Políticas de Gerência de Configuração de Software para Grupos de Pesquisa

Felipe Fernandes da Silva[1],
Aline Maria Malachini Miotto Amaral[1], **Thelma Elita Colanzi**[1]

[1]Universidade Estadual de Maringá (UEM)
Maringá – PR – Brazil

felipe_ffs6@hotmail.com, ammmamaral@uem.br, thelma@din.uem.br

Abstract. *Considering an increasingly dynamic scenario that requires constant change, today's software development requires people and organizations to control the quality of artifacts generated throughout their process. Changes in software artifacts can occur for a variety of factors, whether for correcting functionality or adapting to new business demand. Besides, such artifacts are generally elaborated collaboratively what makes that in order to ensure their quality it is necessary the adoption of Software Configuration Management (SCM) practices. Computing research group environments resemble in many aspects software development environments, in which the main artifact produced is software. These software, elaborated over several researches and by many researchers, need to have their quality controlled. In this sense, the objective of this work is to propose a set of SCM policies for research group environments. It is expected that with the implementation of policies such as the proposals the quality of software developed in academic environments can be improved.*

Resumo. *Considerando um cenário cada vez mais dinâmico que requer mudanças constantes, o desenvolvimento de software atual exige que pessoas e organizações controlem a qualidade dos artefatos gerados ao longo de seu processo. Alterações nos artefatos de software podem ocorrer por vários fatores, seja para corrigir a funcionalidade ou adaptar-se à nova demanda de negócios. Além disso, esses artefatos geralmente são elaborados de forma colaborativa, o que faz com que, para garantir sua qualidade, seja necessária a adoção de práticas de Gerenciamento de Configuração de Software (GCS). Os ambientes dos grupos de pesquisa em computação se assemelham em muitos aspectos aos ambientes de desenvolvimento de software, nos quais o principal artefato produzido é o software. Esses softwares, elaborados em várias pesquisas e por muitos pesquisadores, precisam ter sua qualidade controlada. Nesse sentido, o objetivo deste trabalho é propor um conjunto de políticas de GCS para ambientes de grupos de pesquisa. Espera-se que, com a implementação de políticas como as propostas, a qualidade do software desenvolvido em ambientes acadêmicos possa ser melhorada.*

1. Introdução

Ao longo do ciclo de vida de um projeto de software uma grande quantidade de itens de informação é produzido. É possível criar documentos, código-fonte, manuais e estes podem ser alterados por correções, adaptações do funcionamento e novas implementações.

Com a finalidade de não perder o controle do projeto em consequência das mudanças, é necessário que estas mudanças sejam devidamente controladas e gerenciadas [Figueiredo et al. 2004]. Para lidar com estas complexidades um processo denominado Gerência de Configuração de Software (GCS) foi estabelecido [Crnkovic et al. 2003].

A GCS pode ser definida como o controle da evolução de sistemas complexos ou, de forma mais pragmática, como a disciplina que permite manter produtos de software em evolução sob controle, e, portanto, contribuindo para satisfazer restrições de qualidade e de cronograma [Estublier 2000].

Em um ambiente de desenvolvimento de projetos de pesquisa na área de Computação, a dificuldade de gerenciar artefatos e mudanças é grande. A falta de recursos financeiros, humanos e principalmente de tempo para as atividades de gerenciamento impactam fortemente o dia a dia dos desenvolvedores de sua equipe. Sempre com prazos curtos para suas entregas, pesquisadores de grupos de pesquisa muitas vezes negligenciam as atividades conferidas pela gerência de configuração. Cabe ressaltar que os artefatos produzidos por trabalhos de grupos de pesquisa, normalmente tem continuidade em diversas pesquisas e diferentes pesquisadores, o que faz com que a falta da adoção de práticas de GCS seja um risco para a continuidade de muitos projetos.

Dentro do contexto apresentado, o presente trabalho teve como objetivo a definição Políticas de Gerência de Configuração para ambientes de desenvolvimento de grupos de pesquisa. Além disso, também foi realizado um estudo de caso aplicando as políticas definidas em um ambiente real de grupos de pesquisa.

