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

Uma Análise das Métricas da Evolução do Código Fonte do Joomla

Luiz Fernando Benini do Amaral
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
Ao longo do ciclo de vida de um software, diferentes ações podem melhorar ou deteriorar a qualidade do código fonte. Neste artigo, espera-se mostrar com uma análise desde sua primeira versão, se houve melhora ou degradação da aplicação, analisando as métricas levantadas a cada versão e comparando-as entre si. Desta forma, será possível identificar os atores envolvidos na melhoria ou degradação do código fonte do Joomla. **Objetivo**: O objetivo do projeto é identificar métricas e indicadores que mostram a melhoria ou degradação do código fonte da aplicação analisando diferentes versões e comparando-as entre si fazendo comparando os resultados disponibilizando insumo para a comparação da melhor qualidade de código fonte entre os principais CMS, Joomla, WordPress e Dupral. **Método**: Para visualizar a qualidade do código do Joomla, foi realizado um estudo analisando as diferentes versões do aplicativo ao longo de seu desenvolvimento. Foi realizado um estudo exploratório para caracterizar a qualidade do código, a partir de um conjunto de versões. **Resultado**: Durante a análise do Joomla foi identificado que durante o desenvolvimento das versões da aplicação houve uma grande perda na qualidade geral do código, com pouco ganho na versão 2.5 que logo se perdeu nas versões sucessoras. **Conclusão**: Durante a análise foi observado que ao longo das versões do Joomla houve grandes perdas na qualidade do código que são indicadas pelas métricas extraídas na seção 3. Ao longo do desenvolvimento do Joomla apenas uma versão apresentou melhora real na qualidade do código fonte em diferentes aspectos, a versão 2.5.

Palavras-chave: Joomla, código aberto, qualidade de software, software Métricas, qualidade do código fonte.

ABSTRACT

Throughout a software's lifecycle, different actions can improve or deteriorate the quality of the source code. In this article, it is expected to show with an analysis since its first version, if there is improvement or degradation of the application, analyzing how metrics raised in each version and comparing them with each other. In this way, it will be possible to identify the actors involved in the improvement or degradation of the Joomla source code. **Objective**: The objective of the project is to identify metrics and indicators that show an improvement or degradation of the application code source by analyzing different versions and comparing them to each other, comparing the results, providing input for a comparison of the best source code quality among the main CMS: Joomla, WordPress and Dupral. **Method**: To visualize the quality of the Joomla code, a study was carried out analyzing the different versions of the application throughout its development. An exploratory study was carried out to characterize the quality of the code, based on a set of versions. **Result**: During an analysis of Joomla, it was identified that during the development of the application versions there was a big loss in the overall quality of the code, with little gain in version 2.5 that the logo was lost in the successor versions. **Conclusion**: During an analysis it was observed that over the Joomla versions there were large losses in code quality which are indicated by the metrics extracted in section 3. Throughout the development of Joomla only one version showed real improvement in source code quality in different aspects, a 2.5 version.

1. INTRODUÇÃO

De acordo com WILLIANS et al (2021), ao estudar a evolução da qualidade do software, é necessário adotar uma abordagem analítica do produto durante sua existência. Um código fonte eficaz nos permite analisar métricas ao longo das versões ajudando na visibilidade de novas melhorias e organização das versões. Analisando métricas é possível identificar tanto melhorias quanto depreciação do código ao longo do tempo que pode indicar a qualidade ou más práticas do código respectivamente.

De acordo com SIQUEIRA et al (2021), para que tenhamos um bom nível de qualidade, são necessários alguns aspectos dos dados a considerar como a fonte, sua validade e atualidade, definindo quatro estágios que compõem a avaliação do ciclo de vida.

O estudo tem o intuito de apresentar pontos que indicam a qualidade do código do Joomla mostrando o impacto no ciclo de vida do mesmo a cada versão, trazendo métricas que demonstram esse impacto no desempenho, manutenção, complexidade da aplicação, quantidade de bugs, classes e a forma que essas métricas alteram o mesmo seja positiva ou

negativamente. Em outras palavras um software de qualidade tem um código fácil de entender, fácil de testar, sem blocos duplicados e com débito técnico próximo ou igual a zero horas indicando assim que o código não precisa de ajustes. Analisando esses fatores é esperado que esse artigo sirva de insumo para a comparação da qualidade do código entre os principais CMS do mercado: Joomla, WordPress e Dupral.

Este trabalho está organizado nas seguintes seções. Na seção 1 a introdução, a seção 2 apresenta o trabalho e os processos realizados no estudo. A seção 3 relata a execução do estudo e os resultados da análise experimental e a seção 4 serão apresentadas as considerações finais.

2. METODOLOGIA

Na figura 1 a seguir é possível ver a forma que abordamos a análise do estudo. A abordagem inicial foi revisar a literatura e apresentar os resultados na Seção 3. Essa etapa consiste na coleta e organização dos dados da análise. Para essa etapa foram utilizadas duas ferramentas: [1]SonarQube e [2]PHPMetrics sem calibração usando as configurações padrão das ferramentas. Com isso é possível gerar as métricas de cada uma das 122 versões analisadas.

Na primeira parte da pesquisa (2.1) será explicado como os dados foram coletados para análise, quais ferramentas e versões foram usadas representando a fase 1 do projeto. No subtópico seguinte (2.2) serão abordadas as questões importantes da pesquisa como o intuito da mesma e quais foram as métricas e análises feitas dentro do contexto deste artigo. Na terceira e última parte (2.3) será apresentado o resultado dos testes realizados em cada agrupamento de dados e uma análise de cada um dos resultados mostrando em cada contexto como as versões se comportaram.

