UNIVERSIDADE DA REGIÃO DE JOINVILLE – UNIVILLE

BACHARELADO EM ENGENHARIA DE SOFTWARE

ESTUDO DE CASO DE MÉTODOS E FERRAMENTAS DE DESENVOLVIMENTO DE DOCUMENTAÇÃO NA METODOLOGIA ÁGIL SCRUM

JONATHAN JUARES BEBER

Anderson

Projeto Integrador

Joinville – SC

2015

1. DEFINIÇÃO DO PROJETO

1.1. Título

Estudo de caso de métodos e ferramentas de desenvolvimento de documentação na metodologia ágil Scrum.

1.2. Tema

Um produto de software é caracterizado por diversas partes e entre elas está enquadrada a documentação. A documentação mais do que parte do produto de software, é parte essencial dos requisitos de qualidade do mesmo. Nesse momento não está sendo citado a documentação para clientes, e sim a documentação da aplicação para equipe técnica e gerencial responsável pelo software.

Atualmente a utilização de métodos ágeis está mais comum, prevalecendo o desenvolvimento de forma rápida e eficaz, resuzindo tempos de entrega e prazos. Essa nova metodologia também engloba a definição de que é necessário mais atenção ao software funcional como um todo do que documentação compreensiva.

Dessa forma o tema proposto visa analisar os métodos e ferramentas que possam ser utilizados em conjunto com um gerencimento ágil de projeto de software, trazendo a capacidade de equipes de manutenção, implantação e personalização do software serem capazes de compreender o processo de desenvolvimento e detalhes da funcionalidade do produto sem a necessidade de analisar todo o código, o que muitas vezes não se torna possível e traz sérios riscos para segurança da aplicação ou agride regras de modularização do código.

1.3. Problema

A documentação de software vem sendo tranformada pela utilização de metodologia ágeis de gerenciamento de projetos de software, como o Scrum. Essas metodologias trazem em si a ideia de não serem como as metodologias tradicionais, também chamadas de “metodologias baseadas em documentação.”

As metodologias mais tradicionais acabavam por engessar o desenvolvimento, tornando-o demorado e cheio de processos lentos, muitas vezes, gerando documentação desnecessária. Cabe relembrar que o uso dessas metodologias é extramente necessário em casos de sistemas críticos. A quebra de paradigma, saindo de meotodologias tradicionais para novas metodologias traz alguns pontos que merecem destaque:

- Como gerar documentação, de forma ágil, sem engessar o desenvolvimento, porém, provendo todas as informações necessárias para equipes de implantação, manutenção conseguirem realizar suas tarefas, incluindo pequenas personalizações?

- Quais os métodos e ferramentas disponíveis que podem ser utilizados junto a metodologias ágeis, capazes de gerar documentação também de forma ágil?

- Qual a melhor forma de avaliar ferramentas e métodos já especificados?

1.4. Objetivo geral

Identificar e analisar ferramentas e métodos para geração de documentação em ambientes de desenvolvimento com a metodologia ágil scrum.

1.5. Objetivos específicos

Identificar qual a documentação necessária para equipes de manutenção e implantação realizarem sua tarefas.

Identificar métodos e ferramentas disponíveis para documentação que possam ser incluídos no framework scrum.

Verificar a melhor forma para avaliar as ferramentas e métodos já identificados.

Avaliar as ferramentas e métodos já identificados.

1.6. Justificativa

A utlização de métodos ágeis traz como padrão a norma de que o próprio código é a documentação. Essa prática auxilia no desenvovlimento de forma rápida e em teoria força o desenvolvedor para utilização de boas práticas em seu código.

Porém a documentação ainda se faz, muitas vezes, necesária. Sem ela equipes de suporte necessitariam analisar todo o código para realizar implantações de funcionalidades ou pequenas personalições. O manifesto ágil deixa claro que o software funcionalé mais importante do que a documentação clara, porém, a probletimzação já apresentada demonstra a necessidade da documentação e a necessidade de um processo capaz de gerar documentação também de forma ágil.

