Engenharia de Software

Engenharia de software

Roque Maitino Neto

Dados Internacionais de Catalogação na Publicação (CIP)

Maitino Neto, Roque
M232e Engenharia de software / Roque Maitino Neto. –
Londrina : Editora e Distribuidora Educacional S.A., 2016.
216 p.

ISBN 978-85-8482-416-8

1. Engenharia de software. 2. Software - Desenvolvimento. 3. Software – Testes. 4. Software – Controle de qualidade. I. Título.

CDD 005.1

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

Palavras do autor

Caro aluno, seja bem-vindo!

Você pode imaginar que desenvolver uma solução computacional que esteja apta a atender expectativas e necessidades de quem a demandou não é tarefa simples. A dificuldade de comunicação entre equipe de desenvolvimento e cliente, a adoção de procedimentos equivocados (ou a falta deles) e a dificuldade natural em se criar um programa correto são condições que, consideradas de forma isolada ou combinada, podem acarretar expectativas frustradas e necessidades mal atendidas.

Mas será que não temos como mudar isso? Há uma saída: o aprimoramento das práticas de trabalho e a estruturação de metodologias eficientes de desenvolvimento de *software* têm ajudado a reduzir os casos de insucesso nos projetos ou, na pior das hipóteses, mitigar seus efeitos. Desde seu surgimento no final da década de 1960, a Engenharia de *software* tem se mostrado o melhor modo – senão o único – de tornar viável um projeto de *software* e de possibilitar a entrega de produtos com qualidade e confiabilidade satisfatórias.

Como é possível, então, utilizarmos a Engenharia de *software* para obter sucesso no desenvolvimento de soluções computacionais? Como você proporcionaria à sua equipe uma maneira segura de atingir os objetivos do projeto sem que seus membros se percam entre procedimentos complexos, normas rígidas de processo, documentação excessivamente detalhada e intermináveis seções de planejamento? Como a Engenharia de *software* tem tentado, por meio de metodologias mais leves, ágeis e focadas no desenvolvedor, corrigir erros cometidos no passado?

Você vai constatar que este material didático trata dessas e de outras questões ao apresentar conceitos e propor desafios para você, aluno. A unidade 1 aborda questões introdutórias da Engenharia de *software*, as principais e mais tradicionais metodologias de desenvolvimento e as dificuldades em adequá-las às características atuais das equipes de desenvolvimento e às expectativas dos clientes.

A unidade 2 trata das novas práticas de desenvolvimento e como, por meio delas, você poderá tornar mais ágil e flexível o processo

de construção de uma solução computacional. As metodologias *Extreme Programming* e *Scrum* estão em foco nesta unidade e você poderá, por meio das práticas que adotam, decidir se uma equipe de desenvolvimento deve evitar a todo custo mudanças nos requisitos iniciais do sistema ou simplesmente tratá-las como inevitáveis.

A qualidade do produto de *software* é o tema da unidade 3. Você terá contato com algumas das mais eficientes práticas de garantia da qualidade e normas relacionadas a elas, com enfoque nas revisões e métricas voltadas à construção de produto que ofereça confiabilidade e perfeita adequação ao seu propósito.

Por fim, na unidade 4, assuntos relacionados ao teste de *software* são desenvolvidos de forma a proporcionar a você visões e métodos diversificados de como submeter um produto executável a verificações e validações. A participação do cliente e/ou usuário neste processo é mostrada como fundamental para seu sucesso.

Pois bem, esse é o desafio. O bom aproveitamento nesta disciplina – assim como nas demais - pressupõe sua dedicação nas atividades de pré e pós-aula, bem como sua efetiva participação nas discussões que se seguirão nos encontros presenciais. As atividades de autoestudo terão importância destacada na construção de sua competência para distinguir e compreender as metodologias de desenvolvimento de *software* abordadas e, de acordo com a necessidade do caso real, empreendê-las para o sucesso do seu projeto.

Bom trabalho!

Fundamentos de Engenharia de *software*

Convite ao estudo

Olá, seja bem-vindo ao módulo de introdução à Engenharia de *software*! Você sabia que as boas práticas associadas à Engenharia de *software* têm servido como apoio para todos os envolvidos no processo de desenvolvimento de um produto de *software*? Pois é! Porém, essas práticas não se consolidaram da noite para o dia. Ao contrário, foram sendo construídas e estruturadas conforme as experiências em projetos se acumulavam entre as equipes. Em nossos dias, o desafio do desenvolvimento de um produto de *software* não pode ser enfrentado sem condutas estruturadas e padrões mínimos de procedimentos.

Esta primeira unidade será um dos passos para que você seja capaz de conhecer as principais metodologias de desenvolvimento de *software*, normas de qualidade e processos de teste de *software*.

Para esta unidade foi preparado conteúdo que vai colocar você diante dos tópicos iniciais da Engenharia de *Software*, apresentados na medida certa para proporcionar seu primeiro contato com o tema. Estudaremos juntos os conceitos iniciais da disciplina, seus paradigmas e objetivos, sempre com foco na sua preparação para transitar com naturalidade pelas novas metodologias de desenvolvimento e pelos padrões de qualidade que o desenvolvimento moderno impõe.

São objetivos de aprendizagem desta seção que você conheça aspectos gerais, conceitos, objetivos e paradigmas da Engenharia de *software*, assim como fatos históricos do tema e a crise pela qual o desenvolvimento de *software* passou.

Os próximos parágrafos apresentam situação comum a muitas organizações que assumiram a missão de automatizar processos

por meio de programas de computador. Tal situação dará base para nossa caminhada pelas demais unidades deste material didático.

Vamos iniciar pela seguinte situação: você é sócio-proprietário de uma *startup* de desenvolvimento de *software*, chamada "*XAX-Sooft*". Um grande cliente, especializado em venda de games, solicitou um projeto de *software* para cadastro de clientes por interesse, o que vai possibilitar contatos mais assertivos e direcionamentos seguros de campanhas. Acontece que para atender esse cliente com a qualidade que ele demanda, mudanças nos processos de desenvolvimento da *XAX-Sooft* deverão ser implementadas. Não há metodologia formal de desenvolvimento implantada na empresa e as atividades são executadas sem acompanhamento e validação.

Para que sua tarefa seja cumprida você deverá apresentar:

1. Levantamento do cenário atual do processo de desenvolvimento utilizado na *XAX-Sooft*.

2. Proposta de melhoria do processo de *software* utilizado pela *XAX-Sooft*.

3. Levantamentos dos requisitos do *software* e esboço do projeto.

4. Definição do processo de implantação do *software*.

Nas seções seguintes você terá à disposição textos estruturados em formatação padronizada que o levarão a conhecer as motivações da existência da Engenharia de *Software*. No "Diálogo aberto", as situações da realidade profissional serão problematizadas e, na sequência, será abordada a teorização que nos dará base para os exercícios e o desenvolvimento das outras situações do dia a dia que se seguirão.

Vale a pena, de fato, investir tempo em aplicação de metodologia? Temos feito a divisão correta das etapas de desenvolvimento de um produto de *software*? Ao aprofundar-se nos temas aqui abordados, você será capaz de responder a essas e outras tantas questões.

Seção 1.1

Introdução à Engenharia de *software*: aspectos gerais, objetivos, evolução do *software* e crise do *software*

Diálogo aberto

Nossa empresa *XAX-Sooft* cresceu! Agora bons contratos são fechados e clientes importantes estão sendo conquistados.

Temendo a perda de controle sobre os projetos e buscando bom atendimento à demanda do cliente de venda de games, os dirigentes da *XAX-Sooft* resolvem que os processos de desenvolvimento e o gerenciamento dos projetos devem ser repensados. Após algum tempo de discussão, constatam que os processos de desenvolvimento que adotam são caóticos e que o início da solução passa pelo levantamento de como os procedimentos são executados atualmente.

Sua tarefa é fazer o levantamento dos procedimentos de desenvolvimento atuais adotado na "*XAX-Sooft*". Utilize os conhecimentos que serão abordados nesta seção 1.1, principalmente os relacionados às características do período chamado "crise do *software*".

Será que uma nova metodologia deve ser discutida com todos os envolvidos antes de sua adoção? Assumindo que a empresa hoje cumpre com a maioria dos prazos e tem clientes minimamente satisfeitos, qual o motivo de mudar o que está dando certo? Essas questões terão suas respostas direcionadas nas próximas seções.

Não pode faltar

Adiante serão descritos conceitos, objetivos e paradigmas relacionados à Engenharia de *software* que embasarão temas mais avançados e pavimentarão o caminho para metodologias mais adequadas ao cenário atual de demandas dos clientes.

Ensina Schach (2008), em sua obra "Engenharia de *software*: os paradigmas clássico e orientado a objetos", que "Engenharia de *software* é uma disciplina cujo objetivo é produzir *software* isento de

falhas, entregue dentro do prazo e orçamentos previstos, e que atenda às necessidades do cliente. Além disso, o *software* deve ser fácil de ser modificado quando as necessidades dos usuários mudarem".

Alternativamente, para uma melhor definição do conceito de Engenharia de *software*, faz-se necessária a explicação isolada dos termos que o compõem. De forma genérica, pode-se definir *software* como (i) instruções que, quando executadas, produzem a função desejada, (ii) estruturas de dados que possibilitam que os programas manipulem a informação e (iii) documentação relativa ao sistema. Engenharia diz respeito ao projeto e manufatura, circunstâncias nas quais os requisitos e as especificações do produto assumem importância crítica na qualidade final do produto. Trata-se da definição clássica de Engenharia. Por ser imaterial, um programa de computador não passa por um processo de manufatura como se conhece no meio industrial de produtos complexos. Fica claro também que, apesar da semelhança com a engenharia tradicional, a produção de programas de computador possui situações particulares.

A IEEE Computer Society (2004) define Engenharia de *software* como: "A aplicação de uma abordagem sistemática, disciplinada e quantificável de desenvolvimento, operação e manutenção do *software*, além do estudo dessas abordagens".

Fica claro, então, que o objetivo da Engenharia de *Software* é a entrega de produto de qualidade, respeitados os prazos e os limites de dispêndio de recursos humanos e financeiros.

Assimile

Por se tratar de assunto amplamente abordado na literatura, a Engenharia de *software* acumulou várias definições durante seus anos de existência como disciplina. Vale a pena conhecer mais uma:

"Engenharia de *Software* é a profissão dedicada a projetar, implementar e modificar *software*, de forma que ele seja de alta qualidade, a um custo razoável, manutenível e rápido de construir." (LAPLANTE, 2007, p. 39)

Como toda disciplina, a nossa também apresenta aspectos que a norteiam, comumente referenciados como princípios ou paradigmas. Vale a menção de alguns deles:

Abstração: para resolver um problema, deve-se separar os aspectos que estão ligados a uma realidade particular, visando representá-lo em forma simplificada e geral.

Formalidade: significa seguir uma abordagem rigorosa e metódica para resolver um problema.

Dividir para conquistar: resolver um problema complexo dividindo-o em um conjunto de problemas menores e independentes que são mais fáceis de serem compreendidos e resolvidos.

Organização hierárquica: organizar os componentes de uma solução em uma estrutura hierárquica tipo árvore. Assim, a estrutura pode ser compreendida e construída nível por nível, cada novo nível com mais detalhes.

Ocultação: esconder as informações não essenciais. Permitir que o módulo "veja" apenas a informação necessária àquele módulo.

Localização: colocar juntos os itens relacionados logicamente (o usuário não pensa como o analista!).

Integridade conceitual: seguir uma filosofia e arquitetura de projeto coerentes.

Completeza: checar para garantir que nada foi omitido.

Agora que você já teve contato com os pilares da Engenharia de *Software*, vale a pena focar naquele que é seu produto final. Conheça algumas das principais categorias de *software*, classificadas segundo sua aplicação:

***Software* básico**: apoio a outros programas. Forte interação com o *hardware*. Exemplos: compiladores, *device drivers*, componentes de sistema operacional.

***Software* em tempo real**: trata-se de um tipo de *software* que monitora eventos por meio de coleta e análise de dados, tais como temperatura, pressão, vazão, entre outros. Usa-se a expressão "tempo real" por conta da resposta imediata (um segundo ou menos) que o *software* deve fornecer.

***Software* comercial**: caracteriza-se pela manipulação de grande volume de dados e uso em aplicações comerciais. Exemplos: folha de pagamento, estoque, recursos humanos. Forte interação com banco de dados.

***Software* científico**: algoritmos de processamento numérico. Usados na astronomia, mecânica e projeto auxiliado por computador.

***Software* de computador pessoal**: forte interação com o ser humano. Deve ser fácil e amigável. Exemplos: Planilhas, editores de texto, *browsers*, entre outros.

Assimile

A maioria dos programas com os quais temos contato são de computador pessoal. Nesta categoria de programas podemos destacar ainda o *software on-line*, que necessita de conexão com a internet para funcionar. Ele normalmente é executado em um computador distante fisicamente do usuário, mas usa a máquina local para apresentação das entradas e saídas de dados.

Em algum momento da sua vida profissional você estará envolvido com o desenvolvimento de um ou mais desses tipos de programas de computador. Aliás, desenvolver *software* é uma atividade que tem deixado de ser artesanal e empírica para se tornar sistemática e organizada. No entanto, logo em seus primeiros anos, a produção de *software* enfrentou tempos turbulentos, nos quais a chance de insucesso nos projetos era grande.

Na década de 1960, alguns atores do processo de desenvolvimento de *software* cunharam a expressão "Crise do *Software*" na intenção de evidenciar o momento adverso que a atividade atravessava. Em seu sentido literal, crise indica estado de incerteza ou declínio e, de fato, esse era o retrato de um setor inapto a atender demanda crescente por produção de *software*, que entregava programas que não funcionavam corretamente, construídos por meio de processos falhos e que não podiam passar por manutenção facilmente. Além disso, a incerteza causada pela imprecisão nas estimativas de custo e prazo afetava a confiança das equipes e principalmente dos seus clientes.

A precária – e muitas vezes ignorada – comunicação entre cliente e equipe de desenvolvimento contribuía para que a qualidade do

levantamento dos requisitos fosse perigosamente baixa, acarretando consequente incorreções no produto final.

O cenário era agravado pela inexistência de métricas que retornassem avaliações seguras – fossem qualitativas ou quantitativas – dos subprodutos gerados nas fases de requisitos, projeto, implementação e testes. Em seções seguintes trataremos em detalhes dessas fases.

Não havia, ainda, dados históricos de outros projetos que ajudassem nas estimativas para os projetos atuais e na adequada avaliação da eficácia da aplicação de uma ou outra metodologia no desenvolvimento.

Quando, apesar das adversidades, o programa era entregue, o processo de implantação tendia a ser turbulento, já que raramente eram considerados os impactos que o novo sistema causaria na organização. Treinamentos aos usuários após a implantação não era atividade prioritária e o fator humano era ignorado com frequência, gerando insatisfações nos funcionários impactados.

Por fim, há que se considerar a dificuldade e o alto valor em se empreender manutenção nos produtos em funcionamento, normalmente ocasionados por projetos mal elaborados.

Estava claro que algo deveria ser feito. Ações que aprimorassem e dessem segurança ao processo de desenvolvimento deveriam ser tomadas. Sob pena de verem seu negócio naufragar, empreendedores de TI deveriam a todo custo entrar em sintonia com seus clientes, trazendo-os para dentro do processo e dando voz ativa a eles. Na Unidade 2 tais ações serão abordadas em detalhes.

Reflita

Em 2002, uma empresa global de pesquisa em Tecnologia da Informação chamada *Cutter Consortium*, constatou que 78% das empresas de TI pesquisadas fizeram parte de ações judiciais motivadas por desavenças relacionadas a seus produtos. Na maioria desses casos (67%), os clientes reclamavam que as funcionalidades entregues não correspondiam à suas demandas. Em outros tantos casos, a alegação era que a data prometida para entrega havia sido por várias vezes desrespeitada e, por fim, em menor quantidade, a reclamação se originava do fato de o sistema ser tão ruim que simplesmente se podia utilizá-lo.

Estarrecedor, não acha?

Pesquise mais

O texto "Uma breve história da Engenharia de *Software*" foi publicado pelo renomado professor suíço Niklaus Wirth, em 2007. O texto original está disponível em: <http://www.helwan.edu.eg/university/staff/Dr.WaleedYousef.../DownLoads/SoftwareEngineeringI/Wirth2008BriefHistorySWE.pdf>. Acesso em: 27 set. 2015.

Exemplificando

Certa equipe de desenvolvimento construiu um processador de texto. Ao planejar o *menu* da ferramenta, a funcionalidade de classificação por ordem alfabética acabou ficando no *menu* "*Layout* de página". Tempo depois, a mesma equipe foi chamada a construir um sistema acadêmico. Para a fase de análise, a equipe escolheu a metodologia da Análise Estruturada e, para o projeto, a metodologia de Projeto Orientado a Objeto.

A Engenharia de *Software* norteia-se por princípios que devem ser respeitados para que sua prática leve ao cumprimento de seus objetivos. No caso apresentado, dois desses princípios foram ignorados pela equipe de João.

Ao pensar no *menu*, a equipe não considerou que a localização dos elementos no programa final é de suma importância para que o usuário o utilize com desenvoltura. Quando um *menu* de programa é identificado como "*Layout* de página", não se espera que a funcionalidade de classificação por ordem alfabética lá esteja localizada.

No segundo projeto, a equipe simplesmente ignorou que o projeto deve respeitar o princípio da integridade conceitual ao não considerar a adoção de uma metodologia do início ao fim do desenvolvimento.

Faça você mesmo

A fim de prepará-lo para as próximas seções deste material didático, um breve exercício é proposto: se você recebesse a missão de construir o sistema de controle acadêmico de uma faculdade recém-aberta por um conhecido seu, por onde você começaria o desenvolvimento? Qual seria o seu primeiro passo? Responda em no máximo cinco linhas de texto.

Sem medo de errar

Todos na *XAX-Sooft* concordam que o momento é apropriado para a implantação de novos procedimentos. Nunca antes haviam se preocupado com a formalização de seus procedimentos, nem com a documentação relativa a eles. Nem sequer um treinamento foi disponibilizado à equipe até então. Objetivamente, o cenário descrito demanda quebra de paradigmas, com o consequente aumento do esforço empreendido que toda mudança acarreta.

No entanto, as coisas podem começar a ser mudadas por meio do levantamento da situação atual. Este levantamento deve responder às seguintes questões:

Como as funções do sistema a ser desenvolvido são descobertas? Como são coletadas? Quem as informa?

O que a equipe faz depois que entende o que deve ser feito? Inicia imediatamente a programação?

Durante o desenvolvimento do programa, o cliente é novamente chamado em caso de dúvida da equipe?

Com o sistema pronto, a equipe simplesmente o disponibiliza ao cliente? Treinamentos são previstos?

Não lhe parece, caro aluno, que estamos diante de uma situação em que os projetos são desenvolvidos unicamente ao sabor da experiência da equipe? Caso um membro da equipe deixe a empresa, seu substituto será capaz de continuar seu trabalho da forma como era feito?

Pois é, a situação demanda providências e você é chamado a tomá-las.

O levantamento dos procedimentos de desenvolvimento atuais adotados na *XAX-Sooft* apontou que:

1. Quando a equipe de desenvolvimento recebe um novo projeto, um de seus membros – normalmente o que se encontra com carga de trabalho menor no momento – é destacado para visitar o cliente e reunir-se com ele. Durante a reunião, todos os requisitos do sistema

são discutidos e anotados. Em suma, a única fonte dos requisitos é o cliente e a coleta que se faz pela anotação em uma folha de papel.

2. Uma vez coletados os requisitos, o membro da equipe que o fez os repassa a dois ou mais colegas para que sejam imediatamente traduzidos em uma linguagem de programação. A quantidade de profissionais que cuidarão do projeto e as tecnologias a serem utilizadas são determinadas por um dos sócios da *XAX-Sooft*.

3. Caso ocorram dúvidas durante o desenvolvimento da solução, uma ligação ou alguns e-mails enviados ao cliente servirão como meios para saná-las. Caso o cliente não seja encontrado ou não seja capaz de resolver o questionamento da equipe, o caso é decidido pelo programador mesmo. Há consenso na equipe que, embora o cliente não tenha solicitado determinadas funções, elas serão desenvolvidas por precaução e para o caso de serem solicitadas no futuro.

4. Terminado o desenvolvimento, agenda-se a implantação do sistema e a equipe aguarda novos contatos do cliente.

Atenção

Nosso objetivo, no momento, é iniciar a colocação da nossa empresa no caminho da criação de produtos de qualidade e que estejam perfeitamente adequados ao seu propósito. Não nos cabe, por ora, fornecer a solução completa para o caso, pois isso demandaria outros conhecimentos que ainda não temos.

Lembre-se

A criação de conteúdos compartilhados tem crescido nas empresas. Ferramentas do pacote *Office* e do *Google* possuem funcionalidades que permitem a criação de *wiki* corporativa. Para saber mais, acesse link disponível em: <http://www.sabesim.com.br/o-que-e-um-wiki-corporativo/>. Acesso em: 21 jun. 2016.

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.

Nova metodologia para colaboradores experientes	
1. Competência geral	Conhecer os fundamentos da Engenharia de *Software* e sua importância no processo de desenvolvimento de sistemas.
2. Objetivos de aprendizagem	Transferência dos conceitos aprendidos para situação-problema semelhante.
3. Conteúdos relacionados	Introdução de nova metodologia de desenvolvimento, problemas de comunicação, fundamentos da Engenharia de *Software*.
4. Descrição da SP	Uma outra empresa de desenvolvimento de *software*, em situação semelhante à que temos descrito, também decide transformar sua metodologia de trabalho. Logo após ser iniciado o processo de mudança, os gestores percebem resistência à novidade, imposta justamente pelos colaboradores mais experientes e com carreira mais longa, perfil mais comum entre os membros das equipes. Apesar de obrigá-los a adotar os novos procedimentos, os gestores têm conseguido obter os resultados desejados de seus colaboradores.
5. Resolução da SP	Essa situação é bastante comum. A atitude menos aconselhada é obrigar a adoção do procedimento sem que sejam demonstradas as reais vantagens que todos terão com ele. Cabe aos gestores investir tempo em esclarecimentos e simulações de como a nova metodologia irá permitir diminuição do retrabalho, padronização das tarefas e consequente redução de problemas após a entrega. A solução, neste caso, não pode deixar de passar pela conscientização dos envolvidos e, se for o caso, pela implantação gradual do novo processo.

Lembre-se

A boa comunicação no ambiente de trabalho é fundamental para a assimilação de um novo procedimento ou de novas maneiras de agir. Quer saber mais? Sugerimos a leitura de um texto curto e objetivo que pode ser consultado em: <http://www.vert.com.br/blog-vert/a-importancia-de-ter-uma-boa-comunicacao-com-a-sua-equipe/>. Acesso em: 3 jun. 2016.

Faça você mesmo

Embora profissionais mais experientes possam ser propensos a oferecer resistência à algumas novidades, ainda assim você poderá encontrar entre eles colaboradores vivamente interessados no sucesso da nova metodologia. No entanto, por receio de represálias dos colegas, essas pessoas acabam não manifestando sua motivação. Na condição de gestor do empreendimento, elabore um plano em poucas linhas que lhe permitirá identificar esses profissionais e, por meio deles, disseminar os novos procedimentos.

Faça valer a pena

1. De acordo com os conceitos apresentados, pode ser classificado como objetivo da Engenharia de *software*:

a) Melhoria da comunicação entre a equipe de desenvolvimento.

b) Cumprimento de prazos.

c) Entrega de *software* adequado ao seu propósito, respeitados prazo e orçamento estabelecidos.

d) Aprimoramento dos conhecimentos da equipe em programação de computadores.

e) União da equipe de desenvolvimento.

2. São situações típicas da chamada "Crise do *software*":

a) Entregas pontuais e clientes satisfeitos.

b) Métricas não confiáveis e histórico de projetos anteriores disponíveis.

c) Limites orçamentários respeitados e treinamento adequado aos usuários.

d) Projetos mal elaborados e geração de produtos de difícil manutenção.

e) A crise do *software* nunca existiu.

3. Assinale a alternativa que contém os tipos de *software* que completam corretamente as lacunas nas frases abaixo.

I- _____ monitora eventos do mundo real.

II- _____ manipula grandes quantidades de dados e tem alto nível de comunicação com sistemas de banco de dados.

III- _____deve ter interface amigável e interativa.

a) *Software* em tempo real, *Software* comercial, *Software* de computador pessoal.

b) *Software* em tempo real, *Software* em tempo real, *Software* científico.

c) Software científico, Software básico, Software de computador pessoal.

d) *Software* básico, *Software* de computador pessoal, *Software* de computador pessoal.

e) *Software* básico, *Software* em tempo real e *Software* científico.

Seção 1.2

Fundamentos dos processos de desenvolvimento de *software*: conceitos, métodos, ferramentas, procedimentos e principais atividades. Os produtos e o ciclo de vida. Projetos, atividades e estruturas analíticas. Modelos de referência e fatores de produção

Diálogo aberto

Seja bem-vindo de volta! Nesta segunda seção, retomaremos a situação abordada na seção anterior.

Com o crescimento da *XAX-Sooft*, a condução sistemática e correta de um processo de desenvolvimento deverá ganhar importância destacada na empresa. No cenário atual, há risco de perda de controle dos projetos por conta da desorganização na aplicação dos procedimentos vigentes. Nós sabemos que esses procedimentos eram minimamente adequados para a realidade da empresa no início das suas atividades, quando os projetos eram simples e demandavam pouco esforço da equipe. Uma metodologia formal de desenvolvimento deverá ser escolhida para fins de sistematização dos procedimentos de desenvolvimento de *software*. Será nossa missão dar bons motivos para que os envolvidos nos projetos a adotem e os dirigentes a comuniquem aos colaboradores.

Feito o levantamento dos procedimentos adotados atualmente na *XAX-Sooft*, é hora de você propor soluções para melhoria do processo de desenvolvimento praticado na empresa, utilizando os conhecimentos adquiridos nesta seção.

Esta nova parte do nosso material pretende oferecer soluções viáveis para situações em que o desenvolvimento do sistema passa por dificuldades e incertezas, bem destacadas em nosso estudo sobre a crise do *software*. Sem procedimentos formais, alguns erros comuns poderão ser cometidos mais do que uma vez durante o desenvolvimento, a necessidade de retrabalho estará sempre presente, as ações de prevenção de erros dificilmente serão implantadas e, mais importante, o cliente não estará satisfeito com o produto entregue.

Para que a proposta desta seção seja contemplada, você deverá dominar os conceitos de processo de desenvolvimento de *software* e conhecer seu ciclo de vida tradicional. Da mesma forma, na *XAX-Sooft* essa necessidade também se faz presente. Trataremos de explorar os

conceitos de processo, fases, atividades, recursos e tudo o que se insere no contexto das boas práticas de desenvolvimento.

Nosso desafio agora é estruturar as novas práticas, cuidando para que a equipe toda se sinta motivada a adotá-las e a cuidar da sua manutenção e evolução. Os desenvolvedores já se acostumaram à não formalidade dos seus meios e neles confiam, já que, bem ou mal, têm funcionado até o momento. Esse cenário impõe necessidade de planejamento, método e comunicação eficiente sobre a nova metodologia. O estudo do ciclo de vida de um *software* e de seu processo de desenvolvimento é ponto de partida para a inflexão que a empresa precisa. O que é um processo de *software*? Como implantá-lo de forma eficiente? Vamos adiante!

Não pode faltar

Conforme já mencionamos, o desenvolvimento de um *software* pode ser considerado como um processo e não um conjunto de ações isoladas. A previsibilidade e a economia de recursos propiciada por um processo bem estruturado conferem segurança ao desenvolvimento. Ao resumirmos e agruparmos os muitos conceitos que a literatura disponibiliza, teremos que, no âmbito da Engenharia de *Software*, **processo é a sequência de passos que visam a produção e manutenção de um *software* e que se inter-relacionam com recursos (humanos e materiais), com padrões, com entradas e saídas e com a própria estrutura da organização**.

Esse conceito pode ser estendido a outros ramos da atividade humana, como a fabricação de um bem de consumo. A Figura 1.1 mostra o processo de montagem de um automóvel.

Figura 1.1 - Processo de montagem de um automóvel

Fonte: <https://pixabay.com/pt/f%C3%A1brica-carro-motor-montagem-35104/>. Acesso em: 28 out. 2015.

O termo "projeto", comumente utilizado neste âmbito, precisa ser diferenciado de "processo". De acordo com o Guia do Conhecimento em Gerenciamento de Projetos (PMBOK, 2013, n. p.), projeto "é um conjunto de atividades temporárias, realizadas em grupo, destinadas a produzir um produto, serviço ou resultado únicos". Ensina Wazlawick (2013), que processo é o conjunto de regras que definem como um projeto deve ser executado. Um processo é adotado pela organização para que seja praticado e aperfeiçoado pelos seus colaboradores durante o desenvolvimento de um projeto.

Ainda de acordo com Wazlawick (2013), há vantagens em se definir o desenvolvimento de *software* como um processo. O autor apresenta três delas:

a) Redução no tempo de treinamento, já que funções e procedimentos bem definidos e documentados facilitam a inclusão de novo membro na equipe de trabalho;

b) Produção de artefatos mais uniformizados, já que a previsibilidade do processo ajuda a equipe a trabalhar de forma mais padronizada;

c) Transformação de experiências em valor, já que a sistemática utilização do procedimento poderá aperfeiçoá-lo com o tempo.

Os processos podem conter divisões em sua estrutura. Para iniciarmos efetivamente o estudo do processo de *software*, convém analisarmos algumas delas.

Fases: um conjunto de atividades afins e com objetivos bem definidos são realizados em uma fase do processo. O modelo cascata de desenvolvimento, por exemplo, apresenta fases bem definidas, quais sejam a fase dos requisitos, a fase do projeto, da programação e assim por diante (WAZLAWICK, 2013).

Atividades ou tarefas: comumente descritas com conceitos semelhantes, uma atividade ou uma tarefa constitui um projeto em pequena escala. Ela visa promover modificações nos artefatos do processo, que podem ser descritos como diagramas, documentos, programas e tudo o que puder ser desenvolvido no processo. As atividades devem possuir entradas, saídas, responsáveis, participantes e recursos bem definidos (WAZLAWICK, 2013).

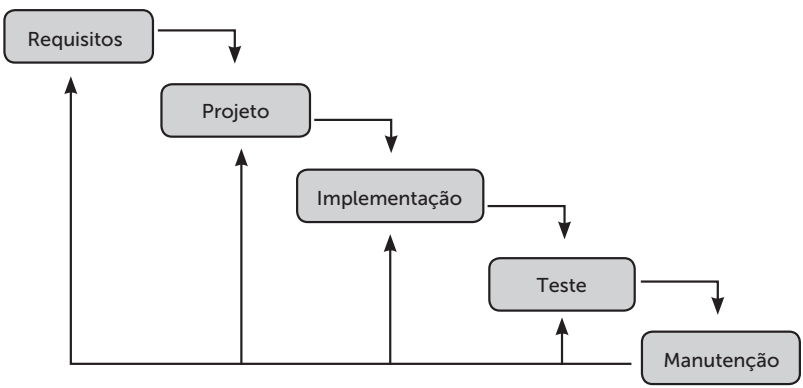
Em suas regras processuais, a organização pode determinar que seja adotado um documento que descreva a atividade. Por meio dele, a equipe tomará conhecimento da tarefa, seus responsáveis, objetivos, recursos a serem utilizados e tudo o que a caracteriza por completo.

Sabemos até o momento que um processo é um conjunto disciplinado e articulado de tarefas, que serve para sistematizar o desenvolvimento de *software*. Esse tipo de processo é genérico, sendo definido e aplicado pela organização, o que o torna, de certo modo, sua filosofia de trabalho.

Há, no entanto, certos modelos de processos ditos **prescritivos**, que contêm descrições de como as atividades são realizadas. O modelo Cascata, também conhecido como modelo tradicional, é o mais conhecido e ainda bastante utilizado para desenvolvimento de produtos de *software*. Ele descreve, por meio de etapas bem definidas, o ciclo que o *software* cumprirá durante o período compreendido entre sua concepção e sua descontinuidade.

O ciclo de vida natural de um *software,* de acordo com Rezende (2005), abrange as seguintes fases: concepção, construção, implantação, implementações, maturidade, declínio, manutenção e descontinuidade. Estas fases são mais comumente descritas como fase de requisitos, projeto, implementação, teste e manutenção. A Figura 1.2 mostra esquema geral das fases do modelo Cascata.

Figura 1.2 | Representação das fases do modelo Cascata



Fonte: elaborada pelo autor.

Note que uma fase do processo depende do resultado ou do produto gerado pela fase anterior. As setas de retroalimentação, dispostas no sentido contrário à cascata, indicam a possibilidade de retornos às fases anteriores, considerando nelas a ocorrência de falhas. Em outras palavras, o retrocesso a uma fase anterior serve, em tese, para sanar problemas que, se levados adiante, acarretariam mais prejuízo ao desenvolvimento.

Assimile

Um ciclo de desenvolvimento de sistemas mais detalhado é apresentado por Rezende (2005), como segue:

Estudo de viabilidade, análise de sistemas, projeto, implementação, geração de teste de aceite, garantia de qualidade, descrição de procedimentos, conversão de bases de dados e instalação.

Para que a proposta de ensino desta seção seja contemplada, descreveremos sinteticamente cada uma das fases que, nas aulas seguintes, serão mais bem detalhadas.

Requisitos

A fase de requisitos de *software* preocupa-se com a descoberta, análise, especificação e validação das propriedades que devem ser apresentadas para resolver tarefas relacionadas ao *software* que será desenvolvido. Requisitos são as condições necessárias para que um determinado evento aconteça. Tome como exemplo uma aula presencial de Engenharia de *Software*. Para que ela aconteça, é necessário professor, alunos, lousa, giz, carteiras. Todos esses itens formam o conjunto de requisitos da aula. No desenvolvimento de *software* acontece o mesmo. Fazem parte dos requisitos de um *software* suas funções, suas características, restrições e todas as demais condições para que ele exista e cumpra seu objetivo.

O projeto de um *software* fica vulnerável quando o levantamento dos requisitos é executado de forma não apropriada. Os requisitos expressam as necessidades e restrições colocadas num produto de *software* que contribuem para a solução de algum problema do mundo real.

Reflita

Caso uma falha seja cometida na fase de levantamento dos requisitos do produto, ela deverá se propagar nas fases seguintes de projeto e implementação. Assim, quanto antes a falha for corrigida, menos dispendioso seu reparo será. Schach (2008) assinala que 60% a 70% de todas as falhas detectadas em grandes projetos acontecem nas fases de levantamentos de requisitos, análise ou de projeto. Durante 203 inspeções do *software* da *Jet Propulsion Laboratory* para o programa interplanetário não tripulado da NASA, em média foram detectadas cerca de 1,9 falhas por página de um documento de especificações, 0,9 falha por página de um projeto, mas apenas 0,3 falha por página de código.

Na definição dos requisitos, o profissional envolvido (chamado de especialista em requisitos, engenheiro de *software* ou analista de requisitos) tem a oportunidade de aprimorar a alocação das funções do *software*, além de supri-lo com a especificação de requisitos, documento fundamental no relacionamento cliente/desenvolvedor.

