DEDMD

LIVRO

UNIDADE 2

Engenharia de Software

Desenvolvimento ágil de *software*

Roque Maitino Neto

2016
Editora e Distribuidora Educacional S.A
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: http://www.kroton.com.br/

Sumário

DESENVOLVIMENTO ÁGIL DE *SOFTWARE*

Convite ao estudo

Olá! Seja bem-vindo à segunda unidade do curso de Engenharia de Software.

Foi durante a primeira parte dos estudos que tivemos a oportunidade de estruturar os meios de produção de software da X-Soft e transformá-la em uma empresa organizada do ponto de vista metodológico. Por conta de sua implantação descomplicada, a metodologia em cascata foi escolhida e, de certa forma, cumpriu seu papel de conferir ordem a um processo até então caótico. Acontece, no entanto, que a possibilidade de se organizar uma rotina de desenvolvimento está longe de ficar restrita a uma só metodologia. Além disso, por mais conhecido e utilizado que seja, o meio tradicional apresenta falhas graves na comunicação com o cliente e períodos excessivamente longos entre a produção e validação dos produtos entregues.

Pois bem, chegou o momento de conhecermos algumas metodologias mais modernas e ágeis de desenvolvimento de software. Elas reúnem segmentos bem-sucedidos das metodologias mais antigas, suprimem hábitos ineficientes e incorporam práticas que dão agilidade ao processo, preveem interação constante entre cliente e equipe, aprimoram métodos de teste e, enfim, tornam mais viável e seguro o caminho até um produto de boa qualidade.

Vale a pena, antes de tudo, resgatarmos a competência geral e introduzirmos novas competências e objetivos desta unidade. Em aspectos gerais, nosso estudo visa apresentar um conteúdo que te permitirá conhecer as principais metodologias de desenvolvimento de software, normas

de qualidade e processos de teste de software. Em especial, podemos destacar a competência técnica desta unidade de ensino como sendo o conhecimento das principais metodologias ágeis de desenvolvimento e a habilidade em contrapô-las com a metodologia tradicional.

Esta segunda unidade estudará os métodos ágeis e suas características. Assim será possível realizar a comparação entre os modelos tradicional e o ágil, além de apresentar os valores e práticas das metodologias XP (Extreme Programming), Scrum e FDD (Feature-Driven Development). Sua atuação prática ainda orbitará a X-Soft e sua missão é implantar metodologia ágil na X-Soft, em quatro etapas:

1. Levantar pontos frágeis da metodologia atual.

2. Planejar a introdução de práticas do XP relacionadas aos princípios da comunicação e feedback.

3. Planejar a introdução de práticas do XP relacionadas à integração contínua e testes.

4. Adotar práticas contínuas de aprimoramento do modelo e de encantamento de novos clientes.

Nas próximas páginas será detalhada a primeira parte de nossa missão, proposto conteúdo teórico sobre metodologias ágeis e novas abordagens da situação problema. Bom trabalho!

Seção 2.1

Introdução a metodologias ágeis e comparações com a tradicional

Diálogo aberto

Os conteúdos apresentados na unidade 1 abordaram os fundamentos da Engenharia de Software, passando pelo seu conceito, objetivos e princípios. Os processos de software receberam o devido destaque e fundamentaram o estudo do ciclo de vida tradicional de desenvolvimento de um produto de software. A exposição de todas as etapas deste ciclo deu a você a exata noção do modelo tradicional de criação de um sistema e te tornará apto a compará-lo com o que será estudado nesta unidade.

Por conta do sucesso do projeto de controle de clientes, a X-Soft assumiu outros compromissos com a empresa de games. Conforme a demanda cresce e a natureza das solicitações se concentra em torno de entregas mais rápidas, vai ficando mais clara a necessidade de adoção de procedimentos que acompanhem tais mudanças.

Embora a metodologia cascata tenha sido apropriada para determinada circunstância da vida da X-Soft, seus dirigentes entendem que é chegado o momento de adotarem práticas que se apoiem em contatos regulares e produtivos com o cliente, em produção de programas executáveis em menor tempo e em estimativas mais realistas.

Deste cenário naturalmente despontam as metodologias ágeis. Elas diferem dos processos mais tradicionais em muitos aspectos, mas sobretudo naqueles que compõem as aspirações da X-Soft.

Utilizando os temas que estão sendo abordados nesta seção e sua habilidade em diagnosticar procedimentos ativos, você deve criar um documento que contenha as fragilidades do processo atual, visando justamente justificar a mudança para uma modelo ágil.

Este documento também lhe servirá de base para providências futuras, incluindo a adoção das novas práticas. Eis o desafio!

Na seção "Não pode faltar" você terá contato com fundamentos das metodologias ágeis, comparações com o modelo tradicional e a descrição de alguns princípios e práticas do XP, Scrum e FDD.

Bom trabalho!

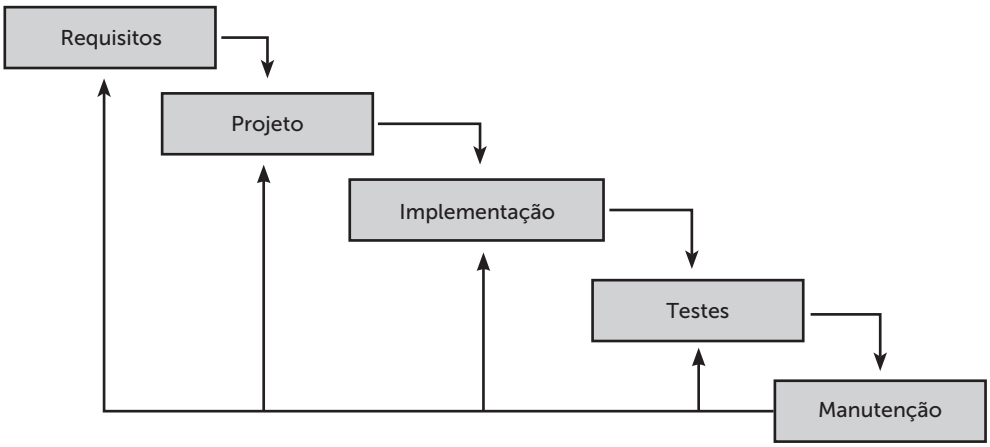
Não pode faltar

É possível que você conheça alguma organização que desenvolve softwares. Se conhece, é bastante provável que já tenha ouvido de algum membro desta organização que o caminho entre o início do projeto e a entrega do produto esteja sendo trilhado com dificuldade crescente e que, embora exista processo formal de desenvolvimento, nem sempre os clientes estão satisfeitos com os resultados. Quais são os problemas das metodologias tradicionais? Por que tais metodologias têm dificuldade em acompanhar as mudanças de planos de qualquer cliente em relação ao produto? Por que o cliente tem dificuldade em reconhecer valor no que está sendo desenvolvido? Mais do que responder a estas questões, esta seção tem a intenção de apontar soluções. Adiante com a introdução às Metodologias Ágeis.

Processo Tradicional de Desenvolvimento

Conforme estudamos na unidade 1, o processo tradicional de desenvolvimento baseia-se na construção linear do sistema, seguindo sequência definida de fases, como mostra a figura 2.1.

Figura 2.1 | etapas do modelo tradicional de desenvolvimento



Fonte: Adaptado de Teles (2004).

Além da linearidade, outras características – com raízes nas formas tradicionais de fabricação de bens de consumo – costumam estar presentes no desenvolvimento tradicional, cujo foco contempla o determinismo, especialização e foco na execução (TELES, 2004).

Para explicar estes conceitos cabe a analogia com o processo de montagem de veículos. Vejamos:

• Materiais alimentam um processo de fabricação. Ao final, temos um automóvel terminado. As alterações pelas quais os materiais passam são DETERMINÍSTICAS e devem sempre gerar um resultado conhecido. Acarreta segurança, redução de tempo e custo.

• A indústria tradicional divide o processo de montagem em inúmeras atividades ESPECIALIZADAS, desenvolvidas por trabalhadores igualmente especializados.

• FOCO NA EXECUÇÃO: a fórmula é simples: determinismo + especialização = não há o que pensar. Basta executar.

Não é difícil de inferir que o modo tradicional tenha sido concebido com base nessas ideias de desenvolvimento industrial em linha. De acordo com elas, a mera obediência a eventos consecutivos (de requisitos até implantação), a especialização (funções de analista, projetista, programador, testador) e o foco na execução seriam capazes de criar um produto de qualidade, no tempo estipulado e sem ultrapassar o orçamento.

Havia, e ainda há, a presunção de que a sequência de etapas do projeto será transformada corretamente em software (TELES, 2004).

Pesquise mais

Os benefícios da divisão de trabalho e da especialização para a produtividade do setor industrial são defendidos em um trecho do livro A Riqueza das Nações, escrito em 1776 por Adam Smith. Para conhecê-lo melhor, acesse https://economianostra.wordpress.com/2013/05/28/a-fabrica-de-alfinetes-de-adam-smith/. Acesso em: 15 nov. 2015.

Agora, considere a possibilidade de dividirmos nossos recursos humanos em dois tipos: trabalhadores manuais e trabalhadores do conhecimento. O primeiro tipo desempenha trabalhos repetitivos, pré-determinados e dependem principalmente das suas habilidades manuais e físicas para a execução de suas tarefas. Os trabalhadores do conhecimento, por sua vez, cumprem a missão com base em seu raciocínio, devem ter oportunidades de praticar sucessivas revisões em sua obra e não seguem processo linear em seus processos de criação.

Em qual dos dois tipos você classificaria um desenvolvedor de software? Acertou se pensou no trabalhador do conhecimento. No entanto, pela metodologia tradicional de desenvolvimento, são tratados como trabalhadores manuais. O erro é tratado como "pecado" e o medo de errar torna os desenvolvedores defensivos. Há pouca possibilidade de reverem sua obra depois de pronta e seu trabalho se baseia num processo linear (TELES, 2004).

Processo Ágil de Desenvolvimento

Em sua essência, os métodos ágeis têm menos ênfase nas definições de atividades e mais ênfase nos fatores humanos do desenvolvimento (WAZLAWICK, 2013). São claramente mais adequados à natureza do trabalho de profissionais de TI, já que se baseiam na necessidade de sucessivas revisões na obra. Atividades intelectuais não são executadas de forma linear e não são determinísticas.

Durante a construção de um software, há que se considerar uma infinidade de detalhes que nem sempre são lembrados logo na primeira oportunidade. Da mesma forma, ao tratar pela primeira vez das funcionalidades que deseja para o produto, o cliente ainda não as conhece por completo e não consegue ter visão global do que necessita. Que tal darmos a ele a oportunidade de aprender e mudar de ideia ao longo do processo de desenvolvimento?

Assimile

De acordo com documento intitulado "Manifesto Ágil", os métodos ágeis valorizam:

- indivíduos e interação entre eles mais que processos e ferramentas;
- software em funcionamento mais que documentação abrangente;
- colaboração com o cliente mais que negociação de contratos;
- responder a mudanças mais que seguir um plano (BECK, 2001. Disponível em: < http://www.manifestoagil.com.br/>. Acesso em: 15 nov. 2015).

Você certamente não se deparou com esta preocupação com o cliente quando estudou o modelo tradicional na unidade anterior.

"O aprendizado do qual estamos tratando decorre do feedback que o software fornece ao cliente quando este o manipula. No desenvolvimento ágil, o conceito de feedback está presente ao longo de todo o desenvolvimento do software e exerce um papel fundamental" (TELES, 2004, p. 42).

Como podemos proporcionar esta chance ao cliente? As práticas relacionadas aos métodos ágeis responderão. Nas próximas páginas serão oferecidas em linhas gerais três processos de desenvolvimento ágeis: Extreme Programming, Scrum e Feature-Driven Development.

