Technical Debt in Microservices: A Mixed-Method Case Study

Roberto Verdecchia*[0000−0001−9206−6637], Kevin Maggi*, Leonardo Scommegna[0000−0002−7293−0210], and Enrico Vicario[0000−0002−4983−4386]

Department of Information Engineering, University of Florence, Italy
`roberto.verdecchia@unifi.it`,
`kevin.maggi@edu.unifi.it`, `leonardo.scommegna@unifi.it`, `enrico.vicario@unifi.it`

Abstract. *Background:* Despite the rising interest of both academia and industry in microservice-based architectures and technical debt, the landscape remains uncharted when it comes to exploring the technical debt evolution in software systems built on this architecture. *Aims:* This study aims to unravel how technical debt evolves in software-intensive systems that utilize microservice architecture, focusing on (i) the patterns of its evolution, and (ii) the correlation between technical debt and the number of microservices. *Method:* We employ a mixed-method case study on an application with 13 microservices, 977 commits, and 38k lines of code. Our approach combines repository mining, automated code analysis, and manual inspection. The findings are discussed with the lead developer in a semi-structured interview, followed by a reflexive thematic analysis. *Results:* Despite periods of no TD growth, TD generally increases over time. TD variations can occur irrespective of microservice count or commit activity. TD and microservice numbers are often correlated. Adding or removing a microservice impacts TD similarly, regardless of existing microservice count. *Conclusions:* Developers must be cautious about the potential technical debt they might introduce, irrespective of the development activity conducted or the number of microservices involved. Maintaining steady technical debt during prolonged periods of time is possible, but growth, particularly during innovative phases, may be unavoidable. While monitoring technical debt is the key to start managing it, technical debt code analysis tools must be used wisely, as their output always necessitates also a qualitative system understanding to gain the complete picture.

Keywords: Technical Debt · Microservices · Software Evolution

1 Introduction

As companies seek to take advantage of their many benefits, microservice-based architectures are becoming more and more adopted. As often referenced, the microservice architecture style offers several advantages, such as scalability, flexibility, and the ability to develop and deploy individual components independently [11]. Albeit the many benefits microservice-based systems offer, the architectural style also comes with its own set of challenges, including increased complexity, the need for effective management of eventual consistency, and additional effort required for integration and system testing.

* The first two authors contributed equally to this work.

In this context, we can intuitively conjecture that, in order to cope with the increased complexity and potential loss of the bigger architectural picture, developers may tend to adopt suboptimal implementation expedients. While providing temporary benefits, such expedients may tend to make future development harder or even impossible. This concept of software quality issues related to temporary expedients is commonly referred to as technical debt (TD) [5].

TD is one of the paramount factors in software development practice. If left unmanaged, TD can lead, among other consequences, to lower development speed, raise of a high number of bugs, or even completely crystallized architectures [39]. TD has been widely covered in academic literature [3, 27, 41] and is increasing in research popularity. Similarly, albeit the adoption of microservice architectural style could be considered as a relatively new phenomenon, its widespread adoption recently drew a considerable academic interest [12, 17, 34].

Surprisingly, while both TD and microservices could be considered as popular topics in current academic literature, there has been relatively little focus on the relationship between TD and microservice architectures. To date, few studies have considered how TD evolves in microservice systems and, to the best to our knowledge, none have quantitatively studied in depth the characteristics of such evolution.

With this research, through a mixed-method case study on an open-source project comprising 12 microservices (see Section 3.4), we make a first step towards understanding how TD changes as microservice-based systems evolve. Our goal is to pave the way for future empirical studies that investigate the evolution of this relationship. By understanding how TD evolves in microservice architectures, and gaining insights into the characteristics of such evolution, we might be able to shed light on how TD can be effectively managed in microservice architectures, with the end goal of better supporting the long-term success of software-intensive systems based on such architectural style.

The main contributions of this research are (i) a quantitative case study reporting TD measurements through the evolution of a microservice-based software system, (ii) a thorough statistical analysis complemented by a manual code inspection and discussion of the results, (iii) a qualitative assessment of the gathered results *via* an interview with the leading developer and subsequent reflexive thematic analysis, and (iv) a replication package containing the entirety of the raw, intermediate, and final data, analysis traces, and code used for this study.

This work extends the preliminary case study presented at the first International Workshop on Quality in Software Architecture (QUALIFIER) [40] by complementing the investigation with an interview with the leading developer of the case study, analyzed *via* reflexive thematic analysis, which is used to gain further insights into the results and assess our conjectures on their interpretation.

2 Related Work

By considering the academic literature that focuses on TD in microservice-based systems we can identify, to the best of our knowledge, only a handful of studies.

The research of Lenarduzzi *et al.* [25], where the effects of migrating from a monolithic to a microservice architecture can have on TD are investigated, might potentially be the most similar to this work. As our study, the research presents a case study based on repository mining and static code analysis. In contrast to such study however, we (i) do not focus on the effects of migrating from monolithic to microservice architecture, (ii) aim to study various characteristics affecting TD (*e.g.,* number of microservices), and (iii) consider as case study a software-intensive system which comprises 13 microservices instead of the 5 studied by Lenarduzzi *et al.* [25].

By inspecting the other related literature, TD in microservices appears to be investigated primarily from a qualitative point of view. In a study by Toledo *et al.* [35], a multiple case study based on 25 interviews investigating architectural TD (ATD) in microservices is reported. The results of the investigation identified ATD issues, their negative impact, and the common solutions used to repay each debt type. Differently to such study, we focus on code TD [19, 27], utilize a quantitative rather than qualitative research method, and focus on a case study. In a similar work by Toledo *et al.* [16], through a qualitative analysis of documents and interviews, ATD in the communication layer of microservice-based architecture is investigated. The main contribution of the paper is a list of debt types specific to the communication layer of a microservice-based architecture, as well as their associated negative impact, and solutions to repay the debt types. Regarding the differences w.r.t. our work, the same considerations previously elicited for the other study of Toledo *et al.* [35] apply.