Exemplificando

Um bom exemplo de como ações simples podem evitar grandes transtornos durante e depois do desenvolvimento de um *software* pode ser resumido como segue. Logo após concluir o levantamento dos requisitos junto ao cliente e sua posterior análise e especificação, uma certa empresa de desenvolvimento iniciava imediatamente a fase de projeto. A certeza de que tudo que haviam conversado com o cliente havia sido corretamente compreendido os autorizava a investir pouco tempo no projeto e, sem demora, começar a fase de programação. Ocorre que, com a sucessão de retrabalhos em seus projetos, decidiram rever o processo desde o início. O que poderia estar errado? Conforme será abordado em detalhes na próxima seção, o processo de requisitos inclui também a validação do que foi definido como funcionalidade do sistema, antes que a elaboração do projeto seja iniciada. Antes de qualquer nova ação, há que se validar os requisitos, o que significa conferi-los em reuniões com as equipes e, principalmente com o cliente. A partir do momento que a empresa passou a incluir a conferência dos requisitos antes de dar o próximo passo no projeto, a quantidade de retrabalho diminuiu, na mesma proporção em que a comunicação com o cliente melhorou.

Projeto

Uma vez levantados, analisados, especificados e validados os requisitos, o processo deverá nos levar ao desenho do produto. Se os requisitos nos mostram "o que" o sistema deverá fazer, o projeto deverá refletir "como" ele o fará. Por meio do projeto, os requisitos são refinados de modo a se tornarem aptos a serem transformados em programa. O trabalho principal de um projetista é o de decompor o produto em módulos, que podem ser conceituados como blocos de código que se comunicam com outros blocos por meio de interfaces.

Implementação

Nesta fase, o projeto é transformado em linguagem de programação para que, de fato, o produto passe a ser executável. Para que se possa avaliar se os requisitos foram corretamente traduzidos, a equipe pode

optar por construir protótipo do sistema, ou seja, uma versão com funcionalidades apenas de tela para proporcionar entendimento e validação das entradas e saídas do *software*. Como estratégia de implementação, a equipe poderá dividir o trabalho de forma que cada programador (ou um pequeno grupo deles) fique responsável por um módulo do sistema. Idealmente, a documentação gerada pela fase de projeto deve servir como principal embasamento para a codificação, o que não afasta a necessidade de novas consultas ao cliente e à equipe de projetistas.

Testes

Testar significa executar um programa com o objetivo de revelar a presença de defeitos. Caso esse objetivo não possa ser cumprido, considera-se o aumento da confiança sobre o programa. O processo de teste deve incluir seu planejamento, projeto de casos de teste, execução do programa com os casos de teste e análise de resultados. As técnicas Funcional e Estrutural são duas das mais utilizadas técnicas de se testar um *software*. A primeira baseia-se na especificação do *software* para derivar os requisitos de teste e a segunda baseia-se no conhecimento da estrutura interna (implementação) do programa.

Manutenção

Os esforços de desenvolvimento de um *software* resultam na entrega de um produto que satisfaça os requisitos do usuário. Espera-se, contudo, que o *software* sofra alterações e evolua. Uma vez em operação, defeitos são descobertos, ambientes operacionais mudam, e novos requisitos dos usuários vêm à tona. A manutenção é parte integrante do ciclo de vida do *software* e deve receber o mesmo grau de atenção que outras fases. A manutenção de *software* é definida como modificações em um produto de *software* após a entrega ao cliente a fim de corrigir falhas, melhorar o desempenho ou adaptar o produto a um ambiente diferente daquele em que o sistema foi construído.

Pesquise mais

Você pode encontrar mais sobre requisitos de *software* no texto contido em: <http://www.bfpug.com.br/islig-rio/Downloads/Ger%C3%AAncia%20de%20Requisitos-o%20Principal%20Problema%20dos%20Projetos%20de%20SW.pdf>. Acesso em: 12 nov. 2015. Ele destaca a importância do processo de requisito na qualidade do produto de *software*.

Faça você mesmo

Na seção 1.1 foi solicitado que você descrevesse seu primeiro passo caso recebesse a missão de construir o sistema acadêmico da faculdade de seu amigo. No contexto em que se apresentava o exercício, você tinha pouco conhecimento de processo de *software* e do ciclo de vida de um produto.

O desafio agora é que você refaça aquele exercício, colocando em prática o que você já conhece da teoria necessária para iniciar o desenvolvimento de um sistema. Mãos à obra!

Sem medo de errar

Com o levantamento da atual situação em mãos, a *XAX-Sooft* precisa agora definir procedimento padronizado e universal para construção de seus produtos. A fase do desenvolvimento artesanal e baseado apenas na experiência acumulada de sua equipe deve ficar para trás. Os gestores definiram que qualquer novo procedimento deve ser implantado gradualmente e a necessidade de evolução do processo deve ser considerada, a fim de terem, depois de um ano, um processo sedimentado, amplamente praticado e em constante evolução.

Como podemos, então, iniciar a implantação de um procedimento uniforme?

A primeira providência é definir que as novas práticas devem ser agrupadas e registradas em documento próprio. Este documento conterá as seguintes grandes seções:

1. **Introdução e objetivo**: aqui será descrito o objetivo do documento, que é justamente a definição do procedimento padrão da empresa. Deverá ficar evidente que o processo prescritivo a ser adotado é o modelo tradicional de desenvolvimento.

2. **Definição dos papéis**: nesta seção cada função da equipe será descrita. As características pessoais de cada membro da equipe devem ser levadas em conta no momento de definir áreas de atuação. Quem já tinha facilidade em se relacionar com o cliente deverá ser destacado para a fase de requisitos; o programador experiente deverá continuar a ser líder na fase de implementação e assim por diante. Serão considerados como funções da equipe: engenheiro de requisitos, projetista e desenvolvedor. Este último deverá também ser destacado nas atividades de teste e manutenção.

3. **Definição detalhada do processo de desenvolvimento**: por fim, nesta seção serão definidos os caminhos para a produção de *software* de qualidade. O processo deverá prever que:

3.1. Definida a viabilidade de se assumir o trabalho, a criação do produto deverá ser iniciada pela fase de levantamento e tratamento dos requisitos. Um profissional previamente destacado deverá visitar o cliente e, por meio de reuniões presenciais, coletar todas as suas necessidades em relação ao programa. Os requisitos então serão consolidados em documento e revisados pela equipe.

3.2. Levantados os requisitos, um projeto modular do sistema deverá ser criado. Os grandes módulos e suas interfaces deverão ser definidas e um desenho do produto deverá ser gerado.

3.3. Dependendo da complexidade e da extensão do produto, os módulos definidos na fase de projeto são distribuídos para um ou mais desenvolvedores para tradução em linguagem de programação.

3.4. Terminada a implementação do módulo, os desenvolvedores envolvidos deverão delegar aos colegas a tarefa de testar suas funções, com base nos requisitos revisados. Terminados esses testes, o sistema é integrado e o teste geral é aplicado. Havendo necessidade, os desenvolvedores darão suporte à manutenção ao sistema.

Em todas as fases do processo, prazo e recursos necessários deverão ser explicitados.

Atenção

Sintetizar o novo processo de desenvolvimento em documento é a melhor providência a se tomar. Ele servirá, inclusive, de referência para novos colaboradores e como fonte de consulta em caso de dúvida sobre os procedimentos.

Lembre-se

A especificação dos requisitos se dá pela criação de documento que os contém e define. Um exemplo de documento de especificação de requisitos pode ser encontrado em: <http://www.cin.ufpe.br/~rpgl/smartclinic/documentoRequisitos.doc>. Acesso em: 12 nov. 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.	
"Adquirindo a *You-Soft*"	
1. Competência geral	Conhecer o ciclo de vida de um produto de *software* e o processo que o embasa.
2. Objetivos de aprendizagem	Transferência dos conceitos aprendidos para situação-problema semelhante à apresentada no início da seção
3. Conteúdos relacionados	Processo de desenvolvimento de *software* e seu ciclo de vida.
4. Descrição da SP	Como parte do seu projeto de expansão, a *XAX-Sooft* acaba de adquirir a *You-Soft*, startup que desenvolve exclusivamente um Sistema de Gerenciamento de Conteúdo (do inglês *Content Management System – CMS*). A *XAX-Sooft* pretende manter a *You-Soft* como empresa autônoma, preservando seus funcionários e os processos de aperfeiçoamento, implantação e manutenção de seu produto. No entanto, em rápido levantamento, os gestores da *XAX-Sooft* notaram que as funções dos envolvidos na operação da ferramenta nos clientes da *You-Soft* não estavam sendo orientadas da melhor maneira. Ao assumir que a ferramenta dispensaria a contratação de profissionais de *internet* para operá-la, os clientes que adquiriam o CMS da *You-Soft* viam-se frustrados com o baixo retorno que o produto lhes proporcionava. Com base nas características próprias de um CMS, como você ajudaria os clientes da *You-Soft* a definirem os papéis dos profissionais envolvidos com a operação da ferramenta?
5. Resolução da SP	Por falta de orientação dos seus desenvolvedores, os clientes que adquiriram o CMS da *You-Soft* não estruturaram equipe para operacionalização da ferramenta. Trata-se de caso típico de falha no processo de implementação de *software* no cliente, provavelmente reflexo de falha de definição do processo de desenvolvimento. A despeito da necessidade de reestruturação do processo da *XAX-Sooft*, a consequência dessa situação no cliente pode ser sanada pela contratação de profissionais que cuidarão da evolução da ferramenta, da criação de conteúdos para a internet e da aprovação desses conteúdos antes da sua divulgação. Sem essa estrutura, a ferramenta deixará de atingir seus objetivos, já que não há CMS que não requeira responsável pela criação e fornecimento de conteúdo.

Lembre-se

As ferramentas de CMS têm sido amplamente utilizadas por empresas que desejam controlar com autonomia os conteúdos que publicam na internet. Saiba mais em: <http://computerworld.com.br/gestao/2008/03/05/superguia-infoworld-avalia-sistemas-de-gestao-de-conteudo>. Acesso em: 12 nov. 2015.

Faça você mesmo

Como forma de aprofundar seu conhecimento sobre o modelo Cascata, pesquise a existência de ao menos uma variação dele e resuma-o em, no máximo, 15 linhas.

Faça valer a pena

1. O desenvolvimento de um *software* feito num contexto de processo organizado apresenta vantagens em relação ao desenvolvimento informal. Em relação a esse tema, analise as afirmações que seguem:

I - Redução no tempo de assimilação da metodologia, já que o processo bem documentado facilita o trabalho de quem ainda não o conhece.

II - Transformação das experiências vividas em valor, já que a sistemática utilização do procedimento poderá aperfeiçoá-lo com o tempo.

III - Descoberta de maus profissionais da organização, já que o processo padronizado ajuda a destacar programadores sem afinidade com linguagens de programação.

IV - Produção de artefatos mais uniformizados, já que a previsibilidade do processo ajuda a equipe a trabalhar de forma mais padronizada.

V - Possibilidade de se aperfeiçoar o processo, já que ele deve estar em constante evolução.

É correto o que se afirma apenas em:

a) I, II e IV.

b) I, II e III.

c) I, II, IV e V.

d) II, III e IV.

e) II, III, IV e V.

2. Analise as afirmações sobre processos de *software*:

I - Processo é um conjunto de atividades e resultados associados que geram um produto de *software*.

II - São componentes de um processo de *software* as entradas e saídas e os responsáveis pelas tarefas.

III - Um modelo de processo prescritivo não apresenta descrição formal das atividades, já que se baseiam na prescrição dos gestores do negócio para funcionarem.

IV - Uma fase corresponde a um período no qual determinadas atividades com objetivos bem específicos são realizadas.

V - Na definição de uma atividade, não há necessidade prévia de se definir responsáveis e participantes, já que eles serão naturalmente escolhidos pela equipe durante seu andamento.

É verdadeiro o que se afirma apenas em:

a) I, II e III.

b) I, II e IV.

c) I, III e IV.

d) III, IV e V.

e) III e IV.

3. Em relação ao ciclo de vida de um *software*, assinale a afirmação verdadeira:

a) Trata-se de um modelo de processo que prevê as chances de um sistema ter sobrevida em períodos de crise durante sua elaboração.

b) Trata-se de um processo que se encerra quando o produto é entregue ao cliente.

c) O ciclo de vida tradicional de um *software* inclui fase em que os requisitos levantados são especificados e validados.

d) No ciclo de vida tradicional de um *software* não se pode retornar às fases já cumpridas, mesmo quando falhas são encontradas em fases posteriores.

e) O ciclo de vida tradicional não apresenta a linearidade como característica, já que sua representação nos mostra processo em forma de cascata.

Seção 1.3

Modelos e etapas do processo de *software*: características,requisitos, projeto de sistema, desenvolvimento, integração, instalação e manutenção

Diálogo aberto

Para iniciarmos a terceira seção desta unidade, vale breve retomada de como a situação da *XAX-Sooft* tem evoluído. Ao receber demanda de grande cliente especializado em venda de games, a empresa viu-se na necessidade de deixar o modo artesanal de produção de *software* e buscar aplicação de metodologia formal. Como primeiro passo, fez levantamento da situação atual e, ato contínuo, esboçou processo uniforme de desenvolvimento.

Então, na condição de ator do processo, chegou o momento de você iniciar trabalho relacionado aos requisitos e de esboçar o projeto do *software*, com base nos estudos desenvolvidos nesta seção e com base no processo previamente definido. Na resolução dessa atividade, esperamos que você aplique as práticas adequadas para levantamento dos requisitos e para sua classificação. Será necessário também esboçar os módulos que comporão o sistema. Ao final, você deverá gerar documento de especificação de requisitos devidamente revisado e os módulos do sistema.

Você pode observar que sem a competente descoberta, análise, documentação e conferência dos requisitos, a perfeita adequação do produto ao seu propósito ficará comprometida. Da mesma forma, sem a criação de desenho correto da solução, a implementação será feita com base em dados incorretos.

Pois é, esses desafios nos remetem a outro de igual importância: você deverá dominar os conceitos de requisitos e de projeto de *software*, que formam justamente os objetivos de aprendizagem desta seção. O entendimento teórico e a habilidade prática para utilizar as diversas formas de levantamento dos requisitos farão toda a diferença no correto cumprimento de parte da sua tarefa. Da mesma

forma, a capacidade de sintetizar os requisitos em uma solução de implementação viável garantirá seu sucesso.

Não podemos nos esquecer que o estudo dos conteúdos desta primeira unidade tem construído, passo a passo, sua capacidade de avaliar e adequar a metodologia tradicional de desenvolvimento às situações que certamente encontrará em seu futuro profissional. Seu crescimento nesse tema lhe possibilitará, inclusive, comparar essa metodologia com outras mais atuais, além de capacitá-lo no entendimento das normas de qualidade e dos processos de teste de *software*.

O desafio está novamente lançado. Seja bem-vindo e vamos adiante!

Não pode faltar

É importante você observar que a abordagem dos requisitos e a elaboração do projeto constituem, normalmente, as duas primeiras etapas dos modelos de *software* tradicionais. Não por acaso, são tidas também como as mais críticas no processo. Roger Pressman, em sua clássica obra "Engenharia de *Software*", destaca que não importa o quão bem codificado seja, mas um programa mal analisado e mal projetado desapontará o usuário e trará aborrecimentos ao desenvolvedor (PRESSMAN, 1995). Nas próximas páginas caminharemos juntos pelos detalhes das fases de requisitos e de projeto, de modo a prepará-lo para aplicar seus conhecimentos na solução dos problemas da *XAX-Sooft*.

Requisitos de *software*

A área de requisitos de *software* preocupa-se com o levantamento, análise, especificação e validação das propriedades que devem ser apresentadas para resolver tarefas relacionadas ao *software* em desenvolvimento. Requisitos são a expressão mais detalhada sobre aquilo de que o usuário ou cliente precisa em termos de um produto de *software* (WAZLAWICK, 2013).

Além das funcionalidades, eles relacionam-se também aos objetivos, restrições e padrões do sistema. Em outras palavras, os requisitos de *software* expressam as necessidades e restrições colocadas num produto de *software* que contribuem para a solução de algum problema do mundo real. São subetapas da fase de requisitos:

1. Levantamento de requisitos

Schach (2008) aponta ações que devem nortear o trabalho de levantamento de requisitos. A primeira é determinar o que o cliente precisa, em vez do que o cliente quer. No entanto, é muito comum que os clientes não saibam do que precisam ou que tenham dificuldade em expressá-lo.

É necessário também que o grupo destacado conheça o campo de aplicação, ou seja, a área geral em que o *software* será aplicado. Incluem exemplos de campos de aplicação o setor bancário, o comércio varejista e o setor acadêmico, entre tantos outros.

Por fim, é indispensável que o engenheiro de requisitos conheça as regras (ou modelo) de negócios do cliente. Trata-se da descrição dos processos que compõem o negócio da organização.

Técnicas de levantamento de requisitos

O levantamento de requisitos é atividade essencialmente humana, que requer habilidade em trabalhar com especialistas humanos e com o conhecimento tácito, que é trivial para quem conhece a informação, mas não é trivial para quem procura obtê-la.

O documento "Guide to the Software Engineering Body of Knowledge" (IEEE, 2004) aborda algumas técnicas que facilitam esse trabalho.

Entrevista

Antes da sua aplicação, a entrevista deve ser planejada. Seus objetivos devem ser fixados, seu local e roteiro definidos e os entrevistados criteriosamente escolhidos. A interação entre entrevistado (especialista do conhecimento) e entrevistador (engenheiro de requisitos) deve buscar revelar conceitos, objetos e a organização do domínio do problema, além de buscar soluções ou projeções de soluções que comporão o domínio da solução (SCHACH, 2008).

Assimile

As entrevistas mais usuais são as tutoriais, informais e estruturadas. Nas entrevistas tutoriais, o entrevistado fica no comando, praticamente lecionando sobre um determinado assunto. Nas entrevistas informais ou não estruturadas, o entrevistador age espontaneamente, perguntando ao entrevistado, sem obedecer a nenhuma organização. Já as entrevistas estruturadas são preparadas pelo entrevistador, que define previamente o andamento do procedimento de aquisição de conhecimento.

Aplicação de questionário

O questionário geralmente é aplicado em formulário distribuído para os futuros usuários do sistema. Seu elaborador deve utilizar questões diretas e objetivas, dispostas preferencialmente na mesma ordem para todos os participantes e que consigam extrair deles respostas sobre o procedimento atual adotado (SCHACH, 2008).

Pesquise mais

Para melhor compreensão da realidade, a utilização de casos de uso tem sido frequente como meio de representar a interação entre o *software* e o ambiente no qual ele opera ou irá operar. Serve também para explicitar a interação entre o produto de *software* e seus usuários. Saiba mais em: <http://sistemasecia.freehostia.com/component/jccmultilanguagecontent/article/34-engenhariasoft/73-reqs-casosuso-desenv.html>. Acesso em: 12 nov. 2015.

Análise de documentos, observações pessoais e reuniões estruturadas são outras três técnicas utilizadas na fase de levantamento de requisitos.

2. Análise de requisitos

Concluída a fase de levantamento, tem início a análise de requisitos. Aqui os requisitos serão analisados e classificados e, como resultado, serão divididos principalmente em requisitos funcionais e não funcionais. Os primeiros descrevem as funções que o *software* irá executar. Por exemplo, formatar algum texto ou imprimir um relatório de vendas. Os requisitos funcionais definem a funcionalidade que o *software* deve prover com o objetivo de capacitar os usuários a realizar suas tarefas. Eles descrevem o comportamento planejado do sistema, podendo ser expressos como os serviços, tarefas ou as funções que o sistema irá executar.

Requisitos não funcionais são aqueles que restringem a solução de um problema. Não se referem às funções específicas do sistema, mas sim a propriedades, tais como tempo de resposta, requisitos de armazenamento, restrições de entrada e saída, memória, entre outras.

3. Especificação dos requisitos de *software*

Refere-se a produção de um documento contendo os requisitos levantados e analisados, que podem ser sistematicamente revistos,

avaliados e aprovados. Ele estabelece um contrato entre cliente e desenvolvedor. Geralmente escrito em linguagem natural, forma base realista para estimativas de custos, riscos e cronograma.

Uma boa especificação de requisitos de *software* pode propiciar diversos benefícios aos clientes e demais envolvidos no projeto, a saber:

I) Estabelece a base para a concordância entre clientes e fornecedores, naquilo que o *software* deve produzir;

II) Reduz o esforço para o desenvolvimento. Uma revisão cuidadosa dos requisitos na ERS pode trazer à tona omissões e falhas em fases iniciais no ciclo de desenvolvimento quando esses problemas são mais fáceis de corrigir;

III) Fornece base para estimativa de custos e agendas. A descrição do produto a ser desenvolvido é uma base realista para a estimativa dos custos do projeto e pode ser usada como referência de preço do produto; e

IV) Fornece uma linha de base para validação e verificação. As organizações podem desenvolver seus planos de validação e verificação de forma muito mais produtiva a partir de uma boa ERS (IEEE, 2004).

Assimile

A IEEE padronizou o formato do ERS por meio da norma IEEE 830:1998, substituída posteriormente pela ISO/IEC/IEEE 29148:2011.

Validação dos requisitos

Como última etapa da fase do processo, a validação deve incidir sobre o documento de especificação.

Validação: "Aquilo que fiz é o que deveria ser feito? Aquilo que fiz corresponde aos requisitos?"

A validação serve para assegurar que o engenheiro compreendeu os requisitos e se estes são compreensíveis, consistentes e completos. No processo de validação a participação do cliente é fundamental. A revisão é feita por profissionais designados para assegurar que não há no documento falta de clareza, sentido dúbio e desvio dos padrões.

Uma maneira eficaz de validar os requisitos é pela prototipação. Trata-se da criação de versão simplificada de determinadas funções do sistema, indicada para capturar a real compreensão do engenheiro em relação aos requisitos levantados. O comportamento de uma interface de usuário também pode ser mais bem compreendida através de um protótipo (IEEE, 2004).

Projeto de *software*

O projeto é o primeiro passo da fase de desenvolvimento de qualquer produto ou sistema de engenharia. Sua meta é produzir modelo ou representação do produto a ser construído. Para tanto, o projetista deve lançar mão de sua experiência, intuição e metodologias consagradas. Modelos ou representações podem ser analisados quanto a sua qualidade e são base para as etapas de codificação, teste, validação e manutenção (PRESSMAN, 1995).

Projeto é o processo pelo qual os requisitos são traduzidos numa representação do *software*.

A avaliação da qualidade de um projeto implica que ele deve exibir uma organização hierárquica do *software*, deve ser modular e que deve haver distinção entre dados e procedimentos.

O nível de detalhamento deve ser suficiente para permitir sua realização física e, de acordo com o modelo de processo tradicional, o projeto se inicia depois de levantados, analisados e especificados os requisitos.

Aspectos fundamentais de projeto

Pressman (1995) destaca os aspectos fundamentais de um projeto de *software*:

1. Abstração: solução modular leva a vários níveis de abstração. Abstrair é concentrar-se em certos aspectos relevantes ao ambiente do problema. Cada passo da Engenharia de *Software* diminui o nível de abstração em direção à solução do problema. O nível mais baixo de abstração é o código-fonte. A abstração procedimental refere-se à sequência de instruções com funções específicas.

Exemplificando

Como exemplo, têm-se dois níveis de abstração procedimental para programa de esboço de CAD:

Abstração 1 - O *software* terá uma interface gráfica que possibilitará acesso a função de desenhos de linhas retas e curvas. Os desenhos serão armazenados numa pasta de desenhos.

Abstração 2 - Tarefas do *software* CAD:

início tarefa

interação com o usuário

criação de desenhos

gerenciamento de arquivos de desenho

fim tarefa

2. Abstração de dados: coleção de dados que descreve um objeto. Exemplo: contracheque contém o nome do beneficiado, quantia bruta, imposto de renda, contribuição sindical (conjunto de dados). Assim, referencia-se o conjunto de dados pelo nome da abstração (contracheque).

3. Modularidade: divisão do *software* em componentes chamados módulos. Permite a administração intelectual do *software*, já que um grande *software* monolítico (composto por apenas um módulo) é praticamente incompreensível. O desafio do projetista, no entanto, é dimensionar a quantidade de módulos a serem construídos. Quantitativamente, há que se comparar o esforço em se construir interface para um módulo comparado ao benefício em poder contar com ele.

Qualitativamente, a noção de qualidade passa pelos conceitos de coesão e acoplamento. De forma simplificada, coesão é o grau de interação dentro de um módulo e o acoplamento é o grau de interação entre dois módulos.

4. Procedimento de *software*: focaliza os detalhes de processamento de cada módulo da hierarquia. O procedimento oferece especificação precisa do processamento do módulo.

Exemplificando

A classificação dos requisitos não se resume apenas na separação entre funcionais e não funcionais. Requisitos voláteis, estáveis, de usuário, de sistema e inversos são outros tipos de classificação. Imagine que você tenha recebido a incumbência de classificar requisitos apresentados da seguinte maneira:

1. O sistema deve ser capaz de cadastrar um cliente;

2.O sistema não emite nota fiscal;

3. O cálculo da taxa de juros varia de acordo com a lei vigente.

Tendo como base os tipos estudados, classificamos o primeiro como um requisito funcional puro. O segundo se trata de requisito inverso, justamente aquele que não deve ocorrer no produto. Por último, o cálculo da taxa de juros não deixa de ser um requisito funcional, mas que também deve ser subclassificado como um requisito volátil, pois varia conforme evento externo.

Faça você mesmo

Você faz parte da equipe de engenheiros que cuida dos requisitos de um sistema acadêmico e foi designado para classificá-los, logo após o levantamento ter sido feito. Com base nos seus conhecimentos sobre o tema, classifique os requisitos que seguem:

1) Apenas o diretor da unidade terá acesso ao módulo de concessão de descontos na mensalidade;

2) O sistema poderá ser executado também pela internet, via navegador web;

3) Não haverá exclusão de aluno. Ao deixar a instituição, ele receberá status de formado, desistente ou trancado;

4) O sistema atual deverá contar com interface para sistema de biblioteca;

5) O usuário autorizado poderá emitir relatório de melhores alunos por período.

Sem medo de errar

Na condição de dirigente da *XAX-Sooft*, você sabe que a atenção empreendida nas fases de requisitos e projeto será fundamental para a continuidade do trabalho. O cliente espera que tudo sobre o seu programa de manutenção de clientes seja descoberto e documentado nesta etapa e frustrá-lo não seria boa ideia.

O processo se inicia pela tarefa de descoberta dos requisitos que, uma vez terminada, possibilitará que você classifique os requisitos em funcionais e não funcionais. Depois disso, você deverá criar documento de especificação de requisitos, que deverá ser devidamente revisado.

Tal documento é, afinal, um dos objetivos finais dessa tarefa. Findada essa fase e, com base nos requisitos funcionais, você deverá descrever os principais módulos do sistema.

O documento de especificação de requisitos poderá ter o seguinte conteúdo:

1. Breve descrição do sistema: o sistema a ser desenvolvido deverá

possibilitar a manutenção dos clientes da empresa, viabilizando o agrupamento por área de interesse nos games por ela comercializados, o que deverá possibilitar contatos mais assertivos e direcionamentos seguros de campanhas.

2. Requisitos funcionais: (i) em relação aos novos clientes, o sistema deverá permitir o cadastramento, alteração de dados, exclusão e consulta aos seus detalhes. (ii) O sistema deverá emitir relatórios das preferências dos clientes por tipo de games. (iii) O sistema deverá enviar mensagens de e-mails à sua base de clientes com lançamentos e atualizações de games.

3. Requisitos não funcionais: (i) o sistema deverá fazer uso da base de clientes já mantida pelo sistema de vendas. (ii) O sistema deverá ser desenvolvido para a plataforma web.

4. Considerações gerais de projeto: (i) fica decidido que, pelo perfil do cliente e do produto a ser gerado, serão utilizadas as técnicas de reuniões e de aplicação de questionários para a descoberta dos requisitos. (ii) O cliente irá indicar seus representantes para atuarem como fontes de informações.

Para efeito de esboço do projeto do *software*, deverão ser previstos os módulos de manutenção do cliente, emissão de relatório de preferências de clientes por tipo de games e módulo de envio de mensagens via e-mail.

Exemplificando

Você poderá incluir mais itens em sua especificação, tais como a descrição do modelo de negócio da empresa, requisitos de *hardware* do novo sistema, *layouts* de tela e descrição das funções em linguagem natural. Pode ser uma boa fonte de consulta o documento disponível em: <http://www.fazenda.sp.gov.br/sat/RequisitosSATv1.pdf>. Acesso em: 12 nov. 2015.

Atenção

O documento de especificação de requisitos pode se tornar base para contrato entre o desenvolvedor e o cliente, o que reforça a necessidade de cuidado em sua elaboração.

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.	
"Retomando um projeto interrompido"	
1. Competência geral	Conhecer as principais metodologias de desenvolvimento de *software*, normas de qualidade e processos de teste de *software*.
2. Objetivos de aprendizagem	Transferência dos conceitos aprendidos para situação semelhante à apresentada no início da seção.
3. Conteúdos relacionados	Fases de requisitos e de projeto de *software*. Documento de especificação de *software*.
4. Descrição da SP	Conforme apresentado na seção anterior, a *XAX-Sooft* acaba de adquirir a *You-Soft*, empresa que desenvolve ferramenta de CMS. Terminado o ajuste nas atividades dos operadores da ferramenta em determinado cliente, os gestores da *XAX-Sooft* passaram a fazer levantamento dos projetos da *You-Soft* que não tiveram êxito e, por isso, foram interrompidos. Concluíram em um desses projetos que o fracasso havia se dado por falha no processo de descoberta de requisitos, motivada pela distância geográfica da sede do cliente, da incerteza do mesmo sobre os benefícios que a ferramenta poderia lhe trazer e pela má condução das poucas reuniões que tiveram com o cliente. Sua missão aqui é sugerir meios de sanar esses problemas, recolocar o projeto em andamento e reconquistar a credibilidade do cliente.
5. Resolução da SP	Uma vez identificadas as causas, a solução dos problemas passa pelo ajuste pontual dos itens que levaram à interrupção do projeto. A primeira providência deve ser a de trazer o cliente novamente para o trabalho, com o agendamento de ao menos uma visita presencial e de conferências regulares via *Skype*. A distância geográfica entre cliente e fornecedor não deve mais ser obstáculo para bons contatos entre ambos. É comum também que o cliente, passada a euforia da compra, arrume motivos para questionar a validade de tê-la feito. Você, na condição de gestor da *XAX-Sooft*, deverá demonstrar que a ferramenta é necessária e que trará vantagens em relação ao processo de atualização de conteúdo que o cliente atualmente adota. Por fim, a coleta dos requisitos deverá ser retomada, com o uso de questionários enviados e respondidos por e-mail e entrevistas com usuário-chave via *Skype*.

Lembre-se

"De forma ideal, o projeto do produto deve ser flexível, significando que futuros aperfeiçoamentos (manutenção pós-entrega) poderão ser feitos acrescentando-se novas classes ou substituindo-se classes existentes, sem afetar o projeto como um todo" (SCHACH, 2008).

Faça você mesmo

Faça breve levantamento sobre as causas mais frequentes que levam os projetos de *software* a fracassarem, no Brasil ou no mundo.

Faça valer a pena

1. Em relação à especificação de requisitos de software, analise as afirmações que seguem:

I - Seu formato é livre, embora haja padronização sugerida pela IEEE.

II - Pode constituir base para contrato entre cliente e o fornecedor do *software*.

III - É de elaboração opcional, já que serve de base apenas para a fase de testes.

IV - Pode servir de base para elaboração de cronograma do projeto.

É verdadeiro o que se afirma em:

a) I, II e III.

b) I, II e IV.

c) II e III apenas.

d) I e III apenas.

e) II, III e IV.

2. Sobre projeto de *software*, analise as afirmações que seguem:

I - A representação de *software* gerada na fase de projeto pode ser um algoritmo escrito em pseudolinguagem.

II - A abstração procedimental refere-se à sequência de instruções com funções específicas.

III - A modularidade é um conceito que prevê diretamente a divisão de trabalho entre membros da equipe.

IV - O procedimento de *software* é um dos aspectos fundamentais do projeto e prevê a descrição detalhada de cada módulo.

V - De acordo com o processo tradicional de desenvolvimento, a fase de projeto vem após a fase de requisitos.

É verdadeiro o que se afirma apenas em:

a) I e II.

b) I, II, IV.

c) I, II, IV e V.

d) II e III.

e) III e IV.

3. Assinale a alternativa que contém expressões que completam corretamente as lacunas nas frases abaixo.

I) Entende-se que o _____ seja a expressão do que o produto deve fazer.

II) O _____ expressa como o *software* poderá fazê-lo.

III) Um requisito pode expressar também uma _____ do sistema a ser desenvolvido.

a) projeto, requisito, inversão.

b) projeto, atributo, inversão.

c) projeto, código, restrição.

d) requisito, projeto, indefinição.

e) requisito, projeto, restrição.

Seção 1.4

Modelos dos processos de *software*: aplicabilidade e evolução

Diálogo aberto

Seja bem-vindo à quarta aula desta primeira unidade do livro didático de Engenharia de *Software*!

Desde a primeira seção, sua ajuda tem sido decisiva na reestruturação do processo da *XAX-Sooft*. Por conta de sua boa intervenção, a empresa passou de um cenário de desenvolvimento indisciplinado e artesanal para um ambiente de procedimentos estruturados, papéis conhecidos e resultados mais previsíveis. Você sabe que as coisas ainda podem ser melhoradas, mas o primeiro passo foi dado. Afinal, o primeiro grande cliente já foi conquistado e, dependendo do sucesso dessa empreitada, outros se seguirão a ele.

O desenvolvimento do *software* para controle de clientes já passou pelas etapas de requisitos e projeto. Até agora, tudo vai bem. No entanto, o projeto ainda precisa ser implementado, passar pela integração dos módulos e, por fim, ser implantado.

Entretanto, a prova de fogo começa agora: sua missão é planejar e gerenciar a execução das fases finais do processo, cuidando para que um produto de qualidade seja entregue ao cliente. Entregue para o seu cliente um relatório de projeto de *software* para cadastro de clientes por interesse, contendo todas as fases desenvolvidas até o momento.

Para executar essas ações com competência, você deve conhecer elementos básicos de implementação de *software,* tais como: integração, as melhores práticas de implantação e como se faz a manutenção de *software*.

A tarefa deve ser realizada pela elaboração de um documento que contenha aspectos principais da implementação, incluindo a definição dos programadores, escalonamento das atividades, controle de versão do sistema, treinamento dos usuários, implantação e linhas gerais do processo de manutenção. Vale a sugestão de que ao relatório criado na seção anterior sejam acrescidos os itens ora desenvolvidos, de modo a construir um único documento.

Mãos à obra e bom trabalho!

Não pode faltar

Conforme temos estudado, o modelo tradicional de desenvolvimento de sistemas caracteriza-se pelo aproveitamento do resultado da etapa anterior na construção da etapa seguinte. Por exemplo, é por meio do que foi entregue na etapa de projeto que o pessoal da implementação construirá o programa. Esta seção trará a você conteúdo teórico necessário para se sair muito bem em nosso desafio de estruturar a etapa de implementação do nosso programa de controle de cliente.

Implementação de *software*

Implementação é o processo de converter o projeto detalhado em código. Com as etapas anteriores de requisitos e projeto bem-sucedidas, a implementação tende a ser relativamente bem compreendida. Alguns pontos do processo devem ser alvo de atenção (SCHACH, 2008):

a) Escolha da linguagem de programação: esta decisão passa pela vontade expressa do cliente, pela experiência da equipe em determinada linguagem e pela necessidade pontual do projeto. Em todos os casos, a escolha deverá ter sido registrada previamente pela equipe e pelo cliente. Caso o cliente não faça menção da linguagem a ser usada e o projeto não demande nada específico, a escolha certamente recairá sobre a linguagem que for de amplo conhecimento da equipe;

b) Práticas de programação adequadas: o estilo usado no desenvolvimento do *software* deve obedecer, preferencialmente, a padrões consagrados de codificação.