Visão geral do *Extreme Programming* (XP)

O XP é uma metodologia adequada para projetos que possuem requisitos que se alteram constantemente, para equipes pequenas e para o desenvolvimento de programas orientados a objetos. É indicado também para ocasiões em que se deseja partes executáveis do programa logo no início do desenvolvimento e que ganhem novas funcionalidades assim que o projeto avança.

Reflita

O XP, assim como as outras metodologias ágeis, defende que a criação de um software segue a mesma dinâmica da criação de uma obra de arte. O trecho que segue ilustra este fato: "Escrever uma redação, um artigo ou um livro é uma atividade puramente intelectual que se caracteriza pela necessidade de sucessivas revisões e correções até que a obra adquira sua forma final. [...] Quando um pintor cria um novo quadro, é comum começar com alguns esboços, evoluir para uma representação mais próxima do formato final, fazer acertos, retoques e afins até que a obra esteja concluída" (TELLES, 2004. p. 39).

Você conhecerá agora como se compõe uma equipe de trabalho no XP e os valores do modelo.

Equipe de trabalho

Embora a especialização não seja estimulada nas metodologias ágeis, há necessidade de se estabelecer funções entre os participantes do projeto. Uma típica equipe de trabalho no XP tem a seguinte configuração (TELLES, 2004):

Gerente do projeto: responsável pelos assuntos administrativos, incluindo relacionamento com o cliente. Opera nos bastidores do projeto.

Coach: responsável técnico pelo projeto. Deve ser tecnicamente bem preparado e experiente. Compete a ele assegurar o bom andamento do processo.

Analista de teste: ajuda o cliente a escrever os testes de aceitação e fornece

feedback para a equipe interna de modo que as correções no sistema possam ser feitas.

Redator técnico: ajuda a equipe de desenvolvimento a documentar o sistema, permitindo que os desenvolvedores foquem a construção do programa propriamente dito.

Desenvolvedor: realiza análise, projeto e codificação do sistema. No XP, não há divisão entre estas especialidades.

Valores do XP

O Extreme Programming apoia-se em quatro pilares para atingir seus objetivos:

Feedback: quando o cliente aprende com o sistema que utiliza e reavalia suas necessidades, ele gera feedback para sua equipe de desenvolvimento.

Comunicação: entre equipe e cliente permite que os detalhes sejam tratados com atenção.

Simplicidade: implementar o que é suficiente para atender a necessidade do cliente.

Coragem: para melhorar o que já está funcionando.

Visão geral do *Scrum*

Scrum é um modelo ágil para a gestão de projetos de software que tem na reunião regular dos seus desenvolvedores para criação de funcionalidades específicas sua prática mais destacada. Suas práticas guardam semelhança com as próprias do XP, mas possuem nomes e graus de importância diferentes nos dois contextos. Na sequência você terá contato com os principais elementos do Scrum.

Principais elementos

Product Backlog: trata-se da lista que contém todas as funcionalidades desejadas para o produto. O Scrum defende que tal lista não precisa ser completa logo na primeira vez em que é feita. "Pode-se iniciar com as funcionalidades mais evidentes [...] para depois, à medida que o projeto avançar, tratar novas funcionalidades que forem sendo descobertas" (WAZLAVICK, 2013).

Sprint Backlog: lista de tarefas que a equipe deverá executar naquele Sprint. Tais tarefas são selecionadas do Product Backlog, com base nas prioridades definidas pelo

Product Owner.

Sprint: o *Scrum* divide o processo de efetiva construção do software em ciclos regulares, que variam de duas a quatro semanas. Trata-se do momento em que a equipe se compromete a desenvolver as funcionalidades previamente definidas e colocadas no Sprint Backlog. Se alguma funcionalidade nova for descoberta, ela deverá ser tratada no Sprint seguinte. Cabe ao Product Owner manter o Sprint Backlog atualizado, apontando as tarefas já concluídas e aquelas ainda por serem concluídas.

Membros da equipe

Embora o modelo Scrum defenda que as equipes sejam auto organizadas, ainda assim apresenta três perfis profissionais de relevância:

Scrum Master: trata-se de um facilitador do projeto, um agente com amplo conhecimento do modelo e que preza pela sua manutenção durante todas as etapas do projeto. Deve atuar como moderador ao evitar que a equipe assuma tarefas além da sua capacidade de executá-las.

Product Owner: é a pessoa responsável pelo projeto propriamente dito. Ele tem a missão de indicar os requisitos mais importantes a serem tratados nos Sprints.

Scrum Team: é a equipe de desenvolvimento, composta normalmente por seis a dez pessoas. A exemplo do Extreme Programming, não há divisão entre programador, analista e projetista (WAZLAVICK, 2013).

Exemplificando

A Vodafone é uma das maiores empresas de telecomunicações do mundo. Sua divisão da Turquia, fundada em 2006, era, no ano de 2014, a segunda maior empresa do ramo naquele país. Por meio de processo baseado em três passos básicos, a corporação tem buscado encantar seus clientes por meio do que chamam "transformação ágil". No primeiro passo, uma equipe piloto de desenvolvimento foi estabelecida e, em um número determinado de Sprints, seu progresso foi medido e registrado.

Devido às melhorias observadas na produtividade da equipe piloto – que havia triplicado seu desempenho ao final dos primeiros três meses – ficou definido que o passo dois seria iniciado, com a criação de novas equipes de Scrum. Cerca de cinco meses após o aumento das equipes em quantidade, observou-se que o desempenho havia aumentado em duas vezes. Além disso, foram observadas reduções significativas nas reclamações dos clientes em relação a estas equipes. Este sucesso

levou a organização a criar uma unidade ágil autônoma, chamada "Agile Solutions", que atualmente funciona com seis equipes Scrum. Novas equipes, com novas responsabilidades, logo serão criadas. O próximo passo é crescer e fortalecer a "Agile Solutions" e, em seguida, avançar para o terceiro passo, que é a adoção da metodologia em toda a organização, a fim de fazer crescer a cultura ágil na Vodafone.

Mais informações em http://www.scrumcasestudies.com/wp-content/uploads/2014/10/AgileTransformationInVodafoneTurkey.pdf. Acesso em: 15 nov. 2015.

Alternativamente, você também pode consultar

http://computerworld.com.br/adote-metodologia-agil-para-viabilizar-transformacao-digital. Acesso em: 15 nov. 2015.

Visão geral do *Feature-Driven Development* (FDD)

O FDD ou Feature-Driven Development (Desenvolvimento Dirigido por Funcionalidade) é um método ágil que enfatiza o uso de orientação a objetos e possui apenas duas grandes fases:

a) Concepção e planejamento: o modelo sugere que se conceba e planeje o produto por uma ou duas semanas antes de começar a construir.

b) Construção: desenvolvimento por iterações do produto em ciclos de uma a duas semanas.

A primeira fase inclui três subfases:

DMA (Desenvolver Modelo Abrangente): etapa na qual especialistas estabelecidos em grupos desenvolvem um modelo de negócio amplo, representado por diagramas.

CLF (Construir Lista de Funcionalidades): atividade inicial que abrange todo o projeto, com o objetivo de identificar todas as funcionalidades que satisfaçam os requisitos levantados.

PPF (Planejar por Funcionalidade): nesta etapa é definida a ordem em que as funcionalidades serão implementadas, com base na disponibilidade da equipe de desenvolvimento, na complexidade e nas dependências entre as funcionalidades.

A fase de construção inclui outras duas subfases:

DPF (Detalhar por Funcionalidade): momento em que o design de implementação da funcionalidade é criado, por meio de diagramas de sequência ou comunicação.

CPF (Construir por Funcionalidade): fase em que se produz efetivamente código para as funcionalidades escolhidas (WAZLAVICK, 2013).

Faça você mesmo

A fim de colocá-lo em contato com situações em que a mudança para o modelo ágil foi encarada como solução para pontos frágeis do processo de desenvolvimento, pesquise outros casos de sucesso na adoção do modelo ágil. A internet está repleta deles.

Sem medo de errar

Alguns clientes importantes da X-Soft, incluindo o vendedor de games, têm demonstrado certa insatisfação com o fato de não verem, logo em etapas iniciais dos projetos, seus investimentos revertidos em funcionalidades que possam ser usadas e experimentadas. Argumentam que, caso pudessem ter testado certas partes do programa antes de atingirem seus respectivos formatos finais, não teriam demandado tantas mudanças nas fases finais do projeto e, por consequência, dispendido mais recursos.

Se a X-Soft tem seguido com rigor e competência os procedimentos definidos, por qual motivo tantas alterações são solicitadas pelos clientes? Que indicação de solução você daria ao caso?

Na seção "Diálogo Aberto" foi proposto o desafio de relatar os pontos de atenção e as fragilidades da metodologia assumida, a fim de justificar intenção de mudança futura de modelo.

Idealmente, este relatório deverá pontuar que:

1. o modelo tradicional, por sua própria natureza, não apresenta pontos de feedback regulares com o cliente, de modo a deixá-lo a par do que está sendo desenvolvido e oportunizar a utilização de algumas funcionalidades já em condições de serem executadas.

2. em função desta característica, o modelo atual não oferece ao cliente momentos em que ele possa aprender com o que já está pronto e mudar o rumo – se for o caso – das próximas funcionalidades. Em outras palavras, o cliente tem pouca chance de mudar de ideia.

3. por conta da estrutura linear do modelo tradicional, as etapas do processo devem

ser integralmente concluídas antes de serem sucedidas pela próxima, o que implica na falta de oportunidades para rever o trabalho feito naquela fase e na propagação de uma falha em etapas mais adiantadas do projeto.

4. a implementação das funcionalidades do sistema nunca é feita pelo mesmo profissional que levantou os requisitos e desenhou a solução, já que todas as funções são especializadas. Há pouca comunicação entre os membros da equipe e não raro algumas tarefas são repetidas ou sequer cumpridas.

Atenção!

A mudança de um modelo para outro não é uma atividade simples e imediata. Ao contrário, implica em mudanças culturais, de atitude e de desapego ao que estava funcionando na metodologia antiga. É comum que alguns interesses sejam contrariados durante o processo.

Lembre-se

O cliente deixou de ser figura indesejada durante o processo de desenvolvimento para se tornar peça importante na validação das funcionalidades e na correção pontual de falhas derivadas da má qualidade da comunicação entre ele e equipe.

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu transferindo seus conhecimentos para novas situações que pode encontrar no ambiente de trabalho. Realize as atividades e depois compare-as com a de seus colegas.

Aprimorar ao invés de trocar	
1. Competência Geral	Conhecimento das principais metodologias ágeis de desenvolvimento e habilidade em contrapô-las com a metodologia tradicional.
2. Objetivos de aprendizagem	Apresentar as principais metodologias ágeis de desenvolvimento e a habilidade em contrapô-las com a metodologia tradicional.
3. Conteúdos relacionados	Métodos ágeis - Conceito, histórico e aplicabilidade. Introdução ao XP (Extreme Programming), Scrum, FDD (Feature-Driven Development). Comparação entre metodologia tradicional e ágil.

4. Descrição da SP	Uma empresa desenvolvedora de soluções de software planeja aprimorar o modelo que vem seguindo desde sua criação. É desejo dos seus diretores que haja completa integração do cliente ao processo de desenvolvimento, de modo a aprimorar a comunicação entre os atores do projeto e evitar que a equipe desenvolva as funcionalidades com base apenas no que imaginam terem escutado um dia do cliente. O modelo atual, que é baseado na metodologia cascata, não prevê regularidade na comunicação, o que tem causado, inclusive, retrabalho na criação do programa. Sua missão é sugerir formas de aproximar o cliente do processo, tornando-o corresponsável pelo sucesso do projeto.
5. Resolução da SP	A solução do caso requer as seguintes providências: - Escolha de um membro da equipe para promover e manter o contato com cliente, preferencialmente pessoal e no realizado no próprio ambiente de desenvolvimento. - Promoção de revisões sucessivas em partes pequenas do produto, com a presença do cliente. - Solicitação de que o cliente escreva, de próprio punho, outras eventuais funcionalidades que deseja para o programa em desenvolvimento.

