engenharia de software magazine

Requisitos
Trabalhando com casos de uso na prática – Parte 2

Requisitos
Veja como detalhar os requisitos de software

DevMedia group Ano I - Edição 11

Manutenção de Software

- Definições, evolução e desafios
- Uma visão geral sobre evolução e reengenharia de software

Planejamento
Estimativas de Software - Fundamentos, Técnicas e Modelos

Validação, Verificação e Testes
Introdução a testes de software

Processo
Entenda a importância de se aplicar controle estatístico em processos de software

Aulas desta edição:

- Introdução à Construção de Diagrama de Classes da UML – Parte 21
- Introdução à Construção de Diagrama de Classes da UML – Parte 22
- Introdução à Construção de Diagrama de Classes da UML – Parte 23

ISSN 1983127-7

engenharia de software magazine

Ano 1 - 11ª Edição 2009 **Impresso no Brasil**

Atendimento ao Leitor

A DevMedia conta com um departamento exclusivo para o atendimento ao leitor. Se você tiver algum problema no recebimento do seu exemplar ou precisar de algum esclarecimento sobre assinaturas, exemplares anteriores, endereço de bancas de jornal, entre outros, entre em contato com:

Carmelita Mulin – Atendimento ao Leitor
www.devmedia.com.br/central/default.asp
(21) 2220-5375

Kaline Dolabella
Gerente de Marketing e Atendimento
kalined@terra.com.br
(21) 2220-5375

Publicidade

Para informações sobre veiculação de anúncio na revista ou no site entre em contato com:

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique a vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site SQL Magazine, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Rodrigo Oliveira Spínola - Colaborador
editor@sqlmagazine.com.br

EDITORIAL

"O desenvolvimento de qualquer software, exceto programas muito simples, é uma tarefa bastante complexa. Esse fato se tornou evidente com a "crise de software", o que originou o conceito de engenharia de software como uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, manutenção e descarte de software."

"A engenharia de software surgiu inicialmente mais como uma promessa do que uma realidade. De fato, muitos dos problemas ligados ao desenvolvimento e manutenção de software continuam sem solução, apesar de muitos conceitos terem evoluído."

"A manutenção de software é hoje um assunto presente em organizações que desenvolvem e mantém software. Isso se deve à necessidade de sempre ajustar e melhorar o produto de software de acordo com as mais diversas necessidades. Diante desse fato, entender o significado e abrangência do termo manutenção de software pode auxiliar organizações e profissionais interessados no tema a melhor conduzir seus esforços quando precisam manter seus produtos."

Neste contexto, a Engenharia de Software Magazine destaca nesta edição duas matérias muito interessantes:

• Introdução à Manutenção de Software: este artigo trata do assunto manutenção de software, englobando suas definições, evolução e desafios. É apresentado também, de maneira breve, a norma ISO/IEC 12207 e sua estrutura para o processo de software

• Reengenharia de Software: este artigo tem como objetivo apresentar uma visão geral sobre softwares legados, evolução de software e reengenharia de software. Para isso, é apresentado o conceito de softwares legados, as categorias da evolução de software, que são: manutenção de software, modernização de software e substituição de software. Além disso, são apresentadas atividades e técnicas de reengenharia de software.

Além destas matérias, esta edição traz mais cinco artigos:

• UML – Casos de Uso: Entendendo os casos de uso na prática – um estudo de caso – Parte II;
• Especificação de Casos de Uso: Detalhando os Requisitos de Software;
• Estimativas de Software - Fundamentos, Técnicas e Modelos;
• Introdução a Testes de Software;
• Controle Estatístico de Processos Aplicado a Processos de Software.

Desejamos uma ótima leitura!
Equipe Editorial Engenharia de Software Magazine

Rodrigo Oliveira Spínola
(rodrigo@sqlmagazine.com.br)

Doutorando em Engenharia de Sistemas e Computação (COPPE/UFRJ). Mestre em Engenharia de Software (COPPE/UFRJ, 2004). Bacharel em Ciências da Computação (UNIFACS, 2001). Colaborador da Kali Software (www.kalisoftware.com), tendo ministrado cursos na área de Qualidade de Produtos e Processos de Software, Requisitos e Desenvolvimento Orientado a Objetos. Consultor para implementação do MPS.BR. Atua como Gerente de Projeto e Analista de Requisitos em projetos de consultoria na COPPE/UFRJ. É Colaborador da Engenharia de Software Magazine.

Marco Antônio Pereira Araújo
(maraujo@devmedia.com.br)

É Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ – Linha de Pesquisa em Engenharia de Software, Especialista em Métodos Estatísticos Computacionais e Bacharel em Matemática com Habilitação em Informática pela UFJF, Professor dos Cursos de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora e da Faculdade Metodista Granbery, Analista de Sistemas da Prefeitura de Juiz de Fora. É editor da Engenharia de Software Magazine.

Eduardo Oliveira Spínola
(eduspinola@gmail.com)

É Editor das revistas Engenharia de Software Magazine, SQL Magazine, WebMobile. É bacharel em Ciências da Computação pela Universidade Salvador (UNIFACS) onde atualmente cursa o mestrado em Sistemas e Computação na linha de Engenharia de Software, sendo membro do GESA (Grupo de Engenharia de Software e Aplicações).

Caro Leitor

Para esta edição, temos um conjunto de 3 vídeo aulas. Estas vídeo aulas estão disponíveis para download no Portal da Engenharia de Software Magazine e certamente trarão uma significativa contribuição para seu aprendizado. A lista de aulas publicadas pode ser vista ao lado:

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 21
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade ao estudo de caso de um sistema de gestão de festas. Neste estudo de caso, partiremos dos requisitos funcionais do software. A partir deles, será elaborado o diagrama de casos de uso do sistema. Com o diagrama de casos de uso elaborado, serão especificados três casos de uso. Por último, será elaborado o diagrama de classes com base nos requisitos funcionais e casos de uso especificados. Nesta aula, o foco é a elaboração da versão inicial do diagrama de classes na Visual Paradigm contendo as classes e atributos identificados até o momento.

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 22
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade ao estudo de caso de um sistema de gestão de festas. Neste estudo de caso, partiremos dos requisitos funcionais do software. A partir deles, será elaborado o diagrama de casos de uso do sistema. Com o diagrama de casos de uso elaborado, serão especificados três casos de uso. Por último, será elaborado o diagrama de classes com base nos requisitos funcionais e casos de uso especificados. Nesta aula, o foco é a identificação das operações que estarão presentes nas classes a partir dos requisitos funcionais do software.

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 23
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade ao estudo de caso de um sistema de gestão de festas. Neste estudo de caso, partiremos dos requisitos funcionais do software. A partir deles, será elaborado o diagrama de casos de uso do sistema. Com o diagrama de casos de uso elaborado, serão especificados três casos de uso. Por último, será elaborado o diagrama de classes com base nos requisitos funcionais e casos de uso especificados. Nesta aula, o foco é a identificação das operações que estarão presentes nas classes a partir dos casos de uso especificados.

ÍNDICE

Você não está mais sozinho.

Introdução à Manutenção de Software

Mateus Maida Paduelli

Bacharel e mestre em computação pela Universidade de São Paulo. Já atuou em diversas organizações com o tema manutenção de software e atualmente é servidor público federal.

O desenvolvimento de qualquer software, exceto programas muito simples, é uma tarefa bastante complexa. Esse fato se tornou evidente com a "crise de software", o que originou o conceito de engenharia de software como uma abordagem sistemática, disciplinada e quantificável para o desenvolvimento, manutenção e descarte de software.

A engenharia de software surgiu inicialmente mais como uma promessa do que uma realidade. De fato, muitos dos problemas ligados ao desenvolvimento e manutenção de software continuam sem solução, apesar de muitos conceitos terem evoluído.

Entender o que significa manutenção de software, e principalmente a abrangência do significado do termo, constitui passo fundamental para o estudo e aprofundamento de soluções para os problemas atrelados a essa tarefa.

Faz-se importante destacar nesse ponto que o autor deste trabalho vem trabalhando diretamente com manutenção de software no meio organizacional há alguns anos. Esta experiência acaba por

Neste artigo veremos

O artigo trata do assunto manutenção de software, englobando suas definições, evolução e desafios. É apresentado também, de maneira breve, a norma ISO/IEC 12207 e sua estrutura para o processo de software.

Qual a finalidade

Este texto é indicado para aqueles profissionais que atuam com manutenção de software no seu dia-a-dia e que desejam entender melhor a evolução histórica e os atuais desafios em torno desse tema.

Quais situações utilizam esses recursos?

A manutenção de software é hoje um assunto presente em organizações que desenvolvem e mantém software. Isso se deve à necessidade de sempre ajustar e melhorar o produto de software de acordo com as mais diversas necessidades. Diante desse fato, entender o significado e abrangência do termo manutenção de software pode auxiliar organizações e profissionais interessados no tema a melhor conduzir seus esforços quando precisam manter seus produtos.

influenciar a percepção dos fatores mais importantes envolvidos na prática com a manutenção de software, contribuindo para a concretização deste artigo.

Atividade de Manutenção de Software

Este tópico tem por objetivo destacar as características, tipos e desafios para a manutenção de software. Trata-se de um tópico importante para esclarecer alguns pontos pertinentes ao assunto que serão considerados ao longo deste artigo.

Definições

A atividade de manutenção de software é caracterizada pela modificação de um produto de software já entregue ao cliente, para a correção de eventuais erros, melhora em seu desempenho, ou qualquer outro atributo, ou ainda para adaptação desse produto a um ambiente modificado (IEEE, 1998).

Embora a definição trate genericamente qualquer produto de software, existem diferenças entre a manutenção de softwares com propósitos distintos. Essa distinção é explicada por Pfleeger (2001), que estabelece três categorias de sistemas.

Uma primeira classificação representa aqueles softwares construídos com base em uma especificação rígida e bem definida, cujos resultados esperados são bem conhecidos. Por exemplo, um software construído para realizar operações com matrizes (adição, multiplicação e inversão). Nesse tipo de software, uma vez que tenha sido construído considerando a correta implementação do método, dificilmente haverá a necessidade de manutenções.

Já em uma segunda classificação, são agrupados os softwares que constituem implementações de soluções aproximadas para problemas do mundo real, uma vez que soluções completas somente são conseguidas na teoria nesses casos. Como exemplo, pode-se citar um jogo de xadrez. Embora suas regras sejam bem definidas, não é possível construir um software que calcule a cada passo todos os possíveis movimentos de peças do tabuleiro, de forma a determinar o melhor movimento. Isso porque o número de movimentos possíveis é muito grande para ser calculado em um intervalo de tempo relativamente curto. A técnica utilizada para desenvolver esse tipo de solução baseia-se em descrever o problema de forma abstrata e então definir os requisitos de software a partir dessa abstração.

Percebe-se que esse tipo de sistema já abre espaço para diferentes interpretações por parte do desenvolvedor, o que tende a produzir software com maior necessidade de manutenção do que quando comparado aos da classificação anterior. Por considerar uma abstração para especificação de requisitos, a necessidade de mudança pode aparecer caso a abstração mude, na medida em que um maior entendimento do problema seja alcançado.

Finalmente, uma terceira e última classe de softwares considera mudanças no ambiente onde o software vai ser utilizado, característica não existente nas duas classificações anteriores.

Um software integrante da terceira classificação corresponde àquele criado com base em um modelo dos processos abstratos envolvidos no sistema, e precisará mudar sempre que ocorrer mudanças nesses modelos, sendo, portanto, parte do ambiente que ele modela. Um exemplo desse tipo de software seria aquele que apresenta informações da economia de um país. À medida que a economia passa a ser mais bem compreendida, o modelo muda e com ele a abstração do problema, causando uma necessidade inevitável de manutenção no software. Esse tipo de software, comumente encontrado no dia-a-dia das organizações, tem interesse particular neste trabalho.

Além de considerar tipos diferentes de software, o processo de manutenção não corresponde a uma atividade isolada. Durante a sua execução, diferentes partes precisam interagir de forma que os objetivos da manutenção sejam entendidos e os resultados esperados alcançados. Essas partes são apresentadas em Polo et al. (1999), que as define como:

Organização Cliente: essa organização corresponde ao adquirente do software, conforme definido pela norma ISO/IEC 12207. Isso significa a organização que possui o software e requisita o serviço de manutenção.

Mantenedor: trata-se da organização que oferece o serviço de manutenção.

Usuário: representa a organização ou pessoa que utiliza o software em questão, valendo-se de suas funcionalidades para automatizar e facilitar tarefas.

Ainda segundo esse autor, existe um relacionamento entre essas partes e a constituição de um fluxo para os pedidos de manutenção, conforme é esquematizado na **Figura 1**.

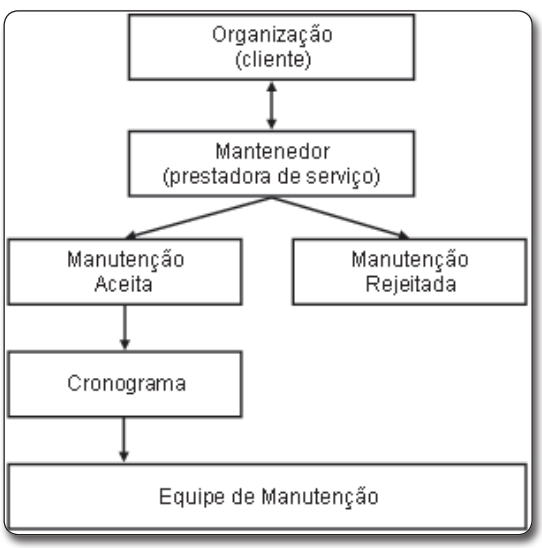


Figura 1. Fluxo envolvido na manutenção de software (adaptação Polo et al. 1999)

A organização solicitante deve possuir algum responsável pela solicitação de manutenção, que será encarregado de verificar os requisitos e informar o mantenedor. Esse responsável deverá considerar os erros e dificuldades apontadas pelo help-desk, que corresponde ao departamento diretamente encarregado do diálogo com os usuários.

Do lado do mantenedor, a organização que presta o serviço deve possuir alguém responsável por avaliar as requisições,

julgando-as apropriadas ou não para os objetivos do software e da organização solicitante. Uma vez que os pedidos de manutenção são aceitos, deve existir alguém responsável por estabelecer um cronograma de entrega, que deverá considerar as prioridades e interesses de ambas as partes. Esse cronograma deverá ser seguido pela equipe de manutenção, que compreende o pessoal envolvido diretamente em atender às solicitações.

Finalmente, o papel do usuário consiste em utilizar o software reportando problemas para o help-desk, que por sua vez informará o responsável pelas solicitações de manutenção, fechando o ciclo.

Uma observação faz-se necessária com relação ao termo falha e defeito de software. Conforme explica Pressman (2005), a IEEE define para o contexto de hardware a explicação de que um defeito constitui uma anomalia do produto. Já uma falha pode significar um defeito em um dispositivo ou componente, ou uma definição incorreta de passo, processo ou de dados em um programa de computador.

No contexto de manutenção de software, os termos falha e defeito são sinônimos, e ambos se referem a algum problema de qualidade, identificado após o software ter sido entregue ao usuário.

Evolução de Software

As mudanças que ocorrerão em um software para deixá-lo mais completo, livre de erros, ou adaptado ao seu ambiente, podem ser definidas como atividades de evolução de software, conforme explica Sommerville (2003).

A decisão de investir na evolução de um software, ou abandoná-lo para iniciar um projeto novo, construído com base nos requisitos atuais, não é uma tarefa trivial (Pfleeger, 2001). Essa decisão deve considerar fatores como o custo envolvido, a confiabilidade do software após a manutenção, a capacidade que possuirá de se adaptar a mudanças futuras, seu desempenho, limitações de suas atuais funções, e ainda as opções existentes no mercado que atendam o mesmo problema de maneira mais completa, rápida e barata.

Se o software atual funciona adequadamente, supõe-se que a organização que o utiliza terá uma preferência em aplicar a ele apenas os ajustes necessários em função de mudanças nos negócios ou outras necessidades de adaptações, do que substituí-lo por completo. Abandonar sistemas existentes para iniciar projetos a partir do zero representa uma perda considerável (Silva & Santander, 2004).

Visando entender melhor as características de evolução de software, Lehman (1996), publicou suas oito "leis da evolução de software". De acordo com esse trabalho, seriam oito as principais características que regem o envelhecimento de um software, e foram determinadas após mais de vinte anos de estudos e observações. Inicialmente foram definidas apenas três leis (Lehman, 1974), que passaram a ser cinco (Lehman, 1980), alcançando seis leis (Lehman, 1991). Finalmente, após estudos mais longos com processos de software, as atuais oito leis foram publicadas. Essas leis consideram não apenas o software em si, mas as características da organização que o desenvolve e mantém.

A primeira lei, caracterizada pela evolução contínua, explica que um software desenvolvido para tratar um problema do mundo real, precisa ser constantemente adaptado, pois caso contrário, ele se tornará progressivamente menos satisfatório. Essa lei baseia-se na experiência, sendo que a evolução somente pode acontecer por meio de um processo de manutenção controlado e dirigido.

Na segunda lei, o autor explica que existe um aumento progressivo de complexidade do software à medida que este se desenvolve, o que pode ocasionar uma desestruturação crescente. Esse problema será inevitável, a menos que algum trabalho seja feito para reduzi-lo, e é ocasionado por alterações efetuadas em cima de outras alterações, na medida em que o software é adaptado ao seu ambiente.

A terceira lei prega uma auto-regulação do processo de evolução do software. Isso significa que a organização responsável pelo desenvolvimento desse software estabelecerá normas e verificações capazes de assegurar o entendimento das dificuldades e prover respostas que serão utilizadas para o crescimento e estabilização tanto do processo, como do produto de software. O gerenciamento de retornos positivos e negativos dos pontos de controle é um exemplo de mecanismos de estabilização. Existem vários outros, e juntos eles estabelecem uma dinâmica disciplinada cujos parâmetros são, pelo menos em parte, normalmente distribuídos.

A quarta lei trata da conservação da estabilidade organizacional, no sentido de que ao longo de todo ciclo de vida do software, a forma como a organização irá trabalhá-lo sempre produzirá resultados constantes. A lei sugere que mudanças de pessoal e recursos têm pouco efeito sobre a evolução do sistema.

Na quinta lei, é apresentada a característica de conservação de familiaridade, o que significa dizer que o conhecimento dos objetivos da organização ao lançar uma nova atualização do software, é conhecido entre os membros da equipe que trabalha com esse software. Quanto mais mudanças forem necessárias ao software, maior será a dificuldade de manter toda a equipe ciente dos objetivos organizacionais. A taxa, qualidade de progresso e outros parâmetros são influenciados, até mesmo limitados, pela taxa de aquisição da informação necessária pelos participantes coletivamente e individualmente.

A sexta lei consagra a característica do contínuo crescimento do software, decorrente de novas funcionalidades, a fim de manter a satisfação do cliente. Essas mudanças podem ser correções de erros, adições de novas funcionalidades ou melhorias em funções pré-existentes. A maioria certamente não foi planejada pela equipe de desenvolvimento na época da primeira versão, ou foram causadas devido a alguma mudança no domínio operacional em que o software está inserido. Essas mudanças não planejadas inicialmente geram a necessidade de módulos extras para o software, causando assim um inevitável aumento do conteúdo funcional.

Na sétima lei, é apresentada a característica de qualidade decrescente do software, à medida que evolui. Isso ocorre uma vez que o software está inserido em um ambiente imprevisível, no qual existem possibilidades ilimitadas que entrarão em

contraste com as características de evolução do software, limitadas pelo número de recursos e tempo disponível, expondo a suscetibilidade do software a eventos externos. Ainda que esse software venha a funcionar satisfatoriamente por muitos anos, isto não será um indicativo de que continuará funcionando a contento nos anos vindouros. Essa característica somente poderá ser evitada com um esforço para detecção e correção contínuas.

Finalmente, na oitava lei, publicada somente na década de 1990, o autor apresenta a característica de sistema de retorno, o que significa que o sistema de evolução de um software será constantemente realimentado pelo retorno de seus usuários. A vida de um software é um ciclo bem estabelecido de retornos positivos e negativos dos usuários entre cada versão do software. Em longo prazo, a taxa de crescimento do sistema será estabelecida pela quantidade de retornos negativos e positivos, e controlado por fatores como quantidade de recursos (verba), número de usuários solicitando novas funcionalidades ou reportando algum erro, interesses administrativos e tempo entre uma versão e outra.

De uma maneira geral, as leis de Lehman abordam características aplicáveis a grandes corporações, que desenvolvem softwares complexos e com longo ciclo de vida. Não ficam claras quais as dificuldades para a manutenção de software. O que se tem é um panorama geral da evolução de software, apresentando em diversas leis a necessidade de intervenção no software para que problemas sejam evitados, mas nada diz a respeito das dificuldades que emergem dessa intervenção.

Natureza e Características

Atividades de manutenção de software são caracterizadas por intervenções no produto de software de forma a evitar sua deterioração. Um software não se desgasta como peças de um equipamento, mas se deteriora no sentido de os objetivos de suas funcionalidades cada vez menos se adequarem ao ambiente externo.

Do ponto de vista de desenvolvimento, Pfleeger (2001) explica que o foco das atenções está em produzir código que atenda aos requisitos e funcione corretamente. Isso inclui dizer que a cada estágio do desenvolvimento, haverá uma referência contínua a componentes produzidos em estágios anteriores. O desenvolvimento levará a uma integração dos componentes desenvolvidos, passando por etapas de revisão e testes para certificação de sua corretude, adequação aos requisitos a ao projeto. Resumindo, o desenvolvimento terá foco em considerar estágios anteriores do processo de maneira controlada.

A manutenção de software, por sua vez, incluirá não apenas considerar resultados de etapas anteriores, mas atividades do presente de forma a conseguir compreender o nível de satisfação dos usuários em relação ao software. Uma característica importante é também considerar o futuro durante a manutenção, buscando antever necessidades de mudanças nos negócios, o que causaria mudanças nesse software.

Outra característica fundamental está relacionada à imprevisibilidade da manutenção de software. Esse fato está relacionado à influência que o software sofre de fatores externos, naturalmente imprevisíveis (Bhatt et al., 2004).

Na **Figura 2** é mostrada a curva de esforço despendido com a atividade de manutenção de software, destacando os picos de esforço ocasionados por fatores externos.

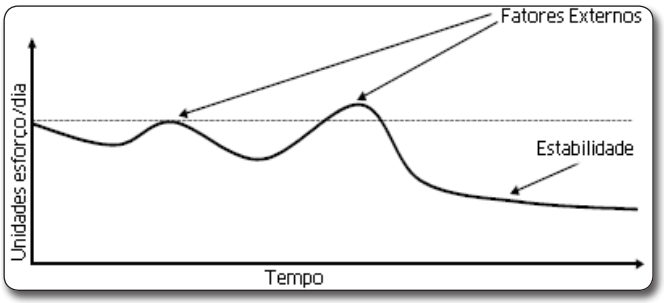


Figura 2. Curva de esforço para manutenção de software (adaptação de Bhatt et al. 2004)

Pela figura percebe-se que o software é sensível ao contexto no qual está inserido, estando sujeito a mudanças na medida em que seu ambiente muda, causando a necessidade de considerar e tentar prever esses fenômenos, tanto durante o desenvolvimento como durante a manutenção.

Um modelo de ciclo de vida de manutenção de software é proposto por Kung e Hsu (1998), no qual se destacam quatro fases de vida para uma aplicação de software: (i) introdução; (ii) crescimento (iii) maturidade; (iv) declínio. Para cada fase, os autores indicam qual o tipo de tarefa que será mais incidente, conforme mostrado na **Figura 3**.

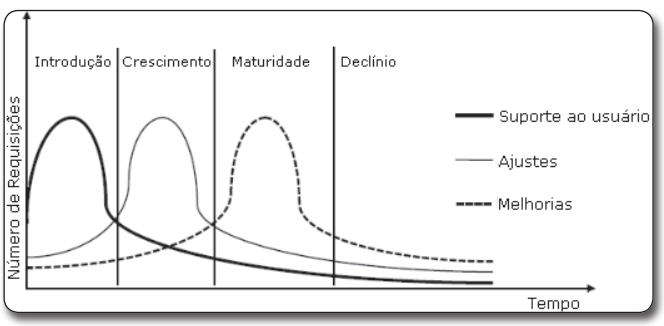


Figura 3. Ciclo de vida de manutenção de software (adaptação de Kung & Hsu, 1998)

Quanto tempo o software vai levar para entrar na fase de declínio é uma previsão quase impossível, visto que, uma vez na fase de maturidade, o software adquiriu certa estabilidade na organização, possivelmente já se enquadrando na classificação de sistema legado. Nesse sentido, a fase de declínio corresponderia àquela na qual uma decisão a respeito de manter ou substituir o software deve ser tomada, conforme exposto com mais detalhes mais adiante neste artigo.

Tipos de Manutenção

As ações ligadas à atividade de manutenção de software foram classificadas de acordo com sua natureza em três categorias

(Lientz & Swanson, 1980): corretivas, adaptativas e perfectivas.

• Manutenções do tipo corretivas visam corrigir defeitos de funcionalidade, o que inclui acertos emergenciais de programa. Pfleeger (2001) expõe um exemplo desse tipo de manutenção, que consiste em um usuário apresentando um problema de impressão em um relatório. O número de linhas impresso por folha é muito grande, o que causa sobreposição de informações. O problema foi identificado como uma falha no driver da impressora, provocando a necessidade de se alterar o menu do relatório para aceitar um parâmetro adicional que determina o número máximo de linhas impressas por folha.

• Manutenções do tipo adaptativas referem-se a adequar o software ao seu ambiente externo. O exemplo apontado por Pfleeger (2001) ilustra bem essa categoria. Suponha um gerenciador de banco de dados, que faz parte um sistema maior de hardware e software. Em uma atualização do gerenciador, os programadores perceberam que as já existentes rotinas de acesso a disco precisavam agora de mais um parâmetro adicional. Essa manutenção corresponde a uma manutenção adaptativa, uma vez que teve por finalidade adequação do software ao seu ambiente e não a correção de um defeito.

• Manutenções do tipo perfectivas têm por objetivo acrescentar novos recursos de funcionalidade ao software, normalmente em razão de solicitações dos usuários. Significam ainda re-projetar partes de um software, de forma a tornar mais simples a compreensão e utilização do mesmo. Como exemplo, pode-se citar o pedido do usuário por um novo relatório com informações que até então não podiam ser obtidas do banco de dados.

A união das categorias adaptativa e perfectiva é sugerida por Pigoski (1996), que propõe uni-las em uma única denominada aprimoramentos. Essa classificação estaria de acordo com organizações que geralmente utilizam o termo manutenção para se referir à execução de pequenas mudanças no software, enquanto o termo desenvolvimento é usado para os demais tipos de modificações.

Uma quarta categoria de manutenção é apresentada por alguns autores, como Pressman (2005). Essa categoria se refere a manutenções do tipo preventivas que buscam identificar previamente possíveis fontes de problemas no software e corrigi-las antecipadamente.

A IEEE (1998) modifica a definição apresentada inicialmente por Lientz e Swanson (1980), definindo uma categoria a mais chamada emergencial. Essa categoria é caracterizada pela execução de uma manutenção corretiva não planejada, com o intuito de manter o software operacional. Tal classificação insere a idéia de manutenção planejada e não-planejada, bem como manutenção reativa e pró-ativa, como é ilustrado na **Tabela 1**.

	Planejada	Não-Planejada
Reativa	Corretiva Adaptativa	Emergencial
Pró-ativa	Perfectiva	

Tabela 1. Definições IEEE para as categorias de manutenção de software

Para efeito de estudo, neste artigo será considerada a classificação de tipos de manutenção definida por Pressman (2005): corretivas, adaptativas, perfectivas e preventivas.

Dentro dessa classificação adotada, um estudo com empresas de software (Souza et al., 2004), constatou que entre as organizações pesquisadas, as manutenções do tipo corretivas prevaleceram com 50% dos resultados, seguido pelas perfectivas (30%) e adaptativas (20%). Na pesquisa de Souza não foi identificada nenhuma organização realizando manutenção do tipo preventiva.

Custos e Desafios

A manutenção de software é uma operação importante, pois consome a maior parte dos custos envolvidos no ciclo de vida de um software (SWEBOK, 2004), e a falta de habilidade em mudar um software rapidamente, e de maneira confiável, pode causar a perda de oportunidades de negócio (Bennett & Rajlich, 2000).

Embora não exista um consenso sobre o valor exato do custo atrelado à atividade de manutenção, as pesquisas na área apontam, na totalidade dos casos, sempre mais de 50% dos investimentos realizados no software. De fato, para Bhatt et al. (2004), esse percentual corresponde a algo entre 67% a 75% do investimento total, enquanto para Polo et al. (1999) corresponde a um valor entre 67% e 90%. Ainda de acordo com esse último autor, a razão do custo elevado deve-se, em parte, à própria natureza da atividade de manutenção, caracterizada principalmente pela imprevisibilidade. Além dos altos custos financeiros, essa é também a atividade que exige maior esforço dentre as atividades de engenharia de software, conforme apontado por Sneed (2003). Pressman (2005) ainda completa que o grande esforço necessário na manutenção se justifica pela abrangência do significado desse termo no contexto de software.

A importância financeira atrelada à manutenção de software é ainda agravada quando se leva em consideração o risco para as oportunidades de negócio, que podem ser causadas pela falta de gerenciamento e compreensão total da dinâmica da atividade. De acordo com a pesquisa realizada por Dart et al. (2001), esse gerenciamento deve considerar três fatores: (i) ferramentas, (ii) pessoas, (iii) processos, revelando-se, pois, uma atividade gerencial complexa.

Se por um lado a atividade de manutenção é dispendiosa, por outro ela é um desafio para as organizações que precisam considerá-la em seu dia-a-dia. Não é de se esperar que uma empresa de grande porte troque todos seus sistemas somente pelo fato de que a tecnologia neles empregada está ultrapassada. Esses sistemas representam ativos importantes da organização e ela estará disposta a investir de maneira a manter seus valores (Sommerville, 2003).

Prever quando uma manutenção precisará ser conduzida é uma tarefa geralmente muito difícil de ser realizada. Essa dificuldade de previsão, e também controle sobre a manutenção, é explicada pelo fato de muitas vezes ficar a cargo de empresas terceirizadas a manutenção, revelando-se um problema já que normalmente essas organizações não possuem nenhum

contato com o projeto inicial do software (Bhatt et al., 2004). Ainda segundo esse autor, o aumento na complexidade dos softwares produzidos (tanto em termos de funcionalidades, como de técnicas) torna a previsão de esforços de manutenção muito vaga. Essa dificuldade em estimar esforços torna-se mais evidente quando se trata de sistemas legados.

Norma ISO/IEC 12207

Em 1987 a ISO e a IEC estabeleceram um Comitê Técnico Conjunto – JTC (Joint Technical Committee) sobre tecnologia da informação, com o objetivo de efetuar a normatização no campo de sistemas de tecnologia da informação (incluindo microprocessadores) e equipamentos (Singh, 1996).

Dentre os objetivos do comitê estava o estabelecimento de um padrão para processos de ciclo de vida de software, que culminou com a norma ISO/IEC 12207, que teve início em 1989. Esta norma deveria ser estabelecida como um framework adaptável, que pudesse ser usado para auxiliar na gerência e na construção do software.

Assim, a ISO/IEC 12207 foi publicada em 1 de agosto de 1995, todavia está em constante evolução, como pode ser comprovado pelas duas emendas que já recebeu (amendment 1, publicada em 2002, e a amendment 2, publicada em 2004).

Essa norma provê um conjunto de processos de engenharia de software que uma organização deve utilizar para adquirir, fornecer, desenvolver ou manter software, ou seja, ela documenta os processos do ciclo de vida de software em um modelo de referência de processos. Sua organização se dá de acordo com o exposto na **Figura 4**.

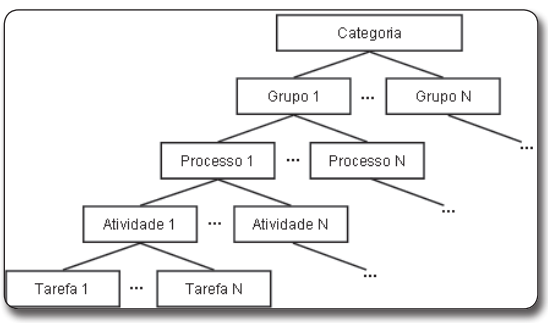


Figura 4. Estruturação da norma ISO/IEC 12207

Ao todo são três categorias, totalizando dez grupos de processos com 47 processos no geral. Na **Figura 5**, extraída da norma ISO/IEC 15504, é mostrada a organização da norma.

