Exploring the Connection between Design Smells and Security Vulnerabilities

Mamdouh Alenezi, Mohammed Akour

***Abstract*: Software quality aims at having quality as part of all aspects of the developed software. Design smells are considered enemies of the software source code quality. There are verities of design problems with different terminologies. Researchers and practitioners accept it as true that whenever there is a design smell, there is a security issue or concern. In this work, we want to explore the connection between design smells and security vulnerabilities. This work provides experimental evidence about this connection. We conducted an empirical study to explore the connection between design smells and security issues by evaluating four C# open-source systems. We found interesting results that show classes with design smells have more chances of having security issues**.**

***Keywords*: Design Smell, Empirical Study, Software Evolution.**

I. INTRODUCTION

Software systems are continually evolving, requiring constant maintenance and development. In this context, many e-services have been made to find approaches that can detect source code fragments that are difficult to maintain or are more likely to have security issues. Source code fragments that contain design smells usually hurt the quality attributes such as maintainability and flexibility [6].

Finding security vulnerabilities is a very important task where researchers have tried different techniques and examined different correlations to make the task more efficient. Security is a software property just as correctness or efficiency. However, correctness and efficiency can be easily measured whereas it is very difficult to measure security directly. Design smells have been found to be strong symptoms of poor design and implementation decisions [6]. These smells are indicators for refactoring needs, which can serve as a proxy for design quality. The investigation of connections between code smells and vulnerabilities can shed light on our understanding of vulnerabilities and their causes.

Design smells or code smells [6] in the source code that can give indications of problems in the design can be solved by refactoring [6]. Design smells represent principles of violations design, making the software difficult to understand, maintain, and evolve [3]. Although the concept of design smells is used to assess the software design, there is little empirical evidence relating design smells with important software quality attributes, such as maintenance effort and security issues [9]. One of the pieces of evidence was given by D'Ambros et al. [3] by showing the emergence of design smells classes of software systems over the cycle contributed to generate security issues.

Li and Shatnawi [9] have also studied the relationship between design smells and the likelihood of security issues. Some design smells were positively associated with the likelihood of security issues in classes. However, some other studies have refuted the idea of accepting the design smells indicators of potential problems as the design [1, 18, 21]. Olbrich et al. [13] For example, evaluated the effect of two design smells (God Class and brain class) with security issues. The authors concluded that design smells are not necessarily harmful. Most studies are studied a few designs smells, such as God Class and Brain Class, while other design smells have not been studied much [23]. Therefore, it is necessary to conduct more studies on the impact of design smells from a security perspective.

The overall goal of our work is to explore the connection between design smells and security issues. We conducted an experimental study of four C# open-source projects. The article is organized as follows: Section 2 discusses related work. Section 3 discusses the aim of the study and research questions. Section 4 discusses the results. Section 5 concludes the paper.

II. RELATED WORKS

Researchers have studied Design smells and how they can be detected. Design smells are potential causes of future problems. The term was first used by Fowler [6] in his prominent work of refactoring where he cataloged potential problems in designing software that caused long-term problems. The author called these pieces of software as points of immediate refactoring. It is worth mentioning that these refactoring should be made so that the internal structure of the system is improved, however, without any external change, and consequently the functionalities are changed. They can be identified through the use of rules-based software as a means of quality intrigues, known as detection strategies [7, 11].

Mumtaz et al. [8] studied the lifespan of code smells in seven open-source systems. They have found that smells are removed because of maintenance requests. The commits that changed these smells are not specifically for these smells. This shows that smells can last for a long time in software systems. Palomba et al. [10] tried to improve bug-predication by studying the intensity of smells.

Revised Manuscript Received on May 20, 2020.

***** Correspondence Author

Mamdouh Alenezi, College of Computer and Information Sciences, Prince Sultan University, Riyadh, Saudi Arabia. E-mail: malenezi@psu.edu.sa

Mohammed Akour*, Al Yamamah University, Riyadh, Saudi Arabia, Yarmouk University, Irbid, Jordan. E-mail: mohammed.akour@yu.edu.jo

Javid et al. [28] studied the relationship between complexity and software vulnerabilities, and their experiment reveals a strong connection between these two factors. The experiments are conducted by utilizing multiple classifiers and multiple feature selection algorithms. Moreover, they addressed the factors that might make the software more vulnerable

Akour and Alsmadi [29] and Alrawais et al. [30] assure the importance of conducting vulnerability assessment in a frequent matter. As their studies reveal how a large number of applications suffer from a wide variety of attacks which lead to financial losses. In their works a security-testing framework for web applications, networks, and computer infrastructure is proposed with an argument that the security of an application should be tested at every stage of the software development life cycle (SDLC).

