Waste Definition and Identification in Software Engineering: A Review

Khalifa Ahmed
College of Engineering, Design and Physical Sciences
Brunel Univeristy of London
London, UK

Nour Ali
College of Engineering, Design and Physical Sciences
Brunel Univeristy of London
London, UK

Wasan Awad
College of Information Technology
Ahlia University
Manama, Bahrain

Abstract— **Defining and identifying waste associated with software is a hot topic in the information technology industry. Therefore, on this paper we went through literature searching for definitions of software waste in general, then with more focus on waste in the context of software architecture. Then, we tried to identify the influential factors for the presence of waste along with its relationships with architectural technical debt and architectural bad smell. The majority of the definitions of waste focused on value delivery aspects of waste, as if the software component is not adding value to the end-users is considered as waste. Adding to that there were links between the presence of waste in the context of software architecture and architectural technical debt and architectural bad smell.**

Keywords— Software Waste, Software Architecture, Software Engineering

I. Introduction

Referring to Cambridge English Dictionary, the word “waste” is defined as “an unnecessary or wrong use of money, substances, time, energy, abilities, etc.” [1]. This definition triggered the demand for further investigation and finding the definition of waste in the context of software architecture.

In this paper we are presenting our findings based on readings and studies conducted related to the presence of software waste in the context of software development and specifically software architecture. Initially we defined a set of research questions that we tried to answer based on the existing literature.

The research questions that we are trying to answer in this paper include the following:

RQ1: How has software waste been defined in the context of software development in general?

RQ2: How has software waste been defined in the context of software architecture specifically?

RQ3: What are the factors that influence the presence of software waste?

RQ4: Is there a link between Architectural Technical Debt and software waste?

Fig. 1. Illustrates the structure of the review conducted and the sequence for answering the research questions. RQ1 and RQ2 will be address on the following section “Software Waste Definition”, then RQ 3 and RQ 4 will be answered on the following sections “Influential Factors” and “Waste and Software Architectural Debt” respectively. After that, the results generated from the literature will be summarized and discussed in the “results and discussion” section.

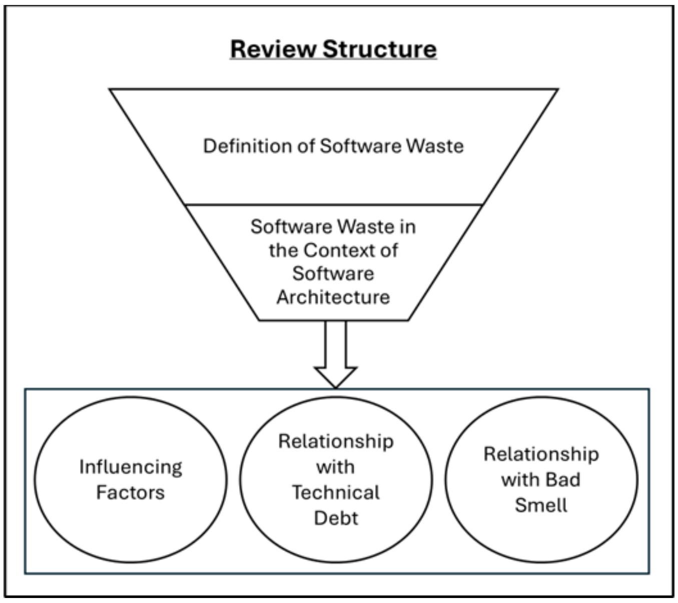
Review Structure
Definition of Software Waste
Software Waste in the Context of Software Architecture
Influencing Factors
Relationship with Technical Debt
Relationship with Bad Smell

Fig. 1. Review Structure

II. Software Waste Definition

Based on the literature review conducted, waste was defined from multiple aspects. However, most of the definitions are based on Lean Approach. In table I, few definitions of waste ware presented.

The definitions mentioned in table I focused more on the concept of waste from Lean perspective. The focus of lean principles focusses on waste elimination, reducing cycle time, and increasing the flow of value delivery to customers. Based on Toyota Production System the term “Muda” refers to the activities that doesn’t add any value to the customer. This will lead to unnecessary financial costs, storage costs, etc. The main categories of waste are Defects, Over-production, Over-processing, Waiting, Motion, Inventory, and Transportation [2].