As práticas e ferramentas para geração de documentação auxiliam a cumprir essa tarefa de forma prática, não comprometendo a agilidade do desenvolvimento e facilitando a manutenção, implantação e customização. Porém é necessário um estudo voltado para as mesmas, definindo formas de comparação e qualidade, sem ferir o manifesto ágil.

1.7 Processos de software

Em meio de projetos, seja esses grandes ou pequenos, são listada atividades, ações e tarefas que são necessárias para cumprimento de uma meta que chama-se produto de trabalho, uma tradução de work product. A diferenciação entre os três termos citados é claramente apresentada por Pressmann (2011) afirmando que uma atividade busca “atingir um objetivo amplo” enquanto uma ação busca atingir um objetivo menos amplo enquanto constrói um artefato de software fundamental. O autor ainda afirma que a tarefa tem um objetivo simples e bem definido, capaz de gerar um resultado tangível, que possa ser medido.

Todas as ações, tarefas e atividades precisam ser definidas, assim como os recursos que o processo de desenvolvimento de software contemplará. Esse conjunto é dito como o processo de software em si. Reis (2003) define processo de software como “um conjunto de atividades realizadas para construir software, levando em consideração os produtos sendo construídos, as pessoas envolvidas, e as ferramentas com as quais trabalham” e Somerville (2011) como “um conjunto de atividades que leva à produção de um produto de software”. Somerville (2011) ainda comenta que é cada vez mais comum que processos de software foquem em modificar e ampliar sistemas já exisitentes e da realização de ajustes em componentes de softwares ou softwares comerciais.

Dessa forma o processo de software guiará o desenvolvimento, porém, não poderá descrever o desenvolvimento como um todo e sim o contrário, como afirma Pressman (2011) onde diz que o processo precisa ser adaptável, dando liberdade as pessoas envolvidas de escolher um pacote de atividades, tarefas e ações que forem necessárias. Assim, Pressman (2011) mostra que o processo de software não visa engessar o desenvolvimento ou prescrever todas as etapas, mas sim, auxiliar na entrega “dentro do prazo e com qualidade suficiente para satisfazer àqueles que patrocinaram sua criação e àqueles que irão utilizá-lo”.

As metodologias de processos de software visam englobar as atividades essenciais para engenharia de software e dessa forma apresentar um alicerce

para o crescimento do processo de desenvolvimento. Além dessa atividades essenciais, Pressman (2011) cita que as metodologias também abordam atividades de apoio que são necessárias em todo o processo de software.

As atividades essenciais anteriormente citadas são definidas por Reis (2003) como “fundamentais para contrução de software” e o mesmo autor apresenta algumas delas como análise e definição de requisitos, projeto arquiteturado e detalhado, codificação, teste de unidade, de integração e de sistema e lançamento. Nesse quesito é possível perceber que até mesmo os autores não concordam na nomenclatura das atividades essenciais. Observar-se que Pressman (2011) cita como atividades essenciais a comunicação, planejamento, modelagem, contrução e emprego. Somerville (2011) cita especificação de software, projeto e implementação de software, validação de software e evolução do software.

Por mais que os autores usem nomes diferentes para as tarefas por eles consideradas vitais sua estrutura e objetivo são semelhantes. Pressman (2011) define que a comunicação é a etapa responsável por manter um contato inicial com o cliente e partes interessadas, *stakeholders*, de forma clara ele define que essa atividade “a interção é compreender os objetivos das partes interessadas para o projeto e fazer o levantamento das necessidades que ajudarão a definir as funções e características do software”. Tal definição tem semelhança com a atividade de análise e definição de requisitos, definida por Reis (2003): “é o processo de descobrir e detalhar quais funcionalidades (e quais limitações funcionais) o software requer. Somerville (2011) se assemelha muito em sua definição de especificação de software, novamente destacando que essa etapa tem a função do levantamento de requisitos.