Bogner *et al.* [10] adopted 17 semi-structured interviews to study how the sustainable evolution of 14 microservice-based systems was ensured. Albeit from the results ATD emerged as a relevant issue, differently from our study, the work of Bogner *et al.* does not explicitly focus on TD. As additional difference w.r.t. our work, while tool-based DevOps processes were often mentioned as a mean to assure evolvability, the study is based on a qualitative rather than quantitative empirical research method. Bogner *et al.*, in a different study [9], surveyed 60 software professionals *via* an online questionnaire to learn how technical debt can be limited through maintainability assurance. Results indicated that using explicit and systematic techniques benefits software maintainability. As for the previous studies, also this work by Bogner *et al.* [9] adopts a qualitative rather than quantitative research approach. In addition, albeit both the study of Bogner *et al.* [9] and this work consider TD in microservices, the primary focus of Bogner *et al.* is on studying maintainability assurance techniques, while the one of this work is on TD evolution in microservice-based software-intensive systems.

Related to the concept of TD in microservice-based systems, Pagazzini *et al.* [31] present the extension of the tool Arcan [20] to detect microservice smells. As main difference with this work, the Arcan extension focuses on architectural smells rather than focusing explicitly on TD, and does not carry out a case study on TD evolution.

A more systematic literature review on TD in microservices w.r.t. this related work section is conducted by Villa *et al.* [43]. Based on the analysis of 12 primary studies, Villa *et al.* corroborate the intuition grounding this study, namely the absence of qualitative studies focusing on the evolution of TD in microservice-based systems. From the results of Villa *et al.*, ATD and code debt result to be the most frequently reported debt types for microservices. Such finding, which reflects the general trend observed for TD in developer discussions [2, 24], provides further support to the focus of this work on the evolution of code TD in microservice-based systems.

3 Study Design and Execution

In this section, we document the research design and execution of the study, in terms of research goal (Section 3.1), research questions (Section 3.2), and research process (Section 3.3).

3.1 Research Goal

The goal of this research is to conduct a preliminary investigation into the evolution of TD in software-intensive systems utilizing a microservice architecture. By using the Goal-Question-Metric framework of Basili *et al.* [8], our goal is:

Analyze *software evolution*
For the purpose of *studying trends and characteristics*
With respect to *technical debt*
From the viewpoint of *software engineering researchers*
In the context of *microservice-based software-intensive systems.*

In this study, we opt to focus on code TD [27], rather than other TD types (*e.g.,* ATD) guided to multiple factors, namely (i) code TD is one of the most frequent TD types appearing in microservice-based systems [43], (ii) in contrast to the other TD types, code TD is supported by a vast range of consolidated off the shelf tools, which are vastly used both in academic research and industrial practice [6], (iii) the focus on code TD allows for the natural extension of this preliminary case study to future heterogeneous case studies.

3.2 Research Questions

Based on the goal of our study, we can derive the main research question (RQ) and sub-research questions which guide our research.

The main RQ on which this study is based can be formulated as follows:

RQ: *How does code technical debt evolve in a microservice-based software-intensive system?*

With this main RQ, which encompasses the overall goal of the study, we broadly express our intent to study the evolution of code TD in microservice-based systems. To be more systematic, we decompose our main RQ into two sub-RQs, each one considering a different facet of TD evolution in microservice-based software intensive systems.

$\mathbf{RQ_1}$: *What is the evolution trend of TD in a microservice-based software-intensive system?*

With RQ_1, we aim to understand the overall evolution trend of TD in microservice-based systems, *e.g.,* if TD is constant through the evolution of a microservice-based software-centric system, if TD showcases a growing trend in time, or if the system is characterized by a seasonal TD trend (e.g., if developers are more prone to incur in TD before/after seasonal holidays).

$\mathbf{RQ_2}$: *Is there a relation between TD evolution and number of microservices?*

With RQ_2, we aim to understand if a relation exists between the evolution of TD and the number of microservices composing a software-intensive system. As example, we could conjecture that, due to suboptimal practices, as the number of microservices grows, TD grows at a higher rate (*e.g.,* TD is in superlinear or even exponential relation with the number of microservices).

3.3 Research Process

An overview of the process followed in this study is depicted in Figure 1, and is further documented below.

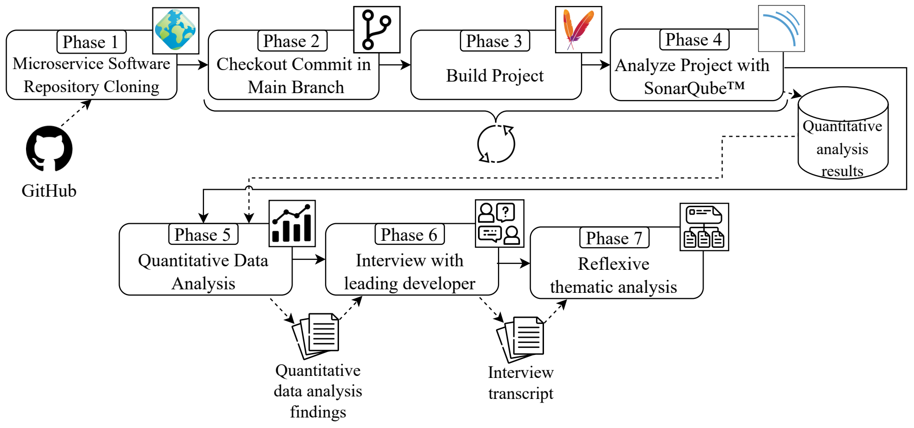
Phase 1
Microservice Software Repository Cloning
Phase 2
Checkout Commit in Main Branch
Phase 3
Build Project
Phase 4
Analyze Project with SonarQube™
GitHub
Quantitative analysis results
Phase 5
Quantitative Data Analysis
Phase 6
Interview with leading developer
Phase 7
Reflexive thematic analysis
Quantitative data analysis findings
Interview transcript

Fig. 1. Research process overview.

The research process consists of five phases, from cloning the case study software repository containing a microservice-based software project, to the static analysis of its source code, and the concluding statistical and manual analysis of the collected data. Each research phase is described in detail in the following.

3.4 Phase 1: Microservice Software Repository Cloning

The first step of our research process consists in cloning a repository containing a microservice-based software project. For this preliminary case study, we select the software repository `Cloud Native GeoServer`.[1] The project is a microservice implementation of GeoServer, an open source server for sharing geospatial data. `Cloud Native Geoserver` splits the original GeoServer geospatial services and API offerings into individually deployable components of a microservices based architecture.

