DEDMD

LIVRO

UNIDADE 1

Engenharia de Software

Fundamentos de Engenharia de Software

Roque Maitino Neto

2015
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 destas 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, este 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.

Essa 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 essa 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 desse material didático.

Vamos iniciar pela seguinte situação: você é sócio-proprietário de uma startup de desenvolvimento de *software*, chamado "*X-Soft*". 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 este cliente com a qualidade que ele demanda, mudanças nos processos de desenvolvimento da *X-Soft* 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 *X-Soft*.

2. Proposta de melhoria do processo de *software* utilizado pela *X-Soft*.

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 estas 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 *X-Soft* 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 *X-Soft* 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 "*X-Soft*". Utilize os conhecimentos que serão abordados nessa 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 destes 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 destes 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 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 da 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 *X-Soft* 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 fora 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 *X-Soft* 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 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 *X-Soft*.

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 http://www.sabesim.com.br/o-que-e-um-wiki-corporativo/

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	Esta 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 a 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? Sugiro 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/.

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. Introdução ao ciclo de vida do *software*

Diálogo aberto

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

Com o crescimento da *X-Soft*, 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 *X-Soft*, 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 *X-Soft* esta 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. Este 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**.

Este 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), Projeto é um conjunto de atividades temporárias, realizado em grupo, destinado 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 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).

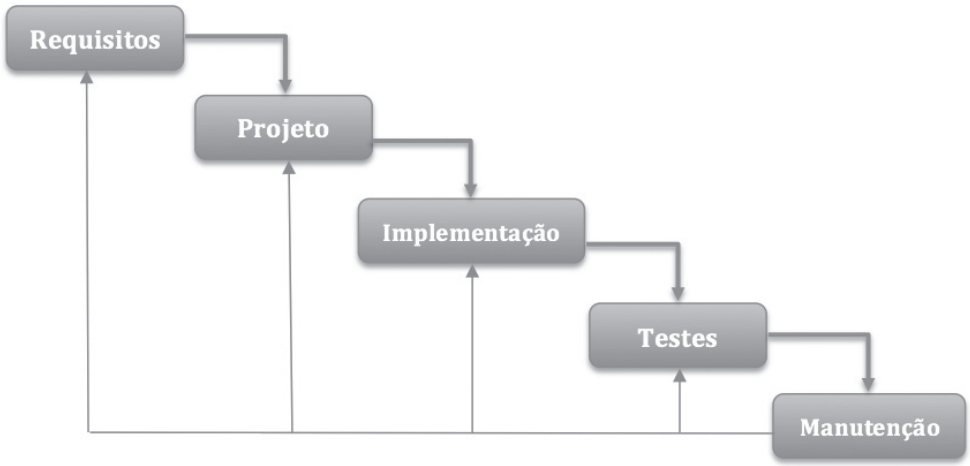
Em suas regras processuais, a organização pode determinar que seja adotado um documento que descreva a atividade. Por meio dele, a equipe tomará conhecimento da tarefa, seus responsáveis, objetivos, recursos a serem utilizados e tudo o que a caracteriza por completo.

Sabemos até o momento que um processo é um conjunto disciplinado e articulado de tarefas, que serve para sistematizar o desenvolvimento de *software*. Este 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: O autor (2013).

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 dessa 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 estes 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 satisfaz 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 *X-Soft* 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**: nessa 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, nessa 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. Terminado a implementação do módulo, os desenvolvedores envolvidos deverão delegar aos colegas a tarefa de testar suas funções, com base nos requisitos revisados. Terminados estes 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.

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: 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 *X-Soft* acaba de adquirir a *You-Soft*, startup que desenvolve exclusivamente um Sistema de Gerenciamento de Conteúdo (do inglês *Content Management System – CMS*). A *X-Soft* 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 *X-Soft* 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 a 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 *X-Soft*, a consequência desta 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 autonomamente 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 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 este 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 à 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

Etapas iniciais do processo de software: requisitos, projeto de sistema

Diálogo aberto

Para iniciarmos a terceira seção do seu material, vale breve retomada de como a situação da *X-Soft* 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, estes 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 neste tema lhe possibilitará, inclusive, comparar esta metodologia com outras mais atuais, além de capacitá-lo ao 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, 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 *X-Soft*.