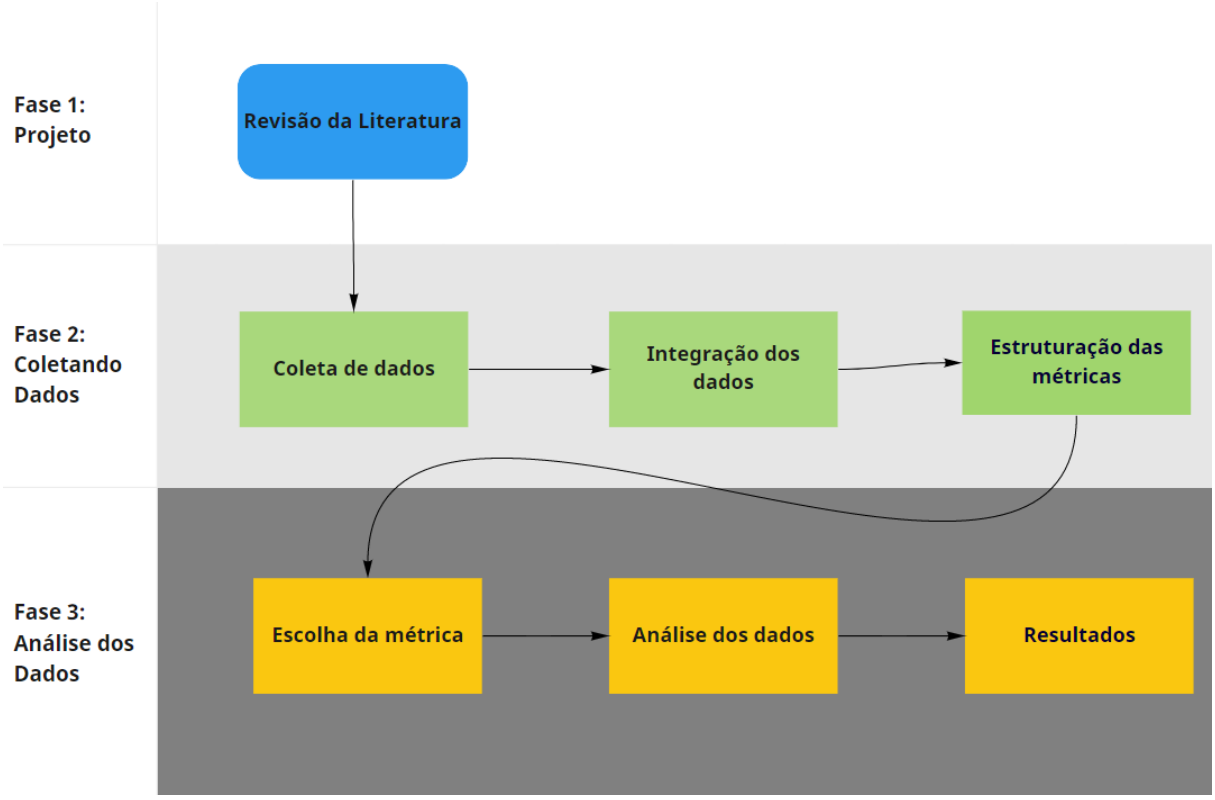
Na figura 1 é possível observar as fases que consistem no projeto como um todo. Na fase um foi revisada a literatura identificando artigos relacionados a análise do ciclo de vida de um software indicados na seção de bibliografia. Na fase 2 estarão presentes três etapas que representam a fase de coleta de dados. Na coleta de dados cada versão do Joomla foi mapeada pelas ferramentas escolhidas e incluídas em uma planilha, os dados foram integrados e as métricas foram estruturadas como é possível observar na subseção 2.3. Na fase 3 foram selecionadas as métricas como os testes de Kruskal-Wallis e Kolmogorov-Smirnov seguindo as orientações de análise presentes em AMORIM et al (2017). Após a

[1] SonarQube. Disponível em: https://www.sonarqube.org. Acessado em 25 de novembro de 2021

[2] PHPMetrics. Disponível em: https://phpmetrics.org. Acessado em 25 de novembro de 2021

escolha das métricas as mesmas foram inseridas em gráficos e os testes foram feitos em cada umas das versões. Por último o resultado completa a fase 3 do projeto.

Figura 1: Estratégia de pesquisa.



Fonte: Elaboração própria.

2.1. Coleta de Dados

Durante o desenvolvimento deste estudo foram analisadas 122 versões do Joomla. Nas versões, desde a primeira versão 1.7.3 até a última versão estável 3.9.27 (24 de maio de 2021) utilizamos as ferramentas PhpMetrics e Sonarqube sem qualquer tipo de calibração das ferramentas usando as configurações originais de cada uma delas.

De acordo com SANTANA (2012) o Sonarqube é uma ferramenta de análise de métricas relacionadas a saúde do código. Essa ferramenta pode ser usada em diversas linguagens, incluindo PHP. Métricas são levantadas durante a análise do Sonarqube como code smell (conhecido também como mal cheiro de código), pontos sensíveis para segurança da informação, duplicidade de código, cobertura da análise entre outros pontos. Assim como no PhpMetrics no Sonarqube também é gerada uma pasta com páginas HTML que mostram o resultado da análise permitindo navegar entre as informações detalhando cada uma. Já o PhpMetrics é uma ferramenta de análise estática para projetos PHP. No PhpMetrics, para

cada análise, uma pasta com as métricas é gerada, contendo os resultados em um relatório baseado em HTML. Durante o processo de coleta de dados, foram realizados os seguintes passos:

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

A análise dos dados coletados, em especial, o número de linhas de código, modularidade e complexidade permite estimar o potencial de desenvolvimento, manutenção, modificação e adaptação do sistema de sistemas, apresentando uma visão geral da qualidade do código fonte do Joomla. Os dados coletados no estudo estão disponíveis em: https://github.com/LuizFernandoBenini/MetricasCicloDeVidaJoomla/blob/main/Metrics_Joomla.xlsx

2.2. Questões de pesquisa

Nesta pesquisa serão analisadas eficiência, complexidade, compreensão, reutilização, manutenção e sustentabilidade como os indicadores que compõem a qualidade do código fonte. Esses indicadores serão utilizados pois não foi encontrado, no contexto do Joomla, um amplo conjunto que poderia definir com precisão atributos para análise de longevidade dessa aplicação.

Foram definidos como base de pesquisa para conduzir a análise do código do Joomla, um conjunto de dados principal (PR) e um conjunto de secundários (SQ). Também foram definidas as hipóteses (H1 e H2).

- **PR -A longevidade do Joomla mudou ao longo do tempo?**

  H0: A longevidade do Joomla diminui ao longo do tempo.

  H1: A longevidade do Joomla permaneceu estável / crescendo ao longo do tempo.

  Dúvidas secundárias da pesquisa:

- **SQ 1- A eficiência do Joomla mudou ao longo do tempo?**

  H0: A eficiência do Joomla permaneceu estável ao longo do tempo.

  H1: A eficiência do Joomla não tem se mantido estável ao longo do tempo.

- **SQ 2- A complexidade do Joomla mudou ao longo do tempo?**

  H0: A complexidade do Joomla permaneceu ao longo do tempo.

  H1: A complexidade do Joomla não foi mantida ao longo do tempo.

- **SQ 3- O entendimento do Joomla mudou ao longo do tempo?**

  H0: A compreensão do Joomla permaneceu a mesma ao longo do tempo.

  H1: A compreensão do Joomla não tem sido a mesma ao longo do tempo.

- **SQ 4- A capacidade de reutilização do Joomla mudou ao longo do tempo?**

  H0: A reutilização do Joomla se manteve ao longo do tempo.

  H1: A reutilização do Joomla não foi mantida ao longo do tempo.

- **SQ 5- A capacidade de manutenção do Joomla mudou ao longo do tempo?**

  H0: A manutenção do Joomla foi mantida ao longo do tempo.

  H1: A manutenção do Joomla não foi mantida ao longo do tempo.

Com as dúvidas citadas acima, foi possível analisar as constantes mudanças do software a cada uma das 122 versões.

Para esclarecer as questões de pesquisa, será usado um protocolo de análise estatística, detalhando e comparando versões dentro da mesma versão principal e entre outros grupos, conforme descrito nos estudos realizados por SIQUEIRA et al. (2020). Os dados coletados foram usados para avaliar os atributos da qualidade do ecossistema Joomla com base nas métricas encontradas por BEDOYA et al (2020), MENS et al (2017) e WILLIANS et al (2009).

