

# Tracking technical debt in agile low code developments

Renato Domingues<sup>1,2</sup>, Miguel Reis<sup>2</sup>, Miguel Araújo<sup>2</sup>, Marcelo Marinho<sup>1</sup>, Mário J. Silva<sup>2,3</sup>

<sup>1</sup>Departamento de Computação – Universidade Federal Rural de Pernambuco (UFRPE)  
Recife – PE – Brazil

<sup>2</sup>Truewind, Axians Portugal  
Recife – PE – Brazil  
Lisboa – Portugal

<sup>3</sup>INESC-ID, IST, Universidade de Lisboa  
Lisboa – Portugal

renato.domingues@ufrpe.br, miguel.reis@axians.com,

miguel.araujo@axians.com, marcelo.marinho@ufrpe.br, mjs@inesc-id.pt

**Abstract.** *We present a new practice for tracking technical debt in agile low code developments using new metrics obtained from the development environment. The automated collection of some of the metrics has significantly reduced the effort devoted to collecting project meta-data and makes it possible to run frequent self-assessments and internal audits with little effort. We have obtained evidence that more frequent audits during development help to identify and reduce technical debt, thus improving the quality of delivered software. In addition, we witness a general increase in the quality score of projects developed under the new practice.*

## 1. Introduction

Software organizations aim to deliver high-quality products and services, on time and at the lowest cost. However, in practice, design trade-offs and unplanned events frequently force developers to take shortcuts when developing their code. These time-saving shortcuts, which solve near-term problems but do not account for long-term implications, are one of the causes of what is known as technical debt [Cunningham 1992].

Low code is becoming prevalent for fast development of enterprise application software. However, higher productivity and reduced costs of low code development could be achieved at the expense of quality management practices established for software engineering. We believe that the quality of reliable complex enterprise software solutions based on agile low code platforms does not need to be sacrificed for speed. In fact, low code development methods, when supported by an appropriate methodology relying on automated metrics collection, should enable software engineers to achieve higher productivity while delivering high-quality complex enterprise solutions.

Some low code platforms help manage tech debt. AI Mentor Studio, from the OutSystems platform for enterprise software development, provides tech debt management support<sup>1</sup>. However, managing technical debt growth is hard to achieve, since ke-

---

<sup>1</sup>[https://success.outsystems.com/documentation/11/managing\\_the\\_applications\\_lifecycle/manage\\_technical\\_debt/get\\_an\\_overview\\_of\\_the\\_overall\\_technical\\_debt/](https://success.outsystems.com/documentation/11/managing_the_applications_lifecycle/manage_technical_debt/get_an_overview_of_the_overall_technical_debt/)

eping it under control depends on teams behavior. Hence, establishing practices and software audits remains necessary to make the technical debt visible beyond the scope of individual project teams.

To address these challenges, we developed a lightweight, largely automated software auditing process tailored to our practice. This new process complements the data provided by the Outsystems platform with self-reported assessments and automated metrics.

The main objective of this paper is to introduce *QSP - Quality Solutions Program*, our approach for tackling technical debt in low code-based software solutions. QSP includes a tool that collects data from developers and exposes the internal audit results to end-users quickly and visually. QSP has been developed and used for the last two years at Truewind<sup>2</sup>, an international company focused on agile transformation and agile low code development.

With QSP, we reduce the effort and time costs of carrying out project audits and allow them to be more frequent. Using the plots and charts generated in QSP, we have observed a general improvement in tech-debt scores over time, showcasing the success of our approach to managing tech debt with minimal additional effort (cost).

## 2. Background

### 2.1. Technical Debt

Cunningham introduced technical debt stating that *Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite... The danger occurs when the debt is not repaid. Every minute spent on not quite right code counts as interest on that debt* [Cunningham 1992].

McConnell later defined technical debt as *delayed technical work that is incurred when technical shortcuts are taken, usually in pursuit of calendar-driven software schedules* [McConnell 2008].

In 2016, at the Dagstuhl Seminar 16162, academic and industrial experts proposed the most widely accepted definition of technical debt in software systems as *a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability* [Avgeriou et al. 2016].

Kruchten classified the entries in a backlog according to their Visibility and value [Kruchten et al. 2012]: *Green/Visible* features are visible and have positive value; *Red/Visible* defects are visible and have negative value; *Yellow/Architectural* features are invisible and have positive value; and *Black/Technical* debt is invisible and has negative value. When determining the work of an iteration, the Product Manager wants more green items to be worked on, Customer Support wants more red items, and Architects the yellow ones, but no one wants to deal with the black items.

Li et al. carried out a systematic review to identify and classify the types of tech debt and tech debt management. Ten types of tech debt were identified: Requirements,

---

<sup>2</sup><https://www.truewindglobal.com/>

Architectural, Design, Code, Test, Build, Documentation, Infrastructure, Versioning and Defect, as well as eight types of management: Identification, Measurement, Prioritization, Prevention, Monitoring, Repayment, Representation and Communication [Li et al. 2015].

