Associação Propagadora Esdeva
Centro Universitário Academia (UniAcademia)
Curso de Engenharia de Software
Trabalho de Conclusão de Curso – Artigo

Dívida técnica: estudo das métricas da evolução do código fonte do i-Educar

Gabriel Silva Moreira
Centro Universitário Academia*, Juiz de Fora, MG*
Tassio Ferenzini Martins Sirqueira
Centro Universitário Academia*, Juiz de Fora, MG*

Linha de Pesquisa: *Engenharia de Software*

RESUMO

O presente artigo relata como foi implementado o estudo exploratório sobre a evolução do software i-Educar. O objetivo do estudo é analisar as métricas comparando-as entre si. Na metodologia as métricas foram extraídas do software por meio da ferramenta SonarQube para analisar se houve uma evolução ou decaimento do débito técnico do software ao longo do tempo, para isso, foram analisadas métricas indicadas em RODRIGUES VIEIRA (2014). Como resultado após a análise dessas métricas, notou-se que houve uma evolução na qualidade do código, com uma grande melhora a partir da versão 2.2.0. Constatou-se que, ao longo do desenvolvimento do i-Educar há indícios de melhora na qualidade de código, e que mesmo sem seguir uma evolução gradativa, o software evoluiu em diferentes aspectos de qualidade, principalmente da versão 2.2.0 em diante.

Palavras-chave: Dívida técnica. Engenharia de software. Gerenciamento de débito técnico. Desenvolvimento de software. Métricas de software

ABSTRACT

This article reports on how the exploratory study on the evolution of the i-Educar software was implemented. The objective of the study is to analyze the metrics by comparing them to each other. In the methodology, the metrics were extracted from the software using the SonarQube tool to analyze whether there was an evolution or decay of the technical debit of the software over time, for this, metrics indicated in RODRIGUES VIEIRA (2014) were analyzed. As a result after the analysis of these metrics, it was noted that there was an evolution in the quality of the code, with a great improvement starting with version 2.2.0. It was found that throughout the development of i-Educar there are signs of improvement in code quality, and that even without following a gradual evolution, the software evolved in different aspects of quality, especially from version 2.2.0 onwards.

KEY-WORDS: Technical debt. Software engineering. Technical debt management. Software development. Software metrics.

1 INTRODUÇÃO

De acordo com LAGE, TREVISAN, KALINOWSKI(2020) com o mundo cada vez mais digital e automatizado e com a crescente demanda das empresas por softwares que entregam mais valor e de forma mais ágil, além do desenvolvimento de software, o conhecimento sobre a dívida técnica é desejado para que o software consiga entregar o que foi planejado e sem um grande aumento no custo final, e para que no futuro ele seja simples e fácil para se fazer manutenções e atualizações.

Tem crescido o interesse por pessoas da área de desenvolvimento sobre a dívida técnica no processo de desenvolvimento de software, sobre como diminuir ou evitar gerar mais dívida técnica, diminuir o retrabalho e demonstrar para a área de negócios a importância de gerenciar a dívida técnica e que essa ação pode evitar o aumento do custo do projeto LAGE, TREVISAN, KALINOWSKI(2020).

> *A dívida técnica é similar à dívida financeira, ou seja, exige o pagamento de juros. Esses juros vêm na forma de esforço extra, que devem ser pagos em desenvolvimentos futuros por conta da escolha de um design mais rápido e de baixa qualidade. Nós podemos optar por continuar pagando esses juros ou quitar de uma vez a dívida fazendo uma refatoração, transformando um design de baixa qualidade em um design melhor. Apesar dos custos para saldar a dívida, ganhamos reduzindo os juros no futuro.*
> *(FOWLER, 2009, apud HAZRATI 2009)*

De acordo com Pina(2013), com a dificuldade em gerenciar a dívida técnica no desenvolvimento do software, o time de desenvolvimento muitas vezes sabe que está entregando o software com melhorias e correções que precisam ser feitas antes de gerar um grande risco de que a maneira que o software for crescendo essa dívida vai ficando cada vez mais cara e difícil tecnicamente de ser resolvida.

Em uma pesquisa feita em 2020 pela CISQ nos Estados Unidos foi mostrado que o custo da baixa qualidade de software chega a 2,84 trilhões de dólares, neste estudo foram considerados: Sistemas legados, falhas catastróficas e o custo de projetos problemáticos e cancelados. O que mostra como a qualidade de software pode impactar negativamente no crescimento de uma empresa, já que este valor poderia ser investido em inovação ou em outras áreas da empresa. Para gerenciar os impactos gerados pela dívida técnica, alguns pontos são importantes, como a capacitação da equipe, a identificação e classificação da dívida técnica e um bom gerenciamento da dívida técnica FONSECA (2020).

Para EVALDO JUNIOR(2014), um dos problemas de código duplicado é a manutenção que será necessária em várias partes semelhantes do software, além de aumentar o custo/tempo, também pode criar bugs de software já que a correção terá que ser feita em vários lugares que usam lógica semelhante e algum pode acabar sendo esquecido.

Além desta introdução esse trabalho é dividido da seguinte forma: seção 1 Introdução. Seção 2 Referencial Teórico. Seção 3 Metodologia, apresenta os métodos usados e o estudo de caso. Seção 4 apresenta Resultados e Análise dos dados, do estudo experimental . Seção e 5 Considerações finais, com o resumo das constatações e os impactos dos resultados do software analisado.

2 REFERENCIAL TEÓRICO

A fundamentação teórica foi elaborada com os seguintes tópicos: processo de desenvolvimento de software, o que é dívida técnica, classificação da dívida técnica, gerenciamento da dívida técnica.

2.1 Processo de desenvolvimento de software

De acordo com PERONDI(2014) o processo de desenvolvimento de software ou processo de software pode ser definido como como um passo a passo de ações e tarefas para a criação de um produto de software, para alcançar um objetivo que foi proposto, independente do tamanho, complexidade ou das regras de negócio do projeto. Nesse processo são utilizadas técnicas e ferramentas para escolher as melhores formas de executar o projeto dentro do prazo e com qualidade para atender as demandas daqueles que investiram e daqueles que vão utilizar o software.

