Implementando o Gitflow para Gerencia de Configuração em um Projeto de Desenvolvimento de Software Ágil: Um Relato de Experiência

Luis Rivero
PPGCC/UFMA
São Luis, MA, Brasil
luis.rivero@ufma.br

Italo Silva
DEINF/UFMA
São Luis, MA, Brasil
francyles@nca.ufma.br

Pedro Cutrim
DEINF/UFMA
São Luis, MA, Brasil
thiagocutrim@nca.ufma.br

Domingos Dias
PPGCC/UFMA
São Luis, MA, Brasil
domingos.adj@gmail.com

Geraldo Braz Junior
PPGCC/UFMA
São Luis, MA, Brasil
geraldo.braz@ufma.br

Anselmo Paiva
PPGCC/UFMA
São Luis, MA, Brasil
paiva@nca.ufma.br

Erika Alves
Equatorial Energia S/A
São Luis, MA, Brasil
erika.assis@equatorialenergia.com.br

Milton Oliveira
Equatorial Energia S/A
São Luis, MA, Brasil
milton.oliveira@equatorialenergia.com.br

ABSTRACT

In software engineering, Software Configuration Management is a set of support activities that allows for the orderly absorption of changes inherent to software development. For that, organization models for code versioning like Gitflow have been proposed. In Gitflow, two fixed branches (master and develop) are used to store the project history and be the starting point for changes. Despite the popularity of Gitflow for being considered a simple workflow, there are few: (a) reports of its use in practice and / or (b) documentation on how to deploy it in a real environment. This paper presents the process of adapting Gitflow and creating rules for its application in a real software development project. This adaptation took into account the opinions of managers and developers of a team of approximately 30 people within an agile Scrum life cycle. As a result, definitions and documents were generated to keep track of development, in addition to defining the necessary steps for its application considering the development process adopted by the team.

PALAVRAS-CHAVE

Gerência de Configuração, GitFlow, Processo de Desenvolvimento, Relato de experiência.

1 INTRODUÇÃO

Na engenharia de software, o processo de desenvolvimento de software envolve várias etapas e vários profissionais. Durante a análise, projeto, implementação, avaliação, manutenção e implantação do software, vários artefatos de software são criados e consumidos. Consequentemente, à medida que o time cresce e/ou se dispersa é necessário controlar quem trabalha em que parte do software [1]. Nesse contexto, a gerência de configuração de software é uma atividade abrangente que ocorre ao longo de todo o ciclo de vida de um software, e que pode ajudar a resolver problemas relacionados com [2]: (a) o aumento da introdução de defeitos à medida que alterações são efetuadas sem uma analise; (b) atrasos em entregas; (c) degradação da qualidade do software; e (d) custos com retrabalho. A gerência de configuração permite identificar e controlar modificações, garantir a implementação adequada das modificações e relatar as modificações a outros que possam ter interesse [3]. Entre as principais vantagens da aplicação de conceitos de gerência de configuração podem ser citadas [1]: (a) permitir que os diversos envolvidos na criação e manutenção do software tenham acesso ao histórico destas modificações; (b) fornecer subsídios para o entendimento do sistema na sua forma atual, e também nas suas formas anteriores; e (c) melhorar a qualidade do software.

Para alcançar os objetivos propostos acima, é necessário controlar a evolução do software, através do controle de versões, solicitações de mudanças e construção do software [4]. Nesse contexto, o controle de versões combina procedimentos e ferramentas para gerir diferentes versões de itens de configuração que são criados durante o processo de software. Um item de configuração é cada elemento criado durante o processo de engenharia de software, ou que seja necessário para este processo; e pode ser um arquivo, uma aplicação corporativa, uma parte de um documento, uma sequência de casos de teste, entre outros [5]. Um sistema de controle de versões pode ser utilizado para disponibilizar [6]: (a) um banco de dados dos itens de configuração relevantes; e (b) uma forma de coletar todos os itens de configuração relevantes e construir uma versão especifica do software. Nesse contexto, o Git é um dos sistemas modernos de controle de versão mais adotado, visto que apresenta código aberto maduro e com manutenção ativa [7].

Existem diferentes formas de trabalhar com o Git, considerando diferentes contextos de desenvolvimento e as especificidades dos projetos de software [6]. Portanto, modelos de fluxo de trabalhos têm sido desenvolvidos para orientar equipes de desenvolvimento na realização de atividades como [8]: (a) criação de *branches* (ramificações), isto é, inicia uma nova linha de desenvolvimento em paralelo a uma principal já existente; (b) execução de *push*, que é

usado para enviar conteúdo do repositório local para um repositório remoto; ou (c) execução de *merges*, que faz a fusão de dois arquivos que estão sendo alterados simultaneamente por pessoas diferentes, garantindo que a versão final contenha todas as alterações.

Entre os modelos existentes, um dos modelos que tem se destacado por sua facilidade de implementação é o *Gitflow* [9]. O *Gitflow* é um modelo fortemente baseado em ramificações, mas focado em entregas de projetos que define os papéis de cada ramificação e como eles devem interagir [10]. Entre as ramificações utilizadas, são sugeridas: ramificações de funcionalidades, ramificações de lançamento e ramificações de manutenção. O *Gitflow* tem sido adotado em vários projetos de desenvolvimento [11–13]. No entanto, os artigos relatando a aplicação do *Gitflow* não descrevem com detalhes como o mesmo pode ser aplicado, citando apenas sua utilização. Além disso, existem poucos relatos da aplicação deste modelo de fluxo de trabalho no contexto de desenvolvimento ágil, que tem ganhado popularidade no últimos anos [14]. Diante do exposto, este artigo relata uma experiência prática de adoção do modelo *Gitflow* em um contexto de desenvolvimento ágil real. Uma subequipe de gerentes e desenvolvedores analisou e adaptou este modelo no desenvolvimento de um sistema inteligente para análise e predição de resultados de litígios e sugestão de tratativas em uma empresa de energia multinacional. O processo de desenvolvimento Scrum adotado pela equipe de desenvolvimento de software foi adaptado para incorporar as atividades do *Gitflow*. Neste artigo, são apresentadas as adaptações incorporadas ao processo de desenvolvimento e as lições aprendidas desta adaptação, do ponto de vista de gerentes e desenvolvedores envolvidos na implantação do *Gitflow*.