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 sub etapas 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, ao invés 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 este 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 do participante 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 (Aceso 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 estes 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 a 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 sub classificado 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 *X-Soft*, 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 desta tarefa. Findada esta 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. O documento encontrado em http://www.fazenda.sp.gov.br/sat/RequisitosSATv1.pdf (Acesso em: 12 nov. 2015) pode ser boa fonte de consulta.

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 *X-Soft* 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 *X-Soft* 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 cliente 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 *X-Soft*, 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

As etapas finais do processo de desenvolvimento de *software*

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 *X-Soft*. 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 desta 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 estas 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.

Em http://www.devmedia.com.br/padroes-de-codificacao/16529 (acesso em: 2 nov. 2015), você poderá encontrar parte da padronização para a linguagem *Delphi*.

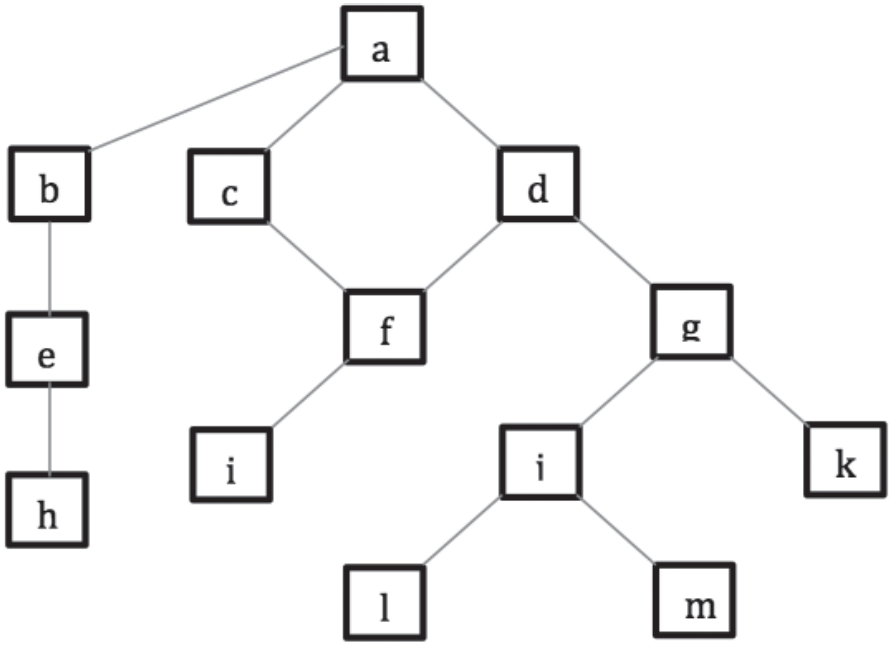
Os nomes das variáveis e demais componentes do programa devem ser coerentes com sua real função. Recomenda-se que sejam dados nomes universalmente significativos aos identificadores, ou seja, que façam sentido tanto para o programador que trabalha naquele projeto como para os que eventualmente lhe darão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

Também estão inclusas nestas 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, esta 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 esta forma de integração é a falta de isolamento de falhas. Ou seja, caso o produto falhe, esta 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. 479).

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 afim 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 a 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 as 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 esta 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 nessa 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 esta 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.

Lembre-se

A entrega contínua de *software* ao cliente tem sido expediente usado para manter o programa sempre atualizado com novas funcionalidades. Saiba mais em http://www.devmedia.com.br/entrega-continua-de-software-revista-net-magazine-100/26312. Acesso em: 2 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.	
"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 *X-Soft* 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 ora 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 *X-Soft*.

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

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

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

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

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

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