PROCESSOS DE SOFTWARE

Professora Esp. Janaina Aparecida de Freitas

GRADUAÇÃO
Unicesumar

UNICESUMAR
Av. Guedner, 1610 - Jardim Aclimação
Cep 87050-900 - MARINGÁ - PARANÁ
unicesumar.edu.br
44 3027.6360

UNICESUMAR EDUCAÇÃO A DISTÂNCIA
NEAD - Núcleo de Educação a Distância
Bloco 4 - MARINGÁ - PARANÁ
unicesumar.edu.br
0800 600 6360

FICHA CATALOGRÁFICA

C397 **CENTRO UNIVERSITÁRIO DE MARINGÁ.** Núcleo de Educação a Distância; **FREITAS**, Janaina Aparecida de.

Processos de software. Janaina Aparecida de Freitas. Florianópolis, SC: Arqué, 2025.

181 p.

"Graduação - EaD".

1. processos 2. software 3. gerenciamento 4. EaD. I. Título.

ISBN 978-65-279-1326-9

CDD - 22 ed. 620
CIP - NBR 12899 - AACR/2

Ficha catalográfica elaborada pelo bibliotecário
João Vivaldo de Souza - CRB-8 - 6828

Impresso por:

Professor
Wilson de Matos Silva
Reitor

Viver e trabalhar em uma sociedade global é um grande desafio para todos os cidadãos. A busca por tecnologia, informação, conhecimento de qualidade, novas habilidades para liderança e solução de problemas com eficiência tornou-se uma questão de sobrevivência no mundo do trabalho.

Cada um de nós tem uma grande responsabilidade: as escolhas que fizermos por nós e pelos nossos farão grande diferença no futuro.

Com essa visão, o Centro Universitário Cesumar assume o compromisso de democratizar o conhecimento por meio de alta tecnologia e contribuir para o futuro dos brasileiros.

No cumprimento de sua missão – "promover a educação de qualidade nas diferentes áreas do conhecimento, formando profissionais cidadãos que contribuam para o desenvolvimento de uma sociedade justa e solidária" –, o Centro Universitário Cesumar busca a integração do ensino-pesquisa-extensão com as demandas institucionais e sociais; a realização de uma prática acadêmica que contribua para o desenvolvimento da consciência social e política e, por fim, a democratização do conhecimento acadêmico com a articulação e a integração com a sociedade.

Diante disso, o Centro Universitário Cesumar almeja ser reconhecido como uma instituição universitária de referência regional e nacional pela qualidade e compromisso do corpo docente; aquisição de competências institucionais para o desenvolvimento de linhas de pesquisa; consolidação da extensão universitária; qualidade da oferta dos ensinos presencial e a distância; bem-estar e satisfação da comunidade interna; qualidade da gestão acadêmica e administrativa; compromisso social de inclusão; processos de cooperação e parceria com o mundo do trabalho, como também pelo compromisso e relacionamento permanente com os egressos, incentivando a educação continuada.

Professor
Fabrício Lazilha
Diretoria de
Planejamento de Ensino

Professora
Kátia Solange Coelho
Diretoria Operacional
de Ensino

Seja bem-vindo(a), caro(a) acadêmico(a)! Você está iniciando um processo de transformação, pois quando investimos em nossa formação, seja ela pessoal ou profissional, nos transformamos e, consequentemente, transformamos também a sociedade na qual estamos inseridos. De que forma o fazemos? Criando oportunidades e/ou estabelecendo mudanças capazes de alcançar um nível de desenvolvimento compatível com os desafios que surgem no mundo contemporâneo.

O Centro Universitário Cesumar mediante o Núcleo de Educação a Distância, o(a) acompanhará durante todo este processo, pois conforme Freire (1996): "Os homens se educam juntos, na transformação do mundo".

Os materiais produzidos oferecem linguagem dialógica e encontram-se integrados à proposta pedagógica, contribuindo no processo educacional, complementando sua formação profissional, desenvolvendo competências e habilidades, e aplicando conceitos teóricos em situação de realidade, de maneira a inseri-lo no mercado de trabalho. Ou seja, estes materiais têm como principal objetivo "provocar uma aproximação entre você e o conteúdo", desta forma possibilita o desenvolvimento da autonomia em busca dos conhecimentos necessários para a sua formação pessoal e profissional.

Portanto, nossa distância nesse processo de crescimento e construção do conhecimento deve ser apenas geográfica. Utilize os diversos recursos pedagógicos que o Centro Universitário Cesumar lhe possibilita. Ou seja, acesse regularmente o AVA – Ambiente Virtual de Aprendizagem, interaja nos fóruns e enquetes, assista às aulas ao vivo e participe das discussões. Além disso, lembre-se que existe uma equipe de professores e tutores que se encontra disponível para sanar suas dúvidas e auxiliá-lo(a) em seu processo de aprendizagem, possibilitando-lhe trilhar com tranquilidade e segurança sua trajetória acadêmica.

AUTORA

Professora Esp. Janaina Aparecida de Freitas

Possui graduação em Informática pela Universidade Estadual de Maringá (2010). Especialização em MBA em Teste de Software pela Universidade UNICEUMA, Brasil. (2012). Trabalhou na iniciativa privada, na área de Análise de Sistemas e Testes de Software. Têm experiência na área de Engenharia de Software com ênfase em Análise de Requisitos, Gestão de Projetos de Software, Métricas e Estimativas, Qualidade e Teste de Software. Atualmente cursando Mestrado em Ciências da Computação UEM e Licenciatura em Letras - Português/Inglês na UniCesumar. Trabalha como Professora Mediadora no NEAD - Núcleo de Educação a Distância da UniCesumar.

Conferir mais detalhes em: http://lattes.cnpq.br/4906244382612830

SEJA BEM-VINDO(A)!

Caro(a) Aluno(a).

Seja bem-vindo(a) à disciplina de Gerenciamento de Software. Neste curso, iremos abordar alguns conceitos e práticas envolvidas do gerenciamento de projetos e que são usadas nas metodologias ágeis. Gerenciar projetos é importante quando temos que desenvolver um sistema complexo, compartilhar recursos, controlar prazos e custos para que o sistema seja confiável e instável.

A disciplina de gerenciamento de software tem como objetivo a melhoria do desempenho de um projeto como um todo, a partir do uso de metodologias, tanto tradicionais como ágeis. O gerenciamento software é uma atividade de apoio da Engenharia de Software, pois ele inicia antes de qualquer atividade técnica e prossegue ao longo da modelagem, construção e utilização do software.

As unidades do livro foram organizadas de forma a estarem vinculadas, ou seja, que a unidade seguinte sempre está vinculada com a unidade anterior, portanto, é bom que você leia e entenda todo o conteúdo de uma unidade para passar para a próxima.

Vamos iniciar vendo na Unidade I a diferença entre projetos e processos e os conceitos que envolvem cada um. Veremos que os projetos estão relacionados a algo novo, com início e fim definidos, e os processos ocorrem de maneira contínua, enquanto são, normalmente, estabelecidos quando um projeto é concluído.

Seguindo para a Unidade II vamos conhecer conceitos e práticas de algumas metodologias ágeis para o desenvolvimento de software e como elas são utilizadas nas empresas. Algumas metodologias ágeis que serão estudadas: Processo Unificado (UP), Extreme Programming (XP), FDD, DSDM e Iconix Process.

Assim, na Unidade III, vamos conhecer a metodologia ágil Scrum. Será apresentada uma visão geral do Scrum, seus conceitos básicos e como é o seu funcionamento. Veremos porque o Scrum é considerado uma importante metodologia ágil, e como ela foi fortemente influenciada pela indústria japonesa que utilizavam equipes pequenas e multidisciplinares em seus projetos.

Na Unidade IV será apresentada uma visão comparativa e as diferenças entre as metodologias tradicionais e ágeis no gerenciamento de software. Passaremos a entender as características das metodologias tradicionais e das ágeis e como isso irá lhe ajudar na melhor escolha da metodologia para determinado projeto.

E, por fim, na Unidade V, será apresentada uma visão geral de outras metodologias ágeis de desenvolvimento presentes no mercado. As outras metodologias que aprenderemos são: OpenUp (Processo Unificado Aberto), Metodologia Crystal, Test Driven Development (TDD), Lean Software Development (LD) e Kanban.

Assim, nós te convidamos a entrar nessa jornada com empenho, dedicação e muita sede por conhecimento!

Boa leitura!

SUMÁRIO

UNIDADE I

INTRODUÇÃO AO GERENCIAMENTO DE SOFTWARE

UNIDADE II

INTRODUÇÃO A MODELOS DE DESENVOLVIMENTO DE SOFTWARE

SUMÁRIO

UNIDADE III

INTRODUÇÃO AO GERENCIAMENTO ÁGIL DE SOFTWARE

UNIDADE IV

SUMÁRIO

UNIDADE V

OUTRAS METODOLOGIAS ÁGEIS

Professora Esp. Janaina Aparecida de Freitas

INTRODUÇÃO AO GERENCIAMENTO DE SOFTWARE

UNIDADE I

Objetivos de Aprendizagem

- Compreender as diferenças entre projetos ligados à área de software e projetos de desenvolvimento de software.
- Desenvolver conceitos básicos sobre o Gerenciamento de Projetos (PMI) e PMBOK (Project Management Body of Knowledge).
- Introduzir o aluno ao Gerenciamento Ágil de Projetos, seus conceitos, práticas e princípios.
- Discorrer sobre alguns Modelos (cronograma, orçamento, diagramas etc.) adotados no Gerenciamento de Software, aspectos da Equipe de GP e Métricas no auxílio do GP.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Conceitos de Gerenciamento de Projeto
- PMI – O Gerenciamento de Projetos
- Agile Project Management (APM)
- Práticas do Gerenciamento de Software

INTRODUÇÃO

Olá, aluno(a)! Esta unidade tem por objetivo apresentar uma introdução ao gerenciamento de projetos e alguns conceitos relacionados, que são usados tanto nas metodologias clássicas quanto ágeis.

Vamos começar entendendo a diferença entre projetos e processos. Projetos estão relacionados a algo novo, com início e fim definidos e os processos ocorrem de maneira contínua, normalmente estabelecidos quando um projeto é concluído.

Ainda vamos também compreender as diferenças entre projetos e projetos de desenvolvimento de software e projeto de software. Um dos pontos principais para a diferença entre eles é a natureza dos projetos. O projeto de manufatura é algo físico, palpável, mensurável e o projeto de software é algo subjetivo, invisível, impalpável e imensurável.

O Gerenciamento de Projetos de Software é uma atividade de apoio da Engenharia de Software, pois ele inicia antes de qualquer atividade técnica e prossegue ao longo da modelagem, construção e utilização do software. Gerenciar projetos é importante quando temos que desenvolver um sistema complexo, compartilhar recursos, controlar prazos e custos para que o sistema seja confiável e instável.

Vamos conhecer também que existem diversas instituições que se preocupam em estudar, formalizar e divulgar boas práticas de Gerenciamento de Projetos, como o PMI e PMBOK *(Project Management Body of Knowledge)*.

Outro ponto que veremos nesta unidade é sobre o Gerenciamento Ágil de Projetos, seus conceitos, práticas e princípios. Vamos estudar que a característica que mais atrai nessa metodologia é a habilidade em se reduzir custos, que podem ocorrer durante o processo de desenvolvimento de software.

Vamos conhecer alguns Modelos (cronograma, orçamento, diagramas) adotados no Gerenciamento de Software, como aspectos da equipe e métricas no auxílio do gerenciamento de projetos.

Preparado(a) para começar? Então, vamos seguir em frente. Boa leitura e bons estudos!

CONCEITOS DE GERENCIAMENTO DE PROJETO

O Gerenciamento de Projetos, conforme Sbrocco (2012, p. 27) "é uma disciplina que vem sendo formada há muito tempo por pessoas de diversas áreas de conhecimento e especializações". É a área onde ocorre a aplicação de conhecimentos, habilidades e técnicas na elaboração de atividades relacionadas para atingir um conjunto de objetivos pré-definidos.

Para entender melhor, vamos analisar alguns conceitos relacionados ao Gerenciamento de Projetos.

CONCEITO DE PROJETO

Genericamente, conforme Valle et al (2010), projeto significa "empreendedorismo", que propõe a criação de um produto ou a execução de um serviço, que seja temporário e não repetitivo. E como qualquer empreendimento, este precisa ser bem planejado, programado e sua execução controlada.

Um projeto é um empreendimento com características próprias, tendo princípio e fim, conduzido por pessoas, para atingir metas estabelecidas dentro de parâmetros de prazo, custo e qualidade. Segundo Valle et al (2010), podemos definir um projeto como:

> [...] um projeto é formado por um esforço não permanente, ou seja, temporário, para a criação de um produto ou serviço. Como não é permanente, podemos afirmar que todos os projetos deveriam conter um início, um desenvolvimento e um fim bem definidos. O projeto é finalizado quando seus objetivos são alcançados, quando não for mais necessário ou quando ficar bem claro que seus objetivos não poderão ser atingidos ou não é compensador ir em frente (Valle et al 2010, pg. 35).

Um projeto é um empreendimento, que para Valle et al (2010), tem um ciclo de vida definido, composto por fases com início, etapas intermediárias e término, em que cada etapa gera recursos para as fases seguintes.

Se pensarmos nos conceitos explorados aqui, podemos dizer que os projetos são ubíquos (presente em todos os lugares ao mesmo tempo), pois estão em todo o lugar e são executados por todos (DINSMORE, 2009).

Para Pressman (2016) a principal razão de conduzirmos projetos planejados e com controle é administrar a complexidade. Pois para gerenciar um projeto de software que seja bem sucedido é necessário saber o que pode dar errado e com isso evitar problemas que possam vir a ocorrer. Os propósitos de um Projeto de Software:

- Transformar os requisitos em um projeto que permitirá construir o sistema.
- Nivelar os artefatos existentes com o ambiente em desenvolvimento.
- Identificar os componentes de software e seus relacionamentos.

E quando pensamos em desenvolvimento de software, pensamos em tarefas complexas, por envolver várias equipes que trabalham por um longo período de tempo e, por isso, os projetos de software precisam ser bem gerenciados.

PROCESSOS X PROJETOS

Para Sbrocco (2012) projetos são diferentes de processos. Para ele, projetos estão relacionados a algo novo, com início e fim definidos e os processos ocorrem de maneira contínua. Os processos normalmente são estabelecidos quando um projeto é concluído. Já os projetos produzem um produto único e que é diferente de outros existentes, em pelo menos uma característica. Apesar das diferenças, podemos perceber que os processos e os projetos são limitados em escopo e recursos que podem ser empregados. (SBROCCO, 2012).

O processo, ainda segundo Sbrocco (2012, p. 29), "passa por fases que vão desde a concepção até a sua conclusão". No caso particular de processos de Gerenciamento de Projetos, utilizamos um conjunto de fases organizadas que

tem por função atingir os objetivos que foram definidos. Essas fases organizadas formam o Ciclo de Vida do projeto que serve para definir o início e fim de um projeto, definindo qual atividade que deve ser realizado em cada fase e quem deve ser envolvido.

DIFERENÇA ENTRE PROJETOS E PROJETOS DE DESENVOLVIMENTO DE SOFTWARE

Apesar das técnicas de Gerenciamento de Projetos poderem ser aplicadas a qualquer tipo de projeto, da engenharia civil ao software, existem algumas diferenças entre projetos que precisamos conhecer.

Qual a diferença entre projetos e projetos de desenvolvimento de software? Vamos imaginar um projeto de construção de um prédio, por exemplo, em que os requisitos são facilmente identificados e é raro quando ocorrem erros que sejam derivados do projeto. Agora, quando imaginamos um projeto de desenvolvimento de software, podemos observar que eles apresentam um desafio distinto, muitas vezes complexo, quando comparado à maioria dos outros tipos de projetos existentes.

Não é normal, nos projetos de manufatura, que as especificações sejam alteradas nas fases de construção ou produção, porque caso isso ocorra, os estouros de orçamento se tornam gigantescos. Em projetos de software, as mudanças nas especificações dos requisitos, seja por parte do cliente ou de regras tributárias por parte do governo, seja a maior preocupação.

Um dos pontos principais para todas as diferenças é a natureza dos projetos. O projeto de manufatura é algo físico, palpável, mensurável e o projeto de software é algo subjetivo, invisível, impalpável e imensurável.

E por ser um produto impalpável, o software impõe algumas dificuldades com relação a alguns aspectos importantes do gerenciamento de projetos. Entre esses aspectos, temos a definição do escopo do projeto, a garantia de qualidade do produto e controle de progresso do que está sendo desenvolvido.

Vamos pensar em um projeto de construção de uma casa. Durante a sua construção, pode-se definir com precisão, se a casa está próxima de ser acabada.

Agora imagine em um sistema de software. Isso se torna mais difícil, pois ele é invisível aos nossos olhos, impalpável e não temos como medir seu progresso e saber quando ele estará pronto para ser utilizado.

Mas você já pensou que os softwares estão em todos os lugares? Que a cada dia se tornam sem fim, e a necessidade do cliente de ter novas funcionalidades incluídas ou de alterações são intermináveis. Pois, para o cliente, qual a dificuldade de incluir um novo campo na tela do sistema? E com isso apertam-se os prazos, implantação do sistema semi-acabado e custo altíssimos.

Um bom caminho, seria antes de iniciar qualquer projeto de software, conhecer a regra de negócios, ferramentas e recursos que poderão ser utilizados. Logo em seguida, desenvolver um bom plano de Gerenciamento de Projeto, informando como serão tratados os requisitos, pontos de gestão e seus relacionamentos, recursos e processos empregados durante o desenvolvimento do software.

DIFERENÇA ENTRE PROJETO PARA A DESENVOLVIMENTO DE SOFTWARE E PROJETO DE SOFTWARE

Além da diferença entre um projeto de manufatura e um projeto de um sistema de software, ainda temos a diferença entre um projeto para o desenvolvimento de software e entre o projeto de software.

O projeto de desenvolvimento de software é algo subjetivo, invisível, impalpável e imensurável e isso impõe algumas dificuldades com relação a alguns aspectos importantes do gerenciamento de projetos.

Para o profissional de software, conforme Fernandes (2011, p. 35) a aplicação dos conceitos significa determinar o tipo de software que queremos, quais os processos de construção e gestão de software, quais as linguagem de programação e quais os sistemas automatizados que serão usados como apoio. Além de como será tratado os defeitos e falhas e qual a infraestrutura, redes necessárias e quais os perfis dos profissionais envolvidos.

E quanto ao projeto de software? Para compreendemos o projeto é necessário que revisemos alguns conceitos do processo de software que já foram estudados.

Conforme Sommerville (2011), um processo de software é considerado um

conjunto de atividades, que pode levar a construção de um software, e embora existam processos diferentes, algumas atividades fundamentais são comuns a todos, como a especificação de software que define a funcionalidade do mesmo e as restrições sobre sua operação devem ser definidas. O Projeto e a implementação de software que definem as funcionalidades para que esse atenda à especificação dada pelo cliente. A validação de software que garante que o sistema faça o que o cliente deseja; a evolução desse que define o que deve evoluir para atender às necessidades mutáveis do cliente.

Assim, após os Requisitos de Software terem sido especificados e modelados é iniciada a primeira atividade, o Projeto de Software, onde é definido como ele será construído, sua arquitetura, interfaces, componentes que poderão ser utilizados e outras características que podem ser determinantes para se gerar um software.

Conforme Pressman (2016, p. 206), é na fase de requisitos que é convertido o "que" é para ser feito e detalhado e na fase de projeto é indicado o "como" deverá ser o desenvolvimento, fornecendo detalhes da arquitetura e os componentes essenciais para a implementação do sistema. O profissional responsável por essa fase é conhecido como o Arquiteto de Software, que possui experiência como programador e possui amplos conhecimentos sobre a Gerência de Projetos.

GERENCIAMENTO DE PROJETOS

O Gerenciamento de projetos envolve o planejamento de processos, de pessoas e eventos que possam ocorrer à medida que o software é desenvolvido. Esse gerenciamento é uma atividade de apoio da Engenharia de Software, pois ele inicia antes de qualquer atividade técnica e prossegue ao longo da modelagem, construção e utilização do software.

Pressman (2016), escreve que os quatro Ps (pessoas, produto, processo e projeto) tem grande influência sobre o Gerenciamento de Projetos de Software:

> [...] as pessoas devem ser organizadas em equipes eficientes, devem ser motivadas a realizar um trabalho de software de alta qualidade e devem ser coordenadas para uma comunicação eficaz. Requisitos de produtos devem ser comunicados do cliente ao desenvolvedor; decomposto em partes e posicionados para a execução pela equipe de software. O pro-

> cesso deve ser adaptado às pessoas e ao problema. Uma metodologia de processo comum é selecionada, um paradigma de engenharia de software é aplicado e um conjunto de tarefas é escolhido para que o trabalho se realize. Por fim, o projeto deve ser organizado de modo a capacitar a equipe de software a ser bem sucedida (PRESSMAN, 2016, p. 700).

De acordo com o PMBOK (Project Management Body of Knowledge) – Quinta Edição, Gerenciamento de Projetos "é a aplicação de conhecimento, habilidades, ferramentas e técnicas às atividades do projeto a fim de atender aos seus requisitos" (PMI, Guia PMBOK, 2013).

Os projetos de software envolvem decisões, tempo, custo, escopo, qualidade e alguns aspectos relacionados às necessidades e expectativas dos clientes e partes interessadas envolvidas.

Na figura a seguir, são apresentados os três pilares básicos que sustentam o projeto: o escopo, custo e o prazo e esses pilares se apoiam pela qualidade.

Figura 1 – Três pilares de um projeto: escopo, custo e prazo
Fonte: adaptado Sbrocco (2012 p. 32).

Usar Gerenciamento de Software é importante, segundo Sbrocco (2012) quando pensamos em desenvolver um produto que seja complexo, quando é necessário ter controle sobre os prazos e custos do produto, ou ainda, quando for necessário compartilhar recursos.

E quais problemas que são considerados típicos que podem aparecer durante o desenvolvimento de um projeto?

- Cronogramas com atrasos.
- Produtos com mal funcionamento.
- Falta de equipe qualificada.
- Muitas mudanças nos requisitos e especificações.
- Baixa qualidade do produto.
- Custos acima do que foi orçado.
- Projetos cancelados.

Os projetos normalmente são divididos em fases para facilitar o seu controle e gerenciamento. Para Sbrocco (2012), o conjunto de fases que compõem um projeto é chamado de Ciclo de Vida do projeto. No projeto, cada fase é marcada pela conclusão de um ou mais produtos, sendo que o Ciclo de Vida do mesmo possui uma sequência lógica de atividades para a sua execução e, a cada mudança de fase, temos uma revisão, que determina se devemos prosseguir para a próxima fase.

As fases do projeto, segundo Sbrocco (2012, p. 33) "geralmente são identificados pelo próprio nome dos itens produzidos e das atividades realizadas, como fase de requisitos, fase de implementação, etc.". O nome dado a cada fase do projeto pode variar conforme a área de aplicação e conforme o modelo de ciclo de vida que foi adotado para o projeto.

Algumas características que devemos pensar quando definimos um Ciclo de Vida de um projeto, (SBROCCO, 2012, p. 33):

- O Ciclo de Vida deve definir o início e o fim do projeto.
- Determinar ações de transição entre as fases do projeto ou entre projetos.
- Definir o trabalho ou esforço técnico de cada fase do projeto.
- Determinar a equipe envolvida em cada fase do projeto.
- Mecanismos de controle e aprovação para cada fase do projeto.

Até os projetos de software que são de pequeno porte e simples, precisam de uma gestão de projeto.

> Um projeto de software de grande porte complexo exigiria algumas sofisticadas habilidades de gestão de projeto e um esforço considerável, além de ferramentas que auxiliassem as tarefas de gestão. A chave é encontrar o equilíbrio entre uma gestão de projetos enxuta e uma gestão de projetos excessiva, evitando que se torne pouco ou demasiadamente exigente (TSUI, 2013, p. 191).

Conforme Sbrocco (2012, p. 30) "existem diversas instituições que se preocupam em estudar, formalizar e divulgar boas práticas de Gerenciamento de Projetos". Nos próximos tópicos, vamos citar algumas delas.

ANOTAÇÕES

PMI – O GERENCIAMENTO DE PROJETOS

Conforme Martins (2007, p. 25), "o PMI é uma entidade internacional sem fins lucrativos que congrega os profissionais de áreas relacionadas a Gerência de Projetos (*Project Management*)". Seu principal objetivo é promover e difundir a gestão de projetos no mundo, ampliando aos profissionais o conhecimento sobre gerenciamento de projetos como disciplina e profissão. O PMI não apresenta regras e sim melhores práticas.

O PMI define um conjunto de procedimentos que ajudam a promover e padronizar as teorias que estão ligadas ao gerenciamento de projetos. E essas teorias padronizadas estão registradas no documento chamado PMBOK *(Project Management Body Of Knowledge)*.

O PMI tem como foco a expansão do conhecimento da profissão de gerenciamento de projetos, e para isso, investe em conferências, bienais, simpósios, subsídios e livros voltados para a pesquisa.

PMBOK *(PROJECT MANAGEMENT BODY OF KNOWLEDGE)*

É um guia, segundo Sbrocco (2012, p. 72), "de conhecimento e de melhorias práticas para a profissão de gerência de projetos". Esse guia reúne as informações comprovadas internacionalmente sobre o conhecimento relacionado a práticas tradicionais, inovadores e mais avançadas de gestão de projetos. Além disso, o PMBOK é um guia de padronização de termos que são utilizados na área de gerência de projetos.

> O PMBOK Guide 5º Edição, é um guia de Gerenciamento de projetos internacionalmente reconhecido, desenvolvido pelo PMI (Project Ma-

> nagement Institute) que fornece os conceitos fundamentais de projetos. O Guia PMBoK possui diversos processos, ferramentas e técnicas úteis para a gerência de qualquer projeto. O guia não determina como será gerenciado um projeto, ele apenas dá boas práticas, deixando livre para o gerente de projeto e a equipe escolherem aquilo que melhor se adapte ao seu projeto. O PMBOK Guide fornece também uma terminologia e um glossário comum, dentro da profissão e práticas, para a linguagem oral e escrita sobre gerenciamento de projetos. Ele possui 47 processos abordados que são divididos em 10 áreas de conhecimentos, formando um fluxo de processo. Uma área de conhecimento é definida por seus requisitos de conhecimentos e descrita em termos dos processos que a compõem, suas práticas, entradas, saídas, ferramentas e técnicas (VARGAS, 2014, p. 17).

Para Sotille (2014), PMBOK é um guia genérico, que pode ser usado por todas as áreas de conhecimento, das engenharias, indústrias, até o desenvolvimento de software. Os conceitos e técnicas contidos no PMBOK são válidos para o projeto de desenvolvimento de software, mas a produção de um software possui naturezas particulares e específicas.

Segundo Sbrocco (2012, p. 72), "de acordo com o PMBOK, os processos de gerência de projetos se resumem" em:

> Processos de Iniciação.
>
> Processos de planejamento.
>
> Processos de execução.
>
> Processos de controle.
>
> Processos de finalização.

O Gerenciamento de Projetos, na visão do PMI, de acordo com o Guia PMBOK identifica e descreve as principais áreas de conhecimento. Cada uma dessas áreas (no total de 9) é descrita por meio de processos e se refere a um aspecto a ser considerado dentro da gerência de projetos. As áreas de conhecimento que compõem o modelo proposto pelo PMBOK são:

- Qualidade.
- Recursos Humanos.
- Escopo.

- Aquisições.
- Integração.
- Comunicações.
- Custo.
- Riscos.
- Tempo.

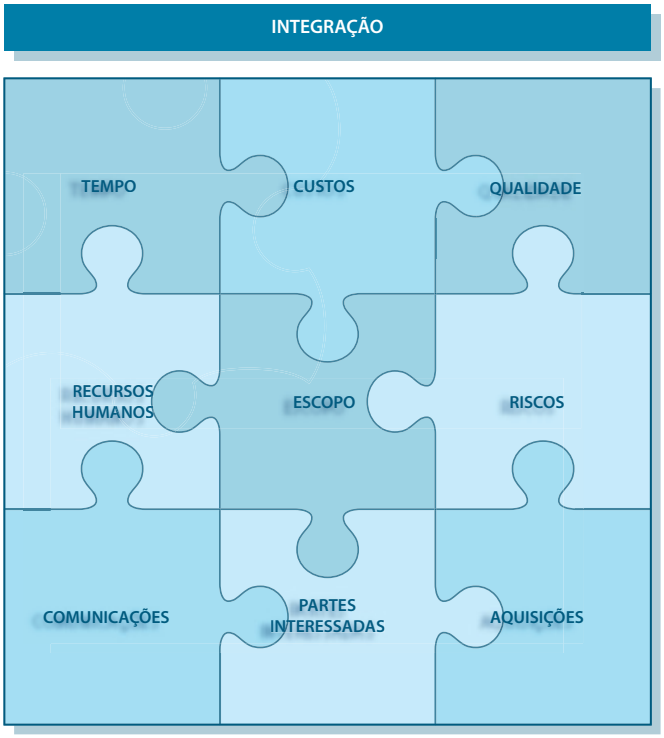


Figura 2 - As áreas de conhecimento do gerenciamento de projetos
Fonte: Vargas (2014, p. 22).

REFLITA

Na Engenharia de Software o projeto começa com uma necessidade de negócio e uma visão da solução, também considerando restrições de tempo e custo.

(José Carlos Cordeiro Martins)

AGILE PROJECT MANAGEMENT (APM)

O termo Gerenciamento Ágil de Projetos (APM), segundo Amaral et al (2011) surgiu devido a um movimento iniciado pela comunidade de desenvolvimento de sistemas de informações, que sentiu a necessidade de um novo enfoque de avanço no software, que se baseasse na "agilidade", na "flexibilidade" nas comunicações e a capacidade de desenvolver novos softwares em um curto período de tempo.

Conforme Pressman (2016, p. 66) "A Engenharia de *Software* ágil combina filosofia com um conjunto de princípios de desenvolvimento [...]. Os princípios de desenvolvimento priorizam a entrega mais do que a análise e o projeto, e também a comunicação ativa e contínua".

A definição Gerenciamento Ágil de Projetos segundo Amaral et al:

> [...] o gerenciamento ágil de projetos é uma abordagem fundamentada em um conjunto de princípios, cujo objetivo é tornar o processo de gerenciamento de projetos mais simples, flexível e iterativo, de forma a obter melhores resultados em desempenho (tempo, custo e qualidade), menor esforço em gerenciamento e maiores níveis de inovação e agregação de valor ao cliente (AMARAL et al, 2011, pg 21).

O desenvolvimento ágil, segundo Pressman (2016, pg. 67), "não significa que nenhum documento é criado, significa que, apenas, os documentos que vão ser consultados mais adiante no processo de desenvolvimento são criados". E uma das características da metodologia ágil que mais atrai, é a habilidade em se reduzir os custos das alterações que podem ocorrer em um processo de software.

Como os métodos ágeis reduzem o custo das alterações? Porque o software é liberado de forma incremental, fazendo com que as alterações passem a ser mais controladas. Para Massari (2014, p. 18) "ser ágil é utilizar um conjunto recomendado de processos e ferramentas de gerenciamento de projetos, para aqueles projetos onde reinam riscos, incertezas e altas possibilidades de mudanças, e também focados muito mais no fator humano, na interatividade entre as pessoas do que em processos e ferramentas".

Ao falarmos em mudança, queremos uma equipe ágil que seja rápida e eficaz em responder a essas possíveis mudanças, pois o software pode ter várias mudanças durante o seu desenvolvimento. A mudança conduz a agilidade, e a metodologia ágil pode ser aplicada a qualquer processo durante o desenvolvimento de um projeto de software. (PRESSMAN, 2016).

Conforme Pressman (2016), para ser aplicado o desenvolvimento ágil, é essencial que:

- Seja projetado de forma que a equipe se alinhe e se adapte às tarefas.
- Que o planejamento possa ser conduzido com fluidez da metodologia de desenvolvimento ágil.
- Que possa eliminar tudo, menos os artefatos considerados essenciais.
- Enfatizar a estratégia de entrega incremental.
- Software operacional para o tipo de produto e ambiente.

PRINCÍPIOS DA AGILIDADE

Conforme Pressman (2016, p. 70), "a Agile Alliance estabelece 12 princípios para alcançar a agilidade", que são:

- Prioridade é satisfazer o cliente com a entrega adiantada e contínua.
- Aceite os pedidos de alterações do cliente, pois os processos ágeis deve aproveitar essas mudanças como vantagem competitiva.
- Entregar software em funcionamento frequente (intervalos custos).
- Equipe do comercial e de desenvolvedores devem trabalhar em conjunto.
- Ambiente e apoio necessários para uma equipe sempre motivada.
- Conversa aberta para comunicação mais efetiva e eficiente.
- Medida de progresso é um software em funcionamento.
- Processos ágeis promovem o desenvolvimento sustentável.
- Excelência técnica em bons projetos aumenta a agilidade.
- Simplicidade é essencial (maximizar o volume de trabalho).
- Equipes auto-organizadas (arquiteturas, requisitos).
- Avaliação da equipe em intervalos regulares.

Não se aplica, necessariamente, esses doze princípios a todo modelo de processo ágil. Mas é importante, usar ou aplicar um ou mais desses princípios.

Conforme Sommerville (2011), os métodos ágeis podem ser bem sucedidos quando aplicados para alguns tipos de desenvolvimento de sistemas:

1. Desenvolvimento de produtos para venda.
2. Desenvolvimento de sistemas personalizados para uma empresa.

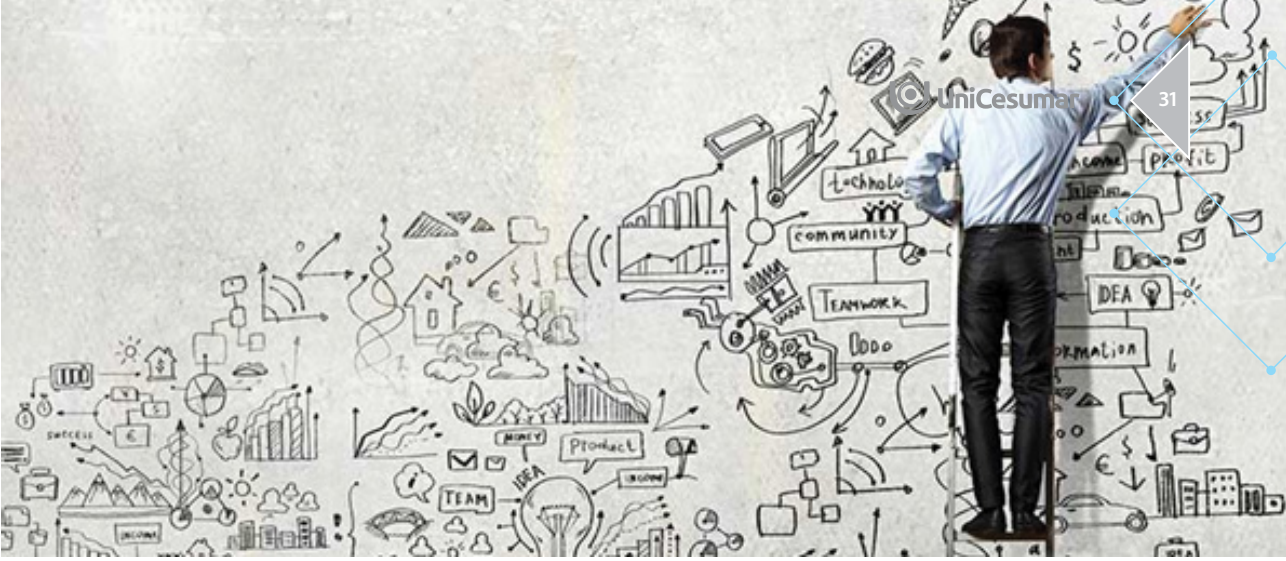
Ninguém é contra a agilidade, mas para Pressman (2016), o importante é saber qual é a melhor forma de atingi-la e, também, como desenvolver um software que atenda com qualidade as necessidades dos clientes e que possa ser estendido e alterado ao longo do prazo.

Conforme Pressman (2016), existem vários modelos de processos propostos e muitos são adaptações de conceitos da engenharia de software. A conclusão, segundo Pressman (2016), é que devemos considerar o que há de melhor nas duas metodologias.

REFLITA

Você não tem de escolher entre agilidade ou engenharia de software. Em vez disso, defina uma abordagem de engenharia de software que seja ágil.

