Associação Propagadora Esdeva
Centro Universitário Academia – UniAcademia
Curso de Bacharelado em Sistemas de Informação
Trabalho de Conclusão de Curso – Artigo

Reutilização de Software em Ambientes Distribuídos: uma proposta de framework conceitual

Oswaldo Vargas Vilas[1]
Centro Universitário Academia, Juiz de Fora, MG
Jacimar Fernandes Tavares [2]
Centro Universitário Academia, Juiz de Fora, MG

Linha de Pesquisa: Engenharia de *Software*

RESUMO

O desenvolvimento distribuído de *software* tem se tornado, progressivamente, uma estratégia adotada em empresas de diversos setores, porém a adoção dessa estratégia necessita de instrumentos que a viabilizem, considerando as principais vantagens e os desafios dessa prática. No contexto deste estudo, considera-se que o desenvolvimento distribuído de *software* ocorre por meio de atividades de projeto executadas por membros de equipes que podem estar em locais geograficamente distintos. Em vista disso, o presente trabalho propõe produzir um estudo sobre a reutilização de *software* e sua aplicabilidade em ambientes distribuídos, acerca do reuso de *software*. Além disso, contempla fundamentos sobre a reutilização de *software* e sua aplicabilidade em trabalho colaborativo, buscando explorar suas vantagens e desvantagens para o reuso dentro desse contexto. O estudo apresenta, como marcante entrave, a carência de trabalhos acadêmicos sobre o tema. Nota-se, porém, que o trabalho possibilitou um aprimoramento acerca do referencial teórico, reconhecendo atributos críticos necessários às organizações que atuam em ambientes de desenvolvimento distribuído de *software* (DDS) na promoção da aplicabilidade do reuso de *software*, ainda que tratados como uma prática importante para o desenvolvimento da tecnologia atualmente. Nessa Perspectiva, será apresentado um projeto de *framework* para reutilização de *software* que pode ser seguido por equipes de desenvolvedores, a fim de tornar essa prática mais comum no dia a dia.
Palavras-chave: Reutilização de *Software*. Equipes distribuídas. Bibliotecas de *software*. Componentes.

ABSTRACT

Keywords: Distributed software development has progressively become a strategy adopted by companies in different sectors. Therefore, the adoption of this strategy needs instruments to make it feasible, considering the main advantages and challenges of this practice. In the context of this study, distributed software development is considered to occur through project activities performed by team members who may be in geographically different locations. The present work proposes to produce a study on software reuse and its applicability in distributed

[1] Discente do Curso de Bacharelado em Engenharia de Software do Centro Universitário Academia – UniAcademia Endereço: Rua Ribeiro Junqueira, 199, Centro, Leopoldina, Minas Gerais. Celular (32) 98703-4303 E-mail: ovargasvilas@yahoo.com.br
[2] Docente do Curso de Bacharelado em Engenharia de Software do Centro Universitário Academia. Orientador.

environments from a literature review, from an overview of software reuse. In addition, it also includes fundamentals about software reuse and its applicability in collaborative work, seeking to explore its advantages and disadvantages for reuse within this context. The study presents, as a marked obstacle, the lack of academic works on the subject. However, the work made it possible to improve the theoretical framework under study, recognizing critical attributes necessary for organizations that work in distributed software development environments to promote the applicability of software reuse, although treated as an important practice for the development of technology today. From this perspective, a framework project for software reuse that can be followed by developer teams in order to make this practice more common in everyday life will be presented.
Key-words: Software Reuse, Distributed Environment. Software Libraries. Components

1 INTRODUÇÃO

Em busca de vantagem competitiva, dos avanços tecnológicos e econômicos da última década, diversas organizações optaram por distribuir o processo de desenvolvimento de *software*, suprindo a necessidade de atualização de métodos e técnicas de desenvolvimento. Assim, percebeu-se a necessidade de constante evolução nesse setor, o que otimizou e definiu os processos de negócios, bem como ampliou a comunicação com os envolvidos (ENAMI, 2006).

Nessa lógica, para Carmel (1999), citado por Huzita *et al.* (2007, p. 31), a dispersão geográfica e temporal, em vista das diferenças culturais aos quais os lugares e as pessoas estão sujeitos, são algumas das características mais marcantes que diferenciam o desenvolvimento distribuído de *software* (DDS). Tal constatação permite observar que, pelas pessoas estarem em diferentes localidades, implicando culturas distintas e diversas expectativas em relação ao trabalho em torno de projetos distribuídos, os desafios apresentados pela distribuição da equipe envolvida nesse processo tornam-se evidentemente significativos. Assim, torna-se necessária a utilização de ambientes de desenvolvimento que ofereçam o suporte adequado para essa nova realidade.

Em razão disso, a prática de reutilização de *software* – abordada com a utilização de componentes previamente documentados, testados e aprovados, a partir do reuso de um sistema previamente elaborado, adquirido e testado no desenvolvimento de novos produtos – visa melhorar significativamente a qualidade e a produtividade da empresa. Por consequência, pode-se promover uma redução considerável no tempo de execução das tarefas dos analistas e dos desenvolvedores. Vale ressaltar que embora seja um objetivo de fácil entendimento, fatores, como o desconhecimento de técnicas de reuso, a falta de ferramentas especializadas, a ausência de infraestrutura para reuso e outros fatores humanos podem configurar

problemas adicionais, que tornam complexa a realização de um processo de desenvolvimento com base em reuso (GREENFIELD *et al*., 2004). Quando realizada de forma intencional, planejada e controlada, no entanto, proporciona o alcance de benefícios relevantes. Porém, mesmo diante da verificação dos benefícios dessa técnica, a reutilização ainda não é uma prática frequente.

Os conceitos iniciais sobre reutilização de *software* foram idealizados no final da década de 1960 (FRAKES; TERRY, 1996). Entretanto, somente a partir dos anos 1980, frente à complexidade dos sistemas e às necessidades de adequação ao mercado de tecnologia da informação (TI), houve o desenvolvimento de uma demanda por métodos mais eficientes de construção de sistemas nas empresas. A esse respeito, a Revista Engenharia de *Software*, em seu volume 39 (REUTILIZAÇÃO DE SOFTWARE, s./d.), comenta sobre quais partes de um sistema anteriormente desenvolvido poderiam ser reutilizados. Entre eles, a revista cita os módulos de um projeto, o código fonte de um produto, assim como suas especificações. Além disso, aponta para o fato de que a reutilização promove vantagens por relacionar-se ao crescimento do nível de qualidade e de desempenho do desenvolvimento de *software*.

Contudo, conforme citado anteriormente, essa ideia do reuso de *software*, tão elogiada pela Revista Engenharia de *Software* (TEIXEIRA PAULINO; WERNER, 2001), ainda é muito recente, sendo conhecida apenas em 1980, quando ocorreu, nos Estados Unidos, o primeiro projeto de pesquisa universitário, voltado para a reutilização, na Universidade da Califórnia, coordenado por Peter Freeman (TEIXEIRA PAULINO; WERNER, 2001). Atualmente, no entanto, existem duas principais conferências sobre o tema da reutilização:

i) a *International Conference on Software Reuse* (ICSR), considerada, hoje, o principal evento relacionado à área de pesquisa e de tecnologia de reutilização de *software*, cujo foco é apresentar os avanços mais recentes na área, a fim de motivar um diálogo intensivo e contínuo entre pesquisadores e profissionais. A Conferência já está em sua 20ª edição, tendo sido realizada em junho de 2022, em Montpellier, na França[3]. O tema norteador do encontro foi a reutilização de *software* para as próximas gerações; e o

[3] Todas as edições do *International Conference on Software Reuse* (ICSR) encontram-se disponíveis em: < http://www.wikicfp.com/cfp/program?id=1481#:~:text=The%20International%20Conference%20on%20Software%20Reuse%20%28ICSR%29%20is,intensive%20and%20continuous%20exchange%20among%20researchers%20and%20practitioners.>. Acesso em: 28 nov. 2022.

ii) *Symposium on Software Reusability* (SSR) é um evento anual cujo foco é a apresentação de técnicas e ferramentas as quais sejam inovadoras para a avaliação, melhoria, confiabilidade e execução, em segurança, dos produtos de *software*. O Simpósio busca ressaltar os métodos científicos que têm relevância industrial e empírica, a fim de que possam ser adaptados aos contextos cotidianos em sociedade e compartilhados para todos[4].

Em resposta às novas demandas do mercado, muitas unidades de desenvolvimento de *software* se conscientizaram sobre a importância de inovar e de oferecer serviços que fossem diferenciados e personalizados aos seus clientes, desenvolvendo, dessa maneira, sistemas inteligentes que agregassem flexibilidade, agilidade e segurança aos negócios, sobretudo em relação à circulação e ao armazenamento de informações. Em busca de modernização, essas unidades começaram a se aprimorar e, com a motivação de desenvolver sistemas mais contemporâneos e eficientes que respondessem à gestão interna das organizações, passaram a ofertar soluções exclusivas de aplicativos que permitiam redução no tempo de desenvolvimento, qualidade do produto e redução de custos. Isso proporcionou um produto de excelência que atendia aos resultados desejados pelos contratantes/clientes (MOSSE; SIRQUEIRA, 2022).