Este trabalho está organizado da seguinte forma: a Seção 2 apresenta os fundamentos que guiram o desnvolvimento deste trabalho. Na Seção 3 as políticas de GCS propostas são descritas e a Seção 4 demonstra a aplicação das proposta deste trabalho no contexto de um grupo de pesquisa. Finalmente na Seção 5 são apresentadas as conclusões e trabalhos futuros.

2. Contextualização

Esta seção descreve os principais conceitos e características inerentes à Gerência de Configuração (GCS) necessários para a definição das politicas de GCS apresentadas neste trabalho.

2.1. Gerência de Configuração

A gerência de configuração é a arte de identificar, organizar e controlar modificações em um software que está sendo construído por uma equipe de desenvolvimento, com o objetivo de maximizar a produtividade minimizando erros e coordenando o desenvolvimento de software para gerir a confusão produzida pelas muitas mudanças que ocorrem ao longo do desenvolvimento de um software [Leon 2015].

De acordo com [Scott and Nisse 2001], a implementação de um processo de gerência de configuração de sucesso requer planejamento, entendimento do contexto e estrutura organizacional e as relações entre os elementos organizacionais.

A GCS atua ao longo de todo o ciclo de desenvolvimento do software, desde a solicitação de uma alteração até a auditoria dos itens entregues ao final do desenvolvimento [Pressman 2011]. Nesse sentido, pode-se identificar cinco principais ati-

vidades: Identificação dos itens de configuração; Controle de versão; Controle de alteração/mudança; Auditoria; e Relatos das alterações.

Estes itens podem ser documentos, casos de teste, código-fonte, seção de especificações de requisitos criados durante a análise, ferramentas de desenvolvimento, bibliotecas de código de terceiros, documentação da instalação, manutenção, operação ou uma parte de modelo de projeto.

Na GCS, o controle de versões é um sistema que registra as mudanças feitas em um arquivo ou um conjunto de arquivos ao longo do tempo, de forma que se possa recuperar versões específicas, podendo ser utilizado para versionar praticamente qualquer tipo de arquivo em um computador. Para oferecer suporte automatizado a esta atividade foram desenvovidos softwares de controle de versão, que são também conhecidos como CVS (*Concurrent Version System*) [Chacon and Straub 2014].

De acordo com [Sink 2011], existem operações e conceitos que foram implementados em todos os sistemas de controle de versão, porém cada sistema implementa à sua maneira dentro de sua estrutura, são eles:

- *Commit*: um *commit* corresponde a formalização de uma ação no banco de dados do sistema de controle de versão.
- *Pull/Update*: é responsável por realizar a operação de atualização de conteúdo local, aplicando as alterações no repositório e realizando *merging* com as alterações realizadas, quando existem alterações.
- *Checkout/Clone*: trata-se de *checkout* um processo para obter uma cópia de um projeto a partir do repositório onde ele se encontra, a cópia criada é chamada de cópia de trabalho.
- *Push*:A ação de *push* corresponde ao repasse de alterações do repositório local, onde as ações de *commits* foram realizados para um repositório principal.
- *Merge*: Trata-se da junção de trechos que estão divergentes, ou seja, é a fusão de dois *branches* de desenvolvimento.
- *Add*: corresponde a operação de adição que é utilizada quando se tem um arquivo ou diretório que ainda não está sob o controle de versão e deseja adicionar o repositório.
- *Diff*: A operação *diff* corresponde as alterações realizadas na cópia de trabalho.

Além do controle de versões, a GCS também é caracterizada pelo sistema de controle de mudanças. Este sistema é encarregado de executar a função de controle da configuração de forma sistemática, armazenando todas as informações geradas durante o andamento das solicitações.

Segundo Sommerville [Sommerville 2011], o processo de gerenciamento de mudanças inicia quando um cliente submete uma solicitação de mudança (CR *change request*) que descreve a mudança requerida. Este cliente é um *Stakeholder*, que não faz parte da equipe de desenvolvimento, podendo até ser um cliente interno, como a equipe responsável pelos testes do software ou a equipe de marketing, por exemplo.

