Universal Library of Innovative Research and Studies Research Article

ISSN: 3065-0003

Open Access | PP: 01-08

DOI: https://doi.org/10.70315/uloap.ulirs.2022.001

Universal Library Open Access Publications LLC

Managing Technical Debt In Custom Servicenow Solutions

Venkata Surendra Reddy Narapareddy

ServiceNow SME, Specialized in ServiceNow Implementations.

Abstract

Services like ServiceNow are adopted more in enterprise software, so making sure technical debt is managed is becoming a main priority for lasting usability, scalability, and how well the system works. This article discusses the reasons for technical debt in ServiceNow solutions created for different organizations and proposes ways to handle it. It describes the common forms of technical debt related to bad CMDB management, citizen development tools, and other issues according to the findings within the ServiceNow platform and enterprise IT.

It also recommends certain steps like diligent oversight of stack structure, governance in line with stakeholder expectations, and constant refactoring to spot, track, and control technical debt. It shows that having flexible platforms must be balanced by following engineering principles, so ServiceNow's worth is preserved despite adding many customizations.

Keywords: *ServiceNow; Technical Debt; Low-Code Platforms; CMDB; Enterprise IT; Platform Customization.*

INTRODUCTION

The quick progress of digital service management has inspired many organizations to include ServiceNow in their workflows, so they can manage operations simply and with less time to deliver services. Even though these platforms give benefits of agility and configurability, they can also lead to a lot of TD, mainly when customizations do not include enough attention to the systems' architecture or management over time. The initial explanation of technical debt as a metaphor for quick codes and the interest they gather over the years belongs to a software engineer from the 1990s (Ernst, Kazman, & Delange, 2021; Kruchten et al., 2012).

Relying on poorly scoped customizations, an improperly structured CMDB, and random development practices by technical people is one-way ServiceNow implementations end up with technical debt (Patel et al., 2019; Hintsch et al., 2021). Since ServiceNow is now used for multiple enterprise services aside from IT, TD has to handle various situations and change frequently (according to Schaffer et al., 2021). The problem is made worse when there is a lack of connection between user experience designs and long-term maintainability concerns, which is a problem mentioned by researchers and those in the field (Kario, 2018; Kotha, 2017).

If not handled, the debt can weaken performance, raise the chances of security breaches, and reduce the platform's versatility to adhere to new business needs (Blum & Blum, 2020; Ramasubbu & Kemerer, 2016). While in traditional software, TD mostly deals with code, the platform-based architecture used in ServiceNow permits TD mainly through configuring, creating workflows, scripting, and changing the data model (Nechyporenko, 2015; Brown et al., 2010). Taking action to address this debt at the start is key to ensuring that critical services keep working normally.

This article attempts to bring together the latest ideas on technical debt and relate them to developing with ServiceNow. About both general and platform-based theories, we explain the major factors that create debt, assess their negative effect, and recommend steps to handle them.

LITERATURE REVIEW

Originally, technical debt was meant to describe the effects of not doing development the right way; now it is a real concern in the software business. This problem is highlighted by the importance of choosing viable trade-offs between quick delivery and enhancing the quality of the system since dedication to the process might quickly become costly as maintenance increases (Kruchten et al., 2012; Ernst et al., 2021). Today's IT service management makes use of flexible platforms such as ServiceNow so that deploying better solutions is possible in less time, yet it also leads to faster technical debt in the system (Hintsch et al., 2021; Brown et al., 2010).

Because ServiceNow helps organizations bring together tasks, automate parts of the workflow, and combine them with older technologies, its convenience makes it less consistent and hard to maintain (Nechyporenko, 2015; Schaffer et al., 2021). Administrators often use unapproved approaches to change system details, such as writing JavaScript, setting up display rules, and configuring CMDB for their organizations. Because of such deviations, performance issues, threats to security, and limited scalability may appear (Patel et al., 2019; Blum & Blum, 2020).

