A DOCUMENTAÇÃO COMO PILAR DE QUALIDADE NO DESENVOLVIMENTO DE SOFTWARE EDUCACIONAL

DOCUMENTATION AS A PILLAR OF QUALITY IN THE DEVELOPMENT OF EDUCATIONAL SOFTWARE

LA DOCUMENTACIÓN COMO PILAR DE CALIDAD EN EL DESARROLLO DE SOFTWARE EDUCATIVO

Fabiana Silva Botta Demizu[1]
Verônica Graciela Rapcinski[2]

Resumo. Este artigo apresenta a importância da documentação de software, ferramentas que podem ser utilizadas para os registros de requisitos e métodos que irão agilizar na finalização dos projetos para que não seja ultrapassado o prazo de entrega.

Palavras Chaves: Documentação, Modelagem, Metodologia Ágil, Mapeamento de Processos

Abstract. This article presents the software documentation, the tools can be used for the requirements records and the methods that will expedite the completion of the projects so that the deadline is not exceeded.

Keywords: Documentation, Modeling, Agile Methodology, Process Mapping.

Resumen: En este artículo se presenta la importancia de la documentación de software, las herramientas que se pueden utilizar para registrar requisitos y los métodos que agilizarán la finalización de los proyectos para no exceder el plazo de entrega.

Palabras clave: Documentación, Modelado, Metodología Ágil, Mapeo de Procesos

1 . Introdução

A engenharia de software é um ramo da engenharia cujo foco é o desenvolvimento dentro de custos adequados de sistemas de software de alta qualidade. O Software é abstrato e intangível. Não é limitado por materiais ou controlado por leis da física ou por processos de manufatura. De alguma maneira, isso simplifica a engenharia de software, pois não existem limitações físicas no potencial de software. Contudo, a falta de restrições naturais significa

[1]Fabiana Silva Botta Demizu: Licenciada em Pedagogia; Licenciada em Ciências Biológicas; Doutora em Educação para a Ciência e a Matemática; Mestre em Ensino com ênfase em Educação Interdisciplinar. E-mail: fabiana.demizu@fatecie.edu.br
[2]Verônica Graciela Rapcinski: Bacharel em Nutrição; Especialista em Nutrição Clínica e Ambulatorial. E-mail: veronica.rapicinski@fatecie.edu.br

que o software pode facilmente se tornar extremamente complexo e, portanto, muito difícil de ser compreendido. (SOMMERVILLE, 2007).

O Software* permeia o nosso mundo e, algumas vezes, reconhecemos seu papel em tornar nossa vida mais confortável, eficiente e efetiva. Por exemplo, considere uma das tarefas envolvidas na preparação do café da manhã. O código inserido na torradeira controla o grau em que o pão estará torrado e quando o produto final sairá da torradeira. Programas regulam o fornecimento de eletricidade para as residências e outro software nos cobra pelo uso da energia. Na verdade, podemos utilizar programas automáticos para pagar a conta de eletricidade, para solicitar mantimentos e até mesmo para comprar uma nova torradeira! Hoje em dia, o software está presente, explicitamente ou mesmo sem se fazer notar, em todos os aspectos de nossa vida, inclusive nos sistemas críticos que afetam a nossa saúde e o nosso bem-estar. Por essa razão, a engenharia de software é mais importante do que nunca. As boas práticas de engenharia de software devem assegurar que o software tenha uma contribuição positiva em nossas vidas. (PFLEEGER, 2004).

Metodologia Ágil também é uma ferramenta incrível para guardar como documentação, pois para se tornar Ágil tudo devemos registrar e seguir começo meio e fim, e FOGGET apresenta em seu livro o KANBAN, “é uma técnica visual para demonstrar o fluxo que precisa ser seguido para geração de valor. No quadro de Kanban, Valor e fluxo passam a ser elementos concretos e visíveis na rotina de trabalho da equipe”. (FOGGET, 2014).

2. O que é documentação

O documento de requisitos de software (algumas vezes chamado de especificação de requisitos de software ou SRC - Software Requirements Specification) é a declaração oficial do que os desenvolvedores de sistema devem implementar. Deve incluir os requisitos de usuários de um sistema e uma especificação detalhada dos requisitos de sistema. Se houver um grande número de requisitos, os requisitos detalhados de sistema podem ser apresentados em um documento separado (SOMMERVILLE, 2007).

Para FOGGET, “os requisitos de um sistema servem para podermos visualizar o conjunto de funcionalidades que precisa ser desenvolvido para que o software atenda às necessidades do cliente/solicitante” (FOGGETI, 2014).

Segue uma imagem mostrando como é o requisito do usuário e de sistema, cada um tem um tipo de linguagem de levantamento de requisitos diferenciada, como apresenta a imagem de SOMMERVILLE.

Definição de requisitos de usuário

1. O MHC-PMS deve gerar relatórios gerenciais mensais que mostrem o custo dos medicamentos prescritos por cada clínica durante aquele mês.

