

Análise de Métricas Estáticas entre releases em repositórios de linguagem Python no GitHub

Andre Murilo Neves Vasconcelos¹, Leonardo Antunes Barreto Noman¹,
Lorrayne Reis Silva¹, Paulo Henrique Cota Starling¹, Victor Augusto dos Santos¹

¹Pontífica Unidade Católica de Minas Gerais (PUC - MG)

Abstract. *The following work seeks to identify the relationship between software releases carried out in terms of configuration management and software evolution from repositories collected on Github. a software and numerical validation of the increase in compeensibility related to raw metrics.*

Resumo. *O seguinte trabalho busca identificar o relacionamento existente entre releases de software realizadas no quesito de gerência de configuração e evolução de software a partir de repositórios coletados em Github. Com aplicação de análises e buscas por métricas que fornecem com precisão a aproximação da demanda de manutenibilidade de um software e validação numérica do aumento de compreensibilidade relacionado a métricas brutas.*

1. Introdução

Com o desenvolvimento da tecnologia e o boom informacional ocorridos nas últimas décadas, tornou-se indubitável a demanda de como organizar as fontes de códigos contínuos, desenvolvidas de maneira a serem gerenciadas corretamente e de forma compartilhada por uma equipe. Surgiram-se então as plataformas de hospedagem de código fonte e arquivos com controle de versões, dentre elas o GitHub. A partir de tal significação, o conceito de releases é cada vez mais difundido no mundo tecnológico, as mesmas podem ser denominadas como versões de um software lançadas com alterações como crescimento ou exclusão de funcionalidades, sua principal questão de discussão gira em torno da efetividade relacional de um maior número dessas atrelado a uma maior ou menor qualidade de um software.

De acordo com a discussão apresentada é relacionado ao fator de significância de qualidade, uma das características presentes quando esse fator é colocado como prerrogativa diz respeito associação do mesmo com métricas estáticas que se relacionam indiretamente com seus atributos, como por exemplo, pode-se citar números de linhas de código e de métodos. Assim, considerando um projeto x à medida que esse se torna mais robusto a tendência em relação às métricas de qualidade se portaram de maneira com aumento ou diminuição de tais valores numericamente, pensando em tal questão o trabalho tem como objetivo entender como a qualidade de um projeto evolui de acordo com o passar de releases realizadas.

Baseando nos apontamentos apresentados, a principal motivação do trabalhado é se conectar ao desejo de entendimento de diferenças numéricas no gerenciamento e evolução de um software relacionados diretamente a trabalhos com utilização da linguagem Python. Essa sendo uma das linguagens de programação mais utilizadas do mundo, está colocada na lista de 10 mais populares desde 2003 e foi desenvolvida na

década de 80 por Guidovan Rossum. Por ser uma linguagem do paradigma orientado a objetos torna-se mais fácil a computação de métricas de qualidade de software.

Pensando nisso a presente pesquisa se justifica através do entendimento numérico da efetividade do lançamento de releases estarem correlacionadas a um aumento de qualidade em relação a crescente concepção atual de cada vez mais repositórios possuírem um alto número das mesmas. Assim, estariam essas tornando um sistema mais manutenível e proposto a serem menos modificados no qual ambos aspectos se relacionam diretamente a qualidade dos sistemas como um todo.

Partindo das percepções destacadas o objeto geral do trabalho se relaciona a análise de métricas estáticas com o objetivo de se avaliar a compressibilidade, complexidade e manutenibilidade com relação a repositórios famosos na linguagem Python por meio do GitHub no contexto de melhora do software de acordo com a qualidade de releases. De maneira mais específica pode-se obter o correlacionamento desses com bad smells, uma vez que ao demonstrar matematicamente valores não atrelados a qualidade podem ser identificados demandas de refatoração no código fonte, ou seja, ao se encontrar bad smells no início de um projeto por meio de métricas validadas através das primeiras releases realizadas ocorre um alerta de qualidade deste o princípio evitando gastos de refatoração futuros.

A partir das informações percorridas foram geradas três perguntas que visam o entendimento do projeto. A primeira RQ1 “A cada release do software Python, existe o aumento de métricas brutas?”, alinhando-se a métricas brutas o intuito da questão diz respeito a averiguação de aumento ou decréscimo de valores dessas métricas no decorrer de releases geradas para compreensão inicial. Segundamente RQ2 “A cada release, os sistemas se tornam mais fáceis de serem lidos e menos arriscados de serem modificados?”, partindo da prerrogativa de qualidade pretende-se validar o relacionamento dessa com a necessidade de modificação de um sistema, uma vez que a concepção desse em alta estado de qualidade está relacionado a seu facilitamento de ser entendido e diminuição de mudanças. Por último RQ3 “A criação de releases diminui a manutenibilidade do sistema?”, a partir de uma visualização numérica será possível discernir a demanda por manutenibilidade uma vez que tal valor está relacionada a maior ou menor qualidade.

A partir das formulações percorridas o presente trabalho apresentara como embasamento para realização da pesquisa uma sessão de trabalhos relacionados, na qual foram coletados trabalhos de suma relevância de entendimento de projetos e ideias a serem debatidas. Uma sessão metodológica com apresentação das questões de pesquisa realizadas como hipóteses nulas e alternativas, como o experimento foi realizado e os métodos e técnicas aplicados para o problema proposto. Por fim, uma sessão de resultados com apresentação das averiguações realizadas e a resolução das perguntas formuladas.

2. Trabalhos Relacionados

Com o intuito de obter embasamento científico para promulgação de resultados do trabalho foram buscados artigos e dissertações que forneceram assimilação problemática e ideação de questões. Assim, a seguir são apresentados alguns trabalhos relacionados com suas respectivas apresentações, resultados e apontamentos de similaridades existentes.

Este trabalho tem como uma das principais fontes o artigo de [Ramos, M. E., Valente, M. T. 2014]¹, no mesmo os autores coletaram de repositórios do github, escolhidos

a partir de características como alta popularidade medida por mais de 150 commits, que não eram fork e que utilizavam a linguagem JavaScript. Com a ferramenta escomplex, foram analisadas 15 métricas que obtiveram resultados satisfatórios, como em medidas de complexidade ciclomática adequadas se comparadas a sistema Java, outro em que a modularidade tem maior número somente em projetos grandes. O trabalho se relaciona com esse projeto na semelhança em análise de métricas estáticas diferenciando-se no escopo, linguagem de programação escolhida e ferramentas para coleta de métricas.

No artigo Understanding metrics based detectable smells in Python software: A comparative study é abordado o entendimento de métricas baseados em code smells, onde é explorado o efeito da manutenibilidade no código, o qual se obtém como resultado o relacionamento entre bad smells com falhas ou mudanças de modularidade. Atualmente, existem diversas ferramentas para detectar bad smells em linguagens como C e Java, mas como demonstrado e semelhantemente a esse trabalho, existem poucos estudos sobre ferramentas e técnicas que detectam esses bad smells na linguagem de programação python, uma das prerrogativas do atual trabalho discutido.