Por fim, a Auditoria de gerência de configuração visa complementar possíveis revisões técnicas que foram feitas, de modo a garantir a consistência das alterações e do versionamento.

2.2. Git

O Git [Spinellis 2012] é um software livre, liberado sob a licença GNU GPL [GPL 2017], para controle de versão distribuído. Criado por Linus Torvalds, o Git nasceu em 2005 e surgiu de uma necessidade específica de armazenamento do núcleo do Linux (Sistema operacional), que é um projeto de código aberto e razoavelmente grande.

Segundo [Loeliger and McCullough 2012], os objetivos iniciais da implementação do Git eram: desempenho rápido e eficiente, manter integridade e confiança, imutabilidade, transações atômicas, suporte robusto a desenvolvimento não linear (milhares de braços paralelos), facilitar o desenvolvimento distribuído, *Design* interno organizado, repositórios completos e reforçar a responsabilidade.

Por conta dessas características, o Git ganhou um grande espaço no mercado, tendo, segundo [Duck 2017], 47% de usuários, superando, assim, o SubVersion que conta com 40% de usuários. Porém, muito do sucesso do Git se deve ao GitHub [Loeliger and McCullough 2012], pois este permitiu que grandes corporações colocassem seus projetos em repositórios públicos para que desenvolvedores de todas as partes do mundo pudessem contribuir para os projetos, criando uma comunidade de mútua colaboração e desenvolvimento.

GitHub: conforme [Loeliger and McCullough 2012], já existem diversas plataformas para o Git, mas o GitHub se destaca entre elas. GitHub é uma ferramenta de hospedagem de código-fonte com controle de versão utilizando o Git. O desenvolvimento da plataforma começou em 2007. Os projetos no GitHub podem ser acessados e manipulados usando a interface de linha de comando Git padrão e todos os comandos Git padrão funcionam com ele.

3. Políticas de GCS para ambiente de grupos de pesquisa

Esta seção descreve as políticas de GCS criadas para ambientes de grupos de pesquisa, por meio da utilização das ferramentas Git e GitHub.

3.1. Descrição do cenário

As políticas propostaso terão como base a utilização de duas ferramentas, o Git e o GitHub.

Considerando um ambiente em que as ferramentas já estejam configuradas e instaladas, é necessário primeiramente realizar a definição e criação do repositório para o projeto de acordo com a estratégia definida pela organização.

Após a criação do repositório, o mesmo deverá ser configurado de acordo com a estratégia de desenvolvimento que se deseja abordar. No processo aqui definido, sempre que um *merge* for solicitado, tal solicitação deverá ser feita por meio de um *pull request*. A funcionalidade *pull request* é disponibilizada pelo GitHub e nada mais é do que uma solicitação de *merge* realizada pelo desenvolvedor quando finalizou o desenvolvimento de sua alteração e deseja integrar estas alterações ao *branch* principal, para que este esteja disponível na versão de liberação para os clientes.

3.2. Papéis do Processo

Segundo Dart [Dart 1990], um cenário operacional para aplicação da gerência de configuração em uma empresa envolve os papéis de gerente de projeto, gerente de

configuração, engenheiros de software e usuários. No contexto de um projeto de pesquisa, esses papéis não existem. Sendo assim, outros devem ser criados para que possa haver a distribuição das responsabilidades do desenvolvimento.

Este processo conterá apenas três papéis para que seja realizado em sua completude, quais sejam:

Professor orientador: responsável pela criação das solicitações de mudanças, criação do repositório para cada nova versão do software que estará disponível; e definição de cronograma de entregas.
Pesquisador responsável: este papel pode ser realizado por um aluno de mestrado, doutorado ou por um professor; responsável por efetuar revisões técnicas nas alterações realizadas pelo desenvolvedor e caso a alteração seja aprovada, realizar o *merge* da mesma, resolvendo possíveis conflitos.
Pesquisador desenvolvedor: reposável por realizar as alterações e correções conforme solicitado pelo professor orientador e pesquisador responsável após a revisão do artefato modificado.

A Figura 1 demonstra estes papéis, suas relações e as tarefas envolvidas no processo de gerência de configuração definido para grupos de pesquisa conforme descritos anteriormente.