(Roger Pressman)



PRÁTICAS DO GERENCIAMENTO DE SOFTWARE

Ao colocar em prática o gerenciamento de projetos de software, devemos levar em considerações algumas práticas, por exemplo modelos, cronogramas, diagramas, pessoas envolvidas e métricas que auxiliam e ajudam a garantir que o projeto seja realizado corretamente.

PESSOAS

Segundo Pressman (2016, p. 687), "pessoas constroem *software* de computador, e os projetos são bem-sucedidos porque pessoas bem treinadas e motivadas fazem as coisas". Agora vamos conhecer os envolvidos *(stakeholders)* que podem ser divididos em:

- Gerentes: definem os problemas do negócio.
- Gerentes do projeto: planejam, motivam, organizam e controlam a equipe de desenvolvedores.
- Profissionais: possuem habilidades técnicas de um produto ou de uma aplicação.
- Clientes: especificam os requisitos para o software.
- Usuários: que interagem com o sistema.

A equipe do projeto, para ser eficiente, deve estar organizada de forma que a capacidade e habilidades de cada um seja maximizada. Para Pressman (2016),

o gerenciamento de projeto é uma atividade que envolve muitas pessoas e, por isso, um gerente de projetos deve se concentrar em entender o problema a ser resolvido, administrar as ideias e deixar claro à equipe, que o que conta muito é a qualidade.

EQUIPE DE SOFTWARE

Segundo Pressman (2016, p.689), "a melhor estrutura de equipe depende do estilo de gerenciamento das organizações, da quantidades de pessoas na equipe, de seus níveis de habilidades e do grau de dificuldade geral do problema". Alguns paradigmas organizacionais para equipes de engenharia de software:

- **Paradigma fechado:** estrutura uma equipe em termos de hierarquia de autoridade tradicional.
- **Paradigma randômico:** estrutura uma equipe vagamente e depende da iniciativa individual de seus membros.
- **Paradigma aberto:** estrutura a equipe de maneira que consiga alguns dos controles associados ao paradigma fechado, mas com inovação do paradigma randômico.
- **Paradigma sincronizado:** organiza os membros da equipe a trabalhar nas partes do problema.

Mas qual seria o paradigma ideal para uma equipe de alto desempenho? Depende da organização, mas devemos ter membros da equipe que confiem uns nos outros e que as habilidades de cada um sejam divididas adequadamente ao problema (PRESSMAN, 2016).

A MODELAGEM

Um modelo é uma simplificação da realidade que usamos para compreender melhor o sistema que estamos desenvolvendo. Os modelos ajudam de diversas formas, como:

- Ajudam a visualizar o sistema como ele é ou como desejamos que ele fique.
- Ajudam a especificar a estrutura ou o comportamento de um sistema.
- Ajudam a proporcionar um guia para a construção do sistema.
- Ajudam a documentar as decisões tomadas.

Um dos motivos, que a modelagem é muito importante é porque podemos delimitar o problema que estamos resolvendo, dividindo em vários problemas menores, restringindo a atenção a um único aspecto por vez até chegarmos à solução total do problema em questão.

Quando usamos modelagem, qualquer projeto será beneficiado. A modelagem auxilia a equipe a ter uma visão mais abrangente do funcionamento do sistema, e assim, desenvolvê-lo de forma mais rápida e correta. Frequentemente a modelagem de software usa algum tipo de notação gráfica e são apoiados pelo uso de ferramentas, como é o exemplo da UML. A UML (Unified Modeling Language), é uma linguagem para especificação, documentação, visualização e desenvolvimento de sistemas orientados a objetos. A seguir alguns outros exemplos de ferramentas para modelagem de software:

- **JUDE (Atual ASTAH):** é uma ferramenta grátis mais usada e poderosa para modelagem UML.
- **Umbrello UML:** ferramenta open-source de modelagem UML para LINUX.
- **Poseidon para UML:** ferramenta alemã para modelagem de sistemas.

- **Microsoft Visual Modeler:** ferramenta para modelar projeto de software que utiliza componentes.
- **Visio:** ferramenta usada para criar variados tipos de diagramas e faz parte do pacote Office.

DIAGRAMAS

Os diagramas de projetos de software podem representar um grande conjunto de informações, desde o projeto de arquitetura até o código fonte do sistema. A seguir algumas das informações que podem ser representadas visualmente usando diagramas:

- Arquitetura geral do sistema.
- Dependências do sistema.
- Complexidade.
- Fluxo de informações do sistema.
- Requerimentos do negócios.
- Organização e estrutura do banco de dados.
- Código fonte.
- Configurações do sistema.

Temos uma infinidade de tipos de diagramas que podem ser aplicados diversos fins dentro do gerenciamento de projetos. Alguns tipos de diagramas são: de classes, de sequência, de casos de uso entre outros. O interessante é escolher um tipo de diagrama que melhor se adapte no contexto do seu projeto e na metodologia escolhida.

- Diagramas Estruturais.
- Diagramas de Classes.
- Diagramas de Sequência.

- Diagrama de Casos de Uso.
- Diagrama de Implantação.
- Diagrama de Atividades.
- Diagrama de Comunicação.

CRONOGRAMA

Cronogramas de projeto de software, segundo Pressman (2016, p. 757) "é uma ação que distribui o esforço estimado por toda a duração planejada do projeto, alocando esforço para tarefas específicas de engenharia de software". Os cronogramas evoluem com o tempo e, a cada fase, ele é refinado em um cronograma mais detalhado.

Conforme Sommerville (2011, p. 437), "os cronogramas do projeto podem ser representados em uma tabela ou em uma planilha mostrando as tarefas, o esforço, a duração esperada e as dependências de tarefas". Como exemplo de cronograma, veja a tabela a seguir com um conjunto hipotético de tarefas, em que é exibido o esforço estimado, a duração e as interdependências de tarefas.

Tabela 1 - Tarefas, durações e dependências

TAREFA	ESFORÇO (PESSOAS-DIAS)	DURAÇÃO (DIAS)	DEPENDÊNCIAS
T1	15	10	
T2	8	15	
T3	20	15	T1
T4	5	10	
T5	10	5	T2, T3, T4

Fonte: Sommerville (2011, p. 439).

Além desse tipo de visualização, temos outras duas alternativas de representações gráficas de cronogramas de projetos, que conforme Sommerville (2011), muitas vezes são mais fáceis de serem lidas e compreendidas. Essas duas alternativas de representações gráficas são:

- **Gráfico de Barras:** são baseados em calendário e mostram quem é o responsável por cada atividade, o tempo decorrido e quais as atividades que estão para iniciar e terminar. São chamados de "Gráficos de *Gantt*".
- **Redes de Atividades:** são diagramas de rede e mostram as dependências entre as diferentes atividades que compõem o projeto.

Tabela 2 - Exemplo Gráficos de Gantt

ID	Nome da Tarefa	Duração	2016 J	F	M	A	J	A	S	O	N	O
1	Levantamento	120 dias										
2	Análise e Projeto	150 dias										
3	Implementação	120 dias										
4	Teste	60 dias										
5	Implantação	30 dias										
6	Treinamento	30 dias										

Fonte: a autora.

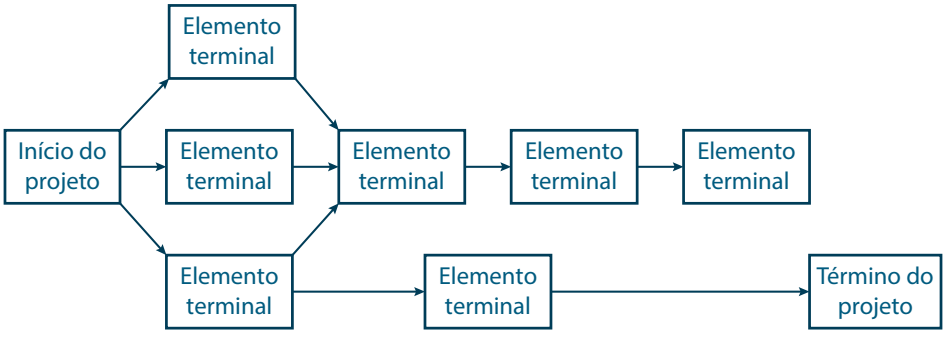


Figura 3 - Exemplo de Rede de Atividades
Fonte: a autora.

Ao desenvolver um cronograma, para Pressman (2016), é necessário que o trabalho seja dividido:

- Anotando as possíveis dependências entre as tarefas.
- Que seja atribuído esforço e tempo para cada uma dessas tarefas.

- Que seja definido as responsabilidades de cada um da equipe.
- Que seja definido os resultados e marcos esperados.

MÉTRICAS NO AUXÍLIO DO GP

Conforme Pressman (2016), a engenharia é uma disciplina quantitativa e um elemento-chave de qualquer processo é a medição. Medimos principalmente para obter controle de um projeto e, portanto, poder gerenciá-lo. Medimos e avaliamos para estimar se estamos pertos ou longe dos objetivos definidos no plano em termos de conclusão, qualidade, compatibilidade com os requisitos etc.

Segundo Pressman (2016), sempre haverá um elemento qualitativo no desenvolvimento do software e em alguns casos, a avaliação qualitativa pode não ser suficiente. A métrica proporciona uma base por meio da qual a análise, projeto, codificação e teste podem ser conduzidos mais objetivamente e avaliados de maneira mais quantitativa.

Métricas coletadas de projetos que já terminaram podem ser usadas como uma base para estimativas de esforço e tempo, para projetos atuais (PRESSMAN, 2016).

Nas métricas de projeto podemos ter relatórios sobre o andamento do projeto, prazos e tempo, segurança, qualidade, confiabilidade, custos e recursos consumidos a mais do que estava planejado.

Conforme Pressman (2016) as medições permitem que se melhore o projeto de software, pois ajudam no seu planejamento, acompanhamento e controle, avaliando sua qualidade. A tabela a seguir lista alguns exemplos dos tipos de métricas relevantes para as necessidades de uma empresa de software:

Tabela 3 - Tipos de Métricas

QUESTÃO	MÉTRICA
Custo	Custo por linha de código, custo por ponto de função, custo por caso de uso.
Tempo de implementação	Tempo decorrido por linha de código ou por ponto de função.
Defeitos no produto liberado	Defeitos descobertos após a liberação por linha de código ou por ponto de função.
Qualidade Subjetiva	Facilidade de uso, facilidade de operação, aceitação do cliente.
Medidas de Melhoria	Tempo e esforço da execução, taxas de defeito, estatística de análise causal, retrabalhos.
Capacidade Tecnológica	Ferramentas mais usadas, maturidade do processo, capacidade do domínio das ferramentas.

Fonte: a autora.

De acordo com Pressman (2016, p. 704), "a mediação é uma ferramenta de gerenciamento. Se for usada adequadamente, ela aumenta o conhecimento do gerente de projetos" e, portanto, auxilia a equipe a desenvolver um projeto bem sucedido.

REFLITA

As métricas de software escolhidas devem ser motivadas pelas metas de negócio e técnicas que deseja atingir.

(Roger Pressman)

SAIBA MAIS

Disciplinas da Gestão por Projetos

Chamamos de gestão por projetos quando uma organização visualiza suas principais demandas como projetos a serem executados e responde a essas demandas se organizando por projetos. As três disciplinas principais da gestão por projetos são visualizadas e descritas a seguir.

Gerenciamento de portfólio: é o processo sistemático pelo qual a organização avalia as oportunidades existentes, transformando-as em projetos mediante a avaliação de seu alinhamento à estratégia da empresa, valor que gera para a organização, o risco e a capacidade de execução.

Gerenciamento de programas: programas são conjuntos de projetos e iniciativas que têm objetivos comuns e que precisam ser coordenados entre si.

Gerenciamento de projetos: disciplina clássica coberta pelos processos do PMBOK, que tem como objetivo principal viabilizar a entrega de projetos individuais que atendam às especificações de prazo, escopo, custo e qualidade acordadas com o cliente.

Fonte: adaptado de Valle et al (2010).

CONSIDERAÇÕES FINAIS

Prezado(a) aluno(a), ao longo desta unidade tivemos a chance de de entender os conceitos relacionados ao gerenciamento de projetos, perceber como são usados nas metodologias clássicas e nas metodologias ágeis de gerenciamento de software.

Foram apresentados aspectos relativos ao projeto e processos e suas diferenças. Aprendemos que os projetos estão relacionados a algo novo que possui um início e um fim bem definidos e que os processos ocorrem de maneira contínua e, normalmente, são estabelecidos quando o projeto é concluído.

Discutimos sobre as diferenças entre projetos aplicados a manufatura, projetos de desenvolvimento de software e projeto de software. Aprendemos que a principal diferença entre eles é a natureza dos projetos, podendo ser físico, palpável, mensurável ou algo subjetivo, invisível, impalpável e imensurável ou que faça parte de um processo.

Caminhando para o final de nossa unidade, conhecemos algumas instituições que se preocupam em estudar, formalizar e divulgar boas práticas de Gerenciamento de Projetos, como o PMI e PMBOK (Project Management Body of Knowledge).

E finalizando nossa abordagem, foi visto uma introdução ao Gerenciamento Ágil de Projetos, seus conceitos, práticas e princípios e também abordamos algumas práticas adotadas no gerenciamento de projetos, como modelos (cronograma, orçamento, diagramas), aspectos da equipe e métricas no auxílio do gerenciamento de projetos.

Depois desta unidade, com o conhecimento que já adquirimos sobre Gerenciamento de software e seus conceitos e metodologias, podemos passar para a próxima unidade para que, desse modo, você se aprofunde ainda mais nos conhecimentos sobre Gerenciamento de software. Preparados? Então vamos em frente! Bons estudos!

ATIVIDADES

1. Marque com **V** para a afirmativa verdadeira e com **F** para a afirmativa falsa, sobre Gerenciamento de Projetos:

   ( ) O Gerenciamento de projetos envolve o planejamento de requisitos, de códigos e eventos que possam ocorrer a medida que o software é desenvolvido.

   ( ) O Gerenciamento de Projetos de Software é uma atividade de apoio da Engenharia de Software, pois ele inicia antes de qualquer atividade técnica e prossegue ao longo da modelagem, construção e utilização do software.

   ( ) Usar Gerenciamento de Software é importante quando pensamos em desenvolver um produto que seja complexo, ou quando é necessário ter um controle sobre os prazos e custos do produto ou quando for necessário compartilhar recursos.

Assinale a opção com a sequência CORRETA:

a. V, F, V.

b. F, V, V.

c. V, V, V.

d. F, V, F.

e. F, F, F

2. Você é o gerente de projeto da empresa X. A empresa X adotou o PMBOK como uma ferramenta padrão para determinar como os projetos devem funcionar e você está envolvido na construção da padronização de todos os projetos futuros. **Considerando-se essa informação, descreva as áreas de conhecimento que compõem o modelo proposto pelo PMBOK.**

3. Segundo o PMI (Project Management Institute), um projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo. **Marque a opção que NÃO representa uma característica de projeto.**

   a) Feito por pessoas.

   b) Elaborado progressivamente.

   c) Repete-se todos o meses.

   d) Tem início e fim (esforço temporário).

   e) Cria um resultado único (criar um produto, serviço ou resultado exclusivo).

4. O Gerenciamento de Projetos possui muitas técnicas que podem ser aplicadas a qualquer tipo de projeto, da engenharia civil ao software. Mas existem algumas diferenças entre projetos que precisamos conhecer. **Com base nisso, qual a diferença entre projetos e projetos de desenvolvimento de software?**

ATIVIDADES

5. Para Pressman (2016) o desenvolvimento ágil não significa que nenhum documento é criado, significa que apenas os documentos que vão ser consultados mais adiante no processo de desenvolvimento são criados. **Diante disso, o que significa ser ágil no contexto da Engenharia de Software*?***

GERENCIAMENTO DE PROJETOS DE DESENVOLVIMENTO DE *SOFTWARE* COM O *RUP* E O *PMBOK*

Nos dias atuais, no âmbito do desenvolvimento de software, uma boa gerência de projetos tem se tornado um fator competitivo no mercado, dado que influencia diretamente na qualidade do produto final. Muitas empresas e organizações voltadas para o desenvolvimento de softwares investem na melhoria de seus processos de desenvolvimento. Existem diversos padrões, referência, ou modelos reconhecidos que podem ser aplicados para atingir esse fim. De modo geral, esses modelos apresentam apenas metas ou estruturas necessárias para que um processo de desenvolvimento apresente excelência na qualidade de seus produtos, mas não determinam como projetar ou implantar as melhorias necessárias no processo de desenvolvimento.

Um projeto consiste em um empreendimento temporário, cujo objetivo é a criação de um produto ou serviço único. Os projetos podem envolver unidades isoladas de uma organização, ou atravessar as fronteiras organizacionais. Muitos projetos são componentes críticos da estratégia de negócios da organização [...]. Na atualidade, o planejamento e execução de projetos nas organizações enfrenta o desafio de desenvolver suas atividades observando os critérios de produtividade, qualidade e cumprimento dos seus planejamentos estratégicos. O gerenciamento de projetos é uma disciplina que tem como objetivo a melhoria do desempenho de um projeto como um todo. As metodologias de gerência de projetos baseadas no PMBOK tem sido muito usadas por diversas organizações. A abrangência dos projetos que podem ser gerenciados consiste no diferencial dessas metodologias. Assim sendo, e considerando especificamente os projetos para o desenvolvimento de software, as organizações cada vez mais investem na melhoria de seus processos de desenvolvimento, o que proporciona grandes benefícios para elas, aumentando a qualidade de seus produtos e diminuindo os esforços para produzi-los e mantê-los [...].

O *Rational Unified Process (RUP)* define um *framework* para um processo de desenvolvimento de software baseado em boas práticas de engenharia de software. Utiliza a abordagem iterativa e incremental de desenvolvimento e é personalizável de acordo com as necessidades específicas de cada projeto de desenvolvimento de software. O RUP implementa as práticas de engenharia de software listadas anteriormente por meio de uma abordagem em duas dimensões [...].

RUP e o PMBOK não "conversam" entre si, ou seja, o RUP não referência o PMBOK quando necessita de algum processo, atividade, ferramenta ou artefato de gerenciamento de projeto; da mesma forma o PMBOK não referencia o RUP em relação à processos, atividades, ferramentas, técnicas e artefatos de engenharia de software, necessários para um projeto de desenvolvimento de software. Um outro ponto a se destacar também, é o fato de que nem o RUP nem o PMBOK indicam o caminho para se realizar um projeto de desenvolvimento de software, usando as boas práticas de engenharia de software do RUP em conjunto com as boas práticas de gerenciamento de projetos do PMBOK

[...]. O planejamento de cada iteração é desenvolvido por meio do uso de técnicas de planejamento de projetos, como as estabelecidas pelo PMBOK, onde cada iteração pode ser vista como um projeto em cascata, tomando por base as atividades e processos previstos no RUP.

Fonte: adaptado de Campos e Lima (2009, on-line)[1].

MATERIAL COMPLEMENTAR

LIVRO

Gerenciando Projetos de desenvolvimento de Software com PMI, RUP e UML.
José Carlos Cordeiro Martins

Editora: Brasport
Sinopse: esse livro propõe a combinação de duas metodologias de gerenciamento de projetos - o RUP e a abordagem do PMI. Apresenta-se dividido em partes, cada qual abordando o assunto pertinente ao gerenciamento de projetos de desenvolvimento de software a partir de um ângulo diferente. Inicialmente, apresenta uma breve descrição da disciplina de gestão de projetos, conforme documentada pelo Project Management Institute (PMI). A seguir, a obra comenta sobre a administração de recursos humanos, que normalmente são o principal insumo no desenvolvimento de sistemas. Por fim, o livro traz uma abordagem de como todas as metodologias e processos apresentados podem ser utilizados de forma combinada, visando aumentar a chance de sucesso no desenvolvimento de sistemas.

LIVRO

Certificação PMP, alinhando com o PMBOK
Armando Monteiro

Editora: Brasport
Sinopse: esse livro tem como diferencial os tópicos mais importantes do PMBOK Guide em uma linguagem menos conceitual e mais prática e próxima das questões do exame. Além disso, as questões resolvidas permitem ao leitor entender como o PMI as desenvolve.

NA WEB

Tradicional vs Ágil
Artigo excelente que fala sobre as diferenças entre um modelo tradicional e um Ágil. A principal diferença entre um modelo Tradicional e um Ágil é o tipo de controle nos processos de cada um. Para saber mais sobre as diferenças acesse : <http://www.culturaagil.com.br/tradicional-vs-agil/>.

PMO: 10 materiais para se tornar um expert
Artigo que fala sobre como se tornar um expert em PMO. Um PMO pode gerar e evidenciar valor dentro da sua organização, simplificando processos e alavancando sua capacidade de gerar resultados. Ele fornece informações precisas, confiáveis e de valor para apoiar a tomada de decisão sobre os projetos e/ou programas, identificando os que trazem mais resultados, dentre outras coisas. Fique por dentro. Para saber mais, acesse: <http://www.projectbuilder.com.br/blog-pb/entry/dicas/kit-tudo-que-voce-queria-saber-sobre-pmo>.

REFERÊNCIAS

AMARAL D. C.; CONFORTO E. C.; BENASSI J. L. G.; ARAUJO C. de. **Gerenciamento Ágil de Projetos – Aplicação em produtos inovadores**. São Paulo: Saraiva, 2011.

DINSMORE, P. C.; CABANIS-BREWIN J. AMA - **Manual de Gerenciamento de Projetos.** Rio de Janeiro: Brasport, 2009.

FERNANDES, A. A.; TEIXEIRA, D. de S. **Fábrica de Software:** implantação e gestão de operações. São Paulo: Atlas, 2011.

MARTINS, J. C. C. **Técnicas para Gerenciamento de Projetos de Software**. Rio de Janeiro: Brasport, 2007.

PRESSMAN, R.; MAXIM, B. R. **Engenharia de Software – Uma abordagem profissional**. 8 ed. Porto Alegre: AMGH, 2016.

PROJECT MANAGEMENT INSTITUTE. **Um Guia do Conhecimento em Gerenciamento de Projetos (Guia PMBOK®)** — Quinta Edição, 2013.

SBROCCO, J. H. T. C.; MACEDO, P. C. de. **Metodologias Ágeis:** Engenharia de Software sob medida. 1 ed. São Paulo: Érica, 2012.

SOMMERVILLE I. **Engenharia de Software.** 9 ed. São Paulo: Pearson, 2011.

SOTILLE, M. **Gerenciamento de Projeto na Engenharia de Software**. Porto Alegre: PM Tech Capacitação em Projetos, 2014.

MASSARI, V. L. **Gerenciamento Ágil de Projetos**. Rio de Janeiro: Brasport, 2014.

TSUI F.; KARAM O**. Fundamentos de Engenharia de Software.** Rio de Janeiro: LTC, 2013.

VALLE, A. B. do; SOARES, C. A. P.; FINOCCHIO, J. Jr.; SILVA L. de S.; Firmino da. **Fundamentos do gerenciamento de projetos.** Publicações FGV Management, 2 ed. Rio de Janeiro: FGV, 2010.

VARGAS R. V. **Manual Prático do Plano de Projeto - utilizando o PMBOK Guide**. 5 ed. Rio de Janeiro: Brasport, 2014.

REFERÊNCIAS ON-LINE

[1]Em: <http://www.aedb.br/seget/arquivos/artigos09/163_seget_2009.pdf>. Acesso em: 12 abr. 2017.

GABARITO

1. B.

2. As áreas de conhecimento que compõem o modelo proposto pelo PMBOK são:

- Qualidade.
- Recursos Humanos.
- Escopo.
- Aquisições.
- Integração.
- Comunicações.
- Custo.
- Riscos.
- Tempo.

3. C.

4. Um dos pontos principais para todas as diferenças é a natureza dos projetos. O projeto de manufatura é algo físico, palpável, mensurável e o projeto de software é algo subjetivo, invisível, impalpável e imensurável.

5. Quando falamos em mudança, queremos uma equipe ágil que seja rápida e eficaz em responder a essas mudanças. E o software tem várias mudanças durante o seu desenvolvimento. E a mudança conduz a agilidade. Assim, a metodologia ágil pode ser aplicada a qualquer processo durante o desenvolvimento de software.

Professora Esp. Janaina Aparecida de Freitas

INTRODUÇÃO A MODELOS DE DESENVOLVIMENTO DE SOFTWARE

UNIDADE II

Objetivos de Aprendizagem

- Introdução ao Processo Unificado (UP), seus conceitos, processos e ferramentas.
- Apresentar as características da metodologia XP, as melhores práticas e exemplos de utilização.
- Apresentar as origens e características básicas da metodologia FDD.
- Apresentar os conceitos básicos e as fases da metodologia DSDM.
- Contextualizar os conceitos e as características da metodologia Iconix Process.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Processo Unificado (UP)
- Extreme Programming (XP)
- FDD - Feature Driven Development
- DSDM - Dynamic Systems Development Methodology
- Iconix Process

UniCesumar

INTRODUÇÃO

Olá, aluno(a)! Na Unidade I você aprendeu alguns conceitos relacionados ao gerenciamento de projetos que são usados tanto nas metodologias clássicas como nas ágeis.

A partir desta unidade, você começará a estudar os modelos de desenvolvimento de software e como são utilizados nas empresas. Alguns modelos que serão estudados: Processo Unificado (UP), Extreme Programming (XP), FDD, DSDM e Iconix Process.

O Processo Unificado (UP) é uma metodologia usada para gerenciar o desenvolvimento de projetos de software e garantir que tenham qualidade, que atenda as necessidades do cliente, dentro de um prazo e custo. O UP utiliza o UML como ferramenta para especificação de sistema e ele pode ser utilizado em conjunto com outras metodologias.

Estudaremos a metodologia Extreme Programming (XP), suas características, seus valores e princípios que caracterizam um projeto, suas práticas que ajudam a garantir um ciclo de desenvolvimento fortemente dependente. Possui um conjunto definido de regras que oferece condições para os desenvolvedores lidarem com as mudanças no projeto de forma eficiente.

A metodologia FDD (Feature Driven Development) ou Desenvolvimento Guiado por Funcionalidades é usado também para o gerenciamento e desenvolvimento de software e ela combina as melhores práticas do movimento ágil com Engenharia de Software Orientada a Objetos. É uma metodologia apreciada pelos desenvolvedores, pois possui um conjunto de regras e técnicas fáceis e com resultados rápidos.

Outra metodologia usada para o desenvolvimento de projetos de software é a DSDM (Dynamic Systems Development Methodology) que é centrada em estabelecer os recursos, tempo fixo, funcionalidades que atendam os prazos estipulados.

A metodologia ágil Iconix Process utiliza a modelagem dirigida por Casos de Uso que tem como objetivo estudar o comportamento do sistema sob o ponto de vista do cliente. É uma metodologia prática e simples, mas poderosa e eficaz.

Preparado(a) para continuar? Então, boa leitura.

PROCESSO UNIFICADO (UP)

Processo Unificado (UP) conforme Martins (2007, pg. 182) "é uma metodologia para gerenciar projetos de desenvolvimento de software que usa o UML como ferramenta para especificação de sistemas". Para Sbrocco (2012) o UP é um processo que contém a prática de uma disciplina em que as tarefas e responsabilidades são atribuídas à empresas que desenvolvem produtos de software.

Para Pressman o Processo Unificado (UP) é:

> [...] é uma tentativa de aproveitar os melhores recursos e características dos modelos tradicionais de processo de software, mas caracterizando-os de modo a implementar muitos dos melhores princípios do desenvolvimento ágil de software (PRESSMAN, 2016, pg. 56).

O Processo Unificado (UP) possui cinco atividades ou fases (figura 1) que fornecem diretrizes para a definição de tarefas e atribuição de responsabilidades e relacionadas com as atividades genéricas de desenvolvimento de software (PRESSMAN, 2016, Pg. 56).

SAIBA MAIS

Processo Unificado é, algumas vezes, chamado de Processo Unificado Racional (RUP, Rational Unified Process) em homenagem a Rational Corporation um dos primeiros contribuidores para o desenvolvimento e refinamento do PU e uma desenvolvedora de ambientes completos (ferramentas e tecnologia) que dão suporte ao processo.

Fonte: Pressman (2016).

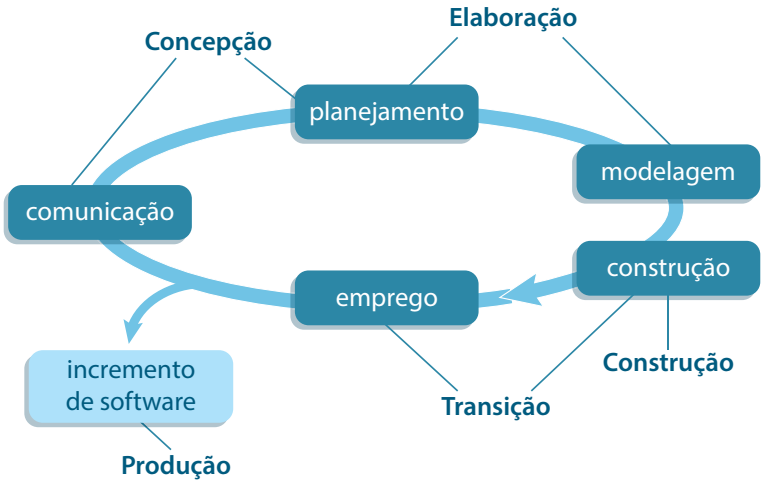


Figura 1 - Fases do Processo Unificado (UP)
Fonte: Pressman (2016, pg. 57).

Conforme Martins (2007), "fases existem em todos os projetos e cada uma termina num marco relevante para o projeto, quando uma decisão deverá ser tomada – continuar projeto e aprovar os recursos para a próxima fase ou cancelar".

Os objetivos de cada Fase do Processo Unificado são:

Na **Fase de Concepção** o Processo Unificado (UP) inclui a atividade de comunicação com o cliente e a atividade de planejamento. Nessa fase, o cliente e os usuários (os envolvidos) colaboram para identificar os requisitos de negócios para o software, é feito uma proposta em forma de rascunho da arquitetura do sistema e é desenvolvido um plano para a natureza iterativa e incremental do projeto. No planejamento são identificados os recursos, avaliado os principais riscos, é definido um cronograma inicial e estabelecido uma base para as fases seguintes que devem ser seguidas à medida que o software é desenvolvido (PRESSMAN, 2016). O objetivo é chegar a um acordo com os *stakeholders* quanto à visão do sistema e aos objetivos e estimativas das demais fases do projeto (MARTINS, 2007).

A **Fase de Elaboração** inclui a atividade de comunicação e atividade de modelagem do processo. Nessa fase, os casos de testes preliminares são refinados e a representação da arquitetura é ampliada (modelo de casos de uso, modelo de

análise, modelo de projeto, modelo de implementação e modelo de implantação). É onde as modificações a serem feitas são oportunas. O Foco dessa fase é em especificar uma arquitetura robusta e confiável para o sistema e fazer o planejamento para o restante do projeto (MARTINS, 2007).

Conforme Pressman (2016, pg. 57) a **Fase de Construção** tem "como entrada o modelo de arquitetura, a fase de construção desenvolve ou adquire componentes de software". Essa fase tem o foco na construção do sistema, no gerenciamento de recursos e, ao mesmo tempo, tenta otimizar o tempo, os custos e a qualidade (MARTINS, 2007).

Na **Fase de Transição** o objetivo é transferir o sistema aos usuários finais para teste beta e relatórios de feedback do usuário. Assim, garantir que tenha um nível de qualidade esperada e que todos os requisitos que foram especificados tenham sido atendidos. Fase onde é feita a correção de erros, migração de dados, treinamentos dos usuários, manuais de instalação e os últimos ajustes no sistema (PRESSMAN, 20016; MARTINS, 2007).

A **Fase de Produção** é onde o uso do software é monitorado, é disponibilizado suporte para o ambiente de operacional (infraestrutura) e os relatórios de defeitos e solicitações de mudanças são submetidos e avaliados (PRESSMAN, 2016).

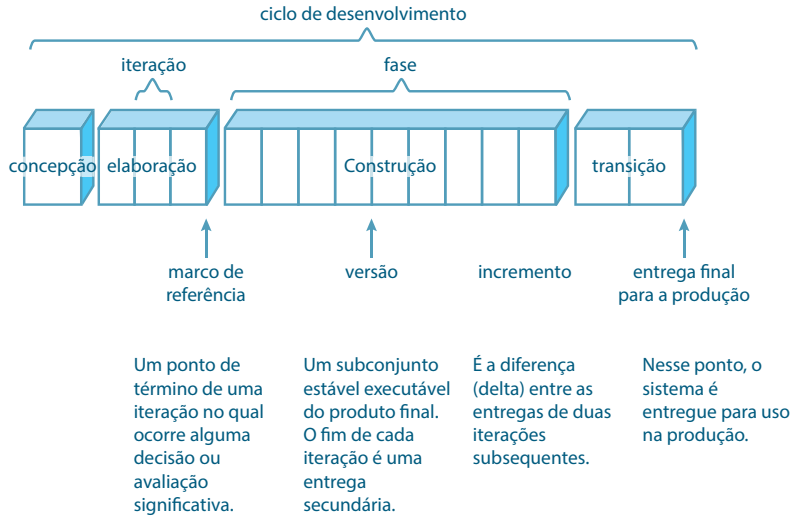


Figura 2 – As Fases do UP
Fonte: Larman, 2007.

> É provável que, ao mesmo tempo em que as fases de construção, transição e produção estejam sendo conduzidas, já se tenha iniciado o incremento de software seguinte. Isso significa que as cinco fases do PU não ocorrem em sequência, mas sim de forma concomitante e escalonada (PRESSMAN, 2016, pg. 58).

Muitos projetos falham e, para Martins (2007) as causas desses fracassos são: gerenciamento informal dos requisitos, falta de entendimentos do que o usuário quer ou necessita, dificuldade em lidar com as mudanças de requisitos, complexidade crescente, qualidade ruim e testes insuficientes ou inadequados. O Processo Unificado tenta resolver esses problemas e muitos outros que surgem ao longo do desenvolvimento do projeto, com o uso de ferramentas e de alguns recursos, como (MARTINS, 2007):

- **Desenvolvimento iterativo:** objetivo é conduzir o projeto em iterações, onde cada iteração é tratada de forma tradicional, alguns requisitos e riscos críticos são abortados, há um pouco de análise, implementação, testes e implantação e assim é feito para cada iteração, até que o produto seja concluído. Um dos benefícios dessa abordagem é que ela permite que a equipe identifique progressivamente os componentes do sistema e decida quais serão desenvolvidos, quais serão reutilizados. Outro benefício é a integração dos elementos de forma progressiva, que é montado por meio das iterações.

- **Gerenciamento de Requisitos:** o projeto precisa ser planejado para que as mudanças sejam incorporadas facilmente e que os requisitos mais importantes e críticos sejam identificados. Os requisitos são dinâmicos e mudam por vários motivos durante o ciclo de vida do projeto, como por exemplo: usuários mudam de ideia, o problema a ser resolvido muda, ocorrem mudanças técnicas e mudanças de mercado e governo.

- **Arquitetura baseada em componentes:** o objetivo dessa abordagem é viabilizar a reutilização de componentes. Permite resolver muitos problemas como: os efeito de mudanças, facilita a reutilização de códigos, facilita o gerenciamento de mudanças e facilita a construção de arquiteturas flexíveis.

- **Organização da especificação em "modelos":** modelagem de sistemas é uma forma de simplificação da realidade e o UP trabalha com os modelos de UML: Modelo de Análise, Modelo de Banco de Dados, Modelo

de Casos de Uso, Modelo de Implantação, Modelo de Implementação, Modelo de Negócio, Modelo de Design e Modelo de Teste.

- **Verificação constante da qualidade:** o foco é a criação de testes para cada cenário, num processo de avaliação contínua da qualidade.
- **Controle de mudanças:** controlar as versões dos artefatos que foram criados e modificados durante o projeto.
- **Organização do sistema com estrutura estática e dinâmica:** o Processo Unificado define: papéis, atividades, artefatos e procedimentos. Os processos definem "quem" está executando "o que" e "quando". Mas a abordagem iterativa acomoda mudanças de objetivos e estratégias dos projetos de desenvolvimento de software.
- **Trabalho com processos focados na arquitetura e nos Casos de Uso:** o Processo Unificado, em grande parte, tem seu foco em modelagem e inclui alguns elementos estáticos e dinâmicos que ajudam a orientar a exploração do sistema em todos os aspectos.