Lembre-se

"Os modelos ágeis de desenvolvimento de software seguem uma filosofia diferente da filosofia dos modelos prescritivos. Em vez de apresentar uma receita de bolo, como fases ou tarefas a serem executadas, eles focam valores humanos e sociais" (WAZLAVICK, 2013).

Faça você mesmo

A fim de prepará-lo para as próximas seções deste material, sua missão é ler sobre "Programação em Par", uma das práticas mais úteis e inovadoras do XP. Comece por http://www.desenvolvimentoagil.com.br/xp/praticas/programacao_par. Acesso em: 15 nov. 2015.

Faça valer a pena

1. Em relação às características dos modelos ágeis, analise as afirmações que seguem:

I) estimulam o desenvolvimento incremental e com intervalos curtos de retornos ao cliente.

II) criados com base nas ideias da produção em série nascidas na Revolução Industrial.

III) apresentam determinismo e a especialização de funções como marcas.

IV) são mais bem adaptadas às mudanças de requisitos que os modelos tradicionais.

É verdadeiro o que se afirma apenas em:

a) II e IV

b) I e IV

c) III e IV

d) II

e) II e III

2. Assinale a alternativa que contém apenas expressões relacionadas aos modelos ágeis de desenvolvimento.

a) trabalhador do conhecimento, especialização, critérios de desenvolvimento

b) feedback regular, trabalhador do conhecimento, determinismo

c) foco na execução, especialização, determinismo

d) cliente presente, trabalhador do conhecimento, desenvolvimento iterativo

e) determinismo, trabalhador manual, desenvolvimento interativo

3. No contexto do modelo XP, assinale a alternativa que contém expressões que completam corretamente as lacunas nas frases abaixo.

I) _____ é o responsável técnico do projeto.

II) _____ é o principal responsável pela documentação técnica do projeto.

III) O principal responsável pelo contato com o cliente é o _____.

a) XP master, desenvolvedor, engenheiro de requisitos

b) desenvolvedor, redator técnico, sponsor

c) coach, redator técnico, gerente do projeto

d) redator técnico, programador, gerente do projeto

e) gerente do projeto, coach, sponsor

Seção 2.2

Métodos ágeis - *Extreme Programming* (XP): valores e práticas

Diálogo aberto

Seja bem-vindo a mais esta aula!

Como se espera de toda organização atenta aos avanços das práticas de desenvolvimento, a X-Soft novamente está prestes a promover mudanças em sua forma de gerir projetos de software. Das experiências passadas deverão permanecer apenas as que contribuíram para tornar a empresa sólida e bem percebida pelo mercado em que atua.

As práticas que remetiam a processos antigos darão lugar a procedimentos ágeis, objetivos, bem focados e em consonância com um novo perfil de cliente e de profissional de TI. A melhor notícia é que você, caro aluno, será o responsável mais direto por essa mudança.

Concluído o levantamento dos pontos ineficientes do modelo atualmente praticado, sua missão agora é promover a gradual, dirigida e irreversível passagem para o modo ágil de se desenvolver produtos de software. Com isso, mais habilidades são desenvolvidas para atender a competência geral que é: conhecer as principais metodologias de desenvolvimento de software, normas de qualidade e processos de teste de software. Ao estudar os métodos ágeis podemos conhecer tecnicamente as principais metodologias ágeis de desenvolvimento e a habilidade em contrapô-las com a metodologia tradicional.

O processo de mudança de modelo deverá ser dividido em duas etapas. Cada uma marcada pela implantação de práticas que estejam relacionadas, respectivamente, com comunicação e feedback.

Para que você se saia bem neste desafio, é preciso que se empenhe na compreensão dos pontos desenvolvidos na seção "Não pode faltar". Sua missão será facilitada se for capaz de entender a real intenção dos criadores do modelo quando propuseram

que o cliente fosse figura presente durante todo o desenvolvimento. Por qual motivo propuseram em seu modelo que a programação fosse feita em pares e que o código fosse propriedade coletiva da equipe? Ousado, não acha?

Em resumo, nesta etapa do desafio você deverá criar documentação que contenha o planejamento da implantação das práticas de: cliente presente, jogo do planejamento, programação em par, código coletivo, stand up meeting e metáforas. Na seção "Sem Medo de Errar" você encontrará as informações que devem constar deste relatório.

Eis o desafio. Bom trabalho!

Não pode faltar

Nesta aula, nosso foco está em apresentar o Extreme Programming. Este modelo fundamenta-se em quatro valores que inspiram suas práticas: feedback, comunicação, simplicidade e coragem (TELES, 2004). Os próximos itens abordarão as práticas que se relacionam mais diretamente com comunicação e ao feedback e que constituirão base para a superação do desafio proposto.

Cliente presente

Esta primeira prática assume o papel principal no processo de quebra de paradigmas proposto pelo XP. Afinal, quando adotamos o modelo tradicional, nunca consideramos como válida – sequer aconselhável - a presença efetiva do cliente durante o desenvolvimento de um programa. Via de regra, a participação dele no projeto se dava apenas no momento da coleta de requisitos e na implantação do sistema.

O modelo ágil, no entanto, sugere que o cliente deve conduzir o desenvolvimento a partir da utilização do sistema e sua proximidade da equipe viabiliza esta condução. Esta prática nos revela que para Teles (2004):

• o distanciamento é natural entre equipe e cliente;

• a proximidade fomenta o feedback, o torna mais constante e evita mudanças bruscas na condução do projeto;

• cliente próximo evita trabalho especulativo;

• quanto mais distante o cliente estiver, mais difícil será demonstrar o valor do serviço;

• para se ter cliente mais presente, deve-se enfrentar desafio cultural, bem como a falta de disponibilidade dele e a distância entre ele e equipe;

• a proximidade provoca aumento da confiança mútua entre cliente e desenvolvedor.

A viabilização da presença do cliente no projeto se dá, entre outros meios, pelo estabelecimento do que se chama sala de guerra, ou war room. Nela deve ser colocada uma mesa onde o cliente poderá desenvolver suas tarefas, próximo da equipe e durante o andamento do projeto. Enquanto trabalha, o cliente poderá ser consultado e, ao escutar informação incorreta, poderá corrigi-la no ato.

Assimile

O XP incentiva que a comunicação seja feita, de preferência, presencialmente. Este trecho sintetiza os motivos. "É importante notar que, em uma comunicação face a face, existe uma riqueza de elementos que facilitam a compreensão da nossa mensagem,além de uma dinâmica que viabiliza questionamentos e respostas (TELES, 2004, p. 47)."

O jogo do planejamento

Que tal se, ao invés de você escutar do cliente o que ele tem a dizer sobre as funcionalidades e fazer suas anotações, a síntese das funções fosse feita por ele, de próprio punho? Pois é assim que o XP orienta que se inicie o planejamento do sistema.

Ao cliente é dado um cartão, com aproximadamente metade de uma folha tamanho A4, no qual ele deverá escrever uma (e apenas uma por cartão) funcionalidade que deseja para o programa, de forma simples e objetiva.

Você pode achar esta prática um tanto diferente, mas vale a reflexão: será que o cliente não passará a dar mais valor àquilo que ele próprio está escrevendo? Não haverá, portanto, maior responsabilidade dele em relação ao que está pedindo? Sem contar que, ao registrar ele próprio um requisito, a possibilidade de mal-entendido em relação a ele cai consideravelmente.

Cada ficha dessa leva o nome de estória. Elas são a base para o planejamento dos desenvolvedores, que podem dividir a estória em tarefas, em nome de um planejamento mais preciso e caso as estórias sejam extensas demais para apenas um dia de trabalho.

Exemplificando

Podem ser considerados bons exemplos de estórias:

• Apresente ao cliente as dez tarifas mais baratas para uma determinada rota.

• Para cada conta, computar o saldo fazendo a adição de todos os depósitos e a subtração de todas as deduções.

• A tela de login deve permitir que o usuário pule o login. Neste caso, o usuário entrará no sistema como convidado.

• O usuário deve poder alterar seu perfil. Dois campos de senha para confirmação.

É comum que, ao considerarmos a duração de uma tarefa, a estimativa seja feita em horas. O XP, no entanto, adota como unidade o dia ideal, que representa um dia no qual o desenvolvedor trabalha apenas nas implementações das funcionalidades, sem telefone, reuniões ou outras interrupções ou imprevistos. A pergunta que se faz é: "Se eu tiver o dia todo para implementar determinada estória, quantos dias levarei para finalizá-la?" Cada dia ideal representa, para a equipe, um ponto, que é a unidade de medida usada para estimar e acompanhar todas as estórias (TELES, 2004).

Para fins de controle, os pontos estimados para aquela estória são registrados no canto superior esquerdo do cartão e os pontos já consumidos são registrados no canto superior direito do cartão.

É normal que neste ponto você esteja questionando a perfeita viabilidade de se estimar o desenvolvimento desta forma. Contudo, alguns procedimentos podem ser adotados pela equipe para tornar este método bem interessante:

• resgatar cartões que já tenham sido implementados e que sejam semelhantes ao que está sendo estimado pode oferecer boa noção sobre investimento de tempo na estória.

• a equipe deve se aproveitar do registro dos pontos efetivamente consumidos para estimar. A experiência com funcionalidades semelhantes também vale.

• a prática recomenda que estimativas sejam feitas em conjunto, como forma de unir experiências e sentimentos e mitigar a possibilidade de erros. E sim, o cliente deve estar presente nas estimativas, tornando menor a chance de premissas incorretas.

Como uma das bases do modelo ágil é a oferta regular de artefato executável ao cliente, a equipe deve planejar muito bem as entregas das funcionalidades, ou os ***releases***.

Exemplificando

Um bom exemplo de como os releases podem ser distribuídos é o que segue. São quatro releases com oito semanas de intervalo entre cada um, para um site de vendas *on-line*:

R1: consulta dos produtos em estoque

R2: processamento das compras on-line

R3: acompanhamento dos pedidos

R4: campanha de marketing de relacionamento

Os releases devem ser oferecidos em intervalos curtos, tipicamente de 2 meses (TELES, 2004).

Confirmando a participação efetiva do cliente no projeto, o XP prevê que ele (cliente) deve estabelecer a ordem dos releases com base em suas necessidades. Devem ser levadas também em conta as dependências técnicas, naturalmente. Durante o desenvolvimento do primeiro release, o cliente usará o sistema várias vezes, o que o ajudará a escolher as estórias das próximas entregas. Durante um release, um cliente poderá alterar as estórias se considerar necessário, fazendo uso do seu aprendizado no sistema, já que o XP não visa blindar a equipe de desenvolvimento das mudanças. Em processos tradicionais, o escopo é fechado no início do projeto. No XP, o escopo deve se alterar ao longo do projeto.

Programação em par

Desenvolvedores não trabalham sozinhos num projeto XP. Você também pode achar pouco convencional, mas na programação em par, dois programadores trabalham em um mesmo problema, ao mesmo tempo. Aliás, quem foi que disse que o XP é convencional? Adiante.

Em determinado momento, o condutor assume o teclado e o navegador acompanha o trabalho, fazendo revisões e sugestões. Em outro, há revezamento de papéis. As correções são feitas no momento da programação, evitando que pequenos erros se tornem grandes problemas no futuro.

A condução da programação deve ser realizada, em tempos alternados, pelos dois programadores. Eles devem se alternar, escrevendo código a cada 15 ou 20 minutos. O programador que não estiver escrevendo verifica cuidadosamente o código, enquanto ele está sendo criado pelo seu companheiro (SCHACH, 2008).

