Processo de Desenvolvimento em Projetos Open Source: Um Estudo Observacional

Thayssa A. Rocha[1], André M. Pinheiro[1]

[1]Instituto de Ciências Exatas e Naturais – Universidade Federal do Pará (UFPA)
CEP 66075-110 – Belém – PA – Brasil
{thayssa.rocha,andremirandap93}@gmail.com

Abstract. *It is not common to discuss the definition or improvement of processes involved in the construction of Open Source projects. However, several mechanisms and procedures are involved in the rules of contribution, especially in large communities. This paper presents an observational study carried out in three large Open Source projects, in order to evaluate if good practices in software engineering are planned in defined contribution rules.*

Resumo. *Não é comum que se discuta a definição ou melhoria dos processos envolvidos na construção de projetos Open Source. Porém, é notável a existência de uma série de mecanismos e procedimentos envolvidos nas regras de contribuição, especialmente em grandes comunidades. Este artigo apresenta um estudo observacional realizado em três grandes projetos Open Source, com a finalidade de avaliar se boas práticas da engenharia de software são previstas nas regras de contribuição definidas.*

1. Introdução

Esforços de melhoria do processo de software, como o CMMI, requerem recursos normalmente encontrados em grandes empresas, como um grupo de melhoria do processo, tempo para treinamento, consultores externos e a disposição de adicionar custos indiretos ao processo de desenvolvimento em troca de gerenciamento de risco (Robbins, 2003).

Nas últimas duas décadas, esforços de melhoria de processo tem apresentado propostas mais ágeis para desenvolvimento de software, com menos formalidades e objetivando maior entrega de resultados. A partir do manifesto ágil (Beck *et al.,* 2001) – que valoriza indivíduos e interações mais que processos e ferramentas, software funcionando mais que documentação abrangente, colaboração com o cliente mais que negociação de contratos e resposta a mudanças mais que seguir um plano – uma série de processos e práticas leves de desenvolvimento foram incluídas no cenário mundial.

Uma analogia clássica acerca do desenvolvimento de software foi proposta por Raymond (1999), quando comparando o que ele inicialmente acreditava ser o ambiente ideal para o desenvolvimento de grandes produtos de software, com a construção de uma catedral: um trabalho habilmente cuidado e desenvolvido por um pequeno grupo de sábios em esplêndido isolamento. Em contraponto, ele apresentou o estilo de desenvolvimento que o Linux utilizava, como um bazar: barulhento e agitado, com agendas diversas e milhares de contribuidores.

Apesar da tendência a se vincular a proposta informalidade dos métodos ágeis ao modelo bazar, é possível perceber que ao longo do tempo, alguma ordem foi acrescida a este ambiente "barulhento e agitado", a partir da entrada de controles e ferramentas como rastreadores de problemas, geradores de código, testes automáticos, gerador de documentação e de empacotamento, entre outros.

Neste sentido, o objetivo deste artigo é verificar se são identificadas – em grandes projetos Open Source – regras que direcionem a implementação de boas práticas propostas por modelos de melhoria de processo mais tradicionalmente vinculados à processos robustos e organizacionais.

2. Fundamentação Teórica

2.1. Software Livre e Open Source

No início do surgimento dos computadores, os softwares que rodavam nestas máquinas eram fortemente acoplados ao hardware, de forma que a sua venda era casada e pouco controle se dava sobre as possíveis modificações que o comprador fizesse no software. Porém, já na década de 70, a partir de movimentos de empresas como a Microsoft, a indústria iniciou o fechamento do código fonte para realização de cópias e alterações (Sabino e Kon, 2009).

Dentro deste cenário, que se intensificou ao longo dos anos, Richard Stallman – um cientista da computação – lançou em meados da década de 80 a ideia de Software Livre com a criação do GNU (Gnu's Not Unix), um sistema operacional que seria disponibilizado livremente a todas as pessoas que tivessem interesse. Stallman fundou a *Free Software Foundation*, em 1985, uma entidade sem fins lucrativos que tem como missão promover a liberdade dos usuários de computador, defendendo os direitos de todos, considerando como base as quatro liberdades essenciais (FSF, 2019):

- Liberdade 0: executar o programa, para qualquer propósito;
- Liberdade 1: estudar o programa e modificá-lo para atender às suas necessidades (pressupõe total acesso ao código fonte);
- Liberdade 2: redistribuir cópias de modo a ajudar os outros;
- Liberdade 3: distribuir suas versões modificadas do programa para que a comunidade possa se beneficiar de suas mudanças.

Com o tempo e a adesão de mais entusiastas, estudiosos e institutos de pesquisa importantes da época percebera que este seria um caminho promissor, especialmente visando a melhoria da segurança e confiabilidade dos produtos. Desta forma, fazia-se importante a adoção de um novo termo, que fosse menos confuso para os indivíduos e empresas que estavam se aproximando (Peterson, 2018).