III. Purpose and Research Issues

This work aims to study the relationship between design smells and the occurrence of security issues in software systems. Therefore, a general question of research that aims to achieve this goal has been defined as follows:

• RQ: The relationship between design smells and security issues?

Design smells are used to identify optical problematic classes in object-oriented systems. Some studies [3, 9] show that feature classes that design smells They are more likely to contain security issues, other classes. However, Olbrich et al. [13] concluded that design smells not necessarily are harmful. Therefore, from this research question, we evaluate the existence of this relationship, comparing the occurrence of security issues into classes design smells and classes without design smells.

Zhang et al. [24] For example, investigated the relationship between six design smells Fowler [6] (Duplicated Code, Data Clumps, Switch Statements, Speculative generality, Message Chains, and Middle Man) and security issues software. The study results showed that the source codes containing Duplicated Code. They are more associated with more security issues the other design smells evaluated. However, this result is considered a small group of design smells and despises other recurring software systems such as God Class and Feature Envy. Therefore, from this research question, we identified the design smell with a higher incidence of security issues, comparing the ratio of security issues in classes affected by each type of design smell under study. The result of this analysis might be used to help developers prioritize refactoring.

Description of selected software systems, with name and structural size information in terms of a number of lines of code (LOC), methods, and classes are in Table 1.

Table 1. Selected open-source C# software systems

System	Version	# of Classes	# of Methods	LOC
ConfigR	1.0.0	125	608	1560
Shadowsocks	4.0.6	117	759	11241
Wexflow	2.0	86	302	6177
Scripty	0.7.4	59	432	4856

Design smells are design system structures, which indicate the compromise of fundamental design principles. They are indicators of poor design quality, which negatively affect design quality [25]. Designite [26] is a software quality assessment tool that comprehensive support detecting architectural and design smells. The tool used in our empirical analysis [26] can detect nineteen design smells. The connection between security and software design has been highlighted in recent years [27]. There is a still need to investigate this relation empirically on real systems.

1. Results

In this section, we discuss the results of our experiments. We report the relationship between design smells and security issues. The following tables report the design smells frequency along with the frequency of security issues.

Table 2. Results of the ConfigR system

Smell	Frequency	Security Issues
Duplicate Abstraction	45	3
Imperative Abstraction	28	1
Unnecessary Abstraction	3	0
Unutilized Abstraction	53	0
Broken Modularization	1	0
Insufficient Modularization	8	0
Rebellious Hierarchy	6	3
Wide Hierarchy	2	0

Table 2 shows the results of the ConfigR system. Security issues are related to 'Duplicate Abstraction' and 'Rebellious Hierarchy'. We found only six security issues that are not part of any of the studied design smells.

Table 3. Results of the Scripty system

Smell	Frequency	Security Issues
Imperative Abstraction	3	0
Unnecessary Abstraction	11	1
Deficient Encapsulation	3	2
Broken Modularization	9	3
Rebellious Hierarchy	4	2
Cyclic Hierarchy	1	1
Broken Hierarchy	1	0

Table 3 shows the results of the Scripty system. Security issues are related to 'Deficient Encapsulation', 'Broken Modularization', and 'Rebellious Hierarchy'. We found only two security issues that are not part of any of the studied design smells.

Table 4. Results of the Shadowsocks system

Smell	Frequency	Security Issues
Duplicate Abstraction	2	0
Imperative Abstraction	6	0
Unnecessary Abstraction	12	2
Unutilized Abstraction	9	1
Deficient Encapsulation	44	9
Broken Modularization	5	1
Cyclically-dependent Modularization	8	3
Insufficient Modularization	5	0
Broken Hierarchy	1	0
Rebellious Hierarchy	7	2
Unfactored Hierarchy	4	0

Table 4 shows the results of the Shadowsocks system. We found only three security issues that are not part of any of the studied design smells.

