Tanzania Journal of Engineering and Technology 2024, **43**(**4**):153-163
OPEN ACCESS articles distributed under Creative Commons Attribution Licence [CC BY-ND]
Websites: https://ajol.org/tjet; https://tjet.udsm.ac.tz

ISSN 1821-536X (**print**); ISSN 2619-8789 (**electronic**)
https://doi.org/10.52339/tjet.v43i4.1147

Regular Research Manuscript

Effects of Technical Debt on Software Interoperability

Leonard Peter Binamungu

Department of Computer Science and Engineering, University of Dar es Salaam, P.O Box 33335, Dar es Salaam, Tanzania

†Corresponding author: lepebina@udsm.ac.tz; ORCID: 0000-0001-6385-1353

ABSTRACT

Technical debt (TD) refers to sub-optimal development decisions that make the software costly to maintain and evolve. Examples of TD include structural complexity, violation of coding styles, and code complexity. Existing research has investigated the nature, causes and indicators of TD, as well as tools and strategies for managing TD. However, although TD could hinder the ability of a software system to be interoperable with others, existing literature has limited evidence on how TD affects systems interoperability. This limits the ability of software engineering teams to manage TD in ways that do not hinder systems interoperability. To fill this void, two system interoperability projects in the health sector, involving 35 systems, were analysed to understand how TD affects systems interoperability. The complexity of the healthcare domain and the diversity of the 35 systems enabled a clear understanding of the intricate interactions between technical debt and systems interoperability. The identified interoperability challenges were mapped to five different TD types, all of which can be linked to software development practices that do not prioritise responsible management of TD. Documentation and requirements debt were identified as the most prevalent barriers to interoperability in the studied healthcare domain. The findings suggest that improving software development processes through interoperability-sensitive TD management strategies could improve software interoperability. The paper makes an empirical contribution by mapping interoperability challenges to technical debt, enabling us to conceptualise system interoperability challenges as consequences of technical debt. The implications of this contribution for domain and research practices are also provided.

ARTICLE INFO

First: Sep. 29, 2024

Revised: Dec. 19, 2024

Accepted: Dec. 30, 2024

Published: Feb. 2025

Keywords: T*echnical debt, systems interoperability, interoperability challenges, technical debt and interoperability, technical debt types.*

INTRODUCTION

Technical debt (TD) refers to sub-optimal software development decisions that could make the software costly to maintain and evolve (Cunningham, 1992; Kruchten *et al*., 2012). The term was first used by Cunningham (1992) to explain to non-software engineers the impact of not performing code refactoring. Later on, it has been refined to refer to sub-optimal software development decisions that could make the software costly (or even prohibitively expensive) to maintain and evolve. Interoperability, on the other hand, denotes the ability of different systems to interact with each other and exchange information in a seamless manner. This is especially important when information stored in different systems is required in a

particular system to facilitate full understanding of a situation and, therefore, better decision making (Binamungu, 2024). Different studies about TD have been conducted. The studies have focused on the nature and causes of TD (Kruchten et al., 2012; Rubin, 2012; Yang *et al*., 2023), as well as the strategies to identify and manage TD (AlOmar *et al*., 2022; Clark, 2018; Lenarduzzi *et al*., 2021; Sharma, 2019; Sierra *et al*., 2019). Tools for dealing with TD have also been proposed (Lenarduzzi *et al*., 2021; Saraiva *et al.*, 2021).

Although technical debt could hinder the ability of a software system to be interoperable with others, existing literature has limited evidence on how technical debt affects systems interoperability. Specifically, existing attempts to relate technical debt and software interoperability challenges (Gallenson *et al*., 2021; Yang *et al*., 2023) have focused on specific kinds of systems and do not offer a comprehensive mapping of interoperability challenges that are attributable to technical debt. This limited focus could impact the ability of software engineering teams to manage technical debt in ways that do not hinder the interoperability of systems of different types, within and across different domains. Specifically, software engineers should be able to appreciate how the different types of TD they incur during software development hinder system interoperability. However, existing studies on TD and interoperability do not help software engineers to appreciate the potential consequences of different types of TD on systems interoperability. This could produce systems that are hardly interoperable, hindering information sharing goals that are achievable through interoperability among systems. To fill this void, the present study analysed two system interoperability projects, involving 35 systems from within and outside the health domain, to understand how technical debt affects systems interoperability. The identified interoperability challenges were mapped to different types of technical debt. Specifically, these challenges were mapped to five different TD types: process debt, design debt, people debt, documentation debt, and requirements debt. All these five TD types can be linked to software development practices that do not handle technical debt responsibly, similar to naïve debt (Rubin, 2012), the addressing of which requires radical improvements of software development processes.