Tabela 1 - Atributos essenciais de um bom software

Características do software	Descrição
Manutenibilidade	O software deve ser escrito de maneira a evoluir para atender às necessidades do cliente.
Confiabilidade e segurança	Inclui características tais como confiabilidade, segurança e proteção.
Eficiência	O software não deve desperdiçar recursos do sistema como memória e ciclos de processador.
Aceitabilidade	O software deve ser aceitável para o usuário para o qual foi projetado.

Fonte: SOMMERVILLE(2010)

2.2 O que é dívida técnica

O termo inicialmente criado por CUNNINGHAM (1992), débito técnico ou Technical Debt (TD) no original em inglês, foi usado como metáfora a empréstimos financeiros, que se gerenciados com sabedoria podem ser úteis e ajudar a resolver problemas a curto prazo, porém a médio e longo prazo, aumentam a complexidade de entendimento e manutenção do software. A dívida técnica pode ser

adquirida por falta de boas práticas de desenvolvimento, planejamento ruim, falta de tempo, como resolver demandas urgentes vindas de pessoas da área de negócios, que não tem muito entendimento da área técnica, utilizando dessa metáfora facilita a comunicação entre as áreas e o melhor entendimento de todos envolvidos no projeto.

Para PINA (2013), adquirir uma dívida técnica pode trazer benefícios a curto prazo, como redução do tempo de desenvolvimento, mas pode influenciar numa menor qualidade de código e um aumento dos custos a longo prazo.

Para PERONDI (2014), a dívida técnica geralmente acontece por fatores como a falta de tempo, custo de ferramentas e também por falta de boas práticas, o que geralmente acontece quando as equipes buscam resolver algum impeditivo e resolver problemas de forma mais rápida com soluções que não usam as melhores práticas. A dívida técnica pode ser um impeditivo para o software evoluir de uma forma mais eficiente, o que gera uma menor produtividade e um maior custo de manutenção. Uma pesquisa feita em 2010 mostrou que o valor do débito técnico era de aproximadamente de 500 bilhões de dólares, podendo chegar a 1 trilhão de dólares em 2015.

2.2.1 Classificação da dívida técnica

McConnell(2007) classificou a dívida técnica em duas categorias: sem intenção e intencional. Se for intencional tem outras duas subcategorias: dívida de curto e a longo prazo.

a) Dívida técnica sem intenção

De acordo com Pina(2013), a dívida técnica sem intenção ocorre por falta de experiência ou descuido da equipe, uma arquitetura mal pensada, ou uma decisão errada ao implementar uma nova funcionalidade, ou ao adicionar um pacote/biblioteca de terceiros sem conhecer a dívida que ele já possui. Por ser sem intenção, tem grandes chances de permanecer invisível e ficar sem o gerenciamento necessário.

b) Dívida técnica intencional

De acordo com Pina (2013), a dívida técnica intencional ocorre com conhecimento da equipe, sabendo que uma ação, construção de software ou funcionalidades ou importação de outras funcionalidades de terceiros vai poupar esforço a curto prazo mas que no futuro será gasto um esforço maior para a manutenção do software. A dívida intencional normalmente é definida em curto ou longo prazo, que indica em quanto tempo a dívida será cobrada.

McConnell(2008), explica da seguinte forma a dívida técnica intencional: “Não tivemos tempo para escrever todos os testes unitários para o código que escrevemos nos últimos 2 meses do projeto. Vamos corrigir esses testes após o lançamento” (tradução livre).

c) Dívida técnica a curto prazo

De acordo com McConnell (2007) é a dívida que causa pouco impacto e que vai ser paga em pouco tempo, geralmente são adquiridas por falta de tempo ou prazos apertados para a entrega do artefato ou de um software completo, um exemplo que costuma entrar nessa dívida é a cobertura de testes do software.

d) Dívida técnica a longo prazo

De acordo com McConnell (2007) são planejadas com antecedência, são adquiridas para necessidades de grande impacto, como uma solução para um grande problema de negócio, priorização de outras funcionalidades ou até para correção de bugs, um exemplo que se enquadra nessa dívida é a refatoração de código, que pode acontecer depois de o software foi entregue.

2.3 Gerenciamento da dívida técnica

De acordo com Ribeiro e Spínola (2016) durante o processo de desenvolvimento de software, devido às várias situações, contratempos e surpresas que podem ocorrer, fazer novas dívidas pode ser algo necessário para o andamento do projeto, desde que todos estejam cientes da dívida e do esforço que será demandado no futuro para o pagamento desta dívida. Já a dívida não gerenciada pode gerar mais custos ao projeto, já que o software vai se tornando mais difícil para fazer sua evolução e manutenção, sendo necessário mais tempo e consequentemente mais dinheiro. Por isso o gerenciamento da dívida se torna tão importante nesse processo, sendo um dos pilares para o sucesso do projeto de software.

De acordo com Ribeiro e Spínola (2016) para o gerenciamento da dívida técnica é necessário identificar, medir e monitorar a dívida técnica. Com isso é possível melhorar a tomada de decisão para saber se a equipe deve ou não trabalhar nessa dívida e se sim qual a melhor forma de fazer o pagamento da dívida técnica.

Segundo PINA (2013) como o débito técnico tem grande relação com o débito econômico, isso possibilita que pessoas que não são do time de desenvolvimento como consultores, gerentes e diretores entendam a necessidade do gerenciamento do débito e possam acompanhar as dificuldades geradas pelo débito técnico, dando mais importância ao seu pagamento.

LAGE, TREVISAN, KALINOWSKI(2020) Mostra como é importante reunir evidências sobre o débito técnico para mostrar o seu comportamento e entender como a equipe tem percebido a evolução do software nesse aspecto para aprimorar processos e ferramentas existentes.

