metrics. These new metrics encompassed Structural Quality, which comprised Architecture Size and Architecture Links, and System Characteristics, which encompassed Domain Characteristics and Agility. The study's findings enabled insights into the efficacy of the proposed metrics during the architecture design phase for software systems.

The study in [15] delved into two metrics concerning the traceability connections between requirements and architectural design. Traced Requirements per Component quantifies the number of requirements linked to a particular component, while Traced Components per Requirement assesses the number of components linked to a requirement. The intricate dependencies observed between requirements and architectural components could potentially serve as indicators of software defects.

The study outlined in [16] presents several architecture-level metrics to assess architecture quality. One such metric is the Decoupling Level, designed to gauge the degree to which a software system is divided into small, autonomous modules. The Propagation Cost metric was also introduced to evaluate the extent of tight coupling within the system. The study also employed two established architecture-level metrics, Component Balance and Dependency Profiles. Their investigation into the effectiveness of these metrics revealed that the measurement outcomes aligned with practitioners' intuitions in understanding architecture and detecting architectural flaws.

The study documented in [17] focused on developing metrics to assess the security level within an architectural model. The authors introduced three metrics specifically aimed at evaluating the vulnerability of entry points within composite components, drawing from the concept of "protected entry points" pattern. This research is notable for its shift towards examining a specific quality attribute within the architectural domain.

The investigation in [18] examined particular metrics tailored for microservices architecture. Should the application's architecture fail to meet requirements during runtime, costly redevelopment efforts may become necessary. Establishing relationships between architectural metrics of microservices-based applications and runtime metrics makes it feasible to conduct design time evaluations.

Nevertheless, it's worth noting that microservices architecture doesn't inherently demand specialised metrics. Recent research [19] indicates that the commonly used structural design property metrics—such as coupling, size, cohesion, and complexity—are suitable for evaluating a service-based architecture.

4. Architectural Metrics Taxonomy

Clustering the metrics used in architectural design creates a metrics taxonomy. Here is a description of this taxonomy.

1. Structural Integrity Cluster focuses on the core structural aspects of the architecture, including how well the components are organised and interact with each other and how stable and balanced the system is.

1.a. Cohesion group assesses the degree to which elements within a module or component are related and work together effectively.

1.a.1. Architecture Relational Cohesion evaluates the degree to which components or modules within a software architecture are functionally related and work together cohesively.

1.a.2. Cohesion defines how well the elements within a component relate to each other and perform a single, focused function.

1.a.3. Lack of Cohesion Metric identifies components with elements that are poorly related to each other.

1.a.4. Ratio of Cohesive Interactions is the ratio of interactions within a component compared to interactions with other components.

1.b. Complexity group assesses the intricacy of the system's design and interactions, which can impact maintainability and comprehensibility.

1.b.1. Bidirectional Component Coupling measures the number of dependencies between two components where each relies on the other.

1.b.2. Coupling is the degree of interdependence between components.

1.b.3. Decoupling Level defines the degree of independence between components.

1.c. Dependency group analyses the relationships and dependencies among various components, which can influence system robustness and modularity.

1.c.1. Dependency Profiles describe the types and patterns of dependencies between components.

1.c.2. Fan-in defines the number of components that depend on a particular component.

1.c.3. Fan-out defines the number of components that a particular component depends on.

1.c.4. Propagation Cost estimates the effort required to propagate changes in one component to other affected components.

1.d. Size and Uniformity group evaluates the size distribution and uniformity of components, which can affect the manageability and balance of the architecture.

1.d.1. Component Balance determines whether the distribution of functionality across components is balanced or skewed.

1.d.2. Component Size Uniformity defines consistency in the size of components measured in the number of methods, interface size, etc.

1.d.3. Number of Components / Connectors / Operations defines the total number of respective elements in the architecture.

1.d.4. Number of Services of Component defines the number of services a single component provides.

1.e. Stability group evaluates how resistant the architecture is to changes and how well it maintains its integrity over time.

1.e.1. Architecture Stability measures the resistance of the architecture to changes and disruptions.

1.e.2. Instability measures the susceptibility of the architecture to failures and errors.

