engenharia de software magazine

DevMedia group Ano I - Edição 06

ISSN 1983127-7

Projeto
Saiba como Analisar o Código de sua Aplicação de Forma Visual

Projeto
Conceitos Introdutórios sobre Arquitetura de Software

TESTE DE SOFTWARE

Veja como elaborar casos de teste a partir da especificação de casos de uso

- **Processo**
  Conceitos Introdutórios sobre Melhoria e Avaliação de Processos de Software
- **Planejamento**
  Conheça os Conceitos Base da Gerência de Projetos
- **Validação, Verificação & Teste**
  Utilizando o framework EasyMock para teste unitário de aplicações Java

Aulas desta edição:

- Introdução à Engenharia de Requisitos - Parte 19
- Introdução à Engenharia de Requisitos - Parte 20
- Introdução à Engenharia de Requisitos - Parte 21
- Introdução à Engenharia de Requisitos - Parte 22
- Introdução à Construção de Diagrama de Classes da UML – Parte 2
- Introdução à Construção de Diagrama de Classes da UML – Parte 3
- Introdução à Construção de Diagrama de Classes da UML – Parte 4
- Introdução à Construção de Diagrama de Classes da UML – Parte 5

Projeto :.
Veja como Avaliar a Usabilidade de suas Aplicações

Ferramenta CASE :.
Conheça a mPrime, Ferramenta de Apoio à Gerência de Riscos

Gestão de Conhecimento :.
Entenda seus Fundamentos e como pode estar Estruturada

engenharia de software magazine

Ano 1 - 6ª Edição 2008 - Impresso no Brasil

Corpo Editorial

Colaboradores
Rodrigo Oliveira Spínola
rodrigo@sqlmagazine.com.br

Marco Antônio Pereira Araújo
Eduardo Oliveira Spínola

Editor de Arte
Vinicius O. Andrade
viniciusoandrade@gmail.com

Diagramação
Roberta F. Leal Arman
roberta.arman@gmail.com

Revisão
Gregory Monteiro
gregory@clubedelphi.net

Na Web
www.devmedia.com.br/esmag

Apoio

Parceiros:

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 Colaborador das revistas Engenharia de Software Magazine, Java Magazine e SQL Magazine. 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).

EDITORIAL

"Se observarmos nos diferentes livros tradicionais de Engenharia de Software, veremos que sempre existe um capítulo ou seção destinado a Teste de software. Porém, eles normalmente apresentam apenas informações básicas sobre esta atividade, como por exemplo, os diferentes níveis de teste que podem ser aplicado, as técnicas de teste que podem ser aplicadas e os critérios para seleção dos testes associados a estas técnicas. Por exemplo, no artigo "Introdução a Teste de Software" publicado na edição 01 da Engenharia de Software Magazine discutimos sobre alguns desses critérios: Particionamento em classes de equivalência, Análise do Valor Limite e Grafo de Causa-Efeito. Para cada critério, vimos como aplicá-los e um exemplo da sua aplicação para a geração de casos de teste."

"No entanto, no desenvolvimento de um software real normalmente os problemas são bem mais complexos do que aqueles tradicionalmente usados quando estamos conhecendo esses critérios para seleção dos testes (ex: indicar qual o maior valor em um conjunto, indicar se um campo número só contém caracteres válidos, dentre outros). Normalmente os problemas a serem resolvidos são representados através de cenários, que podem ser facilmente representados por Diagramas de Casos de Uso da UML (www.uml.org) aliada a uma descrição do que cada caso de uso deve fazer."

Neste contexto, a Engenharia de Software Magazine destaca nesta edição uma matéria muito interessante sobre a elaboração de casos de teste. Será apresentada uma possível estratégia indicando como testes podem ser obtidos a partir dos casos de uso especificados para um projeto.

Além destas duas matérias, esta sexta edição traz mais seis artigos:

- Testes com Objetos Mock: Utilizando o framework EasyMock para teste unitário de aplicações Java;
- Utilizando Visualização de Informação para Compreensão de Software;
- Conceitos Introdutórios sobre Melhoria e Avaliação de Processos de Software;
- Fundamentos de Arquitetura de Software;
- Inspecionando a Usabilidade de Produtos;
- Gerenciamento de Projetos: Entenda alguns dos principais conceitos;
- Soluções para Gerenciamento de Riscos de Projetos.

Desejamos uma ótima leitura!

Equipe Editorial Engenharia de Software Magazine

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

Caro Leitor,
Para esta sexta edição, temos um conjunto de 8 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 abaixo:

01 – Engenharia de Requisitos
Título: Introdução à Engenharia de Requisitos - Parte 19
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula é parte do curso de introdução à engenharia de requisitos. Nesta décima nona parte apresentaremos passo a passo a especificação de um caso de uso considerando o cenário de consulta por clientes cadastrados de uma organização fictícia**.**

02 – Engenharia de Requisitos
Título: Introdução à Engenharia de Requisitos - Parte 20
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula é parte do curso de introdução à engenharia de requisitos. Nesta vigésima parte finalizaremos a especificação do caso de uso iniciada na aula anterior.

03 – Engenharia de Requisitos
Título: Introdução à Engenharia de Requisitos - Parte 21
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula é parte do curso de introdução à engenharia de requisitos. Nesta vigésima primeira parte apresentaremos passo a passo a especificação de um caso de uso considerando o cenário de inclusão de cliente para uma organização fictícia.

04 – Engenharia de Requisitos
Título: Introdução à Engenharia de Requisitos - Parte 22
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula é parte do curso de introdução à engenharia de requisitos. Nesta vigésima segunda parte finalizaremos a especificação do caso de uso iniciada na aula anterior.

05 – Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 2
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula apresenta a notação da UML para elaboração de diagramas de classes considerando os conceitos de classe, atributo e operação.

06 – Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 3
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula apresenta a notação da UML para elaboração de diagramas de classes considerando os relacionamentos de herança e associação.

07 – Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 4
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula apresenta a notação da UML para elaboração de diagramas de classes considerando os relacionamentos de agregação e composição.

08 – Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 5
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula apresenta uma heurística para elaboração de diagramas de classes. Para isto, são apresentados alguns passos que podem ser seguidos para a apoiar a construção passo a passos destes diagramas.

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

ÍNDICE

Cursos Online

Assinatura java magazine PLUS
Mais conteúdo JAVA por muito menos!

A revista **Java Magazine** oferece para seus assinantes uma série de **Cursos Online** de alto padrão de qualidade .
Conheça abaixo o curso já disponível.

Curso em destaque

Introdução ao desenvolvimento para celulares com J2ME

Confira neste curso os principais recursos do J2ME. Aprenda também com 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 **Java Magazine** e comece já seu treinamento!
www.devmedia.com.br/assine

A sua melhor opção de aprendizagem!

Outros cursos disponíveis: **www.devmedia.com.br/curso**

DevMedia Mais Informações: www.devmedia.com.br/central - Tel.: 2220-5375 / 2220-5435

Conceitos Introdutórios sobre Melhoria e Avaliação de Processos de Software

De que se trata o artigo:

Apesar da crescente demanda por software em praticamente todas as áreas do conhecimento, o processo de produção continua sendo um esforço coletivo, criativo e complexo, por isso, precisa ser disciplinado, acompanhado e controlado de forma a se tornar efetivo e eficiente para a organização. O foco no processo permite que um grupo de indivíduos alinhe o comportamento e as atividades de cada membro no sentido de alcançar o objetivo comum. Assim, acredita-se que a qualidade do produto final está fortemente relacionada à qualidade do processo utilizado para o seu desenvolvimento e manutenção. Quando um produto possui algum problema, não se deve corrigir somente o defeito encontrado. É necessário corrigir o processo que permitiu que este fosse inserido, pois, desta forma, não será necessário corrigir os mesmos problemas em trabalhos futuros. Com isto em mente, este artigo apresenta de forma abrangente o assunto melhoria de processo de software.

Para que serve:

Estabelecer boas práticas para facilitar os trabalhos envolvidos na melhoria de processos de software.

Em que situação o tema é útil:

Empresas que estão em busca de excelência no desenvolvimento de software possuem como uma de suas alternativas o trabalho fundamento em processos e sua melhoria contínua.

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 Engenharia de Software Magazine.

A melhoria do processo de software pode ser considerada hoje uma das grandes prioridades para as organizações que trabalham com software. Isto se deve à exigência do mercado por produtos com maior qualidade, que sejam entregues mais rapidamente e com menor custo de desenvolvimento.

Estudos apontam que ao tentarem melhorar seus processos, as empresas estão em busca de:

• entender as características dos processos existentes e os fatores que afetam a sua capacidade;

• planejar, justificar e implementar ações que modificarão os processos, tornando-os mais coerentes com as necessidades de negócios e;

• avaliar os impactos e benefícios ganhos, comparando-os com os custos advindos das mudanças realizadas.

Neste contexto de melhoria de processo, é importante destacar uma das atividades de maior importância: a avaliação dos processos utilizados durante a execução dos projetos.

Com o objetivo de apoiar a melhoria de processo, diversos métodos surgiram ao longo dos últimos anos. Alguns métodos avaliam os processos da organização tomando como base algum modelo de referência, que descreve um conjunto de princípios e práticas e assume que, se devidamente seguidas, irão levar a melhores produtos de software. Outros métodos utilizam as medições para entender e avaliar os processos em uso e, somente então, tomar ações que levem à melhoria do processo.

Neste artigo apresentaremos alguns conceitos relacionados a processos de software e alguns dos principais métodos de avaliação de processo atualmente utilizados para apoiar a melhoria do processo.

Processo de Software

Podemos encontrar na literatura técnica diversas definições para processo de software:

• HUMPHREY (1989) define processo como um conjunto de atividades, métodos e práticas utilizadas na produção e no desenvolvimento de software;

• FLORAC et al. (1997) definem como uma organização lógica de pessoas, materiais, energia, equipamentos e procedimentos empregados na execução de atividades projetadas para produzir um resultado específico;

• Para FUGGETTA (2000), um processo de software é definido como um conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos e artefatos que são necessários para conceber, desenvolver, disponibilizar e manter um produto de software;

• E, finalmente, a ISO/IEC 12207 (1995) define como um conjunto de atividades inter-relacionadas, que transforma entradas em saídas.

Neste sentido, é importante destacar o trabalho da International Standard Organization (ISO) que estabeleceu uma norma padrão para processo de software ISO/IEC 12207 (1995) propondo um framework com terminologia bem definida e contendo processos, atividades e tarefas que devem ser aplicados durante a aquisição, o fornecimento, o desenvolvimento, a operação e a manutenção de software. A norma descreve a arquitetura de um processo de forma geral, mas não especifica em detalhes como implementar ou desempenhar estas atividades, nem descreve formato ou conteúdo da documentação a ser gerada, o que deve ser definido pela organização que pretende utilizá-lo de acordo com suas necessidades e as características particulares de cada projeto.

Outro conceito muito importante para conhecermos neste momento é o de Maturidade do Processo de Software. Este teve sua origem em esforços do Software Engineering Institute (SEI) ao atender uma solicitação da Força Aérea Americana que necessitava de um método para avaliar a capacidade em desenvolver software das organizações que lhe prestavam serviços terceirizados. PAULK et al. (1995) definiram capacidade como o intervalo de resultados esperados que podem ser alcançados com o uso de um processo, e maturidade como a amplitude na qual um processo específico é definido, gerenciado, medido, controlado e executado.

O resultado do trabalho do SEI (HUMPHREY, 1989) representa a base de diversos outros modelos e normas com o objetivo de aumentar a maturidade dos processos de software.

Avaliação de Processo de Software

As avaliações de processo de software são realizadas para atender a diferentes objetivos, geralmente estão delimitadas a diferentes escopos e, a depender das características dos modelos e métodos aplicados, ainda são classificadas como pertencendo a diferentes paradigmas.

Uma vez que este conjunto de características podem afetar de forma diferenciada uma avaliação de processo, existem também na literatura diferentes definições para avaliação de processo:

• Para ZAHRAN (1997), uma avaliação de processo de software é um exame disciplinado do processo de software utilizado pela organização, baseado em um modelo de processo. O objetivo é determinar o nível de maturidade desses processos. O resultado deve identificar e caracterizar as práticas correntes, identificando áreas de força e fraqueza e a eficácia das práticas atuais em controlar ou evitar as principais causas de baixa qualidade, custo e cronograma ultrapassados. Os resultados de uma avaliação também podem ser usados como um indicador da capacidade desses processos em alcançar os objetivos do desenvolvimento de software em relação à qualidade, custo e cronograma com um alto grau de predição.

• Segundo HUMPHREY (1989), uma avaliação do processo de software é um exame aplicado a uma organização que desenvolve software com o objetivo de advertir seus gerentes e profissionais a respeito de como melhorar as suas operações.

• De acordo com a definição da ISO/IEC 12207 (1995), uma avaliação é uma determinação sistemática do grau de atendimento de uma entidade em relação aos critérios para ela estabelecidos.

Neste cenário, KAN (2003) considerou alguns aspectos importantes para as avaliações de processos:

Contexto da avaliação

Uma avaliação de processo pode ser realizada em diferentes contextos, dependendo de quem irá desempenhar os papeis essenciais durante a avaliação. Dessa forma, uma avaliação pode ocorrer:

• internamente, quando é realizada uma auto-avaliação onde os principais

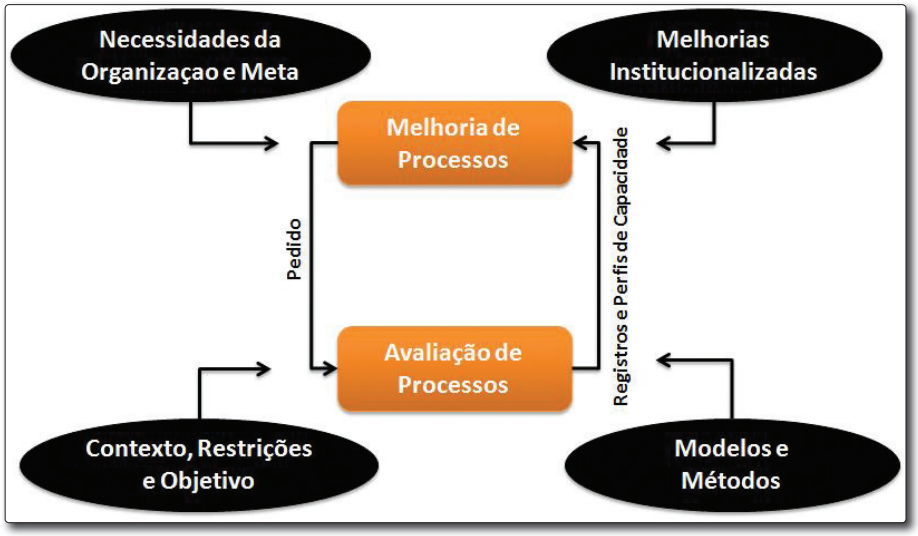


Figura 1. Melhoria de processo com a norma ISO/IEC 15504 (ISO, 1998).

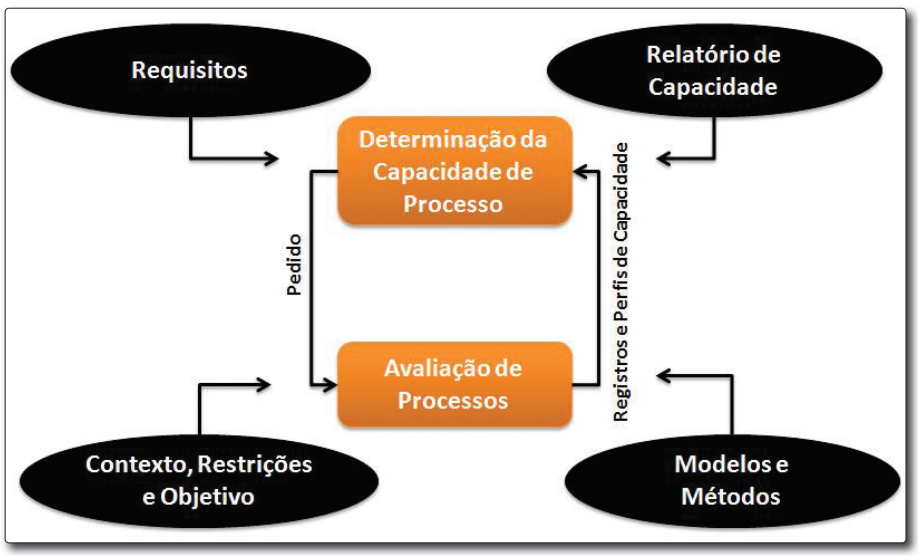


Figura 2. Determinando a capacidade através do uso da ISO 15504 (ISO, 1998).

papéis são desempenhados por uma equipe que pertence à própria organização sendo avaliada;

• externamente, sendo realizada por uma equipe de avaliação externa à organização;

• ou ainda pode ser realizada por terceiros, quando um fornecedor é avaliado por uma equipe externa para que seja averiguada a sua capacidade em atender aos requisitos da organização contratante.

Objetivo da Avaliação

Geralmente, uma avaliação de processo é realizada para atender a dois objetivos: a melhoria dos processos e a determinação da capacidade dos processos de uma organização.

A **Figura 1** mostra como a norma ISO/IEC 15504 (1998) é utilizada para a melhoria de processo. De acordo com a norma, a organização deve definir os objetivos e o contexto, escolher o modelo e o método para a avaliação e definir os objetivos de melhoria (ROCHA et al., 2001).

No segundo caso, determinar a capacidade dos processos de uma organização, o objetivo é avaliar um fornecedor em potencial para obter seu perfil de capacidade. A **Figura 2** mostra como a ISO/IEC 15504 (1998) é utilizada para determinar a capacidade de processos. De acordo com a norma, a organização deve definir os objetivos e o contexto da avaliação, os modelos e métodos de avaliação e os requisitos esperados (ROCHA et al., 2001).

Escopo da Avaliação

O escopo de uma avaliação do processo de software pode cobrir todos os processos da organização, um subconjunto selecionado dos processos ou um projeto específico (KAN , 2003). Para a maioria das avaliações de processo baseadas nos conceitos de maturidade ou capacidade (por exemplo, CMMI, Bootstrap, ISSO/IEC 15504, MR MPS), a unidade de análise é normalmente o nível organizacional.

Quando o alvo da avaliação é a organização, os resultados de uma avaliação de processo podem ser diferentes, mesmo com sucessivas aplicações do mesmo método. Isso acontece pelo fato que, em grandes empresas, varias definições de organização são possíveis e o escopo da avaliação pode ser diferente em avaliações sucessivas. Outra fonte de variação é a amostragem de projetos escolhida para representar a organização; isso pode afetar o escopo e os resultados.

Quando a unidade de avaliação é apenas um projeto, os problemas associados com a avaliação a nível organizacional deixam de ser relevantes. Uma avaliação de projeto de software deve incluir todos os fatores significativos que contribuem para o sucesso ou falha de um projeto. As avaliações de projeto tratam, em profundidade, não somente "quais" atividades foram realizadas, mas também do "como" e "por que" foram realizadas. Dessa forma, a investigação exaustiva é uma característica chave de avaliação de projeto de software.

A avaliação de processo baseada em maturidade de processo torna-se relevante quando uma organização tem a intenção de embarcar em uma estratégia geral de melhoria a longo prazo. Porém, os dois tipos de avaliação podem ser complementares: a avaliação da maturidade do processo para uma estratégia geral de melhoria para a organização e avaliações de projeto para direcionar ações de melhoria imediatas e específicas no nível de projeto.

Abordagens de avaliação

Vários modelos, métodos e técnicas de melhoria estão disponíveis e podem ser divididos em duas grandes vertentes:

• A abordagem top-down, que é fortemente baseada em avaliações e benchmarking. São os casos do CMM (PAULK et al., 1993), ISO/IEC 15504 (2003), o BOOTSTRAP (KUVAJA, 1994), CMMI (CMU/SEI, 2002) e do MR mpb (Sociedade SOFTEX, 2004a) (Sociedade SOFTEX, 2004b).

• A abordagem bottom-up, que utiliza principalmente a medição como o guia para a melhoria de processo. Por exemplo, o GQM (BASILI et al., 1994).

Na abordagem top-down, normalmente se aplica um modelo normativo que é assumido como a melhor maneira de se desenvolver software. Avaliando uma organização utilizando-se este modelo, torna-se possível identificar a maturidade desta organização e propor melhorias relevantes. Já a abordagem bottom-up é baseada na análise cuidadosa das práticas de processo aplicadas, na seleção de objetivos de melhoria derivados dessa análise e na gerência de atividades de melhoria apoiadas por medições.

Modelos de Avaliação de Processo de Software

A partir de agora serão apresentados alguns modelos de apoio à melhoria de processo e como é realizada a avaliação em cada um deles.

CMMI

Desde a década de 90, baseado no sucesso alcançado pelo SW-CMM (CMM para software), um número significativo de modelos de maturidade de processo foi desenvolvido para diferentes disciplinas. Assim surgiram os seguintes modelos (CMU/SEI, 2002):

• Software Acquisition CMM (AS-CMM) – usado para avaliar a maturidade de uma organização em seus processos de seleção, compra e instalação de software desenvolvido por terceiros;

• Systems Enginnering CMM (SE-CMM) – usado para avaliar a maturidade da organização em seus processos de engenharia de sistemas, incluindo o hardware, o software e quaisquer outros elementos que participam do produto completo;

• Integrated Product Development CMM (IPD-CMM) – ainda mais abrangente que o SE-CMM, inclui também outros processos necessários à produção e suporte ao produto, tais como suporte ao usuário, processos de fabricação, etc;

• People CMM (P-CMM) – usado para avaliar a maturidade da organização em seus processos de administração de recursos humanos no que se refere a software: recrutamento e seleção de desenvolvedores, treinamento e desenvolvimento, remuneração, etc.

Apesar dos modelos serem úteis para muitas organizações, o uso de múltiplos modelos gerou alguns problemas devido às diferenças de arquitetura, conteúdo e abordagem. Além disso, a aplicação de diversos modelos não integrados em uma organização aumenta os custos de treinamento, das avaliações e das atividades de melhoria.

Por estas razões, o SEI iniciou o projeto do CMMI (CMM Integration), com o objetivo de integrar as práticas de forma que organizações que almejem melhorar seus processos nas diferentes disciplinas tenham a disposição um único modelo consistente.

Sendo assim, o CMMI integra os diversos CMMs numa estrutura única, todos com a mesma terminologia, processos de avaliação e estrutura. O projeto também se preocupou em tornar o CMM compatível com a norma ISO/IEC 15504, de modo que avaliações em um modelo sejam reconhecidas como equivalentes aos do outro (CMU/SEI, 2002).

Para permitir esta compatibilidade, o CMMI oferece duas representações diferentes para a sua abordagem de melhoria de processos. Estas duas representações são conhecidas como o "modelo contínuo" e o "modelo em estágios". A representação em estágios define um conjunto de áreas de processo para definir um caminho de melhoria para a organização, descrito em termos de níveis de maturidade (melhoria vertical). A representação contínua permite que uma organização selecione uma área de processo específica e melhore com relação a esta área. A representação contínua usa níveis de capacidade para caracterizar a melhoria relacionada a uma área de processo específica.

Ambas as representações contêm essencialmente as mesmas informações e a opção pelo modelo contínuo ou em estágios depende de cada organização. Cada modelo possui características que o tornam mais apropriado em uma situação ou outra (CMU/SEI, 2002).

O modelo em estágios oferece um caminho para melhoria de processos, indicando a ordem de implementação para cada área de processo de acordo com os níveis de maturidade. Essa abordagem minimiza os riscos da melhoria de processos. A representação é indicada para organizações realmente comprometidas com a implantação do CMMI em escala geral.

Nível de Maturidade	Foco	Áreas de Processo
Inicial	Sem foco, processos são ad hoc e caóticos.	Não há áreas de processo neste nível.
Gerencial	O foco está na gerência de projeto.	• Gerência de requisitos • Planejamento de projetos • Monitoração e controle de projetos • Gerência de acordos com fornecedores • Medição e análise • Garantia da qualidade do processo e do produto • Gerência de configuração
Definido	O foco está na institucionalização do processo.	• Desenvolvimento de requisitos • Solução técnica • Integração do produto • Verificação • Validação • Foco no processo organizacional • Definição do processo organizacional • Gerência integrada do produto • Gerência de riscos • Análise de decisão e resolução • Ambiente organizacional para integração (IPPD) • Equipe integrada (IPPD)
Gerência Quantitativa	O foco está na gerência quantitativa.	• Desempenho do processo organizacional • Gerência quantitativa de projeto
Otimizado	O foco está na melhoria contínua do processo.	• Inovação e disseminação organizacional • Análise e resolução de causas

Tabela 1. Níveis de maturidade do CMMI.

O modelo em estágios avalia uma organização como estando nos níveis de maturidade de processo apresentados na **Tabela 1**.

O modelo contínuo oferece uma abordagem mais flexível para a melhoria de processos, embora mais complexo de administrar. É indicado para organizações que desejam dar prioridade à melhoria de uma área de processo ou conjunto de processos, de acordo com seus objetivos de negócio. Este modelo permite fácil comparação à ISO/IEC 15504, porque a organização das áreas de processo é derivada desta norma.

Quando a representação contínua é utilizada numa avaliação, uma área de processo é avaliada como estando em um determinado nível de capacidade. Existem seis níveis de capacidade, numerados de zero a cinco. Para uma área de processo atingir determinado nível de capacidade, os objetivos específicos e, conseqüentemente, as práticas específicas destes objetivos devem ser satisfeitas:

- No nível de capacidade 0 (Incompleto), a área de processo não é realizada ou é parcialmente realizada;
- Uma área de processo alcança o nível 1 de capacidade (Realizado) quando está sendo realizada, ou mais precisamente, quando os objetivos específicos da área de processo são alcançados;
- Alcançando o nível 2 de capacidade (Gerenciado), a área de processo necessita que seu desempenho esteja sendo gerenciado. Diferente do nível 1, uma área de processo no nível 2 dispõe de um plano para a sua realização, assim como um processo concebido para cobrir esta área de processo;
- No nível 3 de capacidade (Definido), a área de processo está sob o controle de um processo padrão organizacional para a área de processo e este pode ser adaptado para necessidades específicas;
- No nível 4 de capacidade (Gerenciado quantitativamente), a área de processo é gerenciada quantitativamente utilizando-se de técnicas estatísticas e outras técnicas quantitativas;
- Ao atingir o nível 5 de capacidade (Otimizado), a área de processo é gerenciada quantitativamente (capacidade nível 4) e alterada e adaptada para adequar-se aos objetivos de negócio da empresa.

Avaliação CMMI

O método de avaliação CMMI padrão para melhoria de processo chama-se SCAMPI. Ele foi desenvolvido para satisfazer os requisitos do modelo CMMI (CMU/SEI, 2002). A avaliação segundo o SCAMPI consiste de três fases: planejamento e preparação, condução de uma avaliação no local de trabalho e a apresentação dos resultados.

Para o planejamento e preparação, as seguintes atividades devem ser realizadas:

- Identificar o escopo da avaliação – onde ocorre o levantamento das necessidades de negócio da unidade organizacional sendo avaliada;
- Desenvolver o plano da avaliação – onde ficam registrados os requisitos do plano de avaliação, acordos, estimativas, riscos, métodos de adaptação e considerações práticas associadas à avaliação;
- Selecionar e preparar a equipe de avaliação – uma equipe treinada, experiente e apropriadamente qualificada é selecionada para conduzir o processo de avaliação;
- Obter e analisar as evidências iniciais - obtém informações que identifiquem áreas potencialmente problemáticas ou falhas na implementação das práticas;
- Preparar para a coleta de evidências – consiste em planejar e documentar a coleta de dados incluindo as fontes de dados, ferramentas e técnicas a serem usadas e contingências para gerenciar o risco da falta de dados.

Para conduzir uma avaliação no local de trabalho, as seguintes atividades devem ser realizadas:

- Examinar as evidências – que compreende coletar as informações a respeito das práticas implementadas na unidade organizacional e relacionar os dados coletados ao modelo de referência;
- Verificar e validar as evidências – consiste em verificar a implementação das práticas nas unidades organizacionais para cada instanciação e validar os resultados da implementação descrevendo as falhas na implementação das práticas do modelo;
- Documentar as evidências – registra as informações obtidas identificando e consolidando os dados e transformados-os em registros que documentem a implementação das práticas, assim como suas forças e fraquezas;
- Gerar os resultados da apresentação - Mede a satisfação dos objetivos baseado na extensão da implementação da prática através da unidade organizacional. A extensão da implementação da prática é determinada baseada nos dados validados, coletados de toda a amostra das unidades organizacionais.

Quanto à apresentação dos resultados, as seguintes atividades são realizadas:

- Apresentar os resultados da avaliação – Provê resultados da avaliação que podem ser usados para guiar ações de melhoria. As forças e as fraquezas dos processos em uso também são apresentadas. Além disso, determina, se planejado, qual o nível de capacidade ou o nível de maturidade dos processos em uso.
- Empacotar e arquivar os resultados da avaliação – guarda registros e dados importantes da avaliação e disponibiliza o material selecionado de maneira apropriada.

ISO/IEC 15504

A ISO iniciou, em janeiro de 1993, o projeto SPICE (Software Process Improvement and Capability dEtermination) com o objetivo de produzir inicialmente um Relatório Técnico que fosse mais geral e abrangente que os modelos existentes e mais específico que a norma ISO 9001. Uma versão do SPICE foi aprovada em 1998 como Relatório Técnico (ISO/IEC TR 15504, 1998) e, apenas em 2003, a Norma ISO/IEC 15504 (ISO/IEC 15504, 2003) foi publicada.

A ISO/IEC 15504 pode ser utilizada para a melhoria de processos e para a determinação da capacidade de processos de uma organização. Quando o objetivo da organização for a melhoria de processos, pode-se avaliá-los, gerando um perfil dos processos a ser utilizado na elaboração de um plano de melhorias. A análise dos resultados identifica os pontos fortes e fracos e os riscos inerentes aos processos. Já quando o objetivo da empresa for avaliar fornecedores para contratação, esta pode obter seus perfis de capacidade.

O modelo de referência da ISO/IEC 15504 (2003) define a dimensão de processo, que corresponde à definição de um conjunto de processos considerados universais e fundamentais para a boa prática da engenharia de software e a dimensão de capacidade, que corresponde à definição de um modelo de medição com base na identificação de um conjunto de atributos que permite determinar a capacidade de um processo para atingir seus propósitos, gerando os produtos de trabalho e os resultados estabelecidos.

Na dimensão de capacidade, as tarefas, atividades e práticas, bem como as características dos produtos de trabalho, são definidas como indicadores que demonstram se determinado processo é adequadamente praticado em um determinado nível de capacidade. Há seis níveis de capacidade em que um processo pode ser avaliado:

- Nível 0 - Processo incompleto: O processo não é implementado ou falha na consecução de seu propósito. Não existe evidência de que os produtos de trabalho sejam adequadamente produzidos ou que os resultados sejam alcançados;
- Nível 1 - Processo executado: O processo implementado alcança seu propósito, mas sua execução talvez não seja rigorosamente planejada e acompanhada;
- Nível 2 - Processo gerenciado: O processo executado anteriormente é agora implementado de forma gerenciada (planejado, monitorado e ajustado) e seus produtos de trabalho são apropriadamente estabelecidos, controlados e mantidos;
- Nível 3 - Processo estabelecido: O processo gerenciado anteriormente é agora implementado utilizando um processo definido e é capaz de alcançar seus resultados de processo;
- Nível 4 - Processo previsível: O processo estabelecido anteriormente opera agora dentro de limites para alcançar os resultados de processo;
- Nível 5 - Processo otimizado: O processo previsível anteriormente é melhorado continuamente para satisfazer os objetivos de negócio atuais e projetados mais relevantes.

