applied sciences

MDPI

Article

Technical Debt Prioritization in Telecommunication Applications: Why the Actual Refactoring Deviates from the Plan and How to Remediate It? Case Study in the COVID Era

Marek G. Stochel [1,2,*], Mariusz R. Wawrowski [1] and Piotr Chołda [2]

[1] Motorola Solutions, 30-392 Kraków, Poland
[2] AGH University of Science and Technology, Institute of Telecommunications, 30-059 Kraków, Poland
* Correspondence: marek.stochel@motorolasolutions.com

Abstract: This paper focuses on application of a technical debt prioritisation technique in telecommunication software managing a fleet of devices for a video surveillance system. Technical debt for this application was gathered, categorised and prioritised according to the Continuous Debt Valuation Approach (CoDVA), previously proposed by the authors. The following research question was posed: Is prioritising technical debt reduction based on CoDVA effective (i.e., executed as per plan, bringing tangible benefits)? The outbreak of COVID-19 pandemic caused unprecedented disturbance to the engineering organisations worldwide, therefore the technical debt identification phase had to be adapted to cope with a switch to forced working-from-home mode. This was achieved by applying the Wisdom of Crowds method, ensuring broad participation of engineers, and providing a fairly complete picture of the accrued technical debt. Nevertheless, the actual technical debt reduction activities did not follow exactly the expected guidelines. The three main causes of this phenomenon were discovered: continuous refactoring approach, sizing of technical debt items, and the broadened scope of refactoring activities. Therefore, as a result of this case study we propose to adopt a specific broadened definition of technical debt and follow a few rules for defining its scope and granularity.

Keywords: COVID experience software development; software engineering; software engineering and debt metaphors; software maintenance and evolution; technical debt; technical debt management; technical debt prioritization; Wisdom of Crowds

Citation: Stochel, M.G.; Wawrowski, M.R.; Chołda, P. Technical Debt Prioritization in Telecommunication Applications: Why the Actual Refactoring Deviates from the Plan and How to Remediate It? Case Study in the COVID Era. *Appl. Sci.* **2022**, *12*, 11347. https://doi.org/10.3390/app122211347

Academic Editor: Paolino Di Felice

Received: 30 September 2022
Accepted: 4 November 2022
Published: 8 November 2022

1. Introduction

The evolution of telecommunication software in use, in an ever-changing landscape of technologies, tools, and business models causes massive creation of technical debt [1]. However, its understanding evolves in time, and a widely used definition is as follows: "*In software-intensive systems, technical debt consists of design or implementation constructs that are expedient in the short term, but set up a technical context that can make a future change more costly or impossible. Technical debt is a contingent liability whose impact is limited to internal system qualities—primarily, but not only, maintainability and evolvability*" [2]. Additionally, technical debt embraces a set of actionable product technical debt items (TD Items, TDIs) indicating these immature artefacts and their deviation from the desired optimal state. In our previous paper, we claim that the technical debt definition should be expanded to embrace all software artefacts responsible for delivering a product to a customer [3], as the business perspective shifts towards a service model, rather than a one-time sale. Moreover, even with a one-time sale, the seller is responsible for providing agreed-upon service (bound by Service Level Agreement) for a predefined period of time. Hence, all software artefacts constituting a stable development environment, ensuring testability (automation), code analysis tools, continuous integration and delivery, etc., may be subject to technical debt dynamics. For this reason, they should be considered a part of the telecommunication software solution, which becomes actually a service offered to the customers. Moreover,

properly targeted improvements in such a broader context may improve an overall solution: its stability, testability, ability to deliver updates fast; additionally improving predictability of the development team [3].

The outbreak of the coronavirus pandemic (COVID-19) has caused an unprecedented worldwide disruption to personal lives, global economy, and operating procedures of organisations. The field of software engineering was not immune to it, and particularly the initial plan for the industrial case study discussed in this article had to be modified, taking into account forced working-from-home policy. This paper presents the initial results from an over one-year-long case study, and describes the application of the technical debt prioritisation technique to a product managing a fleet of connected devices for a video surveillance system. We decided to use the modified Continuous Debt Valuation Approach (CoDVA) technique for technical debt valuation and prioritisation, which we developed earlier [4]. Additionally, a prerequisite to successful application of CoDVA method was a robust technical debt identification step; in our case, it was executed according to the Wisdom of Crowds approach [5], which proved to be effective in other experiments [6].

Ultimately, we posed a research question: Is prioritising technical debt reduction based on CoDVA effective (i.e., executed as per plan, bringing tangible benefits)? In our case study, a few interesting observations were made. Namely, the implementation of technical debt reduction did not follow exactly the expected guidelines. We have discovered three major reasons for that: implementation of continuous refactoring approach, the size of desired technical debt reduction effort (planned refactorings), and additional required activities which were not originally considered a part of technical debt reduction. Now, after considering the results, we propose to adopt a specific definition of technical debt and follow a few rules for defining the scope of technical debt reduction, prioritising it, and executing refactoring efforts to improve overall software quality and maintainability. The initial results of this industrial case study proved that the quality of the product improved over time, as a downward trend of stabilisation issues was observed.

Summarising, this paper provides further details on conceptualisation of our original technical debt valuation and prioritisation (CoDVA) technique and its practical application. The observations made during the case study show how to mitigate potential threats related to the functional use of the concept. Additionally, it proves that the approach can be adjusted to be conducted in forced working-from-home mode imposed on the organisations during COVID-19 pandemic, and despite the challenges, the development team can address significant refactors (technical debt reduction) achieving tangible results.

Paper structure. The remainder of the paper covers the following topics: Section 2 discusses the background and related work. Section 3 presents methodology, including data collection and analysis processes. Discussion on the case study results is shown in Section 4, followed by our analysis of potential threats to validity of the study in Section 5. Finally, conclusions and future work are presented in Section 6.

2. Related Work

The technical debt term is broadly used; however, due to its metaphorical origin it gradually became ambiguous [7]. Therefore, the operational definition used in our work was quoted earlier in Section 1, accompanied by additional explanations. Moreover, the technical debt term also indicates the set of all technical debt items associated with the system [2]. An atomic technical debt item (TDI) points to an immature software development artefact indicating the difference between its current and desired state [4]. The act of reducing technical debt is called *refactoring* [8] or *remediation* [2]. The evolution of software development and delivery techniques, namely the adoption of the DevOps paradigm [9] and its Continuous Integration and Continuous Delivery practices (CI/CD) requires the technical debt perspective to follow. Hence, a broader pool of software development artefacts will be considered a part of telecommunication system or service [3], as more software artefacts undergo the technical debt dynamics. Actually, by monitoring the trends of maintainability metrics, we may determine in advance which parts of the codebase may

likely be abandoned by developers and rewritten deemed non-maintainable [10]. Developers themselves may indicate explicitly introduction of technical debt (i.e., Self-Admitted Technical Debt), and its removal in many cases takes the form of rewriting the impacted code, rather than its correction [11]. However, we should avoid a trap of understanding technical debt only according to the scope presented by the tools, e.g., SonarQube [12]. Our understanding should revolve around the difference between the current and the desired state of the artefacts constituting a system or a service. Therefore, even an evolution of the system architecture model might be constantly monitored to preserve the desired quality level [13]. Moreover, technical debt can be introduced by a shift in the context the product exists in, when the system is used in other circumstances than originally envisaged. Also, new technologies might have emerged, invalidating what was originally deemed a good design decision. Therefore, the context evolution may be perceived as an act of incurring technical debt against the product, as it results in an evolution of its desired optimal state.

