UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA

Renato Cavalcanti Domingues da Silva Filho

Métricas e Qualidade em Ambientes Ágeis Low-Code: Rastreamento de dívida técnica, avaliação contínua e apoio à tomada de decisão

Recife
2025

Renato Cavalcanti Domingues da Silva Filho

Métricas e Qualidade em Ambientes Ágeis Low-Code: Rastreamento de dívida técnica, avaliação contínua e apoio à tomada de decisão

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de mestre(a).

Orientador(a): Marcelo Luiz Monteiro Marinho

Coorientador(a): Mário Jorge Costa Gaspar da Silva

Recife

2025

Dados Internacionais de Catalogação na Publicação
Sistema Integrado de Bibliotecas da UFRPE
Bibliotecário(a): Suely Manzi – CRB-4 809

S586m Silva Filho, Renato Cavalcanti Domingues da.
Métricas e qualidade em ambientes ágeis low-code: rastreamento de dívida técnica, avaliação contínua e apoio à tomada de decisão / Renato Cavalcanti Domingues da Silva Filho. – Recife, 2025.
65 f.; il.

Orientador(a): Marcelo Luiz Monteiro Marinho.
Co-orientador(a): Mário Jorge Costa Gaspar da Silva.

Dissertação (Mestrado) – Universidade Federal Rural de Pernambuco, Programa de Pós-Graduação em Informática Aplicada, Recife, BR-PE, 2025.

Inclui referências e apêndice(s).

1. Administração da qualidade . 2. Desenvolvimento ágil de software. 3. Software - Desenvolvimento. 4. Controle de qualidade 5. Informática - Estudo e ensino. I. Marinho, Marcelo Luiz Monteiro, orient. II. Silva, Mário Jorge Costa Gaspar da, coorient. III. Título

CDD 004

Métricas e Qualidade em Ambientes Ágeis Low-Code: Rastreamento de dívida técnica, avaliação contínua e apoio à tomada de decisão

Dissertação apresentada ao Programa de Pós-Graduação em Informática Aplicada da Universidade Federal Rural de Pernambuco, como requisito parcial para obtenção do título de Mestre(a).

Aprovado(a) em: 25 / 06 / 2025.

BANCA EXAMINADORA

_____

Prof. Dr. Marcelo Luiz Monteiro Marinho (Orientador)

Programa de Pós-Graduação em Informática Aplicada

Universidade Federal Rural de Pernambuco

_____

Prof. Dr. Lucas Albertins de Lima (Examinador Interno)

Programa de Pós-Graduação em Informática Aplicada

Universidade Federal Rural de Pernambuco

_____

Profa. Dra. Suzana Candido de Barros Sampaio (Examinador Externo)

Departamento de Computação

Universidade Federal Rural de Pernambuco

Dedico esta dissertação à minha família, aos amigos e a todos que, de alguma forma, fizeram parte dessa jornada.

AGRADECIMENTOS

Agradeço, em primeiro lugar, à minha família, pelo apoio constante, ao meu orientador o prof. Marcelo e ao meu coorientador o prof. Mario pela orientação, pelas conversas valiosas e pela confiança ao longo deste percurso. Seu apoio foi fundamental para a realização deste trabalho. Ao prof. Lucas por toda a ajuda para conseguir essa oportunidade. Aos colegas e amigos por estarem presentes nos momentos certos. Agradeço também às equipes envolvidas no projeto, às fontes de dados e à instituição que proporcionou a estrutura necessária para o desenvolvimento desta pesquisa.

Por fim, a todos que, de alguma forma, contribuíram para que este trabalho fosse possível, deixo aqui minha sincera gratidão.

It is truly meaningless. It doesn't matter what your hopes and dreams are, or if you got to lead a happy life, or if you got yourself shredded to pieces by rocks. It's all the same — men all eventually die.

Does that mean there is no meaning to life? That even being born at all is meaningless? And if so, is that the same for our fallen comrades? Were those soldiers' deaths also meaningless?

No, absolutely not!!

It's up to us to give meaning to those soldiers' deaths — those brave dead men, those poor dead men! The only ones who are capable of paying respect to them... are we, the living!

We, too, shall all die here today, and entrust those next in line with the same duty of giving meaning to us.

That is the only way for us to fight against this cruel world!!

Rage, my soldiers!!
Scream, my soldiers!!
Fight, my soldiers!! (ISAYAMA, 2016)

RESUMO

O uso de plataformas low-code tem ganhado destaque como alternativa para acelerar o desenvolvimento de software, especialmente em contextos que adotam metodologias ágeis. Esta dissertação, organizada no formato de coletânea, investiga como práticas baseadas em métricas podem apoiar o desenvolvimento ágil com low-code, promovendo a melhoria contínua dos projetos. O primeiro artigo apresenta a automatização de auditorias técnicas no programa Quality Solutions Program (QSP), permitindo a recolha eficiente de métricas e a realização de autoavaliações frequentes com baixo custo. O segundo artigo descreve a evolução do QSP para uma plataforma de monitoramento contínuo da qualidade, utilizada por múltiplas equipes ao longo de dois anos, com resultados positivos na melhoria dos indicadores técnicos. O terceiro artigo realiza uma revisão sistemática da literatura (RSL) sobre métricas em low-code, identificando tendências, lacunas e a falta de padronização nas abordagens atuais. Assim, esta dissertação propõe uma contribuição integrada ao campo do desenvolvimento ágil com plataformas low-code, composta por três eixos principais:(i) o estabelecimento de práticas estruturadas de monitoramento da dívida técnica, (ii) a criação de uma plataforma de avaliação contínua da qualidade; e (iii) a análise crítica do estado atual da literatura sobre o uso de métricas, com identificação de tendências e lacunas de pesquisa. Esses resultados contribuem tanto para o avanço da prática profissional, ao oferecer soluções aplicáveis a projetos reais, quanto para o campo acadêmico, ao mapear lacunas relevantes que ainda demandam investigação. A dissertação é, portanto, relevante para pesquisadores, equipes de desenvolvimento e organizações que buscam maior maturidade na gestão da qualidade em ambientes low-code.

Palavras-chave: Low-code, desenvolvimento ágil, métricas, qualidade.

ABSTRACT

The use of low-code platforms has gained prominence as an alternative to accelerate software development, especially in contexts that adopt agile methodologies. This dissertation, organized as a collection, investigates how metrics-based practices can support agile development with low-code, promoting continuous improvement of projects. The first article presents the automation of technical audits in the Quality Solutions Program (QSP), allowing the efficient collection of metrics and the performance of frequent self-assessments at low cost. The second article describes the evolution of the QSP into a continuous quality monitoring platform, used by multiple teams over two years, with positive results in improving technical indicators. The third article performs a systematic review of the literature (RSL) on low-code metrics, identifying trends, gaps and the lack of standardization in current approaches. This dissertation, therefore, proposes an integrated contribution to the field of agile development with low-code platforms, structured around three main axes: (i) the establishment of structured practices for monitoring technical debt; (ii) the creation of a platform for continuous quality evaluation; and (iii) a critical analysis of the current state of the literature on the use of metrics, highlighting trends and research gaps. These results contribute to both professional practices, by offering applicable solutions to real-world projects, and academic research, by identifying relevant gaps that still require investigation. The dissertation is thus relevant to researchers, development teams, and organizations seeking greater maturity in quality management within low-code environments.

Keywords: Low-code, agile development, metrics, quality.

LISTA DE ILUSTRAÇÕES

DISSERTAÇÃO

LISTA DE ABREVIATURAS E SIGLAS

QSP	Quality Solutions Program
RSL	Revisão Sistemática da Literatura
DSR	Design Science Research

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de software tem passado por transformações significativas impulsionadas pela necessidade de entregas mais rápidas, otimização de custos e aumento da satisfação do cliente, objetivos fortemente associados às práticas ágeis (SANKARAN; GANESAN, 2021). Segundo Rokis, Karlis e Kirikova (2023), plataformas de desenvolvimento low-code se consolidam como alternativas promissoras, permitindo a construção de soluções de forma mais rápida e com menor necessidade de especialistas em codificação tradicional. Essas plataformas viabilizam, além da agilidade, uma maior colaboração entre áreas de negócio e tecnologia, permitindo que até mesmo profissionais sem formação técnica participem do processo de desenvolvimento. Isso fortalece a capacidade de resposta às demandas do mercado, melhora o alinhamento entre TI e negócios e contribui para enfrentar a escassez de desenvolvedores qualificados (ROKIS, KARLIS & KIRIKOVA, MARITE., 2023).

Apesar dos benefícios evidentes, o uso de low-code em ambientes ágeis pode trazer desafios consideráveis no que se refere à garantia de qualidade e ao controle da dívida técnica. A ausência de práticas consolidadas para o monitoramento contínuo da qualidade, assim como a escassez de métricas adaptadas a esse contexto, são fatores que limitam a capacidade de tomada de decisão fundamentada e comprometem a evolução sustentável dos projetos. Essa lacuna foi evidenciada em (DOMINGUES; MONTE; MARINHO, 2025), que identificou a ausência de um conjunto padronizado de métricas e a predominância de estudos com baixo grau de generalização.

Diante desse cenário, torna-se essencial propor uma abordagem que combine métricas automatizadas com práticas ágeis e ferramentas de desenvolvimento low-code, com o objetivo de oferecer suporte prático e estratégico ao processo de desenvolvimento de software.

Partindo dessa necessidade, esta dissertação de mestrado propõe como pergunta de pesquisa central: C*omo aplicar métricas automatizadas para o rastreamento da dívida técnica e da qualidade em projetos ágeis desenvolvidos com plataformas low-code?*

Com o intuito de enfrentar essa problemática, foi desenvolvido a ferramenta Quality Solutions Program (QSP), uma iniciativa voltada para o rastreamento da dívida técnica em projetos de desenvolvimento ágil utilizando plataformas low-code

(DOMINGUES et al., 2024). O QSP propõe a integração de práticas de acompanhamento da qualidade ao ciclo de vida ágil, fundamentadas em métricas específicas coletadas automaticamente do ambiente de desenvolvimento.

Essa abordagem permitiu reduzir o esforço manual de auditorias, aumentar sua frequência e torná-las mais sistemáticas e menos intrusivas na rotina das equipes (DOMINGUES et al., 2024). A aplicação do QSP em diversos projetos reais demonstrou não apenas a diminuição dos índices de dívida técnica, mas também o crescimento consistente da qualidade das entregas.

À medida que os primeiros resultados evidenciaram a eficácia do QSP, tornou-se clara a necessidade de expandir seu escopo para além do rastreamento da dívida técnica, abrangendo o monitoramento contínuo de aspectos gerais da qualidade do desenvolvimento (DOMINGUES et al., 2024). Esta evolução resultou no desenvolvimento de uma plataforma mais abrangente, capaz de integrar dados automatizados sobre engenharia de software, desempenho de entregas, qualidade de produto e gestão da dívida técnica, utilizando dashboards interativos para apoiar a autoavaliação das equipes. Essa plataforma permitiu, por exemplo, o acompanhamento de métricas como a precisão nas estimativas de esforço, o número de incidentes em produção e a cobertura de testes, oferecendo às equipes indicadores objetivos para guiar melhorias contínuas.

Entretanto, ao realizar uma análise crítica do cenário acadêmico, percebeu-se que as práticas de mensuração em ambientes low-code ainda carecem de padronização e maturidade teórica (DOMINGUES; MONTE; MARINHO, 2025). Com o objetivo de entender melhor essas lacunas, foi conduzida uma Revisão Sistemática da Literatura (RSL) sobre métricas no contexto do desenvolvimento ágil em plataformas low-code.

Essa revisão revelou que há uma forte predominância do uso de métricas de desenvolvimento (como esforço e produtividade), enquanto métricas voltadas para usabilidade e experiência do usuário ainda são pouco exploradas (DOMINGUES; MONTE; MARINHO, 2025). Além disso, constatou-se a ausência de práticas padronizadas para coleta e análise de métricas, indicando que a disciplina ainda se encontra em fase inicial de amadurecimento.

Assim, esta dissertação propõe uma contribuição integrada ao campo do desenvolvimento ágil com plataformas low-code, composta por três eixos principais:(i) o estabelecimento de práticas estruturadas de monitoramento da dívida técnica, (ii) a criação de uma plataforma de avaliação contínua da qualidade; e (iii) a análise crítica

do estado atual da literatura sobre o uso de métricas, com identificação de tendências e lacunas de pesquisa.

1.1 OBJETIVOS

Objetivo geral

Investigar e propor uma abordagem metodológica baseada em métricas automatizadas para rastreamento de dívida técnica e avaliação de qualidade em ambientes low-code ágeis.

Objetivos específicos

1. Propor uma metodologia de auditoria automatizada para rastreamento da dívida técnica em projetos low-code ágeis, com base na coleta automatizada de métricas e na realização de autoavaliações e auditorias periódicas por meio do programa QSP (DOMINGUES et al., 2024);
2. Desenvolver e implementar uma plataforma de monitoramento de métricas de qualidade voltada ao contexto de desenvolvimento ágil com low-code (DOMINGUES et al., 2024);
3. Identificar e categorizar as métricas mais utilizadas no desenvolvimento ágil com plataformas low-code por meio de uma revisão sistemática da literatura, mapeando lacunas, padrões e oportunidades para padronização futura. (DOMINGUES; MONTE; MARINHO, 2025)

1.2 METODOLOGIA

Esta dissertação adota a abordagem de Design Science Research (DSR), conforme proposta por (HEVNER et al., 2004), como estrutura metodológica para a condução dos estudos apresentados. A DSR orienta o desenvolvimento e a avaliação de artefatos tecnológicos que visam resolver problemas relevantes da prática, ao mesmo tempo em que contribuem para o avanço do conhecimento científico. No contexto desta pesquisa, os três artigos que compõem esta dissertação se articulam com os principais ciclos e diretrizes da DSR, conforme descrito a seguir e ilustrado na Figura 1:

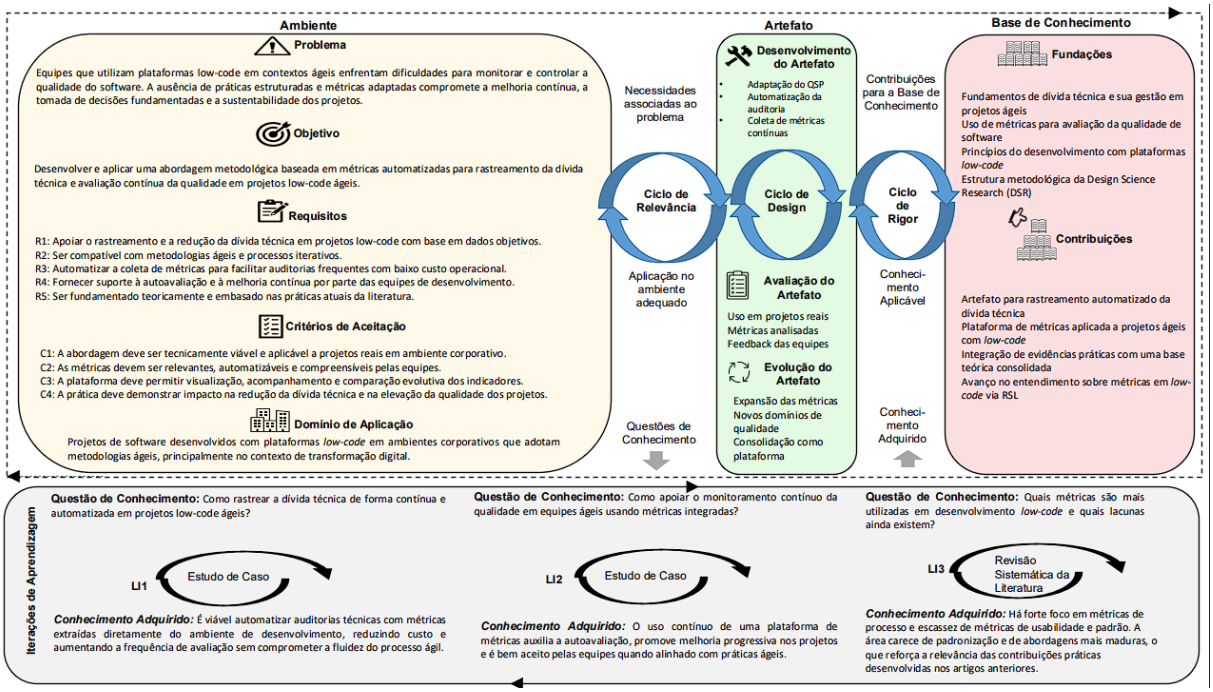


Figura 1 – Ciclo da DSR. Adaptado de (HEVNER et al., 2004).