Este artigo está organizado da seguinte forma. A seção 2 fornece conceitos relacionados ao *Gitflow*, bem como trabalhos relacionados. Na seção 3, é apresentado o contexto de desenvolvimento ágil do sistema inteligente e as ações definidas no contexto do *Gitflow*. Por sua vez, a Seção 4 apresenta a implantação do *Gitflow* e os resultados alcançados. Finalmente, a Seção 5 apresenta as conclusões e perspectivas futuras deste trabalho.

2 BACKGROUND

2.1 O Modelo *Gitflow*

Segundo Driessen [9], o *Gitflow* surgiu como uma alternativa para o controle de versões em projetos de desenvolvimento utilizando o Git. O seu objetivo é manter a integridade do código e permitir que pessoas trabalhem de forma colaborativa no mesmo projeto. A Figura 1 apresenta os diferentes elementos do *Gitflow*. Neste modelo, existem duas ramificações para registrar o histórico do projeto. A ramificação principal (*master*) armazena o histórico do lançamento oficial, e a ramificação de desenvolvimento (*develop*) serve como uma ramificação de integração para recursos. Também é conveniente marcar todas as confirmações na ramificação principal com um número de versão. Além disso, existem as ramificações de recursos (*features*). Cada novo recurso deve residir na própria ramificação, a qual pode ser enviada por *push* para o repositório central para backup/colaboração. No entanto, em vez de serem ramificações da ramificação principal, as ramificações de recurso usam a ramificação de desenvolvimento como ramificação pai. Quando um recurso é concluído, ele é mesclado de volta na ramificação de desenvolvimento. Ainda existem as ramificações de lançamento (*release*) que contêm recursos suficientes para um lançamento. Quando estiver pronta para ser lançada, a ramificação de lançamento é mesclada com a ramificação principal e marcada com um número de versão. Finalmente, é possível ter ramificações de correções prioritárias (*hotfix*), que são usadas para manter os lançamentos de produção de forma eficiente.

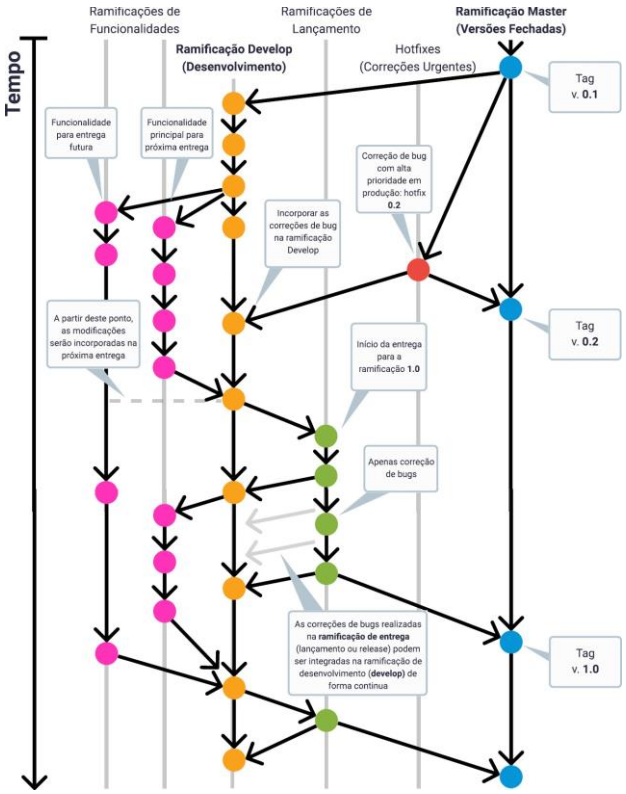


Figura 1: Representação Gráfica do *Gitflow* adaptada de Driessen [9]

Ao utilizar o *Gitflow*, uma equipe de desenvolvimento pode obter as seguintes vantagens [9]: ganhar produtividade; garantir um estado limpo das ramificações em qualquer momento do ciclo de vida do projeto; aumento da facilidade de compreensão das ramificações, visto que estas seguem um padrão sistemático; e aumento da eficácia quando é necessário ter várias versões do sistema em produção. Devido ao potencial do *Gitflow* como modelo de fluxo de controle de versões, várias equipes de desenvolvimento têm utilizado o mesmo em diversos contextos de desenvolvimento [15]. Na subseção a seguir, são descritos estes trabalhos e a necessidade de fornecer um exemplo prático de adoção do *Gitflow*.

2.2 Estudos sobre o *Gitflow*

Segundo Phillips et al. [16] existem vários tipos de modelos de controle de fluxo de ramificações com Git. Por exemplo, o *GitHub flow* oferece a possibilidade de trabalhar em uma ramificação *master* e criar ramificações locais ara manter o controle de alterações. Apesar

de permitir a integração contínua e facilitar a manutenção de uma única versão em produção, este modelo pode ser tornar instável e pode não ser adequado quando há necessidade de manter ramificações de lançamento de versões do produto em desenvolvimento. Por sua vez, o *GitLab flow* é um modelo que combina desenvolvimento orientado a *features* e controle de defeitos em funcionalidades. Além de sugerir o uso de uma ramificação *master*, este modelo sugere o uso de ramificações do tipo *feature* onde devem ser feitas as modificações. Na ramificação *master* devem ser feitas as correções de defeitos e esta versão é a que será entregue ao cliente. Apesar de permitir a integração contínua e permitir um histórico mais limpo, o *GitLab flow* é mais complexo quando for necessário manter mais de uma versão em produção. Por sua parte, o modelo *One flow* é uma alternativa ao modelo *Gitflow* que permite que uma versão de lançamento do sistema seja baseada apenas em outras versões de lançamento, não possuindo uma ramificação *master*. Com isso, este modelo se forna mais flexível e permite obter um histórico limpo quando é necessário manter uma única versão em produção. No entanto, não é sugerido em projetos com integração contínua ou quando é necessário manter mais de uma versão em produção.

Considerando o exposto, o modelo *Gitflow* tem se destacado, levando à sua adoção devido aos benefícios citados na Seção 1. Além disso, o *Gitflow* tem evoluído para outros modelos, como o *GitWaterFlow* [15], que permite lidar com problemas relacionados com o suporte de versões antigas do produto e fornecer entregas urgentes de correções de defeitos. Apesar da criação dos novos modelos, poucos são os exemplos da sua aplicação e é necessário entender sua versão original para facilitar sua implantação.