Em sua segunda atividade essencial Pressman (2011), Somerville (2011) e Reis (2003) definem que, planejamento, projeto e implemnetação de software e projeto arquitetural e detalhado, respectivamente, tem a função de planejar e definir um planejamento completo de como o software deverá ser desenvolvido, porém, nessa etapa Somerville (2011) inclui também o desenvolvimento do software, uma etapa que só será aborda por Pressman (2011) na atividade de codificação e por Reis (2003) na atividade de contrução.

Reis (2003) inclui nessa estapa as tarefas de modelagem do software, que por sua vez, só será abordada por Pressman (2011) na etapa de modelagem, que o mesmo autor considera uma etapa especial e explica que “um engenheiro de software cria modelos para melhor entender as necessidades do software e o projeto que irá atender suas necessidades”.

A etapa de testes também é essencial para garantir a qualidade do software e cumprir a missão da engenharia de software que é manter o foco justamente na qualidade do produto sendo gerado. Pressman (2011) inclui esse teste dentro da etapa de contrução, não criando uma atividade exclusiva para eles como faz Somerville (2011) e Reis (2003) nas atividades de validação de software e teste de unidade e de sistema, respectivamente. Reis (2003) defende uma atividade exclusiva para testes afirmando que “a atividade de teste envolve uma abordagem mais sistemática.” Reis (2003) ainda lembra que a metodologia de XP (eXtreme Programming) define que os testes devem ser definidos antes mesmo da codificação, desta forma, apesar de serem definidos como uma atividade separada, sua separação deve ser feita para fins gerenciais e não para definir que ela não deve ser executada junto a codificação.

Por fim os autores aqui citados definem a entrega do software, definindo essa como etapa também essencial e crítica. Reis (2003) aponta que esse, muito provavelmente, será o primeiro contato do software com o cliente e é nessa etapa que ocorre a “consolidação da documentação e ajuda online”. Somerville (2011) chama essa etapa de evolução de software, lembrando que tarefas de manutenção e aperfeiçoamento continuam a ocorrer para “atender mutáveis do cliente.”

1.8 Metodologias ágeis

Com o passar do tempo e a popularização de projetos de software os métodos da engenharia de software tradicional não atendiam mais pequenas casas de software e empresas que precisavam atender clientes como focos em um software com muitas regras de negócio. Alguns autores como Soares (2004) nomeiam os métodos tradicionais de engenharia de software como pesados ou orientados a objetos. Esses métodos utilizam-se de uma extensa documentação, pois seu foco são sistema críticos e que podem ser desenvolvidos por diversas pessoas em locais diferentes como mostra Somerville (2011): Essa visão vem da comunidade de engenharia de software que foi responsável por desenvolver grandes e duráveis sistemas de software como sistemas aeroespaciais e governamentais.”

Porém, com a evolução da economia , clientes possuem requisitos diferentes em um curto período de tempo e as engenharias de software tradicionais tem dificuldade para lidar com a mudança de requisitos durante o período de desenvolvimento. Após a etapa de levamento de requisitos os métodos tradicionais possuem diversas etapas já definidas e que uma mudança nos requisitos afetaria todo o processo de desenvolvimento subsequente. Pressman (2011) cita que “nos sistemas de negócios com requisitos que mudam rapidamente um processo flexível e ágil é provavelmente mais eficaz.”

A essência dos métodos ágeis é a capacidade de mudança sem aumentar o custo drasticamente. Muitos projetos não podem ter seu levamento de requisitos finalizado antes do início da codificação e, por esse motivo, a equipe de desenvolvimento necessita ser ágil o suficiente para receber o projeto e concluir esse sempre dentro do prazo.

O manifesto ágil é um movimento que busca um revolução nas metodologias de processo de software, realizado em 2001 por 17 programadores experientes em consultoria de software. O manifesto entre outras coisas pregava que processos de software ágeis valorizam:

- Indíviduos e interações acima de processos e ferramentas;
- Software operacional acima de documentação completa;
- Colaboração de cliente acima de negociação contratual;
- Respostas a mudanças acima de seguir um plano.