As technical debt management should be the goal, not the relentless refactoring, many approaches to technical debt prioritisation emerged. The extensive secondary studies on technical debt prioritisation were presented in the work of Lenarduzzi et al. [14], and Pina et al. [15]. Some papers are mostly conceptual; others discuss frameworks standardising the process of prioritisation of technical debt [15]. In our previous work, we introduced the CoDVA technique for prioritising technical debt [16]. This approach enables valuation of technical debt to prioritise its reduction according to the business perspective, i.e., indicating changes with the potentially highest positive impact on the future evolution of the product. The evolution of the product itself should be not only driven by adjusting the codebase to enable future changes (which is a risk in agile development approach), but should ensure an explicit allocation of effort to introduce bigger (e.g., architectural) changes [16,17]. Hence continuous refactoring approach [18] driven within the context of ongoing development, may only be a partial solution. Moreover, the perception of the desired future state of the product is provided by the business representatives in the form of a product roadmap (a pipeline of future functionalities expected to be added to the system). An additional challenge for the case study design was the outbreak of COVID-19 disease and forced working-from-home policy adopted by the organisation. If the COVID-19 related health fears did not exist, the vast majority of companies would not even consider teleworking practices adoption in a massive way [19]. In the literature, the impact of the COVID-19 pandemic, especially remote work, on the operation of organisations and the solutions proposed or adopted is increasingly gaining attention: starting from frameworks to enable remote work adoption [20], through cybersecurity threats and their mitigation [21], up to assessment of the impact of forced working-from-home policy on accumulation of technical debt [22]. Particularly, in the last of these papers, Zabardast et al. concluded that there was no evidence to claim that the change between working from office to working from home resulted in an increased accumulation of technical debt. Additionally, we have found no research papers on assessing COVID-19 impact specifically on technical debt identification and prioritisation. In our case, we observed that communication style and interactions within the team were severely impacted, therefore in order to maintain a highly participative approach focused on technical debt management, the technique called the Wisdom of Crowds [5] was used for technical debt identification. It helped to mitigate communication challenges the team faced in the COVID-19 pandemic times, and to keep all team members engaged in improvement efforts. This technique was already proven successful, e.g., in improving estimation accuracy [6].

In summary, this paper drives further the adoption of the CoDVA concept for technical debt prioritisation and provides insight into further evolution of this original approach. Additionally, the case study analysed in this article proved that the engineering team can jointly prioritise technical debt in the working-from-home mode and effectively refactor it to achieve tangible results.

3. Methodology

A case study is a detailed study of a specific subject like a person, an organisation, a phenomenon, allowing comprehensive explorations of complex issues in their real-life settings. Unlike a classical experiment, where hypotheses are tested and surrounding conditions are to the large extent under control, a case study is a distinct research method that involves contextual analysis of the subject. In the work of R.K. Yin [23] a case study is defined as an empirical inquiry about a contemporary phenomenon, set within its real-world context—especially when the boundaries between phenomenon and context are not clearly defined. Therefore, this research method helps us to understand the organisational context, decisions made, their consequences, and implementation details.

This exploratory case study was conducted in a large, international company, which creates, maintains and operates mission-critical telecommunications systems. The studied application was software managing a fleet of devices constituting a video surveillance system. During a period of one year several versions of the product were released. As the product had a long history of changes and evolved rapidly in line with the company's business strategy, paying off technical debt was a lower priority in the initial phase of its development. In addition, the original team of developers had to be engaged in other tasks; therefore, the responsibility for maintenance and further development of the product was handed over to another team. Moreover, the product responsibility was transferred just before the COVID-19 pandemic started. The development team faced the situation of limited knowledge about the product, the need to expand its size by recruiting new team members, and addressing upcoming requests derived from the defined feature roadmap. Additionally, the company made a series of acquisitions, which resulted in redefining the purpose of the product analysed in this case study, extending the context in which it was supposed to be used. Finally, one development team, cooperating with many stakeholders from North America and Europe (e.g., Product Managers, Sales Representatives, project teams) was responsible for development and maintenance of this product, having a certain budget for refactoring agreed upon with the Product Manager, facing the need to determine and define technical debt management strategy.

The software development process followed the currently dominating Agile software development approach, namely Scrum [24]. This iterative process was optimised towards predictability and risk control. The team was cross-functional, building the necessary knowledge and skills according to the planned and predicted scope, and self-managing, deciding internally on individual work assignments. Agile development is conducted in phases called *sprints*. The result of each sprint, *product increments*, may constitute a new potentially releasable product version. In our case, the Product Manager, who assessed the product's value from the customer's perspective, decided each time whether the new version can be actually released. The development team, called *scrum team*, assessed its performance each sprint during events called retrospectives, and adapted their work style to improve. As the technical debt accumulation and its negative consequences were evident, impacting product quality and scrum team's predictability and *velocity* (amount of work delivered per sprint), the team adapted its development process. The implemented changes were focused on strengthening *definition of done* reflecting criteria defining finalisation of the product backlog item, and enhancing refinement of the work to be addressed in coming sprints. Such refinement sessions were focused also on identification and prioritisation of technical debt items. Moreover, the KPIs related to agile software development, like team velocity, cycle time, or success rate of successful sprint goal realisation were measured to reveal problems, potentially caused by accumulation of technical debt. Generally such practices are positively perceived by developers [25].

Considering the complexity of the situation and seeking optimal allocation of limited resources for refactoring, the team decided to use the CoDVA approach developed earlier to maintain flexibility in a continuously changing business environment. The process adopted to address the challenge of prioritising and refactoring the technical debt had to be both inclusive and providing clear tangible outcomes. The inclusion was understood as

increasing the participation of the team members, who had to grow their knowledge while working remotely. The outcomes were supposed to be evaluated against the expectations set by business optimising value generated for customers, and engineering management focused on continuous improvement. Moreover, engineering perspective on technical debt was expected to embrace a broadened focus on the product, including all software artefacts enabling efficient value creation for a customer in a continuous manner. The technical debt management process, prepared with the Scrum Team is presented in Figure 1.

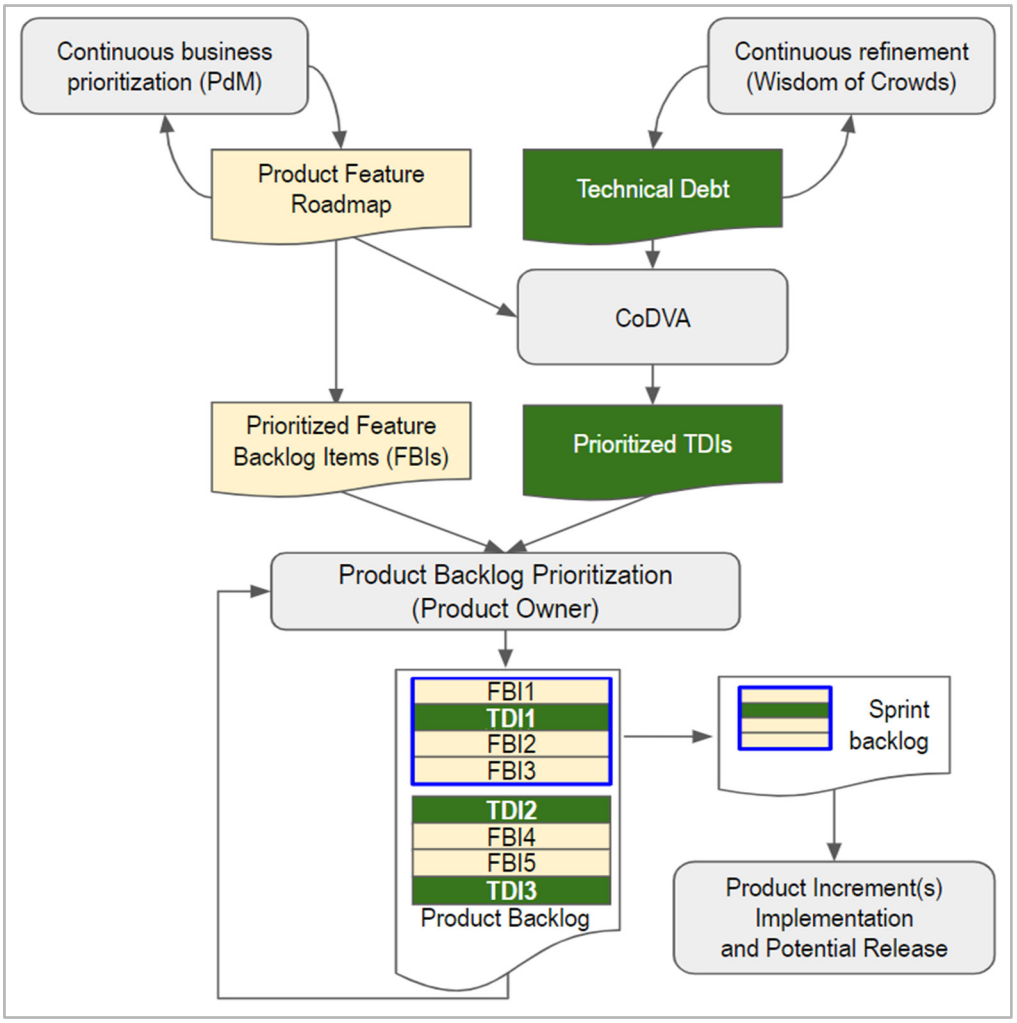

Figure 1. Technical Debt Management Process.