Diante do contexto atual, é notória a necessidade de adaptações dos processos de trabalho adotados por muitas empresas, com a implantação de mudanças, devido à nova realidade inerente à pandemia do novo Coronavírus (Covid-19). Tal fato implicou a tomada de decisão de muitas unidades que prestam serviços de desenvolvimento de *software,* de tal forma que houve a urgência de alterarem sua forma de trabalho, principalmente com a adoção do modelo *home office*, em decorrência da obrigatoriedade de se respeitar o isolamento social, a fim de se conter a disseminação e a contaminação do vírus Covid-19, bem como sua evolução, entre as pessoas (AGUIAR, 2021). Esse novo formato de trabalho remoto – com o qual a maioria das pessoas estava desacostumada – que, antes, era, muitas vezes, ignorado em empresas brasileiras, foi incorporado às suas rotinas, visando manter o distanciamento entre os membros da equipe, como uma atitude de prevenção ao risco de contaminação (MOSSE; SIQUEIRA, 2021) .

[4] Tais informações encontram-se disponíveis em: <https://www.wikicfp.com/>. Acesso em: 18 set. 2022.

Diante dessas considerações, mesmo que tenha havido uma diminuição do número de contaminação – em comparação ao ápice do período pandêmico em meados de 2020 e 2021 – em razão do desenvolvimento de vacinas que permitiram o controle da disseminação do vírus, destaca-se que a nova realidade adversa incentivou a dispersão de um outro tipo de concorrência no mercado (AGUIAR, 2021). O recente contexto tem estimulado as unidades que desenvolvem *software* a criarem novas formas de competição e de cooperação, tornando-se mais eficientes nos processos de desenvolvimento e dando maior atenção aos ambientes projetados para suportar equipes distribuídas (SILVEIRA; MANNAN; ALMEIDA; NAGAPPAN, 2021).

O presente trabalho está dividido da seguinte forma: a primeira seção introduz os conceitos e os objetivos; a segunda seção apresenta o referencial teórico sobre reutilização de *software* em ambientes distribuídos, quais as dificuldades de isso acontecer, bem como as vantagens de se naturalizar essa prática. Além disso, são apresentados trabalhos relacionados à pesquisa. Já a terceira seção mostra o projeto de um *framework* conceitual para reutilização de *software* que pode ser usado como base por equipes de desenvolvedores. A quarta e última seção faz uma análise sobre a reutilização de *software* em ambientes distribuídos e apresenta as considerações finais. Feito o mapeamento sobre a área de reutilização de software e suas especificidades no ambiente distribuído, o projeto de *framework* conceitual busca sintetizar os passos as serem percorridos por uma equipe de desenvolvimento de *software* que busque reutilizá-lo na produção de um novo projeto, especialmente quando as equipes estiverem geograficamente distantes. O uso deste framwork possibilitaria uma melhor organização por parte da equipe, já que este apresenta possibilidades de reuso durante as principais etapas do processo de desenvolvimento e é extensível a inserção de novas etapas de acordo com o que for necessário.

2 REFERENCIAL TEÓRICO

Neste trabalho de conclusão de curso, será apresentado o conceito de reutilização de *software* em ambientes distribuídos e sua importância no que tange à aplicabilidade e às experiências advindas da sua prática, baseadas em seus pressupostos teóricos. Para a discussão do tema, este referencial teórico foi dividido em uma estrutura a qual pretende facilitar não só a leitura, como sua compreensão, conforme segue: a seção 2.1 trata a reutilização de *software*; o desenvolvimento

distribuído é abordado na seção 2.2; a reutilização de *software* em ambientes distribuídos é apresentada na seção 2.3.

Nos últimos anos, pôde-se perceber um grande avanço em direção à globalização dos negócios (SILVEIRA; MANNAN; ALMEIDA; NAGAPPAN, 2021), em que muitas organizações começaram a experimentar o *outsourcing* e a criar instalações para desenvolvimento de *software* em locais remotos. Em *Um Ambiente de Desenvolvimento Distribuído de Software-DiSE* (HUZITA, 2017), é apresentado o DiSEN (*Distributed Software Engineering Environment*), um ambiente cuja arquitetura obedece ao estilo de camadas (dinâmica, aplicação e infraestrutura), oferecendo recursos para comunicação, persistência e cooperação para o trabalho de equipes dispersas geograficamente. Sugerem-se, nesse estudo, a utilização de linguagem Java, a construção de *frameworks* e a aplicação de padrões de projeto, a fim de permitir o reuso, facilitando a manutenção, quer seja quando novos requisitos são adicionados, ou mesmo para dar continuidade à sua construção.

Além disso, o estudo *Análise estatística da qualidade do software Drupal* (MOSSE; SIRQUEIRA, 2021) propõe uma análise estatística sobre um conjunto de métricas que servem como base para a avaliação da qualidade do código fonte do *software* Drupal ao longo de suas versões. Os autores apresentam indícios da confiabilidade de sua aplicação, verificando a viabilidade de sua utilização ou a necessidade de migração para outro CMS, com a utilização de métodos estatísticos baseados na Engenharia de *Software* experimental.

O estudo intitulado *Identificação e mitigação de riscos em projetos de desenvolvimento distribuído de software* (COELHO; FENNER; LIMA, 2015) investigou os riscos envolvidos no DDS relacionados a cinco áreas (comunicação, cultura, tecnologia, colaboração e engenharia de requisitos), propondo medidas para mitigá-los. Realizou-se um estudo de caso real que gerou resultados promissores, em relação a um melhor entendimento sobre os riscos identificados e as estratégias adotadas para aplacá-los. Coelho, Fenner e Lima (2015) afirmam que, entre os riscos associados ao DDS, o mais citado na literatura é comunicação, pois, à medida que o nível de dispersão aumenta, mais difícil se torna gerenciá-lo.

Jimenez *et al.* (2019) apresentam, por sua vez, os resultados de uma revisão da literatura relacionada aos desafios do Desenvolvimento Distribuído de *Software*, cujo objetivo é identificar as soluções e melhorias propostas até os dias atuais. Em *Challenges and improvements in distributed software development: a systematic*

review, acredita-se que a qualidade dos produtos é altamente influenciada pela dos processos que os suportam. Dessa forma, em projetos DDS, o impacto dos problemas pode ser ampliado quando um problema é descoberto, e é mais difícil se recuperar disso do que em projetos com equipes não distribuídas.

Nota-se, porém, que a despeito da grande relevância dos trabalhos citados nesta seção de Revisão de Literatura, em nenhum deles se fala sobre a necessidade de se criar uma metodologia específica que facilite promover uma reutilização de software que minimize erros, para que seja utilizado com mais eficácia entre as empresas. Em vista disso, verifica-se a necessidade de aprofundar-se nesta questão e apresentar, na próxima seção do presente Trabalho de Conclusão de Curso, um *framework* conceitual que permita amenizar tais problemas para o futuro. A proposta de *framework* conceitual que será apresentada neste trabalho pretende mostrar a viabilidade da reutilização de *software* já existente em ambientes distribuídos, para que ele possa ser utilizado pelas empresas com a intenção de auxiliar a sua produtividade.

2.1 REUTILIZAÇÃO DE *SOFTWARE*

Em breve resumo, a reutilização de *software* consiste em reaproveitar elementos de um *software*, o qual tenha sido previamente desenvolvido, em um novo posteriormente. O desenvolvimento distribuído de *software* equivale, dessa maneira, a um processo de desenvolvimento em que os membros de uma equipe não se encontram em um mesmo espaço, estando geograficamente dispersos. Com um mercado cada vez mais exigente, e, conforme citado na introdução do presente Trabalho de Conclusão de Curso, devido às necessidades urgentes de mudanças de logísticas laborais impostas pela Pandemia de Covid-19 nos últimos anos, as empresas têm revisto seus processos de construção de *software.* Elas têm visado não só ao aumento da qualidade de seus produtos, como também de sua produtividade. Isso porque um projeto de *software*, ao longo do seu ciclo de vida, tem uma grande quantidade de itens de informação, tais como documentos, códigos-fonte, dados, manuais e outros, que vão sendo produzidos, e até modificados, pelos mais diversos motivos (FILHO; SPÍNOLA; COSTA; KALINOWSKI, 2022).

Nesse sentido, a ideia de reutilizar um sistema, que já tenha sido desenvolvido anteriormente, adequa-se bem às necessidades do mercado (FILHO; SPÍNOLA; COSTA; KALINOWSKI, 2022). Ressalta-se, no entanto, que essa prática não deve

ser vista como uma falta de originalidade, mas como solução para se criarem *softwares* com maior rapidez, podendo, dessa maneira, ter uma melhora significativa na qualidade do produto a ser desenvolvido, bem como na velocidade de seu desenvolvimento.

Nessa lógica, pode-se considerar a reutilização como um processo vantajoso, visto que, como consequência, pode significar a redução do tempo dos funcionários e custos da empresa, possibilitando a entrega do sistema em menor prazo, o que diminui os gastos (REUTILIZAÇÃO DE SOFTWARE, s./d.). Além disso, ressalta-se que aumenta a confiabilidade do produto em relação aos seus clientes, que utilizarão partes de um produto o qual já tenha sido desenvolvido e testado anteriormente (PRIKLADNICHI, 2002).

No entanto, ainda se verifica pouco apoio com relação a ferramentas especializadas para auxiliar no reuso desses *softwares* (PRIKLADNICHI, 2002). Soma-se a isso a falta de conhecimento das técnicas, o que dificulta ainda mais esse processo, que é complexo e não pode ser alcançado sem planejamento e controle, porque exige maior empenho do desenvolvedor, uma vez que envolve uma série de técnicas a serem utilizadas, que são empregadas desde a fase de modelagem de um projeto até sua implementação (JIMENEZ; PIATTINI; VIZCAINO, 2009).