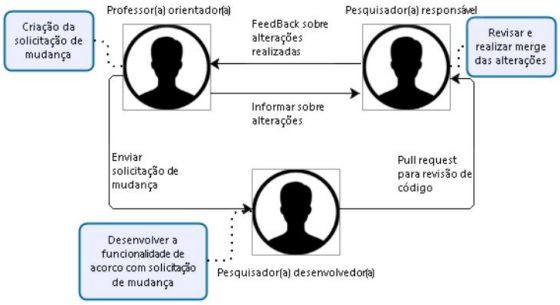

Figura 1. Papéis e tarefas envolvidos no processo de Gerência de Configuração

3.3. Identificação dos Itens de Configuração

A primeira tarefa no processo de gerência de configuração consistem em identificar os itens que devem ser gerenciados. Inicialmente, é necessário identificar quais itens são produzidos pelo processo de desenvolvimento do grupo de pesquisa. Código-fonte, requisitos e testes são exemplos.

Alguns aspectos devem ser considerados para a determinação dos itens: os itens são utilizados por um ou mais grupos de trabalho?; os itens sofram alterações ao longo do tempo? Os itens são críticos ou de importância elevada para o projeto? Os itens são dependentes uns dos outros?

Os itens de configuração identificados são os artefatos que serão passíveis de gerência de configuração, logo, são os arquivos que deverão ser armazenados no repositório remoto. Cada item armazenado no sistema de controle de versão passa a possuir um número de revisão e a cada alteração persistida no sistema de controle de versão este número é alterado. Além da revisão, outros atributos devem ser armazenados, tais como: data e hora, o usuário que realizou a ação, o tipo de ação realizada (inclusão, alteração ou

remoção de um arquivo no repositório), além de uma mensagem que pode ser adicionada pelo usuário que realizou a ação.

3.4. Gerenciamento de Solicitação de Mudanças

O controle de modificação consiste em gerenciar o ciclo de vida de uma mudança desde a sua criação até a sua conclusão. O GitHub possui um mecanismo de registro de solicitação de mudanças denominado *Issues*, e, por meio deste mecanismo, os requisitos e problemas devem ser cadastrados e acompanhados através dos estados de seu ciclo de vida.

Uma solicitação de mudança deve conter: título; descrição da solicitação de mudança; tipo - classificada de acordo com um padrão definido pela organização (por exemplos: Correção, Melhoria, Implementação, etc.); e versão.

Cada *Issue* criada no GitHub possui um número, gerado automaticamente e este pode ser utilizado como referência para o nome do *branch* que será criado para desenvolver determinada funcionalidade.

3.5. Desenvolvimento das Funcionalidades

Após o recebimento da solicitação de alteração, o pesquisador desenvolvedor realizará a implementação das funcionalidades requisitadas.

Para desenvolver a funcionalidade solicitada, o pesquisador desenvolvedor deverá criar localmente um *branch*, gerando assim uma linha de desenvolvimento separada da versão principal, que após a finalização da implementação da funcionalidade conterá a versão atual do software, adicionada a alteração realizada.

O grupo de pesquisa deve estabelecer um padrão para a criação de nomes de *branches*, a fim de que tais nomes sejam reconhecidos nas revisões. Para tanto, deverá ser utilizado o número da *Issue* como identificador para que possa ser possível identificar a origem da mudança e o que a mudança solicita.

Conforme a implementação da funcionalidade evolui, é possível efetivar as alterações localmente atravéspor meio do comando *commit*. A realização de *commits* frequentes é importante para manter as alterações efetivadas no repositório de modo que não se reverta alguma alteração incorretamente, perdendo-se uma implementação importante.

Após a finalização do desenvolvimento da funcionalidade, as alterações deverão ser sincronizadas com o repositório remoto e posteriormente deverá ser aberto o *pull request*, finalizando assim todo o processo de desenvolvimento.

A Figura 2 apresenta o processo em um contexto geral para o desenvolvimento das funcionalidades na visão do pesquisador desenvolvedor.