The case study is selected starting from the manually validated list of microservice-based open-source projects hosted on GitHub elicited by Baresi *et al.* [7]. The project is selected from the list by considering as selection criteria (SC_1) the real use of the application, (SC_2) the number of times the repository is forked and starred, (SC_3) the number of repository commits, and (SC_4) the number of microservices the project comprises. We use SC_1 to exclude as potential case study a toy project and demo. SC_2 instead provides us assurance on the quality and popularity of the repository. Finally, SC_3 and SC_4 guarantee us that the project is representative of a long-lived, complex, software application based on a microservie-based architecture. `Cloud Native GeoServer` to date is forked a total of 52 times and starred 176 times. The repository currently counts a total of 985 commits, comprises 13 distinct microservices, and is composed of 38k lines of code (NCLOC), and is contributed to by 10 developers.[2]

[1] `http://geoserver.org/geoserver-cloud`. Accessed 4th July 2023.
[2] `https://github.com/geoserver/geoserver-cloud`. Accessed 4th July 2023.

3.5 Phase 2: Checkout Commit in Main Branch

The second phase of our research consists in checking out in temporal order the commits of the selected repository. For this process, we opt to consider the commits present in the main branch of the selected repository. While we are aware of the potential pitfalls implied by considering exclusively the main branch during repository mining processes [23], we deem analyzing also other branches as out of scope for this preliminary investigation. Related threats to validity are discussed in Section 5.

3.6 Phase 3: Build Project

After checking out a commit, the project is built by using the build automation tool used by the case study software-intensive system, namely Maven[3]. This step is a prerequisite for the analysis of the project *via* SonarQube (Phase 4, see Section 3.7), as the tool requires the compiled code of the software project in order to analyze it.

During this step, 7 out of 985 builds result to fail (0.7% of all builds). Upon inspection, we identify the failure to be caused by issues related to the Project Object Model (POM) of the Maven build. Rather than using a subjective heuristic to fix the issue, we opt to discard the commits associated to these failing builds. A single commit results instead to be characterized by an erroneous date in the versioning system. In order to avoid to independently estimate the correct commit date *via* some *ad hoc* heuristic, we opt to discard such commit from our analysis. Given the relatively low number of commits skipped due to build failures or dating issues (8/985 in total), we do not deem this factor to noticeably influence our results. Further considerations are reported in the threats to validity section (Section 5). For scrutiny and traceability purposes, the metadata of the 8 commits omitted from our analysis are documented in the replication package of this study.

To iterate and avoid possible confusion, due to the failing builds and an ill-dated commit, 977 out of the 985 total commits are considered for analysis of this study.

3.7 Phase 4: Analyze Project with SonarQube

After obtaining the compiled code of the project, the code is analyzed by utilizing the SonarQube tool. All commits are analyzed by using SonarQube version 9.9 LTS with SonarScanner for Maven version 3.9.1. During this process, in addition to the SQUALE metric measuring TD [26], other metrics and metadata of the project is collected, *e.g.,* the project size in terms of NCLOC, number of files, cognitive complexity, and committer name. To measure TD, the standard out of the box SonarQube rules configuration is used, in order to avoid subjective tempering of the tool settings.

The number of microservices appearing in each commit version is instead collected by following the method first introduced by Baresi *et al.* [7]. Such method relies on the analysis of Docker Compose files, in order to identify *via* parsing the microservices composing a software-intensive system.

Phases 2-4 are repeated for each of the 977 commits of the software project considered for this case study.

[3] `https://maven.apache.org`. Accessed 4th July 2023

3.8 Phase 5: Data Analysis

As last quantitative step of the research process, the data collected through Steps 1-4 is analyzed to answer our RQs.

To answer RQ_1, we decompose the TD evolution trend into its seasonal, trend, and irregular components [18] by utilizing on the STL algorithm [15]. We adopt the STL algorithm as it does not assume a time series distribution, it was successfully used in previous software engineering studies [4, 28], and an open-source implementation is available as an R library.[4] The resulting trend is then inspected qualitatively by graphical means. To gain further insights into the "TD hotspots", *i.e.,* commits showcasing the most outlier values in TD measurements, the content of the outlier commits are manually scrutinized. To identify outlier values, we leverage the STL decomposition, by first removing any seasonality and trend in the TD time series, and subsequently selecting the 10 most anomalous outliers identified in the STL irregular component series for manual scrutiny.

To answer RQ_2, we first study the potential correlation between the number of microservices and TD time series. Afterwards, we analyze the potential correlation between the derivatives of such series, to understand the relation between the growth speed of TD w.r.t. microservice number. For both cases, we test the correlation by using the Multivariate Granger causality analysis [21]. To calculate the optimal lag order for the Granger analysis, we adopt the Akaike Information Criterion [1]. As the Granger test assumes the time series to be stationary, we test such assumption *via* the Augmented Dickey-Fuller test [14]. In case the time series result to be non-stationary, we make them stationary by differencing the data, *i.e.,* by subtracting the value of each observation from the value of the previous observation in the time series.

3.9 Phase 6: Interview with leading developer

As closing phase of our investigation, we complement the quantitative research results collected through Phases 1 to 5 *via* a qualitative research process, namely an interview with the leading developer of `Cloud Native Geoserver`. This final research phase is used to validate the quantitative results collected for RQ_1 and RQ_2, gain further insights into the results, and assess our conjectures on their interpretation.

To identify the leading developer of the software project, the commit authorship of the GitHub repository is analyzed, and the most recurrent committer is contacted for confirmation of being the leading developer. The interview is conducted in a semi-structured fashion [22], with the support of a slide deck which is used to guide the interview.[5] Two weeks prior the interview, the quantitative analysis findings are shared with the leading developer, to ensure the interviewee has sufficient background knowledge and time to prepare for the interview.

The interview is composed of four main portions, namely (i) introduction and background, (ii) questions on the TD evolution in `Cloud Native Geoserver`, (iii) questions on the relation between TD and microservices in `Cloud Native Geoserver`, and (iv) closing remarks. During the introductory interview portion, we verify the familiarity of the leading developer with the repository and the TD metaphor, summarize the

[4] `https://stat.ethz.ch/R-manual/R-devel/library/stats/html/stl.html`. Accessed 4th July 2023.