## **2.2. Software Audits**

Technical debt can be tackled as a quality issue. Asserting the quality of software is a complex task requiring multiple instruments, and one of the most used is audits [Helgeson 2009]. A software audit may be conducted at three levels: product, process, and system. The product audit is the lowest level and focuses on the final product. At the intermediate level, the process audit checks the methodology used in the product and its behavior. This audit yields the most visible results and is thus the most common. Lastly, the system audit (or quality audit) is the most complete, complex, and laborious. A system audit can cover the organization as a whole. Additionally, the development team (internal) can conduct these audit types or require an independent expert (external audits).

Audit processes are often only carried out at the end of a project, since they can be very expensive due to hiring third-party services and consume time. Depending on depth and complexity, an audit could take hours, days, or weeks. Excessive time (and cost) in an agile environment can be prohibitive. On the other hand, when an audit is carried out at the end stage of software development, it is often not possible to invest on substantial code refactoring, whether due to meeting delivery deadlines or lack of budget, which could remove identified technical debts.

## **2.3. Metrics and Audits for Agile Project Management**

Metrics to define the success of an agile project include customer satisfaction, business value delivered, velocity, budget against actual cost, defects into production, cycle time, defects resolution, customer retention, estimation accuracy, test pass/fail overtime, revenue/sales impact, product utilization, and scope change in a release [Mkoba and Marnewick 2020]. Software quality can improve by automating the collection of metrics to measure agile teams' quality as it makes it possible to provide sprint-by-sprint performance feedback [Guerrero et al. 2019]. In a case study of 120 students divided into 20 teams carrying out a project in four two-week iterations, students worked without feedback in the first two iterations. In the two final iterations, they gained access to a software tool providing feedback, and their quality metrics showed a positive increase.

To identify problems earlier and facilitate their correction in agile development projects, it is important to know which metrics have to be audited, and the interaction between the auditor and the team so that the auditor understands what is being audited and the need to conduct audits at each project cycle [Truong 2020].

Joshi identifies three key components for an agile audit [Joshi 2021]: Backlog (collection of scoped work like audit plan), Sprints (scoped items divided into sprints time period), and Scrums (short and concise meetings to evaluate the work done and identify bottlenecks). He also defines five levels of maturity for an audit:

1. Minimal stakeholder engagement.
2. Improving team cooperation and planning methods.
3. Integration with agile practices and stakeholder collaboration.

4. Well-defined metrics are measured.
5. Audit optimization and agile audit tools are applied.

An agile audit has three main phases [KPM 2020]:

**Planing and preliminary research:** Determine what will be audited, what aspects are relevant to the audit, and define some metrics such as Definition of Ready and Definition of Done.

**Audit execution:** Defines how the sprint planning will be carried out, how the workload will be distributed, and based on the results of the previous sprint to schedule the next audit.

**Completion and evaluation:** Once the activities have provided enough information, carry out a product audit shared with all members involved; this audit contains objective, scope, conclusion, recommendations, and action plans.

## 2.4. Low code software development tools

For Waszkowski one of the advantages of Low code platforms is the support to visual programming tools with graphical user interfaces for application design, instead of hard-coded programming techniques [Waszkowski 2019]. Meanwhile, Sahay et al. state that by using low code platforms, developers can focus on the business logic rather than dealing with setting up computing infrastructure, managing data integrity across different environments, and ensuring non-functional aspects [Sahay et al. 2020].

The Outsystems low code development platform offers several tools that help manage software quality. Among these tools, we have the Quality Apps Program<sup>3</sup>, which is used to monitor the quality of low code application developments made on the platform, and AI Mentor Studio<sup>4</sup>, for static code analysis.

## 3. Auditing Low code developments with QSP

In our earlier practice, internal audits were established, but they were not frequent due to the effort of time and personnel needed to conduct them. Audits were normally postponed until the late stages or even after the last iteration before releasing a software project. So, it was also common that some of the problems encountered could not be properly resolved, with a negative impact on the perceived quality of Truewind's agile low code process for complex systems development. Some customers were reluctant to pay for what they perceived as expensive additional work.

Our new methodology for auditing agile low code projects relies on fast and short self-assessments, inspired by Kim et al.'s approach of increasing the frequency of internal audits in the agile development process [Kim et al. 2013]. In addition, new audit metrics are also proposed to better represent the concept of agile, including productivity, team performance, and correct selection of stories.

Our Quality Solutions Program (QSP) is our internal agile low code development process for complex enterprise solutions. QSP presently supports three practices:

---

<sup>3</sup><https://www.outsystems.com/blog/posts/launching-the-quality-apps-program/>

<sup>4</sup><https://www.outsystems.com/product-updates/ai-mentor-studio/>

**Engagement Practice:** monitors the use of agile methodologies and practices such as the use of Key Performance Indicators (KPIs), and the preparation and development of the sprints. The stakeholders conducting this practice are the Engagement Manager and the project's Auditor.

**QA Practice:** manages quality of the development process and mainly takes care of testing. The Stakeholders for this practice are the QA Lead and the Auditor.

