1	Consultant	3	3	Low
2	QA Engineer	5	5	High
3	Developer	4	4	High
4	Developer	5	2	Intermediate
5	Team Lead	13	3	High
6	Developer	5	2.5	Low
7	Developer	3	3	Intermediate
8	Release Manager	2	2	Low
9	Customer Support Specialist	5	2.5	Intermediate

3.2 Interview Questions

We identified the interview questions given in Table 3 to retrieve as much as possible information from the interviewees. These questions were designed in an open-ended structure and applicable for all processes given in Table 1. We specifically aimed not to emphasize the technical debt term presuming that the initial awareness of the interviewees on the subject are low. The interview questions were prepared in English and Turkish, as there were people in the team with nationalities other than Turkish as well.

Table 3. Interview Questions

No	Questions
1	What is your role in the organization, and what kind of responsibilities do you have in the company and the projects?
2	Can you describe the project and the technologies being used in the project?
3	Which of the following Software Life Cycle Processes do you participate in the project? Project Planning, Project Assessment and Control, Stakeholder Needs and Requirements Definition, Systems/Software Requirements Definition, Architecture Definition, Design Definition, Implementation, Integration, Verification, Validation
4	Do you experience any issues with the processes/practices that you participate in the project?
5	Do you practice the tasks/activities defined for each process specified in ISO/IEC 12207?
6	Do you think there are tasks that you fail or don't achieve completely?
7	What kind of problems occur in the project due to these issues?
8	How did you realize these problems? Do you have any further observations on the problems?
9	Do you know if these problems are caused by the SaaS provider or the development team that you belong to?
10	Do you have any recommendations or a way to solve/prevent these problems?
11	Do you experience any further challenges regarding the usage of SaaS applications in the project?
12	Do you sometimes choose easy/limited solutions to problems in your project, instead of using a better approach?
13	Do you have rework or refactoring cycles for your source code?
14	Do you have any problems that you choose not to fix/solve and only monitor the course/trend over time?
15	Are you familiar with the technical debt metaphor?
16	Do you measure the technical debt that is present in your project, using any tools or processes?
17	Do you mostly monitor or take further action on technical debt issues?
18	How do you decide the managerial activities on the technical debt issues?
19	Do you have any metrics or scale to prioritize the technical debt issues?
20	Do you have any technical debt that is expected to become worse in the future?
21	To what extent does technical debt affect your motivation?
22	Is your management aware of technical debt and are they taking any action?
23	Do you have additional issues that you would like to mention/highlight?
24	Do you think that there's room for improvement for your project/organization?

3.3 Validity Threads

In the design stage of this research, another aspect that we considered was the mitigation strategies to the validity threats. As a qualitative research methodology, the case study approach is subject to certain validity challenges. To ensure the integrity of our findings, we have determined and implemented measures to address potential threats to internal, construct, and external validity.

3.3.1. Construct Validity

Construct validity assesses whether the research tools accurately measure the variables or constructs they aim to evaluate [36]. For our study, it was necessary to ensure our interview questionnaire was capable of identifying SaaS-related TD categories without oversight. To this end, we utilized the ISO/IEC 12207 [38] software life cycle process standard as our foundation. We selected ten processes that spanned both technical and managerial categories from ISO/IEC 12207.

Another element that could influence construct validity is the potential ambiguity in terms and definitions used in the research. In particular, as presented in Table 2, team members demonstrate different levels awareness regarding TD. Recognizing that members of the SaaS project development team might interpret the term "technical debt" in various ways, we reframed our questions. Instead of asking, "What specific technical debt issues do you encounter?", we posed, "What kind of problems do you experience in relation to a specific process?".

3.3.2. Internal Validity

Internal validity considers the degree to which the observed effects in the research are actually caused by the manipulated variables or interventions, rather than some other external factors [36]. In our case study, we considered the possibility that the SaaS development team members might provide answers they believe the researchers want to hear. To mitigate this potential bias, we assured the participants that their responses would be kept confidential, and any data published would be anonymized. We believe this approach minimizes participant bias. Furthermore, instead of prompting for confirmatory answers, we encouraged participants to give detailed, descriptive, open-ended responses to our interview questions, elaborating on problematic situations as comprehensively as possible.