Especificações de requisitos de sistema

1.1 No último dia útil de cada mês deve ser gerado um resumo dos medicamentos prescritos, seus custos e as prescrições de cada clínica.
1.2 Após as 17:30h do último dia útil do mês, o sistema deve gerar automaticamente o relatório para impressão.
1.3 Um relatório será criado para cada clínica, listando os nomes dos medicamentos, o número total de prescrições, o número de doses prescritas e o custo total dos medicamentos prescritos.
1.4 Se os medicamentos estão disponíveis em diferentes unidades de dosagem (por exemplo, 10 mg, 20 mg), devem ser criados relatórios separados para cada unidade.
1.5 O acesso aos relatórios de custos deve ser restrito a usuários autorizados por uma lista de controle de geranciamento de acesso.

Figura 1. Requisitos de usuários e de sistema

SOMMERVILLE fala, que cada usuário tem uma necessidade diferenciada um do outro e com isso cada requisito é montado de forma diferenciada, "desde a gerência sênior da organização, que paga pelo sistema, até os engenheiros responsáveis pelo desenvolvimento do software. A Imagem 2, extraída de meu livro com Gerald Kotonya sobre engenharia de requisitos (Kotonya e Sommerville, 1998), ilustra os possíveis usuários do documento e como eles o utilizam" (SOMMERVILLE, 2007).

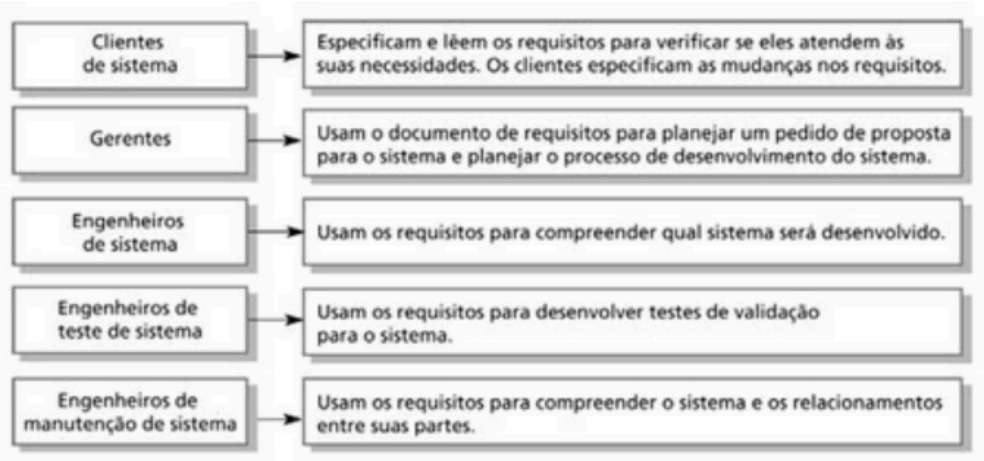


Figura 2. Usuários de um documento de requisitos

A ação da modelagem de requisitos resulta em um ou mais dos seguintes tipos de modelos (PRESSMAN, 2011):

- Modelos baseados em cenários de requisitos do ponto de vista de vários "atores" do sistema.
- Modelos de dados que representam o domínio de informações para o problema.

- Modelos orientados a classes que representam classes orientadas a objetos (atributos e operações) e a maneira por meio da qual as classes colaboram para atender aos requisitos do sistema.
- Modelos orientados a fluxos que representam os elementos funcionais do sistema e como eles transformam os dados à medida que percorrem o sistema.
- Modelos comportamentais que representam como o software se comporta em consequência de "eventos" externos.

FOGGET fala que, "documentos são essenciais para diminuir conflitos e retrabalho no desenvolvimento do software"(FOGGET, 2014).

3. Tipos de documentações

Os requisitos de sistema de software são, frequentemente, classificados em requisitos funcionais, requisitos não funcionais ou requisitos de domínio (SOMMERVILLE, 2007):

1. Requisitos funcionais. São as declarações de serviços que o sistema deve fornecer, como o sistema deve reagir a entradas específicas e como o sistema deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais podem também estabelecer explicitamente o que o sistema não deve fazer.
2. Requisitos não funcionais. São restrições sobre os serviços ou as funções oferecidos pelo sistema. Eles incluem restrições de timing, restrições sobre o processo de desenvolvimento de padrões. Os requisitos não funcionais aplicam-se, frequentemente, ao sistema como um todo. Em geral, eles não se aplicam às características ou serviços individuais de sistema.
3. Requisitos de domínio. São requisitos provenientes do domínio da aplicação do sistema e que refletem as características e as restrições desse domínio. Podem ser requisitos funcionais ou não funcionais.

A Imagem 3 apresenta uma classificação de requisitos não funcionais. Você pode observar, no diagrama, que os requisitos não funcionais podem vir de características necessárias do software (requisitos de produto), da organização que desenvolve o software (requisitos organizacionais) ou de fontes externas.