**Engineering Practice:** The engineering part is responsible for the code, in this part, the issues present and the code developed are checked to verify the existence of possible problems and/or bad practices. The Stakeholders responsible for this practice are the Tech Lead and the Auditor.

### 3.1. Self-assessments in QSP

To guide a QSP self-assessment, we provide a checklist that teams use to evaluate their adherence to QSP best practices. The QSP self-assessment checklist provides a systematic approach to ensuring that crucial quality benchmarks are met throughout the project. Teams can effectively gauge their progress by answering key questions, such as:

- Does the project have a demo?
- Does the project have security tests planned?
- Is the team running sprint reviews?
- Is the team running sprint retrospectives?
- Is the team keeping optimal test coverage rates?

One of OutSystems AI Mentor Studio functions is to identify tech debts present in the code during development, so we can collect data from it and use it as one of the self-assessment inputs metrics to be able to monitor the amount of tech debts present throughout the process.

Most self-assessment questions in audits are framed as Boolean to keep the time required to complete an audit as short as possible. An answer score of 1/0 on a Boolean question means that the requirement was considered met/not met. For non-Boolean questions, answers with half scores (0.5) are also possible. The final score is calculated by averaging the scores of all provided answers:

$$FinalScore = \frac{\sum answer}{\sum Questions} \quad (1)$$

The Quality Apps Program of OutSystems awards a high-quality seal to those projects that obtain a minimum score of 85% in their assessment. In QSP we adopted the same threshold of 85% to award high-quality recognition to our teams' development teams.

All self-assessment scores are saved in the database and compared throughout the project, so a team's progress can be tracked throughout the entire process.

A key aspect of these self-assessments is that audit responses are visible to all company employees. Enabling this type of peer monitoring helps keep the reliability of responses high and helps promote and disseminate team practices that obtain higher QSP assessment scores.

### 3.2. Audits in QSP

A QSP audit lasts a maximum of two hours. If some questions remain unanswered within that time, the audit stops and only the answered questions are considered. To ensure that the most relevant questions are answered more frequently, all questions have a priority value and are asked by the order of priority. Priorities are calculated based on the time it takes to answer a question, its category, and/or internal rules.

Many development services at Truewind do not start with a clean slate. Projects may take on earlier developments by other teams or may even have been initially developed by other organizations. Such projects, in turn, may present different forms of technical debt that would influence both the team's work and the audit scores of the project. We call these debts *legacy items*. As the scores are public within the company, a project team could be unfairly perceived negatively by peers unaware of the project's starting point.

With this in mind, we give two scores to every QSP audit:

**Project score:** Sum of the self-assessment and the audit results considering the legacy issues.

**Team score:** Sum of the self-assessment and the audit results excluding the legacy issues.

All data submitted to QSP by a team and obtained self-assessment results are stored and compared against their previous submission. As a result, teams receive real-time feedback on how their tech debt is evolving. With this information in hand, teams can analyze and better understand what they did right and what needs to be reviewed and improved.

## 4. Metrics collection automation

With the QSP, engagement managers can run a self-assessment of a project, which provides results obtained from automatically collected data. Within a self-assessment, there are six categories of metrics: Lean, Reliability, Methodology, Delivery, Dev-Ops, and Strategy – Value. Each of these metrics may or may not weigh on the final result of the self-assessment. Table 1 shows some of these metrics (those that do not weigh on the final score of the self-assessment are marked with \* at the end of their title).

### 4.1. Information extraction

We initially established the company's Jira<sup>5</sup> issue tracking system as the primary data source. However, after an initial analysis of the historical data, we realized that the lack of common guidelines for filling out Jira forms was destroying the usefulness of the extracted data. Some of the problems encountered are:

- Lack of filling in relevant text fields, such as description and summary, as well as numeric fields, such as time spent and estimated time;
- Multiple interpretations for the meaning of a given field;
- Incorrect use of the issue type when creating an issue.

We then defined new practices for structuring and standardizing the meta-data in Jira projects, enabling automatic extraction of information. The newly adopted practices include, among others:

---

<sup>5</sup><https://www.atlassian.com/software/jira>

<b>Audit Category</b>	<b>Metric</b>
Lean	Project FTE vs User Stories and Cycle time*
Reliability	Reported incidents since last assessment
Reliability	Number of non-trivial bugs in Quality*
Methodology	Definition of Ready
Methodology	Definition of Done
Methodology	Sprint planning done
Methodology	Sprint review done
Methodology	Sprint retrospective done
Delivery	Deploy & Rollback plan available
Delivery	Sprint duration according to the plan
Delivery	Backlog management (prioritized, refined for sprint n and n + 1 and high-level for the remaining)
Delivery	% of Story items done
Delivery	Estimation accuracy (Delivery - fixed price) & Velocity (AMS)
Dev-ops	Time taken from code being deployed into QA to Production
Dev-ops	Time to restore
Dev-ops	Change fail %
Strategy - Value	Identified Business and Operational Outcomes
Strategy - Value	Available Stakeholder map analysis

**Table 1. EM's self-assessment metrics in QSP's Audit categories**

**Project creation centralization:** New projects must be created through a form requesting the necessary information. Previously, we let our project leaders define their projects however they found most suited, thus creating a hard-to-manage variety of project patterns.