Logo, para que a qualidade do *software* seja garantida, é necessário que a Gerência de Configuração e Mudança de *Software* (GCS) exerça sua atuação como responsável pelo controle da evolução das configurações e das mudanças no ciclo de desenvolvimento do *software* (JIMENEZ; PIATTINI; VIZCAINO, 2009). Tal ação deve ser realizada por todos na equipe, além de estar obrigatoriamente presente durante todo esse ciclo. A GCS (BASSO, 2018) é uma área específica da Engenharia de *Software* que controla e acompanha as mudanças estabelecidas em ferramentas, registra a evolução de projetos e estabelece a integridade de um sistema. Em outras palavras, com o intuito de minimizar possíveis problemas durante o desenvolvimento de um *software*, age especificamente no controle de mudança, no controle de versão e na integração contínua do sistema[5]. A GCS, portanto, objetiva gerenciar e controlar

[5] Informações disponíveis em: <https://ebasso.net/wp/2018/03/07/engenharia-de-software-gerencia-de-configuracao-de-software/#:~:text=A%20Ger%C3%AAncia%20de%20Configura%C3%A7%C3%A3o%20de%20Software%20%28GCS%29%20%C3%A9,durante%20o%20desenvolvimento%20e%20controlando%20sistematicamente%20essas%20modifica%C3%A7%C3%B5es.>. Acesso em: 18 set. 2022.

a evolução de um *software*, através do controle formal de versão e configuração de artefatos.

Por isso, ela se revela como uma forma sistemática e organizada de controlar as modificações de um *software*, mantendo-os consistentes, íntegros e adaptáveis ao longo do processo de seu desenvolvimento (FILHO; SPÍNOLA; COSTA; KALINOWSKI, 2022). A quantidade de itens que podem estar sob controle da GCS e os níveis de controle que são exigidos sobre tais itens podem sofrer variações entre as empresas ou, até mesmo, entre os projetos. Assim, o uso de ferramentas adequadas que garantam a qualidade desse processo é fundamental para que uma empresa obtenha êxito e reconhecimento.

Por envolver toda a equipe e por estar fortemente ligada aos artefatos, essa modalidade pode ser responsável por uma maior compreensão do funcionamento de artefatos pela equipe (PRIKLADNICHI, 2002)., sendo isso algo desejável para que se possa aliar a reutilização de *software* com o desenvolvimento distribuído. Contudo, alguns dos problemas potenciais a serem considerados – que tornam complexa a implementação de um processo de desenvolvimento, baseado em reuso – são: o custo de manutenção de uma biblioteca de componentes; o trabalho de encontrar e documentar os artefatos a serem reutilizados; a dificuldade de promover o acesso à infraestrutura para reutilização; e, até mesmo, mal-entendidos, que podem interferir negativamente na comunicação e no desenvolvimento (PRIKLADNICHI, 2002).

Em suma, é interessante dedicar atenção a essa estratégia, além de promover mais estudos e análises sobre todas as questões que envolvam sua execução. Para isso, existem diversos tipos de reutilização no desenvolvimento de *software*, os quais devam ser pesquisados com atenção, como: o uso de bibliotecas; o uso de padrões de projeto; o uso de *frameworks*; a reutilização de componentes, conjuntos de requisitos, testes, microprocessos, especificações, arquitetura, entre outros. Sendo assim, é de sua importância e definir em quais momentos o reuso será realizado, a fim de inseri-lo no processo de desenvolvimento.

2.2 DESENVOLVIMENTO DISTRIBUÍDO

De acordo com Steinmacher *et al.* (2013), citados por Coelho, Fenner e Lima (2015, p.559), o DDS – isto é, *Data Distribution Service* – “é baseado em times geograficamente dispersos trabalhando colaborativamente em um projeto de

software". Nele, os desenvolvedores interagem, de forma distribuída, durante o ciclo de vida do *software*, criando uma rede de times distribuídos.

Como a equipe está distanciada geograficamente, torna-se necessário um bom gerenciamento para atender às dificuldades de controle do projeto (COELHO; FENNER; LIMA, 2015). Isto posto, é sendo importante que este gerenciamento seja efetuado por uma pessoa com que tenha liderança e capacidade para trabalhar com essas diferenças de desempenho, as quais são, muitas vezes, resultantes de elementos subjetivos como cultura, pensamentos e personalidades de cada integrante da equipe (PRIKLADNICHI, 2002). Além disso, esse líder precisa proporcionar uma boa relação entre os membros da equipe, implicando na facilitação da comunicação e da interlocução em todos os aspectos, considerando, até mesmo, que, às vezes, essas equipes podem estar situadas em países diferentes (HERBSLEB; MOITRA, 2001).

Analogamente, de acordo com Herbsleb e Moitra (2001), a separação física entre os membros do projeto pode se refletir em diferentes fatores, como:

i) a decisão de realizar ou não um projeto de forma distribuída;
ii) de que maneira as tarefas serão distribuídas, tendo como base os recursos disponíveis, como infraestrutura e *expertise* em tecnologias;
iii) questões relacionadas aos valores e aos princípios culturais entre os membros da equipe;
iv) questões relacionadas ao gerenciamento de processos, como a dificuldade de sincronismo e a necessidade de marcos comuns ao projeto;
v) por fim, questões relativas à gestão do conhecimento, como o armazenamento e o compartilhamento de informações.

Outros fatores que precisam ser averiguados e tratados em relação a desenvolvimento distribuído dizem respeito à falta de conhecimento técnico da equipe acerca da linguagem de programação que está sendo usada, ou mesmo da falta de conhecimento sobre o processo de desenvolvimento e sobre os artefatos a serem reutilizados. Além disso, pode haver a incompreensão da tarefa a ser desenvolvida, e a própria baixa qualidade do *software* desenvolvido a partir do qual o reuso está sendo efetuado. Em outro aspecto, segundo Herbsleb e Mockus (2003), quando a distância entre os membros da equipe atinge 30 metros ou mais, a frequência de comunicação entre eles cai para um nível equivalente a membros, que estariam a quilômetros de

distância. Isso ficaria ainda mais evidenciado quando não houver uma boa capacitação da equipe para desenvolver o trabalho dentro deste contexto. Isso aumentaria substancialmente o risco de baixa efetividade do projeto.

Nessa perspectiva, entre os documentos pesquisados, destaca-se a abordagem apresentada por Farias Junior *et al.* (2012) que ressalta a importância da comunicação em projetos de DDS como sendo um dos principais obstáculos ao trabalho distribuído. A mesma possui um papel fundamental no desenvolvimento de *software*, permitindo resolução de problemas, entendimento de requisitos e compreensão das tarefas a serem desenvolvidas.

Por isso, Persson *et al.* (2009), citados por Coelho, Fenner e Lima (2015, p.600) enfatizam que, com a expansão da disponibilidade de infraestrutura e dos dispositivos de comunicação, o desenvolvimento distribuído de *software* (DDS) tornou-se mais praticável. Entretanto, apontam que é comum encontrar ainda barreiras, ligadas a falhas de comunicação, diferenças culturais, conforme citado, e até à tecnologia, oferecendo, por sua vez, riscos para o trabalho colaborativo.

Entre os riscos que envolvem o bom andamento do desenvolvimento distribuído, o mais comum – e mais citado na literatura – é o da comunicação, porque essa traz maior complexidade para o gerenciamento da equipe à medida que o nível de dispersão aumenta. Assim, é preciso considerar todos os obstáculos possíveis no exercício do trabalho distribuído, identificando uma maneira de eliminá-los ou de tratá-los, em tempo hábil, a partir da implementação de medidas que possam amenizar e/ou sobrepor-se a essas adversidades (PRIKLADNICHI, 2002).

2.3 REUTILIZAÇÃO DE *SOFTWARE* EM AMBIENTES DISTRIBUÍDOS

Já se sabe que a reutilização de *software* configura uma prática interessante por apresentar benefícios às empresas, aos funcionários e aos seus clientes, porém é relevante apresentar no presente trabalho os possíveis percalços que essa prática pode trazer. Inicialmente, todo código que já tenha sido criado serviu para um propósito específico, de forma que sua codificação inicial, ao ser reutilizada para outros fins, pode não atender completamente aos interesses desejados.

Urge, também, comentar sobre o número de reutilizações de *software*: à medida que seu reuso cresce, aumenta a dificuldade em conciliar as mudanças entre os distintos requisitos e as nuances de cada contexto. Em outras palavras, se, inicialmente, um artefato fora programado para uma única função, mais tarde, lhe são

programadas variadas novas funcionalidades que consubstanciam um conglomerado de mudanças. Por consequência, o código se torna cada vez mais complexo e suscetível, mais facilmente, a erros.

Deve-se atentar, portanto, para os claros perigos de reutilização (PRIKLADNICHI, 2002): o risco de se reutilizar a rotina errada cresce com o tempo, visto que mais longe se encontra da razão para a qual ela fora criada. Tal fato se assemelharia à brincadeira do telefone sem fio, cuja frase inicial vai sendo passada de pessoa a pessoa, mas, muitas vezes, ao chegar ao final, percebe-se que a mensagem do remetente é endereçada totalmente de forma desvirtuada ao seu destinatário. A despeito da analogia, observando-se o contexto do desenvolvimento distribuído, o programador nem sempre conseguirá desempenhar uma função que inicialmente pensara ao reutilizar determinado *software* para um outro contexto, para o qual esse não fora produzido, pelo simples fato de não ter sido ele quem primeiramente desenvolvera o código.