The present paper makes an empirical contribution by mapping interoperability challenges to technical debt, enabling us to conceptualise system interoperability challenges as consequences of technical debt. The implications of this contribution for domain and research practices are provided.

LITERATURE REVIEW

Theorising Technical Debt

The term "technical debt" was coined by Cunningham (1992) to explain to non-software engineers the impact of not performing code refactoring. It was subsequently used to refer to sub-optimal software development decisions, which make the software costly or prohibitively expensive to maintain and or evolve. The cost (e.g., development effort and time) required to pay the technical debt could be very high, sometimes necessitating abandoning the existing software and developing a new one. Realising the potential danger of regarding every sub-optimal decision made during software development as technical debt, Kruchten *et al*. (2012) used a lens of visibility to characterise technical debt. Specifically, they suggested that the definition of technical debt should be limited to invisible issues (known only to the development team) that make a software hard to maintain and evolve (refer to Figure 1). However, while useful in understanding the nature of TD, the visibility lens of TD characterisation, as used in the work of

Kruchten *et al*. (2012), does not facilitate understanding of why software engeers incur TD, and the scope and impact of TD.

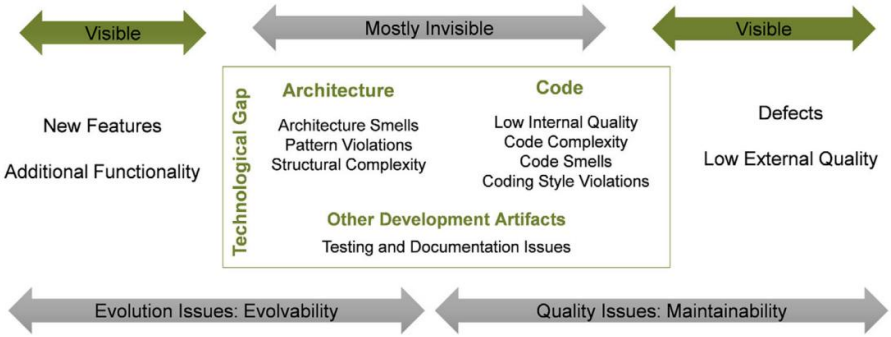

Figure 1: Landscape of technical debt (Kruchten *et al*., 2012; Yang *et al*., 2023).

Rubin (2012) characterised technical debt based on the reasons for incurring it. This led to three types of TD: naïve TD, unavoidable TD, and strategic TD. Naïve TD happens when people involved in software development behave irresponsibly. Causes of naïve TD include bad engineering practices, careless design, inadequate testing, limited project scope, tight deadlines, and constrained budget. Thus, improving a software development process should help the team to avoid naïve TD. Unavoidable TD is caused by an unpredictable future and or complexity of a system under development. It is likely to happen as a result of the need to adapt a system to suit new requirements or technology. Strategic TD is caused by decisions that favour economic gains of an organisation at a particular point in time. For example, a short time to market for a software product could necessitate an organisation to incur TD (Rubin, 2012). Nevertheless, although it facilitates understanding the reasons for incuring TD, Rubin (2012)'s lens of TD characterisation does not enable understanding of the scope and impact of technical debt.

Clark (2018) used a lens of TD scope and impact to characterise TD. This led to four types of TD: local debt, McGyver debt, foundational debt, and data debt. The local debt has a limited scope of impact. McGyver debt refers to temporary solutions that cannot be relied upon in the long term. Foundational debt necessitates future modification of a basic design assumption. Data debt is caused by building the content in a system that contains any of the above debts (Clark, 2018). However, while it enables us to understand the scope and impact of TD, Clark (2018)'s lens of TD characterisation does not pay due attention to how TD affects systems interoperability.

Technical debt is not always bad: the context of its introduction is what matters (Besker *et al*., 2018; Kruchten *et al*., 2012). For example, when the time to market for a software product is key, it might be good to incur technical debt, as long as it is properly documented and payable in the future (Kruchten *et al*., 2012). To properly deal with TD, existing literature has also paid attention to the identification and management of technical debt in software engineering (AlOmar *et al*., 2022; Kruchten *et al*., 2012; Lenarduzzi *et al*., 2021; Saraiva *et al*., 2021; Sharma, 2019; Sierra *et al*., 2019). However, little attention has been paid to understanding how technical debt could affect the interoperability of software systems.