3.3.3. External Validity

External validity concerns the generalizability of the research findings outside the specific context or sample in which the research was conducted [36]. As the study focuses on a single CRM application, the findings might not be generalized to other SaaS applications or other domains. In the findings and results sections, we clearly stated the research findings. These insights are highly valuable; however, they pertain primarily to the specific CRM application studied. Additional qualitative and quantitative research is needed to explore technical debt categories in SaaS application for other domains. We performed the interview with nine employees in a SaaS project and covered a wide diversity of the roles and experience levels within the development team. This has provided a more comprehensive view of the technical debt landscape.

3.4 Case Study Conduct

In this section, we explain the data collection and data analysis stages of the case study.

3.4.1 Data Collection

The data was obtained from semi-structured interviews based on the questions provided in Table 3. We followed the techniques recommended by Patton [39] to ensure comprehensive coverage of every process listed in Table 1, while also retrieving participants' experiences in a flexible way. In total, nine face-to-face interview sessions were conducted, with each taking around 45 minutes. Out of the nine interviewees, two preferred to conduct their interviews in English. Each team member was responsible for at least one SDLC process listed in Table 2. The SDLC processes experienced by the organization, in terms of the number of participants responsible for each process, are detailed below.

At the start of each interview session, the participants were asked about the processes for which they were responsible. To aid in this, the names and purposes of each process were presented to the participants. If a participant requested examples for any process, these were also provided by referring to the activities defined in the ISO/IEC 12207 standard. Table 4 given below shows the SDLC process involvement for each interview participant.

Table 4. SDLC process involvement for each interviewee

Process Name / Interviewee #	1	2	3	4	5	6	7	8	9
Project Planning Process					X			X	X
Project Assessment and Control Process								X	
Stakeholder Needs and Requirements Definition Process			X	X	X		X	X	X
Systems/Software Requirements Definition Process			X	X	X	X	X		

Architecture Definition Process			X	X	X		
Design Definition Process			X	X	X	X	X
Implementation Process	X		X	X	X	X	X
Integration Process	X		X	X	X	X	
Verification Process		X		X	X	X	X
Validation Process		X				X	X

3.4.2 Data Analysis: Coding and Categorization

In analyzing the data, we applied open and axial coding techniques to identify themes, as recommended by [40] and Lochmiller [41]. First, the interview data was first transcribed. The coding and categorization processes were carried out in three iterative and incremental stages.

During the first stage, open coding helped us specify initial TD indicators within the project. Next, we established connections between the codes, focusing on process levels and keywords highlighted by the participants. These stages enabled us to detect TD categories specific to SaaS development, addressing RQ1 and RQ2 respectively. While determining TD categories, we sought patterns that could define TD category themes. To ensure consistent and accurate coding, two review sessions were held: one after the open coding and another at the end of the analysis.
The TD indicators identified from the interview transcripts, along with the resulting TD categories, will be presented in the following section.

4. Findings

SaaS applications are intrinsically complex. They are built to serve clients with diverse needs and designed to be universally accessible. Therefore, they frequently subject to updates and modifications. As businesses race to introduce new features or adapt to market demands, there is often a tendency to make swift decisions, sometimes overlooking long-term implications. Such decisions can inadvertently introduce technical debt.

Table 5 given below presents technical debt indicators and categories for both SaaS applications and generic software development projects. In total, we identified 38 TD categories, and 15 of them are specific to SaaS projects. We also found 104 TD indicators in total, 23 being related only to SaaS projects.
These indicators were derived directly from responses given by interview participants during our case study. Once the indicators were defined, the categories were determined over three iterations. Multiple instances of TD indicators have been removed from Table 5.

The table's first column specifies if the TD category and its associated indicators are common in generic software development or are unique to SaaS projects. This is followed by columns providing the TD category name, its description, and the corresponding TD indicators. The table is organized in ascending order based on the TD categories. SaaS-specific technical debt categories are highlighted with an orange background.

Table 5. TD categories, TD descriptions and associated TD indicators

TD Type SaaS-Specific / Generic	TD Category	Category Description	TD Indicators
Generic	1. Communication Issues	Represents any problems related to conveying or understanding information among team members, stakeholders, or customers.	a. Communication barriers on work tracking b. Miscommunication on customer requests c. Miscommunication with stakeholders/team members
Generic	2. Design Adaptability Challenges	Indicates the difficulties in adjusting designs based on evolving customer requests or requirements.	a. Customer driven volatile design

