Avaliação da Maturidade dos Processos de Gerência de Configuração em Projetos Open Source

Leonardo Júnio Alves dos Santos
Universidade Federal de Viçosa
Florestal, MG-Brasil
leonardo.j.santos@ufv.br

Gláucia Braga e Silva
Universidade Federal de Viçosa
Florestal, MG-Brasil
glaucia@ufv.br

ABSTRACT

With the increasing of contributors and companies in open source projects it is necessary to study the maturity of the adopted development processes. This paper proposes quantitative metrics, based on the CMMI, MPS.BR and OMM models, to evaluate the maturity of Software Configuration Management (SCG) processes in open source projects. The metrics were automated and applied on data from 6 projects hosted on GitHub. The results were unsatisfactory, in terms of best practices of the SCG. Furthermore, for all the projects were found some gaps in terms of the change control flow and the traceability of the releases configuration items.

CCS CONCEPTS

• **Software and its engineering → Software configuration management and version control systems**; **Software libraries and repositories**;

KEYWORDS

CMMI, MPS.BR, OMM, controle de mudanças, métricas quantitativas

ACM Reference format:
Leonardo Júnio Alves dos Santos and Gláucia Braga e SilvaAvaliação da Maturidade dos Processos de Gerência de Configuração em Projetos Open Source. 8 páginas.

1 INTRODUÇÃO

No universo de projetos *Open Source*, apesar de não haver a definição formal e explícita de um processo, os projetos se desenrolam com resultados satisfatórios. Uma das razões para isso pode ser atribuída ao fator motivacional desse universo colaborativo, ou seja, vários desenvolvedores, integrantes efetivos de um projeto ou voluntários, têm o interesse em contribuir, seja reportando ou corrigindo *bugs* ou ainda produzindo novas versões de código-fonte.

No entanto, mesmo os resultados sendo satisfatórios, uma vez que há produtos de qualidade provenientes do universo OSS (*Open Source Software*), como: Apache, Ubuntu, Node.js, dentre outros já conhecidos e consolidados no mercado, alguns ajustes nos processos aplicados certamente contribuiriam para a melhoria da qualidade destes produtos e facilitariam o trabalho dos envolvidos.

Esses tipos de produtos são desenvolvidos em repositórios públicos de desenvolvimento, como o GitHub, uma plataforma de controle de versões amplamente utilizada, onde códigos colaborativos são hospedados. A ferramenta também oferece funcionalidades de *issue tracking*, *merge*, *deploy*, integração contínua, entre outras. No entanto, considerando que a ferramenta deixa a cargo do colaborador o bom uso dessas funcionalidades, quando se avalia o ciclo de vida da mudança de forma completa, existem algumas lacunas que podem prejudicar os resultados dos projetos. Como exemplos de boas práticas não obrigatórias que ficam a cargo dos integrantes do projeto, tem-se: o uso de tags para classificar *issues* e a rastreabilidade de *issues* em *commits* [20]. Diante disso, possíveis melhorias no fluxo da ferramenta aliadas à adoção de boas práticas de desenvolvimento poderiam resultar em contribuições mais eficientes e resultados mais satisfatórios.

Para que as pessoas possam contribuir da melhor forma possível, um conjunto de boas práticas de desenvolvimento de software deve estar definido e acessível a todos. Nesse contexto, destacam-se as diretrizes do processo de Gerência de Configuração de Software (GCS), que norteiam o controle das mudanças e versões do produto de software em desenvolvimento. Tais diretrizes são recomendadas pelos modelos de maturidade de processos de software CMMI (*Capability Maturity Model Intergration*), MPS.BR (Melhoria do Processo de Software Brasileiro) e OMM (*Open Source Maturity Model*).

Assim, este trabalho propõe a avaliação do processo de GCS no universo de projetos *open source*, segundo alguns dos indicadores de qualidade propostos nos modelos CMMI, MPS.BR e OMM, com o intuito de se avaliar a maturidade dos processos envolvidos e identificar pontos de melhoria que podem potencializar os resultados obtidos. Para cada um dos indicadores selecionados, será proposta uma métrica capaz de medi-lo quantitativamente. Por fim, o trabalho compreenderá o desenvolvimento de uma ferramenta de software que automatiza a aplicação das métricas propostas sobre dados de projetos armazenados em repositórios do GitHub.

Este artigo está estruturado da seguinte forma: na Seção 2, serão apresentados os principais modelos de maturidade de processo de software; na Seção 3 é explicado o fluxo da mudança no GitHub; a seção 4 descreve alguns trabalhos relacionados; na Seção 5, apresenta-se o estudo comparativo da maturidade dos processos de GCS em projetos *open source* e a análise dos resultados obtidos da aplicação das métricas; e, por fim, na Seção 6, são apresentadas as conclusões do trabalho e as sugestões de trabalhos futuros.

2 MATURIDADE DE PROCESSOS DE SOFTWARE

Os modelos de maturidade CMMI, MPS.BR e OMM têm como objetivo comum servir de referência utilizando práticas para se medir a

maturidade do processo de desenvolvimento de software de uma empresa.

O CMMI é um modelo de referência, escolhido por ser de cunho internacional e bastante consolidado no mercado global. Possui três modelos para organizações: CMMI para o Desenvolvimento (CMMI-DEV), CMMI para Aquisição (CMMI-ACQ) e CMMI para Serviços (CMMI-SVC). Neste trabalho, será utilizado como base o CMMI-DEV que possui 22 áreas de processo, sendo que cada área é um *cluster* de práticas relacionadas que, quando implementadas coletivamente, satisfazem um conjunto de metas. O CMMI-DEV possui 5 níveis de maturidade (Figura 1), sendo o nível 1, o mais baixo, e o 5, o mais alto (abrange todos os níveis anteriores). Cada nível contém um conjunto de áreas de processo que precisam ter todas suas metas atingidas para que uma organização seja certificada em um nível [24].