Em ESTIMATING COMPLEXITY OF PROGRAMS IN PYTHON LANGUAGE (2011) é realizado uma estimativa da complexidade de programas utilizando como base a linguagem python, assim os mesmos buscam em uma fórmula uma nova métrica de complexidade para linguagem orientada a objetos comparando o resultado com outras métricas. Como resultado uma nova métrica é formulada empiricamente e as análises de métricas estáticas mostram que entre C, Java e Python, a linguagem python apresentou melhor resultado. Como relacionamento estabelecido entre o trabalho postula-se a análise de métricas estáticas realizadas servindo como embasamento.

Nunes, H. (2014) em uma dissertação de mestrado, visa contribuir no aspecto de utilização de métricas de software para identificação de bad smells como buscamos inicialmente neste artigo, ele propôs um método e uma ferramenta para a identificação de bad smells, via métricas de software em sistemas orientados por objetos a partir de modelos UML.

Em Using metrics to evaluate software system maintainability o artigo visa demonstrar como ao se utilizar métricas é possível analisar a necessidade da manutenção de um software, assim a análise automatizada de manutenibilidade de software pode ser usada para orientar a tomada de decisões relacionadas a software. Os resultados indicam que a avaliação de manutenibilidade automatizada pode ser usada para dar suporte a decisões de compra versus construção, análise pré e pós-reengenharia, análise de qualidade de subcomponentes, alocação de recursos de teste e a previsão e direcionamento de subcomponentes propensos a defeitos. O artigo se relaciona ao trabalho proposto com promulgação da RQ3 sobre a necessidade de diminuição de manutenibilidade quanto em maior estágio de desenvolvimento o software se encontra, além da validação de respondimento dessa RQ3 estar também relacionada ao utilização de análise de métricas.

No artigo Cyclomatic complexity density and software maintenance productivity a principal tematização ocorre em decorrência de como dados de complexidade ciclomática podem estar relacionados ou não necessidade de uma produtividade na manutenção de software. Como resultados apreendidos demonstram que a produtividade de manutenção aumenta a medida que aumenta a complexidade ciclomática. O artigo se relaciona ao tra-

balho em relação ao respodimento da questão R3 no qual uma das metricas utilizadas para relacionar a necessidade de modificação ou seja manutenção ser complexidade ciclômática.

Finalmente, é apresentado por Tiago L. Alves(2010) um método que determina limites métricos empiricamente a partir de dados de medição. Os dados de medição para diferentes sistemas de software são agrupados e agregados, após os quais são selecionados os limites que (i) destacam a variabilidade da métrica entre os sistemas e (ii) ajudam a focar em uma porcentagem razoável do volume do código-fonte. Utilizamos como base este trabalho para definição de análise de métricas e definição da ferramenta randon devido contemplar métricas razoáveis ao objetivo

3. Metodologia

Para dar início ao experimento, depois de montar as perguntas a serem respondidas na pesquisa, uma hipótese nula e alternativa para cada quetão foi criada para serem refutadas ou não. Nessa seção será abordado as hipóteses nulas e alternativas, como as perguntas da pesquisa foram respondidas e como cada script criado e seus respectivos entradas e saídas necessários. As hipóteses nulas e alternativas são:

RQ1 - Hipótese nula: As métricas brutas (LOC, LLOC e Comentários) permanecem as mesmas durante as dez primeiras releases dos repositórios coletados.

RQ1 - Hipótese alternativa: As métricas brutas (LOC, LLOC e Comentários) aumentam durante as dez primeiras releases dos repositórios coletados.

RQ2 - Hipótese nula: A cada release os sistemas mantêm o índice de complexidade ciclômática e não se tornam mais fáceis de serem lidos ou menos arricados de serem modificados.

RQ2 - Hipótese alternativa: A cada release os sistemas aumentam o índice de complexidade ciclômática e não se tornam mais complexos de serem lidos e mais arriscados de serem modificados.

RQ3 - Hipótese nula: A cada release os sistemas mantêm o mesmo índice de manutenibilidade do sistema.

RQ3 - Hipótese alternativa: A cada release os sistemas ficam mais difíceis e complexos de serem modificados e de receberem manutenção.

Para coletar e analisar as métricas das dez releases dos repositórios python, foram criados 4 scripts principais em python responsáveis pela coleta dos repositórios, coleta das tags das releases, extração e persistência dos dados com o radon e tratamento dos dados coletados. A partir dos dados tratados, foi utilizado o powerbi para a criação de gráficos para confirmar ou refutar as hipóteses criadas antes do processamento das métricas.

O script `get_repos.py` tem como objetivo coletar os trezentos repositórios mais famosos (com mais estrelas), que tem como linguagem primária o python no GitHub. Para obter esse dados, é realizada uma requisição no GitHub GraphQL Api para que retorne os nomes dos repositórios mais famosos, a quantidade de estrelas e o número de releases. Devido ao limite do tamanho da resposta da API, é necessário criar uma paginação nas requisições. A cada resposta recebida pela API, é aplicado um tratamento em que é apenas coletado os repositórios com no mínimo dez releases. Todos esses repositórios são