Esta mudança não significava nenhum contraponto ao referencial ideológico proposto pelo termo software livre, o ponto era apenas torná-lo objetivo, para agregar mais adeptos (Peterson, 2018).

Assim, o termo Open Source (código aberto) foi cunhado por Christine Peterson no início de 1998, e em seguida foi criada a Open Source Iniciative (OSI), por Eric Raymond e Bruce Perens. A OSI foi concebida como uma organização educacional, de defesa dos benefícios do código aberto e da construção de pontes entre as diferentes vertentes da comunidade (OSI, 2018).

A adoção do termo foi rápida e contou com o apoio de figuras importantes da comunidade computacional. As iniciativas Open Source multiplicaram-se ao longo dos anos 2000, quando grandes companhias lançaram projetos em plataforma aberta e investiram bilhões de dólares neste segmento.

Considerando os conceitos de Software Livre e Open Source, dentro da sua característica de desenvolvimento distribuído e colaborativo através da Internet, é fundamental que existam comunidades virtuais e ferramentas que possibilitem a interação entre os contribuidores. Dentre estas ferramentas podemos citar sites, blogs, chats, repositórios de arquivos, rastreadores de defeitos, etc.

Apesar de parecer à primeira vista que são caóticos, a maioria dos projetos Open Source têm alguma estrutura, regras, normas e expectativas. Alguns projetos maiores, além de tudo ainda apresentam um modelo de governança claramente descrito no seu website ou documentação (Oiser-Mixon *et al.*, 2019).

Segundo o guia de como contribuir para projetos de Open Source (GitHub, 2019), um projeto típico possui cinco tipos diferentes de perfis envolvidos: o autor, o proprietário, os mantenedores, os contribuidores e os membros da comunidade.

O autor é a pessoa que criou o projeto; o proprietário possui propriedade administrativa na organização ou repositório (pode ser o mesmo autor); os mantenedores são responsáveis por direcionar a visão e gerenciar os aspectos organizacionais do projeto (podem ser proprietários ou autores também); os contribuidores são todos os que já agregaram alguma coisa ao projeto e os membros da comunidade são as pessoas que usam o projeto, que podem ser ativos, direcionando-o com suas opiniões. Projetos maiores também tem subcomitês ou grupos de trabalho focados em tarefas técnicas ou de gerenciamento da comunidade.

No que diz respeito à documentação, geralmente os seguintes arquivos estão presentes e listados no primeiro nível do repositório do projeto, a fim de serem facilmente identificados (GitHub, 2019):

- Licença: por definição todo projeto Open Source precisa identificar seu tipo de licença.
- Readme (leia-me): é o manual de instruções que dá as boas-vindas ao novo membro da comunidade interessado no projeto. Explica o quanto o projeto é útil e como ajuda a comunidade.
- Contribuindo: define as regras de contribuição do projeto, como os tipos de contribuições (código, documentação, traduções, etc.) que são aceitas e o funcionamento do processo.
- Código de conduta: estabelece regras básicas para comportamento dos participantes e facilita um ambiente acolhedor e amigável.
- Outra documentação: pode haver documentação adicional como tutoriais, guias passo-a-passo, políticas de governança, etc.

Para o objetivo deste trabalho, é importante ressaltar a importância do processo de contribuição enquanto principal mecanismo de manutenção do caráter colaborativo e comunitário dos projetos de Open Source.

2.2. O Modelo de Referência: CMMI-Dev 1.3

Os modelos CMMI® (Capability Maturity Model® Integration) são coleções de boas práticas que orientam as organizações na melhoria de seus processos. Estes modelos são desenvolvidos por membros da indústria, governo e do SEI (Software Engineering Institute). Dentre os modelos CMMI, o CMMI-DEV é um guia para aplicação das melhores práticas em organizações de desenvolvimento, focadas em atividades para desenvolver produtos e serviços de qualidade que atendam às necessidades dos usuários finais (SEI, 2010).

A proposta do CMMI-DEV, enquanto modelo de maturidade de processos, é melhorar a forma como a organização faz negócios, de maneira integrada, reduzido as barreiras que existem entre as áreas. As práticas que cobrem todo o ciclo de vida do produto desde a concepção, até a entrega e manutenção, estão contempladas.

O CMMI-DEV contém 22 áreas de processo, organizadas por categoria: apoio, engenharia, gerência de processo e gerência de projeto. Fazem parte de cada categoria as seguintes áreas de processo:

Categoria Gerência de Processo: OPF – Foco no processo organizacional. OPD – Definição do processo organizacional. OT – Treinamento organizacional. OPP – Desempenho do processo organizacional e OID – Gerenciamento do desempenho organizacional.

Categoria Gerência de Projeto: PP – Planejamento de projeto. PMC – Acompanhamento e controle de projeto, SAM – Gerenciamento de acordos com fornecedores, REQM – Gerenciamento de requisitos, IPM – Gerenciamento integrado do projeto, RSKM –Gerenciamento de risco e QPM – Gerenciamento quantitativo do projeto.