Entre os trabalhos que aplicaram o *Gitflow*, pode ser citado o trabalho de Wa'el Mohsen et al. [17]. Os autores aplicaram o *Gitflow* como parte de uma metodologia de desenvolvimento baseada nos frameworks Scrum e Nexus. No entanto, não foram apresentados os detalhes do impacto do *Gitflow* no processo de desenvolvimento, ou as etapas realizadas para sua implantação. Outros trabalhos similares têm aplicado o *Gitflow* como parte do processo de desenvolvimento de sistemas *open-source* [18], aplicações de geolocalização [19], ou na implantação de ferramentas [20]. Estes relatos, da mesma forma, também não apresentaram detalhes sobre o processo seguido para implantar o modelo *Gitflow*.

Por sua vez, há trabalhos que relatam a aplicação do *Gitflow* em contextos de desenvolvimento variados voltados para testes ou linha de produto de software. Por exemplo, o trabalho de Konrad et al. [11], apresenta a aplicação prática do *Gitflow* num contexto de desenvolvimento Scrum, utilizando o repositório para construir e executar testes de forma automatizada. Já Montalvillo e Diaz [21] fazem uso do *Gitflow* para gerar um modelo de boas práticas no contexto de linhas de produto de software.

Além dos trabalhos citados, há relatos informais da utilização do *Gitflow* em trabalhos de conclusão de curso ou repositórios de desenvolvimento de software. Por exemplo, Wikström [12] aplicou um estudo de caso cujo objetivo foi encontrar os benefícios e desafios relacionados com a implementação de integração contínua no desenvolvimento de software no contexto de uma empresa finlandesa de software. Finalmente, Bretal [13] desenvolveu uma aplicação web para centralizar, organizar e automatizar projetos de identificação de fitoplâncton. Apesar de terem aplicado o *Gitflow*, os trabalhos se concentram na fundamentação teórica e não mostram os resultados alcançados das experiências.

De um modo geral, é visível a popularidade que o *Gitflow* tem ganho nos últimos anos, seja para a aplicação prática e/ou como base no desenvolvimento de soluções alternativas em diversos contextos de desenvolvimento de software. Apesar disso, os relatos identificados pelos autores deste trabalho não foram suficientes para entender o que seria preciso para implantar o *Gitflow* em um processo de desenvolvimento real, muito menos considerando aspectos específicos de um processo de desenvolvimento ágil. O relato deste tipo de experiência pode ser útil para equipes de desenvolvimento novatas que visem aplicar o *Gitflow* e adaptar atividades de desenvolvimento e artefatos para sua implantação. Nesse contexto, a seção a seguir, apresenta o contexto de desenvolvimento ágil do sistema para o qual foi adotado o *Gitflow*.

3 CONTEXTO DE DESENVOLVIMENTO

O objetivo deste artigo é apresentar um relato de experiência da aplicação do *Gitflow* no desenvolvimento de um sistema de aprendizagem de máquina (*Machine Learning* - ML). Portanto, quanto à abordagem metodológica aplicada nesta pesquisa, podemos classificá-la como uma pesquisa descritiva, uma vez que nosso objetivo é observar, relatar e analisar as experiências de uso do *Gitflow* em um ambiente de desenvolvimento ágil [22]. Os detalhes para caracterizar o contexto de desenvolvimento são explicados a seguir.

3.1 Aplicação Desenvolvida

O mercado de energia em diversos países está passando por mudanças devido ao surgimento de mecanismos legais e gerenciais de proteção ao consumidor [23]. No Brasil, a necessidade de mudanças no mercado de empresas de energia elétrica tem se tornado mais evidente nos últimos anos, principalmente devido ao surgimento de ações judiciais contra essas empresas. Nesse contexto, os clientes costumam entrar com ações judiciais quando estão insatisfeitos com o tratamento e/ou serviço prestado por uma empresa e desejam ser tratados com justiça. Identificar clientes que provavelmente entrarão com uma ação judicial contra a empresa é considerado uma identificação de insatisfação do cliente [24]. Essa insatisfação pode levar a perdas monetárias consideráveis, incluindo a saída de alguns clientes da empresa, além dos custos gerados nos julgamentos. Melhorar a qualidade dos serviços prestados é uma das principais formas de diminuir as ações judiciais dos clientes.

Métodos preditivos foram desenvolvidos para manter o cliente e evitar processos judiciais. A previsão de mudança de serviço é um problema comum, não apenas em empresas de distribuição de energia, mas em empresas que prestam serviços [25, 26]. No contexto do fornecimento de energia brasileiro, as soluções de ML têm sido propostas para investigar a relação entre a satisfação do cliente e as ações judiciais impetradas. Por exemplo, um modelo de satisfação do cliente foi desenvolvido usando modelos de equações estruturais [27]. Os autores do trabalho citado entrevistaram 300 participantes de 62 instalações elétricas brasileiras para avaliar o modelo, mostrando que melhorar a qualidade dos serviços oferecidos é a principal forma de minimizar o número de ações judiciais por parte dos clientes. Outro trabalho propôs uma técnica

de tomada de decisão baseada na Teoria dos Conjuntos Aproximados para extrair regras de decisão importantes relacionadas à rotatividade de clientes [25].

Embora muitos trabalhos proponham formas de tratar os clientes e prever possíveis ações ou abandono de clientes, que podem sair da empresa ou mesmo entrar com ações judiciais contra ela, eles não fornecem uma ferramenta para uma análise individual ou conjunta do cliente. Uma relação conjunta de grupos de clientes precisa ser explorada, uma vez que as características dos grupos de clientes em certas regiões podem ser cruciais na decisão de processos judiciais, especialmente quando se trata de serviços essenciais, como serviços de energia elétrica. O sistema ML para as empresas de energia preverem ações judiciais relacionadas a consumo não registrado e queda de energia foi solicitado como entrega pela concessionária Equatorial Energia. Na subseção a seguir, apresentamos mais detalhes sobre o contexto de desenvolvimento da aplicação.

3.2 Caracterização do Projeto

O projeto de desenvolvimento foi denominado Sistema Jurídico Inteligente para Análise e Predição de Resultados de Litígios e Sugestão de Tratativas (SiJurI). Este projeto está sendo realizado em um período de 18 meses, iniciando em abril de 2020 até o presente momento de submissão deste artigo. Ao todo, a equipe de desenvolvimento contava com 30 engenheiros de software, entre eles: gerentes de projetos, especialistas em ML, desenvolvedores de software, desenvolvedores de interface gráfica, analistas de software, administradores de banco de dados e testadores de software.