Related Work

As regards the effects of technical debt on systems interoperability, only two studies were identified. First is the work of Yang et al. (2023) that proposed a taxonomy for identifying and assessing technical debt in

complex distributed systems. Issues related to interoperability were identified to be among the indicators of technical debt when engineering complex systems. The work suggests that systems interoperability is affected by the complexity of COTS (Commercial Off-the-Shelf) components and higher dependence between system components. However, the work of Yang *et al*. (2023) focuses more on the interoperability among COTS systems (and not custom systems). Second, to manage and mitigate the effects of TD on other systems that are interoperable with a particular system, Gallenson *et al*. (2021) included interoperability in the schemas for the assessment of the risks of technical debt in defence systems. However, these studies focus on specific kinds of systems and do not offer a comprehensive mapping of interoperability challenges that are attributable to technical debt.

MATERIALS AND METHODS

Research Approach

This study followed a qualitative approach and employed an interpretivism research paradigm. Two exploratory case studies (Yin, 2018) were used to understand how technical debt affects systems interoperability.

Data Collection

The data for the present study was collected through participant observation (active involvement) in two systems interoperability projects in the health sector in Tanzania from 2020 until the time of writing this article. This provided the opportunity for the researcher to also have in-situ conversations with technical and non-technical stakeholders of the systems that participated in the two interoperability projects. The first systems interoperability project focused on facilitating interoperability between systems in 13 national, consultant and specialised hospitals with the District Health Information Software Version Two (DHIS2). The second systems interoperability project focused on facilitating interoperability between 22 systems for collecting and managing human resources for health data. This included systems for pre-service data, professional registration data, in-service data, and continuous professional development data. The complexity of the healthcare sector in Tanzania (as characterised by the presence of multiple systems, developed by multiple stakeholders, and serving multiple objectives) and the diversity of the 35 systems enabled a clear understanding of the intricate interactions between technical debt and systems interoperability.

The stakeholders involved in the study include system developers, system analysts, business analysts, system vendors, policy makers, regulators, and other domain experts. The challenges encountered throughout the two interoperability projects were documented and linked to TD, to uncover the TD types that must be paid by software engineers for systems interoperability to succeed. The collected qualitative data were mainly about challenges that hindered smooth interoperability among systems, causes of the interoperability challenges, and how the causes of interoperability challenges were related to technical debt.

Data Analysis

To link the identified interoperability challenges to appropriate types of technical debts, the taxonomy of TD types (Rios *et al*., 2018) was used. The TD types from this taxonomy are summarised in Table 1. However, as can be seen in Table 1, whenever necessary, the definitions of some TD types were extended to accommodate the unique challenges encountered in the present study. Whenever necessary, more conversations were held between the researcher and stakeholders to verify the mapping between the observed interoperability challenges and technical debt related to the studied systems.

Table 1: Types of technical debts (Rios *et al*., 2018)

SN	TD type	Description
1	Design debt	Debt related to violating object-oriented design principles. It is discoverable by analysing the source code. The present study extended design debt to include poor design of business processes by failing to streamline all business processes of an organisation. It can be indicated by complex designs, code smells, and complex methods or classes
2	Code debt	Code violating best coding practices and or rules, producing code that is hard to comprehend, extend and maintain. It can be indicated by poor styling, unnecessary code duplication, and code complexity
3	Architecture debt	Problems in an architecture of a software product, which pose internal quality issues like maintainability. It can be indicated by things like modularity violations, complex architecture, quality issues related to system structure, variations in the use of architectural patterns and policies, paying no attention to non-functional requirements, and the use of architectural techniques that are not mature
4	Test debt	Problems with the quality of software testing activities. Indicators of test debt include a lack of tests of different types (unit, integration, system, and acceptance tests), and deferred testing
5	Documentation debt	Problems related to software documentation. It can be indicated by missing, inadequate, outdated, or incomplete documentation. The present study extended the scope of documentation debt to include policy and legal documentation required to support systems development
6	Defect debt	Known defects that should be fixed but have been deferred to a later time due to competing priorities. Defect debt can be indicated by delayed decisions on fixing defects, bugs or failures in a software product.
7	Infrastructure debt	Infrastructure problems that negatively impact the ability of a software engineering team to produce good quality software. It can be indicated by things like delayed infrastructure upgrades, outdated components of a software development environment, and undesirable configurations of software development tools.
8	Requirements debt	Difference between ideal requirements and the implemented system. The present study extended requirements debt to include incomplete or missing requirements. Requirements debt can be indicated by situations like the presence of partially implemented requirements and implementing the system in a way that does not satisfy all non-functional requirements.
9	People debt	People-related problems that can delay software engineering activities. The present study extended people debt to include the presence of competing interests among people in an