DISCIPLINAS DO UP

O Processo Unificado (PU) possui algumas disciplinas ou Workflow que descrevem sequências de atividades que mostram resultados e interações dos envolvidos e podem ser realizadas em qualquer momento do ciclo de desenvolvimento (AMUI, 2015). **As disciplinas do Processo Unificado(PU) são:**

Modelagem de Negócios: explica como descrever uma visão da organização na qual o sistema será implantado e como usar essa visão como uma base para descrever o processo, papéis e responsabilidades.

Requisitos: explica como levantar pedidos dos stakeholders e transformá-los em um conjunto de requisitos.

Análise e Design: O objetivo da análise e projeto é mostrar como o sistema vai ser realizado.

Implementação: organizar o código em camadas, definir classes e objetos, testar os componentes por unidade e fazer a integração do sistema.

Teste: validar e verificar os componentes e a integração entre os mesmos, verificar se os requisitos foram atendidos, identifica defeitos e garante que sejam corrigidos. .

Implantação: entregar o software com qualidade aos usuários finais.

Temos ainda três Disciplinas de Apoio/Suporte que são:

Ambiente: organiza as atividades e o ambiente de desenvolvimento do sistema. **Configuração e Gerência de Mudança:** envolve os gerenciamentos de configuração, solicitações de mudança e o de status e medição.

Gerenciamento de Projeto: envolve os os aspectos da gestão de riscos e de um planejamento interativo de projeto particular.

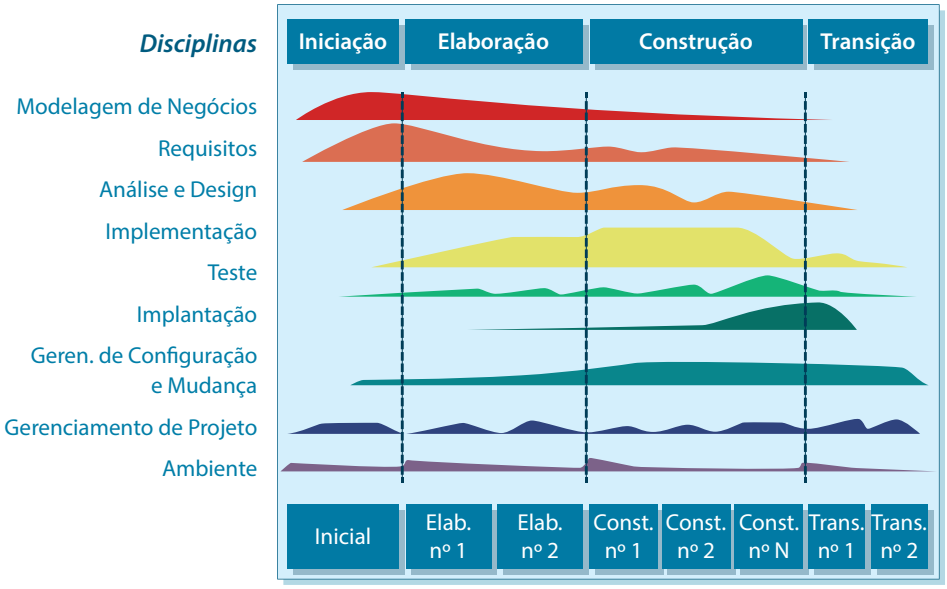


Figura 3 – Disciplinas do UP
Fonte: Martins (2007, pg, 236).

Conforme aponta Martins (2007) o UP pode e deve ser ajustado ao longo de cada projeto, por conta disso, ele pode não ser tão formal e predeterminista (abordagem clássica) e pode ser informar e fluído (abordagem ágil). Segundo o autor, o UP pode ser utilizado em conjunto com outras metodologias, como o XP, o Scrum e o PMI.

SAIBA MAIS

Iterativo e Incremental

O desenvolvimento de um produto comercial de software é uma grande tarefa que pode ser estendida por vários meses, possivelmente um ano ou mais. É mais prático dividir o trabalho em pedaços menores ou miniprojetos. Cada mini projeto é uma iteração que resulta em um incremento. Iterações são passos em um fluxo de trabalho e incrementos são crescimentos do produto. Os desenvolvedores selecionam o que deve ser feito em cada iteração baseados em dois fatores. Primeiro, a iteração deve trabalhar com um grupo de casos de uso que juntos estendam a usabilidade do produto em desenvolvimento. Segundo, a iteração deve tratar os riscos mais importantes.

Um incremento não é necessariamente a adição do código executável correspondente aos casos de uso que pertencem à iteração em andamento. Especialmente nas primeiras fases do ciclo de desenvolvimento, os desenvolvedores podem substituir um projeto superficial por um mais detalhado ou sofisticado.

Fonte: adaptado de Martins (2009, on-line)1.

REFLITA

Em seu intento, as fases do Processo Unificado são similares às atividades metodológicas genéricas.

(Roger Pressman)

EXTREME PROGRAMMING (XP)

A Extreme Programming (Programação Extrema) usa uma abordagem Orientada a Objetos como seu paradigma de desenvolvimento de software. A XP é considerada uma metodologia ágil onde os projetos são conduzidos com base em requisitos que se modificam rapidamente (SBROCCO, 2012).

> Como ocorre com muitas metodologias ágeis, a maioria das regras da XP causa polêmica à primeira vista, e muitas não fazem sentido se aplicadas isoladamente, o que a torna praticável é a sinergia de seu conjunto que sustenta o sucesso da XP, que encabeçou uma verdadeira revolução no conceito de desenvolvimento de software. A ideia básica é enfatizar o desenvolvimento rápido do projeto, visando garantir a satisfação do cliente, além de favorecer o cumprimento das estimativas (SBROCCO, 2012, pg. 143).

A metodologia XP tem como meta atender as necessidades dos clientes, com mais qualidade, com mais rapidez e de forma simples. A XP busca desenvolver os projetos da forma mais rápida, entregando-os dentro do prazo estipulado, mesmo que os requisitos se alterem com frequência. Conforme Martins (2007, pg. 144), "a XP dá preferência ao desenvolvimento orientado a objetos e permite trabalhar com pequenas equipes de até 12 desenvolvedores (programadores)".

A XP utiliza um modelo incremental, que, conforme o sistema é utilizado, novas melhorias ou mudanças podem ser implementadas. Com isso, o cliente sempre tem um sistema que pode utilizar e testar e avaliar se o que foi proposto foi desenvolvido corretamente. A XP é uma metodologia flexível e adaptativa, por isso pode ser usado em vários projetos, desde que o projeto não tenha a necessidade de um maior formalismo e retrabalhos ao longo do seu desenvolvimento (MARTINS, 2007, pg. 144).

VALORES E PRINCÍPIOS DA XP

Durante o desenvolvimento de um software podem ocorrer muitas mudanças em requisitos, projetos, regras de negócios, equipe e tecnologia. A XP ajuda a lidar com essas mudanças constantes e fornece recursos que ajudam a amenizar os problemas que ocorrem nestes casos.

Segundo Sbrocco (2012) a metodologia XP possui valores e princípios que caracterizam um projeto de software que tenha sido desenvolvido com essa metodologia. Valores e princípios que a equipe deve seguir à risca, como forma de diretrizes de desenvolvimento. A seguir, a descrição dos valores e princípios que devem ser seguidos na metodologia XP:

a) Comunicação: na metodologia XP a conversação é a principal forma de comunicação entre os desenvolvedores e clientes. A comunicação é essencial para que não haja mal entendidos, especulações, dúvidas sobre os assuntos relativos ao projeto.

b) Feedback: na metodologia XP o feedback é essencial, pois aproxima os membros da equipe com a equipe do cliente do projeto. Quando mais cedo ocorrer o feedback, mas cedo pode ser corrigido os problemas ou feito adaptações, se necessárias. O feedback permite que o cliente avalie o produto e com isso ele vai aprendendo com o sistema ou solicitando alterações ao longo do seu desenvolvimento.

c) Simplicidade: na XP a equipe é orientada a não usarem recursos desnecessários e que o desenvolvimento deve agir com simplicidade na resolução daquilo que o cliente necessita. O objetivo é atender o cliente com o necessário para que os requisitos sejam atendidos.

d) Coragem: na metodologia XP é necessário que às vezes a equipe tome atitudes complexas ou de alto risco e com isso tenha que ter coragem para modificar, sugerir, cortar e agir sobre o projeto que está sendo desenvolvido.

e) **Respeito:** na XP esse valor é muito importante, pois estimula a equipe a respeitar o cliente e vice-versa e o respeito entre os membros da equipe.

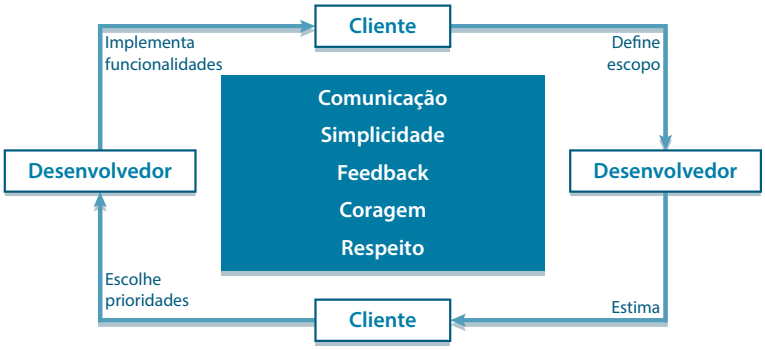


Figura 4 – Valores e Princípios do XP
Fonte: a autora.

PRÁTICAS NA XP

A metodologia XP é apoiada em práticas que tem como objetivo garantir um ciclo de desenvolvimento fortemente dependente e que são à base da metodologia. Cada prática pode desempenhar vários papéis (SBROCCO, 2012). A seguir serão descritas as práticas da metodologia XP, segundo Sbrocco (2012).

Padrões de Desenvolvimento: a equipe deve adotar um padrão de desenvolvimento antes de iniciar um projeto de software, para que todos consigam entender os códigos uns dos outros mais rapidamente. O padrão a ser adotado pode ser um *design pattern* conhecido.

Design Simples: deve-se usar um design simples, para que as mudanças e alterações que o projeto possa sofrer durante o desenvolvimento, se tornem mais fácil de resolver.

Cliente sempre disponível ou presente: a metodologia XP sugere que o cliente acompanhe e participe o desenvolvimento do projeto de software. Na medida em que o sistema sofre alterações, o cliente tem consciência das mudanças e visão de como vai ficar.

Jogo de Planejamento: esta prática ajuda a determinar rapidamente o alcance da próxima entrega do sistema (*release*) combinando as funcionalidades, prioridades, composição e prazos. Os *releases* devem ser planejados quando o projeto se inicia.

Stand Up Meeting: reunião rápida feita normalmente pela manhã e que tem 20 minutos de duração com o objetivo de os participantes troquem as experiências e dificuldades encontradas e relatar o que foi feito no dia anterior o que será feito no dia.

Programação em Pares: a metodologia XP exige que o código que está sendo implementado seja feito em dupla e no mesmo computador. A programação em dupla rende mais do que programar sozinho, pois a dupla troca ideias e experiências. Conceito utilizado também em outras metodologias.

Refatoração: durante os testes no sistema, o cliente pode querer modificar alguma coisa, ou durante as revisões de código o desenvolvedor localiza algum erro ou código mal feito, então cabe melhorá-lo e deixá-lo sem erros usando a prática de correção chamada de *refactoring*.

Desenvolvimento Guiado por Testes: quando se desenvolve um software devemos testar todas as funcionalidades para verificar se o que foi solicitado nos requisitos foi implementado.

Código Coletivo: na metodologia XP todos os desenvolvedores devem ter acesso a todas as partes do código, podendo alterá-lo, se necessário. Todos os desenvolvedores são responsáveis pelo código.

Metáfora: na XP o uso de metáforas dos processo utilizados ajuda na comunicação e no feedback entre a equipe e o cliente.

Ritmo Sustentável: a metodologia XP proíbe que a equipe de desenvolvimento trabalhe fora do horário de trabalho. Isso evita que o desenvolvedor fique cansado física e mentalmente e cause falhas na implementação.

Integração Contínua: possibilita integrar o sistema várias vezes ao dia para deixar a equipe e o cliente atualizado com as novas funcionalidades do sistema. Essa integração faz com que toda a equipe se atualize com as modificações que são feitas diariamente no sistema.

Releases Curtos: a metodologia XP exige que pequenas partes do sistema sejam liberadas ao cliente, após a implementação e em curtos períodos de tempo.

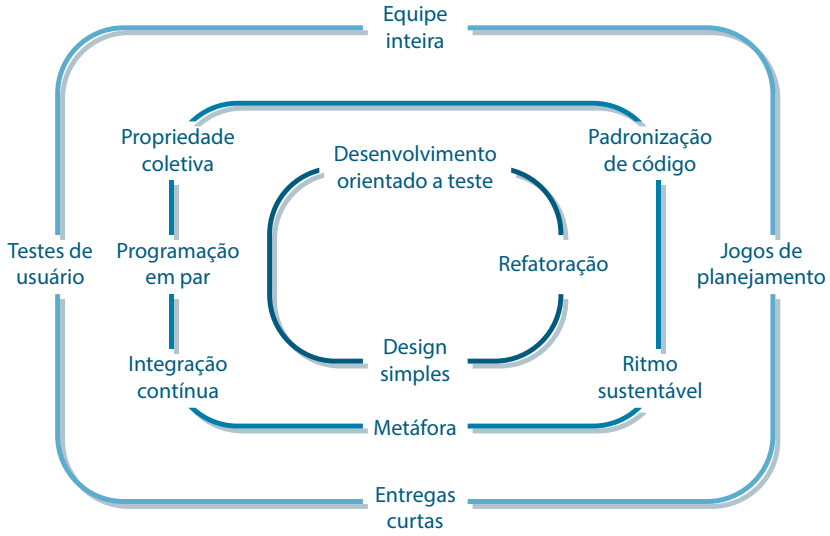


Figura 5 – Práticas XP
Fonte: a autora.

Segundo Martins (2007) uma das características mais marcantes da metodologia XP é com relação aos testes, que estão na base do desenvolvimento e com isso o sistema se torna mais estável e confiável.

REFLITA

Muitas equipes de software são constituídas por individualistas. É preciso mudar tal cultura para que a programação em pares funcione efetivamente.

(Roger Pressman)

SAIBA MAIS

A atividade de planejamento se inicia com ouvir – uma atividade de levantamento de requisitos que capacita os membros técnicos da equipe XP a entender o ambiente de negócios do software e permite obter uma percepção ampla sobre os resultados solicitados. A atividade de ouvir conduz à criação de um conjunto de "histórias" (também denominadas histórias de usuário) que descreve o resultado, as características e a funcionalidade solicitados para o software a ser construído. Cada história é escrita pelo cliente e é colocada em uma ficha. O cliente atribui um valor (uma prioridade) à história baseando-se no valor de negócio global do recurso ou função. Os membros da equipe XP avaliam, então, cada história e atribuem um custo – medido em semanas de desenvolvimento – a ela. Se a história exigir, por estimativa, mais do que três semanas de desenvolvimento, é solicitado ao cliente que ele a divida em histórias menores, e a atribuição de valor e custo ocorre novamente.

Fonte: Pressman; Maxim (2016).

FDD - FEATURE DRIVEN DEVELOPMENT

Feature Driven Development (Desenvolvimento Guiado por Funcionalidades) ou FDD como é comumente chamada, é uma metodologia ágil robusta usada para o gerenciamento e desenvolvimento de software (SBROCCO, 2012). FDD combina as melhores práticas do Gerenciamento Ágil de Projetos com uma abordagem completa da Engenharia de Software Orientada a Objetos.

FDD foi concebido em meados de 1997/1998 em um grande projeto de desenvolvimento de software para um Banco em Singapura. A partir da experiência de análise e modelagem orientada a objetos de Peter Coad, e de gerenciamento de projetos de Jeff De Luca nasceu a FDD (MARTINS, 2007).

Segundo Sbrocco (2012, pg. 99), "o FDD proporciona uma forma de trabalho que agrada todos os envolvidos no projeto, sugerindo formas de interação e controle fáceis e inteligentes". Os envolvidos no projeto apreciam o FDD, porque ele provê resultados significados mais cedo e com isso permite acompanham a evolução do projeto. Os desenvolvedores apreciam o FDD, pois ele possui um conjunto de regras e técnicas que são mais fáceis de entender e com resultados mais rápidos (SBROCCO, 2012).

> O FDD foi concebido para ser utilizado tanto em projetos para desenvolvimento de novos softwares como em projetos para evoluir um software existente. O projeto começa com a definição de uma lista de requisitos funcionais, considerando somente aqueles de valor para o negócio e que podem ser entendidos pelos clientes e usuários. Esses requisitos funcionais são chamados de *Features*, e depois de identificados, são utilizados para planejar e gerenciar o projeto. Os clientes podem priorizar os *Features* em função da sua importância (MARTINS, 2007, pg. 317).

A FDD chama a atenção para algumas características definidas a partir de um processo ágil e adaptativo:

- Iterativo.
- Blocos pequenos de funcionalidades apreciadas pelo envolvidos, chamados “Features”.
- Enfatiza a qualidade.
- Rastreabilidade e relatórios com mais precisão.
- Entrega resultados tangíveis e frequentes.

PROCESSOS DO FDD

O processo do FDD, conforme Martins (2007) começa depois que a equipe envolvida no projeto já possui todas as informações necessárias sobre o negócio ou o domínio do problema e quais os processos que precisam ser automatizados. FDD é considerada uma metodologia objetiva e possui duas fases principais:

- **Concepção e Planejamento**: fase para pensar um pouco antes de fazer (tempo estimado de 1 a 2 semanas).
- **Construção**: fase para fazer de forma iterativa (tempo de iterações de 2 semanas).

A figura abaixo, mostra como é a estrutura dos processos envolvidos no FDD.

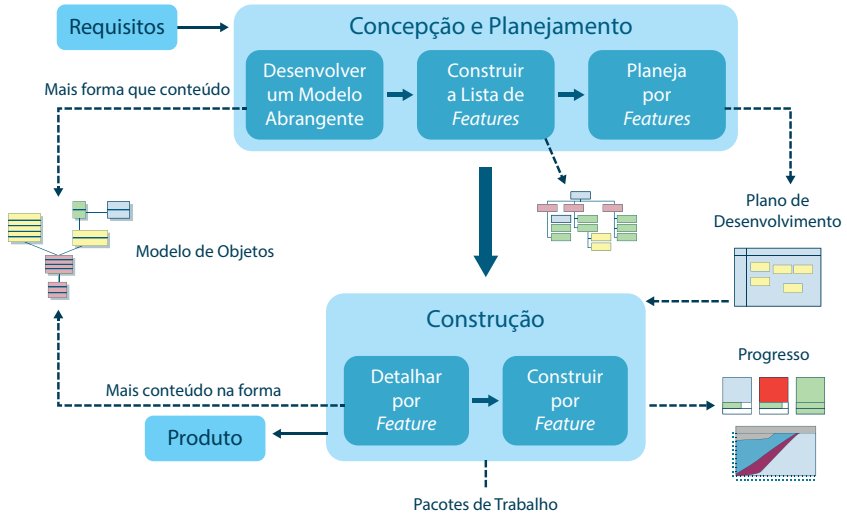


Figura 6 – Estrutura do FDD
Fonte: Heptagon (2017, on-line)[2].

Dentro das duas fases principais, temos algumas etapas com processos que são bem definidos e integrados:

1. **Desenvolver um Modelo Abrangente/Geral (DMA):** etapa onde é criado os diagramas de classes com modelos de objetos de negócio (Análise Orientada por Objetos).

2. **Construir a Lista de *Features* (CLF):** etapa onde é desenvolvida uma lista de funcionalidades agrupadas em conjuntos e áreas de negócios que o novo sistema irá apresentar (Modelagem de Requisitos).

3. **Planejar por *Features* (PPF):** etapa onde são identificados os donos das classes e das funcionalidades (Planejamento Incremental).

4. **Detalhar por *Features* (DPF):** etapa onde são identificados os donos das classes e das funcionalidades (Planejamento Incremental).

5. **Construir por *Features* (CPF):** etapa onde são detalhados os projetos técnicos e os diagramas de sequência (Desenho (Projeto) Orientado por Objetos).

6. **Construir por *Features* (CPF):** etapa onde é construindo/programado as funcionalidades (Programação e Teste Orientados por Objetos).

No FDD, o ideal é que a equipe modele o domínio do problema antes do início do projeto, pois caso precise de mais tempo para fazer o levantamento de um requisito específico, a modelagem para esse requisito em particular, pode ser adiado. Assim, mais tarde, quando os requisitos estiverem mais claros, a equipe volta a completar a modelagem para aquele requisito específico (MARTINS, 2007).

> A equipe desempenha um grande volume de trabalho de levantamento, projeto técnico e planejamento nos processos um, dois e três; isso permite que várias equipes possam trabalhar de forma iterativa e em paralelo nos processos quatro e cinco. O resultado destes processos fazem a equipe convergir rapidamente para o objetivo final (MARTINS, 2007, pg. 321).

DESENVOLVER UM MODELO ABRANGENTE/GERAL (DMA)

Segundo Martins (2007) nessa etapa o especialista do domínio do problema e a equipe de desenvolvimento do projeto trabalham juntos e fazem a análise inicial do escopo do sistema e em que contexto ele será inserido. A equipe de desenvolvimento separa o domínio do problema em partes divididas por assuntos e com vários passo a passos (*walkthrough*) e cada vez com mais níveis de detalhes. Conforme Martins (2007, pg. 322) "a medida que o trabalho se desenvolve a equipe vai construindo o modelo de classes e objetos de negócios. Depois todos os modelos são combinados para formar o modelo final".

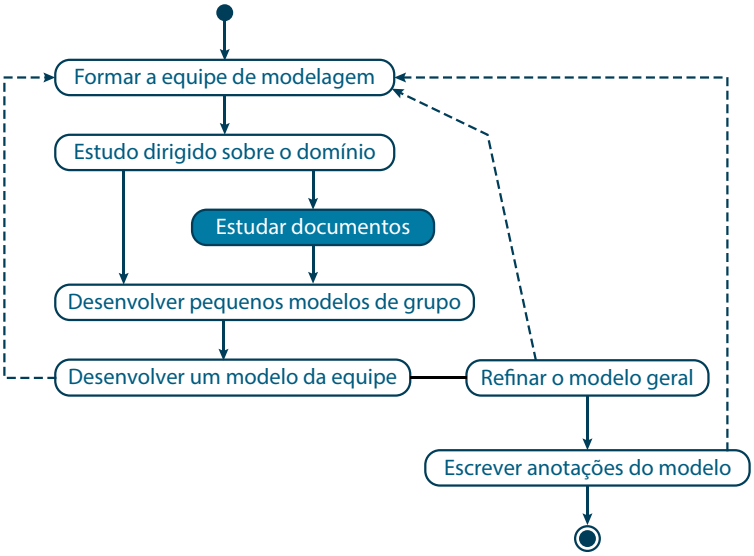


Figura 7 – Etapa Desenvolver um modelo geral
Fonte Martins (2007, pg. 323).

CONSTRUIR A LISTA DE *FEATURES* (CLF)

Nessa etapa, com base no conhecimento já adquirido na primeira fase, é decomposto as funcionalidades do domínio do problema e se passa a construir uma lista de *Features* (funcionalidades) divididas por áreas de negócio. Cada área é decomposta em atividades que formam o grupo de *Features.* Uma *Feature* é expressa na forma <ação> <resultado> <objeto>. Onde cada *Feature* pertence ao conjunto de atividades e a uma área de negócios (MARTINS, 2007).

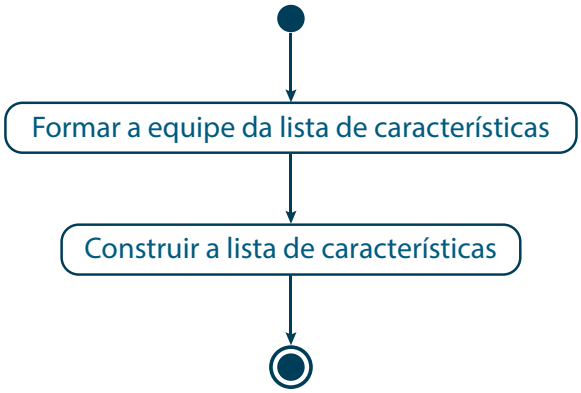


Figura 8 – Etapa Construir uma Lista de *Features*
Fonte Martins (2007, pg. 324).

PLANEJAR POR *FEATURES* (PPF)

Nessa terceira etapa, a equipe e as partes interessadas, conforme Martins (2007, pg. 325) "trabalham em conjunto, planejam a sequência de desenvolvimento dos grupos de Features (atividades de negócio) e dos conjuntos de Features (área de negócio)". Deve-se procurar um balanceamento da carga de trabalho entre os membros da equipe, durante o planejamento, para que não fiquem mais sobrecarregadas que outras (MARTINS, 2007). No final dessa etapa, é produzido o plano de desenvolvimento, que consiste em:

- Conjunto de *Features* divididos por atividades com datas de conclusão.
- Grupos de *Features* divididos por áreas de negócios com datas de conclusão derivadas dos conjuntos de *Features*.
- Atribuição de conjuntos de *Features* aos programadores chefes.
- Atribuição de classes aos desenvolvedores.

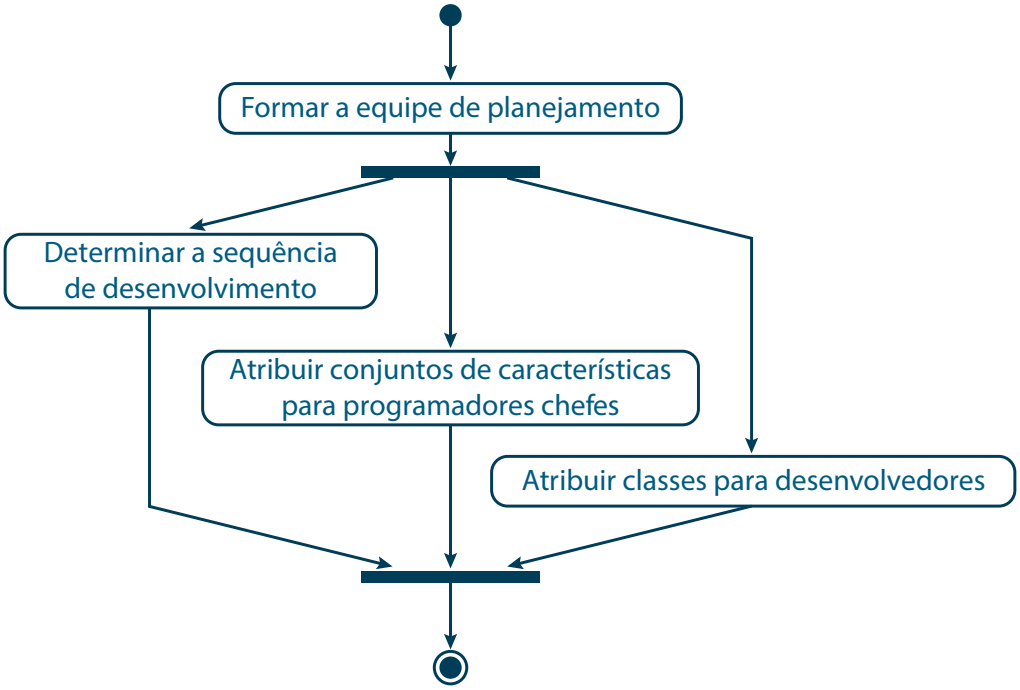


Figura 9 – Etapa Planejar por *Features*
Fonte Martins (2007, pg. 325).

DETALHAR POR *FEATURES* (DPF)

Na quarta etapa são atribuídas as *Features* aos programadores chefes, que seleciona as *Features* a partir de uma lista de prioridades e compõe seus pacotes de trabalho. A equipe pode utilizar documentos complementares sobre as *Features*, como algoritmos, fórmulas, relatórios entre outros (MARTINS, 2007).

Ao final dessa etapa a equipe disponibiliza o projeto técnico, que inclui:

- Um descritivo do conteúdo do projeto técnico.
- Requisitos referenciados na forma de documento de suporte.
- Diagramas de Sequência.

- Diferentes alternativas de projeto técnico.
- Modelo de objetos atualizado (classes, métodos e atributos).
- Esqueleto de classes, métodos e atributos.

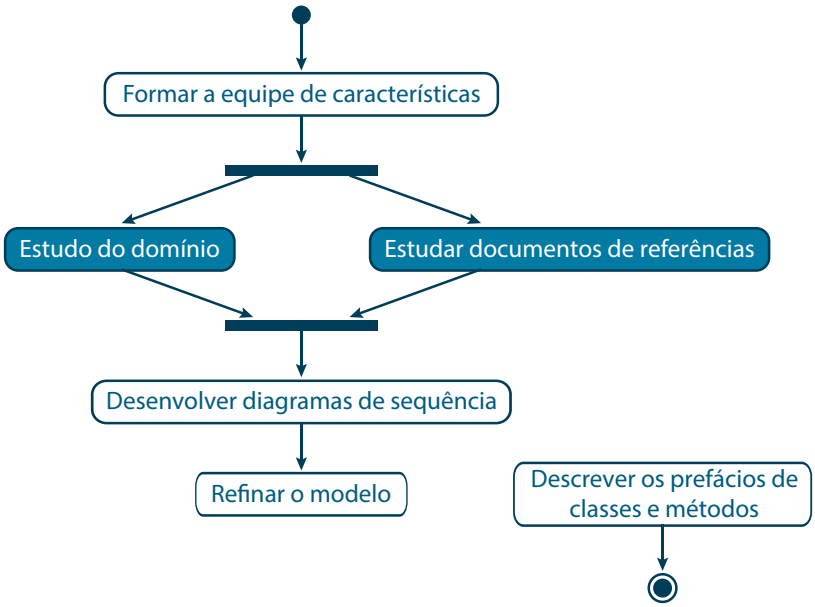


Figura 10 – Etapa Detalhar por *Features*
Fonte: Martins (2007, pg. 327).

CONSTRUIR POR *FEATURES* (CPF)

Na quinta etapa, os donos das classes fazem a programação, usando as especificações do projeto técnico, desenvolvido na etapa anterior. Depois de programado, o código-fonte passa pelos testes de unidade e de inspeção de código e finalmente as *Features* podem ser integradas a versão final.

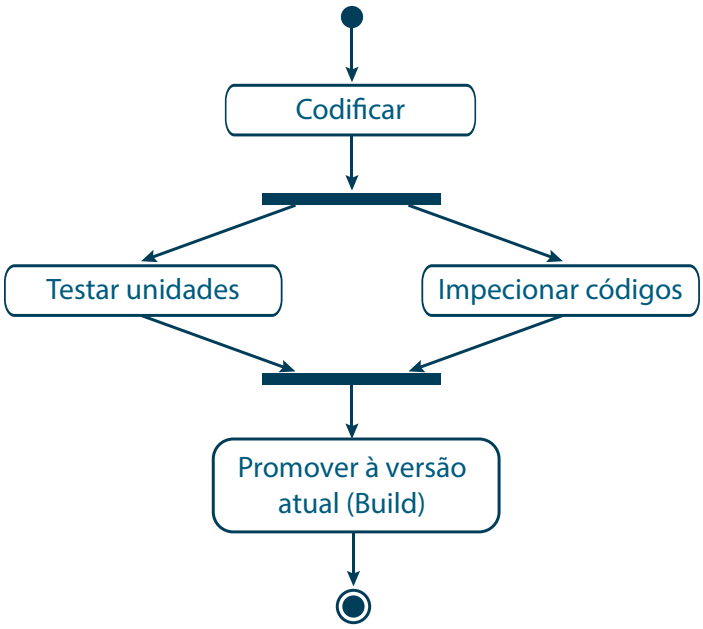


Figura 11 – Etapa Construir por *Features*
Fonte: Martins (2007, pg. 328).

Para Martins (2007, pg. 328) as práticas do FDD podem ser adaptadas conforme as regras de cada equipe e a experiência de cada envolvido. É essa flexibilidade que torna o FDD fácil de ser implantado e utilizado. Mas o autor aconselha que todas as práticas sejam consideradas.

SAIBA MAIS

As responsabilidades de uma equipe no FDD são distribuídas por papéis, onde cada membro da equipe pode assumir mais de um papel simultaneamente e um papel pode ser assumido por mais de um membro da equipe. Os principais papéis são o de Gerente de Projetos, Gerente de Desenvolvimento, Arquiteto, Gerente de Desenvolvimento, Especialista do Domínio, Programador Chefe e Dono de Classe.

Cada um desses papéis, e ainda outros que podem ser criados adicionalmente, caso haja necessidade, tem sua importância nos processos do FDD. O ciclo de vida do desenvolvimento e gerenciamento de projetos de software com a metodologia FDD se resume em duas fases, a primeira composta por três processos, que são referentes a planejamento do projeto e a segunda composta por dois processos, referentes a desenvolvimento das funcionalidades. Cada um dos processos é composto por critério de entrada, atividades, verificação e critérios de saída.

Fonte: adaptado de Cunha (2012).

DSDM - DYNAMIC SYSTEMS DEVELOPMENT METHODOLOGY

Sbrocco (2012, pg. 111) DSDM "é uma metodologia de desenvolvimento de projetos de software centrada em estabelecer os recursos e o tempo fixo para o desenvolvimento de um projeto, ajustando suas funcionalidades de maneira a atender os prazos estipulados". Ela insere-se no conceito das metodologias ágeis de apoio ao desenvolvimento de software.

O DSDM possui uma estrutura que se baseia em nove princípios, que são considerados boas práticas para a utilização da metodologia. Segue os nove princípios (Scrocco, 2012):

1. Participação ativa dos usuários e stakeholders.
2. Abordagem cooperativa e compartilhada.
3. Equipes com poder de decisão.

4. Entregas contínuas que fazem a diferença.
5. Desenvolvimento iterativo e incremental.
6. Feedback.
7. Todas as possíveis alterações durante o desenvolvimento devem ser reversíveis.
8. Fixar os requisitos essenciais.
9. Teste em todo Ciclo de Vida.

Esses princípios têm por objetivo caracterizar a metodologia. Mas, além desses, o DSDM possui alguns conceitos particulares, como o conceito *Timeboxing* Conforme Sbrocco (2012, pg. 113) "trata-se do encapsulamento do tempo reservado para ser desenvolvimento das funcionalidades, por meio da divisão dos projetos em porções, cada uma com um orçamento próprio, e fixas por uma data de entrega definida".

Outro conceito importante é a prototipagem, que possibilita a verificação e a descoberta de problemas e limitações do projeto antecipadamente. Esse conceito é, segundo Sbrocco (2012), um dos principais fatores de sucesso da DSDM. O conceito MoSCoW prioriza os requisitos durante o período de desenvolvimento. A ideia fundamental é priorizar e implementar os requisitos que sejam considerados principais, deixando os menos importantes para depois.

Segundo Sbrocco, não é todo tipo de projeto que se pode adaptar a metodologia DSDM:

> [...] é importante observar que não é todo tipo de projeto de software que pode se adaptar ao DSDM, existindo algumas restrições que devemos conhecer. Por se tratar de uma metodologia ágil que exige pouca formalidade, os sistema de segurança crítica, ou seja, sistemas que não podem falhar por colocar vidas em risco, não devem ser conduzidos por DSDM, bem como por outras metodologias menos rigorosas. Além de sistemas críticos à segurança humana, se não houver a participação efetiva dos clientes (stakeholders), se eles não estiverem disponíveis, e se não houver confiança e responsabilidade, os trabalhos podem ser comprometidos (SBROCCO, 2012, pg. 113).

FASES DA DSDM

Existem cinco fases básicas em relação ao processo do DSDM antecedidas por uma fase de pré-projeto e precedidas pelo pós-projeto que são (SBROCCO, 2012):

- **Estudo da Viabilidade:** nessa fase verifica-se se o DSDM é a solução mais adequada, observando-se a empresa como um todo, listando os envolvidos, mapeando os riscos, analisando as técnicas a serem utilizadas e as ferramentas de trabalho.
- **Estudo de Negócio:** nessa fase são observadas as regras de negócio, os processos envolvidos, aspectos relevantes do sistema e suas necessidades de informação, definindo assim o escopo do projeto.
- **Modelo de iteração funcional:** nessa fase, os requisitos (funcionais e não funcionais) são obtidos, montando uma lista de prioridades e colocando-os no protótipo e documentando-a.
- **Projeto e Construção de iteração:** nessa fase os envolvidos revisam o sistema e comentam o resultado, dando feedback aos desenvolvedores. É a fase em que o sistema é construído.
- **Implementação:** nessa fase, o sistema é colocado em funcionamento, é feita a transição do sistema do ambiente de desenvolvimento para o operacional, cuidando do treinamento e outras tarefas que sejam necessárias.