Pesquise mais

Um padrão de codificação pode ser entendido como um conjunto de regras que disciplinam a criação do código. Um padrão pode conter normas para criação de nomes de arquivos, nomes de classes, endentação e quebras de linha, entre outras.

Você poderá encontrar parte da padronização para a linguagem *Delphi* em: <http://www.devmedia.com.br/padroes-de-codificacao/16529>. Acesso em: 2 nov. 2015.

Os nomes das variáveis e demais componentes do programa devem ser coerentes com sua real função. Recomenda-se que sejam dados nomes universalmente significativos aos identificadores, ou seja, que façam sentido tanto para o programador que trabalha naquele projeto como para os que eventualmente lhe darão manutenção futura.

Também estão inclusas nessas boas práticas a documentação do código. Ela deve ser objetiva, clara e transmitir dados elementares sobre o programa, que incluem descrição da função principal (ou do módulo), programadores envolvidos, *interfaces* externas e as últimas alterações feitas no código (SCHACH, 2008).

Integração de *software*

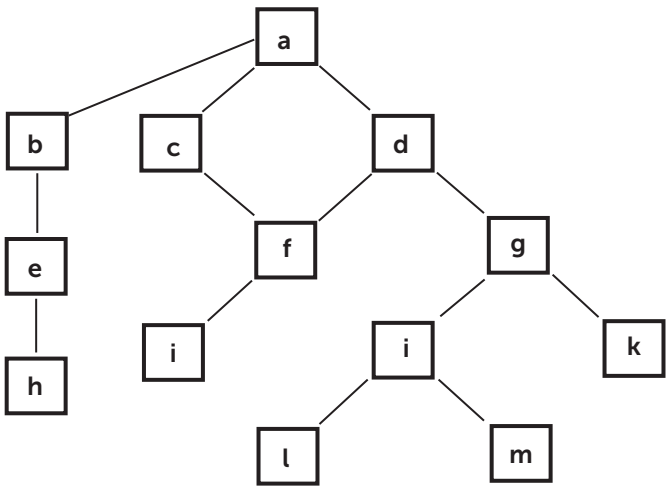
Integrar o *software* significa combinar em um todo os módulos construídos durante a implementação. A figura 1.3 mostra uma típica configuração de comunicação entre módulos de um programa. Uma possibilidade de efetivar a integração do produto é codificar e testar cada um dos artefatos (ou módulo) de código separadamente, unir e testar o produto como um todo.

Também estão inclusas nessas boas práticas a documentação do código. Ela deve ser objetiva, clara e transmitir dados elementares sobre o programa, que incluem descrição da função principal (ou do módulo), programadores envolvidos, *interfaces* externas e as últimas alterações feitas no código (SCHACH, 2008).

Integração de *software*

Integrar o *software* significa combinar em um todo os módulos construídos durante a implementação. A figura 1.3 mostra uma típica configuração de comunicação entre módulos de um programa. Uma possibilidade de efetivar a integração do produto é codificar e testar cada um dos artefatos (ou módulo) de código separadamente, unir e testar o produto como um todo.

Figura 1.3 | Diagrama de interconexões



Fonte: Schach (2008, P. 475)

No entanto, essa abordagem pode apresentar dificuldades em sua execução. O módulo *a*, por exemplo, não pode ser testado isoladamente, pois possui relacionamentos com os módulos *b*, *c* e *d*. Outro problema associado a essa forma de integração é a falta de isolamento de falhas. Ou seja, caso o produto falhe, a falha poderia estar em qualquer um dos treze módulos ou em uma das *treze interfaces* (SCHACH, 2008).

A solução passa pela adoção de técnicas estruturadas de integração, incluindo integração *top-down*, *bottom-up* e sanduíche. O quadro 1.1 resume os três tipos de integração.

Quadro 1.1 | Resumo das técnicas de integração

Abordagem	Pontos fortes	Pontos fracos
Implementação e, depois, a integração		Não tem isolamento de falhas. Falhas de projetos importantes são detectadas muito tardiamente. Artefatos de código com potencial para serem reutilizados não são testados adequadamente.
Integração *top-down*	Isolamento de falhas. Falhas de projetos importantes são detectadas logo no início.	Artefatos de código com potencial para serem reutilizados não são testados adequadamente.
Integração *bottom-up*	Isolamento de falhas. Artefatos de código com potencial para serem reutilizados são testados adequadamente.	Falhas de projeto importantes são detectadas muito tardiamente.
Integração sanduíche	Isolamento de falhas. Falhas de projetos importantes são detectadas logo no início. Artefatos de código com potencial para serem reutilizados são testados adequadamente.	

Fonte: Schach (2008, P. 475)

Reflita

A integração poderá ser bastante problemática no caso de os módulos simplesmente não se comunicarem. Um objeto escrito no módulo *a* passa, por exemplo, dois argumentos para um outro objeto escrito no módulo *b*. No entanto, este segundo objeto está preparado para receber três argumentos. Neste caso haverá falha na integração. Tal situação mostra a necessidade de se implantar gerenciamento do processo de integração, que deve ser conduzido pelo pessoal de qualidade.

Implantação de *software*

A implantação é a última fase de desenvolvimento de um *software*. Embora deva sofrer alterações durante sua vida útil, espera-se que o *software* seja disponibilizado ao cliente em sua versão final. Destaca Rezende (2005), que o envolvimento do cliente deve ser buscado nesta fase assim como o foi nas fases anteriores do projeto e que, da mesma forma que em outras etapas do processo de desenvolvimento, a implantação requer gerenciamento, como será abordado na sequência.

É comum que a implantação de um *software* se dê em substituição a um anterior. Neste caso, os dados mantidos pelo *software* antigo devem ser convertidos para o formato previsto para o atual, seja em forma de digitação ou de conversão via programa.

De acordo com Rezende (2005), a efetiva implantação de um *software* novo no lugar de um antigo pode acontecer das seguintes formas:

a) Direta: o funcionamento do *software* antigo cessa assim que o novo entra em operação. Não há concomitância na operação dos dois;

b) Paralela: os dois sistemas funcionam por um tempo em paralelo, com a base de dados atualizada em ambos. Neste caso, a segurança na implantação de sistema novo é maior;

c) Piloto: neste caso, o sistema atual poderá refazer os processamentos feitos pelo antigo, para fins de comparação de resultados;

d) Parcial ou por etapas: o funcionamento do novo sistema inclui apenas parte do antigo. As novas rotinas substituem aos poucos as antigas.

Manutenção de *software*

Os esforços de desenvolvimento de um *software* devem resultar na entrega de um produto que satisfaça os requisitos do usuário. Adequadamente, espera-se que o *software* sofra alterações e evolua. Uma vez em operação, defeitos são descobertos, ambientes operacionais mudam e novos requisitos dos usuários vêm à tona. A manutenção é parte integrante do ciclo de vida do *software* e deve receber o mesmo grau de atenção que outras fases.

A manutenção de *software* é definida como modificações em um produto de *software* após a entrega ao cliente a fim de corrigir

falhas, melhorar o desempenho ou adaptar o produto a um ambiente diferente daquele em que o sistema foi construído (IEEE, 2004).

Necessidade de manutenção

A manutenção é necessária para assegurar que o *software* continuará a satisfazer os requisitos do usuário. O sistema se altera devido a ações corretivas e não corretivas aplicadas ao *software*. A manutenção deve ser executada a fim de corrigir falhas, melhorar o projeto, implementar melhorias, construir interface com outros sistemas, adaptar programas para que novas facilidades de *hardware* possam ser usadas, migrar *software* legado, retirar *software* de operação.

Assimile

Manutenção de *software* é como se denomina, em geral, o processo de adaptação e otimização de um *software* já desenvolvido, bem como, a correção de defeitos que ele possa ter. A manutenção é necessária para que um produto de *software* preserve sua qualidade ao longo do tempo, pois se isso não for feito, haverá uma deterioração do valor percebido desse *software* e, portanto, de sua qualidade." (WAZLAWICK, 2013, p. 317)

Um *software* legado é um sistema antiquado que continua em uso porque o usuário (tipicamente uma organização) não deseja substituí-lo ou projetá-lo novamente.

Categorias

Manutenção corretiva: modificação reativa em um produto de *software* executada após a entrega a fim de corrigir problemas descobertos.

Manutenção adaptativa: modificação em um produto de *software* executada após a entrega do produto a fim de manter o *software* usável em um ambiente alterado ou em alteração.

Manutenção perfectiva: modificação em um produto de *software* realizada após a entrega a fim de melhorar o desempenho ou a manutenibilidade.

Manutenção preventiva: modificação em um *software* após a entrega a fim de reparar falhas latentes antes que se tornem efetivas (IEEE, 2004).

Exemplificando

O exemplo que segue nos transmite a ideia do quanto a preparação prévia do programa para receber manutenção é importante e sobre o pensamento do cliente em relação à facilidade em se executar mudanças em um *software*.

Um programador foi chamado pelo governo para criar um produto de *software* que mantivesse sete, e exatamente sete, tipos de frutas que eram controladas e comercializadas por certo órgão governamental. Havia ordem expressa de que o banco de dados fosse projetado para sete frutas, sem previsão de expansão. O programa foi entregue e seguia em perfeito funcionamento até que, um ano após sua implantação, o gerente do órgão viu-se na necessidade de incluir mais uma fruta no controle do programa. Chamado a executar a manutenção do produto, o programador constatou que o projetista, desobedecendo às orientações iniciais, havia deixado alguns campos adicionais no banco de dados, o que permitiu a incorporação da oitava fruta.

Mais um ano se passou e, para que a recente mudança na legislação fosse atendida, o programa deveria agora acomodar o controle de mais 26 frutas. Informado sobre essa necessidade, o programador protestou, alegando que tal alteração levaria praticamente o mesmo tempo que a criação de um novo sistema. "Que ridículo!", retrucou o gerente. "Você não teve nenhuma dificuldade para acrescentar a oitava fruta. Simplesmente faça a mesma coisa agora, mais 26 vezes." (SCHACH, 2008).

Manutenibilidade: definida como a facilidade com que um *software* pode sofrer manutenção, melhoramentos, adaptações ou correções para satisfazer requisitos específicos. Ela deve ser perseguida durante o desenvolvimento do *software*, de modo a minimizar os custos futuros de manutenção, que são inevitáveis.

Faça você mesmo

A correta classificação das espécies de manutenção a serem aplicadas no *software* é fundamental para o registro da atividade e formação da documentação apropriada. Com base no que estudamos nesta seção, classifique em corretiva, adaptativa, preventiva ou perfectiva as atividades de manutenção descritas na sequência.

a) Alteração do código para tornar mais rápido o processamento de uma imagem;

b) Manutenção feita para acomodar alteração da moeda corrente;

c) Correção do *layout* do relatório de vendas mensais.

Sem medo de errar

Concluídas as fases de tratamento dos requisitos e de elaboração do projeto do novo *software*, resta cumprir as etapas finais do processo de desenvolvimento e, por fim, entregar o programa ao cliente. Sua missão é a de estruturar o processo, cuidando para que todas as ações planejadas sejam factíveis e que o projeto possa ser levado à conclusão.

Sendo assim, ao trabalho!

O planejamento da implementação de *software* inclui as seguintes atividades:

a) Definição dos programadores: cada um dos responsáveis pela codificação dos módulos projetados deve ser destacado e informado sobre a linguagem de programação a ser utilizada. É boa prática a busca e adequação de funções já codificadas em outros projetos e que eventualmente poderão ser utilizadas neste atual;

b) Escalonamento das atividades: a divisão de tarefas e seus respectivos responsáveis devem ser definidos em documento público. Nesse ponto, deve-se também definir os responsáveis pela integração e teste dos módulos;

c) Controle de versão: durante a codificação, as atualizações feitas no sistema serão informadas a toda equipe de programação, por meio de ferramenta de controle de versão instalada em servidor próprio.

A implantação prevê a efetiva instalação do programa em servidor próprio do cliente e a escala de treinamento aplicado aos usuários. O treinamento será prestado na semana que antecede a colocação do programa em funcionamento, com dois profissionais destacados para o trabalho. Não há sistema em funcionamento que execute as mesmas funções do novo sistema, daí a dispensa de planejamento para conversão de dados e troca de sistemas.

Por fim, a manutenção deverá ser prestada sob demanda, exceto em casos em que correções devam ser feitas. O cliente, sentindo

necessidade de alterar ou aprimorar o sistema, contratará horas adicionais da equipe de programadores. As correções, por sua vez, serão feitas sem custo.

Uma boa referência documental para essa atividade pode ser encontrada em: <http://www2.ati.pe.gov.br/c/document_library/get_file?p_l_id=144654&folderId=144374&name=DLFE-15402.pdf>. Acesso em: 17 nov. 2015.

Atenção

O treinamento das pessoas que utilizarão o novo programa é providência fundamental para sucesso do projeto. A transmissão das práticas relacionadas ao programa, quando feita por profissionais hábeis em motivar o cliente para sua efetiva utilização, pode fazer a diferença entre a plena adesão e o não reconhecimento da utilidade de um programa.

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.	
"Retomando a manutenção do *software*"	
1. Competência geral	Conhecer as principais metodologias de desenvolvimento de *software*, normas de qualidade e processos de teste de *software*.
2. Objetivos de aprendizagem	Transferência dos conceitos aprendidos para situação semelhante à apresentada no início da seção.
3. Conteúdos relacionados	Etapas finais do processo de *software*: implementação, integração, implantação e manutenção.
4. Descrição da SP	Dando continuidade ao processo de resgate de trabalhos interrompidos da *You-Soft*, o pessoal de direção da *XAX-Sooft* deparou-se com um projeto que chegou a ser entregue ao cliente, mas que, por conta da saída de programadores da empresa, ficou carente de manutenção pós-entrega. Os programadores que os substituíram não conseguiram compreender corretamente a escrita do programa e as alterações necessárias no código não puderam ser feitas. Atualmente o programa encontra-se em funcionamento parcial apenas. Nossa missão é planejar e implementar a oferta de manutenção ao cliente da *You-Soft* e, ao mesmo tempo, oferecer mão de obra imediata para que o programa entregue funcione por completo.

5. Resolução da SP	A retomada da normalidade no atendimento deve ser iniciada pelo destaque de profissionais que deverão estudar e compreender o sistema que funciona parcialmente no cliente. A documentação deve ser atualizada ou criada, caso não exista. Para facilitar a manutenção futura, o código deverá passar por processo de refatoração, a fim de se adequar ao padrão de codificação da *XAX-Sooft*.

Lembre-se

Assim como na maioria das áreas de atuação profissional, a rotatividade de programadores é tema que tem preocupado gestores da área de TI. Algumas boas práticas podem, no entanto, reduzir o *turnover*.

Saiba mais em: <http://cio.com.br/gestao/2015/06/11/cinco-dicas-para-reduzir-o-turnover-na-area-de-ti/>. Acesso em: 2 nov. 2015.

Faça você mesmo

A metodologia tradicional de desenvolvimento tem sido questionada quanto à sua efetividade e, aos poucos, tem sido substituída por outras metodologias mais recentes. No entanto, ela ainda é bastante utilizada. Faça levantamento de um caso de sucesso no desenvolvimento de *software* utilizando a metodologia tradicional.

Faça valer a pena!

1. Em relação ao processo de implementação de *software*, analise as afirmações que seguem:

I - A escolha da linguagem de programação deve ser feita sem a participação do cliente, já que não lhe cabe interferir em decisões técnicas.

II - A padronização dos elementos do programa, tais como nomes de variáveis, endentações e nomes de classes deve ser adotada para toda a equipe, já que tal ação tende a facilitar a codificação e manutenção futura.

III - Não há necessidade de se documentar código, já que atualmente as linguagens de programação são autoexplicativas.

É correto o que se afirma apenas em:

a) II e III.

b) I e III.

c) II.

d) I, II e III.

e) I.

2. Em relação à integração de *software*, assinale a afirmação correta.

a) Integrar um *software* significa distribuir à equipe tarefas de implementação, de modo a se ter, ao final do trabalho, um programa integral.

b) Integrar um *software* significa juntar em um só programa todos os módulos que foram construídos separadamente durante a implementação.

c) O processo de integração não prevê a aplicação de testes nas interações entre módulos, já que testar apenas o programa unificado basta.

d) O processo de implementar todos os módulos para então integrá-los é eficiente no isolamento de eventual falha em um módulo.

e) A técnica de integração *bottom-up* é eficiente em detectar falhas de projeto logo no início do processo.

3. Assinale a alternativa que contém expressões que completam corretamente as lacunas nas frases abaixo.

I) A substituição de parte defeituosa de um módulo configura manutenção _____

II) O objetivo da manutenção _____ é melhorar determinada característica ou função do programa.

III) A inclusão de forma alternativa de cálculo de juros determinada em lei constitui manutenção _____

a) corretiva, adaptativa, preventiva.

b) preventiva, perfectiva, adaptativa.

c) perfectiva, perfectiva, preventiva.

d) corretiva, perfectiva, adaptativa.

e) corretiva, corretiva, adaptativa.

Referências

IEEE. **SWEBOK**: a project of the IEEE Computer Society Professional Practices Committee. Los Alamitos: IEEE, 2004.

LAPLANTE, P. A. **What every engineer should know about**: software engineering. London: CRC, 2007.

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

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

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

SOMMERVILLE, I. **Engenharia de *software***. 8. ed. São Paulo: Addison Wesley, 2008. 592 p.

WAZLAWICK, R. S. **Engenharia de *software***: conceitos e práticas. Rio de Janeiro: Elsiever, 2013.

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 *XAX-Sooft* 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 a apresentar um conteúdo que permitirá a você 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 *XAX-Sooft* e sua missão é implantar metodologia ágil nela *XAX-Sooft*, 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, proposta de 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 desse 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 *XAX-Sooft* 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 *XAX-Sooft*, 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.

Desse cenário naturalmente despontam as metodologias ágeis. Elas diferem dos processos mais tradicionais em muitos aspectos, mas sobretudo naqueles que compõem as aspirações da *XAX-Sooft*.

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 um modelo ágil.

Esse 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!

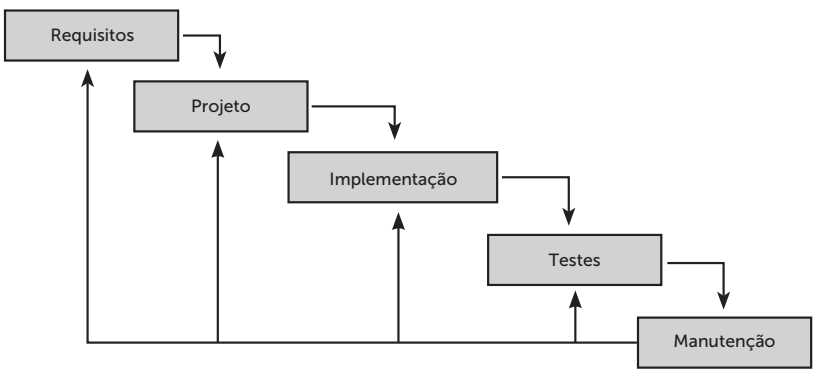
Não pode faltar

É possível que você conheça alguma organização que desenvolve *softwares*. Se conhece, é bastante provável que já tenha ouvido de algum membro dessa 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 essas 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, com sequência definida de fases, como mostra a figura 2.1.

Figura 2.1 | Etapas do modelo tradicional de desenvolvimento



Fonte: adaptada 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 esses 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, predeterminados 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 o 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 essa 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 essa 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 esse 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 à 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." (WAZLAWICK, 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 (WAZLAWICK, 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 essas equipes. Esse 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 (WAZLAWICK, 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 *XAX-Sooft*, 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 *XAX-Sooft* 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 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 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 o 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." (WAZLAWICK, 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 *XAX-Sooft* 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 nesse 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, nessa 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 desse 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. Essa 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 na qual 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, na qual ele deverá escrever uma (e apenas uma por cartão) funcionalidade que deseja para o programa, de forma simples e objetiva.

Você pode achar essa 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 história. Elas são a base para o planejamento dos desenvolvedores, que podem dividir a história em tarefas, em nome de um planejamento mais preciso e caso as histórias sejam extensas demais para apenas um dia de trabalho.

Exemplificando

Podem ser considerados bons exemplos de histórias:

- 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 esse 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 história;

• 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? Veja 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, essa 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: 1 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 registro 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 a 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 *XAX-Sooft* 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 nesse desafio, é preciso que se empenhe na compreensão da teoria desenvolvida na seção "Não pode faltar", na qual 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, nesse 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 a 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 – porque 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 nesse 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 nesse repositório e, no caso de a versão atual apresentar problemas, a anterior será recuperada. Todos os desenvolvedores deverão ter acesso a esse 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 conhecê-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 essa 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 mudança alguma na lógica implementada e ou no funcionamento do programa.

Arriscado, não acha? Pois essa 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 à 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 *XAX-Sooft* 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 *XAX-Sooft* 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 desse 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. Essa 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 a 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

Método*logia 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 *XAX-Sooft* 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 *XAX-Sooft*.

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 dessas ações, deve também compor o documento suas sugestões para que a satisfação e o encantamento do cliente sejam garantidos.

Com essa atividade, você poderá aprofundar-se nos conceitos e na correta operacionalização das práticas do XP implantadas pela *XAX-Sooft* 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, a fim 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 esse 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 essa 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 desses 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 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 história 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 história em *software*. Você se lembra de como a equipe do XP planejava o desenvolvimento das funcionalidades? Pois bem, o princípio 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 nessa 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 a 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 funcionalidades viram programa. Conforme você já estudou, quem determina quais são essas 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 esse fim. Essa é a relação entre *Product Backlog* e *Sprint Backlog*.

É normal e desejável que essa 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 esse 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. Essa lista é derivada da primeira, leva o nome de *Sprint Backlog* e é criada durante a 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*.

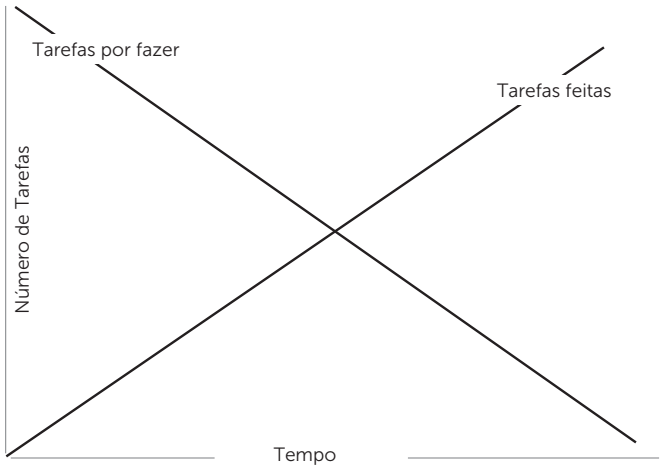
Assimile

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 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 feita e a quantidade de trabalho a ser feita 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 2.2.

Figura 2.2 | Relação ideal entre tarefas



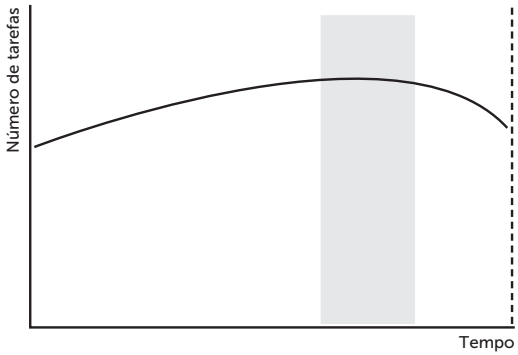
Fonte: Wazlawick (2013, p. 59).

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 (MAR, 2006).

Como exemplo, a figura 2.3 mostra o comportamento de uma equipe que conseguiu entender as funcionalidades depois de passada a maior parte do *Sprint*. Essa curva pode indicar perfil de iniciante nos membros da equipe.

Figura 2.3 | Comportamento típico de equipe iniciante



Fonte: elaborada pelo 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 a sério esse 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 *XAX-Sooft* 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, à medida 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 *XAX-Sooft*.

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 manutenção e aprimoramento das práticas do XP. Como desdobramento direto dessas ações, o estreitamento da relação com os clientes deveria também ser buscado. Deste ponto em diante será apresentada uma forma possível de compor esse relatório.

1. Ao menos dois membros previamente escolhidos da equipe deverão ficar responsáveis pelo treinamento dos novos desenvolvedores que venham a ingressar na *XAX-Sooft*. 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 *XAX-Sooft* (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 *Coore-Sooft*, 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 essa 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 *Coore-Sooft* 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 desse 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*

II. *Product Owner*

III. *Scrum Master*

1. Responsável por determinar quais funcionalidades serão implementadas no *Sprint*.

2. Equipe normalmente estruturada sem clara divisão de especialidades entre seus membros.

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

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

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

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

PAULA FILHO, W. de P.. **Engenharia de software**: fundamentos, métodos e padrões. 2. ed. Rio de Janeiro: Livros Técnicos e Científicos, 2005. 602 p.

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

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

SOMMERVILLE, I. **Engenharia de software**. 8. ed. São Paulo: Addison Wesley, 2008. 592 p.

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

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

Gerenciamento de qualidade de *software*

Convite ao estudo

Olá! Seja bem-vindo à terceira unidade da disciplina de Engenharia de *software*. Nas duas primeiras unidades deste material você teve a oportunidade de conhecer dois modos distintos, quase antagônicos, de desenvolvimento de *software*: o modelo tradicional e o modelo ágil. Antes deles, no entanto, você foi conduzido até os fundamentos da Engenharia de *software* e à teoria associada ao processo de desenvolvimento de um produto; bases necessárias para sua entrada no universo das metodologias de desenvolvimento que seriam abordadas na sequência.

Conforme o conteúdo teórico era apresentado, a história da *XAX-Sooft* era contada. No início, tínhamos uma empresa pequena e sem metodologia formal implantada. Com a chegada de um grande cliente, surgiu também a necessidade de organização dos seus processos e da adoção de um modelo de desenvolvimento para seu produto. A metodologia Cascata foi introduzida e, passado algum tempo, o *Extreme Programming* passou a ser o modelo oficial de desenvolvimento.

É fato que a *XAX-Sooft* evoluiu. No entanto, o modelo utilizado, por melhor e mais moderno que seja, não dispensa uma providência elementar: o gerenciamento da qualidade do produto. O objetivo geral desta unidade é que você se torne capaz de gerenciar por completo a qualidade do produto e do processo utilizado para criá-lo. Cada seção, em específico, tem os objetivos que seguem:

- Abordar os fundamentos de qualidade e da gestão da qualidade do *software*;
- Abordar os fatores que afetam a qualidade e as ações de Garantia da Qualidade do *Software* (SQA);
- Abordar as normas de qualidade mais referenciadas;
- Introduzir métricas e inspeções de *software*.

Como consequência do atingimento desses objetivos, você deverá desenvolver competência técnica para conhecer e conduzir processos de qualidade de *software*. Você atingirá os objetivos e desenvolverá as competências desta unidade por meio da resolução da realidade profissional em quatro etapas, descritas aqui:

1. Criar relatório contendo o levantamento das práticas de qualidade atualmente implementadas;

2. Criar relatório contendo o levantamento dos fatores que afetam a qualidade dos produtos;

3. Criar relatório contendo o processo de implantação de norma de qualidade na *XAX-Sooft*;

4. Criar relatório contendo o processo de implantação de inspeções, medições e métricas de *software*.

Nas próximas páginas serão detalhados a primeira parte de nossa missão, o conteúdo teórico proposto sobre qualidade de *software* e novas abordagens da situação-problema.

Bom trabalho!

Seção 3.1

A gestão da qualidade no processo de desenvolvimento de software

Diálogo aberto

O conteúdo teórico apresentado na unidade 2 abordou três das principais metodologias ágeis usadas para desenvolvimento de *software* e ajudou você a trilhar o caminho até a efetiva implantação do *Extreme Programming* na *XAX-Sooft*.

O modo ágil de se conduzir um projeto já foi incorporado ao cotidiano da equipe e as entregas seguem sendo feitas, na grande maioria das vezes, no prazo e no limite do orçamento combinados. O encantamento do cliente, de certa forma a grande meta a ser alcançada pela direção da empresa, tem sido verificado por meio de bons retornos dados à equipe por quem a contratou.

O aprimoramento da qualidade do processo e do produto deve ser, no entanto, desafio constante na vida da *XAX-Sooft* e de qualquer outra empresa. Por esse motivo – e por alguns outros que lhe serão apresentados no decorrer desta unidade – ações específicas de garantia da qualidade do produto devem ser tomadas. Afinal, a obtenção de produto de qualidade é uma das justificativas da existência da própria Engenharia de *Software* como disciplina estruturada.

Para que seu aproveitamento nesta seção e nas seguintes seja ótimo, há questões importantes a serem respondidas: afinal, o que é qualidade? Estamos tratando de uma percepção subjetiva ou de um conceito puramente objetivo? Um bom produto criado há 30 anos seria hoje considerado um produto de qualidade? Intrigante, não acha?

Utilizando os temas que estão sendo abordados nesta seção e sua crescente habilidade em diagnosticar situações e procedimentos ativos na empresa, você deve criar um relatório contendo o levantamento das práticas de qualidade atualmente implementadas. Nesse relatório devem ser descritas as ações atualmente tomadas para que o produto da *XAX-Sooft esteja* adequado ao seu propósito e em conformidade com os requisitos.

Na seção "Não pode faltar" você terá contato com fundamentos de qualidade de *software* e sua gestão. Um excelente aproveitamento dos seus estudos é o que desejamos a você.

Não pode faltar

Não se pode conceber um produto, qualquer que seja sua natureza, que não seja criado levando-se em conta sua qualidade. A busca por níveis satisfatórios e previamente definidos de excelência deve basear qualquer processo de fabricação em larga escala ou de manufatura, sob pena de se obter produto com baixa aceitação de mercado.

Nesta seção, você terá contato com o conceito de qualidade e com sua gestão no processo de desenvolvimento de um *software*. Entender a importância da qualidade é o primeiro passo em direção à sua elevação de condição indispensável para o sucesso de um produto. Sigamos adiante!

Conceito de qualidade

O que é qualidade de *software* e por que ela é tão importante? Uma boa maneira de começarmos esta discussão é afastando a ideia de que qualidade signifique perfeição. É comum entendermos que sempre será possível encontrar algo a ser melhorado em algo que se reconhece como de boa qualidade.

Também não se pode caracterizar a qualidade como algo absoluto, definitivo e que tenha meios de ser medida universalmente, em parâmetros aceitáveis para todas as pessoas. O que parece ser de boa qualidade para mim pode não parecer para você. E vice-versa.

Por ser considerada um conceito de muitas dimensões, talvez uma boa maneira de começarmos a entender a qualidade, é que esta se realiza por meio de um conjunto de características e atributos de um certo produto. Como qualidade não significa perfeição, é natural que tenhamos que estabelecer um nível aceitável de qualidade para um produto e meios para verificar se esse nível foi alcançado.

Embora saibamos que traços de subjetividade e a perspectiva própria do avaliador farão parte de uma avaliação, o "nível aceitável" de qualidade precisa ser o mais objetivo possível, extraído por meio de processos estruturados e ferramentas apropriadas de medição de qualidade.

Ao longo dos anos, autores e organizações têm definido o termo qualidade de formas diferentes. Para Crosby (1979, p. 23), qualidade é a "conformidade com os requisitos do usuário". Para ele, se o produto

atende aos requisitos explícitos e implícitos, significa que o produto é de qualidade. Por exemplo, ao comprarmos uma televisão, se os nossos requisitos estiverem de acordo com as suas características, o aparelho então nos serve. Caso contrário, escolhemos alguma outra que esteja mais próxima de nossas expectativas.

Humphrey (1989, p. 280) refere-se à qualidade como "a conquista de excelentes níveis de adequação ao propósito", enquanto a IBM cunhou a frase "qualidade dirigida ao mercado", que se baseia na conquista total da satisfação do cliente.

A expressão "adequação ao propósito" pressupõe a existência de um registro da descrição do propósito do produto. Tomando-se como exemplo um secador de cabelos, seu próprio nome carrega sua funcionalidade. Características técnicas são colocadas num manual de instruções. No caso de um *software*, seu propósito está inserido na especificação de requisitos.

Mais recentemente, a qualidade foi definida pela norma ISO 9001-00 como "o grau no qual um conjunto de características inerentes preenche os requisitos".

Assimile

Observe outros dois conceitos e ideias relacionados à qualidade:

"Qualidade de *Software* é um processo sistemático que focaliza todas as etapas e artefatos produzidos com o objetivo de garantir a conformidade de processos e produtos, prevenindo e eliminando ddefeitos." (BARTIÉ, 2002, p. 16).

"Qualidade é a totalidade das características de um produto de *software* que lhe confere a capacidade de satisfazer necessidades implícitas e explícitas." (ISO/IEC 9126-1, 2003, p. 17)

Você já deve ter se perguntado: o que define, afinal, um bom *software*? É de se esperar que nesta resposta apareçam os elementos que compõem o conceito de qualidade dos quais tratamos há pouco. Uma das possíveis medidas de qualidade é, de fato, a adequação ao propósito, o que significa que o *software* funciona efetivamente de acordo com o que foi projetado. Shaffer (2013) propõe uma forma interessante de se medir a qualidade de um *software* que é o seu valor de mercado, especialmente em situações em que alguém

está procurando investir na empresa que o produz. É muito comum também que as pessoas percebam a qualidade de um *software* como reflexo da qualidade do processo usado para criá-lo.

Embora você possa não encontrar uma definição universal e definitiva para a qualidade aplicada a um *software*, fatores como a corretude, a eficiência e a usabilidade são algumas medidas amplamente aceitas como indicadores da qualidade do produto. Esses fatores serão estudados com mais propriedade e detalhes na próxima seção, mas vale uma olhada agora:

• **Corretude**: capacidade do *software* em executar suas funcionalidades conforme elas foram definidas. Se pudéssemos resumir esse fator em uma pergunta, ela seria próxima de: "o *software* faz aquilo que eu quero?"

• **Eficiência**: relaciona-se ao grau de adequação do programa aos recursos de *hardware*, tais como processador e memória. Eficiência é uma palavra muito comumente usada, embora muitas vezes de forma incorreta. Para ele, eficiência é a medida de quantos recursos são usados para que uma tarefa seja completada. Atualmente, com o *hardware* custando tão pouco, não se presta muita atenção no fator eficiência como no passado, exceto em processos que incluem quantidade muito grande de dados, como o *Big Data*, por exemplo (SHAFFER, 2013).

• **Usabilidade**: este fator está relacionado com a facilidade de uso do produto. Em outras palavras, trata-se da medida da capacidade do público-alvo em obter valor do *software* por meio da sua interface.

• **Portabilidade**: é possível usá-lo em outra plataforma? Trata-se da medida de facilidade em mudar o *software* de uma plataforma (*Windows*, por exemplo) para outra (*Mac*, por exemplo).

• **Interoperabilidade**: trata-se da "capacidade de diversos sistemas e organizações trabalharem em conjunto (interoperar) de modo a garantir que pessoas, organizações e sistemas computacionais interajam para trocar informações de maneira eficaz e eficiente". (Disponível em: <http://www.governoeletronico.gov.br/acoes-e-projetos/e-ping-padroes-de-interoperabilidade/o-que-e-interoperabilidade>. Acesso em: 18 fev. 2016).

Com essas características, você começa a entender como a qualidade de um produto é avaliada. Nas páginas seguintes você terá

contato com aspectos da gestão da qualidade do *software*, colocada em duas diferentes perspectivas. Sigamos adiante.

Pesquise mais

Você encontrará questões interessantes sobre qualidade no artigo publicado na "II Escola Regional de Informática da Sociedade Brasileira de Computação Regional de São Paulo – II ERI da SBC" – Piracicaba, SP – junho de 1997, p. 173-189. Disponível em: <https://xa.yimg.com/kq/groups/21646421/371309618/name/Modelos+de+Qualidade+de+Software.pdf>. Acesso em: 18 jan. 2016.