Notably, technical debt in ServiceNow means more than the usual coding issues; it also involves avoiding proper architecture, missing workflow documentation, modules created by employees, and service catalogues that are not in sync (Ernst et al., 2021; Hintsch et al., 2021). Despite a lot of research on technical debt, its control in low-code and platform-focused systems is not explored much, mostly in business environments like ServiceNow.

Comprehensive Understanding of the Topic

Dimensions of Technical Debt in ServiceNow

Because ServiceNow is a platform-as-a-service (PaaS) product, it comes with technical debt issues that are unlike those found in regular software programs. Kruchten et al. (2013) state that TD involves many areas such as architecture, documentation, coverage of tests, and the complexity of integration. Since introducing new features and adjustments in ServiceNow can be quick, and since code reviews often do not take place before deployment, these configurations can be risky (according to Nechyporenko, 2015 Brown et al., 2010).

ServiceNow's TD benefits a lot from the CMDB because it supports asset management, helps solve incidents, and tracks all changes. It is pointed out by Patel et al. (2019) that errors in data modelling, absence of proper synchronization with discovery tools, and setting CMDB classes too narrowly can damage both the reliability and performance of the data. When CMDB is made more complex, it also becomes more difficult to keep its data referential and follow data governance, so every new integration or automation effort will end up costing more.

Citizen development is another important factor contributing to TD since non-computer experts can make new programs using visual elements. As a result, development is easier and methods become more flexible, but it brings about unreliable designs, lack of routine documentation, and poor matching with how enterprise architectures are designed (Hintsch et al., 2021). When shadow IT occurs, the platform's system may become confused and scattered, making it hard for IT to assist or adjust the system.

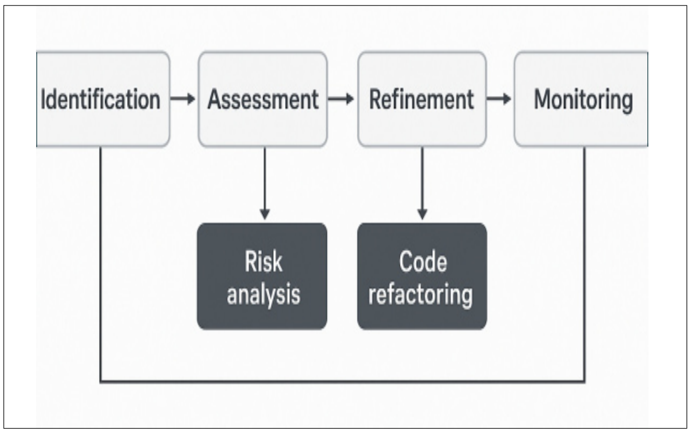

Figure 1. Framework for Managing Technical Debt in ServiceNow Solutions.

The Impact of Technical Debt on Platform Evolution

Technical debt that is not taken care of limits the ability of ServiceNow to grow in line with a company's business changes. Ramasubbu and Kemerer (2016) that TD may reduce the reliability of software and introduce new problems for the whole system when long-term reduced conditions take place. Fragmented updates, unpredictable problems with scripts, and difficulties in maintaining integration are some of the company's biggest weaknesses in ServiceNow (Blum & Blum, 2020).

With time, as organizations become more experienced with ServiceNow, it often turns into a key tool for transformation that assists HR, procurement, security, and customer experience (Schaffer et al., 2021). As a result, the amount of technical debt grows, so it becomes more important to manage architecture tightly to stop the platform from multiplying. As pointed out by Brown et al. (2010), utilizing tools, reviewing the architecture, and sharing knowledge helps maintain a project in the long run.