Generic	3.	Design and Implementation Mismatch	Denotes scenarios where the design and the actual implementation don't align or match as intended.	a. Incomplete implementation wrt design b. Insufficient design maintenance c. Insufficient design verification/validation
Generic	4.	Design Inadequacy	Addresses instances where the design is incomplete, flawed, or based on incorrect assumptions.	a. Design with incomplete/incorrect requirements b. Incomplete design c. Lack of solid design principles
Generic	5.	Design Inefficiency	Covers situations where the design process or outcome is inefficient, overly complex, or doesn't adhere to standard practices.	a. Overengineering
Generic	6.	Design Process Inefficiency	Refers to instances where the design process is flawed or doesn't yield optimal results.	a. Poor design process
Generic	7.	Document Maintenance Deficiency	Indicates a lack of adequate updates, maintenance, or attention to various project or system documents.	a. Insufficient maintenance of design documentation
Generic	8.	Documentation Deficiency	Refers to situations where critical documentation is missing or inadequate, potentially leading to confusion or mistakes.	a. Inadequate architecture definition b. Inadequate design definition c. Insufficient design documentation d. Insufficient requirements documentation e. Missing documentation
SaaS-Specific	9.	Explorative SaaS Challenges	Challenges that emerge when attempting to innovate or implement new methods, features, or processes within a SaaS environment.	a. Exploration of new approaches in SaaS
SaaS-Specific	10.	Inadequate SaaS Support	Occurs where the SaaS provider does not offer enough assistance, guidance, or customer service for users facing issues.	a. Inadequate support from SaaS provider
Generic	11.	Inadequate Technical Debt Management	This theme refers to the challenges of managing, addressing, or accumulating technical debt in the project.	a. Bug-focused releases b. Customer-driven quick fixes c. Delayed problem identification d. Developer-initiated refactoring e. Incomplete solutions f. Rework for fixed problems g. Lack of design refactoring h. Post-verification problem identification i. Quick unconventional fixes j. Refactoring cycles k. Rework time constraints l. Rigid architecture ang coding structure m. Temporary fixes

Generic	12.	Inefficient Work Planning/Estimation	Captures issues related to misestimating work durations, improperly planning tasks, or having conflicts about priorities	a. Decision ambiguities b. Estimation difficulties c. Issue priority conflicts d. Planning conflicts with upper management e. Priority conflicts f. Rising ticket resolution time
SaaS-Specific	13.	Insufficient SaaS Communication	A lack of clarity or adequate information from the SaaS provider. This can be in terms of potential impacts, updates, or changes that might affect users.	a. Insufficient impact information from SaaS
Generic	14.	Issue/Task Description Issues	Highlights problems arising from unclear or incomplete descriptions of tasks or issues.	a. Insufficient issue details
SaaS-Specific	15.	Known SaaS-Specific Issues	Issues that are already identified and recognized with the SaaS provider. These are often listed or documented but remain unresolved.	a. Pre-existing issues with SaaS
Generic	16.	Legacy System Challenges	This theme refers to the difficulties associated with maintaining, upgrading, or working within older systems that may no longer be efficient or compatible with newer technologies.	a. Legacy system maintenance
Generic	17.	Process Adherence Issues	Highlights situations where established processes are not being followed or are incompletely adhered to.	a. Incomplete process adherence b. Ineffective parallel design-implementation cycles c. Neglected backlog maintenance
Generic	18.	Process Definition Ambiguity	Describes situations where processes or criteria are not clearly defined, leading to confusion or misalignment	a. Ambiguous completion criteria b. Undefined checkpoints
Generic	19.	Quality Assurance Inefficiency	Encompasses challenges and inefficiencies in ensuring the quality of code, designs, or implementations.	a. Absent code convention checklist b. Delayed testing cycles c. Ineffective code review d. Self-code verification neglect
SaaS-Specific	20.	Reactive SaaS Adjustments	The need for users to make changes, fixes, or adaptations due to unexpected or uncommunicated changes made by the SaaS provider.	a. Adjustments needed due to SaaS alterations
Generic	21.	Release Management Issues	Denotes difficulties related to managing product releases, updates, or ensuring customers can keep up with the release pace.	a. Mismatch of customer and product needs
Generic	22.	Requirement Issues	Refers to problems related to understanding, gathering, or clearly defining requirements.	a. Inadequate requirement gathering b. Incomplete requirement understanding c. Incomplete requirements d. Requirement consensus lack e. Undefined stakeholder needs f. Vague requirements