Com base nos valores, citados pelos seus autores, é possível perceber que métodos ágeis valorizam a interação com pessoas mais do que a codificação e em etapas a serem percorridas, como afirma Soares (2004): “a ideia das metodologias ágeis é o enfque nas pessoas, não em processos ou algoritmos.” Pressman (2011) também cita que “a maior prioridade é satisfazer o cliente.” Somerville (2011) ainda demonstra que essa preocupação é válida, porém, é necessário cuidado no envolvimento demasiado do cliente pois nem sempre o mesmo terá disponibilidade para o projeto de software, principalemente em projetos realizados por terceiros.

Essa resposta rápida a mudanças, já citada, é essencial para que as metodologias ágeis cumpram seu objetivo, de respeitar prazos e a expectativa de seus clientes. Se analisada a curva, de custo para mudanças, dentro de metodologias de software tradicionais, como o modelo em cascata, quanto maior o tempo passado do projeto, maior o custo para essa mudança. Isso se dá pela necessidade de gerar uma nova documentação completa, análise e revisão de requisitos, aprovação em cada etapa para que a próxima possa ser executada, aumentando em muito o período de desenvolvimento. Segundo Pressman (2011) “um processo ágil bem elabora achata o custo de mudança.” O manisfesto ágil deixa claro que é necessário aceitar as mudanças, acolhendo-as bem, mesmo se atrasadas, pois essa é uma vantagem competitiva para a relação com o cliente.

Toda mudança só é possível de forma rápida através de entregas incrementais, feitas em períodos curtos. Segundo Pressman (2011) um processo precisa ser adaptável incrementalmente. Para atingir essa adaotabilidade a equipe de desenvolvimento conta com feedbacks constantes dos clientes, através de liberações de software operacional, também constante.

Na maioria dos casos são feitas entregas de software operacional em perídos curtos, entre uma e quatro semanas.

Dentro do manifesto da metodologia ágil também está incluído a organização da equipe. Segundo Pressman (2011) a equipe precisa ter como base competência, fico comum, colaboração, habilidade na tomada de decisão, habilidade na solução de problemas ambpiguos, confiana mútua e respeito e auto organização. Esses valores se fazem essenciais, pois as equipes ágeis precisam atuar rapidamente e de forma correta. Métodos ágeis deixam clara a ideia de que as equipes devem ser independentes, como no ponto de auto organização cita Pressman (2011) ao descrever que “nada desmotiva tanto uma equipe como um terceiro assumir compromissos por ela. Nada motiva tanto uma equipe quanto aceitar a responsabilidade de cumprir completamente o prometido feito por ela própria.”

1.9 SCRUM

O scrum é uma das metodologias ágeis mais utilizadas. Ele consiste em fazer o time trabalhar foca em um único objetivo, como já citado por Pressman (2011) para definição de metodos ágeis e como a jogada de rugby, de onde seu deriva segundo Machado (2005): “É uma jogada onde um time de oito integrantes trabalha em conjunto para levar a bola adiante no campo do adversário e concretizar seu único objetivo, o goal.”

O scrum é um framework para gerenciamento de projetos de software que respeita o manifesto ágil, assim como afirma Pressman (2011): “os princípios do Scrum são consistentes com o manifesto ágil e servem para orientar as atividades de desenvolvimento dentro de um processo...”. O scrum não cita atividades técnicas como pair programming ou a prática de primeiro escrever testes, conhecida como test first, como é possível perceber com Somerville (2011): “é um método ágil genérico, porém, focado em gerenciamento de desenvolvimento incremental no lugar de técnicas específicas de engenharia de software ágil.” Porém o scrum é um framework

adaptativo, que permite aos seus usuário utilizar-se de técnicas paralelas, conforme Pressman (2011) novamente: “porém ele pode ser utilizado com técnicas de abordagem ágil.”