[5] For completeness, the slide deck used, including the structured questions asked during the interview, is made available in the replication package of this study (see Section 5.4).

research procedure, and outline the goals of the interview. In the second interview portion, we ask a set of questions designed to gain insights into the TD evolution trends identified in Phase 5 (see Section 3.8). Specifically, for each observed TD evolution trend, we inquire about the development activities conducted in that period, the awareness of the TD trend in that period, and the effects of the TD trend on future development activities. Additionally, we also present our preliminary conclusions on the trend nature, in order to corroborate or disprove our suppositions. In the third interview portion, which regards the relation between TD and microservices in `Cloud Native Geoserver`, we present our quantitative results on the relation and our conjectures on the interpretation of the findings. As for the previous phase, we ask a set of questions designed to gain more insights into the quantitative data collected, and verify if we reached the correct conclusions. Finally, in the closing interview portion, we give the interviewee the opportunity to provide any additional remark on the investigated topic that we might have not covered with our previous interview questions.

Due to geographical distance, the interview is conducted *via* a Google Meet video-call. To ease the interview process, webcams are utilized to observe non-verbal communication of the interviewee, *e.g.,* hand gestures, facial expressions, and posture, and provide silent feedback to the interviewee without interrupting them [13]. The interview lasts approximately 45 minutes.

3.10 Phase 7: Reflexive thematic analysis

The interview is audio-recorded and manually transcribed by utilizing the denaturalism approach, *i.e.,* grammar errors are rectified, disturbances in the interview such as stutters are eliminated, and nonstandard accents (those not belonging to the majority) are normalized while maintaining a comprehensive and accurate transcription [29]. The transcript is then analyzed *via* reflexive thematic analysis [36] based on an open and axial coding process [33]. The adopted qualitative analysis approach allows us to cluster the incidents provided by the interviewee into different themes, and subsequently map the themes to our RQs to report the interview results in an structured and systematic fashion. In addition, adopting a reflexive approach allows us to reflect and reinterpret our conjectures on our quantitative results, revising their potentially subjective interpretation, and gain a more concrete and sound understanding of the studied case.

In the following section, the quantitative results and their reinterpretation based on the qualitative insights offered by the leading developer are reported.

4 Results

In this section we report and discuss the data gathered to answer our RQs. Specifically, in the next section (Section 4.1) we consider the results of RQ_1, while in Section 4.2 we take into account the results of RQ_2.

4.1 Results RQ_1: Evolution of TD in a microservice-based software-intensive system

As described in Section 3.8, in order to study the TD evolution of our case study, namely the `Cloud Native GeoServer` application, we consider three different components, namely the TD evolution trend, seasonality, and irregularities. An overview of such decomposition is depicted in Figure 2.

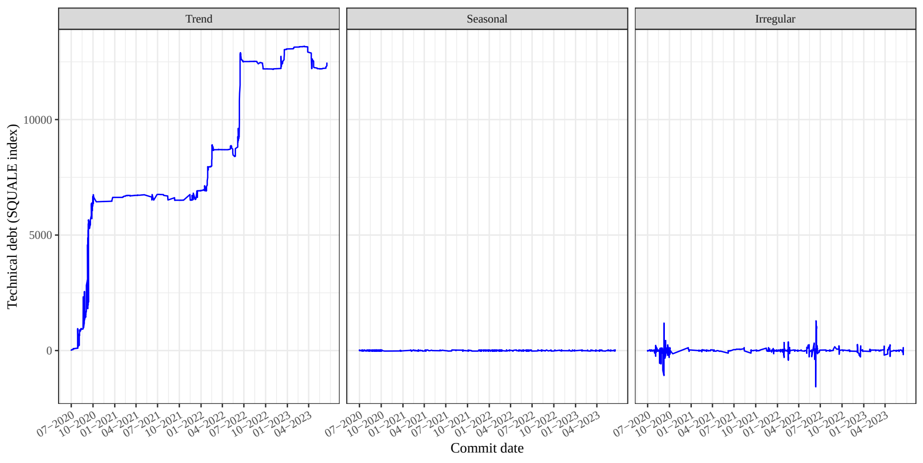

Fig. 2. Decomposition of the `Cloud Native GeoServer` application technical debt evolution *via* the STL algorithm.

As we can observe from the leftmost diagram of Figure 2, the TD evolution showcases an overall growing trend. Interestingly, two outstanding jumps, *i.e.,* sudden increases in TD values, can be be noticed in the plot. By comparing the trend figure with the one reporting the irregularities in TD evolution (rightmost diagram of Figure 2), we note that such outliers are captured by the STL algorithm decomposition. The commits corresponding to such jumps are further analyzed in the second data analysis carried out to answer RQ_1, namely the manual scrutiny of the "TD hotspots" (see Section 3.8). Overall, as could be expected, TD tends to naturally increase during time as the application becomes bigger and more complex. From the qualitative data collected we determine that, according to the leading developer, the main reason behind the overall increasing trend is due to the lack of a systematic TD monitoring process in the development pipeline. By directly quoting the leading developer:
"*The main and very actual reason for the increase of TD is the lack of monitoring it. We could have established a quality assurance policy from the beginning. For one reason or another it's something we always kept on postponing. I think that is the most relevant part.*"

The TD evolution trend also presents noticeable plateaus, *i.e.,* periods where the TD values remain approximately stable. The first plateau, starting from October 2020, lasts approximately one year and three months of development. By inspecting the commit dates, we note that none of the plateaus reported in the trend of Figure 2 is due to periods of development inactivity. Therefore, we conjecture that the development periods associated to the plateaus correspond to development periods where deliberate efforts might be made to prevent a TD increase. As further discussed in the following paragraphs, spikes in TD correspond to periods of innovation and intense activity. Contrary to initial conjectures, insights from the interview reveal that the plateaus are not a result of intentional

efforts to maintain TD constant. As articulated by the lead developer, these plateaus actually represent hard-earned periods of tranquil development, which followed the more tumultuous phases marked by significant jumps in TD. By quoting the leading developer:
"Seeing that almost flat line for that long period makes me happy. It means that, after all that crazyness, our development choices were not that bad."