**Boards template:** Project boards must be created using a pre-defined template for each project type. Previously, leaders could also create several attributes in Jira and name them arbitrarily, making it hard to find what each one represented.

**Issue types standardization:** Allowed issue types have been limited to a pre-defined set. We found that the semantics of each issue type among engagement managers were very subjective, so there was no standard defining what each project metadata attribute represented.

**Standardization of the project title:** The name of projects in the company's Jira repository must follow a defined pattern coding its project type. As the predefined project types in Jira do not comply with our internally defined types, we standardized project naming rules so that it is possible to identify a project's type by scanning a project's name on Jira.

**Burn remaining estimate:** As soon as a ticket enters the Testing phase, any remaining estimates should automatically be burned (dropped to zero) so that the total remaining effort is always up to date. One of the problems identified during the data analysis was that several issues were transferred to the "Done" state but the estimate was not correctly filled, causing the sprint remaining effort on our Jira repository to show incorrect values.

After implementing these new practices in Jira, we waited approximately six months for new data to be generated. With the new data in hand, we observed six projects to assess to what extent the new practices achieved the desired objectives. We concluded that the data of all six tested Jira projects was ready for automation.

## 4.2. QSP metrics collection

Many metrics of QSP self-assessments cannot be obtained from the data in Jira, just as some information that the data contains is not always clear to the Engagement Manager when performing a self-assessment. Therefore, it was necessary to select which metrics could be extracted automatically for QSP self-assessments. We arrived at three categories of metrics:

### 1) User Story Definition metrics:

**INVEST criteria:** User Stories should be focused on features – not tasks, and written in business language. Doing so will enable business people to understand and prioritize the User Stories.

**User stories sizing:** A User story is the smallest unit of work that needs to be done. Taking the INVEST acronym for Story definition, the S (Small) is the most important, and we should guide our teams to cut their work into less than 2 days chunks wherever possible.

**User stories average size:** Purely informative data that compares the average number of hours in user stories for this sprint against the company average.

**Acceptance criteria:** It clearly defines the scope, desired outcomes, and testing criteria for pieces of functionality that the delivery team is working on.

**Definition of Ready (DoR) & Definition of Done (DoD):** Reinforce transparency, assure Built-In Quality, and set the right expectations for the work items to be planned, developed, and completed.

**Work item type:** Purely informative data that shows the number of issues of each type created in this sprint.

### 2) Board Management metrics:

**Reported defects:** It is purely informative information that shows the quantity of each type of defect severity created in this sprint.

**Backlog prioritization or MoSCoW:** Backlog is prioritized in a way that helps build consensus, establish a unified direction, and gain a broader perspective.

**Defects management:** Verify if the defects are being handled as any other Backlog item (sized, classified).

**UX/UI management:** UX/UI Backlog should be handled as any other activity: Planned, developed, and reviewed / validated. As such, it is checked whether there are issues of this type in this sprint.

**Sprint goal defined:** While it's easy to gather a bunch of Backlog Items to work on in a Sprint, it's a little harder (but much more valuable) to have a set of Backlog Items that fit together and in this way, provide more business value.

### 3) Quality & Control metrics:

**Sprint Definition:** Compare the difference between the items that were planned at the beginning of the sprint and how many of them were completed.



**Capacity Tracker:** Calculates the capacity in hours of this team for this sprint, based on the number of sprint days, developers, team members' off days, and a constant to represent the sprint stabilization period, then compares it with the estimated time for the issues planned for this sprint.

**Estimation Accuracy:** Compares the number of worklog hours logged within the sprint period against the capacity value of that sprint. If a defined minimum threshold is not met, it is assumed that the project team did not log the correct number of hours for the sprint and will receive the maximum penalty for this metric (0%). If above the threshold the Estimation accuracy is computed by subtracting from 100% the absolute value of the observed Relative Accuracy Error of the estimation:

$$RelAccuracyError = \frac{(\sum planned - \sum logged)}{\sum planned} \quad (2)$$

$$EstimationAccuracy = 100\% - |RelAccuracyError| \quad (3)$$

**Issues Estimation:** Compares the number of issues that were planned for the sprint and defects with severity *Major* or *Critical* that have effort estimation with the total number of issues of these types.

**# of bugs in Quality:** That is a two-step metric:

1. Get the list of Stories belonging to this project and compare it with the list of issues that have test coverage. If the result of this comparison is larger than the minimum threshold defined, then this sprint has a minimum test coverage and can go to the next step, otherwise, it means that not enough tests were planned for this project and it receives the maximum penalty for this metric.
2. We compare the number of defects that are of *Critical*, *Major*, and *Minor* severity with the number of Increments (Stories and Improvements).