		organisation, which could hinder the efficient and effective implementation of systems. People debt can be indicated by situations like delayed hiring of key software engineering professionals.
10	Build debt	Problems that complicate and delay system building. Build debt can be indicated by situations like manual system building process, having code with no customer value involved in the build process, and the presence of incorrect dependencies that delay the build process
11	Process debt	Inefficient process. It can be indicated by the presence of inappropriate processes, and manual processes.
12	Automation test debt	Work of developing automated tests for functionality developed in the past, to foster faster software development cycles and continuous integration. It can be indicated by the absence of automated tests
13	Usability debt	Problems related to system usability. It can be indicated by the presence of inappropriate usability decisions that have to be fixed later.
14	Service debt	Issues related to incorrect selection and substitution of web services in systems that follow service-oriented architectures. The debt can be indicated by inappropriate selection or replacement of web services.
15	Versioning debt	Issues related to versioning of source code. It can be indicated by situations like unnecessary forking of code.

RESULTS AND DISCUSSIONS

As summarised in Table 2, six key challenges were encountered across the two IS interoperability projects in the health sector. The first challenge was the existence of heterogeneous systems serving the same objectives. Examples of this are four systems for health professional registration councils. Each of the following three councils had its own professional registration and management system: The Medical Council of Tanganyika, the Pharmacy Council of Tanzania, and the Nursing and Midwifery Council of Tanzania. The fourth system served several other health professional registration councils, including the Medical Radiology and Imaging Professio nal Council and the Traditional and Alternative Health Practice Council. Referring to Table 1, this challenge can be mapped to process debt and design debt, because the existence of multiple systems that serve the same objectives (in this case, registration and management of health professionals) indicates the presence of inefficient processes. After process optimisation (e.g., through business process reengineering), one system could serve all health professional registration councils. This could, in turn, simplify interoperability between health professional councils and other related systems, because only one system for the registration and management of health professionals would be involved in a systems interoperability endeavour. It is this failure to streamline all business processes related to health professional registration and management that also led to the mapping of the challenge in question to design debt. Efficient and effective design of health professional registration and management processes should avoid the existence of multiple systems that serve duplicate objectives.

The second systems interoperability challenge was the presence of multiple vendors with different interests and systems development skills. Seventeen (17) different vendors had been involved in developing 35 different systems that were involved in the two interoperability projects. The multiplicity of vendors with different interests and systems development skills was mapped to people debt for two reasons. One, it is because competing interests of people in an organisation (health sector, in this case) could hinder efficient and effective implementation of systems. Thus, the interests of people involved in systems development need to be aligned to produce systems that are easily interoperable with other systems. Two, the presence of system vendors who lack appropriate systems interoperability skills can produce systems that are not interoperable with others. For example, some systems in the studied interoperability projects lacked data-sharing APIs (Application Programming Interfaces) and developers of those systems struggled to produce the required data-sharing APIs. Consequently, this hindered the interoperability of those systems. Therefore, payment of people debt in a software engineering setting is important to ensure that the produced systems are interoperable with others. For example, ensuring that each software development team has skills to develop data-sharing APIs should address interoperability challenges related to the availability and quality of data sharing APIs.

The third encountered challenge was the presence of complex data-sharing policies, which delayed the implementation of interoperability between systems. This challenge was mapped to documentation debt because it is attributed to the lack of appropriate documentation for data sharing such as agreements and policies that delayed the implementation of systems interoperability and sharing of data across interoperable systems. In some cases, preparing and signing data-sharing agreements involved navigating long bureaucratic processes, delaying systems interoperability endeavours. Other systems even lacked data-sharing API documentation. If they remain unpaid, these documentation debts could prevent effective and efficient implementation of interoperable systems. Harmonisation of policies that facilitate sharing of data across multiple systems and organisations, and creating templates for data-sharing API documentation, could help software teams and organisations to navigate these documentation debts that hinder systems interoperability.