By considering the seasonality of the time series (center diagram of Figure 2), we can intuitively observe that no seasonal behavior is present in the TD evolution of the `Cloud Native GeoServer` application. This implies that TD is not more likely to be introduced during a certain period of the year. The leading developer confirms this conjecture by describing a general lack of seasonality in the development activities:
"We don't have an established development roadmap but we do have contracts with costumers. So it's [the release time] more based on a as needed basis and is more agile than having a six month release timeline. We actually release very often. From one week to another we can have a new release because of patching or adding new features."

As second data analysis process carried out to answer RQ_1, we manually inspect the potential "TD hotspots" (see Section 3.8). The most noticeable "TD hotspot" corresponds to a sudden increase in TD values recorded on June 2022 (see Figure 2, rightmost plot). From manual scrutiny, this sudden TD variation results to be due to the upgrade of the JUnit testing framework[6]. The commit also includes the cross-microservice refactoring of test files according to the upgrade.

Other seven "hotspots", which are not graphically appreciable from the TD evolution irregularities depicted in the rightmost plot of Figure 2, happen on the same day as the JUnit upgrade commit. Upon manual inspection, we note that the commits corresponding to these additional seven hotspots involve many microservices of the `Cloud Native GeoServer` application. The commits result to either focus on (i) further refactoring of testing artifacts, (ii) bug fixing, (iii) implementing logging mechanisms, and (iv) introducing automation processes. The leading developer confirms JUnit as being the root cause of the identified "TD hotspots", recalling the high development effort required by the upgrade:
"I did noticed as well as you did that the new JUnit 5 version provoked a jump on TD, because there are a lot of tests and we need to make all test methods and classes package private, which led to a quick [TD] spike."

Via unstructured follow up question on the TD associated to this development period, we learn that the deliberate TD taken on by upgrading JUnit paid off, by considerably easing future development activities. As described by the leading developer:
"It was refreshing. There are [in JUnit 5] new constructs... new ways to test that don't require to launch the whole application. That was so time consuming. There is always a bit of a learning curve, but I would not go back to JUnit 4."

Regarding the sudden increase of TD values in October 2020 results instead, from manual inspection of the quantitative data results to be caused by the addition of 33 new files in a microservice. The commit involves the extension of the `Cloud Native GeoServer` features *via* the binding to a new JSON parser.

The last of the 10 "TD hospots" considered for manual analysis instead corresponds to a sudden increase of TD values happening in September 2020. In this case, the TD increase results to be due to a refactoring activity carried out across 70 files, which involved a considerable number of NCLOC (1.5k NCLOC additions, and 589 NCLOC deletions).

[6] `https://junit.org/junit5/`. Accessed 5th July 2023.

By asking about the "TD hotspost" of September and October 2020 to the leading developer, we understand that both months of noticeable TD increase are due to the early stage of the software project. During this phase, the developers get accustomed to developing the new application, learning as they go on how the project should be shaped. Taking on TD appears to be a natural consequence of this early stage development period. The lead developer recalls on this episode as follows:
"The early stages of the project were subject to a lot of activity. Figuring out mainly how to split up all GeoServer fuctionality into microservices, Spring Boot modules, and configurations. There was a lot of activity, and it was an intense learning period. It makes sense that until we reached some stability... we are talking about the first 3 months... it was crazy having to figure all that out."

As conclusion to RQ_1, we conclude that both working on a single microservice, or multiple ones at the same time, can drastically influence TD. Considerable TD variations can happen independently of the developer activity conducted, *e.g.,* a framework upgrade can have an unforeseen cascading impact on TD, or a refactoring activity could lead to a considerable TD increase. As could be intuitively expected, early stages of development are prone to introduce TD. In addition, taking on TD in a certain period can lead to following periods of TD steadiness, till a new cycle of development innovation is needed. As a word of caution, from the considered case study we learn that TD tool measures can be mischievous. More specifically, a steep code TD increase could also be concurrently associated with a TD decrease that is not measured by the utilized tool, as highlighted by the upgrade of JUnit in the considered case study.

$\mathbf{RQ_1}$ answer (*TD Evolution in a Microservice-based Architecture*)
TD displays an overall increasing trend in time, albeit long periods of continued development without TD increase are noticeable. Considerable TD variations can happen by working on one or multiple microservices, and may occur regardless of the development activity conducted. Taking on in a given development phase can result in subsequent periods of TD stability, until a new wave of development innovation is needed. TD metrics can be deceptive, as a sharp increase in code TD might coincide with an unmeasured decrease in another TD dimension.

4.2 Results RQ_2: Relation between TD evolution and number of microservices

In order to study the potential correlation between TD and number of microservices, we start by graphically inspecting the time series of the two metrics. As can be seen in Figure 3, both TD and microservices seem to display an overall similar growth trend. However, a correlation does not appear always to be present in all commits. As example, by considering Figure 3, we can observe that the removal of one microservice in April 2021 did not correspond to any noticeable TD decrease. In some cases, *e.g.,* April 2023, the addition of a microservice is even associated with a decrease in TD (*i.e.,* microservice number and TD are inversely proportional). This would imply that, while number of microservices could display a strong correlation of directly proportional nature, this might not always be the case.

In order to gain statistical insight into the relation, we follow the analysis process presented in Section 3.8. From the results of the Granger causality test we confidently

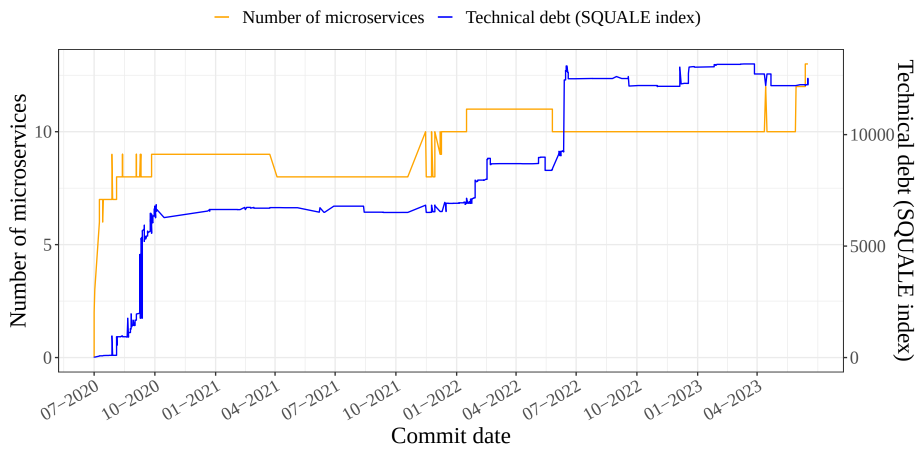

Fig. 3. Overview of the evolution of technical debt and number of microservices of the `Cloud Native GeoServer` application.

reject the null hypothesis and conclude that the evolution of number of microservices and TD are strongly correlated ($p\text{-}value < 4.593^{-6}$). This implies that, albeit seldom irregularities, a growth (or decrease) of microservice number corresponds to a similar change in TD. As additional remark, this indicates that the number of microservices could be used to predict TD values.