2.4 MANUTENÇÃO E EVOLUÇÃO DE SOFTWARE

Para CAVALCANTI JÚNIOR(2012) a manutenção e evolução de software possuem muitas definições, mas possuem 2 aspectos principais: que a manutenção trata de correções e evoluções do produto de software. A definição de Lehman do que é manutenção de software é de que qualquer alteração feita após a primeira instalação

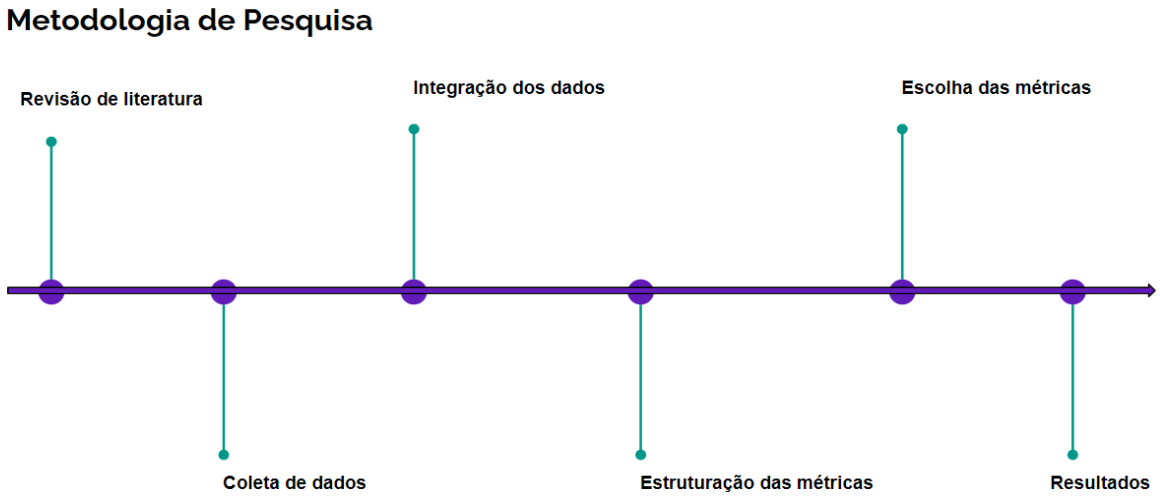
do programa para correção de erros ou alterações feitas por mudanças de ambiente que demandem a adaptação do software.

As intervenções em manutenção são divididas em dois grupos: ações de correção e ações de evolução de software. As ações de correção podem ser corretivas e preventivas, as ações corretivas são as que agem sobre alguma falha encontrada no ambiente de produção, ações preventivas são consideradas proativas, pois corrigem os problemas antes que eles ocorram. As ações de evolução podem ser adaptativas e perfectivas, as adaptativas são adequações do produto a mudanças ambientais como por exemplo, mudar o software para uma nova plataforma, as ações perfectivas são melhorias funcionais, de artefatos, organização ou desempenho do software CAVALCANTI JÚNIOR(2012).

Segundo GABRIEL(2018) manutenção de software é o processo de melhorias e correções de um software após estar disponível em produção. A vida do software não termina após sua implantação, para ele permanecer útil ao longo do tempo é necessário investir em manutenção.

3 METODOLOGIA

Figura 1: Metodologia de pesquisa



Fonte: Elaboração própria.

Na figura 1 é possível observar como foi a evolução do projeto, começando pela revisão de literatura, apresentada na Seção 2, depois na Seção 3, foi feita uma coleta de dados com a ferramenta SonarQube onde foram analisadas 96 versões do versões do software i-Educar e suas métricas foram adicionadas numa planilha. Tais métricas foram integradas e organizadas como é possível ler na seção 3. Na estruturação das métricas foram selecionadas dados específicos para a análise de

débito técnico, como Technical_Deb, Comment Lines, Cyclomatic_Complexity, Code_Smells, Duplicated_Blocks.

3.1 Métricas selecionadas

3.1.1 Technical Debt

O débito técnico será o tempo estimado pelo SonarQube para correção das issues e Code Smells, o tempo pode ser estimado em minutos, horas e dias, sendo que num dia são consideradas 8 horas de acordo com a documentação da plataforma.

3.1.2 Comment

O percentual de comentários é uma métrica que permite avaliar o quão fácil é de entender um código e por consequência evoluí-lo CAVALCANTI JÚNIOR(2012).

3.1.3 Cyclomatic Complexity

De acordo com MATOLA(2020) é a complexidade ciclomática é a quantidade de caminhos independentes que um software pode seguir. Está relacionada ao número de testes que devem ser feitos, quanto maior a complexidade mais testes são necessários.

3.1.4 Code Smells

Segundo Pina(2013) code smells são problemas relacionados com orientação a objetos e outros problemas que envolvem conjuntos de linhas de código e que podem ser resolvidos com refatoração e estão fortemente ligados a manutenção e reuso, além de confiabilidade, segurança e eficiência.

3.1.5 Duplicated Blocks

Código duplicado é o problema mais comum apontado na coleção de más práticas. Além de gerar um aumento na base de código também prejudica a manutenção e o tempo de desenvolvimento pois será necessário alterar vários trechos parecidos no código EVALDO JUNIOR(2014).

3.2 Software analisado: i-Educar

O i-Educar foi selecionado através de resultados de uma busca por softwares livres feita em um buscador online e por ter um repositório público de fácil acesso.

Como informado no próprio site, o [1]i-Educar é um software para gestão escolar, a escolha deste para a análise feita se deu por ser um software livre e por ter seu repositório público na internet. Ele também é usado por prefeituras, como as

[1] i-Educar. Disponível em: https://ieducar.org/index.html. Acessado em 08 de Dezembro de 2022

prefeituras de Duque de Caxias, Botucatu, Criciúma, Balneário Camboriú, Monte Alegre, Paragominas e São Miguel dos Campos, no site também temos a informação de que é usado por mais de 80 municípios e atende mais de 2050 escolas, o que gera uma credibilidade maior para este artigo.