Para um melhor entendimento, é relevante descrever, ainda que em linhas gerais em um primeiro momento, quais os propósitos de cada um dos grupos de processos da norma.

A – Categoria de Processos Fundamentais

Grupo de Processos de Aquisição: inclui processos a serem seguidos pelo cliente com a finalidade de aquisição de um produto ou serviço.

Grupo de Processos de Fornecimento: inclui processos

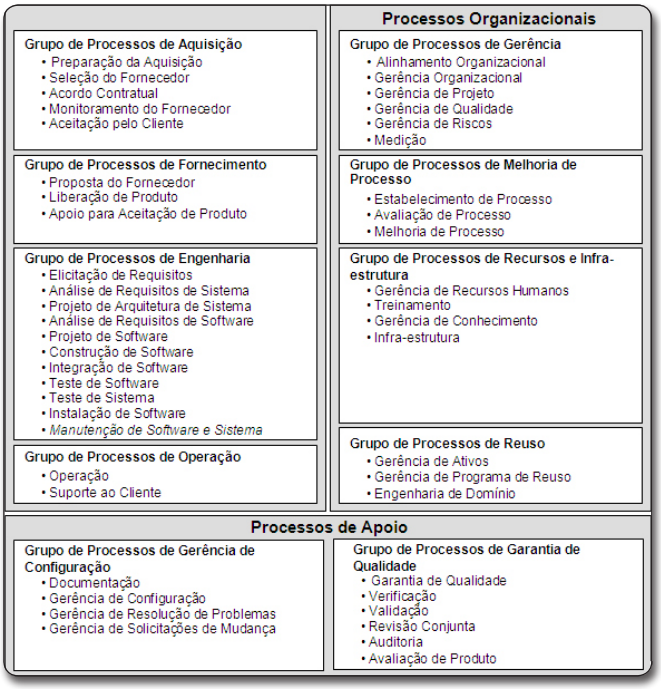


Figura 5. Processos do ciclo de vida da norma ISO/IEC 12207

a serem observados pelo fornecedor, com a finalidade de oferecimento de um produto ou serviço.

Grupo de Processos de Engenharia: inclui processos para elicitação e gerenciamento dos requisitos do cliente, bem como especificação, implementação e/ou manutenção do produto de software, considerando ainda sua relação com o sistema maior do qual fizer parte.

Grupo de Processos de Operação: inclui processos com a finalidade de oferecer suporte à transição do produto para o ambiente do cliente, provendo ainda a correta operação e uso desse produto ou serviço.

B – Categoria de Processos Organizacionais

Grupo de Processos de Gerência: inclui processos que englobam práticas que devem ser observadas por todos aqueles que gerenciam qualquer tipo de projeto ou processo relacionado ao ciclo de vida de software.

Grupo de Processos de Melhoria de Processo: inclui processos destinados à definição, criação e melhoria de processos realizados na organização.

Grupo de Processos de Recursos e Infra-estrutura: inclui processos com a finalidade de prover recursos humanos e infra-estrutura adequada para todos os processos da organização.

Grupo de Processos de Reuso: inclui processos com a finalidade de sistematicamente explorar oportunidades de reuso dentro da organização.

C – Categoria de Processos de Apoio

Grupo de Processos de Gerência de Configuração: inclui processos com a finalidade de controle e manutenção da integridade do produto desenvolvido pelos processos de engenharia.

Grupo de Processos de Garantia de Qualidade: inclui

processos com o intuito de prover garantias de que os produtos de trabalho e processos estejam de acordo com as condições predefinidas e o planejado.

A manutenção de software aparece na norma como um dos processos dentro da categoria de processos fundamentais. Na **Figura 6** é apresentado quais são as atividades pertencentes a esse processo.

Figura 6. Atividades do processo de manutenção de software e sistema (ISO/IEC 12207)

Implantação do processo: essa atividade inclui tarefas para o desenvolvimento de planos e procedimentos para manutenção de software, criando procedimentos para receber, gravar e monitorar pedidos de manutenção, e estabelecer uma interface organizacional com o processo de gerenciamento de configuração. A implementação do processo deve começar cedo no ciclo de vida do software, conforme reforça Pigoski (1996) ao dizer que os planos de manutenção devem ser preparados em paralelo com os planos de desenvolvimento. Essa atividade inclui definir o escopo de manutenção e a identificação e análise de alternativas, bem como organizar e contratar a equipe de manutenção, relacionando recursos e responsabilidades.

Análise do problema e da modificação: em um primeiro momento, essa atividade tem por objetivo analisar a requisição de manutenção para classificá-la, podendo então determinar o escopo em termos de tamanho, custos e tempo necessário, destacando ainda sua prioridade. As demais tarefas dessa atividade focam o desenvolvimento e a documentação de alternativas para mudança de implementação e aprovação das opções adotadas.

• **Implantação da modificação:** essa atividade engloba a identificação dos itens que precisam ser modificados e os processos de desenvolvimento que precisarão ser implementados. Outros requisitos da modificação incluem teste e validação de que as modificações estão corretamente implementadas e que os itens não modificados não foram afetados.

• **Revisão / aceitação da modificação:** as tarefas dessa atividade são dedicadas à confirmação da integridade do software modificado e à conclusão dos negócios com o cliente, quando este concorda e aprova satisfatoriamente a conclusão da requisição de manutenção. Muitos processos de apoio podem ser utilizados aqui, incluindo a garantia da qualidade de processo, o processo de verificação, o processo de validação e o processo de revisão conjunta.

• **Migração:** corresponde à atividade que ocorrerá quando o software for transferido de um ambiente de operação para outro. Será preciso o desenvolvimento de planos de migração e os usuários precisarão estar cientes dos requisitos, dos motivos do antigo ambiente não ser mais suportado e terem à disposição uma descrição do novo ambiente e sua data de disponibilidade. Outras tarefas dessa atividade concentram-se em operações paralelas do novo e antigo ambiente, incluindo a revisão de pós-operação para certificar-se do impacto da mudança de ambiente.

• **Descontinuação do software:** essa atividade consiste em descontinuar o software por meio da formalização, junto ao cliente, de um plano de descontinuação.

Um último comentário referente à norma refere-se ao fato de que ela não representa um modelo fixo ao qual uma organização que a adote se submete. Ao contrário disso, ela funciona como uma estrutura de apoio, devendo a organização que a adotar proceder com adaptações nas recomendações para a sua realidade.

Sistemas Legados

Um sistema legado normalmente representa um dos bens com maior valor econômico de uma organização (Visaggio, 2001).

Essa é a idéia central que envolve as decisões em torno de um software nessas condições, gerando muitas variáveis a considerar, entre elas: (i) é arriscado substituir um software que esteja estável; (ii) os usuários já estão acostumados com a forma de trabalhar com o software e uma mudança normalmente não seria bem vinda; (iii) a mudança vai exigir gastos com treinamento de todos os envolvidos com o uso desse software; (iv) a compra ou construção de um novo software pode extrapolar previsões de custos e prazos; (v) o novo software pode possuir menos funcionalidades, dificultando tarefas dos usuários etc.

Se por um lado o software não pode ficar como está, já que ou possui erros, ou não está mais atendendo às novas necessidades de negócios, por outro, substituí-lo pode trazer problemas ainda maiores.

Neste tópico serão consideradas as características desses softwares e algumas das soluções, e paliativos, já propostos, uma vez que se relacionam diretamente com a atividade de manutenção de software.

Conceitos

Um software desenvolvido no passado e ainda em utilização em uma determinada organização constitui um sistema legado.

Em função da dependência cada vez maior em relação a sistemas de software, as organizações em geral, especialmente as de grande porte, investiram muito dinheiro no desenvolvimento de softwares para atender às suas necessidades operacionais e de gerência. Considerando que a evolução da tecnologia, tanto de hardware como de software, foi muito rápida nos últimos anos, provocando o abandono de linguagens de programação mais antigas e a adoção de novas metodologias de desenvolvimento de software, é fortemente esperado que aqueles softwares, frutos de grandes investimentos das organizações, tenham sido construídos com alguma tecnologia ou método que hoje

é obsoleto. Esses softwares, chamados de sistemas legados, constituem mais um desafio para a engenharia de software, especialmente para a atividade de manutenção de software, uma vez que dificilmente serão abandonados de imediato, justamente por causa do investimento que representam.

A vida útil de um software é muito variada. Alguns softwares de grande porte podem continuar em uso durante mais de dez anos, e em alguns casos organizações podem ainda contar com software desenvolvido há mais de vinte anos (Sommerville, 2003). Uma tentativa de quantificar a quantidade de software legado existente foi feita em 1990, e ainda naquela data, estimou-se que haveria no mundo cerca de 120 bilhões de linhas de código pertencentes a esse tipo de software (Ulrich, 1990).

Problemática dos Sistemas Legados

Várias abordagens foram propostas para gerenciar software legado. Soluções típicas são (Bennett et al., 1999): (i) descartar o software e substituí-lo totalmente (muito inviável na grande maioria dos casos, por restrições financeiras e técnicas); (ii) tornar o software parte integrante de um novo software de maiores proporções; (iii) adiar a manutenção por mais algum tempo (nem sempre isso é possível), e finalmente, (iv) modificar o software para aumentar sua vida útil.

Quando a decisão de modificar o software é tomada, inicia-se um processo de entender esse software, bem como suas estruturas, uma vez que dificilmente o projeto inicial estará disponível, e se estiver, seguramente estará desatualizado.

As razões para explicar a dificuldade de manutenção em sistemas legados apresentada por Sommerville (2003) estão descritas a seguir:

• Diferentes partes do software foram desenvolvidas por equipes diferentes, o que implica em uma não uniformidade no estilo de programação ao longo dele todo;

• Partes do software, ou ele todo, podem ter sido desenvolvidas com o uso de alguma linguagem de programação obsoleta. Isso torna difícil encontrar profissionais com conhecimento e experiência na linguagem, o que pode forçar a terceirização da manutenção (normalmente a um custo elevado);

• A documentação existente é normalmente inadequada e desatualizada. Em muitos casos, a única documentação existente é o próprio código-fonte. Em outras situações mais graves, nem mesmo o código-fonte está disponível, somente a versão executável existe;

• Provavelmente os muitos anos de manutenções alteraram a estrutura do software, tornando-o progressivamente mais difícil de compreender. Novos programas podem ter sido adicionados e sua interface de comunicação com outras partes construída de maneira ad-hoc;

• O software pode ter sido otimizado para reduzir a ocupação de espaço ou melhorar sua velocidade de execução. Isso pode causar problemas de compreensão para profissionais que aprenderam somente as técnicas modernas de engenharia de software, não tendo tido contato com os truques usados no passado;

• Os dados processados pelo software podem estar armazenados em arquivos separados, com estruturas distintas e incompatíveis. Podem ainda existir duplicações de dados, erros e dados incompletos.

Vale aqui uma observação curiosa, referente ao crescimento do número de profissionais da área de software interessados em aprender linguagens e técnicas de programação antigas, justamente para trabalharem no aparente promissor campo da migração de sistemas legados. Organizações de porte médio e grande têm se dedicado à preparação de equipes para conduzir longos e complexos processos de migração de grandes sistemas legados, notadamente aqueles construídos com a linguagem de programação COBOL. Esse é provavelmente o fato que tem feito com que os livros de linguagens antigas estejam voltando a circular, principalmente na indústria, entre profissionais da área.

Gerenciamento de Manutenção de Software

Com o aumento da necessidade de manutenção de software, principalmente de softwares mal desenvolvidos, as organizações têm buscado maneiras de lidar com o problema, já que não é possível trocar o software por um "modelo mais novo", como seria comum pensar em face de um equipamento com defeitos ou com funcionalidades limitadas.

Essa busca por alternativas viáveis economicamente e eficazes tecnicamente, acabam por produzir diferentes abordagens de resposta à necessidade de manutenção, respostas essas que não são sempre corretas e de aplicabilidade universal.

Está a cargo dos tomadores de decisão em relação a software decidir o que fazer e como proceder à necessidade de manutenção de software, o que envolverá sempre questões financeiras e a credibilidade da organização frente a seus clientes.

Lucia et al. (2004) organizaram em quatro grupos as questões importantes a considerar por esses tomadores de decisão:

• **Programadores:** envolve as questões ligadas à equipe de engenheiros de software. Aqui estão incluídos assuntos importantes como a atitude dos profissionais diante de tarefas de manutenção e não de desenvolvimento, ressaltando a necessidade de avaliar questões não apenas técnicas, mas também de compreensão dos negócios do cliente. Nesse grupo estão incluídas ainda eventuais alegações de falta de glamour da atividade de manutenção, inclusive questões como a impossibilidade de desenvolvimento de carreira em manutenção de software, o que acaba por gerar grande rotatividade. Deve-se ainda considerar a habilidade que a equipe de engenheiros de software possui com a atividade de manutenção, avaliando-se a curva de aprendizado para a atividade.

• **Atitude Gerencial:** observa-se que a própria postura gerencial é a principal responsável por ditar os rumos e o sucesso ou fracasso do ambiente de manutenção. Esse grupo envolve as questões relacionadas ao estudo de programas de manutenção, que normalmente são escassas, e seu relacionamento com fatores como tempo e dinheiro. É justamente esse tipo de estudo que permite o sucesso em tomadas de decisão relacionadas a software. Um fator apontando nesse grupo refere-se à atitude gerencial de atribuir o desenvolvimento de projetos importantes a programadores como uma forma de

presenteá-los por bom desempenho profissional. Essa atitude é agravada ainda mais quando é acompanhada de falta de treinamento e oportunidade para equipes de manutenção.

• **Código-Fonte:** definir critérios de qualidade para o código-fonte é outro grupo que envolve questões pertinentes. Arquitetura pobre de software e de estruturas de programa, documentação irregular, falta de padrões e guidelines para atividades de manutenção, incluindo ainda ausência de estudos de impactos de mudanças em outros trechos e módulos do software, representam considerações importantes a fazer.

• **Usuários:** os usuários devem ter participação ao longo de toda atividade de manutenção, para evitar que problemas de manutenibilidade no software possam gerar falta de credibilidade no trabalho do programador. Questões envolvendo confirmação e validação do usuário a respeito do software enquanto em processo de manutenção devem ser consideradas, para evitar erros de interpretação e de especificações. É fundamental reportar aos usuários os erros e dúvidas de interpretação.

Em razão dos muitos critérios a considerar, as decisões gerenciais podem ora fluir por um caminho, ora por outro, sendo importante o acompanhamento do resultado das decisões tomadas, efetuando ajustes o tempo todo, à medida que falhas forem diagnosticadas.

Ao tomador de decisão em organizações de software, é fundamental o conhecimento claro tanto dos objetivos de sua organização, como das características e dificuldades da atividade de manutenção de software, para então somente ser capaz de conduzir adequadamente sua equipe.

Dentro desse contexto, Basili (1990) propôs três paradigmas diferentes para tratar a manutenção de software: (i) Conserto Rápido; (ii) Melhoria Interativa; (iii) Reuso Total.

A idéia de conserto rápido compreende aplicar primeiramente as mudanças necessárias no código-fonte do software e então depois atualizar a documentação, nessa ordem. Essa abordagem corresponde à normalmente preferida pelo mantenedor, justificando a decisão pela urgência da manutenção (Visaggio, 2001). A decisão de alterar a documentação após a manutenção, no entanto, não é comumente uma boa prática, pois pressionado pelo tempo, o mantenedor executa mal, ou não executa, essa tarefa, dificultando cada vez mais manutenções posteriores. Na **Figura 7** é mostrado um paralelo entre o software atual e o derivado da manutenção, notando-se que o ponto de partida é o código-fonte do software antes da manutenção.

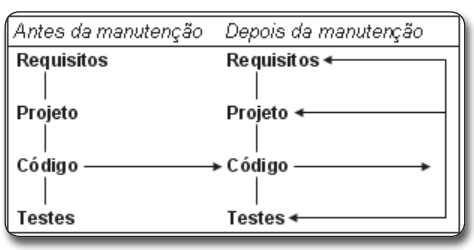


Figura 7. Estrutura da abordagem conserto rápido

A proposta de melhoria interativa, por sua vez, propõe inicialmente adequar e atualizar a documentação de alto nível que será afetada, para então propagar as mudanças até o nível de código-fonte. Na **Figura 8** é representada essa idéia.

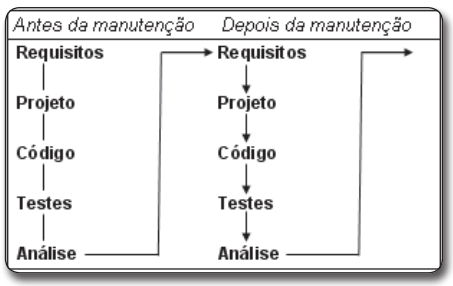


Figura 8. Estrutura da abordagem melhoria interativa

Finalmente, a última proposta, reuso total, consiste em construir um novo software, utilizando componentes do software antigo e outros disponíveis em um repositório. Na **Figura 9** essa idéia é mostrada.

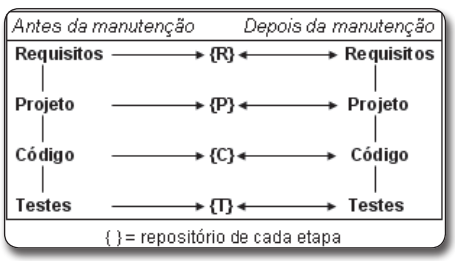


Figura 9. Estrutura da abordagem reuso total

Essa última proposta não é freqüentemente usada no ambiente industrial, e uma das razões seria a deficiência ainda existente na tecnologia para gerenciar o repositório e o reuso de componentes de software.

A comparação entre os dois paradigmas mais comuns, o conserto rápido e a melhoria interativa, foi feito por Visaggio (2001) e o resultado apontou que a primeira abordagem apenas é mais indicada do que a segunda, quando o impacto da modificação envolver poucos componentes, o que pode ser reforçado pelo menor tempo empregado para tornar a manutenção operacional. Porém, e principalmente, para alterações maiores, o emprego da melhoria contínua é a melhor indicação, pois permite antever defeitos e conseqüentemente reduzir re-trabalhos, contribuindo para a garantia de qualidade global do software. Esse último método funciona ainda como uma maneira automática de continuamente revisar a consistência do software.

Nesse ponto, o autor deste artigo tem a acrescentar que verificou, por sua experiência na prática em algumas organizações, o emprego do método de corrigir o problema diretamente no código-fonte, exceção feita em algumas situações mais complexas que envolvem pequenas reuniões entre os profissionais responsáveis pela manutenção, produzindo em algumas delas um roteiro escrito de

passos ou de fatores a observar para a manutenção. No entanto, na maior parte dos casos, esses registros não eram utilizados para a atualização da documentação de software existente. O procedimento mais adotado é o de revisar e completar a documentação de tempos em tempos, com base em visões gerais do software. É justo destacar, no entanto, que algumas organizações decidem por desenvolver internamente seu próprio sistema de controle de manutenções, no qual se registra requisições formais de manutenção, podendo ser mais ou menos completo dependendo da organização, contendo descrição da solução, tempo despendido, estrutura do software entre outros. Essa postura, no entanto, é normalmente adotada por organizações de médio e grande porte. Existem trabalhos dedicados a sistematizar e melhorar a construção desse tipo de ferramenta, como por exemplo, o trabalho de Koskinen et al. (2004), que propõe uma ferramenta para auxiliar a recuperação de diversas informações sobre um software que sofrerá manutenção.

Outro desafio gerencial notável em manutenção de software relaciona-se à necessidade de estimar esforço para atividade, conforme explica De Lucia et al. (2004). Com uma maneira eficaz de estimar o esforço, fica mais fácil aos tomadores de decisão argumentar sobre as melhores opções para a organização, contribuindo para a decisão de terceirizar ou não a atividade. Esse auxílio na tomada de decisão envolve considerações como:

- Reavaliar a eficiência de processos de manutenção;
- Tratar a manutenção ou reengenharia;
- Fundamentar as decisões tomadas;
- Planejar o orçamento de equipe e avaliar a rotatividade.

Definir quem fará a manutenção é um ponto que pode levar ao fracasso ou ao sucesso dos esforços em manter um software em funcionamento e adequado às necessidades. Portanto, decidir entre manter uma equipe interna de manutenção ou submeter o software à manutenção por outra organização exige estudo cuidadoso, e isso se justifica pela simples lembrança de que a organização pode ter dependência enorme em relação a esse software, necessitando de seu correto e adequado funcionamento o tempo todo.

Aplicar à manutenção de software as técnicas, ferramentas, modelos de processos, etc., próprios do desenvolvimento de software não constitui uma decisão correta, pois manutenção e desenvolvimento apresentam características distintas, como afirma Polo et al. (2003).

Algumas razões que justificam essas características distintas são apresentadas pelo mesmo autor:

- O esforço despendido em tarefas de desenvolvimento e manutenção não é igual, constituindo uma prática comum nas organizações concentrar a maior parte do esforço humano disponível em codificação e tarefas de teste durante o desenvolvimento.
- Outro fator de diferença está ligado ao fato de que a manutenção tem mais similaridade com um serviço do que com desenvolvimento, o que acaba implicando na necessidade de não aplicar as mesmas estratégias de desenvolvimento em manutenção.

Essa última razão pode ainda ser apontada como fator que justifica o aumento no número de organizações que oferecem serviços de manutenção de software, e esse aumento estaria ainda ligado à alta demanda já existente.

De fato, a prestação de serviços de manutenção de software, mais do que uma situação nova, na verdade, trata-se de um setor já competitivo, como afirma Bhatt et al. (2006). O autor explica que nos primeiros momentos o que se viu foram clientes contratando organizações de software e pagando com base no número de engenheiros de software envolvidos nas atividades de manutenção. Atualmente, observa uma mudança nesse cenário, com os clientes contratando as organizações por um preço fixo, estipulado por um período determinado e com base em estimativas a respeito da atividade de manutenção que será necessária.

Histórico dos Problemas de Manutenção de Software

Edward Yourdon (Yourdon, 1992), considerado um dos nomes mais importantes da análise estruturada clássica, estimou, ainda no início dos anos 1990, que a manutenção de software viria a ser um dos problemas relevantes no futuro. Essa previsão foi feita com base em constatações da época, como a política de entregar em tempo ao usuário um sistema que funcionasse, não se preocupando com questões de manutenibilidade. Mais do que isso, Yourdon afirmou que a postura da alta direção das organizações que dependiam de software seria a de atacar o problema apenas quando ele emergisse de fato.

Antes, porém, do que afirmou Yourdon, ainda na década de 1980, estudos referentes a problemas com manutenção de software já ocorriam, e um pioneiro e relevante trabalho foi realizado por Lientz e Swanson (1980), publicado na forma de um livro. Esse trabalho descreve os resultados obtidos com estudos no intuito de averiguar a forma como as organizações tratavam a questão de manutenção de software na época.

A pesquisa contou com duas fases. Na primeira, realizada com 69 organizações, foram levantadas as características da atividade de manutenção de software por elas realizada. Essa fase inicial obteve as seguintes conclusões:

- A manutenção de software existente consome uma média de 48% das horas anuais do pessoal de sistemas e programação;
- Aproximadamente 60% dos esforços de manutenção são dedicados a manutenções do tipo perfectivas. Dentre esse percentual, dois terços se referem a esforços de melhoria de software;
- Problemas de natureza gerencial em manutenção foram vistos como mais importantes do que aqueles de natureza técnica.

Com base nessas verificações, uma segunda fase foi conduzida na pesquisa, nesse caso envolvendo 487 organizações, o que abrangia instituições governamentais, bancos, transportes, mineração, educação e indústrias de manufatura em geral, localizadas nos Estados Unidos e Canadá. O intuito dessa segunda fase foi o de validar os resultados obtidos na primeira, buscando uma compreensão mais profunda e abrangente.

Para se obter esses resultados, os autores buscaram saber das organizações as características dos esforços de manutenção empregados em sistemas considerados de grande importância,

que constituíam resultados de investimentos significativos.

A coleta de dados se deu por meio de questionários enviados para 2000 profissionais que ocupavam cargos de diretoria, gerência ou supervisão de departamentos de processamento de dados. Os questionários foram cuidadosamente elaborados. Cada questionário consistia de dezenove páginas, com perguntas que envolviam totais, porcentagens e respostas sim ou não para diversas características da organização e da manutenção por ela praticada.

Uma característica importante do questionário foi considerar a confiabilidade das informações fornecidas, reconhecendo também que nem sempre era fácil ao profissional entrevistado obter os dados para responder algumas questões. Para isso, após cada questão, era fornecido um quadro (ver **Tabela 2**), que visava avaliar o conhecimento que o entrevistado tinha a respeito do que acabava de responder.

Marque a sentença apropriada a resposta anterior é:
a. Razoavelmente precisa, baseada em dados confiáveis.
b. Uma estimativa grosseira, baseado em dados mínimos.
c. Uma suposição, sem base em dados.

Tabela 2. Verificação do conhecimento sobre cada item respondido

Do total de questionários enviados, somente 487 respostas foram obtidas, o que representou uma taxa de resposta menor que 25%.

Nessa segunda fase, os principais resultados obtidos estão resumidos a seguir:

- Mais de um terço dos sistemas descritos tinham sua manutenção realizada por apenas uma pessoa;
- A maioria dos profissionais alocados para manutenção tinha outras tarefas ao mesmo tempo;
- Do total de manutenções, cerca de 20% representavam manutenções corretivas, 25% manutenções adaptativas e mais de 50% manutenções perfectivas (isso está de acordo com aos resultados obtidos na primeira fase da pesquisa);
- Quanto mais velho era o sistema, maior era o esforço que deveria ser empregado em sua manutenção, e mais sujeita a organização estava em perder o conhecimento em torno desse sistema, devido à rotatividade dos profissionais.

Finalmente, os autores chegaram aos principais problemas de manutenção enfrentados pelas organizações na época:

- Baixa qualidade da documentação dos sistemas;
- Necessidade constante dos usuários por melhorias e novas funcionalidades;
- Falta de uma equipe de manutenção;
- Falta de comprometimentos com cronogramas;
- Treinamento inadequado do pessoal de manutenção;
- Rotatividade dos profissionais.

Conforme explicam os autores, desses seis problemas, três estão relacionados com os usuários do software, dois estão relacionados a restrições gerenciais e um trata de problema com documentação, representando uma questão de natureza mais técnica.

No geral, foi verificada uma predominância de problemas não-técnicos sobre problemas técnicos, o que reforça a necessidade de uma maior atenção a questões gerenciais em manutenção de software.

Em 1986, outro estudo (Chapin, 1986) envolveu 250 gerentes de manutenção de software e buscou saber quais eram os principais problemas de manutenção nas equipes por eles lideradas. Dos resultados obtidos, foi estabelecida uma classificação em oito categorias. Entre os problemas apontados, a grande maioria referia-se a características do próprio software, como documentação insuficiente e código excessivamente complexo ou mal estruturado.

Em outra pesquisa (Dekleva, 1990), realizada pelo Software Maintenance Association (SMA), o intuito foi a obtenção de respostas para a seguinte pergunta: "Quais os três principais problemas de manutenção de software no seu departamento de sistemas de informação?", sendo que os resultados permitiram apontar questões ligadas principalmente a dificuldades de gerenciamento, características de desenvolvimento de software, administração de equipes e mudanças no ambiente externo ao software.

Em 1992, valendo-se de uma técnica de pesquisa iterativa denominada delphi, que leva a um consenso entre os entrevistados, Dekleva (1992) obteve uma lista de problemas de manutenção muito parecida com os apresentados por Lientz e Swanson, incluindo apenas o problema da dificuldade da organização em medir o desempenho de sua equipe de manutenção. Novamente, foi constatado que problemas de ordem gerencial são absolutamente dominantes no contexto de problemas de manutenção de software. Na **Figura 10**, extraída do trabalho de Dekleva (1992), é mostrada a relação de causa identificada entre os fatores envolvidos na atividade de manutenção. As más características ou mal gerenciamento de um fator, prejudicam o fator correlacionado.

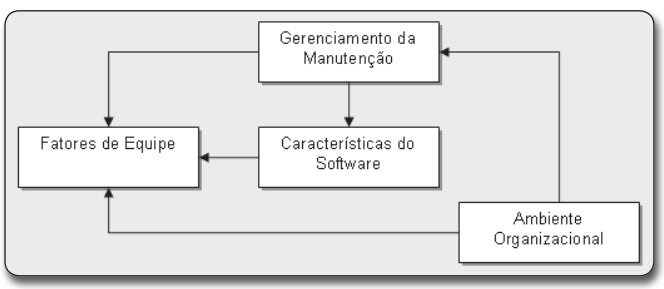


Figura 10. Relação de causa entre os fatores envolvidos na manutenção de software

Existem ainda outros trabalhos com resultados semelhantes, como o de Dart et al. (2001), conduzido pelo Software Engineering Institute (SEI) junto a uma agência do governo dos Estados Unidos. Uma característica interessante apontada nesse estudo refere-se à alegação dos profissionais encarregados de tarefas de manutenção de que acabam não tendo contato com ferramentas que representariam o estado-da-arte da tecnologia. Outro fator apontado refere-se ao fato de que

as lições adquiridas com a experiência de manutenção em um determinado software não estavam sendo disseminadas para as outras equipes dentro da mesma organização.

Considerações Finais

A grande maioria dos estudos existentes na área de manutenção de software utiliza a metodologia de "pesquisar-e-transferir", no sentido de pesquisa fechada dentro do meio acadêmico. No entanto, existe uma necessidade de utilizar a abordagem "indústria-como-laboratório", uma vez que é nas indústrias que a atividade de manutenção existe como atividade real e intensa no dia-a-dia, de acordo com a visão de Niessink (1999).

Nesse sentido, a manutenção de software deve ser considerada, assim como é a engenharia de software, com base em seu ambiente de aplicação, e essa abordagem representa a melhor fonte de evidências para sustentar e guiar as pesquisas de melhoria e sistematização da atividade. ●

Referências

Bennett, K. H.; Rajlich, V. T. (2000) "Software maintenance and evolution: a roadmap", In: Conference on The Future of Software Engineering, Limerick, Ireland, June.

Bennett, K. H.; Ramage, M.; Munro, M. (1999) "Decision Model for Legacy Systems", IEEE Proceedings on Software (TSE), v.146, n. 3, p. 153-159.

Bhatt, P.; Shroff, G.; Misra, A. K. (2004) "Dynamics of software maintenance", ACM SIGSOFT Software Engineering Notes, v. 29, n. 5, p. 1-5.

Basili, V. (1990) "Viewing Maintenance as Reuse-Oriented Software Development", IEEE Software, v. 7, n. 1, p. 19-25.

Chapin, N. (1986) "Software maintenance: A different view", In: Conference on Software Maintenance, Orlando, FL, USA, November.

Dart, S.; Christie, A. M.; Brown, A. W. (2001) "A Case Study in Software Maintenance", Technical Report, Carnegie Mellon University.

Dekleva, S. M. (1990) "Annual Software Maintenance Survey: Survey Results", Software Maintenance Association, Vallejo, California, USA.

_____. (1992) "Delphi study of software maintenance problems", In: Conference on Software Maintenance, Orlando, FL, USA, November.

De Lucia, A., Pompella, E., Stefanucci, S. (2004) "Effort estimation for corrective software maintenance". In: 14th International Conference on Software Engineering and Knowledge Engineering, Ischia, Italy, July.

IEEE (1998) "Std 1219 – IEEE Standard for Software Maintenance", Institute of Electrical and Eletronic Engineers, New York, NY, USA.

ISO/IEC 12207 (1998) "Standard for Information Technology - Software Lifecycle Processes", International Standard Organization, New York, NY, USA.

ISO/IEC 15504 (2003) "Software Process Assessment", International Standard Organization, New York, NY, USA.

Koskinen, J.; Salminen, A.; Paakki, J. (2004) "Hypertext support for the information needs of software maintainers", Journal of Software Maintenance and Evolution: Research and Practice, v. 16, n. 3 , p. 187-215.