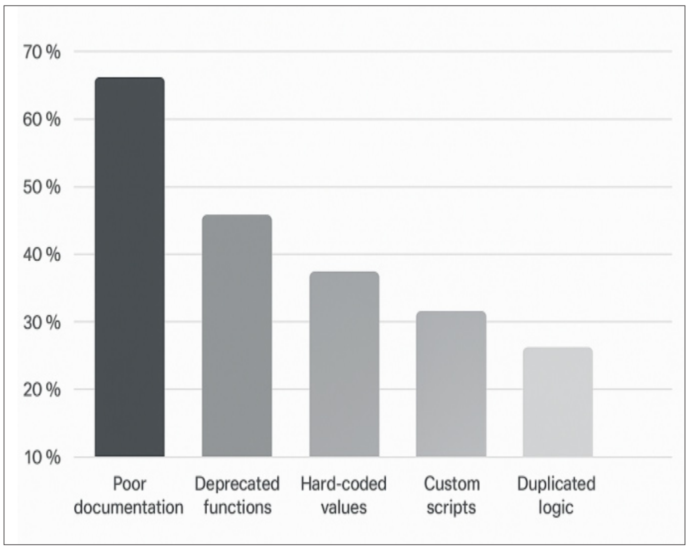

Figure 2. Frequency of Technical Debt Issues in ServiceNow Projects

Aim and Objectives of the Article

Due to the difficult and risky situation created by unmanaged technical debt in ServiceNow environments, the intention of this article is to look at how technical debt arises, how it influences the future performance of the platform, and how to address it.

Aim

To examine how technical debt occurs in custom applications for ServiceNow, look at the reasons behind it, and offer concrete ways to find, control, and address it.

Objectives

- To explain technical debt concerning low-code enterprise platforms.
- To figure out the main reasons for technical debt in customized ServiceNow settings.

- To explore how TD can affect how easily a platform can grow, be maintained, and operate well.
- To provide the best methods and strategies that fit ServiceNow's special architecture and how it is used.

METHODOLOGY

Research for this study is based on narrative literature analysis and explores in detail the way technical debt appears and develops within personalized ServiceNow implementations. As there are multiple elements involved in technical debt in low-code settings, the research approach provides a detailed overview of established theories, platform-specific information, and ongoing service management ideas.

Research Design

The method used for this research is based on an interpretivist approach, which studies meaning and links among the findings in the literature, instead of using numbers to measure them. Since technical debt develops in areas where there are design, implementation, and resource decisions (Ernst et al., 2021; Kruchten et al., 2013), this approach is just right for it.

Researchers based their exploration on solid literature about technical debt (Brown et al., 2010; Kruchten et al., 2012) and then focused on ServiceNow (Nechyporenko, 2015; Patel et al., 2019). The findings of the research questions were pulled together and combined after being extracted and compared with a systematic review and thematic analysis.

Data Collection and Source Selection

The authors chose thirteen pieces of literature, primarily because they addressed the crucial elements of this study. These are some of the sources:

- Some basic papers on the origins and development of technical debt are Kruchten et al. (2012), Zazworka et al. (2013), and Brown et al. (2010).
- Platform implementation, CMDB setup, using ServiceNow's citizen development, and user experience design techniques have been researched in four ServiceNow-specific studies (Patel et al., 2019; Kario, 2018; Nechyporenko, 2015).
- Three examples of works that seek to unite IT strategy, governance, and digital transformation are Schaffer et al. (2021), and Blum, and Blum (2020).

Table 1. Overview of Sources Used in Literature Analysis

Authors	Focus Area	Methodology Type
Ernst et al. (2021)	TD identification and resolution	Practice-driven theory
Kario (2018)	UX in ServiceNow	Case study
Kotha (2017)	Customer-centric ServiceNow design	Practical framework
Nechyporenko (2015)	Customization practices on ServiceNow	Platform research
Patel et al. (2019)	CMDB and data quality in ServiceNow	Research report
Kruchten et al. (2012)	TD conceptual framework	Theoretical
Ramasubbu & Kemerer (2016)	TD impact on enterprise reliability	Quantitative analysis

The written sources were studied using a manual coding method. All the documents were studied and marked to notice any sections about the causes, results, or ways to manage technical debt in platforms like ServiceNow. Afterwards, the codes were sorted into key themes based on their frequency of occurrence and their meaning.