As the business prioritisation is a continuous activity, based on direct discussions with customers and step-by-step engagement in sales processes [26], the engineering perspective should be well aligned with its outcomes. The result of the Product Manager work is a product feature roadmap, which in turn is the source of the prioritised feature backlog expected to be built by the development team. The Product Owner, who is the only source of priorities for the development team, combines the input from various sources into one product backlog. However, all the efforts to address technical debt should be also taken into account. Based on an engineering input, collected using the Wisdom of Crowds approach (technique which will be described in Section 3.1), technical debt is identified. This process is repeated continuously as the knowledge about the codebase and understanding of the desired optimal state of the product evolve. The next step introduces a business perspective into technical debt prioritisation. For that purpose, we deploy the CoDVA technique. CoDVA approach is a continuous assessment and prioritisation of the technical debt items. It is driven by business value, which is directly assigned to each roadmap feature by the

Product Manager. Potential benefits (level of positive impact) resulting from planned refactorings are mapped on roadmap features. This helps to prioritise technical debt items against predicted future benefits (most profitable features), not only by the impact of engineering productivity or potential savings made on development effort. Ultimately, the Product Owner prepares the product backlog for the team taking into account all Product Backlog Items (PBIs), which are Feature Backlog Items and prioritised Technical Debt Items. Having an agreed-upon budget for technical debt reduction, the Product Owner decides on a single prioritised list of items to be developed by the Scrum Team. Subsequently, sprint by sprint, the development team delivers Product Increments which may be shipped to the customer and which contain new functionality and improvements resulting from technical debt reduction efforts. After each sprint, the product backlog is reprioritized based on the new data available: remaining items in the product backlog, new requests coming from the business and newly identified technical debt items. Summarising, business priorities are continuously refined and influence the prioritisation of technical debt items, which in turn land in the product backlog realised iteratively in sprints. The resulting product increments add value for the customers and consist of both new functionality and product improvements. As we can see, the CoDVA technique for technical debt prioritisation drives refactoring effort in parallel with adding new functionality.

3.1. Managing in the COVID Era

The COVID-19 pandemic outbreak in March 2020 forced software development organisations into remote-only mode of operation. In that unprecedented situation, developers' work habits, communication patterns, and daily routine changed dramatically. Uncertainty about the evolution of the situation made the switch even harder to accept. The sudden challenge posed serious risks to organisations stability and profitability. In our case study, half of the team was built right before the pandemic outbreak and further team expansion was conducted remotely. Building the team in such a mode, expecting efficient knowledge sharing became a challenge. Each new developer underwent remote onboarding, knowledge sharing and mentoring process to become a fully productive member of the team. Moreover, the team had to take responsibility for a product undergoing several significant changes caused by redesign of a testing process, acquisitions of other companies producing devices which had to be supported, and development of several new features. Due to a rapid growth of demand for new functionality, where development of the new requests was prioritised, the technical debt grew. Some technical debt was incurred pragmatically (to enable faster support for devices produced by acquired companies), some emerged as a result of changing the desired state of the product (data structures became suboptimal to address broader pool of devices and integration with Video Management Software product) and some was inherited from the software development organisation originally creating the product (related to automation test framework). Therefore finding a way of engaging the new team in sharing their perspectives on technical debt, building its complete picture to prioritise and execute refactorings, led us into adaptation of the process. Thus we established a communication environment resembling as close as possible normal interaction in the office. The video conferencing tools were used to ensure presence. Every participant was visible to the group, so people were fully engaged during the brainstorming sessions. The meetings followed the Wisdom of Crowds approach; conversations were moderated to ensure that each developer was heard and that their input was taken into account. Furthermore, all propositions were equally treated and evaluated, to avoid homogeneity in interpretations. Finally, prioritisation was performed according to the CoDVA approach, switching focus from people proposing changes into agreement on the algorithm used.

3.2. Wisdom of Crowds

The technique called Wisdom of Crowds which is used in our case study for technical debt identification, was already proven to be reliable and accurate in data collection and predictions in other fields. It was proposed by Surowiecki [5], and was also adopted

by one of the authors in one of the previous experiments [6]. Here, we will recall the crucial facts about that. The main advantage of this approach lies in the understanding of sociological aspects of human interaction. Each engineer has a different and unique technical background and understanding of the product: its many hidden characteristics, interactions of its modules, etc. This is the reason why everyone may provide meaningful insight into technical debt presence in the codebase and its consequences, and why better data may be collected from a more diverse and bigger team. In our case, the applied Wisdom of Crowds technique strives for answering how the engineers' continually growing knowledge of the system and understanding of its internal mechanisms and dependencies could be more effectively used in order to improve its quality.

Surowiecki analysed social situations in which a group of people, i.e., crowd, was unable to produce good judgement, because one of several critical factors for good decision making was missing. He called the group of the people, where all necessary aspects of sound judgement are in place, a *Wise Crowd*, and claimed that there are four elements required to form such an entity:

1. Diversity of opinion: each person has private information, which may be just an interpretation of the known facts.
2. Independence: opinions of the people are not determined by the opinions of those surrounding them.
3. Decentralisation: people are able to specialise and make use of local knowledge.
4. Aggregation: some mechanism exists to convert private judgments into a group decision.

Therefore, if the decision making environment is not ready to accept the crowd, failing to represent the majority of opinions, views, or perspectives, the benefits of individual judgments and private information vanish. Surowiecki also discussed common deficiencies, causing crowd to turn into an irrational one:

1. Homogeneity: the lack of diversity within a crowd, reducing variability in viewpoints to be taken into account and resulting in limited amounts of private information to be heard.
2. Centralization: limiting the data collection process to ask only those who are supposed to be a wise representation of the team. Usually if it comes to strategic decisions, the knowledge of lower-level engineers, sometimes very unique when it comes to complex systems or situations, is omitted.
3. Division: the failure to aggregate all pieces of information; for example when all necessary information is available but information held by one organisational department is not accessible to another.
4. Imitation: a sequence of activities may result in an information cascade, where only the first people are adding their knowledge to the common pool used by others. When the first decision seems rational, those who came next are willing to mimic that behaviour.
5. Emotionality: describes even a broader group of effects: e.g., a feeling of belonging leading to peer pressure, and herd behaviour.

Hence, in order to create a wise crowd during the technical debt identification phase, the following aspects need to be preserved:

1. Independent data collection. Each engineer provides his/her view on technical debt, identifying technical debt items and their potential consequences independently. No discussions, or alignment happens at this stage to avoid cascading of assessment. It can be done via online collaboration tools like Jamboard [27] or MURAL [28].
2. In the next step, the list of technical debt items is reviewed to remove duplicates, and identified technical debt items are stored as records in Atlassian Jira [29].
3. No input is considered better than the other, no prioritisation and weighing factors are applied. Each proposal has exactly the same importance for producing the technical debt view, consisting of unique technical debt items. In such a way the final Wisdom of Crowds rule on aggregation is preserved.

4. The next step is focused on mapping technical debt items on the feature roadmap. The weight (cost) of technical debt remediation as well as technical debt impact on each feature on the roadmap (how much the feature will benefit from it (low, medium, high, very high) are estimated independently by engineers. The final values are average of these estimates. This step may be organised slightly differently, estimation of cost may be produced using a technique called Planning Poker (or Scrum Poker) [30].