Para desenvolver o sistema ML, a equipe de desenvolvimento aplicou a metodologia ágil Scrum. Scrum é um dos processos de desenvolvimento ágil mais populares na comunidade de desenvolvimento brasileira [28, 29]. O Scrum é focado em planejamento, e as equipes que o usam normalmente seguem algumas práticas que os levam a uma definição sobre o que será implementado em um *release* e, após a devida priorização, o que será implementado como parte de uma iteração (*sprint*). Ao usar o Scrum, espera-se que as equipes definam um *backlog* do produto, uma lista com todos os requisitos pendentes para um projeto, dimensionados com base na complexidade, dias ou alguma outra unidade de medida que as equipes decidam. Nesse sentido, a equipe deste projeto escreveu frases simples para cada requisito dentro do *backlog* do produto, algo que foi usado pela equipe para iniciar discussões e detalhar o que é necessário para ser implementado pela equipe.

Na metodologia Scrum, algumas funções são definidas: (a) *product owner*, responsável por ser a voz do negócio dentro de um projeto Scrum; (b) Equipe Scrum, formada por seus desenvolvedores, testadores e outras funções dentro do projeto; e (c) Scrum Master, que é responsável por manter a equipe focada nas práticas e valores que precisam ser aplicados dentro do projeto e também é responsável por ajudar a equipe sempre que enfrentar algum problema durante o processo. Em nosso contexto, o *product owner* era um gerente dentro da empresa de energia Equatorial Energia. Ele teria acesso a diversos setores da empresa e possibilitaria reuniões com futuros usuários do sistema, para que os requisitos do sistema fossem discutidos e definidos. O *Scrum master* era um desenvolvedor experiente em ML com mais de 5 anos de experiência no desenvolvimento de modelos de ML. Finalmente, a equipe Scrum era composta por engenheiros de software experientes (mais de 3 anos de experiência) e inexperientes (alunos de graduação em estágio).

Para definir quais entregas seriam produzidas e avaliadas, foram definidas metas para cada *sprint* com relação ao desenvolvimento do modelo de ML e da aplicação Web embutindo o modelo. Assim, as seguintes atividades foram definidas para o projeto: (a) Definição de escopo e arquitetura (*Sprints* 1-3); (b) Desenvolvimento de Base de Dados (*Sprints* 4-10); (c) Desenvolvimento do Módulos para Geração de Relatórios e Tomada de Decisão (*Sprints* 4-14); (d) Desenvolvimento de Interface de Usuário que integraria os dados e produziria relatórios (*Sprints* 08-18); e (e) Teste, Integração e Liberação (*Sprints* 6-18). Após definir as principais atividades a serem executadas no projeto, a equipe de desenvolvimento seguiu as práticas do Scrum. A reunião de planejamento do *Sprint* ocorreu no início de cada *Sprint*. Durante a *Sprint*, a equipe trabalhou em diversas atividades de acordo com o planejamento do projeto. Para acompanhar as decisões de design e requisitos para cada *Sprint*, a equipe utilizou um quadro *Kanban* online, que tem colunas para categorizar as tarefas em: "a fazer", "fazendo", e "concluído". Diagramas de atividades e apresentações foram desenvolvidos a cada *sprint* para documentar os requisitos e o estado atual do projeto. Além disso, foram realizadas reuniões diárias para avaliar problemas durante o desenvolvimento do software.

Conforme descrito acima, durante a atividade de Definição de escopo e arquitetura, quando os modelos de ML e modelos de interação começaram a ser desenvolvidos, foi sugerida a implantação de um modelo de fluxo de trabalho com Git para gerenciamento de configuração. Nesse contexto, a equipe de desenvolvimento decidiu adotar o *Gitflow*, devido aos pontos positivos citados na Seção 2. Além disso, o cenário de desenvolvimento dinâmico e a necessidade de entregas para o cliente, foram fatores decisivos para que a equipe optasse pela utilização deste modelo. Neste artigo, serão relatados os passos que foram necessários para a implantação do *Gitflow* pela organização, além de mostrar o resultado final desta implantação considerando a nova organização do ambiente de trabalho e a opinião dos profissionais que participaram do processo de implantação. A seguir, apresentamos o processo de definição de etapas e artefatos adotados pela organização para a implantação do *Gitflow*.

4 IMPLANTAÇÃO DO *GITFLOW*

4.1 Adaptação do Processo e Artefatos

Como indicado acima, o processo de implantação do *Gitflow* foi sugerido devido a vários problemas enfrentados pela equipe em projetos anteriores. A equipe de desenvolvimento tinha experiência com vários projetos na área de ML. No entanto, a organização enfrentava um problema de rotatividade de profissionais. Além disso, a volatilidade de requisitos dificultava a manutenção de projetos antigos, uma vez que os profissionais que trabalhavam em um projeto, não estavam mais na equipe quando era necessário retomá-lo ou realizar manutenções. Consequentemente, os gerentes do projeto encarregaram a uma sub-equipe a implantação do *Gitflow*. Inicialmente, a proposta era fazer a adoção de um processo de gerenciamento de configuração mais robusto. No entanto, uma mudança radical não seria possível, considerando que a equipe

estava, no momento da implantação, trabalhando em diversas atividades do projeto. Portanto, foi feita a sugestão de adotar mudanças gradativas no processo de desenvolvimento.

Para decidir quais mudanças seriam implementadas, a equipe de gerentes, em conjunto com um subconjunto dos desenvolvedores de maior experiência, realizaram diversas reuniões com sessões de *brainstorming* (chuva de ideias) para definir quais riscos/dificuldades eram frequentes nos projetos. Entre os principais problemas, a equipe destacou: (a) dificuldade de encontrar os arquivos corretos nos diretórios da equipe; (b) dificuldade em identificar quem era o responsável pelas modificações dos artefatos; (c) dificuldade de manter informações sobre resultados obtidos nas *sprints* em termos de funcionalidades desenvolvidas; e (d) dificuldade em manter o controle das versões e a garantia da qualidade das mesmas em termos de execução de testes. Após a identificação dos principais problemas, a subequipe designada analisou as regras específicas do *Gitflow* e as atividades corriqueiras do processo de desenvolvimento ágil adotado pela equipe de desenvolvimento de um modo geral.