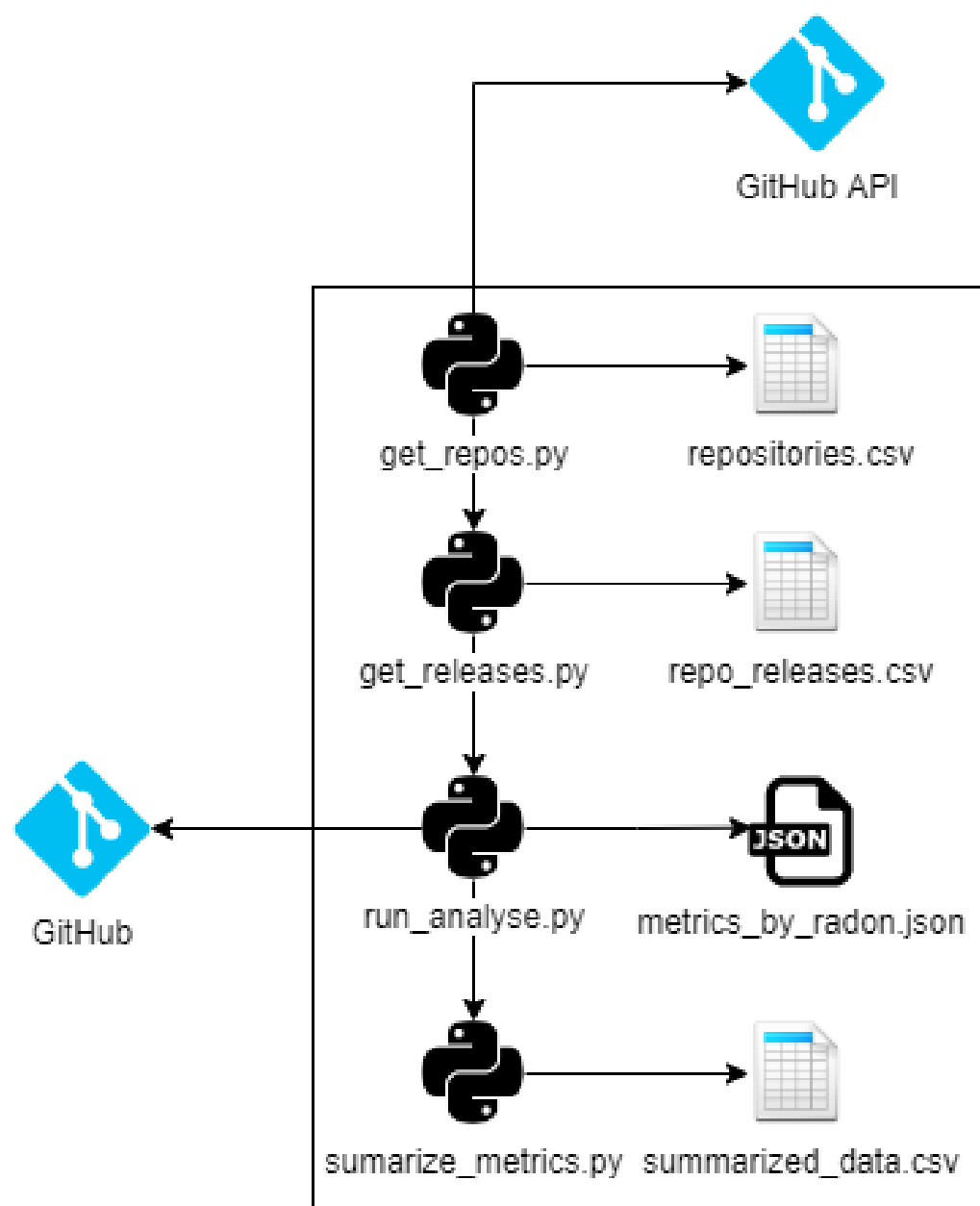
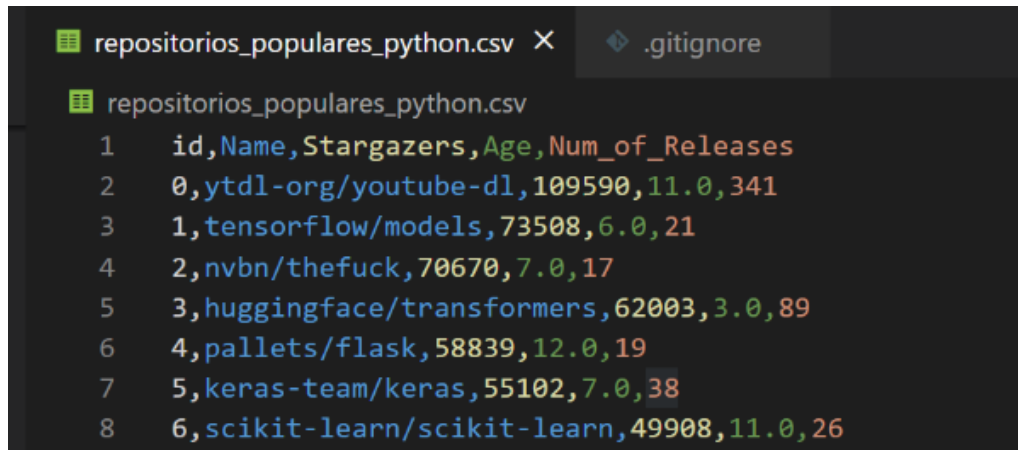


Figure 1. Arquitetura do funcionamento do sistema de coleta de dados

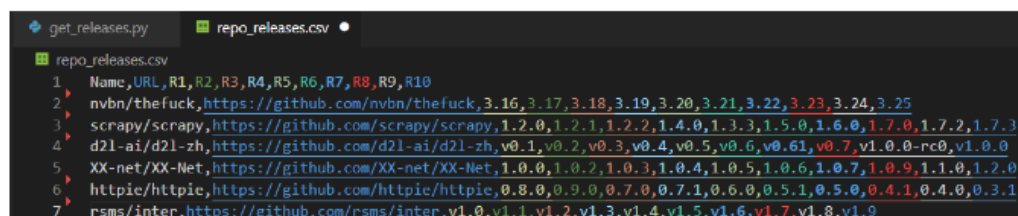
salvos em um arquivo .csv chamado **repositorios_populares_python.csv**. As seguintes informações são salvas de cada repositório: autor, nome, número de estrelas, idade e número de releases.



```
repositorios_populares_python.csv
repositorios_populares_python.csv
1 id,Name,Stargazers,Age,Num_of_Releases
2 0,ytdl-org/youtube-dl,109590,11.0,341
3 1,tensorflow/models,73508,6.0,21
4 2,nvbn/thefuck,70670,7.0,17
5 3,huggingface/transformers,62003,3.0,89
6 4,pallets/flask,58839,12.0,19
7 5,keras-team/keras,55102,7.0,38
8 6,scikit-learn/scikit-learn,49908,11.0,26
```

Figure 2. Exemplo de saída do script **repositorios_populares_python.csv**

O script **get_releases.py** tem como objetivo coletar as dez primeiras tags dos repositórios salvos no arquivo **repositorios_populares_python.csv** e salvá-las em um arquivo chamado **repo_releases.csv**. Para isso, foi criada uma query para consultar o GitHub GraphQL Api que retorna as dez tags de cada repositório. As tags são ordenadas por ordem crescente de criação. A coluna R1 do CSV criada pelo script representa a primeira tag criada no repositório e a coluna R10 é a décima tag.



```
get_releases.py
repo_releases.csv
repo_releases.csv
1 Name,URL,R1,R2,R3,R4,R5,R6,R7,R8,R9,R10
2 nvbn/thefuck,https://github.com/nvbn/thefuck,3.16,3.17,3.18,3.19,3.20,3.21,3.22,3.23,3.24,3.25
3 scrapy/scrapy,https://github.com/scrapy/scrapy,1.2.0,1.2.1,1.2.2,1.4.0,1.3.3,1.5.0,1.6.0,1.7.0,1.7.2,1.7.3
4 d2l-ai/d2l-zh,https://github.com/d2l-ai/d2l-zh,v0.1,v0.2,v0.3,v0.4,v0.5,v0.6,v0.61,v0.7,v1.0.0-rc0,v1.0.0
5 XX-net/XX-Net,https://github.com/XX-net/XX-Net,1.0.0,1.0.2,1.0.3,1.0.4,1.0.5,1.0.6,1.0.7,1.0.9,1.1.0,1.2.0
6 httpie/httpie,https://github.com/httpie/httpie,0.8.0,0.9.0,0.7.0,0.7.1,0.6.0,0.5.1,0.5.0,0.4.1,0.4.0,0.3.1
7 rms/inter,https://github.com/rms/inter,v1.0,v1.1,v1.2,v1.3,v1.4,v1.5,v1.6,v1.7,v1.8,v1.9
```

Figure 3. Exemplo de saída do script **get_releases.csv**

O script **run_analyse.py** tem como objetivo coletar as métricas que o radon extrai de cada release e salvar em formato JSON na pasta **collected-metrics**. Para isso, o script itera sobre cada linha do arquivo **get_releases.csv**. Ao ler uma linha, é utilizada a biblioteca 'os' do python para clonar o repositório referente à coluna URL na determinada tag. O código do repositório é salvo na pasta temp/, o radon coleta as métricas daquela determinada versão e a pasta **temp/** é removida logo depois da coleta. A clonagem e coleta acontecem dez vezes por cada repositório (uma cada tag) e após completadas, as métricas de complexidade ciclomática (CC), índice de manutenibilidade (MI), número de linhas de código (LOC), número de linhas lógicas de código (LLOC) e número de linhas de comentários (Comments) são salvos na pasta **collected-metric** em um arquivo com o nome 'metrics_' mais o nome do repositório. Todos os arquivos tem formato JSON para facilitar o tratamento dos dados coletados pelo radon posteriormente.