A prática influencia na boa modelagem da solução, que passa a ser fruto do entendimento e da conversa entre os pares. Se um parceiro concebe algo muito complexo, o outro pode propor solução mais simples (TELES, 2004).

O par exerce pressão positiva no desenvolvimento, evitando distrações de e-mail,

bate-papos, cansaço, desânimo momentâneo. O compromisso deixa de ser individual e passa a ser também em relação ao par.

Além de reduzir as chances de defeitos na programação, esta prática tende a tornar o conhecimento geral da equipe mais homogêneo e contribuir para o aprendizado contínuo.

Você pode estar quase convencido de que a programação em par é a solução para todos os problemas da criação de um código. Infelizmente esta não é a realidade.

Para boa parte dos gerentes, desenvolvedores são vistos como máquinas e o projeto como uma fábrica e, de acordo com a lógica industrial, deve-se obter mais produção de uma mesma "máquina". Acontece que já estamos alertados: o que se aplica para a produção em massa não se aplica ao desenvolvimento de software, não é mesmo?

Reflita

A programação em par é, de fato, imune a contratempos? Reflita.

Na prática, foram observados alguns inconvenientes na programação em duplas. Por exemplo, ela requer grandes blocos ininterruptos de tempo, e os profissionais poderão ter dificuldades em alocar períodos de tempo de 3 a 4 horas ininterruptas. Além disso, nem sempre funciona bem com indivíduos tímidos ou autoritários, ou com dois programadores inexperientes (SCHACH, 2008, p. 57).

Código coletivo

Já que o XP sugere a programação em par e, neste contexto, o rodízio de programadores, é normal que toda a equipe seja responsável pelo código que está sendo produzido. Sendo assim, não será necessária autorização para que seja alterado arquivo, por quem quer que seja, desde que mantido o padrão estabelecido. Você entende melhor agora o porquê de a coragem ser um dos valores do XP? Estimar estórias na presença do cliente, deixando-o priorizar as funcionalidades e manter o sistema simples são mais indicativos de que o uso do XP requer coragem.

A prática do código coletivo pode ter o efeito benéfico de evitar "ilhas de conhecimento", ou seja, profissionais que detenham conhecimento quase exclusivo sobre determinado projeto. O "sabe tudo" pode se ausentar por doença, férias, viagem ou mesmo deixar a empresa repentinamente e os demais colaboradores poderão não saber alterar partes do projeto cujo conhecimento é quase exclusivo da "ilha".

Outro efeito importante do código coletivo é que, quando todos o acessam, o código tende a ser mais bem revisado.

Stand up meeting

Uma das práticas mais simples e de implementação mais imediata é a stand up meeting (reunião feita em pé). Um dia de trabalho no XP começa com uma reunião rápida, com não mais do que 10 minutos de duração e que serve para que todos os membros da equipe comentem o trabalho do dia anterior e planeje o dia atual. Ao final da stand up, cada membro saberá o que deve fazer ao longo do dia. Esta prática pode gerar bons resultados, já que cada membro da equipe terá a visão geral do andamento do trabalho (TELES, 2004). Em tempo: aconselha-se que a reunião seja feita todos os dias e com todos os participantes em pé, de modo a evitar prolongamento excessivo do tempo da conversa.

Metáforas

Visando promover a boa comunicação entre a equipe, sugere-se a adoção do uso de metáforas em pontos chave do projeto, como na definição de um nome que seja comum à equipe e simbolize algo de fácil assimilação, como, por exemplo: "Vamos chamar nosso projeto de 'cartão de ponto', para um sistema que gerencie as batidas de ponto de funcionários, gerando o provisionamento financeiro e mensal para módulo de folha de pagamento" (Disponível em http://www.devmedia.com.br/extreme-programming-conceitos-e-praticas/1498#ixzz3sViVUxVT. Acesso em: 01 dez. 2015).

A metáfora deve ser utilizada, inclusive, para facilitar o entendimento do modelo XP (ou de uma funcionalidade do sistema) por parte do cliente. Imagine uma equipe de voleibol. Há jogadores mais especializados na defesa, outros no ataque e um que deverá fazer o último passe para o atacante buscar o ponto. No entanto, por conta do rodízio obrigatório de posições, em algum momento um atacante se verá na necessidade de defender uma bola, um levantador de atacar e assim por diante. Alguma similaridade com o XP?

Pesquise mais

Podemos encontrar boa definição formal da figura de linguagem metáfora em http://educacao.uol.com.br/disciplinas/portugues/metafora-figura-de-palavra-variacoes-e-exemplos.htm (Acesso em: 01 dez. 2015).

Faça você mesmo

Uma das bases do pensamento do XP é a de que mais vale um programa rodando do que sua perfeita documentação. Contudo, isso não significa que a documentação deva ser negligenciada ou ignorada. Sua tarefa é fazer levantamento de ao menos 4 documentos que devem ser gerados durante um projeto conduzido pela metodologia XP.

Sem medo de errar

É do senso comum entre profissionais de TI que a metodologia XP, para atingir sua plenitude e ser efetiva, deve ser implantada em sua totalidade. Sua adoção parcial, embora possa trazer algum benefício momentâneo, não deverá ser entendida como regra. Há, contudo, a sensação de que a implantação das práticas deva ser gradual. A troca repentina e integral de uma metodologia em uma organização que sempre a praticou poderá ser, no mínimo, traumática. É justamente a mudança gradual e controlada que nosso desafio propõe.

Apenas para resgatarmos o que foi descrito na seção "Diálogo Aberto", na primeira etapa de mudança para o modelo ágil deverão ser implantadas as práticas mais proximamente relacionadas à comunicação e ao feedback, quais sejam cliente presente, jogo do planejamento, programação em par, código coletivo, stand up meeting e metáforas. Uma abordagem possível para tal inflexão é a que segue:

1. Já que o principal objetivo a ser atingido nesta primeira etapa é a melhoria em todos os meios de comunicação, nada mais imediato do que chamar o cliente ao projeto. Haverá, por certo, resistência em participar tão ativamente do projeto num primeiro momento, mas nada como usar o apelo das entregas regulares e feitas em períodos pequenos para convencê-lo de que, quanto mais próximo ele estiver da equipe, mais corretamente implementadas estarão as funcionalidades.

2. Para a efetivação da nova maneira de coletar e tratar os requisitos – aqui chamada de Jogo do Planejamento – a inclusão do cliente no processo já deverá ter sido consolidada. De novo, é possível que haja resistência em escrever – ele próprio – o que deseja para o projeto.

3. Por demandar providências meramente organizacionais e internas, a programação em par é uma das práticas de mais fácil adoção no contexto. Você deve acompanhar, no entanto, o desempenho dos programadores ao atuarem em duplas, já que esta nova configuração de trabalho pode vir acompanhada de pequenos conflitos entre os colegas, perda de foco, autoritarismo de um dos elementos em relação ao outro, entre outras ocorrências.

4. O código coletivo, a stand up meeting e o uso de metáforas também inspiram relativa facilidade em suas implementações. Vale promover a conscientização da equipe em relação a mudanças inoportunas no código e a eventual fuga do padrão de codificação estabelecido. Embora todos possam ter acesso irrestrito aos arquivos, é necessário o registo das alterações feitas. Em relação às reuniões diárias, na condição de gestor você deve acompanhar os primeiros encontros e atuar como moderador da equipe, cuidando para que a duração da reunião não ultrapasse o estabelecido e que os assuntos devidos sejam tratados.

Atenção!

O que costuma ser mais trabalhoso na venda do XP é mostrar suas vantagens para a área de TI da empresa cliente. A maioria dos profissionais da área teve formação acadêmica voltada para o desenvolvimento tradicional, o que os acostumou a outro conjunto de premissas e pouco propensos a abandoná-las facilmente (TELES, 2004).

Lembre-se

A implantação de prática do XP, em conjunto ou de forma isolada, deve ser norteada pelo bom senso. O gestor não deve colocar rotinas novas para a equipe sem antes motivar seus membros para sua completa adoção.

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu transferindo seus conhecimentos para novas situações que pode encontrar no ambiente de trabalho. Realize as atividades e depois compare-as com a de seus colegas.

Aprimorando a comunicação da equipe	
1. Competência Geral	Conhecimento das práticas ágeis ligadas à comunicação e ao feedback.
2. Objetivos de aprendizagem	Incentivar a investigação e aplicação das práticas do XP relacionadas à comunicação e ao feedback.
3. Conteúdos relacionados	Métodos ágeis - Extreme Programming (XP): valores e práticas
4. Descrição da SP	Uma empresa desenvolvedora de soluções de software, em situação semelhante à que nos serviu como exemplo na aula passada, viu-se na necessidade e aprimorar a comunicação entre os membros da equipe. Durante o processo de desenvolvimento de um produto, os profissionais envolvidos não têm feito reuniões regulares para tratar do andamento do projeto. Esta falta de comunicação presencial, inclusive, acarretou sobreposição de algumas tarefas, que acabaram sendo feitas em duplicidade. Além disso, há clara concentração do conhecimento em um elemento da equipe, justamente o mais experiente e antigo de casa. Sua missão é sugerir formas de estabelecer boa comunicação entre membros da equipe, de modo que os problemas mencionados tenham suas ocorrências minimizadas.
5. Resolução da SP	A solução do caso requer as seguintes providências principais:

	- Agendamento de reuniões diárias, logo na primeira hora da manhã, de curta duração e focadas no planejamento do dia de trabalho. Além disso, todos os participantes deverão informar em que estágio do desenvolvimento se encontram. - Estabelecimento da regra do código coletivo, ou seja, todos os programadores terão acesso às funcionalidades desenvolvidas e em desenvolvimento, a fim de serem capazes de retomar o trabalho em caso de falta do programador mais experiente.

Lembre-se

A efetivação das práticas do XP, quaisquer que sejam, demandam paciência, comprometimento, conscientização da equipe, parceria com o cliente e controle. No entanto, os benefícios derivados delas serão incentivadores permanentes do aprofundamento na metodologia.

Faça você mesmo

A mudança cultural provocada pela implantação do XP e a dificuldade em "vendê-lo" como uma metodologia eficiente ainda são obstáculos no caminho dos modelos ágeis. Para conhecer opinião importante sobre o tema, faça a leitura do capítulo 20 do livro "Extreme Programming: aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade", de Vinícius Manhães Teles. Depois de tomar conhecimento do assunto, resuma-o em ao menos uma página.

Faça valer a pena

1. Em relação à prática do Cliente Presente, analise as afirmações que seguem:

I) visa capacitar o cliente para a codificação do produto.

II) deseja alcançar a inclusão do cliente no processo de desenvolvimento

III) pode auxiliar na melhor compreensão das funcionalidades desejadas pelo cliente.

IV) pode evitar trabalho especulativo por parte da equipe.

É verdadeiro o que se afirma em:

a) I e II apenas

b) II, III e IV apenas

c) I e III apenas

d) III e IV apenas

e) II e IV apenas

2. Em relação à prática do Jogo do Planejamento, analise as afirmações que seguem:

I) estimar uma funcionalidade por comparação significa comparar desempenhos individuais dos desenvolvedores para fins de atribuição de atividades.

II) ponto é uma unidade de medida única usada para estimar todas as estórias.

III) a exemplo do Scrum, durante o desenvolvimento de um release o cliente não poderá alterar a estória.

IV) o período entre um release e outro não importa. Pode ser de muitos meses, desde que ele contenha todas as funcionalidades que o cliente solicitou.