3.2 Coleta de dados

Para este estudo foram analisadas 96 versões do i-Educar, um software livre para gestão escolar. Nas versões, desde a primeira versão 2.0.0 até a última versão estável 2.7.4 (14 de Julho de 2022) utilizamos a ferramenta SonarQube na versão 9.4.0.54424.

De acordo com RODRIGUES VIEIRA (2014) o [2]SonarQube é uma ferramenta para o gerenciamento da qualidade do software para detectar bugs, vulnerabilidades, code smells(mau cheiro), comentários, duplicações, testes unitários, entre outras métricas. É uma ferramenta web que possui uma interface que nos permite a visualização das métricas, e pode ser usada em diversas linguagens, incluindo o PHP e JavaScript, que são as linguagens principais do i-Educar.

Passos realizados na coleta de dados:

1. Baixar o código-fonte de cada versão;
2. Analisar do código fonte através do SonarQube;
3. Converter dados de relatório em uma planilha.

Os dados estão disponíveis em: https://github.com/MoreiraGabriel/metricas-iEducar-sonar

3.3 QUESTÕES DE PESQUISAS

Neste trabalho serão analisados as métricas: Technical Debt(Débito Técnico), que será a principal métrica para análise, e as outras métricas que fazem parte do eixo de qualidade: Comment(percentual de comentários), Cyclomatic Complexity(Complexidade Ciclomática), Duplicate Blocks(quantidade de blocos duplicados) de acordo com RODRIGUES VIEIRA (2014) e PINA (2013) para avaliarmos a evolução do i-Educar ao longo do tempo. O parâmetro Cobertura dos testes (Coverage) que seria utilizado foi excluído da análise porque não foram encontrados testes no i-Educar pelo SonarQube.

Foram definidas as seguintes frentes de pesquisa para ser feita a análise do código fonte do i-Educar. Foram definidos como base de pesquisa um conjunto de dados principal(PR) e um conjunto de dados secundários(SQ). Também foram definidas hipóteses H0 e H1.

H0 (hipótese nula): As amostras apresentam distribuição normal.

[2] SonarQube. Disponível em: https://www.sonarqube.org. Acessado em 28 de novembro de 2022

H1 (hipótese alternativa): As amostras não apresentam distribuição normal.

- **PR - O débito técnico do i-Educar mudou ao longo do tempo?**

  H0: O débito técnico do i-Educar permaneceu ao longo do tempo.
  H1: O débito técnico do i-Educar não permaneceu estável / aumentou ao longo do tempo.

Dúvidas secundárias:

SQ1 - O percentual de comentários do i-Educar mudou ao longo do tempo?

H0: O percentual de comentários permaneceu o mesmo ao longo do tempo.

H1: O percentual de comentários não permaneceu o mesmo ao longo do tempo.

SQ2 - **A complexidade do i-Educar mudou ao longo do tempo?**

H0: A complexidade ciclomática permaneceu a mesma ao longo do tempo.

H1: A complexidade ciclomática não permaneceu a mesma ao longo do tempo.

SQ3 - O número de Code Smells do i-Educar mudou ao longo do tempo?

H0: O número de Code Smells permaneceu ao longo do tempo.

H1: O número de Code Smells não permaneceu ao longo do tempo.

SQ4 - O número de blocos duplicados mudou ao longo do tempo?

H0: O número de blocos duplicados permaneceu ao longo do tempo.

H1: O número de blocos duplicados não permaneceu ao longo do tempo.

4 Resultados

Durante o estudo a análise do débito técnico do software foram levantados os dados presentes na tabela 2, que traz um resumo das métricas levantadas:

Tabela 2 - Estatística Descritiva

Métrica	Min	1° Qu.	Mediana	Média	3° Qu	Max
Technical Debt	282	313	325	377	453	504
Comments_%	6,90	8,30	9,50	10,34	11,90	15,9

Cyclomatic Complexity	33199	34145	35057	38913	45450	46726
Code Smells	6296	8086	8256	9219	10571	13015
Duplicated Blocks	4560	5646	5908	7194	9086	10323

Fonte: Elaboração Própria.

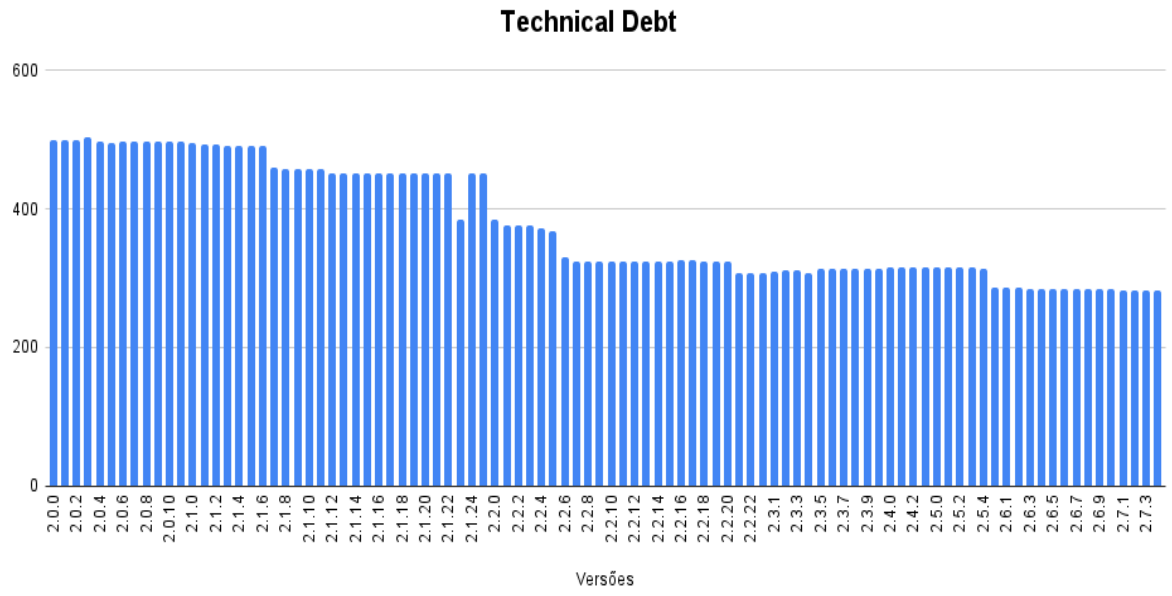
Na tabela 2 temos as estatísticas das métricas que usamos para as análises, que traz no **Technical Debt** uma média de 377 dias, com valor mínimo de 282 dias e maior valor com 504 dias, na métrica **Comments** temos um valor mínimo de 6,9%, uma média de 10,34% e o valor máximo com 15,9%, com a métrica **Cyclomatic Complexity** temos um valor mínimo de 33.199 com média 38913 e valor máximo de 46.726, em **Code Smells** temos um valor mínimo de 6926 e média de 9219 e com valor máximo de 13.015, e **Duplicated Blocks** que tem valor mínimo de 4560, com média de 7194 e valor máximo de 10.323. Esses dados nos mostram que há indícios de uma melhora dessas métricas que como veremos na seção 4 deste artigo análises mais aprofundadas e com imagens que mostram a evolução das métricas a cada versão do software.