Tendo em vista esses problemas (PRIKLADNICHI, 2002), é essencial que se elabore uma metodologia para o reuso – ainda que em pequena escala. Por isso, é de grande importância determinar em quais momentos essa prática de reutilização pode, de fato, apresentar uma vantagem nos processos de engenharia. Nessa lógica, com o planejamento, a deliberação e a sistematização de um programa de reutilização, é possível medir o progresso e identificar as soluções mais eficazes de uma empresa (COELHO; FENNER; LIMA, 2015).

PPara que haja lucro significativo com um processo de reuso, de acordo com Frakes e Isoda (1994), devem ser criadas pesquisas que determinem estratégias mais efetivas para que tal processo se torne mais sistemático. Por esse motivo, essa prática é, no mínimo, desafiadora. No momento atual, são poucos os estudos na área, e faltam metodologias e *frameworks* que sejam adaptados ao desenvolvimento por equipes distribuídas. Ademais, a documentação, muitas vezes precária, e as dificuldades de comunicação entre as equipes distribuídas dificultam ainda mais esse processo. Logo, a utilização de um artefato, eficazmente, por parte de um desenvolvedor, implica sua compreensão em sua totalidade. Caso contrário, poderá reutilizar o artefato incorreto, ou implementá-lo de maneira errada. Em vista disso, será apresentada, na próxima seção, uma proposta de *framework* conceitual que poderá auxiliar equipes de desenvolvimento de *software* a reutilizarem *software* em ambientes distribuídos.

3 DESENVOLVIMENTO DO TRABALHO: PROPOSTA DE *FRAMEWORK CONCEITUAL*

Após todo mapeamento realizado sobre a área de reutilização de *software* e suas nuances no desenvolvimento distribuído, bem como a identificação dos desafios das áreas, parte-se para a definição de um *framework* conceitual que visa sintetizar quais passos e comportamentos uma equipe de desenvolvimento de *software*, que deseje atuar em um novo projeto, deva seguir para se trabalhar a reutilização de *software* considerando o escopo de equipes geograficamente distantes. A proposta do *framework* conceitual surge para autilizar a reutilização de *software* em um ambiente distribuído de desenvolvimento, tornando-a mais prática e organizada.. A maior vantagem na sua utilização está em uma melhor organização das etapas e procedimentos a serem seguidos, na reutilização de software, bem como a possibilidade de serem adicionadas novas etapas para que este se adeque ao processo de desenvolvimento desejado. Considerando que um ciclo de vida de *software* caracteriza-se pela aquisição, passando pelo desenvolvimento, manutenção e fornecimento de *software* (PRIKLADNICKI, 2002, p. 28), os quais são considerados fundamentais e indispensáveis, é possível pensar na criação de um *framework* conceitual para auxiliar equipes de desenvolvedores a produzir um *software* por meio de reutilização de outros anteriores, para que ações sejam realizadas.

Nesse contexto, o *framework* conceitual se insere nos processos de desenvolvimento de *software* para criar uma arquitetura que esteja atenta à análise de riscos, para reduzi-los, transferi-los ou até preveni-los. A intenção, com essa proposta de *framework* conceitual, é que ele formate possibilidades de reutilização dentro do processo de desenvolvimento de *software.* Logo, objetiva-se que o *framework* conceitual ora apresentado seja suficiente para que as equipes que o utilizarem não necessitem buscar outras estratégias de compreensão ou de realização de reaproveitamento, na reutilização de *software* em projetos com equipes geograficamente distribuídas, e caso seja necessário, seja alterado para melhor se integrar com outros processos de desenvolvimento. Sua principal diferença em relação a outros modelos é o foco na reutilização e a possibilidade de ser extensível para se adequar a diferentes processos.

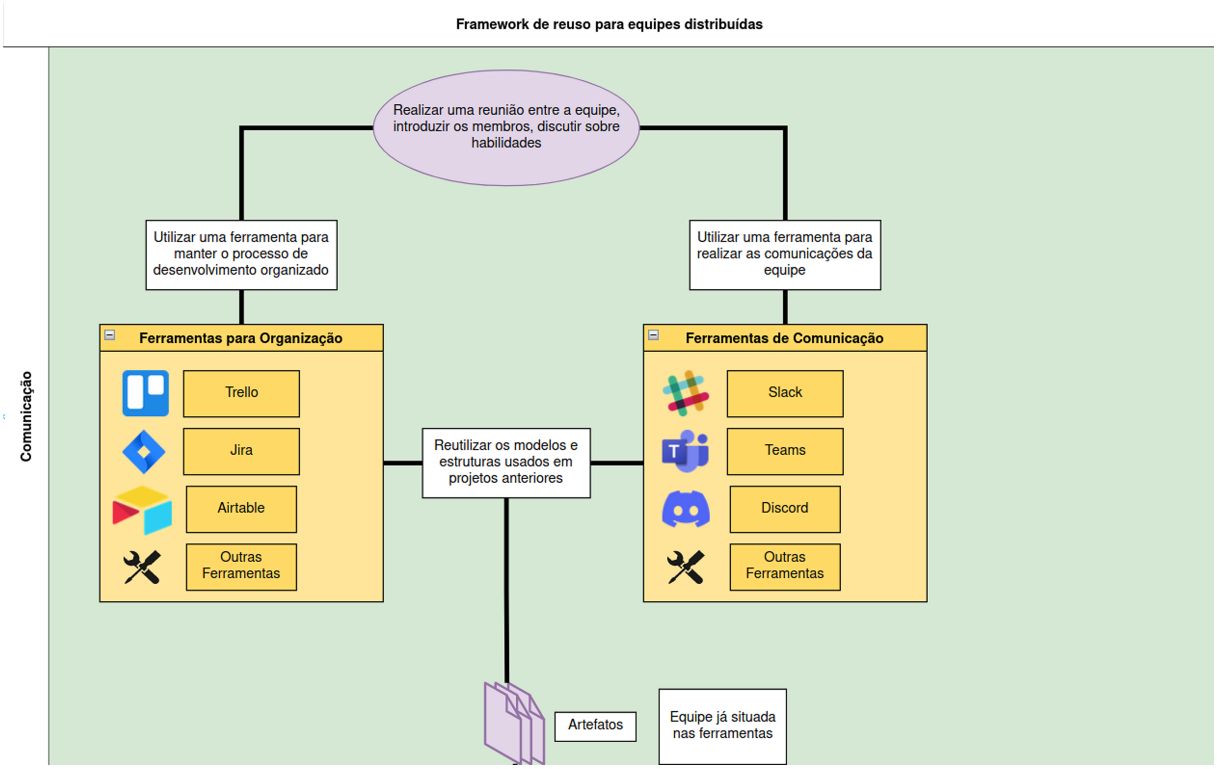
A esse respeito, o *framework* conceitual que se apresenta a seguir divide-se em seis etapas essenciais que buscam localizar a efetivação de sua prática não só

em bases locais, como em distribuídas. Essas etapas encontram-se, então, divididas em: i. Comunicação ii. Levantamento e análise de requisitos; iii. Design; iv. Desenvolvimento; v. Teste; vi. Implantação; viii. Manutenção. Além disso, para facilitar o entendimento e a visualização do *framework* conceitual, optou-se por obedecer à seguinte estrutura: os balões redondos de cor roxa são destinados às equipes que irão desenvolver o *software* e os quadros em amarelo são as opções a serem escolhidas pelas equipes; por fim, os hexágonos em vermelho são considerados casos alternativos. As imagens das etapas bem como o próprio framework conceitual podem ser encontrados acessando o seguinte link: <https://github.com/OVVilas/Framework-conceitual-para-a-reutiliza-o-de-software-em-equipes-distribu-das/tree/main>

3.1 COMUNICAÇÃO

Uma boa comunicação é crucial para que o framework possa ser utilizado com sucesso. Para isto, o diagrama que segue na Figura 1 apresenta a etapa de Comunicação:

Figura 1 – Comunicação



Fonte: Imagem produzida pelo próprio autor.

Conforme o diagrama apresentado na imagem acima, uma reunião inicial deve ser feita entre os membros da equipe, onde eles possam discutir sobre o trabalho a ser feito, suas prórprias experiências, seus planos e ideias. Para que estas sejam possíveis, deve ser usado uma ferramenta de comunicação, como Slack[6], Teams[7], Discord[8] ou alguma outra ferramenta que seja de interesse da empresa. Deve-se também, nessa etapa, determinar uma ferramenta de organização a ser utilizada, como exemplo temos Trello[9], Jira[10] e Airtable[11]. Tais ferramenta permitem uma melhor organização dos processos, e é possível que se reutilizem os modelos e estruturas usadas em projetos anteriores. No fim desta etapa, a equipe já deve estar familiarizada com as ferramentas de comunicação e de organização, sua estrutura e também com seus colegas de equipe.