Avaliação ISO/IEC 15504

Uma avaliação segundo a norma ISO/IEC 15504 (2003) considera três tipos de elementos como importantes para sua realização: um modelo de avaliação; um método de avaliação; um ou mais avaliadores competentes.

Essa norma define um conjunto de requisitos para um Modelo de Avaliação e para um Método de Avaliação. Uma avaliação que esteja de acordo com estes requisitos é referenciada como uma avaliação em conformidade com a avaliação ISO/IEC 15504 (2003).

Ela não define um método de avaliação explícito, definindo apenas os requisitos necessários. Isto significa que as empresas podem desenvolver os seus próprios métodos de avaliação em conformidade com a ISO/IEC 15504 (2003).

GQM

O GQM representa uma abordagem sistemática para adaptar e integrar objetivos de negócio aos modelos de processo de software baseando-se em necessidades específicas de um projeto ou de uma organização (BASILI et al., 1994).

O resultado da aplicação do método do GQM é a especificação de um programa de medição que tem como objetivo investigar determinados assuntos, e um conjunto de regras para interpretar as medidas coletadas.

Dentro de um contexto de avaliação do processo de software, o GQM pode ser utilizado para estabelecer um programa de medição que possibilite investigar o desempenho de determinados processos, tornando-se uma abordagem bastante eficaz para a monitoração e o controle dos processos.

O princípio por trás do método GQM é que as medições sejam orientadas por objetivos. Dessa forma, tanto para avaliar quanto para melhorar seus processos, as organizações devem definir seus objetivos de medição, baseados nos seus objetivos de negócio e transformar esses objetivos em atividades que podem ser medidas durante a execução do projeto.

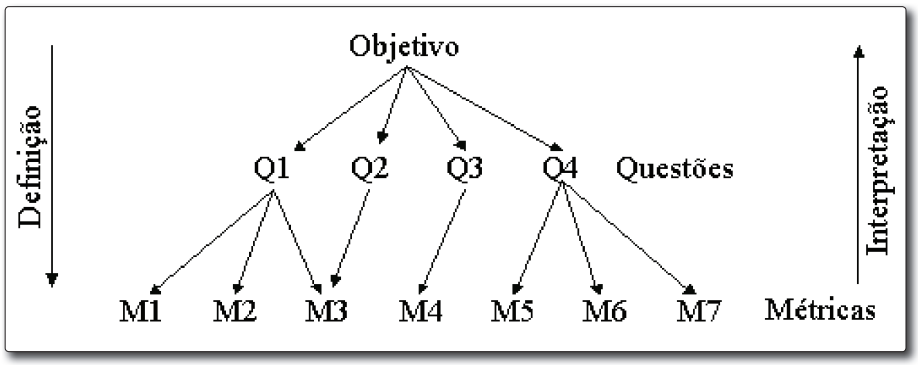


Figura 3. O paradigma GQM (BASILI e WEISS, 1984).

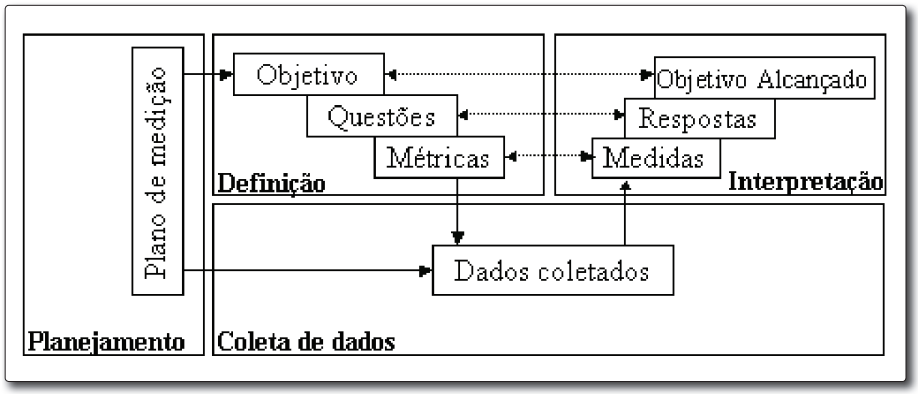


Figura 4. As quatro fases do método GQM (SOLINGER e BERGHOUT, 1999).

Referências

BASILI, V. R., WEISS, D., 1984, "A Methodology for Collecting Valid Software Engineering Data", IEEE Transactions on Software Engineering, Vol. 10, No. 3, Nov, pp. 728-738.

BASILI, V. R., CALDIERA, G., ROMBACH, H.D., 1994, Goal Question Metric Paradigm, Encyclopedia of Software Engineering, 2 Volume Set, John Wiley & Sons, Inc.w

CMU/SEI, 2001, Standard CMMI Appraisal Method for Process Improvement (SCAMPI), Version1.1: Method Definition Document, CMU/SEI-2001-HB-001, Pittsburgh, Software Engineering Institute, Carnegie Mellon University. URL: http://www.sei.cmu.edu

CMU/SEI, 2002, Capability Maturity Model Integration (CMMI), Version 1.1 CMMI for Software Engineering (CMMI-SW, V1.1), Pittsburgh, Software Engineering Institute, Carnegie Mellon University. URL: http://www.sei.cmu.edu

FLORAC, W.A., PARK, R.E., CARLETON, A.D., 1997, Practical Software Measurement: Measuring for Process Management and Improvement, CMU/SEI-97-HB-003, Pittsburgh, Software Engineering Institute, Carnegie Mellon University.

HUMPHREY, W.S. 1989, Managing the Software Process, Addison-Wesley.

ISO/IEC 12207, 1995, Information Technology – Software Life-Cycle Processes.

ISO/IEC PDAM 12207, 2002, "ISO/IEC 12207 Information Technology – Amendment to ISO/IEC 12207", Montreal: ISO/IEC JTC1 SC7.

ISO/IEC TR 15504, 1998, Information technology – Software Process Assessment.

ISO/IEC 15504, 2003, Information Technology – Software Process Assessment, International Standard Organization.

KAN, S. H., 2003, "Metrics and Models in Software Quality Engineering", Second Edition, Addison-Wesley.

KUVAJA, P. et al., 1994, Software Process Assessment and Improvement: The BOOTSTRAP Approach, Oxford, Blackwell Publishers.

PAULK, M. C., WEBER, C. V., CURTIS, B., CHRISSIS, M. B. (eds), 1995, The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley.

ROCHA, A.R., MALDONADO, J.C., WEBER, K.C., 2001, Qualidade de Software – Teoria e Prática, 1a ed., Prentice Hall, São Paulo.

Sociedade SOFTEX, 2004a, "Uma Estratégia para Melhoria de Processo de Software nas Empresas Brasileiras", http://www.softex.br/media/QuaTIC.zip.

Sociedade SOFTEX, 2004b, "Modelo de Referência para Melhoria de Processo de Software: uma abordagem brasileira.", http://www.softex.br/media/artigoCLEI_versao_final.pdf. Acessado em 02/2005.

SOLINGEN, R., BERGHOUT, E., 1999, The Goal/Question/Metric Method: A Practical Guide for Quality Improvement of Software Development, McGrawHill, 1999.

O GQM define um determinado objetivo, refina este objetivo em questões e define métricas que devem propiciar informações que respondam a estas questões.

Respondendo às questões, os dados medidos definem o objetivo operacionalmente e podem ser analisados para identificar se os objetivos foram ou não alcançados. O GQM define as métricas em uma perspectiva top-down e analisa e interpreta os dados medidos numa perspectiva bottom-up, como mostrado na **Figura 3**.

O método GQM é composto de quatro fases (SOLINGEN e BERGHOUT, 1999). Na fase de planejamento, os requisitos básicos para tornar o programa de medição viável são executados, incluindo treinamento, envolvimento da gerência e planejamento do projeto. A fase de definição identifica os objetivos e as questões e métricas associadas a cada objetivo. Durante a fase de coleta de dados os formulários de coleta são definidos, preenchidos e armazenados na base de medições. Durante a fase de interpretação as medidas são utilizadas para responder as questões formuladas, e essas questões são, então, utilizadas novamente para verificar se os objetivos declarados foram atingidos.

As quatro fases do método GQM são mostradas na **Figura 4**.

Considerações Finais

Várias abordagens associadas à melhoria de processo têm ganhado importância na comunidade de software. Os conceitos, métodos, e práticas englobam uma maneira de pensar, de agir e de entender os dados gerados pelos processos que, coletivamente, resultam em melhoria da qualidade, aumento da produtividade e competitividade dos produtos de software.

Vimos neste artigo alguns dos conceitos que norteiam a área assim como uma breve descrição de algumas abordagens para avaliação de processos. ●

Profissional de TI

Conheça o **IBM Rational Team Concert,** a ferramenta que permite a criação de aplicações de negócio de forma rápida e eficaz, acelerando o seu ciclo de desenvolvimento.

Disponível em 3 versões, o **IBM Rational Team Concert** é **gratuíto** para até 3 usuários (versão Community Edition).

Para mais informações, entre em contato conosco e solicite um DVD com conteúdo exclusivo!

PRIMEUP

IBM Business Partner

Endereço:
R. Marquês de São Vicente, 225
Instituto Gênesis, Sala 27B
PUC-Rio, Gávea

Tel: (21) 2512-6005

atendimento@primeup.com.br
www.primeup.com.br

Gerenciamento de Projetos

Entenda alguns dos principais conceitos

De que se trata o artigo:

Nesse artigo foram tratados os conceitos do Gerenciamento de Projetos, a fim de desmistificar o assunto e dar sustentação à decisão de aplicação dos seus conceitos.

Para que serve:

Esclarece a base do Gerenciamento de Projetos e suas ramificações, servindo como ponto de partida para a sensibilização das organizações e profissionais para a aplicação dos conceitos em seu dia a dia.

Em que situação o tema é útil:

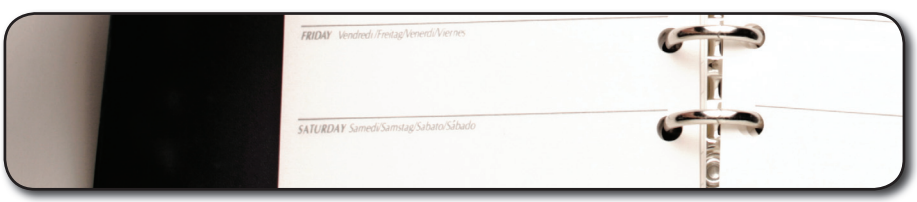
A aplicação dos conceitos de Gerenciamento de Projetos auxilia na gestão de projetos de qualquer tipo e tamanho, aumentando os níveis de qualidade dos produtos ou serviços produzidos e ajudando a atingir os objetivos do projeto.

Neste artigo falaremos um pouco sobre Gerência de Projetos "GP", um assunto que pode parecer novo para alguns por estar sendo foco de muitas discussões atualmente, principalmente dentro das organizações, que têm buscado nas técnicas e conceitos da GP uma forma de reduzir as falhas em seus projetos. Entretanto, como ciência formal, a GP já tem quase meio século e se formos pensar como conceito são alguns milhares de anos, é só lembrarmos da perfeita construção das pirâmides do Egito, da muralha da china e outros grandes projetos que envolveram um número grande de trabalhadores e tiveram êxito.

Andrey Abreu

andreyabreu@gmail.com

Pós graduando em engenharia de software pela universidade GAMA FILHO, graduado em gestão estratégica de organizações pela UNISUL, Gerente de TI da empresa NEXXERA TECNOLOGIA S/A, gerencia atualmente área de Desenvolvimento de Sistemas, composta por 33 profissionais divididos em equipes de desenvolvimento Java e c, que trabalham local e remotamente em 15 linhas de projetos.



Até hoje engenheiros renomados ainda se perguntam como foi possível a construção, por exemplo, das pirâmides, um grande projeto com tamanha precisão e perfeição sem utilização das modernas técnicas de planejamento, construção e controle. Esses fatos nos mostram que muito do que precisamos para ter êxito em nossos projetos já está pronto, já foi testado e funciona, não precisamos recriar novas técnicas ou conceitos, apenas entender e utilizar os conceitos e técnicas existentes da melhor forma possível e com certeza os resultados serão melhores.

Mas o que realmente é um projeto?

Todos nós de forma intrínseca fazemos projetos no decorrer de nossas vidas, seja uma viajem planejada, a manutenção preventiva do carro, um roteiro de férias, tudo isso são pequenos projetos que planejamos e executamos sozinhos. Porém, quando esses projetos começam a envolver mais pessoas é necessária uma organização maior para que o objetivo de todo o grupo seja atingido, e é nesse ponto que entra a Gerência de Projetos.

Segundo o PMBOK (Project Management Body of Knowledge) (2004, p.21), Guia do conjunto de conhecimentos em Gerência de Projetos definido pelo PMI (Project Management Institute), "Um projeto é um esforço temporário, executado por pessoas, restringido por recursos limitados, planejado, executado e controlado, e empreendido para criar um produto, serviço ou resultado exclusivo". Temporário por que cada projeto tem seu início e fim muito bem definidos, chegando-se ao fim quando os objetivos foram alcançados ou quando está claro que não serão ou não poderão mais ser alcançados.

Percebemos então que é necessário definir objetivos, realizar um planejamento de execução, especificar custos, estipular a quantidade de pessoas envolvidas e elaborar um cronograma, delimitando assim a previsão de início e término para produzir o resultado desejado no projeto.

Agora que sabemos o que é um projeto dentro do contexto apresentado, vamos entrar na conceituação do Gerenciamento de Projetos.

O que é Gerência de Projetos?

Segundo o PMBOK (2004, p.6), "Gerência de Projetos é a aplicação de conhecimentos, habilidades, ferramentas e técnicas, a fim de satisfazer ou exceder as necessidades e expectativas dos stakeholders (interessados e envolvidos)".

Satisfazer as necessidades e expectativas dos stakeholders envolve além de tudo equilibrar demandas concorrentes em relação a:

- Escopo, prazo, custo e qualidade;
- Stakeholders com necessidades e expectativas diferenciadas;
- Requisitos identificados (necessidades) e requisitos não identificados (expectativas).

E esse é o desafio da gerência de projetos, alinhar as expectativas e necessidades dos clientes com a realidade do projeto, gerando resultado sem prejudicar a qualidade e mantendo todos os atores envolvidos informados.

Para isso, a Gerência de Projetos utiliza-se de seus processos componentes que podem ser classificados em cinco grupos de processos (iniciação, planejamento, execução, controle e encerramento) e de suas áreas de conhecimento, ao todo nove (gerência de integração, gerência de escopo, gerência de tempo, gerência de custo, gerência de qualidade, gerência de RH, gerência de comunicação, gerência de riscos e gerência de aquisições).

Em resumo, a Gerência de Projetos visa manter os riscos de fracasso em um nível tão baixo quanto necessário durante todo o ciclo de vida do projeto.

A História da Gerência de Projetos

Como ciência foi formalizada na década de 60, quando os negócios e organizações começaram a enxergar o benefício do trabalho organizado em torno de projetos e a entender a necessidade crítica para integrar o trabalho através de múltiplos departamentos e profissões.

Em 1969, no auge dos projetos espaciais da NASA, um grupo de 5 profissionais de gestão de projetos reuniu-se para discutir melhores práticas e técnicas, e foi fundado o Project Management Institute, PMI (EUA) por Jim Snyder.

O PMI é atualmente a maior instituição internacional dedicada à disseminação do conhecimento e aprimoramento das atividades de gestão profissional de projetos e está espalhado por diversos países através de seus grupos disseminadores. Pelos números do PMI (posição jan/2006), passam de 212 mil os membros e são mais de 176 mil profissionais certificados (PMP's, "Project Management Professional") em 160 países.

Áreas de Conhecimento da Gerência de Projetos

Como já comentado no anteriormente, a gerência de projetos é composta por nove áreas de conhecimento, que são a base para estruturação de um projeto de sucesso. A **Figura 1** ilustra a interação entre essas áreas.

Falaremos a partir de agora um pouco sobre cada uma dessas áreas, mas não entraremos nesse artigo no detalhamento

Figura 1. Áreas de conhecimento da gerência de projetos.

de cada uma. O PMBOK (2004, p.71-295) traz com detalhes cada uma das áreas, seus procedimentos, artefatos, entradas e saídas.

Gerenciamento de Integração

Inclui os processos necessários para garantir que os elementos do projeto estão coordenados de maneira apropriada, principalmente no que tange a harmonização das disciplinas centrais (escopo, qualidade, tempo e custo) fazendo compensações entre objetivos e alternativas concorrentes, a fim de atingir ou superar as necessidades e expectativas dos Stakeholders.

Processos envolvidos:

• **Termo de abertura do projeto:** Autorização formal DE um projeto ou uma fase.

• **Declaração do escopo preliminar:** Descrição em alto nível o escopo do projeto.

• **Plano de gerenciamento do projeto:** Listagem das ações de definição, preparação, integração e coordenação, agregadas aos resultados de todos os demais processos, compondo um plano de gerenciamento do projeto.

• **Execução do plano de projeto:** Execução das ações definidas no plano de gerenciamento do projeto a fim de atender aos requisitos definidos na declaração do escopo.

• **Monitorar e controlar o trabalho do projeto:** Monitoramento e verificação do andamento da execução das ações do plano de gerenciamento do projeto.

• **Controle integrado de mudanças:** Coordenação das mudanças de projeto e acompanhamento da aprovação e entrega.

• **Encerrar projeto:** Encerramento formal do projeto ou de uma fase.

Gerenciamento do Escopo

Composto por processos que garantem que o projeto contemple todo o trabalho exigido e somente o trabalho exigido, controlando o que está ou não incluído no projeto, a fim de que o mesmo seja completado com sucesso.

Processos envolvidos:

• **Planejamento do Escopo:** Documentação de como o escopo do projeto será definido, verificado e controlado.

• **Definição do Escopo:** Declaração detalhada do escopo do projeto, que servirá como base para futuras decisões de projeto.

• **Criação da estrutura analítica do projeto (EAP):** Divisão das entregas em partes menores a fim de facilitar o gerenciamento.

• **Verificação do Escopo:** Formalização das entregas do projeto finalizadas, no final do projeto, no final da fase do projeto, ou na liberação das principais entregas.

• **Controle do Escopo:** Garante que mudanças sejam acordadas com todos, determina e gerencia quando uma mudança ocorre e com que freqüência o escopo pode mudar.

Gerenciamento do Tempo

Contém os processos relativos ao controle do término do projeto dentro do prazo previsto, garantindo o cumprimento dos prazos definidos em um cronograma de atividades. É considerada a área de maior exigência dentro do projeto por ser a que é mais visível em sua gestão.

Processos envolvidos:

• **Definição das atividades:** Identificação das atividades dentro do cronograma que precisam ser realizadas para gerar as entregas.

• **Seqüenciamento das atividades:** Identificação das dependências entre atividades no cronograma.

• **Estimativa de recursos das atividades:** Estimativa de recursos (tipos e quantidades) requeridos para a execução de cada atividade do cronograma.

• **Estimativa de duração das atividades:** Estimativa individual do período de trabalho necessário para conclusão de cada atividade do cronograma.

• **Criação do Cronograma:** Análise da seqüência de atividades, dependências, duração e recursos requeridos para a confecção do cronograma.

• **Controle do cronograma:** Controle das possíveis mudanças do cronograma.

Gerenciamento de Custos

Descreve os processos necessários para garantir que o projeto será concluído dentro do orçamento previamente aprovado. Custos e escopo estão fortemente relacionados, uma vez que um escopo mal definido implicará diretamente nas estimativas de custos do projeto.

Processos envolvidos:

• **Estimativa:** Descrição da estimativa de custos relativa à alocação de recursos para execução do projeto.

• **Orçamentação:** Agregação dos custos estimados para estabelecer uma linha base dos custos totais, a fim de servir para a medição de desempenho do projeto.

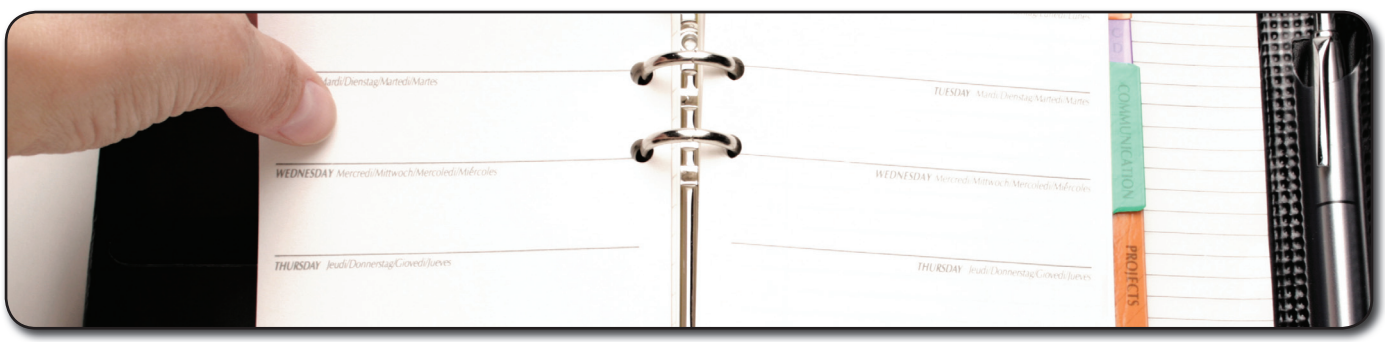
• **Controle de custos:** Controle das variações e mudanças no orçamento e identificação das causas dessas variações seja positiva ou negativa, sendo que a gestão inapropriada pode causar problemas de qualidade e cronograma, e elevar o nível de risco.

Gerenciamento da Qualidade

Objetiva garantir a conclusão do projeto dentro dos níveis desejados de qualidade, garantindo a satisfação de todos os envolvidos no projeto.

Principais Dimensões:

• **Satisfação do cliente:** O projeto deve produzir o que se propôs a produzir e o



produto satisfazer as necessidades reais do cliente.

• **Prevenção de erros**: O cliente sempre é o próximo elemento no processo, o custo da prevenção é menor que o da correção.

• **Responsabilidades:** Todos são responsáveis pelo sucesso do projeto, porém a gerência deve fornecer os recursos necessários para que exista o sucesso.

• **Melhoria contínua:** O mundo está em constante mudança, exigindo o aprimoramento constante dos mecanismos de controle de projetos a fim de garantir a qualidade do produto ou serviço.

Processos envolvidos:

• **Planejamento da qualidade:** Identificação/definição dos padrões de qualidade para o projeto e descrição de como satisfazê-los.

• **Garantia da qualidade:** Aplicação das atividades de qualidade a fim de garantir que serão empregados todos os processos necessários para atender aos requisitos dentro dos níveis exigidos de qualidade.

• **Controle da qualidade:** Monitoramento / Acompanhamento dos resultados do projeto, baseando-se nos padrões estabelecidos de qualidade, garantindo que os mesmos estão sendo satisfeitos e identificando formas de eliminar possíveis resultados insatisfatórios.

Gerenciamento de Recursos Humanos

Compreende organizar e gerenciar a equipe do projeto, essa composta por pessoas com funções e responsabilidades atribuídas e claramente definidas, possibilitando o uso mais efetivo das pessoas envolvidas no projeto.

Processos envolvidos:

• **Planejamento de recursos humanos:** Identificação / definição das pessoas e suas atribuições dentro do projeto, tendo como resultado o plano de gerenciamento de pessoal.

• **Contratar / Mobilizar a equipe do projeto:** Efetiva obtenção dos recursos humanos necessários para conclusão do projeto.

• **Desenvolvimento da equipe:** Trabalhar a melhoria de competências e interação entre membros, a fim de melhorar continuamente o desempenho do projeto.

• **Gerenciamento da equipe do projeto:** Trabalhar / Acompanhar o desempenho individual dos membros da equipe, dar e obter feedbacks, tratar problemas rotineiros e coordenar mudanças, objetivando aumentar o desempenho do projeto.

Gerenciamento das Comunicações

Visa gerar, coletar, distribuir, armazenar e recuperar as informações sobre o projeto de forma oportuna e adequada. Toda a equipe do projeto deve entender que as comunicações afetam o projeto como um todo.

Processos envolvidos:

• **Planejamento das comunicações**: Determina as necessidades de informações e comunicações às partes interessadas no projeto.

• **Distribuição das informações:** Divulgar às partes interessadas as informações necessárias.

• **Relatório de desempenho:** Confecção / Divulgação de relatório de desempenho (andamento do projeto, progresso, previsão).

• **Gerenciamento das partes interessadas:** Gerenciamento junto às partes interessadas quanto à satisfação de seus requisitos, bem como o gerenciamento de problemas do projeto.

Gerenciamento de Riscos

Objetiva aumentar a probabilidade de eventos positivos e diminuir a probabilidade de eventos negativos no projeto, tratando os processos de identificação, análise, resposta, monitoramento, controle e planejamento de riscos.

Processos envolvidos:

• **Planejamento do gerenciamento de riscos:** Definição de como tratar, planejar e executar as atividades de risco do projeto.

• **Identificação de riscos:** Identificação / Documentação dos riscos que podem afetar o projeto.

• **Análise qualitativa de riscos:** Priorização dos riscos do projeto, levando em consideração a probabilidade dos mesmos ocorrerem e sua freqüência.

• **Análise quantitativa de riscos:** Análise do efeito dos eventos de risco e atribuição de classificação numérica a esses riscos, realizada sobre os riscos levantados na análise qualitativa.

• **Planejamento de resposta de risco:** Criação de opções e ações com o intuito de aumentar as oportunidades e reduzir as vulnerabilidades dos objetivos do projeto.

• **Monitoramento e controle de riscos:** Acompanhamento dos riscos identificados, monitoramento dos riscos restantes, identificação de novos riscos, execução / avaliação do plano de resposta de riscos. Este processo é efetuado durante todo o ciclo de vida do projeto.

Gerenciamento de Aquisições

Trata do processo de aquisição de bens, produtos e serviços de fornecedores externos à organização, visando dar condições de realização do projeto.

Processos envolvidos:

• **Planejamento de compras e aquisições:** Definição do que, como e quando comprar.

• **Plano de contratações:** Levantamento e discriminação dos produtos, serviços e identificação de possíveis fornecedores.

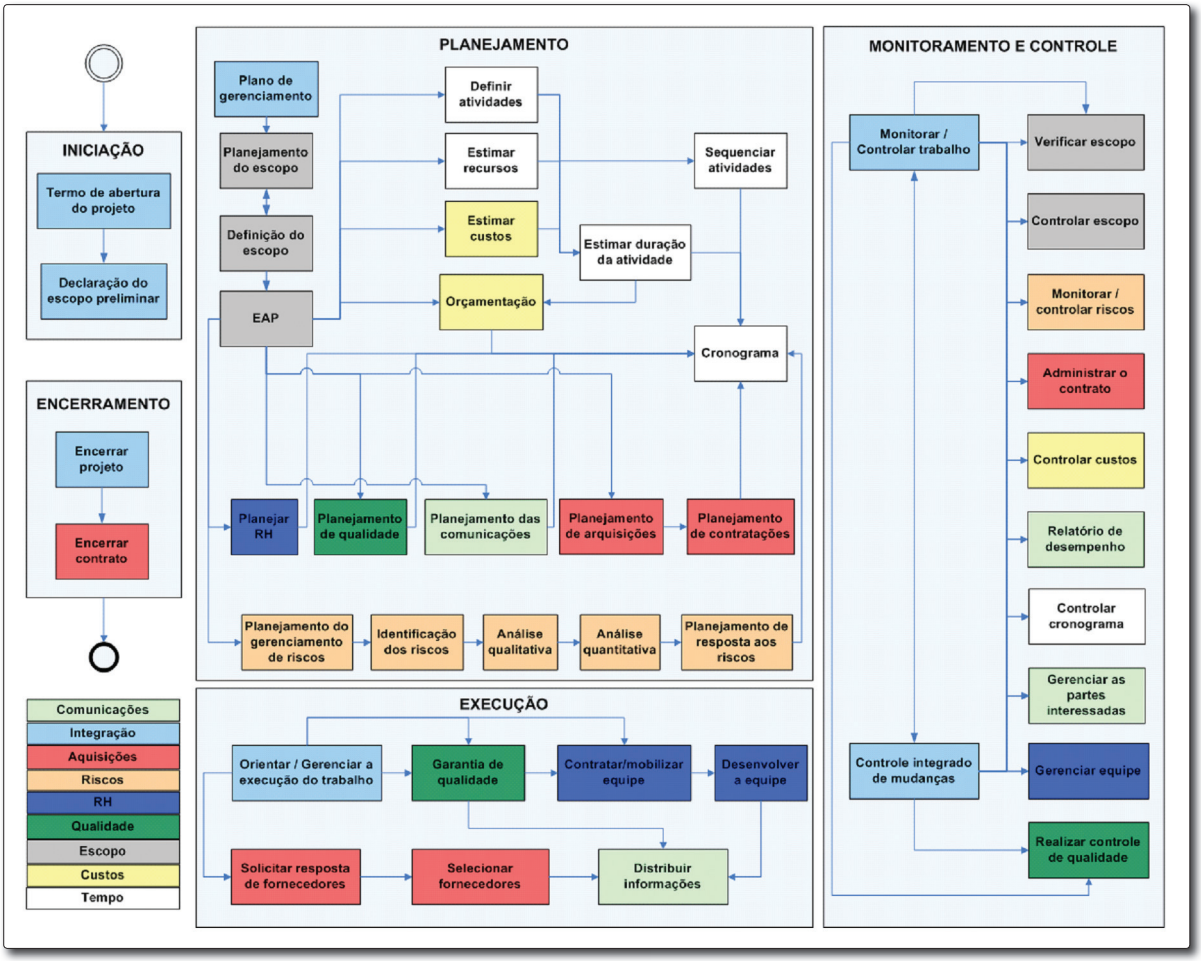
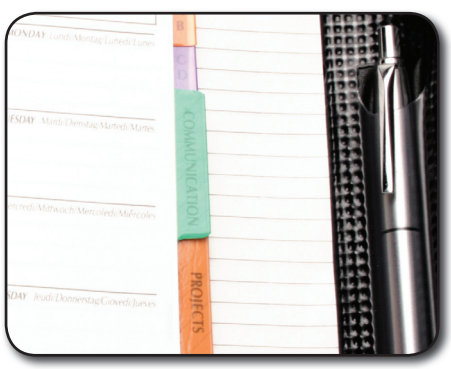


Figura 2. Atividades dos processos da gerência de projetos.

Figura 3. Triângulo do Gerenciamento de Projetos.