É verdadeiro o que se afirma apenas em:

a) I e IV

b) II e IV

c) I

d) II

e) II e III

3. Em relação a Programação em Par, assinale a alternativa correta.

a) presta-se à comparação de desempenho entre dois programadores.

b) pode aumentar a ocorrência de defeitos no programa, já que nenhum dos dois programadores poderá estar preparado para enxergá-los.

c) com a aplicação da prática, as correções são feitas no momento da programação, evitando a continuidade do problema em circunstâncias futuras do projeto.

d) aconselha-se que um dos programadores seja sempre destacado como condutor e o outro como navegador, sem revezamento.

e) a prática estimula a criação de soluções complexas para a funcionalidade que está sendo desenvolvida, já que duas cabeças raciocinam melhor do que uma.

Seção 2.3

Práticas do *Extreme Programming* e suas contra-indicações

Diálogo aberto

Seja bem-vindo de volta!

A segunda etapa do processo de implantação da metodologia Extreme Programming na X-Soft foi concluída. Ela previa a implantação de práticas do XP relacionadas à comunicação e visava trazer o cliente para o centro do processo e aprimorar o relacionamento entre membros da equipe.

Sua experiência em identificar necessidades de mudanças no processo de desenvolvimento e efetivá-las foi decisiva para este início de transição entre o modelo tradicional e o modelo ágil. No entanto, o trabalho ainda não está terminado! Sua missão agora é dar continuidade ao processo de mudança, planejando e realizando a introdução principalmente das práticas do XP voltadas à Integração Contínua e ao Desenvolvimento Guiado pelos Testes. Como trabalho adicional – mas não menos importante – você deverá também iniciar as práticas de refatoração e padrão de codificação.

Além disso, você deverá explicitar as contraindicações da metodologia XP, ao destacar suas vulnerabilidades e pontos de deficiência.

Assim, você poderá aprimorar seus conhecimentos sobre o XP e aumentar sua habilidade em compará-la com a metodologia tradicional. Novamente, para que você se saia bem neste desafio, é preciso que se empenhe na compreensão da teoria desenvolvida na seção "Não pode faltar", onde as práticas serão abordadas.

Em resumo, nesta etapa do desafio você deve planejar e executar a implantação das práticas conhecidas como Integração Contínua, Desenvolvimento Guiado pelos Testes, Refatoração e Padrão de Codificação, por meio de relatório, com isso atendemos ao objetivo de aprendizagem desta aula que é: fornecer subsídios para que os alunos possam planejar a introdução de práticas do XP relacionadas à integração contínua e testes.

Desde já, bons estudos!

Não pode faltar

O Extreme Programming adota uma série de práticas que o tornam um meio moderno, ágil e inovador de se vencer o processo de criação de um programa. Você estudou as práticas de Cliente Presente, Jogo do Planejamento, Programação em Par, Código Coletivo, Stand up Meeting e Metáforas e descobriu que em todas elas há pelo menos uma quebra de paradigma em relação ao modelo tradicional. Afinal, neste modelo não se coloca o cliente no centro do processo de desenvolvimento, os requisitos são levantados e registrados pela própria equipe, a construção de uma parte do programa é feita por uma única pessoa de cada vez e o código não é de propriedade coletiva. Além disso, o modelo tradicional não prevê reuniões diárias (embora não impeça que sejam feitas) e não faz uso de metáforas para aprimorar a comunicação entre os membros da equipe.

Nesta seção serão tratadas outras práticas do XP, tão importantes quanto as primeiras. Adiante!

Desenvolvimento Guiado pelos Testes

Antes de entrarmos especificamente no assunto, vale conceituar dois termos importantes no contexto (WAZLAWICK, 2013):

- Teste de unidade: este teste consiste em verificar se um componente individual do software (uma unidade) foi implementado corretamente. Este componente pode ser uma classe, um método ou um pacote de funções e geralmente está isolado do sistema do qual faz parte. O desenvolvimento guiado pelos testes recomenda, inclusive, que antes de o programador desenvolver uma unidade de software, ele deve gerar o seu driver de teste, que é um programa que obtém ou gera um conjunto de dados que serão usados para testar o componente que ainda será desenvolvido.

- Teste de aceitação: este teste é realizado pelo cliente, que utilizará a interface final do sistema. A aplicação do teste de aceitação visa validar o software em relação aos requisitos e não mais em relação a verificação de defeitos.

Explicados os conceitos, vamos ao nosso assunto principal.

Originalmente identificada como Test-Driven Development (TDD), a adoção desta prática tem como objetivo a identificação e correção de falhas durante o desenvolvimento, não apenas ao final dele.

O processo de teste, quando aplicado da maneira tradicional, tende a ser difícil, especialmente quando você está testando seu próprio código. Escrever um teste para que ele falhe pode ser desencorajador. É muito fácil querer chegar logo ao final de um

teste, esperando que tudo funcione corretamente (O'REILLY, 2009).

O desenvolvimento guiado pelos testes é diferente. Para que a condução desta prática seja feita corretamente e produza bons resultados, o XP propõe algumas regras (WAZLAWICK, 2013):

a) todo código deve passar pelos testes de unidade antes de ser entregue: esta providência ajuda a assegurar que sua funcionalidade seja implementada corretamente. Os testes de unidade também favorecem a refatoração - que será abordada adiante nesta seção – por que protegem o código de mudanças indesejadas de funcionalidade. A introdução de uma nova funcionalidade deverá levar à adição de outros testes ao teste de unidade específico.

b) quando um erro de funcionalidade é encontrado, testes são criados: um erro de funcionalidade identificado exige que testes de aceitação sejam criados. Desta forma, o cliente explica com clareza aos desenvolvedores o que eles esperam que seja modificado.

c) testes de aceitação são executados com frequência e os resultados são publicados. Os testes de aceitação são criados a partir das estórias do cliente. Durante uma iteração, as estórias selecionadas para implementação serão traduzidas em testes de aceitação.

Integração contínua

Conforme tratamos na unidade 1, a integração de um software refere-se ao momento em que ocorre a união dos módulos criados separadamente, de modo a se obter um sistema único. Não há nada de errado com essa abordagem, mas aqui neste contexto a prática será tratada de forma ligeiramente diferente. Prossigamos.

Na forma em que é realizada no âmbito do modelo tradicional de desenvolvimento, é comum que os desenvolvedores convencionem interfaces, de modo que possam fazer a integração em momento futuro (TELES, 2004). No entanto, erros na integração serão frequentes caso os padrões definidos para as interfaces (assinaturas de métodos, por exemplo) não sejam respeitados ou assimilados pela equipe. Quanto maior o intervalo entre uma integração e outra, mais a equipe terá dificuldades em fazer o sistema rodar e os testes funcionarem (TELES, 2004).

O XP adota a prática da integração contínua, que consiste em integrar o trabalho diversas vezes ao dia, assegurando que a base de código permaneça consistente ao final de cada integração (TELES, 2006. Disponível em http://www.desenvolvimentoagil.com.br/xp/praticas/integracao. Acesso em: 18 dez. 2015).

A primeira providência que você deve tomar para promover a integração contínua

é criar um repositório de programas. O sistema em construção deve estar alocado em um repositório comum, disponível para todos os pares de programadores (o código é coletivo no XP, lembra-se?) e cada alteração aplicada neste sistema deve ser controlada por um sistema de controle de versões. O Microsoft Visual SourceSafe e o CVS (do inglês Concurrent Version System) são ótimos produtos de mercado, assim como o Subversion. Este último produto é gratuito e de fonte aberta, capaz de controlar arquivos e diretórios de programas, bem como as alterações feitas neles ao longo do tempo. Ele permite que você recupere versões antigas de seus dados ou examine o histórico de como os programas foram alterados. O Subversion pode operar entre diversas redes, o que permite que seja usado por pessoas em diferentes computadores. Desta forma, a possibilidade de várias pessoas modificarem e gerenciarem o mesmo conjunto de dados de seus próprios locais fomenta a colaboração entre elas.

Para que sua operação seja possível, alguns comandos devem ser usados. Eis alguns exemplos (Disponível em: http://svnbook.red-bean.com/en/1.7/svn-book.pdf. Acesso em: 2 jan. 2016):

- checkout: é um dos comandos principais de qualquer sistema de controle de versão e serve para baixar na máquina local o repositório desejado.

- update: utilizado quando se deseja atualizar o repositório após mudanças serem efetuadas.

- diff: exibe os detalhes de uma alteração em particular.

Assimile

Entenda repositórios de código como uma máquina que centraliza todos os arquivos referentes ao projeto em andamento e que contém sistema de controle de versão. Diversas versões do sistema ficarão gravadas neste repositório e, no caso de a versão atual apresentar problemas, a anterior será recuperada. Todos os desenvolvedores deverão ter acesso a este repositório.

Durante o desenvolvimento do produto, será comum que dois pares de programadores acessem simultaneamente um arquivo para alterá-lo. Através do recurso do checkout, um par de programadores pode tornar o arquivo disponível para si em modo de leitura e alteração e impossibilitar a gravação para todo o resto da equipe. Quando sua tarefa chega ao fim, o sistema deverá integrar o código recém-construído ao repositório.

Neste contexto, uma outra ferramenta digna de sua atenção é o Eclipse. Trata-se de um ambiente de desenvolvimento gratuito e completo para Java, embora suporte também outras linguagens.

Quer conhece-lo melhor? Acesse https://eclipse.org/ ou http://www.devmedia.com.br/conhecendo-o-eclipse-uma-apresentacao-detalhada-da-ide/25589. Acesso em: 14 jan. 2016.

Exemplificando

Um bom exemplo de integração contínua pode ser dado por meio da situação em que dois programadores estejam criando uma calculadora em Java usando o ambiente de desenvolvimento Eclipse. Cada par deverá contar com uma instância diferente do Eclipse em execução e duas áreas de trabalhos diferentes devem ser criadas. Com projetos e diretórios também separados, ambos começam a dar contribuições no projeto. O primeiro par executa aprimoramentos na classe que realiza a multiplicação e o segundo altera o código da classe da divisão. Implementadas e testadas as alterações, basta que atualizem seus códigos no repositório. Ao fazerem pequenas e constantes integrações durante o dia, o esforço de criação do sistema tenderá a ser menor e os inevitáveis erros tenderão a ser menos graves e tratados com mais facilidade.

Padrões de Codificação

O fomento da comunicação num ambiente de XP também se dá através do código dos programas. A equipe deve se comunicar de forma clara através do código e, por isso, o XP sugere a adoção de padrões de codificação. Diferentes programadores têm diferentes estilos de programação, que podem dificultar a compreensão do código por parte dos membros da equipe. A adoção de um mesmo estilo de programação pode evitar esta dificuldade (TELES, 2004).

Características do padrão

Deve-se adotar um padrão fácil e de simples compreensão. Confira algumas dicas que você pode usar na construção de um padrão (TELES, 2004):

- Endentação: mesma largura na tabulação e mesmo posicionamento de chaves (abertura e fechamento de bloco).

- Letras maiúsculas e minúsculas: deve ser mantida a consistência com a caixa da letra (alta ou baixa). Java, por exemplo, tem uma padronização bem definida de maiúsculas e minúsculas.

- Comentários: evite comentários no código. Código simples transmite mais a mensagem do que quaisquer comentários.

- Nomes de identificadores: use nomes que comuniquem a ideia. Se preciso, use nomes longos. Padrões para nomes de tabelas também devem ser seguidos. É essencial que a equipe consiga descrever coisas similares com nomes similares

Mantendo o padrão

Quando alguém identifica um código fora do padrão, deve-se alertar a equipe que o padrão não foi respeitado e orientar sobre a forma de utilizar o padrão

Normalmente, exceções ao padrão só ocorrem no início do projeto. Enquanto a equipe estiver no processo de adaptação, você pode achar interessante publicar as regras do padrão em local visível da sala de desenvolvimento, a fim de que todos possam consultá-las.

Não são descartados ajustes no padrão para que ele se aproxime daquilo que os programadores estão mais habituados.

Dificuldades na adoção de um padrão

Pode haver desconforto quando um programador precisa mudar seu padrão pessoal.