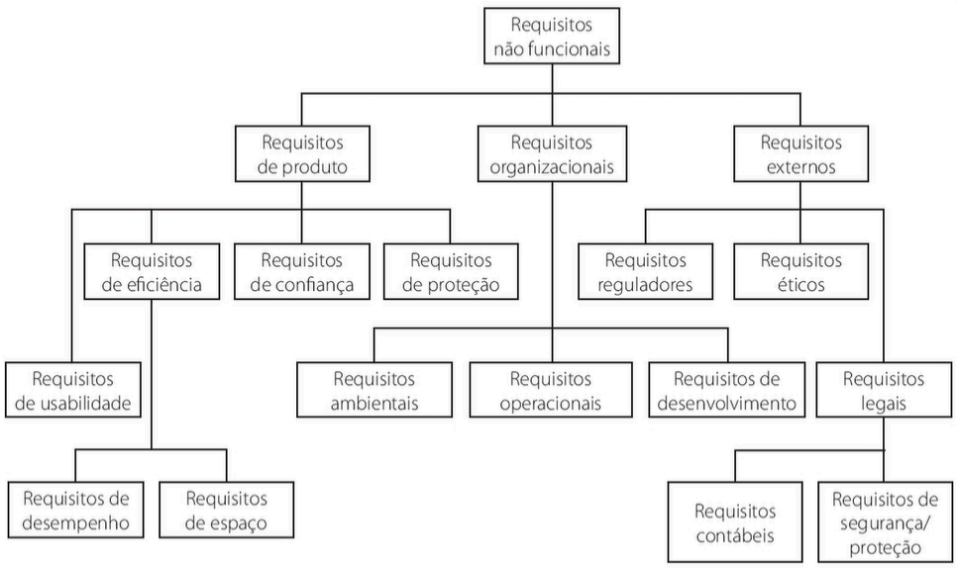


Figura 3. Tipos de requisitos não funcionais

A disponibilização da função de qualidade (quality function deployment, QFD) é uma técnica de gestão da qualidade que traduz as necessidades do cliente para requisitos técnicos do software. O QFD "concentra-se em maximizar a satisfação do cliente por meio dos processos de engenharia de software" [Zul92]. Para tanto, enfatiza o entendimento daquilo que é valioso para o cliente e emprega esses valores ao longo do processo de engenharia. O QFD identifica três tipos de necessidades [ZUL92] (PRESSMAN, 2011):

Requisitos Normais. Refletem os objetos e metas estabelecidos para um produto ou sistema durante reuniões com o cliente. Se esses requisitos estiverem presentes, o cliente fica satisfeito. Exemplos de requisitos normais poderiam ser tipos de displays gráficos solicitados, funções de sistema específicas e níveis de desempenho definidos.

Requisitos esperados. Esses requisitos estão implícitos no produto ou sistema e podem ser tão fundamentais que o cliente não os declara explicitamente. Sua ausência será causa de grande insatisfação. Exemplos de requisitos esperados são: facilidade na interação homem-máquina, confiabilidade e correção operacional global e facilidade na instalação do software.

Requisitos fascinantes. Esses recursos vão além da expectativa dos clientes e demonstram ser muito satisfatórios quando presentes. Por exemplo, o software para um novo celular vem com recursos-padrão, mas junto vem um conjunto de capacidades não esperadas (por exemplo, tecla multitoque, correio de voz visual) que deleitam todos os usuários do produto.

A documentação do usuário não é estritamente parte do projeto de interface com o usuário, mas é uma boa prática projetar o apoio da ajuda on-line em conjunto com a documentação em papel. Os manuais de sistemas devem fornecer informações mais detalhadas do que a ajuda on-line e devem ser projetados para serem utilizados por diferentes classes de usuários finais do sistema (SOMMERVILLE, 2003).

A imagem 4, mostra que para satisfazer a essas diferentes classes de usuários e diferentes níveis de especialidade dos usuários, existem, pelo menos, cinco documentos (ou, talvez, capítulos em um único documento), que devem ser entregues com um sistema de software.

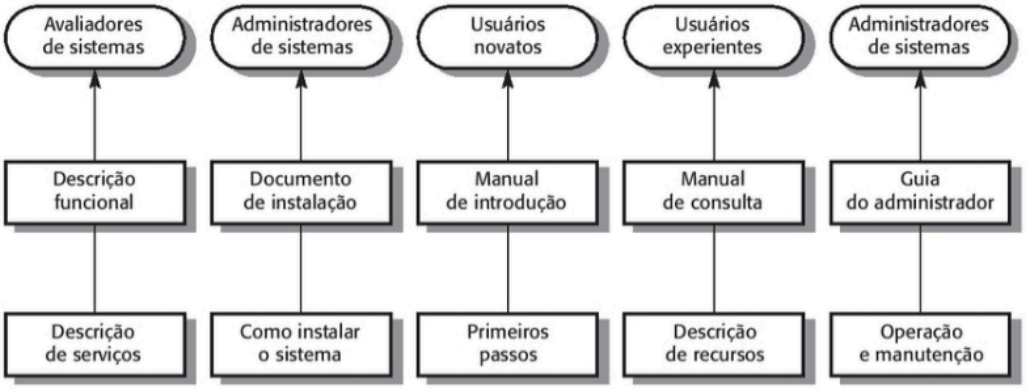


Figura 4. Tipo de documentos produzidos para dar apoio aos usuários

Esses documentos são:

1. Uma descrição funcional, que deve descrever, muito brevemente, os serviços que o sistema fornece. Os Usuários devem poder ler esse documento com um manual de introdução e decidir se o sistema é o que eles necessitam.
2. Um documento de instalação, que deve fornecer detalhes sobre como instalar o sistema. Ele deve descrever os discos nos quais o sistema é fornecido, os arquivos nesses discos e a configuração mínima de hardware exigida. Deve trazer instruções de instalação e detalhes de como configurar arquivos dependentes de configuração.
3. Uma manual de introdução, que deve apresentar uma introdução informal sobre o sistema, descrevendo sua utilização ‘normal’. Ele deve descrever como iniciar o uso do sistema e como os usuários finais podem utilizar os recursos comuns do sistema. Deve ser bem ilustrado com exemplos e incluir informações sobre como fazer a recuperação a partir de erros e reiniciar o trabalho.