As further analysis conducted to answer RQ_2, we study the derivatives of the two time series. This allows us to understand if the two series grow at similar rates, or if a growth in the microservice series correspond to the growth at a higher rate in the TD series. Intuitively, we could expect that, as the number of microservices increases, the software-intensive system becomes more complex, and hence TD grows at a higher rate as the system becomes bigger. From the Granger test results however, we understand that this is not the case. In fact, we observe that also the derivatives of the time series are strongly correlated ($p\text{-}value < 7.896^{-6}$), *i.e.,* we can discard with statistical confidence the null hypothesis. This implies that the impact of adding (or removing) a microservice on TD is similar regardless of the number of microservices already present in a software-intensive system.

As subjective interpretation of this latter result, we can conjecture that this pattern indicates an appropriate adherence to the microservice architectural principles, through which microservices are developed independently by following a loosely coupled and highly cohesive architecture. From the interview with the leading developer however, we learn that this conjecture is partially inaccurate. While in fact the impact on TD of including or excluding microservice is independent of the microservices already present, the motivation behind this phenomenon is not primarily due to the microservice independence. On the contrary, as described by the leading developer, all microservices share a common architectural foundation, and each functionality implemented in the microservice heavily relies on dependencies loaded *ad hoc* from the common architectural layer all microservices share. Therefore, the absent impact of microservice number on TD trends is due to the lean nature of the microservices, which enforce loose coupling by loading the functionalities from a common architectural layer. As described by the leading developer: *"It does not matter how many microservices there are in the ecosystem because architecturally there is a cross-cutting layered design. The microservice functionality itself builds up from dependencies that are usually cross-cutting. There is this cherry-picking approach on what is loaded on each microservice, but all microservices have pretty much the same dependencies on the classpath."*

From the additional insights gained *via* the interview, we note the crucial importance of complementing quantitative source code analyses with qualitative aspects. While, during the qualitative assessment of the TD present in a software-intensive system we can conjecture on the deeper motivations behind TD values, it is only by presenting and discussing the analysis results with the developers of the system that we can get a more detailed and complete picture. Therefore, as a word of caution for software managers, product owners, and alike, it is paramount to always interpret the values provided by static analyzers with caution, and when possible qualitatively complementing the results, before taking decisions on future development activities.

RQ$_2$ answer (*Relation between TD and number of microservices*)
TD and number of microservices are strongly correlated, albeit in seldom cases such relation does not persist. The impact of adding (or removing) a microservice on TD is similar, regardless of the number of microservices already present in a software-intensive system. TD tools based on source code analysis can support TD management processes. However, their results should be complemented with qualitative knowledge before making decisions on future development activities.

5 Threats to Validity

The presented results have to be interpreted in light of potential threats to validity. By following the categorization of Runeson *et al.* [32], we consider four aspects. While documented towards the end of the study, in order to avoid a common pitfall of threats to validity in software engineering research [38], threats were considered from the early stages of this investigation, as further documented below.

5.1 Construct Validity

To answer our RQs, we measured code TD by adopting the SQUALE index, a metric widely used in the literature to study TD [6,25,42]. The number of microservices was measured by utilizing the heuristic first introduced by Baresi [7] (see also Section 3.7). The use of the heuristic might have marginally affected our results, as it relies on the analysis of the Docker Compose file. Therefore, a service could be identified at its insertion in the Docker Compose file, which does not necessarily imply the start of its actual development. At most, this threat could have introduced a lag in the TD timeseries w.r.t. the number of microservice one (corresponding to the time elapsed from the insertion of a microservice in the Docker Compose file, and the start of its actual development). As the potential effects of this marginal threat were not noticeable in our data analysis, we do not deem the threat considerably influenced our results. The threat would at most imply a stronger correlation between microservice number and TD than the one observed. Regarding the focus on the main development branch of `Cloud Native Geoserver` (see Section 3.5), we note that the application possesses other two branches, which are characterized by 2 and 48 commits ahead, and 256 and 28 commits behind the master branch respectively. Given the low number of commits in such branches w.r.t. the master branch (2/985 and 48/985), we do not deem that this research design choice could have drastically influenced our results. To mitigate potential threats to construct validity due to the design of the structured interview questions, we complemented the process with follow-up questions whenever necessary, *e.g.,* when more information was required to completely understand answers. Additionally, the interviewee was frequently asked if they wanted to include any additional information on the studied topic that was not directly covered by the questions posed.

5.2 Internal Validity

To avoid potential confounding factors, we (i) discarded all failing builds, and a commit associated with an incorrect date, (ii) manually scrutinized a set of commits presenting anomalous TD values, (iii) conducted a rigorous statistical analysis on the collected data. Regarding the qualitative analysis, to avoid subjective biases during the data collection and analysis, three researchers took part to the interview, and the relevant findings were jointly discussed before including them in the paper.

5.3 External validity

As any case study, we do not claim the complete generalizability of the obtained results. While comparable results might be observed in software-intensive systems of similar development context and characteristics as `Cloud Native GeoServer`, this could also not be the case. To partially mitigate potential threats to external validity, we selected the case study a software project developed in one of the most popular programming languages (Java), while also ensuring the representativeness of the software project *via* a set of selection criteria defined *a priori* (see also Section 3.4).

