Journal of Computing Theories and Applications ISSN:3024-9104

Review Article

Methodologies of the Validation of Software Architectures

Amina El Murabet * and Anouar Abtoy

SIGL, ENSATE, Abdelmalek Essaadi University, Tetouan, Morocco; e-mail: elmurabet.amina@uae.ac.ma, aabtoy@uae.ac.ma
* Corresponding Author : Amina El Murabet

Abstract: Software architecture validation is the process of assessing whether a software architecture meets its intended requirements and goals. It is an important step in the software development process, as it can help to identify and address potential problems early on before they become more costly and difficult to fix. There are a variety of different methodologies that can be used to validate software architecture. Some of the most common methodologies include Architectural evaluation methods, Architecture tests and reviews, and Model-based validation. This paper will provide an overview of the different methodologies that can be used to validate software architecture. Apart from that, it also analyzes and summarizes the strengths and weaknesses of each method so that it can guide determining the most appropriate methodology for a particular case.

Keywords: Software engineering; Software architecture; Software development process; Validation; Methodologies.

Received: October, 8th 2023
Revised: October, 25th 2023
Accepted: October, 27th 2023
Published: November, 1st 2023

1. Introduction

The essential framework of a software system is its architecture. It outlines the system's constituent parts, their connections, and how they work together. The process of determining if a software architecture satisfies its intended requirements and objectives is known as validation. The failure to validate software architecture can lead to a variety of issues[1]. So, it is a crucial stage in the software development process since it can assist in identifying and addressing possible issues before they become more expensive and challenging to fix. The following are some of the most typical issues:

- Performance problems: A software architecture may not be able to handle the anticipated load if it has not been thoroughly verified. Performance issues like slow response times and outages may result from this.
- Scalability problems: A software architecture may not be able to scale to meet the demands of the business if it is not properly verified. Due to this, there may be issues with scaling, such as the inability to manage growing traffic or data volumes.
- Maintainability problems: If a software design is not properly proven, maintaining it could be time-consuming and expensive. Maintainability issues, such the inability to patch bugs or add new features, may result from this.
- Security problems: If a software architecture isn't properly tested, it could be open to attack. Security issues may result from this, including disruptions in system performance or unauthorized access to data and systems.
- Quality problems: If a software architecture is not properly validated, it may not fulfill the business's standards for quality. Data loss, crashes, and other quality issues like bugs could result from this.

The non-validation of the software architecture can cause very serious problems and can be led to huge losses. For example, a glitch in the rocket's guidance system's software caused the Ariane 5 rocket to explode 40 seconds after its takeoff. A floating-point exception, a type of error that can happen when a computer performs calculations with floating-point values, was what actually triggered the issue. The explosion resulted from the software developers' failure to check the guidance system for this kind of malfunction. The explosion cost over

$370 million[2]. In 2012, Knight Capital Group lost over $440 million in 30 minutes due to a software glitch in its trading program[3]. The glitch caused the algorithm to place thousands of erroneous orders, which led to significant losses for the company. The software issue was caused by a failure to validate the algorithm before it was deployed.

There are a variety of different methodologies that can be used to validate software architecture. Some of the most common methodologies include:

- Architectural evaluation methods: These methods involve a systematic assessment of the architecture against a set of criteria, such as performance, scalability, security, and maintainability. Some examples of architectural evaluation methods include the Architecture Trade-off Analysis Method (ATAM)[4], the Scenario-based Architecture Analysis Method[5], the Software Architecture Analysis Method (SAAM)[6], the Architecture Decision Records (ADR) approach[7], and the Architecture Review Board (ARB) process[8].
- Architecture tests and reviews: These methods involve testing and reviewing the architecture to identify any errors, defects, or deviations from the specifications. Some examples of architecture tests and reviews include unit tests, integration tests, system tests, code reviews, and architecture reviews[9].
- Model-based validation: This involves using architectural models to validate the architecture. Architectural models can be used to simulate the system's behavior and identify potential problems[10].