2.3. Análise de Dados

Os dados coletados foram usados para avaliar os atributos do ecossistema Joomla com base nas métricas extraídas informadas em BEDOYA et al. (2020), MENS et al. (2017) e WILLIANS et al. (2009):

- Complexidade: LdC, CC e RSYSC
- Eficiência: Dt e V

- Compreensibilidade: CC
- Reutilização: RSYSC
- Manutenção: Halstead, CrS, NdC e Dt

A tabela 1 mostra as associações de cada métrica com as ferramentas utilizadas.

Tabela 1. Métricas por ferramenta.

Métricas	PhpMetrics	SonarQube
Linhas de Código (LdC)	-	X
Número de Classes (NdC)	-	X
Complexidade Ciclomática por Classe (CC)	X	-
Complexidade relativa do sistema (RSYSC)	X	-
Média de bugs por classe (Halstead)	X	-
Violações (V)	-	X
Quantidade de Bugs (QB)	-	X
Débito (Dt)	-	X
Quantidade de Blocos Duplicados (QbD)	-	X

Fonte: Elaboração Própria.

3. ANÁLISES E DISCUSSÃO

Durante o estudo da qualidade do código levantamos os dados presentes na tabela 2 que traz um resumo das métricas levantadas:

Tabela 2. Estatística Descritiva.

Métrica	Min	1ª Qu.	Média	Mean	3rd Qu.	Max.	NA's
LdC	130549	176022	257501	241385	291285	302189	-
NdC	765	1003	1442	1347	1583	1667	-
CC	23.40	24.14	24.62	24.55	24.72	28.16	-
RSYSC	257.4	278.3	287.2	295.8	318.2	322.2	-
Halstead	0.5100	0.5225	0.5300	0.5325	0.5400	0.6100	-
V	763	1012	1387	1315	1555	1621	-
QB	775	1780	1792	1777	1820	2113	2
Dt	242.0	336.2	513.0	457.6	536.0	549.0	-
QbD	3214	5508	6518	6166	6669	7275	2

Fonte: Elaboração Própria.

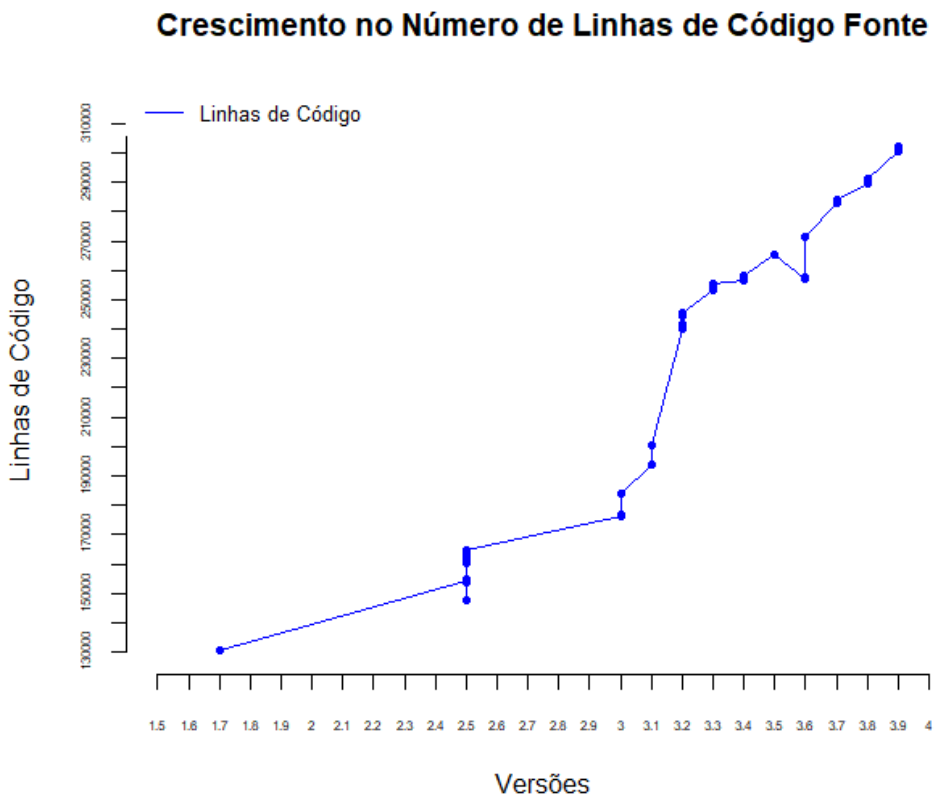
Na sequência, uma análise de cada ponto da tabela apresentada com estatísticas geradas no levantamento de dados para o estudo. Em cada um dos subtópicos abaixo foram abordados os testes de Kolmogorov-Smirnov (K-S) que consiste em analisar se houve uma distribuição normal na variação de cada métrica e Kruskal-Wallis(K-W) como orientado em AMORIM et al (2017). Para maior clareza, a aceitação de hipóteses nulas foi utilizado o nível de significância de 5%.

3.1. Linhas de Código

Este tópico irá apresentar o comportamento do código ao longo do desenvolvimento das 122 versões analisadas. De acordo com a análise apresentada na figura 2, a quantidade de linhas do código mais que dobrou desde sua primeira versão estável. Ao longo do desenvolvimento da aplicação o código cresceu de forma não linear tendo picos de crescimento e até algumas quedas na quantidade de linhas de código como entre as versões 3.5 e 3.6 indicando não 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 < 2.2e-16, ou seja, não apresenta distribuição normal dos dados, assim como no teste Kruskal-Wallis(K-W) no qual obtivemos o valor de p = 2.208e-15.

Figura 2: Crescimento de linhas de código.



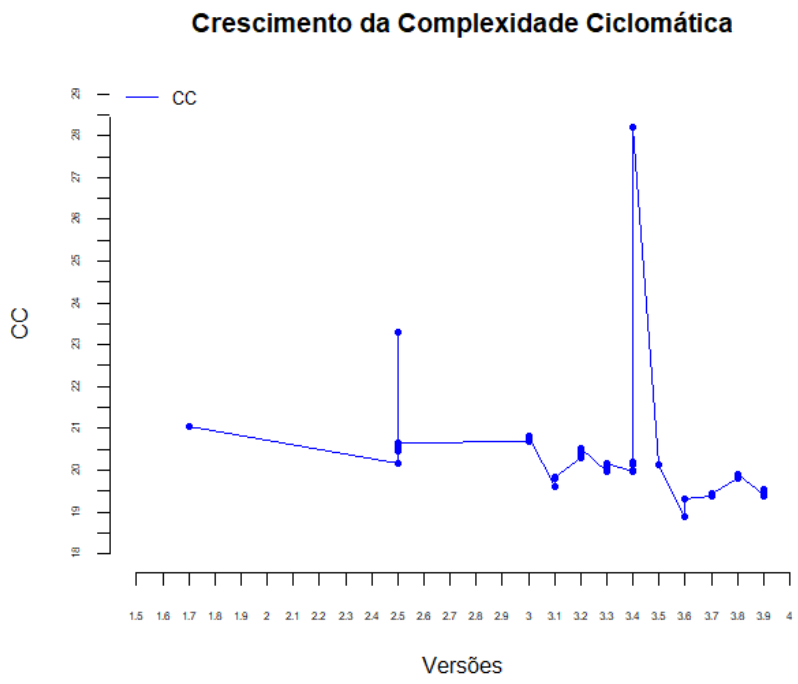
Fonte: Elaboração própria.