Categoria Engenharia: PI – Integração de Produto, RD – Desenvolvimento de requisitos, VER – Verificação, VAL – Validação e TS – Solução técnica.

Categoria Suporte: CM – Gerência de configuração, PPQA – Garantia da qualidade de processo e produto, MA – Medição e análise, DAR – Análise de decisão e resolução e CAR – Análise de causa e resolução.

Cada área de processo agrega um grupo de práticas relacionadas, que quando implementadas corretamente, satisfazem um conjunto de objetivos considerados importantes para alcançar melhorias nesta área.

O CMMI suporta dois caminhos para sua utilização. Um caminho orienta as organizações a melhorar os processos, correspondendo a uma área de processo individual (ou grupo de áreas de processo). O outro caminho orienta as organizações a melhorarem um conjunto de processos relacionados, atendendo conjuntos pré-determinados de áreas de processo. Estes dois caminhos são respectivamente o nível de capacidade e de maturidade (conhecidos como as representações) (Figura 1).

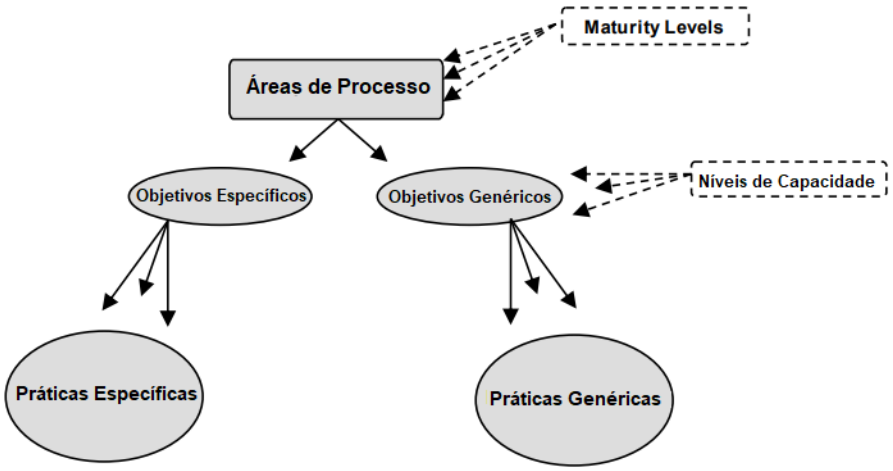

Figura 1 – Estrutura do modelo CMMI-DEV

Níveis de maturidade aplicam-se para obtenção de melhoria de processos através de múltiplas áreas de processo. Estes níveis são uma forma de melhorar os processos correspondentes a um conjunto de áreas de processo (ou seja, nível de maturidade). Os cinco níveis de maturidade são numerados de 1 a 5: inicial, gerenciado, definido, gerenciado quantitativamente e otimizando.

O nível de maturidade estabelece "platôs" evolucionários de melhoria de processo, estabilizando partes importantes do processo organizacional. A abordagem por estágios usa conjuntos pré-definidos de áreas de processo para definir o caminho de melhoria para a organização. Os níveis de maturidade são medidos pelo atendimento de objetivos genéricos e específicos associados com cada conjunto pré-definido de áreas de processo.

Níveis de capacidade aplicam-se para obtenção da melhoria de processos de uma organização em áreas de processo individuais. Estes níveis são um meio de incrementar os processos correspondentes à área de processo em questão. Os quatro níveis de capacidade são numerados de 0 a 3: incompleto, realizado, gerenciado, definido.

Na apresentação contínua a definição do nível de capacidade obtido pela área de processo se dá através do atendimento aos objetivos genéricos, que caracterizam o grau de institucionalização do processo:

- Objetivo Genérico 1: O processo é realizado. Considera-se um processo realizado quando ele satisfaz os objetivos específicos de uma área de processo.
- Objetivo Genérico 2: O processo é gerenciado. Considera-se um processo gerenciado aquele processo realizado, que é planejado e executado de acordo com normas, aplica pessoas qualificadas com recursos adequados para produzir saídas controladas, envolve os patrocinadores relevantes, é monitorado, controlado, revisado e avaliado para aderência na sua descrição.
- Objetivo Genérico 3: O processo é definido. Considera-se um processo definido aquele processo gerenciado, que é instanciado para o conjunto de processos da organização, conforme os guias de adaptação, tem uma descrição de processo.

3. Método

Foram investigados neste estudo, três projetos Open Source de grande porte, que já possuem arquitetura estabelecida e bem definida, cuja evolução - a cargo dos contribuidores vindos da comunidade - consiste basicamente na correção de defeitos (manutenção corretiva), inclusão de novas funcionalidades (manutenção evolutiva) e alterações em documentação. Todos os projetos possuem um comitê de desenvolvedores experientes, responsáveis pela avaliação das solicitações de contribuição.