Com base nestas informações a equipe propôs um processo em que os gerentes, desenvolvedores e equipe de teste teriam atividades designadas relacionadas ao controle de versões dos sistemas em desenvolvimento, seguindo sugestões do *Gitflow* adaptadas às necessidades e realidade da equipe. A Figura 2 apresenta o processo proposto. Para manter o controle das diversas ramificações do sistema em desenvolvimento, foram criadas contas no Git, de modo que ficassem atribuídas as responsabilidades de cada membro da equipe quanto à alimentação do repositório. Além disso, foram designados códigos e estruturas para manter o controle das ações dos membros da equipe. Por exemplo, ao criar uma nova ramificação do tipo *feature*, o desenvolvedor teria de usar a seguinte nomenclatura: "Create Branch - Feature - Nome da Feature". Instruções e estruturas similares foram adotadas ao longo do novo processo definido.

Outro problema importante relacionado com a frequência de atualização dos arquivos e o registro dos responsáveis pelas modificações foi resolvido a partir de instruções específicas de atualização de cadastro e sobre como proceder com a execução de ações como: *push*, *pull*, análise de conflitos, entre outros. Adicionalmente, os membros da equipe foram instruídos a registrar informações relevantes para os gerentes ao longo do clico de vida do projeto. Para isso, um arquivo leia-me foi instanciado, indicando campos e instruções de dados a serem especificados ao longo do projeto. O processo simplificado na Figura 2, apresenta estas sugestões na forma de observações, templates e atividades. Este processo foi validado pelos membros do time para posteriormente ser implantado na equipe. Para a implantação foi preparado um tutorial e um exemplo prático da sua utilização em um contexto fictício.

4.2 Resultados

A Figura 3 apresenta um dos arquivos leia-me criados para um dos módulos do sistema de ML em desenvolvimento. Como pode ser observado, a partir da implantação do *Gitflow* e a melhoria dos artefatos de registro, foi possível ter um novo meio de controle de informações dos módulos do sistema. Entre estas informações, foram registrados dados sobre as funcionalidades desenvolvidas na *sprint*, a versão atual do sistema, registro das configurações necessárias para utilização do módulo, instruções para *Build* e *Deploy*, além de links com acesso a informações externas consideradas importantes.

Por sua vez, a Figura 4 apresenta um exemplo de como ocorreu o registro das atividades dos membros da equipe na ferramenta de controle de versões Git. Neste exemplo, é possível visualizar o fluxo de trabalho de dois desenvolvedores. Inicialmente, *master* e *develop* criam suas ramificações correspondentes, inicializando o projeto. A partir deste ponto, os desenvolvedores podem trabalhar em ramificações de funcionalidades separadas para serem acrescentadas à ramificação *develop*. Quando ocorre uma junção com a ramificação *develop*, os desenvolvedores conversam entre si para resolver conflitos. Além disso, neste exemplo também é apresentada a possibilidade de dois desenvolvedores trabalharem juntos em uma ramificação de uma funcionalidade específica, quando for preciso. Para isso, uma nomenclatura específica foi adotada conforme o processo definido na Seção 4.1, indicando quem é o desenvolvedor que está trabalhando em qual parte da funcionalidade, permitindo um melhor controle das atividades dos membros da equipe. Ao final do processo, ocorrem os *merges* nas ramificações *feature*, *develop* e *master*, respectivamente. O processo adotado em conjunto com a criação de templates e artefatos a serem atualizados permitiu que a equipe pudesse melhorar o controle de versões. Como resultado deste relato de experiência, ainda foi elaborado um tutorial sobre como implantar o *Gitflow* na organização no contexto ágil.

Com o objetivo de obter feedback sobre a adoção do *Gitflow* no contexto de desenvolvimento atual pela equipe de desenvolvimento, foi preparado um questionário contendo perguntas baseadas em indicadores de aceitação de tecnologia [30]. As questões aplicadas foram: (a) Qual é a sua opinião sobre a utilidade da abordagem proposta para a aplicação do *Gitflow*?; (b) Qual é a sua opinião sobre a facilidade de uso da abordagem proposta para a aplicação do *Gitflow?*; (c) Na sua opinião, quais são os problemas que a abordagem proposta pode trazer para a aplicação do *Gitflow* no seu contexto de trabalho?; (d) Quais sugestões você indicaria para melhorar a abordagem proposta?; e (e) Você pretende utilizar a abordagem proposta?. Considerando estas questões, dois desenvolvedores e um gerente que participaram do processo inicial de implantação da abordagem foram questionados.

Com relação à utilidade da abordagem, houve comentários relacionados à organização implantada a partir da adoção do *Gitflow*, além da obtenção de informações necessárias para a gerência.

"*... vejo o Gitflow como esse paradigma para um melhor controle de versionamento. A ferramenta já existe e é muito boa, mas faltava um protocolo procedimental e acredito que o GITFlow vem como uma solução para preencher essa lacuna.*" - Desenvolvedor 1

"*Sem dúvida, o ponto mais forte de aplicar o Gitflow é a utilidade ser bem maior do que apenas utilizar o git de forma despadronizada.*" - Desenvolvedor 2

"*Para mim, sem dúvida ter as informações estruturadas de quem fez o que, e quando o fez, me permite tomar melhores decisões no projeto. Portanto, esta proposta fornecerá indícios para melhor gerenciar a equipe e tomar decisões sobre que módulos precisam de mais suporte.*" - Gerente

Com relação à facilidade de uso, houve críticas quanto ao aprendizado dos passos necessários para a aplicação do *Gitflow*. O gerente indicou que esta dificuldade pode ser negativa a curto prazo, mas