### 4.3. QSP Audit scores

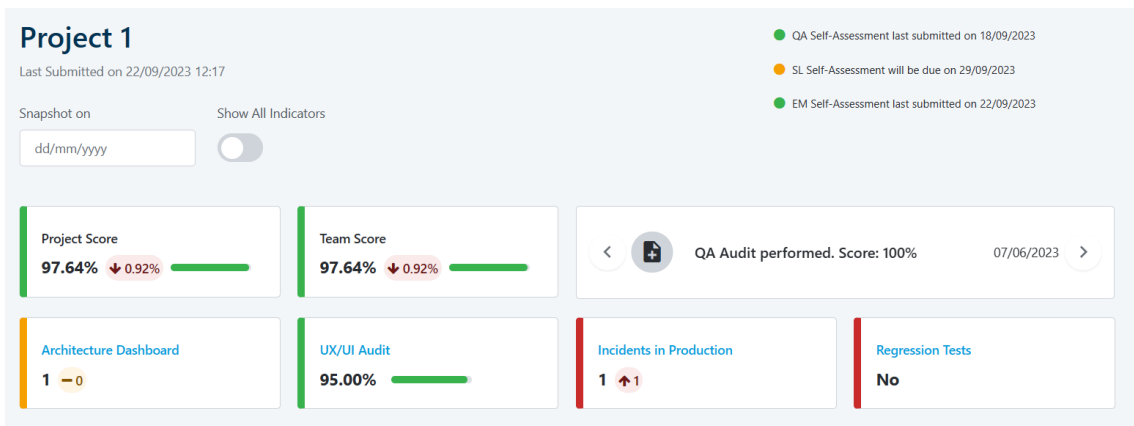
We assign four score grades to each metric: *Excellent*, *Good*, *To improve*, and *Not met*, which corresponds to obtaining 100% of the metric's full score, 2/3, 1/3 and 0%, respectively. Furthermore, we have also developed the metrics so that they are always categorized into one of the four available grades or are in a range of 0 to 100%. By default, we consider *Excellent* those with 85% or above, *Good* from 84 to 75%, *To improve* 74 to 50%, and below that is *Not met*. Similar to the final self-assessment score, we can calculate the final audit score using Equation 4:

$$AuditScore = \sum_{i=1}^f Metric_i * weight_i \quad (4)$$

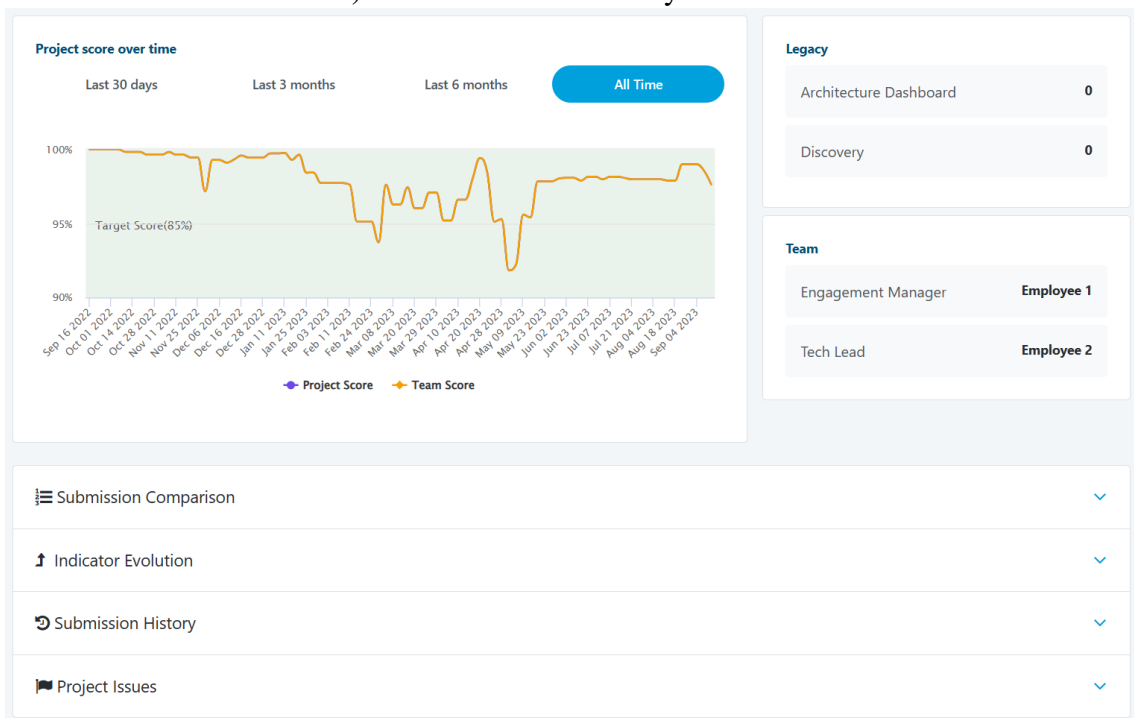
### 4.4. QSP Dashboards

To give a perspective on what data is provided to developer teams, we will be showing throughout this section some of the QSP dashboards of audited projects. While the data are real, all project and employee names given below, as well as some acronyms are replaced with pseudonyms to preserve confidentiality.