Each of these approaches has advantages and disadvantages of its own. However, they can also be more time- and money-consuming. Architectural evaluation procedures are often more thorough than architecture assessments and reviews. In order to identify possible issues early on, model-based validation often necessitates the creation of precise architectural models. The most appropriate methodology for a given situation will depend on a number of factors, such as the size and complexity of the system, the available resources, and the time constraints. In some cases, it may be necessary to use a combination of different methodologies to validate the software architecture comprehensively.

Various works tried to explore the validation of the software architecture. The most recent one goes back to 2002. Dobrica and Niemelä[11] presented a survey of eight of the most representative software architecture analysis methods. The authors selected these methods based on the many criteria (Completeness, Accuracy, Repeatability, Scalability, Scalability). The paper has many limitations: very old, some of the methods that the authors survey may no longer be as widely used or as effective as they once were. Also, it does not provide specific guidance on how to choose the right software architecture analysis method for a particular project.

I. Atoum et al. [12] showed that the challenges of software requirements quality assurance SREQ-QA and validation SREQ-V can significantly impact the validation of software architecture. For example, if the requirements are incomplete or inconsistent, it will be difficult to validate the software architecture against those requirements. Similarly, if there is a lack of tools and support for SREQ-V, conducting the necessary validation activities may be difficult. Additionally, human factors, such as communication problems and lack of experience, can lead to errors in the validation of software architecture. The study does not have a more in-depth discussion about the validation.

Rajabli et al.[13] conducted a systematic literature review to identify the state-of-the-art in software V&V for autonomous cars. They identified 79 primary studies that addressed this topic. The limitation of this review, like many others, is that it gives a very specific domain of software architecture validation.

2. Architectural evaluation methods (AEMs)

Architectural evaluation methods (AEMs) are a set of techniques used to assess the quality of a software architecture by considering a number of criteria, such as performance, scalability, security, and maintainability. Typically, AEMs used in the early of software development process, before the architecture is implemented, to identify and address potential problems that may arise.

One of the most popular AEMs is the Architecture Trade-off Analysis Method (ATAM)[7]. ATAM is a method that that involves multiple stakeholders, including architects, developers, testers, and users. They work together to identify the system's key quality attributes of the system and evaluate the extent to which the architecture can meet those requirements. ATAM is often used to evaluate various alternative architectural solutions as well as to assess the extent to which the architecture is sensitive to quality attributes.

Another popular AEM is the Scenario-based Architecture Analysis Method (SAAM)[5]. SAAM uses scenarios, which are descriptions of how the system will be used, to assess architectural quality. The team identified a number of representative scenarios and then uses those scenarios to evaluate the architecture's performance, scalability, security, and used them to evaluate architectural performance, scalability, security, and other quality attributes. SAAM focuses more on evaluating software architectures by leveraging scenarios and can help in identifying issues such as errors, performance bottlenecks, scalability issues, security issues, security weaknesses, and maintenance challenges. A comparison between ATAM and SAAM can be found in Table 1 based on the focus, goals, and approach.

Table 1. ATAM vs. SAAM

Characteristic	ATAM	SAAM
Focus	General-purpose	Scenarios
Goals	Identify and evaluate trade-offs between different architectural solutions, assess the architecture's sensitivity to quality attributes	Identify and evaluate how the architecture performs under different usage scenarios, identify potential problems
Approach	Structured	Structured

Other notable AEMs include:

- Architecture Decision Records (ADRs): ADRs are a way of documenting the rationale behind architectural decisions[14]. ADRs can be used to assess the quality of the architecture by ensuring that the decisions are well-reasoned and that the trade-offs have been carefully considered.
- Architecture Review Board (ARB): An ARB is a group of experts who review the architecture and provide feedback to the architects[8]. ARBs can help identify potential problems with the architecture that the architects may have overlooked.

Ågren et al.[9] discussed the challenges of architecture evaluation in a continuously developing system, especially AEMs. They propose four principles for continuous architecture evaluation.

3. Architecture tests and reviews

Architecture tests and reviews are a set of techniques for evaluating the quality of a software architecture. They are typically used later in the software development process than architectural evaluation methods, after the architecture has been implemented[7].