O scrum tem em sua base entregas completas de software em períodos curto de tempo, de duas a quatro semanas. Nessa entrega o cliente poderá usufruir de novas funcionalidades, respeitando a regra de ter entregas funcionais. Dentro desse período conhecido com sprint são realizadas as terefas de, segundo Pressman (2011):

- requisitos;
- análise;
- projeto;
- evolução;
- entrega.

Essas atividades já foram descritas pelo mesmo autor como tarefas essenciais para o processo de software. O tamanho da sprint é adaptável e varia conforme a quantidade de funcionalidades e de tarefas há serem realizadas. Os padrões utilizados pelo scrum provam ser bem utilizados para entrega de software em prazos apertados. Parte disso se deve a sua objetividade, conforme cita Machado (2005): “o scrum é bastante objetivo, possuindo metas claras, equipe bem definida, flexibilidade, comprometimento, cooperação e sua curva de aprendizagem é relativamente baixa.”

O próprio criador do scrum define que ele não é previsível e que seu uso é justamente indicado para projetos onde os métodos ágeis são mais eficazes: onde os requisitos não podem ser claramente e/ou completamente definidos no início do projeto. Porém, seu uso permite a toda equipe ter uma visão geral do projeto como um todo, tornando este processo mais claro.

A estrutura do scrum define ainda que são necessárias reuniões diárias, afim de responder três perguntas principais:

- O que você realizou desde a última reunião da equipe?
- Quais os impedimentos encontrados?
- O que irá fazer até a próxima reunião da equipe?

As perguntas permitem a equipe saber aquilo está sendo feito, as dificuldades encontradas e solucionar problemas que possam ocorrer. As mesmas devem ser rápidas, com cerca de 15 minutos apenas. Muitas equipes adotam a cultura de realizar essas reuniões em pé para que a mesma não se prolongue e perca seu foco.

Da reunião diária deve participar apenas o time de desenvolvimento, e é papel do scrum master guiar essa tarefa. O scrum master deve ser um desenvolvedor capaz de guiar a reunião, livrar a equipe, chamada de scrum team, de distrações externas durante a reunião, auxiliar na decisão de quais funcionalidades seram removidos do backlog e jogadas para próxima sprint.

O backlog do produto, dentro da metodologia do scrum é a junção de todas as atividades que devem ser desenvolvidas. Essa pilha cresce e se altera conforme o feedback dos stakeholders é fornecido. É importante perceber que durante a sprint não são inseridas novas funcionalidades ou atividades nela. As funcinalidades e atividades novas são inseridas no backlog e tratadas na próxima sprint se a equipe julgar necessário. Isso, segundo Pressman (2011), permite ao desenvolvedor um periódo breve mas estável, garantindo um foco para o time.

1.10 Documentação de software

A documentação de software é essencial para que o mesmo seja dito como um produto de software completo. Suas várias formas abrem espaço para uma grande área de pesquisa para dicussão sobre qual a forma correta de realizá-la. Falbo (2004) define a documentação como “um aspecto essencial para a qualidade de um sistema de software...”.

Anquetil (2004) define que a documentação de software é “um artefato cuja finalidade seja comunicar a informação sobre o sistema de software ao qual ela pertence.” É importante dar destaque ao termo, utilizado por Anquetil (2004), aterfato, que faz com que a documentação seja vista como uma ferramenta facilitadora, seja esta para manutenção, implementações futuras, implantações ou quaisquer tarefas necessárias. Falbo (2004) destaca essa funções ao relatar que documentações completas “permitem que modificações sejam mais facilmente realizadas, simplificando a tarefa de manutenção.”

Sanches apud Nunes (2005) eleva a documentação a um nível tão importante quanto o processo de desenvolvimento quando afirma que “a criação da documentação é tão importante quanto a criação do software em si”. Isso faz com que o pensamento organizacional seja repensado, dando tanta importância para as tarefas que envolvem o termo de pesquisa. Esse pensamento fica mais claro com a afirmação de Falbo (2004): “a documentação é negligenciada em decorrência de fatores como falta de uma política organizacional que valorize essa tarefa...”. Essa ideia ainda é reforçada por Nunes (2005) que deixa claro que a documentação, ou falta dela afeta organizações de tecnologia da informação pois “a tem deixado de lado, seja por falta de uma política organizacional, por prazos curtos ou inexistência de uma ferramenta específica para essa atividade.”