Etapa 1 – Desenvolvimento do Artefato (Artigo 1)

A primeira etapa corresponde ao ciclo de Design & Development da DSR. Nela, foi desenvolvido um artefato tecnológico na forma de um sistema automatizado para rastreamento de dívida técnica, baseado na plataforma QSP já existente. Esse sistema utiliza métricas de qualidade de software coletadas de forma contínua e auditável para apoiar equipes ágeis em ambientes low-code. A solução proposta endereça um problema prático identificado na organização parceira e representa a materialização concreta de um artefato orientado à ação.

Etapa 2 – Avaliação do Artefato em Contexto Real (Artigo 2)

Na segunda etapa, correspondente ao ciclo de Evaluation, o artefato foi utilizado em seu ambiente real de aplicação ao longo de dois anos. A solução foi integrada ao processo de desenvolvimento ágil de diferentes equipes e monitorada por meio da coleta de métricas em tempo real. Essa fase permitiu avaliar a utilidade, viabilidade e impacto do sistema proposto, gerando oportunidades de evolução incremental a partir do uso contínuo em cenários reais de desenvolvimento.

Etapa 3 – Consolidação do Conhecimento e Rigor Científico (Artigo 3)

A terceira etapa está alinhada ao ciclo de Knowledge Base Contribution proposta por (HEVNER et al., 2004). Nela, foi conduzida uma RSL sobre o uso de métricas em

contextos de desenvolvimento low-code. A RSL forneceu embasamento teórico e científico para o problema investigado, identificando práticas recorrentes, lacunas e oportunidades de avanço. Além disso, serviu como base para a validação das escolhas metodológicas e das métricas utilizadas nos artefatos desenvolvidos.

O projeto segue as sete diretrizes da DSR conforme abaixo:

1. Design como Artefato: A principal entrega da pesquisa é um conjunto de artefatos inter-relacionados que apoiam a gestão da qualidade e da dívida técnica em ambientes low-code ágeis. Isso inclui um sistema automatizado de rastreamento baseado em métricas (Artigo 1), uma plataforma de análise contínua com suporte à autoavaliação das equipes (Artigo 2) e uma base teórica consolidada por meio de uma revisão sistemática da literatura sobre métricas em desenvolvimento low-code (Artigo 3).

2. Relevância do Problema: A gestão da dívida técnica e a mensuração da qualidade em ambientes low-code ágeis representam desafios atuais e práticos da indústria de software.

3. Avaliação Rigorosa: A solução foi avaliada iterativamente, com uso de métricas automatizadas, auditorias recorrentes e monitoramento contínuo do artefato em uso real.

4. Contribuição à Pesquisa: Os artigos contribuem tanto com soluções práticas quanto com evidências científicas, ampliando o conhecimento sobre qualidade e métricas em desenvolvimento low-code.

5. Base Teórica: A RSL fornece a sustentação teórica para a escolha de métricas, práticas e interpretações dos resultados, fortalecendo o rigor da pesquisa.

6. Projeto Científico: O processo seguiu um ciclo iterativo entre desenvolvimento, avaliação e refinamento, garantindo a evolução do artefato com base em evidências.

7. Comunicação Eficaz: Os resultados foram disseminados por meio de três artigos científicos voltados a comunidade acadêmica e profissional.

Dessa forma, a metodologia DSR foi instrumental para conectar a prática organizacional com a produção científica, proporcionando soluções aplicáveis e, ao mesmo tempo, sustentadas por rigor metodológico.

1.3 ESTRUTURA DO DOCUMENTO

Este trabalho está organizado em três capítulos. O Capítulo 1 apresenta a introdução da pesquisa, contextualizando o tema, destacando sua relevância, estabelecendo os objetivos da tese e justificando a adoção do formato coletânea. O Capítulo 2 reúne os três artigos científicos que compõem o núcleo da tese, organizados como subseções: A Seção 2.1 apresenta o primeiro artigo, que introduz o QSP, um programa de gerenciamento da dívida técnica baseado em métricas automatizadas no contexto de desenvolvimento ágil com low-code. A Seção 2.2 apresenta o segundo artigo, que descreve a evolução do QSP para uma plataforma de monitoramento contínuo da qualidade, com foco em apoiar equipes ágeis na autoavaliação e no aperfeiçoamento de seus processos. A Seção 2.3 apresenta o terceiro artigo, que realiza uma RSL sobre métricas em low-code, identificando as métricas mais recorrentes, as lacunas de pesquisa e a falta de padronização na área. O Capítulo 3 apresenta a conclusão geral da tese, discutindo as contribuições integradas dos estudos, suas limitações e as oportunidades para pesquisas futuras.

2 ARTIGOS QUE COMPÕEM ESTA DISSERTAÇÃO DE MESTRADO

Esta dissertação está organizada no formato de coletânea e é composta por três artigos científicos aceitos em eventos relevantes da área de Engenharia de Software. Os artigos abordam, sob diferentes perspectivas, o uso de métricas para apoiar a

melhoria da qualidade e a gestão da dívida técnica em projetos conduzidos com metodologias ágeis e plataformas low-code.

Os estudos foram organizados de forma sequencial, refletindo a evolução prática e teórica da pesquisa ao longo do tempo. O primeiro artigo apresenta uma prática estruturada de auditoria leve e automatizada voltada para o rastreamento de dívida técnica. O segundo expande essa prática por meio da implementação de uma plataforma de monitoramento contínuo da qualidade. O terceiro artigo oferece uma análise sistemática da literatura sobre métricas em low-code, evidenciando tendências e lacunas no estado atual da pesquisa.

2.1 ARTIGO 01

Título: Tracking technical debt in agile low-code developments
Evento: Congresso Ibero-Americano em Engenharia de Software (CIbSE 2024)
Status: Publicado
Apêndice A
Este artigo apresenta o programa QSP, uma prática criada para rastrear e reduzir a dívida técnica em projetos low-code desenvolvidos com métodos ágeis. A abordagem baseia-se na coleta automatizada de métricas extraídas do próprio ambiente de desenvolvimento, permitindo a realização de auditorias frequentes com baixo custo. Os resultados obtidos indicam uma melhora significativa nos indicadores de qualidade dos projetos, além de promover maior conscientização das equipes sobre a importância da gestão da dívida técnica.

2.2 ARTIGO 02

Título: Low-Code Quality Metrics for Agile Software Development
Evento: Simpósio Brasileiro de Qualidade de Software (SBQS 2024)
Status: Publicado
Apêndice B
O segundo artigo apresenta a evolução do QSP para uma plataforma mais abrangente de monitoramento da qualidade de software em ambientes low-code ágeis. A plataforma permite a coleta e análise contínua de métricas em diferentes áreas, como engenharia, testes, entregas e engajamento. Utilizada ao longo de dois anos por

múltiplas equipes que atuam com plataformas low-code, a ferramenta demonstrou efetividade na melhoria de processos e produtos, oferecendo suporte direto à autoavaliação das equipes e ao acompanhamento da evolução dos projetos de forma sistemática. Por ter sido desenvolvida especificamente para ambientes low-code, sua aplicação se mostrou especialmente eficaz nesse contexto; no entanto, a abordagem também pode ser generalizada para projetos desenvolvidos com código tradicional (high-code), desde que sejam feitas as adaptações necessárias.

2.3 ARTIGO 03

Título: Metrics in Low-Code Agile Software Development: A Systematic Literature Review
Evento: International Conference on Software Technologies (ICSOFT 2025)
Status: Aceito para publicação
Apêndice C

O terceiro artigo apresenta uma RSL sobre o uso de métricas em desenvolvimento com plataformas low-code. A revisão identificou um predomínio de métricas voltadas ao processo de desenvolvimento, enquanto métricas qualitativas, como usabilidade, foram pouco exploradas. Além disso, observou-se uma falta de padronização entre os estudos analisados, dificultando a consolidação de práticas consistentes. O artigo complementa os dois primeiros ao posicionar os resultados práticos da tese frente ao estado da arte, reforçando a relevância das soluções desenvolvidas e indicando oportunidades para futuras investigações.

2.4 DECLARAÇÃO DE AUTORIA

O autor desta dissertação é o principal autor dos trabalhos apresentados neste capítulo. Abaixo são descritas as contribuições específicas do autor em cada artigo:

Artigo 01 – Tracking technical debt in agile low-code developments
A ideia do programa QSP e as auditorias internas já existiam na organização antes do início da pesquisa. A contribuição do autor concentrou-se na automatização do processo de auditoria, no desenvolvimento do código necessário para tal fim, pela implementação do processo de auditoria baseado em métricas automatizadas e pela

condução dos experimentos com equipes de desenvolvimento e na coleta e análise dos dados resultantes. O autor também foi o principal responsável pela redação do artigo, bem como pela submissão e resposta aos pareceres do comitê científico.

Artigo 02 – Low-Code Quality Metrics for Agile Software Development
O autor liderou o projeto de evolução do QSP para uma plataforma de monitoramento contínuo da qualidade, com coleta e análise automatizada de métricas. Foi responsável pela consolidação dos dados longitudinais utilizados na análise e pela redação do artigo, bem como pela submissão e resposta aos pareceres do comitê científico.

Artigo 03 – Metrics in Low-Code Agile Software Development: A Systematic Literature Review
O autor participou ativamente da definição do protocolo da revisão sistemática, da seleção e análise dos estudos primários, e da categorização dos dados obtidos. Também foi o principal responsável pela análise dos resultados e pela redação do artigo, bem como pela submissão e resposta aos pareceres do comitê científico.

Nenhum dos artigos apresentados nesta dissertação foi incluído em outros compêndios de dissertação, nem será submetido para compêndios semelhantes no futuro.

3 CONSIDERAÇÕES FINAIS

A partir da análise dos estudos, é possível afirmar que a aplicação de métricas automatizadas em projetos ágeis com plataformas low-code é não apenas viável, como essencial para uma gestão eficaz da qualidade e da dívida técnica. Essa aplicação ocorre de forma integrada a partir da coleta automatizada de dados, da padronização de práticas em ferramentas de gestão (como o Jira) e da execução de auditorias regulares e autoavaliações com base em métricas objetivas.

Esta dissertação teve como objetivo investigar de que forma o uso de métricas pode apoiar o desenvolvimento ágil com plataformas low-code, com ênfase na melhoria da qualidade do software e no controle da dívida técnica. Para isso, foram conduzidos três estudos complementares: a criação e validação do programa QSP para rastreamento da dívida técnica (DOMINGUES et al., 2024), a evolução do QSP para uma plataforma de monitoramento contínuo da qualidade (DOMINGUES et al., 2024), e a realização de uma RSL sobre métricas no contexto de low-code (DOMINGUES; MONTE; MARINHO, 2025).

O primeiro estudo apresentou o QSP, uma prática estruturada e leve para rastreamento da dívida técnica em projetos desenvolvidos com low-code. A proposta destacou-se pela automação da coleta de dados e pela realização de auditorias frequentes com baixo custo operacional. A aplicação prática do QSP em diversos projetos reais demonstrou melhorias consistentes na maturidade dos processos de desenvolvimento e na qualidade dos produtos entregues.

O segundo estudo expandiu essa abordagem, evoluindo o QSP para uma plataforma de monitoramento contínuo, que integra dados sobre produtividade, engenharia de software, qualidade e entrega. Essa plataforma automatizada possibilitou a construção de dashboards e relatórios analíticos utilizados pelas equipes para autoavaliações periódicas, promovendo uma cultura de melhoria contínua.

Por fim, o terceiro estudo realizou uma RSL sobre o uso de métricas em desenvolvimento ágil com plataformas low-code. A revisão revelou que, apesar do crescimento do interesse pelo tema, há predominância de métricas voltadas para o processo de desenvolvimento e escassez de abordagens focadas em usabilidade e experiência do usuário. Também foi constatada a ausência de padronização na definição e aplicação das métricas.

Em conjunto, os três estudos mostraram que a integração de práticas adequadas de mensuração ao ciclo de vida ágil contribui de maneira significativa para a entrega de projetos low-code de alta qualidade, com menor acúmulo de dívida técnica e maior capacidade de evolução. Os resultados mostram que a literatura ainda carece de uma abordagem unificada para o uso de métricas no desenvolvimento low-code, especialmente integrando qualidade, agilidade e dívida técnica. Este trabalho contribui ao propor e validar práticas e ferramentas que preenchem essa lacuna, oferecendo uma base para futuras pesquisas e padrões na área.

3.1 CONTRIBUIÇÕES

As contribuições desta dissertação podem ser agrupadas em três dimensões: práticas, ferramentas e conhecimento teórico. No âmbito prático, o desenvolvimento do QSP propôs um modelo leve, automatizado e de baixo custo para o rastreamento da dívida técnica e monitoramento da qualidade, facilitando ganhos reais de maturidade em processos e entregas de software ágil com low-code.

Em termos de ferramentas, a evolução do QSP para uma plataforma integrada de monitoramento oferece um ambiente robusto para apoio à autoavaliação das equipes, combinando dados objetivos, dashboards interativos e auditorias sistemáticas, que favorecem a melhoria contínua.

No plano teórico, a revisão sistemática da literatura conduzida evidenciou lacunas importantes no uso de métricas no contexto low-code, destacando tanto a predominância de métricas quantitativas tradicionais quanto a ausência de abordagens padronizadas para avaliação da qualidade e da experiência do usuário. Essa análise mapeia o estado da arte e indica áreas pouco exploradas para futuras pesquisas, contribuindo para o avanço acadêmico da área.

Assim, a tese contribuiu para a prática profissional ao destacar soluções aplicáveis que podem ser implementadas em projetos reais, e apoia a pesquisa acadêmica ao identificar lacunas importantes na literatura que ainda precisam ser investigadas, orientando futuros estudos na área.

3.2 LIMITAÇÕES

Durante o desenvolvimento da pesquisa, algumas limitações foram identificadas:
Uma delas refere-se à dependência do QSP em relação a ferramentas específicas para a automação da coleta de métricas, como o Jira (DOMINGUES; SILVA; MARINHO, 2024; DOMINGUES et al., 2024). Em projetos nos quais a integração com tais ferramentas não foi possível, a aplicação da abordagem proposta ficou limitada.

Outra limitação diz respeito à natureza autorreportada de parte das informações coletadas nas autoavaliações, que, em ambientes organizacionais de baixa maturidade, pode comprometer a confiabilidade dos dados (DOMINGUES; SILVA; MARINHO, 2024; DOMINGUES et al., 2024).

Além disso, não foi conduzido um estudo específico para isolar o impacto da ferramenta desenvolvida em relação a outros fatores que possam ter influenciado os resultados observados. Assim, embora tenham sido identificadas melhorias nos indicadores técnicos e nos processos das equipes que utilizaram a ferramenta, não é possível afirmar com total segurança que esses avanços decorreram exclusivamente da adoção do QSP.

Adicionalmente, embora tenham sido observados avanços em diversas dimensões de qualidade, as métricas relacionadas à eficiência de entregas apresentaram evolução mais modesta, indicando a necessidade de estudos adicionais sobre esse aspecto (DOMINGUES et al., 2024).