3.6. Revisão de Código e *Merge* das Mudanças

Com o processo de desenvolvimento finalizado, é imprescindível que se realize uma revisão de código para identificar possíveis falhas ou dúvidas do pesquisador responsável que surgiram em decorrência da revisão. Caso o código falhe na revisão, o pesquisador desenvolvedor deve ser avisado sobre as falhas e como corrigi-las. Este cadastro deverá ser realizado no *pull request* do código alterado.

Após o código passar na revisão de código, será necessário realizar o *merge* das alterações realizadas pelo desenvolvedor e, neste ponto, podem haver conflitos entre o

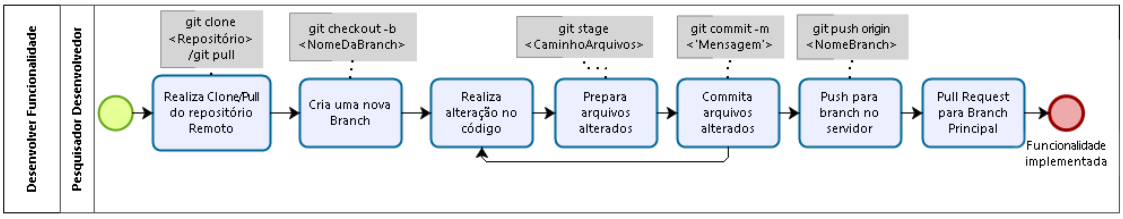

Figura 2. Fluxo do desenvolvimento de novas funcionalidades na ferramenta na visão do pesquisador desenvolvedor

código presente no *branch* de desenvolvimento e as alterações solicitadas pelo desenvolvedor. O pesquisador responsável deverá resolver estes conflitos, de maneira que o código fique coerente e funcional para a ferramenta.

Após a realização da revisão, o *branch* criado para o desenvolvimento deverá ser excluído, pois o mesmo não se faz mais necessário. A Figura 3 demonstra o fluxo do processo de revisão de código e merge das funcionalidades enquanto que a Figura 4 apresenta o processo de desenvolvimento de funcionalidades adaptado para contemplar a revisão de código.

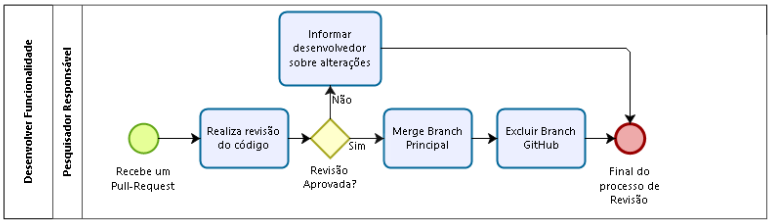

Figura 3. Fluxo do revisão de código e merge de novas funcionalidades na ferramenta na visão do pesquisador responsável

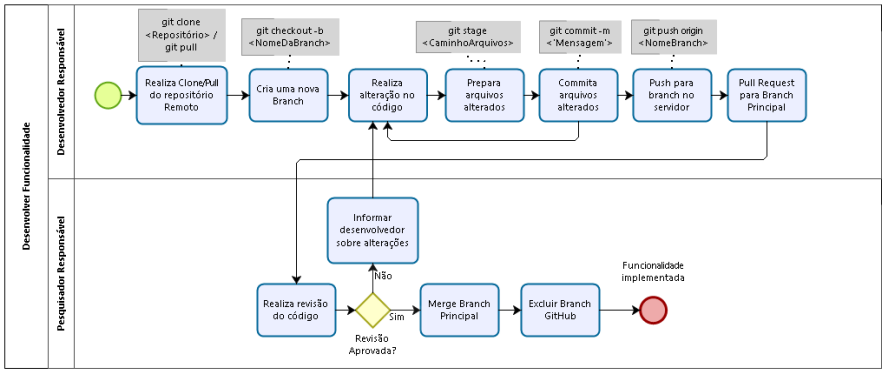

Figura 4. Fluxo de desenvolvimento, revisão de código e merge de novas funcionalidades na ferramenta no contexto geral

3.7. Novas Versões

Considerando o ambiente de grupos de pesquisa, uma nova versão deverá ser disponibilizada após seis meses de desenvolvimento. Será utilizado o seguinte padrão: No fechamento da primeira versão do software no ano, após seis meses de desenvolvimento, será atribuído a versão *Major* o valor do ano corrente, e a versão *Minor* o valor 1. Para abertura da segunda versão no ano, após mais seis meses de desenvolvimento, o valor para a versão *Major* irá se manter e somente a versão *Minor* será incrementada, passando

a assumir o valor 2. Este padrão irá refletir não só na versão do software como também no nome dos *branchs* que devem ser criados para cada versão, para que esta possa ser considerada como finalizada.