3.1. Architecture tests

Involve testing the architecture to identify any errors or defects. This can be done through a variety of types of testing[13], [15], including:

- Unit tests: Unit tests are tests of individual units of code. They can be used to indirectly test the architecture by testing the components that make up it[7].
- Integration tests: Integration tests are tests of how different units of code work together. They can be used to test the architecture by testing the interactions between the different components[7].
- System tests: System tests are tests of the entire system as a whole. They can be used to test the architecture by testing how the different components work together to meet the system's requirements[7].
- Performance tests: Performance tests are used to measure the system's performance under load. They can be used to identify performance bottlenecks in the architecture[6].
- Security tests: Security tests are used to identify security vulnerabilities in the architecture [6].

3.2. Architecture reviews

Involve reviewing the architecture with a team of stakeholders to identify any potential problems[7]. The team comprises architects, developers, testers, users, and other stakeholders[15], [16]. Architecture reviews can be conducted in a variety of ways, including:

- Formal reviews: Formal reviews are conducted according to a defined process. They typically involve a team of reviewers who examine the architecture documentation and other artifacts to identify potential problems[7].
- Informal reviews: Informal reviews are more ad hoc. They may be conducted by a single reviewer or by a small team of reviewers. Informal reviews can be used to get feedback on the architecture early in the development process[7].

The following table compares architecture tests and reviews:

Table 2. Architecture tests vs. reviews

Characteristic	Architecture tests	Architecture reviews
Focus	Assessing the quality of the software architecture	Gathering feedback on the software architecture from experts
Strengths	It can be used to assess a variety of aspects of the software architecture, can be automated.	It can be used to identify potential problems with the software architecture early in the development process, can be used to get feedback from experts
Weaknesses	It can be expensive to implement, can be difficult to develop test cases	It can be time-consuming can be difficult to find experts to participate in the review

Alsaqqa et al.[17] propose a new approach to software architecture analysis that considers modern software systems' dynamic and changing nature. The proposed approach is ATR-oriented and based on four principles: continuous monitoring and evaluation of the program design, use of stakeholder feedback, use of evaluation tasks that works automatically, and integration of research results into the development process. The authors suggest that their approach can help improve the quality and maintainability of software systems by early detection of construction problems and enabling prompt corrective action.

4. Model-based validation

A collection of model-based validation (MBV) methods is used to verify software designs[18]. In order to understand a system's behavior and properties, one might utilize models, which are abstract representations of the system[19]. Model-based verification (MBV) techniques use models to find potential issues in the architecture, like poor design, performance snarls, and security holes.

Various phases of the software development process can use MBV approaches. MBV can be used, for instance, to verify early architectural prototypes and find potential issues before they are codified. MBV can also be used to validate current systems, evaluate the effects of architecture modifications, or make sure the system complies with requirements.

Various MBV techniques are available[20]–[23], each with its own strengths and weaknesses. Some common MBV techniques include:

- Model checking is a method for confirming that a model meets a list of requirements[24]. A variety of attributes, including functional correctness, performance constraints, and security requirements, can be verified using model checkers.
- Simulation is a method for running a model and seeing what happens[25]. Simulation can evaluate a system's performance under various load scenarios or spot potential mistakes or design flaws.
- Run-time analysis is a method for keeping tabs on how a system is being used in order to spot any potential issues[26]. Errors, performance snags, and security flaws can all be found via run-time analysis.

In Table 3, we compare the three MBV techniques based on three characteristics: focus, strength, and weaknesses. MBV is very effective and powerful in validating software architecture. Important factors such as system size and complexity, resource availability, and time constraints must be considered to select an MBV technique to suit a particular situation. MBV can also incorporate artificial intelligence (AI) to increase efficiency when performing its tasks[27].