- **Solicitação de resposta dos fornecedores:** Obter informações gerais, cotações, preços e ofertas.
- **Seleção dos fornecedores:** Análise e escolha de possíveis fornecedores, negociação e confecção de um contrato com cada fornecedor individualmente.
- **Manutenção do contrato:** Gerenciamento das relações entre comprador e fornecedor, bem como das cláusulas constantes no contrato entre as partes e análise de desempenho do fornecedor para contratações futuras e manutenção das contratações atuais.
- **Encerramento do contrato:** Finalizar cada contrato, liquidando todos os itens pendentes.

Grupo de Processos da Gerência de Projetos

Como falamos anteriormente, a gerência de projetos possui cinco grupos de processos que agrupam as atividades das áreas de conhecimento vistas acima em fases bem definidas e complementares:

- **Iniciação:** Define e autoriza formalmente o início de um projeto ou fase, delimitando o escopo e objetivos preliminares;
- **Planejamento:** Define de forma detalhada os objetivos e propicia o planejamento das ações necessárias para que o projeto seja realizado com sucesso;
- **Execução:** Integra recursos e pessoas a fim de realizar o planejamento do projeto com sucesso;
- **Monitoramento / Controle:** Monitora / Mede os resultados do projeto a fim de identificar problemas e solucioná-los gerando o mínimo de impacto no resultado;
- **Encerramento:** Finaliza formalmente o projeto ou fase, englobando a aceitação do produto / serviço e encerramento formal das atividades.

Para esclarecer melhor, a **Figura 2** apresenta o diagrama da interação das atividades apresentadas neste artigo no contexto destes processos.

Os processos interagem entre si através de seus artefatos de entrada, que são processados com o uso de ferramentas e técnicas; e que geram os artefatos de saída que normalmente serão entradas para outros processos, transformando decisões, planos e reações em condições de progresso.

Variáveis do Gerenciamento de Projetos

Principalmente no que se refere a software, as execuções e entregas devem levar em conta algumas variáveis que podem impactar diretamente sobre o resultado final do projeto. Em gerência de projetos temos o que chamamos de Triângulo do Gerenciamento de Projetos, composto pelas variáveis, escopo, tempo e custo, onde cada lado é representado por uma dessas variáveis, sendo que a mudança de um dos lados impacta diretamente nos demais (ver **Figura 3**). Na visão de alguns profissionais, a qualidade é externa ao escopo, logo poderia ser o quarto lado, porém a qualidade não é um fator e sim o centro do triângulo, ou seja, o resultado do que você faz com o custo, tempo e escopo, e é diretamente afetada pelas decisões tomadas.

Esse triângulo está freqüentemente em conflito, pois a mudança em um de seus fatores impactará diretamente nos demais. Dessa forma, o objetivo é conciliá-los para obter o resultado esperado.

Tendo em vista que se o escopo muda, o tempo e custo são diretamente impactados, se o tempo é reduzido por conta de uma expectativa do *stakeholder*, os custos aumentam e o escopo será reduzido, e se o custo está restrito a um orçamento prévio e reduzido, o tempo será maior e o escopo menor. Como fazer para conciliar esses fatores?

O caminho está em escolher qual lado do triângulo é fixo no projeto, ou seja, aquele lado que não poderá ser alterado em hipótese alguma, um prazo pré-definido pelo *stakeholder* por conta de uma particularidade do seu negócio, um orçamento limitado para atendimento da demanda ou um escopo fixo e acordado que não mudará até a entrega. Seja qual for o fator, é importante definir o que não pode mudar e isso o ajudará a identificar no momento de um problema se o lado definido como fixo está em risco, e dessa forma, o que deverá ser feito para que o projeto volte à linha natural e atinja os resultados esperados.

Com essa premissa podemos assumir o controle sobre os impactos do projeto e tomar decisões mais consistentes e embasadas tendo sempre como objetivo a satisfação das expectativas do cliente e o sucesso do projeto.

O Gerente de Projetos

Agora que já temos uma noção do que é gerenciamento de projetos e como podemos usá-lo em nossos projetos e obter resultados, vamos focar um pouco no papel que garante que todos esses processos aconteçam e toma as decisões para que o mesmo atinja seus objetivos.

Estamos falando do Gerente de Projeto. Esse profissional é responsável por gerenciar o progresso do projeto, utilizando como linha base as variáveis citadas anteriormente (qualidade, custo, prazo e escopo) através da verificação de seus desvios, tendo como principal objetivo a minimização das possibilidades de falha do projeto.

A profissão de Gerente de Projetos pode existir em qualquer ramo de atividade, podendo-se alocar um Gerente de Projetos para gerenciar projetos de todas as espécies, sejam esses, na área de TI, construção civil, montagem de automóveis, ou qualquer outra. As habilidades e experiência necessárias para desempenhar esse papel dependem diretamente do tamanho, da complexidade do projeto e de conhecimentos técnicos da área em questão, uma vez que cada área tem suas particularidades específicas, necessitando assim de uma estruturação diferente para cada situação.

Gerenciar um projeto não é simplesmente controlar cronogramas e atividades para que as mesmas não atrasem, é um pouco mais complexo que isso, passa por identificar claramente as necessidades do cliente, estabelecer objetivos claros e com possibilidade de serem alcançados e balancear os conflitos entre custo, prazo e escopo que afetam diretamente na qualidade, além de gerenciar as incertezas do projeto, principalmente no que tange a ocorrência de riscos que precisarão ser avaliados e tratados. Um projeto é considerado de alta qualidade quando é entregue dentro do escopo, no prazo e dentro do orçamento estabelecidos e esse é o desafio do Gerente de Projetos.

Conclusão

O objetivo desse artigo foi desmistificar o assunto Gerência de Projetos sobre o ponto de vista de que essa é uma área de grande importância para todo e qualquer negócio. Para isso, apresentamos alguns dos conceitos básicos da área. ●

Referências

PMBOK 2004: Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos – terceira edição - ISBN: 1-930699-74-3
PMI: www.pmi.org
PMI Brasil: www.pmi.org.br

Utilizando Visualização de Informação para Compreensão de Software

De que se trata o artigo:

Este artigo apresenta como a visualização de informação em conjunto com as métricas de código fonte podem ajudar no processo de compreensão do software.

Para que serve:

A cada dia, com o contínuo aumento na complexidade dos sistemas de software, o processo para análise e compreensão do código visando evolução e manutenção do sistema se torna mais complexo e demorado. Aplicando os conceitos discutidos neste artigo, podemos tornar estas etapas menos difíceis e trabalhosas.

Em que situação o tema é útil:

Atualmente, existem diversas ferramentas que auxiliam a modelagem de sistemas, o controle de versões, testes, que possibilitam o desenvolvimento em grupo, mas ainda não é comum o uso de ferramentas que facilitem o processo de compreensão do software. Talvez por isso, ainda seja comum a existência de código duplicado, com alta complexidade, entre outros pontos negativos que reduzem a qualidade do sistema desenvolvido. A partir deste ponto, percebemos a importância da compreensão do software nas etapas de desenvolvimento e manutenção.

É cada vez mais evidente a importância de se ter controle sobre o modo como um software está sendo desenvolvido. Quando as primeiras linguagens de programação foram criadas, considerando aspectos como tamanho e complexidade dos sistemas desenvolvidos e o conhecimento sobre técnicas de programação, percebeu-se que os desenvolvedores não possuíam uma metodologia para estruturação de seu código. Possivelmente, os desenvolvedores não imaginavam uma forma bem definida na qual fosse possível analisar os códigos escritos sem ter que, necessariamente, olhar as centenas, ou até mesmo milhares de linhas de código (uma tarefa difícil e custosa). Além disso, retirar informações úteis e precisas dessas linhas com objetivo de dar continuidade ou manutenção ao sistema não era uma atividade trivial.

Embora muito tempo tenha se passado e muita evolução tenha ocorrido no desenvolvimento de técnicas para apoiar o

Eduardo Oliveira Spínola
eduspinola@gmail.com
É colaborador das revistas Engenharia de Software Magazine, Java Magazine e SQL Magazine. 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).

desenvolvimento de sistemas, a tarefa de análise de código ainda é um campo com tópicos a serem explorados. Neste contexto, apresentaremos neste artigo o que se tem estudado sobre compreensão de código utilizando técnicas de visualização de informação, métricas e mineração visual de dados e ao final comentaremos sobre um plug-in de visualização desenvolvido para o Eclipse.

Visualização de Informação

Os seres humanos possuem dificuldades em processar dados no formato de textos e tabelas; por isso, freqüentemente, recorremos a meios visuais para interpretar o mundo a nossa volta [1]. Imagine por exemplo, dois mapas geográficos apresentando o número de habitantes (dado em análise) de cada país. No mapa 1, apresentamos sobre cada país o número de habitantes. No mapa 2, utilizamos cores com tonalidades diferentes para indicar o dado em análise, onde, a cor mais escura representa o país com maior população. Considerando que seu interesse seja estudar os países com população entre 50 e 100 milhões, qual mapa você utilizaria para obter esta informação de maneira mais rápida e segura?

O objetivo da visualização de informação é possibilitar que dados sejam apresentados através de formas simples e intuitivas [2].

A grande preocupação da visualização de informação é como os dados serão apresentados, pois uma boa escolha da forma de apresentação resulta na facilidade do entendimento e possibilita a descoberta de (novas) informações. Voltando ao exemplo do mapa, é possível que você "descubra" rapidamente, através do mapa que utiliza cores, que a região com maior concentração populacional esteja no continente asiático.

Como o número de habitantes é um dado comum e bastante discutido, talvez este não seja o melhor exemplo. Agora, como você analisaria o código do seu sistema com milhares de linhas para encontrar os pontos mais complexos, com alto índice de acoplamento e que a refatoração se torna indicada para facilitar a evolução e manutenção do software?

Muitas maneiras podem ser utilizadas para apresentar dados, dentre elas, o formato, cores, movimentos, posicionamento na tela e tamanho. Um exemplo dessa abordagem pode ser visto na **Figura 1**. Através dela, conseguimos notar como a forma, o tamanho e a cor podem ser usados para diferenciar os dados.

Hoje em dia é comum encontrarmos técnicas de visualização de informação até mesmo em players de música para dispositivos móveis, onde cada música é representada por um ponto e este ponto é inserido em um plano. Os pontos mais a esquerda indicam que aquelas músicas possuem um ritmo mais rápido, pontos localizados a direita, músicas lentas e assim por diante. Com isso, para escolher as músicas que você deseja ouvir basta marcar o plano com o ponto, então o programa definirá um raio e tocará apenas as músicas que estiverem dentro da região especificada.

Imaginemos que cada símbolo geométrico da **Figura 1** representa um filme de uma pequena locadora, onde a cor está relacionada com o gênero (comédia-verde; drama-vermelho; ação-azul; e suspense-preto), o tamanho relacionado com a duração e o formato indicando se é um dvd ou uma fita. Agora imagine que você deseja alugar uma comédia, de longa duração e que esteja disponível em DVD. Fácil, não? E se quiséssemos apresentar também o dado que indica o número de vezes que o filme foi locado? Seria necessário relacionar este dado com algum atributo visual existente ou escolher outra técnica de visualização para a apresentação dos dados.

Por isso, é importante ressaltar que o formato visual da apresentação dos dados pode não somente facilitar, mas também dificultar a análise que se deseja fazer. Neste sentido, a escolha correta do paradigma visual e os atributos visuais que serão usados para representar os dados tornam-se uma escolha crucial à atividade de visualização.

O paradigma visual utilizado deve se adaptar à natureza dos dados representados. Por exemplo, grafos são adequados para representar dados que descrevem relacionamentos (por exemplo, sites de relacionamento podem ser representados através de grafos). Árvores são adequadas para representar dados hierárquicos (por exemplo, estrutura de diretórios).

Uma vez definido o paradigma visual, vários atributos visuais podem ser associados aos dados a serem representados. A adequação de um atributo visual ao dado depende da escala e tipo desse último atributo. Para exemplificar

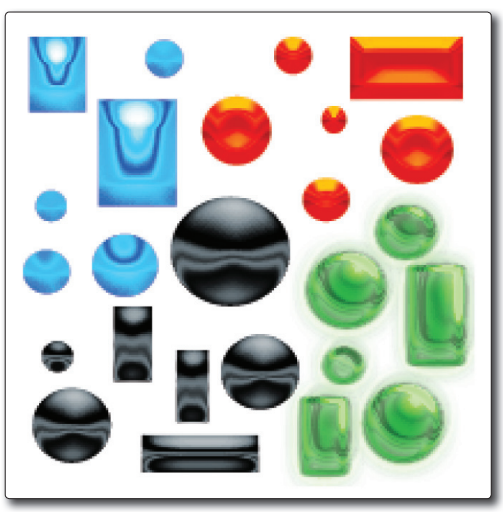


Figura 1. Dados representados através da cor, tamanho e forma.

esta situação, imagine representar todos os países participantes de uma copa do mundo utilizando apenas duas cores.

Os dados podem ser do tipo categórico ou numérico. Enquanto os dados categóricos não possuem um sentido de ordem ou unidade, os dados numéricos possuem. Considere o caso da análise de sistema de softwares que queremos visualizar; neste caso, o nome dos pacotes, classes e métodos são dados categóricos, enquanto o tamanho e a complexidade dos métodos são dados numéricos.

Atributos numéricos podem ser mapeados naturalmente para atributos visuais como tons de cores, tamanho e posicionamento na tela. Atributos categóricos podem ser mapeados naturalmente para formas e conjuntos diferentes de cores. É importante ressaltar que o tipo e a natureza do dado pode também sugerir qual o mapeamento mais indicado para a sua representação visual. Voltando ao exemplo da copa do mundo onde desejamos representar visualmente o registro dos países participantes. Caso se decida representar o país por cores, espera-se que a cor usada para representar cada país esteja relacionada com a nossa percepção natural. Por exemplo, o registro do Brasil poderia estar representado com a cor verde ou amarela e o registro dos Estados Unidos representado pela cor vermelha ou azul.

Dado o cenário acima descrito e a tarefa que temos em mão – representar visualmente atributos do código fonte do software – temos que escolher que paradigma e atributos visuais melhor se adaptam para representar a estrutura e as métricas dos sistemas de software que queremos analisar.

Para uma boa escolha da forma de apresentação, é preciso conhecer a respeito do domínio no qual os dados serão utilizados, como esses dados serão utilizados, as informações que serão apresentadas e o objetivo da apresentação dos dados.

Após escolher a forma de apresentação, o projetista deve levar em consideração alguns dos fatores como: o tipo dos dados, se estes são textuais e/ou numéricos; se existe a necessidade de interação entre o usuário e os dados; com que freqüência os dados apresentados devem ser atualizados; e se o usuário está interessado em informações precisas ou na relação entre o conjunto de dados.

Analisaremos agora como acontece o processo de visualização, discutindo todas as etapas existentes, partindo dos dados brutos até a interação do usuário com as visões.

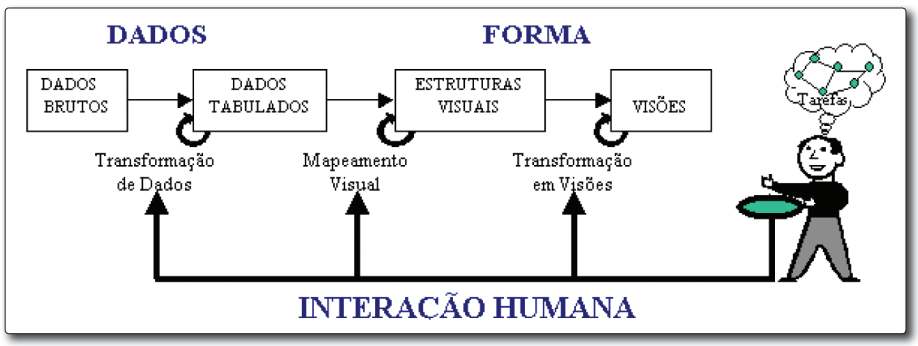


Figura 2. Mapeamento de dados para formas visuais

Processo de Visualização

Para que os dados possam ser apresentados visualmente, eles devem passar por um processo de mapeamento, como pode ser visto na **Figura 2** [1].

A **Figura 2** mostra o processo de visualização de informação, contendo três fases: (1) Preparação dos dados, onde os dados são organizados e formatados para que a aplicação possa entendê-los; (2) Mapeamento, etapa em que os dados

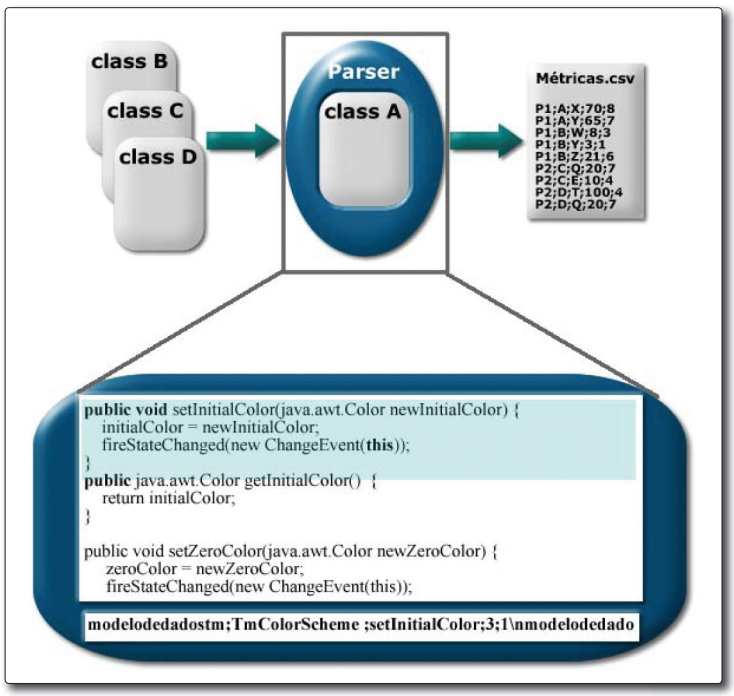


Figura 3. Transformação dos dados

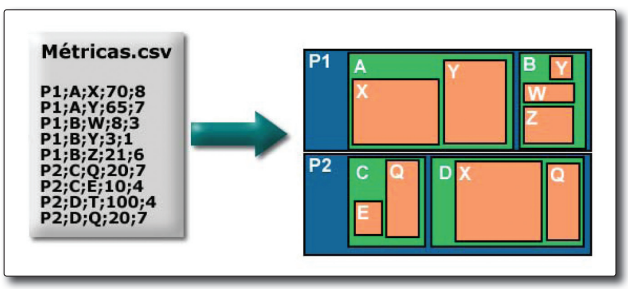
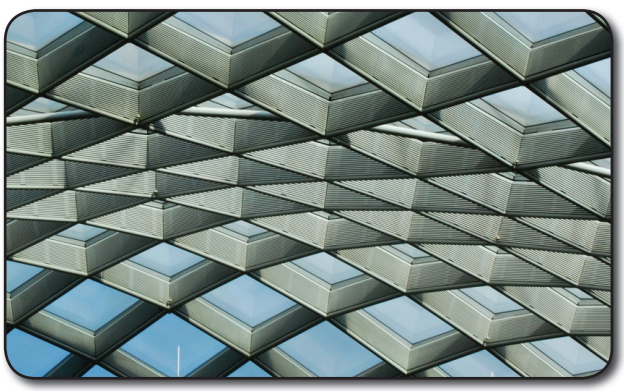


Figura 4. Mapeamento visual



são relacionados com as formas que serão utilizadas para a visualização; (3) Renderização, momento em que a imagem é apresentada na tela. As setas que saem da direita para a esquerda representam a realimentação feita pelo ser humano para explorar os dados que são apresentados de forma visual. A partir de agora analisaremos cada uma das etapas apresentadas na **Figura 2**.

Dados Brutos e Tabela de Dados

Os dados brutos são os dados que queremos analisar ainda no formato de origem. Ele pode estar no formato de tabelas, textos ou formulários. No nosso caso, os dados brutos é o código fonte. A partir do processo de preparação ou transformação, os dados são adaptados para uma forma estruturada, e que permitirá a sua utilização pela ferramenta de visualização.

Normalmente, o resultado do processamento dos dados brutos são representados no formato tabular, onde cada linha representa um registro de dados e cada coluna representa um atributo, ou campo do registro de dados. Seguindo nosso objetivo, o resultado do processamento do código fonte (veja a **Figura 3**) será um arquivo com os dados extraídos das classes (pacote, classe, método, tamanho e complexidade). Estes dados são extraídos utilizando métricas de software, que veremos mais adiante neste artigo.

Estruturas Visuais e Visões

Na visualização, tabelas de dados são mapeadas em estruturas visuais. É neste momento que os dados são relacionados com as formas utilizadas para apresentação da informação (observe a **Figura 4**).

A partir das estruturas visuais montadas, o usuário poderá interagir com as informações, obtendo diferentes visões dos dados apresentados.

Interação Humana

O processo de exploração visual envolve não apenas a construção de estruturas visuais, mas também a interação humana sobre esta estrutura. O processo de interação ocorre conforme mostrado na **Figura 2**, sendo realizado em três momentos. O primeiro momento é na seleção do conjunto de dados que será apresentada visualmente. O segundo ocorre no mapeamento dos atributos dos dados selecionados com atributos visuais que serão apresentados na tela. O terceiro momento acontece na interação do usuário com a visão formada, criando novos cenários visuais.

É importante mencionar que boas ferramentas de visualização permitem a execução de qualquer uma dessas três formas de interação (seleção de dados, o mapeamento visual e a modificação da visão), através de mecanismos fáceis de manusear. Para garantir máxima interatividade, este tipo de ferramenta atualiza o cenário visual no menor tempo possível em resposta a uma ação do usuário. Boas ferramentas oferecem um tempo de resposta praticamente instantâneo para o usuário, garantindo uma interatividade que por vezes lembra a de um vídeo game.

Abordagens para Visualização de Dados

Existem várias abordagens para se criar cenas visuais. As famílias de abordagens mais comuns são as baseadas em iconografia, projeções geométricas, grafos e hierarquias. As abordagens iconográficas usam símbolos ou ícones para representar os dados. Um exemplo de utilização desta técnica são os ícones utilizados na janela de exploração de projeto dos IDEs modernos. Pequenos ícones são utilizados para representar pacotes, serviços, classes, interfaces e outros componentes do software.

Abordagens baseadas em projeções geométricas são as mais comuns no dia a dia das pessoas. Elas utilizam projeções cartesianas ou polares para apresentar gráficos e diagramas aos usuários.

Abordagens baseadas em grafos são utilizadas para apresentar relacionamento entre registros de dados, tais como o acoplamento entre módulos de software ou relacionamento social entre dois programadores. Os registros são representados por nós e o relacionamento por arestas de um grafo.

Como o nome indica, as abordagens hierárquicas são utilizadas para organizar dados de forma hierárquica. Este é o caso de um sistema de arquivos ou a estrutura de pacotes, classes e métodos de um sistema de software.

Poderíamos enumerar várias abordagens para cada uma destas famílias de representação visual. No restante deste artigo utilizaremos *mapas em árvore* para exemplificar a utilização de uma abordagem hierárquica de representação de código fonte.

Mapas em árvore

Uma boa forma de representar dados organizados de forma hierárquica é utilizando estruturas visuais conhecidas como mapas em árvore [3].

A técnica de mapas em árvore, proposta originalmente por *Johnson* e *Shneidermann* [4], consiste em representar o nível mais alto da hierarquia como uma região

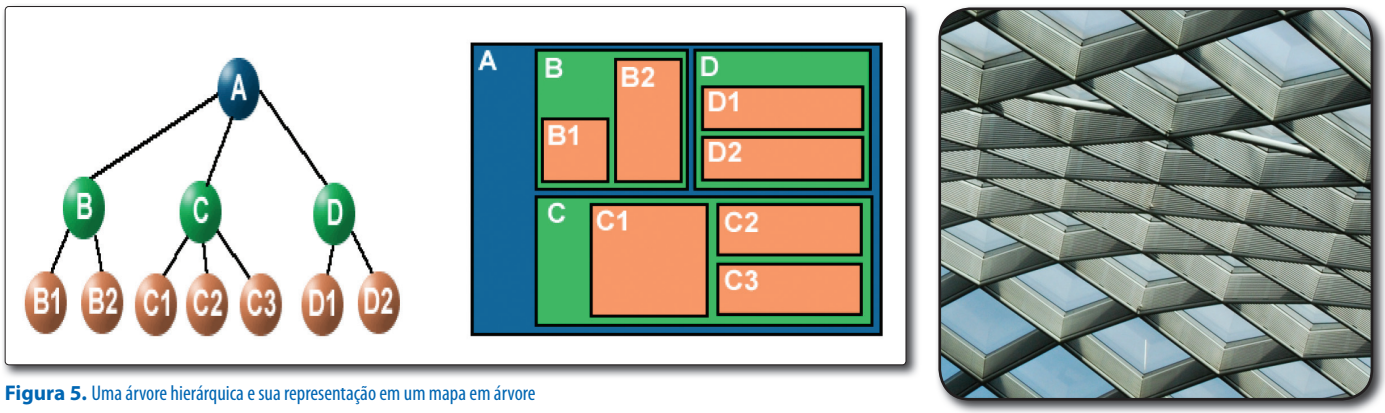
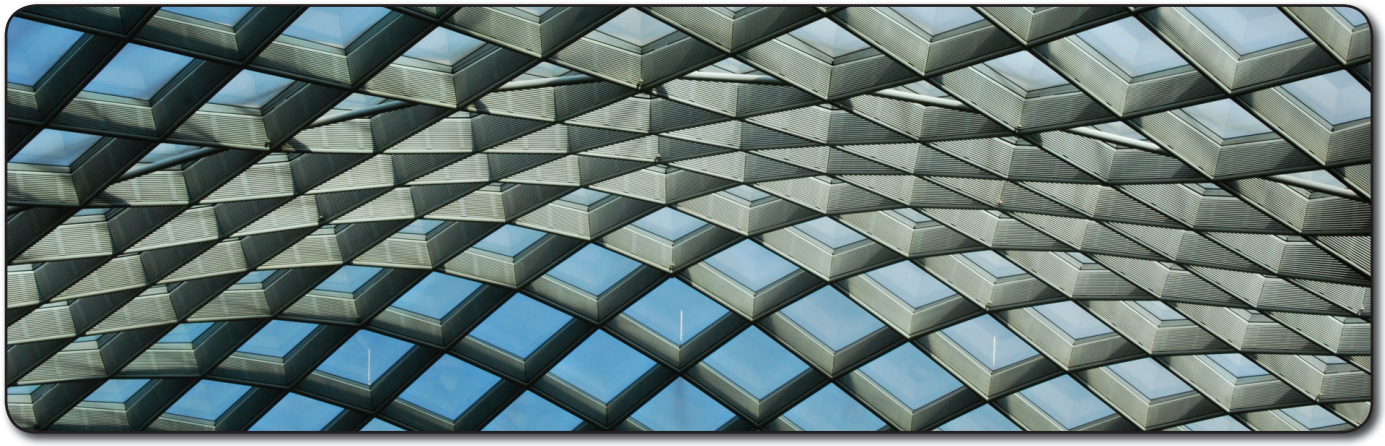


Figura 5. Uma árvore hierárquica e sua representação em um mapa em árvore



retangular que preenche todo o espaço disponível na área de desenho. Os níveis mais baixos são representados por retângulos recursivamente aninhados dentro da região maior, conforme ilustrado na **Figura 5**. O tamanho de cada retângulo é proporcional ao atributo que dimensiona os itens nos níveis imediatamente abaixo na hierarquia.

Esta técnica de visualização permite que todo o espaço disponível na tela de desenho seja utilizado, limitando-se somente à área de exibição das informações dentro de cada grupo (retângulo). Essa abordagem permite que hierarquias com dezenas de milhares de itens possam ser desenhados em uma única cena visual.

Utilizando visualização para compreensão de software

Quando uma pessoa aprende sobre algum assunto, ela se torna capaz de discutir e explicar sobre aquilo que aprendeu. Um processo similar ocorre com os desenvolvedores de software. Quando estes entendem o funcionamento do programa, é normal que saibam explicar como o sistema funciona, qual a estrutura utilizada, e também como modificá-lo para se enquadrar a novos objetivos.

Porém, entender o funcionamento do software não é uma atividade trivial. Dentre os problemas que dificultam o entendimento de um sistema, podemos destacar: o tamanho da aplicação, a sua complexidade, as práticas de programação e as características da linguagem de programação usadas na sua construção.

O processo de compreensão do código é caracterizado pela construção de um modelo mental do funcionamento ou da estrutura do mesmo por um programador. Como temos memória de curto prazo pequena, não somos capazes de armazenar muitos itens de informação durante o processo de compreensão de um sistema [5]. Por essa razão, é normal que os desenvolvedores sintam dificuldade em analisar muitas informações diferentes, e que interagem entre si. A solução para este problema está numa frase bastante conhecida em nossa área: "Dividir para conquistar!". Então, organizamos a informação em níveis diferentes de abstração, onde grandes sistemas são inicialmente percebidos como poucos módulos de grande porte que interagem entre si. Por sua vez, esses módulos podem ser analisados como conjuntos de sub-módulos que interagem entre si, e assim por diante. Dessa forma, a compreensão de um grande sistema pode ser dividida na compreensão de seus sub-módulos.

Esse processo prossegue recursivamente até o nível de detalhe desejado pelo analista. Dessa forma, utilizamos um processo hierárquico, onde um grande problema é recursivamente dividido em problemas menores em um nível de abstração que permite que um ser humano, com limitações que lhe são naturais, possa compreender e analisar grandes sistemas.

Dado esse cenário, pode-se notar a utilidade de uma abordagem hierárquica de visualização software. Tal abordagem permite que o programador entenda o funcionamento e a organização do sistema de uma forma mais rápida, eliminando a maioria das dificuldades citadas no parágrafo anterior. Isto é facilitado se a abordagem possibilita que o usuário interaja com as informações através de mecanismos de análise, exploração e visualização em diferentes níveis de abstração.

Note ainda que abordagens baseadas em relacionamentos (grafos), iconografia e projeções geométricas também têm muito a oferecer para a área.

A partir do momento em que as dificuldades são eliminadas, pode-se esperar vários benefícios com o uso de técnicas de visualização de software. Dentre eles, podemos citar: entendimento do software de forma mais rápida, aumento de produtividade, e melhoria na manutenção, análise, e evolução do software [6].

Tipos de visualização de software

Voltado para a área de software, temos dois tipos de visualização: estática e dinâmica.

A visualização estática é a visualização feita das estruturas estáticas do software. Por exemplo, os dados extraídos da estrutura de um código fonte qualquer. A partir desses dados é montada uma forma visual que representa informações sobre o código a ser analisado.

Neste tipo de visualização, não é apresentado o comportamento de execução do sistema sendo analisado.

A visualização dinâmica tem o objetivo de apresentar estruturas visuais que descrevem o comportamento ou funcionamento do sistema em tempo de execução. Um exemplo desse tipo de visualização são os sistemas de animação de algoritmos computacionais [7]. Outro exemplo são as visualizações de execução construídas pelos sistemas modernos de depuração de código.

Utilizando Métricas na Visualização de Software

Métricas de software têm sido apontadas como um dos principais mecanismos para tornar o desenvolvimento e manutenção de software uma disciplina mais previsível e controlável [8]. Medir é uma prática básica em qualquer tipo de engenharia, e na "engenharia de software" não é diferente. Várias métricas têm sido propostas para se medir atributos como tamanho, complexidade, coesão e acoplamento dos mais variados artefatos de software.