2. Adaptability and Evolvability Cluster addresses the system's ability to adapt to changes and evolve over time, as well as the granularity of its components, which affects flexibility and scalability.

2.a. Adaptability group measures the ease with which a system can be adapted to new requirements or environments.

2.a.1. Absolute System Adaptability Index was designed to quantify the degree to which a software system can adapt to changes in requirements, environments, or technologies.

2.a.2. Evolution Cost Metrics estimates the effort required to modify the architecture in the future.

2.b. Volatility group measures the frequency and extent of changes within the system, indicating how dynamic and change-prone the architecture is.

2.b.1. Design Volatility measures how often the architectural design needs to be changed due to internal or external factors.

2.c. Granularity group assesses the level of detail and decomposition of services or components, which impacts flexibility and scalability.

2.c.1. Service Granularity Metric evaluates the size and scope of services within a software architecture.

2.c.2. System Breakdown measures the overall health and maintainability of the system architecture.

3. Complexity and Maintainability Cluster examines the complexity of the system, which affects how maintainable and comprehensible the architecture is. It also identifies architectural smells that may indicate more profound issues.

3.a. Complexity group assesses the intricacy of the system's design and interactions, which can impact maintainability and comprehensibility.

3.a.1. Complexity of Services measures the complexity of the functionalities provided by services in the architecture.

3.a.2. In-Out Interaction Complexity Metrics measure the complexity of interactions between a component and its external environment.

3.a.3. Impact Scope defines how a change in one component affects other components.

3.b. Architectural Smell group identifies issues or "smells" in the architecture that might indicate deeper problems or areas for improvement.

3.b.1. Architectural Smell Coverage measures the percentage of design patterns identified as potential problems.

3.b.2. Architectural Smell Density counts the number of potential problems found per design unit.

4. The Traceability and Miscellaneous Quality Attributes Cluster includes metrics related to the traceability of requirements and components and other miscellaneous quality attributes that don't neatly fit into the other groups.

4.a. Traceability group measures how well requirements and components can be traced through the system, impacting accountability and understandability.

4.a.1. Traced Components per Requirement measures how many components are associated with a single requirement.

4.a.2. Traced Requirements per Component measures how many requirements a single component is responsible for.

4.b. Miscellaneous group encompasses various metrics that don't neatly fit into the other categories but still provide valuable insights into the architecture.

4.b.1. Depth of Scenario defines the number of components involved in a specific user scenario.

4.b.2. Independence Level is similar to the Decoupling Level and measures how self-contained components are.

4.b.3. Modularization Quality measures how well the architecture is divided into independent, reusable modules.

4.b.4. Protected Entry Points measure the number of well-defined access points for interacting with a component.

Creating a taxonomy for architectural design metrics through clustering enhances our ability to evaluate and improve software architecture. We can systematically assess various aspects of architecture by organising metrics into clusters such as Structural Integrity, Adaptability and Evolvability, Complexity and Maintainability, and Traceability and Miscellaneous Quality Attributes. This structured approach enables more precise identification of architectural instabilities, development of predictive models, and proactive enhancement of software development processes. As a result, this leads to more stable, robust, and maintainable software systems.

5. Architectural Metrics Application

Integrating architectural metrics for fault prediction can improve software development processes by proactively addressing architectural issues and ensuring the software architecture's stability and robustness.

A review of the literature reveals that research primarily focuses on reactive enhancements. Figure 3 illustrates a typical process.