3.2. CC

De acordo com PERLIS (2021), outra métrica que ajuda a rastrear a complexidade do software é Mc-Cabe's cyclomatic complexity. A equação, CC = E - N + P, descreve esta métrica 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 análise é útil para medir a dificuldade de construir testes de unidade em um determinado código, uma vez que determina o número de caminhos linearmente independentes. Como é possível observar na figura 3 houveram picos onde a complexidade ciclomática ficou maior mostrando dificuldade de manutenção como a versão 2.5 e entre as versões 3.4 e 3.5.

Aplicando o teste K-S foram identificados indícios que o código teve grandes picos nos valores de complexidade ciclomatica gerando grande variação na distribuição dos dados. Essa informação mostra que o código não teve uma evolução constante nesse aspecto gerando variações tanto positivas quanto negativas a cada nova versão. Quando aplicado o teste One-sample Kolmogorov-Smirnov (K-S) para analisar essa variação, é obtido o valor de p = 2.863e-05, resultado menor que 0,05, indicando que não houve distribuição normal entre as versões. O teste de Kruskal-Wallis mostra o mesmo resultado com o valor de p < 2.2e-16.

Figura 3: Complexidade ciclomática.



Fonte: Elaboração própria.

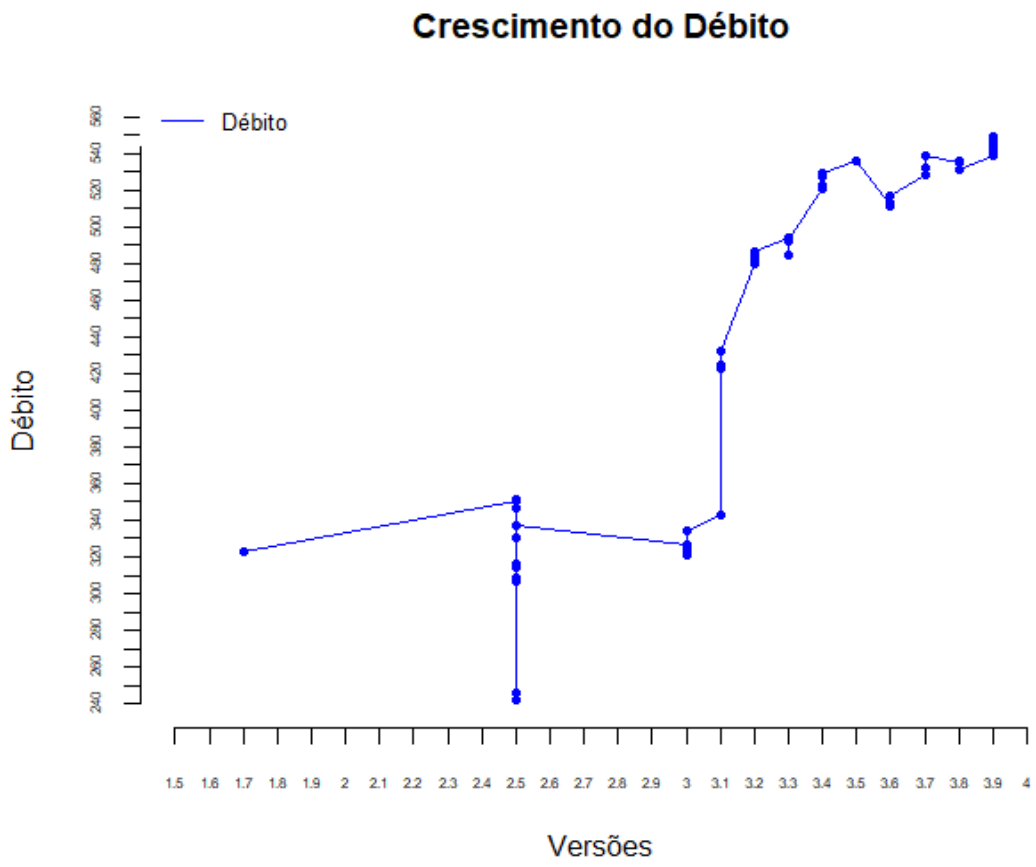
3.3. Débito Técnico

Nesse subtópico será abordada a análise dos dados relacionados ao crescimento do débito do código, ou dívida técnica relacionado ao desenvolvimento. De acordo com SANTANA (2019) este aspecto da análise consiste na soma de más práticas relacionadas ao desenvolvimento do código pensando apenas no presente, que geram uma dívida de correção e otimização do código, essa definição é chamada de dívida técnica.

Analisando o gráfico abaixo e aplicando o teste de K-S é possível observar que temos grandes variações relacionadas a dívida técnica, porém, do contrário que se deseja ao longo

do ciclo de vida de um software, o débito vem aumentando ao longo das versões chegando à marca de 560 horas na versão 3.9.27. Ao executar o teste One-sample Kolmogorov-Smirnov (K-S) obtivemos um valor de P = 2.863e-05 e no teste de Kruskal-Wallis o valor de P sendo menor que 2.2e-16 mostrando em ambos os testes não resultaram em uma distribuição normal do débito entre as versões.

Figura 4: Débito.



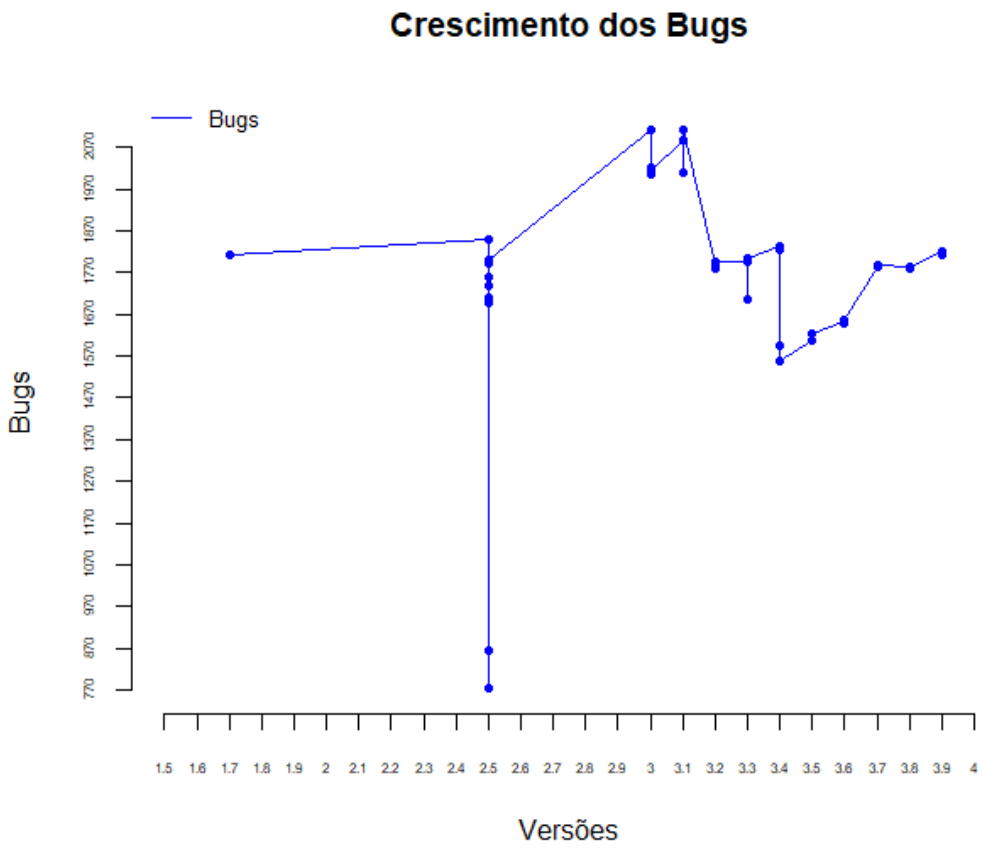
Fonte: Elaboração própria.