As métricas estão relacionadas tanto com o produto, como com processos de desenvolvimento e manutenção do software. A partir delas, conseguem-se dados quantitativos que oferecem uma boa informação sobre o andamento do projeto. Com essas informações, é possível fazer a estimativa de custos, prazos de entrega e até mesmo ter noção sobre a qualidade do sistema [8].

As métricas de produto são de especial interesse na visualização do software. Elas descrevem características como tamanho, complexidade, características de design e níveis de qualidade do software.

Entre as muitas métricas de produto, as métricas de código fonte estão entre as mais importantes. Elas estão associadas ao produto de mais baixo nível de abstração e maior nível de detalhe que é manipulado por um ser humano no processo de desenvolvimento de software. Muitas destas estão diretamente relacionadas ao paradigma da linguagem de programação, como é o caso das métricas orientadas a objeto. Entretanto, existem aquelas que independem do paradigma de desenvolvimento adotado. A **Tabela 1** apresenta algumas métricas de código fonte [8].

A visualização estática de software é a mais atrativa e comum nos dias de hoje. Ela é construída através da análise estática de artefatos de software. Métricas de código fonte podem ser facilmente combinadas a estes paradigmas, pois elas são extraídas diretamente do artefato estático mais completo de um software, o seu código fonte. Estas métricas podem ser usadas para enriquecer as estruturas visuais extraídas da análise estática do software. Em seguida, descrevemos três métricas bastante utilizadas na visualização de software: tamanho, complexidade e coesão.

Métrica de tamanho

Uma das métricas de código fonte bastante conhecida é a métrica responsável pela extração do tamanho de código. Apesar de ser uma métrica bastante simples, ela pode ser feita de várias formas. Uma delas é contar as linhas por statements. Dessa forma, cada statement representará uma linha de código, com isso, pode-se evitar a contagem a mais de linhas, devido a diferentes estilos de programação.

A importância em se conhecer o tamanho do código está na provável relação que indica: quanto maior for o número de linhas, maior será a dificuldade para entender o sistema, quanto maior for o número de linhas de código, mais difícil será para encontrar as linhas que precisam ser alteradas durante atividades de evolução, e mais difícil será para entender a implementação das funcionalidades que se deseja reutilizar.

Métrica de complexidade

A medição da complexidade de software foi proposta por Thomas McCabe e baseia-se na representação do fluxo de controle de um programa. A complexidade pode ser alta ou baixa, a depender do número de estruturas de decisão dentro do código.

Quanto maior a complexidade de um módulo do sistema, mais difícil é sua compreensão e manutenção.

Métrica de coesão

Esta métrica mede a falta de coesão de uma estrutura do código, por exemplo, a falta de coesão de uma classe. Sua importância está no fato de que classes (componentes) com baixa coesão sugerem um projeto inadequado, significando o encapsulamento de entidades de programa não relacionadas entre si e que não deveriam estar juntas.

Métrica	Aplicada em	Atributo	O que medem
Cyclomatic complexity – CC (McCabe, 1976)	Classes e métodos	Complexidade	Complexidade e alternativas possíveis no controle de fluxo
Lines of Code – LOC (Lorenz e outros, 1994)	Classes e métodos	Tamanho	Obter o tamanho das classes e métodos
Depth of Inheritance Tree – DIT (Chidamber e outros, 1994)	Classes e interfaces	Herança	Reuso, compreensão e teste
Number of Children – NOC (Chidamber e outros, 1994)	Classes e interfaces	Herança	Reuso
Response for Classe – RFC (Chidamber e outros, 1994)	Classes	Comunicação	Acoplamento, complexidade e pré-requisitos para teste
Coupling between object classes – CBO (Chidamber e outros, 1994)	Classes	Comunicação	Coesão e reuso
Lack of Cohesion in Methods – LCOM (Chidamber e outros, 1994)	Classes	Comunicação	Coesão, complexidade, encapsulamento e uso de variáveis
Weighted Methods per Class – WMC (Chidamber e Kemerer, 1994)	Classes	Complexidade	Complexidade, tamanho, esforço para manutenção e reuso
Number of Methods – NOM (Lorenz e outros, 1994)	Métodos	Tamanho	Corresponde a WMC onde o peso de cada método é 1
Number of Statements – NOS (Lorenz e outros, 1994)	Métodos	Tamanho	Obter o número de sentenças em um método
Number of Instance Variables – NIV (Lorenz e outros, 1994)	Classes	Tamanho	Obter o número de variáveis de instância
Number of Class Variables - NCV (Lorenz e outros, 1994)	Classes	Tamanho	Obter o número de variáveis de classe
Number of Inherited Methods – NMI (Lorenz e outros, 1994)	Métodos	Herança	Obter o número de métodos herdados e definidos em uma superclasse
Number of Overriden Methods – NMO (Lorenz e outros, 1994)	Métodos	Herança	Obter o número de métodos definidos em uma superclasse e redefinidos na subclasse

Tabela 1. Exemplo de métricas de código fonte [8]

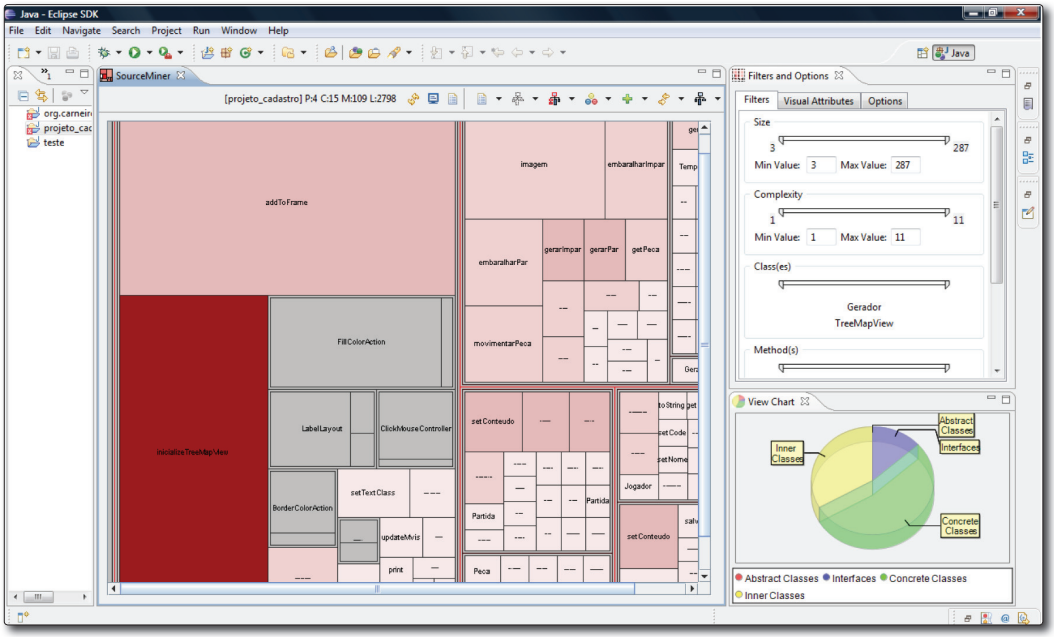


Figura 6. SourceMiner Plugin

Quanto maior for o grau de coesão entre diferentes ações executadas por um componente que contribui para diferentes funcionalidades, mais difícil será para manter ou reutilizar o componente ou uma de suas funcionalidades.

Uma Ferramenta de Compreensão de Software

Com a fundamentação dos principais conceitos, apresentaremos um plug-in chamado SourceMiner, que está sendo desenvolvido pelo GESA, Grupo de Engenharia de Software e Aplicações da Universidade Salvador [9].

O funcionamento do plug-in está dividido basicamente em três etapas, extração das informações e métricas do código fonte, criação das estruturas visuais, e criação dos controles de consulta. A **Figura 6** apresenta o plug-in em execução, utilizando como cenário a técnica de mapas em árvore. Na aba localizada à direita, estão os controles de consulta. Através deles o usuário pode interagir com o cenário padrão criando diferentes cenários, possibilitando a análise do sistema em diferentes níveis de abstração.

Há pouco perguntei como analisar o código de um sistema contendo milhares de linha? Como verificar quais os trechos mais indicados para refatoração?

Agora, observe novamente a **Figura 6**. Neste cenário, o tamanho dos retângulos representa o tamanho dos métodos e a cor representa a complexidade. Considerando que métodos muito grandes e com alta complexidade são um importante indicativo da necessidade de refatoração, fica fácil verificar em apenas uma tela todo o projeto, ao invés de abrir centenas de arquivos para obter tais informações.

Observe também a aba View Chart, nela várias informações sobre o projeto são apresentadas através de gráficos em pizza e em barras, por exemplo: a relação entre interfaces, classes internas, classes abstratas e classes normais de um pacote; a relação entre os pacotes, contabilizando a quantidade de métodos existentes dentro deles; entre outros.

Para obter mais informações sobre o SourceMiner e baixá-lo para teste, acesse o endereço *http://www.nuperc.unifacs.br/tools*.

Conclusão

Apresentamos neste artigo como a visualização de informação pode ser utilizada para facilitar a compreensão de software e tornar menos difícil a tarefa de manutenção de sistemas de software. Para isso, foi discutido sobre o processo de visualização, técnicas de visualização, e métricas de software, mais especificamente, métricas de código fonte.

No final do artigo, exibimos a junção destes conceitos em um plug-in que utiliza diversos paradigmas de visualização com o intuito de fornecer ao engenheiro de software e/ou desenvolvedor, diversas perspectivas sobre a estrutura do código fonte de um sistema de software. Espera-se, com isso, que o processo de desenvolvimento, manutenção e evolução de sistemas se torne uma tarefa menos trabalhosa e cansativa. ●

Referências

Almeida, Márcio Oliveira. Uma Ferramenta para Mineração Visual de Dados Usando Mapas em Árvore e Suas Aplicações. Dissertação de Mestrado. Universidade Salvador. Salvador. Abril, 2003.

Gershon, N; Elick, S.G. Information Visualization. IEEE Computer Graphics and Applications. Los Alamitos, p.29-31, 52-59, 1997.

Shneiderman, B. Tree visualization with tree-maps: 2-d space-filling approach. ACM Transactions on Graphics, v. 11 , n. 1, p. 92-99, Jan. 1992.

Johnson, Brian; Shneiderman, Ben. Tree-Maps: A space-filling approach to the visualization of hierarchical information structures. In: IEEE VISUALIZATION, 1991, San Diego. Proceedings… Los Alamitos: IEEE Computer Society Press, 1991. p.284-291.

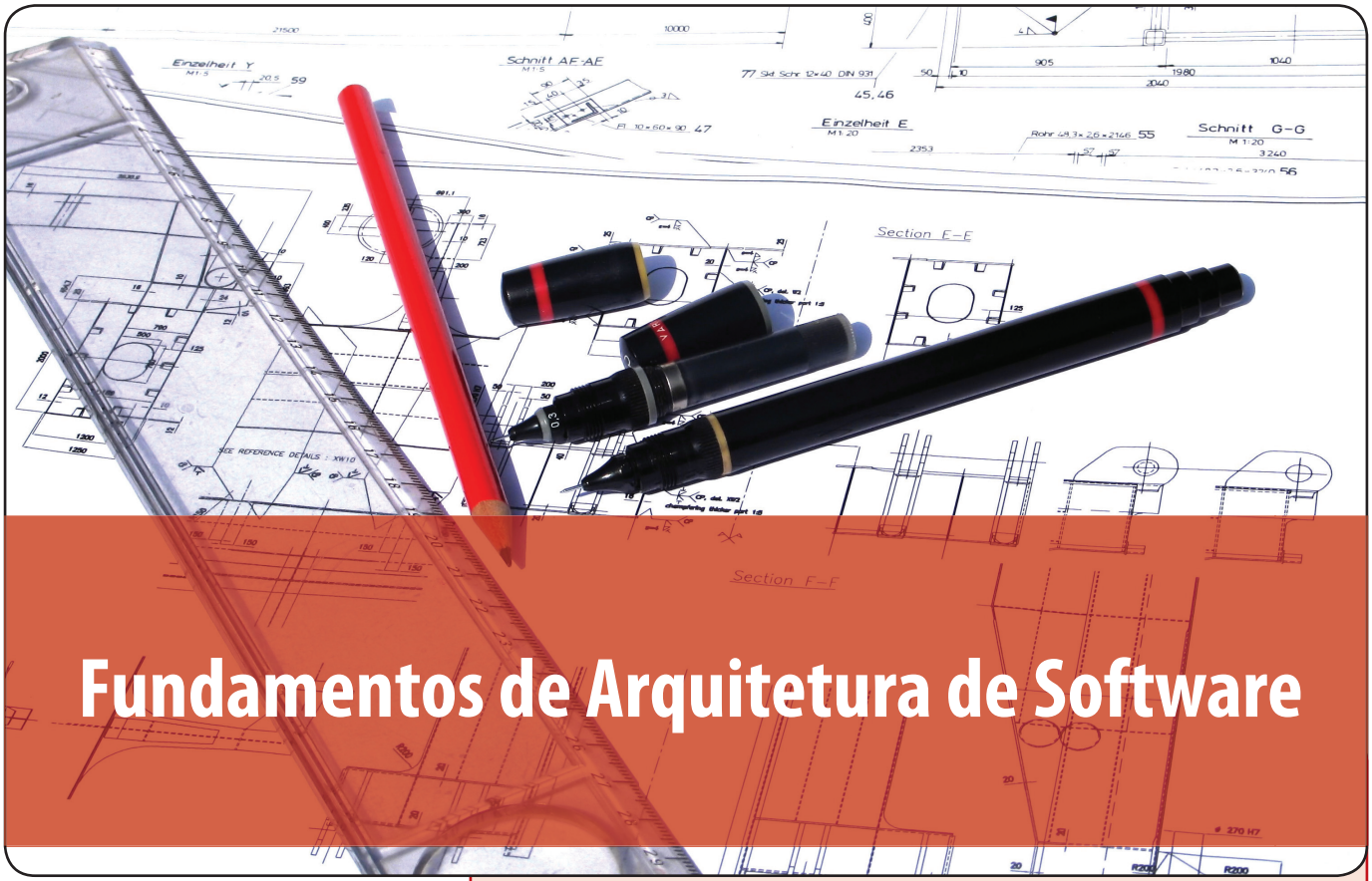
Campos, Marcelo Ricardo. Compreensão Visual de Frameworks através da Introspecção de Exemplos. Tese de Doutorado. UFRGS. Porto Alegre. 1997.

Fyock, Daniel E. Using Visualization to Maintain Large Computer Systems. IEEE Computer Graphics and Applications. Los Alamitos, p.73-75, 1997.

Hamilton, Ashley - Washington University at St. Louis; Kraemer, Eileen; Tudoreanu, Mihail; Wu, Rong - University of Georgia. Empirical Evidence that Algorithm Animation Promotes Understanding of Distributed Algorithms. IEEE Symposia on Human Centric Computing Languages and Environments (HCC'02) p. 236, 2002.

Carneiro, Glauco de Figueiredo. Usando Medição de Código Fonte para Refactoring. Dissertação de Mestrado. Universidade Salvador. Salvador. Abril, 2003.

Carneiro, Glauco de Figueiredo; Magnavita, Rodrigo; Spínola, Eduardo Oliveira; Spínola, Fábio Oliveira; Mendonça, Manoel Gomes. An Eclipse-Based Visualization Tool for Software Comprehension. Publicado no SBES 2008 – Sessão de Ferramentas – Campinas, Brasil.



Fundamentos de Arquitetura de Software

De que se trata o artigo:

Este artigo apresenta os fundamentos da arquitetura de software. São descritos a importância e o papel da arquitetura de software no processo de desenvolvimento. Também são identificadas as principais atividades realizadas durante o processo de especificação arquitetural.

Para que serve:

Quando tentamos solucionar um problema, é possível identificar diversas soluções que poderiam ser utilizadas visando resolvê-lo. Contudo, outros fatores como custo e eficiência influenciam na escolha da solução a ser adotada. No contexto do desenvolvimento de software, o mesmo pode ser observado ao se analisar os requisitos visando a construção de um software: várias soluções computacionais podem ser definidas para atender a esses requisitos, mas uma análise deve ser feita para definir a mais adequada ao contexto de desenvolvimento da aplicação. Para se representar essas soluções, a arquitetura de software é uma das abordagens que podem ser usadas.

Em que situação o tema é útil:

No entendimento dos fundamentos da arquitetura de software. Conhecimento este fundamental na elaboração da arquitetura de aplicações em projetos reais.

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 Engenharia de Software Magazine.

Rafael Ferreira Barcelos

rbarcelos@gmail.com

É Mestre na área de Engenharia de Software da COPPE/UFRJ, atualmente trabalha como Software Development Engineer in Test na Microsoft em Redmond/USA. Possui 5 anos de experiência em desenvolvimento de software tanto para sistemas de informação quanto para sistemas específicos, como por exemplo celular.

Quando tentamos solucionar um problema, é possível identificar diversas soluções que poderiam ser utilizadas visando resolvê-lo. Contudo, outros fatores como custo e eficiência influenciam na escolha da solução a ser adotada. No contexto do desenvolvimento de software, o mesmo pode ser observado ao se analisar os requisitos visando a construção de um software: várias soluções computacionais podem ser definidas para atender a esses requisitos, mas uma análise deve ser feita para definir a mais adequada ao contexto de desenvolvimento da aplicação.

Para se representar essas soluções, a arquitetura de software é uma das abordagens que podem ser usadas. Com isso, para se obter a arquitetura (solução) mais adequada para atender aos requisitos do software (problema), uma avaliação dessa estrutura deve ser realizada.

A arquitetura consiste em um modelo de alto nível que possibilita um entendimento e uma análise mais fácil do software a

ser desenvolvido. O uso de arquitetura para representar soluções de software foi incentivada principalmente por duas tendências (GARLAN e PERRY, 1995; KAZMAN, 2001): (1) o reconhecimento por parte dos projetistas que o uso de abstrações facilita a visualização e o entendimento de certas propriedades do software, e (2) a exploração cada vez maior de frameworks visando diminuir o esforço de construção de produtos através da integração de partes previamente desenvolvidas.

Outra propriedade da arquitetura é a possibilidade de usá-la como ferramenta para comunicar a solução projetada aos diversos stakeholders que participam do processo de desenvolvimento do software (GARLAN, 2000). Contudo, para que essa comunicação seja possível, a arquitetura deve ser representada através de um documento, conhecido como documento arquitetural.

Para se construir a arquitetura de um software, e por conseqüência o documento arquitetural que a representa, os requisitos são as principais informações usadas. Durante o processo de especificação arquitetural (**Figura 1**), além dos requisitos, outras fontes de conhecimento podem ser utilizadas para definir os elementos arquiteturais e a forma como eles devem estar organizados. Entre essas fontes de conhecimento se destacam principalmente a experiência do arquiteto, o raciocínio sobre os requisitos, e os estilos e as táticas arquiteturais.

Contudo, existe uma falta de consenso na comunidade em relação tanto aos conceitos e definições básicas quanto à forma de representar uma arquitetura de software (BUSCHMANN et al., 1996; CLEMENTS et al., 2004). Portanto, na próxima seção são descritos os termos aqui adotados e seus respectivos conceitos associados. Além disso, são descritos a importância e o papel da arquitetura de software no processo de desenvolvimento, e, por fim, são identificadas as principais atividades realizadas durante o processo de especificação arquitetural.

Definição dos conceitos relacionados à arquitetura de software

Nessa seção, são definidos os termos utilizados neste trabalho, evitando ambigüidades, visto que terminologias inconsistentes sobre estes termos podem ser encontradas na literatura.

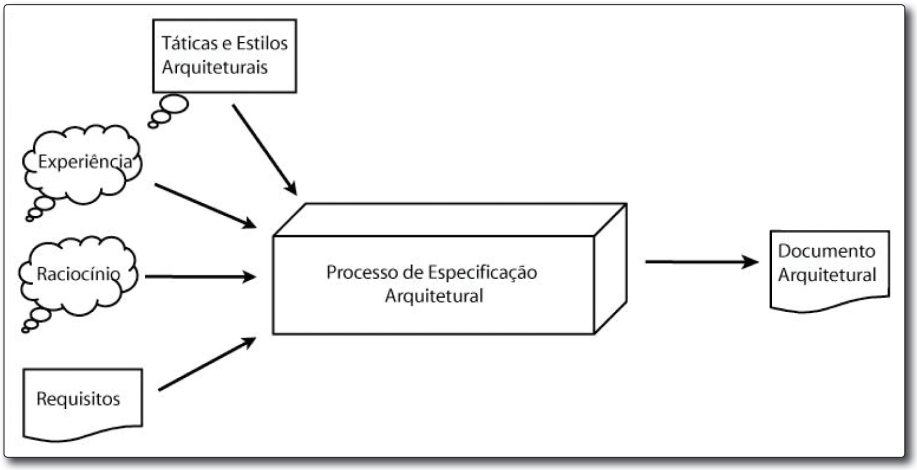


Figura 1. Elementos usados na construção de uma arquitetura.

Arquitetura de software representa a estrutura, ou conjunto de estruturas, que compreende os elementos de software, suas propriedades externamente visíveis e seus relacionamentos (BASS et al., 2003).

Para criar essa estrutura, grande parte dos autores concorda que três tipos de elementos básicos podem ser usados (DIAS e VIEIRA, 2000):

- Elementos de software, podendo também ser chamados de módulos ou componentes, são as abstrações responsáveis por representar as entidades que implementam funcionalidades especificadas;
- Conectores, podendo ser chamados de relacionamentos ou interfaces, são as abstrações responsáveis por representar as entidades que facilitam a comunicação entre os elementos de software;
- Organização ou configuração que consiste na forma como os elementos de software e conectores estão organizados.

Além disso, essa estrutura e as entidades que a compõem devem ser representadas de uma forma que permita utilizar a arquitetura projetada para seus devidos fins, a essa representação é dado o nome de documento arquitetural. Esse documento é composto por um conjunto de modelos e informações que descrevem principalmente a estrutura do software especificado para atender aos requisitos. Para compor um documento arquitetural, podemos nos basear, por exemplo, nas recomendações descritas no padrão IEEE-1471 (IEEE, 2000).

Contudo, mesmo existindo padrões que indicam o tipo de informação que deve ser descrito em um documento arquitetural, não é definido exatamente o nível de abstração que deve ser usado na descrição dessas informações.

A arquitetura de um software começa a ser construída nos estágios iniciais de um processo de desenvolvimento de software com o objetivo de definir e visualizar a solução computacional que será implementada. Neste momento, esse artefato é conhecido como arquitetura inicial, pertence ao escopo do problema, tem como principal característica descrever a solução em um elevado nível de abstração e é utilizado por vários stakeholders como base para tomada de decisões.

Contudo, ao longo do desenvolvimento do software, a arquitetura sofre refinamentos que diminuem o nível de abstração e permitem, por exemplo, a representação dos relacionamentos entre os elementos arquiteturais e os arquivos de código fonte responsáveis por implementá-los (CLEMENTS et al., 2004). Neste momento, a arquitetura passa a pertencer ao escopo da solução e incorpora também informações relacionadas às decisões de projeto, como elementos específicos à tecnologia que será usada para implementar a solução.

O fato da arquitetura representar informações em diferentes níveis de abstração ao longo do processo de desenvolvimento é um dos motivos que leva à falta de consenso na comunidade, pois ainda não se padronizou a granularidade que deve ser usada para descrever esse artefato.

No contexto desse artigo, iremos trabalhar somente com a arquitetura inicial, ou seja, a que representa a estrutura em um elevado nível de abstração. Acreditamos que o uso de arquitetura para representar a solução em um baixo nível de abstração não é adequado devido à existência de diversos tipos de representação de projeto de baixo nível, como diagramas de classe e de seqüências, que permitem uma representação mais completa desse tipo de informação.

A partir de agora, identificaremos os papéis que a arquitetura possui no processo de desenvolvimento de software e os benefícios que podem ser obtidos ao avaliá-la.

Papel da arquitetura em um processo de desenvolvimento de software e os benefícios de sua avaliação

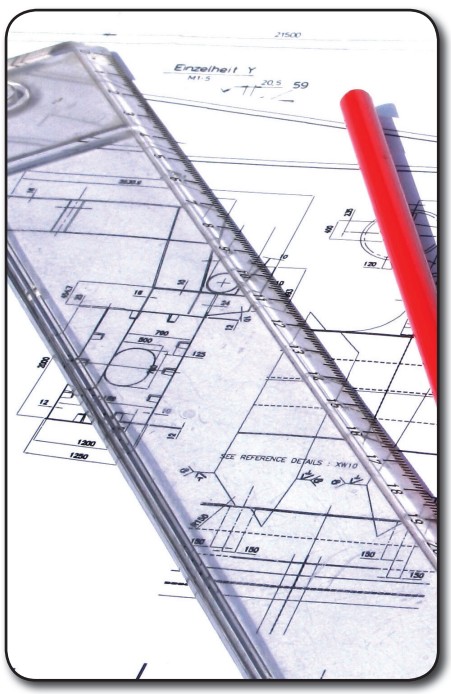
Ao revisar um artefato de software vários benefícios para o projeto e para a melhoria da qualidade do software podem ser obtidos. Contudo, para que essa atividade seja realizada, recursos devem ser alocados, o que pode aumentar o custo final do projeto.

Portanto, antes de realizar a revisão de um artefato, é imprescindível que a importância desse artefato dentro do processo de desenvolvimento seja identificada, permitindo definir o custo/benefício de sua revisão.

A principal motivação para avaliar a arquitetura de um software está relacionada ao seu papel dentro do processo de desenvolvimento.

Possuindo o documento arquitetural do sistema, os stakeholders podem utilizá-lo como artefato de entrada na realização de algumas atividades do processo ou então como base para tomada de decisões no contexto do projeto. Para cada stakeholder, a arquitetura do software é utilizada com diferentes propósitos (GACEK, 1995; XAVIER, 2001; CLEMENTS et al., 2004):

- Cliente. O cliente é a pessoa ou empresa que contrata uma equipe de desenvolvimento para a construção de um sistema de sua necessidade. Na fase inicial do projeto, esse stakeholder necessita de uma estimativa de certos fatores, normalmente econômicos, que podem ser obtidos após a definição da estrutura principal do software. O cliente, por exemplo, tem interesse em estimativas de custo, confiabilidade e manutenibilidade do software que podem ser obtidos principalmente através de uma análise da arquitetura. Portanto, é de extrema importância para o cliente que a arquitetura atenda os requisitos do software de forma a representar suas reais expectativas em relação ao que foi especificado.
- Gerentes. A arquitetura permite aos gerentes tomarem certas decisões de projeto por possibilitar a sumarização das diversas características do sistema. Um gerente pode, por exemplo, usar a arquitetura como base para definir as equipes de desenvolvimento de acordo com os elementos arquiteturais que estão identificados na arquitetura e que devem ser construídos.
- Desenvolvedor. Da arquitetura de um software, o desenvolvedor busca uma especificação que descreva a solução com detalhes suficientes e que satisfaça os requisitos do cliente, mas que não seja tão restritiva a ponto de limitar a escolha das abordagens para a sua implementação. Os desenvolvedores usam a arquitetura como uma referência para a composição e o desenvolvimento dos elementos do sistema, e para a identificação e reutilização de elementos arquiteturais já construídos.
- Testadores. A arquitetura fornece, numa visão de caixa preta, informações aos testadores relacionadas ao correto comportamento dos elementos arquiteturais que se integram. Sendo assim, este artefato pode ser um dos artefatos bases utilizados durante o planejamento e execução de testes de integração e de sistema.
- Mantenedor. A descrição arquitetural do software fornece aos mantenedores uma estrutura central da aplicação que idealmente não deve ser violada. Qualquer mudança deve preservá-la, buscando, se possível, uma



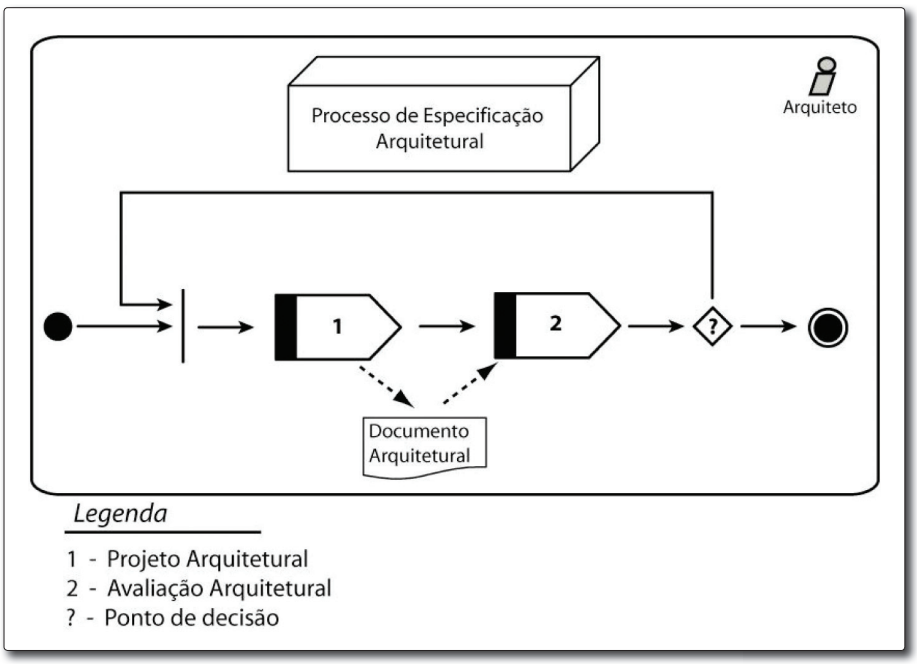


Figura 2. Processo genérico de especificação arquitetural

modificação puramente dos elementos arquiteturais e não da forma como estão organizados.

Visto como os principais stakeholders podem utilizar a arquitetura de um software, percebemos que o principal papel desse artefato é servir como instrumento para comunicar a solução proposta (GARLAN, 2000).

Sendo assim, o principal benefício em se avaliar um documento arquitetural está na diminuição das chances de um stakeholder utilizar um documento defeituoso nas atividades subseqüentes do processo de desenvolvimento de software.

Contudo, para permitir uma melhor compreensão sobre como e o que deve ser avaliado em um documento arquitetural, devemos primeiro entender como esse artefato é criado.

Processo de especificação arquitetural

Existem na literatura diversas abordagens que objetivam a especificação de arquiteturas de software. Após avaliar algumas das principais abordagens (GACEK, 1995; SHAW e GARLAN, 1996; BOSCH e MOLIN, 1999; BACHMANN et al., 2000; BASS et al., 2003) pode-se perceber um processo genérico de especificação arquitetural.

Esse processo é composto principalmente pelos seguintes elementos (**Figura 2**): duas macro-atividades (projeto e avaliação arquitetural) e a tarefa de documentação da arquitetura. O que diferencia essas abordagens é principalmente a forma como cada um desses elementos são realizados.

Nesse processo, a característica comum às duas macro-atividades identificadas é a presença da tarefa de documentação responsável por criar e atualizar o documento que representa a arquitetura de software. Esse documento arquitetural é criado durante a macro-atividade de projeto arquitetural e é responsável por registrar as decisões e os elementos arquiteturais.