Gestão da qualidade do *software*

Com a finalidade de conseguirmos um melhor aproveitamento neste tema, ficaria melhor se definirmos a gestão da qualidade como um sistema. Idealmente esse sistema de gestão da qualidade já deve estar incorporado na organização e, como se espera de qualquer sistema, ele deve incluir processos, pessoas e ferramentas dirigidas à obtenção da qualidade nas entregas.

Quando tomamos um sistema de gerenciamento financeiro ou um sistema de gerenciamento de pessoas em uma organização como exemplos, por si sós eles não deveriam demandar uma nova equipe de gerenciamento ou novos recursos. Ao contrário disso, espera-se que tais sistemas sejam partes integrantes da organização, alinhadas com necessidades particulares de finanças ou de gestão de pessoas.

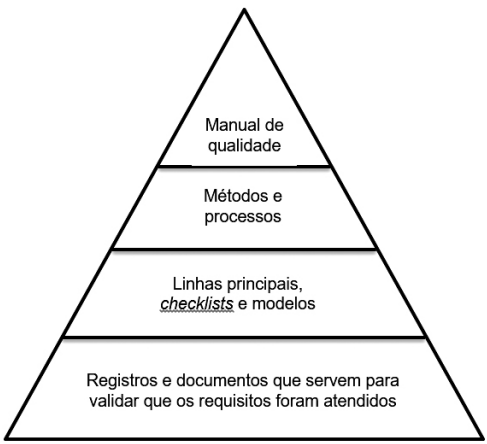
De acordo com Moorthy (2013), um sistema de gestão de qualidade de *software* deve possuir 4 níveis: o nível 1 é composto pelo manual de qualidade da empresa; o nível 2 refere-se aos métodos e processos usados pela equipe para entregar suas tarefas; o nível 3 contém as linhas principais, os *checklists* e os modelos, usados com bastante frequência no dia a dia e importantes na manutenção da consistência das informações e, por fim, o nível 4 refere-se aos registros e documentos usados para fins de validação de um produto, usados como evidências de uma atividade e úteis para referência futura. A figura 3.1 mostra a organização dos níveis de um sistema de gestão de qualidade.

De acordo com SWEBOK (2004), publicação criada pela IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos, do inglês *Institute of Electrical and Electronics Engineers*), o gerenciamento da qualidade de *software* é tratado como um processo, que se aplica a todas as perspectivas de processos de *software*, produtos e recursos. Ele define

os requisitos e os responsáveis pelos processos, suas medições e saídas, além dos canais de *feedback*. O planejamento da qualidade do *software* envolve:

- A definição do produto em termos de suas características de qualidade;
- O planejamento do processo para se obter o produto desejado.

Figura 3.1 | Níveis de um sistema de gestão de qualidade de *software*



Fonte: Moorthy (2013, p. 184).

Pois bem, um processo específico de gestão de *software* é definido no padrão IEEE 12207.0-96 e inclui o **Processo de garantia da qualidade**, mais conhecido pela abreviatura SQA (*Software Quality Assurance* ou Garantia da Qualidade do *Software*).

Esse processo – também composto por várias etapas – visa assegurar que os produtos de *software* e os processos que os constroem estão em conformidade com os requisitos (ou funcionalidades) por meio de planejamento e execução de um conjunto de atividades destinadas a transmitir a certeza de que a qualidade é fator integrante do produto. Isso significa que a equipe poderá se assegurar de que o problema foi clara e suficientemente compreendido e que os requisitos da solução foram definidos e expressos de forma adequada. Em relação ao processo, o papel do SQA é assegurar que tudo será implementado de acordo com o plano traçado. Desafiador, não acha?

Reflita

Sabemos que um processo de gerenciamento de qualidade tem por objetivo a tomada de providências que incluam a qualidade em todas as ações executadas durante o ciclo de desenvolvimento de um produto. No entanto, como garantir que esse processo foi concebido e aplicado corretamente? Como medir a qualidade desse processo? Em que momento a equipe poderá entender que construiu, de fato, algo de qualidade?

Outro processo que visa conferir qualidade ao produto é o que chamamos de **Verificação e Validação**. Para facilitar as referências ao tema, os termos verificação e validação são tratados como apenas um. De acordo com SWEBOK (2004), trata-se de um processo bem estruturado para avaliar os produtos de *software* em todo o seu ciclo de vida, do planejamento até sua efetiva entrega. Em resumo, retrata o esforço da equipe para garantir que a qualidade está embutida no *software* e que ele reflete o desejo do usuário. A V&V, como também é conhecido esse processo, está interessada diretamente na qualidade do produto.

O grupo **revisões e auditorias**, nosso último processo de qualidade abordado aqui e também tratado como um único item, inclui dois principais procedimentos:

• **Revisões técnicas**: o objetivo de uma revisão (ou análise) técnica é o de avaliar um produto de *software* para determinar a sua adequação para a sua utilização pretendida. O objetivo é o de identificar discrepâncias a partir das especificações e dos padrões aprovados. Os resultados devem fornecer evidências que confirmem (ou não) que o produto atende às especificações;

• **Inspeções**: o propósito de uma inspeção é detectar e identificar anomalias no *software*. Esta prática se diferencia das revisões em dois aspectos: alguém que exerça cargo de gestão sobre qualquer membro da equipe de inspeção não deverá participar desse processo, e uma inspeção deve ser conduzida por um facilitador imparcial, treinado em técnicas de inspeção.

As inspeções incluem também um líder, um responsável pelos registros da seção e um número reduzido de inspetores, comumente de 2 a 5 pessoas.

Na próxima seção, dedicada exclusivamente ao estudo da SQA e dos fatores que afetam a qualidade do *software*, você conhecerá de forma mais profunda e abrangente as providências que uma equipe pode tomar para garantir a qualidade do seu produto de *software* e a satisfação do cliente. Na sequência, seu desafio prático para esta seção será mais bem definido e uma solução possível lhe será indicada.

Exemplificando

Um bom exemplo de como um setor pode se organizar para obter *software* de qualidade, é dado pelo caso em que determinado segmento do setor agropecuário, junto com profissionais de TI, estabeleceu um conjunto de características que foram destacadas como imprescindíveis para avaliação da qualidade de um *software* agropecuário. Em linhas gerais, as características eleitas foram: facilidade de uso (a interface é facilmente personalizada), facilidade de operação (é simples a entrada de dados de natureza física, zootécnica, financeira no *software*?) e integridade do sistema (o programa é capaz de manter o processamento, a despeito da ocorrência de ações inesperadas?), entre outras (ROCHA; MALDONADO; WEBER, 2001).

Faça você mesmo

No decorrer desta seção foi feita menção à **Verificação e Validação** como uma das providências para assegurar a qualidade do *software*. Será que V&V são atividades redundantes ou complementares? Executadas em conjunto, podem ser consideradas como teste? Em outras palavras, Verificação + Validação = Teste? Pesquise mais sobre o tema e sintetize o que aprendeu a respeito em relatório.

Sem medo de errar

As práticas ágeis implantadas na *XAX-Sooft* têm assumido importância crescente no processo de desenvolvimento da empresa e se mostrado eficientes no aprimoramento da relação com os clientes. Sabemos que as entregas vêm sendo feitas de forma satisfatória, mas não há na *XAX-Sooft* ainda processo definido e formalizado de gerenciamento da qualidade do produto. Isso significa que, embora os programas atendam à maioria dos requisitos estabelecidos, as providências de qualidade tomadas pela equipe durante sua construção baseiam-se muito mais na percepção subjetiva de seus membros do que em práticas e medidas objetivas e consagradas pela prática.

Neste cenário, fica muito difícil que a equipe e seus gerentes consigam estabelecer níveis aceitáveis de qualidade do produto, já que não contam com meios formais para medi-la.

No item "Diálogo aberto" foi proposto como desafio para esta seção o levantamento das práticas de qualidade atualmente implementadas na *XAX-Sooft* e a geração de relatório contendo tal levantamento.

Uma solução possível para esse desafio é a que segue:

1. O relatório deve ser iniciado com a descrição de seu objetivo, que é o de levantar as práticas de qualidade vigentes na empresa;

2. Um relato de elementos-chave da equipe deve ser colhido. Nesse relato deverá estar presente o que cada um pensa da qualidade e como a coloca em prática;

3. Na sequência, o relatório deverá descrever como atualmente são feitos os testes no produto, com que frequência, em quais ocasiões e por quem;

4. Por fim, o relatório deverá explicitar os pontos falhos nas investidas da equipe em favor da qualidade, tais como falta de revisões nos artefatos liberados, a frequência insuficiente na aplicação de testes e a falta de documentação adequada, por exemplo.

Atenção

A implantação de um processo formal de qualidade pode, num primeiro momento, transmitir a impressão de que tempo precioso de desenvolvimento está sendo desperdiçado em medições, auditorias, inspeções e práticas afins. Cabe aos líderes da equipe conscientizá-la da importância desse processo.

Lembre-se

Embora a maioria dos envolvidos com Tecnologia da Informação sejam capazes de entender a importância da qualidade, é sempre bom reforçar seu valor no processo. Leia o artigo encontrado em: <http://www.devmedia.com.br/qualidade-de-software-parte-01/9408>. Acesso em: 11 jan. 2016, e reforce sua crença na qualidade.

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 as compare com as de seus colegas.	
Aprimorando as práticas atuais para aprimorar a qualidade	
1. Competência geral	Conhecer as principais metodologias de desenvolvimento de *software*, normas de qualidade e processos de teste de *software*.
2. Objetivos de aprendizagem	O objetivo desta seção é apresentar os fundamentos da qualidade e da gestão da qualidade do *software*.
3. Conteúdos relacionados	A gestão da qualidade no processo de desenvolvimento de *software*.
4. Descrição da SP	Uma empresa desenvolvedora de soluções de *software* tem estado parcialmente satisfeita com a qualidade dos seus produtos. Embora as entregas tenham sido na maioria das vezes satisfatórias, faz parte de seus objetivos de curto prazo a redução de ocorrência de falhas e o consequente aprimoramento dos programas. Há, no entanto, certa relutância da direção em adotar procedimento formal de qualidade, já que seus membros entendem que tal providência ocasionaria mudanças bruscas na rotina da equipe e investimento de esforço em processos sem retorno garantido. Seu desafio é indicar, no âmbito do modelo ágil, ações para melhorar o nível de qualidade dos produtos sem que grandes mudanças na equipe e em seus procedimentos sejam feitas.
5. Resolução da SP	A solução do caso requer as possíveis providências: • Durante o planejamento do programa, na fase em que o cliente descreve as funcionalidades desejadas para o programa, deverá haver sempre mais do que um responsável para validar tais funcionalidades, dispensando atenção especial para a viabilidade de sua implantação; • Uma vez validadas as funcionalidades desejadas, também deverá haver rígido acompanhamento do tempo e dos recursos alocados para o cumprimento delas. É comum que estimativas incorretas levem ao apressamento do desenvolvimento e a consequente queda no nível de qualidade do produto; • Acompanhamento rigoroso do processo de testes por parte do analista de testes. O cliente deverá ser orientado a criar testes adequados a cada funcionalidade. No entanto, a equipe deverá testar continuamente cada funcionalidade antes de sua liberação.

Lembre-se

"A qualidade é um aspecto que deve ser tratado simultaneamente ao processo de desenvolvimento do *software*, pois ela não pode ser imposta depois que o produto está finalizado." (ROCHA; MALDONADO; WEBER, 2001, p. 67)

Faça você mesmo

Qual o perfil de um profissional de qualidade? Qual sua atuação? Faça a leitura do artigo publicado em: <http://www.tiegestao.com.br/2013/09/01/carreira-analista-de-teste-o-que-e-e-o-que-faz/>. Acesso em: 20 jan. 2016.

Faça valer a pena

1. No contexto de um processo de gestão da qualidade de um *software*, assinale a alternativa que contém expressões que completam corretamente as lacunas na frase abaixo.

"De acordo com a IEEE, o gerenciamento da qualidade de software é tratado como _____, que se aplica a todas as perspectivas de processos de software, produtos e recursos. _____ da qualidade do software envolve a definição do produto em termos de suas características de _____ e o planejamento do processo para se obter o produto desejado".

a) uma opção, O desenvolvedor, confiabilidade.

b) um processo empírico, A perspectiva, requisitos.

c) um processo, O responsável, qualidade.

d) um processo, O planejamento, qualidade.

e) uma opção, A persecução, requisitos.

2. Em relação aos aspectos gerais do conceito de qualidade, analise as afirmações que seguem:

I - Pode-se considerar que um dos aspectos da qualidade é a conformidade do produto com os requisitos para ele estabelecidos.

II - O conceito de qualidade é universal e absoluto e em relação a ele não existem divergências.

III - Como o nível de excelência que se deseja para um programa é alto, qualidade deve ser sinônimo de perfeição.

IV - No âmbito da Tecnologia da Informação, a qualidade deve ser considerada como um conceito puramente subjetivo.

É verdadeiro o que se afirma apenas em:

a) I e IV.

b) II e III.

c) I.

d) II.

e) II e IV.

3. Em relação ao Processo de Garantia da Qualidade do *Software* (SQA), analise as afirmações que seguem:

I - Visa assegurar que os produtos construídos estão totalmente livres de erros.

II - Visa assegurar que os produtos de *software* e seus processos estão em conformidade com os requisitos.

III - Em relação ao processo, o SQA visa assegurar que tudo será implementado de acordo com o plano traçado.

É verdadeiro o que se afirma apenas em:

a) III.

b) II.

c) I e III.

d) II e III.

e) I.

Seção 3.2

A garantia da qualidade do *software*

Diálogo aberto

Seja bem-vindo de volta!

Em nossa última seção, tivemos a oportunidade de fazer o primeiro contato com um assunto que nos acompanhará até o final do curso: a qualidade relacionada ao produto de *software*, ou seja, a um programa.

Mesmo depois da introdução ao tema, muitas questões ainda ficaram para serem respondidas. Por ora, as que mais nos interessam são as seguintes: o que torna o programa um bom programa? Quais são os parâmetros de qualidade? O entendimento da importância desse tema é bem fácil de ser alcançado. Basta você considerar que, se a dependência das organizações tem aumentado em relação ao uso de *software*, é de se esperar que a exigência sobre a qualidade dos produtos tenha aumentado na mesma proporção.

A *XAX-Sooft*, claro, não deixará de dispensar atenção adequada a essa demanda. Seu corpo gerencial sabe que a implantação de estratégia de qualidade não é apenas uma opção a ser cogitada, mas uma ação necessária à sua própria sobrevivência.

Visando a adoção de procedimento formal de qualidade, você foi chamado a ser o responsável por dar início ao processo de mudança, levantando e relatando o que de efetivo tem sido feito em prol da excelência dos programas criados pela empresa.

Com esse diagnóstico em mãos, o próximo passo será realizar a apuração do nível dos indicadores de qualidade dos programas. Para que esse desafio seja superado com a excelência de sempre, você terá à disposição na seção "Não pode faltar" o conteúdo sobre os fatores que afetam a qualidade de um produto e os procedimentos de garantia da qualidade de um *software*.

Por meio deles, você poderá criar um relatório contendo o nível em que se encontra cada um dos fatores que afetam a qualidade dos produtos da *XAX-Sooft*. Por meio dessa prática, você desenvolverá competência para identificar, uma a uma, as características a serem aprimoradas em um programa, meio pelo qual será possível torná-lo integralmente adequado ao seu propósito, e dessa forma, poderá conhecer e conduzir processos de qualidade de *softwar*e. Eis o desafio.

Bom trabalho!

Não pode faltar

Na seção anterior a esta, foram discutidos conceitos e temas introdutórios da qualidade, indispensáveis para a formação da sua percepção sobre o tema. O caminho que levará você à compreensão dos mecanismos envolvidos num processo de qualidade passa agora pela Garantia da Qualidade do *Software* (SQA) e pelo estudo mais aprofundado dos fatores que exercem influência sobre a qualidade de um produto de *software*. Nos próximos parágrafos, esses assuntos serão abordados de maneira objetiva e dirigida ao melhor aproveitamento do tema para fins de torná-lo apto a transformar o desafio e as questões propostas em missões cumpridas. Siga adiante!

Garantia da Qualidade de *Software* (SQA)

Uma definição aceitável para a SQA (*Software Quality Assurance*) é expressa como "padrão planejado e sistemático de ações que são exigidas para garantir a qualidade do *software*" (PRESSMAN, 1995, p. 733). Em outras palavras, são as providências tomadas pela equipe para assegurar a qualidade do seu produto, de maneira vinculada ao processo que o cria.

Como a abrangência temporal de tais providências se estende das fases iniciais do processo até a finalização do *software*, é esperado que a responsabilidade das equipes do projeto e demais envolvidos também seja extensa. Por exemplo, os engenheiros de *software*, o profissional que gerencia o projeto, o grupo de SQA e o próprio cliente, todos eles são responsáveis pela garantia da qualidade.

A definição de SQA inclui a expressão "padrão planejado e sistemático de ações", não é mesmo? Pois bem, vamos então às ações.

Atividades SQA

Assegurar a qualidade de um *software* envolve algumas tarefas, três das quais são descritas na sequência (PRESSMAN, 1995):

• **Aplicação de métodos técnicos**: a SQA começa a ser aplicada desde a especificação e o desenho do sistema. Uma especificação malfeita – ou uma história mal escrita pelo cliente ou mal interpretada pela equipe – certamente irão comprometer a qualidade do produto final. Assim que uma especificação, protótipo ou *release* de um sistema tiverem sido criados, será apropriado avaliá-los quanto à qualidade.

• **Realização de revisões técnicas formais**: esta é a atividade central no contexto da avaliação da qualidade de um produto. Uma revisão técnica formal é um encontro no qual uma equipe (de 3 a 5 pessoas, normalmente) destacada para o trabalho, concentra-se na busca por problemas de qualidade no produto ou, mais comumente, numa parte específica dele. Elas são aplicadas em momentos variados do processo e são também usualmente referenciadas como *walkthrough* (passo a passo). Se você é fã de *games*, então já ouviu ou leu essa expressão.

• **Atividades de teste de *software***: por melhor que seja sua técnica, por maior que seja sua capacidade de fazer bons programas e por mais que você siga uma metodologia, submeter seu programa a um processo de teste nunca sairá de moda. O motivo é simples: errar é humano.

O teste é uma atividade desempenhada para avaliar a qualidade do produto e para sua melhoria, por meio da identificação de defeitos e problemas. O teste de *software* consiste na verificação dinâmica do comportamento de um programa em um conjunto finito de casos de teste, adequadamente selecionado a partir de um número geralmente infinito de execuções do programa (SWEBOK, 2004). Parece complicado? Como a próxima unidade será inteira dedicada ao assunto, trataremos aqui apenas de alguns elementos essenciais do teste.

O processo de teste envolve quatro etapas: planejamento, projeto de casos de teste, execução e avaliação dos resultados, que devem ser conduzidas ao longo de todo o processo de desenvolvimento (ROCHA; MALDONADO; WEBER, 2001). Testar, não se trata, portanto, de vasculhar o código, linha por linha procurando falhas no programa. Tampouco é executar o programa algumas vezes procurando por erros. Curioso para saber mais? Retomaremos o assunto na seção 3.4, que dará destaque para teste de *software*.

Fatores que influenciam na qualidade do *software*

Já que temos tratado de processos formais de qualidade na construção do nosso conhecimento sobre o tema, nada mais apropriado do que a busca por um modelo de qualidade consagrado para definirmos as características desejáveis em um programa. O modelo de qualidade ISO/IEC 25010:2011 estabelece um conjunto de oito características internas e externas de um *software*, divididas em outras tantas subcaracterísticas. Para facilitar a compreensão global da norma e facilitar sua síntese, a tabela 3.1 apresenta tais características

de qualidade, separadas entre próprias do produto e próprias do uso, conforme será explicado na sequência.

Tabela 3.1 | Modelo de qualidade da ISO 25010:2011

Tipo	Características	Subcaracterísticas
Características do produto	Adequação funcional	Completude funcional, Corretude funcional (acurácia) e funcionalidade apropriada.
	Confiabilidade	Maturidade, disponibilidade, tolerância a falhas e recuperabilidade.
	Usabilidade	Apropriação reconhecível, inteligibilidade, operabilidade, proteção contra erro do usuário, estética de interface com o usuário e acessibilidade.
	Eficiência de Desempenho	Comportamento em relação ao tempo, utilização de recursos, capacidade.
	Segurança	Confidencialidade, integridade, não repúdio, rastreabilidade de uso e autenticidade.
	Compatibilidade	Coexistência e Interoperabilidade.
	Capacidade de manutenção	Modularidade, reusabilidade, analisabilidade, modificabilidade, testabilidade.
	Portabilidade	Adaptabilidade, instalabilidade e substituibilidade.
Características de uso	Efetividade	Efetividade
	Eficiência	Eficiência
	Satisfação	Utilidade, prazer, conforto e confiança.
	Uso sem riscos	Mitigação de risco econômico, mitigação de risco a saúde e segurança e mitigação de risco ambiental.
	Cobertura de contexto	Completude de contexto e flexibilidade.

Fonte: Wazlawick (2013, p. 233).

Os parâmetros de qualidade normalmente são segmentados entre os que possuem mais afinidade com o processo, com o produto percebido pelo usuário ou com o produto sob o ponto de vista da equipe. As medidas de qualidade internas, por exemplo, servem para avaliar aspectos que usualmente são percebidos apenas pelos desenvolvedores. A capacidade de manutenção e a facilidade em se aplicar testes são bons exemplos dessas medidas.

As medidas de qualidade externas alcançam características avaliadas pela equipe do ponto de vista do usuário. Por exemplo, a eficiência e capacidade de operação de um programa.

O modelo ISO/IEC 25010:2011, cujas características foram resumidas na tabela acima, agregou as características internas e externas num único grupo e o chamou de **características do produto**. Elas podem ser avaliadas no ambiente de desenvolvimento, ao passo que as características do ***software* em uso** podem apenas ser avaliadas durante o efetivo uso do sistema.

Parece bastante desafiador construir um produto que tenha boa adequação a todas essas características, não acha?

Pesquise mais

A ISO (Organização Internacional para Padronização ou Organização Internacional de Normalização) é uma organização independente e não governamental, presente em 162 países. Por meio de seus membros, ela desenvolve padrões internacionais aplicáveis em diversas áreas da atividade humana. Pesquise mais sobre a ISO em seu site oficial: <http://www.iso.org/iso/home/about.htm>. Acesso em: 14 jan. 2016. Site em inglês ou no Portal Educação: <http://www.portaleducacao.com.br/administracao/artigos/40732/a-historia-da-organizacao-iso#!1>. Acesso em: 28 jan. 2016.

Para que você possa adquirir preparo para superar o desafio proposto, vale conhecer melhor algumas das características de qualidade mencionadas na tabela 3.1.

1. Adequação funcional: antes conhecida como funcionalidade apenas; ela se refere à existência de um conjunto de funções que satisfazem às necessidades previamente estabelecidas, quando o produto é usado sob condições específicas. Duas de suas subcaracterísticas são a Completude Funcional (o *software* apresenta todas as funções necessárias ao usuário?) e a Corretude Funcional ou Acurácia (o *software* gera dados e consultas corretos segundo o que foi definido?) (WAZLAWICK, 2013).

2. Confiabilidade: se um *software* é capaz de manter comportamento consistente com o que se espera dele ao longo do tempo, então ele pode ser considerado confiável. A confiabilidade tem a ver com o funcionamento do programa em situações incomuns. Ela pode ser medida diretamente e estimada usando-se dados históricos e de desenvolvimento, ou seja, um computador poderá ser considerado

livre de falhas quando não tiver alguma incidência em um determinado ambiente e num determinado período (MUSA et al., 1987 apud PRESSMAN, 1995). Desta definição, o que ainda não temos claro é o que significa *falha*. Quando tratarmos de teste de *software* com mais detalhes, esse conceito será abordado.

Vale aqui também destacar duas de suas subcaracterísticas: a Disponibilidade (avalia o quanto o *software* está operacional e livre para uso quando necessário) e a Tolerância a Falhas (avalia a forma como o *software* reage em situação anormal).

Reflita

Qual é o real interesse de um usuário? Quando ele pensa em qualidade do *software*, em geral, ele se lembra da confiabilidade. No caso de o produto já ser relativamente confiável, a usabilidade é que fará diferença nele.

3. Usabilidade: de forma simplificada, podemos entender essa característica como a facilidade em se usar um programa, do ponto de vista do usuário. Em linhas gerais, o programa é fácil de usar se ele é (REISS, 2012):

• Funcional – ele realmente funciona?

• Responsivo – ele me fornece respostas adequadas?

• Ergonômico – eu posso facilmente ver, clicar, arrastar e girar as coisas?

• Conveniente – tudo está bem onde eu preciso que esteja?

• "À prova de tolos" – o projetista me ajuda a não cometer erros ou quebrar coisas?

A usabilidade também apresenta subcaracterísticas, incluindo (WAZLAWICK, 2005):

• Operabilidade – o produto é fácil de usar e controlar?

• Proteção contra erro do usuário – o programa consegue evitar que o usuário cometa erros?

• Acessibilidade – avalia o grau em que o produto foi projetado para atender usuários com necessidades especiais.

Assimile

"Não é possível conceber um processo de garantia de qualidade de um *software* sem integrá-lo com o ciclo de desenvolvimento de *software*." (BARTIÉ, 2002, p. 35)

4. Segurança: se um programa consegue proteger os dados e as funções de acessos não autorizados, então ele tem um bom nível de segurança. Algumas de suas subcaracterísticas devem ser destacadas. Entre elas:

• Confidencialidade – mede o grau em que os dados e funções ficam disponíveis para quem, de fato, tem autorização para acessá-los;

• Rastreabilidade de uso – mede o grau em que as ações realizadas por uma pessoa ou por um sistema podem ser rastreadas de forma a ser efetivamente comprovado que, de fato, foi essa pessoa ou sistema que realizou tais ações.

5. Capacidade de manutenção: trata de uma característica que atrai interesse direto apenas da equipe de desenvolvimento, já que não afeta a percepção do usuário em relação ao sistema. Como você certamente já inferiu, trata-se da capacidade do sistema em passar por manutenção. Assim como todas as outras características apresentadas, essa também conta com divisões. Vamos a elas:

• Modularidade – o sistema é bem dividido em módulos? Mudanças em um dos módulos deve causar mínimo impacto nos outros.

• Reusabilidade – há partes do sistema que podem ser usadas na construção de outro sistema?

• Analisabilidade – o sistema permite que se faça depuração com facilidade?

Exemplificando

Ainda em dúvida sobre a importância da qualidade em um produto? Em 4 de junho de 1996, o Foguete Ariane 5, construído pela empreiteira EADS SPACE *Transportation*, explodiu 37 segundos após seu lançamento. A comissão responsável pela apuração dos fatos concluiu que houve erro no Sistema Referencial Inercial (SRI). Do relatório, consta o seguinte: "A

anomalia interna de *software* do SRI ocorreu durante a execução de uma conversão de dados de um número de 64 bits em ponto flutuante para um inteiro de 16 bits com sinal. O valor do número em ponto flutuante era maior do que poderia ser representado pelo inteiro de 16 bits com sinal. O resultado foi um operando inválido. A instrução de conversão de dados não estava protegida contra erros de operando".

Disponível em: <http://www.ime.uerj.br/~demoura/Especializ/Ariane/>. Acesso em: 17 jan. 2015.

No âmbito das características de qualidade do *software* em uso, vale destacar (WAZLAWICK, 2013):

6. Efetividade: capacidade que o produto tem de proporcionar ao cliente o atingimento de seus objetivos de negócio.

7. Satisfação: capacidade que o produto tem de satisfazer o cliente em ambiente de uso. Pode ser dividida em Utilidade, Prazer, Conforto e Confiança.

8. Uso sem riscos: o uso do *software* não pode implicar riscos para pessoas, negócios ou ambiente. Divide-se em mitigação do risco econômico, mitigação do risco ambiental e mitigação do risco à saúde e segurança.

Faça você mesmo

O método *Cleanroom* (sala limpa) é uma maneira bastante difundida de se desenvolver *softwares* com qualidade. O método foi publicado pela primeira vez em 1987 por Mills, Dyer e Linger, e tem como princípio básico que os programas devem ser vistos como regras para funções ou relações matemáticas. A atividade aqui proposta é a criação de um relatório de uma página contendo os princípios básicos do modelo. Vale a pena conhecê-lo.

Na sequência, nosso desafio será retomado e uma solução viável para ele será discutida.

Sem medo de errar

Os ventos da mudança passaram novamente pela *XAX-Sooft* carregando o desejo – e a real necessidade – de implantação de procedimento formal de qualidade na empresa. A primeira

providência tomada foi o levantamento do que já tem sido feito em prol da qualidade. Convenhamos, nada melhor que iniciar uma mudança entendendo como as coisas funcionam atualmente.

No entanto, apenas essa providência não basta. Mesmo que em pequenos passos, a implantação da qualidade demanda ações contínuas, tanto processuais como culturais. Ela requer a conscientização de que sempre haverá por onde melhorar um produto ou um processo e que a excelência é um bem que, embora intangível, produz efeitos palpáveis e duradouros no crescimento e na boa imagem da empresa.

Pois bem, o desafio agora é tratar da avaliação dos fatores que afetam a qualidade dos produtos da *XAX-Sooft*. Com base no conteúdo estudado nesta seção, você deve produzir relatório contendo a sua avaliação de algumas características de um software produzido pela *XAX-Sooft*. Como esse software não chegou a ser efetivamente construído, devemos considerar que as avaliações devam ser apenas estimadas, de acordo com os critérios que definem cada uma delas. Vejamos:

1. O relatório deve ser iniciado com a descrição de seu objetivo, que é o de avaliar as principais características de qualidade de um programa produzido pela empresa.

2. Na sequência, você deverá descrever a avaliação das seguintes características do sistema, buscando responder a perguntas relacionadas a cada uma delas:

2.1. Capacidade de manutenção: o programa favorece a manutenção? O sistema possui boa divisão de módulos? É fácil de ser testado?

2.2. Segurança: o programa permite acesso apenas de pessoas autorizadas? Ele oferece condições de rastrear seu uso?

2.3. Usabilidade: o programa é fácil de ser usado? Sua operação é de fácil assimilação?

2.4. Adequação funcional: o software apresenta todas as funções necessárias ao usuário? Ele gera dados e consultas corretos segundo o que foi definido?

Com esse levantamento, você terá um bom panorama da qualidade do produto e poderá direcionar esforços de melhoria com mais precisão.

Atenção

A característica conhecida como "Uso sem riscos" não deve ser confundida com a característica "Segurança do software".

Lembre-se

Alguns dos objetivos das Revisões Técnicas Formais são (i) descobrir erros de implementação em qualquer representação do software; (ii) verificar se o software atende seus requisitos e (iii) garantir que o software tenha sido representado de acordo com os padrões definidos (PRESSMAN, 1995).

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 as compare com as de seus colegas.	
Melhorando a capacidade de manutenção dos programas	
1. Competência geral	Competência para conhecer e conduzir processos de qualidade de *software*.
2. Objetivos de aprendizagem	O objetivo desta seção é abordar os fatores que afetam a qualidade e as ações de Garantia da Qualidade do *Software* (SQA).
3. Conteúdos relacionados	Garantia da qualidade de *software*. Fatores que afetam a qualidade de um *software*.
4. Descrição da SP	A Mimimi-Soft, empresa desenvolvedora de soluções de *software* para consultório médico, atingiu bons níveis de qualidade em seus programas e, com isso, tem obtido bons níveis de satisfação de seus clientes. Em seus produtos, as características que afetam diretamente a experiência e as expectativas do usuário têm sido muito bem avaliadas. Contudo, qualquer necessidade de alteração ou aprimoramento nos produtos tem causado preocupação e trabalho excessivo à equipe, pois aparentemente eles não têm sido bem preparados para sofrer manutenção. O desafio aqui é indicar ações que aumentem a capacidade de manutenção dos próximos sistemas produzidos pela Mimimi-Soft, tornando-os mais flexíveis e aptos para sofrerem mudanças. Consulte a Seção 1.3 da Unidade 1 para relembrar os conceitos de Coesão e Acoplamento e separação de dados e procedimentos, importantes na resolução deste desafio.

5. Resolução da SP	A solução do caso demanda as possíveis providências: 1. Aprimoramento da modularidade do *software*. A divisão lógica dos componentes que executam as funções deve ser feita de forma a se obter alta coesão e baixo acoplamento entre os módulos; 2. O projeto deverá conter representação distinta de dados e procedimentos; 3. Criação ou aprimoramento das interfaces que reduzam a complexidade das conexões entre módulos e o ambiente externo; 4. A reutilização de componentes já testados em outras aplicações deve ser fortemente considerada.

Lembre-se

"O teste de *software* é uma das atividades de validação e verificação e consiste na análise dinâmica do mesmo, ou seja, na execução do produto de *software* com o objetivo de verificar a presença de defeitos no produto e aumentar a confiança de que o produto esteja correto." (ROCHA; MALDONADO; WEBER, 2001, p. 73)

Faça você mesmo

É possível avaliar a qualidade de um *software* se o cliente ficar mudando de ideia sobre o que espera que o *software* faça? Sintetize sua opinião e discuta-a com seus colegas de sala.

Faça valer a pena

1. Em relação aos fatores que influenciam a qualidade de um *software*, analise as afirmações que seguem:

I - Pode-se considerar que um dos fatores que diretamente afetam o produto é a prática exercida pela equipe de se reunir mais do que uma vez por dia.

II - A característica de segurança de um sistema está ligada ao seu uso seguro.

III - A satisfação é um item de qualidade que se mede pelo tempo que o usuário permanece operando o sistema.

IV - Os parâmetros de qualidade são divididos entre os que possuem mais afinidade com o processo e com o produto.

É verdadeiro o que se afirma apenas em:
a) I e IV.

b) II e III.

c) II.

d) IV.

e) II e IV.

2. Assinale a alternativa que contém apenas expressões relacionadas aos fatores que influenciam na qualidade do *software*.

a) Quantidade de linhas de código, compactação do sistema.

b) Conjunto de funções que satisfazem a necessidades previamente estabelecidas, probabilidade de operação livre de falhas de um programa.

c) Disponibilidade de funções substitutas, facilidade de manutenção.

d) Existência de procedimento formal de comercialização do sistema, contratação de profissional de programação com experiência comprovada.

e) Alteração do comportamento estável durante o uso, quantidade de linhas de código.

3. No contexto da usabilidade de um *software*, assinale a alternativa que contém expressões que completam corretamente as lacunas na frase abaixo.

"A usabilidade avalia o grau no qual o produto tem atributos por meio dos quais possa ser _____, _____, usado e que seja atraente ao usuário. Em específico, a _____ avalia o grau no qual o produto é fácil de usar e controlar. A _____ deve proporcionar prazer e uma interação satisfatória."

a) projetado, testado, acessibilidade, rapidez.

b) entendido, aprovado, inteligibilidade, praticidade.

c) visualizado, rastreado, operabilidade, experiência.

d) simulado, prototipado, informalidade, coloração.

e) entendido, aprendido, operabilidade, interface com o usuário.

Seção 3.3

As normas de qualidade aplicadas no desenvolvimento de *software*

Diálogo aberto

Olá! Seja bem-vindo a mais esta seção.

Já discutimos na seção anterior que a percepção sobre o que se costuma chamar de qualidade de um programa pode – e de fato irá – variar de usuário para usuário. No entanto, como você pode imaginar, apenas a subjetividade não é suficiente para a construção de base sólida para a realização de avaliações precisas de um programa. Para que se possa distinguir um bom produto de outro que ainda pode ser melhorado, é necessário o estabelecimento de normas gerais e objetivas de qualidade, universalmente aceitas e capazes de permitir comparações confiáveis entre características correspondentes dos programas.