Kung, H.; Hsu, C. (1998) "Software Maintenance Life Cycle Model", In: International Conference on Software Maintenance, Bethesda, Maryland, USA.

Lehman, M. M. (1974) "Programs, Cities, Students, Limits to Growth?", In: Imperial College of Science Technology, London, England, May.

_____. (1980) "On Understanding Laws, Evolution and Conservation in the Large Program Life Cycle", Journal of Systems and Software (JSS), v. 1, n. 3, p. 213-221.

_____. (1991) "Software Engineering, the Software Process and their Support", IEEE Software Engineering Journal: Special Issues on Software Environments and Factories, v. 1, n. 3, p. 213-221.

_____. (1996) "Laws of Software Evolution Revisited". In: 5th European Workshop on Software Process Technology, Nancy, France, October.

Lientz, B. P.; Swanson, E. B. (1980) "Software Maintenance Management", Reading, MA, Addison-Wesley.

Niessink, F. (1999) "Software Maintenance Research in the Mire?", In: Annual Workshop on Empirical Studies of Software Maintenance (WESS'99), Oxford, United Kingdom, September.

Pfleeger, S. L. (2001) "Software Engineering: theory and practice". Second Edition, New Jersey, Prentice Hall.

Pigoski, T. M. (1996) "Practical Software Maintenance: Best Practices for Managing Your Software Investment", Willey Computer Publishing.

Polo, M.; Piattini, M.; Ruiz, F.; Calero, C. (1999) "Roles in the maintenance process", ACM SIGSOFT Software Engineering Notes, v. 24, n. 4, p. 84-86.

Polo, M.; Piattini, M.; Ruiz, F. (2003) "Using a qualitative research method for building a software maintenance methodology", Software – Practice and Experience, v.32, n. 13, p. 1239–1260.

Pressman, R. S. (2005) "Software Engineering: a practitioner's approach", 6.ed., McGrawHill Higher Education.

Singh, R. (1996). "International Standard ISO/IEC 12207 Software Life Cycle Processes", Software Process Improvement and Practice, vol. 2, p. 35–50.

Sneed, H. M. (2003) "Critical Success Factors in Software Maintenance", In: International Conference on Software Maintenance, Amsterdam, The Netherlands, September.

Sommerville, I. (2003) "Engenharia de Software", 6.ed., São Paulo, Addison Wesley.

Silva, L. de P.; Santander, V. F. A. (2004) "Uma Análise Crítica dos Desafios para Engenharia de Requisitos em Manutenção de Software", In: Workshop em Engenharia de Requisitos, Tandil, Argentina, Dezembro.

Souza, S. C. B. de; Neves, W. C. G. das; Anquetil, N.; Oliveira, K. M. de. (2004) "Documentação Essencial para Manutenção de Software II", In: I Workshop de Manutenção de Software Moderna, Brasília, DF, Brasil, Outubro.

Ulrich, W. M. (1990) "The evolutionary growth of software reengineering and the decade ahead". American Programmer, v. 3, n. 10, p. 14-20.

Visaggio, G. (2001) "Assessing the Maintenance Process Through Replicated, Controlled Experiment". Journal of Systems and Software, v. 44, n. 3, p. 187-197.

Yourdon, e. (1992) "Análise Estruturada Moderna", tradução da terceira edição, Rio de Janeiro, Campus.

Reengenharia de Software

Uma visão geral

Ana Paula Gonçalves Serra
prof.anapaula@usjt.br
Doutora e Mestre em Engenharia de Computação e Sistemas Digitais pela Escola Politécnica da Universidade de São Paulo. Bacharel em Ciência da Computação pela Universidade São Judas Tadeu. Professora na área de Engenharia de Software da Universidade São Judas Tadeu nos cursos de Pós-Graduação e Graduação. Professora de Engenharia de Software da FIAP no curso de Graduação. Atua como consultora na área de Engenharia de Software. Possui 15 anos de experiência profissional em desenvolvimento de software. Possui vários trabalhos nacionais e internacionais publicados nas áreas de Engenharia de Software, Sistemas de Informação, Automação de Sistemas e Qualidade de Serviço.

O que é reengenharia de software? De modo geral, pode-se considerar que reengenharia de software é o re-desenvolvimento de software legado para que sua manutenção seja mais fácil? Mas, o que são softwares legados? Por que eu preciso de uma manutenção mais fácil? Eu não devo desenvolver software e realizar manutenção por um tempo, depois descartá-lo e criar um novo software?

Essas são algumas questões que muitos de nós já fizemos, principalmente pessoas que já tiveram a oportunidade de realizar manutenção em software. Antes de abordar o conceito de reengenharia de software, serão comentados os conceitos de software legado e evolução de software.

Software Legado

Muitas vezes a impressão que temos de software legado é que são softwares antigos, ultrapassados, mas por alguns motivos, como por exemplo: custos, riscos de substituição e regras de negócio complexas, não são substituídos. Além disso, muitas vezes os profissionais não desejam trabalhar em softwares legados,

Neste artigo veremos

Este artigo tem como objetivo apresentar uma visão geral sobre softwares legados, evolução de software e reengenharia de software. Para isso, é apresentado o conceito de softwares legados, as categorias da evolução de software, que são: manutenção de software, modernização de software e substituição de software. Além disso, são apresentadas atividades e técnicas de reengenharia de software.

Qual a finalidade

Introduzir conceitos de softwares legados e evolução de software, com ênfase em reengenharia de software e fornecer diretrizes de tomadas de decisão do que fazer com um software legado.

Quais situações utilizam esses recursos?

Este artigo é útil para pessoas que trabalham com manutenção de software, pois fornece um panorama da evolução de software e pode auxiliar na tomada de decisões do que fazer com o software legado. Devo continuar fazendo manutenção de software? Realizar reengenharia de software? Substituir o software? E no caso da reengenharia de software apresenta algumas atividades que podem ser realizadas.

pois geralmente esses softwares não possuem especificação, modelagem de solução, documentação das regras de negócio, além de um código sem padrão, pois equipes diferentes podem ter feito manutenção do software. O resultado disso é uma manutenção dispendiosa, trabalhosa e incerta no sentido que uma correção ou melhoria pode levar a um erro em outra parte do software, os famosos "efeitos colaterais" de manutenção.

O fato é que softwares legados não são exclusivamente softwares antigos e ultrapassados, e sim qualquer software implantado e que terá modificações, como: correções, melhorias, mudança nas regras de negócio, ou seja, uma evolução natural do software.

No livro *Modernizing Legancy System*, o autor Seacord et al. resume softwares legados em uma frase direta "Um software legado é um software que foi escrito ontem". Sommerville é mais formal e abrangente, e define sistemas legados como "sistemas sócio-técnicos baseado em computadores. Esses sistemas incluem software, hardware, dados e processos corporativos. As modificações em uma parte do sistema envolvem inevitavelmente modificações em outros componentes".

Evolução de Software

A evolução de software é uma atividade de mudança de software, que pode ser desde a inserção de um campo na base de dados até a completa re-implementação do software. A evolução de software pode ser dividida em três categorias: manutenção, modernização e substituição de software.

A manutenção de software é um processo incremental e repetitivo, onde alterações são feitas no software. Essas alterações envolvem eliminação de erros e melhorias funcionais. A manutenção é necessária para suportar a evolução de qualquer software, mas não deve envolver mudanças estruturais, como por exemplo, adoção de novas tecnologias para implantação de uma arquitetura distribuída. A manutenção de software possui ainda quatro categorias:

- **Corretiva:** Alterações para corrigir erros no software;
- **Melhoria:** Alterações para melhorar o software, como por exemplo: novas funções, melhoria do desempenho ou usabilidade;
- **Adaptativas:** Alterações para adaptar o software a novos ambientes, tais como: sistemas operacionais, ferramentas, banco de dados ou componentes;
- **Preventivas:** Alterações para melhorar a manutenibilidade do software. O objetivo desta manutenção é simplificar evoluções futuras.

Com o tempo a manutenção de software pode causar alguns problemas, como: complexidade crescente, o que leva a estrutura deteriorada; qualidade em declínio; custo alto de manutenção, devido à instabilidade da equipe, contrato de manutenção, falta de conhecimento do negócio e do software pelos desenvolvedores que irão fazer a manutenção; entre outros.

Já a modernização de software é utilizada quando o software legado requer mudanças mais extensas e significativas do que aquelas realizadas pela manutenção, mas preserva uma porção significativa do software. A necessidade da modernização pode ocorrer devido, por exemplo, à fragilidade e pouca consistência do software legado, da falta de flexibilidade, isolamento e pouca capacidade de extensão.

A substituição de software requer a construção do software legado desde o início, e é apropriada quando o software legado não consegue mais atender às necessidades do negócio e quando a modernização não é possível ou não vale a pena em relação aos custos.

A **Figura 1** apresenta como as atividades de evolução de software são aplicadas em diferentes etapas do ciclo de vida de um sistema de informação. A linha pontilhada representa a necessidade crescente do negócio. As linhas contínuas representam as funcionalidades fornecidas pelo software. A manutenção constante do software permite atender os requisitos por um determinado período, mas com o tempo a manutenção não é mais eficiente. Com isso, a modernização que demanda mais tempo e esforço que a manutenção será necessária. Finalmente quando o software legado não puder mais evoluir, deverá ser substituído.

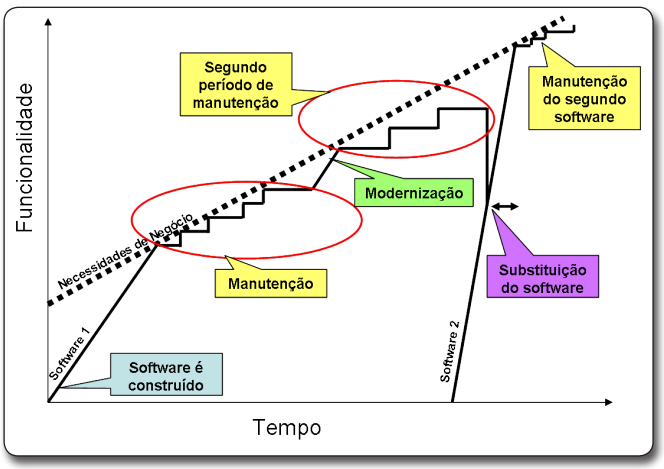


Figura 1. Ciclo de Vida de um Sistema de Informação - Seacord et al. (2003). Modernizing Legancy Systems.

Reengenharia de Software

A reengenharia de software é considerada uma modernização de software, ou seja, uma abordagem disciplinada para migrar softwares legados em softwares evolutivos. Essa opção deve ser escolhida quando a qualidade do software está degradada por modificações regulares e o software possui um alto grau de valor de negócio para empresa, sendo que novas mudanças regulares ainda serão exigidas. Um exemplo disso é um software que serviu às necessidades do negócio de uma empresa por 10 ou 15 anos. Durante esse tempo foi corrigida, adaptada e aperfeiçoada muitas vezes, e nem sempre aplicando boas práticas de engenharia de software, por aspectos de prazo e/ou custo. Com isso, o software está instável, ainda funciona, mas toda vez que uma modificação é tentada, efeitos colaterais inesperados e sérios ocorrem. No entanto a aplicação de software precisa continuar a evoluir.

O processo de reengenharia de software aplica os princípios da engenharia de software em um software legado para atender requisitos existentes e novos requisitos.

Segundo Pressman "a reengenharia tem como objetivo principal melhorar a qualidade global da aplicação, mantendo,

em geral, as funções da aplicação existente. Mas, ao mesmo tempo, pode-se adicionar novas funções e melhorar o desempenho".

Segundo o SEI (*Software Engineering Institute*) "reengenharia é uma transformação sistemática de uma aplicação de software existente para uma nova forma, realizando melhorias na operação, na funcionalidade, no desempenho, na capacidade de evoluir para um novo sistema com menores custos, prazos e riscos".

Segundo Furlan reengenharia é um "Conjunto de técnicas e ferramentas orientadas à avaliação, reposicionamento e transformação de sistemas de informação existentes, com o objetivo de estender-lhes a vida útil e ao mesmo tempo, proporcionar-lhes uma melhor qualidade técnica e funcional".

A **Figura 2** apresenta um esquema diferenciando a engenharia de software em um novo software e a reengenharia de software em um software legado. A principal diferença entre reengenharia e o desenvolvimento de um novo software é o ponto de partida para o desenvolvimento. Isto ocorre, pois na reengenharia o software legado atua como especificação e o processo de desenvolvimento tem como base compreender e transformar o software legado. Essa compreensão e transformação podem ser realizadas através das atividades da reengenharia de software.

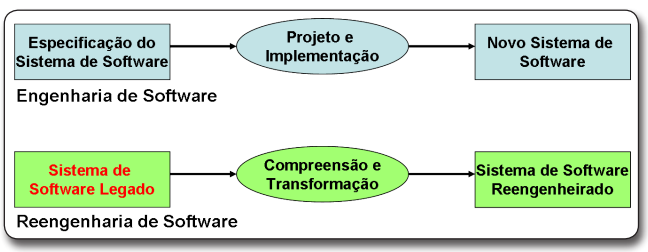


Figura 2. Diferença entre engenharia de software e reengenharia de software

Alguns exemplos de aplicações de reengenharia de software são: migração de softwares legados para uma nova plataforma, alteração da interface humano computador e reestruturação do código e documentação.

Atividades da Reengenharia de Software

Antes de iniciar qualquer atividade de reengenharia de software, deve-se constatar a real necessidade de modernização do software. Determinar as atividades apropriadas em pontos diferentes do ciclo de vida é um desafio. Com isso, surgem algumas perguntas. Deve-se continuar a fazer manutenção no software ou modernizá-lo? Deve-se substituir? As empresas dependem dos sistemas legados e têm um orçamento limitado para evolução de software precisando decidir como obter o melhor retorno de seu investimento. Isso significa que elas devem fazer uma avaliação realista dos softwares legados, e então decidir sobre a estratégia mais apropriada para que esses softwares continuem a evoluir.

Uma das formas de se efetuar esta tarefa é a realização da análise de inventário ou portfólio dos softwares legados da empresa. O inventário dos softwares legados da empresa contém dados importantes como: tamanho, idade, importância para o negócio, tecnologias utilizadas e integração com outros softwares.

Basicamente, quando se está avaliando um software legado, deve-se considerar duas perspectivas. A perspectiva de negócio, que avalia o valor do software para a empresa; e a perspectiva de sistema, que avalia a qualidade do sistema que engloba software, hardware e infra-estrutura de apoio ao software. A combinação dos dois pontos de vista de negócio e de qualidade do sistema pode ser utilizada para ajudar na decisão do que fazer com o software legado. A **Figura 3** apresenta um gráfico que pode auxiliar na análise de inventário dos sistemas de software.

Figura 3. Gráfico de análise de inventário - Seacord et al. (2003). Modernizing Legacy Systems.

Quadrante 1 – Substitua por Pacotes Comerciais: Sistemas de softwares legados com pouco valor de negócio e qualidade técnica pobre são candidatos à substituição por pacotes comerciais. Exemplos desses sistemas de software são: folha de pagamento, recursos humanos, ou serviços similares que não fazem parte do "centro de negócios" da empresa.

Quadrante 2 – Não Faça Reengenharia: Sistemas de softwares legados com alta qualidade técnica e baixo valor de negócio são candidatos a não realização de reengenharia, e sim práticas de evolução de sistemas (manutenções preventivas), pois devido à alta qualidade técnica e baixo valor de negócio não há necessidade de esforço de mudança.

Quadrante 3 – Baixa Prioridade de Reengenharia: Sistemas de softwares legados com alta qualidade técnica e alto valor de negócio são candidatos a evolução do sistema utilizando práticas de evolução de sistemas (manutenções preventivas), devido à alta qualidade técnica e alto valor de negócio há uma preocupação com as mudanças, mas não há necessidade de esforço de reengenharia.

Quadrante 4 – Boa Candidata a Reengenharia: Sistemas de softwares legados com baixa qualidade técnica e alto valor

de negócio são candidatos à reengenharia de software após uma análise do sistema. Essa análise do sistema inclui: custo, planejamento, políticas da empresa, equipe de desenvolvimento disponível, habilidade técnica requerida para esforço de reengenharia, treinamento de usuários, aceitação de usuários, etc. A análise é essencial para priorizar sistemas candidatos à reengenharia e qual será a estratégia de desenvolvimento. A maior justificativa de reengenharia é o alto risco de falhas, problemas de desempenho, confiabilidade, entre outros.

Contudo, em muitos casos, essas decisões não são tão simples e objetivas, mas se baseiam em considerações organizacionais ou políticas. Por exemplo, se existe uma fusão entre empresas, o sistema de software utilizado pela maior empresa poderá ser escolhido. Se uma área toma a decisão de migrar para uma nova plataforma de hardware, isso poderá exigir uma substituição de software. Outros fatores podem contribuir com isto como: mudanças de versão da linguagem de programação atual por uma versão mais recente, padronização da empresa por uma mesma linguagem de programação, falta de suporte de fornecedores a um software pela descontinuidade da versão.

A análise de inventário deve ser revisitada de tempos em tempos, pois as prioridades podem mudar.

Constatada a necessidade de reengenharia de software, quatro atividades básicas podem ser aplicadas:

1. Reestruturação de Documentos

Esta atividade consiste na análise da documentação do software legado, servindo como base para a especificação do software que passará pela reengenharia de software. Infelizmente, mas realisticamente, pouca documentação é uma das características dos softwares legados.

Muitas vezes para iniciar essa reestruturação de documentos é necessário passar pela atividade de engenharia reversa para entender o que está por "trás" do código fonte.

2. Tradução do Código Fonte

Dependendo do processo de reengenharia, pode-se realizar a tradução do software direta de uma linguagem de programação para outra ou para uma versão mais atualizada. Esta atividade pode ser realizada quando não há necessidade de compreender detalhadamente a operação do software ou de modificar a arquitetura do sistema, garantindo que a estrutura e a organização do software permaneçam inalteradas. Algumas situações em que esta atividade pode ser realizada são: mudanças de versão da linguagem de programação atual por uma versão mais recente, padronização da empresa por uma mesma linguagem de programação, falta de suporte de fornecedores a um software pela descontinuidade da versão ou da linguagem de programação.

O mais viável para essa atividade é que se utilize uma ferramenta CASE (*Computer Aided Software Engineering*) para a tradução do código fonte visando a automatização. É importante notar que esta automatização não será total pela ferramenta, pois haverá situações que a ferramenta não será capaz de resolver, como instruções condicionais de compilação que não sejam compatíveis com a linguagem de programação alvo. Entretanto, de modo geral, as ferramentas CASE com esse objetivo facilitam o trabalho, reduzem tempo e custo.

3. Engenharia Reversa

A engenharia reversa é um processo retroativo do ciclo de vida, que parte de um baixo nível de abstração (código-fonte ou executável) para um alto nível de abstração (modelos de projeto e especificação).

Muitas vezes a engenharia reversa é confundida com reengenharia de software. O objetivo da engenharia reversa é derivar o projeto e/ou especificação de um sistema de software a partir de seu código fonte. O objetivo da reengenharia é muito mais amplo, é produzir um sistema de software novo, baseado em um sistema legado com manutenção mais fácil e com qualidade.

A engenharia reversa é uma atividade que pode ser utilizada durante o processo de reengenharia de software a fim de recuperar o entendimento do software antes de reorganizar sua estrutura. Geralmente são utilizadas ferramentas CASE para apoiar esta atividade. Alguns exemplos são traduções do código fonte para diagrama de classes, *script* de banco de dados em modelo lógico de dados.

A engenharia reversa além de ser reutilizada na reengenharia de software, pode ser utilizada em manutenção de software e reestruturação/atualização de documentação.

4. Reestruturação do Código e Dados

Essa é a atividade mais importante e trabalhosa da reengenharia de software. Também chamada de *refactoring*, consiste em um processo de modificação de módulos de software de tal modo que não altere o comportamento externo do código, mas aperfeiçoe a estrutura interna. É um modo disciplinado de "limpar" o código que minimiza a introdução de defeitos e de modularizar os programas, com o objetivo de remover redundâncias nos componentes relacionados, otimizar suas interações e simplificar as interfaces do software. A refatoração pode ocorrer em uma mesma plataforma de desenvolvimento ou em uma nova.

Em paralelo com a reestruturação do código do software deve existir a reestruturação dos dados, ou seja, melhoria do projeto de banco de dados, reorganização dos dados e nomenclatura. Além de uma limpeza e migração dos dados. Geralmente existe uma mudança física nas estruturas de dados, contemplando mudança de formato de arquivos ou mudança de tecnologia de banco de dados.

Toda parte de reestruturação do código e dados geralmente é realizada manualmente, devendo-se aplicar princípios, conceitos, processos e métodos de engenharia de software para recriar um software existente com o objetivo de ampliar a capacidade do software antigo e garantir sua qualidade.

Algumas técnicas atuais que podem ajudar na reestruturação do código e dados são: orientação a objetos, padrões de projetos, COTS (*Commercial Off-The-Shelf*), *frameworks*, interfaces de comunicação (XML, IDL, ...), entre outros.

Conclusão

O cenário atual e real de muitas empresas é a presença de softwares legados essenciais para o negócio que possuem uma manutenção dispendiosa, trabalhosa, com alto custo para empresa e sem nenhuma capacidade de extensão. De acordo com Seacord et al., 2003, estima-se que a participação do custo de manutenção no custo total de um sistema tem crescido de 40%, nos anos 70, até 90%, atualmente. Dos custos com manutenção, segundo Urich, 2002, cerca de 20% são consumidos com correções e 80% com melhorias diversas.

Uma possível solução para o problema de softwares legados é a reengenharia de software, mas que também não é uma solução simples e milagrosa. Segundo uma pesquisa do *Standish Group* realizada em 2001, 23% dos projetos de reengenharia são cancelados antes de serem concluídos, enquanto que 28% terminam no tempo e orçamento previstos com o resultado esperados. Os demais não atendem aos resultados esperados ou terminaram fora do prazo/custo.

Observa-se com isso que a reengenharia de software deve ser aplicada de forma criteriosa e com uma gerência de riscos ativa; iniciando com a análise de inventário, definindo-se uma estratégia de re-desenvolvimento do software, como por exemplo, uma migração gradativa do software a fim de tornar esta tarefa mais confiável, rápida e flexível, realizando as atividades pertinentes de reengenharia de software. Além disso, deve-se utilizar constantemente diretrizes sobre qualidade, processos, métodos, ferramentas e técnicas da engenharia de software. ●

Referências

PRESSMAN, R. S. Engenharia de Software. 6ª. Edição. São Paulo: McGrawHill, 2006.

SEACORD, R. C. et al. Modernizing Legacy Systems. SEI Series in Software Engineering. Addison Wesley, 2003.

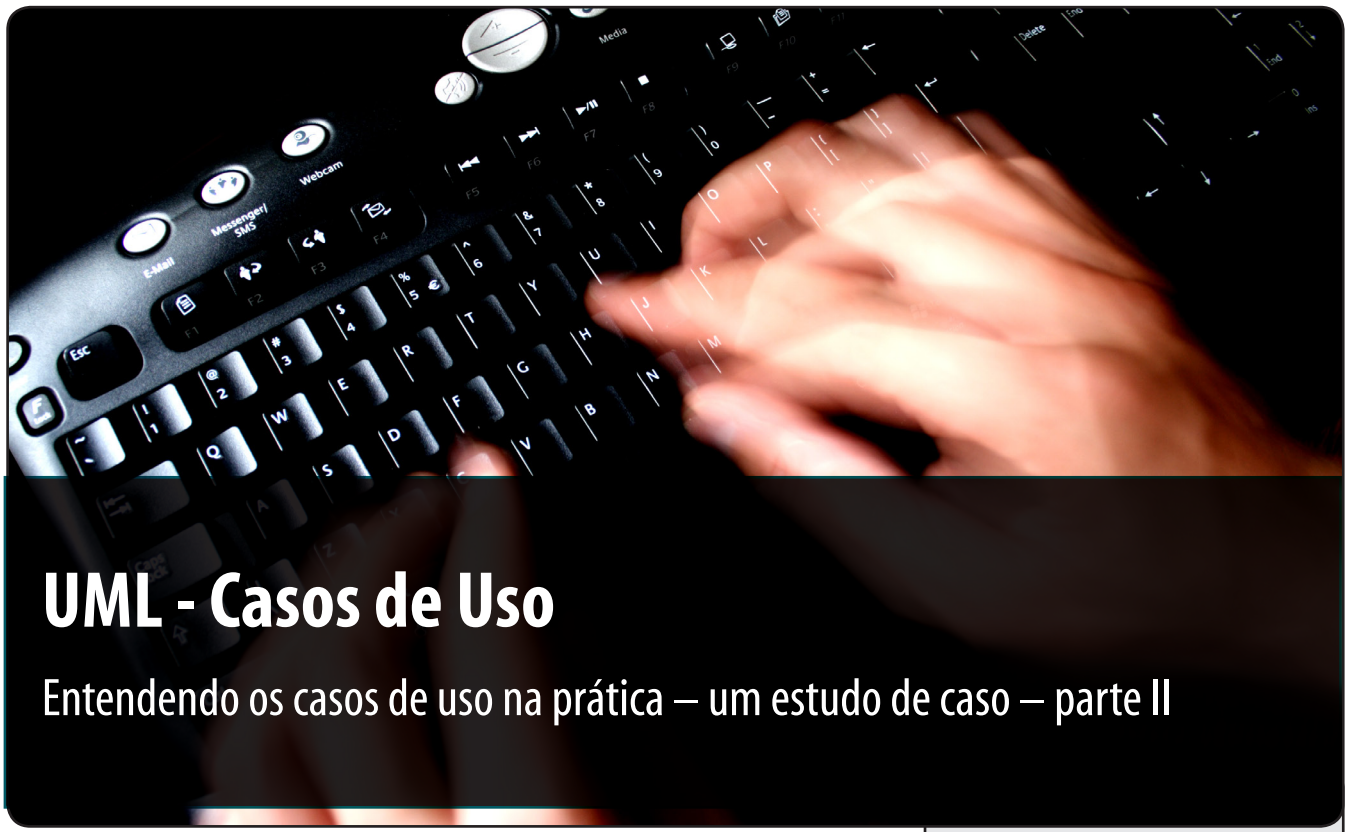
SOMMERVILLE, I. Engenharia de Software. 6ª. Edição. São Paulo: Addison Wesley, 2003.

ULRICH, W. M. Legacy Systems Transformation Strategies. PH PTR, 2002.

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine tem que ser feita ao seu gosto.
Para isso, precisamos saber o que você, leitor, acha da revista!
Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback



UML - Casos de Uso

Entendendo os casos de uso na prática – um estudo de caso – parte II

Neste artigo veremos

Este artigo dá continuidade ao artigo da edição anterior, no qual o caso de uso é apresentado de uma maneira prática.

Qual a finalidade

Fornecer aos desenvolvedores ou estudantes da área de sistemas uma linha de entendimento com o intuito de orientá-los a escrever seus próprios casos de uso.

Quais situações utilizam esses recursos?

Para quem ainda não conhece como escrever um caso de uso, ou para quem já o faz há algum tempo, mas não tem conseguido o sucesso esperado.

Ana Cristina Melo

É especialista em Análise de Sistemas e professora de graduação e pós-graduação da Universidade Estácio de Sá. Atua em análise e programação há 21 anos. Autora do livro "Desenvolvendo aplicações com UML - do conceitual à implementação", na segunda edição, e "Exercitando modelagem em UML" Palestrante em alguns eventos, entre eles, Congresso Fenasoft, OD e Sepai.

No artigo anterior apresentamos o problema de modelagem envolvendo o sistema de uma pequena papelaria. Os requisitos foram listados, contemplando uma sugestão de um formato para ata de reunião. Com base nos requisitos, foram relacionados os casos de uso possíveis, além de ter sido desenhado o diagrama de casos de uso. Desses, selecionamos três casos de uso, para os quais escrevemos todos os cenários, bem como apresentamos seus protótipos.

Nessa segunda parte selecionaremos outros casos de uso, com diferentes formatos, como o de relatório, registro de pagamento (com as diversas formas de pagamento) e o de processamento e controle (como fechamento de caixa).

Relembrando o modelo

Para que possamos acompanhar a escrita desses casos de uso, vamos relembrar o diagrama modelado no artigo anterior. Veja a **Figura 1**.

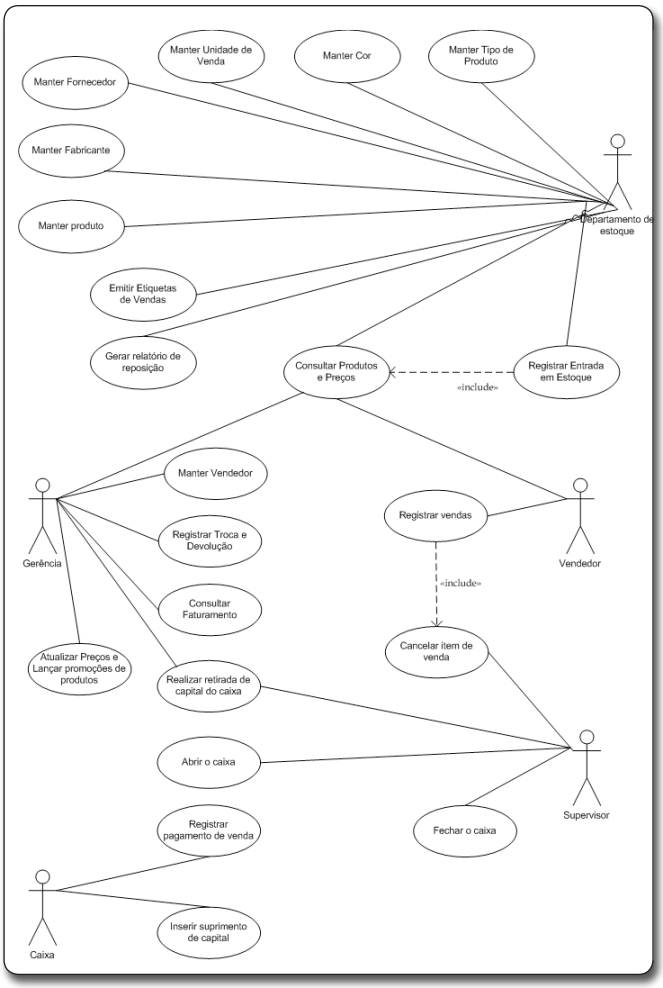


Figura 1. Diagrama de casos de uso para o sistema gestor de papelaria

Escrevendo novos cenários dos casos de uso

No artigo anterior, escrevemos os cenários para os seguintes casos de uso:

- Manter Produto
- Registrar Entrada em Estoque
- Consultar Produtos e Preços

O caso de uso Manter Produto nos apresenta a idéia do caso de uso de manutenção, com as funcionalidades de inclusão, alteração, exclusão (se for o caso) e consulta. O entendimento desse formato leva, por similaridade, à escrita de outros casos de uso: Manter Fabricante, Manter Tipo de Produto, Manter Fornecedor, Manter Unidade de Venda, Manter Cor e Manter Vendedor.

O caso de uso Registrar Entrada em Estoque possibilitou o entendimento de uma funcionalidade na qual se precisa da entrada de dados, e a partir dessa entrada o sistema executa um processamento, gerando uma saída persistida.

Já o caso de uso Consultar Produtos e Preços apresentou um modelo que requer critérios de busca. O resultado é exibido, com possibilidade de seleção e retorno para um caso de uso chamador. Não há processamento nesse tipo de caso de uso, nem persistência de dados, apenas recuperação de informações com base num conjunto de critérios informados pelo usuário.

Seguindo essa linha, apresentaremos outros casos de uso que tragam formatos diferentes, de forma a possibilitar não só o entendimento de todo o sistema, como dos desafios de modelagem que podem surgir em qualquer aplicação.

Na **Listagem 1** está descrito o caso de uso Emitir Etiquetas de Vendas. Para entendermos o que será preciso fazer, vamos relembrar os requisitos que se referem a esse caso de uso:

- Todos os produtos recebem um código que não está associado ao código de barras. São coladas etiquetas com esses códigos em todos os produtos. Da mesma forma, em cada prateleira há uma etiqueta com o nome dos produtos, seu código e o preço atualizado. Assim, nos produtos não há etiqueta de preço.