TABLE I. LIST OF WASTE DEFINITIONS

Author(s)	Definition of waste
Mary Poppendieck and Tom Poppendieck	"Waste is anything that does not add value to a product, value as perceived by the customer" [3]
David Bernstein	"Waste is defined as any work that has begun but is not yet completed" [4]
Todd Sedano et al.	"Waste is any activity that produces no value for the customer or user" [5]
Forbes Technology Council	"The technology they acquire ends up being underutilized or wholly ignored—becoming software waste." [6]
Todd Sedano et al.	"Software waste refers to project elements (objects, properties, conditions, activities, or processes) that consume resources without producing benefits. Wastes are like friction in the development process." [7]

Based on N. Brown [8] "the concept of focusing on waste as a means to reduce cycle time and increase the flow of value is equally applicable to manufacturing and to software development". The author focused on three categories of waste from the perspective of software development generally and software architecture specifically, and these categories of waste are: Defects, Overproduction, and Extra Processing / Excess Complexity. Defect waste in the context of software architecture is caused by the rework required to fix the defect, this has direct negative impact to the flow of value delivery to the end users. Adding additional functionality that are not part of the requirements or developing complex software architecture that is in excess to end users needs, may contribute to waste of team efforts, adding more complexity to the code maintainability and modification, and increase the testing efforts, etc. Hence, all these factors will contribute to increase the financial costs and resources required.

The waste of waiting in the context of software architecture is mainly caused by the developers waiting for the final software architecture developed by the software architects. In many cases, the developers will end-up starting code development which will cause them to underutilize the produced software architecture developed by the software architects. This will result to increase the rework efforts, and/or producing a throwaway code by the developers in many occasions. The best course of action is that all team members to start working on their tasks once all their requirements are ready [9].

III. Influential Factors

In this section, the factors that will influence the presence of software waste in the context of software architecture will be discussed. Starting with the factors that will have bad impacts or implications on software architecture quality. These factors have direct relationships with the presence of waste in the context of software architecture.

These factors include centralization of decision making of decisions related to software architecture and not involving all concerned team members, letting goals such as reusability of software components dictate bad software architecture decisions, imbalance between business decisions and architectural decisions, ignoring the importance of feedback, and over generalization of software architecture [10].

Moreover, improper definition of users requirements (functional and non-functional requirements) could contribute to adding waste from multiple perspectives, leading to wasted related to rework. This is a direct result of software architects being building their decisions without having full picture about the end users requirements, or based on wrong analysis of end users requirements.

In addition to that, there are some non-technical factors that affect negatively on the software architecture and these include not having a software architect, lack of vision, and splitting the resolution of issues over multiple individual teams [11]. These issues also could contribute to increasing waste in the software.

IV. Waste and Software Architectural Debt

In Software engineering, technical debt is defined as "making technical compromises that are expedient in the short term, but that create a technical context that increases complexity and cost in the long term" [12]. Hence, the financial costs, efforts needed, and software complexity will increase in the future for the purpose of maintaining or modifying or adding future enhancements to the software. These additional costs, efforts and extra complexity are considered as waste.

On the other hand, architectural technical debt is defined as "the technical debt incurred at the architectural level of software design, that is, in the decisions related to the choice of structure (e.g., layering, decomposition in subsystems, interfaces), the choice of technologies (e.g., frameworks, packages, libraries, deployment approach), or even languages, development process, and platform" [13]. Therefore, the presence of architecture debt could lead to generating waste in the context of software architecture.

Another very important concept that must be addressed here is Architectural Bad Smells or Architectural Smells. J. Garcia et al. defined architectural smell as "are frequently recurring software designs that can have non-obvious and significant detrimental effects on system lifecycle properties" [14]. There is a relationship between architecture smells, software qualities and refactoring. The presence of architecture smell could lead to the compromise of software qualities [15]. This could lead to defects and overconsumption of resources during the deployment and operations. The impact here exceeds extra costs or extra complexity of software, as it has direct impact to utilization of computing resources and energy consumption. D. Guamán et al. found that The energy consumption of application has link with the complexity and code smells [16]. Adding to that G. Dhaka and P. Singh identified a relationship between removing code smells (god class, feature envy and long method) using recommended refactoring activities and reduction of energy consumption [17].