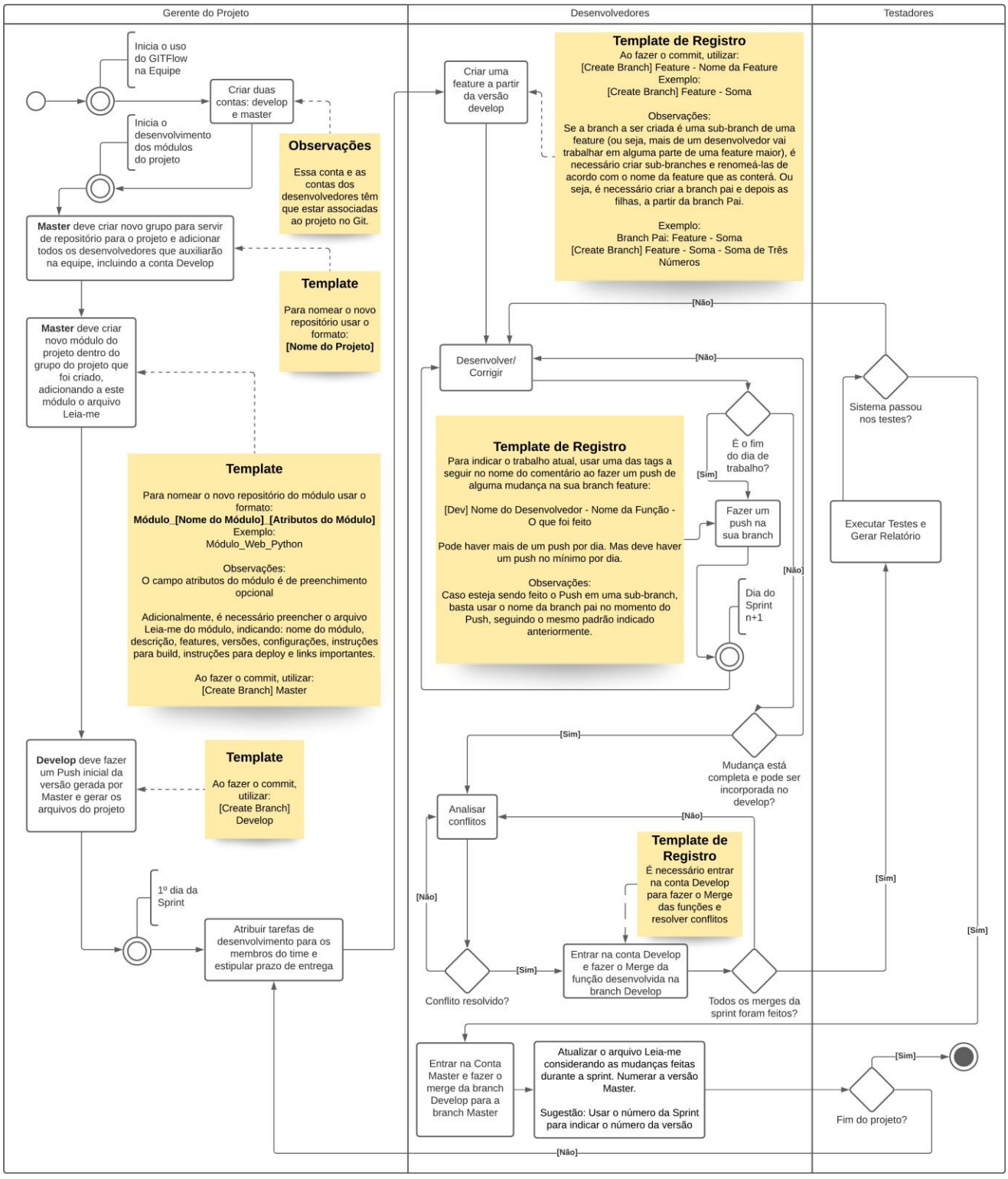
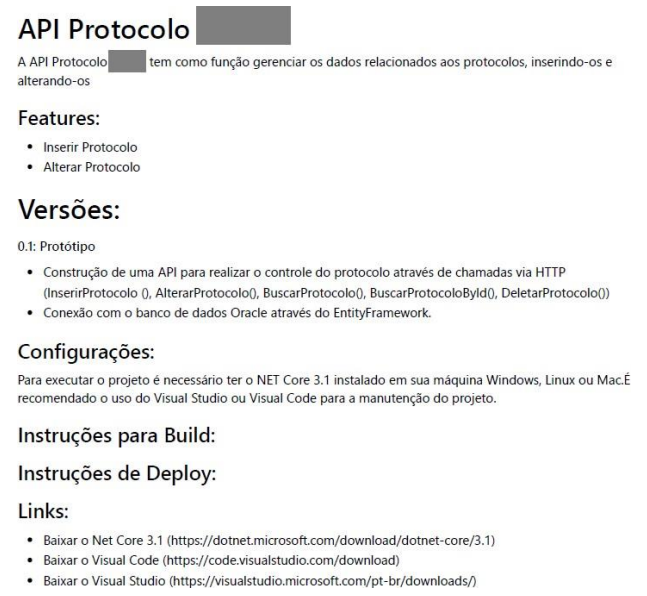


Figura 2: Processo desenvolvido para a implantação do *Gitflow* no contexto ágil de uma organização desenvolvendo sistemas de ML



API Protocolo [redacted]

A API Protocolo [redacted] tem como função gerenciar os dados relacionados aos protocolos, inserindo-os e alterando-os

Features:

- Inserir Protocolo
- Alterar Protocolo

Versões:

0.1: Protótipo

- Construção de uma API para realizar o controle do protocolo através de chamadas via HTTP (InserirProtocolo (), AlterarProtocolo(), BuscarProtocolo(), BuscarProtocoloById(), DeletarProtocolo())
- Conexão com o banco de dados Oracle através do EntityFramework.

Configurações:

Para executar o projeto é necessário ter o NET Core 3.1 instalado em sua máquina Windows, Linux ou Mac.É recomendado o uso do Visual Studio ou Visual Code para a manutenção do projeto.

Instruções para Build:

Instruções de Deploy:

Links:

- Baixar o Net Core 3.1 (https://dotnet.microsoft.com/download/dotnet-core/3.1)
- Baixar o Visual Code (https://code.visualstudio.com/download)
- Baixar o Visual Studio (https://visualstudio.microsoft.com/pt-br/downloads/)

Figura 3: Exemplo de artefato leia-me que será utilizado para manter informações.

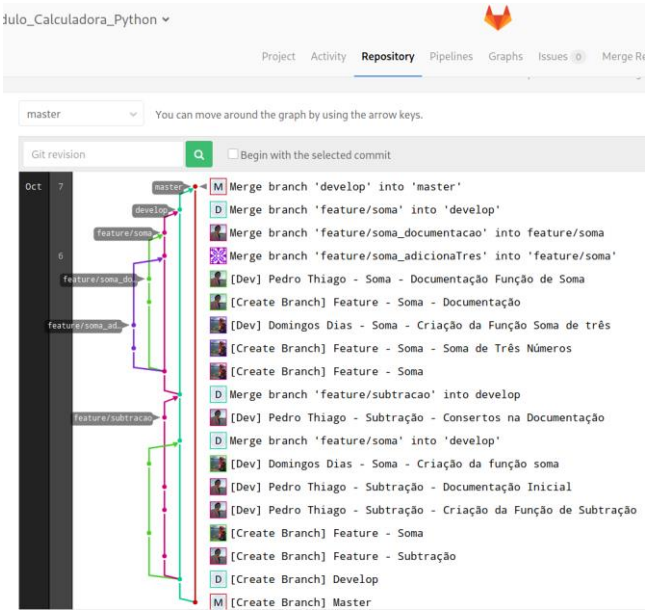


Figura 4: Diagrama contendo histórico das atividades da equipe.

que, fazendo parte da cultura organizacional, pode trazer benefícios a longo prazo para a manutenção dos projetos.