Segundo Dias (2018), em projetos de manutenção de software, os requisitos podem ser alterados devido a necessidade do usuário, mudança de tecnologia e/ou arquitetura ou por defeitos latentes nos componentes do produto, justificando assim – com a possibilidade do surgimento de novos requisitos – a execução das atividades desta área de processo também neste contexto.

A identificação de evidências de compatibilidade com o CMMI foi realizada através da avaliação dos sites ou repositórios dos projetos, focando nos arquivos que determinam as regras de contribuição da comunidade. Não houve participação efetiva em nenhum projeto, nem através da contribuição de código ou documentação e nem através da participação nos fóruns ou listas de discussões. O objetivo era de fato realizar a análise a partir da ótica que o indivíduo – interessado em contribuir – obtém através da leitura inicial das regras publicadas.

Assim, foi realizado um mapeamento das práticas descritas nas **regras de contribuição** dos projetos selecionados, comparando com as **práticas específicas** descritas na área de processo Solução Técnica do CMMI, por se tratar de um modelo de referência internacional, consolidado, de capacidade e maturidade de processos que reúne melhores práticas da engenharia de software.

3.1 CMMI-DEV 1.3 - Área de Processo Solução Técnica

Da categoria *engenharia*, que engloba 5 (cinco) áreas de processo do framework CMMI, a "Solução Técnica" detalha as melhores práticas referentes à seleção, projeto e implementação dos requisitos (SEI, 2010).

O foco desta área de processo está na avaliação e seleção de soluções (projetos preliminares ou abordagens de projeto) que atendem um conjunto de requisitos de qualidade, no desenvolvimento de projetos detalhados para as soluções selecionadas e na implementação do projeto em um produto ou componentes do produto.

Para que a área de processo Solução Técnica seja considerada implementada, é necessário que todas as três metas específicas (Specific Goal – SG) definidas estejam sendo atendidas, o que se caracteriza pela implementação das respectivas práticas específicas (Specific Practice – SP) no processo utilizado no projeto. Estes objetivos específicos e suas práticas estão listados a seguir:

- SG 1: Selecionar as soluções de componentes do produto.
    - SP 1.1: Elaborar as soluções alternativas e os critérios de solução.
    - SP 1.2: Selecionar as soluções de componentes do produto.
- SG 2: Elaborar o Design.
    - SP 2.1: Elaborar o design do produto ou dos componentes do produto.
    - SP 2.2: Estabelecer um pacote de dados técnicos.
    - SP 2.3: Elaborar o design das interfaces usando os critérios.

    - SP 2.4: Desenvolver, comprar ou reusar análises.
- SG 3: Implementar o Design do Produto.
    - SP 3.1: Implementar o design.
    - SP 3.2: Elaborar a documentação de suporte ao produto.

3.2 Caracterização dos projetos

Para este estudo foram selecionados três projetos considerados referência na comunidade Open Source. São eles:

KDE: KDE é uma das maiores e mais ativas comunidade internacional de desenvolvimento e distribuição de Software Livre e Open Source. Enquanto um framework central de desenvolvimento ele prove ferramentas e recursos que permitem trabalho colaborativo para este tipo de software (KDE, 2019). Mais de 2.500 contribuidores participaram do desenvolvimento de softwares KDE e em torno de 20 novos desenvolvedores contribuem pela primeira vez cada mês. Seus projetos juntos consistem em mais de seis milhões de linhas de código e já foram traduzidos para mais de 108 línguas.

A comunidade KDE mantém também um espelho do repositório no Github e possui hoje uma série de produtos, frameworks e aplicações. Um destes projetos, o Plasma Desktop possui atualmente 151 contribuidores e teve 710 contribuições (*commits*) no último ano (KDE, 2019). Como todos os projetos desta comunidade compartilham um mesmo processo de trabalho, para realização deste estudo foi considerado o conjunto de documentos constantes no endereço geral de todo o KDE: https://community.kde.org/Get_Involved.

Kernel Linux: É o núcleo do sistema operacional Linux criado pelo Linus Torvalds, com a primeira versão oficial lançada em 1991 para seu computador pessoal e sem intenções entre plataformas. A família de sistemas operacionais Linux é baseada nesse kernel e implantada em sistemas de computadores tradicionais, como computadores pessoais e servidores, geralmente na forma de distribuições Linux (Kernel, 2019). Ele conta com 21.343 colaboradores e no último ano foram realizadas 71.667 contribuições (*commits*). Para realização deste estudo foi considerado o conjunto de documentos constantes no endereço github: https://github.com/torvalds/linux/tree/master/Documentation/process.

Ruby-on-Rails: É um framework escrito em Ruby para desenvolvimento de aplicações web que suporta banco de dados de acordo com o padrão Model-View-Controller (MVC), simplificando e abstraindo tarefas comuns. Open Source, livre para uso e aberta para receber contribuições de melhoria (Rails, 2019). Foi criado por David Heinemeier Hansson em 2003 e possui um total de 3.834 contribuidores e 2.747 contribuições (*commits*) no último ano. Endereço examinado: https://guides.rubyonrails.org/contributing_to_ruby_on_rails.html.