De acordo com SIRQUEIRA et al (2020) que seguindo a engenharia de software experimental, explica que o objetivo de um estudo experimental é a coleta de dados em um ambiente controlado para confirmar ou negar uma hipótese. Foram selecionadas métricas como os testes de Kruskal-Wallis que é um teste não paramétrico para análise de variância que baseia-se na substituição dos valores por seus rankings no conjunto de todos os valores e Kolmogorov-Smirnov que possibilita a avaliação quanto às semelhanças entre a distribuição de duas amostras e também pode indicar a similaridade na distribuição de uma amostra em relação a uma distribuição clássica. Para maior clareza, a aceitação de hipóteses nulas foi utilizado o nível de significância de 5%.

PR - O débito técnico do i-Educar mudou ao longo do tempo?

Neste subtópico será discutida a análise de dados sobre a métrica débito técnico. Aplicando o teste de K-S vemos que não temos grandes variações relacionadas ao débito técnico, mesmo o débito diminuindo ao longo das versões, como podemos observar na figura 2, saindo de 499 dias na primeira análise na versão 2.0.0 e chegando à marca de 283 dias na versão 2.7.4, como mostra a figura 2. Ao executar o teste One-sample Kolmogorov-Smirnov (K-S) obtivemos um valor de P = 2.174e-06 mostrando que os dados não resultaram em uma distribuição normal entre as versões, e no teste de Kruskal-Wallis o valor de P = 0.4976 mostrando que os grupos são semelhantes.

Figura 2:Evolução Technical Debt



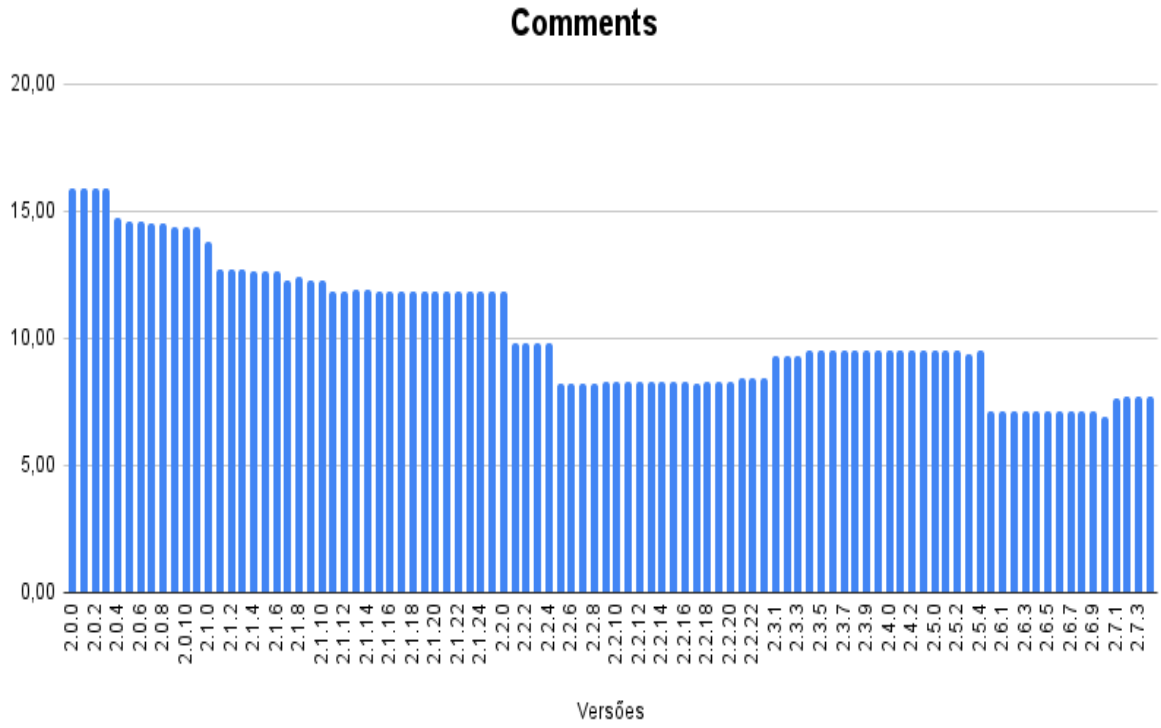
Fonte: Elaboração própria.

SQ1 - O percentual de comentários do i-Educar mudou ao longo do tempo?

Este subtópico irá apresentar a análise do percentual de comentários nas 96 versões analisadas. Para JÚNIOR e OLIVEIRA (2012), essa métrica ajuda a mostrar a dificuldade de entender e evoluir o código, que deve girar em torno de 30%.