"*A curva de aprendizado é um pouco demorada bem no início. Mas depois de entender o objetivo de cada procedimento e praticar um pouco, não há muitas dificuldades.*" - Desenvolvedor 1

"*A facilidade talvez seja um dos pontos mais fracos da aplicação do Gitflow. Não que seja da natureza complicada, mas com toda certeza (pelo menos na minha experiência) é fácil cometer erros, seja em mensagens de commits erradas ou alguma operação utilizando um usuário diferente. O ponto otimista seria que com o uso mais contínuo esses erros sejam minimizados.*" - Desenvolvedor 2

"*Uma das minhas preocupações se encontra na dificuldade de adotar o Gitflow. Desenvolvedores novatos irão cometer vários erros e precisaremos ter uma equipe acompanhando o desenvolvimento. No entanto, acredito que a longo prazo, caso o Gitflow seja adotado na organização, poderemos melhorar o padrão de desenvolvimento e evitar os problemas atuais com relação a controle de versões.*" - Gerente

Com relação às possíveis dificuldades e sugestões de melhoria da abordagem proposta, os membros da equipe indicaram inicialmente a adaptação ao processo, visto que devido ao trabalho necessário para aprender as tecnologias e regras, pode ser dificultada a sua utilização ou desencorajar desenvolvedores com pouca experiência. Adicionalmente, foi sugerida a automação de alguns processos, além de flexibilizar algumas atividades para facilitar o seu uso. Alguns dos comentários relatando estes pontos são descritos a seguir:

"*Acredito que tenha uma certa resistência da equipe em seguir o GITflow bem no início. Isso aconteceria principalmente quando a equipe não era adepta a nenhuma boa prática de controle de versão.*" - Desenvolvedor 1

"*... só se alguns dos procedimentos fossem feitos por alguma ferramenta visual que me impedisse de cometer erros, tais como má nomenclatura, merges indevidos do ponto de vista do protocolo.*" - Desenvolvedor 1

"*Deixar a aplicação do Gitflow apenas um pouco mais dinâmica, afrouxar um pouco mais a padronização de mensagens de commits e outras etapas. Por exemplo, para criar um branch novo é necessário todo um rito com um commit específico e alterações em arquivos específicas, talvez isso acabe desencorajando os integrantes da equipe.*" - Desenvolvedor 2

"*... seria interessante incorporar novas ferramentas ao processo proposto. Adicionalmente, seria necessário criar um pacote de iniciante para facilitar a configuração do ambiente e a adoção. Acredito que isto seria bom para facilitar para quem nunca utilizou um processo como este, que é mais rigoroso.*" - Gerente

Finalmente, com relação à intenção de uso do *Gitflow*, todos os participantes concordaram com o seu impacto positivo nos projetos, inclusive utilizariam a abordagem fora da organização. Alguns trechos corroborando estes pontos são apresentados a seguir.

"*Sim. Usarei em projetos com mais de duas pessoas no time de desenvolvimento. Sozinho ou em dupla, eu faria algumas modificações para praticidade.*" - Desenvolvedor 1

"*Sim! A visualização e controle nos repositórios vale a pena. Os projetos que eu tive experiência sempre tiveram problemas com versionamentos devido à desorganização nos repositórios. Fazer merge e essas operações mais "complexas"no git também costumavam trazer novos problemas devido a um erro na execução das mesmas.*" - Desenvolvedor 2

"*É imprescindível adotar esta tecnologia para facilitar o trabalho do gerente e dos futuros desenvolvedores. A manutenção do software se tornará mais fácil. É evidente que haverá um custo inicial, mas acredito que este custo será compensado a longo prazo.*" - Gerente

5 CONCLUSÕES E TRABALHOS FUTUROS

Este artigo apresentou o relato de experiência da proposta e implantação de um processo baseado no modelo *Gitflow*. Este processo foi aplicado no contexto de desenvolvimento de um sistema de aprendizagem de máquina aplicando uma metodologia Scrum. A partir da análise dos problemas da organização, foi possível definir pontos principais a abordar na implantação do *Gitflow* na organização, considerando as atividades realizadas pela organização. Como resultado, foram definidas atividades para implantar o *Gitflow* e um conjunto de artefatos e templates para acompanhar o registro de atividades ao longo do processo de desenvolvimento.

Ao questionar alguns dos membros da equipe de desenvolvimento envolvidos na implantação, foi possível constatar a aceitação do processo proposto com relação à garantia da qualidade e pontos positivos em relação à organização das atividades e informações disponíveis para os gerentes do projeto. No entanto, foi unanime a preocupação a curto prazo com relação à facilidade de uso e aprendizado das diversas regras necessárias para a adoção da abordagem. Apesar destes pontos, o processo proposto pode ser útil e as vantagens consideradas no processo podem ter um impacto positivo no projeto a longo prazo, que a equipe pretende explorar.

Finalmente, este relato de experiência apresenta mais detalhes dos que os disponibilizados em outros trabalhos. Além disso, informações sobre o processo e artefatos aplicados na prática podem ser úteis para equipes de desenvolvimento interessadas na adoção e implantação do *Gitflow* nas suas organizações. Como trabalho futuro, pretende-se acompanhar a implantação do *Gitflow* a longo prazo, incluindo etapas de correção de defeitos *hotfixes* e controle de alterações das versões de lançamento *releases*, obtendo evidências da sua eficácia e eficiência, além de relatar lições aprendidas da aplicação do *Gitflow* e o que pode ser melhorado para viabilizar sua adoção em outros cenários.