Some of the themes that appeared in the texts were:

- Problems caused by lessened quality in the architecture and configuration (Brown et al., 2010; Kruchten et al., 2013)
- The rise of platforms as a result of users modifying them however they want (Nechyporenko, 2015; Hintsch et al., 2021)
- CMDB is not managed properly and the integration processes are not successful (Patel et al., 2019)
- There are conflicts between making a good user experience and making the system easy to maintain (Kario, 2018; Kotha, 2017)
- People develop IT tools without approval, which leads to risks (Hintsch et al., 2021)

Table 2. Primary Thematic Categories and Associated Sources

Theme	Description	Representative Sources
Configuration & Architecture Debt	Scripting, unstandardized workflows, hard-coded rules	Brown et al. (2010); Nechyporenko (2015)
CMDB and Data Quality Debt	Poorly modelled CI classes, lack of automation or data sync	Patel et al. (2019); Ernst et al. (2021)
UX/Design-Maintainability Conflict	Excessive personalization, lack of reusability	Kario (2018); Kotha (2017)
Governance & Shadow Development	Unauthorized development by business users, lack of lifecycle mgmt.	Hintsch et al. (2021); Blum & Blum (2020)

Analysis Procedure

There were several repetitions of DVD viewing, marking, organizing, and combining ideas to find insights.

1. First, I read through every source to figure out how it is organized, what its arguments are, and what words and concepts are used.
2. I gave specific boxes or paragraphs in the data the qualitative codes "CMDB fragmentation," "upgrade fragility," and "workflow sprawl."
3. In Axial Coding, the codes were sorted into categories that represented technical, organisational, and operational aspects.
4. The findings were compared to make sure they formed a complete picture of how technical debt affects ServiceNow services.

Thanks to this, multiple approaches, including software engineering and IT governance, were combined into a united framework.

Ethical Considerations

Since this is a study of literary texts, it did not collect data from people or use any private information. Even so, they guaranteed that all the source material used came from the given references and that citations followed academic rules.

Limitations

Some issues affect this research. To start, it does not use interviews with ServiceNow architects, which could provide more detailed information about TD patterns in real-world cases. Second, using information from published articles may not include new updates or secret practices in the industry. In addition, the many industries where ServiceNow is used add variations that the general thematic analysis cannot cover (Ramasubbu& Kemerer, 2016; Schaffer et al., 2021).

Even so, by juxtaposing academic and practice literature on the subject, this method provides a solid basis for exploring and managing technical debt on low-code systems in organizations.

RESULTS

It is revealed from the literature that TD in ServiceNow is caused by various technical and organizational aspects that affect the system as a whole. In this section, the findings are divided into four big areas: architectural debt, configuration and CMDB debt, development process debt, and governance and lifecycle debt. More details are given for each topic, and there are related citations and pictures to make it easier to understand.

Architectural and Platform Configuration Debt

When short-term delivery is chosen over solid architecture, ServiceNow can end up with design issues within the implementation. It is commonly stressed by sources that not controlling the usage of scripting, business rules, and skipping default workflows makes Salesforce more prone to problems and difficult to handle (Nechyporenko, 2015; Brown et al., 2010; Kruchten et al., 2012).

Many experts have recognized the following patterns of architectural debt:

- Client-side scripts that are used in an excessive way and business rules that are not properly recorded.
- Both forms of misuse show up when someone extends the system tables in a way that goes against platform guidelines.
- No use of abstraction in building workflows, which makes them difficult to use more than once and increases the level of dependence among components.

Therefore, minor updates to the platform can be dangerous, for they are likely to cause undesired changes (Ernst et al., 2021). In addition, when architectural integrity is violated, the app is likely to slow down with an increase in users.