V. Results and Discussion

Most of the definitions of waste in software development are based on Lean approach which is manly focusing on waste elimination, reducing cycle time, and increasing the flow of value delivery to customers. Most of the definitions found ware inspired by Mary and Tom Poppendieck definition of waste and waste categories on their book "Lean Software Development: An Agile Toolkit". Most of the definitions agree that waste is considered as software components that do not add or represent value to the customer or end users.

If the software component does not represent value for the customer then it is considered as waste, and this could include not finished items or software components, features and items which are not requested by the customer or end users, software components that are not fully utilized, or software components that could consume additional resources without generating extra value for the customers or end users.

Moving to the presence of waste in the context of software architecture, there were very few publications discussing this topic. Most of the available resources are industrial based resources that were presenting subject matter experts experiences and opinions on this topic such as software architects and software developers. There is huge demand for more academic research on this topic to fill the lack of available academic publications. The focus here was more into the additional costs and efforts spent on software development, fixes or repair, modifications and maintainability. The main categories of waste were defects, overproduction, extra processing / excess complexity, and waiting time.

The influential factors for the presence of waste in the context of software architecture could be categorized into two main factors. Technical factors that are directly related to the development cycle of the software that affected the decisions of the development team regarding the software architecture, and non-technical factors that related mostly to the development team. The technical factors are mainly focusing on the software architecture decisions process. These factors mainly concern whether all the right stakeholders were involved in the decision-taking process, and if the right balance between the business goals and software requirements was there during the decision-making process. Adding to that, additional concerns by the software architects such as the reusability of software components and the generalizations of the software architecture. Moreover, building the decisions regarding the software architecture must be based on the full picture of the end users requirements and proper analysis of these requirements. An example here, if software architects built their decisions regarding the design pattern to be used and decided to go for a layered pattern without finalizing all the requirements with customers, and the team proceeded with the development cycle. Later, the customer added a requirement to use microservices pattern as a requirement. This could cause the development team to start all over from the point of software architecture and design phase, and in most cases all work done would be scrapped. This will add extra costs and extra efforts by the team leading into generating waste in the development cycle. Another example, if software architects built their decisions regarding the design pattern to be used and decided to go for the most suitable design pattern at that point of time without finalizing all the requirements with customers, and the team proceeded with the development cycle. Toward the finalization of the requirements gathering and analysis phase, the software architects realized that if they selected another design pattern it would be much better and efficient. Again here, this could cause the development team to start all over from the point of software architecture and design phase, and in most cases all work done would be scrapped. Alternatively, and in most cases the team will proceed with the selected design pattern due to time and schedule concentrations, although it will add extra complexity to the software and improper utilization of computing resources. Adding to that the extra costs and effort that will be required for maintaining and modifying the software in the future.

The non-technical factors – as mentioned in section III based on the inputs from N. Bédard [11] - are mainly concerned with the management, suitability and ability of the development team while taking the decisions that are affecting the software architecture. Having a qualified and dedicated software architect in the development team could help to reduce the possibility of having waste in the context of software architecture. Adding to that having a clear vision for the software and future business goals of the software being developed could help to reduce the possibilities of having in the context of software architecture too. Moreover, the knowledge of the team regarding the software structure and the resolution of the issues that are being addressed by all the relevant development team members also could contribute to reducing the possibility of having waste in the context of software architecture.

Furthermore, the presence of architectural technical debt could lead to the presence of waste in the context of software architecture. This is due to the costs, efforts, and software complexity will increase in the future for the purpose of maintaining, modifying or adding new features to the software. Selecting the most appropriate architecture design pattern will help to reduce the possibility of the presence waste in the context of software architecture. This is possible by involving a well-qualified and experienced software architect in the process of software development process, along with involving all the relevant team members to end up with well designed software architecture with minimum waste in the context of software architecture. This will help to reduce the costs and efforts required to implement and maintain a high quality software.

Moreover, there is a direct link between the presence of waste in the software architecture and architecture bad smell. The expected results include but are not limited to over utilization of computing resources and increase in energy consumption which all adds to customer bill, and software operation and maintenance efforts. Similarly, selecting the most appropriate architecture design pattern will help to reduce the possibility of the presence waste in the context of software architecture and better utilization of available resources in terms of software development and operations.

According to the discussion above and the literature review conducted, there is a big gap in the academic research in terms of how to link architectural technical debt with waste, what are the influential factors that increase waste in the context of software architecture, and what are the best approaches to reduce waste.