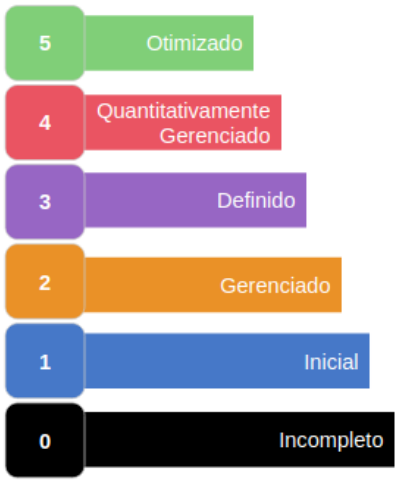

Figura 1: Níveis de Maturidade do CMMI. (Adaptado de *Maturity Levels Summary* [13])

Outro modelo de maturidade que merece destaque é o MPS.BR, que se baseia no CMMI, mas foca em apoiar as micro, pequenas e médias organizações brasileiras. Este modelo possui os modelos de referência: MPS para Software (MPS-SW), MPS para Serviços (MPS-SV) e MPS para Gestão de Pessoas (MPS-RH). O modelo MPS-SW, utilizado durante a pesquisa, possui sete níveis de maturidade, de G (Parcialmente Gerenciado) a A (Em Otimização) [21]. A correspondência entre os modelos MPS.BR e CMMI pode ser vista na Tabela 1.

Tabela 1: Correspondências entre os Modelos CMMI e MPS.BR[2]

CMMI	MPS.BR
Nível 5	Nível A
Nível 4	Nível B
Nível 3	Níveis C, D e E
Nível 2	Níveis F e G
Nível 1	-

Tabela 2: Correspondências entre os Modelos CMMI e OMM[16]

OMM	CMMI
Avançado	Níveis 2 e 3
Intermediário	Nível 2
Básico	Nível 2 (Opcional)

Também merece destaque o modelo OMM[19], que também se baseia no CMMI, mas tem foco em projetos OSS. Diferente dos modelos CMMI e MPS.BR, o modelo OMM possui três níveis de maturidade: Avançado, Intermediário e Básico. Segundo Petrinja [17], esse modelo se destaca dentre os modelos existentes para OSS, pois engloba mais áreas de processo. A correspondência dos níveis dos modelos CMMI e OMM é encontrada na Tabela 2.

2.1 Maturidade em Gerência de Configuração

A Gerência de Configuração é responsável por estabelecer e manter a integridade dos itens de configuração de um projeto[22]. Esses itens podem ser códigos fontes, documentações, modelos, hardware, ferramentas de testes, dentre outros itens referentes ao desenvolvimento de software.

Como os modelos MPS.BR e OMM foram baseados no CMMI eles abordam os mesmos processos, mas de formas diferentes. A Tabela 3 apresenta as correspondências dos indicadores do processo de Gerência de Configuração[25] nesses modelos. Dentre os sete indicadores do CMMI, este trabalho abordará apenas o CM.SP 1.3, CM.SP 2.1 e o CM.SP 2.2, para se avaliar quantitativamente alguns processos de projetos de software armazenados no GitHub. A seguir são apresentados esses indicadores e as práticas necessárias para satisfazê-los.

2.1.1 CM.SP 1.3 Criar ou liberar baselines. No geral, os modelos de maturidade apresentados ressaltam que para estar de acordo com este indicador é necessário que sejam respeitadas as seguintes práticas:

(1) Obter autorização para criação/liberação de uma *baseline*.
(2) Formar as *baselines* somente com itens de configuração que estão no sistema de gerenciamento.
(3) Documentar todos os itens de configuração contidos nas *baseline*.
(4) Disponibilizar as *baselines*.

2.1.2 CM.SP 2.1 Rastrear solicitações de mudança. Para que uma solicitação de modificação seja devidamente implementada ela deve seguir ao menos os seguintes passos[23]:

(1) Documentar a solicitação da mudança.
(2) Analisar o impacto da mudança.
(3) Avaliar a modificação (aprovada ou reprovada).

2.1.3 CM. SP 2.2 Controlar Itens de Configuração. Este indicador orienta que todas as alterações nos itens de configuração sejam controlados, com aplicação das seguintes práticas:

(1) Controlar as mudanças dos itens de configuração durante a vida útil do produto.

Tabela 3: Indicadores de Gerência de Configuração de Software

Indicador	CMMI	MPS.BR	OMM
Identificar itens de configuração.	CM.SP 1.1	GCO2	CM-1.1
Estabelecer um sistema de gerenciamento de configuração.	CM.SP 1.2	GCO1 e GCO6	CM-1.2
Criar ou liberar *Baselines*.	**CM.SP 1.3**	**GCO3**	**CM-1.3**
Rastrear solicitações de mudança.	**CM.SP 2.1**	**GCO5**	**CM-2.1**
Controlar Itens de Configuração.	**CM.SP 2.2**	**GCO5**	**CM-2.2**
Estabelecer registros de gerenciamento de configuração.	CM.SP 3.1	GCO4	CM-3.1
Realizar auditorias de configuração.	CM.SP 3.2	GCO7	CM-3.2

(2) Obter autorização antes de adicionar itens a uma nova versão.
(3) Revisar se as mudanças nos itens não causaram algum impacto negativo nas baselines.
(4) Documentar as alterações realizadas nos itens e seus motivos.