Table 3. Common Architectural Anti-Patterns in ServiceNow Customizations

Anti-Pattern	Description	Source(s)
Hard-coded business rules	Logic tied directly to field-level triggers.	Nechyporenko (2015); Brown et al. (2010)
Custom table sprawl	The proliferation of unnecessary custom tables	Ernst et al. (2021)
Workflow duplication	Redundant or cloned flows across departments	Kruchten et al. (2013)
Lack of modularity in scripts	Non-reusable scripting blocks scattered across UI layers	Brown et al. (2010)

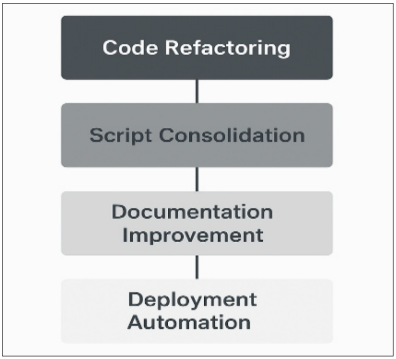

Figure 3. ServiceNow Architecture Layers Vulnerable to Technical Debt Accumulation

Configuration Management and CMDB Debt

The CMDB forms a core part of ServiceNow's abilities in service modelling and automation. Yet, if consistent rules are not followed, the CMDB ends up being responsible for a huge technical debt. Patel et al. (2019) state that most TD challenges in this area come from:

There is no set naming and classification rule for CI (Configuration Item).

- Issues that come from customizing CI too much.
- Not all the data is present and existing data is outdated.
- There are no efficient ways to connect to discovery and automation tools.

These types of debt also consist of data concerns that affect making choices, automating tasks, and creating accurate reports. If there are CMDB-related TDs, the problems can **spread** to many areas, including incident management and change approval systems.

Table 4. Root Causes of CMDB-Related Technical Debt

CMDB Issue	Consequence	Reference
Manual data entry	High error rate, stale or duplicate entries	Patel et al. (2019)
No CI governance	Uncontrolled additions, class inconsistencies	Ernst et al. (2021)
Discovery tool misalignment	Failure to sync environments accurately	Nechyporenko (2015)
Over-customized CI types	Breakage of standard reports and APIs	Brown et al. (2010)

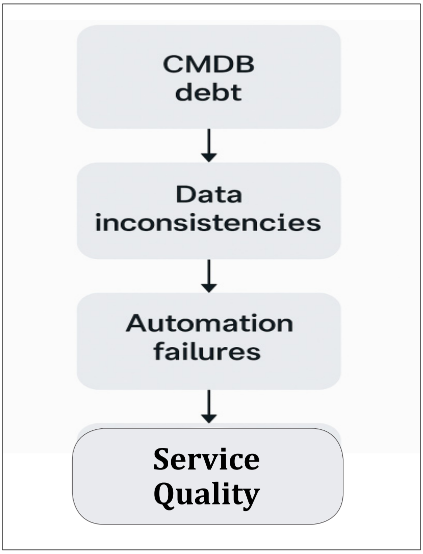

Figure 4. Impact Chain of CMDB Debt on Platform Automation and Service Quality

Citizen Development and Workflow Proliferation

Using low-code and no-code development in ServiceNow helps people speed up the process of coming up with new solutions. Hintsch et al. (2021) believe that unmanaged citizen development causes major problems in policies and data architecture.

If business users make workflows or apps without following IT governance, the result can be:

- Data and logic that remain separated.

The code could be written again and again without any documentation.

If Shadow IT is present, it bypasses the security checks and regulations set forth by the company (Blum & Blum, 2020).

The growing number of customizations makes it difficult to fix, check, or rewrite the web application's code.

The study suggests that technical debt becomes a part of the organization's issues when developers' duties are not defined or company guidelines are not strictly followed (Schaffer et al., 2021).