VI. Conclusion

On this paper we started by gathering data from available literature regarding the definition of waste in the context of software development in general, then in software architecture specifically. After that, the influential factors for the presence of waste were identified and the relationships between the presence of waste and architectural technical debts and architecture bad smells were identified. Finally, the results generated from the literature were discussed in the previous chapter.

Most of the definitions of waste focused on the value generation for the end users and the main categories of waste in the software include defects, overproduction, extra processing / excess complexity, and waiting time. The influence factors were divided into technical factors and non-technical factors.

The topic addressed by this paper requires further studies, especially in the aspect of the presence of waste in the context of software architecture. This topic is well addressed by the industry, however there is a demand for more focus from academic researchers.

References

[1] Cambridge University Press, "Cambridge English Dictionary," [Online]. Available: https://dictionary.cambridge.org/dictionary/english/waste. [Accessed 1 March 2025].

[2] Toyota (GB) PLC, "Muda, Muri, Mura – Toyota Production System guide," 31 May 2013. [Online]. Available: https://mag.toyota.co.uk/muda-muri-mura-toyota-production-system/. [Accessed 1 March 2025].

[3] M. Poppendieck and T. Poppendieck, Lean Software Development: An Agile Toolkit, Addison-Wesley Professional, 2003.

[4] D. Bernstein, "What is Software Waste?," Agile Alliance, 20 April 2016. [Online]. Available: https://www.agilealliance.org/what-is-waste/. [Accessed 1 March 2025].

[5] T. Sedano, P. Ralph and C. Péraire, "Software Development Waste," in 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), Buenos Aires, 2017.

[6] Forbes Technology Council, "12 Tips For Eliminating Software Waste At Your Company," Forbes Media LLC, 8 October 2019. [Online]. Available: https://www.forbes.com/councils/forbestechcouncil/2019/10/08/12-tips-for-eliminating-software-waste-at-your-company/. [Accessed 1 March 2025].

[7] T. Sedano, P. Ralph and C. Péraire, "Removing Software Development Waste to Improve Productivity," in Rethinking Productivity in Software Engineering, Berkeley, CA., Apress, 2019, p. 221–240.

[8] N. Brown, "Lean Principles and Software Architecture: Categories of Waste," 12 May 2011. [Online]. Available: https://insights.sei.cmu.edu/blog/lean-principles-and-software-architecture-categories-of-waste/. [Accessed 1 March 2025].

[9] N. Brown, "Lean Principles and Software Architecture: The Waste of Waiting," 11 June 2011. [Online]. Available: https://insights.sei.cmu.edu/blog/lean-principles-and-software-architecture-the-waste-of-waiting/. [Accessed 1 March 2025].

[10] P. Pureur and K. Bittner, "12 Software Architecture Pitfalls and How to Avoid Them," 13 December 2023. [Online]. Available: https://www.infoq.com/articles/avoid-architecture-pitfalls/. [Accessed 1 March 2025].

[11] N. Bédard, "Non-technical reasons why your software architecture is a mess," 10 October 2023. [Online]. Available: https://normand-bedard.medium.com/non-technical-reasons-why-your-software-architecture-is-a-mess-43b3c1064f11. [Accessed 1 March 2025].

[12] P. Avgeriou, P. Kruchten, I. Ozkaya and C. Seaman, "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)," Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2016.

[13] R. Verdecchia, P. Kruchten, P. Lago and I. Malavolta, "Building and evaluating a theory of architectural technical debt in software-intensive systems," Journal of Systems and Software, vol. 176, no. 110925, pp. 1-23, 2021.

[14] J. Garcia, D. Popescu, G. Edwards and N. Medvidovic, "Identifying Architectural Bad Smells," in 13th European Conference on Software Maintenance and Reengineering, Kaiserslautern, 2009.

[15] J. Reimann and U. Aßmann, "Quality-Aware Refactoring for Early Detection and Resolution of Energy Deficiencies," in 6th International Conference on Utility and Cloud Computing, 2013.

[16] D. Guamán, J. Pérez, P. Valiviezo and N. Cañas, "Estimating the energy consumption of software components from size, complexity and code smells metrics," in 37th ACM/SIGAPP Symposium on Applied Computing, New York, NY, 2022.

[17] G. Dhaka and P. Singh, "An empirical investigation into code smell elimination sequences for energy efficient software," in 23rd Asia-Pacific Software Engineering Conference, 2016.