3 PRÁTICAS DE GCS NO GITHUB

Atualmente, o GitHub representa uma das mais utilizadas plataformas de hospedagem de código aberto e colaborativo, sendo utilizada por empresas como Google, Airbnb, IBM, entre outras. Esse tipo de ferramenta se torna importante no contexto de projetos OSS pois possibilita o controle de versões de um produto em desenvolvimento e facilita a manutenção de um projeto como um todo, supervisionando atividades como *Commits* e *Pull Requests* de uma possível contribuição. Além disso, a plataforma ainda possibilita integração com outras ferramentas de apoio ao desenvolvimento como *Atom*, *Circle CI* e *Slack*, fazendo com que haja um ambiente favorável para desenvolvimento de uma mudança [3]. Em virtude das características citadas acima, este trabalho utilizou projetos armazenados no GitHub para realização das avaliações.

A seguir são listados alguns conceitos importantes no contexto do GitHub[6]:

- ***Issue*** é a documentação de reclamações, sugestões ou tarefas relacionadas ao projeto.
- ***Commit*** é uma alteração em item de configuração.
- ***Pull Request*** é a solicitação da integração das mudanças realizadas na versão atual do sistema.
- ***Release*** é a forma de criar e fornecer versões de software[4].

Uma consideração importante é que os conceitos de *baseline* e *release* são relativamente parecidos. Uma *baseline* representa um conjunto de itens de configuração que passaram formalmente pelo controle da mudança, sendo que tais itens representam um estado do produto em que este já foi devidamente implementado e testado. Quando uma *baseline* é estabelecida, todo o desenvolvimento do produto passa a acontecer com base no itens nela presentes. Já uma *release* representa um produto, ou parte dele, que será apresentado ao cliente, ou seja, a *release* é composta pelos itens de configuração que serão entregues ao cliente final[15].

Os modelos de maturidade utilizados neste trabalho consideram *baselines* para a disponibilização de versões do produto para o cliente. No entanto, como o GitHub trabalha somente com *releases*[4], este será o conceito usado para avaliação.

Para que uma mudança seja aprovada no GitHub, ela deve percorrer um fluxo de seis fases, conforme representado na Figura 2. Esse fluxo engloba desde a solicitação da mudança até a sua integração com o produto principal [11].

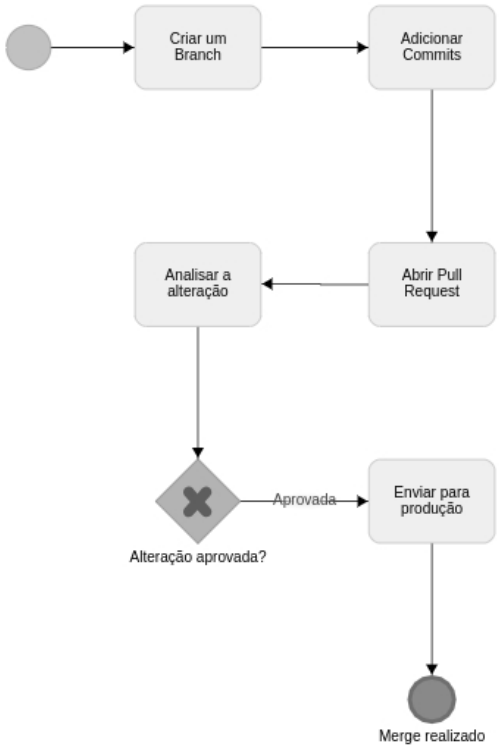

Figura 2: Fluxo da Mudança do GitHub (Adaptado de [11])

As fases do fluxo da mudança no GitHub são descritas a seguir:

(1) **Criar um *Branch*:** Para que uma nova mudança seja desenvolvida, é necessário que ela esteja em uma ramificação diferente e isolada do ramo do produto principal. Isto acontece porque o desenvolvimento desta mudança pode acarretar em *bugs* não só na própria modificação, como também em outra parte do produto. Desta forma, uma modificação é desenvolvida e testada isoladamente, protegendo outras funcionalidades de um possível impacto.

(2) **Adicionar *commits*:** Com o ambiente inicial devidamente preparado, a mudança é de fato desenvolvida. Para isso, *commits* são realizados todas as vezes artefatos do projeto são alterados, criando assim um histórico da mudança. Vale lembrar que cada *commit* deve ter uma mensagem clara o suficiente para indicar o que foi alterado, facilitando o entendimento de terceiros sobre o que está sendo desenvolvido. Outra prática interessante é referenciar *issues* que motivaram a realização de cada *commit*, porém esta prática não é obrigatória e, portanto, não é tão amplamente utilizada como deveria.
(3) **Abrir *Pull Request*:** Uma vez que a mudança já tenha sido devidamente implementada e testada, deve ser solicitado um *Pull Request*. Isto significa que a mudança será aberta para que outras pessoas possam visualizar e realizar outros testes, ou seja, a mudança será validada por outras pessoas e posteriormente poderá ser aprovada ou não. A aprovação de uma requisição pode representar a inclusão desta em uma *release*, entretanto, somente o dono do projeto pode aprová-la ou não.
(4) **Analisar a alteração:** Antes de a requisição de mudança ser aceita, esta será revisada por uma pessoa ou time responsável. Esta revisão tem como objetivo levantar dúvidas, inconsistências, falhas ou outros problemas que podem ocorrer. Isto é importante porque qualquer *bug* encontrado ainda é passível de correção, pois ainda não foi incorporado ao produto principal. Vale lembrar que devem ser abertas *issues* para qualquer problema encontrado.
(5) **Enviar para produção**: Caso o *Pull Request* seja aceito, ele deve ser enviado para produção separadamente do produto principal. Isto acontece porque durante a produção podem ser geradas *issues*, e caso isso ocorra, o processo de produção deve voltar ao produto original. A produção representa a última verificação em uma mudança antes de esta se juntar a uma nova versão do produto principal.
(6) ***Merge* realizado:** Caso a mudança passe pela produção, esta será incorporada (*merge*) ao produto principal. Vale ressaltar que independentemente de uma mudança ser aceita, esta ainda pode ser revogada posteriormente.