3.4. Bugs

O subtópico apresenta as métricas referente ao crescimento de bugs na aplicação que impacta diretamente na qualidade do código fonte como citado em PIANCO JUNIOR (2017). Assim como os demais tópicos existem oscilações consideráveis durante o desenvolvimento. Analisando o gráfico é possível ver que mesmo havendo muitas variações na distribuição dos dados, a comparação entre a primeira e a última versão mostram dados bem parecidos não mostrando um ganho significativo na tratativa de bugs ao longo das versões. Durante a versão

2.5 existiu melhoras de peso no código quando se trata de correções de bugs tendo seu menor valor durante todo o desenvolvimento analisado das 122 versões. Entre as versões 2.5 e 3 houve um crescimento linear na quantidade de bugs sendo que a partir da versão 3 houveram grandes oscilações. Durante a versão 3.4 existiu uma melhora significativa no quadro de bugs do sistema que voltou a crescer entre as versões 3.4 e a última versão tratada 3.9.27. Ao executar os testes obtivemos o valor de p = 1.132e-07 no teste de One-sample Kolmogorov-Smirnov (K-S) e de P = 0.03175 no teste de Kruskal-Wallis.

Figura 5: Crescimento de bugs.



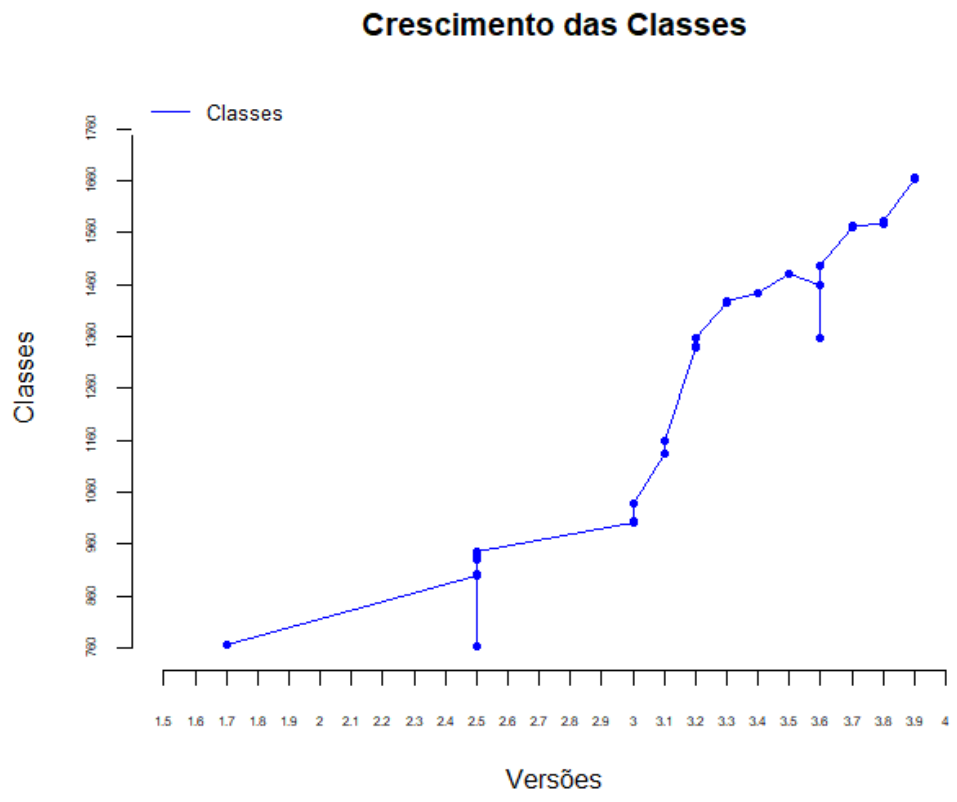
Fonte: Elaboração própria.

3.5. Halstead

De acordo com DINIZ (2018) as métricas de Halstead podem ser úteis durante o desenvolvimento para avaliar a qualidade do código em aplicações computacionais densas ou para acompanhar e medir quantitativamente a complexidade. As métricas de Halstead foram introduzidas em 1977 tendo sido usadas extensivamente desde então e estão entre as

mais antigas métricas de complexidade de programas computacionais hoje em dia. A medida de bugs entregue pela Halstead é uma estimativa do número de erros na implementação, expresso pela equação B = V / 3000, onde V = N * log2 (n), onde N = N1 + N2 e n = n1 + n2 de acordo com SIQUEIRA et al (2020). O valor de N1 é composto pelo número total de operadores e o N2, todos os operandos. O valor de n1 representa o número de diferentes operadores presentes e n2 o número de diferentes operandos. Aplicando o teste de K-S, foi encontrado o valor de p = 8.657e-06 mostrando a não distribuição normal dos dados apresentado variações durante o desenvolvimento.