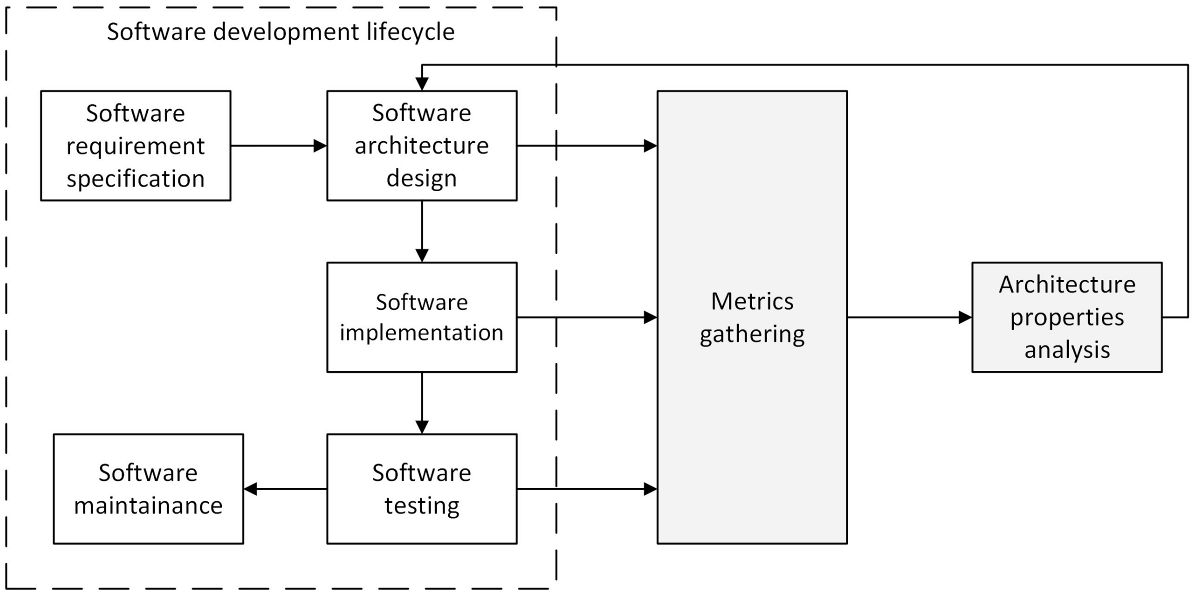
Software development lifecycle
Software requirement specification
Software architecture design
Software implementation
Software testing
Software maintainance
Metrics gathering
Architecture properties analysis

Figure 3: Reactive enhancement process.

Initially, a comprehensive development cycle is executed, during which various measurements are collected. These metrics pertain predominantly to code properties, while design metrics play a supplementary role. The collected data serves as the foundation for identifying potential problematic areas within the architectural project.

The process can be improved. By obtaining sufficient measurements of architectural metrics and utilising an effective model to predict quality characteristics based on these metrics, we can enhance the architectural design before its implementation in code. Figure 2 illustrates such a predictive process.

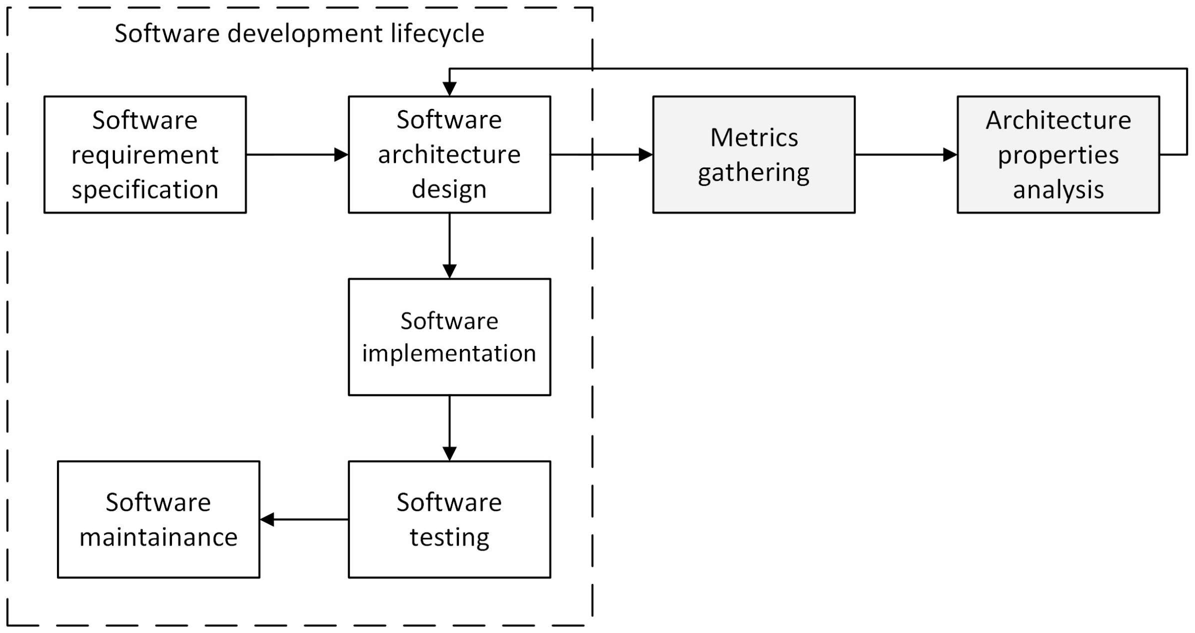
Software development lifecycle
Software requirement specification
Software architecture design
Software implementation
Software testing
Software maintainance
Metrics gathering
Architecture properties analysis