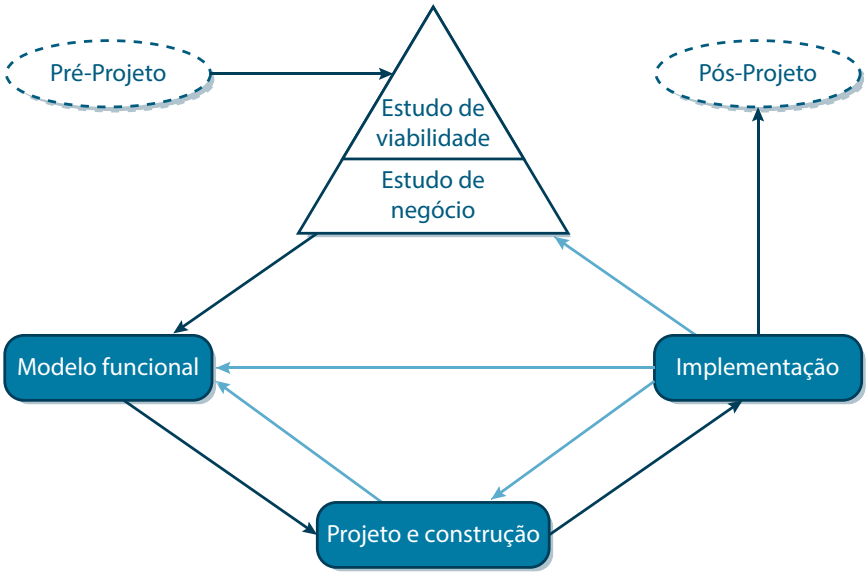


Figura 12 – Fases do DSDM
Fonte: Luna (2009).

EQUIPE DE PROJETO

O DSDM apresenta alguns papéis de usuários (stakeholders) e desenvolvedores, de acordo com o tamanho do projeto. Alguns desses papeis são: Coordenador técnico/arquiteto, Desenvolvedor sênior, Analista, programador, Designers, Testadores, Usuários, Administrador de Banco de dados (DBA), implantador e suporte técnico.

Conforme Sbrocco (2012, pg. 116) os papéis podem ser acumulados entre os envolvidos, dependo do tamanho e amplitude do projeto, ela pode variar entre duas a seis pessoas, podendo existir várias equipes pequenas em um projeto, sendo que em uma equipe de duas pessoas deve existir pelo menos um usuário e um colaborador.

SAIBA MAIS

A DSDM fornece uma framework para uma abordagem iterativa e incremental de desenvolvimento de **Sistemas de Informação (SI)**. Desenvolveu-se nos anos 90 na Inglaterra e foi aplicado pela primeira vez em 1995. Essa metodologia foi desenvolvida por um consórcio de vendedores e peritos no campo dos Sistemas de Informação, no qual partilharam e combinaram as suas melhores técnicas. Assim, a DSDM surge como uma extensão do **RAD (Rapid Application Development)**, focada em projetos de Sistemas de Informação caracterizados por prazos e orçamentos apertados. A DSDM aborda os problemas que, frequentemente, ocorrem no desenvolvimento de informação que se prendem, essencialmente, com a falta de tempo, com orçamentos mais apertados ou com outro tipo de razões para que o projeto falhe, tal como a falta de envolvimento dos encarregados do projeto ou dos utilizadores finais.

Fonte: adaptado de Teixeira (2005).

ICONIX PROCESS

O Iconix Process é uma metodologia ágil que diz respeito a uma modelagem dirigida por casos de uso e que tem como objetivo estudar e comunicar o comportamento do sistema sob o ponto de vista do usuário final (SBROCCO, 2012).

Conforme Sbrocco (2012, pg. 173) "o Iconix é considerado um método ágil não tão burocrático e formal quanto o RUP (*Rational Unified Process*), nem tão radical quanto a XP (*Extreme Programming*)".

Conforme Sousa, as características principais do Iconix Process são:

> [...] utiliza um subconjunto da UML: apenas 3 diagramas (diagramas de Caso de Uso, Classes e Sequência), mas robustez, ao invés dos 14 diagramas da UML existentes; Minimiza a paralisia da análise: ignora a semântica de estereótipos do padrão UML tais como <<*extend*>>, <<*include*>>, etc que faz o desenvolvedor perder tempo, retardando a passagem da análise para o projeto; Possui rastreamento da análise à implementação: todos os requisitos são associados a casos de uso e classes, que formam o eixo de sustentação do processo; É iterativo e incremental: várias iterações ocorrem entre o desenvolvimento do Modelo de Domínio e a modelagem dos casos de uso, enquanto que o mo-

> delo estático é incrementalmente refinado pelo modelo dinâmico; é na baseado nas questões fundamentais da orientação a objetos: o que fazem os usuários? (casos de uso); quais os objetos do mundo real? (Modelo de Domínio) ; quais os objetos relacionados com os casos de usos? (robustez); como os objetos colaboram entre si? (sequência); como realmente será construído o software? (classes) (SOUSA, 2014, pg. 35).

É considerada uma metodologia prática e simples e possui uma quantidade razoável de documentação, poderosa e com um componente de análise sólido e eficaz e, por isso, é caracterizada como um processo de desenvolvimento de software.

A metodologia Iconix Process é dividida em duas visões: Dinâmica e a Estática, onde:

Dinâmica: apresenta os aspectos comportamentais do software, a interação do usuário com o sistema. Os artefatos usados são: Diagrama de Caso de Uso, Diagrama de Robustez e Diagrama de Sequência.

Estática: apresenta os aspectos estruturais do software, mostrando o funcionamento do sistema sem nenhum dinamismo e interação do usuário. Os artefatos usados são: Modelo de Domínio e Diagrama de Classe.

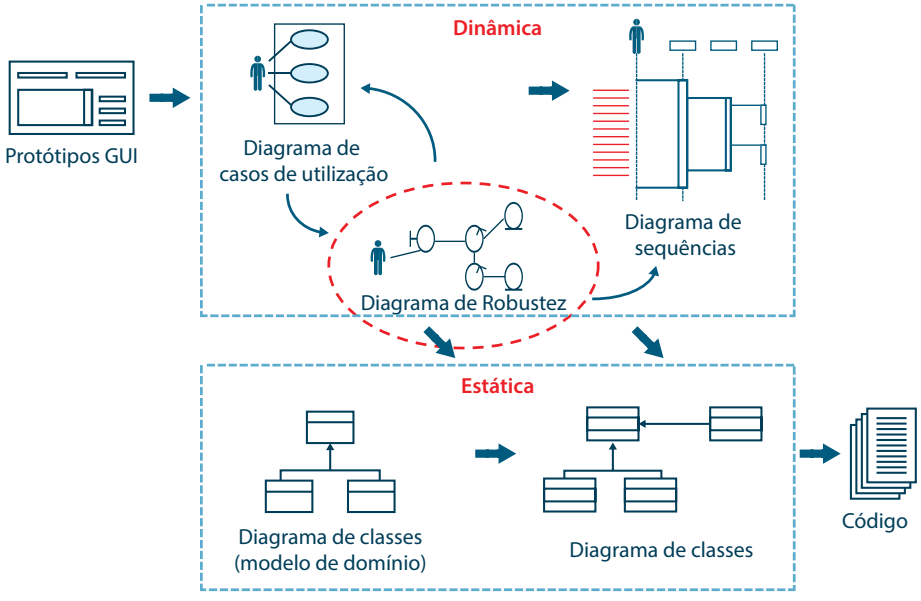


Figura 13 - Visão geral do ICONIX
Fonte: adaptado Rosenberg (2007, pg. 34).

Diagrama de Robustez é um gráfico híbrido entre o Diagrama de Classes e o Diagrama de Atividades da UML. E possui três tipos de estereótipos: objetos de fronteira/interface (*Boundary*), objetos de Entidade (*Entity*) e objetos de Controle (*Control*), conforme imagem a seguir.

Figura 14 – Simbologia do Diagrama de Robustez
Fonte: adaptado Sbrocco (2012, pg. 175).

Segundo Sbrocco, o Iconix Process possui uma característica importante:

> [...] característica exclusiva chamada "rastreabilidade dos requisitos" (*Traceability of Requirements)*. Essa rastreabilidade permite por meio de seus mecanismos, verificar em todas as fases se os requisitos estão sendo atendidos, à medida que o projeto vai sendo desenvolvido (SBROCCO, 2012, pg. 174).

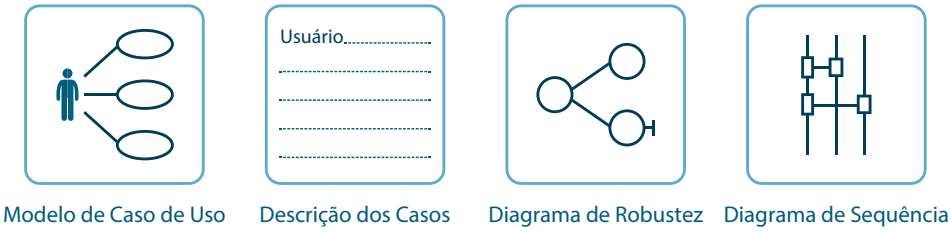


Figura 15 - Rastreabilidade dos Requisitos
Fonte: adaptado Rosenberg (2007, pg. 38).

ESTRUTURA DO ICONIX PROCESS

- **Análise de Requisitos:** etapa em que se identificam os requisitos, os Casos de Uso, objetos e onde se desenvolve de protótipos de interface. Para iniciar a Análise de Requisitos se usa um levantamento de documentos de regras de negócio, entrevistas e questionários aos usuários etc.
- **Análise e desenho preliminar:** etapa onde é descrito os casos de uso usando cenários e Diagrama de Classes.
- **Desenho:** fase onde é especificado o comportamento do Diagrama de Classes o mais detalhado possível.
- **Implementação:** etapa onde é especificado o Diagrama de Componentes e de instalação, onde é escrito o código, realizado os testes unitários, testes de integração e de aceitação.

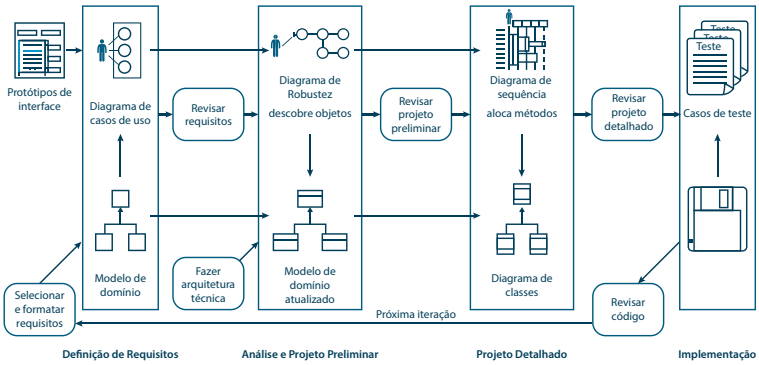


Figura 16 – Iconix Process
Fonte: Sousa (2014, pg. 35).

REFLITA

Os processos usados para desenvolver um projeto de software têm a maior importância na qualidade do software produzido e na produtividade alcançada pelo projeto.

(Cristina Bona)

SAIBA MAIS

O ICONIX é um processo simplificado que unifica conjuntos de métodos de orientação a objetos em uma abordagem completa, com o objetivo de dar cobertura ao ciclo de vida. [...] o ICONIX tem como base responder algumas questões fundamentais sobre o software. [...] As questões e as técnicas são:

1. Quem são os usuários do sistema [...] e o que eles estão tentando fazer? Utilizar **casos de uso**.

2. O que são, no "mundo real" (chamado domínio de problema), os objetos e as associações entre eles? Utilizar **diagrama de classe** de alto nível.

3. Que objetos são necessários para cada caso de uso? Utilizar **análise de robustez**.

4. Como objetos estão colaborando e interagindo dentro de cada caso de uso? Utilizar **diagrama de sequência e de colaboração**.

5. Como será manipulado em tempo-real aspectos de controle? Utilizar **diagrama de estado**.

6. Como realmente será construído o sistema em um nível prático? Utilizar **diagrama de classe** de baixo nível.

Fonte: adaptado de BONA (2002).

CONSIDERAÇÕES FINAIS

Prezado(a) Aluno(a)! Nesta unidade você aprendeu sobre alguns modelos de desenvolvimento de software, seus conceitos, suas características e como são utilizados nas empresas.

Aprendemos que o Processo Unificado (UP) é uma metodologia usada para gerenciar o desenvolvimento de projetos de software e que garante a qualidade do que está sendo construído, que atende as necessidades do cliente, dentro do prazo e custo estipulados para o projeto e que ele utiliza a UML como ferramenta para especificação de sistema e que ele pode ser utilizado em conjunto com outras metodologias, como o XP e o Scrum,

Estudamos também a metodologia Extreme Programming (XP), suas características, seus valores e princípios que caracterizam o projeto que desenvolvido usando esse modelo. Também aprendemos que as práticas do XP ajudam a garantir um ciclo de desenvolvimento fortemente dependente do sistema e que é uma metodologia com um conjunto bem definido de regras e que oferece condições para os desenvolvedores lidarem com as mudanças no projeto de forma eficiente.

Outra metodologia abordada foi a FDD (Feature Driven Development) ou Desenvolvimento Guiado por Funcionalidades, que é usada para o gerenciamento e desenvolvimento de software e que ela combina as melhores práticas do movimento ágil com Engenharia de Software Orientada a Objetos. Considerada uma metodologia apreciada pelos desenvolvedores, pelo conjunto de regras e técnicas fáceis e que mostram resultados rápidos.

Outra metodologia estudada foi a DSDM (Dynamic Systems Development Methodology) que é centrada em estabelecer os recursos, tempo fixo, funcionalidades que atendam os prazos estipulados.

E por último, foi estudada a metodologia ágil Iconix Process, que utiliza a modelagem dirigida por Casos de Uso com o objetivo de estudar o comportamento do sistema sob o ponto de vista do cliente. É uma metodologia prática e simples, mas poderosa e eficaz.

Vamos continuar? Então, vamos para a próxima unidade e bons estudos.

ATIVIDADES

1. O UP possui cinco fases que fornecem diretrizes em todos os projetos, com definição de tarefas e responsabilidade. **Em ordem cronológica, quais são as fases definidas no Processo Unificado (UP)?**

   a) Parte superior do formulário.

   b) Concepção, Requisitos, Implementação e Testes.

   c) Concepção, Elaboração, Construção e Implantação.

   d) Concepção, Elaboração, Construção e Transição.

   e) Elaboração, Concepção, Construção e Transição.

   f) Elaboração, Construção, Implementação e Transição.

2. Marque com V para a afirmativa verdadeira e com F para a afirmativa falsa, sobre Extreme Programming (XP):

( )Tenta resolver problemas e muitos que surgem ao longo do desenvolvimento do projeto, com o uso de ferramentas e de alguns recursos como desenvolvimento iterativo e arquitetura baseada em componentes.

( )Os envolvidos no projeto apreciam o XP porque ele provê resultados significativos mais cedo e possui um conjunto de regras fáceis de entender.

( )Ajuda a lidar com mudanças constantes e fornece recursos que ajudam a amenizar os problemas que ocorrem nestes casos.

3. Assinale a opção com a sequência CORRETA:

a) V, F, V.

b) F, V, V.

c) V, V, V.

d) F, F, V.

e) F, F, F.

3. FDD é uma metodologia ágil para gerenciamento e desenvolvimento de software, criada em 1997 num grande projeto em Java para um banco em Singapura. FDD é uma metodologia objetiva com processos bem definidos e integrados, com duas fases principais. **Explique essas duas fases principais e explique o que é uma *feature*?**

ATIVIDADES

4. Existem cinco fases básicas em relação ao processo do DSDM antecedidas por uma fase de pré-projeto e precedidas pelo pós-projeto. **Marque a opção que <u>NÃO</u> representa uma fase do DSDM.**

   a) Estudo da Viabilidade, Estudo de Negócio, Modelo de iteração funcional, Projeto e Construção de iteração, Implantação.

   b) Estudo da Viabilidade, Estudo de Negócio, Modelo de iteração incremental, Projeto e Construção de iteração, Implementação.

   c) Estudo da Confiabilidade, Estudo de Negócio, Modelo de iteração funcional, Projeto e Construção de iteração, Implementação.

   d) Estudo da Viabilidade, Estudo de Negócio, Modelo de iteração funcional, Projeto e Construção de Estados, Implementação.

   e) Estudo da Viabilidade, Estudo de Negócio, Modelo de iteração funcional, Projeto e Construção de iteração, Implementação.

5. Estudamos alguns modelos de desenvolvimento de software, entre eles, a metodologia Iconix Process, que diz respeito a uma modelagem dirigida por casos de uso e que utiliza alguns diagramas UML em seu processo. **Com base nisso, como podemos descrever a rastreabilidade dos requisitos na metodologia Iconix Process?**

MÉTODO ÁGIL XP (EXTREME PROGRAMMING)

Como o emprego dos métodos para desenvolvimento de software tem se tornado mais popular, existe uma grande demanda, pela indústria, da introdução de práticas de desenvolvimento de métodos ágeis. Essa tendência pode ser encontrada no método *Extreme Programming (XP)* como um método popular, despertando interesse tanto na área acadêmica quanto nas comunidades de programação.

Apesar disso, o XP ainda não é uma realidade na grade curricular das universidades, o que, de certa forma, impede o seu crescimento e a sua aplicação no mercado.

Contudo, além de os métodos tradicionais de desenvolvimento de software terem o foco voltado para a documentação, necessitam de requerimentos completos e fixos, o que os torna uma metodologia pesada e não flexível. Foi contrapondo a esse cenário que surgiu o *Extreme Programming* – uma metodologia ágil, que visa um rápido desenvolvimento, atende às reais necessidades do cliente e, ainda, permite modificações, à medida que novas necessidades apareçam.

Esse foco no desenvolvimento ágil que o XP traz, é de grande interesse para a indústria, podendo ocasionar inúmeros benefícios ao seu processo produtivo.

Porém, atualmente, faltam desenvolvedores capacitados para trabalhar com esse método, gerando uma grande demanda de profissionais com esse perfil. Consequentemente, a inclusão de métodos ágeis de desenvolvimento, no currículo de referência das universidades, é de grande interesse para a indústria, fornecendo-lhe mão-de-obra qualificada e contribuindo para o desenvolvimento da metodologia.

O *Extreme Programming* é um modelo de desenvolvimento de software, criado em 1996, por Kent Bech, no Departamento de Computação da montadora de carros Daimler Crysler, ele possui muitas diferenças em relação a outros modelos, podendo ser aplicado a projetos de alto risco e com requisitos dinâmicos. O XP é um conjunto bem definido de regras, que vem ganhando um grande número de adeptos, por oferecer condições para que os desenvolvedores respondam com eficiência a mudanças no projeto, mesmo nos estágios finais do ciclo de vida do processo, devido à quatro lemas adotados por seus seguidores, que correspondem à quatro dimensões a partir das quais os projetos podem ser melhorados. São eles: Comunicação, Simplicidade, FeedBack e Coragem.

No entanto, o XP não deve ser aplicado a qualquer tipo de projeto. O grupo de desenvolvedores deve formar uma equipe de 2 a 10 integrantes, que devem estar por dentro de todas as fases do desenvolvimento. É necessário realizar vários testes, às vezes, alterar o projeto em decorrência destes. A equipe tem de ser bastante interessada e pró-ativa, para assegurar a alta produtividade e, o cliente, deve estar sempre disponível para tirar dúvidas e tomar decisões em relação ao projeto.

Seguindo os requisitos expostos, anteriormente, o XP poderá trazer inúmeros benefícios ao mercado, de forma que o processo de desenvolvimento se torne mais ágil e flexível.

Um desses benefícios é a agilidade no Planejamento *(Planning Games)* de não definir uma especificação completa e formal dos requisitos, ao contrário das metodologias tradicionais. Outro é a produção de sistemas simples que atendam aos atuais requisitos, não tentando antecipar o futuro e permitindo atualizações frequentes em ciclos bastante curtos.

A comunicação com o cliente, no XP, mostra-se mais intensa que nos métodos tradicionais, devendo o cliente estar sempre disponível para tirar as dúvidas, rever requisitos e atribuir prioridades, utilizando-se de sistemas de nomes, em vez de termos técnicos para facilitar a comunicação. As equipes devem manter-se integradas com os projetos, melhorando a comunicação e a produtividade. Uma outra característica importante do XP é que o código é sempre escrito em duplas, visando a melhorar a qualidade do código por um custo muito baixo, às vezes menor do que o desenvolvimento individual. O código deve estar padronizado, para que todos na equipe possam entender o que está sendo escrito e possa ser validado durante todo o desenvolvimento, tanto pelos desenvolvedores quanto pelos clientes, a fim de se saber se os requisitos estão ou não sendo atendidos.

A inclusão de metodologias de desenvolvimento ágil nos currículos de referência das universidades é de grande valia para a indústria, que busca maior produtividade com a utilização do XP, mas que esbarra na falta de desenvolvedores qualificados para aplicar tais metodologias.

Fonte: adaptado de Souza (2007).

MATERIAL COMPLEMENTAR

LIVRO

Métodos Ágeis para Desenvolvimento de Software

Rafael Prikladnicki, Renato Willi, Fabiano Milani.

Editora: Bookman

Sinopse: esse livro traz informações completas sobre conceitos e práticas ágeis. Reúne a visão de 23 profissionais de todo o país, agregando em uma única obra conhecimentos e experiências sobre o tema, de projetos pequenos a grandes, dos simples aos complexos, de empresas públicas às privadas.

LIVRO

Desenvolvimento Ágil de Software – Guia Prático

Tiago Palhoto.

Editora: Fca

Sinopse: nesta obra, de cariz essencialmente prático e baseada na experiência do autor, demonstra-se como aplicar as práticas mais recomendadas para o desenvolvimento ágil de software. Técnicas e frameworks como Scrum, eXtreme Programming (XP), Disciplined Agile Development e IBM® Rational Unified Process (RUP®), entre outras, são cuidadosamente reunidas e adaptadas de forma a oferecer uma única metodologia integrada que cobre todo o ciclo de desenvolvimento de um sistema de informação.

O livro descreve as principais fases de um projeto. Recorrendo a exemplos práticos que demonstram as principais atividades a executar e os artefatos a produzir em cada uma das fases, o autor aborda também a aplicação da metodologia com recurso a software próprio para o efeito.

NA WEB

Introdução ao FDD - Feature Driven Development

Veja neste artigo uma introdução ao modelo de desenvolvimento FDD - Feature Driven Development. Serão apresentadas as principais características deste modelo e como ele pode ser facilmente integrado ao Scrum, por exemplo. Artigo muito interessante. Para saber acesseo link disponível em: <http://www.devmedia.com.br/introducao-ao-fdd-feature-driven-development/27971>.

Metodologia ágil FDD

Artigo que fala sobre FDD (Feature-Driven Development) que significa desenvolvimento guiado a funcionalidades, assim como o XP, ASD, Scrum e AUP, faz parte das metodologias ágeis originais, sendo esse um modelo incremental e iterativo do processo de desenvolvimento de software que tem como lema Resultados frequentes, tangíveis e funcionais. Fique por dentro e acesse o link disponível em: <http://www.leandromtr.com/gestao/metodologia-agil-fdd/>.

REFERÊNCIAS

AMUI, S. F. **Processos de Desenvolvimento de Software**. Rio de Janeiro: SESES, 2015.

BONA, C. **Avaliação de Processos de Software:** Um Estudo de Caso em XP e Iconix. Florianópolis: Engenharia de Produção da Universidade Federal de Santa Catarina, 2002.

CUNHA, C. R.; FILIPAKIS, C. D. Proposta de Utilização de FDD e APF para Melhoria do Processo de Software. In: Encontro de Computação e INFORMÁTICA DO TOCANTINS, 2012. Palmas. **Artigo**. Palmas. CEULP/ULBRA, 2012.

LUNA, A. J. H. de O. **MAnGve:** Um modelo para governança ágil em tecnologia da informação e comunicação. Recife: Universidade Federal de Pernambuco, 2009.

MARTINS, J. C. C. **Técnicas para Gerenciamento de Projetos de Software.** Rio de Janeiro: Brasport, 2007.

PRESSMAN, R.; MAXIM B. R. **Engenharia de Software** – Uma abordagem profissional. 8. Ed. Porto Alegre: AMGH, 2016.

SBROCCO, J. H.T. C.; MACEDO, P. C. de. **Metodologias Ágeis:** Engenharia de Software sob medida. 1 ed. São Paulo: Érica, 2012.

SOUZA, L. M. de. Método ágil XP (extreme programming). **Revista Eletrônica da FIA**, Vol. III N. 3 Jul-Dez / 2007.

SOUSA, T. C. de. **Um Processo de Desenvolvimento Orientado a Objetos com Suporte à Verificação Formal de Inconsistências**. São Paulo: Universidade de São Paulo, 2014.

TEIXEIRA, D. D.; PIRES, F. J. A.; PINTO, J. P. G. de S.; SANTOS, T. A. G. P. **DSDM** – Dynamic Systems Development Methodology. Portugal: Faculdade de Engenharia da Universidade do Porto, 2005.

REFERÊNCIA ON-LINE

[1]Em: <http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=1227>. Acesso em: 12 abr. 2017.

[2]Em: <http://heptagon.com.br/fdd/>. Acesso em: 12 abr. 2017.

GABARITO

1. d) Concepção, Elaboração, Construção e Transição

2. D. F, F, V.

3. As duas fases principais são: **Concepção e Planejamento**: fase para pensar um pouco antes de fazer (tempo estimado de 1 a 2 semanas) e a fase de **Construção**: fase para fazer de forma iterativa (tempo de iterações de 2 semanas). *Feature é* uma lista de requisitos funcionais, considerando somente aqueles de valor para o negócio e que podem ser entendidos pelos clientes e usuários e depois de identificados, são utilizados para planejar e gerenciar o projeto.

4. e) Estudo da Viabilidade, Estudo de Negócio, Modelo de iteração funcional, Projeto e Construção de iteração, Implementação

5. Rastreabilidade dos requisitos (*Traceability of Requirements)* é uma característica exclusiva que rastreabilidade permite por meio de seus mecanismos, verificar em todas as fases se os requisitos estão sendo atendidos, à medida que o projeto vai sendo desenvolvido.

Professora Esp. Janaina Aparecida de Freitas

INTRODUÇÃO AO GERENCIAMENTO ÁGIL DE SOFTWARE

UNIDADE III

Objetivos de Aprendizagem

- Fornecer uma visão Geral do Scrum, sua origem e seus fundamentos.
- Apresentar os conceitos básicos do funcionamento do Scrum.
- Indicar a importância dos Papéis no Scrum e suas funções.
- Entender os objetivos das Cerimônias do Scrum.
- Compreender os artefatos do Scrum.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Visão Geral do Scrum
- Fundamentos Básicos do Scrum
- Funcionamento do Scrum
- Papéis no Scrum
- Cerimônias do Scrum
- Artefatos do Scrum
- Estudo de Caso

UniCesumar

INTRODUÇÃO

Olá, aluno(a)! Esta unidade tem por finalidade apresentar uma visão geral do *Scrum*, seus conceitos básicos e como é o seu funcionamento. A partir de um pouco de história sobre o *Scrum,* passaremos a entender porque é considerada uma metodologia ágil e como ela foi fortemente influenciada pela indústria japonesa que utilizavam de equipes pequenas e multidisciplinares em seus projetos.

Vamos aprender porque a metodologia *Scrum* tem sido muito usada em variados segmentos e como sua característica iterativa e incremental permite usá-la em qualquer desenvolvimento de produto ou gerenciamento de *software*.

Aprenderemos como o *Scrum* trabalha com o conceito de *Sprints*, conhecidas como corrida de velocidade e como essas representam iterações de trabalho com duração de períodos curtos de tempo, geralmente de duas a quatro semanas.

Apresentaremos a importância dos Papéis no *Scrum* e quais as funções que cada um representa nessa metodologia. O *Scrum* estabelece um conjunto de práticas que devem ser seguidas pela equipe, conforme seus papéis: *Product Owner, Scrum Master* e o *Team*.

Procuraremos entender os objetivos e o funcionamento das cerimônias do *Scrum*. As cerimônias são reuniões que devem ser realizadas em momentos diferentes durante o desenvolvimento de uma *Sprint*. Resumem-se no Planejamento da *Sprint*, na Reunião Diária, na Revisão da *Sprint* e na Retrospectiva da *Sprint*.

A partir do resultado das cerimônias são gerados os artefatos do Scrum, que são documentos que são produzidos em diferentes momentos da *Sprint* e se resumem em: *Product Backlog, Sprint Backlog* e o *Burndown Chart*.

Por fim, vamos aprender sobre os conceitos vistos sobre o Scrum em um Estudo de Caso, mostrando um resumo sobre como implementá-lo e como começar a usá-lo em um projeto.

Preparado para continuar? Então, vamos seguir em frente. Boa leitura e bons estudos!

VISÃO GERAL DO SCRUM

As metodologias consideradas ágeis, assim como *Scrum*, são fortemente influenciadas pelas práticas da indústria japonesa, como as adotadas pelas empresas Toyota e Honda (SBROCCO, 2012). A origem do termo *Scrum* surgiu de um famoso artigo em 1986, escrito por Takeuchi e Nonaka, intitulado *"The new product development game"* - "O novo jogo do desenvolvimento de produtos", em que descreveram que equipes de projeto pequenas e multifuncionais produzem resultados melhores (PHAM, 2011).

O uso dessa metodologia *Scrum*, conforme Pressman (2016), vem de uma formação que ocorre durante uma partida de *rugby*. Essa formação ocorre após uma parada ou quando a bola sai de campo, e o técnico reúne a equipe de jogadores.

A metodologia *Scrum* foi desenvolvida em 1990 por Jeff Sutherland e Ken Schwabe e, segundo eles, *Scrum* é um *framework* para desenvolver e manter produtos complexos. Uma metodologia ágil de *software* que concentra as atenções no produto final com um rápido desenvolvimento e nas interações dos indivíduos.

Essa metodologia segue os princípios do Manifesto Ágil e conforme Pressman (2016, pg, 78) "são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades metodológicas: requisitos, análise, projeto, evolução e entrega".

O *Scrum* é uma metodologia para construir *software* incrementalmente em ambientes complexos, em que os requisitos não são claros ou que mudam frequentemente (BROD, 2013). A filosofia do *Scrum* se baseia em um processo empírico, onde se um processo é totalmente definido, ele pode ser repetido e os resultados podem ser previsíveis e pode ser automatizado. Mas caso, seja um processo caótico e não repetitivo, requer um controle constante e medições (SBROCCO, 2012).

O *Scrum* possui algumas características consideradas importantes, como: uma flexibilidade de resultados e prazos, trabalha com equipes pequenas, uso de revisões frequentes, colaboração dos interessados e segue a orientação a objetos (SBROCCO, 2012).

FUNDAMENTOS BÁSICOS DO SCRUM

Conforme Sbrocco (2012, pg. 161) sobre o *Scrum* ser "conhecido pelo fato de estruturar seu funcionamento por ciclos, chamados de ***sprints*** que representam iterações de trabalho com duração variável". As tarefas desenvolvidas dentro de um s*print* são adaptadas ao problema pela equipe *Scrum* e são realizadas em um período curto de tempo enquanto o projeto se desenvolve.

Conforme Sbrocco (2012), não pode ocorrer nenhuma mudança durante uma *sprint*, pois o produto é projetado, codificado e testado durante a *sprint*. O *Scrum* também estipula um conjunto de práticas e regras que devem ser seguidas pela equipe. No *Scrum* temos três **papéis** importantes: *Product Owner*, *Scrummaster* e *Team* (BROD, 2013).

No *Scrum* também ocorrem às **cerimônias** e são divididas em quatro: *Daily Meeting ou Daily Scrum*, a *Sprint Review*, o *Sprint Planning* e *Sprint Retrospective*. A partir dessas cerimônias são gerados os **artefatos** que os documentos *Product Backlog*, *Sprint Backlog* e o *Burndown Chart* (SBROCCO, 2012).

O *Scrum* reconhece como princípio básico que os clientes podem mudar de ideia durante o desenvolvimento do projeto e, por isso, adota uma abordagem empírica, ou seja, aceitar que o problema do cliente pode não ser totalmente entendido, mostrando atitude**,** (SBROCCO, 2012).

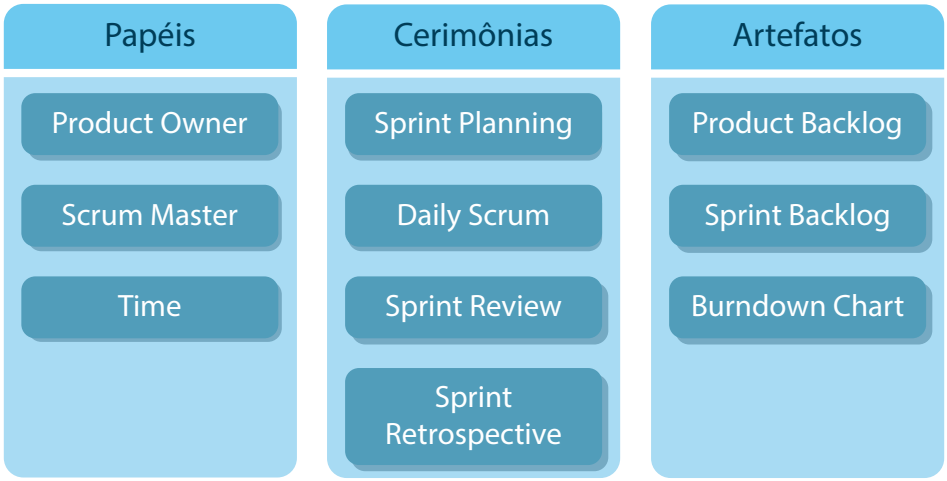


Figura 1 - Fundamentos básicos do *Scrum*

Ao longo da Unidade III, será explicado com mais detalhes cada um dos fundamentos básicos do *Scrum* e suas funções dentro do ciclo de desenvolvimento da metodologia.

FUNCIONAMENTO DO SCRUM

Para entender o funcionamento da metodologia *Scrum*, precisamos conhecer o fluxo geral do processo, que será apresentado na próxima figura, e seus conjuntos de atividades de desenvolvimento.

Tudo começa com o ***Product Backlog*** (*Backlog* do Produto) que é uma lista de prioridades dos requisitos ou funcionalidades para o projeto. O termo *backlog* é um histórico (*log*) de trabalhos realizados em um determinado período de tempo. Também são estimados custos, riscos e definidas as ferramentas de desenvolvimentos, definida a equipe e datas de entrega para os resultados, (SBROCCO, 2012).

Após o *product backlog* ser definido, segundo Sbrocco (2012), a equipe passa a realizar a ***Sprint Backlog***, que é uma lista de funcionalidades que serão realizadas no próximo *Sprint* e onde são definidas as funções de cada membro da equipe.

Ao falarmos em *Sprint*, pensamos em planejamento e em itens do *product backlog,* que são os requisitos do projeto e que serão transformados em funcionalidade dentro da próxima *Sprint* para entrega ao cliente. Mas antes disso, é necessário se preparar para a *Sprint*, pois para se trabalhar nos itens, é preciso fazer o levantamento junto ao cliente, depois analisá-los, procurar entendê-los, além de fazer o detalhamento desses itens.

Após o final de cada *sprint* um incremento do produto é apresentado ao cliente e, caso surjam defeitos, deve ser incluído ao *backlog* do produto. Conforme o ciclo de desenvolvimento ocorre, são adotados mecanismo de controle do *Scrum*, por exemplo, controle de funcionalidades que não foram entregues, mudanças que podem ocorrer devido a defeitos corrigidos, problemas técnicos e controle de riscos, (SBROCCO, 2012).