3.3 TRABALHOS FUTUROS

Dando continuidade a esta pesquisa, propõe-se a realização de novos estudos voltados ao aprofundamento do monitoramento da eficiência de entregas em projetos ágeis com low-code, buscando identificar fatores que impactam os resultados nessa dimensão (DOMINGUES et al., 2024).
Sugere-se também a ampliação do escopo do QSP, explorando sua adaptação para ambientes de desenvolvimento high-code, com vistas à avaliação de sua aplicabilidade em cenários mais tradicionais.

Outro eixo de pesquisa futuro consiste no fortalecimento da padronização das métricas utilizadas, combinando abordagens quantitativas e qualitativas para avaliações mais robustas e integradas (DOMINGUES; MONTE; MARINHO, 2025).

Por fim, destaca-se a importância de fomentar iniciativas que promovam maior abertura e colaboração com organizações parceiras, permitindo a coleta de dados em ambientes variados e fortalecendo a validação empírica das práticas propostas. Além disso, recomenda-se a realização de estudos de replicação para validar os resultados em diferentes contextos, avaliações com usuários finais para assegurar a adequação das soluções às necessidades práticas, e análises de custo-benefício que possam contribuir para a viabilidade e adoção das ferramentas desenvolvidas.

REFERÊNCIAS

1- Domingues, Renato; Reis, Miguel; Araújo, Miguel; Marinho, Marcelo; Silva, Mario J. **Tracking technical debt in agile low code developments**. Congresso Ibero-Americano em Engenharia de Software (CIbSE). SBC, 2024.

2- Domingues, Renato Cavalcanti, Mario J. Silva, and Marcelo Luiz Monteiro Marinho. **Low-Code Quality Metrics for Agile Software Development.** Proceedings of the XXIII Brazilian Symposium on Software Quality. 2024.

3- Domingues, Renato, Iury Monte, Marcelo Marinho. **Metrics in Low-Code Agile Software Development: A Systematic Literature Review**. 20th International Conference on Software Technologies. 2025.

4- Hevner, Alan R., et al. **Design science in information systems research**. MIS quarterly (2004): 75-105.

5- Rokis, Karlis & Kirikova, Marite. (2023). **Exploring Low-Code Development: A Comprehensive Literature Review**. Complex Systems Informatics and Modeling Quarterly. 68-86. 10.7250/csimq.2023-36.04.

6- Mahesh Sankaran, Dr. E.N Ganesan, 2021, **Accelerate & Deliver better Software by Cost Saving and Cost Optimization Continous Improvements through Agile Methods**, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 10, Issue 09 (September 2021),

7- ISAYAMA, Hajime. Attack on Titan. Capítulo 80. Kodansha, 2016.

APÊNDICE A – Tracking technical debt in agile low-code developments

Tracking technical debt in agile low code developments

Renato Domingues[1,2], Miguel Reis[2], Miguel Araújo[2], Marcelo Marinho[1], Mário J. Silva[2,3]

[1]Departamento de Computação – Universidade Federal Rural de Pernambuco (UFRPE)
Recife – PE – Brazil

[2]Truewind, Axians Portugal
Recife – PE – Brazil
Lisboa – Portugal

[3]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[1]. However, managing technical debt growth is hard to achieve, since ke-

[1]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[2], 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].

Krutchen 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,

[2]`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[3], which is used to monitor the quality of low code application developments made on the platform, and AI Mentor Studio[4], 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:

[3] https://www.outsystems.com/blog/posts/launching-the-quality-apps-program/
[4] 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:

$$FinalScore = \frac{\sum answer}{\sum Questions} \tag{1}$$

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[5] 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:

[5] `https://www.atlassian.com/software/jira`

Audit Category	Metric
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:

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

$$EstimationAccuracy = 100\% - |RelAccuracyError| \tag{3}$$

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

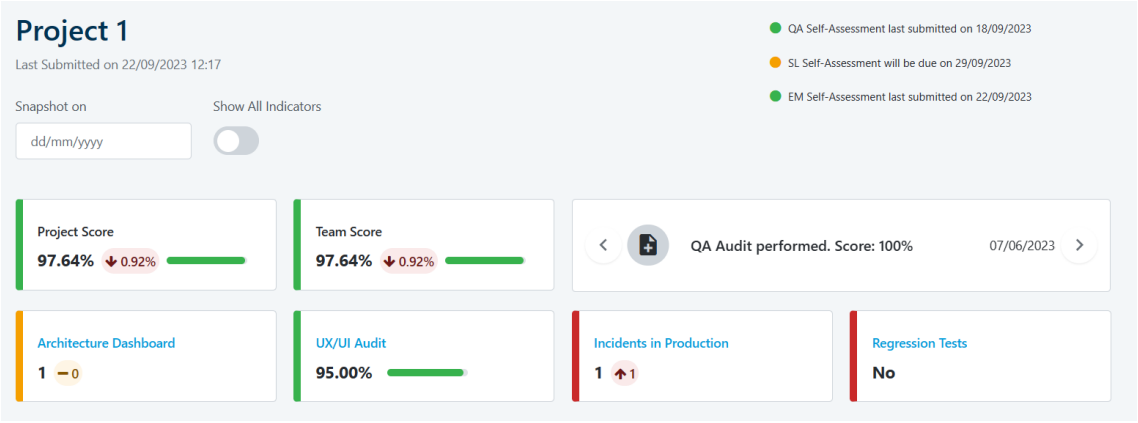
We assign four score grades to each metric: *Excellent*, *Good*, *To improve*, and *Not met*, which corresponds to obtaining 100% of the metric's full score, 2/3, 1/3 and 0%, respectively. Furthermore, we have also developed the metrics so that they are always categorized into one of the four available 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:

$$AuditScore = \sum_{i=1}^{f} Metric_i * weight_i \tag{4}$$

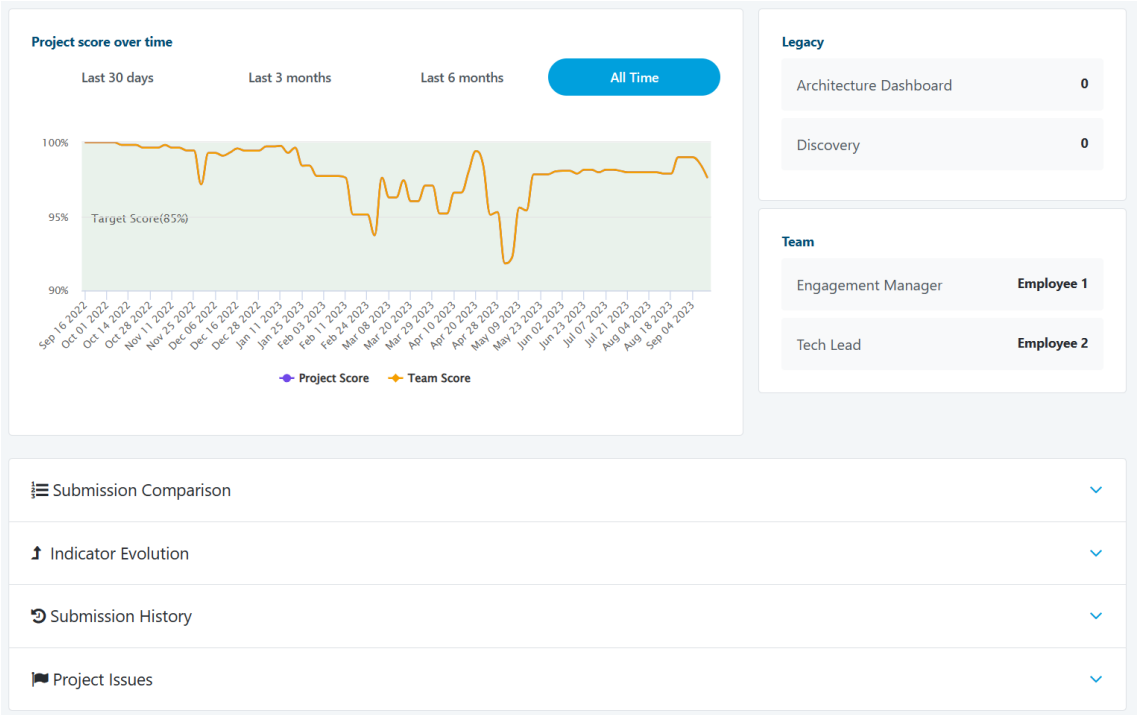
4.4. QSP Dashboards

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 (*n*ot passed), one yellow (*T*o 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.

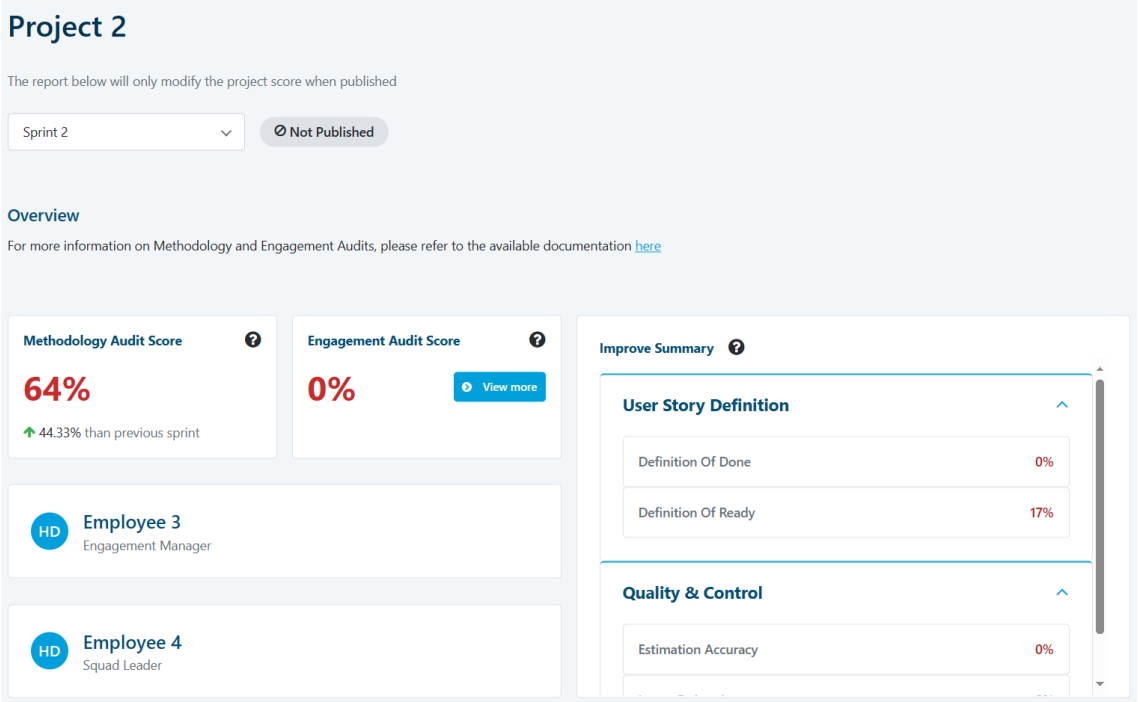


Figure 2. Engagement Manager details of a QSP audited project. The audit score is shown on the left and the metrics that need to be improved on the right.

5. Results

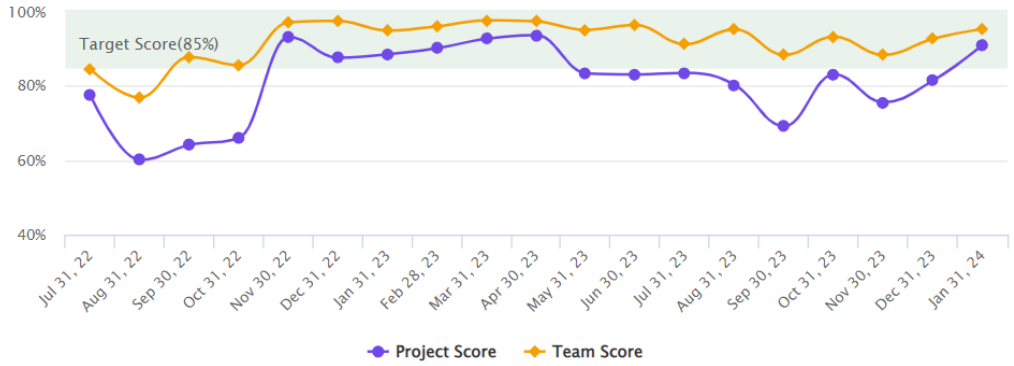
At the time of writing this article, the QSP has been used in more than 30 projects and currently has 8 others active, by teams from different sizes, from 3 to 10 members per team.

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.

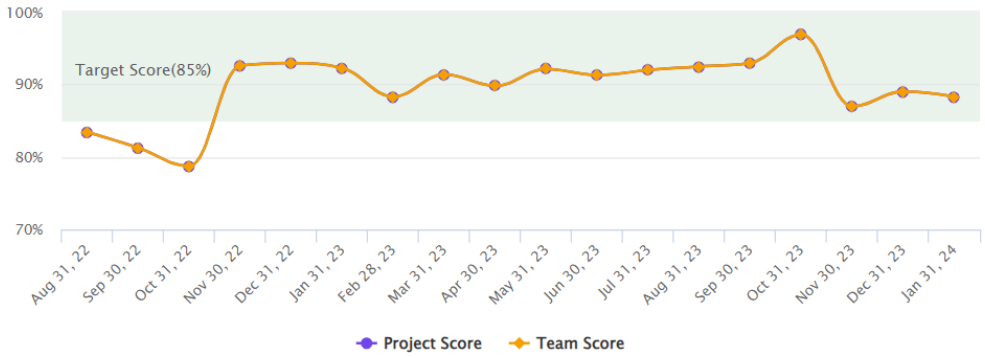
Finally, Figure 3d) shows the mean of the final score of all the projects and teams. Although once again the first four months had the lowest results, since then, we have

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.

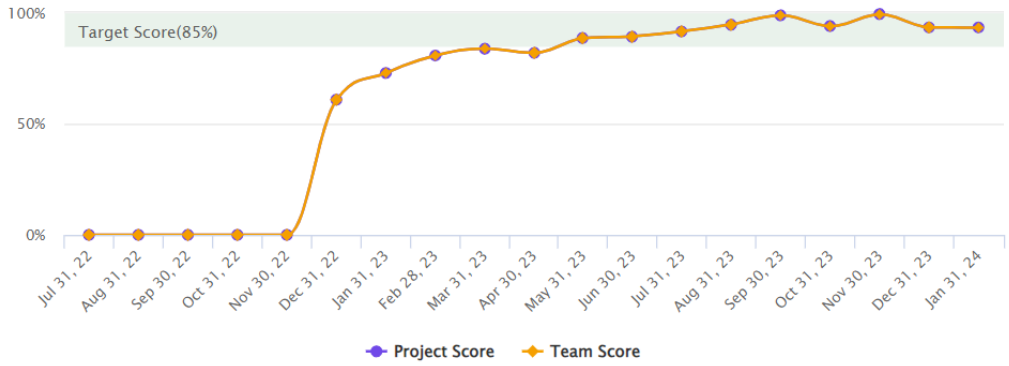
When we take the grades for the year 2022 we obtain an average score of 88.22% for tech debt, 85.80% for audits and 87.64% for the final score, in 2023 these metrics rose to 93.88 %, 91.52% and 91.96% respectively. The quality metric cannot be calculated in this way given that the 2022 sample is only 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.



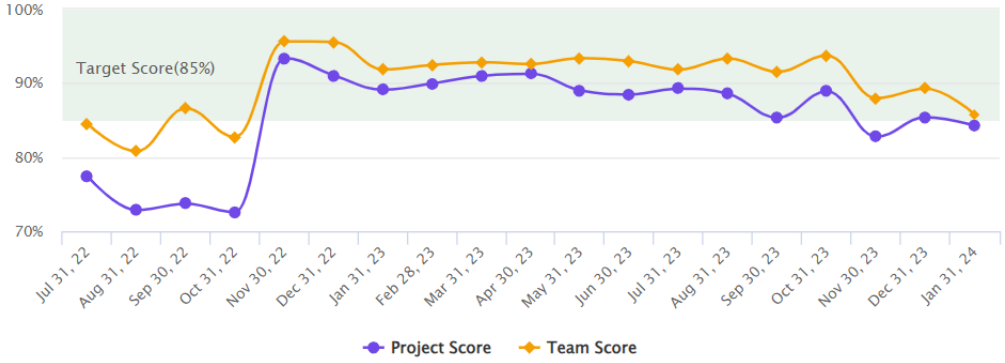
a) Technical debit metrics score



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`. 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 Euromicro 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.