Numa revisão do analista de sistemas com o cliente, percebe-se que exibir apenas o nome do produto causará dúvidas no consumidor, visto que a diferenciação consta não no nome, mas sim nos detalhes, cor e até mesmo no fabricante. Muitas vezes, durante o levantamento de requisitos, esses detalhes podem passar despercebidos pelo cliente e pelo próprio analista. É normal e esperado. Assim, durante a modelagem dos casos de uso, dúvidas podem surgir, e se as respostas não corresponderem ao que já foi modelado ou definido como requisito, deve-se proceder às alterações necessárias.

Desta forma, para o requisito acima, foi feita a seguinte alteração:

- Todos os produtos recebem um código que não está associado ao código de barras. São coladas etiquetas com esses códigos em todos os produtos. Da mesma forma, em cada prateleira há uma etiqueta com o código dos produtos, nome, detalhes, cor, fabricante, unidade de venda e o preço atualizado, incluindo o preço normal e promocional, se for o caso. Assim, nos produtos não há etiqueta de preço.

O caso de uso tem início com o usuário informando os critérios de busca (item 1), a fim de localizar os produtos para os quais deseja imprimir etiquetas. Essa seleção pode ser individual, ou por um grupo de produtos a partir do tipo de produto, ou mesmo por um grupo de produtos sem nenhum critério comum. Repare que o cenário principal se limita a tratar o que há de mais importante como requisito: selecionar os produtos previamente, antes da impressão. Se apenas fossem apresentados os produtos e o usuário fosse selecionando-os com o CTRL, ainda assim esse requisito estaria atendido. Não importa a forma como um requisito será implementado, mas que sua necessidade seja atendida. Contudo, também é um requisito do cliente que essa seleção possa ser feita com a melhor usabilidade possível. Nesse caso, essas características não fazem parte do cenário principal, e são colocadas no cenário alternativo (veja os cenários Pesquisa de tipo do produto, Seleção geral e Seleção incremental).

A partir dos critérios informados pelo usuário, o sistema faz a busca, exibe os produtos encontrados, e habilita a múltipla seleção (itens 2 e 3).

Há agora um processamento batch (item 4), que varrerá toda essa lista selecionada, gerando uma etiqueta para cada item. Essa parte do caso de uso traz uma característica diferente. No seu corpo é apresentado o formato da etiqueta. Não um formato gráfico, com todos os detalhes, mas um formato que especifica o layout dessa etiqueta. Esse tipo de requisito é importante, pois não deixa sob responsabilidade do programador definir qual deve ser o melhor

formato a ser apresentado. Principalmente numa saída desse tipo (etiqueta exposta nas prateleiras), é imprescindível que o gestor do sistema dê a sua aprovação, para que não haja problemas futuros.

Outra característica importante que deve ser notada é que há um requisito de interface relevante para o cliente. O cliente precisa que a etiqueta tenha destaques visuais de tamanho de fonte e cor para chamar a atenção do consumidor. Esses requisitos não são relevantes nem no cenário principal, nem nos cenários alternativos, que devem tratar dos requisitos funcionais apenas. Contudo, não se pode desprezar essas solicitações; e melhor do que criar outro documento para relacionar esses tipos de requisitos, é associá-los a cada rotina pertinente. Sendo assim, esses requisitos são agrupados e colocados numa nova seção do caso de uso (Seção Requisitos de Implementação). Essa seção pode receber outras nomenclaturas, visto não ser um consenso sua inclusão.

A **Figura 2** apresentará o protótipo (1) do caso de uso Emitir Etiquetas de Vendas, na sua parte de pesquisa e seleção. O processamento não necessita de protótipo, visto não haver interface com o usuário. O produto desse processamento, a etiqueta, é apresentado na **Figura 3**, como o protótipo (2) do referido caso de uso.

Listagem 1. UC Emitir Etiquetas de Vendas

```
Descrição: Este caso de uso tem por objetivo emitir etiquetas de
vendas para os produtos da papelaria.
Atores: Departamento de Estoque
Pré-condição: existir cadastro prévio de produtos.
Cenário principal:
1. O usuário informa os critérios de seleção de produtos:
    - código (com a opção de todas as variações do produto
    localizado)
    ou
    - seleção por tipo do produto
2. O sistema pesquisa e exibe os produtos que satisfaçam os
   critérios informados, exibindo para cada um:
   2.1. código do produto
   2.2. nome do produto
   2.3. detalhes
   2.4. cor
   2.5. fabricante
   2.6. preço de venda
   2.7. preço de venda promocional
   2.8. unidade de venda
   2.9. tipo do produto
3. O usuário seleciona um ou mais produtos para geração das
   etiquetas.
4. Para cada produto selecionado, o sistema gera uma etiqueta
   com os dados e formato a seguir:
   4.1. código do produto (linha 1 - primeiro campo)
   4.2. nome do produto (linha 1 - segundo campo)
   4.3. detalhes do produto (linha 2 - primeiro campo)
   4.4. cor (linha 2 - segundo campo)
   4.5. fabricante (linha 2 - terceiro campo)
   4.6. unidade de venda (linha 2 - quarto campo)
   4.7. preço de venda (linha 3 - primeiro campo)

Formato:
-----------------------------------------------------------------
        999999 - XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
     XXXXXXXXXXXXXXXXXX - XXXXXXX - XXXXXXXXXXXX - XXXXXX
                          R$ 999,99
-----------------------------------------------------------------
Cenários alternativos:
Pesquisa de tipo do produto
1.a. A pesquisa de tipo do produto deve ser feita apenas com
     os tipos de produto já cadastrados no sistema.
Seleção geral
1.b. Deve ser exibido ao usuário a opção de selecionar todos
     os produtos, para geração das etiquetas.
Seleção incremental
1.c. Deve ser oferecido ao usuário a opção de fazer nova
     pesquisa sem que sejam desmarcados os produtos já sele-
     cionados. Assim, será possível que ele faça várias pesquisas,
     para seleção de produtos de características diferentes.
```

Preço promocional

4.a. Se o produto estiver em promoção, a linha de preço deve ser impressa no formato abaixo, considerando que o primeiro valor corresponde ao preço sem desconto e o segundo valor corresponde ao preço com desconto:

De: R$ 999,99 Por: R$ 999,99

Requisitos de Implementação:

Formato de exibição da etiqueta

4.a. A linha de preço (3) deve ser exibida com uma fonte maior (mínimo de 4 pts a mais) que as linhas anteriores – código e detalhes (1 e 2).

4.b. A cor do texto da linha de preço (3) deve ser diferente da cor do texto das linhas anteriores – código e detalhes (1 e 2).

4.c. No caso da linha de preço (3) apresentar o valor normal e o valor promocional, o preço normal deverá ser exibido com a mesma fonte e cor das linhas anteriores - código e detalhes (1 e 2), e o preço promocional deverá ser exibido no tamanho e cor já definido para o mesmo (itens 4.a. e 4.b)

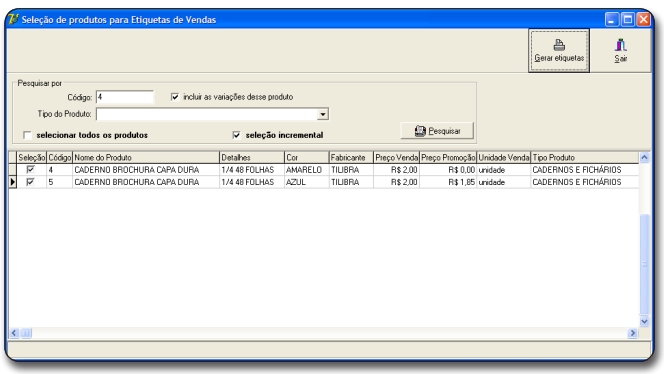


Figura 2. Protótipo (1) do UC Emitir Etiquetas de Vendas – Tela principal de seleção

Figura 3. Protótipo (2) do UC Emitir Etiquetas de Vendas – etiquetas geradas

Na **Listagem 2**, apresentamos o caso de uso Gerar relatório de reposição. Da mesma forma que o caso de uso anterior, vamos relembrar os requisitos pertinentes a esse caso de uso:

• O estoque de cada produto será automaticamente gerado no

momento do cadastramento do produto. O estoque será inicializado com zero, porém deve ser informada a quantidade mínima de estoque, que identificará a necessidade de reposição.

Com base no requisito descrito, entende-se que todo produto possui seu controle de quantidade em estoque e um outro valor que indica o estoque mínimo. Quando a quantidade disponível em estoque alcançar o valor de estoque mínimo, esse produto precisa ser reposto. Para isso, existirá esse relatório de reposição. Da mesma forma como aconteceu com o caso de uso anterior, estão faltando informações, que precisam ser detalhadas pelo analista. Quais campos devem ser exibidos nesse relatório? Qual ordenação o mesmo deve ter? Com base nessas respostas, o documento de requisitos foi alterado, incluindo-se o parágrafo abaixo:

- Para o relatório de reposição, deve-se pesquisar todos os produtos que estejam com a quantidade disponível em estoque igual ou menor que a quantidade mínima. Para cada produto nessa situação é preciso exibir: código, nome, detalhes, cor, fabricante, quantidade mínima em estoque e a quantidade disponível em estoque. O relatório deve vir ordenado por estoque e em ordem decrescente da diferença entre quantidade mínima e quantidade disponível.

Repare que o caso de uso é controlado completamente pelo sistema, com um processamento batch. E pela simplicidade desse processamento, não há cenário alternativo.

A **Figura 4** apresenta o protótipo com a imagem prevista do relatório numa visão macro, e a **Figura 5** apresenta o mesmo relatório num "zoom" mais próximo nas linhas detalhes.

Listagem 2. UC Gerar relatório de reposição

```
Descrição: Este caso de uso tem por objetivo gerar um relatório
com todos os produtos cuja quantidade disponível em estoque
tenha alcançado a quantidade mínima.
Atores: Departamento de Estoque
Pré-condição: existir cadastro prévio de produtos.
Cenário principal:
1. O sistema pesquisa todos os produtos cadastrados cuja
   quantidade disponível em estoque esteja menor ou igual que a
   quantidade mínima de estoque.
2. O sistema prepara um relatório contendo todos os produtos
   encontrados.
   2.1. O relatório deverá conter a seguinte ordenação:
      2.1.1 fabricante
      2.1.2. diferença (quantidade mínima - quantidade disponível em
         estoque), em ordem decrescente
3. O sistema gera o relatório com os dados e formato a seguir:
   3.1. agrupamento por fabricante
   3.2. cada linha de produto:
      3.2.1. código
      3.2.2. nome
      3.2.3. detalhes
      3.2.4. cor
      3.2.5. quantidade mínima em estoque
      3.2.6. quantidade disponível
   3.3. totalização final: quantidade de produtos impressos

   Relatório de Reposição
   ------------------------------------------------------------
   Fabricante:  XXXXXXXXXXX
   ------------------------------------------------------------
   Código Nome          Detalhes   Cor    Qtd Mínima   Qtd Disponível
   9999   XXXXXXXXXXXX  XXXXXXXXX XXXXX    99999          99999
   ------------------------------------------------------------
   Total de produtos: 9999
Cenários alternativos:
   Não se aplica
```

A seguir vemos os requisitos que darão suporte ao caso de uso Registrar Pagamento de Venda, apresentado na **Listagem 3**:

• Após escolher os produtos, com a ajuda ou não de um vendedor, o cliente deve se dirigir a um deles para tirar o pedido de venda. Esse pedido conterá todos os produtos e liberará para o caixa apenas o pagamento. Só existe um caixa na papelaria, responsável apenas pelo recebimento do pagamento.

• O registro da venda será feito pelo vendedor e se dará a partir do código do produto. Se o vendedor não souber o código, poderá fazer a pesquisa pelo nome. Para cada produto é informada a quantidade comprada. Ao final de uma venda, é gerada uma nota com todos os produtos, contendo um número de venda e o total a pagar. Essa nota deve ser paga no caixa. O supervisor poderá autorizar um desconto para a venda total. Isso se dará no momento do registro da venda. São formas de pagamento aceitas: dinheiro, débito e cartão de crédito.

Esse caso de uso tem uma ligação lógica com o caso de uso de Registro de Venda, sendo que nessa situação não há pré-condição. O sistema, num processamento inicial (item 1), verifica todos os pedidos de venda ainda em aberto. O usuário seleciona um pedido e registra o pagamento.

Repare que apesar de termos dois atores, ainda usamos o

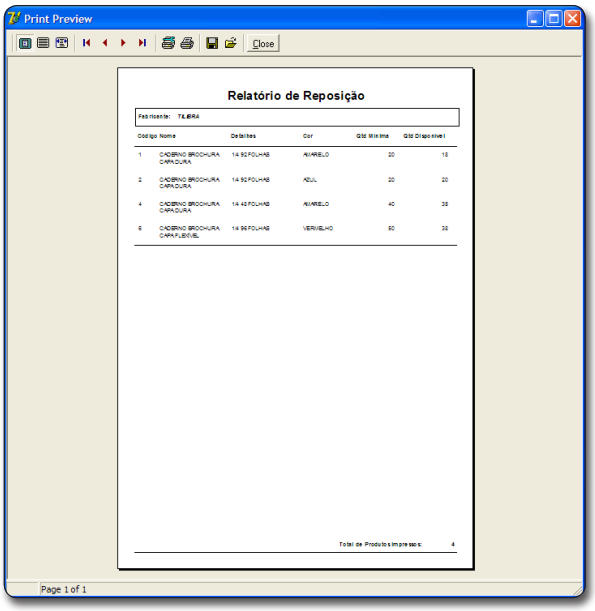


Figura 4. Protótipo do UC Gerar Relatório de Reposição

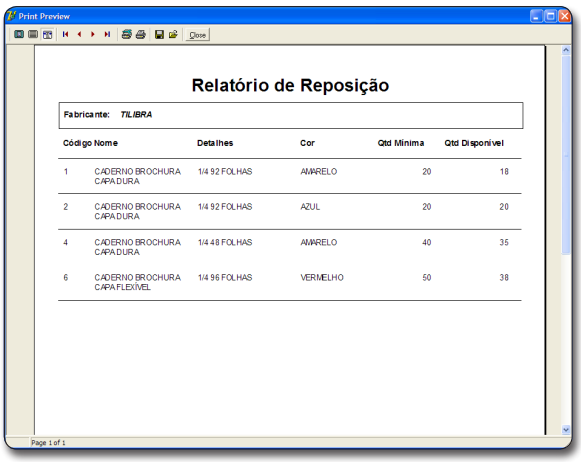


Figura 5. Protótipo do UC Gerar Relatório de Reposição (zoom nas linhas detalhes)

termo "usuário" que abrange ambos os casos, já que um supervisor pode realizar todas as tarefas do Caixa. Somente no momento em que uma rotina é exclusiva do supervisor, seu nome é citado de forma explícita (item 3.1.).

A papelaria não terá um sistema que se comunicará diretamente com os bancos e administradoras de cartões. Mas o caso de uso já está preparado para tal, repassando essa tarefa para um caso de uso de extensão (Realizar comunicação externa) (item 4.2). Nesse caso de uso, quando de sua criação, serão definidos todos os dados que o sistema precisa passar aos sistemas externos (bancos e administradoras de cartões), além da abertura do canal para aguardar a resposta sobre a autorização. Enquanto isso não ocorre, o sistema terá a opção de "autorização manual" que está definida no cenário alternativo. Esta opção também é útil quando houver a comunicação externa, pois problemas podem ocorrer, e a autorização manual é imprescindível para que o pagamento não deixe de ser registrado.

Como a nota fiscal segue as características fiscais de cada cidade, seu detalhamento foi colocado num documento à parte, que conterá todas as regras de cálculo de impostos e apresentação exigidas pelo governo (item 5).

A **Figura 6** apresenta o protótipo (1) desse caso de uso, com a tela inicial, para seleção dos pedidos de venda. O protótipo (2), apresentado na **Figura 7**, já traz o pedido selecionado, com a opção de registro de pagamento em dinheiro. Como variação, o protótipo (3), exibido na **Figura 8**, traz um registro de pagamento com cartão de crédito.

Listagem 3. UC Registrar Pagamento de Venda

```
Descrição: Este caso de uso tem por objetivo finalizar os
pedidos de vendas efetuados pelos vendedores, com o registro do
pagamento.
Atores: Caixa, Supervisor
Pré-condição: existir cadastro prévio de produtos.
Cenário principal:
1. O sistema pesquisa e exibe todos os pedidos de vendas
   cadastrados pelos vendedores, que ainda não tenham sido
   finalizados, ou seja, que não haja registro de pagamento.
   1.1. Para cada venda é exibido:
        - número do pedido de venda
        - nome do vendedor
        - total a pagar da venda
2. O usuário seleciona um pedido de venda.
3. O sistema exibe o número do pedido de venda, o nome do
   vendedor e o total a pagar.
   3.1. O Supevisor pode autorizar um desconto no total,
   informando o valor de desconto.
   3.2. No caso de desconto, o sistema deve calcular e exibir o
   novo valor a pagar.
4. O usuário informa a forma de pagamento, selecionada entre as
   opções: dinheiro, débito e cartão de crédito.
   4.1. Para a forma de pagamento "dinheiro":
        4.1.1. O usuário informa o valor recebido.
        4.1.2. O sistema calcula e exibe o troco (troco = valor
        recebido - valor a pagar).
   4.2. Para as formas de pagamento "débito" e "cartão de
   crédito", o sistema se comunica com um sistema externo e
   aguarda autorização. Extends Caso de Uso Realizar comunicação
   externa.
   4.3. O usuário deve confirmar o pagamento
5. O sistema deve emitir nota fiscal de acordo com o "modelo de
   nota fiscal" constante no documento em anexo.
   5.1. O sistema registrará no estoque de cada produto do
   pedido de venda o débito das quantidades vendidas.
   5.2. O sistema registrará o fechamento do pedido de venda,
   associando o número da nota fiscal correspondente ao referido pedido.
Cenários alternativos:
   Autorização manual
4.a. No caso de não receber autorização do sistema externo, o
     usuário poderá entrar com a informação de autorização
     manual e concluir a venda.
```

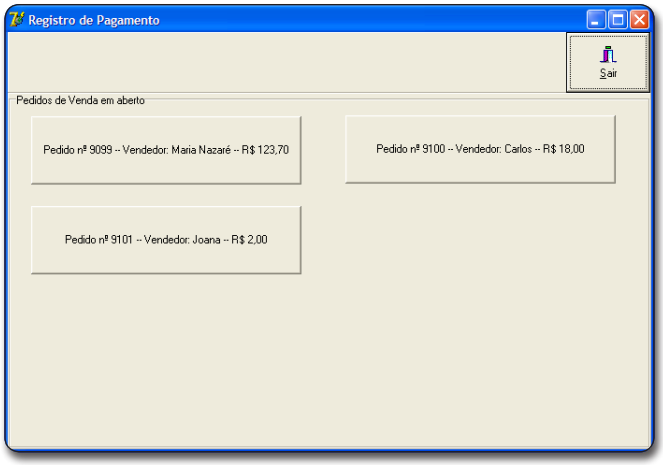


Figura 6. Protótipo (1) do UC Registrar Pagamento de Venda – Seleção dos pedidos em aberto

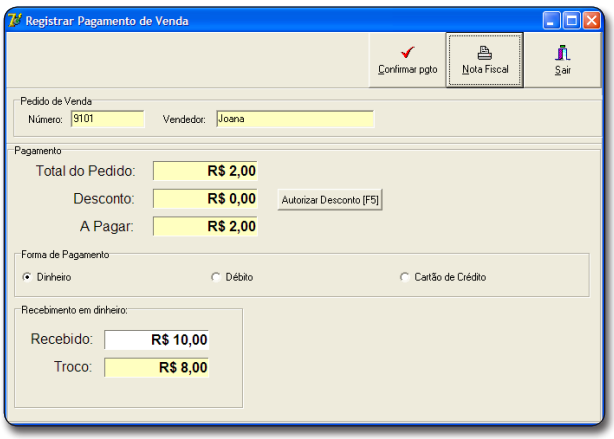


Figura 7. Protótipo (2) do UC Registrar Pagamento de Venda – registro do pagamento em dinheiro

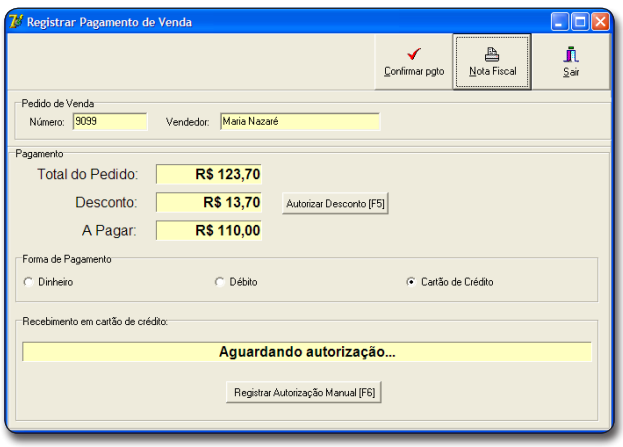


Figura 8. Protótipo (3) do UC Registrar Pagamento de Venda – registro do pagamento em cartão de crédito

Por fim, a **Listagem 4** traz o caso de uso Fechar o caixa. O objetivo deste é totalizar todas as vendas ocorridas, fechando o caixa para novos registros de pagamento. A totalização é importante, para que haja conferência dos pagamentos físicos existentes. Repare que este caso de uso controla um flag de abertura/fechamento, bem como realiza o registro da data e

hora, que será usado para consulta histórica (item 3). Na **Figura 9**, apresentamos o protótipo com a tela de finalização.

Listagem 4. UC Fechar o caixa

```
Descrição: Este caso de uso tem por objetivo encerrar os
registros de pagamento, com a totalização das vendas ocorridas,
para conferência do supervisor.
Atores: Supervisor
Pré-condição: o caixa já ter sido aberto.
Cenário principal:
1. O sistema verifica se não há nenhum pedido de venda pendente.
2. O sistema totaliza e exibe todas as vendas efetuadas com
   pagamento em dinheiro, pagamento em cartão de crédito,
   pagamento em débito, e totalização de todas as vendas.
3. O sistema encerra o caixa, registrando a data e hora do encerramento.
   3.1 O usuário imprime o comprovante de fechamento com os
   seguintes dados:
       - total de vendas em dinheiro
       - total de vendas com cartão de crédito
       - total de vendas em débito
       - total geral das vendas
       - data e hora do encerramento
Cenários alternativos:
   Pedido de venda pendente
1.a. Se houver pedido de venda pendente, o sistema deve exibir a
     lista dos pedidos, com número, nome do vendedor e total a
     pagar, e impedir o prosseguimento do caso de uso.
```

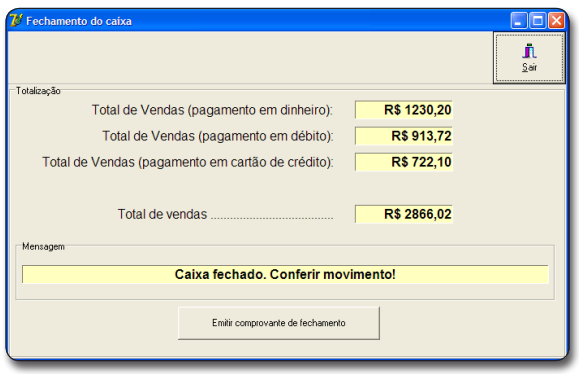


Figura 9. Protótipo do UC Fechar o caixa

Conclusão

Neste artigo apresentamos os cenários completos de sete casos de uso do estudo de caso do sistema gestor de papelaria. Percorremos modelos diferentes que possibilitassem o entendimento de como modelar rotinas de manutenção, entrada de dados, recuperação de informações, processamento batch, utilização por múltiplos atores, chamadas a outros casos de uso, impressão de relatório, impressão de etiqueta e consulta em tela.

Minha experiência trabalhando com casos de uso me permite dizer que essa é uma ferramenta poderosa, que torna maleável não só o canal de contato com o cliente, como possibilita um relacionamento sem ruídos com os desenvolvedores.

Escrever caso de uso é uma tarefa muito pessoal. Não é a toa que a documentação oficial da UML não definiu padrões para essa escrita. Pelo contrário, foi o mercado que instituiu as melhores formas, e apontou as piores práticas. O importante para um analista é usar de bom senso para lapidar o melhor possível o seu produto. Afinal, um caso de uso torna-se a base de um sistema. ●

Especificação de Casos de Uso

Detalhando os Requisitos de Software

Antonio Mendes da Silva Filho
antoniom.silvafilho@gmail.com
Professor e consultor em área de tecnologia da informação e comunicação com mais de 20 anos de experiência profissional, é autor do livros Arquitetura de Software e Programando com XML, ambos pela Editora Campus/Elsevier, tem mais de 30 artigos publicados em eventos nacionais e internacionais, colunista para Ciência e Tecnologia pela Revista Espaço Acadêmico com mais de 80 artigos publicados, tendo feitos palestras em eventos nacionais e exterior. Foi Professor Visitante da University of Texas at Dallas e da University of Ottawa. Formado em Engenharia Elétrica pela Universidade de Pernambuco, com Mestrado em Engenharia Elétrica pela Universidade Federal da Paraíba (Campina Grande), Mestrado em Engenharia da Computação pela University of Waterloo e Doutor em Ciência da Computação pela Univesidade Federal de Pernambuco.

Neste artigo veremos

Apresenta o documento de especificação de casos de uso, fundamental para o processo de desenvolvimento de software, e exemplifica as seções e componentes das seções deste documento.

Qual a finalidade

Informar o leitor sobre quais elementos considerar quando da elaboração de um documento de especificação de casos de uso, apresentando um modelo (template) de documento que pode ser adotado ou customizado.

Quais situações utilizam esses recursos?

Durante o desenvolvimento de um sistema de software no qual há a necessidade de elaborar o documento de especificação de casos de uso, os quais descrevem o conjunto de funcionalidades do sistema a serem implementados pela equipe de projeto.

Desenvolver um sistema de software é uma atividade que requer um conjunto de habilidades com objetivo de entender as necessidades do usuário e/ou cliente, analisá-las, documentá-las num documento de requisitos (discutido na edição anterior), desenvolver o projeto, implementar e testar o software. Perceba que um engenheiro de software é um profissional que possui um conjunto de competências e é comum especializar-se em alguma(s) dela(s). As atividades iniciais são importantes, já que definem quais funcionalidades o sistema deve prover. Portanto, faz-se necessário descrever os requisitos de software com objetivo de facilitar as etapas seguintes (projeto e implementação).

Note que um documento de requisitos de software precisa ser claro, consistente e completo, pois ele servirá de referência aos desenvolvedores, gerente de projeto e engenheiros de software (responsáveis pelos testes e manutenção do sistema). Requisitos bem documentados consideram não apenas consistência, completude e clareza, mas também a evolução dos requisitos e, portanto, características do sistema de

software. Dentro deste contexto, este artigo trata da especificação de casos de uso para detalhamento dos requisitos de software.

Especificação de Requisitos de Software

A especificação de requisitos de um sistema de software descreve as necessidades de usuários do sistema e, mais especificamente, descreve o conjunto de funcionalidades que o sistema deve fornecer. É a partir desta especificação que o projeto de software é desenvolvido para, em seguida, ser implementado. Quaisquer problemas como, por exemplo, de inconsistência ou ambigüidade, resultará em possíveis erros que precisarão ser corrigidos em alguma etapa futura do processo de desenvolvimento.

Adicionalmente ao objetivo acima, uma especificação de requisitos tem ainda outro objetivo, talvez até mais importante: comunicar de maneira efetiva quais funcionalidades o sistema deve prover (ou ainda, o que não irá prover). Perceba que a especificação de requisitos irá apoiar a comunicação entre usuários (clientes), projetistas, programadores, testadores.

O interesse é o de especificar as funcionalidades ou requisitos funcionais de um sistema. Nesse sentido, o sistema irá prover uma determinada funcionalidade quando solicitado ou quando uma condição for alcançada. Dessa forma, o que se tem é uma interação entre a funcionalidade (caracterizada pelo caso de uso) e um ator, que pode ser um usuário (humano), um componente de software ou algum hardware (ou dispositivo). Num exemplo simples do conhecimento do leitor, tem-se a funcionalidade *login* de um sistema Web, a qual serve para autenticar o *login* e senha de usuários e permitir acesso a determinada aplicação. Para essa funcionalidade, ou caso de uso, há uma interação entre usuário e o caso de uso *login*. Para tanto, um conjunto de passos é usado para descrever o fluxo de eventos da interação entre essas duas entidades. A seguir, a especificação de casos de uso é apresentada e exemplificada.

Especificação de Casos de Uso

Caso de uso é uma técnica de especificação que descreve uma seqüência de ações que o sistema deve realizar para produzir uma resposta para um ator. Na realidade, tem-se uma seqüência da interação entre caso de uso e ator. O caso de uso detalha o que um sistema deve fazer, descrevendo como uma determinada funcionalidade é utilizada por um ator.

Cabe destacar que um caso de uso compreende duas partes: o diagrama de caso de uso e o caso de uso propriamente dito. O diagrama de caso de uso é um dos nove diagramas da UML (*Unified Modeling Language*) enquanto que o caso de uso consiste de um *template* (ou modelo), conforme apresentado na seção seguinte, que serve para detalhar a seqüência de passos de execução do caso de uso. Neste artigo, o foco é tratar da especificação de casos de uso, isto é, de apresentar e ilustrar um *template* de especificação e, portanto, o assunto diagrama de casos de uso não é tratado.

Os casos de uso servem para especificar as interações existentes entre o sistema em desenvolvimento e atores (ou entidades externas ao sistema). Os atores compreendem usuários, estímulos externos gerados por dispositivos eletrônicos ou outros sistemas computacionais. Os atores podem ser considerados como entidades externas ao sistema, já que estão fora da fronteira do sistema, e são responsáveis por gerar eventos para iniciar interação com o caso de uso.

Outra recomendação é evitar uso de termos específicos de implementação, procurando manter o nível de detalhamento compreensível ao usuário. É preciso considerar que os casos de uso devem levar em conta o vocabulário do usuário. Além disso, um cenário é uma instancia de um caso de uso e, portanto, um caso de uso pode conter um ou mais cenários, conforme apresentado na seção seguinte.

Casos de uso servem para especificar o comportamento de um sistema de software ou parte dele, isto é, descrevem um conjunto de ações que produzem algum resultado. Note que o objetivo do caso de uso é especificar o *'o que'* e não o *'como'* um determinado sistema realiza um objetivo. Um exemplo de diagrama de caso de uso é mostrado na **Figura 1**.

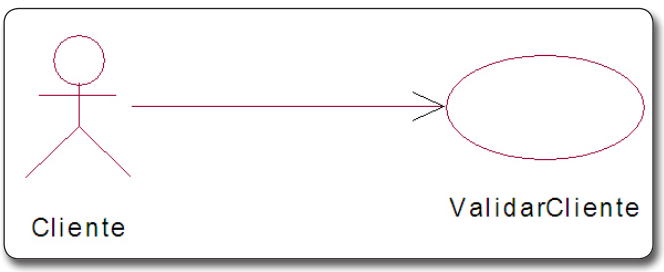


Figura 1. Exemplo de um caso de uso.

A figura contém um ator (Cliente) e um caso de uso (ValidarCliente) com o qual o ator interage. Note que podemos ter interações entre atores e casos de uso, cujo objetivo é alcançar a meta do caso de uso (realizando sua funcionalidade). Essas interações ocorrem através da troca de mensagens onde os atores se comunicam com o(s) caso(s) de uso. O fluxo que descreve as ações (i.e. mensagens) trocadas entre atores e caso de uso é denominado de fluxo principal e compõe o núcleo do documento da especificação de casos de uso.

Esse documento contém um conjunto de informações que são de interesse do cliente, gerente de projeto, gerente de negócios a programadores, analistas e engenheiros de testes. Nesse sentido, o engenheiro de software, ao elaborar esse documento, busca um compromisso de comunicar bem o conjunto de passos de cada caso uso do sistema, bem como prover detalhes com suficiente clareza e consistência para público alvo, ou seja, os programadores e engenheiros de testes (responsáveis pela implementação do sistema e elaboração e execução de plano de testes, respectivamente).