4. Um manual de consulta, que deve descrever as facilidades do sistema e sua utilização, fornecer uma lista de mensagens de erro e suas causas possíveis e descobrir como fazer a recuperação a partir dos erros detectados.
5. Um guia do administrador pode ser fornecido em alguns tipos de sistemas. Ele deve descrever as mensagens geradas quando o sistema interage com outros sistemas e explicar como reagir a essas mensagens.

Conforme Barnes, documentação de classe deve incluir pelo menos (BARNES, 2004):

- O nome da classe;
- um comentário que descreva o propósito geral e as características da classe;
- um número de versão;
- o nome do autor (ou autores);
- a documentação para cada construtor e cada método.

A documentação de cada construtor ou método deve incluir:

- o nome do método;
- o tipo de retorno;
- o nome e os tipos de parâmetro;
- uma descrição do propósito e da função do método;
- uma descrição de cada parâmetro;
- uma descrição do valor retornado.

(SOMMERVILLE, 2007) apresenta um modelo de elicitação e análise de requisitos, o modelo segue conforme mostra a imagem 5.

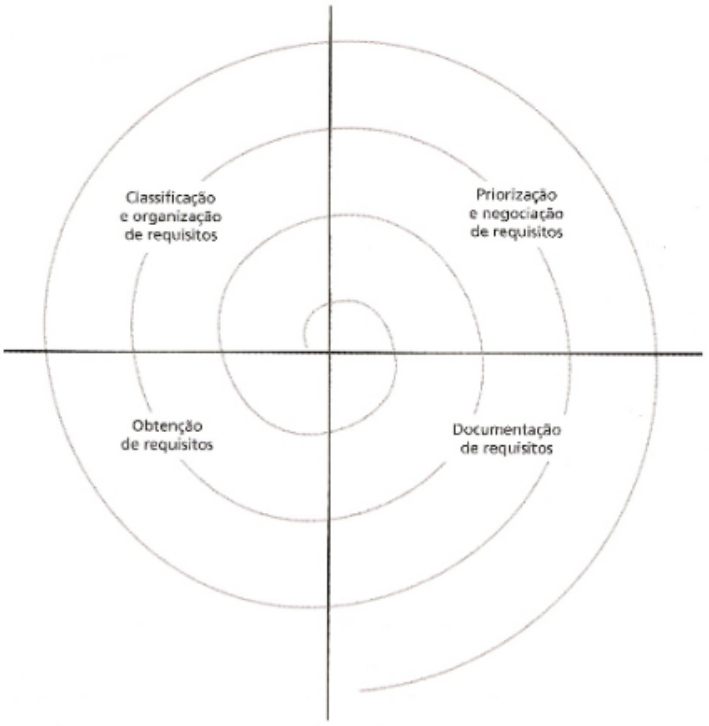


Figura 5. Modelo de elicitação de requisitos

As atividades do modelo proposto são:

1. Obtenção dos requisitos: processo de interação com os stakeholders para a coleta de requisitos.
2. Classificação e obtenção de requisitos: coleta de requisitos não estruturados, agrupando-os e organizando de forma coerente.
3. Priorização e negociação de requisitos: processo de busca de conflitos de requisitos.
4. Documento de requisitos: processo em que os requisitos são documentados.

4. Ferramentas de documentação

Richard fala que a UML é uma linguagem de modelagem para documentar e visualizar os artefatos que especificamos e construímos na análise e desenho de um sistema. Este apêndice propicia um amplo entendimento acerca do que é, e do que não é, a UML (RICHARD, 2001).

A UML é uma linguagem que unifica as melhores práticas de engenharia para fins de modelagem de sistemas(RICHARD, 2001).

- É uma linguagem para capturar conhecimento (semântica parcial) e para expressar esse conhecimento (sintaxe).
- Seu propósito é a modelagem de sistemas.

- Ela é utilizada para documentar de maneira visual aqueles modelos.
- Ela é um conjunto de melhores práticas.
- Ela é utilizada para produzir um conjunto de artefatos que podem ser postos no mercado.
- Ela tem suporte de alcance mundial.

Um workbench CASE é um conjunto de ferramentas que são utilizadas em determinada fase do processo de software, como a fase de projeto, implementação ou teste. A vantagem de agrupar as ferramentas CASE em uma workbench é que as ferramentas podem trabalhar em conjunto, para oferecer um apoio mais abrangente. Serviços comuns podem ser implementados e solicitados por todas as outras ferramentas, As ferramentas de workbench podem ser integradas por meio de arquivos compartilhados, de um repositório compartilhado ou de estruturas de dados compartilhadas (SOMMERVILLE, 2003).