Table 5. Results of the Wexflow system

Smell	Frequency	Security Issues
Duplicate Abstraction	1	0
Imperative Abstraction	4	0
Unnecessary Abstraction	2	0
Unutilized Abstraction	7	1
Deficient Encapsulation	4	2
Unexploited Encapsulation	4	2
Cyclically-dependent Modularization	2	2
Insufficient Modularization	2	0
Wide Hierarchy	1	0

Table 5 shows the results of the Wexflow system. We found only three security issues that are not part of any of the studied design smells. We can see from the previous tables that there is an obvious connection between design smells and security issues. The number of security issues that are not associated with design smells is very small compared to the number of security issues that are associated with security issues. This shows some evidence that there is a great relationship between design smells (low code design quality) with security issues in the software system. The most recurring design smells that have connections with security issues are 'Deficient Encapsulation', 'Duplicate Abstraction', 'Rebellious Hierarchy', and 'Cyclically-dependent Modularization'.

IV. CONCLUSION

In this paper, we discuss the connection between design smells and security issues. In literature, there are very few studies that discuss the impact of design smells on software qualities especially security. Hence, identifying the research gap we conducted experiments to investigate the impact of design smells and security issues. For this purpose, in this work, we selected four open-source C# systems. The tool that we used to investigate the design smells was able to identify nineteen design smells within the open-source software system.

REFERENCES

1. Anda, B. (2007, October). Assessing software system maintainability using structural measures and expert assessments. In *2007 IEEE International Conference on Software Maintenance* (pp. 204-213). IEEE.
2. Cohen, J. (2013). *Statistical power analysis for the behavioral sciences*. Routledge.
3. D'Ambros, M., Bacchelli, A., & Lanza, M. (2010, July). On the impact of design flaws on software defects. In *2010 10th International Conference on Quality Software* (pp. 23-31). IEEE.
4. D'Ambros, M., Lanza, M., & Pinzger, M. (2007, June). " A Bug's Life" Visualizing a Bug Database. In *2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis* (pp. 113-120). IEEE.
5. Fontana, F. A., Braione, P., & Zanoni, M. (2012). Automatic detection of bad smells in code: An experimental assessment. *Journal of Object Technology*, *11*(2), 5-1.
6. Fowler, M. (2018). *Refactoring: improving the design of existing code*. Addison-Wesley Professional.
7. Khomh, F., Di Penta, M., & Gueheneuc, Y. G. (2009, October). An exploratory study of the impact of code smells on software change-proneness. In *2009 16th Working Conference on Reverse Engineering* (pp. 75-84). IEEE.
8. Mumtaz, Haris, Mohammad Alshayeb, Sajjad Mahmood, and Mahmood Niazi. "An empirical study to improve software security through the application of code refactoring." Information and Software Technology 96 (2018): 112-125.
9. Li, W., & Shatnawi, R. (2007). An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution. *Journal of systems and software*, *80*(7), 1120-1128.
10. Palomba, Fabio, Marco Zanoni, Francesca Arcelli Fontana, Andrea De Lucia, and Rocco Oliveto. "Smells like teen spirit: Improving bug prediction performance using the intensity of code smells." In 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 244-255. IEEE, 2016.
11. Marinescu, R. (2004, September). Detection strategies: Metrics-based rules for detecting design flaws. In *20th IEEE International Conference on Software Maintenance, 2004. Proceedings.* (pp. 350-359). IEEE.
12. Moha, N., Gueheneuc, Y. G., Duchien, L., & Le Meur, A. F. (2009). Decor: A method for the specification and detection of code and design smells. *IEEE Transactions on Software Engineering*, *36*(1), 20-36.
13. Olbrich, S. M., Cruzes, D. S., & Sjøberg, D. I. (2010, September). Are all code smells harmful? A study of God Classes and Brain Classes in the evolution of three open source systems. In *2010 IEEE International Conference on Software Maintenance* (pp. 1-10). IEEE.
14. Rahman, F., Bird, C., & Devanbu, P. (2012). Clones: What is that smell?. *Empirical Software Engineering*, *17*(4-5), 503-530.