Como podemos observar na figura 3, o percentual de comentários diminuiu consideravelmente desde sua primeira versão. Ao longo do desenvolvimento da aplicação, o percentual de comentários no código diminuiu de forma linear, apesar de uma queda mais significativa no percentual de comentários entre as versões 2.2.0 e 2.2.1 e um aumento nas últimas versões da 2.7.0 até 2.7.4, as análises indicam uma distribuição normal dos dados, porém para esclarecer foram realizados os testes Kolmogorov-Smirnov e Kruskal-Wallis. De acordo com o teste de One-sample Kolmogorov-Smirnov (K-S) obtivemos o valor de p = 0.5114, ou seja, apresenta distribuição normal dos dados, e no teste Kruskal-Wallis(K-W) no qual obtivemos o valor de p = 0.5305, mostra que os grupos são semelhantes.

Figura 3:Evolução Comments



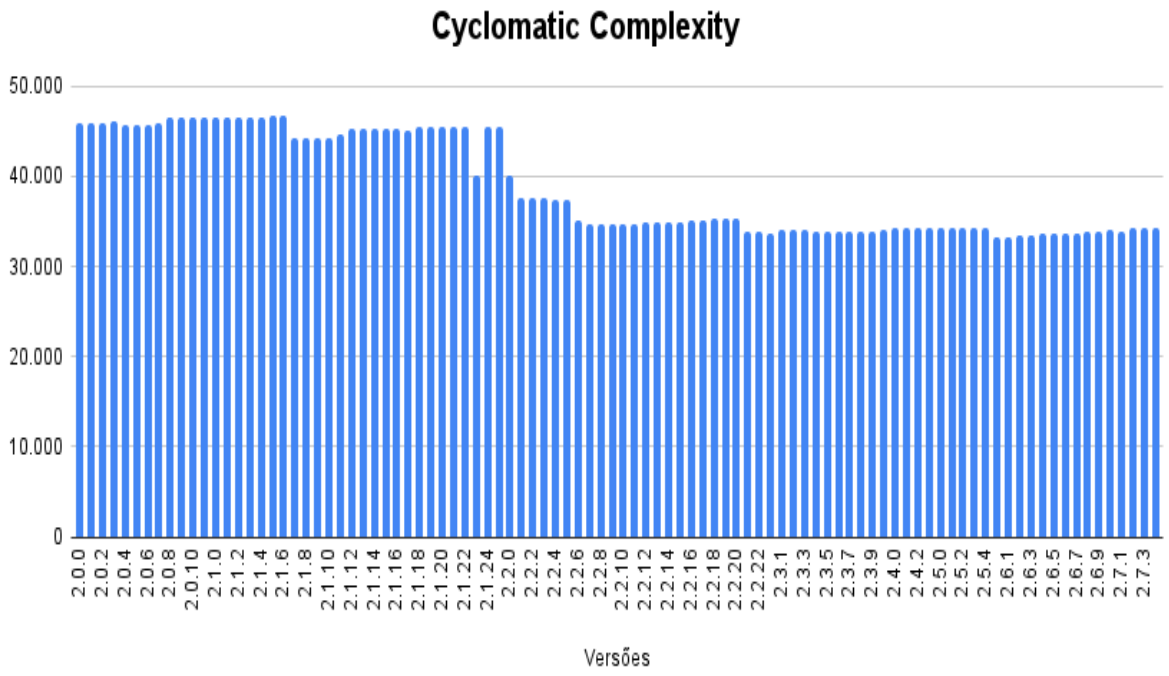
Fonte: Elaboração própria.

A complexidade ciclomática do i-Educar mudou ao longo do tempo?

Segundo (PERLIS), outra métrica que ajuda a rastrear a complexidade do software é Mc-Cabe's cyclomatic complexity. A equação, CC = E - N + P, onde E representa o número de arestas, N o número de nós, e P o número de componentes conectados seja a função analisada como um gráfico. Esta métrica é útil para medir a complexidade de construir testes de unidade em um determinado código, pois mapeia todos os caminhos possíveis que o software pode seguir. Como podemos observar na figura 4, houveram algumas versões onde a complexidade ciclomática ficou maior mostrando dificuldade de manutenção como na versão 2.2.16 e entre as versões 2.2.20, mas voltando a cair a partir da versão 2.2.21.

Aplicando o teste One-sample Kolmogorov-Smirnov (K-S) foi encontrado o valor de p = 4.222e-07, resultado menor que 0,05, essa informação mostra que o código não teve uma evolução constante nesse aspecto gerando variações tanto positivas quanto negativas a cada nova versão. Mas o teste de Kruskal-Wallis mostra que os grupos são semelhantes, com o valor de p = 0.5114.

Figura 4: Evolução Cyclomatic Complexity



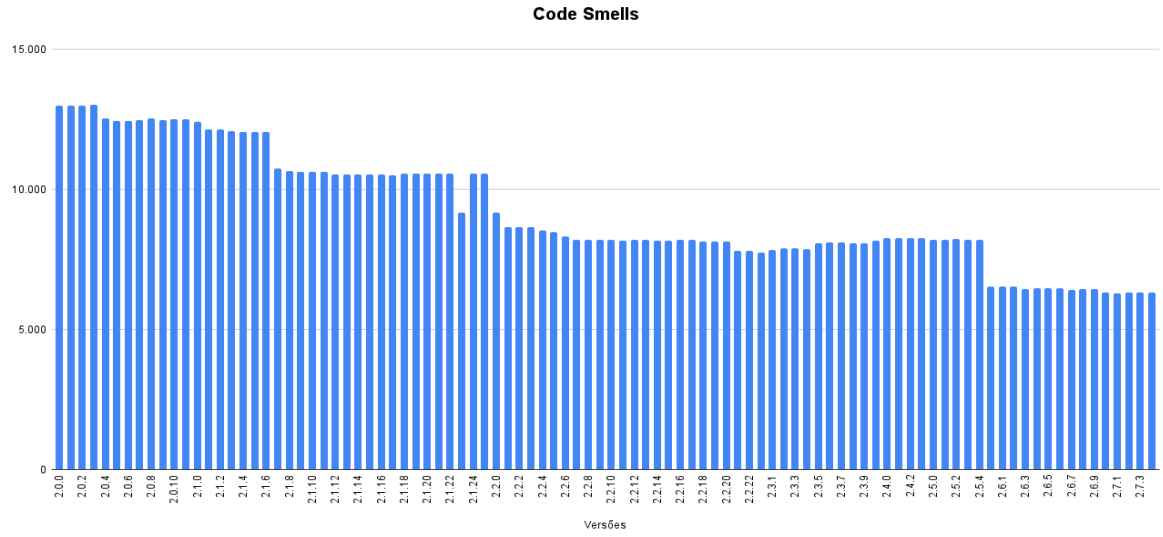
Fonte: Elaboração própria.