Com base no fluxo descrito acima, percebe-se que apesar das inúmeras funcionalidades, o GitHub não obriga o referenciamento de uma *issue* para que uma mudança seja iniciada, o que faz com que a documentação de tal mudança fique incompleta, dificultando a rastreabilidade e o gerenciamento da mesma.

4 TRABALHOS RELACIONADOS

Como um trabalho relacionado pode-se citar a pesquisa de da Silva Oliveira et al. [1], na qual é feita uma análise de ferramentas para o controle de versões de software segundo critérios sugeridos pelos autores, critérios esses baseados no processo de Gerência de Configuração do MPS.BR-SW. Para realizar tal análise, os autores compararam quatro ferramentas de controle de versão e puderam concluir que o Git foi a que atendeu o maior número de critérios. Porém, alguns critérios estão subjetivos em relação ao esperado dos resultados da GCS e também é feita uma análise comparativa sobre ferramentas. Um desses critérios, por exemplo, é a verificação da conexão com a internet para considerar que a ferramenta estabelece canais de segurança.

Outro trabalho que pode ser citado é o de Petrinja e Succi[18] que utiliza como base o modelo OMM para avaliar projetos *open source*. No contexto citado, os autores utilizaram seis projetos para realizar o estudo do OMM no universo FLOSS (*Free/Libre Open Source Software*), concluindo que o modelo é aplicável, além de trazer benefícios, não só para o universo citado como também para o desenvolvimento de *softwares* que serão integrados ao universo FLOSS futuramente. As notas para cada critério são dadas por avaliadores, o que demanda tempo e conhecimento dos mesmos na área.

Por fim, Khraiwesh [14] apresenta métricas para avaliar o processo de Gerência de Configuração com base no CMMI. O autor utilizou o modelo GQM (*Goal Question Metrics*) para avaliar sete práticas de GCS. Para validação das métricas, um questionário foi aplicado em seis empresas de desenvolvimento de *software*, resultando na coleta de 100 questionários respondidos por programadores, *designers* e analistas. Para apuração dos resultados de cada prática, Khraiwesh [14] utilizou o método *Alpha de Cronbach*, que analisa a distribuição dos valores obtidos como resultado e a sua confiabilidade. Esse método tem como resultado um valor entre 0 e 1, onde valores abaixo de 0.5 indicam que os itens avaliados são inconsistentes, sendo necessário valores acima de 0.5 para indicar que o resultado foi aprovado. Assim, após a análise dos coeficientes alpha, percebeu-se que todas as práticas obtiveram resultados acima de 0.698, indicando a aprovação de todas elas.

Neste trabalho, propõe-se a avaliação da maturidade do processo de Gerência de Configuração de projetos *open source*, por meio de métricas baseadas nos modelos de maturidade CMMI, MPS.BR e OMM. Essas métricas serão definidas quantitativamente e implementadas de forma automatizada, sem a necessidade de participação de colaboradores envolvidos nos projetos.

5 ESTUDO COMPARATIVO DA MATURIDADE DOS PROCESSOS DE GCS EM PROJETOS *OPEN SOURCE*

Nesta seção, serão apresentadas as definições e implementações das métricas sugeridas, os projetos selecionados para a avaliação, a descrição da automatização das medições e, por fim, a discussão e análise dos resultados.

5.1 Definição das métricas

Para realizar as medições quantitativas foram propostas métricas aplicadas automaticamente sobre dados de projetos *open source* armazenados no GitHub, sem a necessidade de consulta a especialistas da área ou integrantes dos projetos sob análise. Dessa forma, existe uma dependência direta dos dados fornecidos pela API de acesso à plataforma GitHub, que impacta diretamente o tipo de análises que podem ser realizadas.

Para cada um dos indicadores selecionados, foi proposta uma métrica, com uma fórmula envolvendo os conceitos mencionados na Seção 3, conforme ilustra a Tabela 4. Uma consideração a ser feita é que o símbolo # foi utilizado para representar a cardinalidade (número de elementos) de um conjunto. As métricas utilizam como

parâmetro o conceito de *release*, pois é o implementado no GitHub. A seguir são apresentadas explicações das métricas.

5.1.1 Métrica CM.SP 1.3. Na Subseção 2.1.1 foram listadas as práticas necessárias para atingir o indicador CM.SP 1.3 do CMMI. As práticas (1) e (4) já são atendidas uma vez que as ferramentas de controle de versão são usadas, porque constituem funcionalidades básicas. Já as práticas (3) e (4) são metrificáveis, por isso, a Métrica CM.SP 1.3 calcula a proporção de *releases* que são controladas (RC) verificando se todos os itens de configuração contidos nas mesmas estão sob gerência de configuração. Assim, o intuito é que a métrica seja suficiente para analisar a proporção de *releases* que estão de acordo com o sistema de gerência de configuração, ou seja, todos os itens de configuração que compõe uma *baseline* estiveram sob gerência de configuração por todo o ciclo de vida.

5.1.2 Métrica CM.SP 2.1. Esta métrica objetiva quantificar a proporção de solicitações da mudança que são rastreáveis, ou seja, possuem referências para uma ou mais *issues* de acordo com o descrito na Seção 2.1.1. Para isso, são contabilizados todos os *pull requests* que referenciam uma ou mais *issues*(PRI) e calcula-se a proporção destes dentre o conjunto total de *pull requests* (PRT).