The human variability in the process is an inherent factor for any software engineering activity, but we may build on this phenomenon, and make it the strength of the technique used. For the Wisdom of Crowds any group should be as diverse as possible to take advantage of different points of view. In our case the whole Scrum Team participated in the exercise. This may lead to more variation, but this variation actually constitutes the strength of the Wisdom of Crowds approach. Additional strength of this approach was highly visible during COVID-19 pandemic, when forced working-from-home policy was institutionalised. Having a clearly defined algorithm, a remote data collection and estimation sessions were easy to establish and execute. All team members knew and followed the process.

3.3. CoDVA

The Continuous Debt Valuation Approach (CoDVA) for technical debt prioritisation was introduced in our previous work [4]. For completeness of the overview, the most important facts are quoted in this subsection, and clarifications are added when some modifications were applied. In the proposed approach, we assessed the entire service offered (the product) in terms of desired improvements. Then our approach embraced the business perspective, based on the data provided by the Product Managers. A single business prioritisation value assigned to every feature on the product roadmap aggregated several aspects, such as potential benefits, investment size, strategic aspects, time criticality. In this way, this perspective extended the previous case study where business input was calculated based on financial aspects only. The CoDVA approach served as a reference point for measuring the value of technical debt items at any given point in time.

The CoDVA approach is aligned with a structured sales process, embracing a business perspective. It facilitates continuous technical debt valuation against a predicted product roadmap. Any set of features may have a sales prediction (or business value) associated with them. Therefore, they should not be evaluated in isolation. The same applies to technical debt items, as they may be dependent on each other, and this aspect is still being investigated by us. The software release may contain not only features directly associated with sales opportunities, but also all software artefacts influencing value creation for a customer: improvements for delivery speed (optimization of the software deployment), enhancements in monitoring, and updates of logging capabilities, etc. If the product is offered as a service, the maintenance cost of the software solution directly impacts profitability of the service to which customers subscribe. The CoDVA prioritisation value may change in time, but at any decision point—when improvements are to be prioritised—the current business perspective is considered. The model provides a relevant prioritisation technique. Therefore, valuation, based on business prioritisation, serves a prioritisation purpose: defining the order and a relative distance between TD Items. Moreover, the prioritisation of TD Items is done automatically. The implementation results confirm that close alignment between business and engineering may focus the discussion on the implementation of changes perceived as the most profitable. Even though the product roadmap may change significantly, the consistent stream of improvements will ensure product resiliency.

3.4. Technical Debt Identification and Prioritization

Firstly, technical debt was collected during brainstorm sessions within the development team. The results of these sessions were discussed with stakeholders responsible for business return of engineering investments (Product Managers). Such refined technical debt was stored as separate records in the development database, one record for each atomic TD Item. A single record, stored in Atlassian Jira tool, described in detail: scope of

work, estimated effort, product context pointing to an area requiring update, and potential consequences associated (justification for a change).

Secondly, the prioritised business roadmap was discussed with the Product Managers. The differences between the CoDVA approach presented in our previous work [4] and the currently used one were briefly described earlier. The most important one referred to the refined valuation process. Instead of using pure monetary value of benefits (deriving it from expected sales), a broader perspective was adopted, taking into account parameters like investment size, alignment with overall product strategy, time-to-market expectations, and many more. As a result, the business value—called feature score (*f_score*)—was calculated by the Product Manager for each feature. The way how these parameters were combined to constitute feature score was at the discretion of the Product Management organisation, which is responsible for business return on engineering investments. The normalised f_score value (in the range 0–1) was called relative feature priority *f_priority*. This formed a basis for prioritisation of the product roadmap. Thirdly, a mapping of TDIs on the feature roadmap was prepared (see a sample excerpt in Table 1 below). Such a matrix helped to understand the *TDI_impact* of the refactoring effort on the product (provided as a t-shirt size estimation, respectively: S, M, L, XL), explicitly listing anticipated benefits.

Table 1. Impact of TDIs on feature roadmap.

Technical Debt Items			Features (Id and Size)								
			F10	F11	F12	F13	F14	F15	F16	F17	F18
Id	Type	Size	XL	XL	XL	XL	L	S	M	S	XL
TDI-1	testability	M	XL	XL	XL	-	S	S	M	XL	S
TDI-2	testability	S	XL	L	L	-	S	S	M	XL	S
TDI-6	CI/CD	XL	XL	XL	XL	-	-	S	-	M	-
TDI-3	CI/CD	M	-	XL	XL	-	L	L	L	L	L
TDI-4	maintainability	S	XL	XL	S	-	-	L	S	-	-
TDI-8	CI/CD, Perfomance	S	XL	XL	XL	-	-	-	-	-	-
TDI-15	testability	S	-	XL	XL	-	M	M	M	M	M
TDI-17	CI/CD	S	-	XL	XL	-	S	S	S	S	S
TDI-18	CI/CD	S	-	XL	XL	-	S	S	S	S	S
TDI-7	Development Environment	M	XL	-	XL	-	-	-	-	XL	-
TDI-10	CI/CD	S	L	-	XL	-	M	M	M	M	M
TDI-14	Development Environment	S	-	-	XL	S	-	-	-	S	-
TDI-9	maintainability	S	L	-	S	-	-	-	-	-	-
TDI-19	CI/CD, testability	M	M	XL	XL	-	M	M	M	M	M

Next *CoDVA_INDEX* for each refactoring was calculated, based on: feature priorities, feature sizes (*f_size*), TDI impacts, TDI costs, and a *release_cadence* factor. The more features were positively impacted by a given TDI refactoring, the higher CoDVA_INDEX was associated with it. Additionally, feature scores were modified based on their assignment to future releases of the product (release_cadence). The later in the release chain the feature was planned, the lesser was its value taken into account for calculating CoDVA_INDEX, as the scope volatility for future product releases had to be taken into account. Expecting no changes in the scope of future releases, we may keep the values intact for the complete roadmap (release_cadence $\rightarrow$ 1), expecting ~100% change every release, there is no point for assessing anything beyond the features planned for the next release (release_cadence $\rightarrow$ 0). The parameter release_cadence was calculated in retrospective mode, based on the content of the recent product releases. As a result, a prioritised list of TDIs was created, and a refactoring effort was planned accordingly. The following formula was developed to calculate CoDVA_INDEX for each technical debt item:

$$\text{CoDVA\_INDEX}_i = \frac{\sum_{j=1}^{m} \text{f\_priority}_j * \text{release\_cadence}_j * \text{f\_size}_j * \text{TDI\_impact}_{ij}}{\text{TDI\_cos t}_i} + \sum_{k=0}^{n} \text{CoDVA\_INDEX\_dependent}_k$$

The following parameters: impact of TDIs on feature roadmap, feature size, TDI size, which were provided as t-shirt size values, were mapped on numerical values (in the range 0–1). Taking into account TDI_impact (evaluated against costs related to implementing a new feature, see assessment presented in Table 1), the potential financial benefits might be derived. However, as in this case study a fixed amount of effort was available for improvements, the more important aspect was the relative prioritisation among technical debt items, which ultimately drove the scope planned for implementation. This prioritisation was provided by comparing CoDVA_INDEX for all technical debt items. Ultimately, the last part of the formula, a sum of CoDVA_INDEX values for dependent TDIs, was favouring these investments which were more generic, supporting or enabling other ones.

3.5. Data Collection Process

In this case study, the development of the prioritised sequence of these improvements (TDIs) and their impact on the product was tracked to assess the effectiveness of the CoDVA approach, i.e., whether the improvements were executed as per plan and produced tangible benefits, as it was stated in our research question. As the dynamics of a feature roadmap played an important role in prioritising technical debt items, we were monitoring the volatility of relative feature priorities for both: features already delivered and features still planned in the roadmap. The snapshot of feature priorities was captured each time reprioritization with the business team was made. Next, based on priorities on the feature roadmap we were recalculating the CoDVA_INDEX and prioritised the list of technical debt items accordingly. For each of the TDIs, we monitored the status of implementation, and if refactoring could not be conducted or was delivered prior to expectations, we collected data to understand the causes.