A imagem 6 mostra um diagrama de sequência com objetos ao longo do topo do diagrama. As operações estão indicadas por setas rotuladas e a sequência das operações é descendente. Nesse cenário, o usuário da biblioteca acessa o catálogo para verificar se o item solicitado está disponível eletronicamente e, caso esteja, pede a liberação eletrônica deste item. Por razões de direito de propriedade, essa operação precisa ser licenciada, de modo que exista uma transação entre o item e o usuário, em que a licença é aceita. O item a ser liberado é remetido para um objeto servidor de rede, para que os dados sejam compactados, antes de serem remetidos para o usuário da biblioteca.

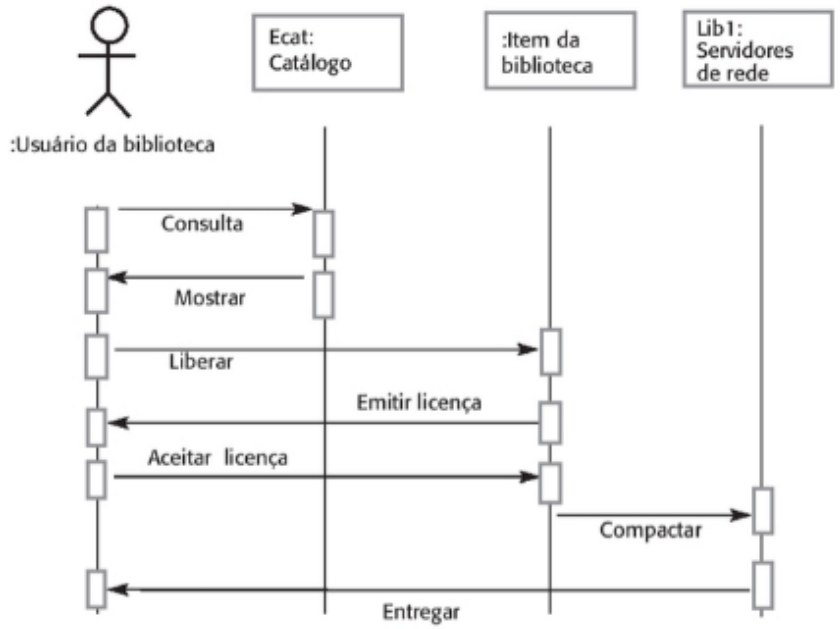


Figura 6. Liberação eletrônica de itens

No exemplo abaixo, temos o diagrama de contexto apresentado pelo Vasques que representa o ambiente no qual o software está inserido e seus principais fluxos de dados, sem abordar o seu processamento interno. Uma vez que se obteve consenso entre as partes interessadas sobre o contexto representado pelo diagrama, ao longo do trabalho de requisitos no projeto é pouco provável que se necessite atualizá-lo (VASQUES, 2016).

O objetivo é que, a partir do diagrama de contexto, possam ser mais facilmente identificadas as partes interessadas para elicitar requisitos e validar os resultados de sua análise, delimitando o escopo com base no trâmite de informações entre o ambiente interno do sistema e externo dos usuários



Figura 7. Um diagrama de contexto estilizado e com menos ênfase nos fluxos de dados específicos com o objetivo de ilustrar os macroprocessos incluídos no escopo.

Outra ferramenta que temos é o caso de uso conforme apresentado na imagem 8, um cenário é uma pequena história que delineia uma sequência esperada de solicitações e respostas entre um usuário (usuários) e o sistema. Ele é utilizado para comunicar como um usuário específico emprega o sistema para atingir um objetivo proveitoso. Escrever um cenário é um simples jogo de “o que acontece a seguir”. A maioria dos cenários é simples; há apenas uma sequência lógica de operações a partir do estado inicial. Outros cenários são mais complicados, com múltiplos casos de exceção (coisas indo mal) ou diferentes caminhos de interações (opções) (RICHARD, 2001).

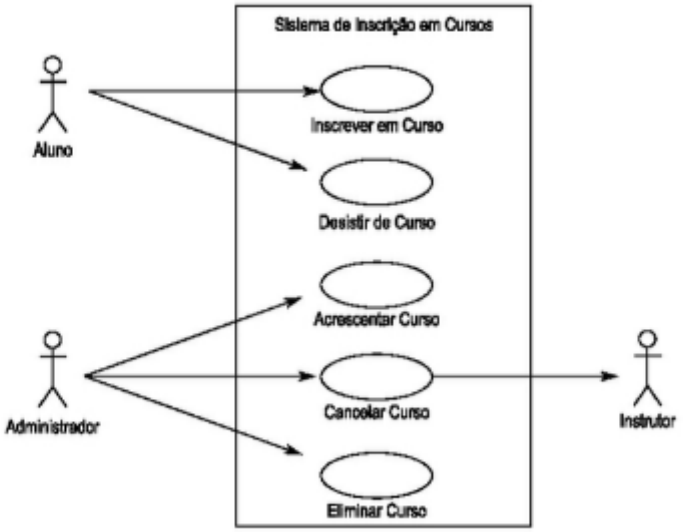


Figura 8. Diagrama simples de caso de uso

Paul e Harvei, fala que o java fornece uma documentação de comentários chamado javadoc, eles são delimitados por //** e */. O compilador ignora todo o texto entre os delimitadores. Os comentários no estilo javadoc permitem-lhe incorporar a documentação do programa diretamente aos seus programas. Esses comentários são o formato de documentação Java preferido na indústria. O programa utilitário javadoc (parte do JDK) lê comentários javadoc e os usa para preparar a documentação do programa no formato HTML (PAUL DEITEL, HARVEY DEITEL, 2017).