5.1.3 Métrica CM.SP 2.2. Para que os itens de configuração sejam considerados sob gerência de configuração, como o detalhado na Seção 2.1.3, é necessário que ocorra controle de todas as questões relacionadas a estes itens. Dessa maneira, esta métrica verifica a proporção dos itens de configuração que estão nas *releases* (CR) e que passaram pelo ciclo completo da mudança (CPRI).

5.2 Implementação das métricas

Esta seção apresenta os pseudocódigos das métricas propostas. Inicialmente, o Algoritmo 1, apresentado a seguir, implementa a métrica CM.SP 1.3.

Algorithm 1 Métrica CM.SP 1.3

```
1:  procedure METRICA1
2:      PRT ← Lista de todos Pull Requests
3:      RT ← Lista de todas Releases
4:      CPR ← ∅                          ▷ Inicializa a lista vazia.
5:      RC ← 0     ▷ Inicializa o contador de Releases Controladas.
6:      for pull ← PRT do
7:          if pull.getStatus() = Aprovado then
8:              CPR ← pull.getShaCommits()
9:      for release ← RT do
10:         if release.getListaShaCommits() ⊂ CPR then
11:             RC ← RC + 1
12:     return (RC/#RT)
```

Esse algoritmo recebe como entrada duas listas: a PRT (lista contendo Todos os *Pull Request*) e RT (lista contendo Todas as *Releases*) de um determinado projeto. É feita uma filtragem dos *pull requests* aprovados da lista PRT e são adicionados sem repetição na lista CPR, incialmente vazia, os identificadores dos *commits* que compõe cada um. Logo após, para cada *release* da lista RT é feita a verificação se os *commits* da mesma estão contidos na lista CPR, ou seja, se todos *commits* da *release* pertencem à lista de *commits* de *pull request* e toda vez que atender a esta condição o contador de *releases* controladas RC é incrementado. Por fim, é calculada a quantidade de *releases* que são controladas (RC) em relação ao total de *releases* (RT).

A métrica CM.SP 2.1 é implementada pelo Algoritmo 2, que recebe como parâmetro uma lista contendo todos os *pull requests* (PRT). Depois disso, é executada uma iteração que verifica para cada *pull* se ele referencia pelo menos uma *issue*. Caso verdadeiro o contador PRI é incrementado. Por fim, é calculada a proporção de *pull requests* que referenciam uma *issue* (PRI) em relação ao total de *pull requests*.

Algorithm 2 Métrica CM.SP 2.1

```
1:  procedure METRICA2
2:      PRT ←Lista de todos Pull Requests
3:      PRI ← 0▷ Inicializa o contador do número de pull requests
    que referenciam issues.
4:      for pull ← PRT do
5:          if fazReferenciaIssue(pull) = Verdadeiro then
6:              PRI ← PRI + 1
7:      return (PRI/#PRT)
```

O Algoritmo 3 implementa a métrica CM.SP 2.2 descrita na Seção 2.1.3. Inicialmente, recebe-se uma lista de todos *pull requests* (PRT) e uma lista dos identificadores (*sha*) dos *commits* que compõe todas as *releases* (CR). Então, a lista PRT é percorrida e para cada *pull request* avalia-se sua aprovação e se o mesmo faz referência a pelo menos uma *issue*. Se atender a estes dois critérios, os identificadores dos *commits* do *pull request* são adicionados à lista CPRI. Logo após, é calculado o conjunto intersecção (IC) entre as listas CPRI e CR. Por fim, retorna-se a proporção de itens itens de configuração que passaram pelo controle completo da gerência da mudança em relação ao total de itens que compõe as versões liberadas (*releases*).

Algorithm 3 Métrica CM.SP 2.2

```
1:  procedure METRICA3
2:      PRT ←Lista de todos Pull Requests
3:      CR ←Lista do Sha dos Commits de todas as Releases
4:      CPRI ← ∅                         ▷ Inicializa a lista vazia.
5:      for pull ← PRT do
6:          if pull.getStatus() = Aprovado then
7:              if fazReferenciaIssue(pull) = Verdadeiro then
8:                  CPRI ← pull.getShaCommits()
9:      IC ← CPRI ∩ CR
10:     return (#IC/#CR)
```

5.3 Seleção dos projetos *open source*

Para a aplicação das métricas, foram selecionados 6 projetos armazenados no GitHub. Tais projetos abrangem diferentes tipos de proprietários: empresa, governo e comunidade. Esses projetos são explicitados a seguir:

- *Brackets* Desenvolvido pela Adobe, este projeto provê um editor de código aberto para a web, escrito em JavaScript,

Tabela 4: Indicadores selecionados de Gerência de Configuração

CMMI	Objetivo	Métrica	Explicação
CM.SP 1.3	Criar ou liberar *Releases* (*Baselines*).	M1 = RC/#RT, sendo RC = quantidade de *releases* controladas e RT = conjunto de todas as *releases*.	Verificar a proporção de *releases* que são controladas.
CM.SP 2.1	Rastrear solicitações de mudança.	M2 = PRI/#PRT, sendo PRI = quantidade *pull request* que referenciam issues e PRT = conjunto de todos *pull requests*.	Verificar a proporção de solicitações da mudança que são rastreáveis.
CM.SP 2.2	Controlar Itens de Configuração.	M3 = #(CPRI inter CR)/#CR, sendo CPRI = conjunto de *commits* de todos os pulls aprovados que apontam uma *issue* e CR = conjunto de *commits* de todas as *releases*.	Verificar a proporção de itens de configuração que passaram pelo ciclo de vida da mudança.

