Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 43 No. 4 (2022)

Review of Software Architecture and Design Practices: Current Trends and Future Direction

[1] Kajal Sharma, [2] Naveen Kumar Yadav, [3] Aishwarya Maloo, [4] Shivam Verma

[1] Asst. Professor
Dept. of Management
Arya Institute of Engineering and Technology, Jaipur
[2] Asst. Professor
Mechanical Engineering
Arya Institute of Engineering Technology & Management, Jaipur
[3] Science Student
Sampoorna Kendra Vidyalaya, Dibrugarh, Assam.
[4] Research Scholar
Arya Institute of Engineering and Technology, Jaipur

Abstract: Software structure and design are foundational components of software program engineering, shaping the shape, capability, and maintainability of software program structures. This evaluates paper provides a comprehensive exam of the modern-day country of software program architecture and design practices, tracing their historical evolution, elucidating fundamental standards, and discussing key components, principles, and styles that underpin effective software design. The paper delves into modern tendencies, together with micro services, server less computing, and occasion-pushed architectures, highlighting their advantages and demanding situations. It explores diverse architectural patterns and processes, presenting insights into their suitability for exclusive mission contexts. Tools, frameworks, and visualization strategies for software program structure and design also are assessed.
Through this overview, we purpose to offer a comprehensive resource for software engineers, researchers, and practitioners, supporting them navigate the dynamic panorama of software program structure and layout, make knowledgeable choices, and envision a future wherein software structures are extra resilient, scale-able, and adaptable to evolving technological demands.
Keywords: Modularity, performance, software architecture, software design, scalability, modularity.

1. Introduction

Software has come to be a critical a part of current existence, powering everything from smartphones to commercial machinery, from monetary systems to healthcare solutions. Behind every piece of software lies a complicated net of architectural choices and design choices that determine its functionality, maintainability, and adaptable-ness. Software architecture and layout, essential pillars of software program engineering, have evolved appreciably over the years, shaping the manner software systems are conceived, structured, and advanced. The importance of software program architecture and design can't be overstated. Effective architectural selections can lead to structures that scale gracefully, are tremendously maintainable, and provide strong overall performance. Thoughtful layout selections can decorate software program's usability, extensibility, and security. However, the panorama of software architecture and layout isn't static; it constantly evolves in response to emerging technology, changing user expectations, and new paradigms of software program improvement. This research paper embarks on a journey into software architecture design, providing a comprehensive understanding of the current state of practice and trends shaping the field We explore the historical roots of software architecture and design, tracing their evolution background, from the early days of computers to gifts. We delve into the key concepts that underpin effective architectural design decisions, from modularity and scalability to maintainability and security. The paper also shines some light on modern trends that are reshaping the software landscape. Micro-service architecture, server low compute, and event-driven systems have emerged as powerful paradigms, offering new ways to design and design software solutions We examine the benefits and challenges associated with each of these elements, helping professionals to they make the right choices in their software business. Real-world case studies show how architectural choices influence the outcomes of software projects, providing valuable lessons from successful implementations. Finally, we cast our gaze toward the future, speculating on the evolution of software architecture and design in the face of emerging technologies like artificial intelligence and the Internet of Things (IoT). We identify areas ripe for future research and innovation, envisioning a landscape where software systems are more resilient, scale-able, and adaptable than ever before. In essence, this review paper serves as a comprehensive resource for software engineers, researchers, and practitioners, guiding them through the ever-evolving landscape of software architecture and design. It equips them with the knowledge needed to make informed decisions, embrace current best practices, and prepare for the exciting challenges and opportunities that lie ahead in the field of software engineering.

2. Literature Review

Fundamental Concepts in Software Architecture and Design:

1. Modularity: Modularity is the exercise of breaking down a software gadget into smaller, self-contained modules or components. These modules can be evolved, tested, and maintained independently, making the gadget extra potential and scalable.
2. Scalability: Scalability refers to a system's capability to address elevated workloads and develop in potential without compromising performance. Architectural choices, such as load balancing and distributed computing, play a critical position in achieving scalability.
3. Security: Security is a paramount issue in software program design. It involves protecting software program systems from unauthorized get right of entry to, information breaches, and vulnerabilities. Security measures need to be included into the architecture from the floor up.
4. Performance: Performance issues involve optimizing a software program system to fulfill particular speed and performance necessities. This consists of optimizing algorithms, records systems, and aid utilization.

Key Components of Software Architecture:

1. Modules/Components: Modules are the constructing blocks of a software gadget, representing discrete, functional gadgets of code. They may be organized hierarchically and speak with each different via defined interfaces.
2. Layers: Layered structure separates a device into horizontal layers, with each layer answerable for a specific set of functions. Common layers include presentation, enterprise common sense, and information get right of entry to layers.
3. Architectural Patterns: Architectural styles provide high-level templates for fixing recurring layout issues. Examples encompass Model-View-Controller (MVC), Micro services, and Event-Driven Architecture.
4. Interfaces: Interfaces outline the contracts and communication factors between additives, permitting them to interact without exposing their inner information.
5. Data Storage: Architectural choices concerning statistics storage, consisting of databases, caching mechanisms, and statistics warehouses, are vital for records-pushed applications.