Table 3. The three techniques of MBV

Characteristic	Architecture tests	Architecture reviews
Focus	Verifying that a model satisfies a set of properties	Observing the behavior of a model
Strengths	It can be used to find subtle defects, can be automated	Can be used to validate complex systems, can be used to predict the performance of a system
Weaknesses	It can be expensive to implement, can be difficult to use	It can be time-consuming and difficult to validate the simulation model

5. Discussion

An informative overview of the three main types of software architecture validation, which include architectural evaluation methods, architecture tests and reviews, and model-based validation, is provided in Table 4 for benchmarking purposes.

Table 4. Benchmarking of types of software architecture validation

Characteristic	Architectural evaluation methods	Architecture tests and reviews	Model-based validation
Purpose	To assess the quality of a software architecture against a set of quality attributes, such as performance, security, and maintainability.	To find defects in a software architecture, such as missing requirements, design errors, and architectural inconsistencies.	To identify potential problems in a software architecture using models by verifying that a software architecture satisfies its requirements and constraints.
Strengths	It can be used to evaluate a wide range of architectural properties, such as functional correctness, performance, security, and maintainability.	It can be used to identify errors and defects in the architecture	Can validate architectural properties that are difficult or impossible to test directly and provide quantitative evidence of the validity of a software architecture.
Weaknesses	It can be time-consuming and expensive to conduct	It can be difficult to identify all potential problems in the architecture	It can be expensive to develop and maintain
Maturity	Well-established and widely used.	Less mature than architectural evaluation methods.	Emerging technology, but gaining popularity
Tool support	A variety of tools are available to support architectural evaluation methods, such as SAAM and ATAM.	Limited tool support is available for architecture tests and reviews.	A number of tools are available to support model-based validation, such as MARTE and UML-RT.
Applicability	It can be used to evaluate architectures of all sizes and complexities	Best suited for small to medium-sized software architectures.	It can be used to evaluate architectures of all sizes and complexities but is particularly well-suited for evaluating early-stage architectures
Contribution to the overall validation success	High	Medium	Medium
Dependence on resources	Requires a team of experienced architects and engineers.	Requires a team of experienced testers and architects.	Requires a team of experienced modelers and architects.
Projects particularity	It can be applied to projects of all sizes and security levels.	Best suited for small to medium-sized projects.	Best suited for medium to large-sized projects, especially those with complex security requirements.
Testing Aspect	It can be used to test both functional and non-functional requirements.	It can be used to test both functional and non-functional requirements but is particularly well-suited for testing non-functional requirements.	It can be used to test functional and non-functional requirements but is particularly well-suited for testing them.
Use Cases	Evaluating a new software architecture before it is implemented.	Assessing the quality of an existing software architecture before making major changes.	Verifying that a software architecture satisfies its requirements and constraints in safety-critical systems.

Each form of validation has advantages and disadvantages of its own. Early in the software development cycle, architectural evaluation techniques are frequently employed to assist architects in making knowledgeable choices on the architecture. Later in the software development process, architecture tests and reviews are usually utilized after the architecture has been put in place. It is possible to apply model-based validation at any level of the software development cycle.

Depending on the project's particular requirements, the optimal validation method will be chosen. Architecture reviews might be the best choice, for instance, if the project has a constrained budget and/or a short deadline. Model-based validation might be the best choice if the project wants to validate architectural properties that are challenging or impossible to test directly.

In practice, it is often best to use a combination of different validation techniques to get a more comprehensive assessment of the quality of a software architecture. For example, an architect might use architectural evaluation methods to identify potential problems in the architecture early in the development process and then use architecture tests and reviews to validate the architecture after it has been implemented. Some additional considerations to keep in mind when choosing a software architecture validation technique:

- Size and complexity of the system: The size and complexity of the system will affect the cost and time required to conduct each type of validation.
- Available resources: The resources available, such as budget and personnel, will also affect the choice of validation technique.
- Time constraints: The time constraints on the project will also affect the choice of validation technique.
- Project objectives: The choice of validation technique will also depend on the project's specific objectives. Architecture tests could be the ideal choice, for instance, if the objective is to locate and correct problems in the architecture. Performance testing might be the ideal choice if the objective is to evaluate the architecture's performance.