Additionally, in order to observe the impact of these changes on the product, we were monitoring a number of stabilisation cycles required to release a new product version. Each stabilisation cycle embraced execution of a test plan and correction of all issues preventing the release of a new product version. Once all critical issues were addressed and the test cycle did not expose any new blockers, the new product version could be released.

3.6. Data Analysis Process

In each sprint, the development team met for an event called *refinement*. Its goal was to decompose the planned functionality in a way it was understood and ready for development in the next one or further sprints. This was also an opportunity to decompose technical debt items the same way. At this moment, the team assessed the potential value of technical debt items, decomposed the top ones and, just before the next sprint started, selected ones to be refactored (incorporating technical debt reduction plans). After the end of the sprint, the product increment was demonstrated to stakeholders, including the results of the technical debt reduction efforts. Additionally, an assessment of prioritised technical debt was made to understand whether it was following the plan (i.e., whether currently prioritised items per alignment with the product feature roadmap were taken for refactoring). Every release, usually after a few sprints, the assessment of the feature roadmap was conducted. As a result, feature priorities were reviewed and the technical debt roadmap was revised to reprioritize existing items and incorporate newly identified ones. Additionally, an ad-hoc update of the technical debt roadmap, or the feature roadmap was called if either the engineering team or the business team were getting new data which was potentially affecting the plans. The engineering team ran into unexpected complexity or wanted to consult their ideas with the Product Manager, and on the other hand, the Product Manager had new updates reflecting the customer feedback and prioritisation. In extreme cases, a sprint or a release was cancelled to prioritise a new urgent business

need. Therefore, when the decision of taking new scope for implementation was made, the data captured at this stage was used to understand the dynamics of the feature roadmap, and assess how the actual technical debt reduction efforts reflected the recommended prioritisation of technical debt based on CoDVA. The results were baselined in the table reflecting relative prioritisation at the moment the decision was made (see Table 2 and discussion in Section 4).

Table 2. Technical Debt Reduction—Prioritized TDIs.

TDI	TDI Type	Status	Comment
TDI-1	testability	done	
TDI-2	testability	done	
TDI-3	CI/CD	done	
TDI-4	maintainability	done	
TDI-5	maintainability	done	
TDI-6	CI/CD	too big	
TDI-7	Development Environment	blocked	
TDI-8	CI/CD, Performance	done	
TDI-9	maintainability	done	
TDI-10	CI/CD		
TDI-11	security		
TDI-12	maintainability		
TDI-13	maintainability		
TDI-14	Development Environment	done	context
TDI-15	testability		
TDI-16	testability		
TDI-17	CI/CD	done	context
TDI-18	CI/CD	done	context
TDI-19	CI/CD, testability		
TDI-20	logging/telemetry		
TDI-21	maintainability		
TDI-22	security		
TDI-23	maintainability		
TDI-24	testability		
TDI-25	maintainability		
TDI-26	maintainability	done	context
TDI-27	CI/CD	done	context
TDI-28	maintainability	done	context
TDI-29	maintainability	done	context
TDI-30	logging/telemetry, performance		
TDI-31	logging/telemetry, performance		
TDI-32	maintainability		
TDI-33	maintainability		
TDI-34	maintainability		
TDI-35	maintainability	too big	
TDI-36	maintainability		

4. Discussion of Results

Technical debt prioritisation is a dynamic process, and should reflect a business perspective to provide the highest return on investment. Observing high dynamics of

the business roadmap, it is evident that we should not base our judgement of benefits on engineering assessment only. Pure engineering perspective may result in creating waste by optimising codebase which will not be changed in the future or, even worse, which will be retired. In the latter case, whatever is the engineering perception of the technical debt associated with the code to be retired, the potential value of refactoring efforts is zero from the customer or business perspective.

As the case study started, the forced working-from-home policy was applied; therefore, our initial assumptions had to be modified to be based only on remote participation, discussions, and data collection processes. For that reason a clear and well-defined data collection process was established, and participation of the team in the data collection was enhanced by using the Wisdom of Crowds approach. This technique, due to its clear guidelines, and encouraging the participation of every engineer involved in development, helped to establish a clear case study protocol for the technical debt identification phase. Fortunately, the technical debt prioritisation phase was based on an automated algorithm implementing CoDVA technique, which was not impacted by changing work conditions.

First, we monitored the trend of the relative value of features from the business perspective that evidently demonstrated dynamics of the product roadmap. For clarity, we present only 13 of them: the ones that exceeded (at least once) the arbitrarily chosen threshold of relative feature priority (0.35). They are presented here separately in two figures: Figure 2 shows how priorities changed over time for six features not yet delivered and Figure 3 presents how priorities changed over time for seven features that were already delivered to customers as a part of subsequently released new product versions.

Moreover, in Figures 2 and 3 only a subset of features is presented for legibility reasons, as the number of features was a few times higher. As we can see, even though the prioritisation was dynamically updated, we could find a situation where two (or more) features were having the same relative priority (FN7 in Figure 3 and F2 in Figure 2). In such a situation, the engineering team consulted with the Product Manager which feature should be selected for implementation. In this case feature F2 was chosen.

An additional noteworthy observation on the feature roadmap dynamics is the fact that features prioritised on top of the list once at a certain point of time may be deprioritized, and not implemented at all. This is the result of the dynamics in the sales process; e.g., the window of opportunity for the sale may disappear (see features F3, F5, F6 in Figure 2).

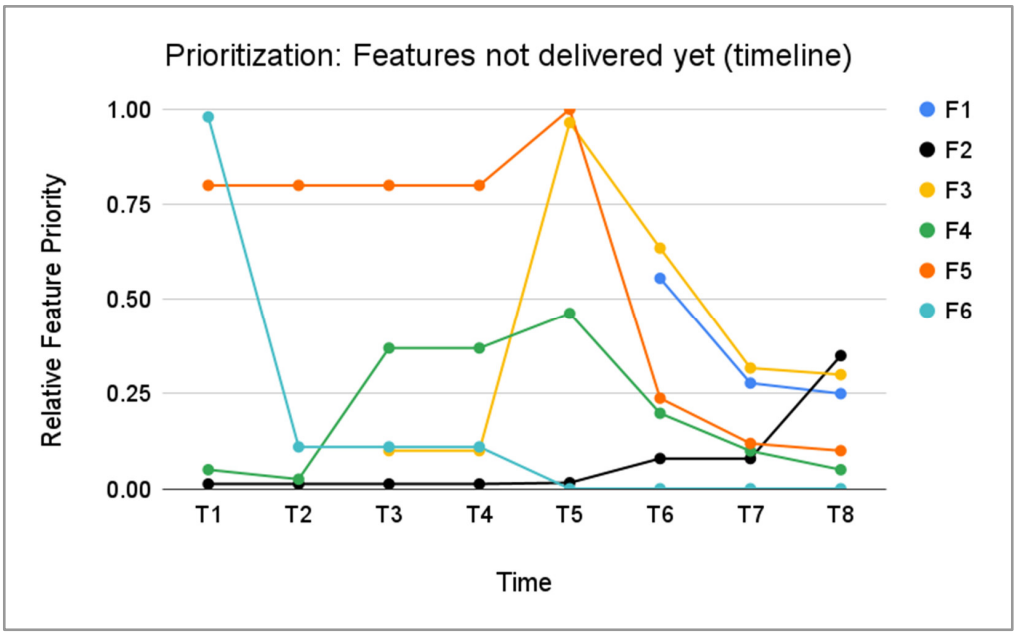

Figure 2. Product roadmap: planned features.