AGRADECIMENTOS

Os autores agradecem à Equatorial Energia pelo apoio financeiro concedido por meio do Programa de Pesquisa e Desenvolvimento (P&D) da Agência Nacional de Energia Elétrica (ANEEL), através do processo de número APLPED00044_PROJETOPED_0036_S01. Adicionalmente, o presente trabalho foi realizado com apoio da Fundação de Amparo à Pesquisa e ao Desenvolvimento Científico e Tecnológico do Maranhão (FAPEMA), da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e do Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

REFERÊNCIAS

[1] Steve Sawyer and Patricia J. Guinan. Software development: Processes and performance. *IBM systems journal*, 37(4):552–569, 1998.

[2] Thorsten Berger, Marsha Chechik, Timo Kehrer, and Manuel Wimmer. Software evolution in time and space: Unifying version and variability management (dagstuhl seminar 19191). In *Dagstuhl Reports*, volume 9. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[3] Hussain Saleem and SM Aqil Burney. Imposing software traceability and configuration management for change tolerance in software production. *IJCSNS-International Journal of Computer Science and Network Security (ISSN: 1738-7906)*, 19(1):145–154, 2019.

[4] Rana Majumdar, Rachna Jain, Shivam Barthwal, and Chetna Choudhary. Source code management using version control system. In *2017 6th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions)(ICRITO)*, pages 278–281. IEEE, 2017.

[5] Shunfang Liu, Wang Shen, Hongbo Cai, Jian Yan Wei, and Cao Li. A software managemente method for the lut. In *2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pages 1624–1628. IEEE, 2018.

[6] Diomidis Spinellis. Git. *IEEE software*, 29(3):100–101, 2012.

[7] Vladimir Kovalenko, Fabio Palomba, and Alberto Bacchelli. Mining file histories: Should we consider branches? In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 202–213, 2018.

[8] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful tools and techniques for collaborative software development*. "O'Reilly Media, Inc.", 2012.

[9] Vincent Driessen. A successful git branching model. *URL http://nvie. com/posts/a-successful-git-branching-model*, 2010.

[10] Mathias Olausson and Jakob Ehn. Source control management. In *Continuous Delivery with Visual Studio ALM 2015*, pages 65–85. Springer, 2015.

[11] Martin Konrad, Dylan Maxwell, and Guobao Shen. Continuous integration and continuous delivery at frib. In *11st Int. Workshop on Personal Computers and Particle Accelerator Controls (PCaPAC'16), Campinas, Brazil, October 25 28, 2016*, pages 145–147. JACOW, Geneva, Switzerland, 2017.

[12] Axel Wikström. Benefits and challenges of continuous integration and delivery: A case study. *Thesis, University of Helsinki*, 2019.

[13] Francisco Bretal Ageitos. Project management platform for the identification of phytoplankton samples. *Thesis, Universidade da Coruña*, 2020.

[14] Viktoria Stray, Bakhtawar Memon, and Lucas Paruch. A systematic literature review on agile coaching and the role of the agile coach. In *International Conference on Product-Focused Software Process Improvement*, pages 3–19. Springer, 2020.

[15] Rayene Ben Rayana, Sylvain Killian, Nicolas Trangez, and Arnaud Calmettes. Gitwaterflow: a successful branching model and tooling, for achieving continuous delivery with multiple version branches. In *Proceedings of the 4th International Workshop on Release Engineering*, pages 17–20, 2016.

[16] Shaun Phillips, Jonathan Sillito, and Rob Walker. Branching and merging: an investigation into current version control practices. In *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 9–15, 2011.

[17] Wa'el Mohsen, Mostafa Aref, and Khaled ElBahnasy. Scaled scrum framework for cooperative domain ontology evolution. In *Proceedings of the 2020 The 6th International Conference on Frontiers of Educational Technologies*, pages 135–143, 2020.

[18] Younes Nejahi, Mohammad Soroush Barhaghi, Jason Mick, Brock Jackman, Kamel Rushaidat, Yuanzhe Li, Loren Schwiebert, and Jeffrey Potoff. Gomc: Gpu optimized monte carlo for the simulation of phase equilibria and physical properties of complex fluids. *SoftwareX*, 9:20–27, 2019.

[19] Thong Nguyen. Developing geolocation chat base application with ionic framework. *PhD. Thesis, Ouli University of Applied Sciences*, 2016.

[20] Richard Simko. Automating traceability in agile software development. *LU-CS-EX 2015-29*, 2015.

[21] Leticia Montalvillo and Oscar Díaz. Tuning github for spl development: branching models & repository operations for product engineers. In *Proceedings of the 19th International Conference on Software Product Line*, pages 111–120, 2015.

[22] Francisco Tarciso Leite. Metodologia científica: métodos e técnicas de pesquisa. *Aparecida: Ideias & Letras*, 2008.

[23] Vanessa Apaolaza Ibáñez, Patrick Hartmann, and Pilar Zorrilla Calvo. Antecedents of customer loyalty in residential energy markets: Service quality, satisfaction, trust and switching costs. *The Service Industries Journal*, 26(6):633–650, 2006.

[24] LADS Gruginskie and Guilherme Luís Roehe Vaccaro. Lawsuit lead time prediction: Comparison of data mining techniques based on categorical response variable. *PloS one*, 13(6):e0198122–e0198122, 2018.

[25] Adnan Amin, Sajid Anwar, Awais Adnan, Muhammad Nawaz, Khalid Alawfi, Amir Hussain, and Kaizhu Huang. Customer churn prediction in the telecommunication sector using a rough set approach. *Neurocomputing*, 237:242–254, 2017.

[26] Abbas Keramati, Hajar Ghaneei, and Seyed Mohammad Mirmohammadi. Developing a prediction model for customer churn from electronic banking services using data mining. *Financial Innovation*, 2(1):10, 2016.

[27] Renato Marchetti and Paulo HM Prado. Avaliação da satisfação do consumidor utilizando o método de equações estruturais: um modelo aplicado ao setor elétrico brasileiro. *Revista de Administração Contemporânea*, 8(4):9–32, 2004.

[28] Claudia de O Melo, Viviane Santos, Eduardo Katayama, Hugo Corbucci, Rafael Prikladnicki, Alfredo Goldman, and Fabio Kon. The evolution of agile software development in brazil. *Journal of the Brazilian Computer Society*, 19(4):523–552, 2013.

[29] Ken Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.

[30] Viswanath Venkatesh and Hillol Bala. Technology acceptance model 3 and a research agenda on interventions. *Decision sciences*, 39(2):273–315, 2008.