Ultimately, the best way to choose a software architecture validation technique is to consider all of the relevant factors and select the technique most likely to meet the project's needs. Based on the results of this paper, we recommend the development of a software architecture validation process to work on the following aspects:

- Focus on automation: It will become harder to validate software systems as they get more complicated manually. Automated validation solutions can enhance the validation process's effectiveness and scalability.
- Utilization of machine learning: New and improved validation approaches can be created using machine learning. By analyzing historical data, for instance, machine learning can be used to spot patterns and trends that can be utilized to anticipate prospective issues with software designs.
- Support for emerging technologies: As new software technologies emerge, new validation methodologies will need to be developed to support them. For example, methodologies must be developed to validate the architectures of microservices-based and AI-powered systems.

By implementing these suggestions, we can develop more effective and efficient methodologies for validating software architectures. This will help improve software systems' quality and reliability, especially as they become more complex and challenging to develop.

6. Conclusions

Software architecture validation is an essential step in the software development process, helping to ensure that the architecture of a software system meets the system's requirements and is of high quality. Various software architecture validation techniques are available, each with its own strengths and weaknesses.

The optimum sort of validation to use will be determined by the project's specific needs, such as money, timeframe, and system complexity. For example, architecture reviews may be the greatest option for projects with limited resources and short deadlines. In contrast, model-based validation may be the best option for projects requiring the validation of architectural aspects that are difficult or impossible to test directly. In practice, several validation approaches are frequently employed to assess the software architecture's quality fully.

This paper has succeeded in presenting an overview analysis of software validation methods and providing a useful overview to highlight the strengths and weaknesses of the three main types of software architecture validation. So it can determined the best and most appropriate software validation method based on particular case to minimize losses resulting from software issues.

Author Contributions: Conceptualization: All.; methodology: All.; validation: All.; investigation: A.E.; resources: A.E..; writing—original draft preparation: A.E.; writing—review and editing: A.E.; supervision: A.A.; project administration: All.

Acknowledgments: This work was supported by Abdelmalek Essaadi University.

Conflicts of Interest: The authors declare no conflict of interest.