The fourth systems interoperability challenge was about the availability and nature of systems support contracts. For some systems, support contracts had expired, hindering systems interoperability. For other systems, making systems interoperable was regarded as a new feature that demanded fresh negotiations. These challenges can be linked to documentation debt because it encompasses issues related to systems support contract documents that hindered smooth systems interoperability. Thus, software engineering teams should pay all documentation debts to ensure effective and efficient systems interoperability. For example, system development and support contracts should designate interoperability as a basic requirement that should be prioritised in all system development and support activities.

The fifth challenge was related to missing data in some systems. The whole point of systems interoperability is to ensure that appropriate data is available across different systems, to inform decision-making. Thus, interoperable systems with incomplete data cannot facilitate the achievement of this objective. However, in the studied systems interoperability projects, some data was missing in some systems. For example, not all data for health workers in private health facilities were available or frequently updated. Providing all important data to other systems through an interoperability link

was not a key requirement during the early stages of developing some of the studied systems. Since this challenge is related to systems requirements, it can be mapped to requirements debt. Thus, requirements debt related to missing data in interoperable systems must be paid before systems interoperability can become a success. For instance, it might be important for each software development team to create a template of all data required in a particular system and other related systems, to ensure the availability of all data required to serve the interoperability objectives.

The sixth and final interoperability challenge was related to tracking data across systems. At the beginning of one of the studied systems interoperability projects, there was an interest in tracking individuals across pre-service, professional registration, and in-service systems. However, not all systems collected data that could be used for tracking an individual across all interoperable systems. For example, not all the pre-service and in-service data required to identify a health worker across different systems (e.g., National Identification Number and Form Four Index Number) was available in all systems, because some data were not considered necessary at the time of specifying requirements for some systems that were involved in the present study. Therefore, because the tracking requirements were not considered at the time of implementing the individual systems, it was practically impossible to track individuals across interoperable systems. This challenge can be mapped to requirements debt, because of the apparent oversight in specifying and implementing tracking requirements at the time of implementing the individual systems.

Table 2: Mapping of interoperability challenges to technical debt

SN	Interoperability challenge	TD mapping	Description	Example(s)
1	Existence of heterogeneous systems serving the same objectives	Process debt, design debt	• Unoptimized processes, leading to multiple duplicate systems serving the same organizational processes • Failure to streamline all business processes of an organization	Unoptimized processes for health professional registration councils, leading to multiple health professional registration systems
2	Multiplicity of vendors with different interests and systems development skills	People debt	• Failure to reconcile competing interests of multiple system vendors • Lack of skills to develop systems by considering interoperability requirements	Unwillingness to make systems interoperable; and absence of (or inability to develop) data sharing APIs
3	Complexity of data sharing policies	Documentation debt	• Unclear or complex data sharing policies, which caused delays and absence of data sharing agreements	• Signing of data sharing agreements with some stakeholders took years, and, in other cases, data

				was never shared at all. • Expiry of support contracts for some vendors at the time of an interoperability project
4	Availability and nature of systems support contracts	Documentation debt	Absence or inadequate support of interoperability in systems support contracts	In some support contracts, making one system interoperable with another was considered to be a new feature, necessitating fresh negotiations between system vendors and clients.
5	Missing data in some systems	Requirements debt	Some data required for effective and interoperable systems were missing	Lack of reliable data for health workers in the private sector
6	Tracking data across systems	Requirements debt	Failure to anticipate and account for the need to track data across interoperable systems	Inability to track health workers across in-service, professional registration and in-service systems due to lack of unique identifiers across systems

Similar to the work of Gallenson *et al*. (2021) and Yang *et al*. (2023), the findings of the present study also emphasize the need to ensure that technical debt in each system does not hinder smooth interoperability with other systems. Therefore, the strategies for incurring and managing technical debt should also consider interoperability requirements. However, different from past studies, the findings of the present study are unique in two ways. First, they are relevant for all kinds of systems, irrespective of the nature (e.g., COTS systems or custom systems) or domain (e.g., defence, health or others). Thus, improper management of technical debt in all kinds of systems could hinder systems interoperability. Second, the present study offers a mapping of some interoperability challenges that are attributable to technical debt. This could help software development teams to deal with technical debt in ways that mitigate interoperability challenges.

CONCLUSIONS

Although technical debt could hinder the ability of a software system to be interoperable with others, existing literature has limited evidence on how technical debt affects systems interoperability. This could impact the ability of software engineering teams to manage technical debt in ways that do not hinder systems interoperability. The

present study analysed two system interoperability projects to understand how technical debt affects systems interoperability. The identified challenges were mapped to different types of technical debt.