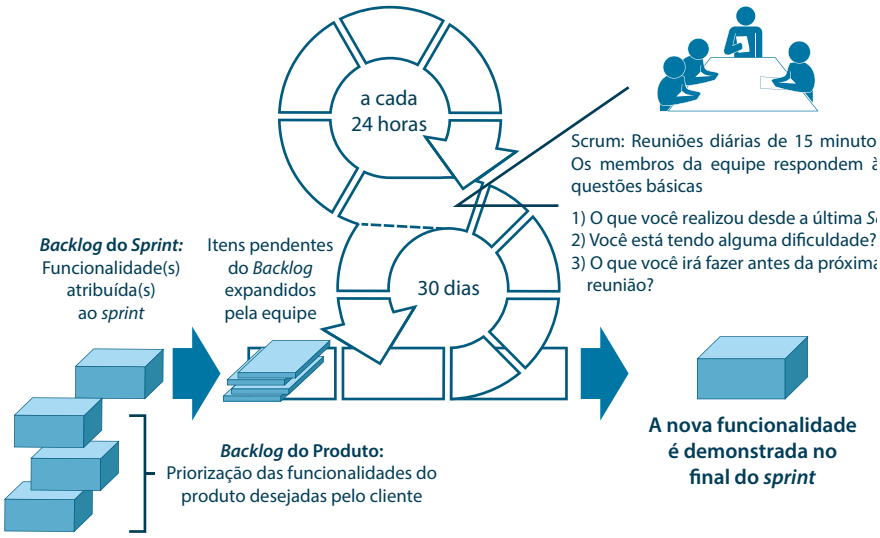


Figura 2 - Fluxo do Processo Scrum
Fonte: Pressman (2016, p. 78).

PAPÉIS NO SCRUM

No método *Scrum*, a organização dos recursos humanos envolvidos no projeto é composta por três papéis: *Product Owner, Scrum Master* e o *Team*.

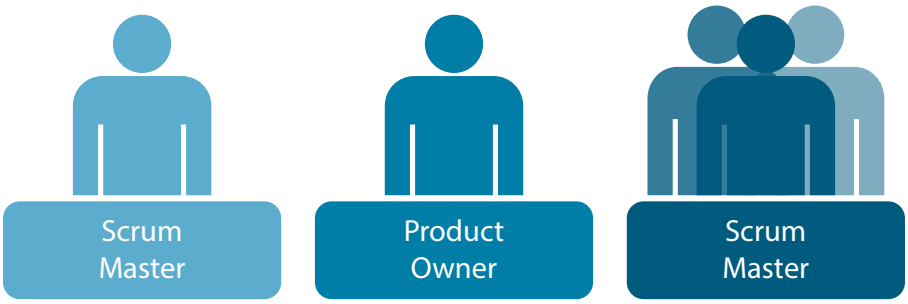


Figura 3 - Papéis do Scrum
Fonte: a autora.

Vamos conhecer cada um destes papéis e suas responsabilidades.

PRODUCT OWNER

Conforme Sbrocco (2012, pg. 163) o "*Product Owner* ("dono" do produto) representa o cliente e é responsável por garantir que a equipe Scrum agregue valor ao negócio". Trabalha junto com as partes interessadas e desempenha um papel de moderador entre o cliente e o *Team*.

De acordo com Pham (2011), o *Product Owner* é responsável por:

- Levantar os requisitos para o *Product Backlog*.
- Colabora com o *Scrum Master* e com a *Team* para o planejamentos do *Sprints*.

- Guia a equipe para alcançar as metas do planejamento de *releases* e *sprints*.
- Acompanha o desenvolvimento do projeto.
- Decide as datas de lançamento do produto.

SCRUM MASTER

Para Sbrocco (2012, pg. 164), "o *Scrum Master* (mestre *Scrum*) é o representante do cliente no projeto e também desempenha um papel importante de facilitator". Trabalha junto com o *Product Owner* protegendo o *Team* de possíveis distúrbios externos, ou seja, o *Scrum Master* é um guardião do processo *Scrum*.

Segundo Pham (2011), o *Scrum Master* é responsável por:

- Guiar e ajudar a treinar o *Team* para alcançar as metas dos lançamentos.
- Organiza a retrospectiva de *Sprint* para ajudar o *Team* a melhorar sua produtividade.
- Garante a colaboração entre as partes interessadas e a equipe e suas responsabilidades.
- Aplica os valores e as práticas *Scrum*.

TEAM

Team (time) ou Equipe *Scrum*, é responsável pelo desenvolvimento do projeto e por demonstrar os resultados dos *sprints* para o *Product Owner* e para o cliente, ao final de cada entrega do *sprint*. Conforme Sbrocco (2012, p. 164) "seus integrantes não devem possuir títulos e os integrantes do time só devem ser trocados após o término de uma *sprint*".

Pham (2011), afirma que o *Team* é responsável por:

- Responsável por fazer as estimativas necessárias.
- Transforma itens do *Product Backlog* em tarefas.
- Apresenta o produto ao cliente.
- Deve ser auto-organizada e multidisciplinar.

Segundo Brod, para entender melhor as funções e responsabilidade de cada um dos papéis no *Scrum*, podemos resumi-los da seguinte forma:

> [...] o coordenador geral do projeto é o *Scrum Master*, responsável por garantir a aplicação da metodologia, podendo atuar como o representante dos desejos do "dono do projeto", o *Product Owner*, quando ele não está presente. A tarefa primordial do *Scrum Master*, porém, é a de remover obstáculos, independentemente de sua natureza. Uma equipe *Scrum* terá membros com especialidades variadas, de acordo com a necessidade do projeto. Os membros da equipe discutem entre si e se auto gerenciam. Não há níveis hierárquicos dentro de uma equipe. Dentro de cada *sprint* os membros da equipe não podem ser substituídos (BROD, 2013, pg. 49).

CERIMÔNIAS DO SCRUM

A equipe *Scrum* possui diversas responsabilidades e uma delas é participar das cerimônias. As cerimônias são reuniões ou eventos que ocorrem em determinados momentos diferentes da *Sprint,* (SBROCCO, 2012). A seguir, falaremos sobre as cerimônias (figura 4) que são divididas em quatro: *Sprint Planning*, *Daily Meeting ou Daily Scrum*, a *Sprint Review*, e *Sprint Retrospective* e a participação da equipe em cada uma delas.

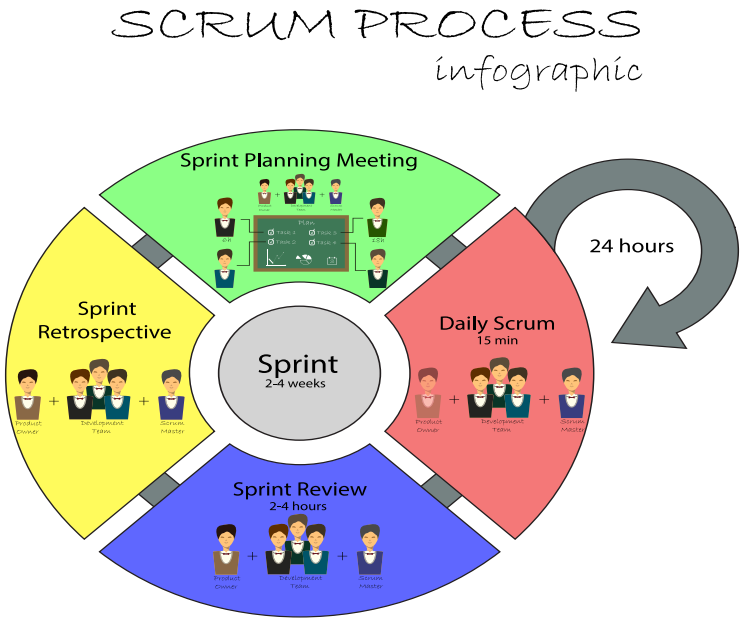


Figura 4 - Cerimônias do *Scrum*

SPRINT PLANNING OU PLANEJAMENTO DA *SPRINT*

Conforme Sbrocco (2012, p. 165), *sprint planning* ou planejamento da *sprint,* "é a primeira reunião do projeto e todos precisam participar; deve ter uma duração de no máximo 8 horas. Esta é a reunião em que o *Produtc Owner* planeja e elabora a lista de prioridades". Essa reunião deve ser dividida em duas partes:

Parte 1: o *Product Owner* define suas prioridades com base na história do usuário (requisitos), selecionando os itens do *backlog* que serão desenvolvidos e qual o objetivo da *sprint*. Conforme Pham (2012), esta primeira parte da reunião responde a questão "o quê" deve ser feito.

Parte 2: a equipe de desenvolvimento define a *sprint backlog*, que é um documento que contém as tarefas necessárias para cumprir a meta, deduz o quanto de tempo (em horas) será levada para concluí-las e transformá-las em incrementos de produto, que sejam potencialmente entregáveis.

Brod apresenta que (2013, p. 54), "o resultado da reunião será o objetivo do *sprint* (o que será produzido) e o *sprint backlog* (a lista de tarefas específicas deste *sprint*)". Assim, ao final da reunião teremos o que cada *sprint* terá como tarefa a ser desenvolvida que deverão ser passadas a equipe de desenvolvedores.

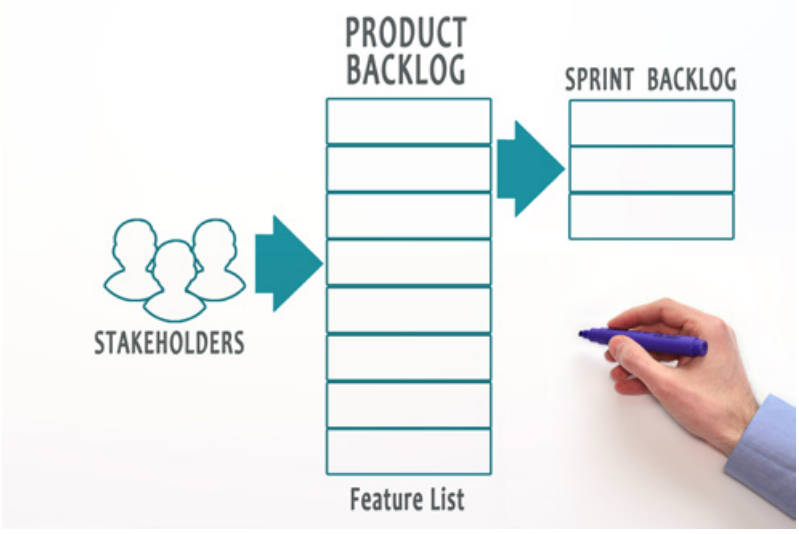


Figura 5: Planejamento da *Sprint*

DAILY MEETING OU *DAILY SCRUM* OU REUNIÃO DIÁRIA

É uma reunião importante e ao mesmo tempo simples, e segundo Sbrocco (2012, p. 166), "nessa reunião cada membro do time deve responder sobre o que já fez, sobre o que pretende fazer e se há algum impedimento para a conclusão da(s) tarefa(s)". É uma reunião que ocorre diariamente e deve ser rápida, no máximo 15 min, tendo como participantes o *Team* e o *Scrum Master*.

Para Brod (2013), é uma reunião em que cada participante deve responder a três perguntas:

1. O que você fez ontem?
2. O que você fará hoje?
3. Quais os obstáculos que impedem (ou podem impedir) seu trabalho?

Um dos benefícios da Reunião Diária é a visibilidade e o acesso a tudo o que está acontecendo pelos membros do projeto.

SPRINT REVIEW OU REVISÃO DA *SPRINT*

Conforme Sbrocco (2012, p. 167), "é uma reunião de balanço sobre tudo o que foi feito durante uma *print*. Nessa reunião o time deve mostrar os resultados da *sprint* para o *Product Owner* e seus convidados". Nessa reunião, são mostrados somente os itens concluídos, ou seja, caso tenha faltado uma atividade, o item não deve ser apresentado.

Participam da reunião: *Product Owner, Scrum Master, Team* e convidados. Ela tem duração estimada de quatro horas e é sempre marcada no final da *sprint*. Após o encerramento da reunião, segundo Sbrocco espera-se alcançar os seguintes objetivos:

> [...] apresentar o que a equipe fez durante a *sprint;* Entregar o produto(software funcionando) ao *Product Owner* (geralmente uma demo da parte implementada). Após a apresentação, o *Product Owner* conversa com seus convidados e tem o direito de aceitar ou rejeitar a *sprint* com base no que foi apresentado. Qualquer necessidade de mudança ou inserção de novas *features* (funcionalidades) será incorporada ao *Product Backlog* em um momento oportuno, além de priorizada novamente (SBROCCO, 2012, pg. 167).

Durante a Revisão da *Sprint*, é avaliado o projeto com base nos objetivos da *Sprint*, que foram determinados durante o Planejamento da *Sprint*. É verificado se o *Team* completou cada um dos itens do *Product Backlog* e se o objetivo geral da *Sprint* foi atingido.

SPRINT RETROSPECTIVE OU RETROSPECTIVA DA *SPRINT*

A reunião Retrospectiva da *Sprint* tem por objetivo verificar o que foi bom e o que pode ser melhorado na *sprint*. Para Brod (2013), é uma reunião em que se lava a roupa suja, ou seja, onde se procura entender o que deu errado e o que pode ser ajustado nas *sprints* para melhorar.

Participam dessa reunião: o *Team* e o *Scrum Master*, o *Product Owner* pode participar, caso seja convidado. Ela acontece após a reunião de revisão da *Sprint* e possui uma duração aproximada de três horas. Segundo Sbrocco (2012, p. 167), durante a reunião "o *Scrum Master* deve tomar nota de tudo e o time deve priorizar os itens apontados em uma ordem de mudança. A retrospectiva é uma excelente forma de garantir a melhoria contínua do processo".

SAIBA MAIS

Durante o planejamento da sprint podemos utilizar uma técnica que estima o tamanho do trabalho a ser realizado, aplicada a uma dinâmica de grupo conhecida como "pôquer do planejamento". Essa técnica dá a impressão de que os jogadores (time de desenvolvimento) estão jogando cartas, como num jogo de pôquer. Como vimos, no SCRUM não pensamos em tempo, ou seja, a duração das tarefas, mas no tamanho delas. Então, para que possamos definir esse tamanho, a equipe senta ao redor de uma mesa e um dos integrantes apresenta para todos uma "história", que está relacionada a uma funcionalidade que deve ser desenvolvida. Após o time de desenvolvimento entender claramente o objetivo da história, cada membro do time escolhe uma carta, que foi previamente distribuída a todos, e coloca na mesa virada para baixo. As cartas apresentam valores numéricos relacionados com o grau de complexidade da funcionalidade a ser desenvolvida, sob o ponto de vista de cada membro da equipe. O objetivo é obter um consenso relacionado ao tamanho do desenvolvimento da funcionalidade.

Fonte: adaptado de Sbrocco, 2012.

ARTEFATOS DO SCRUM

A partir das cerimônias, são gerados os **artefatos** que são documentos produzidos em diferentes momentos de uma *Sprint*. Os artefatos são: *Product Backlog*, *Sprint Backlog* e o *Burndown Chart*.

PRODUCT BACKLOG

O *Product Backlog* é um documento que segundo Sbrocco (2012, p. 168) "representa a visão do produto de forma modular, contendo todos os itens que devem ser desenvolvidos durante o projeto". Ou seja, é uma lista de funcionalidades com prioridades que é feita no início do projeto com o objetivo de esclarecer o que deve ser entregue ao cliente. Os itens do *Product Backlog* devem ser descritos de forma simples e de fácil entendimento para o cliente e para o *Team* e são criados e mantidos pelo *Product Owner*.

SPRINT BACKLOG

Artefato proveniente do Planejamento da *Sprint* e que conforme Sbrocco (2012, p. 168) "representa todas as tarefas que devem ser desenvolvidas durante uma *Sprint* ou iteração". Para um melhor rendimento da *Sprint, o Sprint Backlog* não pode sofrer alterações e é importante que o *Team* conclua o primeiro item, antes de começar o segundo, para evitar riscos na meta. Pode ser subdividida as tarefas entre equipes distintas e, com isso, ter várias *sprints* ocorrendo simultaneamente.

BURNDOWN CHART OU GRÁFICO BURNDOWN

Seu objetivo, conforme Sbrocco (2012, p. 171) é "mostrar o esforço restante para a conclusão da iteração, bem como mostrar o quão próximo ou distante o time está de atingir a meta". Com ele é possível verificar o quanto a estimativa inicial está correta ou não e, com isso, melhorar continuamente a cada novo projeto, (BROD, 2013).

SAIBA MAIS

Task Board

É um quadro utilizado para acompanhamento das sprints, principalmente durante as reuniões diárias. Por meio das informações contidas nesse registro, aliadas ao seu posicionamento no task board, torna-se possível a qualquer um observar o andamento do projeto, de maneira clara e intuitiva. As informações sobre as tarefas a serem realizadas são registradas no task board geralmente por "post-it" ou escritas diretamente em um quadro branco, previamente preparado para esta função. Quando o sprint se inicia, os integrantes do projeto decidem em comum acordo quais atividades irão desenvolver e normalmente registram-nas em pequenos papéis (post-it). Muitas equipes SCRUM fazem uso dos tradicionais "post-it" coloridos encontrados no mercado, nos quais escrevem o que devem fazer a caneta, de forma simples e objetiva.

Fonte: Sbrocco (2012).

REFLITA

Como podemos descobrir algumas regras simples que possam guiar as equipes a entrar em um estado mais produtivo, feliz, incentivador, divertido e arrebatador?

(Jeff Suttherland).

ESTUDO DE CASO

Para explicar melhor o funcionamento do Scrum, vamos a um Estudo de Caso.

Vamos imaginar uma empresa chamada TempoSite. Essa empresa tem um cliente chamado Sr. Dobro que quer lançar um site para a sua empresa. Assim, o gerente de projetos (Product Owner) faz um *briefing* e levanta junto ao Sr. Dobro todas as necessidades que ele precisa para o site (é criado a *Users Stories*).

O gerente se reúne (Reunião de Planejamento) com a equipe de desenvolvimento (*designers*, *front-end* e *back-end*) para juntos definirem todas as tarefas necessárias para produzir o projeto e formalizar tudo em uma lista de tarefas (Product Backlog - exemplificado na Figura 6).

O *Scrum Master* (neste caso, imagine que seja você) fica encarregado de organizar as reuniões e garantir que todos entendam as tarefas. A equipe decide democraticamente que vai entregar a página inicial pronta (Objetivo) até o final da semana (*Sprint*).

A equipe então divide o objetivo em tarefas menores (*Sprint Backlog* - exemplificado na Figura 7) para criar o *layout,* desenvolver os códigos e implantar um sistema de administração de conteúdo e testar tudo em diversos *browsers*, para que não haja erros.

Cada um da equipe escolhe as tarefas que irá realizar, definindo um prazo de duração em horas para cada tarefa. Diariamente a equipe se reúne por 15 minutos para acompanhar o andamento do projeto (*Daily Meeting ou Daily Scrum ou* Reunião Diária).

No final da semana a equipe apresenta o resultado (*Sprint Review* ou Revisão da Sprint) e refletem sobre dificuldades e quais melhorias podem ser feitas na

próxima semana (*Sprint Retrospective* ou Retrospectiva da Sprint) e desenvolvendo o Burndown Chart (exemplificado na Figura 8).

O mais importante é que vocês entregaram um produto parcial, porém funcional (Incremento) para o cliente. A partir do *feedback*, a equipe decide qual vai ser a tarefa a ser realizada na próxima semana. E assim o ciclo do *Scrum* recomeça.

PRODUCT BACKLOG

				Concluído
Produto	Fábrica de Software	SM	<Nome do Scrum Master>	Em andamento
PO	<Nome do Product Owner>	Atualizado em	<Data>	Cancelado

ID	Prioridade	História de Usuário / Requisito / Item	Tipo	Quem	Estimativa	Status
100	Alta	Captação de recursos para o projeto e contratação da equipe	Funcionalidade	CB	3	Concluído
101	Alta	Reunião de início do projeto e definição do primeiro *sprint backlog*	Funcionalidade	CB	2	Em andamento
102	Média	Definição de leiautes, interfaces, levantamento de necessidades e ferramentas adotadas	Funcionalidade	CB	2	Em andamento
103	Alta	Aquisição de equipamento "host", instalação, registro de domínio, implantação do CL	Correção	TBD	1	Em andamento
104	Baixa	Criação do sistema do projeto	Funcionalidade	DEVEL	1	Em andamento

Figura 6 - Exemplo de Product Backlog
Fonte: Adaptado de Brod (2013, p. 51).

Colaborador	Tarefas	Semana 1 (estimada)	Semana 1 (Realizada)	Tarefas	Semana 2 (estimada)	Semana 2 (Realizada)
Vinicius de Moraes	Desenho de interface	40	50	Aplicação do leiaute à interface	40	35
Tom Jobim	Orientação dos elementos que compõem o leiaute e adequação aos formatos padrões	20	30	Criação do paper prototype para etstar com o cliente	40	50
Baden Powel	Montagem da infra-estrutura	20	20	Criação da estrutura	10	15
Total		**80**	**100**		**90**	**100**

Figura 7 - Exemplo de *Sprint Backlog*
Fonte: Adaptado de Brod (2013, pg. 52).

Para uma *Sprint* de quatro semanas, com 160 horas que são consumidas, conforme a tabela abaixo:

Tabela 1- Consumo do Tempo durante uma *sprint*

SEMANA	HORAS RESTANTES	HORAS ESTIMADAS
0	160	160
1	90	100
2	55	60
3	20	25
4	0	0

Fonte: Brod (2013, pg. 60).

Nesse exemplo, foi considerado as horas totais das pessoas envolvidas no projeto, conforme a Figura 7. Começamos a semana com 160 horas e foi previsto o consumo de 40 horas na primeira semana, mas consumimos apenas 50 horas. O gráfico a seguir, mostra o quanto a estimativa inicial foi desviada, mas no final, o que havia sido prometido foi cumprido dentro do prazo estimado no *sprint*.

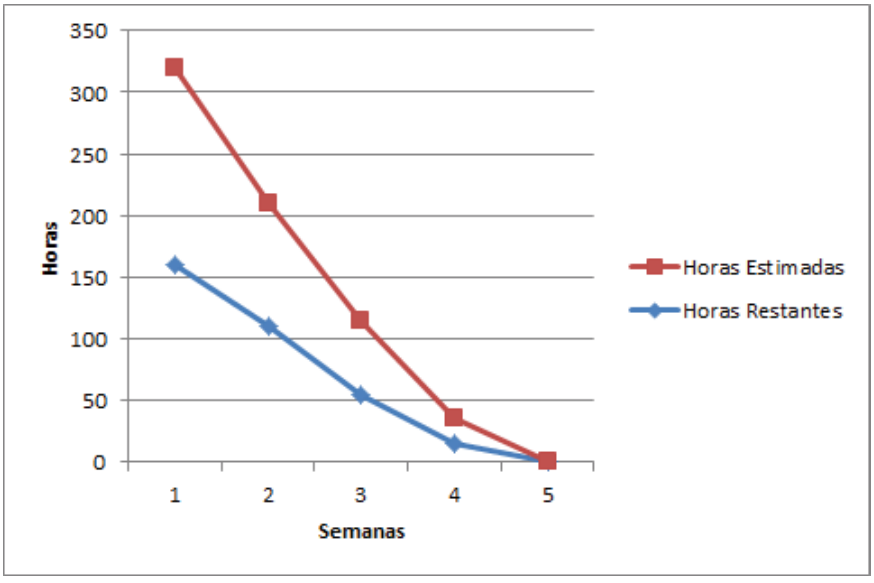


Figura 8 - Exemplo de *Burndown Chart*.
Fonte: Brod (2013, p. 60).

No gráfico, temos a coluna vertical que mostra a quantidade de esforços/horas, a coluna horizontal que mostra as semanas de uma sprint. A linha azul mostra o fluxo ideal de trabalho. Assim, quando o gráfico está acima da linha azul indica que o *Team* está longe da meta, quando o gráfico está em cima da linha azul indica que o *Team* está no fluxo ideal de trabalho e quando o gráfico está abaixo da linha indica que o *Team* está acima das expectativas.

ANOTAÇÕES

CONSIDERAÇÕES FINAIS

Olá, aluno(a)! Chegamos ao final de mais uma unidade, que aprendemos sobre a metodologia ágil *Scrum* e seus conceitos, fundamentos e funcionamento. Conhecemos um pouco da história sobre o *Scrum* e como foi fortemente influenciada pela indústria japonesa, que usavam equipes pequenas e multidisciplinares em seus projetos.

Aprendemos que a metodologia *Scrum* ajuda as empresas a simplificar o processo de desenvolvimento de um projeto, pois é flexível nos prazos, no resultado e possui revisões frequentes.

Na metodologia *Scrum* trabalhamos com o conceito de *Sprints*, ciclos que representam iterações de trabalho com duração variável, onde as tarefas desenvolvidas dentro de um s*print* são adaptadas e são realizadas em um período curto de tempo.

Um ponto importante que deve ser aprendido na metodologia *Scrum* é o conjunto de práticas que a equipe deve seguir, conforme os papéis e as funções que cada um representa nessa metodologia. No *Scrum o*s papéis se dividem em: *Product Owner, Scrum Master* e o *Team*.

As cerimônias no *Scrum* são reuniões que ocorrem em momentos diferentes durante uma *Sprint* e se resumem no Planejamento da *Sprint*, na Reunião Diária, na Revisão da *Sprint* e na Retrospectiva da mesma. Assim, com o resultado das cerimônias os artefatos do *Scrum* são gerados. Os artefatos são documentos que são produzidos em diferentes momentos da *Sprint* e se resumem em: *Product Backlog, Sprint Backlog* e o *Burndown Chart.*

Para Brod (2013), a metodologia *Scrum* exige projetos bem documentados e um conjunto de formalidades em sua prática, apesar do que muitos pensam sobre o *Scrum.*

Para exemplificar os conceitos vistos sobre o *Scrum*, foi usado um Estudo de Caso, mostrando um resumo sobre como implementá-lo e como começar a usá-lo em um projeto para uma hipotética empresa. Preparado para continuar? Então, vamos seguir em frente.

ATIVIDADES

1. No time de *Scrum* temos pessoas com diversas habilidades, como analistas, programadores, testadores e muitas outras que são comprometidas em realizar o trabalho que lhes foi proposto. Um time de *Scrum* possui várias responsabilidades, entre elas, de participar das Cerimônias. **Com base nisso, cite e explique quais são as cerimônias do *Scrum*.**

2. Conforme Brod (2013), o *Scrum* é um método de trabalho para desenvolvimento em tempos curtos e, para pequenas equipes, trabalha com o conceito de *Sprints*. **A partir disso, na metodologia *Scrum*, o que representa uma *Sprint*?**

3. Segundo Sbrocco (2012), após o final de cada *sprint* um incremento do produto é apresentado ao cliente e caso surjam defeitos, deve ser incluído ao *backlog* do produto. **Com base nisso, analise as opções abaixo:**

   I- O *Product Backlog* é uma lista de funções dos membros da equipe do *Scrum*.

   II- *Sprint Backlog* é uma lista de funcionalidades que serão realizadas no próximo *Sprint* e onde são definidas as funções de cada membro da equipe.

   III- Gráfico *Burndown* mostra o esforço restante para a conclusão da iteração e mostra se está próximo ou distante para que o time atinja a meta.

   IV- *Product Backlog* deve ser criado e mantido pelo *Product Owner*.

Assinale a alternativa correta:

   a. Apenas I e II estão corretas.

   b. Apenas II e III estão corretas.

   c. Apenas I está correta.

   d. Apenas II, III e IV estão corretas.

   e. Nenhuma das alternativas está correta.

4. O *Scrum,* segundo Sbrocco (2012), estabelece um conjunto de práticas e regras que devem ser cumpridas pela equipe, em seus respectivos papéis. **Quais os principais papéis utilizados na metodologia *Scrum*?**

   a. *Product Owner, SCRUM Master* e *Sprint*.

   b. *Product Owner, Daily SCRUM* e *Team ou equipe SCRUM*.

   c. *Product Backlog, SCRUM Master* e *Team ou equipe SCRUM*.

   d. *Product Owner, SCRUM Master* e *Sprint Backlog*.

   e. *Product Owner, SCRUM Master* e *Team ou equipe SCRUM*.

5. Na metodologia *Scrum,* o *Product Owner* é responsável por maximizar o valor do produto e o trabalho da equipe de desenvolvimento. O proprietário do produto é a única pessoa responsável pela manutenção do *Backlog* do produto. **Com base nisso, comente o papel do *Scrum Master* na metodologia *Scrum*.**

GESTÃO DE PROJETO COM SCRUM: UM ESTUDO DE CASO

O Scrum (nome derivado de uma atividade que ocorre durante um jogo de rugby) é um modelo ágil de processo que foi desenvolvido por Jeff Sutherland e por sua equipe no início da década de 1990 [...]. Originalmente, o Scrum foi desenvolvido para ser implementado em equipes de desenvolvimento de produtos de software. Porém, pode ser utilizado por qualquer empresa que necessite implementar processos de gerenciamento de projetos, tais como agências de publicidade, projetos de arquitetura e bancos [...].

O Scrum baseia-se em seis características: flexibilidade dos resultados; flexibilidade dos prazos; times pequenos; revisões frequentes; colaboração; orientação a objetos [...]. Esse método não requer ou fornece qualquer técnica específica para a fase de desenvolvimento, apenas estabelece conjuntos de regras e práticas gerenciais que devem ser adotadas para o sucesso de um projeto [...]. O Scrum não é um processo previsível, ele não define o que fazer em toda circunstância. Ele é usado em trabalhos complexos nos quais não é possível prever tudo o que irá ocorrer e oferece um framework e um conjunto de práticas que torna tudo visível. Isso permite aos praticantes do Scrum saber exatamente o que está acontecendo ao longo do projeto e fazer os devidos ajustes para manter o projeto se movendo ao longo do tempo visando alcançar os seus objetivos [...]. O Scrum é um framework dentro do qual pode-se empregar diversos processos e técnicas para desenvolver produtos complexos.

Pereira et al (2007) que relatam que o uso da agilidade traz vantagens como:

· Cria um ambiente propício para definição de mudanças de requisitos e inovação durante o ciclo de desenvolvimento do produto, assim como mais colaborativo e produtivo entre desenvolvedores e cliente, resultando em entregas mais rápidas de produto, melhor adaptados à realidade do cliente e com a qualidade esperada.

· Facilita o gerenciamento do projeto, uma vez que existem maior integração e comprometimento da equipe do projeto, que consequentemente se sente mais motivada: a moral da equipe é elevada.

· Reforça o planejamento constante do projeto, o que minimiza os riscos, considerando que o planejamento é mais importante do que o plano. Não se deve parar de planejar até que se tenha encontrado a satisfação do cliente com a entrega do produto.

· Valoriza a satisfação do cliente em primeiro lugar.

O ciclo do Scrum tem o seu progresso baseado em uma série de iterações bem

definidas, cada uma com duração de 2 a 4 semanas, chamadas Sprints. Antes de cada Sprint, realiza-se uma Reunião de planejamento (Sprint Planning Meeting) onde o time (equipe) de desenvolvedores tem contato com o cliente (Product Owner) para priorizar o trabalho que precisa ser feito, selecionar e estimar as tarefas que o time pode realizar dentro da Sprint [Pereira et al 2007]. A próxima fase é a Execução da Sprint. Durante a execução da Sprint, o time controla o andamento do desenvolvimento realizando Reu-

niões Diárias Rápidas (Daily Meeting), não mais que 15 minutos de duração, e observando o seu progresso usando um gráfico chamado Sprint Burndown. [Pereira et al 2007].

Ao final de cada Sprint, é feita uma revisão no produto entregue para verificar se tudo realmente foi implementado com a realização uma Reunião de Revisão (Sprint Review), onde o time demonstra o produto gerado na Sprint valida se o objetivo foi atingido. Logo em seguida, realiza-se a Reunião de Retrospectiva (Sprint Retrospective), uma reunião de lições aprendidas, com o objetivo de melhorar o processo/time e/ou produto para a próxima Sprint [Pereira et al 2007]. Há seis papéis identificáveis no Scrum que têm tarefas e finalidades diferentes durante o processo e suas práticas: Scrum Master, Product Owner, Scrum Team, o cliente, o usuário e o gerente. A seguir estes papéis são apresentados de acordo com as definições de Schwaber & Beedle (2007):

· Scrum Master: É um novo gerente introduzido ao Scrum. É responsável por garantir que o projeto é realizado de acordo com as práticas, valores e regras ao Scrum e que avança conforme o planejado.

· Product Owner: É oficialmente responsável pelo projeto, pelo gerenciamento, controle e por tornar visível a lista de Product Backlog. Ele é selecionado pelo

Scrum Master, cliente e gerente.

· Scrum Team: É o time do projeto que tem a autoridade para decidir as ações necessárias e organizá-las em ordem para atingir os objetivos de cada Sprint.

· Cliente: Participa nas tarefas relacionadas aos itens do Product Backlog para o sistema ser desenvolvido ou aprimorado.

·Usuário: É responsável por utilizar o produto quando este estiver em produção.

· Gerente: É responsável pela decisão final, juntamente com as normas e convenções a serem seguidas no projeto. O gerente também participa da definição das metas e dos requisitos.

Durante a Sprint, o time, de forma organizada, controla como as tarefas devem

ser executadas. Durante a Sprint não deve existir interferência externa, esse é um dos principais papéis do Scrum Master, blindar o time de qualquer desvio do objetivo traçado. O acompanhamento do progresso é feito realizando reuniões diárias (daily meeting). Todos participam, o Scrum Master e o time. Visitantes são bem vindos, mas devem ser apenas ouvintes, pois o daily meeting resume-se ao time [Pereira et al 2007].

Fonte: adaptado de Andrade et all (2012, on-line)[1].

MATERIAL COMPLEMENTAR

LIVRO

SCRUM - A Arte de fazer o dobro do Trabalho na Metade do Tempo
Jeff Sutherland.

Editora: Casa da Palavra.
Sinopse: a arte de fazer o dobro do trabalho na metade do tempo! É inegável a máxima de que o mundo e tudo a seu redor vêm sofrendo um processo de mudança contínuo cada vez mais rápido. Para aqueles que acreditam que deve haver uma maneira mais eficiente de se fazer as coisas, esse é um livro instigante sobre um processo de gestão que está mudando a maneira como vivemos.

LIVRO

Scrum em Ação - Gerenciamento e Desenvolvimento Ágil de Projetos de software
Andrew Pham e Phuong-Van Pham.

Editora: Novatec.
Sinopse: essa obra pretende fornecer um guia prático para equipes de projeto de software que desejam implantar uma estrutura de software Ágil usando o Scrum. Os autores buscam usar situações da vida real, elaboradas por praticantes corporativos. Descrevem também como extrair o máximo das equipes de projeto, abordando a maneira de como se comunicar com executivos por meio de termos financeiros, como usar uma técnica objetiva de estimativa e onde a arquitetura de software se encaixa dentro do Scrum. Um apêndice fornece estudos de caso sobre como dois produtos de software foram construídos e implantados com sucesso, usando as técnicas e conselhos descritos no livro.

NA WEB

Estória de usuário. Você saberia contar?
Artigo que fala sobre estórias de usuário, que é uma prática oriunda dos métodos ágeis, para levantar requisitos de forma ágil mais simples e mais eficiente. Uma estória de usuário pode ser caracterizada como uma curta e simples descrição da necessidade do cliente, usadas em metodologias ágeis como SCRUM e XP. Muito interessante. Vale a pena ler!
Acesse o link disponível em: <http://www.culturaagil.com.br/estoria-de-usuario-voce-saberia-contar/>.

O que é um Scrum Master e porque ele é importante?
Artigo que fala sobre o que é um Scrum Master, todo gerenciamento de projetos precisa ser eficiente e de qualidade. E isso pode ser mais facilmente conquistado com a figura do Scrum Master. Mas você sabe o que é um Scrum Master, qual o papel desse profissional e sua real importância na gestão de projetos? Entenda melhor a seguir. Para saber mais, acesse o link disponível em:
<http://www.projectbuilder.com.br/blog-pb/entry/conhecimentos/o-que-e-um-scrum-master-e-por-que-ele-e-importante>.

REFERÊNCIAS

BROD, C. **Scrum:** Guia Prático para Projetos Ágeis. São Paulo: Novatec, 2013.

SBROCCO, J.H. T. C.; MACEDO, P. C. de. **Metodologias Ágeis**: Engenharia de Software sob medida. 1 ed. São Paulo: Érica, 2012.

PHAM, A.; PHAM, P. **Scrum em Ação:** Gerenciamento e desenvolvimento Ágil de Projeto de Software. São Paulo: Novatec: Cengage Learning, 2011.

PRESSMAN, R. MAXIM, B. R. **Engenharia de Software – Uma abordagem profissional**. 8 ed. Porto Alegre: AMGH, 2016.

SBROCCO, J. H. T. C.; MACEDO, P. C. de. **Metodologias Ágeis:** Engenharia de Software sob medida. 1 ed. São Paulo: Érica, 2012.