Por fim, o script **extract_metrics.py** tem como objetivo tratar os dados dos arquivos JSON gerados pelo script **run_analyse.py**. Cada arquivo dentro da pasta

collected-metric é lido por vez e, como o radon tem um formato diferente para cada métrica, o script teve que ser dividido em três blocos de lógica diferentes:

- Tratamento das métricas de CC e IM: o radon calcula a complexidade ciclomática e o índice de manutenibilidade por função de cada arquivo. O script soma os valores calculados de todas as funções dos arquivos por releases.
- Tratamento das métricas de LOC, LLOC e Comments: o radon calcula essas métricas por arquivo e então o script salva a soma total de cada uma delas.

4. Resultados

Caracterização dos datasets

Com o intuito validar os datasets foi realizado a caracterização dos mesmos apresentando dados como idade, número de releases e estrelas. Dentro de cada uma dessas observações é analisado conjuntamente o máximo, mínimo, mediana e média dos quesitos. Assim é possível observar que o gráfico apresentado por idade demonstra uma proximidade entre os valores de mediana e média com ambos respectivamente apresentando os valores 6,00 e 6,15 e a máxima idade em 13 anos. Sobre o gráfico de número de releases observa-se um número mínimo de releases de 10 e máximo de 1.224 com mediana na casa dos 32. A respeito do número de estrelas verifica-se que o maior valor de estrelas é aproximadamente 109.000 e com mínimo de 3.808 e mediana de 6.411. Após a análise da caracterização dos dados verifica-se que os repositórios coletados em sua maioria possuem idade entre 6 e 10 anos com número de releases que variam entre 32 e 100 e com número de estrelas entre 6.000 e 10.000.

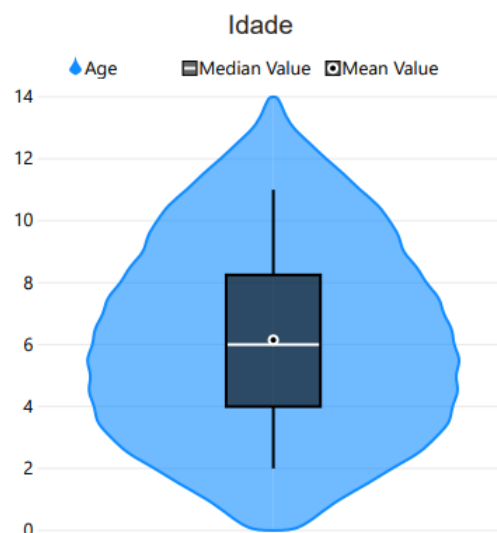


Figure 4. Caracterização do Dataset: Idade

Foi feita também uma análise sobre a diferença de dias entre uma release e outra. Com os dados obtidos foi calculado uma média de 18,4 dias e uma mediana de 19 de diferença entre uma release e outra.

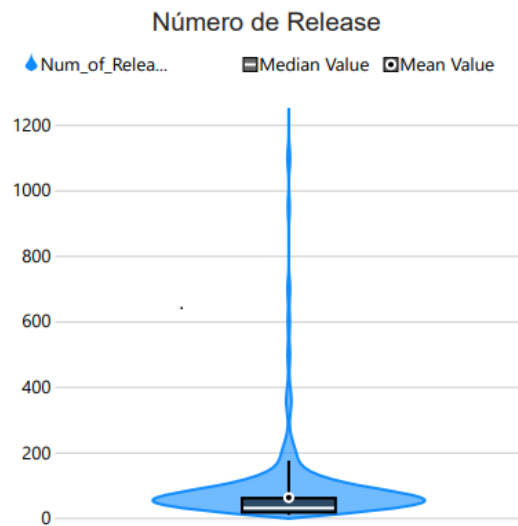


Figure 5. Caracterização do Dataset: Número de Release

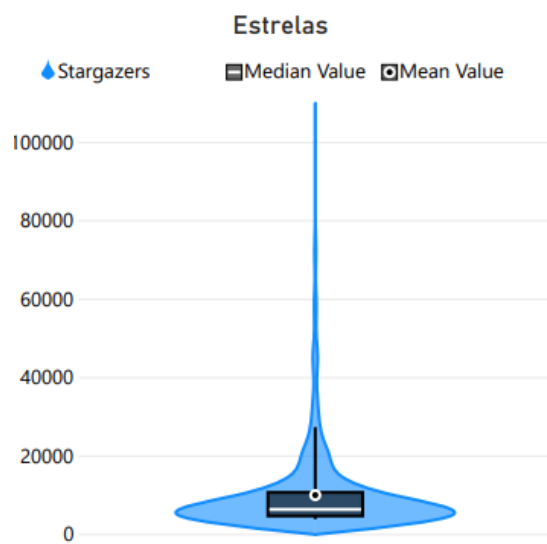


Figure 6. Caracterização do Dataset: Estrelas

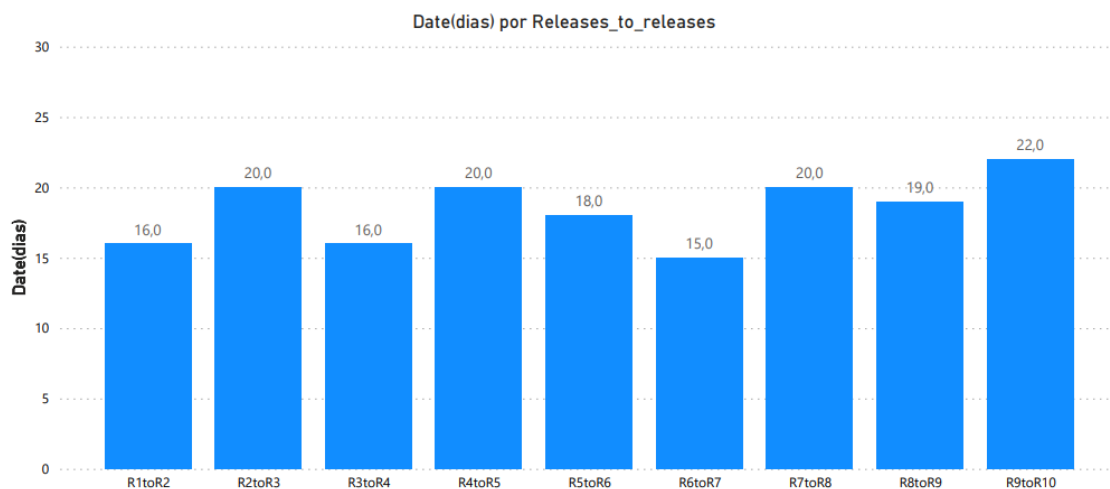


Figure 7. Caracterização do Dataset: Diferença de dias

4.1. RQ1 - A cada release do software python, existe o aumento de métricas brutas?

A resolução dessa RQ acontece por meio da análise de LLOCS, LOCs e COMMENTS.