HTML e CSS. O projeto começou em 2011 e está disponível para contribuição, somando-se, até o momento, 358 contribuidores[5].

- *Material Design Icons* Conjunto de ícones de design da *Google*, dona do repositório. Inicializado em 2014, o projeto conta com 18 colaboradores, até o momento[7].
- SAPL Consiste em um Sistema de Apoio ao Processo Legislativo que começou em 2015 e conta com 20 colaboradores. O proprietário do repositório é o Interlegis, Programa de Modernização do Legislativo Brasileiro[8].
- *SciELO-Manager* Projeto de uma ferramenta de gerenciamento para catalogação de periódicos e artigos, disponível em um modelo SaaS (*Software as a Service)*. Atua como um *backbone* central de metadados para todos os sistemas e serviços do *SciELO (Scientific Electronic Library Online)*, programa responsável pelo projeto. O desenvolvimento do *SciELO-Manager* começou em 2011 e ainda está aberto para contribuições, possuindo, até o momento, 9 contribuidores [9].
- *Streama* Este projeto é um servidor de *streaming* de mídia auto hospedado que conta com 54 colaboradores. O *Strema* está disponibilizado para contribuição desde 2015[10].
- *Windows Community Toolkit* É uma coleção de funções auxiliares, controles personalizados e serviços de aplicativos. Ela simplifica e demonstra tarefas comuns do desenvolvedor, criando aplicativos UWP (*Universal Windows Platform*) para o *Windows* 10. Este projeto começou em 2016, somando neste período, 178 contribuidores[12].

5.4 Automatização das medições

Para automatização das métricas, foi desenvolvida uma aplicação na linguagem Java que utiliza a API GitHub para realizar as conexões com os repositórios de software. Uma das dificuldades de implementação foi a deficiência da API do GitHub em fornecer alguns dados. Assim, algumas das informações necessárias foram baixadas por linha de comando. Isto será melhor explicado a seguir.

Uma das informações necessárias para a realização desta pesquisa são os dados referentes às *releases*. Entretanto, a API não coleta a lista de *commits* que compõe uma *release*. Por este motivo, um *script* foi criado para realizar o download de cada uma das *releases* através de linha de comando, a partir dos nomes que são fornecidos pela API. Para cada uma delas, foi criado um arquivo JSON para ser manipulado na aplicação criada.

Depois disso, com a aplicação desenvolvida, e de posse de todos os dados, as métricas foram aplicadas sobre os mesmos e os resultados encontram-se descritos na próxima seção.

5.5 Discussão e Análise dos Resultados

Os resultados das métricas aplicadas nos processos dos projetos selecionados são apresentados na Tabela 5. As colunas estão divididas em Dados Brutos, Dados Processados e Resultado das Métricas, que por sua vez, são subdivididas em:

- Dados Brutos:
  - #IT - cardinalidade do conjunto que contém Todas as *Issues*;
  - #CT - cardinalidade do conjunto que contém Todos os *Commits*;
  - #PRT - cardinalidade do conjunto que contém Todos os *Pull Requests*;
  - #RT - cardinalidade do conjunto que contém Todas as *Releases*.
- Dados Processados:
  - PRI - Quantidade de *Pull Requests* que referenciam *Issues*;
  - #CPR - cardinalidade do conjunto que contém Todos os *Commits* dos *Pull Requests* aprovados;
  - #CPRI - cardinalidade do conjunto que contém Todos os *Commits* dos *Pull Requests* aprovados que referenciam *Issues*;
  - #CR - cardinalidade do conjunto que contém Todos os *Commits* das *Releases*.
- Resultado das Métricas:
  - M1 - Resultado da Métrica CM.SP 1.3;
  - M2 - Resultado da Métrica CM.SP 2.1;
  - M3 - Resultado da Métrica CM.SP 2.2.

Com os resultados obtidos, é possível perceber que a maioria dos valores para as métricas foi 0.00%, que representa um valor muito preocupante do ponto de vista das recomendações dos modelos de maturidade para os processos de Gerência de Configuração de Software. Na Figura 3, apresenta-se a proporção de processos em projetos cujo resultado atingiu valores não nulos para cada métrica, ou seja, processos que apresentaram algum nível de maturidade.

Tabela 5: Informações dos 6 projetos e resultados obtidos para cada uma das métricas.

Projeto	Dados Brutos				Dados Processados				Resultado das Métricas		
	#IT	#CT	#PRT	#RT	PRI	#CPR	#CPRI	#CR	M1	M2	M3
Brackets	9141	17701	5423	110	586	16894	1239	261	0.00%	≈ 10.81%	≈ 4.60%
Material Design Icons	774	124	91	16	8	37	0	117	0.00%	≈ 8.79%	0.00%
SAPL	1609	4369	796	140	498	2638	1926	285	0.00%	≈ 62.56%	≈ 0.70%
SciELO-Manager	496	2853	937	87	161	2818	241	250	≈ 1.15%	≈ 17.18%	7.20%
Streama	557	1406	140	50	18	1100	249	250	0.00%	≈ 12.86%	34.00%
Windows Community Toolkit	1466	8278	1218	23	67	9128	156	250	0.00%	≈ 5.50%	0.00%

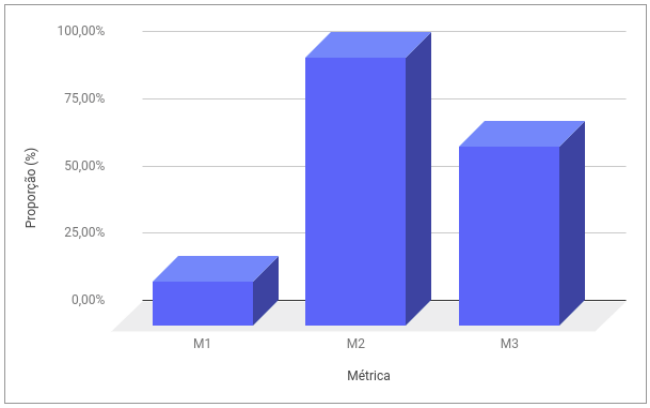

Figura 3: Proporção de resultados acima de 0% para cada uma das métricas.