Alterações podem causar resistências. A equipe deve negociar e entrar em acordo sobre qual padrão adotar.

Reflita

Será que o formato de um padrão de codificação é, de fato, menos relevante do que sua própria adoção? Em outras palavras, será que mais vale adotar um formato não tão claro do que nenhum formato?

Refatoração (Refactoring)

Imagine que você tenha empreendido bastante esforço e conhecimento para criar um programa. Depois de um tempo considerável, ele está funcionando muito bem e executando com perfeição suas funções. No entanto, seu gerente ainda não está satisfeito: agora ele deseja que você altere seu código, apenas para melhorar a escrita e legibilidade, sem que isso implique em mudança alguma na lógica implementada e ou no funcionamento do programa.

Arriscado, não acha? Pois esta é justamente a ideia da refatoração ou, como usualmente dito em nosso meio, do refactoring.

Refatoração é uma palavra sofisticada usada para identificar a melhoria na escrita de código já criado, sem que isso acarrete alterações em seu comportamento (O'REILLY, 2009).

A necessidade de se refatorar um código não é criação do XP, mas foi alçada a uma de suas práticas pela sua importância no contexto do pensamento ágil.

No âmbito de qualquer modelo de desenvolvimento que se use, um código mal formulado poderá gerar dificuldades para quem um dia precisar modificá-lo ou simplesmente compreendê-lo. O código deve estar preparado para receber manutenções corretivas, evolutivas, preventivas ou adaptativas (TELES, 2004).

Pesquise mais

Quer conhecer alguns mitos sobre a refatoração? Acesse http://www.infoq.com/br/articles/RefactoringMyths (Acesso em: 10 dez. 2015) e leia este bom artigo que trata do tema.

A refatoração está ligada ao código coletivo e a implantação de padrões de codificação. Se a ideia é que todos tenham acesso ao código, que ele seja padronizado e ofereça facilidade em sua leitura.

A boa prática indica que o código deve ser refatorado regularmente. Após a execução de um teste, refatore. Após concluir uma tarefa ou uma função, refatore o código novo. Elimine repetições de código, quebre métodos e funções em partes menores, torne mais claros nomes de variáveis e métodos e, finalmente, deixe o código mais fácil de entender e de ser modificado (O'REILLY, 2009).

Lembre-se

A refatoração, se mal aplicada, também pode fazer o código parar de funcionar. Não mexer no código significa não acrescentar mais um risco, contudo significa também manter no sistema risco potencialmente maior de ser mal compreendido e de ter a manutenção dificultada.

Para minimizar riscos, deve-se conhecer bem as técnicas de refatoração e executar testes de unidade após sua execução. Após a refatoração, o desenvolvedor pode ficar à vontade para adicionar ou alterar funcionalidades (TELLES, 2004).

Faça você mesmo

Para praticar o que você sabe sobre refatoração e padrões de codificação, tome um programa do qual você tenha o código-fonte, refatore-o e

coloque o código no padrão recomendado para aquela linguagem. Lembre-se: nenhuma funcionalidade deverá ser alterada, tampouco seu funcionamento.

Sem medo de errar

As práticas do XP mais diretamente relacionadas à comunicação e feedback foram implantadas recentemente na X-Soft e estão caindo no gosto da equipe. Com uma ou outra necessidade de adaptação, cada uma delas foi aos poucos introduzida no cotidiano da equipe e os primeiros sinais positivos da mudança começam a aparecer.

As reuniões diárias têm melhorado a comunicação entre a equipe e a tornado mais coesa. A adoção da programação em par e do código coletivo dão sinais de terem sido ótimas providências para melhorar a qualidade do código e das funcionalidades por ele criadas. Agora a maioria dos erros cometidos na programação são descobertos no ato e a tendência à sua propagação tem diminuído consideravelmente.

E, claro, não podemos nos esquecer do esforço que a X-Soft tem empreendido para trazer o cliente para perto do processo de desenvolvimento. É com boa constância que a equipe tem chamado seu cliente para usar partes prontas do código e opinar sobre necessidades de mudanças pontuais no rumo da criação do produto. Essa participação tem sanado, quase que em tempo real, as dúvidas da equipe e evitado a introdução de funcionalidades equivocadas. Enfim, seu trabalho até aqui tem se mostrado competente e tem gerado bons resultados.

No entanto, você ainda precisa implantar algumas outras práticas do XP para que sua utilização esteja muito perto da plenitude. Na seção "Diálogo Aberto" foi proposto a você dar continuidade ao processo de mudança, planejando e realizando a introdução da Integração Contínua, do Desenvolvimento Guiado pelos Testes, da Refatoração e de um Padrão de Codificação.

Novamente trazemos neste ponto uma abordagem possível para a criação deste relatório:

1. caso ainda não exista, um repositório único dos arquivos de programa do projeto deve ser criado. 2. logo em seguida, deve ser providenciada a implantação de software de controle de versão. Com isso, a integração contínua estará apta a ser implantada.

3. para sua efetivação, haverá necessidade de orientar a equipe para que a integração seja feita várias vezes ao dia. Para que a prática do Desenvolvimento Guiado pelos Testes possa ser efetivada, você deverá orientar os desenvolvedores que desenvolvam e apliquem testes automatizados de unidade. A geração dos testes de unidade deve preceder a criação de cada classe

4. para a implantação da refatoração, você deverá estipular a periodicidade de sua aplicação e as ocasiões em que ela será aplicada.

5. por fim, um padrão de codificação deve ser escolhido e divulgado à equipe. A escolha deverá levar em conta eventual conjunto registrado de boas práticas indicadas pela linguagem de programação utilizada.

Atenção!

Erros na codificação descobertos prematuramente pouparão o tempo da equipe no momento dos testes e da integração.

Lembre-se

O planejamento é indispensável num processo de mudança. A preparação prévia para a mudança inclui a divulgação das práticas à equipe e o estabelecimento de métodos, prazos e objetivos. Boa leitura sobre o tema encontra-se em http://www.significados.com.br/planejamento/ (Acesso em: 18 dez. 2015).

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu transferindo seus conhecimentos para novas situações que pode encontrar no ambiente de trabalho. Realize as atividades e depois compare-as com a de seus colegas.

Aprimorando a aplicação de testes no software	
1. Competência Geral	Conhecimento da prática do XP ligada ao teste de software.
2. Objetivos de aprendizagem	Fornecer subsídios para que os alunos possam planejar a introdução de práticas do XP relacionadas à integração contínua e testes.
3. Conteúdos relacionados	Elementos da metodologia tradicional relacionados a teste de software, práticas dos modelos ágeis, Desenvolvimento Guiado pelos Testes.
4. Descrição da SP	Uma empresa desenvolvedora de soluções de software, em situação semelhante à que nos serviu como exemplo na aula passada, viu-se na necessidade de aprimorar seu processo de teste. Atualmente, os desenvolvedores escrevem as unidades e, só depois de prontas, iniciam os testes. Esta prática, no entanto, tem se mostrado ineficiente: quando o projeto atrasa, o tempo destinado aos testes acaba sendo usado no próprio desenvolvimento do sistema; o defeito no código, quando descoberto tempos depois, tende a ser corrigido com maior

	esforço, já que os desenvolvedores com frequência devem voltar ao contexto daquele código para efetivarem o ajuste e, por fim, a prática atual não toma medida preventiva alguma em relação aos defeitos no código. Sua missão é sugerir mudanças na forma de abordagem do teste, de forma que seu processo seja eficiente na prevenção de ocorrência de falhas no código e que o esforço para sua detecção seja minimizado.
5. Resolução da SP	A solução do caso requer a criação e aplicação de testes antes e durante a criação da unidade, o que deverá incluí-los no processo de desenvolvimento.

Lembre-se

"Quando o desenvolvedor pensa no teste antes de pensar na implementação, ele é forçado a compreender melhor o problema. Ele se vê diante da necessidade de aprofundar o entendimento, entrando nos detalhes e levantando hipóteses (TELES, 2004)".

Faça você mesmo

A prática do Desenvolvimento Guiado pelos Testes inclui a utilização de ferramentas capazes de automatizar o processo. Você está convidado a conhecer o JUnit, por meio de pesquisa própria. Depois de tomar contato com a ferramenta, nossa sugestão é que acesse http://www.desenvolvimentoagil.com.br/xp/praticas/tdd/ (Acesso em: 18 dez. 2015) e verifique exemplo de teste usando o JUnit.

Faça valer a pena

1. Em relação à prática do Desenvolvimento Guiado pelos Testes, analise as afirmações que seguem:

I) prática do XP que deve ser aplicada apenas ao final do processo de desenvolvimento.

II) o teste de unidade visa verificar o produto final, como um item único.

III) o teste de aceitação é realizado pelo cliente, utilizando a interface final do sistema

IV) a adoção do desenvolvimento guiado pelos testes indica o caminho da prevenção de erros, mais do que sua detecção e correção.

É verdadeiro o que se afirma em:

a) I e II apenas

b) II, III e IV apenas

c) I e III apenas

d) III e IV apenas

e) II e IV apenas

2. Em relação à prática da Refatoração, analise as afirmações que seguem:

I) prática que propõe a reconstrução do programa caso a quantidade de erros encontrados no código seja elevada.

II) refatorar significa melhorar o código já criado, sem que seja afetada qualquer funcionalidade.

III) a refatoração é uma das providências adotadas para facilitar manutenção futura do código.

IV) a refatoração é prática ligada ao código coletivo e ao padrão de codificação

É verdadeiro o que se afirma apenas em:

a) I e IV

b) II, III e IV

c) I

d) II

e) II e III

3. No contexto da prática da Integração Contínua, assinale a alternativa que contém expressões que completam corretamente as lacunas na sentença que segue.

"O trabalho de integração requer a criação de _____ no qual o _____ possa ser integrado diversas vezes ao dia. O recurso de _____ evita alterações simultâneas, no mesmo código".

a) padrão, código, concomitância

b) padronização, acesso, integração contínua

c) projeto, padrão, checkout
d) padrão, código, integração
e) repositório, código, checkout

Seção 2.4

Métodologia *Scrum*, suas características e aplicações

Diálogo aberto

Seja bem-vindo à última seção desta unidade!

A terceira etapa do processo de implantação da metodologia Extreme Programming na X-Soft foi concluída com sucesso. Como forma de introduzir o XP como modelo oficial da empresa, você cuidou da implantação do Desenvolvimento Guiado pelos Testes, Padrões de Codificação, Refatoração e Integração Contínua. Atualmente, a equipe já cumpre relativamente bem as novas rotinas e a experiência com o modelo ágil tem agradado os dirigentes da X-Soft.

No entanto, assim como acontece em qualquer processo de desenvolvimento, as práticas do XP precisam passar por maturação e aprimoramento constante. Você sabe que a correta operacionalização do modelo ágil depende, em parte, da contínua reafirmação dos seus valores. No mesmo sentido, a obtenção de bons resultados tem relação estreita com o encantamento que o modelo consegue provocar nos envolvidos e principalmente no cliente.

Está posto, então, seu novo desafio: por meio de documento próprio em que você deverá planejar ações que visem a manutenção e o aprimoramento das práticas do XP. Como consequência direta destas ações, deve também compor o documento suas sugestões para que a satisfação e o encantamento do cliente sejam garantidos.

Com esta atividade, você poderá aprofundar-se nos conceitos e na correta operacionalização das práticas do XP implantadas pela X-Soft e ter condições de, no futuro, propor variações, novidades e melhorias na rotina da equipe.