A **Tabela 1** apresenta uma relação dos itens consideradas imprescindíveis em um documento de especificação de casos de uso. A relação de itens destacados na **Tabela 1** não pressupõe a intenção de ser completo, mas de apontar os itens considerados como mais importantes num

Itens de uma Especificação de Casos de Uso	Conteúdo
1. Introdução	Esta seção contém uma descrição dos objetivos do documento, o público ao qual ele se destina e, em linhas gerais, o propósito projeto a ser desenvolvido. Essa seção, adicionalmente, pode apresentar uma visão geral do sistema a ser desenvolvido, informando as demais seções do documento.
2. (Descrição de) Atores	Esta seção descreve, de maneira sumarizada, o conjunto de atores que interagem com os casos de uso do sistema a ser desenvolvido. A descrição de cada ator, adicionalmente, caracteriza seu respectivo papel.
3. (Descrição de) Casos de Uso	Apresenta-se uma descrição de todos os casos de uso que fazem parte do sistema a ser desenvolvido. Também, é apresentado o conjunto de informações que compõem cada caso de uso. As informações a serem utilizadas na especificação do caso de uso são obtidas do documento de requisitos. O engenheiro de software deve organizar o conjunto de casos de uso do sistema de modo a torná-los mais compreensíveis.
4. Referências (Documentos Complementares)	Exemplos desses documentos complementares compreendem o documento de requisitos, atas de reuniões nas quais ocorreram levantamento e validação de requisitos e, em aplicações específicas, um glossário.
5. Apêndice	Trata-se de uma seção que pode conter informações de apoio para os leitores do documento como, por exemplo, diagramas de casos de uso.

Tabela 1. Relação de itens de um documento de requisitos.

documento da especificação de casos de uso para uma empresa.

O conteúdo exato das seções que compõem um documento de especificação de casos de uso pode variar de empresa para empresa. Todavia, os itens relacionados na **Tabela 1**, geralmente, fazem parte de qualquer especificação de caso de uso. As subseções, destacados nos quadros a seguir, enumerados de 1 a 5, ilustram o conteúdo que compõe um documento de especificação de casos de uso.

Note que o **Quadro 1** descreve o propósito do documento e apresenta uma visão geral do documento do sistema a ser desenvolvido, informando as seções e conteúdo que compõem o documento. Isto visa prover os leitores (projetista, programador ou engenheiro de teste) com as informações corretas das funcionalidades do sistema.

A seção seguinte apresenta a segunda parte do documento de especificação de casos de uso, a qual contém relação e descrição dos atores. Atores podem ser usuários (humanos), outros (sub)sistemas (de hardware ou software) ou ainda uma entidade abstrata que causa algum tipo de estímulo como, por exemplo, um determinado horário ou data que é alcançada, e gera um evento (que interage com um caso de uso do sistema).

Perceba que um caso de uso jamais irá iniciar sozinho qualquer ação. Eles necessitam de algum ator que atua como *'inicializador'* das interações. É importante ainda entender que não apenas um ator inicializa um caso de uso como também pode ser influenciado pela resposta do caso de uso (como ocorre com o *feedback* dado pelo sistema a um usuário). Um exemplo da descrição de atores é ilustrado no **Quadro 2**.

1. Introdução

Este documento especifica os casos de uso de um sistema que provê notícias e conteúdo online, denominado de Sistema Exemplo, a ser desenvolvido para a Empresa XYZ. Seu propósito é prover notícias sobre os mais variados conteúdos, permitindo acesso integral apenas aos usuários leitores cadastrados no sistema. Este documento apresenta descrição dos fluxos de eventos, entradas e saídas de cada caso de uso a ser implementado.

(Note que o propósito desse sistema, usado aqui apenas com fins ilustrativos, é similar ao de um sistema como o de portais de jornais e revistas e outros provedores de conteúdo que permitem o acesso ao conteúdo apenas a clientes devidamente cadastrados no sistema).

Visão geral do documento

Esta introdução fornece as informações necessárias para utilizar este documento, explicitando os objetivos do sistema a ser desenvolvido. As seções abaixo complementam este documento.

- **Seção 2** – Descrição de Atores: apresenta um quadro dos atores que interagem com o sistema, com breve descrição sobre cada um.
- **Seção 3 –** Descrição de Casos de Uso: apresenta a especificação do conjunto de casos de uso do sistema.
- **Seção 4** – Referências: referencia documentos complementares citadas no documento ou necessárias para o entendimento do mesmo.
- **Seção 5** – Apêndice: compreende um conjunto de informações complementares para auxiliar no entendimento do projeto.

Quadro 1. Exemplo da Seção 1 do Documento de Especificação de Casos de Uso

2. Descrição de Atores

Esta seção apresenta a descrição dos atores pertinentes aos casos de uso especificados neste documento, os quais estão relacionados no quadro abaixo.

Ator	Descrição
Usuário Internet	Qualquer usuário que acesse o site do sistema que provê notícias e conteúdo online e venha consultar algum conteúdo gratuito. Este usuário não precisa estar cadastrado no sistema para ter acesso ao conteúdo gratuito.
Usuário Interno	São os funcionários da empresa XYZ.
Usuário Externo	São os usuários cadastrados no sistema de notícias, os quais têm acesso a todo conteúdo via Internet.
Usuário Principal	Usuário Interno que tem a responsabilidade de controlar as liberações/bloqueios de acesso ao conteúdo do sistema.
Usuário Autorizado	Usuário Externo que tenha sido autorizado pelo Usuário Principal para ter acesso às informações e conteúdo do sistema.
Artigos	Componentes que implementam operações de acesso e atualização de artigos do sistema de informações (cujo acesso é autorizado apenas aos usuários cadastrados).

Quadro 2. Exemplo da Seção 2 do Documento de Especificação de Casos de Uso

O **Quadro 2** apresenta a descrição de um conjunto de atores de um sistema Exemplo. Perceba que o objetivo não foi de ser completo, mas o de ilustrar como a seção de atores de uma especificação de casos de uso poderia ser elaborada. Por exemplo, perceba que há cinco atores que são usuários (humanos) que têm diferentes denominações dependendo de suas características. Além disso, há um outro ator (Artigos) que é na realidade um componente de software que implementa um conjunto de funcionalidades. Note ainda que as informações apresentadas nesta seção têm a intenção de comunicar às partes envolvidas e interessadas (isto é, projetista, programadores e engenheiro de teste) o conjunto de atores pertinentes ao sistema a ser desenvolvido.

A seção seguinte apresenta a terceira parte do documento de especificação de casos de uso que compreende a descrição dos casos de uso seguindo um *template* ou modelo que contém um conjunto de itens como descrito abaixo:

- Nome do caso de uso
- Autor
- Data
- Pré-condição
- Descrição ou sumário (do caso de uso)
- Fluxo principal (ou básico) de eventos
- Subfluxo de eventos
- Fluxo secundário (alternativo e de exceção)
- Pós-condição
- Ponto de extensão

3. Descrição de Casos de Uso

Um caso de uso compreende uma seqüência de ações que o sistema realiza produzindo algum resultado para um ator.

Os casos de uso nesse documento são identificados pelo termo "UC" (que denomina Use Case) seguido de um número seqüencial, como [UC001]. Cabe destacar que os identificadores dos casos de uso jamais devem ser modificados ou reaproveitados, para não invalidar referências externas feitas a eles.

O fluxo principal descreve o comportamento "normal" do caso de uso, isto é, os eventos básicos necessários para que o caso de uso seja executado normalmente. Eventualmente, pode-se ter sub-fluxos (SB) que descrevem um fluxo opcional de eventos. Vale ressaltar que um sub-fluxo não pressupõe sua execução já que ele pode ou não ser executado. Um sub-fluxo constitui um fluxo opcional e poderia ser considerado como uma extensão num diagrama de casos de uso. Adicionalmente, um fluxo secundário (FS) que compreende fluxo alternativo e de exceções, isto é, eventos não cobertos no fluxo principal do caso de uso. Além disso, se um passo do fluxo de eventos principal puder levar à execução de um sub-fluxo ou de um fluxo secundário, o identificador do sub-fluxo ou do fluxo secundário deverá ser referenciado, colocando-se ao lado deste passo entre parênteses (como ilustrado nos casos de uso descritos abaixo).
O conteúdo abaixo apresentado descreve um conjunto de requisitos do Sistema Exemplo que provê conteúdo para usuários cadastrados.

[UC001] Solicitar cadastro no sistema
Pré-condições: O usuário deve estar na tela de login.
Pós-condições: Se o caso de uso for realizado com sucesso, um termo de responsabilidade de acesso ao site do sistema de conteúdos será gerado.
Ator: Usuário autorizado e usuário principal

Fluxo de eventos principal
1. O sistema apresenta uma tela contendo os seguintes campos:
    a. Nome *
    b. Endereço *
    c. CPF *
    d. Correio eletrônico *
    e. Telefone *
    f. Indicador se o usuário deverá ser usuário principal *

    Os campos assinalados com * são obrigatórios

2. O usuário preenche os campos ([FS001] Campos de preenchimento obrigatório, [FS002] CPF inválido, [FS003] Formato inválido de correio eletrônico, [FS004] Telefone inválido, [FS005] Usuário não pode ser o principal).
3. O sistema executa a validação da inclusão de usuário, passando os dados (nome, endereço, CPF, correio eletrônico, telefone e tipo de usuário) que devem ser validados. ([FS001] Campos de preenchimento obrigatório; [FS002] Campo de CPF inválido; [FS003] Formato de correio eletrônico inválido; [FS004] Telefone não é numérico; [FS005] Usuário não pode ser o principal).
4. O sistema verifica se o usuário deve ser um usuário principal (SB001 Solicitar usuário principal).
5. O sistema apresenta o termo de responsabilidade de acesso ao site.
6. O sistema retorna para a tela de login.

Sub-fluxos
[SB001] Solicitar usuário principal
1. O sistema verifica autorização para o usuário principal, passando CPF do usuário ([FS005] Usuário não pode ser o principal).
2. O sistema apresenta o termo de responsabilidade de acesso ao site do sistema de conteúdo e de usuário principal.
3. O sistema volta ao passo 5 do fluxo principal

Fluxos secundários
[FS001] Campos de preenchimento obrigatório
1. Se o usuário não preencher algum dos campos obrigatórios da tela, o sistema exibe mensagem solicitando seu preenchimento.

[FS002] CPF inválido
1. Se o dígito verificador do CPF não estiver correto, o sistema exibe mensagem informando que o CPF é inválido.

[FS003] Formato do correio eletrônico inválido
1. Se o correio eletrônico não tiver apenas um caractere @ e não tiver pelo menos um caractere antes e pelo menos um depois do @, então o sistema exibirá uma mensagem de erro: "Formato do correio eletrônico é inválido".

[FS004] Telefone não é numérico
1. Se os dados do telefone não forem numéricos, o sistema exibe a mensagem: "Telefone deve ser numérico".

[FS005] Usuário não pode ser o principal
1. O sistema checa se a resposta obtida na verificação foi "0XY" (isto é, um código de número qualquer 0XY que, para fins de exemplo, pode significar que o "usuário não pode ser o principal") e exibe a seguinte mensagem "O CPF do usuário informado não pode ser o usuário principal da empresa informada".

[UC002] Alterar senha
Pré-condições: O usuário deve estar na tela de login.
Pré-condições: Usuário logado.
Pós-condições: Se o caso de uso for realizado com sucesso, o usuário terá a sua senha alterada.
Atores: Usuário principal, usuário autorizado e usuário interno.

Fluxo de eventos principal
1. O sistema apresenta uma tela com a identificação do usuário e solicita que o mesmo informe os seguintes campos:
    a. Senha de acesso atual *
    b. Nova senha de acesso *
    c. Confirmação de nova senha de acesso *
    d. Lembrete de senha

Os campos assinalados com * são obrigatórios. A senha digitada é exibida para o ator (usuário) em forma de asteriscos.
2. O usuário informa os dados solicitados e confirma a efetivação da alteração da senha. ([FS001] Campos de preenchimento obrigatório; [FS002] Confirmação de senha incorreta).
3. O sistema realiza a alteração de senha, passando como parâmetro a nova senha de acesso e o lembrete de senha ([FS003] Senha inválida; [FS004] Tamanho de senha inválido; [FS005] Senha igual a uma anterior; [FS006] Timeout – Tempo mínimo para alteração de senha não alcançado; [FS007] Usuário excluído do sistema).

Fluxos secundários (alternativos e de exceção)

[FS001] Campos de preenchimento obrigatório
1. Se o usuário não preencher qualquer dos campos obrigatórios da tela do site, então o sistema exibirá uma mensagem solicitando seu preenchimento.

[FS002] Confirmação de senha incorreta
1. O sistema verifica se a nova senha digitada e a confirmação de nova senha estão iguais. Se as senhas não estiverem iguais, então o sistema exibirá a seguinte mensagem: "A confirmação da nova senha está incorreta".

[FS003] Senha inválida
1. O sistema verifica se a senha informada está de acordo com os parâmetros definidos na política de segurança do sistema e exibirá a mensagem de "Senha inválida" caso não esteja em conformidade com esses parâmetros.

[FS004] Tamanho de senha inválido
1. O sistema verifica se o tamanho da senha informada está em conformidade com os parâmetros definidos na política de segurança do sistema e exibirá a mensagem de "Tamanho de senha inválido", caso não esteja em conformidade.

[FS005] Senha igual a uma anterior
1. O sistema verifica se a senha informada está igual a alguma senha anterior já utilizada pelo usuário e exibirá a mensagem de "Esta senha já foi utilizada anteriormente", caso isso seja verificado.

[FS006] Timeout - Tempo mínimo para alteração de senha não alcançado
1. O sistema verifica se o tempo mínimo (timeout) para alteração de senha não foi alcançado e exibirá a mensagem de "Timeout - Tempo mínimo para alteração de senha não alcançado", caso isso ocorra.

[FS007] Usuário excluído do sistema
1. O sistema checa se a resposta obtida na verificação foi "0XY" (isto é, um código de número qualquer 0XY que, para fins de exemplo, pode significar que o "usuário excluído do sistema") e exibirá a mensagem de "Usuário inativo e já excluído do sistema".

[UC003] Efetuar login
Pré-condições: O usuário deve ser cadastrado no sistema.
Pós-condições: Se o caso de uso for realizado com sucesso, o usuário será logado no sistema e terá acesso às funcionalidades do mesmo.
Atores: Usuário principal, usuário autorizado e usuário interno.

Fluxo de eventos principal
1. O sistema disponibiliza as seguintes opções ao usuário:
    a. Solicitar cadastro no sistema
    b. Iniciar sistema de atendimento
    c. Solicitar lembrete de senha
1. De acordo com a opção selecionada pelo usuário, o sistema executa um dos seguintes fluxos:
    a. [UC001] Solicitar cadastro no sistema
    b. [SB001] Iniciar sistema de atendimento
    c. [UC004] Solicitar lembrete de senha

Sub-fluxos
[SB001] Iniciar sistema de atendimento
1. O sistema exibe uma tela solicitando que o usuário preencha os seguintes campos:
    a. Identificação do usuário*
    b. Senha*
    Os campos assinalados com * são obrigatórios.
2. O usuário informa os dados solicitados e confirma o login. ([FS001] Campos de preenchimento obrigatório).
3. O sistema realiza o login do usuário. ([FS002] Usuário inválido, [FS003] Senha inválida, [FS004] Usuário não cadastrado, [FS005] Usuário bloqueado, [FS006] Usuário excluído)
4. Verifica se a condição da senha "descartável" (isto é, uma senha gerada automaticamente pelo sistema a qual o usuário deve alterar dentro de determinado prazo) do usuário ([SB002] Senha descartável expirada).
5. O sistema verifica se a senha de acesso do usuário está expirada.
6. Caso a senha de acesso do usuário tenha expirado, o sistema:
    a. Registra o log de acesso do usuário, passando os seguintes parâmetros:
        i. A identificação do usuário informada,
        ii. O IP da máquina cliente,
        iii. O evento 5 (acesso com senha expirada).
    b. Executa o caso de uso [UC003] Alterar senha.
7. O sistema verifica o conjunto de funcionalidades as quais o usuário tem acesso.
8. O sistema exibe uma tela inicial e conteúdo para o usuário e destaque de últimas notícias.

[SB002] Senha expirada
1. O sistema alterada a condição de senha descartável do usuário ([FS007] Usuário não está com senha descartável).

Fluxos secundários (alternativos e de exceção)
[FS001] Campos de preenchimento obrigatório
1. Se o usuário não preencher qualquer dos campos obrigatórios exigidos pelo sistema, então o sistema exibirá uma mensagem solicitando seu devido preenchimento.

[FS002] Usuário inválido
1. O sistema verifica se a nova senha digitada e a confirmação de nova senha são iguais. Se as senhas não estiverem iguais, então o sistema exibirá a seguinte mensagem: "A confirmação da nova senha está incorreta".

[FS003] Senha inválida

1. O sistema verifica se a senha informada está de acordo com os parâmetros definidos na política de segurança do sistema e exibirá a mensagem de "Senha inválida" caso não esteja em conformidade com esses parâmetros.

[FS004] Usuário não cadastrado

1. O sistema checa se a resposta obtida na verificação foi "0XY" (isto é, um código de número qualquer 0XY que, para fins de exemplo, pode significar que o "usuário não cadastrado") e exibirá a mensagem de "Usuário não cadastrado no sistema. Entre em contato com a Central de Atendimento 0-800-XYZW", caso nenhum dado seja encontrado no sistema.

[FS005] Usuário bloqueado

1. Caso o usuário esteja bloqueado, o sistema deve:
   a. Registrar o log de acesso do usuário, passando os seguintes parâmetros:
      i. A identificação do usuário
      ii. O IP da máquina cliente
      iii. O evento 3 (tentativa de acesso com usuário bloqueado).
   b. Enviar e-mail para a área de segurança da Empresa XYZ, relatando a ocorrência de tentativa de acesso por usuário bloqueado.
   c. Apresentar a mensagem: "Usuário bloqueado. Entre em contato com a Central de Atendimento 0-800-XYZW".

[FS006] Usuário excluído

1. O sistema checa se a resposta obtida na verificação foi "0XY" (isto é, um código de número qualquer 0XY que, para fins de exemplo, pode significar que o "usuário excluído") e exibirá a mensagem "Usuário excluído. Entre em contato com a Central de Atendimento 0-800-XYZW".

[FS007] Usuário não está com senha descartável

1. O sistema checa se a resposta obtida na verificação foi "0XY" (isto é, um código de número qualquer 0XY que, para fins de exemplo, pode significar que o "usuário não está com senha descartável") e exibirá a mensagem "Usuário não possui senha descartável. Entre em contato com a Central de Atendimento 0-800-XYZW".

Quadro 3. Exemplo da Seção 3 do Documento de Especificação de Casos de Uso.

Uma descrição de cada um desses itens é apresentada no Apêndice, conforme ilustrado no **Quadro 5**. Note que essa relação de itens não tem o objetivo de ser completa.

Um conjunto de caso de uso é mostrado no **Quadro 3**, objetivando ilustrar uma descrição de casos de uso que poderia ser elaborada.

O **Quadro 3** apresenta a descrição de um conjunto de casos de uso de um sistema Exemplo. Vale ressaltar que apenas alguns requisitos funcionais são apresentados para ilustrar como essa seção do documento de especificação de casos de uso poderia ser elaborada. Veja também que possíveis fluxos secundários, os quais podem existir numa aplicação, foram ilustrados.

O **Quadro 4** apresenta a quarta parte do documento de especificação de casos de uso que lista um conjunto de documentos complementares, os quais servem de subsídio ao conteúdo neste documento.

Documentação Complementar

Esta seção apresenta a documentação de apoio, referenciando um conjunto de outros documentos que complementam e suportam as informações contidas no documento de requisitos.

1. **Documento de Requisitos do Sistema Exemplo, Versão 00.01 22/01/2009.**
2. **Ata de Reunião – Levantamento de Requisitos do Módulo de Cadastro do Sistema Exemplo, 12/01/2009.**
3. **Ata de Reunião – Levantamento de Requisitos do Módulo Controle de Acesso e Autenticação do Sistema Exemplo, 13/01/2009.**
4. **Ata de Reunião – Levantamento de Requisitos do Módulo de Financeiro do Sistema Exemplo, 14/01/2009.**
5. **Ata de Reunião – Validação de Requisitos do Sistema Exemplo, 15/01/2009.**
6. **Plano de Projeto do Sistema Exemplo.**

Quadro 4. Exemplo da Seção 4 do Documento de Especificação de Casos de Uso

Finalmente, a quinta parte do documento de especificação de caso de uso é opcional, podendo ou não fazer parte deste documento. Ela pode conter, por exemplo, um glossário de termos usados no documento e também um quadro informando o conteúdo de cada parte do *template* da descrição de caso de uso, como ilustrado no **Quadro 5**.

1. Apêndice

Esta seção apresenta conteúdo do Modelo (Template) de Caso de Uso utilizado no documento de especificação de casos de uso.

1. **Nome do caso de uso – trata-se de uma identificação única do caso de uso para ser utilizada ou referenciada em todo o documento.**
2. **Descrição ou sumário (do caso de uso) – Breve descrição do conteúdo e propósito do caso de uso.**
3. **Fluxo principal (ou básico) de eventos – Descrição dos passos que atores e sistema realizam durante execução do caso de uso. Trata-se do fluxo normal de eventos que ocorrem na interação entre ator e sistema para realização de um caso de uso.**
4. **Fluxo secundário (alternativo e de exceção) – Compreendem caminhos alternativos que podem ser tomados durante a realização de um caso de uso.**
5. **Pré-condição – Compreende uma lista de condições necessárias para que um caso de uso aconteça.**
6. **Pós-condição – Trata-se uma lista de condições que são consideradas como atendidas quando um determinado caso de uso é encerrado.**
7. **Ponto de extensão – Consiste de uma seqüência opcional de eventos que é incluída num caso de uso, podendo ocorrer ou não (já que ela é opcional).**
8. **Autor – Identificação de quem elaborou o documento de especificação de casos de uso.**
9. **Data – Refere-se à data na qual o documento foi produzido.**

Quadro 5. Exemplo da Seção 5 do Documento de Especificação de Casos de Uso

Comentários Finais

Requisitos compreendem a essência de um sistema de software já que eles definem as funcionalidades que o sistema deve fornecer e sob quais condições o sistema deve operar. Portanto, é de suma importância documentar de maneira adequada o conjunto de requisitos para que as etapas seguintes do processo de desenvolvimento de software aconteçam de modo satisfatório.

Assim, com o documento de requisitos em mãos, um engenheiro de software deve detalhar o conjunto de requisitos e, para tanto, ele faz uso de um modelo de documento de especificação de casos de uso para elaborar esse documento. A necessidade de trabalhar com um modelo (ou *template*) é fundamental para comunicar bem as funcionalidades do sistema a ser desenvolvido para, por exemplo, o arquiteto de software, gerente de projeto, programadores e engenheiros de testes, os quais irão consultar as informações contidas nesse documento.

Dê seu feedback sobre esta edição!

A Engenharia de Software Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!
Dê seu voto sobre este artigo, através do link:

www.devmedia.com.br/esmag/feedback

Este artigo apresentou e exemplificou um documento de especificação de casos de uso e destacou sua importância para o desenvolvimento de um sistema de software. ●

Links

A Guide to the Software Engineering Body of Knowledge
http://www.swebok.org/

Use Case – An Introduction
http://www.parlezuml.com/tutorials/usecases/usecases_intro.pdf

Understanding Use Case Modeling
http://www.methodsandtools.com/archive/archive.php?id=24

Driving Development with Use Cases
http://www.parlezuml.com/tutorials/usecases/usecases.pdf

Develop Use Case Document Task
http://www.cragsystems.co.uk/development_process/develop_use_case_document.htm

Requirements Engineering – A Roadmap
http://www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf

Estimativas de Software - Fundamentos, Técnicas e Modelos

Como usar de forma consistente PF, COCOMOII, Simulação de Monte Carlo e seu bom senso em estimativas de software

Carlos Eduardo Vazquez

Sócio-fundador da FATTO Consultoria e Sistemas, um dos autores do livro "Análise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de Software", livro com mais de 7.000 exemplares vendidos e atualmente em sua 8ª edição. Pioneiro na aplicação de métricas de software no Brasil possui 20 anos de experiência em TI, notoriamente na aplicação das disciplinas do desenvolvimento e sustentação de sistemas corporativos. Graduado em Processamento de Dados pela PUC-RJ em 1990, já passou com sucesso por quatro vezes pelo processo de certificação de especialista em pontos de função pelo IFPUG – International Function Point Users Group, tendo sido um dos primeiros brasileiros a conquistar essa certificação em 1996. Desde 1993, vem formando profissionais na aplicação da Análise de Pontos de Função, tendo sido professor da UFES, atuado como consultor de grandes projetos de tecnologia em empresas do setor financeiro, bancário e de telecomunicações.

Neste artigo veremos

Estimar é uma atividade cotidiana, sistematicamente evitada por aqueles responsáveis pela sua execução. Na busca por superar isso, uma série de técnicas e ferramentas surge no cenário do desenvolvimento e manutenção de sistemas. Muitas vezes, no desespero por uma solução imediata, elas são adotadas independentemente de sua adequação ao cenário específico em que serão introduzidas, ou mesmo apenas com um conhecimento superficial quanto ao seu funcionamento. Ferramentas como o COCOMOII, Simulação de Monte Carlo e Pontos de Função não substituem a analista responsável pela estimativa, que enfrenta a confusão entre o que seja uma estimativa técnica, um compromisso pessoal ou uma meta corporativa.

Qual a finalidade

Nosso objetivo é diferenciar entre esses diferentes atos e como se portar diante de cada um deles; destacar que simples cuidados podem ajudar a produzir estimativas de muito mais qualidade; apresentar como funciona uma série de ferramentas isoladamente e como integrá-las no estabelecimento de um ambiente propício à melhoria contínua da qualidade das estimativas.

Quais situações utilizam esses recursos?

Nas diferentes situações em que um analista deve se relacionar com seus clientes no sentido de fornecer a sua expectativa para prazo, custo, esforço ou escopo no desenvolvimento e manutenção de software. Visa ajudar a esse analista a identificar os diferentes tipos de solicitação e evitar que ele caia em armadilhas que o leve a assumir compromissos inexeqüíveis. Adicionalmente, é útil também àquele profissional que trabalha na definição de processos de desenvolvimento e seleção de métodos e ferramentas para fins de melhorar o processo de estimativa de sua organização.

Dificuldades ao Estimar

Estimar é um ato cotidiano na vida de todos e isso não é diferente no desenvolvimento e manutenção de sistemas. Apesar disso, uma série de dificuldades de diferentes naturezas faz com que muitos profissionais evitem o seu exercício, ou continuamente adiem a comunicação dos seus resultados. Para ilustrar cinco dessas dificuldades, considere um seguinte pedido por uma estimativa: Qual a duração do deslocamento entre o Rio de Janeiro e Niterói?

Ambigüidade, volatilidade ou falta de clareza

A primeira dificuldade em atender esse pedido é que **(1) os objetivos da estimativa, seu escopo, seus requisitos, não estão claros ou completos**. Por exemplo, qual o meio de transporte para esse deslocamento? Quando se diz Rio de Janeiro, deve-se entender o centro da cidade ou a Barra da Guaratiba (bairro situado a cerca de 60 quilômetros de distância do centro do Rio de Janeiro)? Analogamente, Niterói se refere ao centro da cidade ou a algum outro ponto? O trajeto será feito utilizando um carro ou alguma outra combinação de transportes públicos e privados? Em qual horário?

Veja que essas questões não demandam um grande volume de análise, mas ainda assim muitas vezes elas são desconsideradas, o mesmo acontecendo na prática do desenvolvimento e manutenção de sistemas.

Garantir medições adequadas e referências válidas

A ambigüidade, volatilidade ou falta de clareza do objeto da estimativa, assim como um domínio de problema não muito bem compreendido, não podem ser empecilhos na realização de uma estimativa, afinal estimar também é relativo a predizer face à incerteza.

O conhecimento incompleto não deve ser uma barreira, desde que existam referências nas quais o analista possa se basear. No processo de estimativa, o projeto ou demanda é fracionado em subconjuntos que, individualmente, podem ser designados como uma unidade para execução. Daí surgem outras duas dificuldades: **(2) Garantir que os pacotes de trabalho tenham sido adequadamente medidos**; e **(3) produzir estimativas que estejam consistentes com realizações passadas em outros projetos**.

A construção dessas referências requer dados históricos ou o levantamento dos mesmos – é importante identificar quando não há vontade política para isso, mesmo quando há subsídios técnicos para tal, afinal esse tipo de iniciativa transfere poder de indivíduos para a corporação. Para que esses dados tenham valor efetivo para o processo de estimativa, é necessário que haja um tratamento estatístico dos mesmos, que dificilmente é alcançado na esfera do indivíduo, podendo ser empreendido como uma iniciativa organizacional. Isso porque é tipicamente aí que: (a) Processos são estruturados para estimar e descrever o tamanho do produto de software; (b) as estimativas de custo, esforço, prazo e escopo são relacionados aos valores realizados; (c) os modelos de estimativa utilizados são calibrados às condições locais; e (d) os critérios para normalizar as diferenças entre os projetos e produtos são estabelecidos de tal forma que uma simples extrapolação, não normalizada, de taxa de entrega (Homem-Hora / Ponto de Função) não seja a base para a estimativa.

Diferenciar estimativa, meta e compromisso

Entre aqueles que pedem e fornecem estimativas, existe muita confusão entre o que seja: a) fornecer uma estimativa, um ato técnico que pondera os riscos de escopo e produtividade; b) estipular uma meta para o atendimento de uma demanda ou projeto por uma equipe, um ato gerencial ou político; c) assumir um compromisso, uma decisão pessoal.

Uma estimativa é uma avaliação do provável resultado quantitativo de uma variável de interesse, é a representação de uma chance, um número com uma possibilidade de ser realizado, enquanto meta é um objeto a que se dirige algum intento, e compromisso é uma obrigação tácita ou explícita que pode envolver outras pessoas ou ser auto-imposta.

Outra dificuldade ao estimar é que **(4) nem sempre é fácil distinguir entre esses diferentes atos e a importância dessa distinção está na forma como eles são julgados e os fins para os quais os seus produtos são usados**. Uma estimativa não é um orçamento mal realizado, é uma etapa preliminar. Por exemplo, eliminadas as ambigüidades, nivelado o entendimento necessário para estimar a duração do deslocamento entre o Rio de Janeiro e Niterói, identificadas referências válidas, compatíveis com esse entendimento, ainda assim é um absurdo oferecer uma estimativa de uma hora e quinze minutos, pontual (single point estimate) sem o destaque que se trata de uma chance.

Estimativas diretas versus estimativas paramétricas

As estimativas pontuais são típicas, ainda que não exclusivas das estimativas diretas. Uma estimativa direta é aquela cuja grandeza de interesse (esforço, prazo, custo ou escopo) tem o seu valor estimado de forma direta, sem a utilização de algum outro parâmetro de referência, como no exemplo utilizado anteriormente de 01:15. Um contra-exemplo de uma estimativa direta é como quando se verifica que a distância entre o Rio de Janeiro e Niterói é de 25 quilômetros, considerando uma velocidade média de 25 quilômetros por hora fazendo o percurso todo de carro, e por volta das 15:00, a estimativa seria de 1:00 hora. Esse tipo de estimativa, em que algum outro parâmetro de referência é utilizado para derivar o valor para a grandeza de interesse e é denominada estimativa paramétrica. As estimativas paramétricas são resultados de modelos de estimativas, desde os mais simples como esse exposto, que também produziu uma estimativa pontual, até aqueles que incluem componentes de incerteza.

O aspecto humano é outro fator que deve ser considerado ao receber e elaborar estimativas diretas. Estimar envolve a auto-estima e o reconhecimento profissional. Existem aqueles profissionais que são "orientados ao sucesso"; a estimativa direta produzida por pessoas com esse perfil tende a ser conservadora enquanto aquela produzida por profissionais "viciados em adrenalina" tendem a ser agressivas.

Modelos determinísticos

Os modelos de estimativa determinísticos não incluem qualquer forma de aleatoriedade ou probabilidade em sua caracterização. Independentemente de sua complexidade, as estimativas são determinadas assim que as suas entradas são definidas. Ao oferecer essas estimativas de 01:15 ou 01:00 para a duração do deslocamento entre o Rio de Janeiro e Niterói, aquele que a recebe pode muito facilmente entender que haja certeza dela estar correta.