APÊNDICE B – Low-Code Quality Metrics for Agile Software Development

Low-Code Quality Metrics for Agile Software Development

Renato Cavalcanti Domingues Filho
Department of Computer Science (DC)
Federal Rural University of Pernambuco (UFRPE)
Recife, Pernambuco, Brazil
Truewind
Recife, Brazil
renato.domingues@ufrpe.br

Mario J. Silva
INESC-ID Instituto Superior Técnico
Universidade de Lisboa
Lisboa, Portugal
mjs@inesc-id.pt

Marcelo Luiz Monteiro Marinho
Department of Computer Science (DC)
Federal Rural University of Pernambuco (UFRPE)
Recife, Pernambuco, Brazil
marcelo.marinho@ufrpe.br

Abstract

We present a quality metrics monitoring and analysis platform, which has been developed to support the main goal of delivering customer value at competitive costs without sacrificing quality in the context of fast digital transformation using a low-code platform and agile development methods. The quality platform collects project data and enables the automatic computation of metrics tailored to our software process that development teams can use for self-assessment. Since introducing our platform two years ago, our teams have been using it constantly, demonstrating its effectiveness and importance for our software development process.

CCS Concepts

• **Software and its engineering** → **Agile software development**; *Programming teams*; • **Applied computing** → Enterprise applications.

Keywords

Metrics, Low Code, Agile development

ACM Reference Format:
Renato Cavalcanti Domingues Filho, Mario J. Silva, and Marcelo Luiz Monteiro Marinho. 2024. Low-Code Quality Metrics for Agile Software Development. In *XXIII Brazilian Symposium on Software Quality (SBQS 2024), November 05–08, 2024, Salvador, Brazil*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3701625.3701626

1 Introduction

One of the greatest challenges faced by companies today is ensuring the delivery of value to their customers. This challenge is amplified when we consider that delivered products must also have high quality. Considering that one obvious software development trade-off is exchanging quality for speed to meet delivery time, how can we ensure that our development teams will not compromise quality in exchange for speed?

A possible solution to this dilemma is monitoring agile teams throughout the development of products or services to ensure that the final delivery will be within accepted standards. Agile methodologies enable this through continuous integration[5], pair programming[15], code reviews[11], and continuous feedback [9]. As the number of agile teams increases, the challenge of managing a team of teams (or an Agile Release Train (ART) in the Scaled Agile Framework [1]) becomes more difficult.

We believe that in large settings, a methodology relying on quality metrics should not require too much involvement of project managers in resolving quality aspects, if quality metrics enable identification of problem areas, facilitate decision-making and promote continuous monitoring and improvement [8] [1].

Considering these questions, our organization has been developing a new quality monitoring platform supporting an internal *Quality Solutions Program* (QSP). The QSP platform provides support for conducting self-assessments and software audits as part of the monitoring activities. Self-assessments are questionnaires to be answered at the end of each sprint by the member responsible for the development team, gathering information about how development progressed throughout the sprint. QSP supports two types of audits, manual and automated. Progress is reviewed in manual audits with the participation of the auditor and a team member, while the auditor alone checks the correctness of results at the end. Like self-assessments, audits are carried out at the end of each sprint.

This paper aims to expand on previous work that has introduced the QSP platform [4], describing which metrics are obtained through the self-assessment and how we categorize each of them. We introduce Engagement Outcome Reviews, a new enterprise software development monitoring process for tracking the delivery of business value and include an update to the audit metrics.

2 Background

2.1 Metrics

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,

SBQS 2024, November 05–08, 2024, Salvador, Brazil

ACM ISBN 979-8-4007-1777-2/24/11
https://doi.org/10.1145/3701625.3701626

[1] https://scaledagileframework.com/agile-release-train/

revenue/sales impact, product utilization, and scope change in a release [10].

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 [6]. 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.

2.2 Agile Audits

In agile software projects, to identify issues earlier and facilitate their correction, it is important to establish which metrics will be audited and organize the interaction between the auditor and the team so that both understand what is being audited and the need to conduct audits at each project cycle [13].

Joshi identifies three key components for an agile audit [7]: Backlog (collection of scoped work like audit plan), Sprints (scoped items divided into sprints period), and Scrums (short meetings to evaluate the work done and identify bottlenecks). An agile audit has three main phases [2]:

Planing and preliminary research: Determine what will be audited, what aspects are relevant to the audit, and define some metrics such as the Definition of Ready and the 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 the objective, scope, conclusion, recommendations, and action plans.

2.3 Quality assessment tools for low code software development

Low-code platforms are revered for their grains in productivity. One of the advantages is the support of visual programming tools with graphical user interfaces for application design instead of hard-coded programming techniques [14]. By using low code platforms, developers can focus on business logic rather than dealing with setting up computing infrastructure, managing data integrity across different environments, and ensuring non-functional aspects [12]. Software quality improves in developments with low code platforms, given that there is a higher reuse of tested library modules interconnected by automatically generated code.

In addition, low-code platforms also provide tools for managing the quality of application software. Outsystems, a leading low-code development platform, offers several tools for managing software quality. Among them, the Quality Apps Program (QAP)[2], which is used to monitor the quality of low code application developments , and AI Mentor Studio[3], for static code analysis. Our QSP program uses these Outsystems tools to collect some of its key metrics.

[2]https://www.outsystems.com/blog/posts/launching-the-quality-apps-program/
[3]https://www.outsystems.com/product-updates/ai-mentor-studio/

3 Self-Assessments metrics

The Self-Assessment is the primary method for obtaining metric data in the QSP. Self-assessments are carried out biweekly or at the end of each sprint. The project's engagement manager, i.e. the person responsible for coordinating the team, fills a form containing a list of questions of two possible types Boolean or numeric. This is a deliberate choice, intended to generate easily explainable QSP self-assessment reports.

Each metric can be associated to one or more of five metric groups:

Engagement(1): Project management and control
Engineering(2): Development and performance
Quality Assurance(3): Quality assurance
Technical Debt(4): Technical Debt management
Delivery(5): Encompasses all of the above

Additionally, each of these metrics is of one of two metrics Categories: 1) Factual: such as the number of bugs or technical debts identified. 2) Predictive: for instance if demos are being performed, this is relevant because without a demo some of the development work may not be run-time checked until too late, by the end of the project. By adding predictive metrics we may assess aspects that could have a heavy negative impact later on in the project.

Table 1 summarizes which metrics are obtained, the metrics groups they are associated with, their respective weights for the final score and the type of value received as input. In total, we define 37 metrics, 28 of which in group Delivery, 20 in Engagement, 10 in Quality Assurance, 5 in Engineering and 3 in Technical Debt. The three tech debt metrics: AI Mentor result, AI Mentor security result and Discovery result are specific metrics of the Outsystems low code platform used within the company, which can be obtained with the help of the AI Mentor tool available from Outsystems. For instance, one of the features of the AI Mentor is a static code analysis scans the code for technical debts. The AI Mentor result metric corresponds to the number of technical debts found in this analysis.

All self-assessment questions are evaluated and scored. Boolean questions have 0 or 1 scores. Numerical questions scores can have 0, 0.5 or 1 scores, based on pre-defined assessment criteria. In addition, each metric is weighted by following empirical rules:

- Factual metrics have higher weight than predictive metrics; this ensures they have more impact on the self-assessment score.
- The score is impacted positively when a team improves; this motivates the teams.

3.1 Self-Assessment Scores

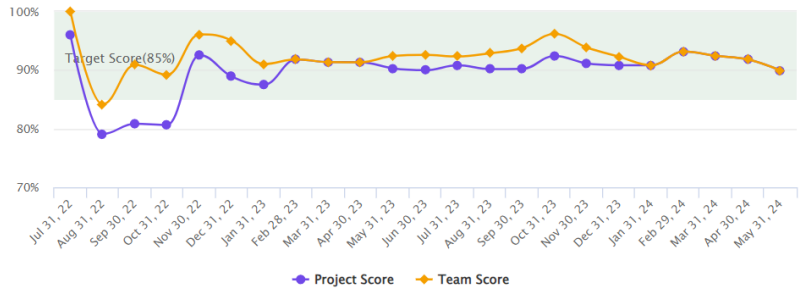
The QSP platform has been used by our ART in more than 40 internal projects by teams from different sizes, from 3 to 10 members per team. Figures 1a to 1e provide average monthly scores in Metric Groups for all projects in the QSP platform. Figure 1d, which shows the QA Metrics Group, the score remains at 0% in the starting months because the data acquisition for this metrics group took a little longer to be developed and was concluded at the end of 2022.

In Figures 1a, 1c and 1e we observe two types of scores: Project score and Team score. We have this differentiation for this metric group because it includes metrics based on legacy items inherited

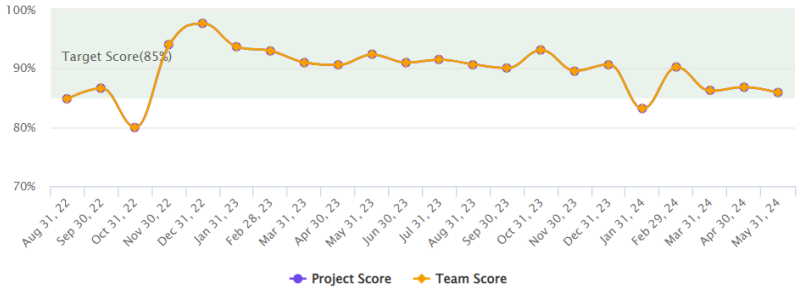
Table 1: Metrics collected in a QSP self-assessment, their Metric Groups(Engagement(1), Engineering(2), Quality Assurance(3), Technical Debt(4) and Delivery(5)), Weight, Input type(Num for numeric and Bool for Boolean) and a brief description of what they mean.

Metric	Metric Groups	Weight	Input	Description
AI Mentor result	2, 4	34	Num	A team member uses the AI Mentor's tech debt identifier and reports the # of tech debt found.
AI Mentor Security result	2, 4, 5	0	Num	A team member uses the AI Mentor's tech debt identifier and reports the # of tech debt related to security found.
Average Execution time	2	34	Bool	The average execution time of project functions is within the acceptable limits defined for that project?
Backlog Management	5	3	Bool	Backlog was managed during this sprint?
Change fail percentage	1	1	Num	% of failures in functions that underwent some update during this sprint.
Definition of Done	1, 5	2	Bool	Definition of Done was written for the artifacts of this sprint?
Definition of Ready	1, 5	2	Bool	Definition of Ready was written for the artifacts of this sprint?
Demos	1, 5	8	Bool	Demo was performed?
Deploy + Rollback plan	1, 5	5	Bool	Deploy or/and rollback plan were made for this sprint?
Deploy frequency	1, 5	1	Num	# of deploys carried out in this sprint.
Discovery result	2, 4	21	Num	A team member uses the AI Mentor's tech debt identifier and reports the # of problems related to the architecture found.
Dry-run	1, 5	5	Bool	Dry-run was performed?
Engagement Outcome Review	1, 5	5	Bool	The Engagement outcome goals for the sprint were achieved?
Impact Matrix	3, 5	1	Bool	Impact matrix was done?
Incidents in Production	1	13	Num	# of non-trivial incidents in production.
Lead time	1, 5	1	Num	What was the lead time for this sprint?
Post-mortems	1, 5	2	Bool	Post-mortems for the incidents were done?
Pre requirements	3	3	Bool	The pre requirements was written for the artifacts of this sprint?
Regression Tests	3	1	Bool	Regression tests were done?
Retrospectives	1, 5	2	Bool	Retrospectives were done?
Risk Assessment	3, 5	5	Bool	The risk assessment was done?
Security Tests	3, 5	5	Bool	Security tests were done?
Smoke Tests	3, 5	3	Bool	Smoke tests were done?
Sprint duration	5	5	Bool	The sprint lasted within the planned deadline?
Sprint planning	5	5	Bool	The sprint met the plan?
Stakeholder Map Analysis	1, 5	3	Bool	Stakeholder map analysis were done?
Story items "Done"	1, 5	8	Bool	The # of stories planned and completed for this sprint was sufficient?
Test coverage	3, 5	5	Bool	Test coverage were done?
Test execution	3, 5	2	Bool	The planned tests were done?
Test plan-Engagement Manager	1, 5	8	Bool	The engagement manager created a test plan for this sprint?
Test plan-QA	3, 5	0	Bool	The member responsible for QA created a test plan for this sprint?
Test Strategy	3, 5	1	Bool	The test strategy were done?
Time to Restore	1, 5	1	Num	How long is the system taking to recover in the event of a failure?
Top error report	2	21	Num	# of error reports received during this sprint.
Velocity & Estimation	1	5	Num	Time of work performed during this sprint.
# of Non-Trivial bugs in Quality	1, 5	3	Num	# of non-trivial quality-related bugs reported during this sprint.
# of Trivial Bugs in Quality	1, 5	0	Num	# of trivial quality-related bugs reported during this sprint.

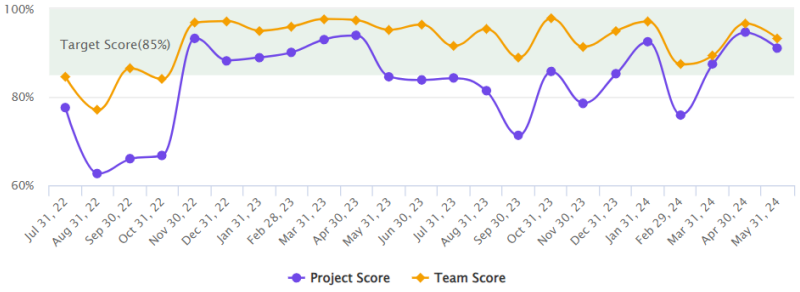
from early developments by previous teams. Such items are artifacts and/or technical debts that already existed in the project before the team's work began. This situation normally occurs when a project does not start from scratch within our ART. With this differentiation we can have a measure reflecting both the real quality level of a project and the quality of the contributions of the team to the project.



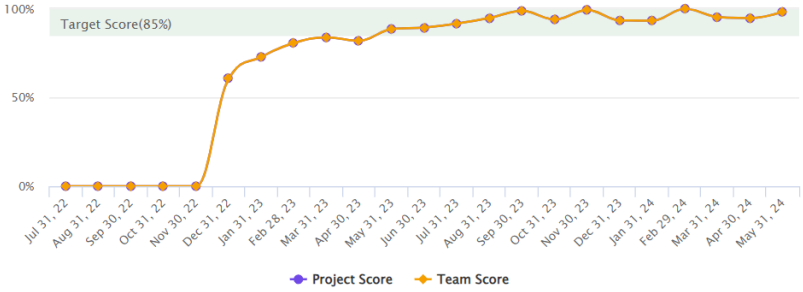
a) Delivery metrics average score over time



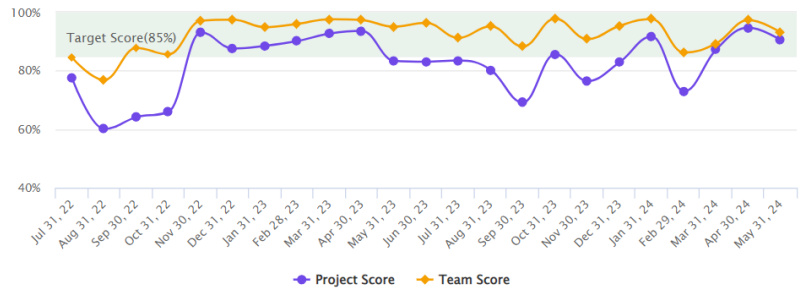
b) Engagement metrics average score over time



c) Engineering metrics average score over time



d) QA metrics average score over time



e) Technical debt metrics average score over time

Figure 1: Self-assessment Metric Groups scores of our ART over time since QSP started

4 Audits metrics

Given the complexity and diversity of domains of expertise present in a typical low code project, it would be very hard to have them all reviewed in a single audit session. Therefore, a QSP managed project is fully assessed under five different types of audits: i) Architecture, ii) Engagement, iii) Front-End, iv) Quality Assurance and v) UX/UI.