Generic	23.	Requirements and Design Mismatch	Highlights situations where there's a disparity between what's required and the resulting design.	a. Insufficient design verification/validation
SaaS-Specific	24.	SaaS Configuration Overhead	Challenges related to the extensive setup, tuning, or customization required on the SaaS side. It indicates that the SaaS might have a complex or not have user-friendly configuration setup.	a. Extensive SaaS-side configurations
SaaS-Specific	25.	SaaS Imposed Limitations	These are strict constraints set by the SaaS provider (such as the number of API calls per hour, storage capacity, customization options, or integration capabilities).	a. Limitations imposed by SaaS governor b. Constraints imposed by SaaS provider c. Inability to exceed SaaS limitations
SaaS-Specific	26.	SaaS-Induced Modifications	Challenges that arise directly due to changes, updates, or decisions made by the SaaS provider. This can include anything from platform changes to the deprecation of certain features.	a. Platform-driven design alterations b. Release-induced incompatibilities c. Deprecation-induced disruptions
SaaS-Specific	27.	SaaS-Platform Specific Limitations	Refers to the constraints that are unique to a specific SaaS platform. Salesforce has its own set of limitations distinct such as platform-specific development languages, modules, or unique architectural elements.	a. Salesforce-specific SaaS constraints
SaaS-Specific	28.	SaaS Testing and Troubleshooting Limitations	Difficulties faced during the testing or troubleshooting phases because of constraints or peculiarities in the SaaS.	b. Testing and troubleshooting issues due to SaaS limits c. SaaS-related troubleshooting difficulty
SaaS-Specific	29.	Setup and Replication Challenges	Difficulties related to creating, setting up, or duplicating SaaS environments. This can range from initial environment setup to trying to reproduce specific conditions for testing or troubleshooting.	a. Time-consuming SaaS environment setup b. Difficulty in replicating live environments c. Challenges in customer environment replication
SaaS-Specific	30.	Shared Accountability	This theme pertains to situations where both the software vendor and the development team share the blame for issues or conflicts. It emphasizes that responsibility is not solely on one party.	a. Accountability on both vendor and development team
Generic	31.	Standards and Best Practices Violation	Represents scenarios where industry standards, best practices, or internal guidelines aren't followed.	b. Absence of architectural standards c. Absent unit test standards d. Best practice implementation neglect e. Flawed testing process f. Incomplete unit tests g. Lack of a defined design process h. Neglect of solid design principles i. Neglected code conventions

				j. Non-conforming code practices k. Undefined test steps
SaaS-Specific	32.	Technological Diversity Challenges	Issues that arise due to using a mix of different technologies, tools, or platforms. This might be the integration of varied systems or the problems that arise from them.	a. Diverse technology hindrances
Generic	33.	Testing Limitations	Deals with challenges in the testing phase, including time restrictions, missing tests, or lack of automation.	a. Insufficient testing b. Manual testing overhead c. Missed edge case testing d. Missing unit test coverage e. Time-restricted testing
Generic	34.	Time Management Issues	Encompasses challenges related to deadlines, frequent releases, or general time constraints.	a. Deadline pressures b. Frequent release pressure c. General time constraints d. Missed deadlines
SaaS-Specific	35.	Trade-offs due to SaaS Architecture	Recognizing both the advantages and disadvantages of using a particular SaaS due to its design or structure, especially with architectures like multi-tenancy.	a. Multi-tenancy-related SaaS trade-offs b. SaaS limits acknowledged, but seen as not positive
Generic	36.	Verification Inefficiency	Highlights inefficiencies or challenges in the verification phase, whether it's customer or internal verifications.	a. Customer-identified issues b. Verification difficulties c. Verification-identified issues
Generic	37.	Workaround Dependencies	Refers to reliance on workarounds, be it due to platform constraints, specific requests, or other limitations.	a. Limitation workarounds b. Platform-specific workarounds c. Request-specific workarounds
SaaS-Specific	38.	Workaround for SaaS Limitations	The effort and methods employed to find alternative solutions or bypasses for issues or limits set by the SaaS provider.	a. Employing workarounds for SaaS limits/issues

5. Discussions

SaaS applications have recently gained significant interest due to their adaptability and scalability characteristics. These applications can serve to many clients with varying needs. However, this flexibility can contribute to increased TD levels in SaaS projects. In this study, we identified technical debt categories specific to SaaS projects. Our study was not limited to SaaS; we also identified categories of technical debt that apply to generic software development. The SaaS-specific TD categories identified in this study are novel, whereas the generic categories are consistent with prior literature on technical debt. However, these generic TD categories are explained at a finer level of abstraction, thereby providing a deeper understanding of the associated challenges.