Em LOCS ao observar a mediana por releases é possível notar um decréscimo de LOC especialmente entre a diferença da 5ª para a 6ª release e da 6ª para 7ª. Em relação ao gráfico de mediana da diferença de release para release verifica-se o mesmo procedimento com uma média igual a 2 até a 6ª release e a partir dessa uma mediana com valor igual a 1.

A respeito do número de LLOCS observa-se uma mediana crescente no total de LLOCS de acordo com o aumento de releases. Quando observado a mediana de diferença de LLOCS de release para release observa-se uma mediana frequente de valor 2 até a release 6 e após essa um decremento no valor da diferença para 1, a partir da diferença entre 5ª e 6ª release.

Sobre COMMENTS verifica-se uma mediana constante de 1 em quase todos os casos. Os casos especiais em que foram 0 são as diferenças da release 7ª para 8ª e da 9ª para 10ª.

É possível inferir que a cada release não existe um fixamento de aumento de métricas brutas no qual algumas aumentam e outras não. Baseando-se nas coletadas, o número de LLOCS e LOC tende decrescer enquanto COMMENTS tende a sofrer uma pequena variação durante as releases feitas por uma busca de maior qualidade.

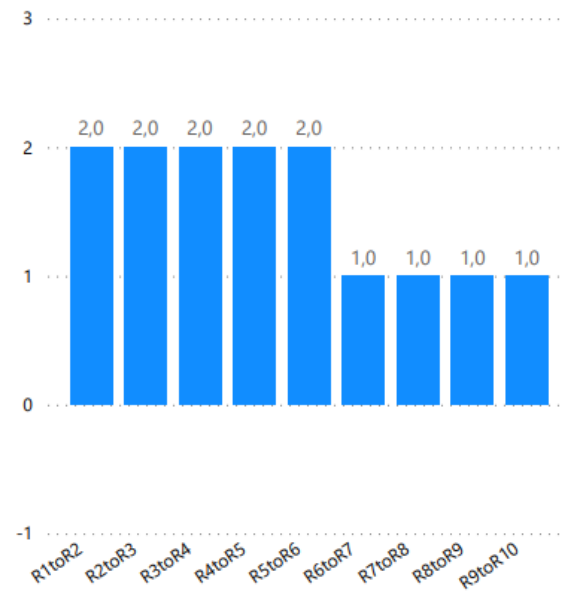


Figure 8. RQ1: LOCS

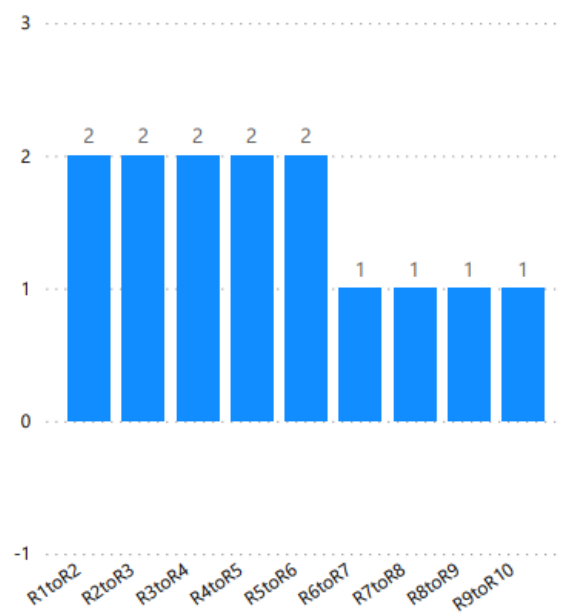


Figure 9. RQ1: LLOCS

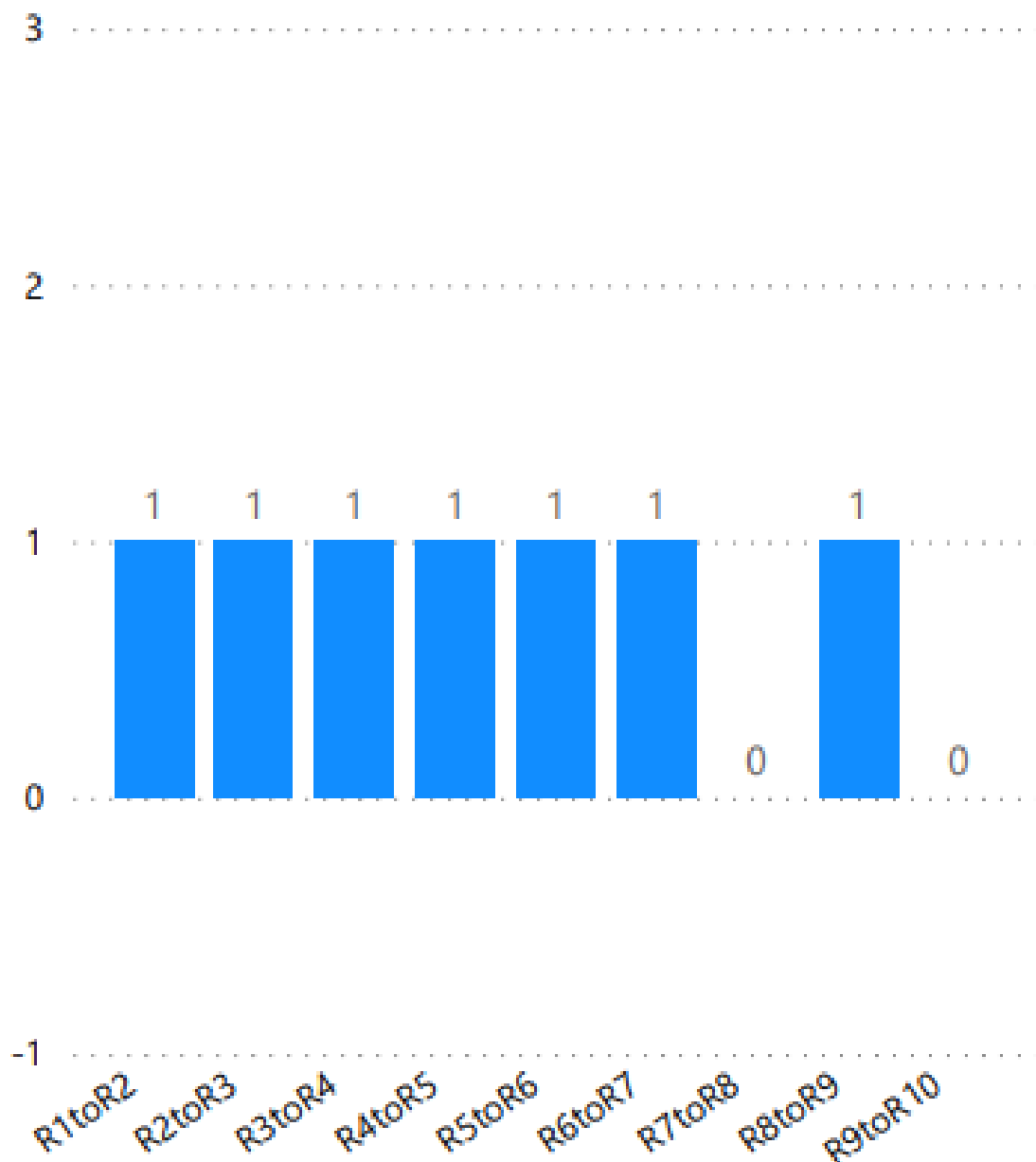


Figure 10. RQ1: COMMENTS

4.2. RQ2 - A cada release, os sistemas se tornam mais fáceis de serem lidos e menos arriscados de serem modificados?

Com o intuito de validar a divergência entre releases para a questão a ser tratada a métrica coletada com esse intuito foi a complexidade ciclomática referente a 1ª e a 10ª release.

Observa-se na 1ª release uma complexidade ciclomática máxima de 50.705 com mediana de 879, o gráfico demonstra uma concentração de valores de CC abaixo de 10.000 com alguns valores destoantes até 50.000. Em relação a 10ª release verifica-se uma complexidade ciclomática máxima de 52.793 com mediana de 1.768, o gráfico demonstra