Depois de levantar as ações que eram em dado momento executadas em prol da qualidade e de avaliar algumas características de seus programas, a *XAX-Sooft* agora foca seus esforços na adoção de uma norma de qualidade que seja capaz de estabelecer parâmetros seguros de avaliação e de proporcionar à empresa o necessário reconhecimento como uma desenvolvedora com altos níveis de excelência.

O desafio que lhe é proposto nesta seção é o de criar um relatório contendo o processo de escolha e implantação de norma de qualidade na *XAX-Sooft*. Para que você possa contar com a base teórica necessária para superá-lo, as próximas páginas trarão os fundamentos das principais e mais aceitas normas de qualidade de *software*, o que dará suporte para a escolha daquela a ser implantada na organização.

Por meio do conhecimento adquirido com essa prática, você terá condições para selecionar o modelo (ou norma) de qualidade que mais se adequa à realidade da sua organização, tornando-a tão bem-sucedida quanto possível na arte de desenvolver programas de computador.

Que o caminho para a escolha do modelo comece a ser trilhado.

Boa sorte e bom trabalho!

Não pode faltar

Em quais parâmetros podemos confiar na hora de escolhermos um modelo de qualidade? Qual instituição cria os padrões, os divulga e os mantém? Há uma norma melhor que a outra? Esta seção do seu livro didático aborda essas questões e alguns dos mais bem-conceituados modelos de qualidade, com destaque para seus pontos fortes e aplicações. Antes de nos debruçarmos sobre eles, vale a pena sabermos um pouco mais sobre a organização que os cria e mantém.

Em Genebra, cidade da Suíça, está localizado o escritório central de uma organização independente e não governamental, com membros em 162 países e que desde fevereiro de 1947 já publicou mais de 20.500 padrões internacionais que cobrem quase todos os aspectos dos ramos da atividade humana, tais como saúde, agricultura, manufatura e, é claro, tecnologia. Essa organização é mundialmente conhecida como ISO, sigla da *International Organization for Standardization*. Por conta da existência de diferentes acrônimos (ou siglas) em línguas diferentes, tais como IOS em inglês e OIN em francês, os fundadores resolveram adotar simplesmente ISO, palavra derivada do grego *isos*, que significa igual. Disponível em: <http://www.iso.org/iso/home/about.htm>. Acesso em: 23 jan. 2016.

Estudaremos na sequência alguns dos principais padrões de qualidade utilizados mundo afora. Eles darão a você base para superar o desafio proposto.

ISO 9001 – Sistema de Gestão da Qualidade

A ISO 9001 é um dos mais conhecidos e utilizados padrões mundiais de qualidade. Atualizado no ano de 2015, ele especifica requisitos para um sistema de gestão de qualidade, com foco naquilo que o cliente exige para que o produto ou serviço seja entregue de acordo com suas necessidades (SEEAR, 2015).

Esse padrão é implantado quando uma organização precisa demonstrar sua capacidade de fornecer produtos e serviços que atendam às exigências de regulamento e estatuto da organização e que pretende aumentar a satisfação do cliente por meio da aplicação eficaz do sistema. O aspecto interessante desse padrão é que todos os

seus requisitos são genéricos e se destinam a aplicação em qualquer organização, independentemente da sua natureza ou tamanho, ou dos serviços ou produtos que disponibiliza. (Disponível em: <http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=62085>. Acesso em: 23 jan. 2016). Não acha que é disso que a *XAX-Sooft* precisa?

Pesquise mais

A ISO 9001 faz parte da família de normas ISO 9000 e, como todas elas, tem passado por atualizações regulares. Rocha, Maldonado e Weber (2001), em sua obra "Qualidade de *Software*: Teoria e Prática", oferecem bom panorama das atualizações recentes, exceto a do ano de 2015. Esta última, aliás, pode ser conhecida em: <http://www.qualityconsult.com.br/index.php/iso-9001-versao-2015/>. Acesso em: 23 jan. 2016.

A ISO 9001:2015 adota uma abordagem de processo para desenvolvimento, implementação e melhoria da eficácia de um sistema de gestão da qualidade, com o objetivo de aumentar a satisfação do cliente por meio do atendimento aos seus requisitos. Para isso, ela se baseia em 7 princípios de gerenciamento de qualidade: Foco no Cliente, Liderança, Engajamento de Pessoas, Abordagem de Processo, Melhoria, Tomada de Decisão Baseada em Evidências, Gestão de Relacionamento.

Esses termos, se considerados de forma isolada, podem não transmitir por completo o "espírito" do padrão. No entanto, pense no foco no cliente, na liderança e nas outras expressões num contexto de processo de qualidade no qual a satisfação dos requisitos colocados pelo cliente é prioridade, a participação efetiva das pessoas é fundamental e a melhoria – seja qual for o produto – é objetivo constante.

ISO/IEC 90003 – Orientações para Qualidade de Processo de *Software*

Nosso último item de estudo tratou de aspectos gerais da ISO 9001 e a colocou como um padrão geral para gestão da qualidade. A ISO/IEC 90003:2014 fornece orientações para aplicação da ISO 9001 nos processos de aquisição, fornecimento, desenvolvimento, operação e manutenção de um programa de computador e serviços de suporte relacionados.

A aplicação desse padrão é apropriada para produtos de *software* que são parte de um contrato comercial com uma outra organização, para produtos disponíveis para um segmento de mercado, produtos usados para dar suporte a um processo em uma empresa, produtos incorporados em um *hardware* específico ou produtos relacionados a um serviço de *software*. Disponível em: <http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=66240>. Acesso em: 24 jan. 2016. Ou seja, trata-se de um padrão de qualidade aplicável a uma gama considerável de programas.

Reflita

A implantação de um padrão de qualidade em uma empresa pequena é necessária? Alguns argumentos talvez ajudem na resposta dessa questão: conquista de novos mercados, criação de condições para o surgimento de clima de incentivo, motivação e participação, direcionamento da empresa para o cliente e até redução de custos.

O padrão ISO/IEC 90003:2014 não está atrelado à tecnologia, modelos de ciclo de vida, processos de desenvolvimento, sequência de atividades e estrutura organizacional.

Que tal um modelo que ofereça melhorias no gerenciamento do processo de *software* e conduza à produção de programas mais confiáveis? Sigamos adiante.

CMMI

O CMMI, sigla de *Capability Maturity Model Integration* é o sucessor do CMM, mantido pela SEI (*Software Engineering Institute*) de 1987 até 1997. Trata-se de um modelo que visa aprimorar a capacidade da maturidade do processo de *software*. Para melhor entendê-lo, voltemos à nossa primeira unidade. Na segunda seção, conceituamos assim processo: sequência de passos que visam a produção e manutenção de um *software* e que se inter-relacionam com recursos (humanos e materiais), com padrões, com entradas e saídas e com a própria estrutura da organização. Pois bem, a capacidade e maturidade de um processo remete à noção do grau de qualidade com o qual um processo atinge um resultado esperado. Disponível em: <http://www.devmedia.com.br/cmmi-uma-visao-geral/25425#ixzz3yBwjcLs1>. Acesso em: 24 jan. 2016.

A versão atual do CMMI, a 1.3, possui três divisões:

• CMMI-ACQ: modelo que foca em atividades de gerenciamento da aquisição de produtos e serviços de *software*.

• CMMI-DEV: criado para o desenvolvimento de produtos e serviços de qualidade.

• CMMI-SVC: modelo que tem o objetivo de fornecer serviços de qualidade para clientes e usuários finais.

O CMMI é um caminho de melhoramento evolucionário, trilhado em 5 níveis de maturidade e 4 níveis de capacidade, feito para que organizações de *software* possam sair de um processo de *software* imaturo para um processo maduro e disciplinado. A tabela 3.2 mostra esses níveis:

Tabela 3.2 | Níveis de capacidade e maturidade do CMMI

Nível	Capacidade	Maturidade
0	Incompleto	-
1	Realizado	Inicial
2	Gerenciado	Gerenciado
3	Definido	Definido
4	-	Quantitativamente gerenciado
5	-	Em otimização

Fonte: Wazlawick (2013, p. 506).

Segundo a descrição do modelo, a qualidade é influenciada pelo processo e seu foco é melhorar o processo de uma organização.

Existem duas estruturas do CMMI: em estágios e contínua. Embora a diferença seja sutil, ela é significativa. A representação (ou estruturação) em estágios usa níveis de maturidade para caracterizar o estado geral dos processos da organização em relação ao modelo como um todo, enquanto a representação contínua utiliza níveis de capacidade para caracterizar o estado dos processos da organização em relação a uma área de processo individual. Em outras palavras, a representação por estágios é aplicada à organização como um todo e permite que se compare à maturidade de diferentes organizações. Já a representação contínua é projetada para permitir à empresa focar em processos específicos que deseja melhorar em função de suas prioridades (WAZLAWICK, 2013).

Assimile

O CMM foi de fato criado pela SEI, que é o Instituto de Engenharia de *Software* da Carnegie Mellon University, mas contou com a participação do governo e da indústria norte-americana. Atualmente o CMMI tem sido mantido pelo CMMI Institute, que também passou a ser responsável pelo treinamento e certificação no modelo.

A essência do modelo é a divisão da capacidade e da maturidade em níveis, conforme mostrado na tabela 3.2. Vejamos os níveis em detalhes. Disponível em: <http://cmmiinstitute.com/system/files/models/CMMI_for_Development_v1.3.pdf>. Acesso em: 24 jan. 2016.

Níveis de Capacidade do Processo: um nível de capacidade do processo é alcançado quando todos os objetivos genéricos para aquele nível são satisfeitos.

Nível 0 – Incompleto: um processo incompleto é um processo que não está sendo colocado em prática ou que é usado apenas parcialmente. Um ou mais objetivos específicos da área de processo não estão sendo satisfeitos e inexistem metas genéricas para serem alcançadas, daí não haver razão para tornar oficial um processo parcialmente executado.

Nível 1 – Realizado: este nível é caracterizado como um processo que está sendo seguido, é capaz de gerar produtos, é viável no atingimento de metas específicas e já pode ser considerado como um fator de melhorias na organização. No entanto, tais melhorias podem ser perdidas ao longo do tempo, já que o processo ainda não foi institucionalizado.

Nível 2 – Gerenciado: um processo no nível 2 já é planejado e executado de acordo com a política definida, emprega pessoas qualificadas que tenham recursos adequados para produzir saídas controladas; envolve as partes interessadas (os *stakeholders*), é monitorado, controlado e avaliado. A disciplina do processo refletido por este nível ajuda a garantir que as práticas existentes são mantidas durante períodos de estresse.

Nível 3 – Definido: trata-se de um processo gerenciado que é feito sob medida a partir do conjunto de processos padrão e diretrizes da organização. Este nível também se caracteriza por manter registros de descrição do processo. No nível de capacidade 2 (Gerenciado), as

normas, as descrições de processos e os procedimentos podem ser bastante diferentes em cada instância específica do processo. No nível de capacidade 3, as normas, descrições de processos e procedimentos para um projeto são feitos sob medida a partir do conjunto de processos padrão da organização para atender um determinado projeto ou unidade organizacional e, portanto, são mais consistentes.

Outra distinção importante é que, no nível de capacidade 3, processos são geralmente descritos de forma mais rigorosa do que no nível 2. Um processo definido estabelece claramente a finalidade, os insumos, os critérios de entrada, as atividades, os papéis de cada membro, medidas, saídas e critérios de saída. Enfim, no nível 3, os processos são geridos de forma mais proativa e disciplinada.

Exemplificando

Uma boa indicação de como o CMMI tem sido bem aceito é dada por sua utilização em grandes organizações. No link: <http://cmmiinstitute.com/who-uses-cmmi>, acesso em: 24 jan. 2016, você poderá conferir que NASA, Siemens, Bosch, Motorola e IBM, entre outras empresas de ponta, adotaram o modelo em seus processos.Em Língua Portuguesa, outro bom exemplo de como algumas práticas do CMMI podem ser aplicadas em pequenas e médias empresas de *software*, sem que o negócio se torne financeiramente inviável, pode ser encontrado em: <http://www.techoje.com.br/site/techoje/categoria/detalhe_artigo/1734>. Acesso em: 5 fev. 2016.

Outros bons exemplos dos benefícios trazidos pela adoção do CMMI são demonstrados em documento preparado pela SEI (*Software Engineering Institute* ou Instituto de Engenharia de *Software*) e disponibilizado em: <http://resources.sei.cmu.edu/asset_files/SpecialReport/2003_003_001_14117.pdf>. Acesso em: 5 fev. 2016. O relatório aborda 12 casos que apontam melhorias obtidas por empresas em suas medidas de desempenho em custo, cronograma, qualidade, satisfação do cliente e no retorno do investimento. Tomemos como exemplo as melhorias obtidas nas medidas de satisfação do cliente e aprimoramento da qualidade, expressas nos seguintes indicadores:

1. Aumento de 55% nas vitórias em concorrências públicas em comparação com CMMI nível 2 de maturidade por parte da Lockheed Martin;

2. Redução de defeitos encontrados de 6,6 para 2,1 em cada mil linhas de código, por parte da Northrop Grumman;

3. Mais de US$ 2 milhões de dólares economizados por causa da detecção e remoção prematura de defeitos no código, por parte da Sanchez Computer Associates, Inc.

Níveis de Maturidade: para dar suporte à representação por estágios, o CMMI contempla níveis de maturidade em sua concepção. Um nível de maturidade é composto por práticas específicas e genéricas relacionadas para um conjunto predefinido de áreas de processos que melhoram o desempenho global da organização. O nível de maturidade fornece uma maneira de caracterizar o seu desempenho da organização. Você sabe: quanto maior o nível de maturidade, mais organizada em seus processos a organização é. Em consequência, a tendência de gerar produtos de qualidade avançada é maior. Os níveis são o seguinte:

Nível 1 – Inicial: o processo de *software* é caracterizado como caótico. Poucos processos são definidos e o sucesso depende de esforços individuais. A organização não provê um ambiente estável para o desenvolvimento e manutenção do *software* e cronogramas e orçamentos são frequentemente abandonados por não serem baseados em estimativas próximas da realidade.

Nível 2 e Nível 3 – Gerenciado e Definido: as características dos níveis de maturidade 2 e 3 são semelhantes às dos níveis de capacidade 2 e 3.

Nível 4 – Quantitativamente gerenciado: nesse nível, a organização estabelece metas de qualidade e usa essas medidas na gestão de seus projetos. A qualidade de processo e de produto é entendida por meios estatísticos e gerenciada de forma que seja quantitativamente previsível.

Nível 5 – Em otimização: nesse nível, a organização melhora continuamente seus processos, com base nas medições quantitativas já obtidas (WAZLAWICK, 2013).

MPS.BR

O MPS.BR ou MR-MPS (Modelo de Referência para Melhoria do Processo de *Software*) é um modelo de avaliação voltado às empresas brasileiras, criado no ano de 2003 pela SOFTEX, em parceria com o Governo Federal e pesquisadores. Sua concepção leva em consideração normas e modelos internacionalmente reconhecidos, além das indispensáveis boas práticas da Engenharia de *Software* e a realidade de negócio dos desenvolvedores nacionais. Sua criação se justifica, inclusive, pelos altos custos de certificação nas normas internacionais.

O guia de implementação da norma se divide em 11 partes, sendo

as 7 primeiras relacionadas aos 7 níveis de maturidade implantados no MPS.BR. As demais são guias de implementação nas empresas.

Esse modelo também é formado por níveis: Em Otimização, Gerenciado Quantitativamente, Definido, Largamente Definido, Parcialmente Definido, Gerenciado, Parcialmente Gerenciado.

Em maio de 2015, haviam sido feitas 596 avaliações de *software* pela MPS.BR, por 13 instituições avaliadoras (WAZLAWICK, 2013). Disponível em: <http://www.softex.br/mpsbr/mps/mps-br-em-numeros/>. Acesso em: 24 jan. 2016.

Faça você mesmo

A norma ISO/IEC 15504, também conhecida como SPICE, foi criada para orientar a avaliação da capacidade das empresas nos processos. Trata-se de padrão bastante utilizado, que vale a pena ser conhecido. A prática aqui proposta é o levantamento e síntese, em relatório, da estrutura da SPICE e dos seus aspectos mais importantes.

Sem medo de errar

Por conta das atividades que você desenvolveu nas duas últimas seções, a *XAX-Sooft* já conseguiu obter o registro das eventuais ações em favor da qualidade que a equipe tem executado e o relatório de avaliação de algumas características de um produto seu. O momento agora demanda empenho para a escolha e implantação de um modelo de qualidade do processo que seja economicamente viável, bem estruturado, reconhecido pelo mercado e que tenha boa adequação à cultura da *XAX-Sooft*. Para que essa escolha seja feita com acerto, você deve recorrer ao conteúdo abordado na seção "Não pode faltar".

Conforme anteriormente colocado, o desafio inclui a criação de relatório contendo o processo de implantação de norma de qualidade na *XAX-Sooft*, que vai desde os critérios para sua escolha, passa pelos objetivos da implantação e termina no planejamento dos passos que deverão ser tomados para sua efetiva adoção.

A indicação do conteúdo desse relatório vem na sequência.

1. O relatório deve ter início com a exposição dos objetivos da implantação de um modelo de qualidade.

2. O segundo item deve revelar a escolha do modelo e os motivos

que levaram a ela. Questões como "a implantação consumirá muito recurso financeiro?", "trata-se de um modelo bem aceito no mercado?", "ele é adequado à realidade da nossa empresa?" deverão ser abordadas neste item.

3. Por fim, o relatório deverá identificar como o modelo será implementado. Aqui, as questões relacionadas à contratação de profissionais externos, disponibilização de treinamento externo à equipe, período de implantação e objetivos a serem alcançados deverão ser abordadas.

Atenção

A ISO 90003 é uma versão mais atual da antiga norma ISO 9000-3:1997, que era um guia para aplicação da ISO 9001 a empresas de *software*. Um dos problemas com a 9000-3 é que ela não tratava a melhoria contínua do processo, mas apenas indicava os processos que as empresas deveriam manter. A 90003 corrigiu tal deficiência (WAZLAWICK, 2013).

Lembre-se

Você encontrará farta documentação sobre o modelo MPS.BR em: <http://www.softex.br/mpsbr/>. Acesso em: 25 jan. 2016. Essa documentação inclui guia geral do modelo para *software* e guias de avaliação, entre outros.

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 as compare com as de seus colegas.	
Decidindo pelo padrão de qualidade adequado	
1. Competência geral	Conhecer as principais metodologias de desenvolvimento de *software*, normas de qualidade e processos de teste de *software*.
2. Objetivos de aprendizagem	O objetivo desta seção é apresentar as normas de qualidade mais referenciadas.
3. Conteúdos relacionados	Introdução às normas de qualidade de *software*: ISO 9001 – Sistema de Gestão da Qualidade, ISO/IEC 90003 – Orientações para *software*, ISO/IEC 25010:2011 – Qualidade de Produto de *software* e CMMI – *Capability Maturity Model Integration*, MPS.BR.

4. Descrição da SP	Conforme relatado na última seção, a *XAX-Sooft* tem experimentado bons níveis de excelência em seus produtos e, com isso, altos índices de satisfação de seus clientes. Há, no entanto, um novo desafio no horizonte: para ser aceita em licitação pública para desenvolvimento de *software* para o Governo Federal, a *Mimimi-Soft* deverá apresentar avaliação MPS-BR no ato da inscrição para o processo seletivo. Sua missão é relatar os passos a serem dados e especificar o órgão competente a ser consultado para que a avaliação seja obtida no menor tempo possível.
5. Resolução da SP	A solução do caso demanda as possíveis providências: 1. Um bom ponto de partida para a obtenção de informações gerais sobre a avaliação pode ser obtido em: <http://www.softex.br/a-softex/>, acesso em: 5 fev. 2016. Você encontrará nessa página informações sobre a SOFTEX, associação que busca promover a excelência do *software* brasileiro. 2. Na sequência, você poderá conseguir informações mais específicas sobre o processo em: <http://www.softex.br/mpsbr/guias/>. Acesso em 5 de fevereiro de 2016. Nessa página você encontrará Guia Geral de Serviços, Guia Geral de *Software* e o Guia de Avaliação, bases importantes para formar conhecimento sobre a avaliação. 3. Para complementar as consultas e obter mais informações a respeito, procure contato com empresa que já adota o MPS.BR. As avaliações realizadas desde setembro de 2005 podem ser encontradas em <http://www.softex.br/mpsbr/avaliacoes/mps-sw/>, acesso em: 25 jan. 2016.

Lembre-se

O processo de *software* pode ser relatado como um conjunto de ações, atividades, procedimentos e transformações utilizadas para construção e manutenção de *software* (REZENDE, 2005).

Faça você mesmo

Faça o levantamento de qual o investimento real aproximado para a obtenção de avaliação CMMI e de uma avaliação MPS.BR.

Faça valer a pena

1. Em relação a ISO, analise as afirmações que seguem:

I - Trata-se de organização governamental e atrelada à indústria norte-americana, responsável por emitir especificamente normas de *software*.

II - ISO significa *International Software Orientation* e trata-se de organização de destaque no ensino de criação de *software*.

III - A ISO é uma organização independente e sem vínculo com governos, que desenvolve e publica padrões internacionais para vários ramos de atividade.

IV - A ISO treina e certifica profissionais para o desenvolvimento de projetos de *software*.
É verdadeiro o que se afirma apenas em:
a) II e IV.

b) II e III.

c) III.

d) IV.

e) I e IV.

2. Assinale a alternativa que contém apenas expressões diretamente relacionadas a norma ISO 9001.

a) Gestão de qualidade do produto e foco no cliente.

b) Baseada em 7 princípios de gerenciamento e destina-se à aplicação em qualquer organização.

c) Dependente do modelo de desenvolvimento de *software* e sem atualização desde sua criação.

d) Tomada de decisão baseada em evidências e gestão de qualidade do produto.

e) Criada pela ISO e mantida pela SOFTEX.

3. No contexto do CMMI, assinale a alternativa que contém expressões que completam corretamente as lacunas na frase abaixo.
"A representação _____ do CMMI usa níveis de maturidade para caracterizar o estado geral dos processos da organização em relação ao modelo como um todo, enquanto a representação _____ utiliza níveis de capacidade para caracterizar o estado dos processos da organização em relação a uma área de processo individual. O nível de capacidade _____ caracteriza-se por manter registros completos de descrição do processo. Estando no nível _____, uma organização já consegue conduzir processos monitorados, controlados e avaliados."

a) em estágios, contínua, definido, gerenciado.

b) complementar, contínua, indefinido, automatizado.

c) contínua, complementar, continuado, gerenciado.

d) em estágios, complementar, gerenciado, definido.

e) controlada, avançada, automatizado, definido.

Seção 3.4

As verificações necessárias na engenharia de *software*

Diálogo aberto

Seja bem-vindo à última seção desta unidade!

Depois de diagnosticar como a qualidade era tratada pela equipe, avaliar seus principais indicadores em um produto e implementar o modelo padrão consagrado, a *XAX-Sooft* mantém passo firme rumo à excelência em seus processos e na busca pela maior qualidade possível em seus produtos.

Como você está lembrado, o desafio da última seção foi justamente escolher o padrão de qualidade que a *XAX-Sooft* adotaria. No entanto, você ainda pode incluir providências na rotina *XAX-Sooft* que certamente implicarão retorno positivo à empresa. Que tal adotar formalmente um processo de detecção de defeitos rigorosos e bem definidos nos produtos? Será possível e viável quantificar algumas características do *software*, tal como complexidade funcional? Em outras palavras: conseguimos medir alguns atributos do programa? Entenda atributos como, por exemplo, número de funções e tamanho do código. Não é difícil imaginar o benefício que tais ações trariam aos produtos.

Pois bem, é hora de formalizar os procedimentos de qualidade do produto. Sua missão é justamente criar relatório contendo o processo de implantação de inspeções, medições e métricas de *software*. Dessa forma, poderá conhecer e conduzir processos de qualidade de *software*, aplicados a um projeto de desenvolvimento de *software*.

Conforme já tratamos, uma das consequências direta dessas ações é melhoria da qualidade do produto. Mas é certo que, indiretamente, ela é obtida pelo aprimoramento das estimativas de prazo, custo e esforço relacionadas ao produto. Estamos diante de algo a ser experimentado, não acha?

Bom trabalho e siga em frente com os estudos.

Não pode faltar

É muito difícil concebermos uma atividade desempenhada profissionalmente que não inclua maneiras de se medir o que é produzido. Essas medidas, sejam relacionadas a tamanho físico, a complexidade funcional ou a desempenho, visam assegurar algo indispensável em um ambiente produtivo: o controle. Nesta seção, você terá contato com o conceito de métrica e medição, com algumas formas de medições de um produto de *software*, além de revisões e inspeções. Com essas providências, a equipe deverá garantir e manter a qualidade de um *software*. Em frente!

Métrica e Medição

A medição é o processo pelo qual os números são atribuídos aos atributos de entidades do mundo real. Na medição, atribui-se um valor numérico a uma grandeza física. Por exemplo, quando medimos a distância entre dois pontos e obtemos 5 metros, estamos atribuindo o valor 5 à grandeza física chamada distância. Usaremos o termo medição tanto para descrever um processo como um valor de atributo. A **medição** é tipicamente uma quantificação direta, que envolve um único valor, ao passo que **métrica** é uma quantificação indireta, que envolve o cálculo e o uso de mais de uma medida. Vamos a um exemplo de métrica: considerando a medida de número de linhas de código de um programa e a medida de número de defeitos encontrados em todo o programa, podemos estabelecer a métrica de quantidade de defeitos por linha de código.

É desejável que as métricas sejam capazes de fornecer informação relevante para a tomada de decisão e para a comparação de desempenhos. É necessário que você tenha em mente um fato importante: existem métricas referenciais, usadas normalmente de forma padronizada pelos desenvolvedores. No entanto, uma métrica pode ser baseada no objetivo da organização e na sua necessidade de informação para a tomada de uma decisão.

Devemos considerar também que uma métrica deve ser calculada com facilidade, que ela tenha condições de ser repetida quantas vezes forem necessárias, que sua unidade seja compreensível e universal e que, por fim, seu processamento possa ser automatizado.

Exemplificando

Observe este exemplo do uso de métrica. Se você quer comprar um carro – e procura por eficiência – você não levará em conta a quantidade de quilômetros que ele é capaz de rodar, nem a quantidade de combustível que ele consegue armazenar no tanque. Na verdade, o que você quer saber é quanto ele consegue rodar com um litro de combustível. Assim, pelo uso de uma métrica, conseguimos chegar a um dado relevante e apto a te ajudar na tomada da sua decisão de compra do carro.

Imagine que dois projetos de *software* – com alguma similaridade funcional entre eles – estejam em construção. Se você quiser saber qual dos dois projetos está sendo mais produtivo, você deve realizar a **medição** do tamanho do projeto e do esforço para produzi-lo. Contudo, sem uma métrica, a comparação não seria viável, tampouco útil (MOORTHY, 2013).

No âmbito da construção de um *software*, as medidas podem ser categorizadas em (SWEBOK, 2004):

- **Medidas de Processo**: a expressão "medida de processo" que usamos aqui significa a coleta, análise e interpretação de informações quantitativas sobre o processo utilizado pelo desenvolvedor. A medição é usada para identificar os pontos fortes e deficiências do processo, além de avaliar o processo após sua implementação ou mudança. Por exemplo, a introdução de inspeções de *software* no processo pode reduzir o esforço para realização de testes. No entanto, pode haver aumento de tempo até a entrega do produto, caso as reuniões de inspeção sejam extensas demais. A decisão de se investir mais tempo nas inspeções do que nos testes, por exemplo, deverá ser tomada com base em métrica aplicada no processo.

- **Medidas de Produto**: as medidas de um produto incluem o tamanho do produto, sua estrutura e sua qualidade. Trataremos melhor dessas medidas e algumas das métricas correspondentes no decorrer desta seção. Outras duas medidas também devem ser consideradas neste contexto (MOORTHY, 2013):

- **Medidas de Projeto**: usadas para quantificar o desempenho de um projeto de *software*. Por exemplo, uma métrica comumente usada neste contexto é a porcentagem de variação no cronograma

do projeto, assim expressa: (data real de término – data planejada de término *100) / (data planejada de término – data real de início).

• **Medidas de Recursos**: usadas para quantificar a utilização de recursos em um projeto de *software*. Imagine que estejamos tratando de recursos humanos, ou seja, pessoas. Uma boa medida de efetividade seria dada pela quantidade de tarefas cumpridas por unidade de tempo, considerando tarefas de complexidade e duração semelhantes.

Pesquise mais

Como uma organização pode definir qual conjunto de métricas é adequado aos seus objetivos? Existe um modelo que ajude a definir e implantar as métricas? O GQM, acrônimo para *Goal/Question/Metric* (Objetivo/Questão/Métrica) é uma abordagem orientada a metas para a mensuração de processos e produtos de *software*. Pesquise mais em:

<http://www.inf.ufsc.br/~gresse/download/cits99.pdf>. Acesso em: 12 fev. 2016.

<http://www.cin.ufpe.br/~scbs/metricas/seminarios/GQM_texto.pdf>. Acesso em: 12 fev. 2016.

WAZLAWICK, R. S. **Engenharia de *Software***: conceitos e práticas. Rio de Janeiro: Elsiever, 2013, p. 247.

Já sabemos que, feitas as medições, podemos (e devemos) utilizá-las para gerar as métricas. Para fins de classificação, algumas métricas são geradas a partir de medidas obtidas diretamente, geralmente por contagem do atributo observado. Às métricas geradas damos o nome de **métricas diretas**. Outras métricas, porém, são obtidas indiretamente. A elas damos o nome de **métricas indiretas**.

Exemplificando

Exemplos de métricas diretas: custo, esforço, quantidade de linhas de código, quantidade de erros, velocidade de execução do programa, entre outras. Em relação às métricas indiretas, os exemplos mais referenciados são funcionalidade, confiabilidade e manutenibilidade.

Vale a pena analisarmos uma dessas métricas tomadas como exemplo. Na sequência, você conhecerá uma métrica de esforço, muito usada para medir o tamanho funcional de um *software* com o

objetivo de obter boa estimativa de custo, antes mesmo da sua efetiva construção.

Análise de pontos por função

Essa técnica se baseia nos requisitos do *software* para a obtenção da métrica. Por isso, ela é aplicável a partir do momento em que os requisitos funcionais do programa (ou as histórias) tenham sido definidos. Esses requisitos (ou funções – daí o nome de Pontos por Função) são convertidos em valores numéricos que, depois de calculados e ajustados, proverão excelente ideia do esforço necessário para desenvolver o sistema (WAZLAWICK, 2013).

Você deve se lembrar de que uma medida direta se baseia geralmente em contagem feita em algum atributo do sistema. Os atributos relevantes neste nosso contexto são os requisitos.

Devemos considerar que os resultados obtidos pela aplicação dessa métrica não se prestam apenas para estimar esforço de desenvolvimento. Dependendo da intenção de uso da métrica, a contagem dos pontos de função também pode ser usada para:

• Melhoria de um produto – neste caso, são contadas as funcionalidades alteradas, suprimidas ou adicionadas durante a manutenção adaptativa.

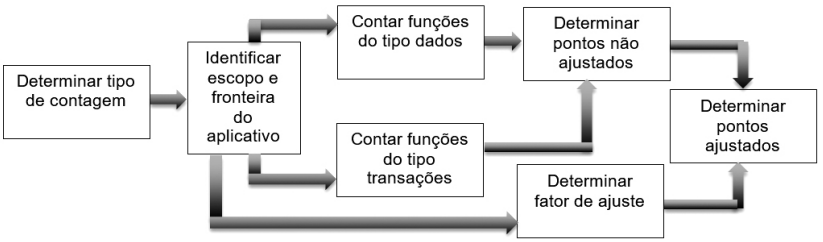
• Contagem de aplicação – técnica usada para contar pontos de aplicações já existentes.

Reflita

Quais as razões que justificam investimento de tempo e dinheiro para medir processos, produtos, projetos e recursos? Imagine que, por meio das medições, você poderá avaliar as atividades em curso, estimar em qual estágio tais atividades estarão no futuro e controlar o uso de recursos destinados ao projeto. Em outras palavras, as vantagens derivadas de um processo estruturado de medições poderão oferecer claras indicações sobre a qualidade do produto, servirão de avaliação da produtividade da equipe e determinarão se um novo método ou ferramenta implementados estão sendo efetivos.

Pois bem, falta-nos então conhecer o procedimento de contagem das funções e do cálculo da métrica. Antes de você chegar a qualquer explicação textual, observe a figura 3.2:

Figura 3.2 | Diagrama do processo de contagem de pontos de função



Fonte: Mecenas; Oliveira (2005, p. 4).

A fase de identificação da fronteira do aplicativo serve, por exemplo, para determinar se a contagem estará concentrada em um ou mais sistemas. Ela serve para estabelecer um divisor entre os componentes do aplicativo e os componentes de outro aplicativo.

Na sequência do processo, vem a contagem das funções do sistema. Vejamos o procedimento (MECENAS; OLIVEIRA, 2005):

• **Funções do tipo dados**: representam as necessidades de dados dos usuários, de acordo com sua visão de negócio. Divide-se em:

Arquivo Lógico Interno (ALI): trata-se de um elemento percebido pelo usuário e mantido internamente pelo sistema. Exemplos: arquivos de cadastro de clientes, cadastro de funcionários, arquivos de mensagens de auxílio e arquivos de mensagens de erros.

Arquivo de Interface Externa (AIE): representa as necessidades de dados externos à aplicação, ou seja, são dados armazenados fora da fronteira da aplicação, mas que não sofrem manutenção internamente.

• **Funções do tipo transação**: representam as funcionalidades de processamento de dados identificados pelos usuários. Existem três funções deste tipo:

Entrada Externa (EE): função que obtém dados informados pelo usuário ou por outra aplicação e os inserem no sistema. A função deve ter como objetivo armazenar, alterar ou remover dados no sistema. O nome de um cliente e seu endereço são exemplos de entradas externas.

Saída Externa (SE): função que obtém dados do sistema e apresentam ao cliente ou enviam a outras aplicações, sendo que pelo menos um valor obtido por cálculo deve existir para que seja considerada saída externa. Exemplo: fatura de um cliente, relação de clientes inadimplentes.

Consulta Externa (CE): função que apresenta dados da mesma forma que foram armazenados, sem cálculos ou transformações. Configura uma consulta externa, por exemplo, informar o CPF de uma pessoa ao sistema para se obter seus dados cadastrais completos.

Uma vez identificadas e contadas as funções, as quantidades apuradas são classificadas como complexidade alta, média e baixa, conforme a tabela 3.3:

Tabela 3.3 | Fatores de complexidade

	Complexidade Funcional		
Tipo de função	Baixa	Média	Alta
Entrada	3	4	6
Saída	4	5	7
Consulta	3	4	6
Arquivo interno	7	10	15
Arquivo externo	5	7	10

Fonte: Waslawick (2013, p. 161).

Assimile

Para efeito de contagem, um requisito do sistema equivale a uma função. No entanto, essa regra não deve ser tomada de forma absoluta. Apenas funções visíveis para o usuário devem ser consideradas. Um cálculo interno, por exemplo, não deve ser contado. Se apenas um requisito trata de cadastro de clientes e de produtos, então teremos aí duas funções. Em resumo, deve-se tomar os requisitos, eliminar os que são funções internas e subdividir aqueles que representam mais do que uma função (WAZLAWICK, 2013).