“Um ponto forte do JavaDoc é o uso de tags especiais a fim de qualificar melhor a informação contida nos comentários. Com as tags podemos especificar, por exemplo, o autor, a versão, links, data, exceções lançadas, lista de argumentos de um método e tipo de retorno de um método. Essas tags são inseridas dentro do bloco de comentários, antecipadas pelo caractere @ (arroba), e após o nome da própria tag” (DEVMEDIA):

tag	
@author	Expecifica o autor da classe ou do método em questão.
@deprecated	Identifica classes ou métodos obsoletos. É interessante informar nessa tag, quais métodos ou classes podem ser usadas como alternativa ao método obsoleto.
@link	Possibilita a definição de um link para um outro documento local ou remoto através de um URL.
@param	Mostra um parâmetro que será passado a um método.
@return	Mostra qual o tipo de retorno de um método.
@see	Possibilita a definição de referências de classes ou métodos, que podem ser consultadas para melhor compreender a ideia daquilo que está sendo comentado.
@since	Indica desde quando uma classe ou métodos foi adicionado na aplicação.
@throws	Indica os tipos de exceções que podem ser lançadas por um método.
@version	Informa a versão da classe.

Tabela 1. Tags Disponíveis pelo JavaDoc

5. Métodos Ágeis para documentação

Conforme Foggetti (2014) fala, a ferramenta Kanban começou a ser usada com o surgimento dos métodos ágeis. “São utilizados quadros visuais, nos quais os desenvolvedores marcam com cartões físicos as funcionalidades ainda não iniciadas, as que estão em desenvolvimento e as prontas” (FOGGETI, 2014).

Ele fala também que a engenharia de requisitos na metodologia ágil como a fase burocrática que torna os projetos menos ágeis. “Contudo, a falta de documentação é um dos maiores problemas da metodologia”.

Existem dois pressupostos básicos para a utilização do Kanban:

1. O trabalho deve poder ser dividido em versões que tenham conteúdo para o cliente e podem ser produzidas e gerenciadas de maneira independente.
2. Essas versões podem ser desenvolvidas em fluxo contínuo.

Assim Foggeti diz que devemos considerar o seguinte antes de adotar o Kanban (FOGGET, 2014):

- Instalações físicas: Visibilidade é essencial. No espaço do grupo de trabalho deve haver um quadro ou uma parede onde possa ser desenhado o mapa do projeto.
- Volume de trabalho - se o volume de itens de trabalho for muito grande e mudar em questão de minutos, o uso do quadro pode não ser tão útil, pois se perderá muito tempo para atualizá-lo.
- Modelo colaborativo - as pessoas precisam estar motivadas a trabalhar colaborativamente em times.
- Foco em times - Os processos devem ser desenhados pelo próprio time.

Foggeti diz que a falta de documentação é notada somente quando há rotatividade de profissionais na equipe e por isso a documentação histórica é necessária.

A imagem 9 mostra um modelo simples do Kanban. Nesta representação, fica fácil identificar o limite de cartões estabelecidos para cada fase. Este limite está representado pelos números em vermelho no cabeçalho. Os cartões ilustrados pelos retângulos representam uma breve história dos usuários, ou seja, as demandas. As imagens com formato de rosto representam os responsáveis pelos trabalhos em andamento. Portanto, este exemplo aplica as três etapas cruciais para obter os benefícios alcançados com o sistema Kanban.