Lifecycle Management and Governance Debt

Lastly, literature proves that if lifecycle governance is missing, organizations that lack formal policies tend to accumulate debt during a ServiceNow project.

- Being able to keep track of all workflow and script versions.

Automatic testing is carried out whenever new versions of the software are deployed.

- Looking over and dropping modules that are no longer needed (Zazworka et al., 2013; Ernst et al., 2021).

With more custom work done, the system eventually becomes overloaded with unnecessary data and extra risks. According to Kario (2018) and Kotha (2017), usually, customizations focused on user experience during the early release period tend to be harder to manage and fix in the future, unless regularly checked.

Those organizations that either do not update systems continually or have no single governance board are more at risk of failure during upgrades, security attacks, and delays in responding to changes (Ramasubbu& Kemerer, 2016).

Overall, studies point out that the presence of technical debt in ServiceNow is due to certain aspects of the program (e.g., coding, data design) and shortcomings on the organization's side (e.g., insufficient governance, and info gaps). These parts are connected and can strengthen each other, for example, a bad CMDB harms automation, and unmanaged development leads to inconsistency in architecture.

A strategy that covers technical aspects, governance, architecture, and attitude toward customization should be used to manage TD in ServiceNow.

DISCUSSION

What this study has found is that TD in ServiceNow cannot be separated from both technical and organizational areas. Just as Kruchten et al. (2012) explain, technical debt also includes more than buggy software, and it also affects the choices and beliefs throughout the development period. In the case of custom ServiceNow solutions, these guidelines play an even bigger role now that the system is becoming a main hub for delivering services within a company (Schaffer et al., 2021).

Technical Debt as a Consequence of Agility and Flexibility

The fact that ServiceNow can develop quickly through low-code/no-code tools and offers many options to configure and change information and data makes its value proposition strong. Still, the ability to do whatever you want without conforming to architecture or written rules often leads to technical debt. Nechyporenko (2015) demonstrates that when platforms are open, people often create custom tables and apply redundant business rules, which eventually results in the system's architecture being weakened.

Ernst et al. (2021) mention that unknown factors and pressure often cause TD, and not necessarily because someone lacks knowledge or expertise. Because of business-driven pressure, these choices in the ecosystem may be to implement a workflow swiftly, respond to a client's demand, or address a workflow issue as fast as possible. Still, when many band-aid policies are introduced, they reduce the system's overall consistency and make it harder to adjust later.

The skills and expectations set by a leader are needed to keep the platform flexible and prevent long-term problems.

Organizational Factors: Governance, Culture, and Lifecycle Thinking

What deserves more attention than the technical aspects are the factors that created or increased technical debt within the company. Experts say that failing to have review boards, strict workflow management, enough quality testing, and control over citizen development are some main reasons behind runaway debt (Hintsch et al., 2021; Blum & Blum, 2020). Because anyone on a team in ServiceNow can create new applications without IT's review, rules can be applied differently across the organization.

Similar to what Kario (2018) thought, Software Engineering in ServiceNow may also become slower and more difficult to maintain after a post-customization support period. Brown et al. (2010) and Zazworka et al. (2013) describe how not including lifecycle principles in design choices might lead to more expenses in the future.

To deal with this, the organization should ensure that people from each area officially care about and monitor the platform. Old modules have to be replaced opportunistically and technical debt should be regularly evaluated and cleared.

CMDB: The Hidden Debt Multiplier

Although ServiceNow relies heavily on the Configuration Management Database (CMDB), this part is also sensitive and can weaken the platform if not looked after well. Patel et al. discovered (in 2019) that it's not only automation that suffers when CI records are incorrect, certain, or duplicated, but it also brings about failures across incidents, change, and asset management.

Specifically, CMDB debt makes everything else worse by enhancing problems associated with automation rules and service catalogues. Since it occupies a vital position in the platform's structure, slight errors may travel to different parts of the workflow.