No entanto, o objetivo de aprendizagem desta seção é apresentar a metodologia Scrum, suas práticas e documentos relacionados, afim de que você conheça e assimile o funcionamento geral do Scrum e seja capaz de estabelecer comparações com o XP. Assim como o XP, ela tem sido utilizada em boa parte das empresas que decidiram adotar o modelo ágil de desenvolvimento. Com este conhecimento, você estará apto a fazer comparações entre as duas metodologias e a realizar escolhas com

toda autoridade. Em outras palavras, nesta seção você estará concluindo a aquisição da competência técnica planejada para esta unidade, que inclui conhecimento das principais metodologias ágeis de desenvolvimento e habilidade em contrapô-las com a metodologia tradicional. Tudo isso sem perder de vista, é claro, a competência geral proposta para o curso, que é conhecer as principais metodologias de desenvolvimento de software, normas de qualidade e processos de teste de software.

Bom trabalho e siga em frente com os estudos.

Não pode faltar

Nesta seção do seu livro didático você tomará contato com a metodologia ágil Scrum, cuja concepção inicial se deu na indústria automobilística em meados da década de 1980 e que tem o Sprint como o conceito mais importante. Com nomes diferentes, mas com ideias semelhantes, suas práticas se aproximam conceitualmente das práticas do XP e tornam esta metodologia bastante aceita entre as empresas de desenvolvimento de software.

Para que o Scrum seja devidamente apresentado a você, começaremos pela descrição do perfil dos seus atores. É importante destacar que, embora exista versão coerente na língua portuguesa para os nomes utilizados pela metodologia, manteremos os originais na língua inglesa.

Pois bem. Três figuras importantes fazem parte do Scrum:

1. Scrum Team: trata-se da equipe de desenvolvimento. A exemplo do XP, aqui também a quantidade de elementos da equipe responsável por um projeto gira em torno de 8 a 10 pessoas. Outra semelhança notável é a falta de especialização entre seus componentes: não há separação clara entre, por exemplo, a funções de analista, programador e projetista, pois todos colaboram para o desenvolvimento do produto em conjunto (WAZLAWICK, 2013).

2. Product Owner: literalmente, o dono do produto. Ele é responsável pelo projeto e por determinar quais funcionalidades serão implementadas em cada Sprint. A propósito, Sprint é o nome que o Scrum dá a cada período em que a equipe se reúne para, de fato, construir o produto. O contato direto com o cliente é mais uma atribuição importante do Product Owner.

3. Scrum Master: por conhecer bem a metodologia ele age como um facilitador no projeto e cuida para que as práticas do Scrum sejam seguidas. Não se trata, no entanto, de um gerente no sentido tradicional do termo. Pode ser um membro qualquer da equipe, preferencialmente bem articulado e experiente.

O Scrum, a exemplo do XP, também se desenvolve com base em práticas e documentos relacionados a elas. Na sequência você irá conhecer alguns destes documentos.

- Product Backlog: é um documento indispensável no modelo. Ele é criado pelo Scrum Master e contém as funcionalidades a serem implementadas no projeto. Você deve ter percebido que a palavra "todas" não antecedeu a palavra "funcionalidades" nesta última frase. Isso tem um motivo: o Product Backlog não precisa ser completo no início do projeto. É recomendável que o documento seja iniciado apenas com as funcionalidades mais evidentes. Depois, à medida em que o projeto avança, ele deverá conter novas funcionalidades que forem sendo descobertas.

Exemplificando

Veja um bom exemplo de Product Backlog (WAZLAWICK, 2013):

Product Backlog					
ID	Nome	Imp	PH	Como demonstrar	Notas
1	Depósito	30	5	Logar, abrir página de depósito, depositar R$ 10,00, ir para uma página de saldo e verificar se ele aumentou em R$ 10,00.	Precisa de um Diagrama de Sequência UML. Não há necessidade de se preocupar com criptografia, por enquanto.
2	Ver extrato	10	8	Logar, clicar em "Transações", fazer um depósito, voltar para "Transações", ver se o depósito apareceu.	Usar paginação para evitar consultas grandes ao BD. Design similar para visualizar página de usuário.

Que tal uma olhada mais cuidadosa nos campos que o Product Backlog contém? Vamos a eles (WAZLAWICK, 2013):

a. Id: trata-se de um contador cujo objetivo é não deixar que a equipe perca a trilha das estórias do cliente em caso de mudança de nome ou descrição. Pense nele como um identificador numérico da funcionalidade.

b. Nome: representa a estória do cliente em uma expressão fácil de ser lembrada.

c. Imp: este campo é bastante útil e interessante, já que expressa a importância que o cliente dá àquela funcionalidade. A equipe pode fazer outras convenções, mas geralmente números mais altos representam importâncias maiores.

d. PH: significa Pontos de Histórias e é a tradução da estimativa de esforço

para se transformar a estória em software. Você se lembra de como a equipe do XP planejava o desenvolvimento das funcionalidades? Pois bem, o principio aqui é o mesmo. Um ponto de história pode ser definido como o esforço de desenvolvimento de uma pessoa durante um dia ideal de trabalho, sem interrupções.

e. Como demonstrar: este campo descreve o que deve ser possível fazer para que a estória possa ser de fato implementada. Esta descrição deve ser de tal maneira bem detalhada a ponto de ser usada como um teste possível para a estória.

f. Notas: campo livre destinado às observações feitas pela equipe.

Reflita

Como você determina o grau de importância da funcionalidade? A observação das reações do cliente quando trata dela, a quantidade de vezes que é repetida por ele e a ênfase dada àquela característica do sistema revelam sua importância ao cliente. Naturalmente que nesta avaliação devem também ser considerados fatores técnicos e funcionalidades cuja existência justificam o produto, estes também têm alto grau de importância no projeto.

Os campos do Product Backlog aqui demonstrados não são os únicos viáveis. Você e sua equipe podem, a critério de ambos, estabelecer outros campos ou níveis de informações mais aprofundados, tudo em nome da boa comunicação e do perfeito entendimento do problema.

Pesquise mais

Quer saber mais sobre o Product Backlog? Acesse https://www.atlassian.com/agile/backlogs (Artigo em inglês. Acesso em: 6 jan. 2015.) e http://www.desenvolvimentoagil.com.br/scrum/product_backlog (Acesso em: 6 jan. 2015).

A prática que mais destaca o Scrum das outras metodologias ágeis é o Sprint. Se você é fã de atletismo ou de ciclismo certamente já relacionou o termo à estas modalidades. No Atletismo, o sprint é momento da corrida em que o competidor aumenta a velocidade para chegar na frente dos outros competidores.

No Scrum, é o momento de esforço concentrado – ou um ciclo de desenvolvimento em que determinadas funcionalidade viram programa. Conforme você já estudou, quem determina quais são estas funcionalidades é o Product Owner. Ele as prioriza e as registra durante o planejamento do ciclo, em uma reunião chamada Sprint Planning

Meeting e em um documento chamado Sprint Backlog. Tal documento nada mais é do que a lista dos itens extraídos do Product Backlog que serão desenvolvidos naquele determinado Sprint. Complicado? Pense então numa lista com as funcionalidades do produto e numa outra lista com itens - escolhidos e extraídos da primeira - que serão implementados na próxima vez em que os desenvolvedores se reunirem para este fim. Esta é a relação entre Product Backlog e Sprint Backlog.

É normal e desejável que esta segunda lista, embora contendo itens extraídos da primeira, os apresente com termos mais técnicos e voltados à maneira como a equipe irá construí-los.

Cada Sprint dura de uma a quatro semanas, dependendo da complexidade e da quantidade das funcionalidades a serem criadas. Durante este período, o Product Owner não irá levar à equipe outras funcionalidades. Ao contrário, as registrará para o próximo Sprint.

Bem, até o momento foi apresentada a você a seguinte rotina: com base nas estórias do cliente, o Product Owner cria uma lista de funcionalidades do sistema chamada Product Backlog. Quando a equipe se reúne para o ciclo de desenvolvimento (Sprint), o Product Owner cria a lista de funcionalidades que serão desenvolvidas naquele ciclo. Esta lista é derivada da primeira, leva o nome de Sprint Backlog e é criada durante a uma reunião chamada Sprint Planning Meeting. O Sprint dura poucas semanas e, enquanto acontece, nenhuma outra funcionalidade é enviada à equipe. Interessante, não acha? Mas ainda há um pouco mais a conhecer sobre o Sprint.

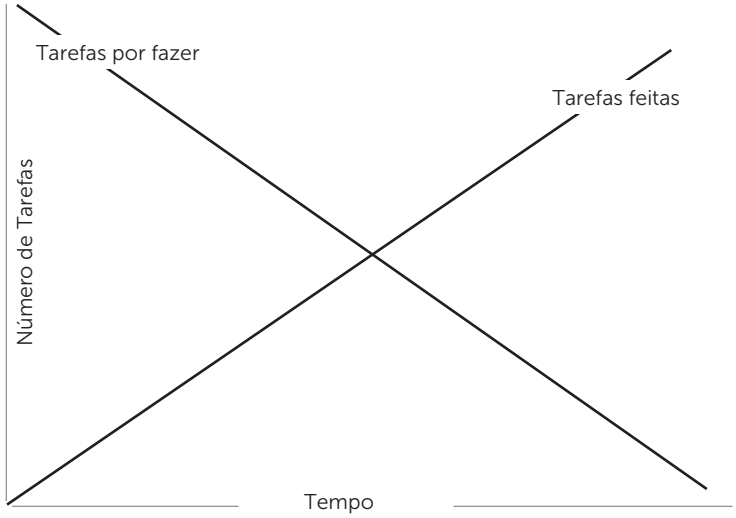
Assimile

A Sprint Planning Meeting é a reunião em que se planeja o próximo Sprint e se decidem as funcionalidades que serão implementadas naquele ciclo. Saiba um pouco mais em http://www.desenvolvimentoagil.com.br/scrum/sprint_planning_meeting. Acesso em: 21 dez. 2015.

À medida em que o Sprint avança e as funções do sistema vão sendo criadas, cabe ao Product Owner manter atualizada a lista de itens daquele ciclo. As tarefas já concluídas e aquelas ainda a serem feitas são mostradas em um quadro atualizado diariamente e à vista de todos. A cada dia pode-se avaliar o andamento das atividades, contando a quantidade de tarefas a fazer e a quantidade de tarefas terminadas, o que vai produzir um diagrama chamado Sprint Burndown (WAZLAWICK, 2013).

Se você considerar que a quantidade de trabalho já feito e a quantidade de trabalho a ser feito geram, cada um, uma linha neste diagrama, é natural pensar que a situação ideal ocorre quando o encontro das linhas se dá no centro do gráfico. Quer entender melhor? Observe a figura 2.4.

Figura 2.4 | Relação ideal entre tarefas



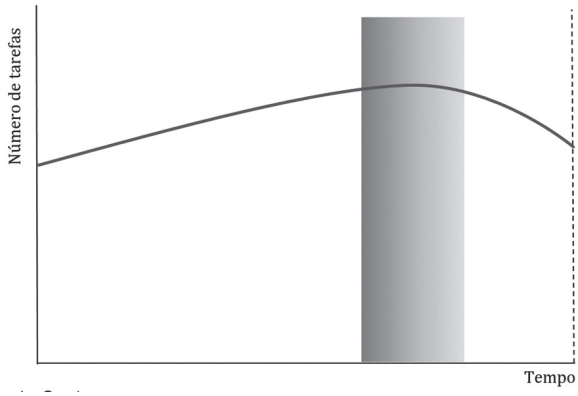
Fonte: Wazlawick (2013, p. 59).

Neste caso, o número de tarefas feitas cresce de forma sustentável em função do tempo, enquanto a quantidade de tarefas a serem feitas cai na mesma taxa.

Ao tomarmos familiaridade com a disposição da linha de "Tarefas por Fazer" no Sprint Burndown, seremos capazes de identificar alguns tipos de comportamentos da equipe (KANE MAR, 2006).

Como exemplo, a figura 2.5 mostra o comportamento de uma equipe que conseguiu entender as funcionalidades depois de passada a maior parte do Sprint. Esta curva pode indicar perfil de iniciante nos membros da equipe.