2024 - Ano - Volume II - Número I ISSN: 2966-0378

Revista Interdisciplinar de Engenharia Civil,

Arquitetura e Urbanismo

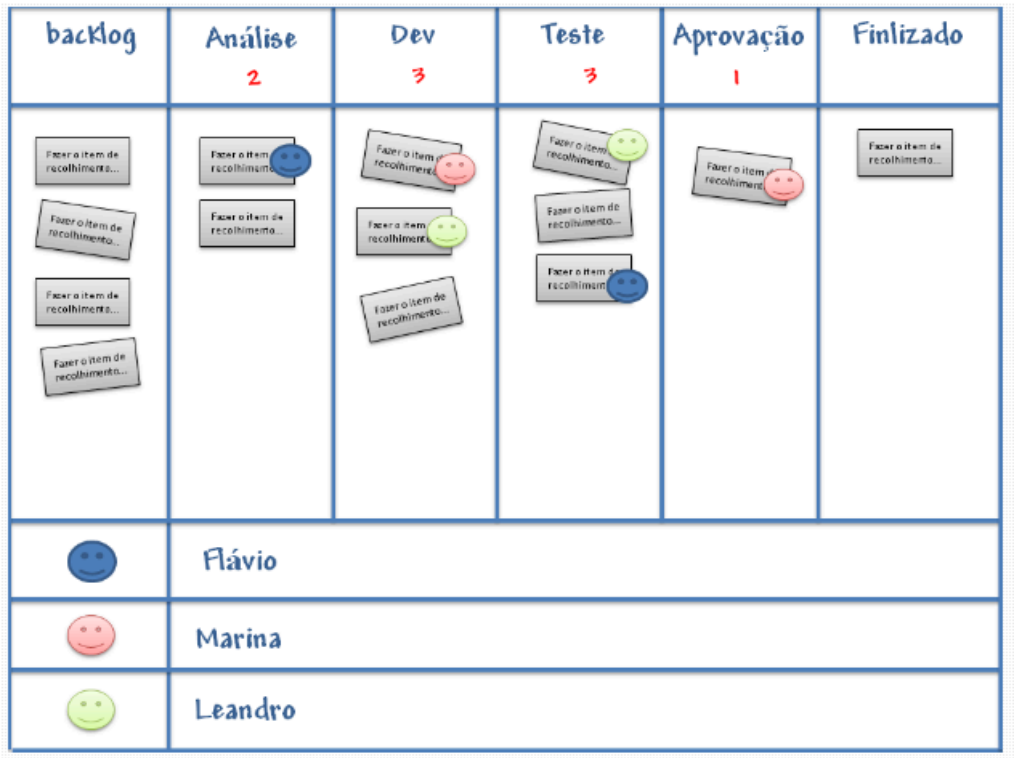


Figura 9. Modelo simples de sinalizador visual Kanban

Conforme palavras de FOGGET existe a metodologia FDD que foi criada em 1999 por Jeff de Luca, Peter Coad e Stephen Palmer,durante um desenvolvimento de projeto em Cingapura. Foi definido um processo ágil com as características:

- Iteração;
- qualidade;
- resultados tangíveis e frequentes;
- possui relatórios de progresso;
- Apreciado por clientes, gerentes e desenvolvedores.

6. Ferramentas mais utilizadas

O Bizagi é um importante software para mapeamento de processos e também para automação destes. Permite ao usuário mapear, documentar e também compartilhar seus processos através de desenhos facilmente interpretados. Segue exemplo:

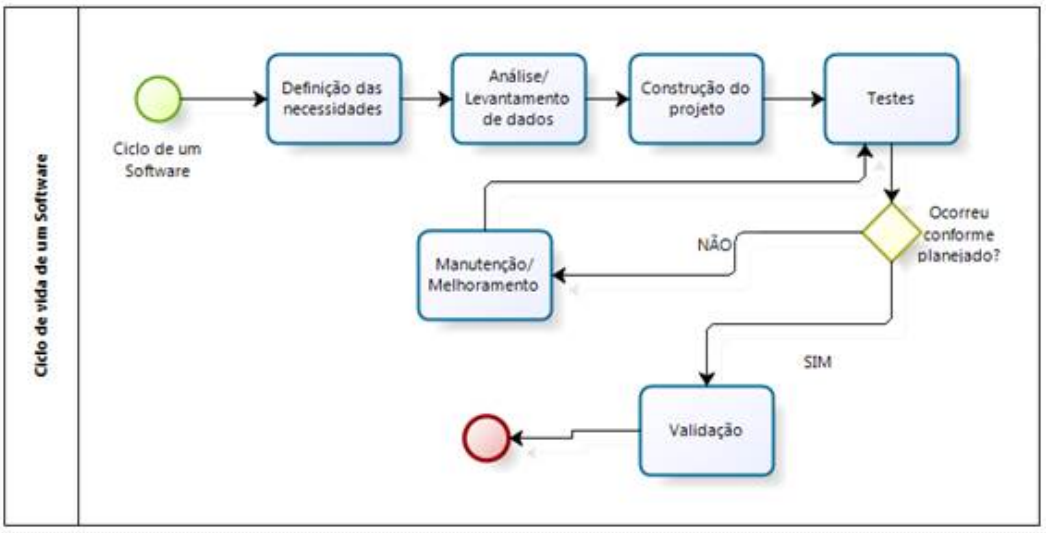


Figura 10. Fluxograma com o ciclo de vida de um software

O redmine é possível dar suporte a vários projetos simultaneamente, ter um maior controle de acesso, configuração de controle por problemas, assim como receber notificações por e-mail. Os módulos do Redmine são:

- **Visão geral:** Este módulo apresenta uma visão geral do projeto, contando com um campo que pode ser editado informando para que serve o projeto. Exibe também um resumo das tarefas e quem são os membros participantes;
- **Atividades:** Relata quais as últimas atividades feitas no projeto; sendo assim, os envolvidos podem acompanhar cada alteração realizada;
- **Planejamento:** Módulo para controle dos sprints, ou seja, as versões dos softwares. Nele, há uma linha informando o percentual de conclusão da versão. Nesta aba, há a possibilidade de inclusão da quantidade de dias remanescentes para a conclusão do projeto, ou até de tarefas relacionadas a ele. É possível atribuir um repositório para cada projeto, já que os mesmo são separados em árvores de diretórios;
- **Tarefa:** Listagens de tarefas cadastradas. O módulo de planejamento exibirá informações sobre tarefas associadas à determinada versão;
- **Nova tarefa:** Essa subdivisão de tarefa foi desenvolvida justamente pensando na agilidade do processo. Ao ser cadastrada em uma nova tarefa, automaticamente ela irá para o módulo de Tarefa. O tipo de tarefa a ser criada é divido entre Defeito, Funcionalidade, Suporte e Administrativa;
- **Gantt:** O diagrama de Gantt é um recurso gráfico utilizado para visualizar a passagem de tempos ao longo das etapas de um projeto;

- **Calendário:** Exibição das datas das versões, assim como cronograma para seu término;
- **Notícias:** As notícias adicionadas podem ser acompanhadas via RsS;
- **Documentos:** O cadastro de documentos pode ser necessário se, por exemplo, tratar-se de um software com um termo de criação mais formal. Há ainda a opção neste módulo para a inclusão do manual do usuário do software, na entrega final;
- **Wiki:** Pode-se inserir páginas relativas a requisitos, mudanças, entre outros aspectos do software. Formulários podem ser criados dentro da própria aplicação, não necessitando de documentos paralelos para tal finalidade;
- **Arquivos:** Pode-se inserir qualquer tipo de arquivo desejado, que estará disponível sempre que necessário;
- **Configurações:** Configurações básicas do projeto como título, descrição, módulos a serem utilizados, pessoas envolvidas, entre outras.