Na Figura 3, é possível verificar que a Métrica CM.SP 1.3 (M1), que calcula a proporção de *baselines* que são controladas, obteve os piores resultados, sendo que somente o projeto *SciELO-Manager* não apresentou resultados nulos. Ainda assim, o processo desse projeto alcançou 1.15% de maturidade para o indicador CM.SP 1.3, ou seja, apenas 1 (uma) das 87 *releases* estava sob gerência de configuração. Isso se deve ao fato de que para ser considerada controlada, uma *release* deve ter todos os itens controlados (*commits*). Assim, se ao menos um único item não for controlado, a *release* não é contabilizada como controlada.

Já a métrica CM.SP 2.1 (M2) obteve 100% dos resultados não nulos. Assim, podemos perceber que a prática de referenciar *issues* nos *pull requests* é a mais adotada, apesar de ainda ficar aquém do recomendado. Somente o SAPL atingiu um resultado satisfatório, cerca de 63% de maturidade no indicador CM.SP 2.1. Um dos fatores possíveis para este resultado é que na descrição do projeto uma das práticas sugeridas é a de solicitar *pull request* somente após a criação de uma *issue*. Como a Métrica CM.SP 2.1 avalia a quantidade de *pull requests* que referenciam pelo menos uma *issue* e os contribuidores do projeto são encorajados a seguir esta prática, o projeto atingiu resultados satisfatórios.

Outra prática ausente nos processos dos projetos avaliados é a gerência dos itens de configuração. Pode-se perceber isso com o resultado da Métrica CM.SP 2.2 (M3), pois apesar de, aproximadamente, 66.67% dos projetos terem obtido resultados positivos, o melhor deles recebeu nota 34%. Isto mostra que grande parte dos itens de configuração dos projetos não passam por todo o ciclo da GCS, não sendo possível rastrear as mudanças ocorridas de forma completa.

Na Figura 4, tem-se um gráfico do nível de maturidade do processo de desenvolvimento para cada um dos 6 projetos avaliados. Nela é possível observar que o SAPL é o projeto que destaca dos demais, porém ele só obteve este resultado por causa do resultado no indicador CM.SP 2.1 (M2 no gráfico), já que para os outros indicadores ele não conseguiu alcançar o mesmo nível. Por outro lado, o projeto *Windows Community Toolkit* foi o pior avaliado em todas as métricas. Este resultado pode ser explicado pelo fato de que os colaboradores são incentivados a utilizar o *Stack Overflow* caso tenham dúvidas.

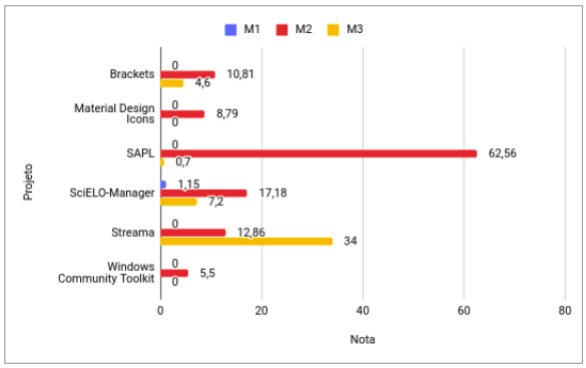

Figura 4: Nível de maturidade dos processos de 6 projetos *open source* para os indicadores estudados.

Um fator que pode ser responsável pelos resultados obtidos é o forte relacionamento dos indicadores selecionados. Ou seja, caso não seja possível realizar a rastreabilidade da mudança pela falta de documentação (CM.SP 2.1), os itens de configuração não serão considerados controlados (CM.SP 2.2) e, por consequência, as *releases* não serão consideradas controlas (CM.SP 1.3). Em razão disso, as métricas são diretamente impactadas, pois se os *pull requests* aprovados não referenciam *issues* (M2), isso reduz o conjunto de *commits* que podem ser avaliados na Métrica CM.SP 2.2 (M3). Por consequência, o número de *releases* totalmente controladas (M1) é também reduzido.

6 CONCLUSÃO E TRABALHOS FUTUROS

O objetivo deste artigo foi avaliar a maturidade dos processos de Gerência de Configuração por meio de métricas quantitativas baseadas nos modelos de maturidade CMMI, MPS.BR e OMM. As métricas foram aplicadas para se avaliar dados de 6 projetos *open source* armazenados no GitHub.

No geral, os resultados obtidos mostraram que o processo de desenvolvimento do GitHub deixa a responsabilidade de praticar a GCS nas mãos dos colaboradores. Em razão disso, as amarrações fracas entre as solicitações das mudanças e as documentações dos itens de configuração impactam diretamente nos resultados dos projetos. Isso porque, a falta de um controle mais rígido, no que compete ao ciclo de vida da mudança, dificulta, ou até impossibilita, a rastreabilidade da mudança do código fonte.

Em virtude dos fatos mencionados, percebe-se que com pequenas mudanças no processo do GitHub o processo de GCS poderia ser melhor controlado. Com a adição de um campo obrigatório para referenciamento de pelo menos uma *issue* durante a criação de um *commit*, por exemplo, já seria suficiente para melhorar a rastreabilidade da mudança.