5.4 Reliability and Replication Package

If and to what extent the results of the study can be independently reproduced by other researchers. With exception of the manual scrutiny conducted to analyze the commits presenting the most anomalous TD values, the quantitative results are completely based on the execution of mining and data analysis scripts. We make all data, scripts, and settings available in a companion replication package[7]. Given that such results are of purely quantitative nature, we deem the reliability of such results as very high. To increase the reliability of the qualitative research process used (see Section 3.9), we make the guiding interview slide deck available for scrutiny in the replication package, and make extensive use of direct quotes in the manuscript to avoid subjective misinterpretation of the data collected through the interview.

[7] **Replication package of this study**: `https://github.com/STLab-UniFI/QUALIFIER-2023-TD-microservices-rep-pkg`. Accessed 6th July 2023.

6 Conclusion and Future Work

In this study, we present a preliminary case study investigating the evolution of technical debt in microservice architectures. The investigation utilizes as case study the application `Cloud Native GeoServer`, which comprises a total of 13 distinct microservices, 977 commits, and 38k NCLOC. The study is primarily based on repository mining and source code analysis. The results show that TD evolution displays a growing trend, mostly due to development innovation periods, followed by moments of TD stability, when no disruptive change is needed. TD variation are independent of the number of microservices and development activities considered in a commit. TD and number of microservices are correlated, and adding or removing a microservice has the same impact on TD regardless the number of microservices already present.

Adhering to microservice architecture principles might keep technical debt compartmentalized within microservices, and therefore make TD more manageable w.r.t. other types of architectures (*e.g.,* monolithic ones). It is crucial for developers to remain aware of the potential TD they may incur in, irrespective of the quantity of microservices they modify or the nature of the development activity they undertake. An intuitively trivial change, such as the upgrade of a testing framework, could have a massive cascading effect on the TD of a microservice-based software-intensive system. While it is feasible to maintain a consistent level of TD during the evolution of a microservice-based application, an increase in technical debt may be inevitable as the software-intensive system grows in size and complexity.

As a word of warning for both researchers and practitioners, through the presented case study, we observe how TD metrics can be deceptive, as they may not always provide a clear and complete picture of the TD present in a software-intensive system. While conducting TD source code analyses it is therefore paramount to consider that (i) a variation in code TD may co-occur with an opposite variation in another TD aspect, (ii) code analysis results should be always complemented by qualitative knowledge (e.g., interviews or focus groups).

The leading developer however also emphasizes the importance of including TD monitoring capabilities in the development pipeline to manage TD and avoid significant consequences, *e.g.,* flaky behaviors or crystallized architectures [39]. As articulated in the interview, the leading developer states:
"It [the quantitative analysis] was quite enlightening to me. I wanted to include a static code analysis for a long time. And maybe it would have never happen or it would have taken me way too long to address if I dind't have this feedback from you".
The leading developer further reports that they are waiting for the project steering committee to integrate the repository with SonarCloud, and is currently working lowering the identified TD. From a email exchange held after the interview, we learn that the the `Cloud Native GeoServer` repository underwent a considerable refactoring, leading the TD measured *via* SonarQube to be close to zero.

The presented case study has to be considered as a stepping stone for future research on TD in microservice-based systems. Several facets which could provide more information on the phenomenon are not considered in the study. As future work, we plan to (i) study the individual contribution of each microservice to the TD measured at system level, (ii) conduct a more in-depth analysis of "TD hotspots", (iii) utilize dedicated tools to measure other types of TD, *e.g.,* by combining different data sources [37] or focusing on ATD *via* the ATDx tool [30], and (iv) extend the research to a multiple-case study.

Acknowledgments

We would like to express our uttermost gratitude to Gabriel Roldan, leading developer of `Cloud Native Geoserver`, for his availability, insightful feedback, and genuine interest in the research project.

Roberto Verdecchia would like to thank Curzio Checcucci for his lighthearted yet insightful feedback on the data analysis process.

This work was partially supported by the European Union under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE0000001 - program "RESTART").

References

1. Akaike, H.: Information theory and an extension of the maximum likelihood principle. In: Selected papers of hirotugu akaike, pp. 199–213. Springer (1998)
2. Aldrich Edbert, J., Jannat Oishwee, S., Karmakar, S., Codabux, Z., Verdecchia, R.: Exploring technical debt in security questions on stack overflow. International Symposium on Empirical Software Engineering and Measurement (2023)
3. Alves, N.S., Mendes, T.S., De Mendonça, M.G., Spínola, R.O., Shull, F., Seaman, C.: Identification and management of technical debt: A systematic mapping study. Information and Software Technology **70**, 100–121 (2016)
4. Atchison, A., Berardi, C., Best, N., Stevens, E., Linstead, E.: A time series analysis of travistorrent builds: to everything there is a season. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories. pp. 463–466. IEEE (2017)
5. Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C.: Managing technical debt in software engineering (dagstuhl seminar 16162). Dagstuhl Reports **6** (01 2016)
6. Avgeriou, P.C., Taibi, D., Ampatzoglou, A., Fontana, F.A., Besker, T., Chatzigeorgiou, A., Lenarduzzi, V., Martini, A., Moschou, A., Pigazzini, I., et al.: An overview and comparison of technical debt measurement tools. Ieee software **38**(3), 61–71 (2020)
7. Baresi, L., Quattrocchi, G., Tamburri, D.A.: Microservice architecture practices and experience: a focused look on docker configuration files. arXiv preprint:2212.03107 (2022)
8. Basili, V.R., Caldiera, G., Rombach, D.: The Goal Question Metric Approach. In: Encyclopedia of Software Engineering, pp. 528–532. Wiley (1994)
9. Bogner, J., Fritzsch, J., Wagner, S., Zimmermann, A.: Limiting technical debt with maintainability assurance: An industry survey on used techniques and differences with service-and microservice-based systems. In: Proceedings of the 2018 International Conference on Technical Debt. pp. 125–133 (2018)
10. Bogner, J., Fritzsch, J., Wagner, S., Zimmermann, A.: Assuring the evolvability of microservices: insights into industry practices and challenges. In: 2019 IEEE International Conference on Software Maintenance and Evolution. pp. 546–556. IEEE (2019)
11. Bogner, J., Fritzsch, J., Wagner, S., Zimmermann, A.: Microservices in industry: insights into technologies, characteristics, and software quality. In: 2019 IEEE international conference on software architecture companion. pp. 187–195. IEEE (2019)
12. Bogner, J., Wagner, S., Zimmermann, A.: Automatically measuring the maintainability of service-and microservice-based systems: a literature review. In: Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement. pp. 107–115 (2017)
13. Brinkmann, S.: Qualitative interviewing. Understanding Qualitative Rese (2013)
14. Cheung, Y.W., Lai, K.S.: Lag order and critical values of the augmented dickey–fuller test. Journal of Business & Economic Statistics **13**(3), 277–280 (1995)