Determinadas as complexidades das funções do tipo dados e transação, a soma dos pontos obtidos é chamada de pontos de função não ajustado. Nessa etapa, já temos o tamanho funcional do aplicativo (MECENAS; OLIVEIRA, 2005).

Ocorre que, para que essa métrica possa ser útil de fato, há que se levar em consideração a real complexidade técnica dessas funções. Por exemplo, um sistema de controle de uma loja de locação de vídeo tende a possuir funções mais simples e estruturadas do que um sistema bancário. O método, então, prevê a aplicação de ajuste neste valor obtido em função de 14 características gerais do sistema, aplicáveis a todo o projeto. Seguem:

1) Comunicação de dados: em que grau a comunicação de dados é requerida?

2) Processamento de dados distribuído: em que grau o processamento distribuído está presente?

3) *Performance*: o desempenho é fator crítico na aplicação?

4) Uso do sistema: o usuário deseja executar a aplicação em um equipamento já existente ou comprado e que será altamente utilizado?

5) Taxa de transações: qual o volume de transações esperado?

6) Entrada de dados *on-line*: são requeridas entrada de dados *on-line*?

7) Eficiência do usuário final: as funções interativas fornecidas pela aplicação enfatizam um projeto para o aumento da eficiência do usuário final?

8) Atualização *on-line*: há arquivos atualizados *on-line*?

9) Processamento complexo: qual o grau de complexidade do processamento interno?

10) Reusabilidade: em que grau o código é reutilizável?

11) Facilidade de instalação: em que grau o sistema é fácil de ser instalado?

12) Facilidade de operação: em que grau o sistema é fácil de ser operado?

13) Múltiplos locais: o sistema é projetado para múltiplas instalações em diferentes organizações?

14) Facilidade para mudanças: a aplicação é projetada de forma a facilitar mudanças?

Como cada um receberá uma nota de 0 a 5, o somatório desses fatores ficará entre 0 e 70. Assim, finalmente, a fórmula para o cálculo dos pontos por função ficará como segue:

$$VAF = 0{,}65 + \left(0{,}01 * \sum_{i=1}^{14} GSCi\right)$$

Onde: VAF = *Value Adjustment Factor* ou Fator de Ajuste da Função;

GSC = *General Systems Characteristics* ou Características Gerais do Sistema.

Por fim, tendo sido obtido os pontos não ajustados, que chamaremos de UFP (*Unadjusted Function Points*, ou pontos por função não ajustados), basta aplicar a fórmula que segue para o número de pontos por função ajustado:

$$AFP = UFP\ x\ VAF$$

Vamos a um exemplo simples de cálculo de pontos por função. Suponha um projeto de programa que gerou as seguintes contagens:

Tabela 3.4 | Cálculo de pontos por função

Componente	Quantidade contada	Complexidade Funcional	Fator de complexidade	Total de complexidade	Total do componente
Arquivo Lógico Interno	3	Baixa	7	21	21
	0	Média	10	0	
	0	Alta	15	0	
Arquivo de Interface Externa (AIE)	2	Baixa	5	10	10
	0	Média	7	0	
	0	Alta	10	0	
Entradas Externas (EE)	4	Baixa	3	12	26
	2	Média	4	8	
	1	Alta	6	6	
Saídas Externas (SE)	2	Baixa	4	8	8
	0	Média	5	0	
	0	Alta	7	0	
Consultas Externas (CE)	2	Baixa	3	6	14
	2	Média	4	8	
	0	Alta	6	0	

Fonte: Wazlawick (2013, p. 161).

Teremos então o valor 79 como contagem dos pontos por função não ajustados. Supondo que tenhamos obtido 55 na soma das características gerais do sistema, a fórmula de cálculo do Fator de Ajuste da Função ficaria como segue:

$VAF = 0{,}65 + 0{,}55$

$VAF = 1{,}2$

Ao aplicarmos a fórmula dos pontos ajustados, obteremos: $AFP = 79 * 1{,}2$. Portanto, os pontos por função ajustados para esse caso é 94,8.

Revisões e inspeções de *software*

Como você bem se recorda, revisões técnicas e inspeções foram apresentadas na primeira seção desta unidade. Dada a importância de ambos e a intenção de introduzir desde já temas relacionados a teste de *software*, uma nova abordagem será feita aqui. Uma inspeção típica apresenta cinco etapas formais (SCHACH, 2008):

1. Uma visão geral do documento a ser inspecionado (histórias, projeto, código ou outro) é dada e, logo após, é distribuído aos presentes na sessão;

2. Por meio de uma lista de imperfeições detectadas em inspeções recentes, a equipe deverá analisar e entender o presente documento;

3. Um participante escolhido deve percorrer o documento e conduzir a equipe de inspeção, garantindo que cada item seja verificado, em busca de falhas. A missão é encontrar falhas, não as corrigir. No prazo de um dia, o líder da equipe dever gerar relatório do que foi discutido;

4. No processo chamado **reformulação**, o responsável resolve as imperfeições encontradas;

5. No **acompanhamento**, o responsável deve garantir que cada questão levantada tenha sido adequadamente resolvida.

Para determinar a eficácia de uma inspeção, você pode usar a métrica da **taxa de inspeção**. Quando, por exemplo, a especificação dos requisitos (ou histórias) passam por inspeção, o número de páginas inspecionadas por hora pode ser medido. Se a inspeção recair sobre o código, uma métrica interessante é a de quantidade de linhas de código inspecionadas por hora.

Lembre-se

Formalmente, Inspeção e *Walkthroughs* (passo a passo) são processos diferentes. Este último é um processo de duas etapas apenas: uma de preparação e outra de análise de documento pela equipe.

Faça você mesmo

Considere um sistema que você conheça bem e separe, classifique e pontue suas funções de acordo com o estipulado para o cálculo de pontos por função. Na sequência, aplique os fatores de ajuste e obtenha o valor ajustado das funções, conforme a fórmula dada.

No próximo item será aplicado o conteúdo que acabamos de abordar com o objetivo de resolver o desafio proposto.

Sem medo de errar

A busca da *XAX-Sooft* pela excelência em seus processos e produtos continua. Com a sua intervenção, a empresa expandiu o conceito de qualidade e agora mira nas medições dos atributos de seus programas para obter controle sobre aquilo que produz. A implantação de medições e as métricas que delas derivam servirão para inspirar ainda mais a equipe na demanda pela qualidade.

No item anterior a este, você teve a oportunidade de absorver um pouco do conteúdo relacionado a métrica de **Pontos por Função**, juntamente com os conceitos de medição e métrica de *software*. Tais conhecimentos, aliados à sua crescente habilidade em planejar implantações de novos procedimentos, permitirá a você superar o desafio que ora é proposto. Conforme já discutido, você deverá relatar seu planejamento para implantar processo de obtenção de métricas de *software* no cotidiano da *XAX-Sooft*. Vamos então a uma possível solução para o desafio, utilizando a métrica de Pontos por Função como objeto.

1. Você deverá definir que o resultado obtido pela aplicação da métrica de Pontos por Função servirá para estimar o esforço para criação dos novos produtos. Em outras palavras, o procedimento deverá ser aplicado antes que o produto fique pronto.

2. A equipe responsável pela coleta e tratamento das histórias ou dos

requisitos deverá ser orientada para que prepare as funções para serem processadas, excluindo cálculos internos da contagem e dividindo mais de uma função no mesmo contexto em duas funções distintas.

3. A equipe responsável pela aplicação da métrica deverá, então, contar as funções, obter os pontos por função não ajustados. Com utilização de critérios próprios e baseados na natureza de cada projeto, a equipe deverá estipular os valores que serão aplicados a cada um dos 14 fatores de ajuste, a fim de obter os pontos por função ajustados.

4. Após sua obtenção, o resultado deverá ser encaminhado para avaliação e análise, de forma que o esforço para criação daquele programa seja quantificado.

Com esse planejamento e com o bom aproveitamento do conteúdo ministrado, você inicia sua preparação para gerenciar processos de aplicações de medições e métricas em sua vida profissional. Dependendo de certas demandas, você poderá também incluir a aplicação de outras métricas no cotidiano da *XAX-Sooft*.

Atenção

A aplicação dos fatores de ajuste nos pontos obtidos não é procedimento obrigatório. Em, por exemplo, uma organização que produz programas com graus de dificuldade homogêneos, a prática poderá ser dispensada.

Lembre-se

A métrica de Pontos por Função não se aplica apenas aos programas em fase de planejamento. Ela poderá ser aplicada também em processos de manutenção, permitindo a estimativa do esforço necessário para se fazer certa alteração no programa.

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 as compare com as de seus colegas.	
Fazendo estimativas com Pontos por Função	
1. Competência geral	Competência para conhecer e conduzir processos de qualidade de *software*.

2. Objetivos de aprendizagem	Introduzir métricas e inspeções de *software*.
3. Conteúdos relacionados	Introdução a revisões, inspeções, medições e métricas.
4. Descrição da SP	A empresa *XAX-Sooft*, devidamente estabelecida no mercado e que vem desenvolvendo cultura de qualidade e de aprimoramento constante de seus produtos ao longo do tempo, também implantou recentemente a métrica de Pontos por Função. No entanto, embora a contagem das funções e o cálculo estejam sendo feitos corretamente, os resultados não têm servido para estimar o esforço efetivo a ser empreendido em seus projetos de criação de programas. Ou seja, o resultado é obtido, mas não é adequadamente utilizado. Sua missão é sugerir um procedimento que permita que a *XAX-Sooft* seja capaz de estimar o esforço em um projeto, com base no resultado obtido pela aplicação da métrica Pontos por Função.
5. Resolução da SP	O esforço total de construção do programa é dado pela multiplicação do Índice de Produtividade (IP) pelo resultado obtido pela aplicação da métrica Pontos por Função (APF). A fórmula então fica: *E = APF * IP* Naturalmente que a variável mais suscetível a grandes variações é justamente o índice de produtividade, que varia em função da experiência da equipe, do ambiente de trabalho e da aptidão da equipe para aquele projeto em específico. Um dos cálculos possíveis para a obtenção do Índice de Produtividade se perfaz pela tomada de um projeto anteriormente desenvolvido, com esforço conhecido como base.

Lembre-se

Medidas e métricas serão valores frios e sem utilidade se não interpretados adequadamente.

Faça você mesmo

Pesquise sobre e resuma os conceitos de Arquivo Lógico Interno (ALI) e Arquivo de Interface Externa (AIE); Entrada Externa (EE), Saída Externa (SE) e Consulta Externa (CE). Inicie sua pesquisa por: <http://www.inf.ufpr.br/silvia/ES/metricas/FP.pdf>. Acesso em: 12 fev. 2016.

Faça valer a pena

1. Assinale a alternativa que descreve o conceito de métrica.

a) Processo pelo qual os números são atribuídos aos atributos de entidades do mundo real.

b) Quantificação direta, que envolve um único valor.

c) Quantificação indireta, que envolve o cálculo e o uso de mais de uma medida.

d) Escala de valores que dá a noção de uma medida de um determinado elemento.

e) Processo pelo qual se avalia a capacidade de cada desenvolvedor com base em seu tempo de experiência na função.

2. Em relação a revisões de *software*, assinale a afirmação verdadeira:

a) Uma revisão típica apresenta duas etapas formais e, por isso, equivale ao *walkthrough*.

b) Numa das etapas da revisão, o responsável pela reunião faz a refatoração do código defeituoso.

c) É na etapa conhecida como acompanhamento que o responsável pela revisão deve garantir que cada questão levantada tenha sido adequadamente resolvida.

d) Por não ser tangível, não há maneiras de se medir um procedimento de revisão.

e) A revisão consiste apenas na leitura cuidadosa do código-fonte, linha a linha, em busca de defeitos.

3. Em relação ao processo de Análise de Pontos por Função, analise as afirmações que seguem:

I - Trata-se de medida de dificuldade em se programar determinadas funções do programa.

II - Os requisitos são convertidos em valores numéricos que, depois de calculados e ajustados, fornecerão estimativa do esforço necessário para desenvolver o sistema.

III - Aplicável a partir do momento em que os requisitos funcionais do programa (ou as histórias) tenham sido definidos.

IV - Os resultados obtidos pela aplicação dessa métrica valem apenas para estimar esforço de desenvolvimento.

É verdadeiro o que se afirma apenas em:

a) IV.

b) II e IV.

c) III e IV.

d) II e III.

e) I.

Referências

ABNT – Associação Brasileira de Normas Técnicas. **NBR ISO/IEC 9126-1**: engenharia de *software* – qualidade de produto – parte 1: modelo de qualidade. Rio de Janeiro: ABNT, 2003.

BARTIÉ, A. **Garantia da qualidade de *software***: as melhores práticas de Engenharia de Software aplicadas à sua empresa. 5. ed. São Paulo: Elsiever, 2002.

CMMI INSTITUTE. **Who uses CMMI?** Disponível em: <http://cmmiinstitute.com/who-uses-cmmi>. Acesso em: 24 jan. 2016.

CROSBY, P. B. **Quality is free**: the art of making quality certain. New York: New American Library, 1979.

DENNIS, R. G; DIANE, L. G. **Demonstrating the impact and benefits of CMMI**: an update and preliminary results. 2003. Disponível em: <http://resources.sei.cmu.edu/asset_files/SpecialReport/2003_003_001_14117.pdf>. Acesso em: 5 fev. 2016.

DEVMEDIA. **CMMI**: uma visão geral. Disponível em: <http://www.devmedia.com.br/cmmi-uma-visao-geral/25425#ixzz3yBwjcLs1>. Acesso em: 24 jan. 2016.

GOVERNO ELETRÔNICO. **ePING – Padrões de interoperabilidade de Governo Eletrônico**. Disponível em: <http://www.governoeletronico.gov.br/acoes-e-projetos/e-ping-padroes-de-interoperabilidade/o-que-e-interoperabilidade>. Acesso em: 18 fev. 2016.

HUMPHREY, W., KITSON, D., KASSE, T. The state of software engineering practice: a preliminary report. In: **proceedings of the 11th international conference on software Engineering**. 1989, p. 277-288.

IEEE (Instituto de Engenheiros Eletricistas e Eletrônicos, do inglês *Institute of Electrical and Electronics Engineers*). **SWEBOK *(Guide to the Software Engineering Body of Knowledge)***. 2004. Disponível em: <https://www.computer.org/portal/web/swebok>. Acesso em: 22 fev. 2016.

IME. **Ariane 5**: um erro numérico (*overflow*) levou à falha no primeiro lançamento. Disponível em: <http://www.ime.uerj.br/~demoura/Especializ/Ariane/>. Acesso em: 17 jan. 2015.

ISO. **About ISO**. Disponível em: <http://www.iso.org/iso/home/about.htm>. Acesso em: 14 jan. 2016.

LEÃO, A. **Aplicação de técnicas de ITIL e CMMI em pequenas e médias empresas de software**. Disponível em: <http://www.techoje.com.br/site/techoje/categoria/detalhe_artigo/1734>. Acesso em: 5 fev. 2016.

MECENAS, I.; OLIVEIRA, V. **Qualidade em software**: uma metodologia para homologação de sistemas. [s. l.]: Alta Books, 2005.

MOORTHY, V. **Jumpstart to software quality assurance**. [S.l. : s.n.], 2013.

PORTAL EDUCAÇÃO. **A história da organização ISO**. Disponível em: <http://www.portaleducacao.com.br/administracao/artigos/40732/a-historia-da-organizacao-iso#!1>. Acesso em: 28 jan. 2016.

PRESSMAN, R. S. **Engenharia de software**. São Paulo: Makron Books, 1995.

_____. São Paulo: Pearson Prentice Hall, 2009. 1056 p.

QUALITY CONSULT. **O que muda na versão 2015 da norma ISO 9001**. Disponível em: <http://www.qualityconsult.com.br/index.php/iso-9001-versao-2015/>. Acesso em: 23 jan. 2016.

REISS, E. **Usable usability**: simple steps for making stuff better. Indianapolis: John Wiley & Sons, 2012.

REZENDE, D. A. **Engenharia de software e sistemas de informação**. 3. ed., rev. Rio de Janeiro: Brasport, 2005.

ROCHA, A. R.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de software**: Teoria e Prática. São Paulo: Pearson, 2001.

RODRIGUES, A. N. *et al*. **Qualidade de software – parte 1**. Disponível em: <http://www.devmedia.com.br/qualidade-de-software-parte-01/9408>. Acesso em: 11 jan. 2016.

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

SEEAR, D. J. **ISO 9001: 2015 Back to the future**. A review of the new ISO Annex SL structure for Certification Standards using the draft ISSO 9001: 2015 to explain the changes. USA: Author House, 2015.

SHAFFER, S. C. **A brief introduction to software development and quality assurance management**. [S.l.: s.n.], 2013.

SILVEIRA, S. G. **Carreira:** Analista de Teste – O que é e o que faz? Disponível em: <http://www.tiegestao.com.br/2013/09/01/carreira-analista-de-teste-o-que-e-e-o-que-faz/>. Acesso em: 20 jan. 2016.

SOFTEX. **MPS.BR**. Disponível em: <http://www.softex.br/mpsbr/mps/mps-br-em-numeros/>. Acesso em: 24 jan. 2016.

TSUKUMO, A. N. *et al*. **Qualidade de software**: visões de produto e processo de software. In: II Escola Regional de Informática da Sociedade Brasileira de Computação Regional de São Paulo – II ERI da SBC – Piracicaba, SP – junho de 1997, p. 173-189. Disponível em: <https://xa.yimg.com/kq/groups/21646421/371309618/name/Modelos+de+Qualidade+de+Software.pdf>. Acesso em: 18 jan. 2016.

WAZLAWICK, R. S. **Engenharia de Software**: conceitos e práticas. Rio de Janeiro: Elsiever, 2013.

Os testes de *software*

Convite ao estudo

Olá! Seja bem-vindo à quarta unidade de Engenharia de *software*.

Antes de chegar à última etapa do curso, você estudou conceitos fundamentais da disciplina e o jeito tradicional de se conduzir um processo de desenvolvimento de *software*. Depois, modelos de desenvolvimento mais atualizados foram abordados e você foi convidado a planejar e executar a passagem da *XAX-Sooft* para uma forma ágil de se criar programas de computador. Em nossa última etapa antes desta, foram abordados os conceitos de qualidade e como atingi-la por meio de padrões consagrados de qualidade. Você foi chamado a colocar como objetivo central da *XAX-Sooft* o atingimento da excelência em processos e produtos e saiu-se muito bem nessa missão.

E por falar em qualidade, eis que ela volta a receber toda a atenção. Nas próximas quatro aulas serão tratados com mais profundidade temas relacionados ao teste e à evolução de um *software*.

O objetivo geral desta unidade é que você domine conceitos e práticas associadas à atividade de teste. Cada seção, em específico, tem como objetivo possibilitar ao aluno:

- Conhecer os fundamentos da atividade de teste de *software*, teste funcional e estrutural.
- Conhecer os conceitos e de que forma se dá a execução do *Test-Driven Development* (TDD).
- Conhecer o que são e como implementar o teste de *release* e teste de usuário.
- Conhecer e saber fazer a manutenção e compreender como isso se aplica em meio à evolução do *software*.

Como consequência do alcance desses objetivos, você deverá desenvolver a competência técnica de conhecer e conduzir processos de teste de software. Você atingirá os objetivos e desenvolverá as competências desta unidade por meio da resolução dos desafios que serão colocados em cada seção da unidade, descritas aqui:

1. Selecionar e registrar os casos de teste que serão utilizados na aplicação de teste funcional num dos programas da *XAX-Sooft*.

2. Aplicar teste funcional em um dos programas da *XAX-Sooft*, com relato das atividades desempenhadas.

3. Relatar a avaliação dos resultados obtidos com a aplicação do teste de *software*.

4. Aplicar modelo de estimação de esforço de manutenção de um produto da *XAX-Sooft*.

Nas próximas páginas será detalhada a primeira parte de nossa missão, proposta de conteúdo teórico sobre teste de *software* e novas abordagens da situação-problema. Bom trabalho!

Seção 4.1

Testes de *software*: fundamentos, casos de teste, depuração, teste estrutural, teste funcional

Diálogo aberto

É notável a evolução pela qual a *XAX-Sooft* tem passado. Desde seu início, como uma empresa que sequer tinha um procedimento definido para desenvolvimento de seus produtos, até sua consolidação como uma organização reconhecidamente apta a entregar *software* de qualidade, a *XAX-Sooft* experimentou a ascensão que só uma empresa liderada por gente competente é capaz de ter. Todos os modelos e procedimentos adotados já fazem parte da rotina da empresa e a equipe sente-se à vontade com eles.

Afinal, o que mais a *XAX-Sooft* deveria incluir em seu dia a dia para manter sua evolução? Qual aspecto do seu modelo de desenvolvimento ainda pode ser melhorado? Ao tratar das questões introdutórias de teste de *software*, esta seção propõe-se a responder essas questões e esclarecer como a atividade de teste pode contribuir de forma decisiva para a qualidade do produto final e para a consequente redução do investimento de tempo em atividades de retrabalho.

Usando como apoio os conceitos que serão tratados nesta seção, você deverá superar o desafio que se apresenta. É sua missão planejar e registrar os casos de teste que serão utilizados na aplicação de teste funcional num dos programas da *XAX-Sooft*. Resumidamente, um caso de teste é um dado de entrada que leva o programa a executar partes de seu código e a respectiva saída esperada para essa entrada. A evolução da aula dará a você melhores condições para entender e aplicar esse conceito.

Bons estudos!

Não pode faltar

Na primeira unidade de nosso curso, o teste foi colocado como uma das fases finais do processo de desenvolvimento de um produto no modelo tradicional. A atividade foi situada como etapa seguinte à codificação e conceituada como a execução de um programa visando à detecção de defeitos em seu código.

Nas aulas seguintes, ao tratarmos de modelos ágeis, os testes foram abordados como atividade vinculada à codificação, com indicações de que o próprio desenvolvimento deveria ser guiado pelos testes. Avançando mais um pouco no conteúdo, as atividades que visam garantir a qualidade de um produto – das quais o teste faz parte – foram diluídas por todo o processo e, de certa forma, desconstruíram a ideia de que o teste deve ser encarado como uma fase estanque, sem comunicação e sem abrangência nas demais fases do processo.

Pois bem, qualquer que seja a forma pela qual encaremos o teste e onde quer que o situemos no processo, ele sempre será atividade indispensável e de alta criticidade na produção de programas de qualidade. Tratá-lo sem a devida importância seria arriscar toda a produção supondo que programadores não cometem erros, o que, de longe, é um equívoco.

Nas linhas seguintes você terá contato com conceitos ligados ao tema e à apresentação de duas técnicas bastante usadas para a realização de testes, o que servirá de base para a superação do nosso desafio. Sigamos.

Fundamentos

Um teste – ou um **processo de teste** – consiste em uma sequência de ações executadas com o objetivo de encontrar problemas no *software*, o que aumenta a percepção de qualidade geral do *software* e garante que o usuário final tenha um produto que atenda às suas necessidades (PINHEIRO, 2015).

É sempre bom destacar que o objetivo do teste é encontrar problemas no *software*, e não garantir que o programa é livre de defeitos. Se o processo de teste não revelar defeitos, há que se aprimorar os casos de teste e o processo empregado. Não se pode acreditar que o sistema não possui problemas se o teste não os revelar.

O processo de teste é normalmente separado em quatro grandes etapas:

- Planejamento: nesta etapa deve ser definido quem executa os testes, em que período, com quais recursos (ferramentas de teste e computadores, por exemplo) e qual será a técnica utilizada (técnica estrutural ou técnica funcional, por exemplo).

- Projeto de casos de teste: aqui são definidos os casos de teste que serão utilizados no processo. No próximo item, este conceito será detalhado.

- Execução do programa com os casos de teste: nesta etapa, o teste é efetivamente realizado.

- Análise dos resultados: aqui verifica-se se os testes retornaram resultados satisfatórios.

Na sequência, um item muito importante para a resolução do problema proposto será abordado.

Casos de teste

Um caso de teste é o par formado por uma entrada no programa e a correspondente saída esperada, de acordo com os requisitos do sistema. Entenda o conceito de entrada como o conjunto de dados necessários para a execução do programa. A saída esperada é o resultado de uma execução do programa ou função específica. Imagine que estejamos colocando sob teste um programa que valida datas inseridas pelo usuário. Um caso de teste possível seria (25/12/2016; válida). Ao receber a entrada 25/12/2016, o programa de validação de data deveria retornar "data válida".

É certo que a boa escolha dos casos de teste é fundamental para o sucesso da atividade. Um conjunto de casos de teste de baixa qualidade pode não exercitar partes críticas do programa e acabar não revelando defeitos no código. Se o responsável pelos testes usasse apenas datas válidas como entradas, a parte do programa que trata das datas inválidas não seria executada, o que prejudicaria a confiabilidade do processo de teste e do produto testado. Observe os casos de teste que seguem:

t1= {(13/2/2016;válida), (30/2/2004;inválida); (25/13/2000;inválida); (29/2/2016;válida), (29/2/2015;inválida), (##/1/1985;inválida)}. Com esse cenário, certamente a maior parte do código será coberta e a chance de detecção de defeitos aumentará.

Parece claro que o procedimento de testes está diretamente relacionado à boa escolha e ao bom uso dos casos de teste. Idealmente,

cada conjunto de casos de teste deverá estar associado a um grande requisito diferente a ser testado. Para que não se corra o risco de defini-los incorretamente, é necessário planejamento e o bom conhecimento da aplicação. Uma boa forma de se abordar o problema é a que segue (PINHEIRO, 2015):

- Definir o ambiente no qual o teste será realizado;
- Definir a entrada deste caso de teste;.
- Definir a saída esperada para cada entrada;
- Definir os passos a serem realizados para executar os testes.

Quando um caso de teste é executado, o seu resultado deve ser coletado. Podemos assumir diferentes abordagens para definir o resultado da aplicação de um caso de teste específico. A mais comum define as seguintes opções (PINHEIRO, 2015):

- Passou: todos os passos do caso de teste foram executados com sucesso para todas as entradas;
- Falhou: nem todos os passos foram executados com sucesso para uma ou mais entradas;
- Bloqueado: o teste não pôde ser executado, pois o seu ambiente não pôde ser configurado.

Pois bem, dessa forma definimos casos de teste. Há, no entanto, três conceitos especialmente importantes no contexto de teste e que são frequentemente confundidos entre si. Vamos a eles.

Defeito, Falha e Erro

Que expressão que você usa quando um programa simplesmente trava ou não produz o resultado que se espera dele? Tudo o que acontece de incomum em um programa pode ser chamado de erro? Observe os conceitos:

Defeito: trata-se de deficiência algorítmica que, se ativada, pode levar a uma falha. Vamos a um exemplo. Observe o trecho que segue, escrito em pseudocódigo:

$a = -1;$

$b = 0;$

enquanto a < 0 faça

b = *b* + 1;

imprima (*b*);

imprima (*a*);

fim_enquanto;

Em algum momento o valor da variável deixará de ser menor que zero? Estamos diante de um defeito, mais precisamente um laço infinito. Se o caso de teste escolhido for capaz de exercitar esse trecho do código, o defeito se manifestará e então teremos uma falha observável. O programa provavelmente será interrompido.

Falha: é tida como um não funcionamento do programa, provavelmente provocada por um defeito. No entanto, uma falha também pode ser atribuída a uma queda na comunicação ou a um erro na leitura do disco, por exemplo.

Erro: ocorre quando o resultado obtido em um processamento e o que se esperava dele não são coincidentes. Um erro também está associado a uma violação nas próprias especificações do programa. Por exemplo, um usuário não autorizado consegue acessar determinado módulo do programa (esse é o resultado obtido), sendo que seu nível de privilégios não deveria permitir que o fizesse (esse era o resultado esperado).

Pesquise mais

Os conceitos de defeito, falha e erro não são uniformes nas referências. Outros conceitos básicos também podem variar entre autores. Uma boa fonte introdutória e conceitual pode ser obtida em: <http://www.devmedia.com.br/artigo-engenharia-de-software-introducao-a-teste-de-software/8035>. Acesso em: 14 fev. 2016.

Embora estejamos diante de conceitos com diferenças sutis e que frequentemente são confundidos, a devida separação em três certamente facilitará o entendimento de outros conceitos e procedimentos mais adiante.

Depuração

Enquanto testar significa executar o *software* para encontrar

defeitos desconhecidos, a depuração é a atividade que consiste em buscar no código a localização desses erros. O fato de saber que o programa não funciona não implica, necessariamente, já se saber também em qual ou quais linhas o problema está. Os ambientes de programação atuais oferecem recursos para depuração do programa. Durante esse processo, o valor assumido pelas variáveis sob inspeção em cada passo do algoritmo pode ser observado. Além disso, alguns pontos de parada da execução do programa podem ser inseridos no código. Tudo para possibilitar que o testador identifique e isole o defeito no código.

Assimile

O término bem-sucedido do processo de compilação significa que o código apresenta correção sintática e semântica e, portanto, pode ser executado pelo computador. No entanto, isso não garante, definitivamente, que o programa esteja livre de erros de lógica ou de cálculo.

Como todo processo desenvolvido com base científica, a atividade de teste também inclui maneiras estruturadas e consagradas para a sua realização. Dependendo da disponibilidade do código-fonte, da ferramenta de teste escolhida e das características do programa a ser testado, a técnica funcional ou a técnica estrutural podem ser aplicadas.

Imagine o programa como uma caixa. Quando o testador não tem acesso ao código-fonte, ele está lidando com uma caixa preta, cujo interior não se consegue ver. Daí o teste funcional também ser conhecido como caixa preta. A técnica estrutural, por sua vez, é também conhecida como caixa branca, cujo interior se pode ver. Essa comparação é feita justamente pela disponibilidade do código-fonte e das estruturas internas do programa.

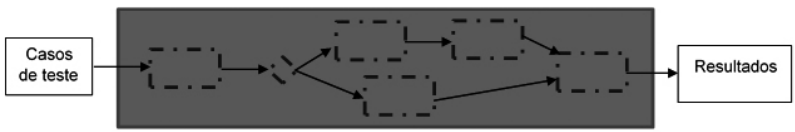
Segue uma descrição mais aprofundada dessas técnicas:

Técnica de teste funcional

Esta técnica baseia-se nas especificações do *software* para derivar os requisitos de teste. O teste é realizado nas funções do programa, daí o nome funcional. Não é seu objetivo verificar como ocorrem internamente os processamentos, mas se o algoritmo inserido produz os resultados esperados (BARTIÉ, 2002).

Uma das vantagens dessa estratégia de teste é o fato de ela não requerer conhecimento de detalhes da implementação do programa. Sequer o código-fonte é necessário. Observe uma representação dessa técnica na figura 4.1:

Figura 4.1 | Visão de teste de caixa preta



Fonte: Bartié (2002, p. 105).

O planejamento do teste funcional envolve dois passos principais: identificação das funções que o *software* deve realizar (por meio da especificação dos requisitos) e a criação de casos de teste capazes de checar se essas funções estão sendo executadas corretamente.

Apesar da simplicidade da técnica e apesar de sua aplicação ser possível em todos os programas cujas funções são conhecidas, não podemos deixar de considerar uma dificuldade inerente: não se pode garantir que partes essenciais ou críticas do *software* serão executadas, mesmo com um bom conjunto de casos de teste.

Exemplificando

Imagine uma função que valida nomes de identificadores em uma linguagem de programação criada por você. As condições para validação são: tamanho do identificador entre 1 e 6 caracteres, primeiro caractere necessariamente deve ser letra e caracteres especiais não são permitidos. A tabela 4.1 resume as condições de validade do identificador e a aplicação do teste.

Tabela 4.1 | Resultado da aplicação da técnica funcional com os casos de teste dados

Condições de Entrada	Classes Válidas	Classes Inválidas
Tamanho *t* do identificador	$1 \leq t \leq 6$ (1)	$t > 6$ (2)
Primeiro caractere *c* é uma letra	Sim (3)	Não (4)
Só contêm caracteres válidos	Sim (5)	Não (6)

Fonte: elaborada pelo autor.

Exemplo de Conjunto de Casos de Teste

T0 = {(a1,Válido), (2B3, Inválido), (Z-12, Inválido), (A1b2C3d, Inválido)}

(1, 3, 5) (4) (6) (2)

Conheça agora uma técnica de teste bastante eficiente na descoberta de defeitos.

Técnica de teste estrutural

Os testes estruturais (ou de caixa branca) são assim conhecidos por serem baseados na arquitetura interna do programa. De posse do código-fonte (ou do código-objeto) e, se for o caso, das estruturas de banco de dados, o profissional designado para a atividade submete o programa a uma ferramenta automatizada de teste.

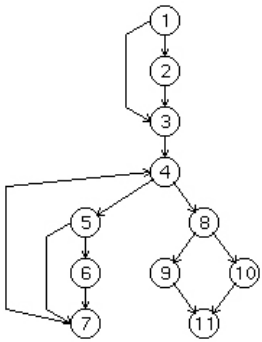
A ferramenta constrói uma representação de programa conhecida como grafo de programa. No grafo, os nós equivalem a blocos indivisíveis, ou seja, não existe desvio de fluxo do programa para o meio do bloco e, uma vez que o primeiro comando do bloco é executado, os demais comandos são executados sequencialmente. As arestas ou arcos representam o fluxo de controle entre os nós. Observe o trecho de código a seguir, escrito em linguagem C. Ele valida identificadores de acordo com os critérios dados no quadro *Exemplificando* (DELAMARO, 2004). Os números à frente de cada linha representam os nós do grafo que vem logo a seguir.

```
/* 01 */ {
/* 01 */ char achar;
/* 01 */ int length, valid_id;
/* 01 */ length = 0;
/* 01 */ printf ("Identificador: ");
/* 01 */ achar = fgetc (stdin);
/* 01 */ valid_id = valid_s(achar);
/* 01 */ if (valid_id)
/* 02 */          length = 1;
/* 03 */ achar = fgetc (stdin);
/* 04 */ while (achar != '\n')
/* 05 */ {
/* 05 */          if (!(valid_f(achar)))
/* 06 */                    valid_id = 0;
/* 07 */          length++;
/* 07 */          achar = fgetc (stdin);
/* 07 */ }
```

```
/* 08 */ if (valid_id && (length >= 1) && (length < 6) )
/* 09 */          printf ("Valido\n");
/* 10 */ else
/* 10 */          printf ("Invalido\n");
/* 11 */ }
```

A submissão do código a uma dada ferramenta de teste criará a representação vista na figura 4.2. Note que cada trecho identificado com um número no código é reproduzido em um nó do grafo.

Figura 4.2 | Representação em grafo de código sob teste



Fonte: Delamaro (2004, p. 11).

A função do testador, então, é encontrar casos de teste que, durante uma execução do programa, sejam capazes de exercitar os caminhos, nós e arcos do grafo. Um caminho simples, por exemplo, seria dado por (2,3,4,5,6,7). O caminho completo é representado por (1,2,3,4,5,7,4,8,9,11).

Aí está, resumidamente, colocada a técnica funcional. Apesar de ser mais eficiente em encontrar defeitos no código, sua aplicação é mais dispendiosa e complexa.

Reflita

Se é certo que um teste, por melhor que seja executado, não conseguirá assegurar que o programa é 100% livre de defeitos, qual a indicação de que é chegado o momento de interromper os testes? Pois é justamente durante o planejamento do teste que os critérios de parada da atividade são definidos.

Pois bem, a devida introdução ao teste de *software* foi dada. Existem outras tantas técnicas e métodos de aplicação de teste cuja abordagem extrapolaria os objetivos desta seção. Embora de difícil execução e de

custo relativamente alto, a atividade de teste é fundamental no processo de criação de programas. É certo que desenvolvedores têm investido em ferramentas e em material humano para conferir graus elevados de qualidade aos seus produtos. E é importante que seja assim.