References

[1] R. F. Schmidt, "Software Architecture," in *Software Engineering*, Elsevier, 2013, pp. 43–54. doi: 10.1016/B978-0-12-407768-3.00003-3.
[2] T. Huckle and T. Neckel, *Bits and Bugs*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2019. doi: 10.1137/1.9781611975567.
[3] C. anton Boiangiu, A. adrian Dinu, M. aura Duican, and R. catalin Raducu, "Comparative Analysis between Mainstream Software Development Methodologies," Apr. 2021, pp. 20–28. doi: 10.12753/2066-026X-21-076.
[4] P. Kruchten, *The rational unified process*, 3rd ed. Boston, MA: Addison-Wesley Educational, 2003.
[5] P. Clements *et al.*, *Documenting software architectures*, 2nd ed. Boston, MA: Addison-Wesley Educational, 2010.
[6] N. Rozanski and E. Woods, *Software systems architecture*, 2nd ed. Boston, MA: Addison-Wesley Educational, 2011.
[7] L. Bass, R. Kazman, and P. Clements, *Software Architecture in Practice*, 3rd ed. Boston, MA: Addison-Wesley Educational, 2012.
[8] J. F. Maranzano, S. A. Rozsypal, G. H. Zimmerman, G. W. Warnken, P. E. Wirth, and D. M. Weiss, "Architecture Reviews: Practice and Experience," *IEEE Softw.*, vol. 22, no. 2, pp. 34–43, Mar. 2005, doi: 10.1109/MS.2005.28.
[9] S. M. Ågren *et al.*, "Architecture evaluation in continuous development," *J. Syst. Softw.*, vol. 184, p. 111111, Feb. 2022, doi: 10.1016/j.jss.2021.111111.
[10] G. Booch, J. Rumbaugh, and I. Jacobson, *The unified modeling language user guide*, 2nd ed. Boston, MA: Addison-Wesley Educational, 2005.
[11] L. Dobrica and E. Niemela, "A survey on software architecture analysis methods," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 638–653, Jul. 2002, doi: 10.1109/TSE.2002.1019479.
[12] I. Atoum *et al.*, "Challenges of Software Requirements Quality Assurance and Validation: A Systematic Literature Review," *IEEE Access*, vol. 9, pp. 137613–137634, 2021, doi: 10.1109/ACCESS.2021.3117989.
[13] N. Rajabli, F. Flammini, R. Nardone, and V. Vittorini, "Software Verification and Validation of Safe Autonomous Cars: A Systematic Literature Review," *IEEE Access*, vol. 9, pp. 4797–4819, 2021, doi: 10.1109/ACCESS.2020.3048047.
[14] O. Kopp, A. Armbruster, and O. Zimmermann, "Markdown architectural decision records: Format and tool support," *CEUR Workshop Proc.*, vol. 2072, pp. 55–62, 2018.
[15] M. Goldstein and I. Segall, "Automatic and Continuous Software Architecture Validation," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, May 2015, pp. 59–68. doi: 10.1109/ICSE.2015.135.
[16] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering – A systematic literature review," *Inf. Softw. Technol.*, vol. 51, no. 1, pp. 7–15, Jan. 2009, doi: 10.1016/j.infsof.2008.09.009.
[17] S. Alsaqqa, S. Sawalha, and H. Abdel-Nabi, "Agile Software Development: Methodologies and Trends," *Int. J. Interact. Mob. Technol.*, vol. 14, no. 11, p. 246, Jul. 2020, doi: 10.3991/ijim.v14i11.13269.
[18] B. Cole, V. Mittal, S. Gillespie, N. La, R. Wise, and A. MacCalman, "Model-based systems engineering: application and lessons from a technology maturation project," *Procedia Comput. Sci.*, vol. 153, pp. 202–209, 2019, doi: 10.1016/j.procs.2019.05.071.
[19] Object Management Group, "OMG Systems Modeling Language (OMG SysML™)," 2018. [Online]. Available: http://www.omg.org/spec/SysML/20161101
[20] A. Bertolino *et al.*, "A Survey of Field-based Testing Techniques," *ACM Comput. Surv.*, vol. 54, no. 5, pp. 1–39, Jun. 2022, doi: 10.1145/3447240.
[21] C. Trubiani, I. Meedeniya, V. Cortellessa, A. Aleti, and L. Grunske, "Model-based performance analysis of software architectures under uncertainty," in *Proceedings of the 9th international ACM Sigsoft conference on Quality of software architectures*, Jun. 2013, pp. 69–78. doi: 10.1145/2465478.2465487.
[22] J. Gonzalez-Huerta, E. Insfran, S. Abrahão, and G. Scanniello, "Validating a model-driven software architecture evaluation and improvement method: A family of experiments," *Inf. Softw. Technol.*, vol. 57, pp. 405–429, Jan. 2015, doi: 10.1016/j.infsof.2014.05.018.
[23] G. Jacobs, C. Konrad, J. Berroth, T. Zerwas, G. Höpfner, and K. Spütz, "Function-Oriented Model-Based Product Development," in *Design Methodology for Future Products*, Cham: Springer International Publishing, 2022, pp. 243–263. doi: 10.1007/978-3-030-78368-6_13.
[24] E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model Checking*, 2nd ed. Carnegie Mellon University; Technion; Oxford University; Bar Ilan University; Technische Universitaet Darmstadt: MIT Press, 2018.
[25] G. S. Fishman, *Discrete-Event Simulation*. New York, NY: Springer New York, 2001. doi: 10.1007/978-1-4757-3552-9.

[26] A. Francalanza, J. A. Pérez, and C. Sánchez, "Runtime Verification for Decentralised and Distributed Systems," 2018, pp. 176–210. doi: 10.1007/978-3-319-75632-5_6.

[27] B. Uzun and B. Tekinerdogan, "Model-driven architecture based testing: A systematic literature review," *Inf. Softw. Technol.*, vol. 102, pp. 30–48, Oct. 2018, doi: 10.1016/j.infsof.2018.05.004.