In the QSP, we follow the approach of carrying out our audits frequently, sometimes at the rate of one per sprint. Each of these audit sessions has its own ways of being conducted, but they are all time-limited to two hours. Without a time limit, we would be overspending on resources we couldn't afford. Hence, to keep the agility of our methodology we restrict the review to the most effort-consuming issues. In this paper, we only detail the Engagement Audit type, due to space limitations. This is, however, the audit-type using the most comprehensive set of QSP metrics.

4.1 Engagement Audit

Within the QSP, we established a practice of carrying out an Engagement Audit on the last day of a development sprint. The audit is based on data automatically extracted from the Jira[4] board of the project. In an Engagement Audit, we consider three categories (User Stories Definition, Board Management and Quality & Control) described below:

4.1.1 User Stories Definition Metrics. With the metrics of this category we seek to evaluate how User Stories (USs) are processed in a sprint. We consider two sub-groups of metrics, those evaluating all the USs of a sprint being audited, and those evaluating a sample. The sample is obtained by taking the three USs with highest estimated effort. Table 2 shows which metrics are in each group, if they affect the audit's Engagement Score and a description of it.

4.1.2 Board Management metrics. For this category we consider information not only from the sprint but also from the backlog and the board. Table 3 shows how we calculate the score of these metrics.

Backlog prioritization or MoSCoW[3]: The Backlog is prioritized in a way that helps build consensus, establish a unified direction, and gain a broader perspective. In order to give teams greater flexibility in how they carry out this prioritization, we adopted a less rigid approach and therefore analyzed it as follows: On our boards we have five possible priority options for an item: *Lowest*, *Low*, *Medium*, *High* and *Highest*. By default, every item that is automatically created is assigned priority *Medium*.

Defects management: Check for defects being handled as any other Backlog item. Similarly to the previous metric, we check the priority of issues of the defect type to verify if at least 80% of them are marked as *Medium*. If not defects management is not being correctly handled.

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 a sprint. On projects that do not involve UX/UI development, the auditor may deactivate this metric and its score is automatically redistributed to the other metrics.

[4]https://www.atlassian.com/software/jira

Reported defects: This is purely informative data showing the quantity and distribution of each type of defect severity created in a sprint. In our boards it is possible to have four different severities: *Trivial*, *Minor*, *Major* and *Critical*.

Sprint goal defined: While it's easy to gather a bunch of Backlog Items to work on in a sprint, it is a little harder (but much more valuable) to have a set of Backlog Items that, by fitting together, provide more business value. To determine this metric, we adopted a similar approach to the metrics of the sample subgroup of the USs Definition category, where we analyzed the textual field of the board called Sprint Goal, to check the number of words in it.

4.1.3 Quality & Control metrics. We consider the following metrics in this category:

Sprint Definition: A period of three days is given at the beginning of each sprint to plan it, after it we take a snapshot of all the items that were programmed for that sprint. At the end of the sprint we check the snapshot and compare it with the items completed over the sprint to measure how many of the planned items were completed or discarded and how many were added in the middle of the sprint.

Capacity Tracker: In equation 1, the *sprintDays* represents the number of work days of that sprint, *teamDevs* the number of working developers for that sprint, and *daysOff* is a manual input that the engagement manager can adjust for accounting absences. The constant 7 represents the number of hours a person works a day. We use 7 instead of the full day to account for project activities not reported in issues, such as meetings, and the constant 0.85 to compensate the three days period of sprint planing. Then we compare the capacity with the estimated time for the issues planned for this sprint.

$$Capacity = ((sprintDays * teamDevs) - daysOff) * 7 * 0.85 \quad (1)$$

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.

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:

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

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

Furthermore, as additional information that does not influence the final score, we show the Root Mean Square Error (RMSE) in hours of the estimations and compare it with a database of pre-selected company projects so team members can have an idea of how they compare to others.

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

Table 2: User Stories Definition Metrics, their name, which user stories are considered, if the metric scores and a brief description.

Metric	Evaluated User Stories	Scores	Description
USs sizing	All	YES	Calculates the distribution of the USs based on the number of hours estimated to be carried out and separates it into three subgroups: below 2 days, between 2 and 3 days and above 3 days.
USs average size	All	NO	Compares the average number of hours in USs for this sprint against the company average.
Work item type	All	NO	Shows the number and distribution of issues of each type created in this sprint.
INVEST criteria*	Sample	YES	USs should be focused on features – not tasks, and written in business language. Doing so will enable business people to understand and prioritize the USs.
Acceptance criteria*	Sample	YES	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)*	Sample	YES	Reinforce transparency, assure Built-In Quality, and set the right expectations for the work items to be planned, developed, and completed.

*This metric is manually reviewed by the auditor in the final stage of the audit.

Table 3: Board Management Metrics, their name, which artifacts are used as input, if the metric scores and a brief description.

Metric	Input	Scores	description
Backlog prioritization or MoSCoW	Backlog issues	YES	We first look at whether all items in the backlog are marked with the default priority, then we look at whether more than 80% of the items have *Highest* priority.
Defects management	Backlog issues	YES	Check the priority of issues of the defect type to verify if at least 80% of them are marked as *Medium*.
UX/UI management*	Sprint issues	YES	Check whether there are issues of this type in the sprint. On projects that do not involve UX/UI development, the auditor may deactivate this metric and its score is automatically redistributed to the other metrics.
Reported defects	Sprint issues	NO	Show the quantity and distribution of each type of defect severity.
Sprint goal defined*	Board information	YES	We analyzed the textual field of the board called Sprint Goal, to check the number of words in it

*This metric is manually reviewed by the auditor in the final stage of the audit.

Table 4: Quality & Control Metrics, their name, if the metric scores and a brief description.

Metric	Scores	description
Sprint Definition	YES	Compare the difference between the items that were planned at the beginning of the sprint and how many of them were completed.
Capacity Tracker	YES	We calculate the team capacity for that sprint following the equation 1.
Issues Estimation	YES	Compares the number issues and defects that have effort estimation with the total number of issues of these types.
Estimation Accuracy	YES	Compares the number of worklog hours logged within the sprint period against the capacity value of that sprint.
Number of bugs in Quality	YES	We first check the coverage of the sprint, then compare the number of non-trivial defects with the number of Increments (Stories and Improvements).

(1) We take two lists, the first contains all the user stories planned for this sprint and the second contains all the user stories

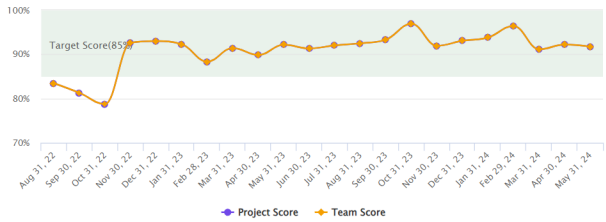


Figure 2: Audit scores of our ART over time since QSP started

that the tests are covering, so we intersect them. The coverage of this sprint can then be calculated by dividing the size of the resulting list with that of the list of planned user stories. If the coverage is above a minimum threshold, then this sprint has a minimum test coverage and can advance 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). To obtain the maximum score in this metric the proportion must be up to 0.5 defects per Increment and anything over 3 is a full penalty.

4.2 Audit Score

Figure 2 shows the audit scores of our ART (all audits) in QSP. The threshold of 85% has not been reached in the first three months of monitoring. However, by the end of 2022, we had reached that target and little by little the score has been rising. Our ART has achieved a score above the threshold in every month of 2023 and 2024.

5 Engagement Outcome Review

With the above metrics, we can monitor each team sprint by sprint. Another equally important aspect of our software services is ensuring that what we deliver returns value to our clients. We established a new quality assurance process dedicated to monitoring value delivery and engagement Outcome Review to meet this goal. Value delivery metrics are specific for each project. At the start of a project, the project manager meets with the project's product owner to identify and define metrics that add value to the project. For example, a client who is in the middle of automating a business process can define the number of manual actions against to the number of automated actions of that process as a proxy for value; another example would be measuring while performing an update to a client application, the percentage of users who began using new features of a business application.

Once the value delivery metrics are defined, it is necessary to define the weight of each metric, as some may be more critical for the customer. Like all other metrics, value delivery metrics are defined at the start of a project. However, client priorities do not always remain the same. The priority of some processes can increase or decrease, and new processes can be added at any time. As a result, we allow modification of value delivery metrics throughout the sprints. It is possible to add a new value delivery metric, remove an existing one, and/or modify the weight of each one. However, such modifications are not retroactive, i.e., scores of previous sprints are not affected.

In our QSP practice, at the end of each sprint, during the manual review stage of the Engagement audit, the same auditor also checks the Engagement Outcome Review metrics to assess the value delivery achieved with that sprint.

Table 5: Average team score per year by category

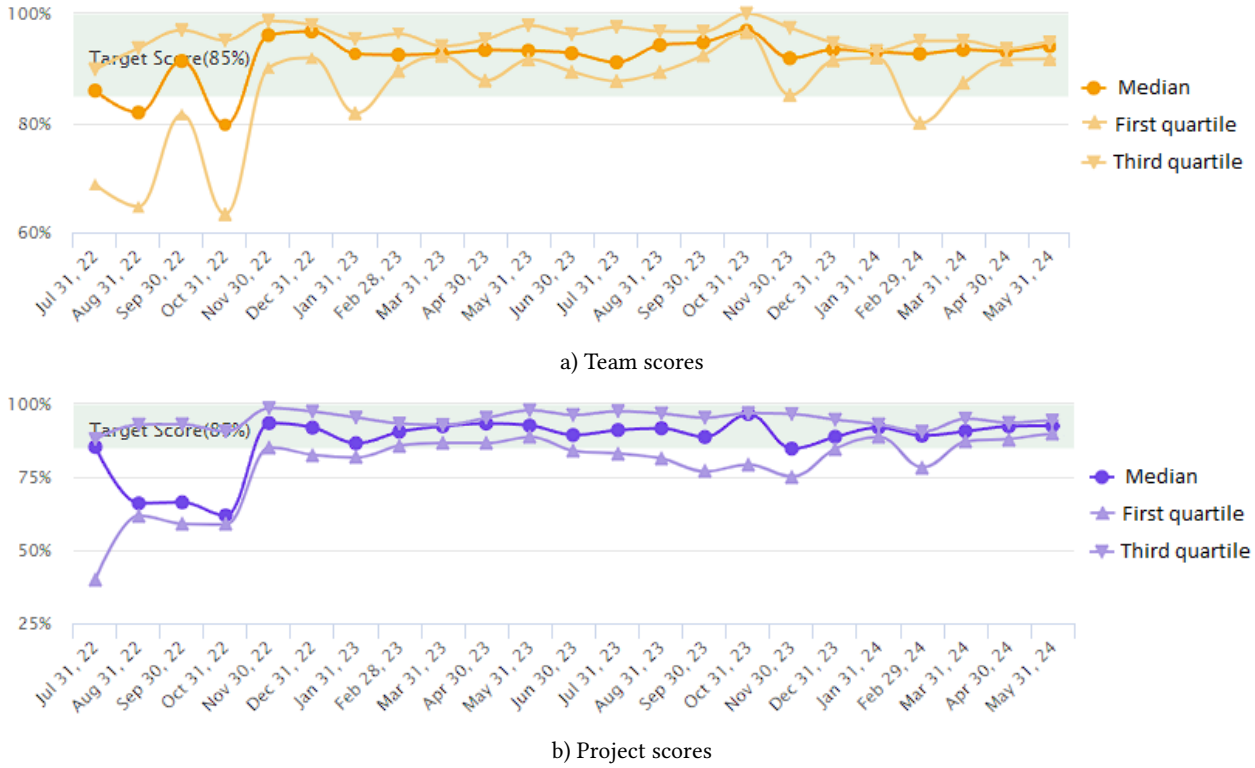
Category	2022	2023	2024*
Audit	85.80%	90.23%	95.40%
Delivery	92.56%	92.19%	92.14%
Engagement	86.38%	90.59%	87.48%
Engineering	87.70%	92.91%	92.78%
QA	60.78%	86.87%	96.31%
Tech Debt	88.22%	92.99%	92.75%
Final	87.64%	91.44%	91.64%

*= Data available until may 2024

6 Quality Performance Results

Figure 3 shows the combined final scores, obtained after weighting all metrics categories, of the quality-monitored projects. We can see the results month by month of the first, second (median) and third quartiles. This plot helps us make considerations about the relative performance of the teams and how quality is improving in different groups of projects and teams. For example, in the period from August to November 2023, it is possible to observe significantly different scores for the first and second quartiles, However, when we checked team scores in the same period we did not find the same variation, This highlights the importance of separating the team and project score so as not to negatively affect the teams' motivation. Although the quality of delivery is the most important factor for our development process, another equally relevant factor is the consistency of delivery of our teams. As all teams are now following the same standard of rules and metrics, the results produced remain consistently within our target score of 85%, even among teams in the first quartile. All medians have been above the target since 2023. Even the group in the first quartile was below expectations only in two months, in February 2023 and March 2024, but still achieved above 80% in the global score.

In Table 5 we can observe the average scores for the years 2022, 2023, and 2024 (until the end of May). The table provides a detailed analysis of our performance in each category, showing where our teams improved or had a reduced score over time. It is possible to observe, for each Audit category, how we are improving over time, as well as in the QA category and in the Final Score. On the Delivery category we are witnessing a drop over time. Despite being almost insignificant, the Audit and QA categories have the largest increases. Audit scores rose by almost 10% going from 85.80% in 2022 to 95.40% in 2024. Although in 2022 we have 60.78% in QA, it is not fair to consider this figure, given that only the month of December had any measurement. Even so, if we focus on 2023 and 2024, we can see that there is still a significant increase of almost 10%, from 86.87% to 96.31%. Delivery was our best category in 2022. Despite a small drop over time, it still did not fall by even 0.5%, still



a) Team scores

b) Project scores

Figure 3: Final overall scores of projects ranked in the 3 first quartiles over time

remaining at 92%. Engagement and Tech Debt present a similar result, where we had an increase compared to 2022 but a decrease compared to 2023 in 2024. Tech Debt variation at just 0.24% below, remains insignificant. However, Engagement had a reduction of 3.11%. In the end, we still observe an increase in the final grade of 4% compared to 2022 and 2024.

7 Conclusion

In this work, we have introduced the 37 metrics of QSP that have been used in self-assessments and how we associate these metrics to five metrics groups: Delivery, Engagement, Engineering, Quality Assurance and/or Technical Debt. We also provided details on how each of the 16 Engagement audit metrics are calculated alongside with the results of the audit metrics. We presented the new Engagement Outcome Review process, a new process for monitoring and measuring the delivered value to our customers. Finally, we have shown overall results containing the final score obtained by all projects within the QSP and compare the results obtained yearly to evaluate our growth and performance.

Overall, the QSP shows that our ART has been improving in all but one of the metrics categories we are tracking. In the single category that has not improved, Delivery, the drop wasn't however significant. These results show the effectiveness of our approach to monitoring the quality of the software services delivered by our development teams.

The QSP platform is being developed with the aim of supporting a large agile low-code ART primarily delivering software based on the Outsystems platform. We believe that our approach can be adapted to other low-code environments or even for environments of traditional (high-code) development.