Modern Software Architecture Trends:

1. Micro services: Micro offerings architecture decomposes a software program system into small, impartial offerings that may be developed, deployed, and scaled in my view. It promotes flexibility and allows for faster improvement cycles.
2. Server much less Computing: Server less architecture abstracts away server management, enabling developers to cognizance on writing code (features) without disturbing about infrastructure. It scales routinely and is price-effective.
3. Event-Driven Architecture: Event-pushed architecture is based on occasions and messages to cause actions and conversation between components. It's well-proper for real-time and reactive systems, like IoT applications.

4. Containerization: Containers, such as Docker, package deal programs and their dependencies into an unmarried unit, making deployment and scaling greener. Kubernetes is a famous box orchestration tool.
5. Edge Computing: Edge computing brings computing resources towards the statistics source or quit-customers, reducing latency and allowing actual-time processing. It's essential for applications like IoT and autonomous motors.
6. Block chain and Distributed Ledgers: Block chain technology is used to create decentralized and tamper-resistant structures. It's gaining traction in industries which includes finance, supply chain, and healthcare.
7. Understanding those essential concepts, key components, and current tendencies in software program structure is essential for making knowledgeable architectural decisions and preserving software program systems applicable and competitive in state-of-the-art fast-paced technological landscape.

Tools and Technologies:

1. Modelling and Design Tools:
   - Unified Modelling Language (UML): UML is a standardized modelling language used for visualizing, specifying, constructing, and documenting software program structures. Tools like IBM Rational Rose, Enterprise Architect, and Visual Paradigm provide UML help.
   - Lucid chart: Lucid chart is a cloud-based diagramming tool that gives UML diagram templates and collaboration functions for designing software architectures.
   - Draw. Io: An open-supply diagramming device that helps various diagram kinds, together with UML, flowcharts, and network diagrams.
2. Version Control Systems:
   - Git: Git is a distributed version manipulate machine extensively used for handling supply code, branching, and collaboration among developers. Platforms like GitHub, Git Lab, and Bit bucket offer Git repository website hosting offerings.
3. Integrated Development Environments (IDEs):
   - Eclipse: Eclipse is an open-source IDE that helps multiple programming languages and offers tools for software development, which include code modifying, debugging, and modelling.
   - Visual Studio: Microsoft's Visual Studio is a powerful IDE that supports various programming languages and provides features for constructing, debugging, and profiling applications.

4. Future Scope

1. Human-Computer Interaction (HCI) and UserCentric Design: Software architects will need to collaborate closely with HCI specialists to layout architectures that provide first rate person experiences, thinking about elements together with accessibility, usability, and user remarks.
2. Evolutionary Architectures: Architectures that can adapt and evolve over time may be in call for. This consists of dynamic reconfiguration, self-recovery systems, and architectures that may accommodate converting commercial enterprise needs.
3. Ethical Software Design: There will be an increased consciousness on ethical considerations in software program structure and design, which includes addressing biases in algorithms, making sure data privates, and adhering to moral ideas in AI and automation.
4. Quantum Computing and Software Design: With the improvement of quantum computing, software program architects will face new demanding situations and opportunities in designing algorithms and structures which can harness the power of quantum computer systems for complicated problem-solving.AI-Driven Architectural Decision Support: Artificial intelligence and device getting to know will increasingly be used to research and optimize software program architectures. AI can assist architects in making statistics-pushed choices, predicting potential issues, and recommending architectural patterns and solutions.

5. Sustainable Software Architecture: Sustainable layout standards will be implemented to software structure, considering energy performance and decreasing the environmental effect of software program structures. This includes designing for low energy intake and optimizing useful resource usage.

5. Conclusions

The subject of software program structure and layout is at the forefront of innovation and transformation inside the ever-evolving panorama of technology. In this evaluate, we've journeyed via its historical roots, essential concepts, key additives, and cutting-edge tendencies, losing light on the dynamic nature of this crucial subject. Fundamental ideas, inclusive of modularity, scalability, maintainability, security, and performance, continue to be the guiding concepts for architects and architects. These principles function the bedrock upon which resilient and adaptable software program systems are constructed. Key components, such as modules/components, layers, architectural patterns, interfaces, facts garage, and communique protocols, provide the tools and frameworks vital for architects to craft state-of-the-art and green architectures. These components empower them to deal with complex design challenges and meet the various needs of modern software program packages. Modern software structure trends have emerged as answers to current demands. Micro services, server much less computing, occasion-pushed architectures, and containerization have redefined how software program structures are conceived and orchestrated. These trends offer flexibility, scalability, and agility, permitting businesses to respond to unexpectedly converting technological landscapes and consumer expectations. In this ever-evolving discipline, architects will include automated structure generation, evolutionary architectures, and the non-stop quest for innovation. The position of software architects will remain pivotal in crafting software program structures that now not best meet purposeful requirements but additionally address moral, environmental, and person-centric issues. In conclusion, software structure and design stand as cornerstones inside the realm of software program engineering. They navigate the tricky interplay of technology, user needs, and moral concerns, shaping a future wherein software structures are not just green and scalable however additionally moral, sustainable, and person-centric.