uma concentração de valores de CC abaixo de 10.000 com alguns valores destoantes até 50.000.

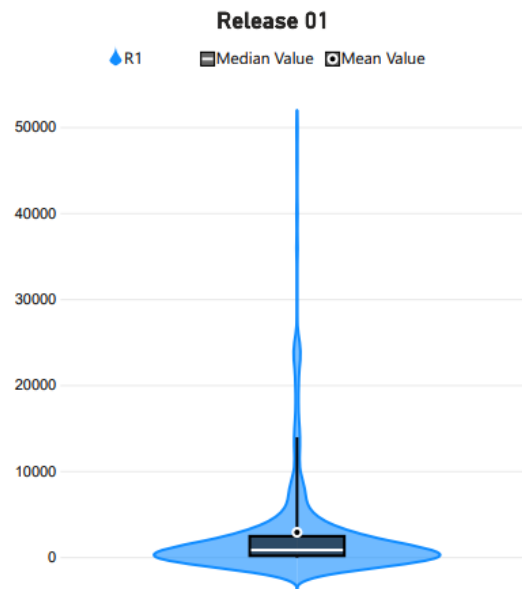


Figure 11. RQ2: CC da release 01.

Ao se fazer a visualização de CC por sua mediana na decorrência da release 1 a 10 verifica-se que os valores sofrem oscilação até a 6ª release com valores ainda muito altos e a partir da 7ª release os valores diminuem drasticamente e tendem a aumentar de maneira padronizada a cada nova release. No gráfico de mediana de diferença de releases observa-se o mesmo comportamento, sendo possível notar o decremento do valor, que da release 9 para 10 apresenta-se como 0,68.

Como um valor de CC mais baixo está relacionado a um melhor valor para entendimento do código e menos risco em modificações pode-se considerar que durante as releases iniciais tais valores oscilam pois tais sistemas ainda estão sofrendo alterações significativas e a partir do momento trazem mais qualidade aos códigos a cada release o sistema se torna mais fácil de ler e menos arriscado de sofrer modificações.

4.3. RQ3 - A criação de releases diminui a manutenibilidade do sistema?

Com o intuito de validar a divergência entre releases para a questão a ser tratada a métrica coletada com esse intuito foi o índice de manutenibilidade referente a 1ª e a 10ª release.

Observa-se na 1ª release um índice de manutenibilidade máximo de 100 com mediana de 71,20 o gráfico demonstra uma concentração de valores de IM acima de 60 com alguns valores destoantes de 0 a 10. Em relação a 10ª release verifica-se um índice de manutenibilidade máximo de 100 com mediana de 71,14, o gráfico demonstra uma concentração de valores de IM acima de 60 com alguns valores destoantes até 10.

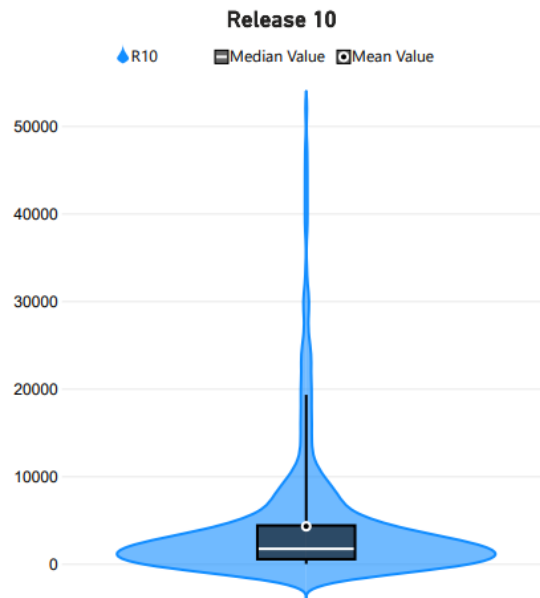


Figure 12. RQ2: CC da release 10.

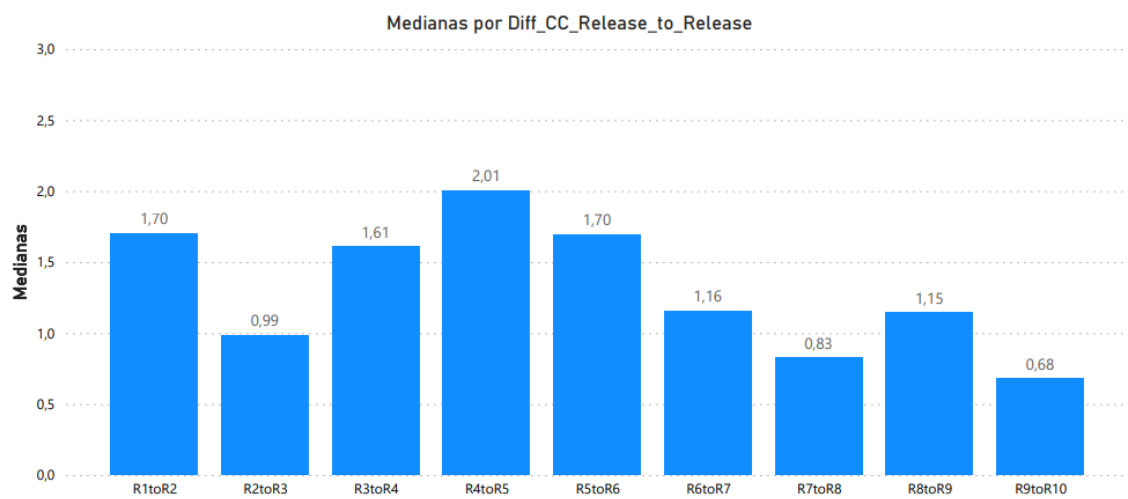


Figure 13. RQ2: CC na diferença das releases.