A estimativa é obtida de forma direta no primeiro caso (01:15) e a partir de um parâmetro, como a média das velocidades, no segundo caso (01:00). Essa média, por sua vez, é obtida a partir de diferentes viagens passadas onde individualmente verificou-se uma determinada velocidade média no percurso. A **Fórmula 1** descreve como ela foi obtida.

$$\overline{Velocidade} = \frac{\sum_{i=1}^{n} Velocidade_i}{n}$$

Fórmula 1. Cálculo da média das velocidades em n diferentes viagens escolhidas aleatoriamente entre aquelas com as condições similares à que se deseja estimar.

A média foi a medida de tendência de centro utilizada na estimativa de 01:00 para a viagem. Essa medida é afetada por valores extremos que podem diminuir a sua representatividade para fins de estimativas. Por exemplo, se houve um dia nessas viagens com uma seqüência de grandes engarrafamentos e, como conseqüência, a viagem durou três horas, a média terá o seu valor majorado já que dificilmente haja uma viagem cuja duração seja curta o suficiente para haver uma compensação.

Outra medida de tendência menos sensível aos valores extremos é a mediana que, considerando uma lista ordenada das diferentes velocidades verificadas, é o elemento central dessa lista no caso dela conter um número ímpar de elementos, ou a média dos dois elementos centrais caso ela contenha um número par de elementos. Uma terceira medida de tendência é a moda. Ainda considerando a lista citada anteriormente, ela é a velocidade mais comum verificada. Por exemplo, se a velocidade de 15 Km/h foi verificada em três ocasiões e nenhuma outra foi verificada mais vezes, ela é a moda.

Independentemente da medida de tendência, no caso a média, o resultado ilustra a aplicação de um modelo determinístico que, assim como em desenvolvimento e manutenção de software, deveria ser a manifestação de uma probabilidade, e não uma certeza. Como tal, uma estimativa para esses fins deve sempre incluir alguma indicação do quão próximo se espera estar do real, de qual a sua acuidade, por exemplo, mais ou menos 10% (±10%).

Modelos estocásticos

Os modelos de estimativa estocásticos são aqueles que incluem em sua formulação componentes de incerteza, as suas saídas são verdadeiras estimativas de um sistema real, tendem a ser uma melhor representação da realidade dada a sua inerente natureza aleatória. Em um modelo estocástico, também se considera o grau de dispersão verificado entre as diferentes viagens utilizadas no cálculo da média das velocidades do percurso.

A medida mais usada para descrever esse grau de dispersão é o desvio padrão. Ele é calculado conforme a **Fórmula 2**.

$$s = \sqrt{\frac{\sum_{i=1}^{n} \left(Velocidade_i - \overline{Velocidade}\right)^2}{n-1}}$$

Fórmula 2. Cálculo do desvio padrão em n diferentes viagens escolhidas aleatoriamente entre aquelas com as condições similares à que se deseja estimar.

A **Fórmula 2** pode assustar um pouco num primeiro momento, mas para os fins desse texto pode ser resumida como a média dos desvios de cada velocidade individual ($Velocidade_i$) em relação à média das velocidades (Velocidade). O desvio padrão é fundamental em modelos estocásticos de estimativa e análise de risco, usado como uma unidade e representado pela letra sigma (∂).

Por exemplo, a soma dos casos com até 1 ∂ de diferença em relação à média em uma distribuição normal representa 68,26% do total dos casos. Em outras palavras, há uma chance de escolher um caso que esteja até 1 ∂ de "distância" da média de 68,26% ou então ±34,13%. Para 2 ∂, esse percentual sobe para 95,44%. A **Figura 1** ilustra a distribuição normal e as probabilidades conforme se aumenta a amplitude da faixa de confiança.

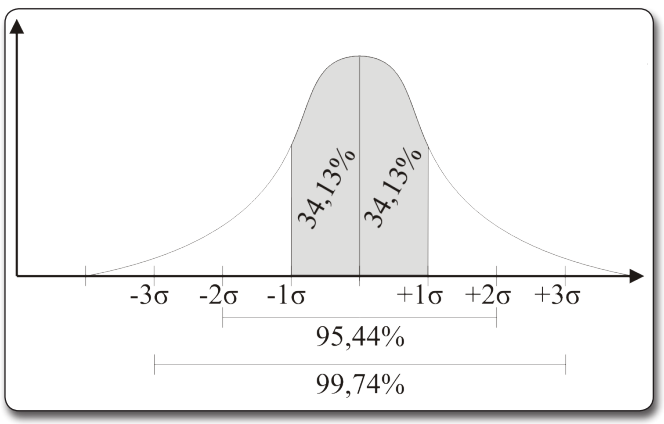


Figura 1. Distribuição de probabilidades considerando uma distribuição normal onde média, moda e mediana convergem para um mesmo ponto e os extremos se compensam mutuamente.

Considere que o eixo X da **Figura 1** represente as diferentes produtividades verificadas em ordem crescente. Como cada viagem tende a ter uma velocidade única, não é necessária uma precisão nesse nível ao estimar. O mais adequado é que no eixo X haja uma faixa ao invés de uma velocidade individual. Por exemplo, o primeiro elemento dessa série é referente à velocidade entre 5 e 10 Km/h, o segundo elemento, para velocidades a partir de 10 Km/h até 15 Km/h, e assim por diante. Observe que a velocidade não deixa de ser uma unidade de produtividade que relaciona um produto – quilômetros percorridos (Km) – com uma unidade de tempo ou custo – hora (h).

Se o exemplo utilizado fosse o desenvolvimento ou manutenção de software, essa produtividade seria adequadamente representada pelo seu inverso – a taxa de entrega expressa pela quantidade de homens-hora (Hh) por pontos de função (PF), verificada nas diferentes demandas consideradas no estudo.

No eixo Y, está a quantidade de vezes em que o deslocamento em particular teve a velocidade enquadrada na respectiva faixa do eixo X. Aproxima-se bastante do desenvolvimento e manutenção de software onde, ao invés da quantidade de vezes, considera-se a soma das horas apropriadas em cada demanda ou projeto, conforme a respectiva faixa de produtividade.

Qual o normal? Que na maior parte das viagens, haja uma convergência para a média. Haverá casos em que uma viagem

foi feita em muito mais tempo e casos em que ela foi feita em muito menos tempo, mas isso será uma exceção. Daí o nome da distribuição apresentada na **Figura 1**: Normal.

Portanto, se o desvio-padrão fosse de 00:10, a estimativa entre 00:33 (00:43 – 00:10) e 00:53 (00:43 + 00:10) teria 68,26% de chance de estar correra. O desvio-padrão é uma forma de determinar o grau de acuidade em uma estimativa.

Outras formas de representar e lidar com a incerteza

O momento em que se estima é implicitamente (e muitas vezes explicitamente) uma indicação de quanta acuidade pode-se esperar. Por exemplo, uma estimativa ao final de um estudo de viabilidade tende a ter uma menor acuidade que outra feita ao final da especificação de requisitos e essa, por sua vez, menor acuidade que uma terceira feita ao final do projeto de alto nível. Esse aumento na convergência entre as estimativas otimistas e pessimistas acontece em função da eliminação tanto de riscos de escopo como de produtividade.

A Associação para o Avanço da Engenharia de Custos Internacional (Association for the Advancement of Cost Engineering International – AACE) recomenda um sistema de classificação de custo específica para a indústria de processos, resumida na **Tabela 1**.

A Associação Americana de Engenheiros de Custo (American Association of Cost Engineers) define três tipos de estimativa de custo: a Estimativa de Ordem de Grandeza ou de Estudo (20 a 25%); Estimativa Preliminar (10 a 15%); e Estimativa Definitiva (5%). Os valores entre parênteses indicam a acuidade esperada nessas estimativas. Em diversos documentos de preparação para certificação como Profissional em Gerência de Projetos (Project Management Professional – PMP) pelo Instituto de Gerência de Projetos (Project Management Institute – PMI), são citados os seguintes tipos de estimativa: Estimativa de Ordem de Grandeza (-25% a +75%); Estimativa de Orçamento (-10% a +25%); e Estimativa Definitiva (-5% a +10%). As faixas entre parênteses também indicam a acuidade da estimativa, destacando um piso (percentual negativo) que representa uma estimativa otimista e um teto (percentual positivo) que representa uma estimativa pessimista. A Figura 2 ilustra essa dinâmica considerando uma Estimativa de Ordem de Grandeza de 01:15 minutos para o deslocamento entre o Rio de Janeiro e Niterói. Nessa figura, ao invés de uma distribuição normal exposta na Figura 1, utiliza-se uma distribuição beta, que modela eventos cuja materialização é restrita a um intervalo definido por um valor máximo e um mínimo.

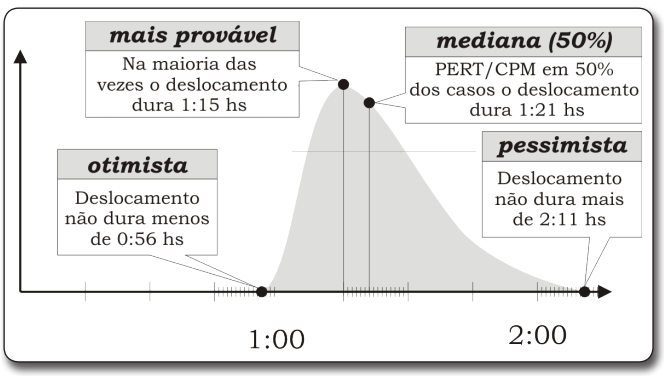


Figura 2. Distribuição de probabilidades considerando uma estimativa de Ordem de Grandeza de 01:15 (-25% a +75%).

Usando estimativas para assumir um compromisso

Uma estimativa não é resultado de uma decisão ou resolução, mas estabelecer uma meta ou assumir um compromisso com determinado prazo, esforço, escopo ou custo é uma decisão. Constrói-se uma realidade e trabalha-se para que ela se mantenha. Serem tomadas com base em informação de qualidade, em uma estimativa de qualidade, é um fator que facilita a manutenção dessa realidade planejada.

Por exemplo, um cliente liga para uma central de rádio-taxi pedindo um taxi com hora marcada para Niterói; fornece as informações necessárias e pergunta com que antecedência é

Classe de Estimativa	Nível de Definição do Projeto	Características Secundárias			
		Uso Final	Metodologia	Acuidade Esperada	Esforço de Preparação
	Expresso como % da definição completa	Típico Propósito da Estimativa	Típico Método de Estimativa	Típica variação entre a base e o teto das faixas	Grau de esforço relativo ao menor índice de custo de 1
Classe 5	Zero até 02%	Planejamento do Conceito	Fatoração da capacidade; modelos paramétricos; julgamento; ou analogia	B: -20% a 50% T: +30% a 100%	01 (0.005% do custo do projeto)
Classe 4	01% até 15%	Estudo de Viabilidade	Fatoração do equipamento; ou modelos paramétricos	B: -15% a 30% T: +20% a 50%	02 a 04
Classe 3	10% até 40%	Orçamento, Autorização ou Controle	Custos unitários semi-detalhados com itens de linha de montagem	B: -10% a 20% T: +10% a 30%	03 a 10
Classe 2	30% até 70%	Controle ou Proposta	Custos unitários detalhados com estimados	B: -5% a 15% T: +5% a 20%	04 a 20
Classe 1	50% até definição completa	Verificação da Estimativa	Custos unitários detalhados com reais	B: -3% a 10% T: +3% a 15%	05 a 100

Tabela 1. Classes de estimativas para a indústria de processo de acordo com a AACE

mais seguro pedir um taxi para chegar lá. Ela soube de uma história em que outra atendente perdeu o emprego por causa da reclamação de um cliente que perdeu um vôo por ter considerado a recomendação de antecedência que ela dera, por isso ela sugere que o cliente marque um horário com uma antecedência de 02:11.

A técnica de estimativa denominada Estimativa de Três Pontos considera não apenas um ponto, mas diferentes pontos que representam a estimativa otimista, mais provável e pessimista. Ela pode ser obtida pela utilização de modelos econômicos que incluam faixas de acuidade pré-estabelecidas conforme o domínio que se deseja estimar (aos moldes daquelas expostas anteriormente) ou solicitando esses pontos para as partes responsáveis pelos diferentes pacotes de trabalho envolvidos. A estimativa otimista considera que tudo sairá tão bem quanto possível e a pessimista que o pior aconteça, porém devem-se desconsiderar eventos muito remotos.

Ao estimar projetos de software, recomenda-se dividir para conquistar, dividir o todo em pacotes de trabalho menores que possam individualmente ser estimados com maior acuidade que o todo. Independentemente da grandeza que se deseje estimar, recomenda-se que cada um desses pacotes de trabalho tenha até 40 horas de esforço. Isso dificulta essa abordagem em momentos preliminares do ciclo de vida de um projeto, o que destaca a importância de haver modelos econômicos para o domínio da engenharia de software.

PERT/CPM

Na Figura 2 está destacado um ponto (01:21) que representa a mediana entre a estimativa otimista e a pessimista; esse ponto idealmente representa a estimativa cuja possibilidade de subestimar é a mesma de superestimar (50%-50%). Idealmente porque ele foi obtido por uma simplificação definida no modelo PERT/CPM (Program Evaluation and Review Technique/ Critical Path Method) que calcula a média ponderada entre a estimativa otimista, a estimativa pessimista e a estimativa mais provável (ou a moda e cujo peso nessa ponderação é quatro vezes o peso da estimativa otimista e da pessimista), conforme a **Fórmula 3**.

$$Mediana = \frac{Otimista + 4 \times Moda + Pessimista}{6}$$

Fórmula 3. Simplificação para obter a mediada (50% de chance) usando o modelo PERT/CPM.

Ao utilizar a fórmula do PERT/CPM, a estimativa para a duração do deslocamento entre o Rio de Janeiro e Niterói é expressa como 01:21 com 50% de chance. Para muitos fins, essa chance pode ser suficiente em tempo de planejamento, esse não era o caso da atendente da central de rádio-taxi!

Simulação de Monte Carlo

Uma terceira técnica que permite obter estimativas pontuais com um nível de confiança, com um percentual de chance nos moldes do exposto na fórmula do PERT/CPM, ou utilizando uma distribuição normal, é a Simulação de Monte Carlo. Ela também requer como entrada as estimativas otimista e pessimista para a grandeza que se deseja estimar, sejam elas obtidas pelo julgamento de um especialista, pela resolução de um grupo de especialistas ou pela utilização de algum modelo de custos.

O projeto é dividido em partes. Cada parte tem a sua estimativa otimista e pessimista elaborada. A simulação consiste em gerar milhares de valores aleatórios entre esses pontos de cada parte, sumarizar o resultado e avaliar a distribuição da amostra gerada.

Por exemplo, voltando ao contexto do desenvolvimento e manutenção de software e considerando que a grandeza que se deseje estimar seja o prazo conjuntamente com o esforço, a Simulação de Monte Carlo permite fornecer uma estimativa pontual de treze meses com uma chance de 75%, ou então de nove meses com 25% de chance. A Figura 3 ilustra os resultados de uma Simulação de Monte Carlo combinando a avaliação de duas grandezas: esforço e prazo.

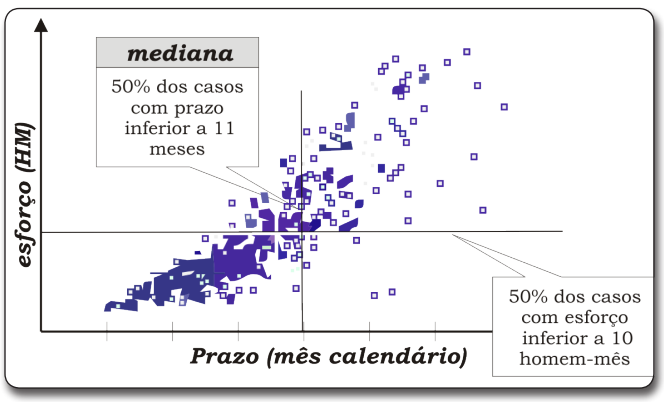


Figura 3. Resultado da Simulação de Monte Claro combinando duas grandezas de interesse, esforço e prazo; em destaque o ponto com a mediana de ambos.

A utilização da Simulação de Monte Carlo não traz resultados relevantes para o processo decisório onde a estimativa esteja num nível de detalhamento muito baixo, com pouco detalhamento. Por exemplo, numa fase bastante preliminar do projeto, não há ainda visibilidade de atividades cujo esforço seja de até 40 horas cada. O seu escopo já está definido, porém ainda não há uma Estrutura Analítica de Projeto (EAP) no nível de granularidade adequado para estimativas diretas de esforço, prazo ou custo. Para suprir a necessidade de estimar também em momentos como esse, os modelos paramétricos foram concebidos.

A Ordem de Grandeza: Uma quinta dificuldade ao estimar

Com base no que foi exposto, percebe-se uma quinta dificuldade ao estimar: **(5) A ordem de grandeza daquilo que será estimado**. Quanto maior essa ordem de grandeza, maior a dificuldade em produzir uma estimativa de qualidade. Como conseqüência disso, uma dinâmica perversa se constrói: Quanto maior a ordem de grandeza daquilo que se deseja estimar, maior é a acuidade exigida e mais difícil é obter essa acuidade usando

estimativas diretas. A **Figura 4** ilustra essa dinâmica e o gap que se forma a partir da interseção das duas curvas.

Observe que a acuidade exigida é de 100% num extremo da **Figura 4**. Tecnicamente no desenvolvimento e manutenção de software, isso é inexeqüível. Se essa ilustração deixa de ser apenas uma tendência e reflete a realidade em que o analista está inserido, é necessária gerência de expectativas no sentido de diminuir esse gap e foge do escopo deste texto, mas tendo tanta, se não mais importância, que as técnicas aqui apresentadas.

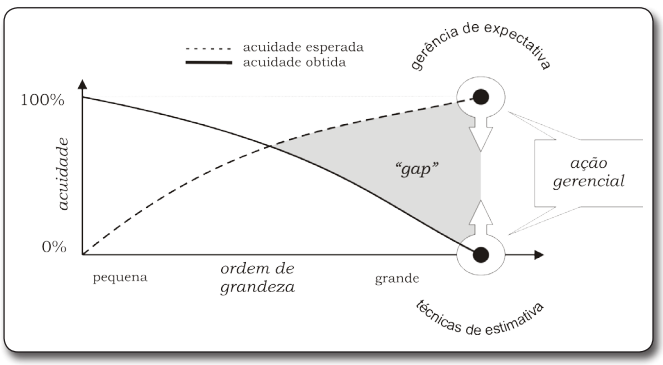


Figura 4. Ilustração empírica da relação entre a dificuldade em estimar, a acuidade obtida, e a maior acuidade esperada conforme aumenta a ordem de grandeza do objeto estimado.

Os modelos de estimativa paramétricos dão insumos para o aumento da acuidade da estimativa e a criação de um ambiente propício a isso, principalmente em projetos de maior porte.

Modelos paramétricos em desenvolvimento e manutenção de software

Iniciativas de introdução de modelos de estimativa paramétrica no desenvolvimento e manutenção de sistemas visam maior assertividade e mais subsídios para uma negociação informada, onde haja trocas e concessões entre as diferentes variáveis do projeto: Se um prazo 20% menor é necessário, quanto do escopo pode ser adiado para outra versão? Por que o custo apresentado é de R$ 120.000,00 e não metade? O que é necessário para que essa seja a estimativa de custo para o projeto? Ao empreender uma iniciativa como essa, respostas são seguidas de uma explicação com base em dados históricos e análise de probabilidades.

Para isso, a sistematização do planejamento do escopo (quilômetros percorridos em nosso exemplo inicial) é um passo necessário e seu início se dá pela introdução de uma unidade de tamanho para o escopo, um fator primário de custo. A partir dessa unidade é que será possível estabelecer as relações de esforço, custo e prazo com o escopo, viabilizando o que está sendo denominado neste documento como decisão informada.

Relação entre escopo e esforço

O fator primário de custo, essa unidade de escopo, será a base para a estimativa de esforço e, associado a outros fatores secundários que aumentem ou diminuam a produtividade observada, são as entradas para modelos de estimativa paramétrica. A maior parte deles utiliza a **Fórmula 4** como base de sua elaboração.

$$PM_{\text{no min al}} = A \times Tamanho^{B}$$

Fórmula 4. Fórmula base para estimativa de esforço.

O Tamanho é o fator de custo primário citado anteriormente, o esforço estimado nessa fórmula está representado pela sigla PMnominal. Ele tem a denominação PM em função da unidade tradicionalmente usada nesse contexto ser o Homem-Mês (Man-Month ou o termo politicamente correto e mais atual Person-Month). O nominal que o qualifica deve-se à consideração de um esforço que não inclua a sobrecarga advinda da compressão ou distensão de cronograma, da solicitação de um prazo mais curto ou mais longo do que aquele onde se equilibra a relação entre prazo e esforço.

Por exemplo, considera-se um desenvolvimento cujo esforço é originalmente estimado em 06 pessoas-mês e o prazo em 03 meses-calendário sendo empreendido com meta de prazo inferior. Ao apresentar essa estimativa, parece razoável que haja uma relação linear entre o esforço, o prazo, e a quantidade de recursos mobilizados. Com base nessa lógica podemos assumir que o mesmo trabalho descrito anteriormente será executado em um prazo de 01 mês-calendário por uma equipe de 18 pessoas.

Frederick Philips Brooks denominou essa lógica de "O Mítico Homem-Mês" (mesmo nome dado ao livro clássico The Mythical Man-Month: Essays on Software Engineering publicado em 1975, mas bastante atual em diversos aspectos). A partir de determinado ponto, na medida em que se colocam mais pessoas em um projeto, isso aumenta o custo em um ritmo superior ao que se aumenta a produção, em outras palavras, o prazo não se reduz proporcionalmente a essa mobilização.

A pesquisa de Putnam Norden identificou que, para projetos onde comunicação e aprendizado são muito necessários (como projetos de software) a curva que representa a relação entre o esforço e o tempo segue a distribuição de Rayleigh. Putnam confirmou que esta curva se aplica aos projetos de software. A curva resultante ficou conhecida como Curva Putnam Norden Rayleigh ou Curva de Staffing PNR, ilustrada na **Figura 5.**

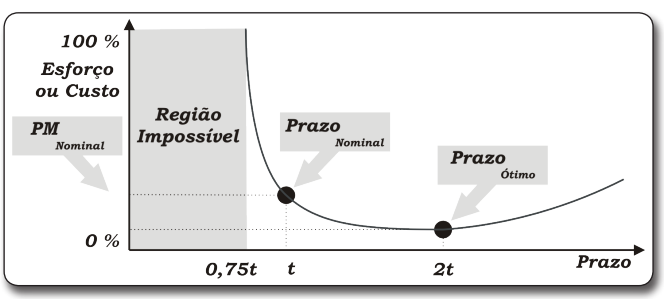


Figura 5. Curva Putnam Norden Rayleigh – PNR.

Os modelos de estimativa como o descrito na **Fórmula 4** produzem estimativas de esforço (PMnominal) referentes a um determinado prazo (t) representando um cenário onde as prioridades entre esforço e prazo estão equilibrados. O esforço

ótimo (e que compromete o prazo) é aquele verificado no dobro do prazo resultante do modelo (2t). A **Figura 5** também ilustra que um prazo inferior a 75% do prazo resultante do modelo (0,75t) não é possível de ser alcançado independentemente de quantos sejam os recursos mobilizados para a empreitada.

Normalizando o esforço entre diferentes projetos

Não existe um padrão que estabeleça quantos Homens-Hora (Hh) equivalham a uma Pessoa-Mês e um cuidado ao analisar dados de produtividade, utilizar ou calibrar modelos de estimativas é verificar se todas as referências estão normalizadas nesse sentido. Por exemplo, um projeto teve o esforço apurado em 20 PM e 160 homens-hora foi considerado como o equivalente a uma pessoa-mês. Para fins de planejamento, trabalha-se com 152 homens-hora. Portanto, para fins de análise dos dados será considerado o esforço de 21,05 PM. A **Fórmula 5** ilustra essa normalização.

$$Esforço_{PM} = \frac{\left(160_{Hh/PM} \times Esforço_{PM}\right)}{152_{Hh/PM}}$$

Fórmula 5. Fórmula para normalizar o esforço em diferentes projetos onde a quantidade de Hh/PM não é uniforme.

Normalizando o prazo entre diferentes projetos

De maneira análoga ao esforço, quando se analisam dados históricos, utilizam-se ou calibram-se modelos de estimativas, é importante conhecer o prazo nominal, que desconsidera a compressão ou distensão de cronograma. Por exemplo, se um projeto durou 12 meses-calendário e houve uma compressão de cronograma de 25% em relação ao nominal, deve-se considerar que este prazo nominal seja 16 meses-calendário. A Fórmula 6 ilustra a normalização.

$$Prazo_{Nominal} = \frac{Prazo_{Medido}}{75\%}$$

ou

$$Duração_{Nominal} = \frac{Duração_{Medido}}{SCED\%}$$

Fórmula 6. Fórmula para obter o prazo nominal a partir do prazo efetivo e a compressão de cronograma nele verificada

Capturando a produtividade

As constantes A e B da **Fórmula 4** são os fatores secundários de custo e buscam capturar a produtividade pela ponderação dos efeitos multiplicativos no esforço conforme o escopo aumenta (A) e pelas economias e deseconomias de escala que têm uma natureza exponencial (B). Essas últimas causadas principalmente: a) pelo aumento no esforço em função da sincronização entre as atividades realizadas concomitante-mente; e b) pelo aumento da quantidade de caminhos de comunicação entre os envolvidos na realização dessas atividades. A **Figura 6** ilustra o aumento da complexidade das comunicações na medida em que aumenta a quantidade de pessoas envolvidas.

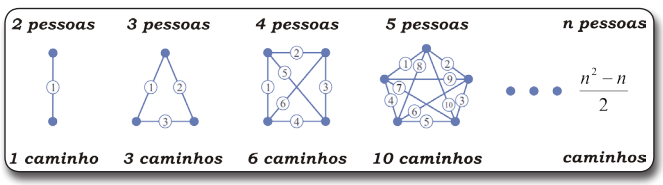


Figura 6. Aumento exponencial da complexidade na medida em que se aumenta a quantidade de pessoas interagindo e trocando informações.

A solução encontrada para lidar com esse tipo de complexidade é a introdução de agentes intermediando, coordenando o trabalho entre essas pessoas, e que não estejam diretamente ligados à produção – o gerente. Ele não produz, contudo, viabiliza a consecução de empreendimentos impossíveis sem esse tipo de ação.

Modelos baseados em simples relações usando a APF

Os modelos de estimativa baseados na análise de pontos de função (**Fórmula 7**) são uma simplificação onde os fatores de escala, que têm um efeito não linear na produtividade, são desconsiderados e a taxa de entrega busca capturar os efeitos de A e B.

$$Esforço_{(H)} = Taxa\ de\ Entrega_{(H/P)} \times Tamanho_{(P)}$$

Fórmula 7. Estimativa de esforço a partir do escopo estimado/medido em pontos de função.

Se considerarmos a aplicação dessa simplificação em um contexto onde o escopo dos projetos/demandas é relativamente uniforme, por exemplo, estejam entre 200 e 500 PF, essa

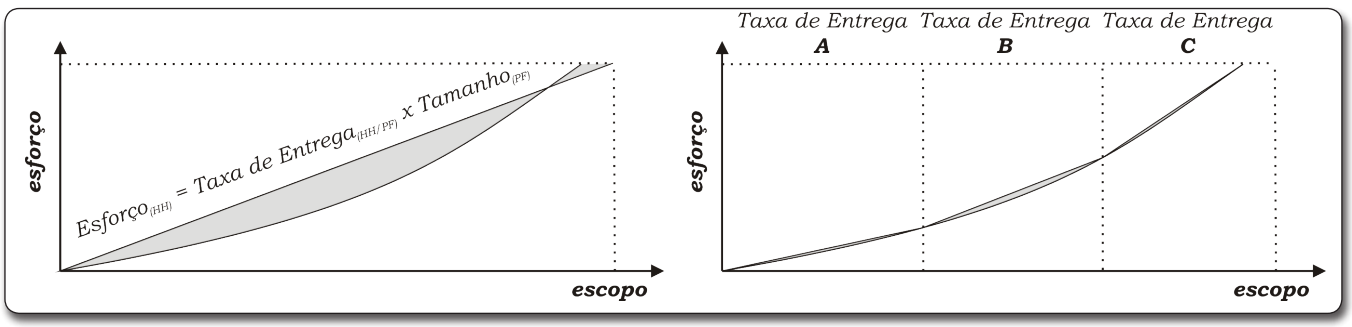


Figura 7. Adequação de um modelo linear de estimativa de esforço a partir do escopo, mesmo quando a realidade modelada tem um comportamento não-linear usando diferentes categorias de tamanho.

simplificação é plenamente adequada para fins práticos de estimativa, como representado na **Figura 7**.

Portanto, ao usar uma taxa de entrega para estimar o esforço, é importante que ela seja adequada à ordem de grandeza do projeto que se deseja estimar. Adicionalmente, os pontos de função, hoje, buscam representar uma das dimensões do tamanho de uma aplicação ou projeto – a funcional. Essa, porém, não é a única a ser considerada.

O fator de ajuste e a captura dos aspectos não funcionais

Quando a análise de pontos de função (APF) foi concebida por Allan J. Albrecht entre 1974 e 1979, sua intenção foi medir a produtividade no desenvolvimento de software e, para isso, a definição de uma unidade de produto foi fundamental. O artigo Measuring Application Development Productivity define uma técnica cujo tamanho resultante inclui tanto aspectos funcionais, afins às práticas e procedimentos do usuário, quanto aspectos técnicos e de qualidade. Aos primeiros, dá-se o nome de pontos de função não ajustados (Unadjusted Function Point Count) e, aos últimos, fator de ajuste (Value Adjustment Factor ou VAF); a combinação dos dois resulta nos pontos de função.

Naquela época, o UFPC era obtido com base na contagem das funções conforme o seu tipo – entradas, saídas, consultas e arquivos mestres (esses arquivos podem ser vistos como os requisitos de armazenamento da aplicação em contraste com os arquivos de movimento que são a mídia pela qual as transações chegam para processamento em sistemas batch). Cada tipo tinha uma quantidade de pontos de função correspondente, respectivamente – 04, 05, 04 e 10. O VAF era obtido pela ponderação do nível de influência de 10 características gerais de sistema no projeto ou aplicação, sendo medidas em uma escala de zero, indicando nenhuma influência, a cinco, indicando uma característica essencial. Pela totalização dos níveis de influência atribuídos a cada uma dessas características, determinava-se o VAF numa faixa entre 0,75 (causando uma redução em relação ao UFPC de 25%) e 1,25 (aumentando em 25%). Não havia orientação para proceder a essa classificação dos níveis de influência.

Em 1984, a técnica ganha a sua forma atual onde, além dos quatro tipos de função originais, também são contabilizadas as necessidades por dados externos à aplicação, os arquivos de interface externa; cada tipo de função passa a ter sua complexidade funcional avaliada como baixa, média, ou alta; e ambos associados – o tipo de função e a complexidade funcional – determinam a contribuição de cada funcionalidade aos UFPC.

Apesar de se manter essencialmente a mesma desde então, nesses 25 anos tem evoluído no sentido de ser uma métrica funcional que pondera apenas as práticas e procedimentos do usuário. Isso não implica que aquelas características ponderadas pelo VAF (e outras mais) não sejam importantes para o processo de estimativa.

As principais críticas à determinação do VAF e os motivos pelos quais a comunidade de usuários de pontos de função deixa de utilizá-la para ponderar o tamanho técnico do produto são: a) todas as 14 características têm o mesmo espectro de impacto de 5% no resultado da medição e, conseqüentemente, na estimativa; b) o peso relativo de cada característica é essencialmente estático e parte da definição da técnica não sendo passível de calibração, por exemplo, não é possível considerar que a complexidade do processamento tenha um impacto entre 0,73 (reduzindo a estimativa em 27%) e 1,74 (causando um aumento da mesma em 74%); e c) as orientações em sua determinação refletem um cenário obsoleto e descolado da realidade atual, por exemplo, um aplicativo onde mais de 30% das transações sejam interativas contabiliza o maior nível de influência ao avaliar a característica entrada de dados on-line. A subjetividade, as diferentes possíveis interpretações na determinação dos níveis de influência com base nas orientações fornecidas, não está nesta lista. Isso porque a taxa de entrega utilizada na elaboração da estimativa de esforço numa mesma organização com uma interpretação local, comum, estabelecida, convencionada, cumpre o papel de calibrar o modelo nesse particular.