Após identificarem que a solução descrita na arquitetura atende a todos os requisitos especificados, os arquitetos dão início à atividade de avaliação arquitetural que utiliza como principal artefato de entrada o documento arquitetural.

Após a avaliação, dependendo da qualidade do documento arquitetural e por conseqüência da arquitetura projetada, o arquiteto decide se o artefato será reavaliado visando atingir a qualidade desejada ou então se o processo de especificação arquitetural será finalizado.

A seguir, é mostrado para cada um dos elementos do processo de especificação arquitetural que tipo de informações e abordagens podem ser utilizadas para realizá-los.

Projeto Arquitetural

O projeto arquitetural consiste na atividade em que a solução computacional e, por conseqüência, a arquitetura do software são definidas. Durante essa atividade, o raciocínio sobre os requisitos é realizado e decisões arquiteturais são tomadas, visando identificar e organizar os elementos arquiteturais para que os requisitos especificados possam ser atendidos.

Ao se analisar como essa atividade é realizada nas principais abordagens de especificação arquitetural, observamos a importância dos requisitos de qualidade no projeto de uma arquitetura e a existência de várias abordagens que podem ser utilizadas para atendê-los.

Requisitos de Qualidade

Os requisitos de um software podem ser classificados, de forma geral, como requisitos funcionais e os não-funcionais.

Os requisitos funcionais são responsáveis por descreverem as funcionalidades que o software deve apresentar. Já os não-funcionais descrevem características que o software deve apresentar, muitas vezes podem ser enxergadas como restrições ou especialidades do produto final. Os requisitos podem ter várias subcategorias como requisitos de qualidade, requisitos legais e etc.

Dentre os diferentes tipos de requisitos, tanto funcionais quanto não-funcionais, os requisitos de qualidade são os que mais influenciam na construção da arquitetura. Isso ocorre visto que, diferente dos requisitos funcionais onde na maioria dos casos uma modificação ocasiona alterações em um conjunto específico de elementos arquiteturais, alterações em um requisito de qualidade podem implicar na total reestruturação da arquitetura (BASS et al., 2003).

Contudo, nem todos os requisitos de qualidade são relevantes a nível arquitetural, pois determinados tipos de requisitos podem ser atendidos somente durante a etapa de codificação ou disponibilização (XAVIER, 2001). Um requisito de inteligibilidade, por exemplo, só poderá ser implementado no momento da definição da interface do sistema com o usuário.

Existem diferentes taxonomias para se classificar requisitos de qualidade (ISO/IEC, 1998; BASS et al., 2003). No contexto desse artigo, adotamos a taxonomia descrita por BASS et al. (2003) visto que ela identifica os tipos de requisitos de qualidade que são relevantes a nível arquitetural, ou seja, quais os tipos de requisitos de qualidade que influenciam na construção da arquitetura de um software.

Portanto, de acordo com BASS et al. (2003), esses tipos de requisitos são:

• Desempenho: Descrevem o comportamento do sistema em relação a restrições de tempo e de recurso computacional;
• Disponibilidade: Descrevem o comportamento de determinada parte do sistema em caso de falha;
• Modificabilidade: Descrevem quais as prováveis modificações que podem acontecer no sistema e as flexibilidades que devem estar nele presentes para que essas modificações sejam facilmente realizadas;
• Segurança: Descrevem o comportamento de determinada parte do sistema em relação ao acesso de seus dados ou funcionalidades;
• Testabilidade: Descrevem o comportamento de determinada parte do sistema em relação às facilidades que elas devem fornecer para a realização de testes;
• Usabilidade: Requisitos desse tipo, em um contexto arquitetural, descrevem facilidades que o sistema deve possuir, mas que não são consideradas funcionalidades do sistema. Exemplo dessas facilidades são operações de *undo* e *redo*.

Atendendo os requisitos de qualidade

Durante o projeto de uma arquitetura, para atender aos requisitos de qualidade, as principais abordagens utilizam diversas fontes de conhecimento, tanto tácito quanto explícito para definir quais serão os elementos arquiteturais e com estarão organizados. Um exemplo de conhecimento tácito seria a experiência do arquiteto, e em relação ao conhecimento explícito teríamos os estilos e as táticas arquiteturais.

A experiência de um arquiteto é uma característica importante para o sucesso do projeto de uma arquitetura, pois a partir de suas lições aprendidas, o arquiteto consegue facilmente identificar que elementos arquiteturais devem ser criados e como eles devem ser organizados. Mas, por ser um conhecimento tácito, é difícil de ser externalizado e utilizado por terceiros.

Por outro lado, estilos e táticas arquiteturais são conhecimentos explícitos amplamente difundidos na literatura e bastante utilizados por arquitetos de software (SHAW, 1995; BUSCHMANN et al., 1996; BASS et al., 2003).

Um estilo arquitetural, ou padrão arquitetural, consiste em um conhecimento que pode ser diretamente aplicado pelo arquiteto na identificação dos elementos arquiteturais. Isso é possível por ele ser composto por um conjunto de regras que permitem a identificação dos tipos de componentes e de conectores que serão usados na composição do software levando em conta as restrições impostas (SHAW e GARLAN, 1996).

Na literatura, existe um outro conceito, chamado de padrões de projeto, que é muito semelhante ao conceito de estilos arquiteturais. Em BUSCHMANN et al. (1996), é feita a diferenciação entre padrões arquiteturais e padrões de projeto. Essa diferença encontra-se principalmente no nível de abstração onde cada um desses padrões atua. Os padrões de projeto são utilizados somente durante a fase de definição do projeto de baixo-nível, onde refinamentos são feitos nos elementos arquiteturais que formam a arquitetura, e que foram definidos com base nos padrões arquiteturais. Contudo, muitos dos conceitos presentes em padrões arquiteturais e padrões de projeto são semelhantes, mas o que os diferencia é o fato de serem utilizados em níveis de abstração diferentes.

No contexto desse artigo abordaremos somente padrões arquiteturais pois são eles que possuem os principais conceitos relevantes a nível arquitetural. Para evitar confusões, utilizaremos a denominação de estilos arquiteturais quando abordarmos esses conceitos.

Com isso, uma característica particular aos estilos arquiteturais é que o uso de um único estilo possibilita o atendimento a vários tipos de requisitos de qualidade. XAVIER (2001), por exemplo, descreve uma abordagem que, a partir dos tipos de requisitos de qualidade que devem ser atendidos pelo software, permite identificar os estilos arquiteturais mais adequados que devem ser usados na construção desse software.

Além dos estilos, outro tipo de conhecimento explícito que pode ser utilizado no projeto arquitetural são as táticas arquiteturais. Uma tática arquitetural consiste em um conhecimento mais abstrato, utilizado principalmente para auxiliar o atendimento a um tipo de requisito de qualidade. Portanto, por serem mais abstratas, essas táticas descrevem principalmente possíveis características que uma arquitetura deve apresentar para atender a um determinado tipo de requisito.

Em BASS et al. (2003), essas táticas são identificadas e categorizadas em grupos, de acordo com os atributos de qualidade que elas influenciam.

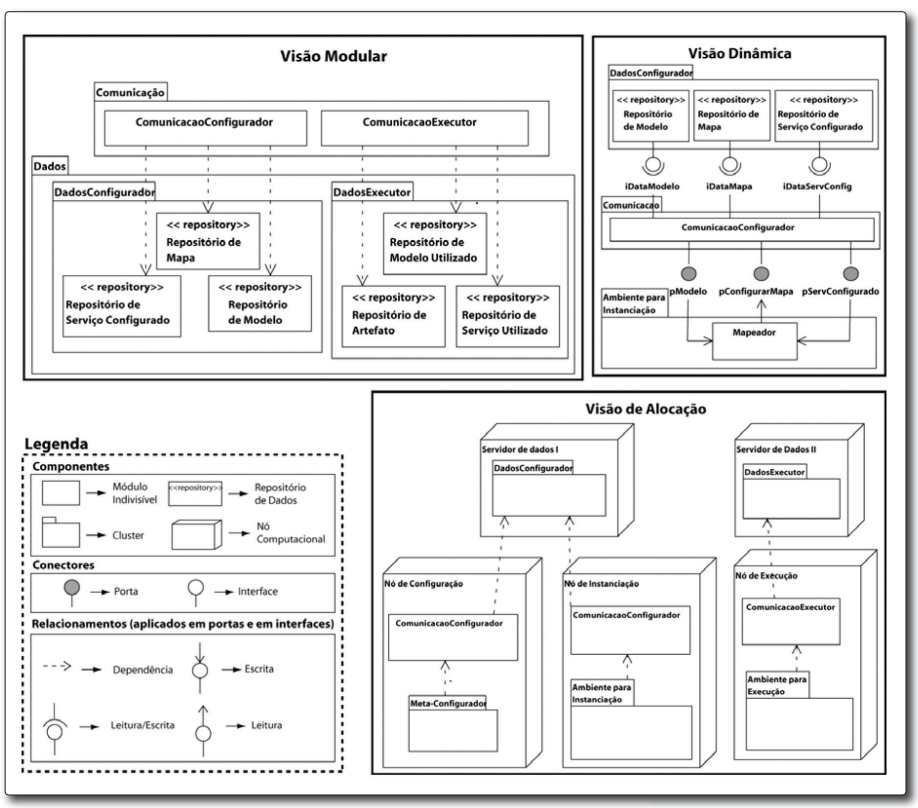


Figura 3. Exemplo de visões arquiteturais

Uma característica particular a essas táticas é que quando agrupadas e especializadas, podem ser usadas como base para a criação de estilos arquiteturais. ZHU (2004), por exemplo, realizou uma análise dos principais estilos arquiteturais por eles utilizados e identificou as táticas arquiteturais que os compõem.

Sendo assim, a partir do uso desse tipo de conhecimento, o arquiteto consegue definir a estrutura principal da arquitetura. Essa estrutura é em seguida povoada com elementos arquiteturais identificados principalmente a partir da análise dos requisitos funcionais.

Documentação Arquitetural

Uma característica única em Engenharia de Software em relação às outras áreas de engenharia é que os produtos por ela construídos não são completamente materializáveis. Diferente de um engenheiro civil que pode inspecionar, por exemplo, as partes de um prédio, um engenheiro de software não consegue inspecionar um pedaço do software em si. Para isso ele deve utilizar representações desse software (LAITENBERGER e ATKINSON, 1999).

A arquitetura é um exemplo da parte de um software que não é materializável. Durante uma inspeção, por exemplo, é o documento arquitetural que deve ser revisado, por impossibilidade de se inspecionar diretamente a arquitetura projetada. Sendo assim, durante o seu projeto, a arquitetura tem que ser documentada para que ela possa ser usada para os seus devidos fins.

A arquitetura é uma entidade complexa que não pode ser descrita de uma forma unidimensional (CLEMENTS et al., 2004). Uma forma efetiva de lidar com essa complexidade é descrevendo-a a partir de diferentes perspectivas, também conhecidas como visões arquiteturais.

Em cada visão, a forma como os elementos arquiteturais e seus relacionamentos são documentados coloca em evidência propriedades distintas do software que eles representam. De acordo com EGYED e MEDVIDOVIC (1999), ao criar uma visão arquitetural, os desenvolvedores conseguem reduzir a quantidade de informação que são obrigados a lidar em um determinado momento. Portanto, essas visões representam um aspecto parcial da arquitetura que mostram propriedades específicas do software.

Na **Figura 3**, podemos identificar três visões arquiteturais usadas para descrever um conjunto de elementos arquiteturais. Independente da notação gráfica utilizada, é possível notar as diferentes propriedades que cada visão permite identificar.

Existe um grande número de visões arquiteturais propostas na literatura que propõem soluções similares para a representação de uma arquitetura (KRUCHTEN, 1995; HOFMEISTER et al., 2000; CLEMENTS et al., 2004). As principais visões são:

- Visão Modular: Esta perspectiva representa os elementos que compõem a arquitetura, responsáveis por realizar um conjunto de funcionalidades, e as dependências entre eles. Para isso, um conjunto de diagramas pode ser criado para representar através de diferentes níveis de abstração, os elementos, seus elementos internos (caso haja) e como eles se relacionam entre si.
- Visão Dinâmica: Esta perspectiva procura descrever o comportamento dos elementos arquiteturais durante a realização dos diferentes fluxos de execução que pertencem ao sistema.
- Visão de Alocação: Esta perspectiva busca representar o mapeamento das unidades de software para elementos físicos do ambiente (hardware, arquivos do sistema, equipe de desenvolvimento).
- Visão de contexto geral: Essa perspectiva tem como objetivo representar uma visão geral dos principais componentes que formam a arquitetura do software e de como ele se relaciona com os elementos externos ao seu contexto (atores e sistemas externos).

A escolha das visões a serem documentadas deve ser feita com base nas características de qualidade que se deseja por em evidência, uma vez que diferentes visões expõem características de qualidade distintas.

Para CLEMENTS *et al*. (2004), documentar uma arquitetura consiste em documentar as visões arquiteturais relevantes, explicar como essas visões se relacionam e como um stakeholder deve utilizar esse material.

No contexto desse artigo, é importante ressaltar algumas das recomendações definidas pelo padrão IEEE-1471, que abordam a descrição arquitetural de sistemas de software, para definir as principais informações que devem ser descritas em um documento arquitetural. Sendo assim, um documento arquitetural deve:

- Identificar os elementos arquiteturais que compõem a solução a ser construída, assim como a forma que esses elementos estão organizados;
- Descrever o papel de cada elemento dentro da arquitetura;
- Identificar como cada requisito relevante a nível arquitetural está sendo atendido através da arquitetura documentada. Essa identificação pode ser feita principalmente através do rastreamento de que requisito está sendo atendido e quais requisitos justificam a criação de determinado elemento arquitetural.
- Representar o software através de diferentes perspectivas, por exemplo, através do uso de visões arquiteturais.

Avaliação Arquitetural

A avaliação arquitetural consiste em caracterizar e avaliar os documentos arquiteturais através de métodos ou procedimentos sistemáticos (BAHSOON e EMMERICH, 2003). Essa avaliação verifica principalmente se as informações descritas no documento estão consistentes e se a arquitetura nele representada atende aos requisitos especificados para o produto.

Visto que são os requisitos de qualidade os que mais influenciam a construção de uma arquitetura, portanto, é principalmente sob a perspectiva desse tipo de requisitos que a avaliação deve ser realizada (DOBRICA e NIEMELA, 2002; BABAR et al., 2004).

A realização da atividade de avaliação é de extrema importância para a melhoria da qualidade do produto de software e para o sucesso do projeto. Esta afirmação é fortalecida se for considerado que (1) a avaliação da arquitetura impede que seus defeitos se propaguem para os demais artefatos, como diagramas de projeto e código fonte, e (2) o custo de correção desses defeitos é bem menor se for realizada durante os primeiros estágios do projeto (BOEHM, 1981).

Além dos benefícios listados anteriormente, MARANZANO et al. (2005) identificaram os seguintes benefícios, após aplicar a avaliação arquitetural em diversos projetos no contexto da empresa em que trabalha, que podem ser obtidos através dessa prática:

- Permite um melhor aproveitamento do conhecimento de seus especialistas, pois são alocados em avaliações arquiteturais que analisam arquiteturas de projetos em que não tiveram participação, utilizando assim suas experiências e conhecimentos para auxiliá-los;
- Permite um melhor gerenciamento dos fornecedores de componentes de software da empresa;
- Permite que a alta gerência tenha uma maior compreensão de problemas, principalmente de ordem técnica, que ocorrem durante a gerência dos projetos da empresa;
- Possibilita a identificação de necessidades de treinamentos ao nível de projeto ou organizacional com base em tipos de problema freqüentemente identificados durante as avaliações. Por exemplo, fornecer cursos em otimização de sistemas quando as avaliações identificarem principalmente problemas arquiteturais relacionados à característica de desempenho.

A avaliação de documentos arquiteturais é um tema que tem sido bastante discutido no contexto de vários grupos de pesquisa, como no grupo do Software Engineering Institute (SEI) (KAZMAN et al., 1994; CLEMENTS et al., 2002), por exemplo.

Conclusão

Ao longo deste artigo foram descritos os principais conceitos em relação à arquitetura de software, dando ênfase principalmente nas atividades que estão relacionadas ao seu processo de especificação.

Através da análise desses conceitos e processos, foi possível identificar (1) a importância da arquitetura dentro do processo de desenvolvimento de software, (2) como esse artefato é construído e principalmente (3) que informações devem estar representadas nesse artefato e que devem ser analisadas durante o processo de avaliação para que se determine a corretude do documento arquitetural. ●

Referências

BABAR, M.A., ZHU, L., JEFFERY, R., 2004, "A framework for classifying and comparing software architecture evaluation methods". In: Proceedings of the Australian Software Engineering Conference, pp. 309-318, Melbourne, Australia, April.

BACHMANN, F., BASS, L., CHASTEK, G., DONOHOE, P., PERUZZI, F., 2000, The Architecture Based Design Method, CMU/SEI, Relatório Técnico, CMU/SEI-2000-TR-001.

BAHSOON, R., EMMERICH, W., 2003, "Evaluating software architectures: development, stability, and evolution". In: Book of Abstracts of the ACS/IEEE International Conference on Computer Systems and Applications, pp. 47, Tunis, Tunisia, July.

BASS, L., CLEMENTS, P., KAZMAN, R., 2003, Software Architecture in Practice, Second Edition, Addison Wesley.

BOEHM, B.W., 1981, Software Engineering Economics, Prentice-Hall.

BOSCH, J., MOLIN, P., 1999, "Software Architecture Design: Evaluation and Transformation". In: Proceedings of the IEEE Engineering of Computer Based Systems Symposium (ECBS´99), pp. 4, Nashville, TN, USA, March.

BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., STAL, M., 1996, Pattern-Oriented Software Architecture: A System of Patterns, Jon Wiley and Sons.

CLEMENTS, P., BACHMANN, F., BASS, L., GARLAN, D., IVERS, J., LITTLE, R., NORD, R., STAFFORD, J., 2004, Documenting Software Architectures, Addison-Wesley.

DIAS, M.S., VIEIRA, M.E.R., 2000, "Software architecture analysis based on statechart semantics". In: International Workshop on Software Specification and Design, pp. 133-137, Washington, DC, USA.

DOBRICA, L., NIEMELA, E., 2002, "A survey on software architecture analysis methods", IEEE Transactions on Software Engineering, v. 28, n. 7, pp. 638-653..

EGYED, A., MEDVIDOVIC, N., 1999, "Extending Architectural Representation in UML with View Integration". In: Proceedings of the 2nd International Conference on the Unified Modeling Language. Beyond the Standard (UML'99), v. 1723, pp. 2-16, Fort Collins, USA, October.

GACEK, C., 1995, On the Definition of Software System Architecture, University of Southern California, Relatório Técnico, USC/CSE-95-TR-500.

GARLAN, D., 2000, "Software architecture: a roadmap". In: Proceedings of The Conference on The Future of Software Engineering, pp. 91-101.

GARLAN, D., PERRY, D., 1995, "Introduction to the Special Issue on Software Architecture". In: IEEE Transactions on Software Engineering, v. 21, April.

HOFMEISTER, C., NORD, R.L., SONI, D., 2000, A study on agreement between participants in an architecture assessment, Addison-Wesley.

IEEE, 2000, "IEEE Recommended Practice For Architectural Description Of Software-Intensive Systems - IEEE Standard 1471-2000", Institute of Electrical and Electronics Engineers.

ISO/IEC, 1998, "International Technology - Software Product Evaluation - ISO/IEC 9126 Part 1: Quality Model".

KAZMAN, R., 2001, "Handbook of Software Engineering and Knowledge Engineering". In: CHANG, S.K. (eds), World Scientific Publishing.

KAZMAN, R., BASS, L., ABOWD, G., WEBB, M., 1994, "SAAM: a method for analyzing the properties of software architectures". In: Proceedings of the International conference on Software Engineering (ICSE), pp. 81-90.

KRUCHTEN, P., 1995, "Architectural Blueprints - The "4+1" View Model of Software Architecture". In: IEEE Software, v. 12, pp. 42-50, November.

LAITENBERGER, O., ATKINSON, C., 1999, "Generalizing Perspective-based Inspection to handle Object-Oriented Development Artifacts". In: Proceedings of the International conference on Software Engineering (ICSE).

MARANZANO, J.F., ROZSYPAL, S.A., ZIMMERMAN, G.H., WARNKEN, G.W., WIRTH, P.E., WEISS, D.M., 2005, "Architecture reviews: practice and experience", IEEE Software, v. 22, n. 2, pp. 34-43.

SHAW, M., 1995, "Some Patterns for Software Architectures".

SHAW, M., GARLAN, D., 1996, Software Architecture - Perspectives on an Emerging Discipline, Prentice Hall.

XAVIER, J.R., 2001, Criação e Instanciação de Arquiteturas de Software Específicas de Domínio no Contexto de uma Infra-estrutura de Reutilização, Dissertação de Mestrado, Programa de Engenharia de Sistemas e Computação - COPPE/UFRJ.

ZHU, L., BABAR, M.A., JEFFERY, R., 2004, "Mining patterns to support software architecture evaluation". In: Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture, pp. 25 - 34, June.

Planejamento de Testes a partir de Casos de Uso

De que se trata o artigo:

O artigo apresenta uma estratégia indicando como testes podem ser obtidos a partir dos casos de uso especificados para um projeto.

Para que serve:

Durante o desenvolvimento de um software, diversas estratégias para teste podem ser aplicadas. Ao longo deste artigo iremos adotar uma estratégia de geração de testes baseada em especificação, representada pelos casos de uso de um sistema. Assim, partiremos desta informação para a geração de casos e procedimentos (roteiros) de teste. Desta forma, este artigo apresenta de forma prática como efetuar a geração de casos de teste.

Em que situação o tema é útil:

A cada dia as atividades de teste de software vêm tendo sua importância aumentada dentro das organizações de desenvolvimento de software. O tema deste artigo agrega conhecimento a este cenário através do apoio às atividades do analista de testes.

Se observarmos nos diferentes livros tradicionais de Engenharia de Software, veremos que sempre existe um capítulo ou seção destinado a Teste de software. Porém, eles normalmente apresentam apenas informações básicas sobre esta atividade, como por exemplo, os diferentes níveis de teste que podem ser aplicado (ex: unidade, integração ou sistema), as técnicas de teste que podem ser aplicadas (ex: técnica funcional ou estrutural) e os critérios para seleção dos testes associados a estas técnicas. Por exemplo, no artigo "Introdução a Teste de Software" publicado na edição 01 da Engenharia de Software Magazine discutimos sobre alguns desses critérios: Particionamento em classes de equivalência, Análise do Valor Limite e Grafo de Causa-Efeito. Para cada critério, vimos como aplicá-los e um exemplo da sua aplicação para a geração de casos de teste. Mais informações sobre esses critérios de seleção dos testes podem ser obtidas em ROCHA (2001).

Arilo Cláudio Dias Neto
ariloclaudio@gmail.com
É Bacharel em Ciência da Computação formado na Universidade Federal do Amazonas, Mestre em Engenharia de Sistemas e Computação formado na COPPE/UFRJ, e atualmente está cursando doutorado na área de Engenharia de Software da COPPE/UFRJ. Possui 5 anos de experiência em análise, desenvolvimento e teste de software. É editor técnico das Revistas SQL Magazine e WebMobile, gerenciadas pelo Grupo DevMedia.

No entanto, no desenvolvimento de um software real normalmente os problemas são bem mais complexos do que aqueles tradicionalmente usados quando estamos conhecendo esses critérios para seleção dos testes (ex: indicar qual o maior valor em um conjunto, indicar se um campo número só contém caracteres válidos, dentre outros). Normalmente os problemas a serem resolvidos são representados através de cenários, que podem ser facilmente representados por Diagramas de Casos de Uso da UML (www.uml.org) aliada a uma descrição do que cada caso de uso deve fazer.

Ao longo deste artigo iremos discutir uma possível estratégia indicando como testes podem ser obtidos a partir dos casos de uso especificados para um projeto. Entendemos que podem existir diferentes estratégias para isso, então iremos apresentar apenas uma possibilidade que pode ser facilmente aplicada para o teste de formulários de cadastro, normalmente existentes em sistemas de informação.

Estratégias de Teste de Software

Durante o desenvolvimento de um software, diversas estratégias para teste podem ser aplicadas. De acordo com PRESSMAN (2005), essas estratégias podem ser categorizadas da seguinte forma:

- **Baseadas em implementação:** utiliza o código como elemento para a geração dos testes. É uma atividade cara, sob o ponto de vista de recursos necessários para a sua realização, e bastante complexa quando o tamanho do código se torna bastante grande. É totalmente dependente de apoio ferramental.
- **Baseadas em especificação:** utiliza um documento de especificação como base para geração dos testes. Assim, tenta-se cobrir as imposições e restrições descritas nos requisitos estabelecidos para o sistema. A automação da geração dos testes nesse caso é mais complicada, caso não se tenha um formalismo para a elaboração da especificação do sistema.
- **Baseadas em modelos:** é uma sub-categoria de estratégias baseada em especificação. Utiliza modelos desenvolvidos ao longo do processo de desenvolvimento que representam o comportamento ou estrutura do software como base para a geração dos testes. Facilita a geração automática dos testes, porem é completamente dependente da facilidade para a construção do modelo adotado e de sua qualidade (corretude).

Cada estratégia apresentada possui sua aplicabilidade, vantagens e desvantagens. Não é propósito deste artigo discutir qual seria a estratégia mais adequada. Ao longo deste artigo iremos adotar uma estratégia de geração de testes baseada em especificação, representada pelos casos de uso de um sistema. Assim, partiremos desta informação para a geração de casos e procedimentos (roteiros) de teste. Relembrando os conceitos associados a esses elementos, conforme descrito no artigo "Introdução a Teste de Software" publicado na edição 01 da Engenharia de Software Magazine, temos:

- **Caso de Teste.** Descreve uma condição particular a ser testada e é composto por valores de entrada, restrições para a sua execução e um resultado ou comportamento esperado (CRAIG e JASKIEL, 2002).
- **Procedimento (Roteiro) de Teste.** É uma descrição dos passos necessários para executar um caso (ou um grupo de casos) de teste (CRAIG e JASKIEL, 2002).

Porém, antes de descrevermos como obter testes a partir da especificação de casos de uso, precisamos primeiramente entender melhor este documento que servirá como entrada para a geração dos testes: a Especificação de Casos de Uso. Este artefato do processo de desenvolvimento servirá como **oráculo** para os testes, ou seja, os resultados obtidos pelo sistema devem sempre estar de acordo com os resultados previstos neste documento. A próxima seção irá discutir sobre a estrutura e conteúdo deste artefato.

Especificação de Casos de Uso

Um Caso de Uso representa uma unidade discreta da interação entre um usuário (humano ou máquina) e o sistema. Ele representa as funcionalidades que um sistema deve prover e uma indicação que qual elemento, denominado de ator, pode acessar uma determinada funcionalidade. Um ator é um humano

Caso de Uso: *<nome>*
Ator: *<nome do ator>*
Pré-Condições: *<pré-condições>*
Fluxo:
1. *<descrição de cada passo>*
2. *<descrição de cada passo>*
3. ...

Pós-Condições: *<pós-condições>*
Fluxos Alternativos:
X.1 *<fluxo alternativo a partir de um determinado passo X>*
Regra de Negócio:
(1) *<regra de negócio associada ao caso de uso>*

Exceções: *<exceções de execução do caso de uso>*

Figura 1. Template para especificação de caso de uso

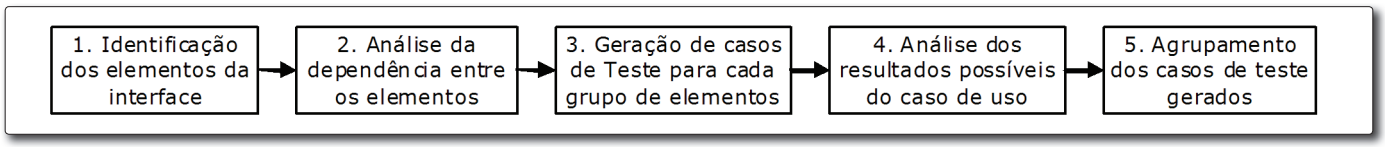


Figura 2. Processo de Geracao de Testes a partir de Casos de Uso

Quadro 1. Descrição do Caso de Uso

Caso de Uso: Cadastrar Funcionário
Ator: Administrador do Sistema
Pré-Condições: O administrador deve estar autenticado no sistema e deve ter acesso a este caso de uso
Fluxo:
1. Ator escolhe a opção Cadastrar Funcionário
2. Sistema abre um formulário a ser preenchido.
3. Ator preenche o formulário com os dados do funcionário a ser cadastrado.
4. Sistema valida os dados do funcionário e solicita confirmação.
5. Ator confirma o cadastro.
6. Sistema armazena os dados do novo funcionário.

Pós-Condições: O funcionário deve estar cadastrado no sistema.
Fluxos Alternativos:
5.1 Ator não confirma o cadastro.
5.2 Sistema volta ao passo 3.
Regra de Negócio:
(1) os campos a serem preenchidos para um formulário são: nome, data de nascimento, cargo (motorista, médico ou técnico em informática), CNH (caso seja motorista), CRM (caso seja médico), naturalidade (brasileira ou estrangeira), CPF (caso seja brasileiro), Passaporte (caso seja estrangeiro) e salário.
(2) os campos nome, data de nascimento, cargo, naturalidade e salário são obrigatórios.
(3) o campo CNH é obrigatório caso o cargo do funcionário seja "motorista" e o campo CREA é obrigatório caso o cargo seja "engenheiro".
(4) o campo CPF é obrigatório caso o funcionário seja "brasileiro" e o campo Passaporte é obrigatório caso seja "estrangeiro".
(5) cada tipo cargo possui uma faixa de salário possível que deve ser respeitada. As faixas são:
- Motorista: entre R$ 1000 e R$ 3000;
- Médico: entre R$ 3000 e R$ 10000;
- Técnico em Informática: entre R$ 1500 e R$ 7000;

Exceções: Dados não preenchidos corretamente ou salário fora da faixa de valores do cargo correspondente.

ou entidade máquina que interage com o sistema para executar um trabalho no contexto do sistema.

Este modelo foi incluído entre os diagramas disponíveis na UML. No entanto, o diagrama sozinho é totalmente limitado e não apresenta qualquer informação sobre o significado de tal funcionalidade. Sendo assim, cada Caso de Uso deve possuir uma descrição que deve descrever a funcionalidade que irá ser construída no sistema proposto.

É importante notar que o caso de uso não descreve como o software deverá ser construído, mas sim como ele deverá se comportar quando estiver pronto. Um software freqüentemente é um produto complexo, e sua descrição envolve a identificação e documentação de vários casos de uso, cada um deles descrevendo uma "fatia" do que o software ou uma de suas partes deverá oferecer.

Uma descrição de um caso de uso deve ser formada pelos seguintes elementos:

– **Nome:** Identificador inequívoco do caso de uso, deve ser escrito em formato de verbo/substantivo e ser suficiente para o utilizador perceber a que se refere o caso de uso.

– **Atores:** perfis de usuários que executam o caso de uso.

– **Pré-condições:** restrições para iniciar a execução de um caso de uso.