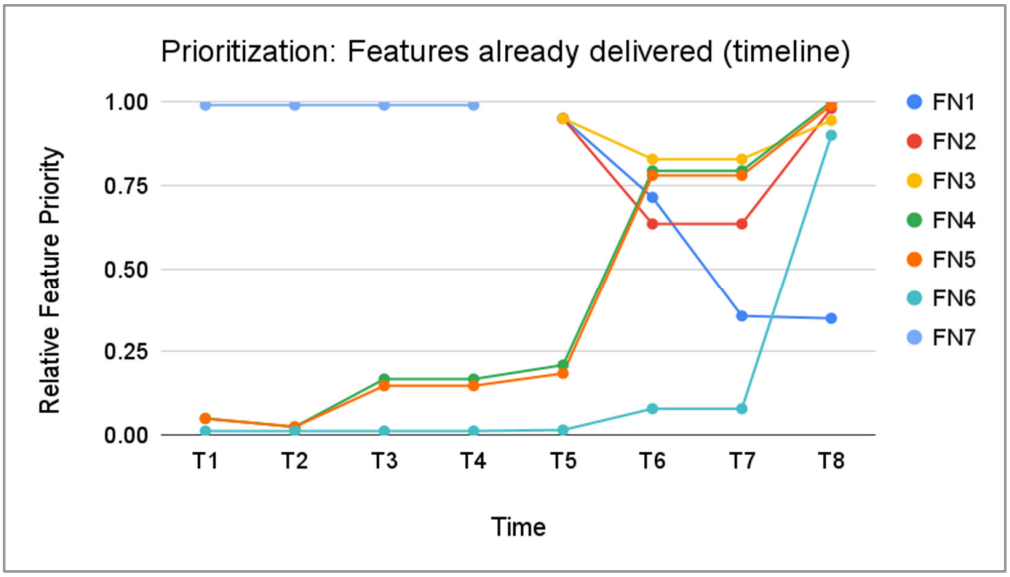

Figure 3. Product roadmap: implemented features.

When we examine the data to understand variability for selected features, the following complementary view may be presented (see Figures 4 and 5 below). Note that T1-T8 reflects the moment in time, when observation was collected.

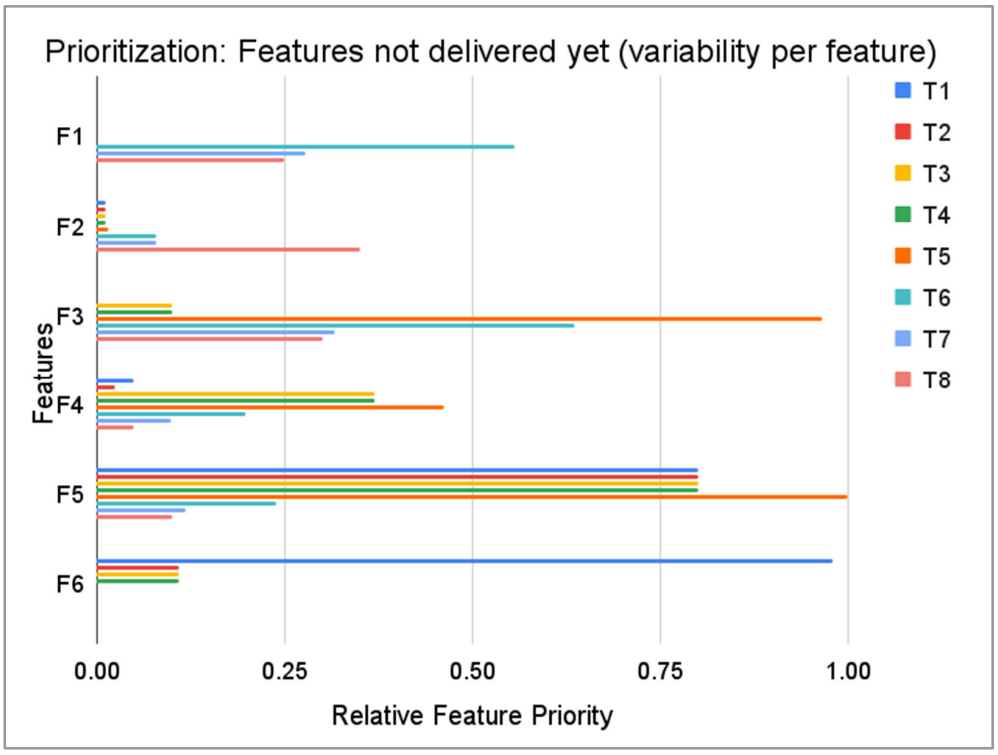

Figure 4. Changes in prioritisation per feature: features not yet delivered.

Next we have collected all TD Items (TDIs) which were identified, gathered, and described by the development team prepared for implementation. This exercise was done during refinement sessions held every sprint. Each item (see: Table 2) was categorised to understand its type and influence over DevOps activities. Furthermore its priority (relative position) reflected the decision time, and status indicates whether TDI was already refactored, and if there was a valid reason not to implement the prioritised TDI, it was explicitly mentioned ("too big": could not have been prioritised over features, "blocked":

in one case TDI awaited other implementations, might be either feature-based or different TDI). As the order of implementation of TDIs was analysed by us in this paper, for each TDI not following the prioritisation (refactored before higher priority ones), we found out that "context" was the reason for such a decision. It means that development in the given area was pending, and the scrum team used a common practice called *continuous refactoring*, addressing smaller-size issues while implementing other product backlog items. Additionally "context" also meant TDI refactorings stemming from broadened product definition (DevOps perspective), they were identified and implemented in parallel with currently implemented functionality.

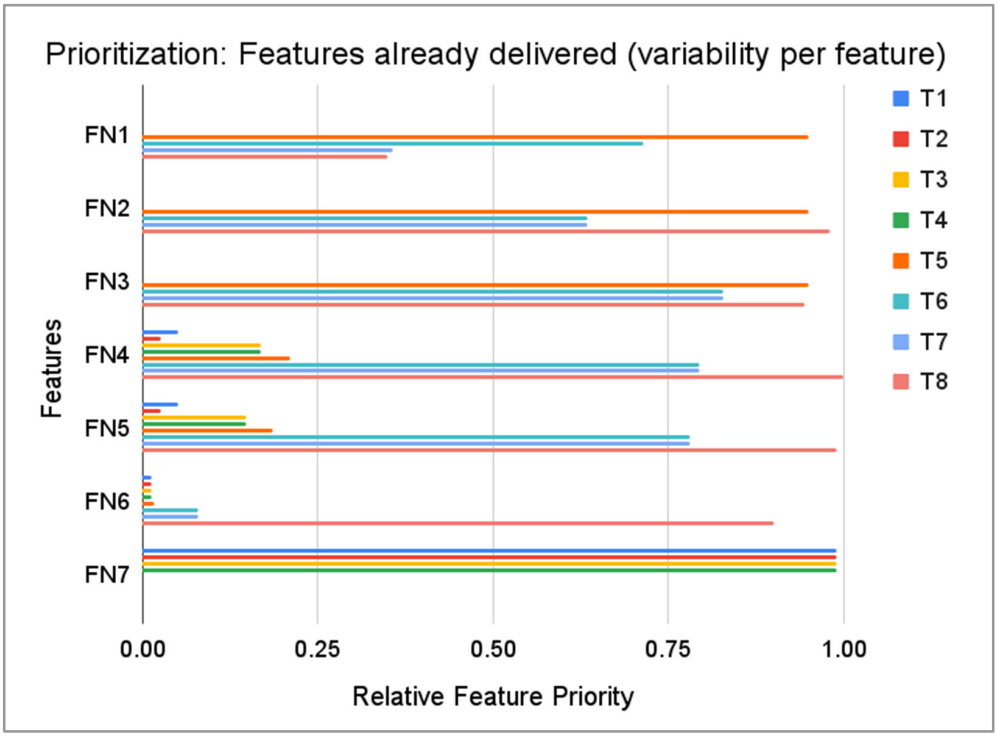

Figure 5. Changes in prioritisation per feature: already delivered features.

We should keep in mind that the roadmap of improvements should reflect dynamics of the feature prioritisation. However, in our case study these changes were negligible against the conclusions made. The reason for this is the fact that the majority of technical debt items actually helped in future development of many features concurrently. In order to answer the question whether execution followed the plan we assessed the moments when a given TDI was selected for refactoring. Only then could we determine whether the decision adhered to the prioritisation. Throughout the period of the case study these TDIs which raised concerns (prioritised low and executed before others) were never prioritised high. Hence the simplified view of the TDIs prioritisation dynamics was presented, and the analysis focused on the reason why several lower priority TDIs were addressed. Therefore, even though the top most items were addressed (TDI1–5, TDI7–8), there were a few TDIs addressed which were prioritised significantly lower on the list.