5.1 SaaS-Specific Technical Debt Indicators and Categories

Technical debt is a well-discussed topic in general software development, but SaaS environments bring their own set of challenges and characteristics. The limitations imposed by SaaS platforms play a significant role in finding

quick solutions and making swift decisions which leading to technical debt. The technical debt categories we specified in this study for SaaS development are;

1. Explorative SaaS Challenges
2. Inadequate SaaS Support
3. Insufficient SaaS Communication
4. Known SaaS-Specific Issues
5. Reactive SaaS Adjustments
6. SaaS Configuration Overhead
7. SaaS Imposed Limitations,
8. SaaS-Platform Specific Limitations
9. SaaS-Induced Modifications
10. SaaS Testing and Troubleshooting Limitations
11. Setup and Replication Challenges
12. Shared Accountability
13. Technological Diversity Challenges
14. Trade-offs due to SaaS Architecture
15. Workaround for SaaS Limitations

Within the technical debt landscape, SaaS-related TD indicators stand out mainly from the characteristics of SaaS platforms. These indicators that range from configuration overheads to platform-specific limitations highlight the balance developers need to ensure between using the benefits of SaaS and managing its inherent constraints.
The *Trade-offs due to SaaS architecture*, *SaaS imposed limitations*, *SaaS platform specific limitations* and *Technological Diversity Challenges* categories are basically related to strict nature of the SaaS platforms. The architectural decisions made during the design and implementation phases of SaaS solutions often lead to inherent trade-offs. By its nature, SaaS usually adopts architectures like multi-tenancy [16]. While this approach offers scalability and maintenance advantages, it can also introduce constraints. For instance, while one tenant (i.e. the client organization) might benefit from a specific feature addition or update, another might find it disruptive. Moreover, customization becomes challenging when trying to adjust to the diverse needs of various tenants without affecting the base architecture. The architectural decisions to use such shared resources might lead to performance bottlenecks or security concerns in SaaS projects. Recognizing these trade-offs is essential to ensure that the advantages of a SaaS solution outweigh its potential limitations. Developers and decision-makers need to be continually be aware of these trade-offs to make informed choices, ensuring that the software remains resilient, secure, and effective.

The SaaS development landscape also involves multiple stakeholders, including SaaS providers, developers, third-party integrators, and end-users [17]. This dynamic nature can lead to situations where no single entity has full ownership of a technical challenge or its resultant debt. The *Shared Accountability* category highlights these blurred lines of responsibility. For instance, a limitation imposed by the SaaS provider might lead the development team to implement a workaround. If this workaround causes performance issues, it would not be clear who is responsible for the issue. This shared accountability can complicate the resolution process and make it challenging to specify the root cause of issues. For effective technical debt management in SaaS projects, it is crucial to establish clear lines of responsibility and communication pathways between all involved parties.

In SaaS projects, the complicated relationship between the *Inadequate SaaS Support, Insufficient SaaS Communication,* and *Known SaaS-Specific Issues* categories highlights the shared challenges that developers face. Although there is a close and symbiotic dependence between SaaS providers and developers, this relationship may lean towards a state of imbalance, often leading to the accumulation of technical debt. Even if SaaS providers communicate well, SaaS developers might still have to deal with known problems with the platform. Knowing about these issues does not always make them easy to avoid.

The *Explorative SaaS Challenges* technical debt occurs when developers try innovated operations within the SaaS environment. Whether employing a novel API or integrating a new feature, uncertainties associated with these actions may cause the development team to adopt shortcuts. The absence of established SaaS development guidelines may increase the technical debt levels.

The categories of *Inadequate SaaS Support* and *Insufficient SaaS Communication* often interact hand-to-hand. Insufficient support from SaaS providers may be in the form of limited guidance or insufficient documentation. This can increase the problem by forcing developers into unexpected actions. Especially, poor SaaS provider communication on platform updates, new features or other changes may result in misinterpreted features or misunderstood functionalities, both of which contribute to an increase in technical debt.

Similarly, the *Known SaaS-Specific Issues* and *Reactive SaaS Adjustments* TD categories represent potential challenges that are recognized but may not be readily resolved. Developers may search for quick, reactive code changes to overcome these issues. While such adjustments serve as temporary solutions, they inadvertently contribute to the accumulation of technical debt.