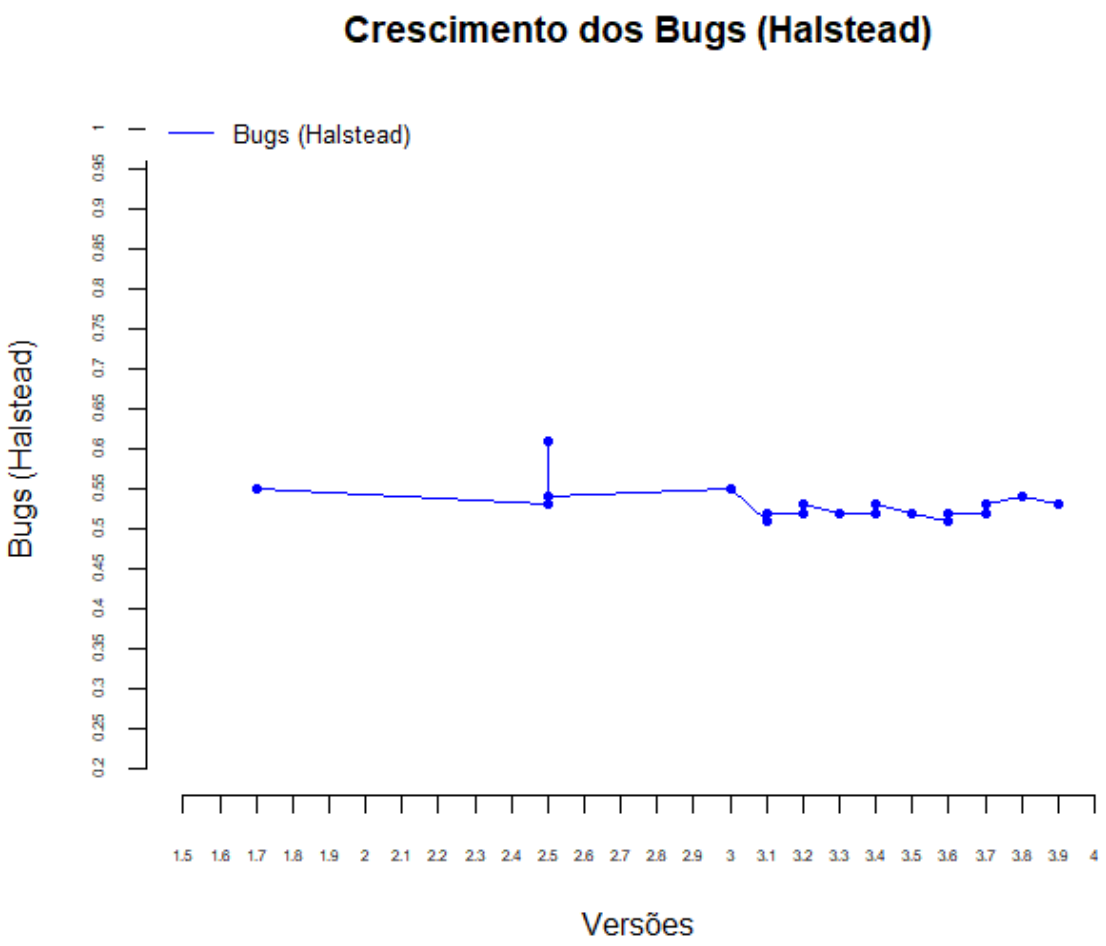
Figura 6: Crescimento da quantidade de classes.



Fonte: Elaboração própria.

No nível de significância, o método não paramétrico de K-W foi usado, com um valor de p de 2.359e-08 mostrando que pelo menos um grupo difere dos demais.

Figura 7: Crescimento do Halstead.



Fonte: Elaboração própria.

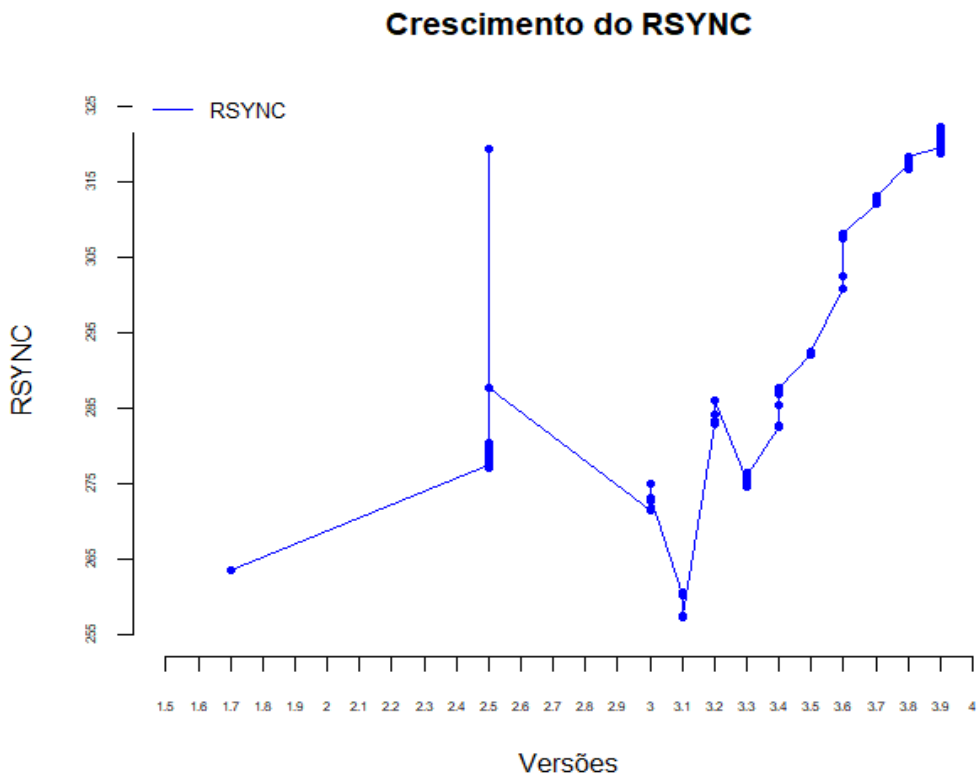
3.6. RSYSC

De acordo com PERLIS (2021) a Complexidade Relativa do Sistema (Relative System Complexity - RSYSC) mede a complexidade de um projeto de software em termos de uso de dados, passagem de parâmetros, e chamadas de procedimento. Será usada a equação do RSYSC = Média (CE + CI).

Nesta métrica, foi considerado que CE é a complexidade estrutural de um procedimento, sendo igual ao número de portas de entrada ao qual ela está conectada ao quadrado, ou seja, o número de outros procedimentos chamados para seu quadrado. A complexidade dos dados (CI) é a complexidade interna para um procedimento, representado por DC = IOvars / (SFOUT + 1). IOvars é equivalente ao número de variáveis de entrada dividido pela saída variáveis de um procedimento.

Para a qualidade do código de uma aplicação, é necessário alcançar um RSYSC o mais baixo possível. Como é possível observar na figura 8, durante a versão 2.5 existe um aumento considerável na complexidade do código sendo parcialmente reduzida no fim do fechamento da mesma. Entre o fim da versão 2.5 e a versão 3 existe uma redução linear e bem distribuída da complexidade tendo seu menor valor durante a versão 3.1. Após seu mínimo valor houve um aumento considerável daí em diante tendo uma pequena queda durante as versões 3.2 e 3.3, porém mantendo um aumento na complexidade até a última versão estável analisada 3.9.27. Para analisar mais precisamente usamos o teste de K-S que mostra uma não distribuição normal dos dados gerando um resultado de P = 8.843e-05 assim como no teste de K-W que mostra um valor de P= 0.0001339 que indica que um grupo pelo menos, difere dos demais.

Figura 8: Crescimento da complexidade relativa do sistema.