Figure 1 shows an example of what information is presented on a project dashboard. Figure 1a, displays the self-assessment delivery deadlines. If a deadline is being reached/has passed and the result has yet to be delivered/has already passed, the orange/red color is used to highlight it. The central part of the panel shows the Project Score and the Team Score of the project in this assessment, together with a comparison with the past score (both scores had a drop of 0.92%). As the grade is above 85%, this card is colored green, otherwise, it would be colored red. Below we can see some project indicators, prioritizing those that did not pass the tests, thus making it clearer what needs to be improved. In the displayed project, two indicators are colored red (*not passed*), one yellow (*To be improved*), and one green. Figure 1b displays the complete history of scores of a project.



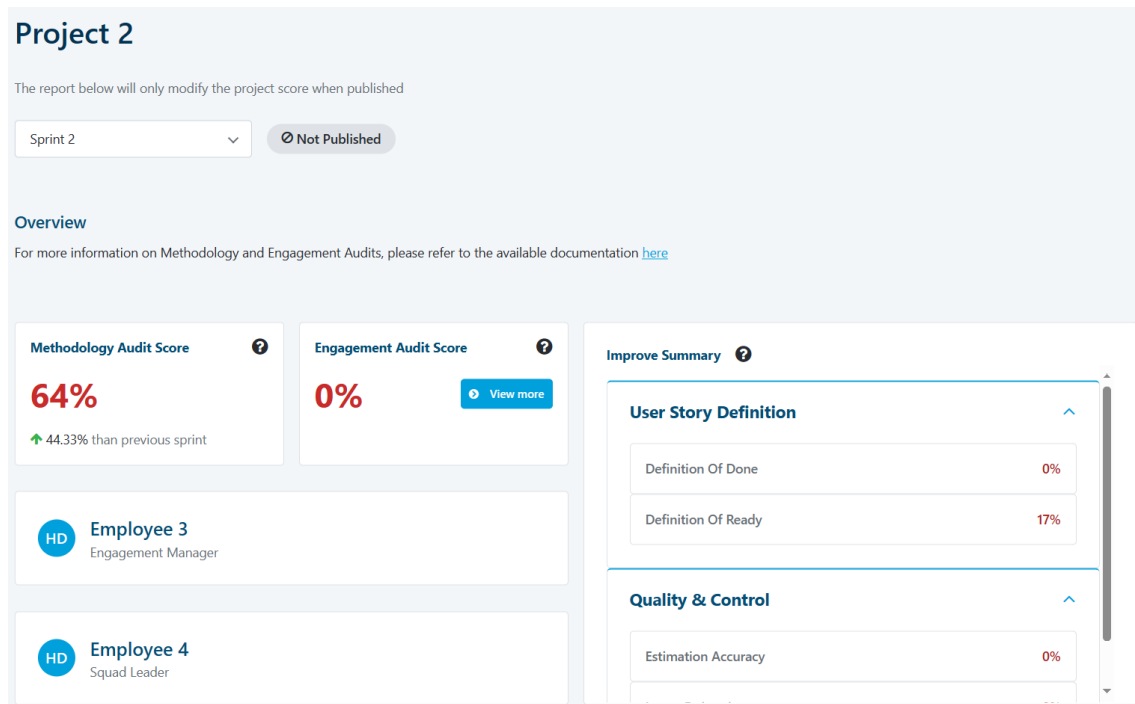
a) self-assessment delivery deadlines



b) QSP scores timeline; as the project does not have legacy items, the Project and Team Scores are always the same

**Figure 1. QSP-audited project summary.**

Figure 2 shows the Methodology audit score and metrics to be improved and presented to Engagement Managers.



**Figure 2. Engagement Manager details of a QSP audited project. The audit score is shown on the left and the metrics that need to be improved on the right.**

## 5. Results

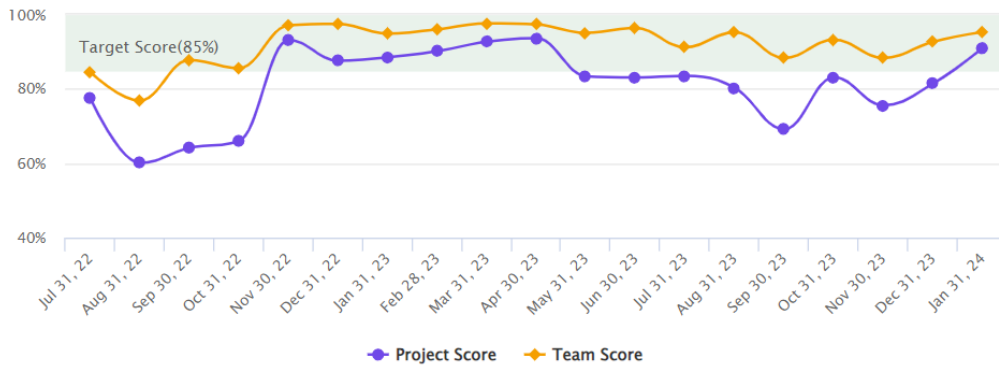
At the time of writing this article, the QSP has been used in more than 30 projects and currently has 8 others active, by teams from different sizes, from 3 to 10 members per team.