Figure 4: Predictive enhancement process.

To establish a predictive process, several preparatory steps are required.

1. Collect historical measurement data on architectural metrics. As mentioned above, architectural design metrics can be leveraged during the design phase to evaluate

architecture quality. These metrics include Decoupling Level, Propagation Cost, Component Balance, and Dependency Profiles. Unfortunately, existing metric repositories contain limited data on these measurements. Additionally, categorising architectural metrics into a hierarchical structure (presented in section 4) and developing new aggregated metrics facilitate a thorough assessment of architecture, encompassing structural quality and overall system characteristics.

2. Identify architectural instability. Utilising these metrics allows for detecting architectural instability growth as the system evolves. It also aids in identifying classes that contribute to architectural inconsistencies, thereby enhancing fault prediction capabilities.
3. Utilise architectural metrics in predictive models. It is essential to develop predictive models that incorporate architectural design metrics. These models enable practitioners to anticipate potential faults and take preventative measures to improve software quality.
4. Enhance software development processes. Integrating these metrics for fault prediction can significantly improve software development processes. By proactively addressing architectural issues, we can ensure the stability and robustness of the software architecture.

Thus, while existing research primarily focuses on reactive enhancements, there is a clear opportunity to shift towards a more predictive approach. Practitioners can enhance architectural design before implementation by collecting comprehensive historical data on architectural metrics, identifying architectural instabilities, and developing predictive models. This proactive approach will ultimately lead to more stable and robust software systems.

6. Discussion and Conclusion

Studies have shown that software architecture design metrics can be valuable for designers. These metrics help designers evaluate two critical aspects of an architecture: its ability to generate different design options and its stability under various external conditions.

We observed that the metrics strategy solutions were the most frequently used in identifying architectural decay. These metrics can determine the growth of architectural instability with system evolution, identify the probability of the classes contributing to architectural inconsistencies, and diagnose the anomalies, whether agglomerations or individuals are more correlated to architectural problems.

We can also see that architectural design metrics can be used to measure quality attributes.

While the introduction discussed the potential of architectural design metrics for fault prediction, no existing research directly addresses this. Existing studies focus on the relationship between metrics from different development stages. However, these studies analyse the "reverse" effect, where metrics from later stages (e.g., code characteristics) are used to identify architectural issues like design erosion or antipatterns.

To summarise, the potential of using architectural design metrics is not fully utilised. Existing studies have shown that one set of relatively simple metrics can be successfully used for various architectural solutions. Therefore, there are no principal obstacles to using these metrics to predict the characteristics of artefacts of subsequent stages of development and software in general. Consequently, further research should be conducted to examine the accumulation of data and investigate models that can be used for prediction.

References

[1] M. Aoyama, Metrics and analysis of software architecture evolution with discontinuity. Proceedings of the International Workshop on Principles of Software Evolution. (2002) 103–107. doi: 10.1145/512035.512059.

[2] J. Muskens, M. R. V. Chaudron, R. Westgeest, Software architecture analysis tool: software architecture metrics collection. Proceedings 3rd PROGRESS Workshop on Embedded Systems. (2002) 128139.