The *SaaS Configuration Overhead, Setup and Replication Challenges* and *SaaS-Induced Modifications* TD categories refer to the additional workload required by the configuration and customization of a SaaS platform to meet particular needs. Especially, a complex configuration structure without sufficient guidance can contribute to the accumulation of technical debt. Lastly, the categories of *SaaS Testing and Troubleshooting Limitations* TD category points to restrictions in effective software verification processes. Impediments in setting up suitable test environments or replicating conditions can lead to less rigorous testing, thereby causing further technical debt.
A comprehensive understanding of these areas is critical for effective management of technical debt which is basically comprised of seven stages: Identification, Measurement, Prioritization, Resolution, Monitoring, Communication, and Prevention [7][42]. Awareness of potential technical debt is critical for a well-executed TD management life cycle. For instance, awareness on the *SaaS Configuration Overhead* technical debt category would enable teams to recognize specific type of indicators, making timely and accurate categorization in a SaaS context. This TD categorization will aid in the efficient allocation of resources, accurate issue prioritization for resolving TDs, and facilitate the development of resolution strategies.

5.3 Generic Software Technical Debt Categories

Our case study revealed 81 distinct indicators associated with 23 generic technical debt categories, as illustrated in Figure 1. This graph, which includes redundant occurrences, aims to capture the frequency of each technical debt category. From the data, we can note that the top three technical categories are *SaaS-Related Technical Debt* (elaborated in Section 5.1), *Inadequate Technical Debt Management*, and *Violations of Standards and Best Practices*, with 23, 13, and 11 instances respectively.

While Alves *et al.* [28] identified 13 high-level technical debt categories based on a systematic literature review, our study takes this a step further. We provide a deeper level of categorization associated with indicators. We argue that just referencing process names in TD categories does not suffice; each technical debt category should clearly hint potential challenges.

In addition to SaaS related TD categories, our study has highlighted a new area of concern in technical debt: *Inadequate Technical Debt Management*. Neglecting to manage technical debt effectively can have a snowball effect, leading to even more debt. The importance here is not just recognizing how technical debt accumulates but also understanding how to manage it properly. We recommend teams not only be cautious about occurring technical debt but also prioritize its effective management through regular audits, targeted sprints, and long-term planning. Lastly, we find categories like 'Verification Inefficiency' and 'Testing Limitations' to be pivotal. These emphasize the crucial role that verification and testing play in managing technical debt. A lack of rigor in these areas can worsen existing technical problems, making them increasingly complex to identify and resolve.