ORTH, L.; DALFOVO O. **Scrum para Desenvolvimento de Projetos de Business Intelligence.** Universidade do Sul de Santa Catarina, 2016.

SUTHERLAND, J. **Scrum:** a arte de fazer o dobro do trabalho na metade do tempo. São Paulo: LeYa, 2014.

REFERÊNCIAS ON-LINE

[1]Em: <http://www.enucomp.com.br/2012/conteudos/artigos/scrum.pdf>. Acesso em: 17 abr. 2017.

GABARITO

1. As Cerimônias do *Scrum* são divididas em quatro: *Sprint Planning*, *Daily Meeting ou Daily Scrum*, a *Sprint Review*, e *Sprint Retrospective*.

Sprint Planning* ou Planejamento da *Sprint: é a primeira reunião do projeto com a participação de todos em que o *Product Owner* planeja e elabora a lista de prioridades a serem cumpridas no projeto.

***Daily Meeting ou Daily Scrum* ou Reunião Diária**: reunião em que cada membro responde sobre o que já fez, o que pretende fazer e se tem algum impedimento e quem participa são o *Scrum Master* e o *Team*.

***Sprint Review* ou Revisão da *Sprint*:** reunião de balanço para apresentar sobre tudo o que foi feito durante a *Sprint*.

***Sprint Retrospective* ou Retrospectiva da *Sprint*:** reunião com o objetivo de verificar o que houve de bom e o que pode a vir ser melhorado em uma *Sprint*.

2. As *sprints* (corrida de velocidade de curta distância) representam iterações de trabalho com duração variável, geralmente de duas a quatro semanas e estabelecem um conjunto de práticas e regras que devem ser cumpridas pela equipe que decidem quantas tarefas serão capazes de executar em duas semanas.

3. d) Apenas II, III e IV estão corretas.

4. e) *Product Owner, SCRUM Master* e *Team* ou *equipe SCRUM.*

5. O *Scrum Master* não é o líder da equipe, ele preocupa-se com o uso correto do processo SCRUM e a aplicação das suas práticas. Atua na definição de funcionalidades de acordo com seu valor para o cliente, planejando e elaborando em conjunto com o *Product Owner* uma lista de prioridades. O *Scrum Master* desempenha um papel mais de responsabilidade técnica na condução do projeto e protege a equipe, mantendo ela focada nas tarefas. para mantendo ela nas suas tarefas.

Professora Esp. Janaina Aparecida de Freitas

DESENVOLVIMENTO ÁGIL DE SOFTWARE

UNIDADE IV

Objetivos de Aprendizagem

- Apresentar uma visão comparativa e as diferenças entre as metodologias tradicionais e ágeis no gerenciamento de projetos.
- Analisar a qualidade de software nas metodologias ágeis.
- Considerações sobre testes de software nas metodologias ágeis SCRUM e XP.
- Entender as estimativas para o desenvolvimento de um projeto ágil e seus riscos.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- Metodologias Tradicionais X Metodologias Ágeis
- Qualidade de Software nas Metodologias Ágeis
- Testes de Software nas Metodologias Ágeis
- Estimativa para Desenvolvimento Ágil
- Gerenciamento de Riscos em Projetos Ágeis

UniCesumar

INTRODUÇÃO

Olá, aluno(a)! Nesta unidade, convido você a dar continuidade em nosso estudos onde será apresentada uma visão comparativa e as diferenças entre as metodologias tradicionais e ágeis no gerenciamento de projetos.

Entender as características das metodologias tradicionais e das ágeis irá ajudar na melhor escolha da metodologia para determinado projeto. O objetivo da comparação entre as metodologias é a redução de erros que possam comprometer a qualidade do produto e a satisfação do cliente.

Falaremos sobre qualidade de software e como ela é determinada pela qualidade dos processos que são usados durante a fase de desenvolvimento do software. Analisaremos se é possível manter a qualidade de software nas metodologias ágeis e qual a visão sobre a qualidade dos envolvidos no projeto.

Um dos objetivos de se controlar a qualidade de software é eliminar problemas como defeitos, falhas e erros e, para isso, deve-se investir em estratégias de teste de software. Apresentaremos nesta unidade algumas considerações sobre testes de software e como eles são importantes nas metodologias ágeis de desenvolvimento como o XP e o *Scrum*.

Além disso, aprenderemos sobre as métricas e medições nas metodologias ágeis. A métrica proporciona uma base por meio da qual a análise, o projeto, a codificação e teste podem ser conduzidos mais objetivamente e avaliados de maneira mais quantitativa em desenvolvimento de um projeto ágil.

Finalmente, para encerrar a nossa unidade, aprenderemos sobre gestão de riscos nas metodologias ágeis e como a simplicidade do processo, a comunicação constante nas reuniões diárias, o feedback do cliente e da equipe, com as entregas frequentes, ajudam a diminuir a complexidade dos sistema e com isso ajudam a detectar os riscos antecipadamente.

Preparado(a) para continuar com os estudos? Então, vamos seguir em frente. Ótima leitura!

METODOLOGIAS TRADICIONAIS X METODOLOGIAS ÁGEIS

As metodologias ágeis, conforme Sbrocco (2012) tem sido muito utilizadas como uma alternativa às abordagens tradicionais, por terem menos regras, menos burocracias e serem mais flexíveis, por permitirem ajustes durante o desenvolvimento do software.

O mercado de desenvolvimento de software está cada vez mais dinâmico e os negócios estão ficando cada vez mais competitivos, assim, quem possui maior acesso a informação de qualidade vai possuir vantagens sobre os concorrentes (SILVA et al, 2013).

As empresas enfrentam algumas dificuldades na hora de escolher uma metodologias disponíveis no mercado para o gerenciamento de projetos de software, conforme Silva et al o tipo de empresa pode influenciar:

> [...] as empresas de grande porte e entidades governamentais têm como prática o detalhamento de vários processos por relatórios, planilhas e gráficos, sendo comumente utilizada a metodologia tradicional para projeto de software.Por outro lado as empresas pequenas e médias encontram dificuldades em utilizar a metodologia tradicional, por ser muito caro a manutenção e controle, e o longo prazo para entrega. A metodologia ágil com custo baixo e entregas rápidas pode contribuir no desenvolvimento de projeto de software para qualquer tipo de empresa e ser agregada à metodologia tradicional. (SILVA et al, 2013, pg. 41).

Segundo Sbrocco (2012, pg. 183), "as metodologias tradicionais são mais "pesadas" e devem ser utilizadas em situações em que os requisitos do sistema são estáveis". Mas se os requisitos forem passíveis de mudanças e a equipe for pequena, é interessante utilizar as metodologias ágeis.

As metodologias tradicionais baseiam-se em um conjunto de atividades predefinidas durante o processo de desenvolvimento, que se inicia com o levantamento de requisitos, segue para o projeto e modelagem, vai para a implementação, teste, validação até chegar a manutenção. Em todas as atividades que estão ligadas ao

processo de desenvolvimento, temos uma forte documentação, que deve ser considerada. Essas documentações são consideradas características limitadoras aos desenvolvedores, pois geram muitos processos burocráticos.

Essa burocracia, segundo Sbrocco (2012, pg. 183), "é uma das principais razões que levam organizações de pequeno porte a não usar nenhum tipo de processo". Não adotando nenhuma metodologia, as empresas de pequeno porte pode ter resultados indesejáveis na qualidade do software e, com isso, dificultar a entrega dentro dos prazos estabelecidos.

DIFERENÇAS ENTRE TRADICIONAL E O ÁGIL

Muitos aspectos diferenciam as metodologias tradicionais das metodologias ágeis, entre eles, as regras que explicam como devem ser executados e conduzidos durante o desenvolvimento de software e na maneira de pensar nos envolvidos do projeto.

Segundo Sbrocco (2012), uma característica importante das metodologias ágeis é que elas estão preparadas para mudanças durante o processo de desenvolvimento de software, enquanto que as metodologias tradicionais são mais rígidas as mudanças.

Outra diferença é que as metodologias ágeis são baseadas em dados estatísticos enquanto que a metodologia tradicional utiliza normas que definem os padrões que devem ser seguidos (SBROCCO, 2012).

Nas metodologias ágeis a estratégia de trabalho adotada, conforme Sbrocco (2012), é determinada pela equipe de desenvolvimento, enquanto que nas metodologias tradicionais a imposição acirrada e as inúmeras normas e políticas devem ser respeitadas. O mesmo ocorre quando se fala em aspectos contratuais, mas as metodologias tradicionais são baseados em contratos rígidos e nas metodologias ágeis os contratos são mais flexíveis e as mudanças são bem-vindas.

Outra diferença que podemos observar, é em relação a participação dos clientes. Nas metodologias tradicionais a participação do cliente não é tão constante e não possuem poder de decisão em relação ao desenvolvimento do projeto. Enquanto que nas metodologias ágeis, o cliente faz parte dos envolvidos no

projeto de desenvolvimento e possui poder de decisão na forma como a implementação é desenvolvida.

Com relação ao tamanho das equipes, nas metodologias ágeis, o número de envolvidos na equipe é reduzido e trabalham em um mesmo local. Nas metodologias tradicionais, geralmente tem um grande número de envolvidos na equipe e estão em locais distantes.

Outra diferença que podemos citar, é com relação ao custo:

> [...] quando analisamos o custo gerado com mudanças ao longo do desenvolvimento do software. Considerando seus princípios, as metodologias ágeis já estão preparadas para aceitar mudanças no projeto, pois estão focadas nas pessoas e não nos processos, e por natureza denotam um controle rígido dele. Nas metodologias tradicionais, à medida que alterações são necessárias na fase próxima de seu encerramento, seu custo tende a crescer exponencialmente. Nas metodologias ágeis o custo não cresce ao final, mesmo que alterações de requisitos devam ser realizadas (SBROCCO, 2012, pg. 184).

Mas qual das metodologias é a melhor? Depende do contexto do software a ser desenvolvido. Mesmo considerando as vantagens das metodologias ágeis no desenvolvimento de software, sempre devemos pensar no que e para que será desenvolvido (SBROCCO, 2012). Softwares que são críticos, cuja precisão, risco e confiabilidade são decisivos, não podemos dispensar as recomendações que são impostas pelas regras e normas das metodologias tradicionais, principalmente com relação a documentação.

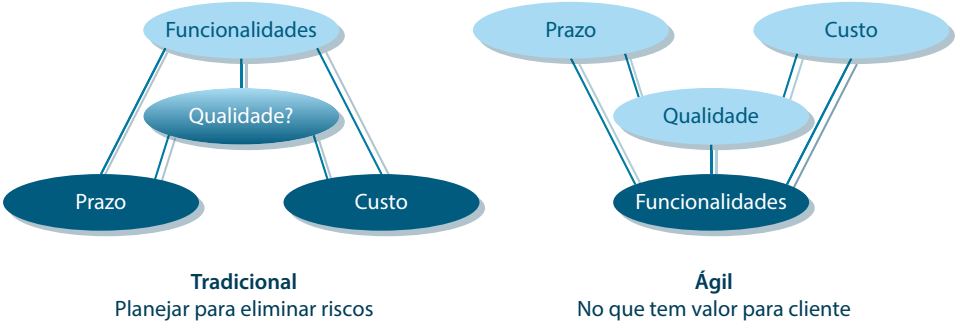


Figura 1 - Metodologia Tradicional X Metodologia Ágil
Fonte: a autora.

COMPARAÇÃO ENTRE METODOLOGIAS ÁGEIS X METODOLOGIAS TRADICIONAIS

Para ficar mais fácil de entender as diferenças entre os aspectos, vamos a um quadro comparativo que nos mostra uma visão geral das principais características entre as metodologias ágeis e tradicionais.

Quadro 1 - Comparação entre as características das Metodologias Tradicionais e Metodologias Ágeis

Aspecto	Metodologia Tradicional	Metodologia Ágil
Objetivo	Orientado por atividades e foco nos processo	Orientado por produto e foco nas pessoas
Projeto	Estáveis e inflexível à mudança	Adaptável a mudanças
Tamanho	Qualquer tamanho, efetivo em projetos de longa duração	Pequenos, mas pode ser usado em projetos de maior porte.
Gerente de projeto	Controle total do projeto	Papel de facilitador ou coordenador
Equipe do projeto	Atuação com papeis claros e bem definidos em todas as atividades	Atuação colaborativa em todas as atividades
Cliente	Participa nas fases inicias do levantamento de requisitos	Parte integrante da equipe do projeto
Planejamento	Detalhado e os envolvidos não participam	Curto e com a participação de todos os envolvidos

Fonte: a autora.

Independente de qual metodologia a empresa escolha o importante, em qualquer projeto de desenvolvimento de software, é a qualidade final do produtos e a satisfação do cliente.

A comparação entre as características das metodologias tradicionais e ágeis irá ajudar na melhor escolha da metodologia em determinado projeto, lembrando que devemos sempre levar em consideração o foco do tipo de projeto que será desenvolvido. O objetivo da comparação é a redução de erros que possam comprometer a qualidade do produto e a satisfação do cliente.

QUALIDADE DE SOFTWARE NAS METODOLOGIAS ÁGEIS

Conforme Sbrocco (2012, pg. 193), ainda enfrentamos "uma série de desafios relacionados ao cumprimento de prazos, redução de custos e obtenção de qualidade" e isso nos leva à necessidade de utilizarmos normas e modelos de qualidade durante o processo de desenvolvimento de software.

Caso esse erro não seja encontrado, o custo gerado com erros aumenta, já que a tendência, conforme Sbrocco sugere, é que ele se propague pelo ciclo de vida do software:

> [...] quanto mais tarde se descobrir um erro, maior será o prejuízo. Quando pensamos em qualidade de software, devemos deixar claro que ela é obtida quando percebemos que o produto desenvolvido está em conformidade com os requisitos funcionais e não funcionais. Portanto, a ocorrência de não conformidades com os requisitos representa a falta de qualidade. Assim, considerando que as metodologias ágeis não são orientadas a documentação [...] é comum refletirmos sobre a adequação de seu uso quando decidimos utilizar um modelo de qualidade (SBROCCO, 2012, pg. 193).

Na metodologia *Scrum* temos iterações rápidas, adaptações às mudanças, uso de padrões de qualidade entre outros aspectos. O *Scrum* possui o foco no controle do projeto e acompanhamento dos resultados. Por ter controle do projeto, ocorre uma constante verificação e validação de tudo o que está sendo desenvolvido e isso é o que representa a dimensão da qualidade no *Scrum*.

Alguns fatores de qualidade podem ser aplicados na metodologia *Scrum*, como por exemplo, compatibilidade, corretude, eficiência, integridade, manutenibilidade, portabilidade, reusabilidade e verificação e validação.

No fator **compatibilidade** cada incremento deve ser testado com todas as funcionalidades já concluídas para verificar a compatibilidade os componentes. Quando pensamos no fator **corretude**, pensamos em usar boas práticas desde o início do projeto. O fator **eficiência** se refere ao compartilhamento contínuo de informações entre os membros da equipe. A **integridade** é o compartilhamento de lições aprendidas, seja de sucessos ou não, que a equipe compartilha. O fator **manutenibilidade** diz que à medida que mais iterações e compartilhamento, entre os membros da equipe, os erros são detectados mais rápidos. Na **portabilidade** são adotados padrões em todas as fases do desenvolvimento e isso facilita a criação de componentes adaptáveis a qualquer tecnologia. Na **reusabilidade** temos a integração de componentes e práticas que podem ser compartilhadas entre as *sprints*. Com o controle do projeto com reuniões diárias, podemos ter o produto **verificado, validado** e testado a cada entrega feita ao cliente.

São vários os benefícios obtidos aplicando normas e modelos de qualidade no *Scrum*, entre eles, temos: melhor integração da equipe, otimização do tempo de trabalho e, com isso, aumentando produtividade, confiabilidade na entrega dos produtos e maior disseminação do conhecimento.

Na metodologia XP, temos a qualidade sendo gerada a partir das práticas, como a programação em par, onde o código é escrito por duas pessoas juntas e isso leva a um desenvolvimento de código melhor. A prática de desenvolvimento orientado a testes, onde os testes são escritos antes da implementação das funcionalidades, ajuda a assegurar que as funcionalidades realmente irão fazer o que foi projetado para fazer. E na refatoração, os desenvolvedores revisam o código constantemente, fazendo ajustes para torná-lo mais claro, legível e simples para adaptá-lo a novas necessidades (TELLES, 2006).

Conforme Sbrocco (2012) devemos visualizar as diversas perspectivas quando pensamos em qualidade de software. Temos o ponto de vista do usuário final, que quer um software que atenda às suas necessidades e seja confiável e fácil de operar. Pelo ponto de vista do desenvolvedor, qualidade se refere a manutenção rápida para atender às solicitações do cliente. No ponto de vista do cliente, a qualidade refere-se ao valor que o software agrega a sua empresa e pelo retorno do investimento.

REFLITA

Todo engenheiro de software deve reconhecer que modificações são naturais. Não tente combatê-las.

(Roger S. Pressman)

TESTES DE SOFTWARE NAS METODOLOGIAS ÁGEIS

No tópico anterior, destacamos os aspectos principais das metodologias ágeis e tradicionais e o que cada uma prioriza no desenvolvimento de software. O que se destaca é que no desenvolvimento, usando a metodologia ágil, a prioridade é a implementação de códigos executáveis e não a documentação do software. Com base nisso, podemos pensar: como garantir que requisitos funcionais e não funcionais sejam atendidos considerando a falta de documentação dos artefatos implementados?

Sobre esse aspecto da falta de amparo de artefatos e de documentação, Sbrocco descreve que:

> [...] tradicionalmente, procuramos garantir a fidelidade do que está sendo desenvolvido em relação aos requisitos pela execução de testes. Essencialmente, os requisitos necessários para realizarmos testes quando utilizamos paradigmas ágeis não são muito diferentes dos métodos de desenvolvimento tradicionais. A grande diferença está no grau de importância que conferimos aos testes, considerando cada uma das abordagens. Nas metodologias tradicionais, percebemos a existência

> de uma extensa lista de artefatos documentais resultantes da análise do projeto que será desenvolvido. Neste contexto, os testes são vistos apenas como mais um artefato produzido pelo processo, que obviamente visa garantir a detecção de erros antes da liberação do produto. Como observamos anteriormente, nos processos ágeis não existe muito amparo documental, e os testes, neste caso,passam a ter um papel importantíssimo para o sucesso da metodologia ágil utilizada (SBROCCO, 2012, pg. 186).

Quando pensamos em testes de software, devemos definir que tipo de teste será executado e, depois, definir estratégias de testes relacionadas ao uso de testes manuais ou automatizados. Testes manuais são realizados pelos desenvolvedores, testadores e pelo usuário final; seu objetivo é encontrar falhas e erros no sistema. O difícil do teste manual é conseguir que alguém teste todo o sistema exaustivamente a procura de falhas. Testes automatizados são scripts pré-configurados capazes de realizar testes exaustivos a procura de erros, mas não encontram todos os erros, já que falhas lógicas podem ocorrer com o código do script automatizado.

O ideal é combinar os testes manuais com os testes automatizados para garantir a maior cobertura de testes possíveis no sistema. Mas independente da estratégia de teste adotada, os testes são essenciais nas metodologias ágeis, pois estão relacionados diretamente com a qualidade do produto. Segundo Sbrocco (2012, pg. 186), "nas metodologias ágeis ocorrem os testes de unidades e também os testes de aceitação, pelo fato de contar com a participação do cliente durante o processo de desenvolvimento". O autor ainda comenta, que pouca ou nenhuma menção à atividade de testes é encontrada nas metodologias ágeis com exceção da metodologia XP, que enfatiza essa atividade.

Para Sbrocco (2012) testar o sistema é procurar por falhas no comportamento, usando estímulos à procura de erros e verificar se o sistema foi implementado conforme o que foi descrito na análise de requisitos. Isso significa que testar aumenta os custos, leva tempo e é necessários vários profissionais capacitados.

As metodologias ágeis buscam minimizar o risco desenvolvendo uma funcionalidade do software em prazos curtos de tempo, cuja duração não é maior que quatro semanas e por causa disso, muitas funcionalidades são finalizadas sem que tenha sido executado nenhum tipo de teste. Alguns testes como os de desempenho e segurança, podem durar mais que duas semanas.

Uma dificuldade que as equipes de testes encontram nas metodologias ágeis,

é com relação às mudanças. Pois, as vezes, após a equipe perder tempo testando o comportamento de determinada funcionalidade, é verificado que o erro foi devido a uma mudança de requisito não documentada e não é um erro.

Algumas estratégias de testes que podem ser utilizados nas metodologias ágeis são: testes estruturais (conhecido como teste da caixa branca) e testes funcionais (conhecido como teste de caixa preta), onde ambos podem ser automatizados. As metodologias ágeis podem utilizar mais de uma estratégia de testes, pois quando combinados, geram resultados melhores e garantem um software com mais qualidade.

TESTES DE SOFTWARE NAS METODOLOGIAS SCRUM E XP

Na metodologia XP é adotado técnicas de testes conhecidas como *Test-First Design* (Design Teste-Primeiro (TFD) ou *Test-Last*, onde qualquer método de um objeto que possa vir a falhar tem de ter um teste que garanta o seu funcionamento (SBROCCO, 2012).

A técnica de Test-First é usada da seguinte forma: a cada nova história de usuário é escrito um teste, onde é executado apenas a nova funcionalidade. Depois é implementado o código que atenta ao teste que foi escrito. O novo código é aceito se passar pelos testes. O *Teste-Last* é uma variação, onde o teste é escrito depois da implementação do código pelo desenvolvedor.

Para Sbrocco os testes unitários na metodologia XP devem fazer parte da modelagem do sistema, porque:

> [...] uma vez que tanto as funcionalidades quanto a documentação de um objeto são expressas por seus testes. Essa estratégia ajuda a diminuir a documentação na XP, pois sempre que alguém precisar saber a função de determinado objeto no contexto do sistema, basta procurar nos testes unitários, que estará registrada. Na XP, quando os programadores recebem a história que deve ser implementada, passam a escrever os testes unitários para elas (SBROCCO, 2012, pg. 192).

No *SCRUM* os vários recursos de comunicação que podem ser utilizados nessa metodologia, permite que as tarefas e os objetivos fiquem claros a todos os envolvidos no projeto. Como temos as reuniões periódicas, os objetivos específicos, a

situação atual do projeto e o andamento do trabalho ficam claros e mostra o que realmente deve ser feito e com isso o gerenciamento dos testes flui juntamente com o andamento o projeto.

Conforme Pham (2011, pg. 131), "em um projeto ágil ou *Scrum*, os testes não são mais realizados ao final do ciclo de vida, e sim "embutidos" ao longo das diferentes iterações e *Sprints*". Os testes no Scrum são focalizados em testes automatizados e testes de integração contínua, onde a sua equipe vai querer se certificar de que software está sempre disponível para ser entregue e com qualidade.

SAIBA MAIS

É indiscutível a importância dos testes nas metodologias ágeis de desenvolvimento. Torna-se inviável entregar um software sem nenhum tipo de teste. Os testes concretizam a ideia de que prevenir é melhor que remediar, ou seja, o tempo gasto testando os componentes de software é economizado após a entrega das funcionalidades. Ainda existem algumas dúvidas e discussões sobre o papel que um analista de testes deve desempenhar ao utilizar uma metodologia ágil. Alguns chegam a achar que não existe espaço para um profissional com essa especialização em uma equipe que utiliza paradigmas ágeis. É bom lembrar que no processo tradicional os testes aparecem no fim do ciclo de desenvolvimento. Nesse caso, o analista de teste é mais reativo. Quando esse profissional utiliza métodos ágeis, ele deixa de ser reativo para ocupar um papel fundamental no que diz respeito à interação dos desenvolvedores e demais integrantes do projeto.

Fonte: adaptado de Sbrocco (2012).

REFLITA

É importante que o ato de testar seja parte natural do ato de programar.

(Vinícius Manhães Teles).

ESTIMATIVA PARA DESENVOLVIMENTO ÁGIL

Conforme Pressman (2016) antes de iniciarmos um projeto de desenvolvimento de software, os envolvidos devem fazer uma estimativa do trabalho, dos recursos necessários e do período de tempo para a sua conclusão. Após ter estabelecido essas atividades, a equipe deve estabelecer um cronograma, com as tarefas e as metas que devem ser atingidas e identificar os responsáveis para a execução de cada uma.

As estimativas dependem de vários fatores que devem ser analisados, como por exemplo: a complexidade do projeto, o tamanho do projeto e o grau de incerteza estrutural e, esses fatores, afetam muito a confiabilidade das estimativas.

Segundo Pressman (2016, pg. 733) "as estimativas de custo e esforço nunca serão uma ciência exata". Fatores humanos, técnicos e físicos podem afetar o custo final e o esforço para o desenvolvimento de um software.

No desenvolvimento ágil, os requisitos são definidos por um conjunto de cenários de usuários (histórias de usuários em XP) e, com isso, é possível, conforme Pressman (2016, pg. 746) "desenvolver uma estratégia de estimativa informal, razoavelmente disciplinada e significativa no contexto do planejamento de projeto para cada incremento de software".

Na metodologia ágil, a estimativa para projetos usa estratégia de decomposição, que segundo Pressman (2016) segue os seguinte passos:

1. Cada cenário de usuário é separado para fins estimativa.
2. O cenário é decomposto em várias tarefas a serem desenvolvidas.

3. Cada tarefa é estimada separadamente (dados históricos, experiência, volume).
4. As estimativas de cada tarefa são somadas e usadas para criar as estimativas de cenário.
5. As estimativas de todos os cenários são somadas para desenvolver a estimativa para o incremento.

No desenvolvimento ágil, a duração para o incremento de software segundo Pressman é:

> [...] como duração do projeto exigida para o desenvolvimento de um incremento de software é muita curta (normalmente, de três a seis semanas), essa estratégia de estimativa tem dois propósitos: (1) garantir que o número de cenários a incluir no incremento esteja de acordo com os recursos disponíveis e (2) estabelecer uma base para a alocação de esforço à medida que o incremento é desenvolvido (PRESSMAN, 2016, pg. 747).

Para Santos (2011, pg. 28) a "estimativa dos custos de um projeto de software deve se basear nas funcionalidades do software" onde cada uma das funcionalidades a ser desenvolvida exige um esforço específico para ser criada, testada e integrada.

Na metodologia ágil temos o princípio que permite mudanças nos requisitos e isso prejudica muito o processo de estimativa de custo de um projeto de software.

Segundo Santos (2011, pg. 31), "quanto maior a experiência do desenvolvedor em projetos diversos, maior a sua assertividade na proposta da estimativa". Para estimar custos, é necessário conhecer as demandas e os requisitos que foram especificados e serão desenvolvidos.

REFLITA

Quanto mais você sabe, melhor você estima. Portanto, atualize suas estimativas à medida que o projeto avança.

(Roger Pressman).

SAIBA MAIS

O planejamento exige que você assuma um compromisso inicial, mesmo que mais tarde ele venha se mostrar errado. Sempre que forem feitas estimativas, deve olhar o futuro e aceitar o certo grau de incerteza. Embora seja muito mais arte do que ciência, ela não precisa ser conduzida de maneira aleatória. Existem técnicas úteis para estimar tempo e esforço. As métricas de projeto e processo podem proporcionar perspectivas históricas e informações valiosas para gerar estimativas quantitativas. A experiência (de todos os envolvidos) pode ajudar imensamente à medida que as estimativas são desenvolvidas e revisadas. As estimativas de recursos, custos e cronogramas para um trabalho de engenharia de software exigem experiência, acesso a boas informações históricas (métricas) e a coragem para se comprometer com as previsões quantitativas.

Fonte: Pressman (2016).

GERENCIAMENTO DE RISCOS EM PROJETOS ÁGEIS

Quando avaliamos os riscos de um projeto, conforme Bastos et al. (2007), estamos buscando aqueles fatos que poderão acarretar em perdas para a empresa. Não podemos sempre aliar um risco a uma perda. Um risco pode estar sempre presente, mas nem sempre ele gera uma perda. Existem riscos que sempre se transformam em perdas. Para Bastos et al. (2007), um avião sempre corre risco de cair, mas a perda só existirá se isso ocorrer. Resumindo, podemos dizer que o risco é uma probabilidade de ocorrência de uma perda para a empresa.

Conforme Bastos et al. (2007) exemplifica que qualquer empresa corre um risco se em algum momento seus computadores pararem de funcionar. Um site fora do ar pode trazer muitos prejuízos para uma empresa. O risco para uma empresa está relacionado ao grau de dependência em relação ao seu

equipamento. Agora, imagine, a ocorrência de erros de software e os prejuízos a uma empresa caso o seu software seja interrompido? Se porventura o sistema de faturamento sofra algum defeito e ele seja interrompido e a empresa deixe de receber o dinheiro?

Segundo Bastos (2007), devido a esses problemas, gerentes de TI passaram a investir para evitar riscos de defeitos em seus softwares, criando planos de contingência para contornar os problemas.

Tratar riscos depende de algumas decisões objetivas que poderão prevenir sua ocorrência ou, na pior das hipóteses, evitar que a sua ocorrência se reverta em sérios prejuízos. Mas o que leva um risco a se tornar uma perda? Ele se torna potencialmente uma perda quando ocorre a ameaça de sua ocorrência. E como elas podem ser reduzidas? Podem ser reduzidas com o uso de mecanismos de controle, que ajudam a controlar as ameaças e com isso evitar a sua ocorrência.

Para Sbrocco (2012), toda empresa deve ter um processo que identifique, qualifique e controle os riscos em uma linguagem que todos os envolvidos possam compreender. Ainda, para o autor, o risco tem três componentes básicos:

1. O fato ou evento que caracterizam um possível risco.
2. Probabilidade de que o fato ou evento realmente venham a acontecer.
3. Impacto financeiro se caso o fato ou evento ter acontecido.

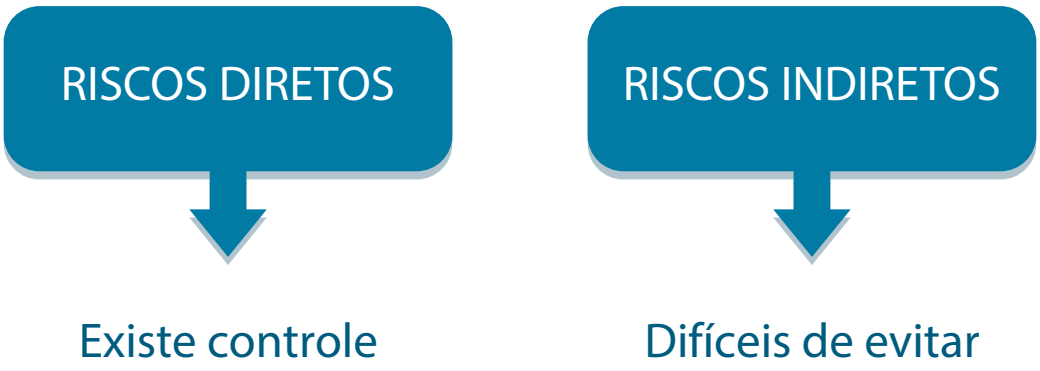


Figura 2 - Classificação dos riscos
Fonte: a autora.

Desenvolver um software é uma atividade arriscada e a empresa deve prever os possíveis riscos que podem afetar o desenvolvimento e sua qualidade. Temos vários tipos de riscos que depende da complexidade e do ambiente de desenvolvimento (SILVA, 2013). Mas alguns riscos são considerados prováveis de acontecer no desenvolvimento de software, conforme o quadro a seguir:

Quadro 2 - Riscos comuns em projetos de Software

TIPO DE RISCO	DESCRIÇÃO DO RISCO
Risco de Projeto	Ameaçam o plano do projeto (cronograma, orçamento, recursos, equipe, clientes e requisitos)
Riscos Técnicos	Ameaçam a qualidade e a data de entrega (análise, projeto, implementação, interface, verificação e manutenção)
Riscos de Negócio	Ameaçam a viabilidade do software, o projeto ou o produto (risco de mercado, risco estratégico, riscos de vendas, riscos gerencial e riscos de orçamento)

Fonte: a autora.

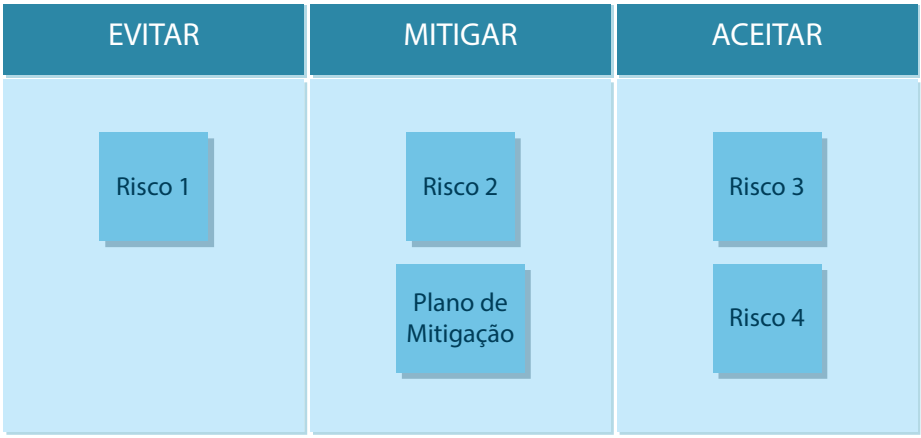
Nas metodologias ágeis, a simplicidade do processo, a comunicação constante nas reuniões diárias, o feedback do cliente e da equipe com as entregas frequentes ajudam a diminuir a complexidade dos sistema e ajudam a detectar os riscos antecipadamente.

Conforme Silva (2013, pg. 25), "o *Scrum* não fala explicitamente sobre um processo formal de Gerência de Riscos, mas é possível realizá-lo sem sacrificar a agilidade desta metodologia". Pode ser realizado de duas maneiras: orgânica - onde os riscos são identificados naturalmente pelas reuniões diárias - *plannings* ou revisões ou, de maneira evidente, onde basta incluir a realização da gerência de riscos, como pauta em alguma cerimônia do *Scrum*.

Uma boa prática, segundo Silva (2013), é adotar a manutenção de um quadro de riscos, como um quadro de tarefas do *Scrum,* que pode ser dividido nas seguintes partes:

- **Mitigação de um risco:** suavização ou diminuição das consequências ou probabilidade de um risco.
- **Aceitação de um risco:** aceitar ou evitar as consequências, podendo haver ou não um plano de contingência para o caso deste ocorrer.

Quadro 3 - Quadro de Riscos



Fonte: adaptado de Silva (2013, pg. 25).

Conforme os riscos vão sendo identificados, eles devem ser inseridos no quadro de riscos e ordenados por probabilidade e impacto para que fiquem classificados e acessíveis para as partes interessadas e para os responsáveis pelo gerenciamento dos riscos.

REFLITA

Embora seja importante considerar os riscos genéricos, são o riscos específicos que causam os maiores problemas.

(Roger Pressman e Bruce R. Maxim)

SAIBA MAIS

Métodos Ágeis e a Engenharia de Requisitos

Embora do métodos ágeis sejam caracterizados pelo foco em princípios, valores e práticas, e não em uma definição rigorosa de processos, esses métodos oferecem diversas orientações sobre como executar atividades da engenharia de requisitos. O elemento básico da engenharia de requisitos em diversos métodos ágeis é a história do usuário, definida como parte do método XP. Uma história do usuário é uma funcionalidade, um requisito de alto nível do sistema, detalhado apenas o suficiente para permitir que os desenvolvedores possam estimar o esforço necessário para implementá-la.

Podemos caracterizar e descrever três aspectos básicos das histórias do usuário:

- **Cartões**: são utilizados como meio de armazenamento das histórias do usuário.
- **Conversação**: o conteúdo da história é definido através da conversação durante o planejamento, ao estimar o esforço e ao planejar a implementação.
- **Confirmação**: é alcançada através da realização dos testes de aceitação, que são criados pelo cliente e podem ser implementados pela equipe de desenvolvimento.

Fonte: Machado (2016).

CONSIDERAÇÕES FINAIS

Olá, aluno(a)! Chegamos ao final de mais uma unidade onde aprofundamos nossos conhecimentos sobre as metodologias ágeis.

Aprendemos por meio de uma visão comparativa entre as metodologias ágeis como é importante entender as características e diferenças entre elas na hora de escolher qual a melhor para ser usada em determinado projeto. Adotar a melhor metodologia para o projeto, ajuda na redução de erros que comprometem a qualidade do produto e a satisfação do cliente.