As CMDB is important, its management should be treated like source code and validated by schemas, automated sync, audit logs, and tracking its life. They note that detecting and monitoring hidden risks requires automated tools that come into action only at the time of crisis (Ernst et al., 2021).

Toward a Platform-Centric View of Technical Debt

In the past, the main techniques for managing technical debt were changing code and conducting tests. Yet, for platforms such as ServiceNow, debt management ought to be driven by taking configurations, flow, permissions, and data as primary pillars of the software creation process.

According to Schaffer et al. (2021), ServiceNow moved from just a back-office system to an important part of digital transformation in organizations. Achieving this effect calls for examining and changing the process of customization. Now, changes should focus on all parts of the platform, the ability of tools to interact, their openness to upgrades, and the alignment of those who use them.

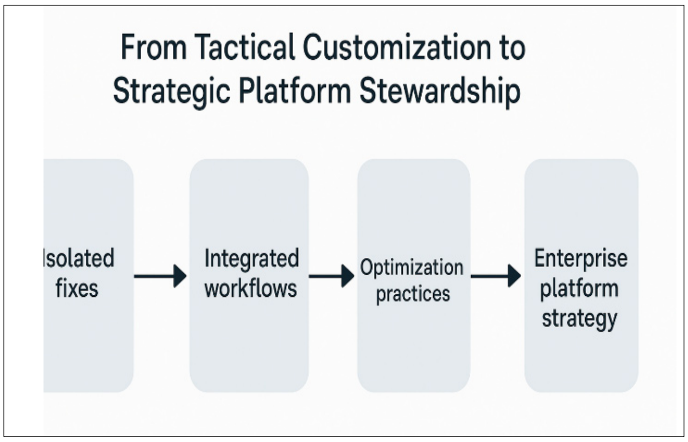
From Tactical Customization to Strategic Platform Stewardship
Isolated fixes
Integrated workflows
Optimization practices
Enterprise platform strategy

Figure 5. From Tactical Customization to Strategic Platform Stewardship

The chart depicts the change from working in seclusion to considering a broad approach as a platform for the whole company.

Implications for Practice and Future Research

The findings are very important for those who work in the field.

Architects have to set up initial design regulations and enforce them in every development.

Documentation should be prioritized by developers and people who create salesforce applications.

Board members and stakeholders should focus on checking both feature deliveries, as well as how much technical debt there is, based on reuse statistics, inactive scripts, and the condition of the CI.

- It is important for platforms to use automated methods to spot hidden debts and suggest how to fix them.

To investigate this further, reviewing real ServiceNow cases may reveal the growth and cost of TD debt, expenses needed to address issues, and development in following good TD governance practices. Total dependence on a government could be monitored over the years to see the changes it makes to TD.

CONCLUSION

It is becoming more difficult to manage technical debt in custom ServiceNow solutions, as the platform helps organizations deliver important services and automates tasks. Based on an analysis of available literature, this research points out that in ServiceNow, TD is more than bad code; it also affects various areas of the platform's architecture, configuration, how rules are followed, and the workplace culture.

A number of important trends can be seen in the literature under consideration. In the beginning, ServiceNow's flexibility for fast delivery and alignment with business can result in harsh customization, strictly set logic, and division of the system, mainly because standards are rarely put in place (Nechyporenko, 2015; Ernst et al., 2021). Also, a poorly managed CMDB often fills up with too much information and configuration data (Patel, Cook, Schabell, & Thomas, 2019). In addition, making technology simple for users leads to more chances for critical issues, especially since citizens can develop solutions that might damage the consistency of the platform (Hintsch et al., 2021; Blum & Blum, 2020).

It is highlighted in this article that the successful management of TD in ServiceNow relies on an approach that joins enterprise architecture principles, managing the platforms throughout the lifespan, and paying close attention to the stakeholders. Automated testing, audits, and well-structured scripts should go along with policies that support refactoring, carrying out debt reviews, and clear rules on applying custom changes (Brown et al., 2010; Kruchten et al., 2012).