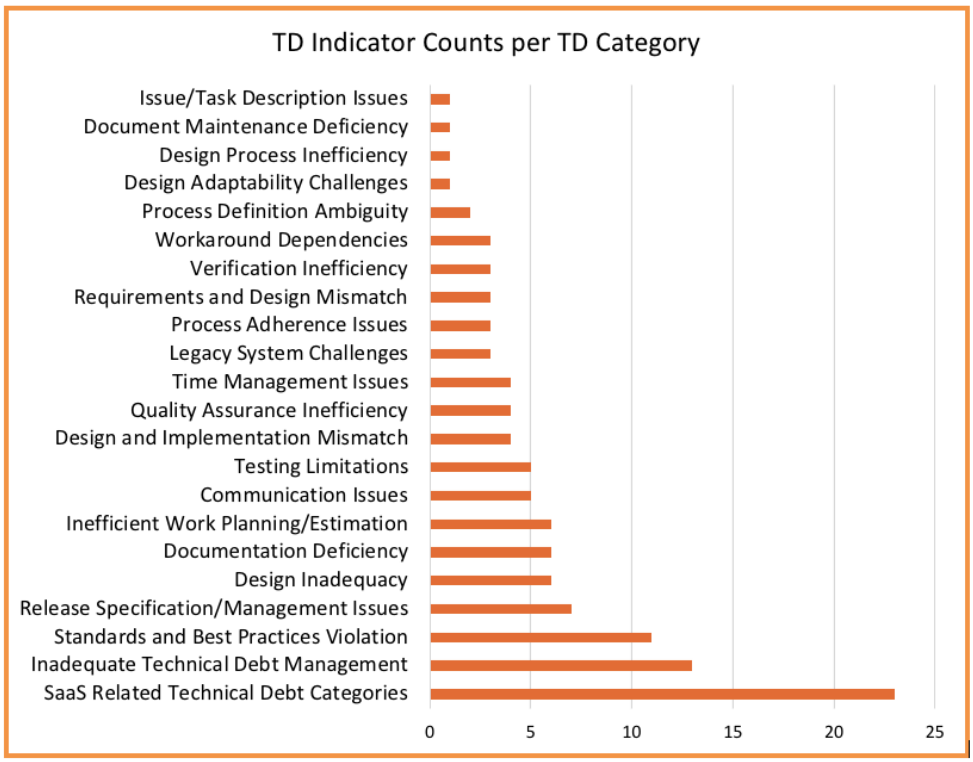

Figure 1. TD indicator counts per TD category

6. CONCLUSIONS

While SaaS development holds significant potential and benefits, it comes with its unique challenges. Such challenges highlight the need for a deeper understanding and strategic approach to SaaS development.
In this study, we approached to the problem from technical debt perspective and aimed to understand SaaS-specific technical debt indicators and categories. We performed an exploratory case study by conducting interviews with nine employees in a SaaS vendor. Based on the data we collected from interviewees, we identified 15 distinct SaaS-specific technical debt categories and 23 corresponding indicators. Additionally, we observed 81 generic software-specific technical debt indicators categorized into 23 types.

Addressing SaaS-specific categories is not just about acknowledging their existence, our study points out several categories that are unique to SaaS development. This emphasizes the distinct challenges of SaaS development and the need for specific strategies to manage its technical debt.

These granular categorizations are valuable in terms of enabling developers to identify, prioritize, tackle and prevent SaaS specific issues effectively. Our findings highlight the multifaceted nature of technical debt and stress the need for an encompassing approach to understanding and managing it. Therefore, our study makes a significant contribution for effective management of technical debt in SaaS applications.
Future studies can be performed with multiple case studies and on understanding the impact of SaaS technical debt on the overall software development lifecycle.

References

[1] Kendall, J. E., & Kendall, K. E. 1993. Metaphors and methodologies: Living beyond the systems machine. *MIS quarterly*, 149-171.

[2] Cunningham, W. 1992. The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger, 4*(2), 29-30.

[3] Kruchten, P., Nord, R. L., & Ozkaya, I. 2012. Technical debt: From metaphor to theory and practice. *IEEE software,* 29(6), 18-21.

[4] Codabux, Z., Williams, B. J., Bradshaw, G. L., & Cantor, M. 2017. An empirical assessment of technical debt practices in industry. *Journal of software: Evolution and Process, 29*(10), e1894.

[5] Tom, E., Aurum, A., & Vidgen, R. 2012. A consolidated understanding of technical debt, *European Conference of Information Systems Proceedings*, https://core.ac.uk/reader/301355650.

[6] Zazworka, N., Spínola, R. O., Vetro', A., Shull, F., & Seaman, C. 2013. A case study on effectively identifying technical debt. In Proceedings of *the 17th International Conference on Evaluation and Assessment in Software Engineering* (pp. 42-47).

[7] Li, Z., Avgeriou, P., & Liang, P. 2015. A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101, 193-220.

[8] Rios, N., Spínola, R. O., Mendonça, M., & Seaman, C. 2020. The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil. *Empirical Software Engineering*, 25, 3216-3287.

[9] Patidar, S., Rane, D., & Jain, P. 2012. A survey paper on cloud computing. In *2nd International Conference on Advanced Computing & Communication Technologies* (pp. 394-398). IEEE.

[10] Sun, W., Zhang, K., Chen, S. K., Zhang, X., & Liang, H. 2007. Software as a service: An integration perspective. In Service-Oriented Computing Proceedings, 5, 558-569, Springer Berlin Heidelberg.

[11] Benlian, A., & Hess, T. 2011. Opportunities and risks of software-as-a-service: Findings from a survey of IT executives. *Decision support systems*, 52(1), 232-246.

[12] Gartner Press Release 2022, https://www.gartner.com/en/newsroom/press-releases/2022-10-31-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-reach-nearly-600-billion-in-2023, (Access Date: 28.07.2023)

[13] Turner, M., Budgen, D., & Brereton, P. 2003. Turning software into a service. *Computer*, 36(10), 38-44.

[14] Vidhyalakshmi, R., & Kumar, V. 2014. Design comparison of traditional application and SaaS. In *2014 International Conference on Computing for Sustainable Global Development (INDIACom)* (pp. 541-544). IEEE.

[15] Cusumano, M. 2010. Cloud computing and SaaS as new computing platforms. Communications of the ACM, 53(4), 27-29.

[16] Chou, S. W., & Chiang, C. H. 2013. Understanding the formation of software-as-a-service (SaaS) satisfaction from the perspective of service quality. *Decision Support Systems*, *56*, 148-155.

[17] Sadiku, M. N., Musa, S. M., & Momoh, O. D. 2014. Cloud computing: opportunities and challenges. *IEEE potentials*, *33*(1), 34-36.

[18] Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., ... & Zazworka, N. 2010. Managing technical debt in software-reliant systems. In Proceedings of the FSE/SDP workshop on Future of software engineering research (pp. 47-52).