A preocupação com a qualidade de software, pelas empresas desenvolvedoras, vem crescendo muito devido às exigências de clientes e do mercado. E independente da metodologia escolhida, tradicional ou ágil, existe um impacto muito grande no custo do desenvolvimento. Vimos que é possível manter a qualidade de software nas metodologias ágeis e a visão sobre a qualidade dos envolvidos no projeto.

E quando pensamos em qualidade de software, pensamos em um software sem problemas, sem defeitos, falhas ou erros. Para isso ocorrer, as empresas desenvolvedoras devem investir em estratégia e técnicas para que o software seja entregue sem problemas. Os testes devem ser realizados independentes da metodologia de desenvolvimento escolhidos, seja tradicional ou ágil.

Quando medimos, entendemos melhor o processo de software. A métrica proporciona uma melhor contínua nos processos de análise, projeto, codificação e teste e podem ajudar de maneira mais quantitativa e avaliativa quando aplicadas no desenvolvimento de um projeto ágil.

Finalmente para encerrar a nossa unidade, falamos sobre gestão de riscos e como elas são aplicadas nas metodologias ágeis. Nas metodologias ágeis a simplicidade, a comunicação constante e os feedbacks ajudam a diminuir a complexidade do sistema e com isso ajudam a detectar e a prevenir os riscos antecipadamente.

Preparado(a) para continuar com os estudos? Então, vamos seguir em frente. Ótima leitura!

ATIVIDADES

1. Os métodos ágeis têm, cada vez mais, despertado o interesse da comunidade de Engenharia de Software e das empresas como uma alternativa para o processo de desenvolvimento de sistemas de uma maneira mais rápida, eficiente e que atenda as reais necessidades dos clientes. Para que esse processo de desenvolvimento seja eficiente e atenda as necessidades dos clientes, é necessário que a empresa gerencie e controle os riscos que possam vir a surgir durante o projeto. **Com base nisso, cite e identifique os três componentes básicos do risco.**

2. Muitas características diferenciam as metodologias tradicionais das metodologias ágeis. Algumas diferenças são com relação a práticas e regras que explicam como devem ser executados e conduzidos durante o desenvolvimento de software e na maneira de pensar dos envolvidos do projeto. **Assinale a alternativa correta com relação às diferenças entre as metodologias tradicionais e ágeis.**

   a) Metodologias ágeis estão preparadas para mudanças durante o processo de desenvolvimento de software e metodologias tradicionais são mais rígidas as mudanças.

   b) As metodologias ágeis são baseadas em dados estatísticos e a metodologia tradicional também se baseia somente em dados estatísticos e normas.

   c) A estratégia de trabalho da metodologia ágil é baseada em inúmeras regras e normas que devem ser respeitadas e as metodologias tradicionais possui uma imposição menos acirrada, com poucas regras e normas políticas.

   d) Metodologias ágeis: os contratos são mais rígidos e as mudanças devem ser muito bem analisadas pela equipe.

   e) Nas metodologias tradicionais a participação do cliente é constante e nas metodologias ágeis também onde o cliente faz parte dos envolvidos no projeto de desenvolvimento.

3. Quando falamos em testes, devemos pensar que eles concretizam a ideia de que prevenir é melhor que remediar, ou seja, o tempo gasto testando os componentes de software é economizado após a entrega das funcionalidades. **Com relação às metodologias ágeis, como são os testes? Possuem algum tipo de atividade de teste que é realizado durante o desenvolvimento do software?**

4. Ao planejarmos em como devemos atingir um objetivo, normalmente pensamos nas tarefas que precisamos realizar para completar este objetivo e o que é necessário, como por exemplo: custo, tempo e o esforço. Ou seja, estamos realizando estimativas para analisar o que precisaremos para atingirmos o objetivo. As empresas também realizam estimativas quando desenvolvem seus produtos. Na metodologia ágil, a estimativa para projetos usa estratégia de decomposição. **Descreva os passos que esta estratégia usa.**

ATIVIDADES

5. O risco para uma empresa está relacionado ao grau de dependência em relação ao seu equipamento. Tratar riscos depende de algumas decisões objetivas que poderão prevenir sua ocorrência ou, na pior das hipóteses, evitar que a sua ocorrência se reverta em sérios prejuízos. **Assinale a alternativa que NÃO corresponde a um risco considerado provável de acontecer em um desenvolvimento de software.**

   a. Riscos de projeto ameaçam o plano do projeto, como por exemplo, o orçamento, recursos e equipe.

   b. Riscos técnicos ameaçam a qualidade e a data de entrega.

   c. Riscos na análise, na implementação e na interface são considerados riscos técnicos.

   d. Riscos de negócio que ameaçam a viabilidade do software, o projeto ou o produto.

   e. Escopo do projeto bem delimitado, possibilitando que se estabeleça um cronograma de testes e entregas no prazo.

UMA PROPOSTA PARA O DESENVOLVIMENTO ÁGIL DE AMBIENTES VIRTUAIS

Processos ágeis de desenvolvimento têm como característica uma abordagem iterativa e incremental dos princípios de Engenharia de Software. Tal abordagem revela-se extremamente adequada aos projetos de Realidade Virtual e proporciona, por sua natureza evolutiva, grandes benefícios associados à gestão de riscos em projetos de software [...]. A palavra ágil foi relacionada, pela primeira vez, à Engenharia de Software em 2001, por um consórcio de especialistas em métodos de desenvolvimento, que elaboraram na ocasião o "Manifesto Ágil". Tal manifesto destaca alguns dos princípios compartilhados por diferentes metodologias de desenvolvimento, a partir de então denominados processos (ou métodos) ágeis [...]:

- Indivíduos e interações são mais importantes que processos e ferramentas.
- Software funcionando é mais importante do que documentação detalhada.
- Colaboração dos clientes é mais importante de que negociação de contratos.
- Adaptação às mudanças é mais importante do que planejamento extensivo.

Por outro lado, o desenvolvimento de aplicações em Realidade Virtual requer pleno conhecimento e entendimento de diversas disciplinas, tais como computação gráfica, modelagem geométrica, interação multimodal, entre outras [...]. Algumas características desses sistemas revelam a necessidade de um aprimoramento contínuo em seu processo de desenvolvimento. Dentre essas características, destacam-se:

- Rápida evolução da tecnologia relacionada à visualização e à capacidade gráfica dos computadores [...].
- Indecisão e mudanças de opinião por parte dos clientes, problema agravado em sistemas compostos por equipamentos de alto custo [...].
- Necessidade da implementação contínua de protótipos, visando a avaliação da viabilidade do sistema pelo cliente.

O desenvolvimento evolucionário, o planejamento adaptativo e o acolhimento de alterações nos requisitos apresentam-se como possíveis melhorias associadas ao desenvolvimento ágil de ambientes virtuais.

O desenvolvimento ágil de software é uma abordagem de desenvolvimento caracterizada pela adaptabilidade, ou seja, pela capacidade de absorver mudanças, sejam elas nas forças de mercado, nos requisitos do sistema, na tecnologia de implementação ou nas equipes de projeto [...].

O desenvolvimento de sistemas de Realidade Virtual, assim como o desenvolvimento de qualquer sistema de software, demanda um processo, uma metodologia de desenvolvimento. No entanto, tal processo deve se adequar à rápida evolução da tecnologia

associada a esses sistemas. [...] apresentado um processo de desenvolvimento que agrega características de prototipagem, desenvolvimento iterativo e desenvolvimento evolucionário de sistemas de software. Esse processo baseia-se em conceitos e modelos da Engenharia de Software, adaptados às peculiaridades dos Sistemas de Realidade Virtual (SRV). Esse processo compõe-se de cinco etapas, realizadas iterativamente: Análise de requisitos, Projeto, Implementação, Avaliação e Implantação.

Nenhum processo de desenvolvimento pode garantir, simplesmente por sua aplicação, o sucesso do produto de software ao qual foi aplicado [...]. No entanto, é possível destacar algumas características comuns a alguns processos bem sucedidos software ao qual foi aplicado. No entanto, é possível destacar algumas características comuns a alguns processos bem sucedidos [...]:

- Desenvolvimento iterativo : projetos maiores, com vários componentes, são mais propensos a problemas de integração. Um planejamento adequado de iterações reduz os problemas de integração e facilita o acompanhamento do processo de desenvolvimento pelos gerentes de projeto.
- Avaliação contínua do processo: nenhum processo de desenvolvimento de software pode garantir a total imunidade do projeto face a problemas como mudanças na equipe de desenvolvimento ou nos requisitos do usuário. A avaliação (e consequente adaptação) do processo de desenvolvimento é de fundamental importância ao longo do ciclo de vida do projeto.
- Boas práticas ("Best practices"): as melhorias associadas à utilização de boas práticas de desenvolvimento [...] ou de padrões de projeto [...] não devem ser negligenciadas em projetos de software.

A abordagem sugerida para a adaptação de um processo existente a determinado contexto consiste em personalizar um processo existente, testando e refinando-o de maneira iterativa, de acordo com as características de cada projeto.

Fonte: adaptado de Mattioli et al. ([2017], on-line)[1].

MATERIAL COMPLEMENTAR

LIVRO

AGILE Scrum Master no Gerenciamento Avançado de Projetos
Vitor L. Massari

Editora: Brasport
Sinopse: muitos livros comentam sobre o framework Scrum e a filosofia Agile, mas poucos entram mais a fundo em questões sobre como implementá-los em ambientes que já possuem outra metodologia – ou mesmo nenhuma metodologia –, quais os cuidados que devemos ter nessa implementação, que resistências iremos enfrentar, que técnicas de motivação e engajamento podemos usar, como fazer uma transição de papéis como gerente de projetos, líder técnico e analista de negócios para os papéis do Scrum, como a área de PMO se encaixa dentro de uma organização Scrum/Agile, como combinar modelos de tal forma a criar um modelo híbrido e como criar uma gestão de serviços de TI ágil.

LIVRO

Gestão de Riscos: Método Ágil Para Identificação De Riscos Em Projetos De Ti
Marcelo Caixeta

Editora: Prismas
Sinopse: o gerenciamento de riscos é considerado por alguns autores como um processo-chave e que faz a diferença em gerenciamento de projetos, além de ser atualmente de grande importância para governos, economias e empresas. A presente dissertação partiu do estudo de diversos padrões de conhecimento e metodologias de gerenciamento de projetos, tanto de métodos tradicionais quanto de métodos ágeis. Foram estudados também os processos de gerenciamento de riscos desses métodos. Durante os estudos foi possível conhecer e apresentar alguns modelos de gestão de riscos em projetos de desenvolvimento de software já estudados por outros autores. A partir daí passou-se à análise de alguns métodos utilizados para a identificação de riscos em projetos. Ao concluir os estudos citados e, após a pesquisa de campo, foi possível conhecer a realidade de algumas empresas consideradas como referência / líderes em sua área de atuação, observando-se na prática de que forma elas utilizam tanto métodos tradicionais como métodos ágeis para gerenciamento de projetos e de riscos. Por fim, a partir da investigação realizada, foi elaborada a proposta de um método ágil para a identificação de riscos em projetos de desenvolvimento de software, alinhando os conhecimentos dos métodos tradicionais e dos métodos ágeis.

MATERIAL COMPLEMENTAR

NA WEB

Desenvolvimento tradicional x desenvolvimento ágil de software
Artigo que aborda um tema que gera muitas controvérsias hoje em dia na área desenvolvimento de software: desenvolvimento tradicional x desenvolvimento ágil. Excelente artigo.
Para saber mais, acesse o link disponível em: <http://blog.ceviu.com.br/info/artigos/desenvolvimento-tradicional-x-desenvolvimento-agil-de-software/>.

Agile, Scrum e Burocracia
Artigo que fala sobre métodos ágeis, sobre Scrum e sobre a burocracia no desenvolvimento de software. É mostrado que as vezes as gerências desconhecem a essência da filosofia ágil e sua forma de trabalho e esse desconhecimento leva a julgamentos equivocados. Por outro lado, muitos agilistas julgam os modelos tradicionais como burocracia. Ótimo artigo. Para saber mais, acesse o link disponível em:
<https://www.infoq.com/br/articles/agile-scrum-burocracia>.

Scrum e o gerenciamento de projetos
Este artigo tem como meta apresentar, de forma prática, como um mesmo projeto pode ser beneficiado com a aplicação das técnicas ágeis de gerenciamento de projetos, unidas e aliadas às tradicionais, de forma a se complementarem mutuamente nos aspectos de papéis e responsabilidades do Scrum, com os papéis e responsabilidades do gerente de projetos e sem haver conflitos de interesse. Para saber mais, acesse o link disponível em:
<http://www.devmedia.com.br/scrum-e-o-gerenciamento-de-projetos/22526>.

REFERÊNCIAS

BASTOS A.; RIOS E.; CRISTALLI R.; MOREIRA T. **Base de Conhecimento em Teste de Software**. 2 ed. São Paulo: Editora Martins, 2007.

BONATO A. S. F. **Extreme Programming e Qualidade de Software**. São Paulo: USP, 2002.

SANTOS, A. B. **Um método para gerenciamento de custos em projetos de software desenvolvidos com metodologia Scrum**. São Paulo: Instituto de Pesquisas Tecnológicas do Estado de São Paulo (IPT), 2011.

MACHADO, F. N. R. **Análise e Gestão de Requisitos de Software - onde nascem os sistemas.** 3 ed. São Paulo: Érica, 2016.

SILVA, D. E. dos S.; SOUZA, I. T. de.; CAMARGO, T. Metodologias ágeis para o desenvolvimento de software: Aplicação e o uso da Metodologia SCRUM em contraste ao Modelo Tradicional de Gerenciamento de Projetos. **Revista Computação Aplicada**, Rio de Janeiro: v. 2, n. 1, 2013.

SILVA, T. M. V. **Gerência de Riscos aplicada a Metodologias Ágeis de Desenvolvimento**. São Paulo: FATEC, 2013.

SBROCCO, J. H. T. C.; MACEDO P. C. de. **Metodologias Ágeis:** Engenharia de Software sob medida**.** 1. ed. São Paulo: Érica, 2012.

PRESSMAN, R.; MAXIM B. R. **Engenharia de Software –** Uma abordagem profissional. 8. ed. Porto Alegre: AMGH, 2016.

PHAM A.; PHAM P. **Scrum em Ação:** Gerenciamento e desenvolvimento Ágil de Projeto de Software. São Paulo: Novatec: Cengage Learning, 2011.

TELES, V. M. **Extreme Programming** - Aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec, 2006.

REFERÊNCIAS ON-LINE

[1]Em:<https://www.researchgate.net/profile/Alexandre_Cardoso2/publication/255640731_Uma_Proposta_para_o_Desenvolvimento_Agil_de_Ambientes_Virtuais/links/0deec5357a588361d3000000.pdf. Acesso em: 18 abr. 2017.

GABARITO

1.Os três componentes básicos do risco são:

1. O fato ou evento que caracterizam um possível risco.
2. Probabilidade de que o fato ou evento realmente venham a acontecer.
3. Impacto financeiro se caso o fato ou evento ter acontecido.

2. a) Metodologias ágeis estão preparadas para mudanças durante o processo de desenvolvimento de software e metodologias tradicionais são mais rígidas as mudanças.

3. Nas metodologias ágeis ocorrem os testes de unidades e também os testes de aceitação, pelo fato de contar com a participação do cliente durante o processo de desenvolvimento. Mas possuem pouca ou nenhuma menção a atividade de testes em si, com exceção da metodologia XP, que enfatiza essa atividade.

4. Os passos que a estratégia de decomposição usa são:

1. Cada cenário de usuário é separado para fins estimativa.
2. O cenário é decomposto em várias tarefas a serem desenvolvidas.
3. Cada tarefa é estimada separadamente (dados históricos, experiência, volume).
4. As estimativas de cada tarefa são somadas e usadas para criar as estimativas de cenário.
5. As estimativas de todos os cenários são somadas para desenvolver a estimativa para o incremento.

5. e) Escopo do projeto bem delimitado, possibilitando que se estabeleça um cronograma de testes e entregas no prazo.

Professora Esp. Janaina Aparecida de Freitas

OUTRAS METODOLOGIAS ÁGEIS

UNIDADE V

Objetivos de Aprendizagem

- Introduzir os conceitos do Processo Unificado Aberto (OpenUP).
- Apresentar as origens e características básicas da metodologia Crystal.
- Entender as origens e características básicas da metodologia TDD.
- Compreender as origens e características básicas da metodologia LD.
- Apresentar os conceitos básicos do Kanban.

Plano de Estudo

A seguir, apresentam-se os tópicos que você estudará nesta unidade:

- OpenUP – Processo Unificado Aberto
- Metodologia Crystal
- Test Driven Development (TDD)
- Lean Software Development (LD)
- Kanban

UniCesumar

INTRODUÇÃO

Olá, aluno(a)! Estamos chegamos ao final do nosso livro e, nesta unidade, será apresentada uma visão geral de outras metodologias ágeis de desenvolvimento presentes no mercado.

Para ser um bom profissional na área é necessário ser multidisciplinar e, além do que foi aprendido, procurar estudar sempre, pesquisar e aprender, além das metodologias ágeis que estão citadas no livro.

Nesta unidade, aprenderemos sobre as metodologias ágeis: OpenUp (Processo Unificado Aberto), Metodologia Crystal, Test Driven Development (TDD), Lean Software Development (LD) e Kanban.

Nosso estudo se inicia com a metodologia OpenUp (Processo Unificado Aberto) possui uma abordagem iterativa e incremental para conduzir os projetos de desenvolvimento de software. A sua qualidade é com base o modelo RUP, a agilidade da metodologia XP e usa as melhores práticas de gerenciamento do método Scrum.

Seguindo, na unidade, aprenderemos sobre a Metodologia Crystal, onde será apresentada suas origens e características básicas. A metodologia Crystal é dividida em cores, onde quanto mais escuro mais complexo e crítico será o sistema a ser desenvolvido. Esta metodologia atende diferentes tipos e tamanhos de projeto.

Também vamos conhecer o TDD ou *Test Driven Development,* conhecida como uma técnica de desenvolvimento orientada a testes, considerada uma metodologia de desenvolvimento de software ágil derivada da metodologia XP e do Agile Manifesto. Finalmente, para encerrar a nossa unidade, vamos estudar a metodologia Lean Software Development (LD) que tem como foco a eliminação de desperdícios, a excelência na qualidade e a velocidade dos processos. Também veremos sobre o Kanban que é considerado uma ferramenta visual que ajuda no acompanhamento do fluxo de trabalho e do controle do trabalho em progresso.

Preparado(a) para continuar? Então, vamos seguir em frente. Ótima leitura e bons estudos!

OPENUP – PROCESSO UNIFICADO ABERTO

OpenUp ou Processo Unificado Aberto é uma metodologia ágil com abordagem iterativa e incremental usada para guiar projetos de software criada em 2005 pela *Enterprise Foundation* em parceria com a *IBM Rational Software*. Conforme Sbrocco (2012, pg. 235) "preza pela qualidade encontrada no RUP, com a agilidade da metodologia *Extreme Programming* e com as melhores práticas de gerenciamento do método *Scrum*".

Considerada uma metodologia leve, que aplica as abordagens iterativa e incremental em um Ciclo de vida estruturado com abordagem ágil e pragmática, possui foco no desenvolvimento colaborativo de software e valorização da equipe, trazendo muitos benefícios aos clientes. Para Sbrocco (2012), a metodologia OpenUp está dividida em quatro grandes áreas: comunicação e colaboração, objetivo, solução e gerência. Essas áreas atuam interativamente no Ciclo de vida e baseiam-se nas seguintes disciplinas:

- Análise e Projeto.
- Gerência de Configuração e Mudança.
- Implementação.
- Gerência de Projetos.
- Requisitos.
- Teste.

Cada disciplina possui um número de tarefas que indica como um papel (*role)* deve realizar o seu trabalho. As tarefas são definidas como uma série de passos que mostram a criação e atualização de um ou mais produtos de trabalho.

Na metodologia OpenUp há tarefas que são realizadas pelos papéis executores primários, que executam as tarefas e outras tarefas pelos executores suplementares, que ajudam a fornecer informações e dar suporte na execução das tarefas (GAVA, 2009).

Um papel (*role*) indica quem realiza o trabalho, define o comportamento e descreve as responsabilidades de um indivíduo ou um mais indivíduos da equipe.

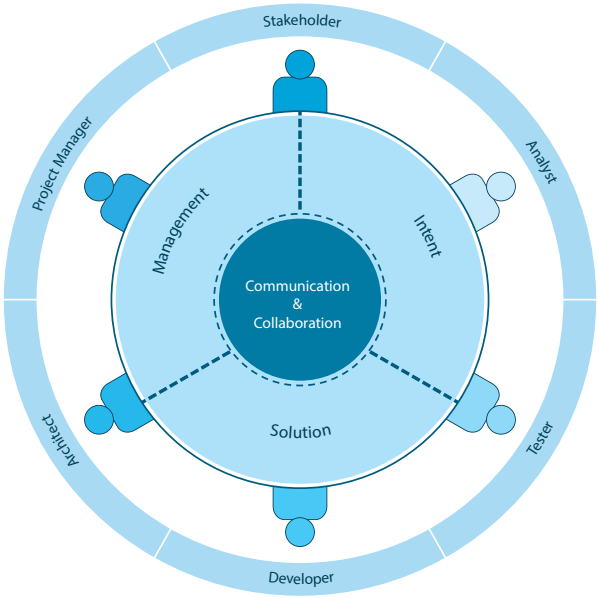


Figura 1 - Papéis no OpenUp
Fonte: Gava (2009, pg. 25).

Na Figura 1, temos os papéis definidos para a metodologia OpenUp, no círculo exterior temos os papéis definidos e, no círculo interior, temos o que cada papel tem como foco. Temos: *Intent* (intenção - o que deve ser executado), *Management* (Gerenciamento do projeto) e *Solution* (soluções do software). Na parte central temos a comunicação e a colaboração como foco dos papéis. Os sete papéis propostos pelo OpenUp são descritos a seguir, conforme Gava (2009):

- ***Stakeholder*** : se comunica e colabora com o *Management* (Gerente de Projeto) e com os analistas.
- **Analista (*Analyst*):** coleta informações com os *stakeholder* e seu foco principal é a Intenção (*Intent*).

- **Arquiteto (*Architect*)**: responsável pela arquitetura, design e a codificação do projeto. Ele se comunica com o *Management* e com os desenvolvedores.
- **Desenvolvedor (*Developer*)**: responsável por escrever os códigos, os testes unitários e a integração entre os componentes. Seu foco é a solução (*Solution*).
- **Testador (*Tester*):** responsável pelos testes no código. Se comunica com o Analista (*Intent*) e com os desenvolvedores (*Solution*).
- **Gerente de Projeto *(Project Manager)***: responsável por liderar, planejar, coordenar os *stakeholders* e o time do projeto.
- **Qualquer papel (*Any Role*)**: qualquer membro da equipe que pode executar tarefas gerais.
- A metodologia OpenUp possui quatro princípios:
  - Igualar as prioridades para maximizar o benefício aos stakeholders.
  - Colaboração constante para alinhar os interesses.
  - Focar na arquitetura para reduzir o risco.
  - Evolução contínua com feedbacks para promover melhorias.

VISÃO GERAL DO OPENUP

A metodologia Openup se baseia em um ciclo iterativo e incremental com foco em criar um modelo ágil e enxuto. O diferencial da metodologia é que ela possui a iteração normal e mais um micro incremento. Segundo Diedrich (2011, pg. 38), "os micro incrementos são pequenas unidades de trabalho pessoais de um ou mais membros da equipe, as unidades de trabalho podem representar algumas horas a alguns dias de trabalho em algum item de produto de entrega aceitável".

Na figura a seguir, temos o funcionamento completo do Ciclo de vida do Openup.

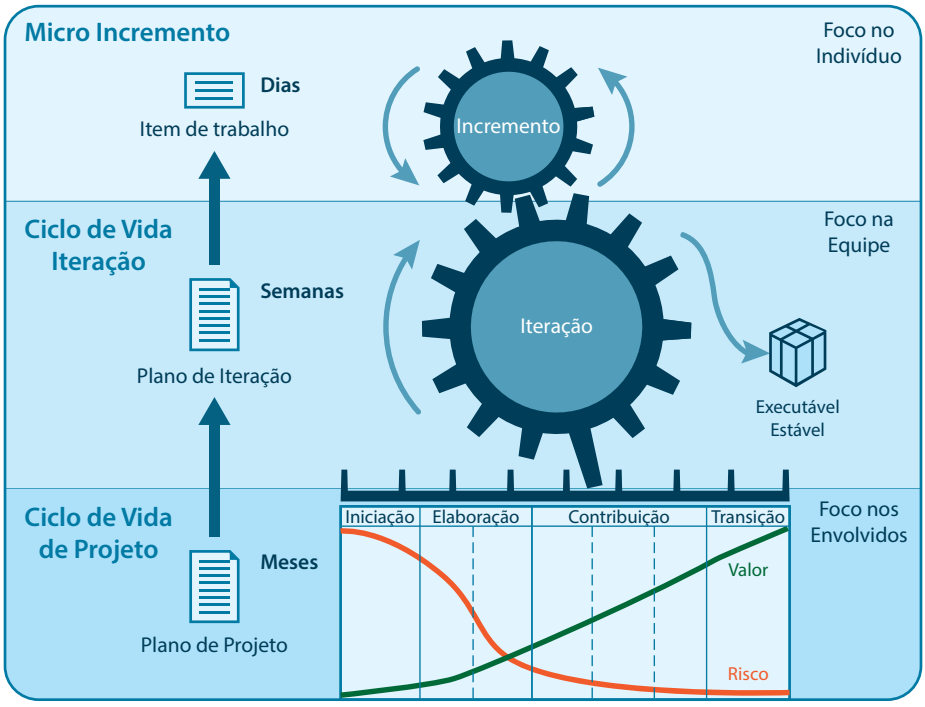


Figura 2 - Ciclo de Vida OpenUP
Fonte: Diedrich (2011).

No Ciclo de vida do projeto temos o artefato Plano de Projeto que possui todas as datas de atualização e fases a serem seguidas até o seu término.

No Ciclo de Vida de Iteração, as iterações mantém a equipe focada na entrega de valores para o cliente. Essas entregas ocorrem dentro de semanas, onde é entregue um protótipo funcional da aplicação.

Segundo Gava, o conteúdo do OpenUp é organizado da seguinte forma:

> [...] em três níveis, cada um com um enfoque em particular. Estes níveis são: pessoal, time e stakeholder. No nível pessoal (*Personal Focus)*, cada indivíduo contribui com micro-incrementos, que devem auxiliar o time a atingir os objetivos estabelecidos. O time de desenvolvimento (*Team Focus*), liderado pelo gerente, faz um plano de iteração tendo como base os objetivos e as prioridades definidas. Os *stakeholders* (*Stakeholder Focus*), finalmente, contribuem ao fazer a avaliação dos resultados de um release para fornecer o feedback sobre o produto de software funcional (GAVA, 2009, pg.26).

Podemos dizer, com o que vimos, que a metodologia OpenUp pode ser utilizada para o desenvolvimento de qualquer tipo de projeto com uma maior agilidade e podendo ser utilizado com qualquer tipo de recurso.

METODOLOGIA CRYSTAL

O método Crystal ou família Crystal, criado por Alistair Cockburn em 1998, tem o foco no valor da comunicação e nas interações entre as pessoas durante o processo de desenvolvimento de software. Conforme Filho (2008) com mais comunicação entre as pessoas, o número de artefatos gerados para a obtenção do produto final será menor e o trabalho será mais objetivo.

Para Sbrocco (2012, pg. 133), "trata-se de uma família de metodologias com um código genérico comum que atende diferentes tipos e tamanhos de projetos". O método Crystal, para Filho (2008) possui o foco nos talentos e nas habilidades das pessoas envolvidas no projeto, permitindo que o desenvolvimento ocorra conforme as características da equipe.

A metodologia Crystal foi dividida em cores para seu melhor entendimento, quanto mais escuro, mais crítico o sistema será. Para Sbrocco essa divisão de cores pode ser descrita onde,

> cada cor tem um propósito diferente e elas são separadas de acordo com a criticidade, ou seja, projetos menores envolvem poucos desenvolvedores, e no caso de problemas, o prejuízo tende a ser menor. Já os projetos maiores ou de segurança crítica normalmente envolvem mais profissionais, portanto o prejuízo deve ser muito maior, podendo colocar a vida de pessoas em risco (SBROCCO, 2012, pg, 134).

A tabela a seguir mostra a divisão de cores do método Crystal:

Tabela 1 – Divisão da família Crystal

Cores	Número de desenvolvedores	Em caso de falha...
Clear	1-6	Perdem dinheiro, mas recuperam facilmente.
Yellow	7-20	Perdem dinheiro discretamente
Orange	21-40	Perdem dinheiro substancialmente
Red	41-100	Há perda substancial de dinheiro e possivelmente, vidas humanas.

Fonte: Sbrocco (2012, pg. 134).

O projeto pode ser classificado usando as seguintes características: criticidade e tamanho da equipe. Pois quanto mais pessoas envolvidas no projeto, mais artefatos e formalismos são necessários para possibilitar a comunicação entre os membros da equipe.

PRINCÍPIOS E FILOSOFIA DO MÉTODO CRYSTAL

O método Crystal, independente da divisão de cores, conforme Sbrocco (2012) possui sete princípios básicos:

1. Trabalho direto com o cliente: envolver o cliente nas iterações e decisões.
2. Maior complexidade, maior o custo.
3. Equipes maiores pedem metodologia diferenciada.
4. Muita cerimônia gera maior criticidade: mais comunicação com os envolvidos.
5. Comunicação eficiente (feedbacks).
6. Habilidade em lidar com pessoas.
7. Eficiência no desenvolvimento.

O método Crystal possui algumas propriedades a serem seguidas durante o desenvolvimento do projeto pela equipe:

1. Entregas Frequentes: software testado e sendo funcional deve ser entregue a intervalos de tempo (poucos meses).

2. Melhoria Reflexiva: a equipe deve identificar formas de melhorar, rever falhas, verificar o que deu certo.

3. Comunicação Intensa: comunicação cara a cara, focada nas pessoas e na troca constante de informações.

4. Segurança Pessoal: descobrir as fraquezas, as pessoas devem se sentir à vontade para dar opiniões, ambiente de confiança entre os membros da equipe.

5. Foco: saber no que trabalhar, ter prioridades do projeto bem definidas.

6. Acesso Fácil a Usuários Experientes: usuários com experiência e com especialidades diferentes devem estar disponíveis para serem consultados em caso de dúvidas.

7. Integração Contínua com Testes: ambiente deve ser assegurado por testes automatizados, com gerenciamento de configuração e com integração contínua.

EQUIPE CRYSTAL

Ao usarmos a metodologia Crystal, segundo Sbrocco (2012, pg. 136) "devemos considerar uma "equipe base" desenvolvimento que deve ser formada". Esta equipe deve possuir os seguintes profissionais:

- **Patrocinador:** também chamado de Stakeholder, é o responsável pelo financeiro da empresa.
- **Coordenador de Projeto:** realiza ações de coordenação e liderança da equipe de desenvolvimento.
- **Analista de Negócios:** responsável por levantar os requisitos do sistema e pela modelagem desses requisitos.
- **Usuário Stakeholder:** o número de usuários depende do tamanho do projeto e de desenvolvedores envolvidos. Deve haver pelo menos um usuário responsável e que conheça bem o processo para acompanhar o projeto.
- **Designer/Projetista:** responsável pela arquitetura do sistema e pelos elementos da interface/humano computador.
- **Programador/Designer:** responsável pela implementação dos códigos do

sistema, adaptação de componentes e configurações iniciais do sistema. Essa metodologia sugere que use padrões de codificação,ou seja, adoção de "*Designers Patterns*".

- **Testador:** responsável por testar o sistema. Essa metodologia considera obrigatório que os testes de regressão sejam executados.
- **Redator:** responsável por documentar os fatos durante o desenvolvimento do projeto.

CICLO DA VIDA

A metodologia Crystal sugere um Ciclo de Vida baseado em integrações e nas seguintes práticas:

- **Encenação (Staging)**: onde ocorre o planejamento do próximo incremento do sistema. Nessa fase a equipe seleciona os requisitos que irão ser implementados na iteração e determinam o prazo para sua entrega.
- **Construção, Demonstração e Revisão**: fase onde é editado e revisado os objetivos do incremento.
- **Monitoramento do Processo**: fase onde o processo é monitorado levando em consideração o progresso e estabilidade da equipe.
- **Paralelismo e Fluxo**: nessa fase as diferentes equipes podem operar com máximo paralelismo.
- **Inspeções de usuários**: inspeções feitas por usuário a cada incremento.
- **Workshops refletivos**: são reuniões que devem ocorrer antes e depois das interações para analisar o progresso do projeto.
- **Local matters**: são os procedimentos a serem aplicados, que variam de acordo com o tipo de projeto.
- **Produtos de Trabalho (Work Products):** são seqüências de lançamento, modelos de objetos, manual do usuário, casos de teste e migração de código. Para o Clear, são especificamente Casos de uso e descrição de funcionalidade e para o Orange, são documentos de requisitos.

- **Padrões (Standards)**: padrões de código, convenções de produto, formatação e normas de qualidade usadas no projeto.
- **Ferramentas (Tools)**: são as ferramentas mínimas utilizadas no desenvolvimento do projeto, como compiladores, gerenciadores de versão e configuração, ferramentas de versão, programação, teste, comunicação, monitoramento de projeto, desenho e medição de desempenho.

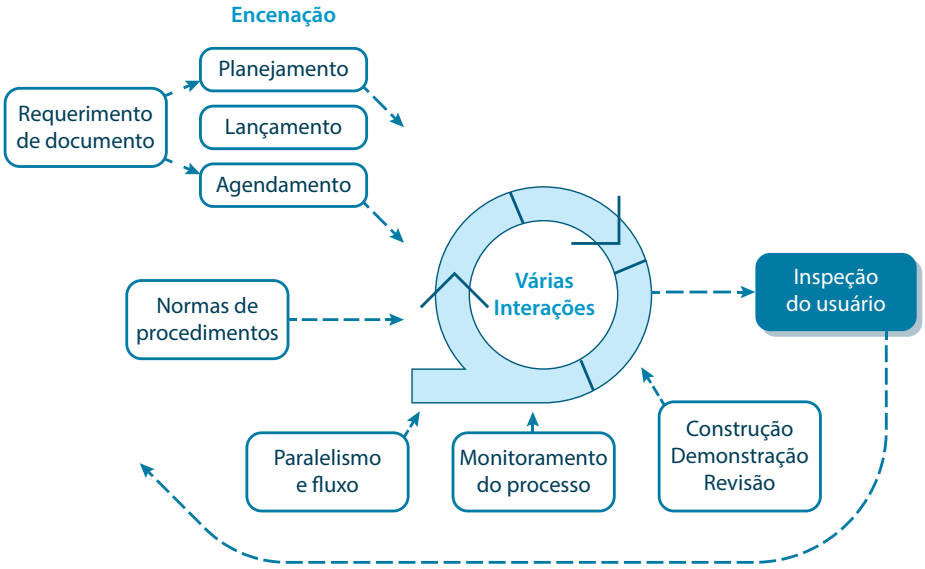


Figura 3 - Ciclo de Vida Metodologia Crystal
Fonte: Maestre (2015, on-line)[1].

O método ágil Crystal adota uma estratégia iterativa e se ajusta a qualquer processo ou projetos de tamanhos e complexidades diferentes.

REFLITA

Devemos considerar que nem todo os desenvolvedores trabalham da mesma forma e não possuem os mesmo conhecimentos.

(José Henrique Teixeira Carvalho Sbrocco).

TEST DRIVEN DEVELOPMENT (TDD)

O TDD ou *Test Driven Development* é uma técnica de desenvolvimento orientada a testes, proposta por Kent Beck em 2003, considerada uma metodologia de desenvolvimento de software ágil derivada da metodologia Extreme Programming (XP) e do Agile Manifesto (Agile Alliance 2000), que busca antecipar a identificação e a correção de erros durante o desenvolvimento de software. Conforme Sbrocco (2012, pg. 236) "baseia-se em pequenas iterações, que começam pela implementação de um caso de teste, depois pelo código necessário para executar o testes e terminando com o aperfeiçoamento do código criado".

O TDD é uma abordagem iterativa usada para desenvolver software e seu princípio básico se baseia em escrever testes automatizados para a funcionalidade antes de ser implementada. Mas você deve estar se perguntando: como testar antes da funcionalidade ser implementada? A ideia consiste em aplicar pequenos ciclos iterativos de teste-codificação-refatoração (*red-green-refactor*) conforme a funcionalidade proposta for sendo desenvolvida.

