Unit 1: Introduction to Secure Software Development

Required Reading

Core Text: Pillai, A.B. (2017) Software Architecture with Python. Birmingham, UK. Packt Publishing Ltd.

- Chapter 1.
- Chapter 6.

Chapter 1:
Chapter 1 of the book introduces the fundamental principles of software architecture, particularly focusing on the Python programming language. It emphasizes the importance of quality attributes such as Modifiability, Testability, Scalability/Performance, Security, and Deployability. The chapter defines these attributes and discusses their interrelations, providing a foundation for understanding how they impact software design and architecture. It also outlines the different levels of changes that can occur in software architecture—local, non-local, and global—highlighting the varying degrees of risk and complexity associated with each type of change. The chapter sets the stage for a deeper exploration of these quality attributes in subsequent chapters.
Reflection:
Chapter 1 effectively establishes the groundwork for understanding software architecture by clearly defining key concepts and quality attributes. The discussion on the levels of changes provides valuable insights into the complexities architects face when modifying systems. This chapter resonates with the challenges many developers encounter, as it underscores the necessity of considering architectural quality attributes in the design process. The focus on Python as a context for these principles makes the content relatable and applicable for practitioners in that ecosystem.
Chapter 6:
Chapter 6 focuses on the importance of writing secure code and implementing robust security practices in software architecture. It outlines various strategies to mitigate vulnerabilities, such as validating inputs, keeping designs simple, applying the principle of least privilege, and sanitizing data. The text emphasizes the need for effective quality assurance, threat modeling, and layered security approaches to protect applications from attacks like Denial of Service (DoS) and Cross-Site Scripting (XSS). Additionally, it highlights the significance of defining security requirements early in the development lifecycle and maintaining consistent security policies across systems.
Reflection:
The insights provided in Chapter 6 are crucial for anyone involved in software development. As cyber threats continue to evolve, the need for secure coding practices becomes increasingly urgent. The strategies outlined serve as a comprehensive guide for developers to proactively address security concerns, rather than reacting to breaches after they occur. By fostering a culture of security awareness and implementing these best practices,

development teams can significantly reduce the risk of vulnerabilities in their applications. This proactive approach not only protects sensitive data but also enhances user trust and the overall integrity of software systems.

Asghar, M. Z., Alam, K. A. & Javed, S. (2019) Software Design Patterns Recommendation : A Systematic Literature Review, 2019 International Conference on Frontiers of Information Technology.

The document presents a systematic literature review (SLR) on software design patterns recommendation, focusing on studies published between 2010 and 2019. The authors, Muhammad Zeeshan Asghar, Khubaib Amjad Alam, and Shahzeb Javed from the National University Fast in Islamabad, Pakistan, aim to analyze existing research on design pattern recommendation techniques, identify gaps, and summarize findings.
Design patterns are established solutions to recurring problems in software development, enhancing reusability, modularization, and consistency. However, selecting the appropriate design pattern can be challenging, especially for less experienced developers. The review identifies 22 relevant studies, highlighting that techniques such as anti-pattern detection and fuzzy methods are prevalent, while traditional recommendation techniques remain underutilized.
The research is structured into sections detailing related work, research methodology, and findings. Key research questions focus on the techniques used for recommendation, contributions to the field, and overall productivity in research. The findings indicate a decline in research activity in the last two years of the review period, suggesting a significant gap in ongoing studies.
Reflection
This literature review underscores the importance of design patterns in software engineering and the necessity for effective recommendation systems to aid developers in selecting appropriate patterns. The identification of a research gap, particularly in the last two years, raises concerns about the stagnation in this area, suggesting a need for renewed focus and innovation in design pattern recommendation techniques.
The reliance on specific methodologies, such as fuzzy c-means and anti-pattern detection, while beneficial, indicates a potential limitation in exploring a broader range of traditional recommendation strategies. Future research could benefit from integrating diverse techniques, including content-based and collaborative filtering approaches, to enhance the effectiveness of design pattern recommendations. Overall, the study serves as a valuable resource for both researchers and practitioners, providing insights into current methodologies and highlighting areas for future exploration.

Royce, W. W. (1970) Managing the Development of Large Software Systems.

Dr. Winston W. Rovce discusses his insights on managing large software development projects, drawing from his extensive experience in spacecraft mission software. He emphasizes that successful software development requires more than just analysis and coding; it necessitates a structured approach that includes requirements analysis, program design, testing, and thorough documentation. Rovce outlines five critical steps to mitigate risks in software development:

1. Program Design Comes First: A preliminary design phase should precede detailed analysis to ensure that storage, timing, and operational constraints are considered early on.

2. Document the Design: Comprehensive documentation is essential for effective communication among team members and for maintaining clarity throughout the development process.
3. Do It Twice: Developing a pilot model or simulation allows for early identification of potential issues, ensuring that the final product is more robust.
4. Plan, Control, and Monitor Testing: Testing is the most resource-intensive phase, and it should be managed carefully, utilizing specialists and thorough checks to identify errors.
5. Involve the Customer: Engaging the customer throughout the development process helps align expectations and reduces the risk of misinterpretation of requirements.

Rovce concludes that while these steps may incur additional costs, they are necessary to avoid the far greater expenses associated with recovering from failures in large software projects.