The main limitation is the need for manually introduced self-assessment data by a responsible team member. Sometimes, engagement managers forget or delay the reporting progress data despite automatic email reminders whenever a delivery is late. As teams are not forced to report metrics data on time, when delivery is very late, someone higher up must intervene to ensure a self-assessment is performed. Another limitation is that the Engagement audit automation still depends on access to Jira data from that project. Without access to Jira's API, the automation does not work, and the Engagement audit becomes unavailable.

In future work, we intend to research why we have not witnessed improvements in the Delivery Metrics group since the QSP was introduced and then identify ways to improve it. We also intend to improve automation on the other four QSP audit types that this paper has not covered.

Acknowledgments

This work was possible thanks to the people at Truewind who helped in the development of the QSP project, especially Miguel Araújo and Miguel Reis. We acknowledge financing from Truewind, Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES), and national funds through FCT, Fundação para a Ciência e a Tecnologia, under project IDB/50021/2020 (DOI:10.54499/UIDB/50021/2020).

References

[1] [n. d.]. Agile Project Metrics Formulas Explained. https://teamhub.com/blog/agile-project-metrics-formulas-explained/. Accessed: 2024-05-31.

[2] 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: 2024-05-31.

[3] Dai Clegg and Richard Barker. 1994. *Case method fast-track: a RAD approach*. Addison-Wesley Longman Publishing Co., Inc.

[4] Renato Domingues, Miguel Reis, Miguel Araújo, Marcelo Marinho, and Mário J Silva. 2024. Tracking technical debt in agile low code developments. In *Congresso Ibero-Americano em Engenharia de Software (CIbSE)*. SBC, 226–240.

[5] Martin Fowler and Matthew Foemmel. 2006. Continuous integration.

[6] Alejandro Guerrero, Rafael Fresno, An Ju, Armando Fox, Pablo Fernandez, Carlos Muller, and Antonio Ruiz-Cortés. 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*. 1139–1143.

[7] Prem Lal Joshi. 2021. A review of Agile internal auditing: Retrospective and prospective. *International Journal of Smart Business and Technology* 9, 2 (2021), 13–32.

[8] Eetu Kupiainen, Mika V Mäntylä, and Juha Itkonen. 2015. Using metrics in Agile and Lean Software Development–A systematic literature review of industrial studies. *Information and software technology* 62 (2015), 143–163.

[9] Agile Manifesto. 2001. Manifesto for agile software development. (2001).

[10] Elizabeth Mkoba and Carl Marnewick. 2020. Conceptual framework for auditing agile projects. *IEEE Access* 8 (2020), 126460–126476.

[11] Peter C Rigby and Christian Bird. 2013. Convergent contemporary software peer review practices. In *Proceedings of the 2013 9th joint meeting on foundations of software engineering*. 202–212.

[12] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. 2020. Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 171–178.

[13] Linh Truong. 2020. Agile auditing: More value, less resources. *Edpacs* 62, 1 (2020), 4–7.

[14] Robert Waszkowski. 2019. Low-code platform for automating business processes in manufacturing. *IFAC-PapersOnLine* 52, 10 (2019), 376–381.

[15] Laurie A Williams and Robert R Kessler. 2000. All I really need to know about pair programming I learned in kindergarten. *Commun. ACM* 43, 5 (2000), 108–114.

APÊNDICE C – Metrics in Low-Code Agile Software Development: A Systematic Literature Review

Metrics in Low-Code Agile Software Development: A Systematic Literature Review

Renato Domingues[1,2][a], Iury Monte[1][b] and Marcelo Marinho[1][c]
[1]*Universidade Federal Rural de Pernambuco, Recife, Brazil*
[2]*Axians Low Code Brasil, Recife, Brazil*
{*renato.domingues, iury.tavares, marcelo.marinho*}*@ufrpe.br*

Keywords: Low-Code, Software Metrics, Systematic Review.

Abstract: Low-code development has gained traction, yet the use of metrics in this context remains unclear. This study conducts a systematic literature review to identify which metrics are most used in low-code development. Analyzing 17 studies, we found a strong focus on Development Metrics, while Usability Metrics were underexplored. Most studies adopted quantitative approaches and fell into the Lessons Learned category (58.8%), suggesting an exploratory phase with little metric standardization. Future work should focus on standardizing metrics and incorporating qualitative insights for a more comprehensive evaluation.

1 INTRODUCTION

The software industry is experiencing rapid growth, and IT spending is expected to increase from $5.11 trillion in 2024 to $5.61 trillion in 2025 (Gartner, 2025). However, challenges such as rising complexity, developer shortages, and the demand for faster deliveries persist alongside this expansion. In addition, the need for fast deliveries without compromising quality is increasingly important. Faced with these difficulties, low-code development platforms (LCDPs) emerged as an alternative to address these problems. These platforms allow for the creation of applications through visual interfaces and reusable components, significantly reducing the need for manual programming. This allows both experienced developers and users without technical training, commonly called citizen developers, to advance in software development more quickly and efficiently.

Low-code development is inherently aligned with agile methodologies, as its visual programming approach and reusable components enable rapid prototyping, continuous user feedback, and incremental software delivery, reducing development time and enhancing adaptability to changing requirements. The benefits of using LCDPs include flexibility and agility, fast development time, quick response to market demands, reduced bug fixing, lower deployment effort, and easier maintenance. Hence, the industry of low-code development is gaining popularity rapidly (Al Alamin et al., 2021). According to (Gartner, 2021) by 2025, 70% of new applications developed by organizations will use low-code or no-code technologies, up from less than 25% in 2020.

Low-code platforms often emphasize speed and productivity in their marketing materials. For instance, OutSystems claims ”Build better apps – 10x faster. Your stakeholders will hug you for it”[1], while Genio advertises ”Up to 10x faster developing new projects with 1/10 of resources.”[2]. In both examples we can see the mention that development is 10x faster, but just developing faster is not necessarily enough, other factors such as usability, security, and product performance are equally important.

This work seeks to better understand how researchers and industry members measure these factors. To do so, we reviewed the literature with the following research question: *“What metrics are described in the literature for the agile low-code software development process?”*. The rest of this work follows the following structure: Section 2 introduces some concepts and related works, Section 3 talks about the methodology applied in this work, Section 4 shows the results obtained, Section 5 brings a brief discussion of the results, Section 6 brings threats to

[a] https://orcid.org/0009-0003-0455-609X
[b] https://orcid.org/0009-0001-7966-3653
[c] https://orcid.org/0000-0001-9575-8161

[1] https://www.outsystems.com/
[2] https://genio.quidgest.com/?lang=en

the validity of our work, and finally Section 7 brings a brief reflection as well as our future works.

2 BACKGROUND

2.1 Agile

Agile methodology was introduced in 2001 by the agile manifesto (Beck et al., 2001); this methodology aims at greater flexibility, collaboration, and speed of delivery. Since then, agile has been increasingly used among companies in the software development process, according to the most recent state of agile report (Digital.ai, 2024) 71% of respondents were already using agile in their software development life cycle, with SCRUM being the methodology most used by teams and SAFe the most used at the enterprise level.

2.2 Low-Code

The term "low-code" was created by Forrester Research in 2014 to describe platforms that enable software development with minimal need for manual coding (Forrester, 2014). Unlike traditional programming languages such as Python, Java, C, etc., which have their source code built from a series of logical instructions written by a programmer, low-code platforms use visual interfaces, pre-built components, and configurable logic to minimize the amount of manual coding. The goal is to speed up the development process and require less team effort.

As described by (Sahay et al., 2020), a LCDP typically has four layers: application, service integration, data integration, and deployment. In the application layer, users interact directly with the graphical environment to define their applications; in the service integration layer, it is used to integrate the application with different services, usually via APIs; in the data integration layer, it is used to homogeneously manipulate data from different sources; and finally, the deployment layer is used to deploy the application in dedicated cloud infrastructures or on-demand environments.

According to Gartner (Gartner, 2024), low-code platforms can be divided into four types: challengers, niche players, visionaries, and leaders, based on their completeness of vision and ability to execute. Some examples of these platforms are: Mendix[3], OutSystems[4], Appian[5], Oracle Apex[6] and salesforce[7]

2.3 Related Work

The topic of low-code is relatively new compared to other branches of technology. As mentioned, the term was only created slightly more than 10 years ago, but its interest has grown. Given this growth, several studies have sought to consolidate existing knowledge about low-code through literature reviews. These reviews analyze the state-of-the-art, identify challenges and trends, and provide a basis for future research.

(Rokis and Kirikova, 2023) provides a summary of the knowledge about low-code in the literature, covering topics such as what low-code is, its platforms, application areas, benefits and challenges. Based on a review of the literature, the authors analyzed 39 articles to group the information found, for example, in a table that contains the benefits of low code and which works provide this information.

(Prinz et al., 2021) conduct a literature review on LCDPs, analyzing academic research and industry reports to map the state-of-the-art. The study uses a systematic review of the literature, reviewing 32 publications and categorizing them according to the sociotechnical system model. The authors identify that most current research is focused on the technical system (technology and tasks), while only three studies explicitly address the social system (organizational structure and people).

(Bucaioni et al., 2022) perform a multi-vocal systematic review, where they analyze both peer-reviewed publications and gray literature studies with the aim of providing a comprehensive view on low-code development. In total, 58 primary studies were analyzed and based on the results found, they framed low-code as a set of methods and tools inserted in a broader methodology, often associated with Model-Driven Engineering (MDE).

(Khalajzadeh and Grundy, 2024) presents a systematic review of the literature that, at the end of the search, backward and forward snowballing stages, analyzed 38 primary studies related to the topic of low-code and accessibility of low-code platforms. The results obtained show that despite there being some concern with the topic of accessibility in low-code platforms, few studies actually address this issue and even those that do have some limitations in their approach.

(Curty et al., 2023) focuses on MDE and low-code/no-code platforms applied to the development

[3]https://www.mendix.com/

[4]https://www.outsystems.com/

[5]https://appian.com/

[6]https://apex.oracle.com/en/platform/low-code/

[7]https://www.salesforce.com/

of blockchain-based applications. Through a structured literature review, the authors analyzed 177 academic publications to categorize the focuses of these works and identified that there is a difference between academic and industrial approaches, where academic approaches are more conceptual and experimental, while industrial low-code and no-code platforms are more mature, but with less flexibility for detailed modeling.

It is already possible to find in the literature several studies that carry out some form of literature review as demonstrated in the papers cited above, some more generalized such as (Rokis and Kirikova, 2023; Prinz et al., 2021) that deal with low-code as a whole, while others have a more specific focus on certain topics such as (Khalajzadeh and Grundy, 2024) that addresses accessibility in a low-code context and (Curty et al., 2023) that deals with works related to blockchain technology. However, during our searches we were unable to identify any work that was specifically focused on metrics, so in order to address this gap in the literature we decided to carry out this review.

3 METHODOLOGY

To ensure a rigorous and reliable approach, this study adopted the Systematic Literature Review (SLR), as defined by (Kitchenham and Charters, 2007). This methodology allows for the structured evaluation and interpretation of all available research that is relevant to a research question, thematic area, or phenomenon of interest. Furthermore, the SLR seeks to present a fair and impartial assessment of the topic, following a rigorous, auditable, and reproducible methodological process, ensuring greater transparency and reliability in the results.

This study followed the guidelines proposed in (Kitchenham and Charters, 2007) and consisted of three main stages: review planning, review implementation, and review findings reporting. Each stage was subdivided into smaller steps as shown below:

- Planning the review: (i)Recognizing the need for a SLR; (ii) Formulating research question(s) for the review.
- Implementing the review: (i)Performing a search for relevant studies; (ii)Evaluating and documenting the quality of included studies; (iii)Categorizing the data needed to address the research question(s); (iv)Extracting data from each incorporated study.
- Reporting of review findings:(i)Summarizing and synthesizing study findings; (ii)Interpreting the results to determine their applicability; (iii)Developing a comprehensive report of study results.

With this, we sought to answer our research question: *"What metrics are described in the literature for the agile low-code software development process?"*. To perform the search, we defined a comprehensive search string to ensure that we would capture the largest possible number of results: *("low code" OR "low-code") AND ("agile software development" OR "agile method" OR agile OR SCRUM OR Kanban OR Lean OR "lean software development")*. We used this string to search for articles, conferences and journals in three databases: ACM, IEEE and SpringerLink.

3.1 Paper Selection

Our search yielded 1,369 references from 2001 to 2024, including 788 from ACM, 344 from IEEE, and 237 from SpringerLink. The paper selection process involved two phases of filtering the results:

- **Phase 1:** An initial selection of papers that reasonably met the inclusion and exclusion criteria based on reading their titles, abstracts and keywords.
- **Phase 2:** A more detailed selection, where the criteria were applied more rigorously, and the introductions and conclusions of the initially selected papers were read.

Following these phases, we conducted both forward and backward snowballing on the papers accepted in both phases. At each stage, two authors reviewed all the material, and in case of disagreement regarding the acceptance of a paper, a third author made the final decision.

In Phase 1, we included studies that broadly aligned with the inclusion criteria in Section 3.2, particularly those related to low-code development. We focused on the titles, abstracts, and keywords during this phase.

In Phase 2, we became more selective and only included papers that, in addition to meeting the criteria from Phase 1, also addressed metrics or presented a metric related to low-code. We read each paper's title, abstract, keywords, introduction, and conclusion in this phase.

3.2 Inclusion and Exclusion Criteria

For this work we defined the following criteria:

We *included*: (i) Studies that were published in journals and peer-reviewed conferences; (ii) Studies directly related to the research questions; (iii) Studies that approach research topics, such as low-code and low-code metrics; and (iv) Studies that are available, via the university library services, to the authors during the time of search or available on the web.

We *excluded*: (i) Studies not written in English; (ii) Documents that are books, short papers (¡=4 pages), theory papers, workshop papers, technical reports, students experiments; (iii) Studies that presents personal viewpoints or specialists opinions; and (iv) Studies not related to Software engineering and computer science.

Before accepting an article into the final set for review, we checked for replication, e.g. if a given study was published in two different journals with a different order of lead authors, only one study would be included in the review. In addition, we checked for duplication, e.g. if the same article was listed in more than one database, only one study would be included in the review.

After defining all inclusion and exclusion criteria, we initiated Phase 1 by reviewing the relevant sections (title, abstract and keywords) of the retrieved studies, resulting in 109 preliminarily accepted papers. These were then evaluated more thoroughly in Phase 2, leading to a final selection of 12 papers, an acceptance rate of approximately 0.88%. Table 1 presents the number of studies accepted from each search engine across both phases.

Table 1: Papers by search engine.

Engine	Selection	Phase 1	Phase 2
ACM	788	73(9.26%)	8(1.02%)
Springer	344	21(8.86%)	2(0.58%)
IEEE	237	15(4.36%)	2(0.84%)
Total	**1369**	**109(7.96%)**	**12(0.88%)**

After completing this step, we started the snowballing backward and snowballing forward on these 12 articles. Through backward snowballing, we identified and accepted one additional paper. Forward snowballing yielded seven potential studies; however, one was inaccessible, and two were excluded for being undergraduate and master's theses, as per our exclusion criteria. This resulted in four additional accepted papers from forward snowballing. In total, these steps added five papers to our selection, leading to a final set of 17 accepted studies.

3.3 Paper Quality

The quality assessment criteria used in this study are based on established principles and good practices for conducting empirical research in software engineering defined by (Dyba et al., 2007). For this purpose, we used a list of 12 questions that can be answered with Yes, Partially and No, we assigned the values 1, 0.5 and 0 respectively, to each of these questions and at the end we calculated the score for that study. The 12 questions used were: (i) Is there a clear definition of the study objectives? (ii) Is there a clear definition of the justifications of the study? (iii) Is there a theoretical background about the topics of the study? (iv) Is there a clear definition of the research question (RQ) and/or the hypothesis of the study? (v) Is there an adequate description of the context in which the research was carried out? (vi) Are used and described appropriate data collection methods? (vii)Is there an adequate description of the sample used and the methods for identifying and recruiting the sample? (viii) Is there an adequate description of the methods used to analyze data and appropriate methods for ensuring the data analysis were grounded in the data? (ix) Is provided by the study clearly answer or justification about RQ / hypothesis? (x) Is provided by the study clearly stated findings with credible results? (xi) Is provided by the study justified conclusions? (xii) Is provided by the study discussion about validity threats?