Figure 3a) shows the average score of the projects in the metrics most directly related to technical debts, such as the number of debts and discovery, which represent the number of simple technical debts found in the project and an analysis of the application architecture in search of debts, respectively. Initially, in the first four months, we had the lowest team scores, but since the third month, all team scores are already above the desired 85%. Meanwhile, project scores are most often below expectations, this may be because the project was started by the team at that time but is a project with many legacy items. This shows the importance of using the two scores to account for legacy items, as this could negatively impact team morale and mislead managers. Figure 3b), shows the average of the scores of all internal audits carried out by all QSP projects, such as EM Audits, Quality Audits, and UX/UI audits, when applicable. Initially, the projects were not performing so well, but since December 2022 they have reached values above the target score. The quality part took a little longer to be developed and that is why we only have data from 2023, but as shown in Figure 3c), despite having only reached the target in June, from that month onwards, all scores reached the target value.

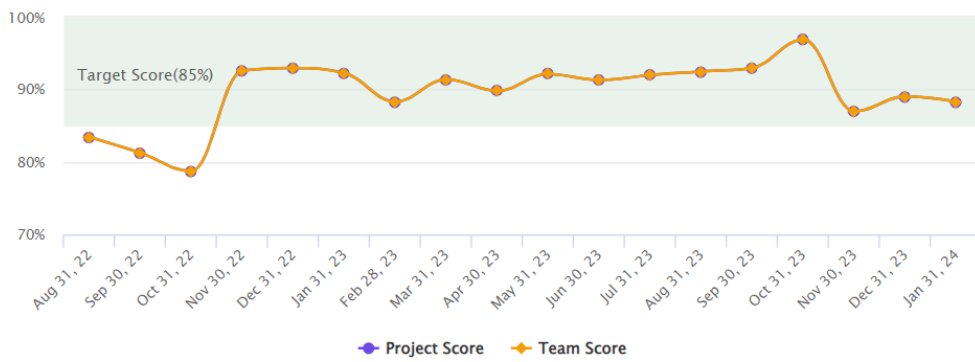
Finally, Figure 3d) shows the mean of the final score of all the projects and teams. Although once again the first four months had the lowest results, since then, we have

always achieved the target score. The Project score on the other hand, despite reaching the 85% mark in every month except the first four and December of 2023, likely because the teams inherited a new project with high technical debt, still reached the target score in all other months.

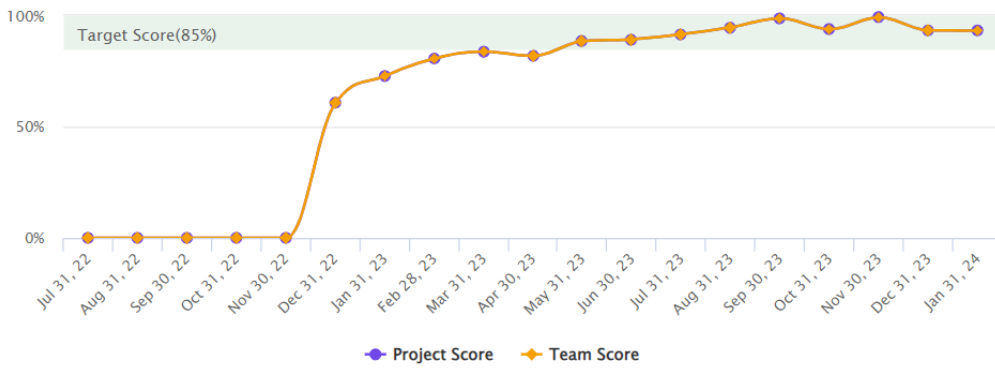
When we take the grades for the year 2022 we obtain an average score of 88.22% for tech debt, 85.80% for audits and 87.64% for the final score, in 2023 these metrics rose to 93.88 %, 91.52% and 91.96% respectively. The quality metric cannot be calculated in this way given that the 2022 sample is only one month, however if we take the first 6 months and compare them with the following 6 months we obtain 78.12% and 94.62%. With this we can see that the score of projects in year 2023 were better than those in 2022 in all aspects, where the tech debt score increased by 6.42%, audit score 6.67% and the final score in 4.93%. The sum of all these indicated results, together with other non-measurable results such as greater interest among employees in talking about how to improve, lead us to consider the QSP as a successful project in our environment.