The present paper makes an empirical contribution by mapping interoperability challenges to technical debt. Doing so enables us to conceptualise system interoperability challenges as consequences of technical debt. For researchers, this contribution implies more opportunities to theorise about how technical debt affects systems interoperability, producing theories to better explain, predict and analyse the impacts of technical debt on effective and efficient systems interoperability. For software engineering practitioners, the empirical contribution of the present study implies opportunities for benchmarking and improving the management of technical debt by considering interoperability aspects.

Although only two interoperability projects were analysed, a total of 35 systems from within and outside the health domain were involved in the two interoperability projects. This provided an opportunity to study diverse sets of systems developed and maintained by stakeholders of different domains. The number and diversity of studied systems increase the credence of the findings of the present paper. However, in the future, it might be interesting to study more systems from more domains to gain a better understanding of how technical debts affect systems interoperability. Predicting system interoperability based on TD could be another area of research in the future. For instance, it might be interesting to develop predictive models to quantify the impact of documentation debt on system integration time. Software engineering professionals should also pay due attention to different types of technical debt and how to manage them, to prevent interoperability challenges. Specifically, software teams should avoid naïve debt by improving TD management practices and the overall software development process. For example, software engineers should incorporate interoperability requirements into technical debt management tools.

REFERENCES

AlOmar, E. A., Christians, B., Busho, M., AlKhalid, A. H., Ouni, A., Newman, C., & Mkaouer, M. W. (2022). SATDBailiff-mining and tracking self-admitted technical debt. *Science of Computer Programming*, **213**. doi.:10.1016/j.scico.2021.102693

Besker, T., Martini, A., Lokuge, R. E., Blincoe, K., & Bosch, J. (2018). Embracing technical debt, from a startup company perspective. *Proceedings - 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018*. doi.:10.1109/ICSME.2018.00051

Binamungu, L. P. (2024). The Role of Source Systems Strengthening in the Effective Interoperability of Digital Health Systems. *International Conference on Implications of Information and Digital Technologies for Development*, 309–324. doi:https://doi.org/10.1007/978-3-031-66986-6_23

Clark, B. (2018). *A Taxonomy of Technical Debt*. https://technology.riotgames.com/news/taxonomy-tech-debt

Cunningham, W. (1992). The WyCash portfolio management system. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, *Part F129621*. doi.:10.1145/157709.157715

Gallenson, A., Miller, S., & Higgins, S. (2021). *Addressing Software-Based, Platform Interoperability Risks in Defense Systems by Using Distressed Debt Financial Strategies: A Technical Debt Mitigation Concept*. Acquisition Research Program.

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *IEEE Software,* **29**(6). doi.:10.1109/MS.2012.167

Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F.A. (2021). A systematic literature review on Technical Debt

prioritization: Strategies, processes, factors, and tools. *Journal of Systems and Software*, **171**. doi.:10.1016/j.jss.2020.110827

Rios, N., Mendonça Neto, M. G. de, & Spínola, R. O. (2018). A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information and Software Technology,* **102**: 117-145. doi.:10.1016/j.infsof.2018.05.010.

Rubin, K. S. (2012). Essential Scrum: A Practical Guide to the Most Popular Agile Process - ISBN: 9780137043293. In *Scrum Framework*.

Saraiva, D., Neto, J. G., Kulesza, U., Freitas, G., Reboucas, R., & Coelho, R. (2021). Technical Debt Tools: A Systematic Mapping Study. *International Conference on Enterprise Information Systems, ICEIS - Proceedings*, *2*. doi.:10.5220/0010459100880098

Sharma, T. (2019). How deep is the mud: Fathoming architecture technical debt using designite. *Proceedings - 2019 IEEE/ACM International Conference on Technical Debt, TechDebt 2019*. doi.:10.1109/TechDebt.2019.00018

Sierra, G., Shihab, E., & Kamei, Y. (2019). A survey of self-admitted technical debt. *Journal of Systems and Software*, **152:** 70-82. doi.:10.1016/j.jss.2019.02.056

Yang, Y., Verma, D., & Anton, P. S. (2023). Technical debt in the engineering of complex systems. *Systems Engineering*, *26*(5). https://doi.org/10.1002/sys.21677

Yin, R. K. (2018). Case study research and applications: Design and methods. In *Journal of Hospitality & Tourism Research,* **53**(5). doi.:10.1177/109634809702100108.