– **Seqüência de Ações (Fluxo principal):** passos ordenados para execução de um caso de uso.

– **Pós-condições:** condição final a ser estabelecida ao final da execução do caso de uso.

– **Fluxos Alternativos:** fluxos de ações que ocorrem paralelamente ao fluxo principal, dada uma ação específica.

Regras de negócio: restrições (regras) para execução de um ou mais passos do fluxo principal ou alternativo.

– **Exceções:** estados inválidos para o sistema.

A **Figura 1** descreve um exemplo de *template* para a especificação de um caso de uso.

Gerando Testes a partir de Casos de Uso

Uma vez garantida a qualidade da Especificação de Casos de Uso, artefato de entrada para a geração dos testes, devemos seguir alguns passos visando a obtenção de casos e procedimentos de teste para avaliação de cada funcionalidade do sistema, representada pelos casos de uso. A **Figura 2** representa os passos que compõem esta estratégia.

Para entendermos melhor esses passos, seguiremos um estudo de caso referente a um caso de uso bastante comum no nosso dia-a-dia de desenvolvedores: um formulário de cadastro.

Estudo de Caso para Geração de Testes: Cadastrar Funcionário

O caso de uso a ser testado está descrito no **Quadro 1**.

Além dessas informações, algumas suposições devem ser feitas para viabilizar o planejamento dos testes:

a) Na interface do sistema, o preenchimento incorreto de um campo será mostrado item a item, ou seja, se todos os campos forem preenchidos incorretamente, o sistema apresentará mensagem de dados inválidos para todos eles.

b) Existem campos que são obrigatórios a partir de certas condições (ex: CNH para motoristas), mas o sistema não impede que esse campo seja preenchido em outras situações.

c) Iremos assumir que os campos CNH e CRM não possuem regra de formação, como sabemos que existe para CPF. Assim, se qualquer valor for preenchido, ele será considerado válido.

d) Os campos numéricos só aceitarão valores numéricos (não preciso testar o contrário) e o campo data só aceitará datas (válidas ou inválidas).

e) Os campos cargo e naturalidade serão opções a serem escolhidas (com alguma das opções já selecionada previamente). Assim, estes campos nunca serão deixados em branco.

A seguir iremos passo a passo gerar os testes para este formulário.

Passo 1: Identificação do Elementos de Interface

Os elementos de interface que compõem um caso de uso são, por exemplo, campos, menus, links ou botões. Precisamos identificá-los para podermos iniciar o processo de geração dos testes para o caso de uso.

No nosso estudo de caso, partiremos da idéia de que o caso de uso só é executado após escolhermos a opção CADASTRAR FUNCIONÁRIO no menu principal da aplicação. Assim, os elementos de interface que fazem parte dessa interface são:

- Nome (String);
- Data de Nascimento (tipo Data);
- Cargo (lista de opções);
- CNH (String);
- CRM (String);
- Salário (Numérico);
- Nacionalidade (lista de opções);
- CPF (String);
- Passaporte (String).

Passo 2: Análise da Dependência entre os Elementos

Em seguida, precisamos observar as dependências entre os elementos de interface, como por exemplo, uma regra indicando que um campo só pode ser preenchido caso um outro campo tenha sido preenchido previamente.

No nosso estudo de caso, olhando as regras de negócio observamos as seguintes dependências:

- Os campos CNH, CRM e Salário dependem do Cargo selecionado.
- Os campos CPF e Passaporte dependem da Naturalidade selecionada.

Assim, este caso de uso possui 4 grupos de elementos independentes:

- Nome;
- Data de Nascimento;
- Cargo, CNH, CRM e Salário;
- Naturalidade, CPF e Passaporte.

Passo 3: Geração dos Casos de Teste para cada Grupo de Elementos

Identificados os grupos de elementos, devemos aplicar algum dos critérios de

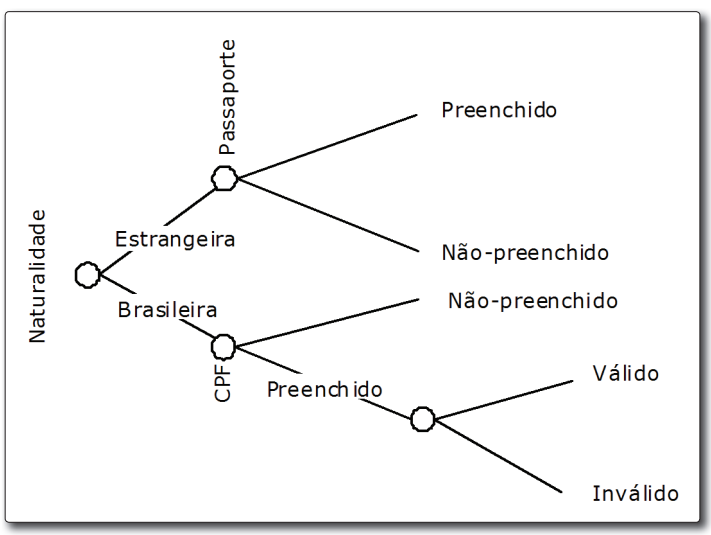
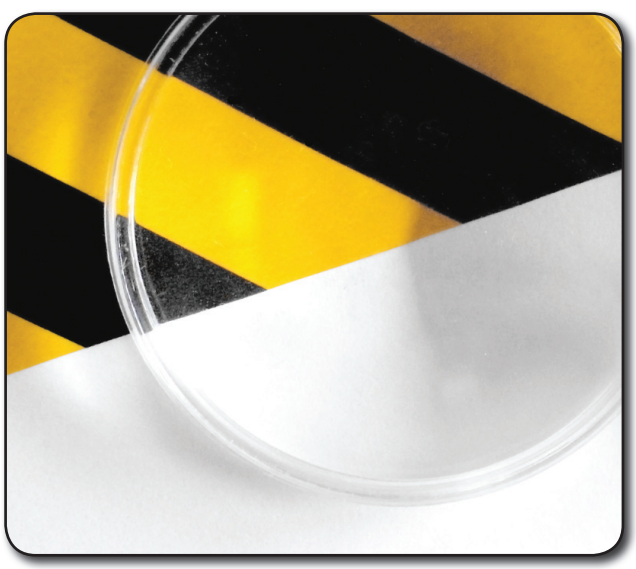


Figura 3. Grafo de Causa-Efeito para o Grupo (Naturalidade, CPF, Passaporte)



seleção de testes funcionais que vimos no artigo "Introdução a Teste de Software" para a geração de casos de teste para cada grupo. Essa seria uma forma de dividir o problema em partes menores para simplificar o processo de geração dos testes.

Os casos de teste gerados para cada grupo são os seguintes:

• (Nome): aplicamos o critério de Particionamento em Classes de Equivalência e obtivemos dois casos de teste, uma para a classe válida e outro para classe inválida (não preenchido).

T_{nome} = {(" ", INVÁLIDO); ("Arilo",VÁLIDO)}

• (Data de Nascimento): aplicamos o critério de Particionamento em Classes de Equivalência e obtivemos três casos de teste, uma para a classe válida, outro para data inválida (classe inválida) e outro para data não preenchida (classe inválida).

T_{data}={("",Inválido);("30/02/1982",Inválido); ("13/09/1982",Válido)}

• (Nacionalidade,CPF,Passaporte): aplicamos o critério de Grafo de Causa-Efeito e obtivemos cinco casos de teste, dois para o caso de nacionalidade estrangeira (um com passaporte preenchido e um com passaporte em branco) e três para o caso da nacionalidade brasileira (um com CPF não preenchido, um com CPF inválido [dígito verificador incorreto] e outro com CPF preenchido e válido), de acordo com a **Figura 3**.

$T_{nacionalidade}$ = {("Brasileira", "", --, [INVÁLIDO]); ("Brasileira", "782622652-97", --; [INVÁLIDO]); ("Brasileira", "636.112.337-57", ""; [VÁLIDO]), ("Estrangeira", --, "", [INVÁLIDO]); ("Estrangeira", "", "23243"; [VÁLIDO])}

• (Cargo,CNH,CRM,Salário): aplicamos o critério de Grafo de Causa-Efeito, sendo que para o campo Salário aplicamos ainda o critério de Análise do Valor

#	Caso de Teste (N)	Caso de Teste (D)	Caso de Teste (C)	Caso de Teste (N)	Resultado Esperado
1	""	""	("Motorista", "", --, --)	("Brasileira", "", --)	DADOS INVÁLIDOS
2	"Arilo"	30/02/1980	("Motorista","123456334", --, 999.99)	("Brasileira", "782622652-97", --)	DADOS INVÁLIDOS
3	""	14/05/2007	("Motorista","123456334", --, 3000.01)	("Brasileira", "636.112.337-57","")	DADOS INVÁLIDOS
4	"Arilo"	""	("Médico", "", --, --)	("Estrangeira", --, "")	DADOS INVÁLIDOS
5	""	30/02/1980	("Médico", --, "7625-2", 2999.99)	("Estrangeira", "", "23243")	DADOS INVÁLIDOS
6	"Arilo"	14/05/2007	("Médico", --,"7625-2"; 10000.01)	("Brasileira", --, "")	DADOS INVÁLIDOS
7	""	""	("Técnico", "", "", 1499.99)	("Brasileira", "782622652-97", --)	DADOS INVÁLIDOS
8	"Arilo"	30/02/1980	("Técnico", "", "", 7000.01)	("Estrangeira", --, "")	DADOS INVÁLIDOS
9	"Arilo"	14/05/2007	("Motorista","123456334",--, 1000)	("Brasileira", "636.112.337-57","")	CADASTRO REALIZADO
10	"Arilo"	14/05/2007	("Motorista","123456334", --, 3000)	("Estrangeira", "", "23243")	CADASTRO REALIZADO
11	"Arilo"	14/05/2007	("Médico", --, "7625-2", 3000)	("Brasileira", "636.112.337-57","")	CADASTRO REALIZADO
12	"Arilo"	14/05/2007	("Médico", --,"7625-2"; 10000)	("Estrangeira", "", "23243")	CADASTRO REALIZADO
13	"Arilo"	14/05/2007	("Técnico", "", "", 1500)	("Brasileira", "636.112.337-57","")	CADASTRO REALIZADO
14	"Arilo"	14/05/2007	("Técnico", "", "", 7000)	("Estrangeira", "", "23243")	CADASTRO REALIZADO

Tabela 1. Conjunto final de casos de teste para o caso de uso CADASTRAR FUNCIONÁRIO

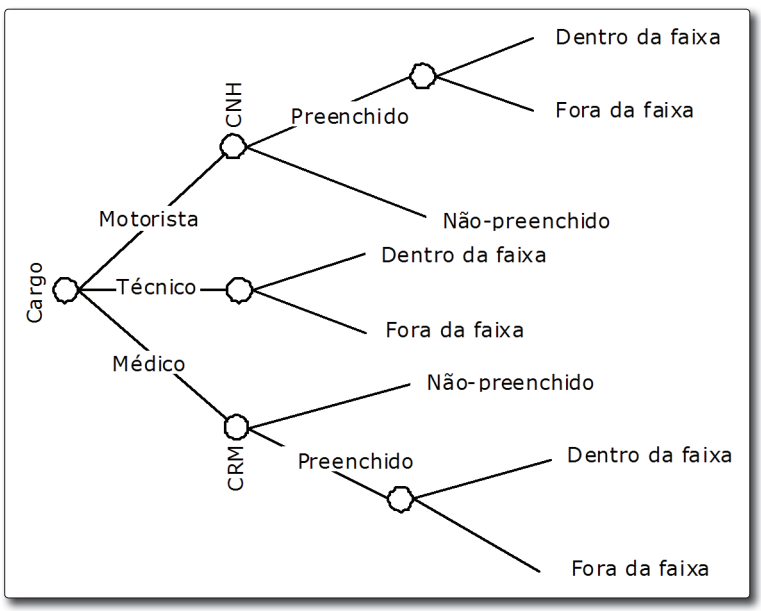


Figura 4. Grafo de Causa-Efeito para o Grupo (Cargo,CNH,CRM,Salário)

Limite. Com isso, obtivemos catorze casos de teste, cinco para o caso de cargo ser motorista, cinco para o caso do cargo ser médico e quatro para o cargo técnico em informática (de acordo com a **Figura 4**).

T_{cargo} = {("Motorista", "", --, --; [INVÁLIDO]); ("Motorista", "123456334", --, 999.99; [INVÁLIDO]); ("Motorista", "123456334", --, 1000; [VÁLIDO]); ("Motorista", "123456334", --, 3000; [VÁLIDO]) ; ("Motorista", "123456334"; --, 3000.01; [INVÁLIDO]); ("Médico", "", --, --; [INVÁLIDO]); ("Médico", --, "7625-2", 2999.99; [INVÁLIDO]); ("Médico", --,"7625-2"; 3000; [VÁLIDO]); ("Médico", --,"7625-2"; 10000; [VÁLIDO]) ; ("Médico", --,"7625-2"; 10000.01; [INVÁLIDO]); ("Técnico", "", "", 1499.99; [INVÁLIDO]); ("Técnico", "", "", 1500; [VÁLIDO]); ("Técnico", "", "", 7000; [VÁLIDO]); ("Técnico", "", "", 7000.01; [INVÁLIDO])}

Passo 4: Análise dos resultados possíveis do caso de uso

Gerados os casos de teste para cada grupo de elementos, o passo seguinte consiste na análise dos possíveis resultados providos pelo caso de uso. Para a geração dos casos de teste para um caso de uso, devemos atender às duas regras seguintes:

1. Deve cobrir todos os casos de teste gerados para cada grupo de elemento.
2. Deve existir ao menos 1 caso de teste para cada resultado possível gerado pelo caso de uso.

No nosso estudo de caso, os resultados possíveis da execução do cadastro de funcionários são apenas dois: CADASTRO REALIZADO COM SUCESSO ou DADOS INVÁLIDOS NO FORMULÁRIO a partir da violação de alguma regra de negócio.

Passo 5: Agrupamento dos Casos de Teste Gerados

O último passo é o agrupamento dos casos de teste gerados no passo 3. O agrupamento entre os casos de teste não precisa seguir uma regra, desde que atenda às duas anteriores descritas no passo 4. No entanto, precisamos considerar o seguinte cenário:

- Integrar dois casos de teste de resultados inválidos irá gerar um novo caso de teste também de resultado inválido.
- Integrar dois casos de teste de resultados válidos irá gerar um novo caso de teste também de resultado válido.
- Integrar um caso de teste de resultado inválido com um de resultado válido irá gerar um novo caso de teste de resultado inválido.

Sendo assim, o conjunto completo e mínimo de casos de teste para avaliação deste caso de uso está descrito na **Tabela 1**.

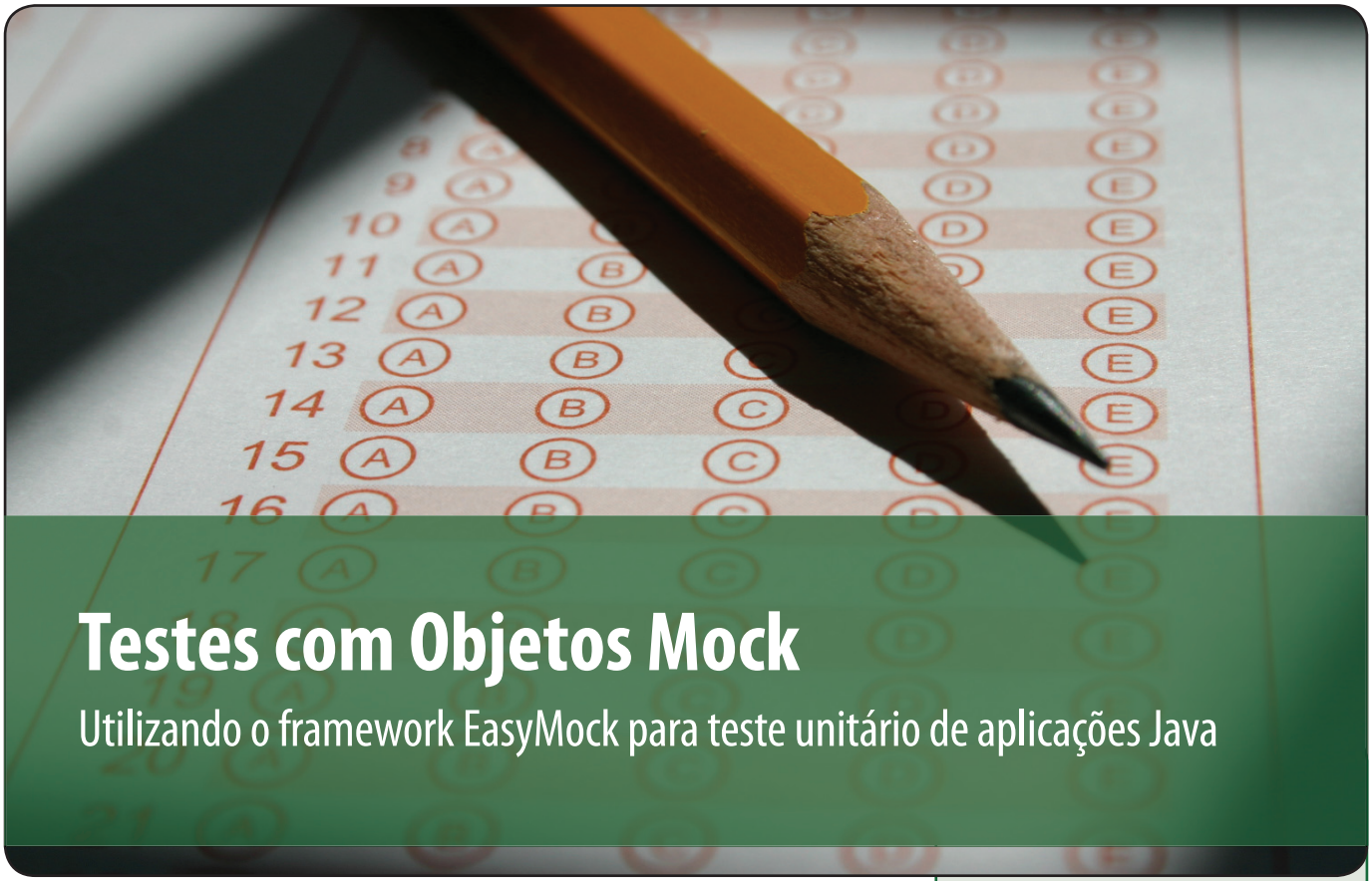
Conclusões

Este artigo apresentou uma possível estratégia para geração de casos de teste a partir de casos de uso, mas é preciso destacar mais uma vez que outras estratégias podem ser adotadas. Ao longo das nossas atividades do dia-a-dia, podemos nos deparar com situações que requeiram uma estratégia diferente da estratégia aqui apresentada. Cabe ao responsável pelos testes tomar a decisão de quais passos irá adotar para a geração dos testes, desde que mantenha em mente o objetivo de gerar testes de qualidade e que realmente possibilitam a avaliação de uma funcionalidade de um software.

Como passo seguinte do processo de teste, devem ser gerados os procedimentos de teste a fim de definir o roteiro/ordem de execução dos casos de teste gerados. As diretrizes para definição dos procedimentos de teste podem ser tema de um próximo artigo em uma edição futura. ●

Referências

CRAIG, R.D., JASKIEL, S. P., "Systematic Software Testing", Artech House Publishers, Boston, 2002.
PRESSMAN, R. S., "Software Engineering: A Practitioner's Approach", McGraw-Hill, 6th ed, Nova York, NY, 2005.
ROCHA, A. R. C., MALDONADO, J. C., WEBER, K. C. et al., "Qualidade de software – Teoria e prática", Prentice Hall, São Paulo, 2001.



Testes com Objetos Mock

Utilizando o framework EasyMock para teste unitário de aplicações Java

Bárbara de Melo Quintela
bmquintela@yahoo.com.br
É Mestranda em Modelagem Computacional na UFJF, cursando especialização em Engenharia de Software e Bacharel em Sistemas de Informação pelo CES/JF - Centro de Ensino Superior de Juiz de Fora, Analista de Sistemas e Programadora da CPM Braxis.

Marco Antônio Pereira Araújo
maraujo@cesjf.br
É Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Matemática com Habilitação em Informática pela UFJF, Professor do Curso de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora, Analista de Sistemas da Prefeitura de Juiz de Fora e Editor da Engenharia de Software Magazine.

Resumo DevMan

Os testes unitários são essenciais para garantir que menores unidades do software sejam testadas, mas essas unidades podem depender de outras partes do código que ainda não estão prontas. Outra situação refere-se à propagação do erro, onde é importante conseguir isolar uma determinada classe a ser testada, independente daquelas que são chamadas por ela, eliminando-se dúvidas sobre a origem do erro. Uma solução para esses casos é apresentada através da utilização de objetos mock, com o *framework* EasyMock. Neste artigo simulamos uma requisição web utilizando um objeto mock nos casos de teste para testar um método de um *servlet*.

De que se trata o artigo:

Uso do framework EasyMock para teste unitário de software Java, utilizando objetos mock. Neste artigo foi realizado o teste unitário de um servlet, sem precisar executar a aplicação web, utilizando objetos mock através do framework EasyMock para simular a requisição.

Para que serve:

Fornecer um meio para construir casos de teste utilizando objetos mock de forma ágil, permitindo que partes críticas da aplicação possam ser testadas de forma automatizada. Mostrar uma solução de teste para casos naturalmente difíceis de serem testados como, por exemplo, partes de código que dependem de outras partes que ainda não estão prontas.

Em que situação o tema é útil:

Em testes unitários de software Java para melhoria da qualidade do produto de software final, permitindo que partes críticas possam ser testadas desde o início do desenvolvimento.

Todo processo de software deve envolver em algum momento a fase de testes. O ideal seria que simplesmente todo o código pudesse ser testado exaustivamente para garantir que um software sem nenhum defeito fosse entregue ao cliente. Mas sabemos que mesmo com uma aplicação pequena, um teste completo, executado de forma exaustiva, seria inviável. Então, a forma que os analistas de teste, e outros profissionais que tenham que desempenhar este papel no processo de software, encontram para identificar os defeitos no software é concentrar os testes nas áreas mais críticas, como partes que serão mais utilizadas pelo usuário e partes que contenham um processamento mais complexo. Existem vários tipos de testes que podem ser utilizados de acordo com a necessidade.

Os testes unitários são essenciais para que seja possível testar a menor unidade do software como um método, uma classe ou mesmo um objeto. Mas essas unidades a serem testadas, principalmente as mais complexas, podem depender de outras partes do código que não queremos testar no momento, por que não estão prontas ou por que podem comprometer os resultados do teste gerando dúvidas sobre qual é a origem do erro. Uma solução para estes casos é a utilização de objetos mock.

Este artigo mostra um exemplo passo a passo da implementação de um teste unitário que utiliza objetos mock.

Objetos Mock e o Framework EasyMock

Os objetos mock são objetos "falsos" que simulam o comportamento de uma classe ou objeto "real" para que possamos focar o teste na unidade a ser testada. Os objetos mock podem ser extremamente úteis nas situações acima citadas onde é necessário criar um caso de teste e existem dependências entre os objetos. Como mostraremos a seguir, sua utilização é bastante simples e agiliza bastante o processo de construção de testes unitários.

Antes dos objetos mock, uma alternativa eram os stubs, objetos criados para substituir aqueles que seriam chamados numa troca de mensagem. Estes tipos de objeto, por um lado, facilitavam os testes,

Listagem 1. Implementação do método processRequest do Servlet

```
protected void processRequest(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html;charset=UTF-8");
String mensagem = "";
if (verificarAprovacao(request)){
    mensagem = "Aluno foi Aprovado!";
}else{
   mensagem = "Aluno foi Reprovado!";
}
request.setAttribute("mensagem",mensagem);
getServletContext().getRequestDispatcher("/index.jsp").forward(request,response);
}
```

Listagem 2. Implementação do método verificarAprovação do Servlet

```
public boolean verificarAprovacao(HttpServletRequest request) {
       String vBuffer_notafinal = "";
       String nome = request.getParameter("vNome");
       float nota1 = Float.parseFloat(request.getParameter("vNota1"));
       float nota2 = Float.parseFloat(request.getParameter("vNota2"));
       vBuffer_notafinal = request.getParameter("vNotaFinal");
       int frequencia = Integer.parseInt(request.getParameter("vFrequencia"));
       float notafinal;
       if (vBuffer_notafinal == ""){
          notafinal = 0;
       }else{
          notafinal = Float.parseFloat(vBuffer_notafinal);
       }
       Aluno objAluno = new Aluno(nome, nota1, nota2, notafinal, frequencia);
       return objAluno.calcularAprovacao();
}
```

Listagem 3. Classe Aluno

```
public class Aluno {
       private String nome;
       private float nota1;
       private float nota2;
       private float notaFinal;
       private int frequencia;

       public Aluno(String nome, float nota1, float nota2, float notaFinal, int frequencia){
              this.nome = nome;
              this.nota1 = nota1;
              this.nota2 = nota2;
              this.notaFinal = notaFinal;
              this.frequencia = frequencia;
       }

       public boolean calcularAprovacao() {
       float media;
       if (this.frequencia < 75) {
                return false;
       } else {
                media = (this.nota1 + this.nota2) / 2;
                if (media < 30) {
                       return false;
                } else {
                       if (media >= 70) {
                              return true;
                       } else {
                              if (((media + this.notaFinal) / 2) >= 50) {
                                     return true;
                              } else {
                                     return false;
                              }
                       }
                }
       }
}
```

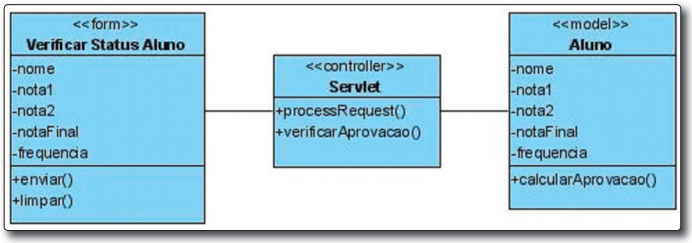


Figura 1. Interação entre os objetos envolvidos, sendo o formulário web, o servlet e classe Aluno

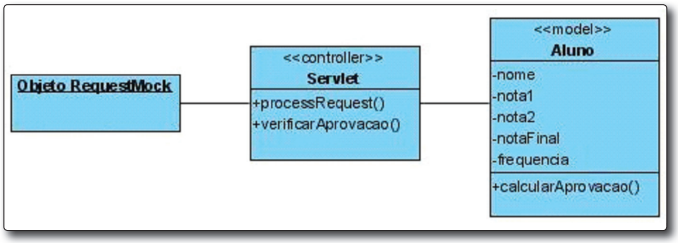


Figura 2. Representação do objeto mock no teste da aplicação

Listagem 4. Criação da classe de teste

```
import junit.framework.TestCase;
import static org.easymock.EasyMock.*;

public class TestesAprovacao extends TestCase{

}
```

mas ao mesmo tempo podiam demandar geração de muito código extra, já que em alguns casos era necessário gerar cópias das classes reais para realizar os testes. Os objetos mock não são stubs, mas poderíamos dizer que são tipos de stubs que requerem muito menos código. A principal exigência na utilização de objetos mock é a implementação de interfaces, que já são utilizadas em várias situações, por serem uma boa prática de programação orientada a objetos, e a sua inclusão no modelo de classes não requer muitas alterações.

Para criarmos os objetos mock utilizaremos o framework EasyMock, gratuito e de código aberto, que está disponível em http://sourceforge.net/projects/easymock.

Basta realizar o download, extrair os arquivos e adicionar os arquivos com extensão ".jar" ao projeto para começar a criar os objetos mock. É necessário também ter uma versão atualizada do JUnit instalada, uma vez que o EasyMock utiliza-se deste outro framework.

Estudo de Caso

Tomemos como exemplo uma aplicação web composta por uma página com um formulário de dados de alunos que chama um servlet ao clique de um botão. A página chama o servlet passando a requisição contendo os dados do formulário. O método que recebe esta requisição é o processRequest() do servlet, mostrado na **Listagem 1**.

O método processRequest() do servlet chama um outro método do servlet, verificarAprovacao() (ver **Listagem 2**), passando o objeto request que foi enviado pela página web. O objeto request contém os parâmetros com os dados que foram inseridos no formulário da página.

O método verificarAprovacao() é o alvo dos nossos casos de teste. Este método recebe o objeto request e obtém, a partir dele, os dados inseridos no formulário através da chamada ao getParameter(), passando o nome do respectivo parâmetro do formulário. Então é criado o objeto aluno chamado "objAluno", passando a freqüência e as notas informadas para o construtor do objeto e é feita uma chamada ao método calcularAprovacao() da classe Aluno (ver **Listagem 3**) que retorna *true* se o aluno foi aprovado ou *false* caso não tenha sido aprovado.

Teste Unitário utilizando Objetos Mock

O nosso objetivo é implementar os casos de testes para o método verificarAprovacao() do servlet. Mas para isso, seria necessário passar uma requisição HttpServletRequest no momento do teste, como o request passado pela página web na aplicação. Para isso, num primeiro momento, seria necessário executar a aplicação a partir do browser, uma vez que este objeto é passado pela página web para o servlet (**Figura 1**). Entretanto, para a construção de testes automatizados para o servlet, será necessário substituir o objeto criado pelo browser por um objeto mock.

O objetivo é realizar testes no método verificarAprovacao() do servlet e, para isso, podemos criar uma requisição mock. Será criada uma requisição falsa simulando uma requisição do tipo HttpServletRequest que é, na verdade, uma Interface, o que facilita a criação do objeto mock a partir dela.

Durante os testes, o formulário que envia os dados para o servlet não será executado em nenhum momento e o browser sequer será aberto. O objeto mock que será criado enviará a requisição diretamente para o servlet como se fosse a requisição do formulário da aplicação Web, conforme esquema da **Figura 2**.

Para quem conhece o JUnit, a estrutura do teste é semelhante e, para quem está começando nos testes unitários, não tem grande mistério. Será criada uma classe de teste que herda da classe TestCase do JUnit. Esta classe será chamada de TestesAprovacao.

É necessário incluir um import estático na biblioteca do EasyMock para criar os objetos mock, como mostrado na **Listagem 4**. Lembrando que os arquivos do framework EasyMock devem ter sido adicionados ao projeto anteriormente.

Casos de Teste

Para identificar quais casos de testes serão necessários para cobrir o método verificarAprovacao(), pode-se calcular sua Complexidade Ciclomática (CC) utilizando alguma ferramenta de métricas, ou então pode-se descobrir quais são as possibilidades de retorno de valor para este método, de acordo com os dados informados. Observando atentamente o método calcularAprovacao() da classe Aluno, verifica-se que se tem cinco possibilidades de retorno:

1. Freqüência inferior a 75, a função retorna *false*;
2. Freqüência igual ou superior a 75 e média inferior a 30, a função retorna *false*;
3. Freqüência igual ou superior a 75 e média igual ou superior a 70, a função retorna *true*;
4. Freqüência igual ou superior a 75 e média final igual ou superior a 50, retorna *true*;
5. Freqüência igual ou superior a 75 e média final inferior a 50, a função retorna *false*.

Com base nessas informações, conclui-se que, para este exemplo, tem-se cinco possibilidades de retorno da função verificarAprovacao() e serão implementados casos de teste para cada uma destas possibilidades.