3.2 LEVANTAMENTO E ANÁLISE DE REQUISITOS

As etapas do presente *framework* conceitual foram escolhidas a partir das definições de Prikladnicki (2002) sobre *framework* conceitual em desenvolvimento distribuído de *software* que considera a utilização de uma variabilidade de mecanismos, a partir do seu potencial de utilização. Para isso, é importante que, no modelo de *framework* conceitual que as equipes seguiriam para desenvolver o *software*, inicialmente, sejam feitos um levantamento e uma análise dos requisitos, conforme diagrama que se segue na Figura 2:

Figura 2 – Levantamento e análise de requisitos

[6]Disponível em <https://slack.com/> Acesso em 07/12/2022

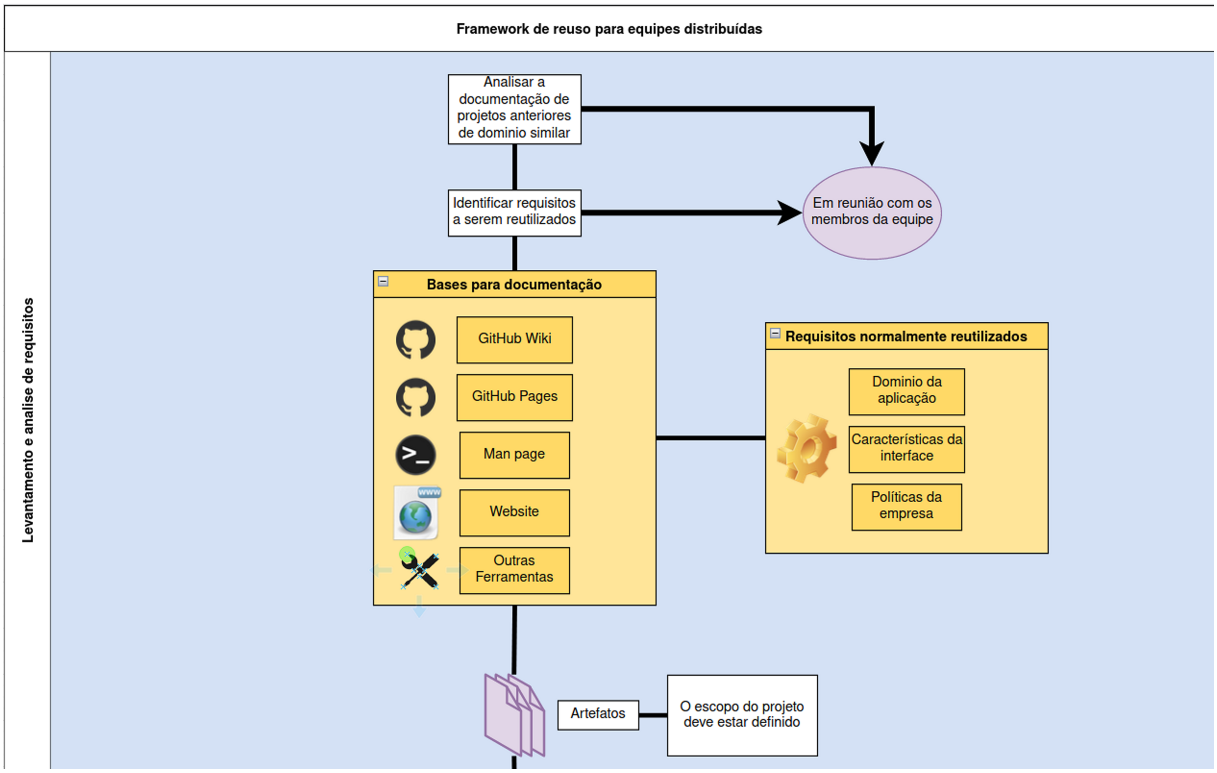
[7]Disponível em <https://www.microsoft.com/en-us/microsoft-teams/> Acesso em 07/12/2022

[8]Disponível em <https://discord.com/> Acesso em 07/12/2022

[9]Disponível em <https://trello.com/> Acesso em 07/12/2022

[10]Disponível em <https://www.atlassian.com/software/jira> Acesso em 07/12/2022

[11]Disponível em <https://www.airtable.com/> Acesso em 07/12/2022



Fonte: Imagem produzida pelo próprio autor.

Conforme o diagrama que se apresenta na imagem acima, consideram-se necessários os principais requisitos a serem normalmente reutilizados no desenvolvimento de *software* pelas equipes – tendo em vista, novamente, bases locais e distribuídas. Tais requisitos se dividem em domínio da aplicação, características da interface, políticas da empresa e linguagem de programação. Essa primeira etapa é fundamental especialmente para essas equipes tenham ciência dos requisitos sejam claramente identificados e dos objetivos a serem buscados.

Utiliza-se, normalmente, como base para a documentação o GitHub Wiki[12], GitHub Pages[13], Man Page[14], Websites, ou outras ferramentas que possam auxiliar na reutilização dos requisitos. É interessante, para isso, que o escopo do projeto esteja bem definido pela equipe, desde o início, uma vez que, ao se definir claramente qual será a utilidade final do *software* e qual ou quais problemas ele pretende resolver,

[12]Disponível em <https://docs.github.com/en/communities/documenting-your-project-with-wikis/about-wikis> Acesso em 29/11/2022

[13]Disponível em <https://pages.github.com/> Acesso em 29/11/2022

[14]Disponível em <https://www.kernel.org/doc/man-pages/> Acesso em 29/11/2022

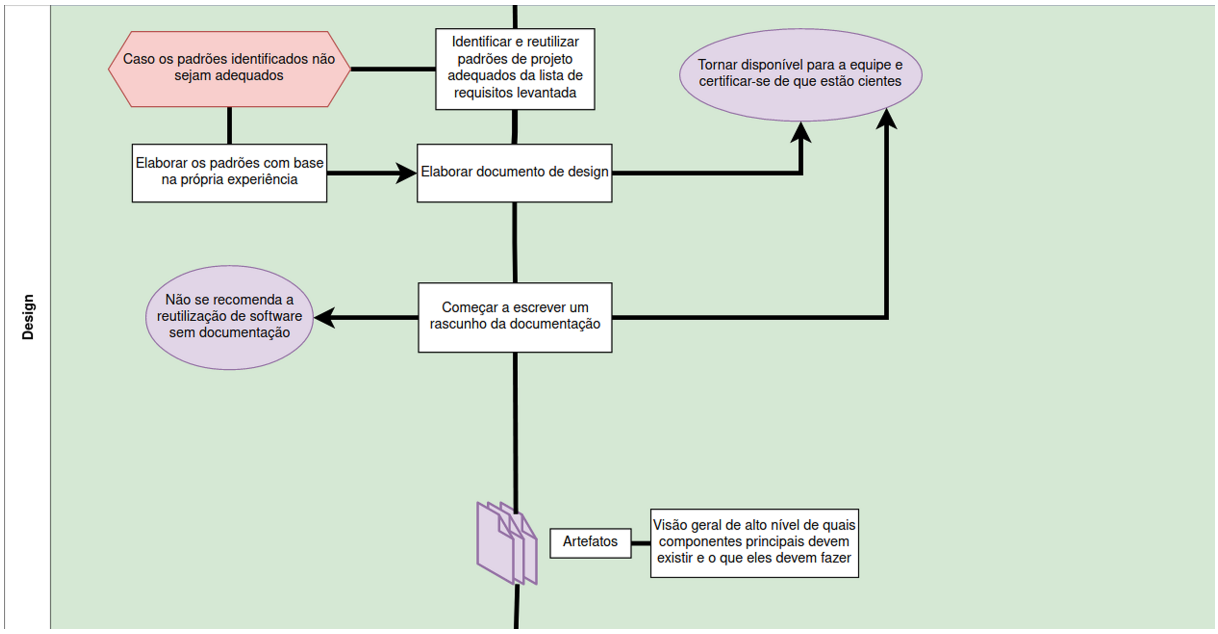
torna-se mais fácil escolher a base para a documentação, bem como quais requisitos se pretende reaproveitar ou melhorar.

Por isso, optou-se por pensar em quatro requisitos base que sejam completos e consistentes para o ciclo de desenvolvimento de *software* a partir deste modelo de *framework* conceitual. Nessa perspectiva, é importante que as equipes configurem as políticas que regem a empresa nas quais eles atuam, definam as características essenciais da interface escolhida para a utilização, em que domínio essa reutilização será aplicada e qual será a linguagem de programação escolhida, para que então sejam reutilizados. Isso tudo poderá ser realizado a partir de reunião com a própria equipe após a análise de documentos e projetos anteriormente realizados com domínios e características similares.

3.3 DESIGN

Após levantamento e análise de requisitos, caminha-se para a segunda etapa do processo que é o Design do projeto que só poderá ser pensado após definidas as modalidades previamente discutidas. Nesta etapa, serão identificados e reutilizados os padrões de projeto adequados conforme a Figura 3, para, após, elaborar o documento de design. É importante, nessa etapa que esse documento de design esteja disponível para a equipe, a fim de que todos estejam cientes do design que devem elaborar. É válido ressaltar, conforme se encontra no hexágono em vermelho, abaixo que, caso os padrões de projeto identificados para serem reutilizados não sejam adequados, convém que a equipe elabore seus próprios com base em sua própria experiência.

Figura 3 – Design



Fonte: Imagem produzida pelo próprio autor.

Para isso, é importante que seja produzido, em seguida, um documento de design o qual esteja baseado em documentos de projetos anteriormente identificados, com a ciência prévia de toda a equipe. Nesse momento, é importante que o desenvolvedor tenha em vista qual é o produto o qual pretende desenvolver, bem como o propósito que busca atingir, para que seja mais bem delineada a arquitetura do *software* em meio à qual serão firmados os requisitos previamente analisados na etapa anterior.