Reflection:

Rovce's insights resonate strongly in today's software development landscape, where complexity and scale continue to grow. His emphasis on early design and thorough documentation highlights the importance of proactive management in mitigating risks. The iterative nature of software development, as described, underscores the need for flexibility and adaptability in response to unforeseen challenges.

Moreover, involving the customer throughout the process is a crucial reminder that software is ultimately built for users, and their input can significantly enhance the final product's relevance and usability. As software development methodologies evolve, the principles outlined by Rovce remain relevant, serving as a foundation for best practices in managing large-scale projects. His approach advocates for a balance between creativity and structured processes, which is essential for achieving successful outcomes in complex software systems.

OMG (2017) Unified Modelling Language, Version 2.5.1.

The Unified Modeling Language (UML) v2.5.1 is a comprehensive modeling language standard developed by the Object Management Group (OMG). It provides a unified framework for visualizing, specifying, constructing, and documenting the artifacts of software systems. UML 2.5.1 includes essential updates and structural refinements, ensuring clarity in syntax and semantics across its modeling diagrams. This version covers elements like classes, states, activities, interactions, and components to support diverse system development needs.

Reflection:

UML v2.5.1's structured approach greatly benefits software design by fostering clear, organized, and scalable model representations. Its detailed semantics and visual notations provide both consistency and flexibility, making UML a valuable tool for both complex systems architects and developers. By adhering to UML standards, teams can maintain interoperability and enhance communication, ultimately reducing ambiguity and errors across the software development lifecycle

Unified Modelling Homepage

Mitre (2017) Weaknesses in OWASP Top Ten.

The "Weaknesses in OWASP Top Ten (2017)" document highlights critical security vulnerabilities in web applications, outlining categories such as injection flaws, broken authentication, sensitive data exposure, XML external entities (XXE), and cross-site

scripting (XSS). These vulnerabilities, if left unaddressed, can lead to severe security breaches, underscoring the importance of implementing secure coding practices and proactive system monitoring.
Reflection:
Serving as both a technical resource and an educational tool, this guide is intended for developers, product customers, and educators. It provides not only the foundational knowledge needed to recognize and prioritize security weaknesses but also actionable insights that can significantly reduce risks and improve application resilience. By adhering to the OWASP Top Ten guidelines, stakeholders can establish more secure development practices, ultimately enhancing software reliability and protecting against prevalent threats

Pohl, C. & Hof, H-J. (2015) Secure Scrum: Development of Secure Software with Scrum, in Proc. of the 9th International Conference on Emerging Security Information, Systems and Technologies.

The paper "Secure Scrum: Development of Secure Software with Scrum" by Christoph Pohl and Hans-Joachim Hof presents an extension of the Scrum framework called Secure Scrum, which focuses on integrating security considerations throughout the software development process. Traditional software development often requires detailed security planning upfront, which can be challenging in agile methodologies like Scrum that prioritize rapid development and flexibility. Secure Scrum addresses this by introducing four key components: identification, implementation, verification, and definition of done, all designed to enhance security awareness among developers, even those without specialized security expertise.
The identification component helps teams mark security-relevant user stories in the Product Backlog, while the implementation component ensures that security issues remain visible during sprints. The verification component focuses on testing security aspects, and the definition of done component incorporates security requirements into the completion criteria for tasks. The paper also discusses the integration of external security resources when necessary.
A field test involving 16 developers demonstrated that Secure Scrum not only improved the security level of the developed software but was also easy to understand and implement, even for teams with limited security knowledge. The results indicated that teams using Secure Scrum identified more security-related requirements and produced software with significantly fewer vulnerabilities compared to those using standard Scrum or no structured approach.
Reflection:
The development of Secure Scrum is a significant step toward addressing the growing need for security in software development, particularly in agile environments. As software becomes increasingly integral to everyday life, the potential for security vulnerabilities poses serious risks. The findings from the evaluation highlight that even novice developers can effectively incorporate security practices into their workflow when provided with the right tools and frameworks.
This approach not only enhances the security of the software but also fosters a culture of security awareness within development teams. The use of S-Tags to annotate security concerns is particularly innovative, as it keeps security at the forefront of the development process without overwhelming the team with additional complexity. Overall, Secure Scrum represents a practical solution for integrating security into agile methodologies, ensuring that security is not an afterthought but a fundamental aspect of software development.

ISO/IEC (2018) ISO/IEC Standard 27000 Section 3.

Additional Reading

Fowler, M. (2006) Code Smell.

Mougouei, D., Sani, F. M., & Almasi, M. (2013) S-Scrum: a Secure Methodology for Agile Development of Web Service, Computer Science and Information Technology.

Wichers, D. (2008) Breaking the Waterfall Mindset of the Security Industry. In: OWASP AppSec USA, New York.

Manadhata, P. & Wing, J. M. (2004) Measuring a System's Attack Surface, Carnegie Mellon.

Ofcom (2019) O2 Network Outage.

IBM (2015) 10 Essential Security Practices from IBM.

Open Group (2016) The SOA Source Book.

Casteren, W. Van. (2017) The Waterfall Model and the Agile Methdologies: A Comparison by Project Characteristics.