15. Cleveland, R.B., Cleveland, W.S., McRae, J.E., Terpenning, I.: Stl: A seasonal-trend decomposition. J. Off. Stat **6**(1), 3–73 (1990)
16. De Toledo, S.S., Martini, A., Przybyszewska, A., Sjøberg, D.I.: Architectural technical debt in microservices: a case study in a large company. In: 2019 IEEE/ACM International Conference on Technical Debt. pp. 78–87. IEEE (2019)
17. Di Francesco, P., Lago, P., Malavolta, I.: Architecting with microservices: A systematic mapping study. Journal of Systems and Software **150**, 77–97 (2019)
18. Dickey, D.A., Fuller, W.A.: Distribution of the estimators for autoregressive time series with a unit root. Journal of the American statistical association **74**(366a), 427–431 (1979)
19. d'Aragona, D.A., Pecorelli, F., Baldassarre, M.T., Taibi, D., Lenarduzzi, V.: Technical debt diffuseness in the apache ecosystem: A differentiated replication. In: 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). pp. 825–833. IEEE (2023)
20. Fontana, F.A., Pigazzini, I., Roveda, R., Tamburri, D., Zanoni, M., Di Nitto, E.: Arcan: A tool for architectural smells detection. In: 2017 IEEE International Conference on Software Architecture Workshops. pp. 282–285. IEEE (2017)
21. Granger, C.W.: Investigating causal relations by econometric models and cross-spectral methods. Econometrica: journal of the Econometric Society pp. 424–438 (1969)
22. Kallio, H., Pietilä, A.M., Johnson, M., Kangasniemi, M.: Systematic methodological review: developing a framework for a qualitative semi-structured interview guide. Journal of advanced nursing **72**(12), 2954–2965 (2016)
23. Kovalenko, V., Palomba, F., Bacchelli, A.: Mining file histories: Should we consider branches? In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering. pp. 202–213 (2018)
24. Kozanidis, N., Verdecchia, R., Guzmán, E.: Asking about technical debt: Characteristics and automatic identification of technical debt questions on stack overflow. In: Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. pp. 45–56 (2022)
25. Lenarduzzi, V., Lomio, F., Saarimäki, N., Taibi, D.: Does migrating a monolithic system to microservices decrease the technical debt? Journal of Systems and Software **169**, 110710 (2020)
26. Letouzey, J.L.: The sqale method for evaluating technical debt. In: 2012 Third International Workshop on Managing Technical Debt. pp. 31–36. IEEE (2012)
27. Li, Z., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. Journal of Systems and Software **101**, 193–220 (2015)
28. Malavolta, I., Verdecchia, R., Filipovic, B., Bruntink, M., Lago, P.: How maintainability issues of android apps evolve. In: 2018 IEEE international conference on software maintenance and evolution. pp. 334–344. IEEE (2018)
29. Oliver, D.G., Serovich, J.M., Mason, T.L.: Constraints and opportunities with interview transcription: Towards reflection in qualitative research. Social forces **84**(2), 1273–1289 (2005)
30. Ospina, S., Verdecchia, R., Malavolta, I., Lago, P.: ATDx: A tool for providing a data-driven overview of architectural technical debt in software-intensive systems. In: European Conference on Software Architecture (2021)
31. Pigazzini, I., Fontana, F.A., Lenarduzzi, V., Taibi, D.: Towards microservice smells detection. In: International Conference on Technical Debt. p. 6. ACM, Seoul, Republic of Korea (2020). https://doi.org/10.1145/3387906.3388625
32. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical software engineering **14**, 131–164 (2009)
33. Saldaña, J.: The coding manual for qualitative researchers. sage (2021)
34. Soldani, J., Tamburri, D.A., Van Den Heuvel, W.J.: The pains and gains of microservices: A systematic grey literature review. Journal of Systems and Software **146**, 215–232 (2018)

35. de Toledo, S.S., Martini, A., Sjøberg, D.I.: Identifying architectural technical debt, principal, and interest in microservices: A multiple-case study. Journal of Systems and Software **177**, 110968 (2021)
36. Vaismoradi, M., Turunen, H., Bondas, T.: Content analysis and thematic analysis: Implications for conducting a qualitative descriptive study. Nursing & health sciences **15**(3), 398–405 (2013)
37. Verdecchia, R.: Architectural technical debt identification: Moving forward. In: 2018 IEEE International Conference on Software Architecture Companion (ICSA-C). pp. 43–44 (2018). https://doi.org/10.1109/ICSA-C.2018.00018
38. Verdecchia, R., Engström, E., Lago, P., Runeson, P., Song, Q.: Threats to validity in software engineering research: A critical reflection. Information and Software Technology **164**, 107329 (2023)
39. Verdecchia, R., Kruchten, P., Lago, P., Malavolta, I.: Building and evaluating a theory of architectural technical debt in software-intensive systems. Journal of Systems and Software **176**, 110925 (2021)
40. Verdecchia, R., Maggi, K., Scommegna, L., Vicario, E.: Tracing the footsteps of technical debt in microservices: A preliminary case study. International Workshop on Quality in Software Architecture (2023)
41. Verdecchia, R., Malavolta, I., Lago, P.: Architectural technical debt identification: The research landscape. In: Proceedings of the 2018 International Conference on Technical Debt. pp. 11–20 (2018)
42. Verdecchia, R., Malavolta, I., Lago, P., Ozkaya, I.: Empirical evaluation of an architectural technical debt index in the context of the apache and onap ecosystems. PeerJ Computer Science **8**, e833 (2022)
43. Villa, A., Ocharan-Hernandez, J.O., Perez-Arriaga, J.C., Limon, X.: A systematic mapping study on technical debt in microservices. In: 2022 10th International Conference in Software Engineering Research and Innovation. IEEE (2022)