SQ3 - O número de Code Smells do i-Educar mudou ao longo do tempo?

De acordo com PINA (2013), Code Smell está fortemente ligado ao reuso e a manutenção do código, além disso é muito utilizado para critérios de confiabilidade, segurança e eficiência.

Como podemos observar na figura 5, apesar de estar diminuindo, houveram algumas versões onde tivemos um aumento de code smells, entre as versões 2.3.5 e 2.5.4, mas voltando a cair a partir da versão 2.6.0. Aplicando o teste One-sample Kolmogorov-Smirnov (K-S) foi encontrado o valor de p = 0.0003788, resultado menor que nível de significância de 5%, o que mostra que mesmo com a quantidade de code smells caindo, o código não teve uma evolução constante nesse aspecto gerando variações negativas nas versões citadas anteriormente, gerando uma distribuição não normal. Mas o teste de Kruskal-Wallis mostra que os grupos são semelhantes, com o valor de p = 0.4993.

Figura 5: Evolução Code Smells

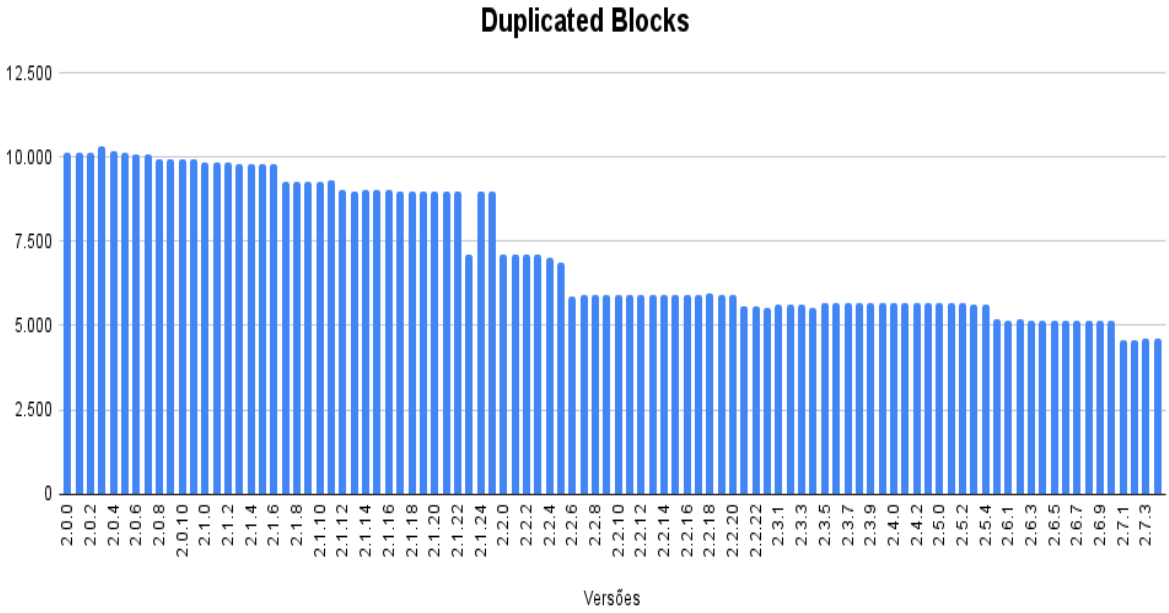


Fonte: Elaboração própria.

SQ4 - O número de blocos duplicados mudou ao longo do tempo?

Como mostrado na figura 6, de estar diminuindo de versão para versão, de acordo com o teste Kolmogorov-Smirnov (K-S) foi encontrado o valor de p = 7.419e-07, que indica que os dados não seguem uma distribuição normal e no teste de Kruskal-Wallis temos o valor de p = 0.4921 que indica que os grupos são semelhantes

Figura 6: Evolução Duplicated Blocks



Fonte: Elaboração própria.

4.1 Análise dos Resultados

Analisando o débito técnico, podemos perceber que ele vem diminuindo ao longo das versões, caindo de 499 dias na versão 2.0.0 para 283 dias na versão 2.7.4, se mantendo estável nas últimas 3 versões, mostra que o software está evoluindo nesse quesito e que estão tendo cuidado para não aumentar quando lançam novas versões.

Na parte de compreensão do código, utilizamos as métricas de porcentagem de comentários e complexidade ciclomática CC. Apesar de instáveis, com as métricas aumentando e diminuindo durante o ciclo de vida analisado, podemos observar uma queda geral tanto dos comentários quanto da CC da primeira 2.0.0 até a última versão 2.7.4. Com a porcentagem de comentários caindo de 15.9% para 7.7% e com nível CC caindo de 45.922 para 34.190, o que mostra que uma boa evolução nesse quesito.

Na análise da qualidade do código usaremos as métricas de code smells e blocos duplicados. Obtivemos uma queda significativa no número de blocos duplicados saindo de 10.133 para 4.616, também obtivemos uma boa queda no número de code smells caindo de 12.996 para 6.320, mostrando uma grande melhora na qualidade de código do software.

5 Considerações finais

Este trabalho teve como objetivo analisar de maneira exploratória, algumas características de débito técnico e mostrar a importância do gerenciamento do débito técnico e como ele influencia na qualidade do software, através da identificação e documentação de métricas, que foram extraídos do i-Educar (um software livre para gestão escolar) pelo SonarQube.