The reason for that was continuous refactoring effort, a practice well aligned with agile development pace. Once a development context was open, some TDIs might be addressed with minimal cost while adding new functionality. However, bigger changes (e.g., some significant architectural improvements) could not be addressed this way, as the major focus of the team, within a sprint lasting two weeks, is to deliver an explicitly planned scope (sprint goals).

Another observation was the fact that some technical debt items would not be addressed at all, as their sheer size was too big to justify investments over new roadmap features awaiting implementation. In our case the larger size correlated with an unclear

scope of work. An example here could be refactoring of all regression tests to use a new simulator or be aligned with a different test framework. In such a situation, the solution was to decompose it into smaller TDIs and propose alignment of such effort with ongoing product development (new features).

An additional early observation (already included in Table 2) was the broadened scope of technical debt reduction activities to embrace also CI/CD perspective. If these improvements were not included in the scope, the engineering team would have (or at least should have) tried to prioritise them over the ones explicitly listed anyway. This was because the faster feedback from the development and greater knowledge on the product usage (for instance, due to telemetry) could cause the better codebase quality, and finally the product.

Finally, we observed that the investments made for this product started paying off and resulted in less error-prone code and faster time-to-market (Table 3). For each product version we checked the number of stabilisation cycles required to remove significant issues before the release. We discovered that in this one-year-long case study the number of stabilisation cycles dropped to 1, due to improvements made in overall product quality (including automation of testing, improvements in development environment and other activities influencing delivery of value to customers).

Table 3. Stabilisation cycles required to release a new version of the product.

Product Version	Stabilisation Cycles
v1	6
v2	2
v3	2
v4	2
v5	1
v6	1

5. Threats to Validity

In this section, following the recommendations of R. Yin on case study research [23], the major threats to validity of the case study are discussed, namely: construct validity, internal validity, reliability, and external validity. Additionally, means to mitigate these threats are presented. The purpose of this section is to recognize and describe any factors that might have undeserved influence on the research or distort the data collection process and hence the findings. That in turn should increase the credibility of the analysis and presented results.

Construct validity assesses to what extent our scales, metrics and instruments actually measure the properties they are expected to measure [31]. We have collected the complete list of technical debt items identified and acted upon by the engineering team. That is subjective by its nature, however the goal was to understand how actual technical debt mitigation efforts were executed. The major threat associated with this approach is the fact that over time priorities may change. However, in our case study these changes were negligible against the conclusions made, even though prioritisation of features changed. The reason for this is the fact that the majority of technical debt items actually helped in future development of many features at the same time. The situation might be very different if specific TD Items were very narrow-focused and referred to rarely touched parts of the code. Therefore, we claim that the actual prioritisation and execution of refactorings provided sufficient insight into the process and its results.

An additional measure on quality—stabilisation cycles required to release a new version—can be considered as a key performance indicator of screening effectiveness against subsequent product releases. Ultimately, we claim that both metrics sufficiently support the conclusions on process effectiveness of prioritisation of technical debt reduction based on CoDVA approach.

Internal validity assesses the level of confidence that the causal relationship being tested is not influenced by other factors or variables. Our case study was not a laboratory experiment, but was conducted in a complex environment over a one-year-long period. Even though some external factors might influence the process, the results were very much focused on causality of the refactoring process. The Agile Scrum Team was following the strict *definition of done* for the work, the development process was standardised and followed continuous integration practices, which were subject to a change mainly—if not only—according to technical debt reduction activities. Furthermore these activities were the subject of our study. An additional factor strengthening the conclusions was the fact that the very same scrum team was solely responsible for both the product implementation and the development process. Therefore, we consider the presence of a cause-and-effect relationship.

Reliability focuses on the ability to replicate results, i.e., that similar results may be produced under consistent conditions by different researchers. To mitigate the threat of lack of replicability, the case study protocol was described in Section 3. The whole Scrum Team was present during the technical debt identification phase, and followed the Wisdom of Crowds approach. Prioritisation of technical debt items was conducted according to the CoDVA approach defined by the authors. Therefore, the process was standardised, and metrics on prioritisation changes were collected automatically from the tools: Atlassian Jira [29] and Aha! [32]. This helped with close alignment of the engineering technical debt items and the business priorities for the product feature roadmap.

External validity addresses the generalisability, i.e., the extent to which the results and conclusions of a case study can be applied in different contexts. The CoDVA prioritisation method presented in this case study was already applied in another software domain, addressing prioritisation of technical debt in the large wireless telecommunication system [4]. The effectiveness of the approach was confirmed there as well; it was defined as the profitability of technical debt repayment (actual implementation of changes perceived as the most beneficial taking into account business financial perspective). However, in the case study analysed in this paper, a closer look into specific implementation steps provided us with deeper insight into the actual dynamics of the technical debt reduction process and allowed us to answer questions about some deviations from the expected implementation of the technical debt reduction process.

Therefore, the challenges discussed for this particular product development process might be considered as representative ones, and we assume that this approach can be replicated and its results may be generalised.

6. Conclusions and Future Work

COVID-19 pandemic outbreak triggered a search for a solution to address inter-team dynamics of communication caused by forced working-from-home mode, especially that this team took at this very moment the responsibility for product development and its maintenance strategy. Therefore, in order to ensure a highly participative approach focused on technical debt management, the Wisdom of Crowds technique was used for technical debt identification and its alignment with business roadmap. It helped to mitigate communication challenges the team faced during the COVID-19 pandemic time, and keep all team members engaged in improvement efforts.

An extensive discussion on technical debt prioritisation techniques was covered by the initial paper introducing the CoDVA approach. We perceive the originality of the CoDVA approach in continuous prioritisation of TD Items relying on the business perspective. This enables relative comparison between TD Items, which in turn drives prioritisation [4]. The techniques of prioritising technical debt as well as understanding the dynamics of its reduction process allow for rational spending of limited funds and reasonable balancing of efforts between the software product development and optimization of the current codebase. Adopting the CoDVA approach for technical debt prioritisation resulted in tangible benefits. Particularly, one year after technical debt reduction effort was incorporated into the process,

releasing a new product version required 6 times less stabilisation cycles (removal of key issues discovered at the final stages of testing) than initially. The engineering team improved the product in a way that adding new functionality was easier and less error-prone, hence achieving faster time-to-market.

Technical debt refactoring was following the plan in general (top-most improvements were applied); however, a few lower priority TDIs were refactored as well. Considering the overall cost of investments this is a desired situation, as the engineering team used the opportunity of ongoing development not only to add new functionality but also to improve the exact part of the code they were currently working on. The cost of adding a refactoring change in the same context as a new development is usually significantly lower than opening a new context and introducing the change there, which makes the continuous refactoring approach useful and desired.

Another factor which should be considered is sizing of technical debt items. They cannot be significantly bigger than the effort required to develop a new feature, as they will be constantly postponed. The biggest size of a single TD Item implemented in this case study was not surpassing the cost of a middle-size feature. Moreover, refactoring of these bigger TD Items should be explicitly planned and agreed upon with the Product Manager. Finally yet importantly, we also observed some minor dynamics of TDI priorities presented in Table 1 caused by the changing feature roadmap priorities. However, as the fluctuation was negligible, we refrained from explicitly showing this dynamics.

In order to achieve better predictability of the execution of technical debt roadmap we claim that the following rules should be applied:

- Proper sizing of TD Items small enough to be considered for implementation,
- Shifting focus towards more costly TD Items,
- Limiting the number of explicitly allocated TD Items for refactoring (limiting work-in-progress),
- Embracing continuous refactoring approach for smaller TD Items,
- Broader, service-like, product context: CI/CD related refactorings should be treated with the same focus as the sheer product codebase (they should be considered a part of a product by definition), and prioritised accordingly.

As this is the second application of the CoDVA technique, we assume that the proposed approach can be replicated, and the results may be generalised for telecommunication software. The future work will embrace finalisation of this case study, further analysis of TDI prioritisation dynamics considering a more diverse set of TDIs, as in this case mostly generic improvements (affecting a higher amount of features) were implemented. Finally, the results of applying the CoDVA approach should be compared with other prioritisation techniques.