References

[1] Babar, M.A., Dingsøyr, T., Lago, P., Vliet, H.v. (Eds.), 2009, Software Architecture Knowledge Management: Theory and Practice. Springer-Verlag, Berlin, Heidelberg.

[2] Babar, M.A., Boer, R.C.d., Dingsøyr, T., Farenhorst, R., 2007. Architectural knowledge management strategies: approaches in research and industry. In: Proceedings of Second Workshop on SHAring and Reusing architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI 2007).

[3] Babar, M.A., Gorton, I., 2007. A tool for managing software architecture knowledge. In: Proceedings of the 2nd Workshop on Sharing and Reusing Architectural Knowledge (ICSE Workshops).

[4] Babar, M.A., Gorton, I., Kitchenham, B., 2006. A framework for supporting architecture knowledge and rationale management. In: Dutoit, A.H., McCall, R., Mistrik, I., Paech, B. (Eds.), Rationale Management in Software Engineering.

[5] Springer, pp. 237– 254. Baldwin, C.Y., 2010. When Open Architecture Beats Closed: The Entrepreneurial Use of Architectural Knowledge. Harvard Business School, Massachusetts. Becker, C., 2014.

[6] Sustainability and longevity: two sides of the same quality? Mental vol. 20, 21. Berg, M.v.d., Tang, A., Farenhorst, R., 2009. A constraint-oriented approach to software architecture design. In: Proceedings Of the Quality Software International Conference (QSIC 2009), pp. 396–405.

[7] Bonnema, G.M., 2014. Communication in multidisciplinary systems architecting. Procedia CIRP 21, 27–33. Borches, P.D., Bonnema, G.M., 2010. A3 architecture overviews: focusing architectural knowledge to support evolution of complex systems.

[8] 20th Annual International Symposium of INCOSE, Chicago (USA), July 12–15. Bosch, J., 2004. Software architecture: the next step. In: Proceedings of Software Architecture: First European Workshop, EWSA 2004.

[9] St Andrews, UK, pp. 194–199. Burge, J., 2005. Software Engineering Using Design RATionale.

Doctor of Philosophy, Computer Science, Worcester Polytechnic Institute.

[10] Burge, J.E., Brown, D.C., 2008. SEURAT: integrated rationale management. In: Proceedings of the 30th International Conference on Software Engineering, pp. 835–838. Capilla, R., 2009. Embedded design rationale in software architecture.

[11] In: Proceedings of Joint Working IEEE/IFIP Conference on presented at the Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Capilla, R., Nava, F., Carrillo, C., 2008. Effort estimation in capturing architectural knowledge.

[12] Proceedings of the Automated Software Engineering (ASE'08), pp. 208–217. Capilla, R., Nava, F., Pérez, S., Dueñas, J.C., 2006. A web-based tool for managing architectural design decisions.

[13] In: Proceedings of the 1st Workshop on Sharing and Reusing Architectural Knowledge. Capilla, R., Zimmermann, O., Zdun, U., Avgeriou, P., Küster, J.M., 2011. An enhanced architectural knowledge metamodel linking architectural design decisions to other artifacts in the software engineering lifecycle. Software Architecture.

[14] Springer, pp. 303–318. Chen, L., Babar, M.A., Liang, H., 2010. Model-centered customizable architectural design decisions management. In: Proceedings of 21st Australian Software Engineering Conference (ASWEC), 2010, pp. 23–32.

[15] Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., et al., 2011. Documenting Software Architectures: Views and beyond, 2nd ed. Addison Wesley. Clements, P., Shaw, M., 2009. "The golden age of software architecture" revisited.

[16] Sharma, R., Kaushik, M. and Kumar, G. (2015) "Reliability analysis of an embedded system with multiple vacations and standby", International Journal of Reliability and Applications, Vol. 16, No. 1, pp. 35-53.

[17] Kaushik, M. and Kumar, G. (2015) "Markovian Reliability Analysis for Software using Error Generation and Imperfect Debugging" , International Multi Conference of Engineers and Computer Scientists 2015, vol. 1, pp. 507-510.

[18] Jain, B.B., Upadhyay, H. and Kaushik, R., 2021. Identification and Classification of Symmetrical and Unsymmetrical Faults using Stockwell Transform. Design Engineering, pp.8600-8609.

[19] T. Manglani, A. Vaishnav, A. S. Solanki and R. Kaushik, "Smart Agriculture Monitoring System Using Internet of Things (IoT)," *2022 International Conference on Electronics and Renewable Systems (ICEARS)*, Tuticorin, India, 2022, pp. 501-505.

[20] R. Kaushik *et al*., "Recognition of Islanding and Operational Events in Power System With Renewable Energy Penetration Using a Stockwell Transform-Based Method," in *IEEE Systems Journal*, vol. 16, no. 1, pp. 166-175, March 2022.