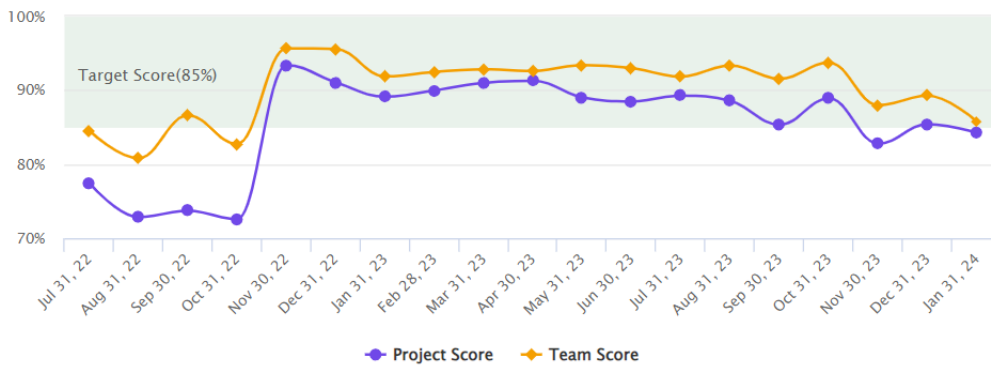
a) Technical debt metrics score



b) Project audit score



c) Quality metrics score



d) Final score

Figure 3. Overall performance of projects in the QSP

## 6. Conclusion

Our new QSP is supported by an app that easily and graphically presents the technical debts of a project. The app includes an automatic metrics collection module that enables fast and frequent self-assessments. We presented various metrics used both in internal audits and self-assessment questionnaires, along with the new practices established across software teams in our organization.

A notorious outcome of introducing QSP is that it helps trigger technical conversations among development teams. Previously it was uncommon to observe co-workers talking about their difficulties. Co-workers often did not even know they could obtain help until it was too late. Through QSP they are now able to find and contact those who had succeeded in overcoming similar hardships.

It is important to also highlight some of the limitations and points for improvement identified throughout the development of QSP. The first major current limitation is the fact that we are heavily dependent on the API provided by Jira. The second is also related to a recurring situation in projects developed for customers who required that all the project development artifacts be maintained in their Jira instance, or in some other repository that they use, and do not provide open access to the API by QSP. Therefore, we do not have direct access to project data and are unable to perform automatic extraction and auditing. This problem, however, could be mitigated by informing customers of the benefits of using QSP to track tech debt and offering them access to the collected data, in exchange for opening access to their repository for collecting project data. A third problem is somewhat more complex and does not directly involve our code. In our self-assessments, lies in the answers are not encouraged nor rewarded. However, if in our working culture lies becomes pervasive and unreported, auditors would need to constantly check the veracity of obtained answers. This would create an even greater workload on auditors than if no self-assessment had been made, leading to a reduction of our auditing capacity and consequently removing the advantage of our approach.

There are some points where data collection automation needs improvement. For example, for String fields the automation only checks if they are not empty, no content validation is performed. Presently, there is an optional manual intermediate step in the audit, where an auditor can manually look at the values of each field and change their scores if he deems it necessary.

## Referências

- (2020). Agile internal audit: White paper on working agile within internal audit functions (part ii: Concrete guidance for the set-up of the agile internal audit function and the execution of agile audits). KPMG, <https://assets.kpmg.com/content/dam/kpmg/nl/pdf/2020/sectoren/agile-internal-audit-2.pdf>. Accessed: 2023-09-23.
- Avgeriou, P., Kruchten, P., Ozkaya, I., and Seaman, C. (2016). Managing technical debt in software engineering (dagstuhl seminar 16162). In *Dagstuhl reports*, volume 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
- Cunningham, W. (1992). The wycash portfolio management system. *ACM Sigplan Oops Messenger*, 4(2):29–30.

- Guerrero, A., Fresno, R., Ju, A., Fox, A., Fernandez, P., Muller, C., and Ruiz-Cortés, A. (2019). Eagle: A team practices audit framework for agile software development. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1139–1143.
- Helgeson, J. W. (2009). *The software audit guide*. Quality Press.
- Joshi, P. L. (2021). A review of agile internal auditing: Retrospective and prospective. *International Journal of Smart Business and Technology*, 9(2):13–32.
- Kim, D. H., Kim, D. S., Koh, C., and Kim, H. W. (2013). An information system audit model for project quality improvement by the agile methodology. *International Journal of Information and Education Technology*, 3(3):295.
- Kruchten, P., Nord, R. L., and Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21.
- Li, Z., Avgeriou, P., and Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101:193–220.
- McConnell, S. (2008). Managing technical debt. Construx Software white paper: [https://www.construx.com/wp-content/uploads/2019/02/CxWhitePaper\\_TechnicalDebt.pdf](https://www.construx.com/wp-content/uploads/2019/02/CxWhitePaper_TechnicalDebt.pdf). Accessed: 2023-09-27.
- Mkoba, E. and Marnewick, C. (2020). Conceptual framework for auditing agile projects. *IEEE Access*, 8:126460–126476.
- Sahay, A., Indamutsa, A., Di Ruscio, D., and Pierantonio, A. (2020). Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euro-micro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 171–178. IEEE.
- Truong, L. (2020). Agile auditing: More value, less resources. *Edpacs*, 62(1):4–7.
- Waszkowski, R. (2019). Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine*, 52(10):376–381.