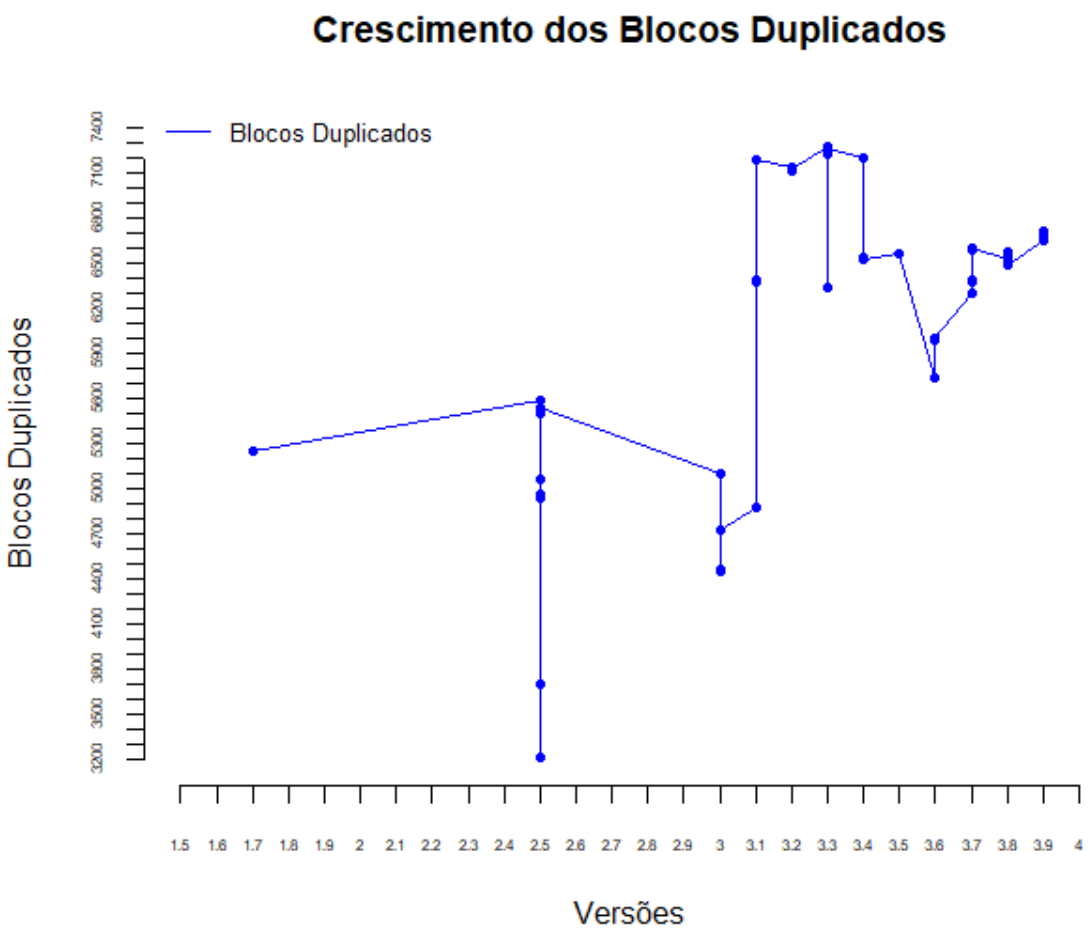
Fonte: Elaboração própria.

3.7. Blocos Duplicados

Durante o desenvolvimento do código prezamos por escrever códigos limpos, legíveis com alta coesão e baixo acoplamento. De acordo com EVALDO JUNIOR (2014) evitar códigos

duplicados também faz parte da saúde do código fonte do software. Neste subtópico foi analisado o crescimento de blocos duplicados ao longo das versões. Como pode ser observado na imagem abaixo na versão 2.5 existe uma queda expressiva de blocos duplicados mostrando melhora na reutilização de métodos e classes que voltou a crescer em seguida. Durante a versão 3.1 houve um grande aumento em blocos duplicados gerando uma mudança brusca no gráfico. Executando o teste K-S foi obtido o valor de P = 1.27e-05 mostrando que os dados não seguem uma distribuição normal. Já no teste de K-W foi obtido o valor 1.489e-11 que indica que pelo menos um grupo difere dos demais analisados.

Figura 9: Crescimento de blocos duplicados.

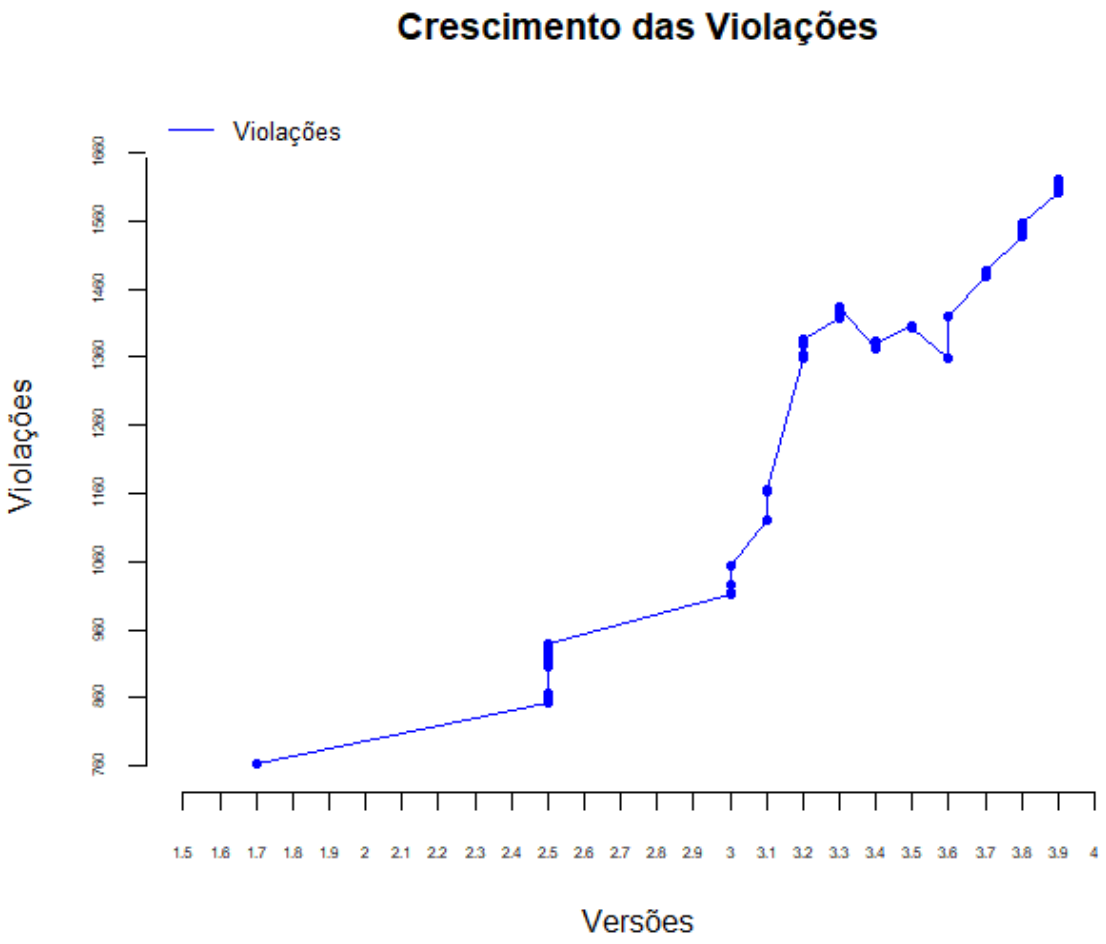


Fonte: Elaboração própria.

3.8. Violações

Este subtópico avalia o crescimento de violações de segurança ou vulnerabilidades no sistema. Essa métrica ajuda a entender como a preocupação com boas práticas de desenvolvimento seguro se manteve ao longo das versões no ecossistema do software. Analisando o gráfico é possível identificar que o crescimento das vulnerabilidades na aplicação se manteve ao longo do tempo salvo duas quedas curtas e pontuais na versão 3.3 e entre a versão 3.5 e 3.6. Executando o teste de K-S foi obtido o valor de P sendo 2.391e-05 mostrando diferença significativa, portanto não distribuição normal dos dados. No teste de K-W mostra que pelo menos um grupo difere dos demais mostrando um valor de P = 2.185e-15.