Author Contributions: Conceptualization, M.G.S.; methodology, M.G.S.; software, M.G.S.; validation, M.G.S. and P.C.; formal analysis, M.G.S.; investigation, M.G.S.; resources, M.G.S. and P.C.; data curation, M.G.S.; writing—original draft preparation, M.G.S. and P.C.; writing—review and editing, M.G.S. and P.C.; visualisation, M.G.S.; supervision, M.R.W. and P.C.; project administration, M.G.S. and P.C.; funding acquisition, M.G.S. and P.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work is funded by the Ministry of Education and Science in Poland under grant no. 45/DW/2017/01 (Industrial Doctorate Programme) in cooperation with Motorola Solutions, Kraków, Poland, and AGH University of Science and Technology, Institute of Telecommunications, Kraków, Poland.

Acknowledgments: We want to express our gratitude to Elżbieta Stochel who supported the work, relentlessly editing and reviewing the paper.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Kruchten, P. The end of agile as we know it. In Proceedings of the ACM International Conference on Software and System Processes (ICSSP), Montreal, QC, Canada, 25 May 2019; IEEE: New York, NY, USA, 2019; p. 104. [CrossRef]
2. Kruchten, P.; Robert, N.; Ozkaya, I. *Managing Technical Debt: Reducing Friction in Software Development*, 1st ed.; Addison-Wesley Professional: Boston, MA, USA, 2019.
3. Stochel, M.G.; Chołda, P.; Wawrowski, M.R. Adopting DevOps Paradigm in Technical Debt Prioritization and Mitigation. In Proceedings of the 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Maspalomas, Spain, 31 August–2 September 2022; pp. 306–313.
4. Stochel, M.G.; Chołda, P.; Wawrowski, M.R. Continuous Debt Valuation Approach (CoDVA) for Technical Debt Prioritization. In Proceedings of the 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Portoroz, Slovenia, 26–28 August 2020; IEEE: New York, NY, USA, 2020; pp. 362–366. [CrossRef]
5. Surowiecki, J. *Wisdom of Crowds. Why the Many Are Smarter Than the Few*, 1st ed.; Abacus: London, UK, 2005.
6. Stochel, M.G. Reliability and Accuracy of the Estimation Process—Wideband Delphi vs. Wisdom of Crowds. In Proceedings of the 35th Annual Computer Software and Applications Conference, Munich, Germany, 18–22 July 2011; IEEE: New York, NY, USA, 2011; pp. 350–359. [CrossRef]
7. Stochel, M.G.; Chołda, P.; Wawrowski, M.R. On Coherence in Technical Debt Research: Awareness of the Risks Stemming from the Metaphorical Origin and Relevant Remediation Strategies. In Proceedings of the 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Portoroz, Slovenia, 26–28 August 2020; IEEE: New York, NY, USA, 2020; pp. 367–375. [CrossRef]
8. Avgeriou, P.; Kruchten, P.; Ozkaya, I.; Seaman, C. Managing technical debt in software engineering (dagstuhl seminar 16162). *Dagstuhl Rep.* **2016**, *6*, 4. [CrossRef]
9. What is Devops? Available online: https://azure.microsoft.com/overview/what-is-devops (accessed on 2 April 2022).
10. Karanikiotis, T.; Papamichail, M.D.; Symeonidis, A.L. Analyzing Static Analysis Metrics Trends towards Early Identification of Non-Maintainable Software Components. *Sustainability* **2021**, *13*, 12848. [CrossRef]
11. Aversano, L.; Iammarino, M.; Carapella, M.; Vecchio, A.D.; Nardi, L. On the Relationship between Self-Admitted Technical Debt Removals and Technical Debt Measures. *Algorithms* **2020**, *13*, 168. [CrossRef]
12. SonarQube. Available online: https://www.sonarqube.org (accessed on 16 September 2022).
13. Alti, A.; Boukerram, A.; Roose, P. Context-aware quality model driven approach: A new approach for quality control in pervasive computing environments. In Proceedings of the 4th European Conference on Software Architecture, Copenhagen, Denmark, 23–26 August 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 441–448.
14. Lenarduzzi, V.; Besker, T.; Taibi, D.; Martini, A.; Fontana, F.A. Technical debt prioritization: State of the art. A systematic literature review. *arXiv* **2019**, arXiv:1904.12538.
15. Pina, D.; Goldman, A.; Tonin, G. Technical Debt Prioritization: Taxonomy, Methods Results, and Practical Characteristics. In Proceedings of the 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Palermo, Italy, 1–3 September 2021; IEEE: New York, NY, USA, 2021; pp. 206–213. [CrossRef]
16. Kruchten, P. Voyage in the Agile Memeplex: In the world of agile development, context is key. *Queue* **2007**, *5*, 38–44. [CrossRef]
17. Stochel, M.G.; Wawrowski, M.R.; Waskiel, J.J. Adaptive agile performance modelling and testing. In Proceedings of the 36th Annual Computer Software and Applications Conference Workshops, Izmir, Turkey, 16–20 July 2012; IEEE: New York, NY, USA, 2012; pp. 446–451. [CrossRef]
18. Vassallo, C.; Palomba, F.; Gall, H.C. Continuous refactoring in CI: A preliminary study on the perceived advantages and barriers. In Proceedings of the International Conference on Software Maintenance and Evolution (ICSME), Madrid, Spain, 23–29 September 2018; IEEE: New York, NY, USA, 2018; pp. 564–568. [CrossRef]
19. Belzunegui-Eraso, A.; Erro-Garcés, A. Teleworking in the Context of the COVID-19 Crisis. *Sustainability* **2020**, *12*, 3662. [CrossRef]
20. Saraiva, C.; Mamede, H.S.; Silveira, M.C.; Nunes, M. Transforming physical enterprise into a remote organization: Transformation impact: Digital tools, processes and people. In Proceedings of the 16th Iberian Conference on Information Systems and Technologies (CISTI), Chaves, Portugal, 23–26 June 2021; IEEE: New York, NY, USA, 2021; pp. 1–5. [CrossRef]
21. Alghamdi, A. Cybersecurity threats to Healthcare Sectors during COVID-19. In Proceedings of the 2nd International Conference on Computing and Information Technology (ICCIT), Tabuk, Saudi Arabia, 25–27 January 2022; IEEE: New York, NY, USA, 2022; pp. 87–92. [CrossRef]
22. Zabardast, E.; Gonzalez-Huerta, J.; Palma, F. The Impact of Forced Working-From-Home on Code Technical Debt: An Industrial Case Study. In Proceedings of the 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Maspalomas, Spain, 31 August–2 September 2022; pp. 298–305.
23. Yin, R.K. *Case Study Research: Design and Methods*, 6th ed.; Sage Publications: Thousand Oaks, CA, USA, 2018.
24. Scrum Guide. Available online: https://scrumguides.org/download.html (accessed on 24 September 2022).
25. Holvitie, J.; Leppänen, V.; Hyrynsalmi, S. Technical Debt and the Effect of Agile Software Development Practices on It—An Industry Practitioner Survey. In Proceedings of the Sixth International Workshop on Managing Technical Debt, Victoria, BC, Canada, 30 September 2014; IEEE: New York, NY, USA, 2014; pp. 35–42. [CrossRef]
26. Eades, K. *The New Solutions Selling: The Revolutionary Sales Process That Is Changing the Way People Sell*, 2nd Revised ed.; McGraw-Hill: New York, NY, USA, 2003.

27. Google Jamboard. Available online: https://workspace.google.com/products/jamboard/ (accessed on 28 September 2022).
28. MURAL Documentation. Available online: https://www.mural.co/ (accessed on 28 September 2022).
29. Atlassian Jira Software. Available online: https://www.atlassian.com/software/jira (accessed on 22 September 2022).
30. A brief Overview of Planning Poker. Available online: https://atlassian.com/blog/platform/a-brief-overview-of-planning-poker (accessed on 28 September 2022).
31. Ralph, P.; Tempero, E. Construct validity in software engineering research and software metrics. In Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, Christchurch, New Zealand, 28 June 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 13–23. [CrossRef]
32. Aha! Suite Overview. Available online: https://www.aha.io/suite-overview (accessed on 23 September 2022).