Figura 2.5 | comportamento típico de equipe iniciante



Fonte: O autor.

Terminado o Sprint, a equipe deve realizar nova reunião, esta chamada de Sprint Review Meeting, na qual será avaliado o produto gerado pelo Sprint.

Lembre-se

De maneira geral, o modelo ágil recomenda que seja entregue ao cliente, o mais cedo possível, um programa executável para que ele possa usar, testar e aprender mais sobre o que está sendo produzido. O Scrum leva à sério este princípio e entrega ao cliente parte do programa após o término de cada ciclo de desenvolvimento.

O Scrum, a exemplo do XP, também prevê a realização diária de reunião na qual o dia atual é planejado e o dia anterior é revisado. A recomendação é que este encontro, chamado de Daily Scrum - também seja feito com seus participantes em pé e que dure apenas alguns minutos.

Faça você mesmo

Busque em http://kanemar.com/2006/11/07/seven-common-sprint-burndown-graph-signatures/#more-113 ou em https://books.google.com.br/books?id=Qtg4VUkE0V0C&pg=PT95&lpg=PT95&dq=comportamento+da+equipe+sprint+burndown&source=bl&ots=N7QxESknpq&sig=vWJVzxJBfTYJPUI4kumc571Yt8A&hl=pt-BR&sa=X&ved=0ahUKEwiAyby09ZXKAhXIQZAKHY60DXkQ6AEIJTAB#v=onepage&q&f=false (Acesso em: 6 jan. 2016) os outros comportamentos de equipes sugeridos pela linha de tarefas por fazer em um gráfico de acompanhamento de Sprint e coloque-os em um relatório de até duas páginas.

Sem medo de errar

Todas as práticas do XP que a X-Soft planejou implantar já estão fazendo parte da rotina das equipes de desenvolvimento. Num primeiro momento, as práticas mais diretamente relacionadas à comunicação e ao feedback tomaram seu lugar no cotidiano dos desenvolvedores e, na medida em que experimentavam maturação, outras relacionadas ao código do programa, testes e integração foram também sendo introduzidas.

As coisas andam bem, mas você entende que elas ainda podem ser melhoradas e que as práticas implantadas podem contribuir ainda mais com o sucesso da sua empresa. Aprimorar os processos internos tem relação quase direta com o encantamento que o XP se propõe a provocar nos clientes. E é exatamente isso que

você quer para a X-Soft.

Novamente você deve recorrer ao que foi proposto no "Diálogo Aberto" para retomar o desafio desta seção. Em um relatório você poderá planejar ações que impliquem na manutenção e no aprimoramento das práticas do XP. Como desdobramento direto destas ações, o estreitamento da relação com os clientes deveria também ser buscado. Deste ponto em diante será apresentada uma forma possível de compor este relatório.

1. Ao menos dois membros previamente escolhidos da equipe deverão ficar responsáveis pelo treinamento dos novos desenvolvedores que venham a ingressar na X-Soft. O sucesso das práticas não pode estar relacionado à pessoa que as realiza, mas à cultura implantada na empresa.

2. Em intervalos regulares, a direção da X-Soft (ou alguém designado por ela) deverá colher percepções e sugestões da equipe sobre sua rotina. Com esta providência, espera-se que eventuais insatisfações em relação ao trabalho venham à tona, sejam sanadas na medida do possível e não se tornem pretexto para descumprimento das práticas implantadas.

3. Deve ser disponibilizado um repositório de experiências e melhores práticas, que possa ter seu conteúdo editado por todos. Assim, a comunicação terá chance de ser aprimorada e experiências vividas no modelo poderão ser facilmente compartilhadas.

4. Por fim, a parte do relatório que aborda o encantamento do cliente deve prever meios de se estabelecer relacionamento duradouro e profícuo com ele. As entregas regulares e corretamente testadas, a cobrança de valor justo, o estabelecimento de regras claras desde o início do projeto e o definitivo posicionamento do cliente como centro do projeto são ótimas providências para que ele continue a encomendar novos projetos e a chamá-lo para aprimorar os já entregues, o que gera valor à empresa.

Atenção!

Não se pode esperar que um cliente, que eventualmente já tenha participado de projetos no modelo tradicional, incorpore imediatamente o modo de operação do modelo ágil. Nas primeiras experiências, ele precisará de motivação e de confiança no trabalho da equipe e de seus gestores.

Lembre-se

O encantamento do cliente gera fidelização. Descubra mais em http://blogbrasil.comstor.com/como-a-sua-revenda-de-ti-pode-fidelizar-e-cuidar-melhor-dos-clientes (Acesso em 21 de dezembro de 2015). Opcionalmente, você pode acessar também http://www.administradores.com.br/artigos/negocios/5-dicas-para-atender-bem-o-cliente/33105 (Acesso em: 6 jan. 2016), artigo indicado pelo próprio autor do blog.

Avançando na prática

Pratique mais
Instrução Desafiamos você a praticar o que aprendeu transferindo seus conhecimentos para novas situações que pode encontrar no ambiente de trabalho. Realize as atividades e depois compare-as com a de seus colegas.

Colocando o desenvolvimento na trilha da previsibilidade	
1. Competência Geral	Conhecer as principais metodologias de desenvolvimento de software, normas de qualidade e processos de teste de software.
2. Objetivos de aprendizagem	Apresentar a metodologia Scrum, suas práticas e documentos relacionados, a fim de que o aluno conheça e assimile o funcionamento geral do Scrum e seja capaz de estabelecer comparações com o XP.
3. Conteúdos relacionados	Métodos ágeis - Scrum: práticas, características e aplicabilidade. Comparação com o XP.
4. Descrição da SP	A empresa Core-Soft, há pouco estabelecida no mercado de TI, vem sofrendo com as constantes mudanças de ideia dos seus clientes em relação às suas necessidades. Esta instabilidade tem gerado interrupções e mudanças bruscas de direção no processo de criação do software. Mesmo depois de todos os requisitos descobertos, tem sido muito comum que o cliente os reforme, obrigando a equipe a refazer partes do sistema e mudar suas prioridades. Sua missão é sugerir ações – e colocá-las em relatório - para que esta realidade seja alterada para um cenário no qual o desenvolvimento não passe por interrupções e mudanças tão frequentes. Ao contrário, a implementação do software deverá trilhar o caminho da constância e da previsibilidade.
5. Resolução da SP	Não se pode imaginar que, de um dia para o outro, a realidade atual de desenvolvimento da Core-Soft será mudada. As ações planejadas, por mais bem especificadas que estejam, levarão tempo até se tornarem efetivas. No entanto, o caminho da mudança tem que ser indicado. Na sequência você terá algumas sugestões de como iniciar a mudança deste cenário. A intenção de especificar todas as funcionalidades do sistema logo no início do processo de desenvolvimento tem grande chance de causar retrabalho e frustrações na equipe e no cliente. O Scrum sugere que as necessidades do cliente sejam colocadas de forma gradual, com o estabelecimento de prioridades de desenvolvimento e a previsão de que ele (cliente) poderá mudar de ideia. Depois que uma funcionalidade é esboçada pelo cliente – mesmo de forma incompleta e passível de mudanças – as partes deverão acordar prazos para seu término e período no qual nenhuma alteração será acatada. Esta providência, que adota característica do Sprint, deverá blindar a equipe da insegurança causada pelas quase certeiras interferências no processo de desenvolvimento. Por fim, é sempre bom lembrar, qualquer ação que passe pela mudança de mentalidade do cliente requer estreitamento das relações e sua elevação para a condição de ator central do processo.

Lembre-se

"Ao final de um Sprint, a equipe apresenta as funcionalidades implementadas em uma Sprint Review Meeting. Finalmente, faz-se uma Sprint Retrospective e a equipe parte para o planejamento do próximo Sprint. Assim reinicia-se o ciclo". (Disponível em http://www.desenvolvimentoagil.com.br/scrum/. Acesso em: 22 dez. 2015).

Faça você mesmo

A adoção do Scrum como modelo de desenvolvimento tem sido cada vez mais constante e bem-sucedida mundo afora. Relate ao menos um caso em que o Scrum tenha sido introduzido com sucesso em uma organização, nacional ou não.

Faça valer a pena

1. Em relação aos perfis presentes numa equipe Scrum, assinale a alternativa que contém a associação correta entre as colunas abaixo.

I. Scrum Team	1. Responsável por determinar quais funcionalidade serão implementadas no Sprint.
II. Product Owner	2. Equipe normalmente estruturada sem clara divisão de especialidades entre seus membros.
III. Scrum Master	3. Responsável pela correta aplicação das práticas do Scrum.

a) I – 1; II – 3; III – 2
b) II – 3; I – 2; III – 1
c) III – 3; I – 2; II – 1
d) II – 2; I – 1; III – 3
e) III – 1; I – 3; II – 2

2. Assinale a alternativa que descreve situações própria do Sprint.

a) As funcionalidades específicas a serem desenvolvidas estão contidas no

Product Backlog.

b) O Product Owner se compromete a não priorizar funcionalidades no período em que o Sprint ocorre.

c) A Sprint Planning Meeting acontece assim que o Sprint termina.

d) As tarefas a serem cumpridas em um Sprint são transferidas do Sprint Backlog para o Product Backlog.

e) As tarefas a serem cumpridas em um Sprint são transferidas do Product Backlog para o Sprint Backlog.

3. Em relação ao Scrum Master, analise as afirmações que seguem:

I) profissional com talento notável para programação e referência técnica na equipe.

II) líder técnico e com capacidade para cuidar da manutenção das práticas do Scrum durante o projeto.

III) dono do produto e aquele que paga por ele.

IV) auxiliar de programação e hierarquicamente situado abaixo de qualquer membro da equipe.

V) profissional responsável pela captação de projetos e com perfil de vendas.

É verdadeiro o que se afirma apenas em:

a) I

b) II

c) III

d) IV

e) V

Referências

BECK, Kent et al. **Manifesto para o desenvolvimento ágil de software**. 2001. Disponível em: http://www.manifestoagil.com.br/index.html. Acesso em: 18 jan. 2016.

HIBBSs, C.; JEWETT, S.; SULLIVAN, M. **The Art of Lean Software Development**: A Practical and Incremental Approach, 1ª Edição, O'Reilly Media, Inc., CA. 2009

MAR, Kane. **Seven common sprint burndown graph signatures**. 2006. Disponivel em: http://kanemar.com/2006/11/07/seven-common-sprint-burndown-graph-signatures/#more-113. Acesso em: 21 jan. 2016.

REZENDE, Denis Alcides. **Engenharia de software e sistemas de informação**. 3. ed. rev. e ampl. Rio de Janeiro: Brasport, 2005.

SCHACH, Stephen. **Engenharia de software**: os paradigmas clássico e orientado a objetos. 7.ed. São Paulo: McGraw-Hill, 2008.

SOMMERVILLE, Ian. **Engenharia de software**. 8.ed. São Paulo: Addison Wesley, 2008. 592 p. ISBN 85-88639-07-6.

TELES, Vinicius Manhães. ***Extreme Programming***: aprenda como encantar seus usuários desenvolvendo software com agilidade e alta qualidade. São Paulo: Novatec, 2004. 316 p. ISBN 8575220470.

PRESSMAN, Roger S. **Engenharia de software**. São Paulo: Pearson Prentice Hall, 2009. 1056 p. ISBN 8534602379.

PAULA FILHO, Wilson de Pádua. **Engenharia de software**: fundamentos, métodos e padrões. 2. ed. Rio de Janeiro: Livros Técnicos e Científicos, 2005. 602 p. ISBN 8521613393.

WAZLAWICK, Raul Sidnei. **Engenharia de software**: conceitos e práticas. 1.ed. Rio de Janeiro: Elsiever, 2013.