In the end, ServiceNow's technical debt can be a challenge or a chance for further improvement. If technology is managed carefully, it helps design decisions, increases how reliable the platform is, and keeps development related to the company's long-term goals. Since the move from specialization to platformization (Schaffer et al., 2021), companies have to learn that managing technical debt is no longer optional and needs to be treated as a significant function of platform management.

REFERENCES

1. Blum, D., & Blum, D. (2020). Simplify and rationalize IT and security. In Rational cybersecurity for business: The security leaders' guide to business alignment (pp. 199–225). Apress. https://doi.org/10.1007/978-1-4842-5952-8_7
2. Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., ... &Zazworka, N. (2010, November). Managing technical debt in software-reliant systems. In Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research (pp. 47–52). https://doi.org/10.1145/1882362.1882373
3. Ernst, N., Kazman, R., & Delange, J. (2021). Technical debt in practice: How to find it and fix it. MIT Press. https://doi.org/10.7551/mitpress/12440.001.0001
4. Hintsch, J., Staegemann, D., Volk, M., & Turowski, K. (2021). Low-code development platform usage: Towards bringing citizen development and enterprise IT into harmony. Australasian Conference on Information Systems (ACIS). https://aisel.aisnet.org/acis2021/11/
5. Kario, P. (2018). Service design approach to understand the user experiences on the ServiceNow cloud solution [Bachelor's thesis, Laurea University of Applied Sciences]. Theseus. https://urn.fi/URN:NBN:fi:amk-2018113019182
6. Kotha, V. (2017). Customer-centric service management using ServiceNow [Master'sthesis, St. Cloud State University]. The Repository at St. Cloud State. https://repository.stcloudstate.edu/msia_etds/35/
7. Kruchten, P., Nord, R. L., &Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. IEEE Software, 29(6), 18–21. https://doi.org/10.1109/MS.2012.167
8. Kruchten, P., Nord, R. L., Ozkaya, I., &Falessi, D. (2013). Technical debt: Towards a crisper definition. Report on the 4th International Workshop on Managing Technical Debt. ACM SIGSOFT Software Engineering Notes, 38(5), 51–54. https://doi.org/10.1145/2507288.2507326
9. Nechyporenko, T. (2015). ServiceNow as a platform – Practical research [Bachelor's thesis, Haaga-Helia University of Applied Sciences]. Theseus. https://www.theseus.fi/bitstream/handle/10024/102830/Thesis_Tamara_Nechyporenko.pdf?sequence=1
10. Patel, M., Patil, S., Tzanavara, K., Chauhan, M., Wang, Y., Xie, H., & Shi, L. (2019). ServiceNow: CMDB research. Clark University. https://commons.clarku.edu/sps_masters_papers/50/

11. 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. https://doi.org/10.1287/mnsc.2015.2196

12. Schaffer, N., Ritzenhoff, M., Engert, M., & Krcmar, H. (2021). From specialization to platformization: Business model evolution in the case of ServiceNow. In Proceedings of the European Conference on Information Systems (ECIS). https://www.fortiss.org/fileadmin/user_upload/06_Ergebnisse/Publikationen/FROM_SPECIALIZATION_TO_PLATFORMIZATION___BUSINESS_MODEL_EVOLUTION.pdf

13. Zazworka, N., Spínola, R. O., Vetro', A., Shull, F., & Seaman, C. (2013, April). 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).https://dl.acm.org/doi/abs/10.1145/2460999.2461005

Citation: Venkata Surendra Reddy Narapareddy, "Managing Technical Debt In Custom Servicenow Solutions", Universal Library of Innovative Research and Studies, 2022; 01-08. DOI: https://doi.org/10.70315/uloap.ulirs.2022.001.