A documentação tem uma característica importantíssima que é o acompanhamento da mesma por todo o clico de vida do software como novamente afirma Falbo (2004) e confirma Murta (1999): “A documentação dentro do desenvolvimento de software não faz parte de uma fase definida, mas ocorre durante toda a sua existência, em paralelo com todas as fases do ciclo de vida.

1.11 Ferramentas para documentação de software

Processos de desenvolvimento podem tomar grandes dimensões, dessa forma são criadas técnicas e ferramentas, muitas vezes embutidas nas

linguagens de modelagem e/ou de programação. Conforme apresentado por Nunes (2005) “devido à complexidade dos sistemas atuais e à contínua necessidade de armazenamento e utilização do conhecimento nas organizações, uma boa documentação tem se mostrado essencial”.

Como forma de facilitar esse processo utiliza-se de técnicas e ferramentas de documentação, porém, esse processo acaba por ser negligenciado como já apresentado por Falbo (2004). Ferramentas também facilitam a atualização da documentação uma vez que as geram de forma automática. Segundo Anquetil (2004) a documentação pode apresentar vários problemas como “documentação desatualizada e de baixa qualidade, processos de documentação dispendiosos e caros, documentação em abundância e sem propósito, dificuldade de acesso, entre outros”. Dessa forma vários estudos são feitos sobre documentação, afim de unir os melhores métodos e ferramentaas possíveis para realizar a mesma.

Alguns estudiosos apontam o uso de hipertexto como uma boa técnica de documentação, fazendo com que o nível de profundidade da documentação seja avançado conforme a necessidade de quem a lê, como afirma, também, Murta (1999) ao compara o hipertexto com a forma linear: “o texto linear força ao leitor um nível de abstração pré-determinado, não dando a ele a opção de, em um determinado momento, saber mais sobre um determinado tema”. O mesmo autor ainda cita que a troca de hipertexto por textos lineares causa a disjunção da documentação, causando textos monolíticos.

Outros estudiosos da engenharia de software, como como Tilley e Müller (1991) apontam para o uso de documentação, comentários de código e trechos de código como uma forma mais eficaz para documentação, ainda utilizando-se do hipertexto.

Nesse momento é importante desatacar que Murta (1999) expressa a importancia de hipertextos quando afirma que “O pensamento humano não trabalha de forma seqüencial”, novamente fazendo uma comparação ao texto lineares, declarando sua ineficiência: “todo mecanismo relacionado à manipulação de informações deve ser o mais próximo possível do modo que trabalha o pensamento.”

Anquetil (2004) também ressalta a existência de uma norma de uma orgão regulador para a documentação de software: “A norma ANSI/ANS 10.3-1995 recomendou a divisão da documentação em quatro categorias: resumo, informação da aplicação, definição das funcionalidades e informação do programa”.

As ferramentas para documentação de software auxiliam ao programador realizar essa tarefa de grande importância. Segundo Falbo (2004) a ”falta de ferramentas para apoiar a documentação faz com que desenvolvedores optem por deixar a documentação em segundo plano”.

.

REFERÊNCIAS

SOMMERVILLE, Ian. **Engenharia de software**. 9. ed. São Paulo: Pearson Education, 2011. 544 p.

MACHADO, Marcos; MEDINA, Sérgio Gustavo. **SCRUM – Método Ágil**: uma mudança cultural na Gestão de Projetos de Desenvolvimento de Software. Intraciência, [s.l.], p.58-71, jul. 2009. Disponível em: <http://www.faculdadedoguaruja.edu.br/revista/downloads/edicao12009/Artigo_5_Prof_Marcos.pdf>. Acesso em: 9 mar. 2015.