Faça você mesmo

Pesquise sobre a ferramenta de teste chamada JaBUTi e prepare um resumo sobre seu funcionamento. Comece sua pesquisa por: <http://ccsl.icmc.usp.br/pt-br/projects/jabuti>. Acesso em: 14 fev. 2016.

Sem medo de errar

A atividade de teste, quando tratada de modo correto e profissional, deve ser bem planejada e estruturada antes de ser executada. Não se pode conceber que uma empresa que busque a excelência em seus produtos negligencie essa etapa tão importante do desenvolvimento.

A preparação da atividade não pode deixar de prever os recursos que serão utilizados nos testes, o prazo para sua execução, a técnica a ser utilizada e, em estágio mais avançado de planejamento, os casos de teste a serem utilizados.

A *XAX-Sooft*, na condição de organização que busca constante aprimoramento em seus processos e produtos, não poderia deixar de investir boa energia no teste dos seus produtos. Como desafio proposto, é sua missão planejar e registrar os casos de teste que serão utilizados na aplicação de teste funcional num dos programas da *XAX-Sooft*. Com a finalidade de dar suporte a você, as duas funções do programa a serem testadas são descritas na sequência:

Função 1: validação de CPF do cliente, segundo seu estado de origem. O terceiro dígito da direita para a esquerda identifica a unidade federativa na qual a pessoa foi registrada, de acordo com o quadro 4.1. Exemplo: o CPF 000.000.008-00 é de alguém cujo estado de origem é São Paulo.

Quadro 4.1 | Unidade Federativa de registro do CPF

0 - Rio Grande do Sul	3 - Ceará, Maranhão e Piauí	6 - Minas Gerais	9 - Paraná e Santa Catarina
1 - Distrito Federal, Goiás, Mato Grosso do Sul e Tocantins	4 - Paraíba, Pernambuco, Alagoas e Rio Grande do Norte	7 - Rio de Janeiro e Espírito Santo	
2 - Amazonas, Pará, Roraima, Amapá, Acre e Rondônia	5 - Bahia e Sergipe	8 - São Paulo	

Fonte: <http://www.geradordecpf.org/>. Acesso em: 28 mar. 2016.

Função 2: verifica atraso no pagamento e aplica acréscimo de 1 ponto percentual sobre cada dia de atraso verificado no pagamento. Uma solução possível para esse desafio é a que segue:

Exemplo de conjunto de casos de teste da Função 1:

Formato geral: (número_do_cpf, estado_de_origem; saída_esperada)

t_1={(937.599.133-42, Ceará; válido), (831.469.521-14, Tocantins; válido), (858.178.888-23, São Paulo; válido), (300.168.443-78, Rio Grande do Sul; inválido)}

Exemplo de conjunto de casos de teste da Função 2:

Formato geral: (data_do_vencimento, data_do_pagamento, valor_do_debito; valor_a_ser_pago)

t_2={(10/2/2016, 14/2/2016, 500; 520), (5/2/2016, 2/2/2016,125;125), (15/2/2016, 5/2/2016, 68; 68)}

Atenção

A conferência da saída obtida pela aplicação do caso de teste é que indicará a existência de problema no código ou não.

Lembre-se

Quanto maior a qualidade do conjunto de casos de teste, maiores serão as chances de o teste detectar defeitos.

Avançando na prática

Pratique mais	
Instrução Desafiamos você a praticar o que aprendeu, transferindo seus conhecimentos para novas situações que você pode encontrar no ambiente de trabalho. Realize as atividades e depois as compare com as de seus colegas.	
Aumentando a qualidade dos casos de teste	
1. Competência geral	Conhecer as principais metodologias de desenvolvimento de *software*, normas de qualidade e processos de teste de *software*.
2. Objetivos de aprendizagem	Conhecer os fundamentos da atividade de teste de *software*, teste funcional e estrutural.
3. Conteúdos relacionados	Testes de *software*: fundamentos, casos de teste, depuração, teste estrutural, teste funcional.
4. Descrição da SP	A RRHH-Soft, empresa desenvolvedora de sistemas feitos para departamentos de recursos humanos, adotou recentemente procedimento de teste em seus produtos. Depois de codificado o sistema, a empresa passa a executar alguns testes antes da entrega. No entanto, o procedimento não tem evitado que o código permaneça com defeitos que deveriam ter sido descobertos por meio dos testes. Como consequência, a frequência com que erros têm se manifestado em ambiente de produção tem sido alta. Seu desafio aqui é indicar, no âmbito das atividades do teste funcional, ações para tornar mais eficiente a descoberta de problemas no código.
5. Resolução da SP	A solução do caso requer as possíveis providências: • Planejamento da atividade: além da definição de prazos e recursos a serem usados no teste, durante o planejamento há também que se buscar na especificação de requisitos a totalidade das funções que deverão ser testadas. • Seleção dos casos de teste: com base nas funções do sistema, a equipe deverá selecionar casos de teste abrangentes o suficiente para exercitar maior número possível de trechos do código, principalmente aqueles que produzem resultado numérico provenientes de cálculos. Encontrado o problema, a depuração no código deverá ser usada para saná-lo. • Análise dos resultados: aplicados os testes, a equipe deverá avaliar se os casos de teste foram eficientes na detecção dos defeitos. Baixa detecção não significa, necessariamente, programa de alta qualidade.

Lembre-se

O conjunto de casos de teste deve abranger tanto entradas corretas como as incorretas. Para ambas, a saída obtida deverá ser igual a saída esperada, o que é indício da corretude da função testada.

Faça você mesmo

Qual o perfil de um profissional que realiza os testes? Quais as habilidades requeridas para essa função? Faça a leitura dos artigos publicados em: <http://www.devmedia.com.br/artigo-engenharia-de-software-18-profissional-da-area-de-testes/14801> e <http://crowdtest.me/caracteristicas-testador-infografico/> (acesso em: 15 fev. 2016) e fique sabendo.

Faça valer a pena

1. Em relação aos fundamentos da atividade de teste, assinale a alternativa que contém expressões que completam corretamente as lacunas na frase a seguir.

"O processo de teste consiste em executar um programa com o objetivo de revelar a presença de _____; ou, falhando nesse objetivo, aumentar a _____ sobre o programa. Após revelada a presença do defeito, o processo de _____ auxilia em sua busca e correção."

a) funções recursivas; *performance*; teste.

b) defeitos; exatidão; teste exaustivo.

c) casos de teste; confiança; apuração.

d) defeitos; confiança; depuração.

e) defeitos; exatidão; casos de teste.

2. Em relação aos casos de teste, analise as afirmações a seguir:

I - Um caso de teste é o par formado por uma entrada no programa e a correspondente saída esperada.

II - Um caso de teste equivale a uma seção em que os testes são realizados.

III - A escolha correta dos casos de teste tem importância relativa no processo, já que são selecionados pelo cliente.

IV - Os casos de teste são específicos para cada programa submetido a teste.

É verdadeiro o que se afirma apenas em:

a) I e IV.

b) II e III.

c) IV.

d) II.

e) II e IV.

3. Em relação aos conceitos de defeito, falha e erro, analise as afirmações a seguir:

I - Um laço de repetição construído de forma a deixar o processamento preso em laço infinito configura um defeito.

II - Falha é um não funcionamento do programa.

III - Uma falha no programa pode ser provocada por um problema no *hardware*.

É verdadeiro o que se afirma em:

a) III, apenas.

b) I, II e III.

c) I e III, apenas.

d) II e III, apenas.

e) I, apenas.

Seção 4.2

O *Test-Driven Development* (TDD)

Diálogo aberto

Além de introduzir conceitos fundamentais de teste, nossa última aula serviu para desconstruir a impressão de que verificação em um programa pode ser feita apenas ao se vasculhar o código ou executar o programa algumas vezes, sem critério ou formalidade. Duas das formas mais comuns de executar verificações num *software* – o teste funcional e o teste estrutural – foram introduzidas e a importância da correta escolha de casos de teste foi destacada.

Pois bem, a percepção geral na *XAX-Sooft* é que a introdução de procedimento formal de teste começará em breve a produzir bons resultados. A fase inicial do novo procedimento incluiu a seleção de casos de teste e aquele desafio tornou você e a equipe de desenvolvimento aptos para dar continuidade ao processo de teste. O desafio que se coloca nesta seção é a efetiva aplicação do teste funcional em um dos produtos da *XAX-Sooft*, incluindo o relato das atividades desempenhadas e as ações empreendidas pelos envolvidos no procedimento.

Além de fornecer nova abordagem do *Test-Driven Development* (TDD) e apresentar esta técnica como própria do modelo ágil de desenvolvimento, é também objetivo desta seção apresentar o tratamento que um erro encontrado por um testador recebe após sua descoberta. Esse tratamento desenhará o que chamaremos de ciclo de vida de um erro e você, na condição de participante ativo do processo de teste, será convidado a sugerir aprimoramentos no desenho desse ciclo, que nasce na descoberta do erro e termina em sua completa correção.

Com esse conteúdo, abordado principalmente no item "Não pode faltar", você seguirá aprimorando sua competência para conhecer e conduzir processos de testes de *software*, sempre mirando a excelência nos procedimentos e na qualidade dos produtos da *XAX-Sooft*.

Adiante e bom trabalho!

Não pode faltar

Na Unidade 2 deste material, uma abordagem preliminar de Desenvolvimento Guiado pelos Testes (ou TDD – *Test-Driven Development*) foi feita. O assunto se encaixava perfeitamente no contexto das metodologias ágeis, que na ocasião eram tratadas em detalhes. Depois de termos tratado da maioria das práticas do XP (*Extreme Programming*) e, já na Unidade 3, passado por assuntos relacionados à qualidade, eis que novamente o TDD assume lugar de importância em nosso estudo.

É natural que, depois de conhecer aspectos de qualidade e do teste de *software*, você encare o assunto de outra maneira e esteja mais apto a expandir seus conhecimentos sobre ele.

Esta seção tratará do caminho que um erro percorre após sua identificação e os responsáveis por cada ação empreendida para corrigi-lo. Cada metodologia de desenvolvimento guarda particularidade na execução do processo de teste e o TDD será abordado como procedimento de teste do modelo ágil, já estudado na Unidade 2.

Na sequência, o conteúdo teórico desses assuntos será abordado.

Engenharia de Testes

Em nossa última seção, quando tratamos de temas iniciais de teste de *software*, foi dada ênfase às definições de defeito, erro e falha. Entre um exemplo e outro, o texto procurou deixar clara a sutileza entre os conceitos, sem que as diferenças entre os três se perdessem.

No entanto, fora dos limites da esfera acadêmica, qualquer problema que se manifeste no código tende a ser chamado simplesmente de *bug*, ou como referenciaremos aqui, de erro. A intenção do que chamamos de Engenharia de Testes é simples: oferecer entendimento do que é um erro e oferecer formas de encontrá-lo.

Em seções anteriores, foi discutida a impossibilidade de assegurar a completa e irrestrita ausência de erros em um programa, justamente pela condição falível dos programadores. Mas será que apenas os programadores falham?

Um erro ocorre quando uma ou mais das opções a seguir for verdadeira (PINHEIRO, 2015):

- O *software* **não faz** algo que a especificação estabelece que ele deveria fazer;

- O *software* **faz** algo que a especificação estabelece que ele não deveria fazer;
- O *software* **faz** algo que a especificação não menciona;
- O *software* **não faz** algo que a especificação não menciona, mas deveria mencionar;
- O *software* é **difícil de usar**, entender ou, na visão do testador, pode ser visto pelo usuário final como não estando correto.

Fica claro que a especificação incorreta é uma das grandes causas de erros em programas. Mas não só ela. Em sistemas pequenos, erros na codificação respondem por aproximadamente 75% das ocorrências. A tendência é simples de ser entendida: quanto maior o projeto, menor a quantidade de erros na codificação e maior na especificação.

Pelo seu alto potencial em causar problema para equipe e clientes, muito se investe na busca por um perfil ou um padrão na ocorrência de erros. Sabe-se que 85% dos erros podem ser corrigidos em uma hora. É sabido que 80% do esforço é concentrado em apenas 20% do código, o que nos leva ao raciocínio de que os erros estão concentrados em partes específicas do código (PINHEIRO, 2015).

O ciclo de vida de um erro

Compreende o período entre a identificação do erro e sua efetiva correção, sempre durante o processo de testes.

Em cada fase de seu ciclo, o erro recebe uma denominação diferente, providência importante para que os envolvidos identifiquem seu estado com rapidez e correção. O ciclo inclui os seguintes estados, descritos no quadro 4.2:

Quadro 4.2 | Estados assumidos por um erro em seu ciclo de vida

Estado	Descrição	Definido por
Novo	Novo erro encontrado pelo testador	Equipe de teste
Aberto	Erro que foi revisado e confirmado como um defeito real	Líder do teste
Rejeitado	Erro que não foi confirmado como tal	Líder do desenvolvimento
Atribuído	Erro confirmado e já designado (ou atribuído) a um desenvolvedor para correção	Líder do desenvolvimento
Corrigido	Erro já corrigido e pronto para passar por novo teste	Desenvolvedor
Reaberto	Erro que se manifestou novamente durante o novo teste	Equipe de teste
Fechado	Erro que passou com sucesso pelo novo teste	Equipe de teste
Postergado	O erro será tratado em versões futuras do programa, seja por baixa prioridade ou tempo escasso	Gerente de teste do cliente

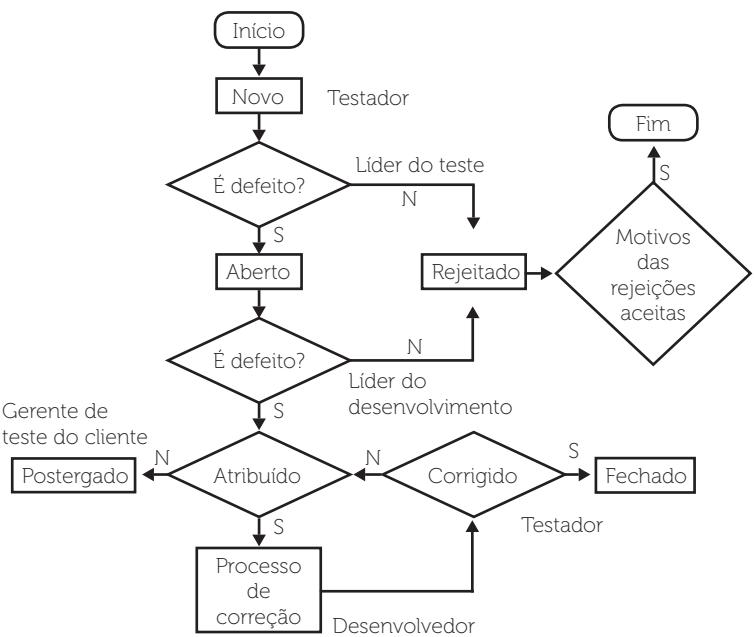
Fonte: <http://learndatamodeling.com/blog/what-is-bug-life-cycle-in-software-testing/>. Acesso em: 20 fev. 2016.

Exemplificando

Vamos ao exemplo de como o erro percorre seu caminho da descoberta até sua correção. Imagine um aplicativo de envio de *e-mail* sendo testado. Assim que o programa é terminado, o responsável confere a página de *login* e descobre que o campo de nome de usuário permite nomes duplicados. A partir da constatação, o responsável registra o erro e o reporta ao líder do teste. O líder então verifica o erro e, caso o valide, este receberá o *status* de "novo" e será passado à equipe de desenvolvimento. O líder da equipe de desenvolvimento faz nova verificação e, se for constatado que se trata de fato de um erro no programa, o erro é atribuído a um desenvolvedor – no caso um especialista em banco de dados – para correção. O desenvolvedor cria uma chave primária na tabela, altera o *status* do erro para "corrigido" e reporta a ação para o líder do teste que, por sua vez, muda o *status* para "novo teste" e atribui a tarefa de revisão a um testador. Se, depois disso, o erro tiver sido sanado, seu *status* passará para "fechado".

Como o caminho entre a descoberta do erro e sua efetiva correção percorre um caminho definido, podemos representá-lo conforme ilustrado na figura 4.3. Observe o fluxo:

Figura 4.3 | Fluxograma de ações para tratamento de um erro



Fonte: <http://learndatamodeling.com/blog/what-is-bug-life-cycle-in-software-testing/>. Acesso em: 21 fev. 2016.

Pois bem, esse é um meio possível de se tratar um erro. Sejam quais forem as ações escolhidas, elas devem ser registradas, estruturadas e, naturalmente, desempenhadas por todos os envolvidos no processo de teste. Há, no entanto, algumas outras situações que podem ocorrer durante o tratamento de um erro. Vejamos:

- O testador reporta um erro ao líder do teste, que não interpreta a situação como erro e o rejeita;
- O líder do teste valida o erro encontrado pelo testador e o reporta ao líder do desenvolvimento, que o rejeita;
- Tanto o líder do teste quanto o líder do desenvolvimento validam o erro. O desenvolvedor trabalha para corrigi-lo e reporta ao líder do teste que ele está resolvido. Mais uma vez, o testador realiza o teste e verifica que ele não foi resolvido. O erro, então, reassume o status de "aberto";
- Com base em sua prioridade ou gravidade, o erro pode ser postergado. A classificação da gravidade pode ser baixa, média, alta ou crítica.

(Disponível em: <http://learndatamodeling.com/blog/what-is-bug-life-cycle-in-software-testing/>. Acesso em: 21 fev. 2016).

Assimile

A intenção do que chamamos de Engenharia de Testes é simples: oferecer entendimento do que é um erro e oferecer formas de encontrá-lo.

Por mais que os procedimentos de teste sejam tidos como universais, não há como não os vincular ao modelo de desenvolvimento adotado para a construção do programa que será testado. Os dois modelos que já estudamos – tradicional e ágil – apresentam particularidades no trato do processo de teste, tais como quais testes serão executados e quando. Vejamos em detalhes como os modelos Cascata e *Extreme Programming* (XP) conduzem os testes.

Modelo Cascata

Teoricamente, a fase de testes nesta metodologia concentra-se logo após o término da fase de codificação do programa. Na prática,

contudo, o que se tem adotado é a revisão dos artefatos criados em cada uma das fases, sejam eles documentos, especificações, projetos ou um programa. Ao longo dos anos, a aplicação de verificações e validações (nas quais o teste está incluído) apenas no programa executável mostrou-se ineficiente, principalmente por deixar incorreções da especificação dos requisitos propagarem-se pela fase de implementação (PINHEIRO, 2015).

Na sequência, abordaremos o já conhecido TDD, meio pelo qual o XP conduz seu processo de teste.

TDD - *Test-Driven Development* (Desenvolvimento Guiado pelos Testes)

Trata-se de um formato de teste muito parecido com o "codificar e testar", modelo de desenvolvimento no qual não se dá ênfase a outras etapas, senão as de codificar e testar.

Pesquise mais

O formato "codificar e testar" é também conhecido como metodologia codifica-corrige. Pesquise mais sobre ela em: <https://marcelocoelhao.wordpress.com/2010/01/06/metodologias-de-desenvolvimento-de-software/>. Acesso em: 22 fev. 2016.

Outra fonte interessante pode ser consultada em: <http://www.devmedia.com.br/test-driven-development-tdd-simples-e-pratico/18533>. Acesso em: 26 fev. 2016.

Para a realização dos testes, o TDD apoia-se no fato de que a descrição dos requisitos – ou as estórias escritas pelo cliente – não constitui documento com excessiva formalidade e detalhamento de funções e que, por esse motivo, o cliente deve acompanhar o processo de codificação e teste. Em tempos passados, o teste era realizado tomando-se como base o código completo, ou quase completo, já que as linguagens de programação dificultavam a execução de apenas partes do programa. Hoje em dia, tecnologias que suportam linguagens orientadas a objeto (como o Java, por exemplo) permitem não só a automatização dos testes – ação tão importante no âmbito do TDD – como também a execução de partes autônomas de um programa, como uma classe, por exemplo.

O teste de parte do código segue-se à sua criação, ou seja, o teste e a integração são feitos logo após sua codificação.

Os passos do desenvolvimento guiado pelos testes incluem:

- A seleção de um conjunto de casos de teste;
- A execução do caso de teste. Encontrando-se o defeito, o código deverá ser ajustado. Caso não se encontre defeito, novo conjunto de casos de teste deve ser selecionado e o processo deve ser reiniciado (PINHEIRO, 2015).

Reflita

Por que a figura do testador é necessária no processo de testes? Por que os desenvolvedores, eles próprios, não assumem os testes? Reflita:

Os testes feitos por desenvolvedores tendem a verificar apenas trechos do código que, provavelmente, não conterão defeitos. Desenvolvedores só conseguem enxergar de 50% a 60% dos casos de teste. O testador, por sua vez, tem perfil diferente. São exploradores, gostam de encontrar problemas, são criativos nas execuções do *software* e possuem visão das diferentes situações em que o programa pode ser usado (PINHEIRO, 2015).

Esse foi o conteúdo teórico preparado para esta aula. Na parte da aula reservada às questões práticas, trataremos da aplicação de um teste funcional.

Faça você mesmo

Sua tarefa é conceber e relatar uma sequência de tratamento do erro encontrado no código, seja por meio de fluxograma, seja por meio de descrição textual. O tratamento deve incluir as ações e seus responsáveis.

Sem medo de errar

A afirmação de que a atividade de teste deve ser bem planejada e estruturada antes de ser executada, feita na aula anterior, ainda soa atual aqui. A seleção dos casos de teste foi o primeiro passo tomado pela XAX-Sooft para disciplinar o processo de teste. Agora chegou o momento de efetivamente aplicar o teste funcional, com o cuidado de estabelecer critérios para o tratamento dos erros encontrados e responsáveis por cada etapa do processo.

Como desafio proposto, sua missão então é a de aplicar o teste funcional num dos produtos da empresa, obedecendo aos critérios previamente definidos. Tome por base o cenário descrito na sequência para a resolução do caso. Como as circunstâncias não são as mesmas especificadas na seção anterior, assuma que os casos de teste apropriados para esse teste já foram selecionados.

- A equipe designada para a tarefa conta com um testador, dois desenvolvedores e um líder de desenvolvimento;
- Existem três funções a serem testadas no programa: manutenção de dados do contato, geração de relatório dos clientes mais rentáveis por período e envio automático de mensagem ao cliente aniversariante.

Uma solução possível para esse desafio é a que segue:

Com base nos casos de teste criados, o testador deverá executar o programa e conferir as saídas que cada uma das funções fornece quando acionadas.

Caso um erro seja encontrado pelo testador, ele deverá ser verificado pelo líder do desenvolvimento. Se for constatado que, de fato, se trata de um erro, sua descrição será passada a um dos desenvolvedores, com indicação em qual função ele foi encontrado.

O desenvolvedor procede a correção e repassa a informação de ajuste feito ao líder do desenvolvimento que, por sua vez, realiza nova verificação. Se, na visão do desenvolvedor e de seu líder, o erro não existir mais, nova informação de ajuste feito é emitida, desta vez ao testador, que faz a validação ou não da correção.

Durante o processo, os envolvidos deverão anotar o tempo utilizado em cada correção, em qual função o erro foi encontrado, as ocorrências de não validação do erro e as ocorrências de não validação do ajuste feito.

- Caso o código tenha sido escrito na linguagem Java, este procedimento poderá ser desenvolvido na *JUnit*, ferramenta gratuita desenvolvida por Erich Gamma e Kent Beck para criação de testes automatizados.

Atenção

Por causa das suas especificidades, o tempo empenhado nos testes e nas correções varia de função para função.

Lembre-se

Não se deve assumir que um erro foi corrigido sem que um novo teste seja feito.

Avançando na prática

Pratique mais	
Instrução Desafiamos você a praticar o que aprendeu, transferindo seus conhecimentos para novas situações que você pode encontrar no ambiente de trabalho. Realize as atividades e depois as compare com as de seus colegas.	
Ajustando o processo de teste	
1. Competência geral	Conhecer as principais metodologias de desenvolvimento de *software*, normas de qualidade e processos de teste de *software*.
2. Objetivos de aprendizagem	Conhecer os conceitos e de que forma se dá a execução do *Test-Driven Development* (TDD).
3. Conteúdos relacionados	Desenvolvimento dirigido a testes – *Test-Driven Development* (TDD): conceitos, processo e benefícios.
4. Descrição da SP	A RRHH-Soft, empresa desenvolvedora de sistemas feitos para departamentos de recursos humanos, adotou recentemente procedimento de teste em seus produtos. O processo de descoberta e correção do erro pode ser resumido assim: uma vez descoberto pelo desenvolvedor, o erro é informado a outro membro da equipe, que o corrige e relata a correção ao primeiro desenvolvedor. Mesmo com esse tratamento, os erros continuam se manifestando após a entrega feita ao cliente. Seu desafio aqui é indicar o procedimento adequado de teste, de modo a minimizar a continuidade do erro após a entrega.
5. Resolução da SP	A solução do caso requer as possíveis providências: um especialista em teste (ou testador) deve ser incorporado à equipe. As verificações feitas pelo desenvolvedor tendem a buscar funções e partes do sistema que apresentam menor chance de retornar um erro. Após a informação de que o erro foi corrigido, um outro profissional – possivelmente um segundo testador ou um líder do desenvolvimento – deve fazer nova verificação no programa ajustado. Com essas pequenas providências, o processo de teste pode evitar retrabalho.

Lembre-se

A figura do testador é indispensável no processo de teste. Programas conferidos por quem o desenvolveu tendem a continuar mantendo os erros após os testes.

Faça você mesmo

Faça a leitura e resuma o artigo encontrado em: <http://revistapensar.com.br/tecnologia/pasta_upload/artigos/a80.pdf>. Acesso em: 8 mar. 2016. Ele trata da importância de uma equipe de testes de *software* presente em uma organização, além de abordar o processo de teste.

Faça valer a pena

1. Em relação à Engenharia de Testes, assinale a alternativa que contém a afirmação verdadeira.

a) Ramo da Engenharia de *Software* que cuida da busca e formação de engenheiros de testes.

b) Área que cuida da criação de fluxogramas para processos de teste.

c) Área que cuida da busca e entendimento do erro em um *software* e das formas de eliminá-lo.

d) Trata-se de expressão sinônima de Engenharia de *Software*.

e) Área que tem como objetivo eliminar a necessidade de testes em um processo de desenvolvimento.

2. Em relação às ocasiões em que um erro ocorre, analise as afirmações a seguir:

I - Um erro ocorre quando o usuário falha na operação do sistema.

II - Um erro ocorre quando o sistema executa uma operação não prevista nos requisitos.

III - Um erro ocorre quando o sistema não executa uma função prevista nos requisitos.

IV - Um erro ocorre quando o sistema executa uma função vetada nos requisitos.

É verdadeiro o que se afirma apenas em:

a) I e IV.

b) II e III.

c) IV.

d) II.

e) II, III e IV.

3. Em relação ao ciclo de vida de um erro, analise as afirmações a seguir:

I - O ciclo de vida de um erro acompanha o ciclo de vida do sistema, de acordo com o modelo de desenvolvimento adotado.

II - Um erro rejeitado é aquele comportamento supostamente anômalo cujo teste mostra não ser um erro.

III - Um erro atribuído é aquele já corrigido.

IV - Um erro fechado é aquele sobre o qual não se pode aplicar métodos de correção, por não estar acessível ao testador.

É verdadeiro o que se afirma em:

a) II, apenas.

b) I, II, III e IV.

c) II e IV, apenas.

d) II e III, apenas.

e) I, apenas.

Seção 4.3

Os testes de *release* e de usuário

Diálogo aberto

Nesta etapa do nosso estudo, já concordamos que os testes são fundamentais no processo de desenvolvimento de um produto. Sem que as devidas verificações sejam feitas, não há a mínima segurança de que o sistema se comportará conforme o esperado.

A expressão "devidas verificações" demanda atenção especial de nossa parte: ela indica, entre outras coisas, a necessidade de aplicação do teste certo, pela pessoa certa e no tempo devido.

Antes de iniciarmos a abordagem teórica, é bom que nos lembremos que temos adquirido, ao longo do curso, conhecimento das principais metodologias de desenvolvimento de *software*, das normas de qualidade e, nesta unidade, dos processos de teste de *software*. Temos como objetivo específico, nesta aula, conhecer o que são e como implementar o teste de *release* e o teste de usuário.

Conforme constataremos na sequência, não há apenas um tipo de teste a ser feito no produto. As verificações devem focar desde o desempenho do programa até sua adequação à máquina em que será executado. Não se pode conceber também que apenas uma pessoa ou grupo realize os testes. Diferentes pontos de vista e interesses devem ser acomodados pelo processo, fato que também deu motivo para a criação de várias modalidades de teste.

Ainda resta uma questão a ser abordada: é válido e útil o registro dos resultados obtidos no teste? Podemos considerar esses registros base para criação de histórico do processo? Desta maneira, será nesta seção que abordaremos esses assuntos que circunstanciam o processo de teste e que serão argumento para o desafio desta aula.

Novamente se faz necessário resgatarmos o desafio que dá sentido ao nosso curso: fazer com que a *XAX-Sooft* atinja bom nível

de excelência em seus processos e produtos, tendo como ponto de partida uma organização sem processo definido e sem qualquer padronização de qualidade. Em específico, nossa missão nesta seção é fazer o registro dos resultados obtidos com a aplicação do teste de *software* como meio eficiente de formar uma base para manutenção de histórico dos processos. Pelos próximos parágrafos você terá contato com a base teórica sobre modalidades de teste e sobre registros de atividades no processo.

Bons estudos!

Não pode faltar

É justamente por causa de sua elevada importância e criticidade que a atividade de teste conquistou grande abrangência no processo de desenvolvimento de um produto. Os testes há muito não estão mais restritos apenas às ações da equipe de desenvolvimento e as ocasiões em que são executados não se limitam simplesmente à pós-codificação.

Diversas modalidades de testes vêm sendo aplicadas por diferentes atores do processo, o que visa garantir que pessoas com diferentes visões do produto e em tempos distintos possam avalizá-lo.

Os testes escritos para o usuário final (ou para o cliente) serão abordados nos próximos parágrafos e seu conteúdo teórico dividirá espaço com outros conceitos relacionados à atividade e com boas práticas de análise dos resultados obtidos pela aplicação de um teste funcional, de modo a capacitá-lo para superar o desafio proposto nesta seção. Ao trabalho!

Testes de unidade

Para entender um teste de unidade, imagine o *software* como um conjunto de partes que, juntas, formam o todo. Essa modalidade de teste é direcionada a uma rotina, classe ou pequena parte de um produto e é normalmente executada pelo próprio desenvolvedor, de modo a assegurar que determinada unidade poderá ser integrada ao resto do *software*.

No contexto dos testes de unidade (ou testes unitários) insere-

se um elemento conhecido como *stub*. Ele simula resultados que são retornados por um determinado componente do qual o *software* depende (PINHEIRO, 2015). Em outras palavras, um *stub* é um trecho de código que substituirá as entradas, dependências e comunicações que a unidade deveria receber em uma execução do programa. Quando um componente A que vai ser testado chama operações de outro componente *B* que ainda não foi implementado, pode-se criar uma implementação simplificada de *B*, chamada *stub*. Neste caso, devemos entender que o componente *A* é a unidade a ser testada.

Exemplificando

Veja um bom exemplo de aplicação de um *stub*: Suponha que o componente *A* é uma classe que deve usar um gerador de números primos *B*. A implementação de *B* ainda não foi feita. No entanto, para testar a unidade *A* não será necessária a geração de alguns poucos números primos. Assim, *B* pode ser substituído por uma função *stub* que gera apenas os 5 primeiros números primos (WAZLAWICK, 2013).

Há situações, no entanto, em que o módulo *B* já está implementado, mas o módulo *A* (que aqui representa nossa unidade) que chama as funções de *B* ainda não foi implementado. Deverá ser implementada, então, uma simulação do módulo *A*, que recebe o nome de *driver*. A diferença entre *stubs* e *drivers* concentra-se na relação de dependência entre os componentes.

As técnicas de teste funcional e estrutural, ambas abordadas anteriormente nesta seção, podem ser utilizadas para a execução de um teste de unidade. O teste funcional serve para validar a função específica que está sob teste. Quando utilizada, a técnica de teste estrutural (ou caixa-branca) viabiliza a validação dos fluxos de execução da unidade. Ambas as técnicas podem ser utilizadas de forma combinada ou isolada num teste de unidade.

Reflita

Os *stubs*, como sabemos, geram valores de entrada para unidades que serão testadas. Você entende como legítimo que um *stub* seja construído também para gerar valores errados para a classe, por exemplo?

Testes de integração

Trata-se de teste executado pelo testador para garantir que vários componentes do sistema funcionem corretamente juntos (PINHEIRO, 2015). Eles são feitos quando as unidades (as classes, por exemplo) já foram testadas de forma isolada e precisam ser integradas para gerar uma nova versão do *software*.

No caso de as classes a serem testadas precisarem de comunicação com outros componentes ou classes que ainda não foram testadas – ou mesmo implementadas –, os *stubs* mais uma vez serão necessários. O problema com um *stub* é que se investe tempo para desenvolver uma função que não será efetivamente entregue. Além disso, nem sempre é possível saber se a simulação produzida pelo *stub* será suficientemente adequada para os testes (WAZLAWICK, 2013).

Com o sistema integrado e testado, chega o momento de entregá-lo ao usuário para que ele o teste também.

Teste de usuário (ou teste de aceitação)

Quando já se dispõe da interface final do sistema, o teste de aceitação já pode ser aplicado. Como o próprio nome nos faz supor, esse teste é executado pelo usuário e não pela equipe de testadores ou desenvolvedores. Para entendermos melhor essa modalidade de teste, é interessante que a coloquemos em oposição ao teste de sistema.

O **teste de sistema** é feito quando todas as unidades e as integrações entre elas já foram testadas. Pode ser que a equipe já se sinta confiante o bastante nesse ponto para assumir que seu produto é de boa qualidade. No entanto, é necessário que ele passe por esse novo teste depois de integrado. O objetivo da sua aplicação é o de verificar se o programa é capaz de executar processos completos, sempre adotando o ponto de vista do usuário.

Pois bem, o teste de usuário pode ser planejado tal qual o teste de sistema. A diferença é que ele será executado pelo usuário. Em outras palavras, enquanto o teste de sistema faz a *verificação* do sistema, o teste de aceitação faz sua *validação*.

Exemplificando

Um plano viável para a realização de teste de aceitação em um programa de vendas pela internet é o que segue no quadro 4.3.

Quadro 4.3 | Exemplo de roteiro para teste de aceitação

Como testar	Resultado
Um cliente cadastrado informa livros válidos, indica um endereço e cartão de crédito válidos e a operadora (um *stub*, possivelmente) autoriza a compra.	Compra efetuada.
Um cliente cadastrado informa livros válidos e guarda o carrinho.	Carrinho guardado.
Um cliente não cadastrado informa livros válidos e guarda o carrinho.	O cliente é cadastrado e o carrinho é guardado.
Um cliente cadastrado informa livros válidos, indica um endereço inválido e depois um endereço válido; indica um cartão válido e a operadora autoriza a compra.	O endereço inválido é atualizado e a compra é efetuada.
Um cliente cadastrado informa livros válidos, indica um endereço e cartão válidos e a operadora não autoriza a compra. O cliente informa outro cartão válido e a operadora autoriza a compra.	Compra efetuada com o segundo cartão informado.

Fonte: adaptado de Wazlawick (2013, p. 296).

O teste de aceitação é chamado de **Teste Alfa** quando feito pelo cliente sem planejamento rígido ou formalidades. Se o teste é ainda menos controlado pelo time de desenvolvimento e as versões do programa são testadas por vários usuários, sem acompanhamento direto ou controle da desenvolvedora, então o procedimento é chamado de **Teste Beta**. Nesta modalidade, as versões disponibilizadas costumam expirar depois de certo tempo de uso.