4. Resultados

Os resultados da verificação dos processos descritos para cada projeto serão apresentados conforme cada prática específica definida para o processo de Solução Técnica do CMMI, agrupando-as nos objetivos específicos.

Para cada prática específica, algumas considerações referentes à interpretação utilizada para projetos Open Source são apresentadas. Em seguida, é tabelado o mapeamento das regras que foram identificadas como possível implementação da prática para cada projeto examinado.

4.1 Objetivo Específico: SG1 – Selecionar Soluções de Componentes de Produto

Prática Específica: SP 1.1 Desenvolver Soluções Alternativas e Critérios de Seleção

Identificar e analisar soluções alternativas para que seja escolhida a opção mais aplicável em termos de custo, prazo e desempenho técnico, considerando inclusive a possibilidade da utilização de produtos "comerciais de prateleira".

Considerações para o mapeamento:

- Por motivos intrínsecos ao desenvolvimento de software Open Source, nenhum dos três projetos considera a utilização de produtos comerciais de prateleira;
- Não foi possível identificar em nenhum dos 3 projetos os critérios pré-estabelecidos para avaliação de arquitetura - provavelmente - por motivos diversos, como o fato de estes projetos já terem a arquitetura estabelecido e as regras para contribuições que foram consideradas para a análise serem basicamente de manutenção corretiva e evolutiva de novas funcionalidades sobre a mesma plataforma arquitetural previamente definida;
- Para esta prática, foram selecionadas algumas regras de submissão de funcionalidades que alteram requisitos já estabelecidos de segurança ou aspectos da arquitetura – que seriam alternativas às soluções já existentes. Vale a pena ressaltar mais uma vez que não foi possível identificar os critérios de avaliação destas propostas de maneira explícita;
- Padrões de codificação e projeto também foram considerados como critérios para aplicação da seleção de soluções, pois caso o envio de contribuições seja feito fora dos padrões dificilmente serão aceitas.

Tabela 1 - Mapeamento SP1.1

KDE	Não foram localizadas nos sites examinados as regras sobre a inclusão de novas funcionalidades ou funcionalidades core que alterem segurança ou arquitetura.
Linux – Kernel	Detalha algumas questões que devem ser respondidas antes de sugerir uma solução nova e também propõe iniciar um diálogo com a comunidade de desenvolvedores, através das listas de discussão já estabelecidas conforme o assunto que será tratado. Ressalta que realizar estas atividades antes de começar a codificar a mudança pode evitar a proposição de trabalhos duplicados ou que estejam divergentes da arquitetura ou tecnologia adotada. https://github.com/torvalds/linux/blob/master/Documentation/process/3.Early-stage.rst

Ruby-on-Rails	Sugere diretamente que contribuições que podem ter impacto no desempenho façam um benchmark do código para avaliação do impacto https://guides.rubyonrails.org/contributing_to_ruby_on_rails.html (Seção 5.6 Benchmark Your Code)

Prática Específica: SP 1.2 Selecionar Soluções de Componentes de Produto

Selecionar as soluções mais adequadas conforme os critérios previamente estabelecidos.

Considerações para o mapeamento:

- A seleção da solução adequada acontece no aceite das contribuições enviadas. Este processo não está definido nos projetos examinados, mas foi considerado que pelo menos os critérios citados na prática anterior são observados.
- Não há uma seleção de projeto (design), mas sim uma seleção já da contribuição implementada em código fonte.

Tabela 2 - Mapeamento SP1.2

KDE	Explica como se dá o processo de envio da solução pelos contribuidores, não detalha como é feita a escolha ou o que acontece após o aceite da solução proposta. https://community.kde.org/Get_Involved/development (Submit a patch)
Linux – Kernel	Explica como funciona o processo de aceite de uma solução e as possíveis interações para ajustes conforme os padrões de qualidade estabelecidos https://github.com/torvalds/linux/blob/master/Documentation/process/6.Followthrough.rst
Ruby-on-Rails	Explica como se dá o processo de envio da solução pelos contribuidores e os retorno sobre seu aceite com possíveis ajustes mediante os critérios utilizados pela equipe principal de desenvolvimento https://guides.rubyonrails.org/contributing_to_ruby_on_rails.htm (Seção 5.14 Issue a Pull Request e Seção 5.15 Get some Feedback)

4.2 Objetivo Específico: SG2 – Desenvolver Design

Prática Específica: SP 2.1 Desenvolver o Design do Produto ou dos Componentes de Produto

Elaborar documentação do produto ou componente a fim de fornecer informações adequadas para implementação e outras fases do processo como modificação, manutenção, etc.