Figura 10: Crescimento de violações no sistema.



Fonte: Elaboração própria.

Como é possível observar, utilizamos os testes de Kolmogorov-Smirnov e Kruskal-Wallis para analisar cada uma das métricas que serão usadas na análise geral do estudo na seção 4.

4. CONSIDERAÇÕES FINAIS

Ao longo da seção 4, foram analisados os dados coletados para o estudo exploratório da qualidade do código fonte do Joomla que tem o intuito de produzir conhecimentos específicos sobre um determinado assunto e nesse caso estamos falando da saúde do código fonte da aplicação, para isso foram utilizadas todas as métricas analisadas durante a seção citada.

Quando é analisado a parte da complexidade do Joomla se trata falando das métricas LdC, CC e RSYSC. Foi possível observar nesse ponto que a LdC do Joomla tem aumentado ao longo do tempo, porém a CC apesar de um aumento brusco entre as versões 3.4 e 3.5, teve uma queda real na última versão mesmo que baixa mostrando que a complexidade ciclomática do código vem diminuindo sendo um aspecto positivo para o SECO. Em contra partida a RSYSC aumentou muito durante o tempo que é ruim para o SECO. Esses dados mostram que a complexidade do código tem mudado negativamente durante o tempo.

Analisando a eficiência foi levado em consideração as métricas Dt e V. Analisando a parte de débito técnico (Dt) da aplicação onde mostra más práticas, code-smell (mau cheiro de código) e erros de performance; é possível ver que houve grande aumento no valor do mesmo em horas mostrando que ao longo das versões as más práticas cresceram diminuindo assim a eficiência do código.

Na compreensibilidade da aplicação foi utilizada a métrica CC. Analisando a complexidade ciclomática é possível ver aumento na versão 2.5 e nas versões 3.4 e 3.5, porém com uma queda pequena até a última versão mostrando uma melhora nesse aspecto mesmo que pequena como apresentado na seção 4.

Na análise da reutilização também houve grande variação como é possível ver analisando a métrica RSYSC mostradas na seção 4. Essa análise mostra o quanto a complexidade relativa do sistema vem aumentando mostrando dificuldade em reutilizar aspectos da aplicação.

Por último, mas não menos importante foi analisada a manutenção baseada nas métricas Halstead, CrS, NdC e Dt. Foi identificado que a CrS mesmo tendo uma queda entre as versões 2.5 e 3.0, 3.0 a 3.1 e 3.2 e 3.3, cresceu muito ao longo das versões sendo sinal

negativo para o Seco. A quantidade de classes do sistema aumentou consideravelmente ao longo das versões tendo queda apenas nas versões 2.5 e 3.6. O Halstead mostra o resultado dessa relação apresentando um crescimento abrupto na versão 2.5, mas no geral se manteve estável mostrando a queda na manutenção do Joomla. Analisando o gráfico de débito é possível ver também um aumento considerável na quantidade em horas de débito técnico da aplicação mostrando uma degradação da qualidade do código fonte da aplicação.

Analisando essas métricas indicadas temos indícios que o código fonte do Joomla tem sido afetado negativamente ao longo das versões mostrando uma piora na qualidade do mesmo. Com todas as métricas específicas levantadas durante esse artigo temos indícios que o mesmo trouxe uma análise do código fonte do Joomla através de métricas específicas. Essa perspectiva não implica na qualidade da aplicação em termos de usabilidade, aderência de mercado por exemplo; implicando apenas na análise do código fonte do mesmo.

Outra ferramenta interessante que pode auxiliar na profundidade do estudo é o PHP DEPEND o qual detalha mais os pontos abordados pelo Sonarqube e PHPMetrics podendo ser uma análise complementar desse artigo.

REFERÊNCIAS

Aida Sefic Williams. 2009. Life cycle analysis: A step by step approach. TechnicalReport. Champaign, IL: Illinois Sustainable Technology Center.

Alan Perlis. McCabe Cyclomatic Complexity.2021. Disponível em http://www.chambers.com.au/glossary/mc_cabe_cyclomatic_complexity.php . Acessado em 25 de novembro de 2021.

Evaldo Junior. Os problemas do Código Duplicado. 2014. Disponível em http://blog.evaldojunior.com.br/desenvolvimento/melhores%20práticas/2014/01/23/os-problemas-do-codigo-duplicado.html . Acessado em 25 de novembro 2021.

Flávio Roberto Ianague Diniz. 2008. CE-230: Qualidade, Confiabilidade e Segurança de Software.Disponível em http://agileg.wdfiles.com/local--files/projfinalexam/ExameFinal_rev00.pdf Acessado em 17 de novembro de 2021

Marcus Aurélio Cordeiro Piancó Júnior. (2017). Analisando a relação entre Mudanças no código fonte de Bugs no software. Disponível em http://www.repositorio.ufal.br/bitstream/riufal/2922/1/Analisando%20as%20relações%20entre%20mudanças%20no%20código%20fonte%20e%20bugs%20no%20software.pdf. Acessado em 19 de novembro 2021.

Oscar Franco-Bedoya, Oscar Cabrera, and Sandra Hurtado-Gil. 2020. QuESo-Process: Evaluating OSS Software Ecosystems Quality. In Proceedings of the 10th Euro-American Conference on Telematics and Information Systems (Aveiro, Portugal) (EATIS '20). Association for Computing Machinery, New York, NY, USA,Article 27, 7 pages. https://doi.org/10.1145/3401895.3402056

Otávio Golçalves de Santana. 2012. Sondando qualidade de código com o sonar. Disponível em https://www.devmedia.com.br/sondando-qualidade-de-codigo-com-o-sonar/24239. Acessado em 24 de novembro de 2021.

Simone da Silva Amorim, Félix Simas S. Neto, John D. McGregor, Eduardo Santana de Almeida e Christina von Flach G. Chavez. 2017.How Has the Health of Software Ecosystems Been Evaluated? A Systematic Review. 31º Simpósio Brasileiro de Engenharia de Software (Fortaleza, CE, Brasil) (SBES'17). Association for Computing Machinery, Nova York, NY, EUA, 14–23. https://doi.org/10.1145/3131151.3131174

Tassio Siqueira, Regina Braga e José Maria N. David. WordPress: An Exploratory Study of the Ecosystem Life Cycle.2021. In SBES ’21: XXXV Simpósio Brasileiro de Engenharia.

TFM Sirqueira, MA Miguel, HLO Dalpra, MAP Araujo e JMN David. 2020. Aplicação de Métodos Estatísticos em Engenharia de Software: Teoria e Prática. In Engenharia no Século XXI. Volume 18, Editora Poisson, 2020, pages 228-246.

Tom Mens, Bram Adams, and Josianne Marsan. 2017. Towards an interdisci-plinary, socio-technical analysis of software ecosystem health. arXiv preprint arXiv:1711.04532 (2017).

Wellington Santana.2019. O que é débito técnico? Saiba como tratar. Disponível em https://ezdevs.com.br/o-que-e-debito-tecnico-saiba-como-tratar. Acessado em 20 de novembro de 2021.