[19] Besker, T., Martini, A., & Bosch, J. 2017. The Pricey Bill of Technical Debt: When and by Whom will it be Paid?. *2017 IEEE International Conference on Software Maintenance and Evolution* (ICSME).

[20] Suryanarayana, G., Samarthyam, G., & Sharma, T. 2014. Refactoring for software design smells: managing technical debt. Morgan Kaufmann.

[21] Falessi, D., & Kazman, R. 2021. Worst smells and their worst reasons. *In IEEE/ACM International Conference on Technical Debt* (TechDebt) (pp. 45-54). IEEE.

[22] Ramač, R., Mandić, V., Taušan, N., Rios, N., Freire, S., Pérez, B., ... & Spinola, R. 2022. Prevalence, common causes and effects of technical debt: Results from a family of surveys with the IT industry. *Journal of Systems and Software*, 184, 111114.

[23] Zazworka, N., Izurieta, C., Wong, S., Cai, Y., Seaman, C., & Shull, F. 2014. Comparing four approaches for technical debt identification. *Software Quality Journal*, 22(3), 403-426.

[24] Iuliia, G. 2017. Technical Debt Management. In Russion Software Development Companies. Master's Thesis. St. Petersburg University Graduate School of Management.

[25] Alzaghoul, E., & Bahsoon, R. 2013. CloudMTD: Using real options to manage technical debt in cloud-based service selection. In 2013 4th International Workshop on Managing Technical Debt (MTD) (pp. 55-62), IEEE.

[26] Klinger, T., Tarr, P., Wagstrom, P., & Williams, C. 2011. An enterprise perspective on technical debt. In Proceedings of the 2nd Workshop on managing technical debt (pp. 35-38).

[27] Nugroho, A., Visser, J., & Kuipers, T. 2011. An empirical model of technical debt and interest. In Proceedings of the 2nd workshop on managing technical debt, 1-8.

[28] Alves, N. S., Ribeiro, L. F., Caires, V., Mendes, T. S., & Spínola, R. O. 2014. Towards an ontology of terms on technical debt. In 6th International Workshop on Managing Technical Debt (pp. 1-7). IEEE.

[29] Ramasubbu, N., & Kemerer, C. F. 2016. Technical debt and the reliability of enterprise software systems: A competing risks analysis. *Management Science*, 62(5), 1487-1510.

[30] Mcconnell, S. 2008. Construx,. Available from: https://www.construx.com/resources/whitepaper-managing-technical-debt/. (Access Date: 23.08.2023).

[31] Fowler, M. 2009. Technical Debt Quadrant, Available from: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html. (Access Date: 31.07.2023).

[32] Kumar, S., Bahsoon, R., Chen, T., & Buyya, R. 2019. Identifying and estimating technical debt for service composition in SaaS cloud. *In 2019 IEEE International Conference on Web Services* (ICWS) (pp. 121-125). IEEE.

[33] Agarwal, P. 2011. Continuous Scrum: Agile management of SAAS products. *In Proceedings of the 4th India Software Engineering Conference* (pp. 51-60).

[34] Liu, F., Tong, J., Mao, J., Bohn, R., Messina, J., Badger, L., & Leaf, D. 2011. NIST cloud computing reference architecture. *NIST special publication*, 500(2011), 1-28.

[35] Baxter, P., & Jack, S. 2008. Qualitative case study methodology: Study design and implementation for novice researchers. *The qualitative report*, *13*(4), 544-559.

[36] Runeson, P., & Höst, M. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, *14*, 131-164.

[37] Easterbrook, S., Singer, J., Storey, M. A., & Damian, D. 2008. Selecting empirical methods for software engineering research. Guide to advanced empirical software engineering, 285-311.

[38] ISO/IEC/ IEEE 12207 2017. Systems and software engineering — Software life cycle processes.

[39] Patton, M. Q. 2002. Qualitative interviewing, in *Qualitative Research and Evaluation Methods*. Thousand Oaks, CA, USA: SAGE, pp. 344–347.

[40] Williams, M., & Moser, T. 2019. The art of coding and thematic exploration in qualitative research. *International Management Review*, 15(1), 45-55.

[41] Lochmiller, C. R. 2021. Conducting thematic analysis with qualitative data. *The Qualitative Report*, *26*(6), 2029-2044.

[42] Kruchten, P., Nord, R., & Ozkaya, I. 2019. Managing Technical Debt: Reducing Friction in Software Development. Software Engineering Institute.