ZANATTA, Alexandre Lazaretti; VILAIN, Patrícia. **Uma análise do método ágil Scrum conforme abordagem nas áreas de processo Gerenciamento e Desenvolvimento de Requisitos do CMMI**. In: WER. 2005. p. 209-220.

RÊGO, Claudete M. et al. **Qualidade de software: visões de produto e processo de software**. CITS, 1997. Disponível em: <https://xa.yimg.com/kq/groups/21646421/371309618/name/Modelos+de+Qualidade+de+Software.pdf>. Acesso em: 10 mar. 2015.

OLIVEIRA, Ebenezer Silva de. **Uso de Metodologias Ágeis no Desenvolvimento de Software.** 2003. 38 f. Monografia (Especialização) - Curso de EspecializaÇÃo em InformÁtica: Ênfase: Engenharia de Software, Departamento de CiÊncia da ComputaÇÃo, Universidade Federal de Minas Gerais, Belo Horizonte, 2003. Disponível em: <http://www.cpdee.ufmg.br/~renato/TesesEDissertacoesOrientadas/Monografia-EbenezerSilvaOliveira.pdf>. Acesso em: 10 mar. 2015.

SOARES, Michel dos Santos. Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software. **Resi: Revista Eletrônica de Sistemas de Informação,** Conselheiro Lafaiete, v. 3, n. 1, nov. 2004. Quadrimestral. Disponível em: <http://189.16.45.2/ojs/index.php/reinfo/article/download/146/38>. Acesso em: 3 mar. 2015.

BISSI, Wilson. **SCRUM:** Metodologia de desenvolvimento ágil. 2007. 4 f. TCC (Graduação) - Curso de Tecnologia em Análise e Desenvolv Imento de Sistemas, Centro Universitário de Maringá, Maringá, 2007. Disponível em: <http://revista.grupointegrado.br/revista/index.php/campodigital/article/viewFile/312/146>. Acesso em: 11 mar. 2015.

PRESSMAN, Roger S. **Engenharia de software.** McGraw Hill Brasil, 2011.

COSTAL, Graziele Cristina Silveira Zerbini; TURRIONI, João Batista and MARTINS, Roberto Antonio. **Adaptação de um wiki para a informatização da documentação do sistema de gestão da qualidade**. *Gest. Prod.* [online]. 2013, vol.20, n.4, pp. 963-978. ISSN 0104-530X.

LOBO, Edson JR. **Guia prático de engenharia de software**. Universo dos Livros Editora, 2009.

SOUZA, Sérgio Cozzetti Bertoldi, et al. **Documentação Essencial para Manutenção de Software II.** IV Workshop de Manutenção de Software Moderna (WMSWM), Porto de Galinhas, PE. 2007.

FALBO, Ricardo A. NUNES, Vanessa B. SOARES, Andrea O. **Apoio à Documentação em um Ambiente de Desenvolvimento de Software.** VII Workshop Iberoamericano de Ingeniería de Requisitos y Desarrollo de Ambientes de Software, IDEAS. 2004.

NUNES, Vanessa Battestin. **Integrando gerência de configuração de software, documentação e gerência de conhecimento em um ambiente de desenvolvimento de software.** Universidade Federal do Espírito Santo (2005).

AGUIAR, Ademar Manuel Teixeira de. **Framework documentation: A minimalist approach.** (2012).

MURTA, Leonardo Gresta Paulino. **FRAMEDOC:** Um FrameWork para a Documentação de componentes Reutilizáveis. Diss. Universidade Federal do Rio de Janeiro, 1999.

SANCHES, R. **Documentação de software e Qualidade de Software:** Teoria e Prática, Prentice Hall, São Paulo (2001): 54-59.

TILLEY, Scott. MULLER, Hausi. **Info: a simple document annotation facility.** Proceedings of the 9th annual international conference on Systems documentation. ACM, 1991.

REIS, Christian Robottom. FORTES, Renata Pontin de Mattos. **Caracterizaç ao de um Processo de Software para Projetos de Software Livre**. Diss. PhD thesis, University of Sao Paulo, Brazil, 2003.