Reprovação por Freqüência Insuficiente

O primeiro caso de teste que será criado é o que testa a reprovação por freqüência insuficiente. Para isso, a **Listagem 5** apresenta o método, na classe de testes TestesAprovacao, chamado testAlunoReprovadoInfrequencia().

O método inicia com a criação do objeto requestMock. A sua criação é bem simples, a única diferença da criação de um objeto comum está no fato de que, ao invés de chamar o construtor, chama-se o método createMock() passando como parâmetro a Inteface HttpServletRequest, ou seja, de onde será criado o objeto falso.

Entretanto, o objeto falso, como não foi instanciado a partir da classe original, não sabe como responder às chamadas de métodos. Então, as linhas seguintes, através do método expect, instruem o objeto mock como ele deve se comportar quando forem feitas requisições a ele, retornando o especificado no método andReturn(). Trata-se do treinamento do objeto mock e deve-se fazer isso por que estamos utilizando um objeto falso, que não veio de uma página web. Este treinamento termina com o método replay, estando o mock pronto para ser utilizado no teste. Desta forma, o mock foi instruído para retornar valores específicos para os métodos que deve responder. Deve-se perceber que é necessário fazer o treinamento para cada método chamado contendo a assinatura completa do mesmo, incluindo os parâmetros, caso existam.

Para que se possa então testar a reprovação por freqüência insuficiente, precisa-se apenas retornar um valor inferior a 75, no caso foi utilizado o valor imediatamente inferior.

Finalmente, nas duas últimas linhas estão respectivamente a criação do servlet servletControllerWeb e a execução do teste com a chamada ao verificarAprovação() deste servlet, passando o objeto mock que foi criado no teste. Como na reprovação por freqüência insuficiente sabe-se que o verificarAprovacao() deve retornar false, utilizou-se a função assertFalse() para fazer esta verificação. Ao clicar com o botão direito na classe de teste e executá-la, no NetBeans o resultado deve ser algo como exibido na **Figura 3**.

Listagem 5. Implementação do Caso de Teste testAlunoReprovadoInfrequencia()

```
public void testAlunoReprovadoInfrequencia() {
        HttpServletRequest requestMock = createMock(HttpServletRequest.class);
        expect(requestMock.getParameter("vNome")).andReturn("Marco");
        expect(requestMock.getParameter("vNota1")).andReturn("0");
        expect(requestMock.getParameter("vNota2")).andReturn("0");
        expect(requestMock.getParameter("vNotaFinal")).andReturn("0");
        expect(requestMock.getParameter("vFrequencia")).andReturn("74");
        replay(requestMock);
        ServletControllerWeb servletControllerWeb = new ServletControllerWeb();
        assertFalse(servletControllerWeb.verificarAprovacao(requestMock));
}
```

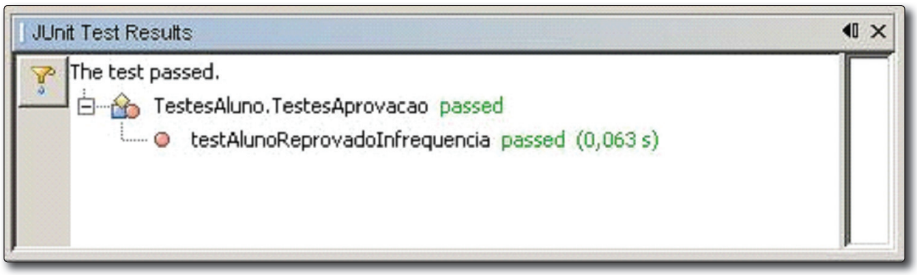


Figura 3. Resultado da execução de testAlunoReprovadoInfrequencia()

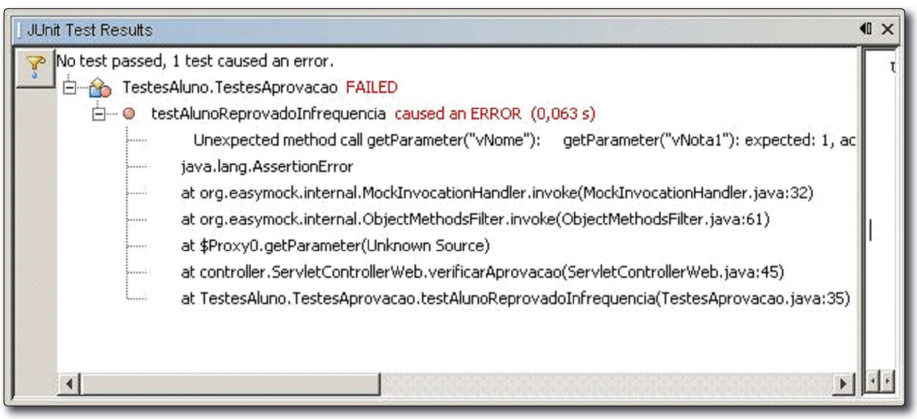


Figura 4. Resultado da execução de testAlunoReprovadoInfrequencia() com o teste modificado

O framework EasyMock permite ainda a criação de objetos mock que não retornem uma exceção caso ele não tenha sido treinado para uma determinada situação prevista na sua classe original. Para visualizar esta situação, pode-se comentar a primeira linha de treinamento do mock que contém o comando expect, fazendo o treinamento para o retorno do parâmetro que contém o nome do aluno.

Ao executar novamente o conjunto de testes, gera-se uma exceção por não estar preparado para responder quando é feita a requisição ao objeto com o parâmetro vNome (**Figura 4**).

Substituindo o construtor do mock de createMock para createNiceMock, pode-se perceber que o teste voltará a passar, mesmo com a linha comentada do treinamento para o parâmetro nome. Claro que este teste só pode ser feito uma vez que o parâmetro nome não era mais necessário no restante do teste.

Reprovação por Nota

O segundo caso de testes é bem semelhante ao primeiro. Será criado um novo método na classe de testes TestesAprovacao chamado testAlunoReprovadoNota(). Sua implementação é apresentada na **Listagem 6**.

Agora será necessário passar os parâmetros vNota1 e vNota2. O método deve retornar *false* se a freqüência for maior ou igual a 75 e a média dos parâmetros Nota1 e Nota2 for inferior a 30. Para a nota 2, está sendo passado o valor "29", para testarmos o valor limite. A utilização de testes exercitando os valores limite das condições é muito importante, pois, caso contrário, os testes podem estar retornando um resultado não verdadeiro.

Listagem 6. Implementação do Caso de Teste testAlunoReprovadoNota()

```
public void testAlunoReprovadoNota() {
       HttpServletRequest requestMock = createMock(HttpServletRequest.class);
       expect(requestMock.getParameter("vNome")).andReturn("Marco");
       expect(requestMock.getParameter("vNota1")).andReturn("30");
       expect(requestMock.getParameter("vNota2")).andReturn("29");
       expect(requestMock.getParameter("vNotaFinal")).andReturn("0");
       expect(requestMock.getParameter("vFrequencia")).andReturn("75");
       replay(requestMock);
       ServletControllerWeb servletControllerWeb = new ServletControllerWeb();
       assertFalse(servletControllerWeb.verificarAprovacao(requestMock));
}
```

Listagem 7. Implementação do Caso de Teste testAlunoAprovadoNota()

```
public void testAlunoAprovadoNota() {
       HttpServletRequest requestMock = createMock(HttpServletRequest.class);
       expect(requestMock.getParameter("vNome")).andReturn("Marco");
       expect(requestMock.getParameter("vNota1")).andReturn("70");
       expect(requestMock.getParameter("vNota2")).andReturn("70");
       expect(requestMock.getParameter("vNotaFinal")).andReturn("0");
       expect(requestMock.getParameter("vFrequencia")).andReturn("75");
       replay(requestMock);
       ServletControllerWeb servletControllerWeb = new ServletControllerWeb();
       assertTrue(servletControllerWeb.verificarAprovacao(requestMock));
}
```

Listagem 8. Implementação do Caso de Teste testAlunoReprovadoFinal ()

```
public void testAlunoReprovadoFinal() {
       HttpServletRequest requestMock = createMock(HttpServletRequest.class);
       expect(requestMock.getParameter("vNome")).andReturn("Marco");
       expect(requestMock.getParameter("vNota1")).andReturn("30");
       expect(requestMock.getParameter("vNota2")).andReturn("30");
       expect(requestMock.getParameter("vNotaFinal")).andReturn("69");
       expect(requestMock.getParameter("vFrequencia")).andReturn("75");
       replay(requestMock);
       ServletControllerWeb servletControllerWeb = new ServletControllerWeb();
       assertFalse(servletControllerWeb.verificarAprovacao(requestMock));
}
```

Listagem 9. Implementação do Caso de Teste testAlunoAprovadoFinal ()

```
public void testAlunoAprovadoFinal() {
       HttpServletRequest requestMock = createMock(HttpServletRequest.class);
       expect(requestMock.getParameter("vNome")).andReturn("Marco");
       expect(requestMock.getParameter("vNota1")).andReturn("30");
       expect(requestMock.getParameter("vNota2")).andReturn("30");
       expect(requestMock.getParameter("vNotaFinal")).andReturn("70");
       expect(requestMock.getParameter("vFrequencia")).andReturn("75");
       replay(requestMock);
       ServletControllerWeb servletControllerWeb = new ServletControllerWeb();
       assertTrue(servletControllerWeb.verificarAprovacao(requestMock));
}
```

Para exemplificar isso, basta substituir no código fonte um sinal de "<" por um de "<=", situação não muito difícil de ser confundida quando da implementação de um algoritmo. Executando os casos de teste novamente, tem-se o resultado dos dois casos de teste implementados até agora.

Aprovação por Nota

A terceira situação refere-se à aprovação de alunos por nota. Assim, será criado mais um método na classe TestesAprovacao chamado testAlunoAprovadoNota(). A diferença deste para o anterior está nos parâmetros vNota1 e vNota2, que agora devem retornar média igual ou superior a 70 e, no retorno do método verificarAprovacao() que agora deverá ser verdadeiro, sendo verificado pelo comando assertTrue(). Serão considerados os valores "70" para ambas as notas, para testar o valor limite. O código deste caso de teste é apresentado na **Listagem 7**.

Reprovação por Nota Final

Para ser reprovado por nota final, a média final ((média anterior + nota final) / 2) deve ser inferior a "50". Para criar o caso de teste que represente esta situação, vamos seguir o modelo na **Listagem 8**. O método se chamará testAlunoReprovadoFinal().

Aprovação por Nota Final

Finalmente, para a implementação do último caso de testes do estudo de caso, seguiremos a implementação descrita na **Listagem 9**.

Para nos certificarmos que será executado o trecho do método verificarAprovacao() que verifica se o aluno passou com a nota da final (e não somente com as duas primeiras notas), foram utilizados valores para vNota1 e vNota2 que retornariam que o aluno foi reprovado por nota. Ao informar o valor "70" no parâmetro vNotaFinal, verifica-se que o aluno foi aprovado.

Com isso, a classe que testa o método verificarAprovacao() do servlet está concluída, e com testes unitários automatizados, sem a necessidade de execução da aplicação via browser. Perceba que o intuito desde o início era testar o servlet, não a classe de domínio. Apenas para testar a classe de domínio, o framework

JUnit seria suficiente. Para finalizar, a **Figura 5** mostra a execução de todos os casos de teste implementados.

Verificando se todos os métodos treinados foram executados

Uma situação que ainda vale ser explorada é se todos os métodos treinados foram realmente executados em um teste. A **Listagem 10** apresenta uma modificação feita no método testAlunoReprovadoFinal. Perceba que foi inserida uma linha antes do comando replay, treinando o mock para retornar a matrícula de um aluno.

Neste caso, o teste continuará executando normalmente, pois a nova linha inserida não será chamada em nenhum momento pelo servlet, quando o teste for executado. Entretanto, poderíamos querer saber se todos os métodos treinados foram executados ao menos uma vez. Para isso, pode-se utilizar o comando verify(), conforme a última linha da **Listagem 11**.

Executando os testes desta forma, pode-se verificar que o EasyMock irá apresentar uma falha, uma vez que um determinado método que foi treinado não foi executado.

Conclusão

O objetivo deste artigo foi demonstrar a importância dos testes unitários automatizados no processo de desenvolvimento de software e como sua utilização pode ser implementada. Foi mostrada uma solução eficaz para situações onde é necessário testar objetos que necessitam de outros que ainda não foram implementados, ou simplesmente quando se deseja isolar determinados objetos a serem testados, eliminando a dependência deles com outros existentes. Esta solução baseia-se na utilização de objetos mock, ou objetos "falsos", através de frameworks específicos para este fim, como o EasyMock, apresentado neste artigo. ●

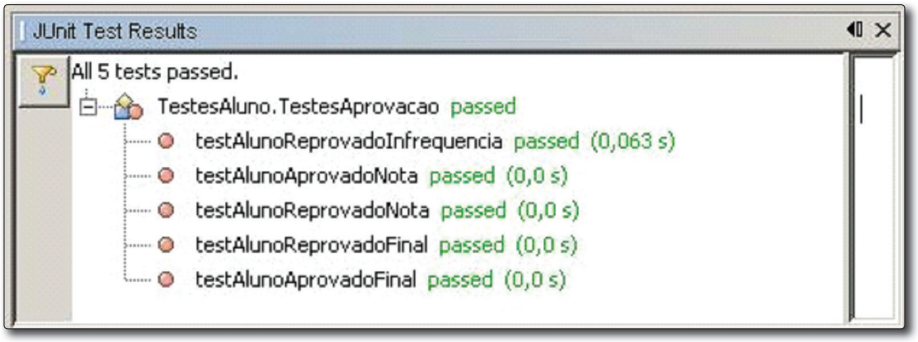


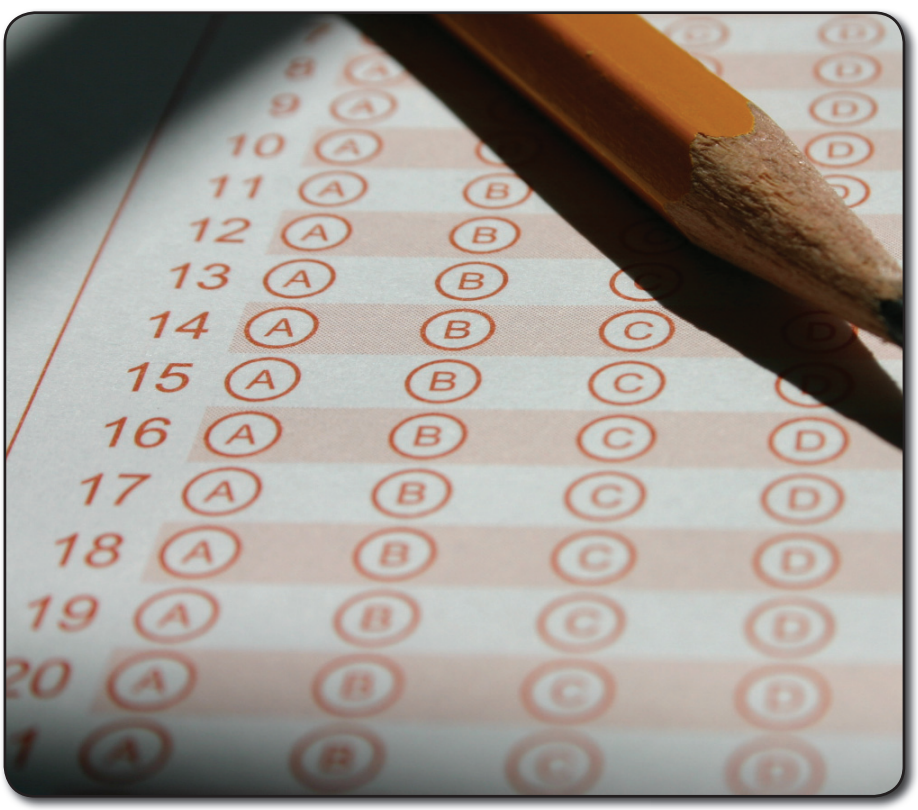
Figura 5. Resultado Final da Execução do Arquivo TestesAprovacao

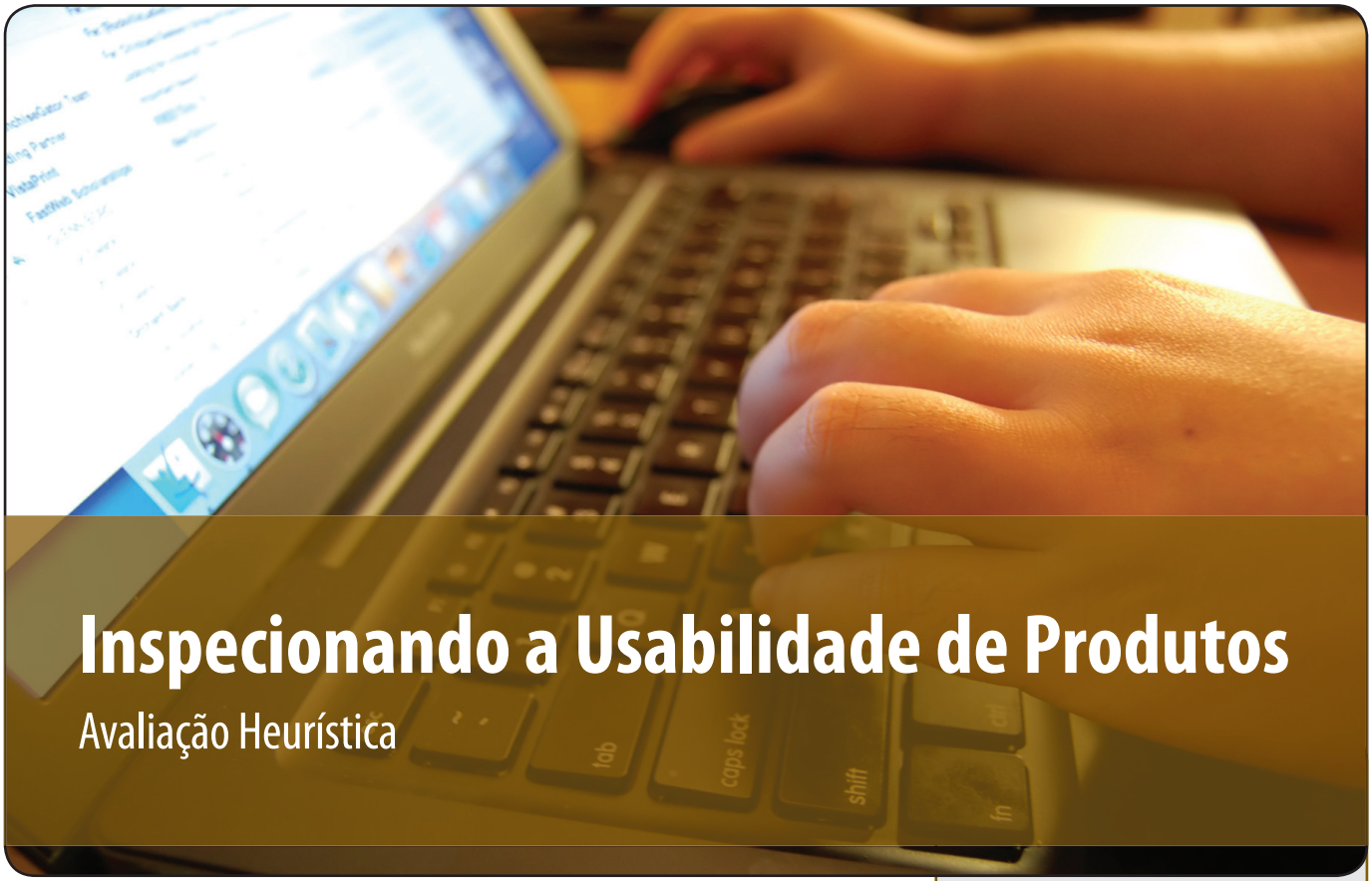
Listagem 10. Caso de Teste testAlunoReprovadoFinal () modificado

```
public void testAlunoReprovadoFinal() {
        HttpServletRequest requestMock = createMock(HttpServletRequest.class);
        expect(requestMock.getParameter("vNome")).andReturn("Marco");
        expect(requestMock.getParameter("vNota1")).andReturn("30");
        expect(requestMock.getParameter("vNota2")).andReturn("30");
        expect(requestMock.getParameter("vNotaFinal")).andReturn("69");
        expect(requestMock.getParameter("vFrequencia")).andReturn("75");
        expect(requestMock.getParameter("vMatricula")).andReturn("200821010");
        replay(requestMock);
        ServletControllerWeb servletControllerWeb = new ServletControllerWeb();
        assertFalse(servletControllerWeb.verificarAprovacao(requestMock));
}
```

Listagem 11. Caso de Teste testAlunoReprovadoFinal () com o comando verify()

```
public void testAlunoReprovadoFinal() {
        HttpServletRequest requestMock = createMock(HttpServletRequest.class);
        expect(requestMock.getParameter("vNome")).andReturn("Marco");
        expect(requestMock.getParameter("vNota1")).andReturn("30");
        expect(requestMock.getParameter("vNota2")).andReturn("30");
        expect(requestMock.getParameter("vNotaFinal")).andReturn("69");
        expect(requestMock.getParameter("vFrequencia")).andReturn("75");
        expect(requestMock.getParameter("vMatricula")).andReturn("200821010");
        replay(requestMock);
        ServletControllerWeb servletControllerWeb = new ServletControllerWeb();
        assertFalse(servletControllerWeb.verificarAprovacao(requestMock));
        verify(requestMock);
}
```





Inspecionando a Usabilidade de Produtos

Avaliação Heurística

De que se trata o artigo:

Uso do método de avaliação heurística para inspecionar a usabilidade de software. Neste artigo, um conjunto de heurísticas de projeto de interface de usuário é apresentado e vinculado ao método de avaliação para mostrar como a usabilidade de software pode ser avaliada.

Para que serve:

Prover um método através do qual um engenheiro de software ou avaliador pode avaliar a usabilidade de um produto de software. Este método pode ser empregado para identificar a maioria dos problemas de usabilidade de um produto.

Em que situação o tema é útil:

Pode ser empregado ao longo do desenvolvimento de um sistema de software, bem como após se obter o primeiro protótipo da interface de usuário do software, objetivando identificar problemas de usabilidade.

Simplicidade é um desejo de todo ser humano quando utiliza um produto. Simplicidade é um objetivo de projeto que todo engenheiro de software deve ter em mente quando projeta um novo produto. O engenheiro deve se colocar no lugar do usuário final do produto, buscando entender se as funcionalidades implementadas pelo sistema e a maneira pela qual elas podem ser acessadas são facilmente assimiladas pelos usuários. Ter essa preocupação é considerar a usabilidade como determinante no processo de desenvolvimento de um produto e, especificamente, de um sistema de software. Essa atitude impacta diretamente sobre a aceitabilidade e sucesso do produto. A edição anterior dessa revista tratou do processo de desenvolvimento de sistemas interativos e discutiu

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 60 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.

um conjunto de métodos de avaliação da usabilidade. Este artigo apresenta um método de avaliação da usabilidade, denominado avaliação heurística.

Usabilidade

Usabilidade é definida como a facilidade de aprender e usar um produto, que pode ser, por exemplo, um painel de um carro ou um sistema de software. Sua importância nos produtos tem sido explorada ao longo das últimas duas décadas e tem se tornado cada vez mais um determinante no sucesso dos produtos. Usabilidade conquista usuários e abre novos mercados. O que dizer do site de buscas do Google? Há tarefa mais simples do que fazer uma busca no Google? Considere o caso mais recente da Apple ao lançar o iPhone, como ilustrado na **Figura 1**. Ele possui um conjunto de funcionalidades que podem ser acessadas através do toque. O usuário seleciona as aplicações (como, por exemplo, ouvir música, enviar mensagem e visualizar fotos) que deseja utilizar apenas tocando e manipulando a tela. Ele pode ainda fazer zoom, ampliando ou reduzindo imagens, e tudo de forma intuitiva.

Perceba que a usabilidade é uma característica através da qual o usuário expressa sua satisfação em utilizar um produto ou serviço como, por exemplo, um site de uma instituição, um celular, um painel de carro ou um software instalado na sua máquina. Usabilidade resulta em simplicidade e agilidade. A simplicidade agrada os usuários e agilidade (leia-se e entenda produtividade) agrada as organizações.

Cabe aqui destacar que a usabilidade, um dos atributos da qualidade de um produto, está inserida no campo de projeto, de um sistema ou produto. Profissionais da área também costumam denominar de *design*. O design é um campo interdisciplinar que compreende atividades de concepção e projeto de um novo produto que pode, por exemplo, ser um veículo ou apenas seu painel, um novo modelo de telefone celular, uma interface gráfica de usuário de alguma loja virtual ou de um sistema de software (que você usa em seu computador pessoal ou notebook).

Design é, na realidade, uma atividade que abrange uma ampla gama de aplicações e, ao mesmo tempo, requer do designer projetista uma atenção especial com os aspectos funcionais e estéticos do produto. Isso também requer muita imaginação e habilidade para criar modelos e fazer ajustes iterativos e re-design. Os designers, assim como os artistas, têm sempre sido influenciados pelo ambiente onde vivem, e isto reflete exatamente o tempo e o lugar. Em outras palavras, o design, similarmente à arte, acompanha as necessidades de sua época. Antoine de Saint-Exupéry expressa isso bem quando diz:

> *"You know you've achieved perfection in design,*
> *Not when you have nothing more to add,*
> *But when you have nothing more to take away."*
>
> *[A perfeição no design não é alcançada*
> *Quando você percebe que não tem mais nada a adicionar,*
> *Mas quando você não tem mais nada a remover.]*

Segundo esse pensamento, é difícil saber quando se alcançou a perfeição no design. Entretanto, ele apontou que ela pode ser alcançada tornando-o mais simples, que podemos interpretar como *facilitar a vida do usuário*. Simplicidade no design é essencial para a satisfação do usuário e sucesso de um produto.

Uma questão que costumo fazer em diversas ocasiões é: *Por que o Google se tornou no site de buscas mais usado?* A resposta é: a necessidade humana por acesso a informação e simplicidade. O segundo motivo é, na realidade, o principal. Não há nada mais fácil do que fazer uma busca no Google. Você tem uma tela quase toda branca e apenas uma janela na qual você digita as palavras referentes ao conteúdo de seu interesse, como ilustra a **Figura 2**. Você, nem qualquer outra pessoa, precisa de treinamento ou auxílio para

Figura 1. iPhone da Apple.

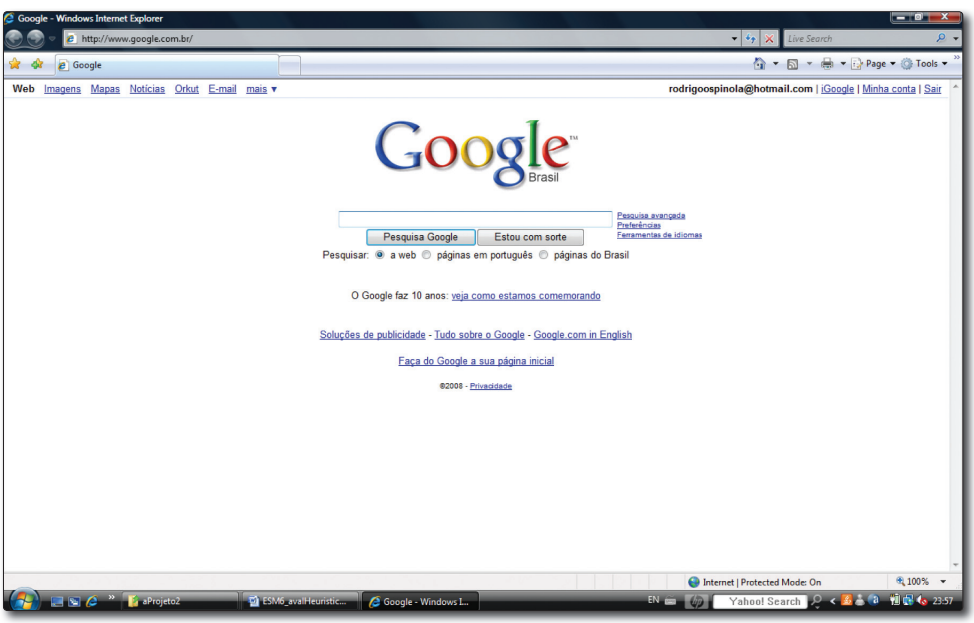


Figura 2. Simplicidade no site de buscas do Google.

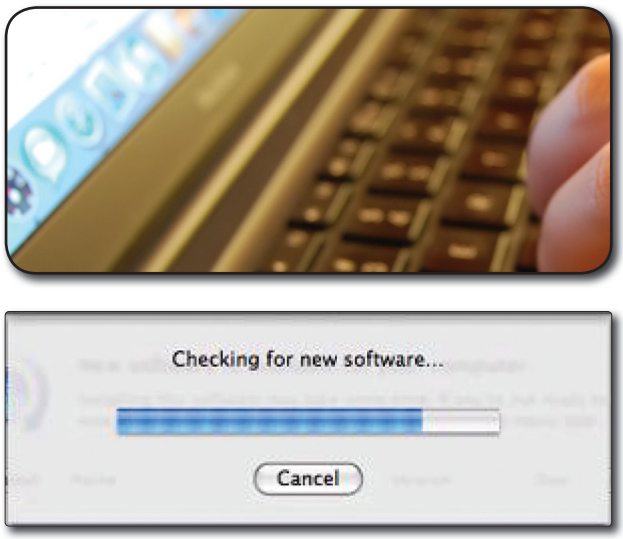


Figura 3. Uso de barra de progresso na checagem de novo software.

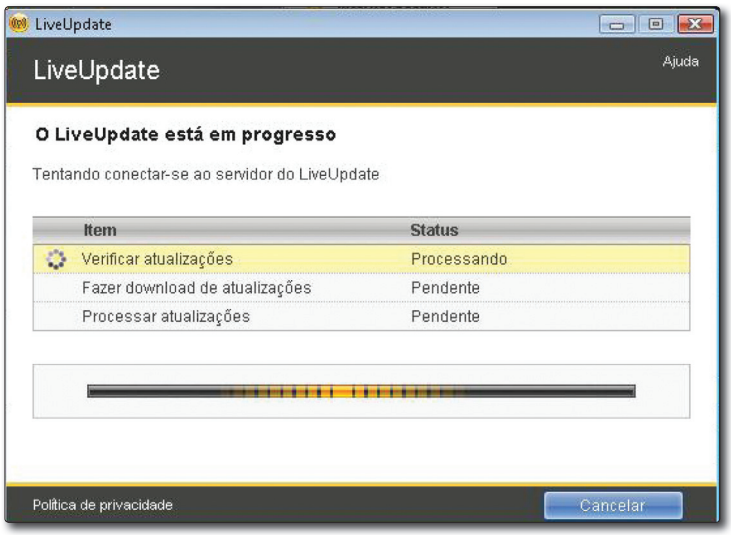


Figura 4. Uso de barra de progresso na checagem de novo software.

fazer isso. Por quê? Trata-se de um único serviço que é disponibilizado ao usuário e da forma mais simples possível.