For our study we consider a paper as Excellent if it obtains a score equal to or greater than 11, Good if is between 8.5 and 11, Regular if between 6 and 8.5 and Insufficient if it is below 6.

3.4 Papers Rigor and Relevance

Our research evaluation process is based on the rigor and relevance assessment methods proposed by (Ivarsson and Gorschek, 2011). Rigor aspects were assessed on a scale of 0 ('weak'), 0.5 ('moderate') and 1 ('high') and had three dimensions: (i) Context; (ii) Study design; and (iii) Threats to validity.

On the other hand, industry relevance concerns the impact a study can have on industry and academia, considering relevant research topics and real industry scenarios. The relevance aspect has a binary score, 1 for present and 0 for not present. The aspects are: (i) Subjects of the study, described as the people involved in the case, e.g., industry professionals; (ii) Context in which the study was conducted, e.g., industrial settings; (iii) Scale of applications used in the study, e.g., realistic industrial applications; (iv) Research method used. The maximum Rigor value is 3

while the maximum Relevance value is 4.

3.5 Data Extraction

We examined each selected paper to extract the following elements: (i) Study objective or research question; (ii) Results relevant to the study; (iii) Potential themes emerging from the study findings.

We synthesized the data by first identifying the indicators and their purpose. Because we assigned equal weight to each occurrence, the frequency of occurrence only reflects how many articles mention a given practice, so frequency reflects the popularity of a topic rather than its potential importance;

We classified all included articles into one of the research type facets derived from (Wieringa et al., 2006). All reviewed studies were also classified using contribution type facets derived from (Petersen et al., 2008) .

4 RESULTS

4.1 Overview of the Papers

Table 2 presents the list of all 17 selected studies, along with their assigned IDs, which will be used throughout this section for easier reference.

Table 2: List of selected studies and their corresponding IDs.

ID	Reference
1	(Al Alamin et al., 2021)
2	(Martins et al., 2020)
3	(Alamin et al., 2023)
4	(Overeem et al., 2021)
5	(Domingues et al., 2024)
6	(Hagel et al., 2024)
7	(Kakkenberg et al., 2024)
8	(Varajão et al., 2023)
9	(Guthardt et al., 2024)
10	(Luo et al., 2021)
11	(Calçada and Bernardino, 2022)
12	(Kedziora et al., 2024)
13	(Trigo et al., 2022)
14	(Sahay et al., 2023)
15	(Haputhanthrige et al., 2024)
16	(Pacheco et al., 2021)
17	(Bexiga et al., 2020)

Although we searched for articles from 2001 to 2024, in our final result we did not have any articles until 2019, only from 2020 to 2024. The figure 1

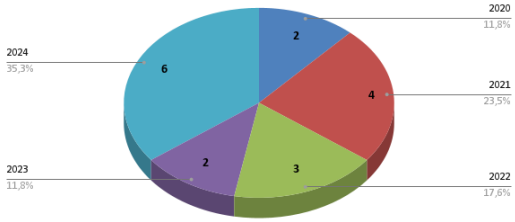


Figure 1: Papers distribution per year.

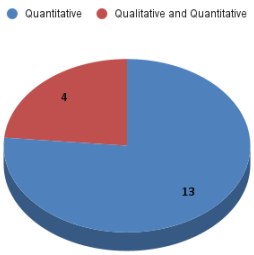


Figure 2: Papers study type.

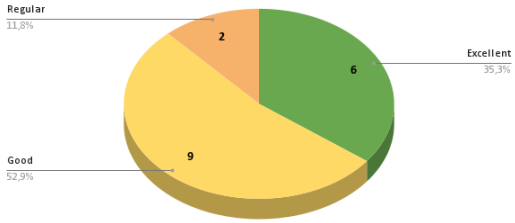


Figure 3: Papers quality.

shows the distribution of papers in each year, where we had 2 papers in 2020 and 2023, 3 in 2022, 4 in 2021 and 6 in 2024.

Regarding the type of study, as we can see in the figure 2 we did not have any study that was only qualitative, however we had 4 studies that were qualitative and quantitative and 13 that were quantitative.

The results from Section 3.3 approach are presented in Figure 3, showing the classification of the analyzed papers. Six were rated as Excellent, nine as Good, two as Regular, and none received a score of Insufficient. As no work was classified as Insufficient, none of the papers had to be discarded.

As mentioned in the Section 3.4 the papers were classified by their rigor and relevance according to the method of (Ivarsson and Gorschek, 2011). Figure 4 show the distribution of scores of the papers, the caption shows the number of papers with that score, in it we can see that no paper scored below 1.5 when we refer to Rigor, nor below 3 when we talk about relevance. The Rigor value that had the highest frequency was 3 (maximum score) with 7 papers, followed by 2 with 6 papers, 1.5 with 3 papers and 2.5 with 1 paper. For Relevance, we had a less wide distribution with 9 papers obtaining the score 4 (maximum score) and 8

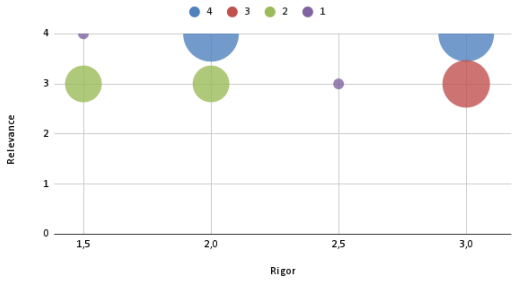


Figure 4: Papers Rigor and Relevance distribution.

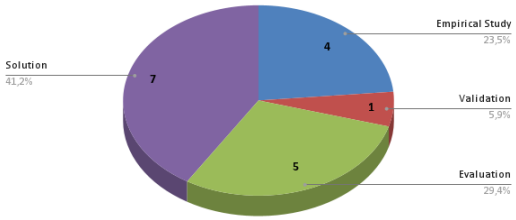


Figure 5: Paper research facet distribution.

with the score 3. When we look at both dimensions at the same time (Rigor as X and Relevance as Y) we have a tie in the highest frequency where (2,4) and (3,4) both have 4 papers, followed by (3,3) with 3 papers, (1.5,3) and (2,3) with 2 papers each and 1 single paper with (2.5,3).

For the categorization of (Wieringa et al., 2006) by the type of research facet, that was described in Section 3.5, figure 5 shows the distribution of the results, where the most present type was Solution with 7 papers, followed by Evaluation with 5 papers, Empirical study with 4 and Validation with 1 paper.

And for the categorization of (Petersen et al., 2008) by type of research contribution, the figure 6 shows that there was a predominance for the Lessons Learned category with 10 papers, followed by Framework with 3 and finally we had a tie between Tool and Model where each of them had 2 papers.

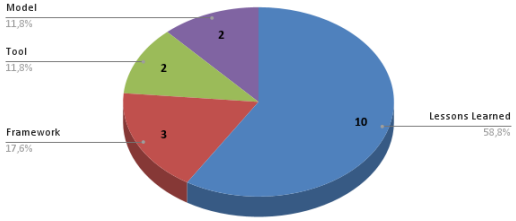


Figure 6: Paper research contribution distribution.

4.2 Thematic Analysis of the Papers

To answer this question, we decided to group our metrics into two categories: Main topic and Off-topic. The metric groups marked as Main topic are those that help us answer our RQ, while the Off-topic, are those that talk about low-code metrics but not specifically about the development process and therefore do not help us with the RQ. The 3 table shows the metric groups we defined as well as in which paper we found it:

Table 3: Metric groups, their categories and in which paper it was found.

Metric group	Category	Paper ID
Productivity Metrics	Main topic	8,11,13,17, 18,20
Development Metrics	Main topic	2,5,6,8,9,11, 13,17,18,20
Quality Metrics	Main topic	5,8,12,13,18
Performance Metrics	Main topic	8,9,11,12,13, 14
Usability Metrics	Main topic	6,12
Popularity & Engagement Metrics	Off-Topic	1,3,10
Model Metrics	Off-Topic	7
Compliance Metrics	Off-topic	4

To clarify the categorization in Table 3, we briefly define each metric group: Productivity — metrics related to output and efficiency of development (e.g., effort, speed); Development — metrics describing the process and activities involved in building software; Quality — metrics assessing software correctness, reliability, and defect levels; Performance — metrics evaluating system behavior under operation (e.g., execution time, resource use); Usability — metrics reflecting user satisfaction and interface quality; Popularity & Engagement — metrics based on public interest and discussion around low-code; Model — metrics focused on the structure and representation of visual models; Compliance — metrics measuring adherence to platform or organizational standards.

4.2.1 RQ: What Metrics Are Described in the Literature for the Agile Low-Code Software Development Process?

In this subsection we will describe the metrics of each group that we defined as Main topic: Productivity, Development, Quality, Performance and Usability.

Productivity: The table 4 briefly shows all the metrics found for this group.

Table 4: Productivity metrics found.

Paper ID	Metric
8,13,11	Lines of code
8,13	Use case points analysis (UCPA)
8,13	Constructive Cost Model (COCOMO II)
8,13	Function point analysis
8,13	Productivity factor
17	Skill value
17	Industry skill value
18	# of pages/Time
20	New results/Old results

(Varajão et al., 2023) is a continuation of (Trigo et al., 2022) made by the same authors, changing the context in which the experiment is carried out and therefore the metrics found were the same. Among the metrics referenced, Use Case Points Analysis (UCPA) and a specifically defined Productivity Factor are the only ones of this metric group effectively employed in the study. Lines of code, the Constructive Cost Model (COCOMO II), and Function Point Analysis are mentioned solely as potential alternatives, without being applied in the actual analysis. The UCPA is used as a basis for calculating the use case points (UCP) and by consequence the Productivity factor. In a simplified way, the Productivity factor can be calculated from the amount of total effort divided by the UCP and the result is adjusted by a quality factor, that will be better described in the Quality topic.

The Lines of code metric does not make much sense for our low-code context, it is traditionally used for traditional high-code environments, but we decided to add it to our list because we could find it in 3 papers (Varajão et al., 2023; Trigo et al., 2022; Calçada and Bernardino, 2022), being merely cited in papers (Varajão et al., 2023) and (Trigo et al., 2022), but actually used in (Calçada and Bernardino, 2022). In the context of (Calçada and Bernardino, 2022), it makes comparisons between low-code, Java Swing and JavaScript development; in this comparison, one of the metrics used is the number of lines of code written by hand.

In (Haputhanthrige et al., 2024) it was possible to find 2 metrics, which are very similar. Originally, the Skill Value metric is divided into three that measure the skill of a team member at the beginning, during and at the end of a project in a quantified way. However, for this work we decided to group them into just one metric. The Industry Skill Value is a scale from 0 to 100, calculated based on the industry average, considering time of experience and salary. Values above 50 indicate that the person is above the industry average.

(Pacheco et al., 2021) presents a single straightforward productivity metric, comparing the number of screens delivered with and without their UX/UI solution over the same period. In contrast, (Bexiga et al., 2020) introduces its own solution and applies it to past projects, comparing the new results with previous ones.

Development: The table 5 briefly shows all the metrics found for this group.

Table 5: Development metrics found.

Paper ID	Metric
2,6,8,9,11, 13,18,20	Development time
8,13	Total effort
8,13	UCP
6,9	Task completion time
5,17	# of project hours
2	# of backlog items
5	A set of 6 user stories centric metrics
5	A set of 5 project board centric metrics
5	Development score
9	# of questions asked
9	Task success %

The Development Time metric is the most recurrent among all groups analyzed, being identified in a total of eight studies (Martins et al., 2020; Hagel et al., 2024; Varajão et al., 2023; Guthardt et al., 2024; Calçada and Bernardino, 2022; Trigo et al., 2022; Pacheco et al., 2021; Bexiga et al., 2020), its high frequency shows the popularity of this metric, which may also suggest its importance.

As mentioned previously, papers (Varajão et al., 2023) and (Trigo et al., 2022) present the same metrics, UCP and Total effort, which are used to calculate the Productivity factor metric. Because the calculation involves dividing Total Effort by UCP, a lower result indicates better performance.

Both (Hagel et al., 2024) and (Guthardt et al., 2024) use the time required to complete a task as a metric, but only (Guthardt et al., 2024) verifies the success rate of these completed tasks. Another metric identified in (Guthardt et al., 2024) is the number of questions asked, which is particularly relevant in the context of this study. In this experiment, the researchers recorded the number of questions from participants before the tests began.

The other metric that was found in more than one paper is # of project hours, in (Haputhanthrige et al., 2024) it is related to the Skill value metric for the distribution of tasks, where more relevant skills would have a greater allocation of hours. In (Domingues et al., 2024) it is one of the factors used in the cal-

culation of one metrics that will be better described in the quality topic.

In the paper (Domingues et al., 2024) we were able to identify a total of 17 metrics, 12 of which are development metrics and 5 of which are quality metrics. In the development metrics, we separated them into three: those specific to user stories, those related to the project board, and the development score. In the context of this work, a score from 0 to 100% is given for each project and team based on an evaluation; the Development score metric is the combination of these results. As for the sets of metrics for user stories, the size, its size in relation to the others, the type, and the completion of some specific text fields are observed. For board metrics, it is observed whether the management and completion of some specific parts of the project, such as UX/UI or the backlog, are present.

Quality: The table 6 briefly shows all the metrics found for this group.

Table 6: Quality metrics found.

Paper ID	Metric
8,13	Quality factor
18	Precision
18	Recall
12	Transaction success %
5	A set of 5 quality centric metrics

As already mentioned in the productivity topic, the papers (Varajão et al., 2023) and (Trigo et al., 2022) present a metric called Quality Factor. This metric was initially proposed in (Trigo et al., 2022) and considers three factors when evaluating the quality of the developed project: Mockups, Use case description and software errors. In (Varajão et al., 2023) they developed this metric to consider a fourth factor, which is the performance.

In (Pacheco et al., 2021) they developed a tool to assist in UX/UI development, in this context they adapted the use of two metrics, traditionally from models, to calculate the success of the tool: Precision and Recall. Despite using both, the one selected to be the main factor was Precision, as it was better for the tool to only assist when it was sure that something was correct.

In the context of (Kedziora et al., 2024) they use a robot as a service approach, and to calculate the service level agreement they defined 4 metrics, 3 of which were for Performance and 1 for Quality. The quality metric was Transaction success %, something that they themselves define as an uncommon thing to do.

The third set of metrics defined in (Domingues et al., 2024) contains 5 metrics related to quality that cover the estimation of project issues, their accuracy, the number of bugs, test coverage, whether what was defined at the beginning of the sprint was fulfilled and the team's work capacity.

Performance: The table 7 briefly shows all the metrics found for this group.

Table 7: Performance metrics found.

Paper ID	Metric
8,11,13,14	Execution time
9,14	Resource usage
11	Runtime
12	Uptime
12	Recovery
12	Availability

Execution time was the second most mentioned metric, seen in 4 papers (Varajão et al., 2023; Calçada and Bernardino, 2022; Trigo et al., 2022; Sahay et al., 2023). This metric was generally used when they wanted to refer to the amount of time needed to complete a given task.

The Resource usage metric was presented in two different ways. In (Guthardt et al., 2024) it is only mentioned in the interviews conducted in this work as a relevant metric to be implemented in their tool. In (Sahay et al., 2023) they bring it up in the context of BPMN and say that although BPMN models are not completely measurable, Execution time and amount of required memory are valid ways to measure it.

(Calçada and Bernardino, 2022) in its comparisons of results between low-code, Java Swing and JavaScript this work brought both Execution time and Runtime as performance metrics.