15. Schumacher, J., Zazworka, N., Shull, F., Seaman, C., & Shaw, M. (2010, September). Building empirical support for automated code smell detection. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement* (p. 8). ACM.
16. Silva, A. L., Garcia, A., Reioli, E. J., & De Lucena, C. J. P. (2013, October). Are domain-specific detection strategies for code anomalies reusable? An industry multi-project study. In *2013 27th Brazilian Symposium on Software Engineering* (pp. 79-88). IEEE.
17. Silva, L. L., Valente, M. T., & Maia, M. D. A. (2014, April). Assessing modularity using co-change clusters. In *Proceedings of the 13th international conference on Modularity* (pp. 49-60). ACM.
18. Sjøberg, D. I., Yamashita, A., Anda, B. C., Mockus, A., & Dybå, T. (2012). Quantifying the effect of code smells on maintenance effort. *IEEE Transactions on Software Engineering*, *39*(8), 1144-1156.
19. Walker, R. J., Rawal, S., & Sillito, J. (2012, November). Do crosscutting concerns cause modularity problems?. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering* (p. 49). ACM.
20. Wu, R., Zhang, H., Kim, S., & Cheung, S. C. (2011, September). Relink: recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering* (pp. 15-25). ACM.
21. Yamashita, A., & Moonen, L. (2012, September). Do code smells reflect important maintainability aspects?. In *2012 28th IEEE international conference on software maintenance (ICSM)* (pp. 306-315). IEEE.
22. Zazworka, N., Shaw, M. A., Shull, F., & Seaman, C. (2011, May). Investigating the impact of design debt on software quality. In *Proceedings of the 2nd Workshop on Managing Technical Debt* (pp. 17-23). ACM.
23. Zhang, M., Hall, T., & Baddoo, N. (2011). Code bad smells: a review of current knowledge. *Journal of Software Maintenance and Evolution: research and practice*, *23*(3), 179-202.
24. Zhang, M., Hall, T., Baddoo, N., & Wernick, P. (2008, July). Do bad smells indicate trouble in code?. In *Proceedings of the 2008 workshop on Defects in large software systems* (pp. 43-44). ACM.
25. Suryanarayana, G., Samarthyam, G., & Sharma, T. (2014). *Refactoring for software design smells: managing technical debt*. Morgan Kaufmann.
26. Sharma, T., Mishra, P., & Tiwari, R. (2016, May). Designite: a software design quality assessment tool. In *Proceedings of the 1st International Workshop on Bringing Architectural Design Thinking into Developers' Daily Activities* (pp. 1-4). ACM.
27. Feng, Q., Kazman, R., Cai, Y., Mo, R., & Xiao, L. (2016, April). Towards an architecture-centric approach to security analysis. In *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)* (pp. 221-230). IEEE.
28. Yasir Javed, Mamdouh Alenezi, Mohammed Akour, Ahmad Alzyod. Discovering The Relationship Between Software Complexity And Software Vulnerabilities, Journal of Theoretical and Applied Information Technology 31st July 2018. Vol.96. No 14
29. Akour, Mohammed, and Izzat Alsmadi. "Vulnerability assessments: a case study of Jordanian universities." 2015 International Conference on Open Source Software Computing (OSSCOM). IEEE, 2015.
30. Alrawais, Layla Mohammed, Mamdouh Alenezi, and Mohammad Akour. "Security Testing Framework for Web Applications." International Journal of Software Innovation (IJSI) 6.3 (2018): 93-117.

AUTHORS PROFILE

Dr. Mamdouh Alenezi is currently the Dean of Educational Services at Prince Sultan University. Dr. Alenezi received his MS and Ph.D. degrees from DePaul University and North Dakota State University in 2011 and 2014, respectively. He has extensive experience in data mining and machine learning where he applied several data mining techniques to solve several Software Engineering problems. He conducted several research areas and development of predictive models using machine learning to predict fault-prone classes, comprehend source code, and predict the appropriate developer to be assigned to a new bug.

Dr. Mohammed Akour is an associate Professor of Software Engineering at Al Yamamah University (YU). He got his Bachelor's (2006) and Master's (2008) degree from Yarmouk University in Computer Information Systems with Honor. He joined Yarmouk University as a Lecturer in August 2008 after graduating with his master's in Computer Information Systems. In August 2009, He left Yarmouk University to pursue his Ph.D. in Software Engineering at North Dakota State University (NDSU). He joined Yarmouk University again in April 2013 after graduating with his Ph.D. in Software Engineering from NDSU with Honor. He serves as Keynote Speaker, Organizer, a Co-chair and publicity Chair for several IEEE conferences, and as ERB for more than 10 ISI indexed prestigious journals. He is a member of the International Association of Engineers (IAENG). Dr. Akour at Yarmouk University served as Head of accreditation and Quality assurance and then was hired as director of computer and Information Center. In 2018, Dr. Akour has been hired as Vice Dean of Student Affairs at Yarmouk University. In 2019, Dr. Akour joins Al Yamamah University -Riyadh Saudi Arabia- as an associate professor in Software Engineering.