[3] H. Venkitachalam, J. Richenhagen, A. Schlosser, Th. Tasky, Metrics for Verification and Validation of Architecture in Powertrain Software Development. Proceedings of the First International Workshop on Automotive Software Architecture. (2015) 27–33. doi: 10.1145/2752489.2752496.

[4] O. Al-Debagy, P. Martinek, A Metrics Framework for Evaluating Microservices Architecture Designs. Journal of Web Engineering. 19(3–4) (2020) 341370. doi: 10.13052/jwe15409589.19341.

[5] M. G. Moreira, B. B. N. De França, Analysis of Microservice Evolution using Cohesion Metrics. Proceedings of the 16th Brazilian Symposium on Software Components, Architectures, and Reuse. (2022) 40–49. doi: 10.1145/3559712.3559716.

[6] K. Sethi, Y. Cai, S. Wong, A. Garcia, C. Sant'Anna, From retrospect to prospect: Assessing modularity and stability from software architecture. Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture. (2009) 269272. doi: 10.1109/WICSA.2009.5290817.

[7] E. Bouwers, J. P. Correia, A. v. Deursen, J. Visser, Quantifying the Analyzability of Software Architectures. Ninth Working IEEE/IFIP Conference on Software Architecture. (2011) 8392. doi: 10.1109/WICSA.2011.20.

[8] E. Bouwers, A. van Deursen, J. Visser, Evaluating usefulness of software metrics: An industrial experience report. 35th International Conference on Software Engineering. (2013) 921930. doi: 10.1109/ICSE.2013.6606641.

[9] D. Perez-Palacin, R. Mirandola, J. Merseguer, Software architecture adaptability metrics for QoS-based self-adaptation. Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS. (2011) 171–176. doi: 10.1145/2000259.2000288.

[10] U. Tiwari, S. Kumar, In-out interaction complexity metrics for component-based software. SIGSOFT Softw. Eng. Notes. 39(5) (2014) 1–4. doi: 10.1145/2659118.2659135.

[11] S. Stevanetic, T. Haitzer, U. Zdun, Supporting Software Evolution by Integrating DSL-based Architectural Abstraction and Understandability-Related Metrics. Proceedings of the 2014 European Conference on Software Architecture Workshops. (2014) 1–8. doi: 10.1145/2642803.2642822.

[12] O. Zimmermann, Metrics for architectural synthesis and evaluation: requirements and compilation by viewpoint: an industrial experience report. Proceedings of the Second International Workshop on Software Architecture and Metrics. (2015) 8–14.

[13] D. Le, Architectural-Based Speculative Analysis to Predict Bugs in a Software System. IEEE/ACM 38th International Conference on Software Engineering Companion. (2016) 807810.

[14] S. Orlov, A. Vishnyakov, Decision Making for the Software Architecture Structure Based on the Criteria Importance Theory, Procedia Computer Science. 104 (2017) 2734. doi: 10.1016/j.procs.2017.01.050.

[15] B. Nassar, R. Scandariato, Traceability Metrics as Early Predictors of Software Defects? IEEE International Conference on Software Architecture. (2017) 235238. doi: 10.1109/ICSA.2017.12.

[16] W. Wu et al., Software Architecture Measurement—Experiences from a Multinational Company. In: Cuesta, C., Garlan, D., Pérez, J. (eds) Software Architecture. Lecture Notes in Computer Science. 11048 (2018) 303–319. doi: 10.1007/9783030007614_20.

[17] M. Buitrago, I. Borne, J. Buisson., Deriving metrics for software architectures from the "protected entry points" security patterns. Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. (2023) 1473–1475. doi: 10.1145/3555776.3577816.

[18] N. Knoll, R. Lichtenthäler, An Experimental Evaluation of Relations Between Architectural and Runtime Metrics in Microservices Systems. Proceedings of the 13th International Conference on Cloud Computing and Services Science. (2023) 147154. doi: 10.5220/0011728600003488.

[19] M. H. Hasan, M. H. Osman, N. I. Admodisastro, M. S. Muhammad, From Monolith to Microservice: Measuring Architecture Maintainability. International Journal of Advanced Computer Science and Applications. 14(5) (2023) 857–866. doi: 10.14569/IJACSA.2023.0140591.