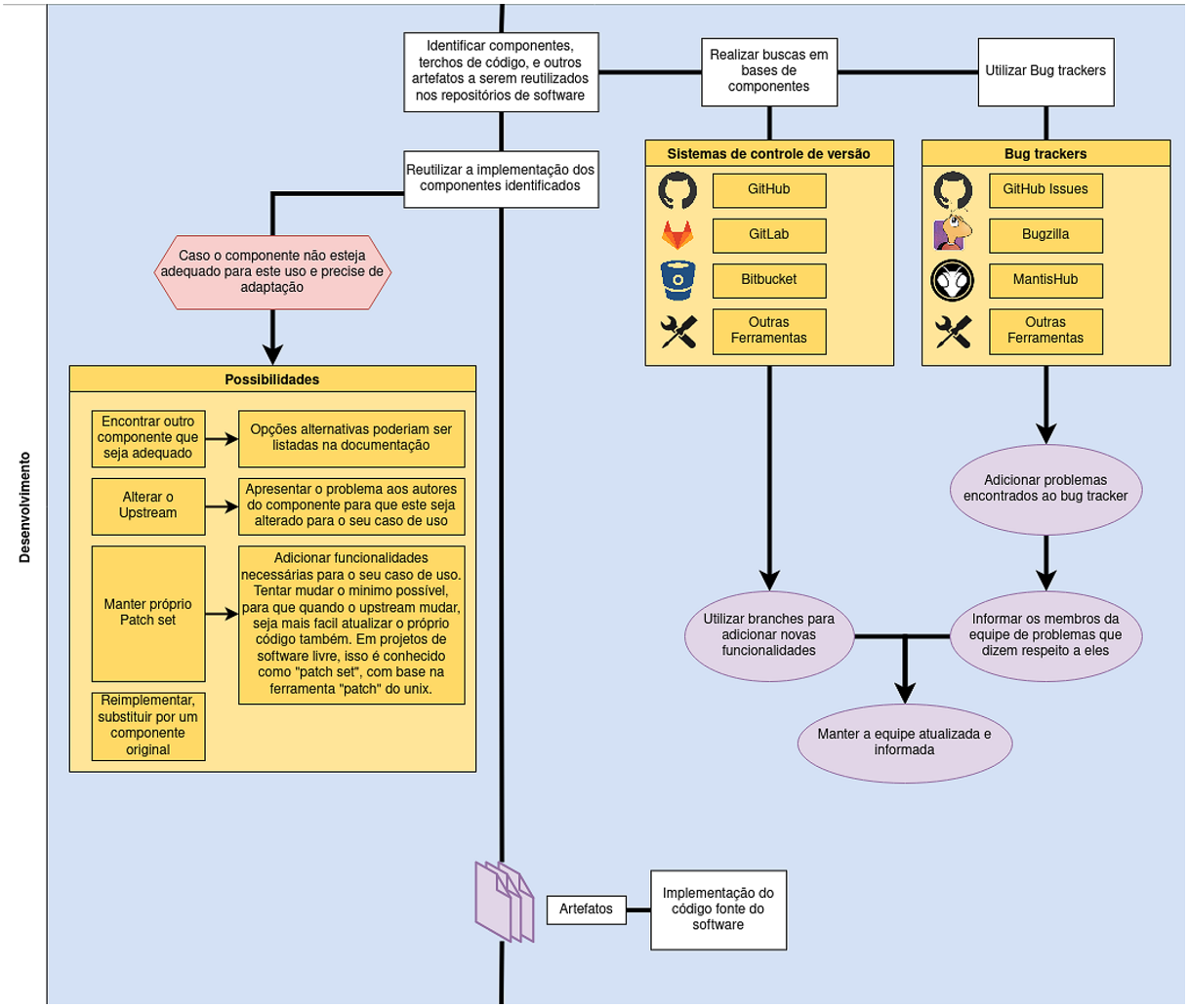
A documentação produzida nessa etapa, a qual será discutida pela equipe, deve ser de fácil leitura, compreensão e navegação. Além disso, deve ser fácil e rápida a busca e o compartilhamento de trechos da documentação com colegas de equipe. É importante que essa documentação seja produzida, porque um software sem uma documentação adequada dificilmente será entendido por seus desenvolvedores e mantenedores. Além disso, todo *software*, para que possa ser mais tarde reutilizado, necessita de uma documentação robusta para fundamentação.

3.4 DESENVOLVIMENTO

Nessa etapa, a equipe de desenvolvedores deve, por meio do sistema de controle de versão utilizado, acessar os repositórios contendo os componentes, os trechos de código, bem como outros artefatos que serão reutilizados. Uma vez

identificados esses componentes, haverá a sua reutilização a partir de um sistema de controle de versão como o GitHub[15], GitLab[16] ou o Bitbucket[17].

Figura 4 – Etapa de desenvolvimento



Fonte: Imagem produzida pelo próprio autor.

[15]Disponível em <https://github.com/> Acesso em 29/11/2022

[16]Disponível em <https://about.gitlab.com/> Acesso em 29/11/2022

[17]Disponível em <https://bitbucket.org/> Acesso em 29/11/2022

É necessário também que a equipe faça o uso de *Bug trackers*, como o GitHub Issues[18], o BugZilla[19] ou o MantisHub[20], com a intenção de identificar as atuais falhas e informar sobre quaisquer problemas que possivelmente ocorram aos que dizem respeito. Assim, todos os problemas encontrados devem ser adicionados ao *bug tracker* pela equipe, que informará a todos os membros sobre sua existência. Além disso, é importante que a equipe esteja sempre informada e atualizada a respeito de quaisquer mudanças e problemas que possam ocorrer.

Ressalta-se, no entanto, que se os componentes identificados para a implementação não estiverem adequados para o uso, deve-se decidir qual opção é mais viável:

- Encontrar outro componente que seja adequado, possivelmente listado na documentação,
- Apresentar o problema aos autores do componente para que estes alterem o componente de acordo com o seu caso de uso
- Adicionar funcionalidades necessárias para o seu caso de uso. Tentar mudar o mínimo possível, para que quando o *upstream* mudar, seja mais fácil atualizar o próprio código também. Em projetos de software livre, isso é conhecido como "patch set", com base na ferramenta "patch" do unix.
- Criar um componente original para ser usado em seu lugar

3.5 TESTE

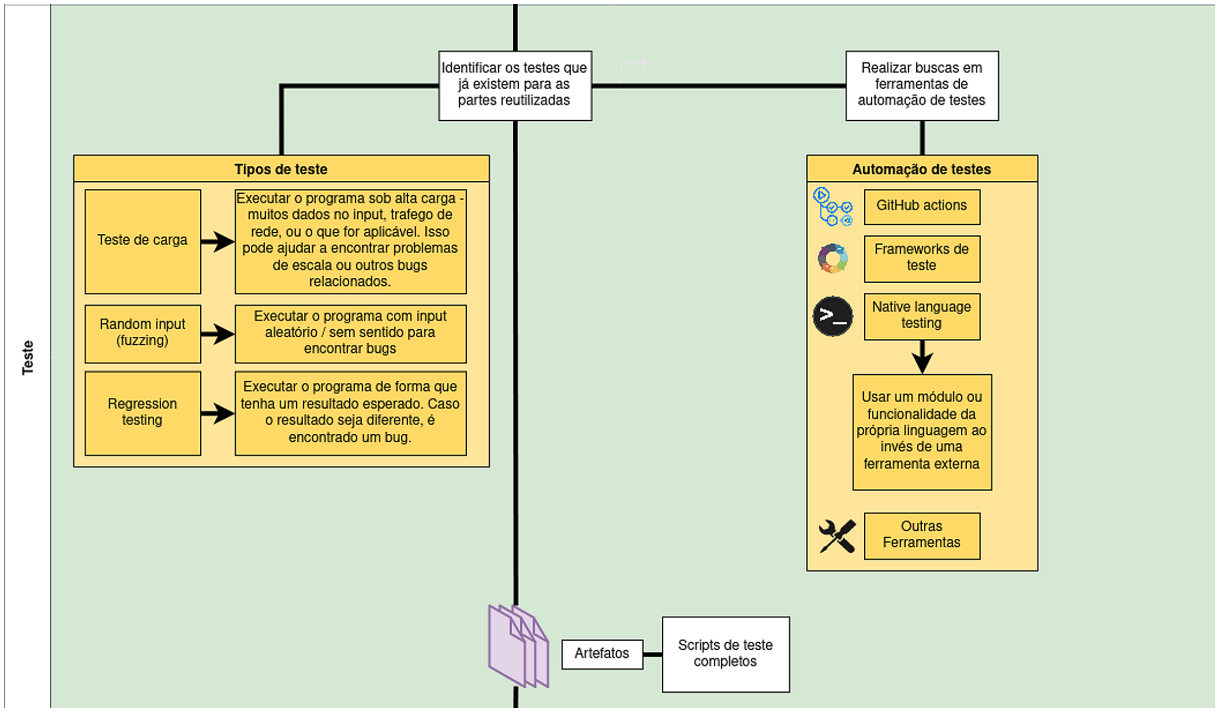
Na etapa de teste, como o próprio nome sugere, a equipe busca identificar aqueles já existentes para as partes reutilizadas, com a intenção de observar possíveis erros e/ou defeitos. Nesse momento, será avaliada a sua usabilidade. Portanto, é importante que as interfaces e funções do *software* reaproveitado sejam intuitivas, para que a equipe possa realizar o teste de carga, conforme apontado na Figura 5.