Figure 14. Raw

Release 01

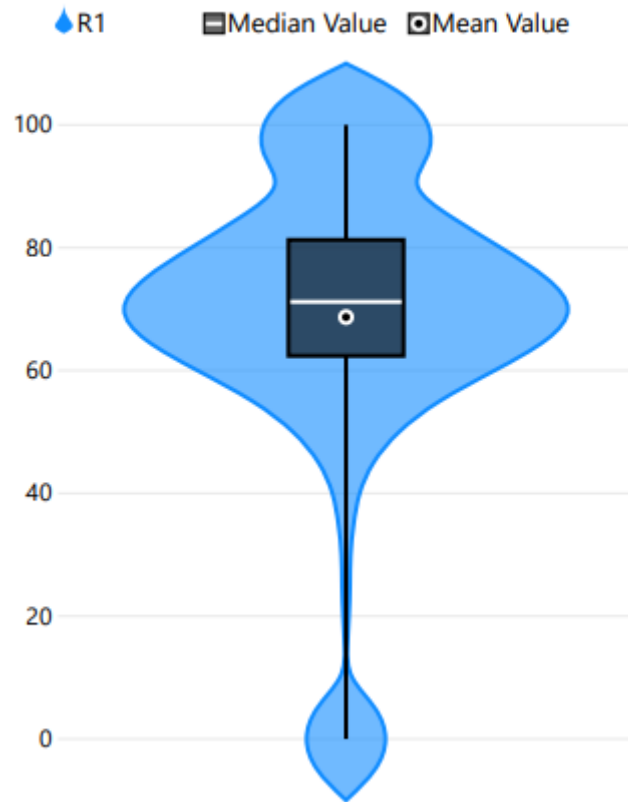


Figure 15. RQ3: Índice de manutenibilidade da release 01.

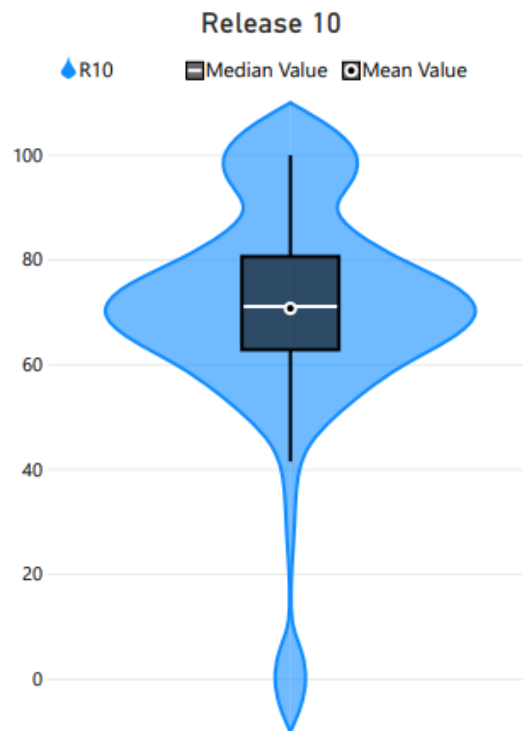


Figure 16. RQ3: Índice de manutenibilidade da release 10.

Por meio da visualização de ocorrência de releases da 1ª a 10ª observa-se um valor constante da mediana, onde os valores são sempre 0.

É possível afirmar que a criação de releases não diminui a manutenibilidade do sistema, uma vez que o valor é sempre 0.

5. Discussão

RQ1 - A hipótese nula feita para essa RQ prevê que as métricas brutas permanecem as mesmas durante as dez primeiras releases dos repositórios coletados. Com os resultados obtidos podemos observar uma leve variação das métricas a partir de uma determinada release, sendo essa diferente para cada métrica. Conclui-se então que a hipótese é válida e prevê bem o comportamento dessas métricas ao longo do tempo. Isso se deve ao fato de que é comum o uso de metodologia ágil para o desenvolvimento de software, o que o torna uma prática de certo modo uniformizada.

RQ2 - Nesta RQ, a hipótese nula feita foi que o índice de complexidade ciclomática se mantém estável e, portanto, não há aumento na facilidade de serem lidos ou menos arriscados de serem modificados. Segundo os resultados, a complexidade ciclomática aumentou razoavelmente da primeira para décima release. O que vai contrário a nossa hipótese nula, sendo assim podemos concluir que de fato há um certo aumento na dificuldade da leitura, compreensão e o quão arriscado é modificar o software em questão. Todavia, esse aumento foi progressivamente menor, o que implica que a parte mais complexa do processo de desenvolvimento de software é no início.

RQ3 - Para a RQ3 foi formulado a seguinte hipótese nula: a cada release os sis-

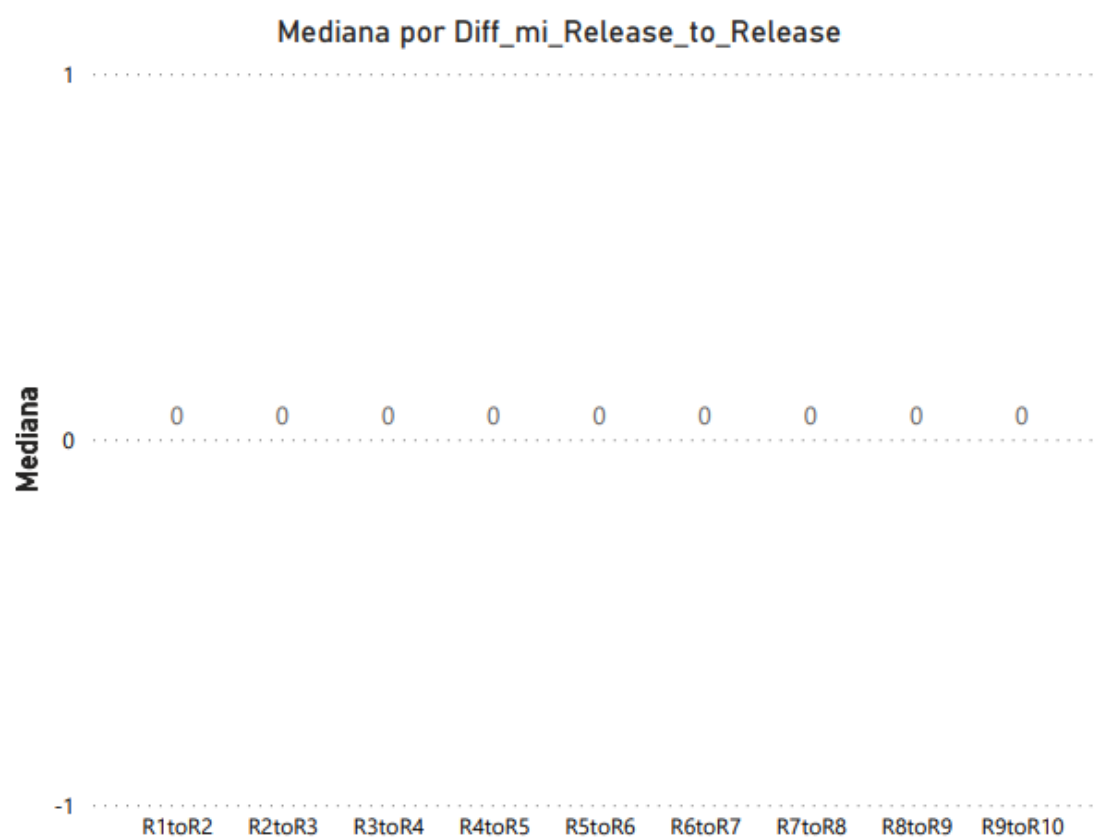


Figure 17. RQ3: Mediana do índice de manutenibilidade das releases 01 a 10.