7. Conclusão

Nesta pesquisa conclui-se a necessidade que tem em documentar os projetos desenvolvidos, pois caso haja necessidade de efetuar manutenções ou novas implementações, será possível outros programadores, que ficaram fora destes projetos, analisar o código com cautela, observando frequentemente as documentações, sem risco de efetuar algo errado, colocando código fora do padrão no ato das manutenções, ou no momento de desenvolver novas funcionalidades.

A documentação apresentada neste contexto, conforme os autores dizem, são registros levantados pelos analistas de sistema, que buscam diretamente com os usuários do sistema, o que especificamente eles precisam neste novo software, que atenda as expectativas no momento da utilização.

No artigo também apresenta, as ferramentas utilizadas para documentação, como exemplo, o diagrama de sequência, que conseguimos ver passo a passo do que o usuário faz, no seu hábito de trabalho, diagrama de contexto, que fala o que irá ter no sistema como um todo, o diagrama caso de uso, o que cada setor faz pelo usuário de cada setor, aqui mostra também a documentação JavaDoc, que é utilizada frequentemente em sistemas JAVA.

Percebe-se que a Metodologia Ágil também é importantíssima para registro de desenvolvimento, e ajuda muito em visualizar os itens mais críticos e pouco críticos no

momento da implementação dos softwares, por exemplo, o kanban, quem costuma alimentar esta ferramenta, é o analista de sistemas, que levanta todos os requisitos, e coloca em uma coluna para fazer, então é preparada uma reunião e nesta conversa é para saber quem irá ficar responsável por cada tarefa, e depois deste momento, começa as implementações dos programadores, e assim que vai finalizando cada atividade, coloca como finalizado, e todos os envolvidos consegue ver que o desenvolvimento por respectivo programador foi finalizado, e ele já irá começar a outra atividade que está no cronograma do mesmo, trabalhando desta forma percebe-se a agilidade da utilização deste método.

Atualmente conforme mostrado no artigo, foi demonstrado as Ferramentas mais utilizadas para documentações, uma delas é o Bizagi que mapeia os processos de todo o sistema, por representação de um desenho facilmente interpretado.

O Redmine é outra ferramenta que contribui principalmente na metodologia ágil de trabalho, o desenvolvedor vai utilizando e registrando todos os passos executados no desenvolvimento, e assim que vai finalizando cada tarefa vai colocando como finalizado, e o gerenciamento dele é excelente, pois vai mostrando os gráficos de como está o trabalho de cada usuário, é uma ferramenta muito interessante também.

REFERÊNCIAS

SOMMERVILLE, I. **Engenharia de Software**. São Paulo, SP, Brasil, 2007.

PEFLEEGER, S. L. **Engenharia de Software Teoria e Prática**. São Paulo, SP, Brasil, 2004.

VAZQUEZ, C. E.; GUILHERME, S. S. **Engenharia de Requisitos**. Rio de Janeiro, RJ, Brasil, 2016.

RICHARD, L. **Guia Prático de Desenvolvimento Orientado a Objeto**. São Paulo, SP, Brasil, 2001.

FOGGETI, C. **Gestão ágil de projetos**. São Paulo, SP, Brasil, 2014.

DEITEL, P.; DEITEL, H. **Java: como programar**. São Paulo, SP, Brasil, 2017.

BARNES, D. J.; KOLLING, M. **Programação Orientada a objetos com java - uma introdução prática usando BLUEJ.** São Paulo, SP, Brasil, 2004.

DEVMEDIA. JavaDoc - **Implementando documentação através do Netbeans**: Disponível em: <https://www.devmedia.com.br/javadoc-implementando-documentacao-atraves-do-netbeans/2495>. Acesso em: Outubro. 2018.

DEVMEDIA. **Kanban: o ágil adaptativo** - Revista Engenharia de Software Magazine 45: Disponível em:<https://www.devmedia.com.br/kanban-o-agil-adaptativo-revista-engenharia-de-software-magazine-45/23560>. Acesso em: Outubro. 2018.

DEVMEDIA. **Entendendo os requisitos de software** - Revista .Net Magazine 97: Disponível em:<https://www.devmedia.com.br/entendendo-os-requisitos-de-software-revista-net-magazine-97/24715>. Acesso em: Outubro. 2018.

DEVMEDIA. **Ferramenta CASE: Conhecendo algumas boas opções:** Disponível em:<https://www.devmedia.com.br/ferramentas-case-conhecendo-algumas-boas-opcoes/32034>. Acesso em: Outubro. 2018.