Para isso, quando finalizado o processo de desenvolvimento e testes de todas as funcionalidades de uma versão, deverá ser aberto um novo *branch* no qual será retirada a versão oficial a partir do *branch* de desenvolvimento, com o número da versão sendo referenciada. Este processo garante que uma versão antiga não será perdida por sobrescrita de *branches* no GitHub. Essa abertura de versões deverá ser realizada pelo professor orientador ou pelo pesquisador responsável.

A Figura 5 demonstra a como é definida a abertura de versões, e os nomes a serem utilizados para cada versão e cada *branch*.

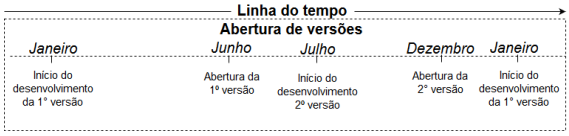

Figura 5. Período de início do desenvolvimento e abertura de versões

Ao realizar uma correção em uma versão, a mesma deverá ser replicada para as versões seguintes e posteriormente, no *branch* de desenvolvimento, para que essas correções mantenha a integridade nas próximas versões que serão abertas. A figura 6 demonstra a maneira correta de propagar as correções realizadas em versões que o desenvolvimento já foi finalizado.

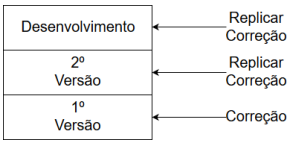

Figura 6. Esquema de propagação das correções realizadas em versões

4. Relato de Experiência: Implantação das Políticas de GCS

Considerando as políticas definidas, este relato de experiência apresenta a implantação destas políticas de GCS em um ambiente de grupo de pesquisa que tem como foco a realização de pesquisas em SBSE (Engenharia de Software Baseada em Busca).

Para atuar neste campo foi criado dentro do Departamento de Informática da Universidade Estadual de Maringá um grupo de pesquisa que tem como objetivo o desenvolvimento de ferramentas de suporte a problemas de SBSE. No referido grupo de pesquisa atuam professores, alunos de graduação e alunos de mestrado da Universidade Estadual de Maringá. A principal ferramenta desenvolvida por este grupo é denominada OPLA-Tool [Féderle et al. 2015].

Durante o desenvolvimento da OPLA-Tool não houve uma preocupação com processos de gerência de configuração. O processo de desenvolvimento de uma nova funcionalidade passava por um processo não estruturado e não padronizado, permitindo que cada aluno desenvolvedor atuasse da maneira que achasse conveniente. Ao final do desenvolvimento todos as alterações de código eram enviados a um aluno de mestrado que era responsável por realizar *merges* manuais para que todas as funcionalidades desenvolvidas fossem integradas a uma única versão do software, o que gerava um grande esforço e aumentava o risco de problemas no software.

Ao realizar a identificação dos itens de configuração, foi constatado apenas um tipo de artefato que seria relevante para ser gerenciado pelo controle de versão, sendo este o código-fonte. Assim sendo, o código fonte do projeto foi versionado em um repositório público no GitHub, e este repositório foi configurado para que desenvolvedores possam contribuir com o projeto de maneira ordenada por meio do processo definido neste trabalho.

Os papéis foram definidos exatamente como proposto na seção 3.2, ficando sobre responsabilidade dos professores a criação das solicitações de mudança que devem ser abertas no GitHub, bem como a abertura das versões para liberações oficiais. As revisões de código e *merges* ficaram sob responsabilidade de um aluno de mestrado (mais experiente e designado pelos professores orientadores) que possui conhecimento prévio da ferramenta. O desenvolvimento foi alocado a alunos que desenvolveram os trabalhos de conclusão de curso com foco na OPLA-Tool, bem como alunos de iniciação científica e mestrado que fazem parte do projeto. Estes fizeram uso do Git para realizar o versionamento dos artefatos. A abertura de novas versões foi realizada como descrito na Figura 7 e a a propagação de correções foi definida conforme apresentado na Figura 8.