Considerações para o mapeamento:

- Em nenhum projeto analisado foi identificada a exigência do envio de projeto técnico antes da geração de código.
- Para as contribuições baseadas em código fonte, foram consideradas as orientações de envio de atualização da documentação técnica envolvida com o código submetido.

Tabela 3 - Mapeamento SP2.1

KDE	No guia de desenvolvimento há uma orientação de boa prática para documentação de API, levando para uma página específica detalhada de como documentar as APIs e sua importância https://community.kde.org/Get_Involved/development / Seção "Best practices & other useful information" cita a página https://community.kde.org/Guidelines_and_HOWTOs/API_Documentation
Linux – Kernel	O processo de codificação ressalta a necessidade de o código submetido ser documentado, inclusive como pré-condição para aceite da contribuição. A orientação cita inclusive diversos tipos de documentação necessários conforme o tipo de contribuição que está sendo feito. https://github.com/torvalds/linux/blob/master/Documentation/process/4.Coding.rst (Seção: Documentação)
Ruby-on-Rails	Oferece orientação para se atualizar toda documentação afetada pela atualização a ser enviada. https://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html (Seção: Write Your Code)

Prática Específica: SP 2.2 Estabelecer Pacote de Dados Técnicos

Elaborar um pacote de dados técnicos para fornecer ao desenvolvedor uma descrição detalhada do produto à medida que ele é construído.

Considerações para o mapeamento:

- Considerando que a documentação técnica dos projetos examinados é integrada às versões juntamente com o código fonte, podemos estabelecer que os pacotes de dados técnicos sugeridos nesta prática são caracterizados pelas versões mais atuais em cada ramo de desenvolvimento ativo. Esta interpretação configura o caráter dinâmico do desenvolvimento Open Source em comunidades;
- Além dos próprios repositórios de trabalho, com as versões mais atuais, listamos os sites que oferecem para o público geral as versões disponibilizadas de cada projeto.

Tabela 4 - Mapeamento SP2.2

KDE	Repositório de trabalho: https://github.com/KDE/ (espelho somente leitura) Site com versões disponíveis: https://kde.org/announcements/
Linux – Kernel	Repositório de trabalho: https://github.com/torvalds/linux Site com versões disponíveis: https://www.kernel.org/
Ruby-on-Rails	Repositório de trabalho: https://github.com/rails/rails https://www.ruby-lang.org/en/downloads/

Prática Específica: SP 2.3 Projetar Interfaces Utilizando Critérios

Projetar as interfaces dos componentes do produto a partir dos critérios estabelecidos e mantidos.

Considerações para o mapeamento:

- Nos projetos examinados foram consideradas apenas as interfaces entre componentes de software;
- As interfaces dos componentes de software são projetadas e avaliadas conforme os mesmos critérios estabelecidos para o restante do código fonte;

Tabela 5 - Mapeamento SP2.3

KDE	Não identificado.
Linux – Kernel	Não identificado.
Ruby-on-Rails	Não identificado.

Prática Específica: SP 2.4 Analisar Alternativas: Desenvolver, Comprar ou Reusar

Avaliar se os componentes do produto devem ser desenvolvidos, comprados ou reusados, com base em critérios estabelecidos.

Considerações para o mapeamento:

- Por motivos intrínsecos ao desenvolvimento de software Open Source, nenhum dos projetos considera a possibilidade de comprar componentes;
- Não foi possível mapear atividades referentes a esta prática em nenhum dos projetos examinados.

Tabela 6 - Mapeamento SP2.4

KDE	Não identificado.
Linux – Kernel	Não identificado.

Ruby-on-Rails	Não identificado.

4.3 Objetivo Específico: SG3 – Implementar Design do Produto

Prática Específica: SP 3.1 Implementar Design

Implementar efetivamente o componente de software conforme projetado, considerando a realização de testes unitários prévias à integração e elaboração da documentação final.

Considerações para o mapeamento:

- São considerados nos projetos examinados as práticas referentes à efetiva codificação do produto pelo contribuidor, a realização dos testes unitários e geração da documentação de usuário.;
- Regras para reuso, uso de padrões e revisões em pares também caracterizam esta prática.

Tabela 7 - Mapeamento SP3.1

KDE	Orienta a criação e preparação de ambiente, o uso de frameworks, bem como oferece diretrizes para realização de testes unitários, escrita de documentação e revisão de código. A atividade de realização da revisão não está descrita no guia, mas é possível identificar no site*abaixo listado. https://community.kde.org/Get_Involved/development https://community.kde.org/Guidelines_and_HOWTOs/UnitTests *https://community.kde.org/Infrastructure/Phabricator#How_to_review_someone_else.27s_patch
Linux – Kernel	O processo de codificação como um todo é descrito e são apresentados links para páginas com padrão de codificação e documentação do código https://github.com/torvalds/linux/blob/master/Documentation/process/4.Coding.rst https://github.com/torvalds/linux/blob/master/Documentation/process/coding-style.rst https://github.com/torvalds/linux/blob/master/Documentation/doc-guide/kernel-doc.rst Sugere, no padrão de envio da contribuição, que os testes devem ter sido realizados e o código revisado também oferecendo até um modelo de “declaração de supervisão do revisor” para ser utilizado. https://github.com/torvalds/linux/blob/master/Documentation/process/5.Posting.rst https://github.com/torvalds/linux/blob/master/Documentation/process/submitting-patches.rst