Como trabalhos futuros, sugere-se uma análise considerando um maior número de projetos com características distintas, tais como: maior número de colaboradores, mais populares, com diferentes tipos de donos dos repositórios (empresa privada, governo, comunidade, entre outros), para que se possa analisar os impactos dessas características na maturidade dos processos praticados. Outra possibilidade de trabalho futuro seria avaliar dados de outra ferramenta de controle de versão e comparar os resultados com os obtidos para o GitHub. Dessa forma, seria possível analisar a influência do fluxo de trabalho das ferramentas na maturidade dos processos adotados.

REFERÊNCIAS

[1] Danne da Silva Oliveira, Heitor Costa, and Paulo Afonso Parreira Júnior. 2016. Análise de Ferramentas para Controle de Versões de Software no Contexto do MPS. BR. *I Workshop sobre Aspectos Sociais, Humanos e Econômicos de Software (WASHES 2016)* (2016).

[2] W. F. FERREIRA. 2009. MPS.BR – um estudo do modelo MPS.BR como beneficio para as pequenas e médias empresas. Retrieved Dez 12, 2018 from http://www.softwarepublico.gov.br/file/17234990/MONOGRAFIA_WILKER_-_MPS.BR.pdf

[3] GitHub. 2018. Build software better, together. Retrieved Nov 11, 2018] from https://github.com

[4] GitHub Inc. 2018. About Releases - User Documentation. Retrieved Nov 07, 2018 from https://help.github.com/articles/about-releases/

[5] GitHub Inc. 2018. adobe/brackets: An open source code editor for the web, written in JavaScript, HTML and CSS. Retrieved Nov 28, 2018 from https://github.com/adobe/brackets

[6] GitHub Inc. 2018. GitHub Glossary - User Documentation. Retrieved Nov 07, 2018 from https://help.github.com/articles/github-glossary/

[7] GitHub Inc. 2018. google/material-design-icons: Material Design icons by Google. Retrieved Nov 28, 2018 from https://github.com/google/material-design-icons

[8] GitHub Inc. 2018. interlegis/sapl: Sistema de Apoio ao Processo Legislativo. Retrieved Nov 11, 2018 from https://github.com/interlegis/sapl

[9] GitHub Inc. 2018. scieloorg/scielo-manager: Management tool for cataloging journals and articles, available in a SaaS model. It acts as a central metadata backbone for all the systems and services of SciELO. Retrieved Nov 28, 2018 from https://github.com/scieloorg/scielo-manager

[10] GitHub Inc. 2018. streamaserver/streama: Self hosted streaming media server. Retrieved Nov 28, 2018 from https://github.com/streamaserver/streama

[11] GitHub Inc. 2018. Understanding the GitHub flow · GitHub Guides. Retrieved Nov 11, 2018] from https://guides.github.com/introduction/flow

[12] GitHub Inc. 2018. windows-toolkit/WindowsCommunityToolkit: The Windows Community Toolkit is a collection of helper functions, custom controls, and app services. It simplifies and demonstrates common developer tasks building UWP apps for Windows 10. The toolkit is part of the .NET Foundation. Retrieved Nov 28, 2018 from https://github.com/windows-toolkit/WindowsCommunityToolkit

[13] CMMI Institute. 2018. Instituto CMMI - Modelos pré-definidos - níveis de maturidade e capacidade. Retrieved Nov 11, 2018 from https://cmmiinstitute.com/products/cmmi/cmmi-v2-products/appendices/predefined-model-views-maturity-and-capability-l

[14] Mahmoud Khraiwesh. 2017. Configuration Management Measures in CMMI. *International Journal of Applied Engineering Research* 12, 18 (2017), 7546–7557.

[15] Alexis Leon. 2015. *Software configuration management handbook*. Artech House.

[16] Etiel Petrinja, Ranga Nambakam, and Alberto Sillitti. 2009. Introducing the opensource maturity model. In *Proceedings of the 2009 ICSE workshop on emerging trends in free/libre/open source software research and development*. IEEE Computer Society, 37–41.

[17] Etiel Petrinja, Alberto Sillitti, and Giancarlo Succi. 2010. Comparing OpenBRR, QSOS, and OMM assessment models. In *IFIP International Conference on Open Source Systems*. Springer, 224–238.

[18] Etiel Petrinja and Giancarlo Succi. 2012. Assessing the Open Source Development Processes Using OMM. *Advances in Software Engineering* 2012 (2012), 1–17. https://doi.org/10.1155/2012/235392

[19] Qualipso. 2018. OMM. Retrieved Nov 07, 2018 from http://qualipso.icmc.usp.br/OMM/

[20] Risto Salo, Timo Poranen, and Zheying Zhang. 2015. Requirements management in GitHub with a lean approach.. In *SPLST*. 164–178.

[21] Softex 2018. Guias – Softex. Retrieved Nov 01, 2018 from https://www.softex.br/mpsbr/guias/#toggle-id-7

[22] Ian SOMMERVILLE. 2011. *Engenharia de Software* (9nd ed.). Vol. I. Pearson Prentice Hall, São Paulo, BR.

[23] T View. 2005. IEEE Standard for Software Configuration Management Plans. *IEEE Std* 2005 (2005), 1–19.

[24] wibas 2018. CMMI. Retrieved Nov 01, 2018 from ttps://www.wibas.com/cmmi/

[25] wibas 2018. Configuration Management (CM) (CMMI-DEV). Retrieved Nov 01, 2018 from ttps://www.wibas.com/cmmi/configuration-management-cm-cmmi-dev