[18]Disponível em <https://docs.github.com/en/issues> Acesso em 29/11/2022

[19]Disponível em <https://www.bugzilla.org/> Acesso em 29/11/2022

[20]Disponível em <https://mantishub.com/> Acesso em 29/11/2022

Figura 5 – Etapa de teste



Fonte: Imagem produzida pelo próprio autor.

Nesse aspecto, os testes podem ser feitos de maneira automatizada, como, por meio de *GitHub actions*[21], utilizando outros *frameworks* de teste, ou utilizando *native language testing*, por exemplo. Esta etapa é necessária para que a equipe consiga avaliar se o *software* produzido tenha algum problema de usabilidade, antes de seguir a próxima etapa que é a sua implementação. Existem diversos tipos de teste possíveis, entre eles foram destacados três: o teste de carga, já que ajuda a encontrar problemas de escala ou outros *bugs* relacionados; o *random input*, que, como o próprio nome faz alusão, executa programa com um input aleatório, a fim de encontrar *bugs*; por fim, o *regression testing*, que executa o programa com o objetivo de se encontrar um resultado esperado, logo, caso o resultado for diferente, um *bug* é encontrado.

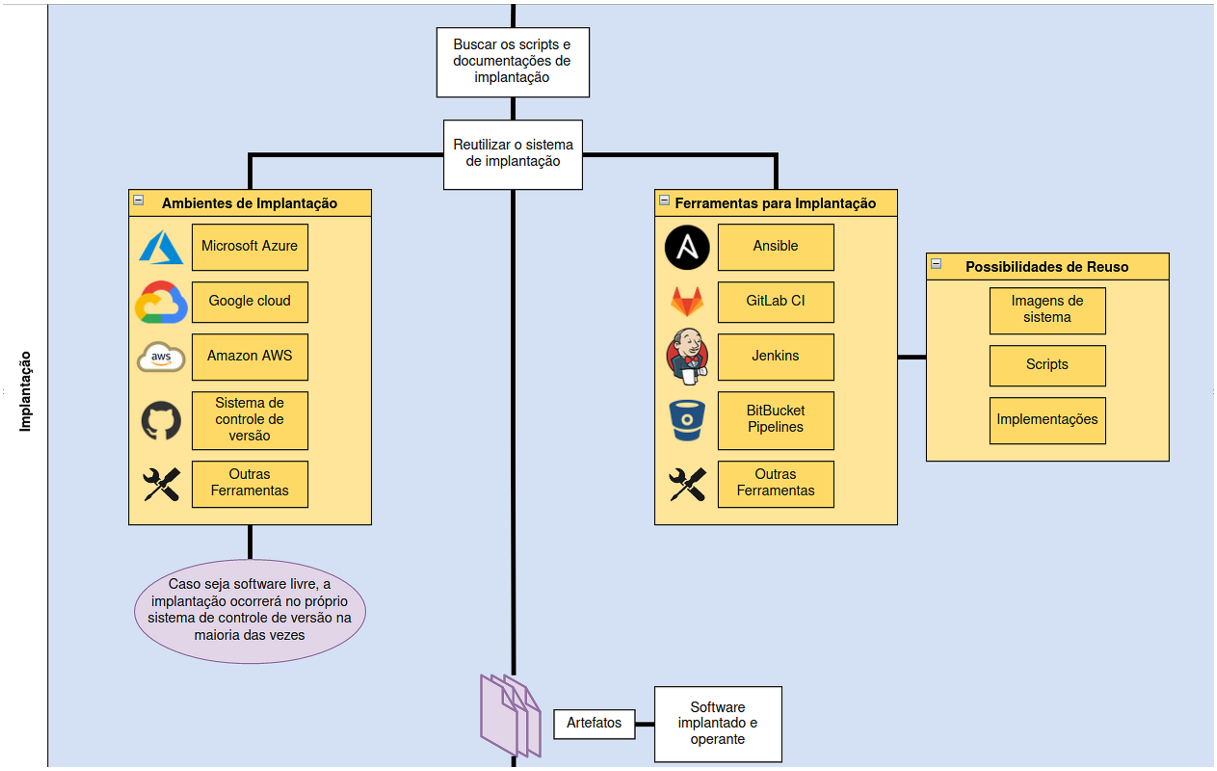
3.6 IMPLANTAÇÃO

Na etapa de implantação, a equipe finalmente realizará a implantação do código desenvolvido, podendo optar por algumas ferramentas, para este fim, como

[21]Disponível em <https://docs.github.com/en/actions> Acesso em 29/11/2022

Ansible[22], GitLab CI[23], Jenkins[24], BitBucket Pipelines[25] ou até outras ferramentas que considerar mais viáveis. É interessante que a equipe reutilize os scripts anteriores de implantação, imagens de sistema e outras implementações conforme apontado na Figura 6, a seguir, para agilizar o processo.

Figura 6 – Etapa de implantação



Fonte: Imagem produzida pelo próprio autor.

[22]Disponível em <https://www.ansible.com/> Acesso em 07/12/2022

[23]Disponível em <https://docs.gitlab.com/ee/ci/> Acesso em 07/12/2022

[24]Disponível em <https://www.jenkins.io/> Acesso em 07/12/2022

[25]Disponível em <https://bitbucket.org/product/features/pipelines> Acesso em 07/12/2022

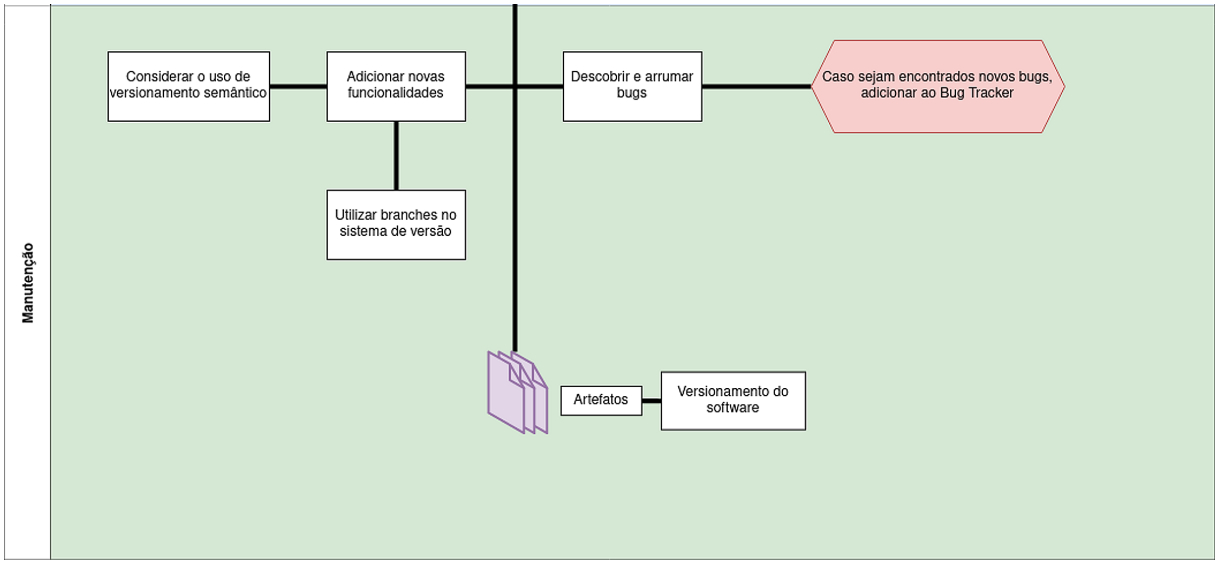
Pode-se também reutilizar um ambiente de implantação como as plataformas em cloud Microsoft Azure[26], Google Cloud[27], Amazon AWS[28], ou até próprio sistema de controle de Versão

Caso o software desenvolvido seja *software* livre, a sua implementação geralmente ocorrerá no próprio sistema de controle de versão.

3.7 MANUTENÇÃO

A manutenção é a última etapa do projeto de desenvolvimento do software que só acontecerá após esse ter sido desenvolvido, testado, ~~e~~ aprovado e implantado pela equipe. Nesta etapa, conforme a Figura 7, abaixo, caso sejam encontrados erros – ou *bugs* - estes poderão ser adicionados ao *bug tracker* e poderão ser consertados.

Figura 7 – Etapa de manutenção



Fonte: Imagem produzida pelo próprio autor.

Ademais, com o tempo, é possível que novas funcionalidades possam ser adicionadas ao *software* reaproveitado.

4 CONSIDERAÇÕES FINAIS

[26]Disponível em <https://azure.microsoft.com/> Acesso em 29/11/2022

[27]Disponível em <https://cloud.google.com/> Acesso em 29/11/2022

[28]Disponível em <https://aws.amazon.com/> Acesso em 29/11/2022

O presente Trabalho de Conclusão de Curso buscou mostrar a possibilidade da aplicabilidade da Reutilização de *Software* em Equipes Distribuídas. Para isso, recorreu-se às consultas bibliográficas as quais, embora tenham sido importantes para o desenvolvimento da proposta de *framework* conceitual apresentado neste trabalho, evidenciaram que ainda é um assunto que se apresenta de maneira escassa na área de Desenvolvimentos de *Software*. Além disso, tais referências não foram conclusivas no que concerne à experimentação prática de sua aplicabilidade, de maneira que, mesmo que se aponte a vantagem na redução de custos no processo de trabalho, uma série de riscos – que ultrapassam as questões tecnológicas – é ponderada. Muitos desses riscos associam-se, principalmente, às dificuldades de comunicação e de colaboração entre os envolvidos.