As mentioned in the previous topic, (Kedziora et al., 2024) brought 3 performance metrics, they are Uptime which is the percentage of time that the service is available, Recovery speed which is how long it takes for the system to recover after a service interruption, and Availability which they define as the time difference between the initial time of the request and the time in which the request was actually initiated.

Usability: The table 8 briefly shows all the metrics found for this group.

Table 8: Usability metrics found.

Paper ID	Metric
12	Customer satisfaction
12	Customer impact
6	System Usability Scale (SUS)

For this group we found only 3 metrics, 2 of them from (Kedziora et al., 2024) and 1 from (Hagel et al., 2024). The ones from (Kedziora et al., 2024) are not

explicitly found in the work, they can be found implicitly in the statements present in the text, for example:"...much more valuable for customers...", "We take responsibility for the impact to Customer..." and "...as well as cost, quality, and customer experience.".

(Hagel et al., 2024) in their approach they carried out experiments with 18 participants, where, at the end of each task that was performed they sent a SUS-type questionnaire for the participants to answer about the usability.

4.2.2 Off-Topic Metric Groups

In this subsection, the groups of metrics that were found but do not help us to answer our RQ will be described, they are: Popularity & Engagement, Compliance and Model Metrics. Although these metrics do not directly contribute to answering our research question, they provide context on how low-code is being discussed and evaluated in different domains. Understanding these broader metrics can help researchers identify new perspectives on low-code evaluation.

Popularity & Engagement: The table 9 briefly shows all the metrics found for this group.

Table 9: Popularity & Engagement metrics found.

Paper ID	Metric
1,3,10	Popularity of topics
1,3,10	Development questions
1,3	Difficulty of questions
1	Classification questions

The papers (Al Alamin et al., 2021), (Alamin et al., 2023) and (Luo et al., 2021) are works that have as a methodology to analyze and extract data from questions that were posted on online sites. (Al Alamin et al., 2021) and (Alamin et al., 2023) are in a similar situation to (Varajão et al., 2023; Trigo et al., 2022), where (Alamin et al., 2023) is a continuation of (Al Alamin et al., 2021) with same authors, they search on stackOverflow (SO)[8], while (Luo et al., 2021) searched both on SO and on reddit[9], within reddit they filtered into three subreddits: "Low Code"[10], "No Code"[11], and "nocodelowcode"[12].

For Popularity of topics we find metrics such as: number of posts, mention frequency, frequency over time, average views and average favorites. Development questions encompass issues such as benefits, challenges, and limitations of using low-code and LCDPs, and distribution of issues throughout the agile life cycle. Difficulty of questions deals with metrics like: % of unanswered questions, time to get an accepted answer, and % of questions with no accepted answer. Classification questions are metrics that classify questions as number of questions per topic over time and distribution of questions over topics.

Model: The table 10 briefly shows all the metrics found for this group.

Table 10: Model metrics found.

Paper ID	Metric
7	Model
7	Visualization

For this topic, only (Kakkenberg et al., 2024) found metrics. In it, they develop a tool for analyzing code architecture and define some metrics for it. We separated the metrics found into 2 subtopics: Model, which is for metrics that analyze the model itself, and Visualization, which is for metrics focused on the visualization part of the model. For Model, we have metrics such as: number of input and output dependencies, number of screens and entities, file size, and cohesion. For Visualization, we have the colors and dimension of the graph nodes.

Compliance: The table 11 briefly shows all the metrics found for this group.

Table 11: Compliance metrics found.

Paper ID	Metric
4	A set of 20 compliance metrics

(Overeem et al., 2021) brings a study on the coverage of practices for 4 LCDPs (Mendix, OutSystems, Betty Blocks[13] and Pega[14]), it divides the practices into 6 groups: Lifeciclye management, Security, Performance, Observability, Community and Commercial. For Lifecycle management we have: (i) version management (4 practices), (ii) Decoupling API & application (4 practices) e (iii) Update notification (4 practices); Security: (i) Authentication (3 practices),(ii) Authorization (4 practices), (iii) Threat detection & protection (6 practices) and (iv) Encryption (3 practices); Performance: (i) Resource management (4 practices) and (ii) Traffic management (7 practices); Observability: (i) Monitoring (3 practices), (ii) Logging (4 practices) and (iii) Analytics (5 practices); Community: (i) Developer onboarding (4 practices),

[8]https://stackoverflow.com/

[9]https://www.reddit.com/

[10]https://www.reddit.com/r/lowcode/

[11]https://www.reddit.com/r/nocode/

[12]https://www.reddit.com/r/nocodelowcode/

[13]https://www.bettyblocks.com/

[14]https://www.pega.com/

(ii) Support (3 practices), (iii) Documentation (3 practices), (iv) Community Engagement (5 practices) and (v) Portfolio management (3 practices); Commercial: (i) Service-level agreements (4 practices), (ii) Monetization strategy (4 practices) and (iii) Account management (4 practices). This means we have coverage of 81 practices across these 20 metrics.

5 DISCUSSION

This study aims to identify how metrics are being used in the context of low-code software development and which metrics appear most frequently in the literature. Although our search covered publications from 2001 to 2024, all relevant results were published within the last five years, with 2024 alone accounting for 35.3% (Figure 1) of the selected studies. This trend suggests a growing interest in the topic, particularly in recent years, as more researchers and practitioners explore the role of metrics in low-code development.

One notable pattern is the overwhelming reliance on quantitative approaches, with only four studies incorporating qualitative elements (Figure 2). This suggests that research in this field prioritizes numerical evaluation over experiential insights. This may be partially influenced by our study's focus on metrics, which naturally emphasizes quantitative research. However, it may also reflect a broader trend in low-code research, where qualitative evaluations are under explored.

The high proportion of studies categorized as Lessons Learned (58.8%) (Figure 6) reflects a field still in its early stages, where foundational understanding is prioritized over standardized practices. A notable trend is the emphasis on productivity and development efficiency, while usability and qualitative assessments remain underexplored. This suggests a research bias toward measurable outcomes rather than user-centered evaluation.

Among the metric groups, Development Metrics appeared most frequently, reinforcing the emphasis on measuring effort, speed, and output in low-code development. In contrast, Usability Metrics were the least represented, indicating that the end-user experience may be undervalued or assumed as inherent to low-code platforms. This imbalance highlights a potential blind spot in the literature.

Another key observation is the low recurrence of specific metrics across different studies, suggesting a lack of standardization. Without commonly accepted metrics, it becomes difficult to compare findings or establish best practices. This limitation reduces the generalizability of current research and may hinder the integration of low-code development practices in more structured or regulated environments.

Future research should investigate not only the definition and validation of standardized low-code metrics, but also how these metrics are adopted in real-world agile environments, how they influence team performance, and how they align with end-user satisfaction. In particular, mixed-method approaches could help bridge the current gap between quantitative rigor and qualitative insight. Studies exploring the organizational, cultural, and collaborative aspects of metric adoption in low-code teams may also contribute to a more holistic understanding of this field.

6 THREATS TO VALIDITY

While this study provides valuable insights into how metrics are used in low-code software development and that we have followed the standards of the methodology defined in (Kitchenham and Charters, 2007), it is important to acknowledge its limitations.

Internal Validity: While we conducted a rigorous paper selection process, there is always a risk of selection bias. Our inclusion and exclusion criteria may have led to the omission of relevant studies that did not explicitly mention low-code metrics. Additionally, our categorization of metrics was based on our interpretation, which could introduce classification bias.

External Validity: Our findings are limited to studies published in ACM, IEEE, and SpringerLink, potentially excluding relevant research from other sources. While our focus on peer-reviewed publications ensures academic rigor, it may underrepresent industry best practices and informal yet valuable research.

Construct Validity: This study primarily analyzes quantitative research, which may lead to an underrepresentation of qualitative perspectives on how developers perceive and use metrics. Furthermore, the lack of standardized definitions for metrics across studies could introduce inconsistencies in the interpretation of findings.

Reliability: Although our SLR follows established guidelines, the inherent subjectivity in study selection and data extraction could impact reproducibility. To mitigate this, we adhered to a structured process, with multiple authors reviewing the results to ensure consistency.

7 CONCLUSION

This study systematically reviewed recent research on low-code development to understand how metrics are being applied in the software development process and which metrics appear most frequently. Our findings indicate a growing interest in the topic, with papers first appearing in 2020 and a notable increase in 2024. This trend suggests that the academic community is starting to focus on evaluating and improving low-code development practices.

One key observation is the predominance of quantitative approaches, with only a few studies incorporating qualitative elements. This reflects a strong focus on measurable aspects of low-code development, while more subjective factors, such as developer experience and usability, remain underexplored. Additionally, the distribution of research contributions reveals that most studies fall into the Lessons Learned category, suggesting that the field is still in an exploratory phase, with relatively few works proposing practical frameworks or tools.

In terms of specific metric groups, Development Metrics was the most frequent, reinforcing the emphasis on measuring efficiency and productivity in low-code development. Usability Metrics on the other hand, was the group with the least frequency, indicating a gap in research regarding how developers and end-users interact with low-code platforms. Similarly, the lack of standardization in metric usage suggests an opportunity for future research to establish more consistent evaluation methods.

Despite its contributions, this study has limitations, particularly regarding the underrepresentation of qualitative research and the potential exclusion of industry-driven studies. Future work should explore standardized metric definitions, qualitative insights into metric adoption, and industry best practices to complement the findings presented here.

By addressing these gaps, future research can provide a more comprehensive understanding of low-code metrics, ultimately contributing to more effective measurement, better tool development, and improved software quality in low-code environments.

REFERENCES

Al Alamin, M. A., Malakar, S., Uddin, G., Afroz, S., Haider, T. B., and Iqbal, A. (2021). An empirical study of developer discussions on low-code software development challenges. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*, pages 46–57. IEEE.

Alamin, M. A. A., Uddin, G., Malakar, S., Afroz, S., Haider, T., and Iqbal, A. (2023). Developer discussion topics on the adoption and barriers of low code software development platforms. *Empirical software engineering*, 28(1):4.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., et al. (2001). Manifesto for agile software development.

Bexiga, M., Garbatov, S., and Seco, J. C. (2020). Closing the gap between designers and developers in a low code ecosystem. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 1–10.

Bucaioni, A., Cicchetti, A., and Ciccozzi, F. (2022). Modelling in low-code development: a multi-vocal systematic review. *Software and Systems Modeling*, 21(5):1959–1981.

Calçada, A. and Bernardino, J. (2022). Experimental evaluation of low code development, java swing and javascript programming. In *Proceedings of the 26th International Database Engineered Applications Symposium*, pages 103–112.

Curty, S., Härer, F., and Fill, H.-G. (2023). Design of blockchain-based applications using model-driven engineering and low-code/no-code platforms: a structured literature review. *Software and Systems Modeling*, 22(6):1857–1895.

Digital.ai (2024). The 17th state of agile report. https://info.digital.ai/rs/981-LQX-968/images/RE-SA-17th-Annual-State-Of-Agile-Report.pdf?version=0. Accessed: 2025-02-14.

Domingues, R. C., Silva, M. J., and Marinho, M. L. M. (2024). Low-code quality metrics for agile software development. In *Proceedings of the XXIII Brazilian Symposium on Software Quality*, pages 417–425.

Dyba, T., Dingsoyr, T., and Hanssen, G. K. (2007). Applying systematic reviews to diverse study types: An experience report. In *1st Int'l Conference on Empirical Software Engineering and Measurement (ESEM) 2007.*, pages 225–234, Madrid, Spain. IEEE.

Forrester (2014). New development platforms emerge for customer-facing applications. https://www.forrester.com/report/New-Development-Platforms-Emerge-For-CustomerFacing-Applications/RES113411. Accessed: 2025-02-14.

Gartner (2021). Gartner says cloud will be the centerpiece of new digital experiences. https://www.gartner.com/en/newsroom/press-releases/2021-11-10-gartner-says-cloud-will-be-the-centerpiece-of-new-digital-experiences. Accessed: 2025-02-14.

Gartner (2024). Gartner magic quadrant for enterprise low-code application platforms. https://www.gartner.com/en/documents/5844247. Accessed: 2025-02-14.

Gartner (2025). Gartner forecasts worldwide it spending to grow 9.8% in 2025. https://www.gartner.com/en/newsroom/press-releases/2025-01-21-gartner-forecasts-worldwide-

it-spending-to-grow-9-point-8-percent-in-2025. Accessed: 2025-02-14.

Guthardt, T., Kosiol, J., and Hohlfeld, O. (2024). Low-code vs. the developer: An empirical study on the developer experience and efficiency of a no-code platform. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, pages 856–865.

Hagel, N., Hili, N., and Schwab, D. (2024). Turning low-code development platforms into true no-code with llms. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, pages 876–885.

Haputhanthrige, V., Asghar, I., Saleem, S., and Shamim, S. (2024). The impact of a skill-driven model on scrum teams in software projects: A catalyst for digital transformation. *Systems*, 12(5):149.

Ivarsson, M. and Gorschek, T. (2011). A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering*, 16:365–395.

Kakkenberg, R., Rukmono, S. A., Chaudron, M., Gerholt, W., Pinto, M., and de Oliveira, C. R. (2024). Arvisan: an interactive tool for visualisation and analysis of low-code architecture landscapes. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, pages 848–855.

Kedziora, D., Siemon, D., Elshan, E., and Sońta, M. (2024). Towards stability, predictability, and quality of intelligent automation services: Ecit product journey from on-premise to as-a-service. In *Proceedings of the 7th ACM/IEEE International Workshop on Software-intensive Business*, pages 15–23.

Khalajzadeh, H. and Grundy, J. (2024). Accessibility of low-code approaches: A systematic literature review. *Information and Software Technology*, page 107570.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. 2.

Luo, Y., Liang, P., Wang, C., Shahin, M., and Zhan, J. (2021). Characteristics and challenges of low-code development: the practitioners' perspective. In *Proceedings of the 15th ACM/IEEE international symposium on empirical software engineering and measurement (ESEM)*, pages 1–11.

Martins, R., Caldeira, F., Sa, F., Abbasi, M., and Martins, P. (2020). An overview on how to develop a low-code application using outsystems. In *2020 International Conference on Smart Technologies in Computing, Electrical and Electronics (ICSTCEE)*, pages 395–401. IEEE.

Overeem, M., Jansen, S., and Mathijssen, M. (2021). Api management maturity of low-code development platforms. In *International Conference on Business Process Modeling, Development and Support*, pages 380–394. Springer.

Pacheco, J., Garbatov, S., and Goulão, M. (2021). Improving collaboration efficiency between ux/ui designers and developers in a low-code platform. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 138–147. IEEE.

Petersen, K., Feldt, R., Mujtaba, S., and Mattsson, M. (2008). Systematic mapping studies in software engineering. In *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, EASE'08, page 68–77, Swindon, GBR. BCS Learning & Development Ltd.

Prinz, N., Rentrop, C., and Huber, M. (2021). Low-code development platforms-a literature review. In *AMCIS*.

Rokis, K. and Kirikova, M. (2023). Exploring low-code development: a comprehensive literature review. *Complex Systems Informatics and Modeling Quarterly*, (36):68–86.

Sahay, A., Di Ruscio, D., Iovino, L., and Pierantonio, A. (2023). Analyzing business process management capabilities of low-code development platforms. *Software: Practice and Experience*, 53(4):1036–1060.

Sahay, A., Indamutsa, A., Di Ruscio, D., and Pierantonio, A. (2020). Supporting the understanding and comparison of low-code development platforms. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 171–178. IEEE.

Trigo, A., Varajão, J., and Almeida, M. (2022). Low-code versus code-based software development: Which wins the productivity game? *It Professional*, 24(5):61–68.

Varajão, J., Trigo, A., and Almeida, M. (2023). Low-code development productivity: " is winter coming" for code-based technologies? *Queue*, 21(5):87–107.

Wieringa, R., Maiden, N., Mead, N., and Rolland, C. (2006). Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requir. Eng.*, 11:102–107.