Ruby-on-Rails	Todo o processo de criação do ambiente, construção do código fonte e realização de testes unitários é mencionada, incluindo padrões de codificação que devem ser considerados. https://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html (Seção 5 Contributing to the Rails Code)

Prática Específica: SP 3.2 Elaborar Documentação de Suporte ao Produto

Elaborar e atualizar a documentação utilizada para instalar, operar e manter o produto.

Considerações para o mapeamento:

- Quando se trata de projetos Open Source, as documentações também são consideradas como ativos, passíveis de contribuições, e desenvolvidas - em alguns casos - por equipes específicas para este fim. Cabe aos desenvolvedores contribuidores as documentações mais técnicas ou manuais para uso de outros desenvolvedores, já descritos como atendidos no SP 3.1;
- Os manuais de apoio à manutenção são basicamente todo o arcabouço de instruções (inclusive do próprio processo examinado) que dá apoio às atividades de interação da comunidade de desenvolvedores;
- Os manuais de operação e de instalação serão tratados respectivamente como (manual do usuário final) e guia de instalação das versões atuais dos softwares gerados pela comunidade;
- Um aspecto importante também identificado foram as iniciativas de tradução destas documentações

Tabela 8 - Mapeamento SP3.2

KDE	São citados os dois tipos de documentação que existem no KDE, os "*context helps*" que explicam itens individuais da interface na tela e manuais de aplicativos. Esta página orienta os passos para contribuir com a documentação, desde as listas para comunicação com outros contribuidores, passando pelas orientações para construir uma versão atualizada da aplicação a ser documentada, pelo formato padrão que deve ser utilizado e por uma lista de tarefas de documentação pendentes que podem ser executadas. https://community.kde.org/Get_Involved/documentation
Linux – Kernel	Referência uma página que coloca em termos de recomendação a inclusão de documentação de manual de uso quando novos recursos são adicionados ao software. Também recomenda envio de informações ou alterações já sugeridas para os mantenedores das páginas de manual quando a alteração realizada altera as interfaces com o usuário. https://github.com/torvalds/linux/blob/master/Documentation/process/8.Conclusion.rst

	https://github.com/torvalds/linux/blob/master/Documentation/process/howto.rst O processo também aponta o site que fornece a documentação mais atualizada e em constante atualização. https://github.com/torvalds/linux/blob/master/Documentation/process/8.Conclusion.rst aponta para https://www.kernel.org/
Ruby-on-Rails	Aponta os processos e padrões para apoio na geração dos dois principais conjuntos de documentação: os guias e as documentações de APIs. https://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.htm (Seção: 3 Contributing to the Rails Documentation) Orienta também como traduzir os guias para outras línguas. https://edgeguides.rubyonrails.org/contributing_to_ruby_on_rails.html (Seção: 4 Translating Rails Guides)

5. Discussão

A partir da verificação das regras de contribuição dos projetos foi possível mapear a presença (ou não) das boas práticas listadas no processo de Solução Técnica do CMMI. A tabela de mapeamento (tabela 9) aponta, de maneira objetiva, quais práticas foram identificadas em quais projetos.

Tabela 9 - Mapeamento Regras x Práticas

Prática	KDE	Linux - Kernel	Ruby-on-Rails
SP1.1	**Não**	Sim	Sim
SP1.2	Sim	Sim	Sim
SP2.1	Sim	Sim	Sim
SP2.2	Sim	Sim	Sim
SP2.3	**Não**	**Não**	**Não**
SP2.4	**Não**	**Não**	**Não**
SP3.1	Sim	Sim	Sim
SP3.2	Sim	Sim	Sim

Dos três objetivos genéricos, podemos perceber que dois foram integralmente atendidos pelos projetos Kernel Linux e Ruby-on-Rails, restando apenas o KDE, que não

apresentava nas regras publicadas os critérios de avaliação para funcionalidades que alterassem os requisitos.

Apenas um objetivo genérico foi contemplado parcialmente por todos os projetos examinados e podemos justificar a não aderência integral por características intrínsecas ao desenvolvimento colaborativo comum aos projetos Open Source.

Uma das práticas não identificadas nas regras refere-se à avaliação de comprar, reusar ou fazer (*make or buy*) que obviamente não é uma opção para os projetos Open Source devido à sua natureza livre de compartilhar e alterar o código fonte com a comunidade, impossibilitando a inclusão de componentes proprietários ou de código fechado.