O papel das categorias

A definição de diferentes categorias de projetos, cada uma com uma taxa de entrega específica, conforme os diferentes contextos representativos dos requisitos de qualidade e técnicos, é uma alternativa para capturar o impacto desses aspectos não funcionais e as economias ou deseconomias de escala usando o modelo com simples relações entre pontos de função e esforço. A **Figura 8** ilustra uma estratégia para isso.

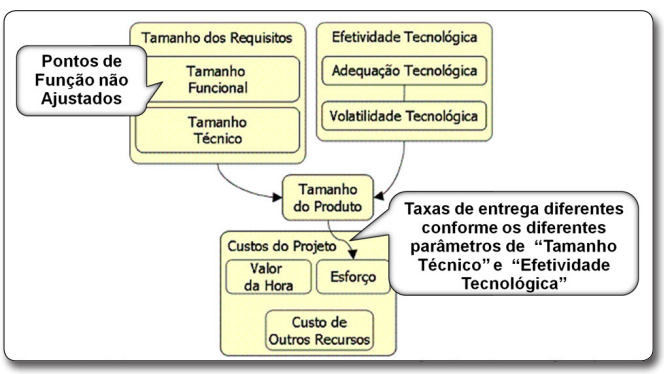
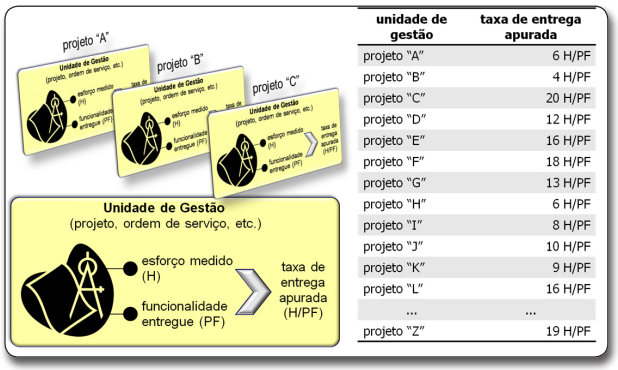


Figura 8. Uma estratégia para representar os aspectos não funcionais para fins de estimativas de software.

A prática observada no mercado é a definição dessas categorias apenas com base na linguagem de programação ou plataforma. Contudo, nem sempre essas duas dimensões são suficientes para capturar outros requisitos como, por exemplo: a complexidade do produto; o processo de desenvolvimento ou gestão utilizado; o grau de confiabilidade exigido; entre outros. Para tanto, seria necessário empreender um estudo que buscasse a determinação dessas categorias com base nas diferentes produtividades verificadas e na identificação dos aspectos comuns que levaram projetos diferentes a apresentar o mesmo nível de produtividade. A **Figura 9** ilustra a definição de unidades de gestão na qual atributos são coletados e tabulados visando identificar as diferentes categorias de produtividade verificadas, e os critérios para que um projeto ou demanda seja enquadrado numa categoria.



unidade de gestão	taxa de entrega apurada
projeto "A"	6 H/PF
projeto "B"	4 H/PF
projeto "C"	20 H/PF
projeto "D"	12 H/PF
projeto "E"	16 H/PF
projeto "F"	18 H/PF
projeto "G"	13 H/PF
projeto "H"	6 H/PF
projeto "I"	8 H/PF
projeto "J"	10 H/PF
projeto "K"	9 H/PF
projeto "L"	16 H/PF
...	...
projeto "Z"	19 H/PF

Figura 9. Coleta e apuração das taxas de entrega em diferentes projetos em busca das diferentes categorias de produtividade e critérios para o enquadramento de um projeto nessa categoria.

COCOMOII

Uma alternativa ao empreendimento de um trabalho de definição de categorias, ou pelo menos uma plataforma para isso, é a utilização de modelos pré-definidos de estimativa. O trabalho mais abrangente sobre engenharia de software nessa perspectiva econômica é a definição do modelo denominado Constructive Cost Model (COCOMO), coordenado pelo Professor Barry Boehm na Universidade do Sul da Califórnia (UsC), e que originou o livro Software Engineering Economics publicado em 1981. O trabalho relativo a essa iniciativa não parou e hoje o modelo encontra-se em sua versão COCOMOII publicada em 2000, assim como houve uma série de suplementos também publicados desde então. Ele é um modelo aberto ao público e bem definido, com uma diversidade de parâmetros adequados aos níveis de informação disponíveis em diversos momentos do ciclo de vida, podendo ser calibrado às condições locais e estruturado conforme os diferentes segmentos do mercado de software.

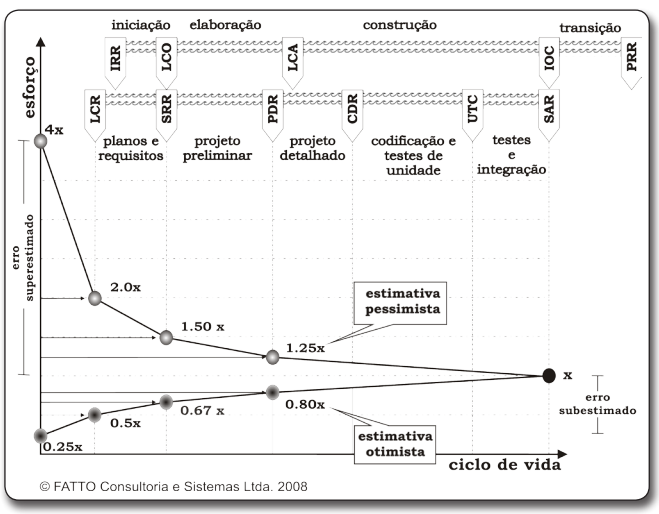


Figura 10. Uniformização dos marcos entre os diferentes processos de desenvolvimento; explicitação das fases; e demonstração da acuidade na estimativa conforme a fase do ciclo de vida de acordo com o COCOMOII

O primeiro valor do COCOMOII está na definição de suas premissas. Elas estruturam de maneira unificada as principais correntes metodológicas na engenharia de software. Essa estruturação consiste de: (a) Marcos comuns, onde determinados artefatos são entregues; (b) Fases numa perspectiva externa à função de desenvolvimento para fins de planejamento e controle; (c) Atividades cujo escopo, esforço, custo e prazo serão considerados na aplicação e geração de indicadores de produtividade; e (d) Grau de acuidade, a amplitude da incerteza conforme a fase em que a estimativa será realizada.

O modelo não se limita a isso, nem tão pouco esse é o seu objetivo final. Porém, ao fazer isso, constrói todo um arcabouço, toda uma fundação, que facilita a prática de estimativa independentemente da aplicação do modelo propriamente dito. A **Figura 10** ilustra esse arcabouço.

Conclusão

Este artigo procurou definir o que é uma estimativa e diferenciá-la de outras ações com as quais ela se confunde, apresentar uma breve introdução sobre uma série de ferramentas e técnicas para sua realização, e desfazer alguns mal entendidos sobre a dinâmica do uso da análise de pontos de função em estimativas de software, especialmente o papel do valor do fator de ajuste.

Em um espaço como este seria impossível esgotar o assunto em profundidade. Em próximos artigos vamos especificamente tratar de cada uma dessas técnicas sem perder o foco em como integrá-las com outras técnicas, ferramentas e práticas que visam melhorar a qualidade das estimativas de software, começando pelo COCOMOII. ●

Referências

L.H. Putnam. "A General Empirical Solution to the Macro Software Sizing and Estimation problem", IEEE Transactions of SW Engineering, 1978.

Boehm, Barry et al. "Software Cost Estimation With COCOMO II", Prentice Hall, 2000.

Robert E. Park. "A Manager's Checklist for Validating Software Cost and Schedule Estimates", CMU/SEI-95-SR-004, 1995.

Brooks, Frederick P., Jr., "The mythical man-month: essays on software engineering – Anniversary ed.", Addison Wesley Longman, Inc., 1995. Edward Yourdon, "Death March: The Complete Software Developer's Guide to Surviving Mission Impossible Projects", Prentice Hall, 1997.

IFPUG – International Function Point Users Group, "Framework for Functional Sizing", 2003.

Allan J. Albrecht, "Measuring Application Development Productivity", 1979.

AACE International, "Recommended Practice No. 18R-97, Cost Estimate Classification System – as applied in engineering, procurement, and construction for the process industries", 2005.

Thomas A. Willian, "Estatística Aplicada: à Administração e Economia", 2ª Edição, Cengage Learning, 2007.

Daniel D. Galorath e Michael W. Evans, "Software Sizing, Estimation, and Risk Management", Auerbach Publications, 2006.

Steve McConnell, "Software Estimation: Demystifying the Black Art", Microsoft Press, 2006.

Carlos Vazquez, Guilherme Simões, Renato Albert, "Análise de Pontos de Função: Medição, Estimativas e Gerenciamento de Projetos de Software", Editora Érica, 2008, 8ª Edição.

Cursos Online

Web Mobile PLUS

A **Revista WebMobile** oferece para seus assinantes uma série de Cursos Online de alto padrão de qualidade .**Conheça abaixo os cursos já disponíveis.**

Curso de .net em destaque

Aplicação completa de orçamento Doméstico no Visual Studio 2005

Confira neste curso online como criar uma aplicação completa no Visual Studio 2005, usando ASP.NET, Web Services, Mobile e muito mais! Veja como criar uma aplicação de orçamento doméstico, usando diagrama de classes de uma forma muita produtiva, criar classes de negócios de acesso a dados.

Confira o plano de aula completo:
www.devmedia.com.br/domesticovs

Curso de Java em destaque

Introdução ao desenvolvimento para celulares com J2ME

Confira neste curso os principais recursos do J2ME. Aprenda também o passo a passo para criar sua primeira aplicação J2ME. Neste curso você irá aprender diversas funcionalidades desta tecnologia para desenvolvimento de dispositivos móveis.

Confira o plano de aula completo:
www.devmedia.com.br/celularesj2me

Assine a **WebMobile** e comece já seu treinamento!
www.devmedia.com.br/assine

A sua melhor opção de aprendizagem!

Em breve mais novidades para você! www.devmedia.com.br/curso

DevMedia Mais Informações: www.devmedia.com.br/central - Tel.: 21 3382-5038/ 3382-5025

Introdução a Testes de Software

Melissa Barbosa Pontes

melissa.pontes@cesar.org.br

Aluna de Mestrado Profissional em Engenharia de Software do Cesar.EDU, onde realiza pesquisas sobre predição de defeitos em software. Certificada em testes de software pelo ISTBQ (International Software Testing Qualification Board), atualmente trabalha como engenheira de testes no CESAR (Centro de Estudos e Sistemas Avançados do Recife), onde desempenha atividades de definição de processos, liderança e consultoria. Possui artigos e trabalhos publicados em eventos internacionais. Participa da organização de EBTS - Encontro Brasileiro de Testes de Software, que já se encontra na quarta edição.

A qualidade de um software pode estar fortemente relacionada à existência de defeitos inseridos durante o desenvolvimento ou manutenção de um produto. Uma das maneiras de identificar os defeitos de uma aplicação de forma que eles possam ser corrigidos é através das atividades de teste de software. Devido à sua importância e complexidade, teste de software pode ser responsável por uma parcela considerável dos custos de um projeto, por isso, este assunto merece atenção por parte das organizações que desejam sucesso com a condução das atividades do processo de testes.

O que é Teste de Software

Falar de testes de software é muito mais do que falar de execução de testes. Testar um software e relatar impressões e não conformidades é fornecer um diagnóstico do estado da aplicação, e é muito importante que este diagnóstico seja o mais completo e preciso possível, porque provavelmente ele vai servir de base para tomadas de decisões em relação ao projeto que está sendo analisado.

Neste artigo veremos

Neste artigo é abordado o assunto testes de software sem o relacionar diretamente com o processo de testes ou ferramentas de testes. Daremos ênfase ao que o testador faz no seu dia-a-dia, que dificuldades ele pode encontrar e de que maneira ele pode vencer essas dificuldades. Além disso, mostraremos como o desenvolvimento de habilidades como comunicação, pensamento crítico, pro atividade e criatividade pode ajudar o testador a desempenhar melhor suas atividades.

Qual a finalidade

Testes de software são atividades que contribuem com a qualidade do software a ser testado. Um bom planejamento dessas atividades pode significar economia para um projeto, visto que a identificação de defeitos no inicio do ciclo de desenvolvimento do produto pode reduzir os custos da sua correção, além de sua confiabilidade.

Quais situações utilizam esses recursos?

Além do conhecimento técnico necessário para a realização das atividades de testes de software, o testador pode buscar o desenvolvimento de habilidades pessoais para desempenhar melhor seu trabalho. Buscar aprimoramento além das dimensões técnicas ajuda a tornar um engenheiro de testes um profissional diferenciado na área, e faz com que algumas dificuldades que podem surgir no seu dia-a-dia sejam superadas com maior facilidade.

Testar uma aplicação é questioná-la, através de casos de teste e principalmente de observações, para analisar as respostas obtidas, pois estas podem revelar defeitos. Podemos entender por defeito tudo o que ameaça a qualidade do produto, levando em consideração que qualidade é o que o cliente quer. E o que o cliente quer é relativo, porque vai depender da finalidade da aplicação a ser desenvolvida. Por exemplo, uma aplicação pode estar em conformidade com a documentação de requisitos e ainda assim o cliente achar que ela não tem qualidade, pois apresenta demora em realizar determinada operação. Dependendo de onde a aplicação será utilizada, restrições de tempo podem ser cruciais.

O que não é Teste de Software

Para um melhor entendimento do que é teste, podemos analisar o que não é ou pelo menos não deveria ser teste de software.

Teste de software não é:

• Um processo burocrático: Apesar de ter um processo bem definido para a realização das atividades de testes de software, com atividades e papéis bem explicados, este processo não deve ser burocrático, nem inibir a criatividade do testador, que deve se sentir a vontade para explorar a aplicação em busca de defeitos, mesmo que exista um roteiro de testes a seguir. Isso quer dizer que o testador não precisa deter-se a documentações, pois nem sempre estas estão atualizadas e livres de erros. O testador pode seguir sua intuição, desde que utilizando experiência e bom senso. Neste caso, a documentação do projeto pode servir como guia, mas não deve ser considerada uma verdade absoluta e inquestionável;

• Atividade limitada a seguir um roteiro: Se os testadores se limitam a fazer somente o que pede o roteiro com os casos de teste, provavelmente deixarão de perceber situações estranhas merecedoras de investigação na aplicação. A preocupação em ler o passo a passo do roteiro que é designado ao testador pode tirar seu poder de pensamento crítico em relação ao que está sendo avaliado. O testador no exercício da execução de testes deve ter plena liberdade de experimentar situações novas, mesmo que estas lhe venham na cabeça naquele momento de sua atividade. Ele deve também ter liberdade de discutir situações duvidosas que possa encontrar, mesmo que estas venham de uma observação feita por ele, que não estava previamente planejada pela pessoa que escreveu os casos de teste;

• Uma atividade que pode ser bem desempenhada por qualquer pessoa: Com isso, quer-se dizer que as atividades de testes não serão bem desempenhadas se quem as tiver realizando não seja um profissional preparado para tal. Para se testar eficientemente uma aplicação, é preciso possuir algumas habilidades, um conjunto de conhecimentos específicos de técnicas e alguma experiência no assunto;

• Uma atividade de garantia de qualidade: Não devemos confundir informar sobre a qualidade do produto com garantir a qualidade do produto gerado. As atividades de teste podem contribuir para a qualidade do produto final, mas a garantia dessa qualidade é de responsabilidade de toda a equipe do projeto. Dizer que testes são atividades de garantia de qualidade pode dar a impressão de que um software que não tem qualidade é um produto mal testado;

• Uma atividade que pode ser deixada para o final do projeto: Quando se trata testes de software como apenas uma fase do ciclo de desenvolvimento de um projeto, corre-se o risco de ter o tempo dedicado a essas atividades reduzido, seja por atrasos no cronograma de desenvolvimento ou alocação tardia de testadores no projeto;

• Uma atividade destrutiva: O livro Lessons Learned in Software Testing traz uma visão interessante sobre testes como sendo uma atividade tão construtiva quanto o desenvolvimento do software que será testado. No livro, os autores afirmam que teste não quebra um software, mas nos tira a ilusão de que ele funciona corretamente.

A Importância de Testes de Software

Quanto mais cedo um defeito for encontrado no ciclo de vida de um software, mais barato é o custo da sua correção. De acordo com Myers, autor do livro The Art of Software Testing, corrigir um defeito que se propagou até o ambiente de produção pode chegar a ser cem vezes mais caro do que corrigir este mesmo defeito em fases iniciais do projeto. Muitos dos defeitos que são encontrados no código da aplicação são originados na fase de levantamento de requisitos. Alguns estudos publicados no Chaos Report, um relatório elaborado pelo grupo de pesquisa Standish Group, em 2004, revelam que aproximadamente 45% dos recursos de um projeto de software nos Estados Unidos e Europa chegam a ser dedicados a atividades de testes e simulações. Assim, testes de software devem ser encarados como investimento em qualidade, e até mesmo economia, já que é uma atividade que identifica defeitos o quanto antes em um projeto.

Então, como podemos testar um software?

Existe mais de uma maneira de se conduzir as atividades de testes de software em um projeto. Bret Pettichord, co-fundador da WatirCraft (http://www.watircraft.com/), uma empresa de consultoria em automação de testes, separa testes em escolas, de acordo com algumas características, tais como afinidade intelectual, objetivos definidos, hierarquia de valores, vocabulário comum, dentre outras. Essas escolas, através de suas diferentes visões, fornecem cinco maneiras distintas de entender e realizar testes de software. As escolas são: Escola Analítica, Padronizada, de Qualidade, Dirigida a Contexto e Escola Ágil. Segundo Pettichord, a Escola Analítica deu origem à Escola Padronizada, e esta, originou as demais [Fonte: Schools of Software Testing, Bret Pettichord].

A **Escola Analítica** vê teste como uma atividade altamente técnica e rigorosa, uma visão comum no meio acadêmico. Os casos de teste nessa escola possuem somente uma resposta certa. Um alto nível de documentação é requerido e os testadores devem verificar se o software está de acordo com esta documentação.

A **Escola Padronizada** vê teste como uma maneira de medir progresso com ênfase em custo, se utilizando de padrões e repetições. O processo de testes deve ser previsível, planejado e reproduzível. As certificações de testes existentes hoje no mercado seguem nessa linha.

A **Escola de Qualidade** enfatiza o processo e requer disciplina.

Algumas vezes se posiciona como um gargalo, pois em alguns casos, os testadores têm que proteger o usuário de um software ruim, e os engenheiros de qualidade que decidem se o software pode ser liberado.

A **Escola Dirigida a Contexto** tem o foco nas pessoas. Os testadores se colocam no lugar do cliente ou usuário na hora de procurar por defeitos. Admite que teste é uma atividade que requer experiência, habilidades, aprendizado rápido, atividade mental e multidisciplinaridade. Preocupa-se em focar no que agrega valor no momento. Os testes geralmente são projetados no mesmo momento da sua execução.

A **Escola Ágil** enfatiza automação de testes. Vê testes como um complemento do desenvolvimento, uma atividade que indica que o desenvolvimento está finalizado. Geralmente usa desenvolvimento dirigido a testes.

Na prática, visando o mercado, a escola dirigida a contexto traz muitos benefícios, pois seus adeptos procuram desenvolver diversas habilidades que lhes permite superar algumas dificuldades nos projetos, como por exemplo, documentação ruim ou inexistente, e até mesmo testar sob pressões de prazo, com custo direcionado a testes reduzido. Adeptos dessa escola assumem que a aplicação deve ser explorada além das fronteiras da documentação existente, e fazem isso testando além dos roteiros, quando estes existem. Daqui em diante, seguiremos a linha de raciocínio dos adeptos da Escola Dirigida a Contexto, e vamos então, entender porque testar bem uma aplicação requer experiência, habilidades, aprendizado rápido, atividade mental e multidisciplinaridade.

Como Testar sem o Documento de Requisitos?

Não é uma situação rara a equipe de projeto de software receber uma documentação de requisitos confusa, cheia de ambigüidades, e com situações que não são testáveis. Muitas vezes uma documentação pobre distrai testadores inexperientes, que podem até utilizar requisitos incorretos como fonte para casos de teste. Dependendo da maneira como a documentação está escrita, pode ser que o testador perca o foco do que a aplicação de fato deve fazer, e preste atenção em detalhes que não são os mais importantes do ponto de vista do cliente. Sem falar que uma documentação pode dar a idéia de cem por cento de cobertura dos testes em relação aos requisitos, quando podem existir muitas funcionalidades que ainda não foram testadas na aplicação. Pode também dar margem para que o testador derive casos de teste incompletos, já que o seu foco está voltado para cobrir somente a documentação. Vários problemas podem ocorrer quando temos casos de testes incompletos. Estes testes podem passar, mesmo existindo defeitos no software.

O que fazer então se o testador estiver numa situação parecida com essa? Testar! Como? Utilizar-se de heurísticas é uma boa saída. De acordo com Michael Bolton, consultor da empresa Developsense (www.developsense.com), heurística é um método falível de se resolver um problema. Sendo falível, não é perigoso utilizar esse método para testes? Absolutamente! Ainda segundo Michael, heurística quase sempre funciona, e somente às vezes falha. A utilização de heurísticas esta diretamente relacionada com uma abordagem exploratória, uma maneira de ver a aplicação com a visão do usuário. O testador então assume certas condições como verdade, e parte delas para analisar a aplicação. Qualquer não conformidade com o que era esperado, não deve ser imediatamente considerada um problema. Pelo fato de se estar usando um método falível, essa situação deve ser mais bem analisada, muitas vezes discutida com o restante da equipe, para somente então tomarmos uma atitude de registrar um defeito. Uma maneira de entender bem como se utilizar de heurísticas é utilizar um exemplo do nosso dia-a-dia. Um bom exemplo é: Em épocas de fim de ano, muitas lojas de Shopping Centers contratam trabalhadores temporários. Existe a possibilidade de esse fato ocorrer todo ano? Sim, existe! Mas não dá para ter certeza de que isso sempre irá ocorrer. Porém, a chance de que ocorra pode direcionar nossas ações. Podemos tomar a decisão de elaborar bem nosso currículo, para que fique atrativo para os empregadores. Para isso, podemos investir em algum treinamento, com a intenção de mostrar que estamos capacitados para tais vagas. Assim, essa heurística termina nos servindo de guia, assim como nas atividades de teste.

Na prática então, a utilização de heurísticas nas atividades de testes de software pode dar ao testador noções de como a aplicação deveria se comportar sob determinadas circunstâncias. Através de seu uso, o testador vai analisando a aplicação e construindo um modelo mental. Por exemplo, ele pode começar a testar o produto em relação à consistência com suas próprias funcionalidades. Qualquer fluxo que esteja fora do padrão do resto da aplicação é passível de investigação. Também pode analisar se a aplicação está consistente com seu padrão visual, com o que ela se propõe a fazer, com a maneira que um usuário comum esperaria utilizá-la, e muitas outras conformidades podem ser testadas, sem a necessidade de uma vasta documentação que explique cada item acima. Testar dessa maneira pode fazer com que um testador envie seu relatório de casos de teste executados, todos passando, e ainda assim escrever uma seção com vários defeitos encontrados, provenientes dessa exploração. Não testar dessa maneira pode fazer com que um testador envie seu relatório de casos de teste executados, todos passando, e ainda assim a equipe entregue uma aplicação cheia de defeitos para o cliente.

Sinal Verde, Perigo!

Se existe algo em toda a documentação gerada pelo processo de testes de software que não recebe muita atenção, é a seção verde do relatório de defeitos. Os testes que estão falhando geralmente ficam marcados de vermelho, para que recebam atenção especial, pois esses testes revelaram defeitos que vão precisar ser analisados. Então, por que deveríamos nos preocupar com os testes que estão passando? Porque, dependendo da maturidade do time que executou os testes, ou da maneira como a organização encara a execução de testes, pode ser que muita coisa importante tenha sido deixada de lado. Algum comportamento estranho na aplicação pode não

ter sido percebido, porque em muitos casos, os testadores ficam mais preocupados em seguir o roteiro de casos de teste, tentando fazer exatamente o que o passo a passo manda fazer, do que ver a aplicação tentando imaginar como realmente ela deveria funcionar em certas circunstâncias. Nesses casos, não é raro casos de testes incompletos darem a falsa idéia de estabilidade do sistema, enquanto defeitos continuam escondidos, a espera de alguém experiente para revelá-los.

Qual a saída? Existem alternativas para contornar este problema. Uma delas é utilizar o teste como base para a execução, e não se limitar ao que ele manda fazer. Entender qual o objetivo do teste e tentar alcançá-lo não somente através do passo a passo que foi projetado, mas imaginar outras maneiras de chegar ao mesmo objetivo, por outros caminhos na aplicação. Esse tipo de atividade requer atenção aos detalhes, diversas vistas de um só ponto (funcionalidade em questão), facilidade de reproduzir com clareza situações vividas, boa memória, clareza de escrita, pensamento crítico, etc.

O que pode ajudar muito a tornar a equipe de testes proativa e experiente para que realizem as atividades dessa maneira é buscar o desenvolvimento de cada um. Buscar um time multi-disciplinar pode trazer muitos benefícios para uma organização. Assim como em qualquer profissão, o desenvolvimento de habilidades pessoais pode fazer a diferença para um time.

Habilidades do Testador

Diversas são as habilidades que podem ser desenvolvidas por cada integrante de um time de testes. Muitas habilidades estão diretamente relacionadas com algumas atividades de testes, e podem dar um diferencial ao profissional que busca desenvolvê-las, tornando suas atividades mais completas. A seguir, mencionaremos algumas habilidades e como o desenvolvimento delas pode ajudar nas atividades de teste.

Comunicação

Uma das principais atividades do engenheiro de testes é comunicar. Seja comunicação escrita ou oral, esta deve ser feita com muita clareza e precisão. Um testador deve relatar suas observações de maneira que outros integrantes do projeto possam entender rapidamente, pois o desenvolvedor, por exemplo, terá que examinar as informações que os testadores o enviam para entender o que tem de errado na aplicação. O testador deve munir a equipe do projeto de contexto, acerca do que foi encontrado. Deve relatar em que condições estava a aplicação no momento que ocorreu a falha, qual o fluxo percorrido, listar tudo o que foi encontrando pelo caminho, mesmo que não esteja descrito no caso de testes.

Se um defeito não é bem descrito, pode não receber a atenção merecida no momento em que a equipe decide priorizar defeitos para correção. Também, um desenvolvedor pode não conseguir reproduzir o problema encontrado, por falta de informações relevantes no relatório de defeitos. A comunicação escrita merece um cuidado especial, pois os relatórios de defeitos devem ser escritos de maneira imparcial. Não se deve criticar o desenvolvedor por um defeito encontrado, pois todos, desenvolvedores e testadores, estão no projeto com o mesmo objetivo, que é entregar um produto de qualidade ao cliente. Então, é necessário muito cuidado com os termos que serão utilizados, principalmente se testadores e desenvolvedores são de países diferentes, se possuem culturas diferentes. Nesses casos, a utilização de certos termos pode causar mal entendidos difíceis de serem resolvidos. A comunicação oral também tem sua importância. Testadores devem comunicar o estado da aplicação com o intuito de ajudar, de mostrar o caminho percorrido em busca do defeito.

Um relatório de defeitos deve ser escrito para seu público-alvo. O que isso quer dizer? Um relatório feito para os desenvolvedores deve ser escrito em nível técnico. Já um relatório feito para a coordenação do projeto, talvez precise ser escrito de maneira gerencial. Em um relatório gerencial, pode ser interessante descrever como um problema em uma funcionalidade pode aborrecer o cliente, ou porque isso pode atrasar a saída do produto para o mercado, por exemplo.

Pensamento Crítico

Pensamento crítico é o que pode fazer com que um testador, ao final da execução de um ciclo de testes, perceba que todos os casos de teste da suíte estão passando, e ainda assim, ter um relatório de defeitos sem casos de testes associados a eles. Esse olhar crítico pode impedir que um produto seja lançado no mercado mesmo que a maioria de seus casos de teste esteja passando.

Também é entender que não adianta sair executando todos os testes que existem na ferramenta de gerenciamento de testes, se estes não fizerem sentido para o momento que o time está passando. Testes devem ser selecionados com critério para compor um ciclo de execução. Ao invés de colocar todos os casos de testes existentes no ciclo, muitas vezes é mais prudente analisar as funcionalidades que estão sendo desenvolvidas pela equipe. Devem-se analisar quais funcionalidades são críticas ou quais funcionalidades estão prestes a ser liberadas para o mercado. Áreas de risco devem ter prioridade, e caso o tempo seja curto para a execução dos testes, uma boa estratégia é essencial. Deve ser analisado o que será testado primeiro, o que pode ser deixado para depois, e o que pode ficar sem ser testado. Tomar esse tipo de decisão não é fácil e exige preparo e conhecimento do projeto.

Proatividade

Uma pergunta muito comum em eventos de testes de software no Brasil e no mundo é: De que maneira você mede a produtividade ou eficiência dos seus testadores? É muito comum testadores serem medidos pela quantidade de casos de testes que conseguem executar em um período de tempo. Que tipos de problemas isso pode acarretar se essa for a única maneira de avaliar a equipe?

Testes mal executados, visto que os testadores podem realizar suas atividades em busca de números, e não de qualidade. Perda do foco, porque os testadores vão ficar preocupados em executar a quantidade de casos de testes que lhes é atribuída por dia/semana. Podem deixar de perceber se algo estranho acontecer e não estiver descrito no caso de teste, pois eles podem considerar

que não tem tempo a perder analisando coisas extras. Além de que um testador proativo pode ter seu trabalho considerado igual ao de um testador que não tem proatividade alguma, porque ambos terão executado seus casos de testes no tempo acordado.

Mas o trabalho do testador proativo está por trás disso tudo. Ele deve encontrar mais defeitos na aplicação, relatar em detalhes tudo o que observou e encontrou durante seus testes, fornecer os dados de testes em arquivos para facilitar a reprodução do defeito, estar disponível para que o desenvolvedor esclareça alguma coisa que tenha sido mal entendida, etc. Por isso, somente contar quantos casos de testes foram executados pode esconder todas essas atividades que também merecem muita importância. Existem gerentes de testes, principalmente em multinacionais, que analisam as impressões do cliente, depois que o produto é entregue, como forma de analisar o desempenho da sua equipe. E, apesar de saber que é impraticável testar todas as combinações de todos os fluxos da aplicação, de maneira que não podemos garantir que uma aplicação não tenha nenhum defeito, os defeitos que são encontrados pelo cliente também servem de indicativo de como a equipe está se saindo na realização das suas atividades.

Criatividade

Uma pergunta importante. Quem usa a criatividade no seu time de testes? Quando estamos seguindo um processo de testes bem definido, cada um tem seu papel dentro do time. Tem o arquiteto, que organiza a estratégia de testes e os planeja. Depois o projetista de testes pega esse planejamento e escreve os casos de teste, e então os executores pegam esses casos de testes projetados e os executa. Se os executores só executam o que o projetista determinou que ele iria fazer, e se o projetista só projeta o que o arquiteto determinou que ele iria projetar, o uso da criatividade fica restrito a um terço do time. No máximo o projetista pode interferir em alguma coisa, mas os executores ficam somente seguindo um roteiro, algo que o mandaram fazer. Quais são as conseqüências desse tipo de processo? A lista é grande, mas aqui vão alguns itens que merecem atenção especial:

1. Com o tempo a execução de testes pode se tornar uma atividade cansativa, à medida que os testes sofrem poucas modificações, ou o testador já tenha executado aquela suíte algumas vezes;
2. É gerada uma forte dependência de roteiros de teste, que por conseqüência devem depender de uma documentação de requisitos;
3. Manutenção difícil da suíte de testes. Tendo em vista que requisitos mudam com freqüência, e isso impacta diretamente na suíte de testes gerada.