Durante a seção 4 analisamos as métricas para identificarmos a evolução do débito técnico durante o desenvolvimento do software I-Educar. Além do próprio débito técnico, utilizamos outras métricas do eixo de qualidade como percentual de comentários, complexidade ciclomática, code smell e quantidade de blocos duplicados.

Ao analisar a evolução do software i-Educar a partir das métricas extraídas, podemos ver que o débito técnico vem diminuindo ao longo das versões, podendo haver indícios de uma melhora na qualidade e uma diminuição da complexidade do código. Com as métricas levantadas nesse artigo podemos afirmar que o mesmo trouxe uma cuidadosa análise do débito técnico do i-Educar e que é importante o uso de métricas para análise e gerenciamento do débito técnico.

Como trabalho futuro fica como sugestão usar outras ferramentas existentes para análise de código como PHP Metrics para analisar os dados gerados por essa ferramenta, gerando métricas e indicadores que podem ser usados por equipes e gerentes de projeto em reuniões para avaliar esses dados e para serem usados para uma evolução do produto.

Por fim um estudo analisando outras métricas de outro software, como a métrica de cobertura de testes que não foi encontrada no iEducar pelo SonarQube, e analisar o impacto dessas outras métricas na evolução do software.

REFERÊNCIAS

CAVALCANTI JUNIOR, Antonio Luiz de Oliveira. **Métricas como Ferramenta de Auxílio para o Gerenciamento de Dívida Técnica em Produtos de Software**. 2012. [*s. l.*], 2012.

CUNNINGHAM, Ward. The WyCash Portfolio Management System. [*s. l.*], 1992.

FONSECA, Luan. **Débito Técnico: por que isso vai estragar teu software | Labcodes**. [*S. l.*], [*s. d.*]. Disponível em: https://www.labcodes.com.br. Acesso em: 13 nov. 2022.

FOWLER, Martin. **Bliki: TechnicalDebtQuadrant**. [*S. l.*], [*s. d.*]. Disponível em: https://martinfowler.com/bliki/TechnicalDebtQuadrant.html. Acesso em: 29 ago. 2022.

GABRIEL. **Manutenção de Software - Definição e melhores práticas**. *Em*: OPUS SOFTWARE. 8 nov. 2018. Disponível em: https://www.opus-software.com.br/manutencao-de-software-definicao/. Acesso em: 30 nov. 2022.

HAZRATI, Vikas. **Analisando a Dívida Técnica**. [*S. l.*], 2009. Disponível em: https://www.infoq.com/br/news/2009/10/dissecting-technical-debt/. Acesso em: 29 ago. 2022.

JUNIOR, Evaldo. **Os problemas do Código Duplicado**. [*S. l.*], 2014. Disponível em: http://blog.evaldojunior.com.br/desenvolvimento/melhores%20pr%C3%A1ticas/2014/01/23/os-problemas-do-codigo-duplicado.html. Acesso em: 1 nov. 2022.

LAGE, Luiz Carlos Da Fonseca; TREVISAN, Daniela G.; KALINOWSKI, Marcos. **Dívida Técnica de Usabilidade em Projetos de Software: Um Estudo de Casos Múltiplos**. 2020. [*s. l.*], 2020. Disponível em: https://sol.sbc.org.br/journals/index.php/isys/article/view/753. Acesso em: 5 jun. 2022.

MATOLA, Rodrigo. **Complexidade ciclomática: Por que QAs (e Devs) devem se preocupar com isso | LinkedIn**. [*S. l.*], 2020. Disponível em: https://www.linkedin.com/pulse/complexidade-ciclom%C3%A1tica-por-que-qas-e-devs-devem-se-preocupar/. Acesso em: 28 nov. 2022.

MCCONNELL, Steve. **Managing Technical Debt**. [*s. l.*], 2008.

PERLIS, Alan. **McCabe's Cyclomatic Complexity | Software Quality Metric | Quality Assurance | Complex System | Complex | Software Engineering**. [*S. l.*], [*s. d.*]. Disponível em: http://www.chambers.com.au/glossary/mc_cabe_cyclomatic_complexity.php. Acesso em: 30 nov. 2022.

PERONDI, Ramon. **UNIVERSIDADE FEDERAL DA FRONTEIRA SUL CAMPUS CHAPECÓ CURSO DE CIÊNCIA DA COMPUTAÇÃO**. 2014. [*s. l.*], 2014.

PINA, Diogo de Jesus. **Dívida Técnica: Identificando, Medindo e Monitorando**. 2013. [*s. l.*], 2013.

RIBEIRO, Leilane Ferreira; SPÍNOLA, Rodrigo Oliveira. Um Survey sobre a Pertinência e Relevância de Critérios de Decisão para Apoiar o Gerenciamento de Itens de Dívida Técnica. *Em*: XV SIMPÓSIO BRASILEIRO DE QUALIDADE DE SOFTWARE, 2016, Brasil. **Anais do XV Simpósio Brasileiro de Qualidade de Software (SBQS 2016)**. Brasil: Sociedade Brasileira de Computação - SBC, 2016. p. 256–270. Disponível em: https://sol.sbc.org.br/index.php/sbqs/article/view/15139. Acesso em: 5 jun. 2022.

RODRIGUES VIEIRA, Igor. **Um estudo experimental para avaliar características da dívida técnica em produtos de código aberto**. *Em*: , 2014. **Anais [...]**. [*S. l.: s. n.*], 2014. p. 135–148. Disponível em: https://ww2.inf.ufg.br/~erigo/4/o/anais-ii-erigo-2014.pdf#page=135.

SIRQUEIRA, Tassio *et al.* **Application of Statistical Methods in Software Engineering: Theory and Practice**. [*S. l.: s. n.*], 2020.

SOMMERVILLE, Ian. **Engenharia de Software**. [*s. l.*], n. 9ª edição, 2010.