Figura 7. Período de início do desenvolvimento e abertura de versões definidos para OPLA-Tool

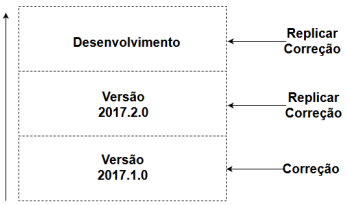

Figura 8. Esquema de propagação das correções realizadas em versões anteriores definidos para OPLA-Tool

5. Considerações Finais

O presente trabalho teve como objetivo propor políticas de GCS para ambientes de grupos de pesquisa. Para tanto foi realizada uma pesquisa sobre o que é e como ocorre o processo de gerência de configuração, bem como identificado, como funciona, em geral, o ambiente de desenvolvimento de grupos de pesquisa, em especial o grupo de pesquisa em SBSE do Departamento de Informática da Universidade Estadual de Maringá.

Dentro deste contexto, foi definido um processo de gerência de configuração segundo o que propõe a literatura. Esse processo propõe mudança nos papéis necessários para implantação de GCS em relação aos especificados na literatura e modifica as tarefas para que possam ser executadas por estes papéis. Também especifica os fluxos de desenvolvimento que devem ser aplicados, propondo revisões técnicas como parte da garantia da qualidade do software e como deve ser a abertura de novas versões do software bem como o fluxo de correção de problemas.

Como trabalhos futuros pretende-se aplicar o processo de GCS definido em ambientes de grupo de pesquisa diferente do grupo de SBSE, para permitir uma melhor

adequação das políticas de GCS já definidas, bem como a definição de novas práticas, relevantes para o contexto de grupos de pesquisa. Além disso, pretende-se elaborar um processo de *deploy* automatizado com entrega contínua para a ferramenta OPLA-Tool e implantar testes automatizados como pré-requisito para realização do *Merge* no GitHub.

Referências

Chacon, S. and Straub, B. (2014). *Pro git*. Apress, New York, NY, USA.

Crnkovic, I., Asklund, U., and Dahlqvist, A. P. (2003). *Implementing and integrating product data management and software configuration management*. Artech House, Norwood.

Dart, S. (1990). Spectrum of functionality in configuration management systems. Technical Report CMU/SEI-90-TR-011, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA.

Duck, B. (2017). Compare repositories.

Estublier, J. (2000). Software configuration management: A roadmap. *In ICSE '00 Conference on The Future of Software Engineering*, pages 279–289.

Féderle, E. L., do Nascimento Ferreira, T., Colanzi, T. E., and Vergilio, S. R. (2015). Oplatool: A support tool for search-based product line architecture design. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 370–373, New York, NY, USA. ACM.

Figueiredo, S., Santos, G., and Rocha, A. R. (2004). Gerência de configuração em ambientes de desenvolvimento de software orientados a organização. *Simpósio Brasileiro de Qualidade de Software, Brasília*.

GPL, G. (2017). General public licence.

Leon, A. (2015). *Software configuration management handbook*. Artech House, Norwood.

Loeliger, J. and McCullough, M. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development*. O'Reilly Media, Sebastopol, 2 edition.

Pressman, R. S. (2011). *Engenharia de software: uma abordagem profissional*. Amgh Editora, Porto Alegre, 7 edition.

Scott, J. A. and Nisse, D. (2001). *Guide to Software Engineering Body of Knowledge*. IEEE Computer Society Press, Livermore.

Sink, E. (2011). *Version Control by Example*. Pyrenean Gold Press Champaign, IL, 1 edition.

Sommerville, I. (2011). *Engenharia de software*. Pearson Prentice Hall, São Paulo, 9 edition.

Spinellis, D. (2012). Git. *IEEE Software*, 29(3):100–101.