Assimile

O teste de aceitação visa à validação dos requisitos implementados no *software*, não mais a verificação de defeitos (WAZLAWICK, 2013).

Um termo bastante comum no contexto dos testes de aceitação é o ***release***. Literalmente, o termo refere-se a uma liberação ou lançamento de uma nova versão de um produto de *software*. Os testes aplicados sobre um *release* seguem o padrão dos testes de usuário. Usualmente, um lançamento obedece às seguintes fases:

- **Alfa**: nesta fase, o lançamento ainda está em processo de testes, que são realizados por pessoas situadas normalmente fora do processo de desenvolvimento. Os produtos em fase de teste alfa podem ser instáveis e sujeitos a travamento.

- **Beta**: trata-se da classificação posterior a Alfa. Um lançamento em beta teste é avaliado por testadores beta, normalmente clientes em potencial que aceitam a tarefa de teste sem cobrar por isso. Uma versão Beta é utilizada para demonstrações dentro da organização e para clientes externos.

- ***Release Candidate***: trata-se de versão do sistema com potencial para ser a versão final. Neste caso, todas as funcionalidades já foram devidamente testadas por meio das fases Alfa e Beta.

Um *release* pode ser interno ou externo. O primeiro é usado somente pela equipe interna de desenvolvimento para fins de controle ou para demonstração a clientes e usuários. Um *release* externo é uma versão do produto distribuída para os usuários finais (BARTIÉ, 2002).

De uma forma ou de outra, as modalidades de teste apresentadas até aqui relacionam-se diretamente às funcionalidades do sistema. Em outras palavras, elas fazem a verificação dos processos que efetivamente devem ser realizados pelo programa. No entanto, a abrangência dos testes é maior e demanda a aplicação de outros tipos de verificações no produto. A esses tipos damos o nome de testes suplementares. Vejamos dois exemplos:

a) **Teste de Desempenho**: tipo que tem por objetivo determinar se, nas situações de pico máximo de acesso e concorrência, o desempenho ainda permanece consistente com o que foi definido para essa característica do sistema. Assim, o critério de sucesso aqui é a obtenção de tempo de resposta compatível com o que se espera do produto, quando o sistema trabalha em seu limite. Um plano possível para esse teste está descrito a seguir (BARTIÉ, 2002):

- Validar todos os requisitos de desempenho identificados;

- Simular *n* usuários acessando a mesma informação, de forma simultânea;

- Simular *n* usuários processando a mesma transação, de forma simultânea;

- Simular *n%* de tráfego na rede;
- Combinar todos esses elementos.

Quando o procedimento eleva ao máximo a quantidade de dados e transações às quais o sistema é submetido, o teste realizado é chamado de **Teste de Estresse**. Ele é realizado para que se possa verificar se o sistema é suficientemente robusto em situações extremas de carga de trabalho.

b) **Teste de recuperação**: seu objetivo é avaliar o *software* após a ocorrência de uma falha ou de uma situação anormal de funcionamento. Algumas aplicações demandam alta disponibilidade do programa e, no caso de falha, o *software* deve ter a capacidade de se manter em execução até que a condição adversa desapareça.

Os testes de recuperação devem prever também os procedimentos de recuperação do estado inicial da transação interrompida, impedindo que determinados processamentos sejam realizados pela metade (BARTIÉ, 2002).

Normalmente, o teste de recuperação trata de situações referentes a (WAZLAWICK, 2013):

- Queda de energia na organização em que o sistema está em funcionamento;
- Discos corrompidos;
- Problemas de queda de comunicação;
- Quaisquer outras condições que possam provocar a terminação anormal do programa ou a interrupção temporária em seu funcionamento.

Pesquise mais

É possível que, ao se aplicar uma manutenção num programa, outros defeitos sejam gerados no código? Acertou se respondeu que sim. Pesquise sobre Teste de Regressão para saber mais. Comece por: <http://gtsw.blogspot.com.br/2009/03/importancia-dos-testes-de-regressao.html> e <http://www.testavo.com.br/2010/05/desmistificando-testes-de-regressao.html>. Acesso em: 1 mar. 2016.

Pois bem, eis então algumas das outras modalidades de teste e outros conceitos relacionados à atividade. Chega a hora, então, de tratarmos dos registros das atividades de teste, visando capacitá-lo ainda mais para o desafio desta seção. Vamos a eles?

Os relatórios da qualidade do *software*

O registro das atividades relacionadas ao processo de qualidade, sobretudo os testes, são feitos em relatórios específicos. Além do efetivo registro, eles servem também como instrumentos de medição e análise. A execução do teste deve gerar o que chamamos de *log*, que nada mais é do que o registro das atividades desempenhadas. Vamos a alguns deles:

Log de execução: este documento é criado para registrar todos os procedimentos realizados durante a execução de um ciclo de testes, bem como apontar as eventuais interrupções ocorridas. O *log* de execução certifica que, de fato, o teste foi realizado.

Ocorrências da validação: neste documento registram-se todas as ocorrências geradas durante um teste. Uma ocorrência pode ser, por exemplo, a identificação de um defeito. Neste caso, alguns dos dados a serem relatados serão: um nome ou número de identificação do defeito, a data em que foi encontrado e a sequência de ações capazes de reproduzi-lo (BARTIÉ, 2002).

Um dado também bastante importante a ser registrado é a classificação do defeito. Apesar de haver muitas classificações, alguns deles são mais relevantes:

a) **Falha na descrição funcional**: esta classificação refere-se à discrepância entre a descrição da funcionalidade e sua efetiva implementação. Por exemplo: a descrição da funcionalidade estabelece que o programa deveria promover acréscimo trimestral automático do salário dos gerentes de seção e o sistema aplica o aumento a cada quatro meses;

b) **Falha no controle, lógica ou sequenciamento do algoritmo**: um laço de repetição infinito é um exemplo desse tipo de falha;

c) **Falha nas estruturas de dados**: trata-se da definição incorreta, por exemplo, de matrizes e tabelas de banco de dados;

O relatório de teste, em resumo, serve para registrar a maior quantidade possível de dados acerca do processo.

Faça você mesmo

A norma IEEE 829 é o padrão criado pela IEEE para documentação do processo de teste. Um dos relatórios recomendados pelo padrão é o de incidente de teste, que prevê a descrição dos seguintes itens, entre outros, em caso de ocorrência de um incidente: entradas, resultados esperados e resultados encontrados. Vale registrar que um incidente pode ser entendido como discrepância entre o resultado esperado e o encontrado na execução dos casos de teste. Crie um exemplo de um processo de teste que contemple uma entrada de caso de teste, faça previsão de um resultado esperado para aquela entrada e forneça a descrição do resultado obtido.

Sem medo de errar

A caminhada da *XAX-Sooft* rumo à excelência não para! Depois de fazer a seleção dos casos de teste e de efetivamente aplicar teste funcional em um dos seus produtos, a *XAX-Sooft* vê-se na necessidade de fazer um registro criterioso das atividades desempenhadas no processo de teste, com a finalidade de criar base de comparação para testes futuros. Afinal, como saber se a aplicação de um método foi bem-sucedida se não pela comparação entre dois ou mais resultados?

Vale a pena dizer que, na prática, os registros dos resultados obtidos no teste são feitos no ato da sua aplicação ou em situação inserida em seu contexto. Por exemplo, as anotações referentes à ocorrência de uma falha no algoritmo devem ser feitas assim que ela acontece, no momento do teste. Para efeito de organização didática, esta seção coloca o registro das ocorrências e resultados como etapa posterior à aplicação do teste, embora não o seja de fato.

Pois bem, o desafio lançado aqui é justamente o de registrar o processo e os resultados obtidos com a aplicação do teste de *software* em um dos seus produtos. O que deve ser registrado? O que não deve ser registrado? Há formato definido?

Uma solução possível para esse desafio é a seguinte:

- A equipe deve criar, oficializar e publicar um *log* de execuções, ou seja, um formulário apropriado para que todos os registros de ocorrências sejam feitos;
- O formulário de registro deve conter, no mínimo, os seguintes campos:

Responsáveis pelo teste

Data e horário de início

Data e horário de término

Função, unidade ou sistema testados

Fase do teste: teste de unidade, teste de integração ou teste de sistema.

Falha na interpretação da função: nesta seção devem ser relatadas as divergências entre a função especificada nos requisitos e a função de fato implementada.

Falha na construção de estrutura de dados: erros na criação de tabelas e outras estruturas devem ser registrados aqui.

Defeito algorítmico: devem ser identificados erros de lógicas, laços infinitos, por exemplo.

Travamento de origem não identificada

Outras ocorrências relevantes

Esse formulário pode – e deve – passar por constante aprimoramento em seus campos e em sua utilização.

Atenção

O formato do *log* de execuções é livre e deve ser definido pela equipe, segundo critérios próprios.

Lembre-se

A norma IEEE 829 é o padrão criado pela IEEE para documentação do processo de teste.

Avançando na prática

Pratique mais	
Instrução Desafiamos você a praticar o que aprendeu, transferindo seus conhecimentos para novas situações que você pode encontrar no ambiente de trabalho. Realize as atividades e depois as compare com as de seus colegas.	
Incluindo nova etapa no processo de teste	
1. Competência geral	Conhecer as principais metodologias de desenvolvimento de *software*, normas de qualidade e processos de teste de *software*.
2. Objetivos de aprendizagem	Conhecer o que são e como implementar o teste de *release* e o teste de usuário.
3. Conteúdos relacionados	Testes de *release* e de usuário: conceitos, características e tipos.
4. Descrição da SP	A Mimimi-Soft, empresa desenvolvedora de sistemas para consultórios médicos, adota procedimento de teste que prevê a verificação isolada de cada função do sistema. Na ocorrência de erro, ele é registrado, atribuído e corrigido. Após sua integração, o programa é submetido ao cliente para teste de aceitação. Mesmo com os procedimentos seguidos como se deve, muitas ocorrências de erros continuam se manifestando no teste de aceitação, causando frustração no cliente e na equipe. Seu desafio aqui é indicar procedimento adequado de teste, com as etapas corretamente estipuladas, de modo a minimizar a continuidade dos erros nos testes de aceitação.
5. Resolução da SP	A solução para o caso requer que seja incluída etapa de **teste de integração** no processo, com o objetivo de verificar as comunicações e interfaces entre os módulos ou unidades do sistema. Sem a execução dessa etapa, não haverá confiança suficiente de que os dados de saída de uma função, por exemplo, estejam no formato e quantidade adequados para servirem como dados de entrada de outra função, subordinada àquela.

Lembre-se

No teste de integração, os módulos são combinados e testados conjuntamente. Trata-se de etapa indispensável no processo e que antecede o teste de sistema.

Faça você mesmo

Imagine que você crie um sistema, submeta-o a teste, certifique-se de que ele funciona corretamente e, tempos depois que o instala no servidor situado na organização do cliente, descobre que seu funcionamento não está correto por causa de questões relacionadas ao *hardware*.

Para evitar essa situação, é costume proceder o **teste de instalação**. Pesquise em *sites* e livros especializados mais sobre o conceito e sobre o procedimento correto para esse tipo de teste.

Faça valer a pena

1. Em relação ao Teste de Unidade, assinale a alternativa que contém a afirmação verdadeira.

a) Modalidade de teste que toma o sistema todo como uma unidade e aplica sobre ele um teste funcional.

b) Atividade de teste que consiste em verificar se um componente individual do *software* (ou unidade) foi corretamente implementado.

c) Atividade de teste que consiste em verificar se várias unidades interligadas foram corretamente implementadas.

d) Tipo de teste em que o cliente valida uma única funcionalidade do sistema.

e) Tipo de teste em que o cliente valida várias unidades interligadas do sistema.

2. Em relação a um *stub*, analise as afirmações a seguir:

I - Trata-se de um tipo de defeito que ainda não foi revelado pelos testes.

II - Um *stub* é um trecho de código que substituirá as entradas, dependências e comunicações que a unidade deveria receber em uma execução do programa.

III - Um *stub* não será aproveitado como parte integrante do sistema que será entregue ao cliente.

IV - Um testador reconhece um *stub* pelo tipo de manifestação do defeito durante a execução do programa.

É verdadeiro o que se afirma apenas em:

a) I e IV.

b) II e III.

c) IV.

d) II.

e) II, III e IV.

3. Em relação ao teste de aceitação, analise as afirmações a seguir:

I - O teste de aceitação é executado pela equipe de desenvolvimento e constitui a última tentativa da equipe de descobrir defeitos no código antes da entrega.

II - O teste de aceitação é executado pelo usuário final e visa à validação dos requisitos previamente estipulados.

III - Executado o teste de aceitação, o cliente não poderá ainda utilizar o produto, já que ele deverá passar pelo teste de sistema antes da entrega.

IV - O teste de aceitação é executado pelo usuário final e visa à verificação de falhas ainda não manifestadas no programa.

É verdadeiro o que se afirma em:

a) II, apenas.

b) I, II, III e IV.

c) II e IV, apenas.

d) II e III, apenas.

e) I, apenas.

Seção 4.4

A manutenção e evolução de *software*

Diálogo aberto

Aqui estamos, em nossa última aula.

No trajeto que se iniciou pelo desenvolvimento de produtos com base apenas no talento individual dos seus desenvolvedores até o atingimento da maturidade processual, a *XAX-Sooft* percorreu um longo caminho.

Assim que constatou a necessidade de organizar seus processos, a empresa adotou e consolidou metodologia consagrada de desenvolvimento de *software* e, no momento seguinte, enxergou a necessidade de evoluir para um modelo ágil e moderno de criação de programas. Ato contínuo, passou a adotar medidas e padrões de qualidade com fins de atingir a excelência em seus produtos. Por fim, como parte do esforço para aprimoramento da qualidade, implantou procedimento formal de testes. Você, claro, foi figura fundamental nessa transformação.

Este é o retrato da *XAX-Sooft* atual: uma empresa sólida, reconhecida em seu meio, amadurecida em seus processos e preocupada com sua constante evolução. E é justamente de evolução que trata esta última seção. Como meio de completar seu ciclo virtuoso de desenvolvimento, a organização pretende adotar meios eficientes de manutenção dos seus produtos e implantar modelo para estimar o esforço que esta atividade acarretará à equipe.

Em resumo, o objetivo desta seção é abordar a manutenção e a evolução do *software* e o desafio que se propõe é justamente implantar um modelo de estimação de esforço de manutenção num dos produtos da *XAX-Sooft*. Com essa medida, a manutenção deixará de ser encarada como um retrabalho e passará a ser tida como meio de proporcionar evolução ao produto.

Por fim, nesta última aula, vale a pena resgatarmos as competências que motivaram nosso estudo até aqui. Você está lembrado? Pois bem, esperamos que o conhecimento das principais metodologias de desenvolvimento de *software*, normas de qualidade e processos de teste de *software* tenha sido adquirido.

Parabéns pelo bom trabalho feito até aqui e vamos em frente!

Não pode faltar

A expressão manutenção de *software*, quando entendida da forma costumeira, remete apenas a correções no programa provocadas pela ocorrência de erros durante a operação feita pelo usuário final. No entanto, essa atividade pode e deve assumir caráter mais significativo do que o meramente corretivo. Ao mudar sua concepção sobre a atividade, a equipe poderá colocar-se diante da oportunidade de fazer o produto evoluir sem fazê-lo perder suas características principais, o que certamente terá boa consequência na utilidade do produto e na satisfação do cliente.

Quando estudamos o modelo tradicional de desenvolvimento, o tema foi abordado principalmente de forma conceitual e circunstanciado como parte final do ciclo de criação de um programa. Nesta seção, contudo, você terá embasamento para perceber a manutenção de forma diferente e poderá ter contato com um meio de dimensionar o esforço que deverá ser exigido para a conclusão de uma atividade de manutenção. Ao trabalho!

A manutenção pós-entrega

Assim que o produto passa pelo teste de aceitação, ele está apto a ser entregue ao cliente. Embora longe de ser considerada o encerramento definitivo do ciclo do *software*, a entrega marca o término do desenvolvimento de uma versão apta a ser utilizada em meio produtivo e adequada ao propósito de resolver as situações colocadas pelo cliente como requisitos do produto.

Esse marco no processo caracteriza-se também por ser o ponto de início de outra etapa igualmente desafiadora: a manutenção pós-entrega.

Hoje em dia, a expressão "manutenção de *software*" vem sendo substituída ou usada em conjunto com "evolução de *software*". Evolução talvez seja o termo mais adequado, já que as atividades de modificação de um produto que já está em operação não visam mantê-lo em seu estágio atual, mas fazê-lo evoluir de forma a adaptar-se a novos requisitos ou ainda corrigir defeitos (WAZLAWICK, 2013). Usaremos aqui as duas expressões como sinônimas.

Reflita

É correto imaginar que, após seu desenvolvimento, um *software* terá seu valor diminuído com o passar do tempo? Se imaginou que sim, está correto. Conforme falhas são descobertas no programa, conforme seus requisitos originais mudam e conforme produtos menos complexos, mais eficientes e mais avançados são lançados, o valor (não necessariamente financeiro) do *software* original vai se perdendo. Daí a necessidade de se aplicar melhorias que façam o produto evoluir.

Conforme estudamos na primeira unidade, existem basicamente três razões que justificam o investimento na aplicação de modificações em um produto. Uma falha de compreensão de um requisito, de projeto, codificação, de documentação ou de qualquer outro tipo deve ser corrigida. Tal ação configura uma **manutenção corretiva**. Quando o código ou outro artefato é modificado para que a eficiência do produto aumente, temos uma **manutenção de aperfeiçoamento**, ou perfectiva. Se o programa deve se adaptar a uma mudança no ambiente em que ele opera (uma mudança na quantidade de dígitos das linhas telefônicas, por exemplo), então ocorre uma **manutenção adaptativa** (SCHACH, 2008).

É saudável que toda iniciativa de manutenção passe pela avaliação de viabilidade em se aplicar modificações em um programa ou se construir um novo produto. Fatores como dificuldades técnicas na manutenção, inadequação do produto original às necessidades do cliente e custo devem ser levados em conta antes da decisão.

Exemplificando

A diferença entre a necessidade de novo desenvolvimento e a necessidade de aplicação da manutenção em produto já existente pode parecer clara, mas vale a pena contrastarmos as duas situações,

considerando outros fatores que não o aspecto técnico apenas. Leia com atenção o que segue.

Imagine que uma mulher teve seu retrato pintado aos 18 anos. Anos mais tarde, já casada, ela decide que o marido deve ser incluído no quadro, ao seu lado. Grandes dificuldades acompanham essa decisão:

- A tela não é grande o suficiente para acomodar o marido ao lado dela;
- Com o tempo, as cores do quadro ficaram menos vivas. Haveria possibilidade de retoque, mas a tinta usada já não é mais fabricada;
- O artista original se aposentou e foi morar em outro país;
- O rosto da mulher envelheceu e não há garantia de que a figura do quadro possa ficar parecida com a feição atual da mulher.

Considere agora a manutenção de um programa de computador que custa R$ 2 milhões para ser desenvolvido. Quatro dificuldades se apresentam:

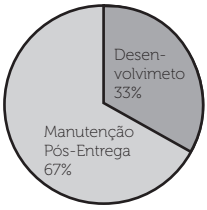
- O disco rígido no qual a base de dados está armazenada está quase cheio e novos dados não podem mais ser incluídos;
- A empresa que fabricava o disco faliu e outro disco de maior capacidade, de outro fabricante, já foi comprado. No entanto, há incompatibilidade entre o novo disco e o *software* e as adequações custarão R$ 100 mil;
- Os desenvolvedores originais já se aposentaram e a manutenção deverá ser feita por profissionais que jamais cuidaram do programa antes;
- O produto original foi construído por meio do uso do modelo clássico de desenvolvimento. No entanto, há real necessidade de que o paradigma de orientação a objeto seja usado no produto que surgirá depois da manutenção.

Percebe a correspondência entre os itens do quadro e os itens do programa? A conclusão para o primeiro caso é que, partindo da estaca zero, um outro artista deverá pintar o retrato do casal para atender ao pedido da mulher. Isso também significa que, ao invés da manutenção de R$ 100 mil, um novo produto de R$ 2 milhões deve ser criado? Independentemente de estarmos tratando de quadros ou de programas de computador, muitas vezes parecerá mais simples e viável a criação de um novo produto. No entanto, considerações financeiras podem tornar a manutenção muito mais imediata do que um novo desenvolvimento (SCHACH, 2008).

Por mais improvável que possa parecer, a manutenção pós-entrega consome mais tempo e recurso do que qualquer outra atividade do ciclo do produto. Atualmente, aproximadamente dois terços do custo total de um programa estão relacionados à manutenção (SCHACH, 2008). As Figuras 4.4a e 4.4b mostram a relação de custos entre atividades de desenvolvimento e manutenção em dois momentos passados.

Figura 4.4a | Relação entre custo e manutenção entre 1976 e 1981

Dados obtidos entre 1976 e 1981

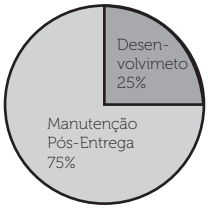


Desenvolvimento
Manutenção Pós-Entrega

Fonte: Schach, 2008.

Figura 4.4b | Relação entre custo e manutenção entre 1992 e 1998

Dados obtidos entre 1992 e 1998



Desenvolvimento
Manutenção Pós-Entrega

Fonte: Schach, 2008.

Fica claro que o investimento em técnicas, ferramentas e boas práticas que levem à redução desse custo se justifica plenamente. Uma dessas boas práticas é a incorporação da facilidade de manutenção do produto desde o início do seu desenvolvimento, por meio da correta modularização do produto e das atividades de verificação e validação desempenhadas durante todo o processo.

A correção de um erro apontado pelo usuário final demanda conhecimento não só do código-fonte, mas de todos os demais aspectos relacionados à sua produção, tais como as especificações dos requisitos, o projeto e a documentação relacionada.

Assimile

Já que o produto de *software* é formado por outros elementos além do código-fonte, qualquer alteração nos manuais feita após a entrega pode ser considerada manutenção.

Na Seção 4.2 estudamos tratamentos possíveis para erros no programa descobertos durante seu processo de desenvolvimento. Embora estejamos tratando agora de um programa já entregue, não haveria motivo relevante o suficiente para nos fazer mudar o fluxo para correção do erro neste atual contexto.

Como qualquer atividade desenvolvida durante a criação de um produto de *software*, a manutenção também requer processo formal e gerenciamento. Um elemento bastante importante neste gerenciamento é o **relatório de defeitos**. Via de regra, ele é preenchido pelo usuário final e contém informações suficientes para permitir que o programador de manutenção recrie o problema descoberto pelo cliente. Confirmada a existência do erro, o programador poderá classificá-lo como crítico, importante, normal, secundário e trivial, conforme sua relevância (SCHACH, 2008).

Exemplificando

Se um programa usado para folha de pagamento apresenta erro um dia antes da data do cálculo dos salários dos funcionários, então estaremos diante de um erro crítico e a intervenção da equipe deverá ser imediata.

Há, no entanto, a possibilidade de o usuário final ter interpretado mal um item do manual do produto e, por isso, entendido que determinada situação relacionada a ele se tratava de um erro. Neste caso, ao invés de correção, cabe apenas esclarecimento.

Suponhamos que o programador de manutenção tenha constatado a anomalia no programa. Ele deverá iniciar, então, a correção, sempre tomando precaução para que o processo não acabe introduzindo novos erros.

Assimile

A inclusão ou alteração de uma função ou módulo no sistema requer a execução de **testes de regressão**, que visam evitar erros de integração com os módulos originais do sistema.

Feita a correção, o programador (ou outro membro da equipe) deverá realizar novo processo de teste, objetivando assegurar que o problema original foi sanado e nenhum outro foi criado em decorrência do ajuste.

São claras, portanto, as semelhanças entre o processo de correção de um erro antes da entrega do programa e o processo de correção desenvolvido após a entrega. No primeiro caso, entretanto, o testador é o responsável principal pela descoberta do erro. No segundo, o próprio usuário final o encontrará.

Lembre-se

Já que o produto de *software* é formado por outros elementos além do código-fonte, qualquer alteração nos manuais feita após a entrega pode ser considerada manutenção.

Pois bem, parece-nos claro agora que a manutenção é atividade fundamental no ciclo de vida do *software* e que, dada a sua contribuição para o aprimoramento constante do produto, ela justifica plenamente ser chamada de evolução do *software*. Em que pese sua importância no processo, há que se pensar no esforço necessário para empreender tal atividade. Afinal, como você pode imaginar, investimento em recursos humanos e financeiros sempre serão necessários.

Na sequência, será abordado um modelo de estimação de esforço de manutenção. Através da aplicação dessa técnica, você poderá quantificar qual o trabalho previsto para a atividade, com base na porcentagem de linhas de código que sofrerão alteração.

ACT - Modelo de Estimação de Esforço de Manutenção

O modelo ACT (*Annual Change Traffic* ou Tráfego Anual de Mudança) foi proposto por Boehm (1981) e se baseia em uma estimativa de porcentagem de linhas de código que passarão por manutenção. Para efeito de contagem, são consideradas como *linhas em manutenção* tanto as linhas a serem alteradas quanto as novas linhas criadas. O valor da variável ACT reflete o número de linhas que sofrem manutenção dividido pelo número total de linhas do código em um ano típico (WAZLAWICK, 2013). A fórmula criada é $E = ACT * SDT$, em que E representa o esforço, medido em desenvolvedor/mês, a ser aplicado no período de um ano nas atividades de manutenção, *ACT* representa a porcentagem esperada de linhas modificadas ou adicionadas durante um ano em relação ao tamanho do *software* e *SDT* é o tempo de desenvolvimento do *software*, ou *Software Development Time*.

Pesquise mais

Entre os anos de 1974 e 1996, o pesquisador alemão Meir M. Lehman publicou oito leis que colocam a evolução do *software* como necessária e inevitável. Você pode pesquisar sobre o tema em:

<http://homepages.dcc.ufmg.br/~figueiredo/disciplinas/aulas/evolucao-leis-lehman_v01.pdf>. Acesso em: 10 mar. 2016.

<http://www.rafaeldiasribeiro.com.br/downloads/testesw5.pdf>. Acesso em: 24 mar. 2016.

WAZLAWICK, R. S. **Engenharia de Software**: conceitos e práticas. Rio de Janeiro: Elsiever, 2013, páginas 318 e seguintes.

Como exemplo, se tomarmos um programa que foi desenvolvido com o esforço de 80 horas de desenvolvimento/mês, seu *SDT* será igual a 80. Se a taxa anual esperada de linhas em manutenção (*ACT*) for de 2%, então o esforço anual esperado de manutenção para esse programa será dado por $E = 0{,}02 * 80$; $E = 1{,}6$.

De acordo com essa técnica, espera-se esforço anual de 1,6 horas de desenvolvimento/mês destacadas para atividade de manutenção (WAZLAWICK, 2013).

É claro que você deve considerar a criticidade da manutenção, a experiência dos programadores, a complexidade do programa e outros fatores que poderão influenciar na necessidade de mão de obra. Contudo, não se pode negar que essa estimativa irá te ajudar a ter boa noção da quantidade de programadores que deverá destacar para a atividade.

Pois bem, chegamos assim ao fim do nosso conteúdo teórico. Torcemos vivamente para que a compreensão dos assuntos abordados em nossos 16 encontros tenha sido satisfatória e que o conhecimento adquirido seja decisivo na realização do seu projeto de vida. Fica a seguinte sugestão: não pare por aqui! A Engenharia de *Software* é um campo vastíssimo e tópicos mais avançados desta disciplina devem merecer sua atenção.

Faça você mesmo

O modelo de estimativa de esforço de manutenção chamado CII ou Cocomo II toma como entrada o número de linhas de código a serem desenvolvidas ou Pontos por Função convertidos em milhares de linha de código-fonte (KSLOC). Descubra a fórmula utilizada neste modelo e descreva cada um dos seus itens.

Comece por: <http://fattocs.com/pt/cursos/nossos-cursos/cocomoii.html>. Acesso em: 24 mar. 2016.

Sem medo de errar

Não nos enganemos: a manutenção é uma daquelas atividades que apresentam grandes desafios em sua execução e que não conferem grande reconhecimento a quem as pratica. Não é incomum que programadores sem muita experiência ou sem grande capacidade técnica sejam destacados para cuidar dos ajustes em um programa. Isso, contudo, está longe de diminuir sua importância.

É por meio da manutenção que o programa se torna mais adequado à sua função, mais confiável, melhora seu desempenho e ganha inovações interessantes. Claro que tudo isso tem seu custo, tanto material, quanto relacionado à mão de obra. Sob essa perspectiva, é natural que uma organização queira fazer estimativas de quanto esforço deverá ser empreendido na atividade e, com elas, dimensionar seu investimento.

Depois de adotar as medidas de qualidade necessárias para o atingimento de nível satisfatório de excelência, a atenção da *XAX-Sooft* volta-se agora justamente para um meio de estimar o esforço que empreenderá em suas manutenções. A métrica a ser adotada é de aplicação bem simples e a apuração de seu resultado servirá para parametrizar as decisões relacionadas à atividade.

A maneira que se propõe para a resolução do desafio é igualmente simples. Imaginemos um programa relativamente complexo da *XAX-Sooft* que tenha sido desenvolvido com o esforço de 100 horas de desenvolvimento no mês. Por causa da complexidade das suas funções, a taxa esperada de linhas que passarão por manutenção é de 6%.

Aplicando esses valores na fórmula $E = ACT * SDT$, teremos que:

$E = 0{,}06 * 100$

Espera-se, então, esforço anual de 6 horas de desenvolvimento por mês, no intervalo de 12 meses.

Atenção

Existem outras técnicas importantes de estimativa de esforço de manutenção. Uma delas leva o nome de CII.

Lembre-se

Conforme abordado na Unidade 3, a métrica de Pontos por Função não se aplica apenas a programas em fase de planejamento. Ela poderá ser aplicada também a processos de manutenção, permitindo a estimativa do esforço necessário para se fazer certa alteração no programa.

Avançando na prática

Pratique mais	
Instrução Desafiamos você a praticar o que aprendeu, transferindo seus conhecimentos para novas situações que você pode encontrar no ambiente de trabalho. Realize as atividades e depois as compare com as de seus colegas.	
Estimando o esforço de manutenção por Pontos por Função	
1. Competência geral	Competência para conhecer e conduzir processos de qualidade de *software*.
2. Objetivos de aprendizagem	Conhecer e saber fazer a manutenção e compreender como isso se aplica em meio à evolução do *software*.
3. Conteúdos relacionados	Necessidade de manutenção e Evolução de *Software*. Classificação das atividades de manutenção. Processo, ferramentas e tipos de atividades para manutenção de *software*.
4. Descrição da SP	Mais uma vez colocamos a Mimimi-Soft, desenvolvedora de programas para consultório médico, à frente da situação a ser tratada. A empresa vem adotando com relativo sucesso a técnica ACT para estimar os esforços de manutenção. No entanto, esse modelo de estimativa tem falhado em sistemas que demandam adição de novas funcionalidades, o que levou a estimativas incorretas e consequente necessidade de deslocamento de outros programadores de suas funções originais para a manutenção. Sua missão é propor a adoção da métrica de Pontos por Função para que a equipe possa ter a correta noção do investimento das atividades de manutenção.

5. Resolução da SP	A métrica de Pontos por Função pode ser eficiente para estimativa de inclusão de novas funcionalidades em sistemas já existentes. Vale lembrarmos que a criação de novas funções também é classificada como manutenção. Aqui, o processo de cálculo passa pelas mesmas etapas da aplicação da métrica em sistemas ainda não construídos. Assim, a estimativa de esforço de manutenção pode ser obtida pela sequência: Identificação da fronteira do aplicativo -> contagem das funções do tipo dados e do tipo transações -> obtenção dos pontos não ajustados -> determinação do fator de ajuste - > obtenção dos pontos ajustados. A fórmula para o cálculo dos pontos por função ficará como segue: $$VAF = 0{,}65 + \left(0{,}01 * \sum_{i=1}^{14} GSCi\right)$$

Lembre-se

A técnica de Pontos por Função pode ser aplicada em processos de manutenção, pois permite estimar o esforço para implantação de nova funcionalidade no sistema.

Faça você mesmo

Há situações em que ajustar partes do código simplesmente não basta para que a manutenção em um programa fique completa. Sistemas antigos, sem a devida documentação, poderão requerer o que chamamos de reengenharia. Pesquise sobre o tema e resuma-o em uma página. Você pode começar sua pesquisa por <http://www.inf.ufpr.br/silvia/ES/reengenharia/reengenharia.pdf>. Acesso em: 11 mar. 2016.

Faça valer a pena

1. Assinale a alternativa que descreve o conceito de manutenção pós-entrega.

a) Processo de revisão pelo qual passam as especificações de requisitos após terem sido aprovadas pela equipe de desenvolvimento.

b) Atividade que inclui melhorias, adequações e correções em um programa feitas após ele ter sido entregue ao testador.

c) Processo de inclusão de funções logo após a entrega do programa ao líder do desenvolvimento ter sido feita.

d) Atividade que inclui melhorias, adequações e correções em um programa feitas após ele ter sido entregue ao usuário final.

e) Processo de troca do local de instalação do programa.

2. Assinale a alternativa que contém apenas situações determinantes para que um gestor tome a decisão de desenvolver um novo sistema ao invés de aplicar manutenção no já existente:

a) Alto custo de desenvolvimento integral do programa, alta disponibilidade de desenvolvedores.

b) Programa atual obsoleto, necessidade de inúmeras alterações e acréscimos de funções.

c) Programa atual construído a partir de plataforma moderna, alta disponibilidade de desenvolvedores.

d) Falta de espaço no disco rígido para a inclusão de novos dados, programa atual construído com base no paradigma de orientação a objeto.

e) Baixa disponibilidade de testadores, falta de espaço no disco rígido para a inclusão de novos dados.

3. Em relação à origem dos problemas que levam à necessidade de manutenção, analise as afirmações a seguir:

I - Apenas travamentos do programa durante a execução justificam a aplicação de manutenção.

II - Não só o código-fonte, mas incorreções nas especificações dos requisitos, no projeto e na documentação podem ensejar manutenção.

III - Aplica-se manutenção, inclusive, quando um requisito não foi corretamente traduzido para uma função executável.

IV - A necessidade de manutenção nasce de uma manifestação do testador.

É verdadeiro o que se afirma apenas em:

a) IV.

b) II e IV.

c) III e IV.

d) II e III.

e) I.

Referências

BARTIÉ, A. **Garantia da qualidade de *software***: as melhores práticas de engenharia de *software* aplicadas à sua empresa. 5. ed. São Paulo: Elsiever, 2002.

DELAMARO, M. E. **Introdução ao teste de *software***. Rio de Janeiro: Elsevier, 2004.

MECENAS, I.; OLIVEIRA, V. **Qualidade em *software***: uma metodologia para homologação de sistemas. [S. l.]: Alta Books, 2005.

MOORTHY, V. **Jumpstart to software quality Assurance**. [S.l : s.n.], 2013.

PINHEIRO, V. Um comparativo na execução de testes manuais e testes de aceitação automatizados em uma aplicação web. Simpósio Brasileiro de Qualidade de *Software* – SBQS 2015. **Anais**... Manaus: Uninorte, 2015.

PRESSMAN, R. S. **Engenharia de software**. São Paulo: Pearson Prentice Hall, 2009.

ROCHA, A. R.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de *software***: teoria e prática. São Paulo: Pearson, 2001.

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

SHAFFER, Steven C. **A brief introduction to software development and quality assurance management**. [S.l : s.n.], 2013.

WAZLAWICK, R. S. **Engenharia de *software***: conceitos e práticas. Rio de Janeiro: Elsiever, 2013.

ISBN 978-85-8482-416-8