temas mantêm a manutenibilidade do sistema. Isso significa que o índice de manutenibilidade se manterá constante e que, portanto, ele não progride nem retrocede nesse aspecto. Olhando os resultados vemos que de fato o índice se manteve como zero em toda release, o que vai de acordo com a hipótese citada acima. O índice de manutenibilidade calcula quão fácil é mudar e dar suporte ao código, então esse resultado é preocupante como um todo.

Indiscutivelmente o resultado de maior expressão é o índice de manutenibilidade ser 0 durante todas releases. Este fato é curioso e se deve ao volume de Halstead, índice de complexidade ciclomática total, SLOC e porcentagem de linhas de comentário. Como trabalho futuro sugere-se a investigação de por que o resultado permaneceu constante em zero.

6. Ameaças à Validade

Nesta seção, relatamos as ameaças à validade, bem como os respectivos tratamentos.

Validade de construção: A validade de construção em estudos controlados refere-se à mensuração correta das variáveis dependentes, que no nosso caso, qualidade do software pela quantidade de releases. Uma possível ameaça ao procedimento experimental é no momento de extração de métricas com o `randon` e repositórios do github, algum defeito com a máquina ocorrer e interferir no processo. Para prevenção, utilizamos meios para estabilizar a corrente elétrica.

Validade interna: A validade interna está relacionada a aspectos não controlados que podem afetar os resultados experimentais, como repositórios com métricas estáticas enormes, gerando outliers comparado a outros repositórios. Buscando mitigar essas ameaças, foi sumarizado os resultados das métricas, apresentados no gráfico em mediana, pois um conjunto de informações numéricas, o valor central corresponde à mediana desse conjunto. Outra possível ameaça é o comportamento inesperado da ferramenta `Radon` no momento de coleta das métricas, no qual não entendendo o código em um repositório, pode gerar valores irregulares.

Validade externa: A validade externa está relacionada com a possibilidade de generalização dos resultados. Como o nosso experimento foi realizado com 384 amostras do github na linguagem python e as 10 primeiras releases, esse número pode não ser uma amostra de releases representativa. No entanto, as amostras são diversificadas, os repositórios têm níveis variados de características, como idade e estrelas.

Além disso, existe a validade de conclusão: que é o grau para qual as conclusões sobre a relação entre as variáveis com base nos dados são corretas ou “razoáveis”, como foram avaliados as 10 primeiras releases dos repositórios, não temos conhecimento se o comportamento de qualidade iriam impactar no resultado final da mediana. Para isto coletamos os repositórios com mais estrelas e idade do github, buscando manter um padrão.

7. Conclusão

Com os resultados e a geração dos gráficos, é possível observar que os sistemas aumentaram de tamanho no geral. O aumento de linhas lógicas de código observado na RQ1 está relacionado também com o aumento da complexidade ciclomática que podemos observar nos resultado da RQ2. É bem provável que quando aumentamos a quantidade de linhas de

código nos sistemas, a tendência é que aumente a métrica de complexidade ciclomática. Apesar do aumento no código, não é possível concluir nesse estudo, se a adição de novas linhas está relacionado com cobertura de testes, novas features, correção de bugs ou refatorações.

Apesar dos crescimento dos softwares analisados, é observado na RQ3 que o índice de manutenibilidade não melhora e nem piora entre uma release e outra. Essa métrica é interessante de ser observada já que mede a manutenção (fácil de suportar e alterar) do código-fonte.

Ter o controle da evolução do software em relação à sua qualidade é necessário para um projeto de software. Através dessas métricas é possível avaliar se com o passar da sua evolução, o sistema está aumentando a sua qualidade, o que pode refletir no sucesso de um projeto ou não.

8. Referências

Ramos, M. E., Valente, M. T. (2014). Análise de Métricas Estáticas para Sistemas JavaScript. In Anais do II Workshop on Software Visualization, Evolution and Maintenance (pp. 30-37).

Fowler, M. (1999). Refactoring: improving the design of existing code. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Marinescu, R. (2002). Em Measurement and Quality in Object-Oriented Design, Tese de Doutorado. Timisoara, Romênia. University of Timisoara

Lanza, M.; Marinescu, R. Ducasse, S. (2006). Object-Oriented Metrics in Practice. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Oliveira, P., Valente, M. T., and Lima, F. (2014). Extracting relative thresholds for source code metrics. In IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), pages 254–263.

Gill, G. K. and Kemerer, C. F. (1991). Cyclomatic complexity density and software maintenance productivity. IEEE Transactions on Software Engineering, 17(12):1284–1288.

Coleman, D., Ash, D., Lowther, B., and Oman, P. (1994). Using metrics to evaluate software system maintainability. Computer, 27(8):44–49.

Alves, T. L., Ypma, C., and Visser, J. (2010). Deriving metric thresholds from benchmark data. In 2010 IEEE International Conference on Software Maintenance (ICSM), pages 1–10. IEEE

McCabe, T. J. (1976). A complexity measure. IEEE Transactions on Software Engineering, SE-2(4):308–320.

Chen, Z., Chen, L., Ma, W., Zhou, X., Zhou, Y., Xu, B. (2018). Understanding metric-based detectable smells in Python software: A comparative study. Information and Software Technology, 94, 14-29.

Chong, T. Y., Anu, V., Sultana, K. Z. (2019, August). Using software metrics for predicting vulnerable code-components: a study on java and python open source projects. In 2019 IEEE International Conference on Computational Science and Engineering (CSE)

and IEEE International Conference on Embedded and Ubiquitous Computing (EUC) (pp. 98-103). IEEE.

Misra, S., Cafer, F. (2011). Estimating complexity of programs in Python Language. *Tehnički vjesnik*, 18(1), 23-32.

Chen, Z., Chen, L., Ma, W., Xu, B. (2016, November). Detecting code smells in Python programs. In 2016 international conference on Software Analysis, Testing and Evolution (SATE) (pp. 18-23). IEEE.

Orrú, M., Tempero, E., Marchesi, M., Tonelli, R., Destefanis, G. (2015, October). A curated benchmark collection of python systems for empirical studies on software engineering. In *Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering* (pp. 1-4).

Nunes, H. (2014). Identificação de bad smells em software a partir de modelos UML, Dissertação (mestrado). Belo Horizonte, Minas Gerais. Universidade Federal de Minas Gerais-Departamento de Ciência da Computação.