Esses fatores podem estar relacionados especialmente às questões sociais e humanas, o que certamente representam dificuldades ao gerenciamento do projeto. Logo, se, na teoria, a reutilização de dados de *software* se apresente como uma interessante ideia, na prática, sua aplicabilidade nem sempre se efetiva de maneira adequada, principalmente, quando se trata de bases distribuídas. No entanto, vale ressaltar que os resultados são diferentes para cada caso. Na verdade, a reutilização se torna inviável quando ocorrem dificuldades em se integrar uma equipe distribuída, ou quando há falta de estrutura para o reuso (isto é, quando falta documentação, ou o *software* desenvolvido não foi pensado para ser reutilizado). Nessa perspectiva, é de se considerar que qualquer forma de reuso – além de bibliotecas – se tornaria impraticável.

Assim sendo, é necessário aprofundar a aprendizagem não só sobre técnicas de compartilhamento de conhecimento, como sobretudo de gerenciamento e de avaliações de custos e desempenho, uma vez que esse procedimento tem representado uma das soluções para a crise de *software* ainda presente. Assim, os benefícios da reutilização de *software* que são caracterizados pela redução de esforços, evitarão o retrabalho, diminuindo custos em sua operacionalização e prometendo melhorar a qualidade do *software* desenvolvido e a produtividade da equipe.

REFERÊNCIAS

AGUIAR, Estela. Após começo turbulento, empresas se adaptam ao home-office e planejam mantê-lo. CNNBrasil**. São Paulo, 2021. Disponível em: <** https://www.cnnbrasil.com.br/business/apos-comeco-turbulento-empresas-se-adaptam-ao-home-office-e-planejam-mante-lo/**>. Acesso em: 28 nov. 2022.**

ASPRAY, F. M. W.; VARDI, M. Y. **Globalization and offshoring of *software***: a report of the ACM Job Migration Task Force. New York: Association for Computing Machinery, 2006.

BASSO, Enio. Engenharia de Software: gerência de configuração de software. **Ebasso.net**, 2018. Disponível em: < https://ebasso.net/wp/2018/03/07/engenharia-de-software-gerencia-de-configuracao-de-software/#:~:text=A%20Ger%C3%AAncia%20de%20Configura%C3%A7%C3%A3o%20de%20Software%20%28GCS%29%20%C3%A9,durante%20o%20desenvolvimento%20e%20controlando%20sistematicamente%20essas%20modifica%C3%A7%C3%B5es.>. Acesso: 28 nov. 2022.

COELHO, P. H. V.; FENNER, G.; LIMA, A. S. Identificação e mitigação de riscos em projetos de desenvolvimento distribuído de *software.* **Revista Eletrônica Sistemas & Gestão**, v. 10, n. 4, p. 599-607. 2015. Disponível em: <https://www.revistasg.uff.br/sg/article/view/543/368>. Acesso em: 02 jun. 2022

DAVISON, R. Offshoring information technology: sourcing and outsourcing to a global workforce. **Information Technology for Development**, v. 13, n. 1, p. 101–102, 2007.

DAY, R. One Big Lesson on Creating a Software Library. **Towards data science**. 2021. Disponível em: <https://towardsdatascience.com/one-big-lesson-on-creating-a-software-library-723c36180941>. Acesso em: 03 jun. 2022.

ENAMI, L. N. M. **Um Modelo de Gerenciamento de Projetos para um Ambiente de Desenvolvimento Distribuído de *Software***. Dissertação (Mestrado). Programa de Pós-Graduação em Ciência da Computação. UEM-DIN. Maringá/PR, 2006.

FILHO, A. B.; SPÍNOLA, R. O.; COSTA, M. N.; KALINOWSKI, M. Gerência de Configuração Definições Iniciais, Ferramentas e Processo. **Engenharia de Software Magazine** - Gerência de Configuração, v . 24, 2010. Disponível em: <http://www-di.inf.pucrio.br/~kalinowski//publications/ BollerSCK10.pdf>. Acesso em: 06 jun. 2022.

FRAKES, W., TERRY, C. *Software* Reuse: Metrics and Models. **ACM Computing Surveys**, v. 28, n. 2, p. 415-435, 1996.

GREENFIELD, J. *et al*. ***Software* Factories**: Assembling Applications with Patterns. Models, Frameworks, and Tools. New York: John Wiley & Sons, 2004.

HERSELB, J.; MOITRA, D. Global Software Development, ***Software**, IEEE*, v.18, n. 2, p.16-20. 2001.

HUZITA, E. H. M. *et al*. Um Ambiente de Desenvolvimento Distribuído de Software – DiSEN. **I Workshop de Desenvolvimento Distribuído de Software**, 2007. Disponível em: <chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://www.sesos-wdes.icmc.usp.br/pdf/wdds/2007/2007-WDDS-04.pdf>. Acesso em: 17 nov. 2022.

IMOIZE, L.; IDOWU, D.; BOLAJI, T. A Brief Overview of Software Reuse and Metrics in Software Engineering. **World Science News,** v. 122, p. 56.70, 2019. Disponível em: < chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/http://www.worldscientificnews.com/wp-content/uploads/2019/01/WSN-122-2019-56-70.pdf>. Acesso em: 17 nov. 2022.

JIMENEZ, M.; PIATTINI, M.; VIZCAINO, A. Challenges and improvements in distributed software development: a systematic review. **Advances in Software Engineering**, p. 1-14, 2009. Disponível em: <https://www.hindawi.com/ journals/ase/2009/710971/>. Acesso em: 04 jun 2022.

MOSSE, B. G..; SIRQUEIRA, T. F. M. Análise estatística da qualidade do *software* Drupal. **Caderno de Estudos em Engenharia de Software**, v. 3, n. 2, 2021. Disponível em: <http://seer.uniacademia.edu.br/index.php/engsoftware/article/view/3013/2248>. Acesso em: 17 nov. 2022.

MURTA, L. G. P; BARROS, M. de O.; WERNER, C. M. L. Token: Uma Ferramenta para o Controle de Alterações em Projetos de Software em Desenvolvimento. XIV Simpósio Brasileiro de Engenharia de Software. João Pessoa, 2000. Disponível em: <https://bibliotekanauki.pl/articles/1071402>. Acesso em: 02 jun. 2022.

_____; WERNER, C. M. **Odyssey-SCM:** uma abordagem de gerência de configuração de software para o desenvolvimento baseado em componentes. Exame de Qualificação, COPPE, UFRJ, Rio de Janeiro. 2004. Disponível em: <http://www.ic.uff.br/ ~leomurta/papers/murta2004b.pdf>. Acesso em: 02 jun. 2022

PRIKLADNICHI, Rafael. **Desenvolvimento distribuído de software e Processos de desenvolvimento de software.** Porto Alegre, 2002. Disponível em: <chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.inf.pucrs.br/volut/docs/TI2.pdf>. Acesso em: 15 nov. 2022.

_____. **MuNDDoS:** um modelo de referência para desenvolvimento distribuído de *software*. 145 f Dissertação (Mestrado). Programa de Pós-Graduação em Ciência da Computação. Porto Alegre, 2003. Disponível em: <https://tede2.pucrs.br/tede2/handle/tede/5092>. Acesso em: 04 jun. 2022

_____; DAMIAN, D.; AUDY, J. L. N. Patterns of volution in the practice of distributed software development: quantitative results from a systematic review, in Proceedings of the 12th. **Conference on Evaluation and Assessment in Software Engineering** (EASE 08), Italy, 2008. Disponível em: < https://www.researchgate.net/publication/262218707>. Acesso em: 17 nov. 2022.

REUTILIZAÇÃO DE *SOFTWARE.* **Revista Engenharia de Software Magazine 39**. s./d. Disponível em: <https://www.devmedia.com.br/reutilizacao-de-software-revista-engenharia-de-software-magazine-39/21956>. Acesso em: 03 jun. 2022

SILVEIRA, Paulo; MANNAN, Umme Ayda; ALMEIDA, Eduardo Santana de; NAGAPPAN, Nachi. A Deep Dive into the Impact of COVID-19 on Software Development. **IEEE Transactions on Software Engineering**, 2021. Disponível em: < https://www.researchgate.net/publication/352344336_A_Deep_Dive_into_the_Impact_of_COVID-19_on_Software_Development> Acesso em: 28 nov. 2022.

SOMMERVILLE, Ian. **Software Engineering.** Boston, Massachusetts: Addison-Wesley Longman, 2006. Disponível em: < https://www.academia.edu/es/58171756/Software_Engineering_9th_Edition_by_Ian_Sommerville>. Acesso em: 17 nov. 2022.

TEIXEIRA, H. V.; PAULINO, L. G.; WERNER, C. M. L. *LockED:* Uma Abordagem para o Controle de Alterações de Artefatos de *Software*. **Library.org.** s./d. Disponível em: http://www.ic.uff.br/ ~leomurta/papers/teixeira2001.pdf. Acesso em: 29 maio 2022.