A outra prática que não foi caracterizada nas regras examinadas explica-se pela condição do desenvolvimento colaborativo, uma vez que as mesmas regras que se aplicam à codificação também se aplicam às interfaces dos componentes. Em verdade, uma outra interpretação poderia ser a de considerar esta prática atendida através dos mesmos padrões de codificação dos componentes do produto, listadas na SP2.1.

É importante mencionar a interpretação aplicada na avaliação da prática SP2.2, que consiste na exigência de versões estáveis de projeto (documentação de requisitos e *design*) como insumo para a posterior construção do software. Aparentemente este poderia ser o principal ponto de divergência entre os processos mais estruturados e o desenvolvimento colaborativo dos projetos examinados, porém, quando se considera que todo o código fonte, os manuais, guias e demais documentações (quando existentes), além de todos os registros de defeitos (em aberto e já corrigidos) são de livre acesso à toda a comunidade é possível se constatar que um pacote de dados técnicos está SEMPRE atualizado e disponível para quem atua contribuindo com código fonte.

Analisando os resultados é possível perceber que, apesar da aparente falta de rigor nas regras de contribuição nos projetos Open Source, e a histórica relação do projeto do Kernel Linux com o estilo de desenvolvimento “bazar”, há quase totalidade de cobertura para o processo de Solução Técnica do CMMI, um dos mais robustos modelos de referência para melhoria de processos.

6. Conclusão

Foi apresentado neste trabalho um estudo observacional acerca da existência de boas práticas de processo descritas em regras de contribuição de grandes projetos Open Source.

Como referência foi utilizado o processo Solução Técnica do CMMI-DEV 1.3, por se tratar de um modelo já estabelecido e tradicionalmente vinculado à processos robustos e organizacionais. A avaliação foi realizada em três projetos considerados referência na comunidade Open Source: O Kernel Linux, o Ruby-on-Rails e a comunidade KDE, que agrega uma série de projetos sob as mesmas regras de contribuição.

Após a análise das regras de contribuição dos referidos projetos, em busca de orientações que convergissem para as práticas específicas do processo de Solução Técnica

do CMMI, foi verificada uma adesão quase total, levando à conclusão de que apesar de os processos de desenvolvimento de grandes comunidades Open Source serem geralmente descritos – inclusive na literatura histórica de referência – como pouco rigorosos, pode-se dizer que as suas regras de contribuição contemplam as boas práticas da engenharia de software para o processo de desenvolvimento da solução técnica.

Como trabalho futuro, pode ser citada a expansão deste estudo para os demais processos de engenharia ou mesmo de gerenciamento, visto que há, em todas as comunidades analisadas, um time de referência, responsável por dar o direcionamento técnico e estratégico dos grandes projetos Open Source.

Referências

Beck K., Beedle M., Bennekum A. v., CockburnWard A., Cunningham W., Fowler M., Grenning J., Highsmith J., Hunt A., Jeffries R., Kern J., Marick B., Martin R. C., Mellor S., Schwaber K., Sutherland J., Thomas D. (2001) "Manifesto para o desenvolvimento ágil de software", https://www.manifestoagil.com.br/, Julho.

KDE (2019) "About KDE", https://kde.org/community/whatiskde/, Julho.

Dias, E.C. (2018). Processo de Solução Técnica do CMMI-DEV par Projetos de Manutenção de Software: um Estudo de Caso de Implementação. [Dissertação]. Belém: Universidade Federal do Pará; 2018.

FSF (2019) "Free software is software that gives you the user the freedom to share, study and modify it. We call this free software because the user is free", https://www.fsf.org/about/what-is-free-software, Julho.

GitHub (2019) "How to Contribute to Open Source", https://opensource.guide/how-to-contribute/, Julho.

Kernel (2019) "The Linux Kernel Archives" https://www.kernel.org, Julho.

Oiser-Mixon J., Yehuda G., Martin G., Ruff N., Varley I. (2019) "Recruiting Open Source Developers", https://www.linuxfoundation.org/resources/open-source-guides/recruiting-open-source-developers/, Julho.

OSI (2018) "History of the OSI", https://opensource.org/history, Julho.

Peterson, C. (2018). "How i coined the term 'open source'", https://opensource.com/article/18/2/coining-term-open-source-software, Julho.

Robbins J. E. (2003). "Adopting Open Source Software Engineering (OSSE) Practices by Adopting OSSE Tools", http://www.methodlabs.com/jrobbins/papers/robbins-msotb.pdf, Julho.

Rails (2019) "Rails", https://rubyonrails.org/, Julho.

Raymond, E. S. (1999). The Cathedral and the Bazaar. O'Reilly & Associates, Inc.,Sebastopol, CA, USA, 1st edition.

Sabino, V. and Kon, F. (2009). Licenças de Software Livre - História e Características. Technical Report RT MAC-IME-USP 2009-01.

SEI (2010) "CMMI Product Team. CMMI for Development - version 1.3". http://cmmiinstitute.com/resources/cmmi-development-version-13, Julho.