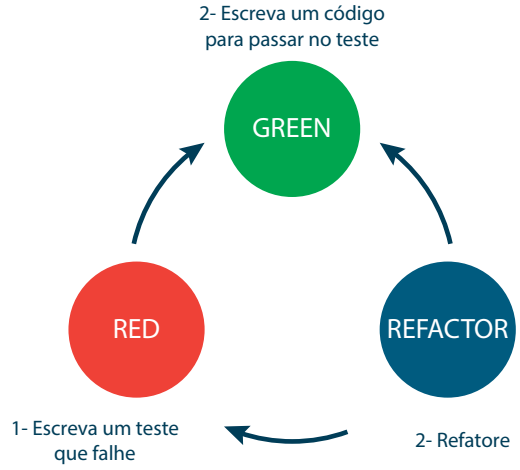


Figura 4 - Os ciclos iterativos do TDD
Fonte: a autora.

Para Ribeiro os ciclos iterativos de teste-codificação-refatoração do TDD funcionam da seguinte forma:

> [...] o vermelho simboliza a fase inicial do ciclo TDD onde o teste escrito falha. Ele falha porque o sistema não está funcional; ele não possui as funcionalidades que desejamos que ele tenha. Em alguns ambientes de desenvolvimento, essa falha é evidenciada através da exibição de uma barra vermelha. Na segunda fase implementamos as funcionalidades que faltavam para fazer todos os testes passarem, ou seja, os testes que já existiam e o novo teste recém-introduzido. Neste momento, a barra visual deve se tornar verde. Somente se passa para a próxima etapa quando nenhum teste estiver falho. Na última parte do ciclo é feita a refatoração. Refinamos o desenho do código sem alterarmos seu comportamento externo, mantendo todos os testes passando e a com a barra visual exibindo a cor verde (RIBEIRO, 2010, pg. 5).

Vimos na Unidade II que na metodologia XP (*Extreme Programming*) os programadores utilizam o TDD como parte do conjunto de práticas, onde eles iniciam escrevendo um teste de unidade que falhe para alguma funcionalidade que ainda não foi desenvolvida, depois, caso o teste de unidade falhe, eles então escrevem o código para fazer o teste passar. Quando o teste passar é refatorado o código, onde é enxugado e eliminado códigos-fonte duplicados ou malfeito, fazendo com que o código fique mais legível e sem erros.

Segundo Ferreira (2011) no TDD temos alguns pontos que devem ser observados quando um teste é executado:

- Muitas linhas de código para um objeto, pode haver algo de errado.
- Não sendo encontrado um lugar comum para o código de inicialização, significa que existem muitos objetos fortemente acoplados.
- Testes que se quebram inesperadamente mostram partes da aplicação sendo afetada por outra parte. Necessário eliminar essas quebras.

Os testes executados devem seguir um modelo chamado de *F.I.R.S.T.,* que se resume em:

- **F** (Fast) Testes rápidos que testam apenas uma unidade.
- **I** (Isolated) Testes unitários isolados, ou seja testes individuais.
- **R** (Repeated) Teste de repetição para avaliar comportamento.

- **S** (Self-verifying) Teste para verificar se passou ou se deu falha no teste.
- **T** (Timely) Teste deve um por unidade.

Segundo Sbrocco (2012, pg. 236), "o TDD apresenta vários benefícios, como a eliminação do medo relacionado à alteração de algo que funciona, considerando que um novo problema pode ser introduzido". Quando adotamos o TDD como um padrão no desenvolvimento de software, aumentamos o nível de compreensão do código escrito pelos desenvolvedores e, com isso, reduzimos a complexidade do software.

REFLITA

É importante ressaltar que o TDD não é um método para testar software, mas sim para construí-lo.

(José Henrique Teixeira Carvalho Sbrocco)

SAIBA MAIS

O uso do TDD diminui muito o custo com a manutenção do software após, o mesmo, ser entregue para o cliente, correções de bugs, inserção de novas funcionalidades serão realizados em um tempo muito menor. O tempo de desenvolvimento utilizando o TDD para um desenvolvedor novo na metodologia será maior, porém após um tempo utilizando a metodologia o desenvolvedor ficará bem mais ágil, aumentando assim sua produtividade e a qualidade de seus testes, ocasionando assim o aumento da qualidade do software final. No final do Projeto o uso do TDD se dá como muito válido, pois o tempo com manutenção será bem menor, a satisfação do cliente a cada entrega de módulos (devido à entrega contínua) será maior, além da maior confiança dos desenvolvedores em seu código dão ao software final uma qualidade muito mais significativa do que um software que não foi desenvolvido utilizando o TDD.

Fonte: Ferreira (2011).

LEAN SOFTWARE DEVELOPMENT (LD)

A metodologia Lean Software Development (LD), conforme Sbrocco (2012, pg. 237) "foi inspirada em metodologias desenvolvidas pela Toyota. Tem como foco a eliminação de desperdício, a excelência na qualidade e também o aumento da velocidade dos processos".

Segundo Fadel e Silveira (2010) a metodologia Lean procura fornecer algo com valor aos clientes, com custos mais baixos, com melhoria constante do processo e do suporte por meio do envolvimento de pessoas qualificadas, motivadas e com iniciativa. Para os autores Fadel e Silveira (2010, pg. 08) "o pensamento Lean foca em oferecer o que o cliente quer, onde e quando quiserem sem haver qualquer desperdício".

PRINCÍPIOS DO LEAN (LD)

A metodologia LD descreve sete princípios para serem usados durante o desenvolvimento de software enxuto (SILVA, 2011), como mostra o quadro a seguir.

Quadro 1 - Princípios e Ferramentas do LD

Princípio
P1-Eliminar desperdícios
P2- Amplificar Aprendizado
P3- Decidir o mais tarde possível
P4- Entregar o mais rápido possível
P5- Dê poder à equipe
P6- Construir com integridade
P7- Ver o todo

Fonte: Silva (2011).

- **P1- Eliminar desperdícios:** desperdícios (atividade sem valor ao cliente) devem ser eliminados. Dentro desse princípio temos sete desperdícios que devem ser eliminados:

  D1: Trabalho Parcialmente Executado: requisito que não foi integrado, executado pelo cliente, validade e aceito é considerado uma software que foi parcialmente executado.

  D2- Processos Extra: na metodologia LD documentação é considerado um processo desnecessário, um desperdício, pois fica defasado e obsoleto gerando retrabalho. A metodologia LD aceita documentos desejados pelo cliente, pequenos, de alto nível e que sejam produzidos depois do processo.

  D3- Funcionalidades Extra: funcionalidade que não são solicitadas pelo cliente, são desperdícios.

  D4- Chaveamento de Tarefas: desenvolvedor trabalhando em várias tarefas, fazendo que elas demorem mais do que o necessário.

  D5- Espera: como o objetivo é o desenvolvimento mais rápido possível, esperas desnecessárias devem ser eliminadas, como por exemplo: muitas reuniões no início do projeto, documentação, revisões, aprovações, testes para entrega de Software.

  D6- Movimento: para melhorar a concentração dos desenvolvedores deve-se diminuir a movimentação física e a movimentação de artefatos entre desenvolvedores.

  D7- Defeitos: procurar descobrir defeitos imediatamente para não haver perda de tempo e impactos negativos no cliente.

- **P2- Amplificar Aprendizado:** devemos aprender junto ao cliente desenvolvendo, de forma crescente, gerando resultados e obtendo feedbacks, assim minimizamos riscos e aumentamos a qualidade.

- **P3- Decidir o mais tarde possível:** decisões de projeto devem ser tomadas o mais tarde possível, até que seja obrigatória tomar uma decisão.

- **P4- Entregar o mais cedo possível:** entregar o mais rápido possível um software, temos menos riscos de mudanças e menor trabalho para consertar erros encontrados.

- **P5- Dar poder à equipe:** o desenvolvedor deve assumir responsabilidade por tomar decisões técnicas.
- **P6- Construir com integridade:** identificação do valor desejado pelo cliente é essencial para a construção e desenvolvimento do software.
- **P7- Ver o todo:** necessário pensar em desempenho, otimização e melhorias do processo o tempo todo, durante o desenvolvimento do software.

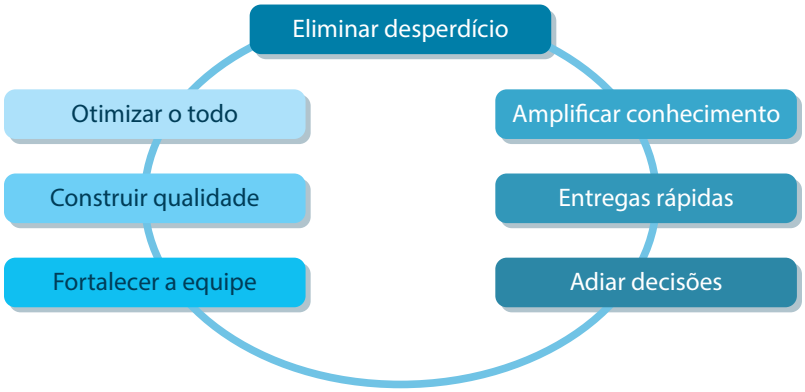


Figura 5 - Princípios do Lean Software Development
Fonte: a autora.

A metodologia Lean é considerada uma filosofia de gestão, onde acredita que toda ação que não tem um valor ao cliente é considerada um desperdício. Quando a metodologia Lean é aplicada corretamente, o processo de desenvolvimento é de alta qualidade, realizada rapidamente e com um baixo custo.

KANBAN

Kanban é um termo japonês, que conforme Silva et al (2012, pg. 337), ”significa sinal visual. Uma das grandes características deste método é evidenciar os problemas existentes no processo”. O Kanban é considerado uma ferramenta visual que ajuda no acompanhamento do Fluxo de Trabalho e do controle WIP (*Work in Progress*) ou Trabalho em Progresso (RIBEIRO e SOUSA, 2015).

Mas segundo Boeg (2012, pg. 5, On-line)[2], “Kanban é um método de gestão de mudanças. O que o Kanban faz, em primeiro lugar, é servir como um catalisador para introduzir ideias Lean na entrega de sistemas de software”.

Das metodologias ágeis para desenvolvimento de software que já estudamos, o método Kanban é o considerado menos prescritivo, sem muitas regras. Essa característica tem estimulado as equipes de desenvolvimento a adotarem esse método (SILVA et al, 2012). O Kanban dá ênfase aos seguintes princípios:

- Visualizar o fluxo de trabalho (*workflow*).
- Acompanhar cada passo da cadeia de valor do software, do início ao fim do seu desenvolvimento.
- Visualizar e limitar a quantidade de Trabalho em Progresso (WIP) em cada fase.
- Lembrar sempre das políticas que estão sendo seguidas.
- Medir e gerenciar o Fluxo de Trabalho.
- Identificar as melhorias que podem ser feitas (cultura *Kaizen* - melhoria contínua).

Considerado um método mais adaptativo do que prescritivo, o Kanban acabou se tornando bastante empírico. Ou seja, as fases do processo e os valores limitados de cada item de trabalho devem ser testados pela equipe de desenvolvimento até encontrarem o valor ideal do WIP. No Kanban não temos uma fórmula para chegar a esse valor ideal, restante a equipe de desenvolvimento experimentar e encontrar números que melhor se adaptam a seu projeto (SILVA et al, 2012).

O método Kanban, segundo Silva et al, após ter sido dada às prescrições:

> [...] percebe-se que o Kanban baseia-se em um processo onde o fluxo de trabalho é contínuo, ou seja, diferente de Scrum, não há um ciclo de trabalho definido onde, conhecida uma atividade, a mesma é estimada e colocada neste ciclo. Kanban controla as entradas de itens de trabalho e a vazão que é dada de acordo com o WIP definido. A esta vazão dos itens de trabalho dá-se o nome de *leadtime*, que representa o tempo de um item de trabalho desde a sua entrada no fluxo de trabalho mapeado até a sua saída. Kanban permite a combinação de ferramentas de diversos métodos até se obter o processo adequado (SILVA et al, 2012, pg. 342).

Para entender e aplicar os conceitos na prática, segundo Boeg (2012, on-line)[2], é necessário conhecer alguns passos do Kanban e segui-los. A seguir os passos do Kanban:

- Compreender o seus sistema de entregas e mapear a cadeia de valor.
- Definir e limitar o Trabalho em Progresso (WIP) para as fases. O WIP descreve o total de trabalho em progresso no sistema Kanban.
- Definir e entender as classes de serviço e seus critérios de seleção e ajustar as cadências para cada tipo de atividade. No Kanban, os itens de trabalho são tratados de acordo com suas características e esse tratamento é chamado de classes de serviço.
- Procure acompanhar o tempo de entrega por meio de um quadro, chamado de quadro Kanban.
- Procure mudar o Trabalho em progresso (WIP) e veja o impacto que ele causa no fluxo da entrega.

O método Kanban auxilia a equipe a assimilar e controlar o progresso das tarefas de uma forma visual. O funcionamento do Kanban funciona, normalmente, com o uso de um quadro branco e com alguns pequenos papéis (Post-it) que possam ser colados. Os papéis vão representar as tarefas a serem executadas pela equipe de desenvolvimento e devem ser separadas por cores, para facilitar a identificação. Ao término de cada tarefa, cada papel é empurrado para a etapa seguinte até que a mesma seja finalizada novamente. Quando analisamos um quadro Kanban fica fácil visualizar o fluxo de trabalho da equipe e o seu progresso. Na imagem a seguir, podemos visualizar um exemplo de um quadro Kanban.

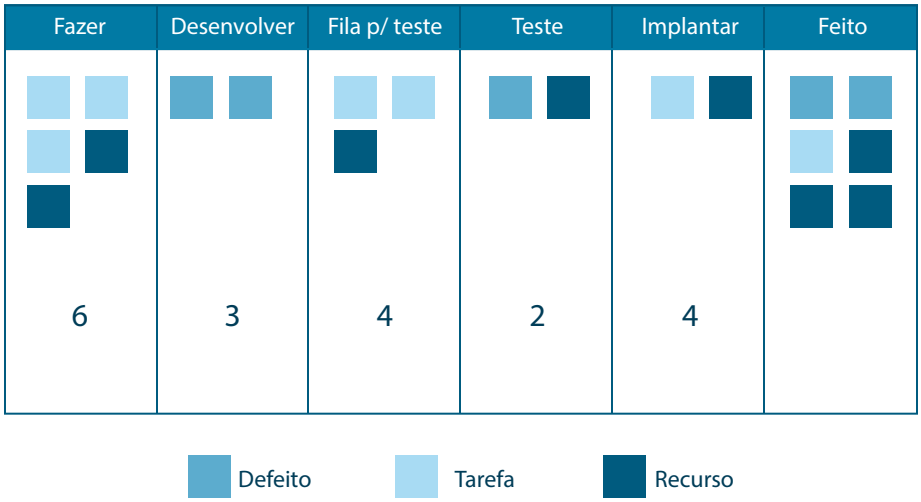


Figura 6 - Exemplo de Quadro Kanban
Fonte: Kleber (2014, on-line)[3].

No exemplo de Quadro Kanban, temos 6 colunas divididas em: Fazer, Desenvolver, Fila para Teste, Teste, Implantar e Feito). Para cada coluna, temos os papéis com as tarefas a serem executadas e estas devem ser escritas de forma simples e de fácil entendimento. A priorização das tarefas podem ser feitas utilizando cores para facilitar a visualização e para que a equipe saiba em que trabalhar primeiro. m cada coluna existe um número. Para cada coluna temos um número que representa o limite WIP, ou seja, a quantidade de tarefas em andamento que pode estar naquela coluna e ser executada pela equipe.

SAIBA MAIS

Mitos e fatos do Kanban

Mito: o Kanban é adequado apenas para equipes trabalhando com tarefas pequenas e padronizadas, como as de operações e manutenção.

> **Fato**: o Kanban é altamente inspirado pelo trabalho de Don Reinertsen, sobre o Desenvolvimento Lean de Produtos. E se provou ser tão boa opção para o desenvolvimento de software quanto é para operações e manutenção.

Mito: o Kanban e o Scrum são opostos.

> **Fato**: nenhum dos princípios do Kanban restringem o uso do Scrum. O Kanban funciona como um agente de mudanças e os princípios do Scrum devem, portanto, ser usados apenas nos casos em que ajudam a otimizar o fluxo de trabalho [...].

Mito: as equipes que utilizam Kanban não utilizam períodos fixos de tempo (timeboxes).

> **Fato**: os timeboxes não são exigidos no Kanban, mas devem ser utilizados quando ajudarem a otimizar o fluxo, o nível de feedback e a qualidade.

Mito: as equipes Kanban não fazem estimativas.

> **Fato**: as estimativas não são obrigatórias, mas devem ser usadas quando apropriadas.

Fonte: Boeg (2012, On-line)2.

O foco do método Kanban, conforme Boeg (2012, on-line)2 é conduzir mudanças evolucionárias de forma diferente que a maioria dos métodos ágeis, fazendo com que ele se torne o método ideal para ser utilizado com qualquer conjunto de processo existente, desde o Scrum até o modelo em Cascata.

O método Kanban permite a combinação de outras ferramentas de diversos métodos para juntos chegarem ao processo de desenvolvimento adequado, pois ele estimula a melhoria contínua do processo.

CONSIDERAÇÕES FINAIS

Prezado(a) aluno(a)! Chegamos ao final da última unidade do nosso livro. Estudamos outras metodologias ágeis de desenvolvimento presentes no mercado e esperamos que a partir do conhecimento adquirido com essas metodologia ágeis, você procure pesquisar mais, estudar mais, além das que foram citadas no livro, pois sempre surgem novas metodologias ágeis e técnicas utilizadas para otimizar o trabalho de desenvolvimento de software.

O mercado de metodologias ágeis e técnicas usadas para o gerenciamento de softwares é amplo e com ótimas opções. Portanto, quanto mais conhecimento você tiver sobre elas, mas fácil se torna seu trabalho, pois muitas delas ajudam a otimizar o tempo e melhorar a qualidade, além de evitar problemas durante o desenvolvimento dos seus projetos.

Em muitas empresas, a junção de metodologias ágeis vem apresentando bons resultados no desenvolvimento de software. A união de práticas de distintas metodologias ágeis pode ajudar a agregar mais conhecimento e qualidade ao desenvolvimento de software.

Nesta unidade foi apresentada uma visão geral de conceitos básicos e princípios das metodologias ágeis: OpenUp (Processo Unificado Aberto), Metodologia Crystal, Test Driven Development (TDD), Lean Software Development (LD) e Kanban.

Finalmente, para encerrar a nossa unidade, lembre-se: para ser um bom profissional na área de gerenciamento de software é necessário ser multidisciplinar e, além do que foi aprendido, procurar estudar sempre, pesquisar e aprender, além das metodologias ágeis que estão citadas no livro.

Procure sempre testar diferentes metodologias, técnicas e abordagens até encontrar a que funciona melhor para você, sua empresa, sua equipe e seus clientes. Se desafie a implantar metodologias ágeis em seu local de trabalho e com a sua equipe.

Use a sua imaginação, a sua criatividade e inove! Desejamos sucesso em todos os seus projetos!

ATIVIDADES

1. As metodologias ágeis para o desenvolvimento de software surgiram para suprir as necessidades do mercado, como: inovação e redução de prazos na execução de projetos. As metodologias tradicionais começaram a ser questionadas, devido ao seu alto grau de documentação. **A partir dessa informação, avalie a opção correta:**

   a) A metodologia Lean procura fornecer algo com valor aos clientes com custos mais baixos, com melhoria constante do processo, mas não dá suporte aos envolvidos no processo.

   b) O método Crystal tem seu foco nos talentos e nas habilidades das pessoas envolvidas no projeto, permitindo que o desenvolvimento ocorra conforme as características da impostas pela empresa, sem levar em consideração as características da equipe.

   c) A metodologia OpenUp pode ser utilizada para o desenvolvimento de qualquer tipo de projeto com uma maior agilidade, mas não pode ser utilizado com qualquer tipo de recurso.

   d) O funcionamento do Kanban funciona, normalmente, com o uso de um quadro branco e com alguns pequenos papéis (Post-it) que possam ser colados.

   e) O TDD é uma abordagem iterativa usada para desenvolver software e seu princípio básico se baseia em escrever testes automatizados para a funcionalidade depois de ser implementada.

2. A metodologia Lean é considerada uma filosofia de gestão, onde acredita que toda ação que não tem um valor ao cliente, é considerada um desperdício. Quando a metodologia Lean é aplicada corretamente, o processo de desenvolvimento é de alta qualidade, realizado rapidamente e com um baixo custo. **Com base nestas informações, cite e descreva os princípios da metodologia Lean Software Development (LD).**

ATIVIDADES

3. O TDD ou *Test Driven Development* é uma técnica de desenvolvimento orientada a testes, proposta por Kent Beck em 2003, considerada uma metodologia de desenvolvimento de software ágil derivada da metodologia Extreme Programming (XP) e do Agile Manifesto. **Com base nessa informação, avalie qual das opções está correta para o projeto de desenvolvimento de software que utilize TDD:**

   a) Deve realizar *sprints* a cada quinzena.

   b) Deve desenvolver pequenos *releases* no decorrer do projeto.

   c) Deve apresentar uma grande quantidade de testes unitários de código-fonte que foram previamente desenvolvidos.

   d) Deve apresentar em seu desenvolvimento uma linguagem de programação estruturada.

   e) Deve recomendar que os testes sejam preparados antes do desenvolvimento do código.

4. A metodologia Open UP possui alguns princípios fundamentais que compõem os pilares de seu processo e que define as suas intençõe. Esses princípios são fundamentais para a interpretação dos conceitos de Papéis, Produtos de Trabalho e as Tarefas que os geram. **Cite e descreva os sete papéis propostos pelo OpenUp.**

5. O Kanban é considerada uma ferramenta visual e um método ágil de desenvolvimento de software que tem como uma de suas práticas o gerenciamento do fluxo de trabalho, que deve ser monitorado, medido e reportado a cada fase do fluxo. **Com base nessas informações, quais os princípios que o Kanban dá ênfase?**.

OPENUP

Openup é uma metodologia que segue o Processo Unificado de engenharia de software porém com uma cultura aberta e enxuta, ele utiliza a filosofia ágil a qual foca na natureza colaborativa de desenvolvimento de software. A metodologia é construída através de processos simples os quais podem ser expandidos para quaisquer tipos de softwares.

O Openup foi criado como um projeto de exemplo aplicando as melhores práticas de engenharia de software do EPF, assim sendo, o Openup é um conjunto das metodologias ágeis como Scrum e XP (Extreme Programming) somado a um pouco da arquitetura do Processo Unificado.

Openup possui seu núcleo em desenvolvimento aberto pela comunidade, assim ele pode ser facilmente adaptado para qualquer tipo de desenvolvimento de software, torna-se possível também adicionar práticas e processos únicos aplicados pela cultura organizacional da empresa.

Em seu ciclo de vida existem micro incrementos organizados visando diminuir o feedback.

ESCOPOS DO FRAMEWORK

A melhor maneira de entender o Openup é saber quais são as necessidades e os objetivos da equipe para desenvolver o software, uma vez que o Openup é aberto para alterações é possível que seja facilmente adaptável para equipe ou pode-se escolher um escopo de conhecimento mínimo, porém, para chegar a esta conclusão se faz necessário que sejam respondidas as três perguntas [...]

1. Faz-se necessário somente o mínimo dos processos para atingir algum valor?
2. Faz-se necessário se não utilizar alguns produtos de trabalho a fim de não ser sobrecarregado?
3. Faz-se necessário utilizar um processo que pode ser alterado e extensível para necessidades adicionais que podem ocorrer durante a execução do projeto?

Almejando a capacidade de aplicação ao maior número de projetos possíveis, bem como, à maior gama de equipes possíveis, independentemente de suas culturas de trabalho o Openup busca atingir as seguintes características [...]:

Metodologia Mínima: Somente os conteúdos fundamentais são adicionados.

Metodologia Completa: É o conjunto de processos completos para desenvolver um sistema.

Metodologia Extensível: Podem ser utilizados como uma base para outros processos criados pela própria empresa onde cada processo pode ser alterado de acordo as necessidades. Desta maneira, torna-se possível limitar ou estender a metodologia facilmen-

te, quando, por exemplo, não há grandes recursos disponíveis ou o projeto não possui grande complexidade pode-se utilizar todos os conteúdos que são ditos pelo Openup como essenciais. **DISCIPLINAS**

O grupo de ações a serem realizadas durante o ciclo de vida do projeto é dividido em várias disciplinas e essas representam os produtos de trabalho que devem ser completados antes do término do ciclo, cada disciplina dependendo do estágio no qual o projeto se encontra pode possuir uma ação e contribuição da equipe do que as outras disciplinas. A disciplina de **Requisitos** agrupa todas as tarefas relativas ao processo de Análise de Negócios e de Análise e Especificação de Requisitos do Openup, tendo maior ênfase na fase de Iniciação do Processo. Para atingir os objetivos finais do projeto e a satisfação da parte interessada é necessário que se tenha bem estabelecidos logo no começo do projeto as metas, para isso é necessário identificar os stakeholders, possíveis riscos e não só as necessidades que o mesmo relata, mas também se devem identificar outras necessidades que possam ser levantadas durante a execução do projeto. Com base nessa disciplina são tomadas as decisões para os estágios futuros e as decisões de arquitetura do projeto.

A disciplina de **Arquitetura** tem como principal objetivo apresentar uma arquitetura estável para o desenvolvimento do sistema, baseando-se nos requisitos especificados. A disciplina de Arquitetura tem maior ênfase na fase de Elaboração. Neste estágio são convertidos os requisitos e necessidades levantadas no design do próprio projeto e, assim são definidas as alterações na arquitetura para suprir as necessidades do projeto.

A disciplina de **Implementação** organiza as tarefas que irão transformar a arquitetura proposta na implementação do sistema, buscando atender os requisitos definidos pelos stakeholders. Nessa fase são desenvolvidos os requisitos utilizando como base de informação os documentos gerados na fase da elaboração, os requisitos são transformados em componentes executáveis e testáveis pelos clientes. A disciplina de **Teste** agrupa as tarefas relacionadas a teste, que se preocupa em prover feedback sobre a maturidade do sistema, projetando, implementando, executando e avaliando testes. Essa disciplina visa criar projetos de qualidade aceitáveis pelos clientes realizando testes precoces a cada iteração e micro incrementos também são realizados testes de capacidade para ver se os sistemas conseguem suprir as necessidades do ambiente no qual será utilizado. A disciplina de **Gerência de Projeto** tem como objetivo apresentar técnicas para que o gerente de projetos possa liderar facilitar e oferecer suporte à sua equipe, auxiliando-a a lidar com os riscos e obstáculos encontrados durante o processo de desenvolvimento de software. Nesta disciplina é necessário que haja um fator líder na equipe, esse líder deve motivar os membros desenvolvedores e analistas assim como enfrentar a parte interessada no projeto em casos de riscos identificando requisitos desnecessários e que possam atrasar o projeto. É o papel do gerente do projeto estabelecer metas e ações corretivas a fim de sempre proporcionar um ambiente colaborativo e um foco para atender todas as soluções propostas.

Fonte: DIEDRICH (2011).

MATERIAL COMPLEMENTAR

LIVRO

Test-Driven Development -Teste e Design no Mundo Real
Mauricio Aniche

Editora: Casa do Código
Sinopse: por que não testamos software? Porque é caro? Porque é demorado? Porque é chato? Testes automatizados são a solução para todos esses problemas. Aprenda a escrever um programa que testa seu programa de forma rápida, barata e produtiva, e aumente a qualidade do seu produto final. Neste livro, você aprenderá sobre TDD, uma das práticas ágeis de desenvolvimento de software mais populares, por meio da linguagem Java, mas poderá aplicar o conceito aprendido em qualquer outra linguagem. TDD faz o desenvolvedor escrever o teste antes mesmo de implementar o código. Essa simples inversão na maneira de se trabalhar faz com o que o desenvolvedor escreva código mais testado, com menos bugs, e inclusive com mais qualidade. Seja profissional, teste seu software! Todos os exemplos deste livro foram escritos em Java.

LIVRO

Título: Kanban - A Simplicidade do Controle da Produção
Reinaldo A. Moura

Editora: Imam
Sinopse: origem do Sistema Toyota de Produção e do Sistema Kanban: uma Visão do Sistema 'Just-In-Time', Sistemas de Controle da Produção, Kanban: um Sistema para Estimular a Produtividade, Kanban e os Inventários, Implementação do Sistema Kanban· Kanban - MRP - MRP II, MRP Sincronizado, Estudo de Caso.

NA WEB

TDD: fundamentos do desenvolvimento orientado a testes
Artigo que fala sobre os fundamentos de TDD (Test Driven Development / Desenvolvimento orientado a teste), bem como o motivo de utilizar a metodologia para o desenvolvimento de sistemas. Excelente artigo. Disponível em: <http://www.devmedia.com.br/tdd-fundamentos-do-desenvolvimento-orientado-a-testes/28151>.

Kanban: Do início ao fim!
Artigo que mostra o funcionamento do Kanban do início ao fim, seus conceitos, princípios e benefícios. Comenta também como o Kanban vem ganhando respeito na comunidade de desenvolvimento de software e mais empresas passaram a adotá-lo. Ótimo artigo, vale a pena conferir. Disponível em: <https://www.culturaagil.com.br/kanban-do-inicio-ao-fim/>.

REFERÊNCIAS

DIEDRICH, L. G. **Integração da Metodologia Ágil OpenUp nos Processos de Engenharia de Software.** Medianeira: UTFPR, 2011.

FADEL, A. C.; SILVEIRA, H. da M. **Metodologias Ágeis no contexto de Desenvolvimento de Software: XP, Scrum e Lean.** São Paulo: UNICAMP, 2010.

FERREIRA, A. R. **Utilização da metodologia TDD para desenvolvimento de software**. 2011. Marília: Centro Universitário Eurípides de Marília, Fundação de Ensino "Eurípides Soares da Rocha", 2011.

FILHO, D. L. B. **Experiências com desenvolvimento ágil**. São Paulo: Universidade de São Paulo (USP), 2008.

GAVA, K. A. **Adaptação Processo OpenUp para o Desenvolvimento de Sistemas Seguros**. São Paulo: Pontifícia Universidade Católica de São Paulo, 2009.

RIBEIRO, C. L. **A Relação entre Desenvolvimento Orientado a Testes e Qualidade de Software**. Goiânia: Instituto de Informática – Pontifícia Universidade Católica de Goiás (PUC Goiás), 2010.

RIBEIRO, R. D.; SOUSA, H. da C. **Gerenciamento de Projetos com Métodos Ágeis**. Rio de Janeiro: Ed. Horácio da Cunha e Sousa Ribeiro: 2015. Disponível em: <http://www.rafaeldiasribeiro.com.br/downloads/livros/livro_metodosageis.pdf>.Acesso em: 19 abr. 2017.

SILVA, F. L. de C. **Mapeamento e Aplicação da Produção Enxuta para o Processo de Desenvolvimento de Software.** Rio de Janeiro: UENF, 2011.

SILVA, D. V. de S.; SANTOS, A. de O.; NETO P. S. Os benefícios do uso de Kanban na Gerência de Projeto de Manutenção de Software. In: **VIII Simpósio Brasileiro de Sistemas de Informação**, SBSI, 2012.

SBROCCO, J. H. T. C.; MACEDO P. C. de. **Metodologias Ágeis:** Engenharia de Software sob medida. 1 ed. São Paulo: Érica, 2012

REFERÊNCIAS ON-LINE

[1]Em: <http://slideplayer.com.br/slide/2459776/>. Acesso em: 19 abr. 2017.

[2]Em: <https://www.infoq.com/br/minibooks/priming-kanban-jesper-boeg>. Acesso em: 19 abr. 2017.

[3]Em: <https://www.culturaagil.com.br/kanban-do-inicio-ao-fim/>. Acesso em: 19 abr. 2017.

GABARITO

1. d

2. Os princípios da metodologia Lean são:

P1- Eliminar desperdícios: desperdícios (atividade sem valor ao cliente) devem ser eliminados.

P2- Amplificar Aprendizado: devemos aprender junto ao cliente desenvolvendo de forma crescente, gerando resultados e obtendo feedbacks, assim minimizamos riscos e aumentamos a qualidade.

P3- Decidir o mais tarde possível: decisões de projeto devem ser tomadas o mais tarde possível, até que seja obrigatório tomar uma decisão.

P4- Entregar o mais cedo possível: entregar o mais rápido possível um software, temos menos riscos de mudanças e menor trabalho para consertar erros encontrados.

P5- Dar poder à equipe: o desenvolvedor deve assumir responsabilidade por tomar decisões técnicas.

P6- Construir com integridade: identificação do valor desejado pelo cliente é essencial para a construção e desenvolvimento do software.

P7- Ver o todo: necessário pensar em desempenho, otimização e melhorias do processo o tempo todo, durante o desenvolvimento do software.

3. e) Deve recomendar que os testes sejam preparados antes do desenvolvimento do código.

4. Os sete papéis da metodologia Open Up são:

- ***Stakeholder* :** se comunica e colabora com o *Management* (Gerente de Projeto) e com os analistas.
- **Analista (*Analyst*):** coleta informações com os *stakeholder* e seu foco principal é a Intenção (*Intent*).
- **Arquiteto (*Architect*)**: responsável pela arquitetura, design e a codificação do projeto. Ele se comunica com o *Management* e com os desenvolvedores.
- **Desenvolvedor (*Developer*)**: responsável por escrever os códigos, os testes unitários e a integração entre os componentes. Seu foco é a solução (*Solution*).
- **Testador (*Tester*):** responsável pelos testes no código. Se comunica com o Analista (*Intent*) e com os desenvolvedores (*Solution*)
- **Gerente de Projeto *(Project Manager)***: responsável por liderar, planejar, coordenar os *stakeholders* e o time do projeto.
- **Qualquer papel (*Any Role*)**: qualquer membro da equipe que pode executar tarefas gerais.

5. O Kanban dá ênfase aos seguintes princípios:

- Visualizar o fluxo de trabalho (*workflow*).
- Acompanhar cada passo da cadeia de valor do software, do início ao fim do seu desenvolvimento.
- Visualizar e limitar a quantidade de Trabalho em Progresso (WIP) em cada fase.
- Lembrar sempre das políticas que estão sendo seguidas.
- Medir e gerenciar o Fluxo de Trabalho.
- Identificar as melhorias que podem ser feitas (cultura *Kaizen* - melhoria contínua).

CONCLUSÃO

Chegamos ao final mais uma etapa! Espero que você tenha aprendido um pouco mais com os temas abordados ao longos das unidades e espero que contribuam com a sua formação. Espero ter esclarecido suas dúvidas e despertado em você a curiosidade de se aprofundar mais neste conteúdo e a vontade de colocar o que aprendeu em prática.

Quando falamos em gerenciamento de *software* o mais importante é entender os conceitos, as práticas e o funcionamento das metodologias ágeis e como elas são necessárias para o processo de desenvolvimento de um sistema com qualidade e que obtenha sucesso junto ao cliente.

Foram apresentados conceitos teóricos importantes, como processos e projetos, seguidos de tópicos relacionados com as metodologias ágeis da área da engenharia de software, os quais precisam ser entendidos. Bem como, uma visão comparativa entre as metodologias tradicionais e as metodologias ágeis, mostrando que essa comparação pode ser utilizada como pré-requisito para o entendimento dos critérios que ajudam na escolha da metodologia ágil que melhor se ajusta às necessidades de determinado projeto.

Aprendemos ao longo das unidades várias metodologias ágeis como: XP, *Scrum*, Kanban, FDD, DSDM, Iconix Process, LD, TDD entre outras. Fica a dica: vale a pena estudar, pesquisar e procurar sempre testar diferentes metodologias ágeis e abordagens que surjam até encontrar a que funciona melhor para você, sua equipe e seus clientes.

Para ser um engenheiro de software é necessário ser multidisciplinar e, além do que foi aprendido, procurar estudar, sempre, pesquisar e aprender, além do que foi citado no livro. Continue sua busca por conhecimento pois este livro representa somente o primeiro passo da sua jornada rumo ao sucesso.

Espero ter alcançado o objetivo proposto, que era mostrar a importância do Gerenciamento de Software e desejo que o aprendizado adquirido com o conteúdo apresentado ao longo das unidades lhe garanta sucesso e realização na sua vida profissional. Muito obrigada pela atenção em todas as unidades e até uma próxima.

Profª. Esp. Janaina.