Como o desenvolvimento e uso da criatividade podem ajudar a todo o time? À medida que os testadores, nesse caso, os executores, se sentem livres para explorar a aplicação da maneira que acham mais conveniente, podem utilizar o roteiro de testes somente como uma garantia de cobertura do cenário do requisito correspondente, e depois, começar a investigar detalhes na aplicação. Para desempenhar essa atividade, é fundamental que os executores tenham um bom conhecimento do domínio da aplicação, habilidades técnicas, envolvimento com o setor de marketing da empresa, para entender o que o mercado espera do produto, etc. Uma reunião de brainstorm com todo o time, principalmente com arquitetos e desenvolvedores, pode gerar muitas idéias de cenários que podem ser explorados. O testador usa a sua imaginação, mas pode ser ajudado pelo resto da equipe. Se colocar no lugar de grupos de pessoas que vão utilizar a aplicação também pode fazer surgir novas idéias. Entender como se comportam esses grupos vai dar uma noção do que eles vão buscar na aplicação, ou o que eles gostariam de ver.

Conclusões

Testes de Software estão muito mais relacionados com o que o testador pode fazer no seu dia-a-dia do que com documentações e realização de atividades em seqüência. Apesar da enorme importância de ter um processo como base, é imprescindível o desenvolvimento das habilidades das pessoas, para que elas saibam realizar bem suas atividades até mesmo em empresas que não têm nenhum processo bem definido. É importante a consciência de que testar um software exige experiência, e que isso faz a diferença no momento em que existem pressões de prazo, orçamento, ausência de ferramentas, enfim, limitações de recursos no projeto. Em suma, trata-se de uma atividade que requer um perfil próprio para o profissional, diferenciado do perfil do desenvolvedor, e que requer também muita especialização, tendo em vista a quantidade de áreas distintas que existem dentro da área de testes de software, por exemplo: Testes de Unidade, Testes de Funcionalidades, Testes de Desempenho e Testes de Segurança, dentre varias outras áreas.

Referências

Lessons Learned in Software Testing - Cem Kaner, James Bach, Bret Pettichord.

Black Box Software Testing - http://www.testingeducation.org/BBST/

Rapid Software Testing - http://www.satisfice.com/info_rst.shtml

Foundations of Software Testing: ISTQB Certification - Dorothy Graham, Erik van Veenendaal, Isabel Evans, Rex Black

The snake in the monkey's shadow. Pradeep Soundararajan – http://testertested.blogspot.com

Schools of Software Testing - Bret Pettichord

The Art of software Testing - Glenford J. Myers

Chaos Report 2004 - www.standishgroup.com

Controle Estatístico de Processos Aplicado a Processos de Software

Do "Chão de Fábrica" para as Organizações de Software

Monalessa Perini Barcellos

monalessa@inf.ufes.br

Doutoranda em Engenharia de Sistemas e Computação (COPPE/UFRJ), Mestre em Engenharia de Sistemas e Computação (COPPE/UFRJ), Bacharel em Ciência da Computação (UFES). Professora do Departamento de Informática, área Engenharia de Software, da Universidade Federal do Espírito Santo (UFES). Atuante, desde 1999, em projetos, consultorias, treinamentos e pesquisas da área de Engenharia de Software.

Os avanços tecnológicos e a alta competitividade do mercado estão, continuamente, aumentando a demanda para que os softwares produzidos sejam cada vez maiores, mais robustos e mais adaptáveis às mudanças dos ambientes em que operam. Mesmo diante de todos os avanços tecnológicos, os projetos de software ainda são caracterizados, em boa parte das organizações, por testes insuficientes, treinamento inadequado, cronogramas irreais e arbitrários, falta de controle de mudanças, orçamentos não cumpridos e estabelecimento e utilização de padrões de qualidade insuficientes. Isso faz com que a demanda pelo bom gerenciamento dos projetos e processos de software também aumente.

A medição é um dos caminhos que apóiam o gerenciamento dos processos e projetos de software. As medidas, ao serem coletadas e armazenadas, podem ser analisadas através de métodos e fornecerem, assim, informações importantes para a realização de ações corretivas e preventivas que orientem os projetos e processos a alcançarem seus objetivos.

Neste artigo veremos

Este artigo apresenta os principais conceitos e questões relacionadas ao controle estatístico de processos e sua utilização em processos de software.

Qual a finalidade

Fornecer conhecimento essencial sobre a utilização do controle estatístico de processos para organizações, pesquisadores, estudantes e profissionais de software envolvidos em atividades relacionadas aos níveis gerencial e/ou estratégico das organizações.

Quais situações utilizam esses recursos?

Para organizações que estão a caminho dos níveis mais elevados de maturidade de seus processos e precisam implantar o controle estatístico de processos; para organizações que estão iniciando o caminho de melhoria de processos e desejam, desde já, conhecer e se preparar para o que será necessário no futuro, nos níveis mais elevados; para profissionais envolvidos na implementação de programas de melhoria de processos de software; para estudantes e pesquisadores que desejam explorar o tema.

A utilização do controle estatístico de processos, tema deste artigo, trata do que foi descrito no parágrafo anterior, considerando, nesse caso, que os métodos utilizados são métodos estatísticos. Ou seja, são utilizados métodos estatísticos específicos para analisar os dados coletados para medidas ao longo de projetos, a fim de analisar o comportamento dos processos e determinar seu desempenho para, a partir daí, fornecer diretrizes para, caso seja necessário, realizar ações corretivas e de melhoria que levem os processos a alcançarem os objetivos para eles estabelecidos. O desempenho de um processo pode ser definido como uma medida dos resultados atuais que o processo alcançou e pode ser caracterizado por medidas de processo (por exemplo: esforço, prazo e eficiência de remoção de defeitos) e por medidas de produto (por exemplo: confiabilidade e densidade de defeitos) (CMMI, 2006).

O controle estatístico de processos foi originalmente desenvolvido para implementar um processo de melhoria contínua em linhas de produção na área de manufatura, envolvendo o uso de ferramentas estatísticas e técnicas de resolução de problemas com o objetivo de detectar padrões de variação no processo de produção para garantir que os padrões de qualidade estabelecidos para os produtos fossem alcançados. É uma metodologia utilizada para determinar se um processo está sob controle, sob o ponto de vista estatístico.

O sucesso da aplicação do controle estatístico de processos na manufatura levou à sua aplicação em outras áreas, como química, eletrônica, alimentação, negócios, saúde e desenvolvimento de software, entre outras.

A principal diferença entre o controle estatístico de processos e a estatística clássica é que esta, tipicamente, utiliza métodos baseados em dados 'estáticos no tempo', ou seja, os testes estatísticos consideram um agrupamento de dados ignorando sua ordem temporal. Na maioria das vezes, independente da ordem em que esses dados forem analisados, o resultado da análise será o mesmo. Esses testes são relevantes especialmente quando se deseja comparar quão similares ou diferentes são dois grupos de dados, porém, não são capazes de, sozinhos, revelarem se houve melhoria ou não. Em contrapartida, o controle estatístico de processos combina o rigor da estatística clássica à sensibilidade temporal da melhoria pragmática, onde a ordem temporal dos dados é fator relevante para sua representação e análise. Assim, através da associação de testes estatísticos com análises cronológicas, o controle estatístico de processos habilita a detecção de mudanças e tendências no comportamento dos processos.

Estabilidade e Capacidade dos Processos

Considerando o comportamento dos processos, dois conceitos são importantes: estabilidade e capacidade.

Um processo é considerado estável se o mesmo é repetível. A estabilidade permite prever o desempenho do processo em execuções futuras e, com isso, apóia a elaboração de planos que sejam alcançáveis. Por outro lado, um processo é capaz se ele, além de ser estável, alcança os objetivos e metas da organização e do cliente.

Em relação à estabilidade, é importante destacar que é intrínseco aos processos apresentar variações em seu comportamento. Sendo assim, um processo estável não é um processo que não apresenta variações e, sim, um processo que apresenta variações aceitáveis, que ocorrem dentro de limites previsíveis, que caracterizam a repetitividade de seu comportamento.

As variações aceitáveis são provocadas pelas chamadas causas comuns. Elas provocam desvios dentro de um limite previsto para o comportamento do processo. São variações que pertencem ao processo, ou seja, é o resultado de interações normais dos componentes do processo: pessoas, equipamentos, ambientes e métodos. Considerando processos de software, um exemplo de causa comum pode ser a diferença de produtividade entre os membros de uma equipe relativamente homogênea. Cada membro executa um certo processo com uma determinada produtividade, sendo assim, as variações causadas no comportamento desse processo, por essa diferença de produtividade entre seus executores, são previsíveis.

Por outro lado, as chamadas causas especiais provocam desvios que excedem os limites de variação aceitável para o comportamento do processo, revelando um processo instável. As causas especiais são eventos que não fazem parte do curso normal do processo e provocam mudança no padrão de variação esperado. Considerando processos de software, a inclusão de um membro inexperiente na equipe poderá alterar o comportamento do processo além do esperado, caracterizando uma causa especial. Dificuldades na utilização de um novo aplicativo de apoio à execução do processo também podem levar a variações não previstas, sendo assim outra causa especial.

A estabilização de um processo depende da eliminação de suas causas especiais, ou seja, das causas das variações não aceitáveis para o comportamento do processo. Quando todas as variações no comportamento de um processo são aceitáveis, isto é, quando não há causas especiais, o processo é dito estar sob controle estatístico. Um processo sob controle estatístico é um processo estável.

A partir do momento que um processo torna-se estável, uma baseline que caracteriza o comportamento atual do processo pode ser definida. Esse comportamento descreve o desempenho com o qual as próximas execuções do processo serão comparadas.

Mas, como saber qual é a variação aceitável para o comportamento de um processo?

O comportamento de um processo é descrito através de medidas de produto e/ou de processo. Por exemplo, o comportamento do processo de Inspeção pode ser descrito através da medida densidade de defeitos, definida pela razão entre o número de defeitos detectados em uma inspeção e o tamanho do produto inspecionado . Para cada processo, os limites de variação são calculados aplicando-se métodos estatísticos como, por exemplo, gráficos de controle aos dados coletados para as medidas que o descrevem, que serão melhor descritos mais adiante neste artigo. Esses dados são coletados durante a execução do processo nos projetos da organização. A aplicação do(s) método(s) estatístico(s) adequado(s) determinará, a partir

desses dados, os valores dos limites de variação esperados. Em um processo estável, todos os valores considerados na análise do comportamento estarão dentro desses limites de variação que são uma baseline do desempenho do processo.

Considerando o processo de Inspeção citado anteriormente, suponha que tenha sido aplicado um método estatístico apropriado aos dados coletados para a medida densidade de defeitos, que foram obtidos os limites de variação 7 e 12, e que todos os valores analisados encontravam-se dentro desses limites. Isso quer dizer que os valores 7 e 12 compõem a baseline de desempenho do processo de Inspeção considerando a medida densidade de defeitos, e que esse é o comportamento esperado para o processo quando executado nos projetos da organização. Ao ser executado nos projetos, caso o processo apresente comportamento diferente do estabelecido pelos limites, deve ser feita uma investigação, pois alguma causa especial provocou esse comportamento.

É importante reforçar que a baseline só pode ser estabelecida para processos estáveis. Logo, ainda considerando o exemplo citado para o processo de Inspeção, caso a aplicação dos métodos estatísticos tivesse revelado dados fora dos limites, as causas desses pontos deveriam ser tratadas e o comportamento do processo deveria ser novamente analisado. Essas ações devem se repetir até que todos os dados considerados na análise estejam dentro dos limites estabelecidos, ou seja, até que o processo seja estabilizado. Nesse momento é determinada a baseline que descreve o desempenho previsto para o processo nos projetos da organização.

Uma vez estabilizado, a capacidade do processo deve ser analisada.

A capacidade descreve os limites de resultados que se espera que o processo alcance para atingir os objetivos estabelecidos. Caso o processo não seja capaz, ele deve ser alterado através da realização de ações de melhoria que busquem o alcance da capacidade desejada. Melhorar a capacidade de um processo significa diminuir os limites de variação que são considerados aceitáveis para seu comportamento, ou seja, consiste em tratar as causas comuns.

O processo de Inspeção do exemplo citado apresentou-se estável e com baseline definida pelos limites 7 e 12 para a medida densidade de defeitos. Suponha, agora, que o objetivo estabelecido para esse processo tenha sido apresentar uma densidade de defeitos entre 8 e 10. Nesse caso, apesar de estável, o processo de Inspeção não é capaz de atender ao objetivo para ele estabelecido. Para torná-lo capaz, são necessárias ações incidentes sobre as causas comuns (pessoas, equipamentos, ambiente), a fim de diminuir a variação esperada para o comportamento do processo.

A capacidade do processo é conhecida como voz do processo. Por outro lado, a capacidade desejada para o processo, estabelecida levando-se em consideração os objetivos da organização e do cliente, é chamada de voz do cliente. Obviamente é desejável que a voz do processo atenda a voz do cliente. Porém, isso nem sempre é possível.

É preciso considerar que, algumas vezes, a capacidade desejada para o processo não é possível de ser obtida. Nesse caso, os objetivos estabelecidos devem ser revistos, pois não são realistas. Ou seja, algumas vezes é preciso rever a voz do cliente, considerando a voz do processo.

Bom, então é isso? Uma vez que os processos estão estáveis e são capazes, o papel do controle estatístico de processos se encerra?

Nada disso! Como comentado no início deste artigo, o controle estatístico de processos surgiu para apoiar a implementação de programas de melhoria contínua nos processos da manufatura. Logo, se o controle estatístico de processos apóia melhoria contínua, seu papel não se encerra quando processos capazes são obtidos.

Quando um processo torna-se capaz, um novo ciclo de melhoria pode (e normalmente deve) ser iniciado, estabelecendo-se novos objetivos para que o processo possa ser melhorado continuamente.

Muitas organizações conhecidas pela excelência de seus programas de qualidade estabelecem limites de variação bastante 'estreitos' para os processos e, uma vez que estes são alcançados, a organização obtém processos estáveis, capazes e com um grau de variabilidade consideravelmente baixo.

Quanto menor a variação dos processos, menores são as chances de desvios entre os valores planejados e realizados nos projetos, ou seja, maior é a aderência dos projetos aos cronogramas, orçamentos e demais planejamentos estabelecidos. Consequentemente, melhor será a gerência dos projetos e processos.

Os princípios do controle estatístico de processos são encontrados em outras abordagens da indústria. O Six Sigma, por exemplo, que também teve sua origem na área de manufatura, é uma abordagem para melhoria de processos baseada na medição do desempenho dos processos. Ele foca a melhoria da satisfação do cliente através da prevenção e eliminação de defeitos e, consequentemente, melhoria dos processos organizacionais.

Então, recapitulando: considerando-se os conceitos relacionados ao comportamento de um processo (capacidade e estabilidade), no controle estatístico de processos, a análise do comportamento de um processo pode levar a três direções: (i) remover/tratar as causas especiais para tornar o processo estável; (ii) mudar o processo, quando este é estável, mas incapaz, para que o mesmo torne-se capaz de atender os objetivos do cliente e da organização; e, (iii) melhorar continuamente o processo, quando este é estável e capaz.

Métodos Estatísticos

Existe uma variedade considerável de métodos estatísticos que podem ser utilizados como ferramentas analíticas para representar e analisar os dados coletados para as medidas.

Exemplos de métodos estatísticos são os gráficos de barras, diagramas de tendências, histogramas e gráficos de controle, entre outros. Os gráficos de controle, muito utilizados no controle estatístico de processos, são capazes de medir a variação dos processos e avaliar sua estabilidade. Esses gráficos associam métodos de controle estatístico e representação gráfica

para quantificar o comportamento de processos auxiliando a detectar os sinais de variação no comportamento dos processos e a diferenciá-los dos ruídos. Os ruídos dizem respeito às variações que são aceitáveis e são intrínsecas aos processos (causas comuns). Já os sinais indicam variações que precisam ser analisadas (causas especiais).

Existem diversos tipos de gráficos de controle e cada um deles é melhor aplicável a determinadas situações. O primeiro passo para utilizar um gráfico de controle é selecionar o tipo adequado às medidas, dados e contexto a serem analisados. Em seguida, os dados devem ser plotados e os limites calculados.

A maneira como os dados serão plotados, se serão agrupados e como os limites de controle serão calculados, são definidos pelo tipo de gráfico de controle que será utilizado. Cada tipo possui um conjunto de métodos quantitativos e/ou estatísticos associados.

A representação gráfica facilita a percepção de valores fora dos limites esperados, ou seja, a presença de causas especiais. Porém, as causas especiais nem sempre aparecem fora dos limites, por isso, existem métodos de análise estatística que orientam sobre como identificar pontos que merecem atenção nos gráficos de controle, mesmo quando não estão fora dos limites permitidos à variação.

O layout básico de um gráfico de controle é ilustrado na **Figura 1**. Tanto a linha central quanto os limites superior e inferior representam estimativas que são calculadas a partir de um conjunto de medidas coletadas. Os limites superior e inferior ficam a uma distância de três desvios padrão em relação à linha central. A linha central e os limites não podem ser arbitrários, uma vez que são eles que refletem o comportamento atual do processo. Seus valores são obtidos aplicando-se as expressões e constantes definidas pelo tipo de gráfico de controle a ser utilizado.

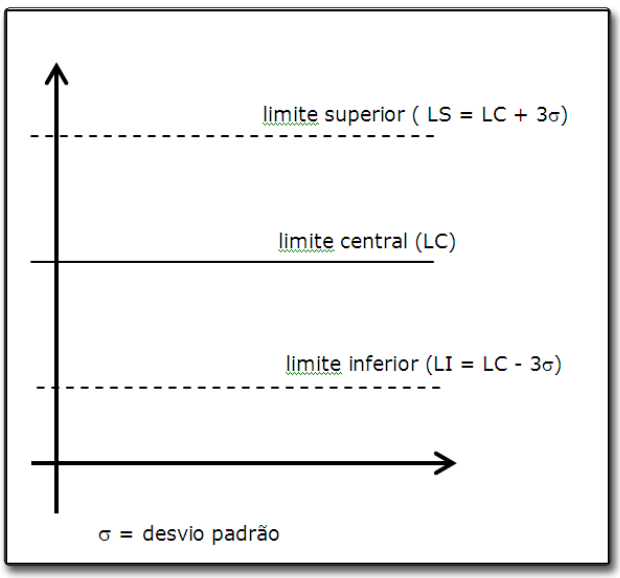


Figura 1. Layout básico de um gráfico de controle.

A **Figura 2** ilustra um exemplo de aplicação de um gráfico de controle para representar as medidas coletadas em um processo estável, ou seja, onde não há causas especiais. O gráfico representa a média diária de horas dedicadas a atividades de suporte por semana em uma determinada empresa.

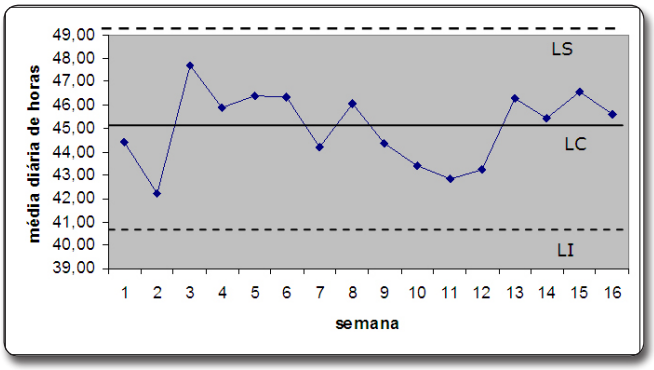


Figura 2. Processo estável.

Na **Figura 3** é apresentado um gráfico que ilustra um processo cujo comportamento extrapolou os limites de variação aceitáveis, sendo identificados pontos cujas causas de variação (causa especial) devem ser investigadas. O gráfico representa o número de problemas relatados pelos clientes diariamente à área de suporte de uma organização que não foram resolvidos (PNR).

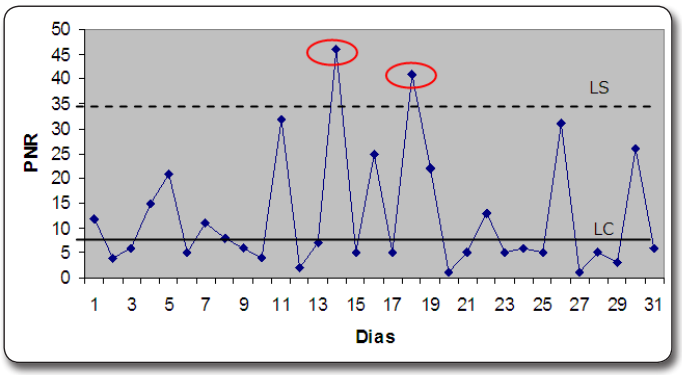


Figura 3. Processo com causas especiais explícitas.

Na **Figura 3**, os pontos que caracterizam a instabilidade do processo estão bastante explícitos, acima do limite superior de variação permitido. Porém, conforme mencionado anteriormente, os pontos de instabilidade, ou seja, aqueles gerados por causas especiais, nem sempre aparecem fora dos limites, existindo outros sinais que revelam a instabilidade de um processo. Para verificar a estabilidade de um processo, quatro testes podem ser aplicados (WHEELER e CHAMBERS, 1992):

Teste 1: presença de algum ponto fora dos limites de controle 3σ.

Teste 2: presença de dois ou três valores sucessivos do mesmo lado a mais de 2σ da linha central (chamada zona C).

Teste 3: presença de quatro ou cinco valores sucessivos do mesmo lado a mais de 1σ da linha central (chamada zona B).

Teste 4: presença de oito pontos sucessivos no mesmo lado da linha central.

A **Figura 4** ilustra um processo hipotético instável que apresenta as situações descritas nos quatro testes de WHEELER e CHAMBERS (1992).

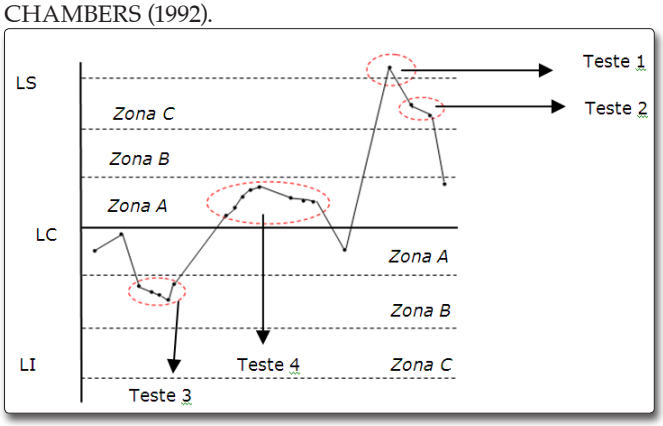


Figura 4. Testes para analisar estabilidade de processos.

Para exemplificar a utilização dos testes de estabilidade, considere o gráfico apresentado na **Figura 2**. Suponha que esse gráfico tenha sido gerado por um gerente que desejava analisar a quantidade de horas semanais dedicadas a atividades de suporte em sua empresa. Para isso, foram registradas as horas de trabalho despendidas com suporte diariamente e, em seguida, foram obtidas as médias diárias de cada semana e o gráfico foi plotado.

Percebendo a ausência de pontos fora dos limites de controle, mas desejando se certificar da aparente estabilidade do processo, o gerente decidiu dividir as distâncias dos limites de controle em três zonas (A, B e C) e aplicar todos os testes de estabilidade. A **Figura 5** apresenta o gráfico com as zonas A, B e C identificadas.

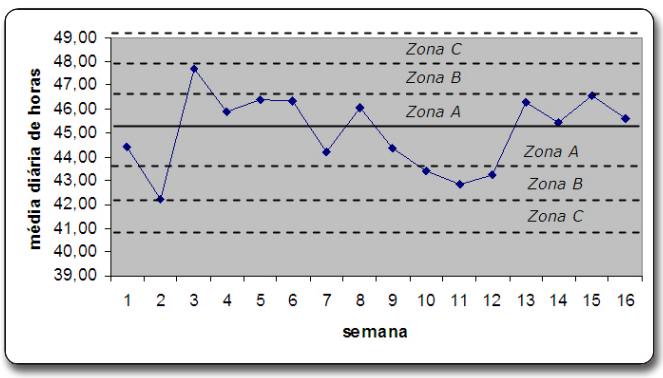


Figura 5. Gráfico com as zonas A, B e C identificadas.

Ao aplicar os testes 1, 2, 3 e 4, percebe-se que o processo realmente é estável.

Vale destacar que nem todos os testes são aplicáveis a todos os tipos de gráficos de controle. Os tipos de gráficos de controle, suas aplicações e particularidades são assuntos para outros artigos.

Cuidados para Realizar Controle Estatístico de Processos de Software

Considerando que a Engenharia de Software é uma área recente, a utilização do controle estatístico de processos em organizações de software pode ser encarada como uma área em seus passos iniciais.

Mas, apesar de ser bastante recente, a utilização do controle estatístico de processos tem sido impulsionada pelos principais modelos e normas de apoio à definição e melhoria de processos que orientam a melhoria de processos através de níveis de maturidade e capacidade. Exemplos notáveis desses modelos e normas de apoio são o MPS.BR - Melhoria de Processo de Software Brasileiro, o CMMI - Capability Maturity Model Integration e a norma internacional ISO/IEC 15504 - Avaliação de Processos de Software. Em todas essas iniciativas, o controle estatístico de processos é utilizado para atender as exigências dos níveis mais altos de maturidade, que incluem gerência quantitativa de projetos e melhoria contínua.

O movimento das organizações em direção à qualidade de software e, especialmente, à melhoria de processos é indiscutível. No Brasil, esse movimento tem recebido cada vez mais adeptos e, inclusive, possui um modelo próprio, o MPS.BR - Melhoria de Processo de Software Brasileiro, que foi elaborado considerando-se as características específicas das empresas brasileiras.

Apesar desses modelos e normas orientarem as organizações sobre o que é necessário realizar para alcançar os níveis mais elevados de maturidade, ainda há lacunas sobre como realizar as ações que levam a esses níveis. Com isso, muitos erros são cometidos durante o percurso dos níveis iniciais aos níveis mais elevados, prejudicando a realização do controle estatístico de processos.

Por exemplo, organizações que, durante a realização das atividades requeridas aos níveis iniciais, onde a medição começa a ser exigida, simplesmente se preocupam em atender os requisitos das áreas de processo, acabam criando um repositório de medidas inúteis ao pensamento estatístico exigido nos níveis superiores. Essas organizações, normalmente, acumulam dados incorretos, capturados em medidas inúteis, definidas sem visar utilização futura; não possuem dados suficientes registrados ou, quando possuem, são registrados de maneira inadequada.

Como conseqüência, essas organizações acabam tendo que despender tempo (e dinheiro!) para arrumar a casa antes de, efetivamente, realizar o controle estatístico de processos. O controle estatístico de processos requer uma fundação, ou seja, processos caracterizados por medidas válidas e dados de qualidade que possam ser utilizados para analisar a previsibilidade dos processos.

Além disso, é preciso conhecer e aplicar corretamente os métodos estatísticos. Mas, admitamos: a utilização desses métodos não é uma tarefa trivial e aplicá-los a processos de software envolve um pouco mais de critério do que aos processos da manufatura.

Sem dúvidas há diversos aplicativos disponíveis no mercado que realizam as operações estatísticas para um conjunto de dados oferecidos como entrada para um determinado método. Alguns exemplos desses aplicativos são Minitab e Excel (considerando a utilização de Macros para apoio ao controle estatístico de processos). Mas isso não é suficiente. Analisar estatisticamente o comportamento de um processo não consiste

apenas em jogar os dados coletados para as medidas em um aplicativo de representação e análise estatística e, como em um passe de mágica, obter os resultados desejados. Os aplicativos, sozinhos, não são capazes de analisar o contexto dos dados utilizados. Essa é uma tarefa que requer intervenção humana e conhecimento organizacional. A utilização dos métodos não apropriados, ou uma interpretação equivocada, podem revelar um comportamento irreal para os processos e, consequentemente, contribuir para o estabelecimento de metas não factíveis.

Então, quer dizer que todo aquele 'papo' de estabilidade, capacidade e melhoria contínua não funciona para processos de software?

Errado! Quer dizer que, para aplicar controle estatístico a processos de software é preciso começar a pensar nele desde as primeiras ações de definição e estabelecimento de um programa de medição. Quanto antes uma organização vislumbrar a realização do controle estatístico de processos menor será o custo de transição da melhoria de processos baseada em medição tradicional para a melhoria de processos através do controle estatístico de processos. Mas, atenção! Pensar no controle estatístico desde o início da implantação da medição não significa medir tudo ou qualquer coisa. É preciso identificar os processos da organização que são mais críticos para o alcance dos objetivos técnicos e de negócio e determinar como medir esses processos corretamente, através de medidas significativas.

E mais, é preciso que os gerentes de projetos sejam capazes de utilizar os métodos estatísticos, entendendo e interpretando sua representação e chegando às conclusões necessárias, considerando o contexto dos dados utilizados. Isso não quer dizer que o gerente precisa ser um expert em estatística. Se fosse assim, seria exigido um estatístico e não um gerente de projetos.

Exemplos de algumas empresas que implantaram programas de melhoria apoiados pelo controle estatístico de processos de software: IBM, Motorola, Hitachi Software Engineering, EDS, Tata Consultancy Services, Unisys, CPM Braxis e BRQ, entre outras.

Conclusão

As exigências cada vez maiores do mercado de software têm levado engenheiros e gerentes de software a atentarem, ainda mais, à necessidade de possuir processos de software maduros, capazes de atender às demandas de qualidade e produtividade.

A aplicação do controle estatístico de processos utiliza os dados coletados ao longo dos projetos para analisar o comportamento dos processos da organização e identificar as ações necessárias para sua melhoria. Além disso, estabelece baselines que refletem o comportamento dos processos nos projetos da organização e apóiam a previsão de seu desempenho em projetos futuros, a fim de que alcancem os objetivos para ele estabelecidos.

O controle estatístico de processos permite quantificar as características dos projetos, produtos e processos de software, representando-as graficamente e propiciando uma análise mais objetiva e eficiente. Dessa forma, o desempenho das atividades que produzem os produtos pode ser previsto, controlado e guiado para alcançar os objetivos técnicos e de negócio.

E vale a pena quantificar?

Como disse Galileu Galilei: "meça o que é mensurável e torne mensurável o que não é". ●

Referências

CMMI – Capability Maturity Model Integration, 2006, SEI – Software Engineering Institute, CMU - Carnegie Mellon University, v 1.2.

Fenton, N., Marsh W., Neil, M., Cates, P., Forey, S., Tailor, M.:, 2004, "Making Resource Decisions for Software Projects", Proceedings of the 26th International Conference on Software Engineering - ICSE'04, pp. 397-406.

Florac, W. A., Carleton, A. D., 1999, Measuring the Software Process: Statistical Process Control for Software Process Improvement, Addison Wesley.

ISO/IEC, ISO/IEC TR 15504-2003, 2003, Information Technology – Software Process Assessment, International Organization for Standardization and the International Electrotechnical Commission, Geneva, Switzerland.

Kitchenhan, B., Kutay, C., Jeffery, R., Connaughton, C., 2006, "Lessons Learnet from the Analysis of Large-scale Corporate Databases", Proceedings of the 28th International Conference on Software Engineering – ICSE'06, pp. 439-444.

Komuro, M., 2006, "Experiences of Applying SPC Techniques to Software Development", Proceedings of the 28th International Conference on Software Engineering - ICSE'06, pp. 577-584.

MPS.BR – Melhoria de Processo do Software Brasileiro, Guia Geral, v. 1.2, 2007.

MPS.BR – Melhoria de Processo do Software Brasileiro, Guia de Implementação – Parte 6: Nível B - v. 1.0, 2007.

Sargut, K. U., Demirors, O., 2006, "Utilization of Statistical Process Control (SPC) in Emergent Software Organizations: Pitfalls and Suggestions", Software Quality Journal, pp. 135-157.

Wheeler, D. J., Chambers, D. S., 1992, "Understanding Statistical Process Control", 2nd ed. Knoxville, SPC Press.

Conhecimento faz diferença!

Mais de 60 mil downloads na primeira edição!

Faça já sua assinatura digital! | *www.devmedia.com.br/es*

Faça um *up grade* em sua carreira.

Em um mercado cada vez mais focado em qualidade ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lança para os desenvolvedores brasileiros sua primeira revista digital totalmente especializada em Engenharia de Software. Todos os meses você irá encontrar artigos sobre Metodologias Ageis; Metodologias tradicionais (*document driven*); ALM (*application lifecycle management*); SOA (aplicações orientadas a servicos); Analise de sistemas; modelagem; Métricas; orientação à objetos; UML; testes e muito mais. **Assine já!**

DevMedia GROUP