Perceba que usabilidade é o que os usuários desejam, isto é, facilidade de uso e facilidade de aprender a usar, que se traduz em simplicidade. Para produzir interfaces de usuário que ofereçam suporte à usabilidade, recomendações e heurísticas de usabilidade são usadas pelos projetistas. As heurísticas compreendem conjunto de diretrizes, geralmente, que formam o conhecimento sobre para o tratamento de determinado problema. Esse problema poderia ser o projeto de uma rede neural, o projeto de uma casa ou projeto de uma interface de usuário de um sistema de software. Essas mesmas heurísticas servem de critério na avaliação heurística da usabilidade de um produto, como discutido a seguir.

Heurísticas de Usabilidade

Heurísticas de usabilidade são princípios resultantes da experiência e, geralmente, aceitos que são aplicados no desenvolvimento de sistemas interativos. Elas também são empregadas durante a avaliação de produtos. Um exemplo é oferecer recursos para desfazer ações quando usando um software, comumente conhecido como *Undo*, que permite você navegar até a revisão anterior de um documento (ou arquivo) de modo que você possa aceitá-la ou rejeitá-la. Portanto, prover mecanismo que permite ao usuário desfazer uma ação recente, como o exemplo do *Undo*, constitui uma das heurísticas de um conjunto a ser apresentado no artigo. Isto visa tornar a interface mais fácil de usar ao dar ao usuário a sensação de que ele está no controle do sistema e de que ele pode se recuperar de uma situação de erro, ou desfazer uma ação que ele fez de modo desatento e/ou indesejável.

Desde os primeiros esforços dos projetistas de interface de usuário para desenvolver produtos e sistemas de fácil uso, sempre houve a preocupação em prover suporte à usabilidade. Exemplo disso compreende as diretrizes de projeto, conhecidas como heurísticas ou *guidelines*, que contêm recomendações de projeto. Um exemplo de heurística é:

Prover os usuários com informações do estado do sistema.

Esta recomendação deve ser considerada, principalmente, em situações onde se têm operações de duração longa. Trata-se de um *feedback* que a aplicação (isto é, o sistema de software) provê ao usuário enquanto ele realiza uma tarefa no sistema. Isto é ilustrado na **Figura 3**, que mostra uma barra de status, comunicando ao usuário o quanto da tarefa (de checar pela disponibilidade de novo software) já foi realizada.

Outro exemplo é apresentado na **Figura 4** quando o usuário tem o *feedback* da tarefa de atualização de um software antivírus. Neste caso, trata-se de uma barra de progresso indeterminado que informa o usuário que a aplicação (isto é, software antivírus) está realizando a tarefa. Todavia, ela tem duração indeterminada já que depende da conexão de Internet para verificação da existência de disponibilidade de novas atualizações, bem como do download da respectiva atualização.

Os objetivos dessas *guidelines* são de munir o projetista de recomendações que pode orientá-lo durante o projeto da interface de usuário de um produto. Esses documentos contêm informações que orientam o projetista quanto a:

1. Organização (ou *layout*) da tela como, por exemplo, no uso de formatação e cores;
2. Navegação na interface, padronizando seqüência de realização de tarefas em situações semelhantes;
3. Facilitando acesso a funcionalidades através da flexibilidade de controle do usuário para entrada de dados ou execução de tarefas;
4. Mantendo a atenção do usuário com uso de diferentes fontes, cores e sons, visando alertar e dar *feedback* ao usuário.

Objetivando compilar essas recomendações, Jakob Nielsen apresentou um conjunto de 10 (dez) heurísticas para o projeto de interface de usuário, as quais estão disponíveis em http://www.useit.com/papers/heuristic/heuristic_list.html. As dez heurísticas de usabilidade, consideradas princípios para o projeto de interface de usuário são apresentadas na **Figura 5**.

As heurísticas mostradas no quadro acima servem a uma ampla variedade de sistemas de software e é um dos ingredientes do método de avaliação heurística apresentado a seguir. Exemplos

identificando problemas relacionados a essas heurísticas são mostrados e discutidos na seção final do artigo.

Avaliação Heurística

A avaliação heurística é um método de avaliação de usabilidade de sistemas e produtos que permite ao avaliador detectar problemas de usabilidade. Esses problemas, em geral, estão relacionados com alguma das dez heurísticas de usabilidade apresentadas anteriormente. O processo da avaliação heurística é apresentado a seguir.

Características

• Para aplicar o método de avaliação heurística, recomenda-se trabalhar com 3 a 5 avaliadores. Esses avaliadores são especialistas em usabilidade, como um engenheiro de usabilidade, ou engenheiro de software ou profissional similar que possua competência na área de usabilidade.

• A avaliação é feita individualmente por cada um dos avaliadores envolvidos e eles não mantêm qualquer contato ou colaboração antes da realização da avaliação. A comunicação entre os avaliadores é apenas permitida após o término da avaliação.

• Durante a avaliação, um especialista de domínio pode também ser consultado pelo avaliador a fim de esclarecer eventuais dúvidas sobre o sistema que está sendo avaliado.

• Cada avaliação individual pode durar de uma a duas horas, período no qual o avaliador obtém informações do sistema a ser avaliado e realiza a avaliação.

• Estudos (realizados por Jakob Nielsen) indicam que cerca de 75% dos problemas de usabilidade são detectados quando se tem a participação de cinco avaliadores.

Preparação para Avaliação

• Obter e ler a descrição do sistema a fim de entender seu propósito e conjunto de funcionalidades oferecidas.

• Conhecer os perfis de usuários e conjuntos de tarefas típicas realizadas por eles. Essas tarefas podem ser ilustradas através de cenários de interação que caracterize o uso do sistema.

• Selecionar o conjunto de heurísticas de usabilidade a serem utilizadas durante a avaliação. Essas heurísticas compreendem heurísticas gerais, como as apresentadas na **Figura 5** e recomendações ou *guidelines* referenciados no quadro de *links* no final do artigo.

Procedimento de Avaliação

• Reunião e estudo inicial do material disponibilizado durante a preparação para a avaliação.

• Cada avaliador deve realizar sua avaliação de maneira independente e sem qualquer participação dos demais. Isto visa evitar que haja qualquer avaliação 'viciada' em observações de problemas apontados por outros avaliadores.

• A avaliação é realizada em duas etapas: na primeira, o avaliador percorre a interface explorando as principais tarefas, objetivando ter uma visão geral da interface; na segunda etapa, percorre cenários de tarefas específicas, checando se esses cenários consideram os critérios (heurísticas de usabilidade) considerados.

• O avaliador faz anotações de problemas de usabilidade observados e atribui grau de severidade a esses problemas. A atribuição do grau de severidade do problema leva em conta a freqüência de ocorrência do problema, o nível de insatisfação do usuário (causado pelo problema) e a dificuldade do usuário em contornar o problema. Há cinco graus de severidade:

1. O problema é irrelevante e, portanto, desconsiderado.
2. Problema estético, o qual será apenas corrigido se o prazo permitir.
3. Problema simples (que tem baixa prioridade para correção).
4. Problema difícil (que tem alta prioridade para correção).
5. Problema danoso que requer correção imediata.

• Após serem encerradas as avaliações individuais de todos os avaliadores, estes se reúnem com os projetistas e eventuais observadores (especialistas do domínio), quando se tem início uma reunião que visa compilar as avaliações feitas pelos avaliadores e consolidar a avaliação final. Nesse momento, sugestões de como corrigir os problemas de usabilidade detectados são discutidas e apresentadas para que as correções possam ser implementadas.

Note que a avaliação heurística não deveria ser empregada isoladamente. Geralmente, ela é usada antes de se realizar testes com usuário. Isso objetiva identificar problemas grosseiros ou outros difíceis de serem detectados em testes com usuário como, por exemplo, aqueles que ocorrem em intervalos curtos de tempo e outros que acontecem de modo não freqüente. Além disso, evita-se um custo maior com testes com usuários que podem exigir mais tempo.

- **Visibilidade do estado do sistema** – Prover o usuário de *feedback* apropriado.
- **Casamento do sistema com o mundo real** – Utilizar termos, objetos e conceitos familiares à linguagem do usuário.
- **Controle e liberdade de escolha do usuário** – Oferecer recursos como *Undo* que permita o usuário desfazer ações realizadas e retornar, por exemplo, a revisão anterior de um documento.
- **Consistência e aderência a padrões** – Seguir recomendações dadas em *guidelines* e guias de estilo, visando prover suporte à consistência.
- **Prevenção de erros** – Identificar e eliminar situações que possam levar a erros do usuário.
- **Flexibilidade e eficiência de uso** – Considerar a diversidade de usuários (novatos e experientes), provendo mecanismos apropriados, como o uso de teclas de atalho, ícones e menus.
- **Estética e projeto minimalista** - Não adicionar informações desnecessárias ou raramente utilizadas, que competem com informações relevantes.
- **Prover ajuda aos usuários de reconhecimento, diagnóstico e recuperação de erros** – Mensagens de erro devem ser construtivas, sugerindo solução para o usuário.
- **Ajuda e documentação** – Prover documentação e recursos de ajuda, facilitando a busca e com foco nas tarefas do usuário.

Figura 5. Heurísticas de Usabilidade propostas por Jakob Nielsen.

Figura 6. Exemplo de site de um Buffet.

Figura 7. Exemplo de site de instituição de ensino superior.

Identificação de Problemas de Usabilidade com Heurísticas

Considere a **Figura 6** que ilustra parte de um site de Buffet. Note que o *layout* do espaço tem as principais informações distribuídas na tela e uso de cores é adequado. O projetista fez uso de pequena quantidade de cores que não excede a quatro que é quantidade recomendada. Além disso, ele utiliza figuras, relacionadas às informações que são disponibilizadas para os usuários, que estão distribuídas de modo adequado na página principal.

Entretanto, perceba na **Figura 6** que há o uso das palavras "Clique aqui" que não deveriam ser utilizadas como elementos do design. Evitar o uso do "Clique aqui", como texto âncora para um link de hipertexto, é uma das recomendações mais conhecidas do design. Note que esse problema está vinculado à quarta heurística (vide **Figura 5**), que recomenda a aderência a padrões (isto é, conjunto de recomendações ou *guidelines* de projeto). Numa situação como essa, o projetista poderia, por exemplo, ter usado:

> Conheça o nosso variado número de **cardápios** para todas as ocasiões.

Na sugestão acima, é feito o uso de um texto âncora, servindo de link de um hipertexto que o usuário pode ter acesso. Note que fica mais intuitivo essa forma de acesso para o usuário.

Outro exemplo ilustrado na **Figura 7** mostra parte do site de uma instituição de ensino superior. Alguns problemas de usabilidade estão indicados na própria figura.

Primeiramente, o site contém um excesso de informações na página principal. Muitas informações irrelevantes estão

apresentadas logo na página principal. A oitava regra diz que devemos considerar a estética e fazer um projeto minimalista, ou seja, informações desnecessárias ou de menor relevância deveriam ser tratada em segundo plano e não serem exibidas na página principal. Isto porque essas informações competem pela atenção do usuário com informações relevantes.

Note que o projeto deste site reserva espaço no lado inferior direito para cursos específicos de Fisioterapia e Direito. E quanto aos demais cursos? Além disso, no lado esquerdo há dois locais através dos quais usuários podem acessar o correio eletrônico da instituição (webmail) e um clube (específico) da instituição. O que isso tem de relevante para outros usuários interessados em obter informações da instituição. Perceba que o *layout*, embora exibido apenas parcialmente, precisa ser repensado e re-projetado, objetivando atender a heurística de estética e projeto minimalista.

Um bom exemplo de um projeto de site de instituição de ensino é o da University of Stanford disponível em http://www.stanford.edu, cuja página é reproduzida na **Figura 8**. Perceba o *layout* da página que faz distribuição do conteúdo, há preocupação com a estética e o projeto também é minimalista, pois apenas informações consideradas mais importantes são colocadas em primeiro plano, isto é, na página principal. Além disso, a página tem um recurso de mapa do site no canto superior direito, que permite ao usuário visualizar mais informações e, eventualmente buscar por informações mais detalhadas. Recurso adicional que oferece mais detalhes ao usuário encontra-se no canto superior esquerdo que permite ao usuário ter uma visão expandida do menu. Esta adequação do site ocorre porque o projetista considerou a terceira heurística que recomenda prover o usuário de controle e liberdade escolha.

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

Comentários Finais

Considerar as dez heurísticas de usabilidade apresentadas anteriormente e outras recomendações é dever do projetista e estas são consideradas pelo avaliador quando avalia a interface de um software ou outro produto. O usuário também percebe isso quando manifesta sua insatisfação no uso do software ou quando comete erros em situações nas quais ele não pode desfazer alguma ação realizada anteriormente, em situações que um usuário (mais experiente) não dispõe de teclas de atalho que otimizariam seu tempo, em situações onde um ícone da barra de ferramentas não tem sua funcionalidade reconhecida pelo usuário, ou ainda quando ele acha difícil localizar um funcionalidade num software. Neste contexto, as heurísticas e outras diretrizes (ou *guidelines*) de usabilidade servem para guiar tanto projetistas quanto avaliadores em suas atividades. ●

Figura 8. Exemplo de site de instituição de ensino superior.

Links

Apple Computer, Inc., Introduction to Apple Human Interface Guidelines
http://developer.apple.com/documentation/UserExperience/Conceptual/AppleHIGuidelines/XHIGIntro/chapter_1_section_1.html

Fundamentals of Designing User Interaction do livro Microsoft Windows User Experience
http://www.microsoft.com/mspress/books/toc/2466.aspx

Microsoft, Windows Vista User Experience Guidelines
http://msdn.microsoft.com/en-us/library/aa511258.aspx

NASA Goddard Space Flight Center - Usability Engineering Center – Handbook for Designing a Usable Web Site
http://software.gsfc.nasa.gov/AssetsApproved/PA2.3.1.2.pdf

Sun Microsystems, Inc., Java Look and Feel Design Guidelines
http://java.sun.com/products/jlf/ed1/dg/higtoc.nf.htm

Bad Human Factors Design
http://www.baddesigns.com/

Heuristics for User Interface Design
http://www.useit.com/papers/heuristic/heuristic_list.html

Design Guidelines for the Web
http://www.usabilitynet.org/tools/webdesign.htm

The Usability Methods Toolbox
http://jthom.best.vwh.net/usability/

Usability.gov
http://www.usability.gov/

Soluções para Gerenciamento de Riscos de Projetos

De que se trata o artigo:

Apresentação de uma ferramenta para apoiar o gerenciamento de riscos.

Para que serve:

Fornecer exemplo de aplicativo para suporte às atividades de gerenciamento de riscos em projetos.

Em que situação o tema é útil:

Em uma organização um programa de gerenciamento de riscos tem o objetivo de avaliar e controlar os riscos existentes e assim decidir como serão tratados. Desta forma, o uso de ferramental de apoio favorece o acúmulo de experiências, ajuste da organização no nível de maturidade próprio e, por conseguinte, adequação do respectivo processo de Gerência de Riscos, de acordo com as limitações organizacionais.

Cristine Gusmão
cristine@dsc.upe.br
Professora Assistente do Departamento de Sistemas e Computação da Escola Politécnica da Universidade de Pernambuco (POLI – UPE), onde leciona várias disciplinas na graduação e pós-graduação (especialização e mestrado) e das Faculdades Integradas Barros Melo. Doutora e Mestre em Ciência da Computação pela Universidade Federal de Pernambuco. Graduada em Engenharia Elétrica – Eletrotécnica pela Universidade Federal de Pernambuco.

A Gerência de Riscos é uma disciplina bastante importante em ambientes de desenvolvimento de software, permitido aos gerentes e membros de equipes o alcance de seus objetivos na execução de um projeto, contribuindo para um melhor tratamento das ameaças e das oportunidades. Ao disponibilizar visões de suporte e favorecer o compartilhamento das informações geradas, o gerenciamento dos riscos permite uma melhor tomada de decisão.

Com base nessas premissas, foi idealizada a ferramenta *m*PRIME *Tool* (www.cin.ufpe.br/~suppera) que através de suas funcionalidades dá suporte à avaliação, tratamento e controle dos riscos em um ambiente organizacional. A problemática tratada inclui o gerenciamento dos riscos nas atividades organizacionais nos níveis operacional, tático e estratégico.

Visão Geral

A *m*PRIME *Tool* é uma ferramenta de gestão de riscos para ambientes de múltiplos projetos de desenvolvimento de software, desenvolvida como *add-in* para o Microsoft® Project, uma ferramenta já bastante difundida no ambiente de Gestão de Projetos. Sua definição teve por base estudos acadêmicos dentro do Centro de Informática da Universidade Federal de Pernambuco (CIn –UFPE).

Utilizando componentes de inteligência artificial – ontologia (uma ontologia é uma visão simplificada de um domínio de conhecimento [Gruber 1995]) de riscos fundamentada na Taxonomia de Riscos (*Taxonomy Risk-based*) do *Software Engineering Institute* (SEI) – a *m*PRIME *Tool* auxilia na execução das atividades de identificação, monitoração e controle dos riscos.

O principal diferencial está na disponibilidade de funções que levantam situações de riscos, através de listas de verificação, facilitando a identificação de riscos de projetos e riscos entre projetos. Além disso, a *m*PRIME *Tool* é aderente ao *Capability Maturity Model Integration* (CMMI) [SEI 2001], pois suporta as fases definidas por este modelo através de funcionalidades para a área de processo de gerenciamento de riscos, contribuindo na qualidade do processo utilizado.

O conjunto das principais características da *m*PRIME *Tool* foi definido para suportar os níveis de uma hierarquia organizacional e suas respectivas fontes de riscos.

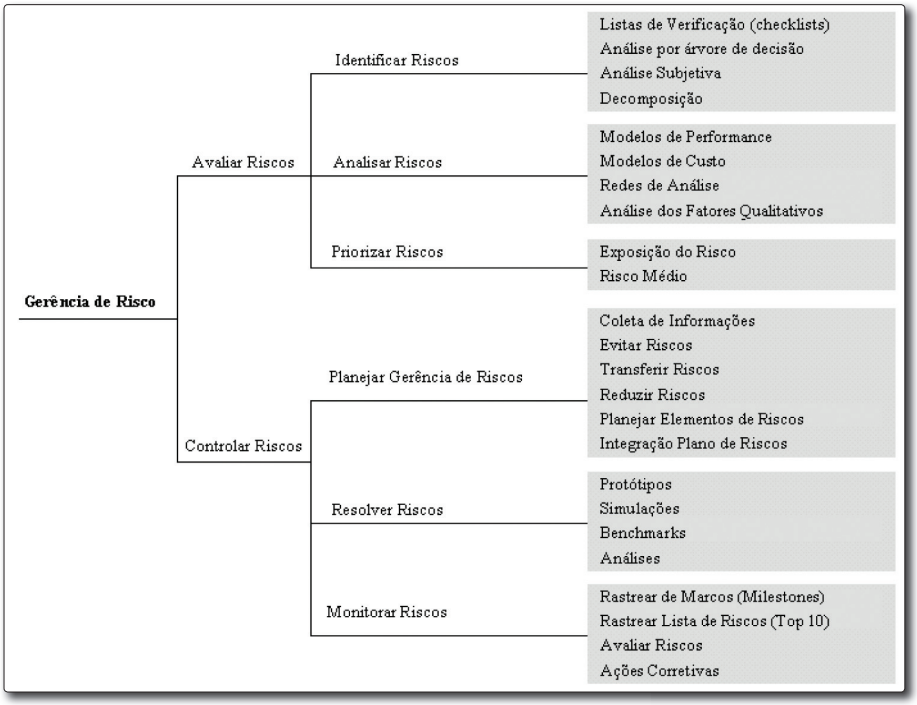


Figura 1. Atividades comuns de um processo de Gerenciamento de Riscos

Principais Funcionalidades

A *m*PRIME *Tool* foi desenvolvida para suportar um processo integrado de Gerência de Riscos Organizacional. Dentro de uma organização podem-se ter vários tipos de riscos associados aos níveis estratégico, tático e operacional. Devido à complexidade na construção dessas funcionalidades, os requisitos essenciais foram divididos entre os níveis organizacionais, facilitando a construção de versões diferenciadas e evolutivas da *m*PRIME *Tool*. Para cada uma das versões definidas, foram modeladas as funcionalidades requeridas da ferramenta.

A versão que trata das questões associadas às necessidades do nível operacional foi desenvolvida e avaliada. As principais funcionalidades estão associadas à disponibilização das atividades necessárias a um processo de gerenciamento de riscos.

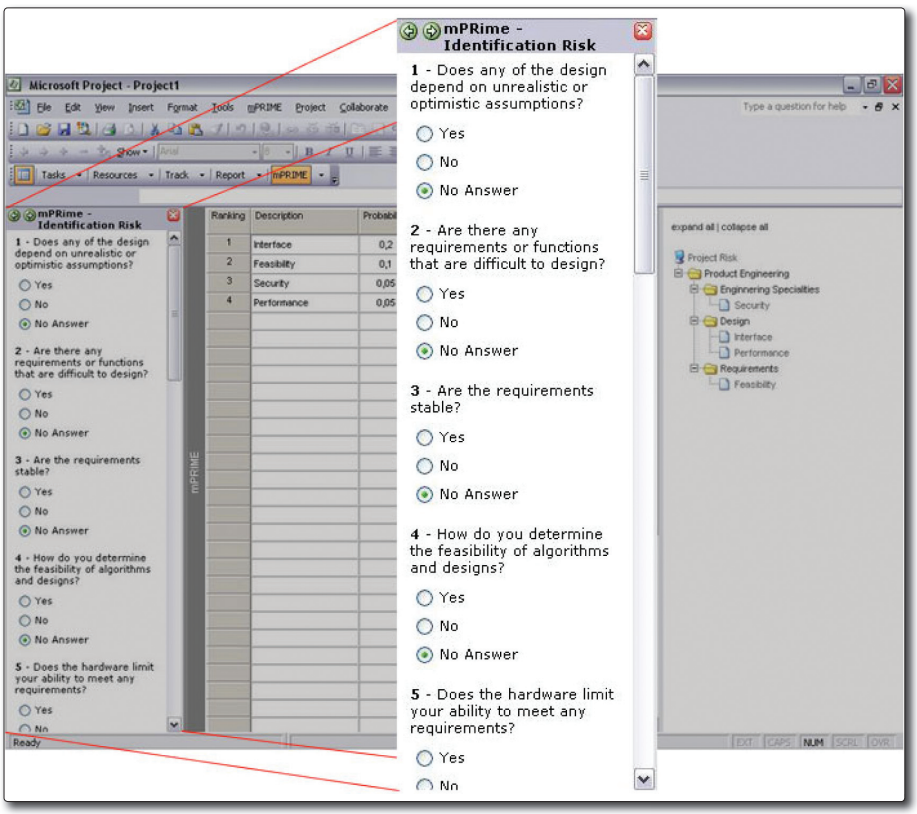


Figura 2. *m*PRIME *Tool*: Tela de Identificação de Riscos

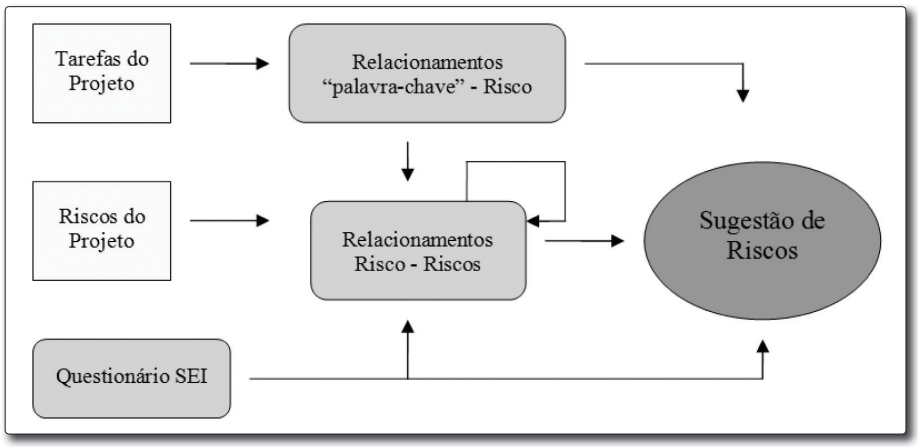


Figura 3. Modelo de uso da *m*PRIME *Ontology*

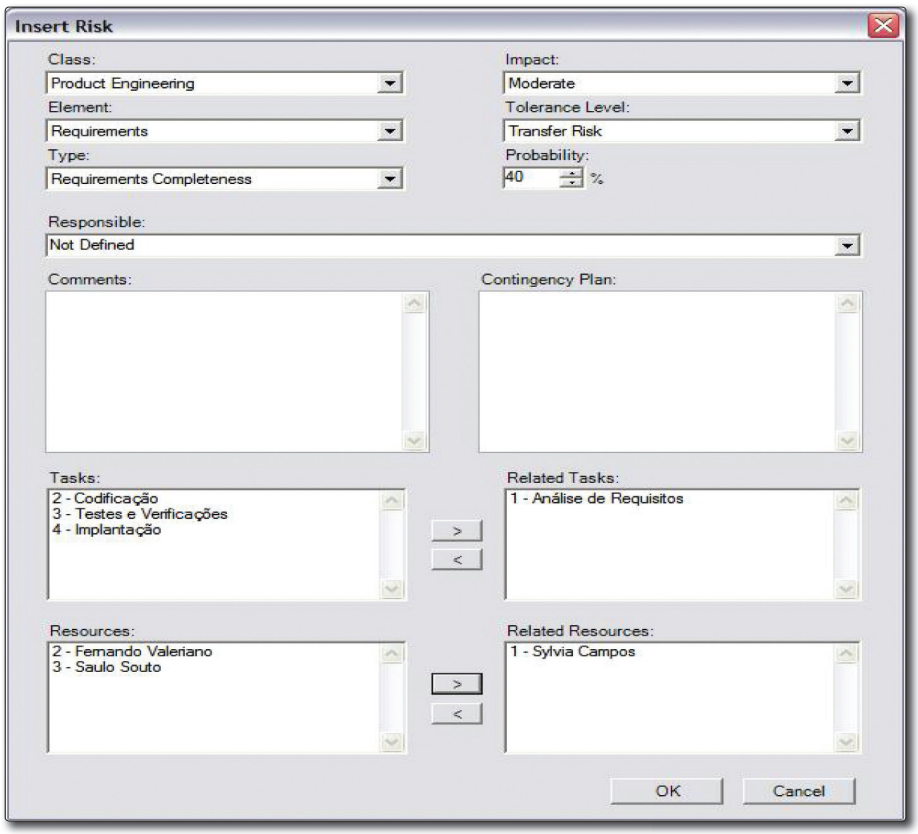


Figura 4. Tela para análise qualitativa do risco.

Vários processos/abordagens de apoio ao gerenciamento de riscos de projetos podem ser encontrados na literatura [SEI 2001, PMI 2004, Gusmão 2007].

A seguir será apresentado conjunto de atividades que são comuns nesses processos/abordagens, como pode ser visualizado na **Figura 1**.

1. *Planejar a Gerência de Riscos*

A *m*PRIME *Tool*, por ser integrada ao MS Project, suporta o planejamento dos recursos de hardware, software e pessoal necessário à realização da Gerência de Riscos, auxiliando o gerente do projeto a definir um plano estratégico para tratar cada um dos riscos.

2. *Identificar Riscos*

Para auxiliar o gerente no momento da identificação dos riscos, a *m*PRIME *Tool* oferece uma série de possibilidades, tendo como forma mais elementar a inserção manual de riscos que o próprio gerente tenha identificado. Porém, a forma mais relevante de identificação é através do uso de uma ontologia de riscos (*m*PRIME *Ontology* [Gusmão 2007; Costa e Gusmão 2008]), onde são sugeridos riscos referentes ao projeto. Essa sugestão pode ser feita através de uma análise inteligente da lista de tarefas do projeto (WBS – *Work Breakdown Structure*) ou através das listas de verificação (*checklists*) contidas na ferramenta, como mostra a **Figura 2**.

Através da visualização da **Figura 2** é possível identificar três partes: (i) à esquerda (ressaltada) está a lista de verificação apresentando o conjunto de situações que associam riscos a questões de requisitos de um produto; (ii) na parte central é visualizada a lista dos quatro riscos analisados até o momento para o projeto, e; (iii) à direita é apresentada a árvore de relacionamento entre os riscos identificados/analisados até o momento para o projeto.

O processo é composto por um conjunto de atividades que permitem identificar, analisar, documentar, acompanhar e monitorar os riscos. A *m*PRIME *Ontology* é usada principalmente na fase de identificação de riscos, permitindo à *m*PRIME *Tool* sugerir riscos para o projeto que podem ser aceitos pelo gerente ou não. Estas sugestões podem ser feitas através do uso de seis funcionalidades

Ranking		Exposure	Description	Probability	Impact	Responsible
1			Coding Implementation	90%	Catastrophic	Sylvia Campos
2			Communication	55%	High	
3			System Integration	40%	Catastrophic	Hermano Perrelli
4			Product Integration	40%	Catastrophic	
5			Testing	40%	Significant	
6			Requirements Completeness	40%	Moderate	Hermano Perrelli
7			Testability	25%	Low	
8			Validity			

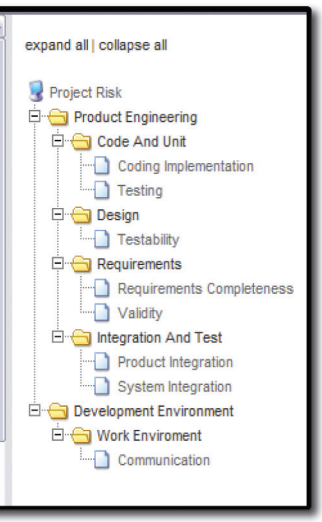


Figura 5. Matriz e Árvore de Riscos

diferentes, apoiadas por três componentes dentro do sistema, como representado na **Figura 3**.

Os três componentes centrais são:

Questionário SEI: perguntas relacionadas aos riscos presentes na taxonomia do SEI, onde as questões são divididas de acordo com sub-ontologias.

Relacionamentos Risco – Riscos: São ligações, definidas na *m*PRIME *Ontology*, que relacionam um risco a um conjunto de outros riscos. Ou seja, caso um projeto tenha um risco X, haverá a possibilidade do mesmo também ter um risco Y, sempre lembrando que um risco pode gerar ou não outro risco.

Relacionamentos "palavra-chave" – Riscos: São ligações, também definidas na *m*PRIME *Ontology*, que relacionam certas palavras-chaves, que podem estar contidas nas tarefas do projeto, com um conjunto de riscos. Ou seja, caso esse projeto tenha um risco X associado a uma palavra-chave, haverá a possibilidade do mesmo também ter um risco Y, sempre lembrando que uma palavra pode gerar ou não outro risco.

Com o uso desses três componentes foi possível gerar seis formas diferentes de identificação de riscos no projeto:

• **Sugestão de riscos pelo uso do questionário SEI:** o usuário pode responder o questionário fornecido (completa e/ou parcialmente) e, a partir dessas respostas, a *m*PRIME *Tool* irá sugerir os riscos relacionados.

• **Sugestão de riscos pelo questionário SEI, com recursão:** com esta opção, ao se responder o questionário, os riscos gerados irão passar pelo componente de Relacionamentos Risco – Riscos, podendo gerar uma quantidade maior de riscos.

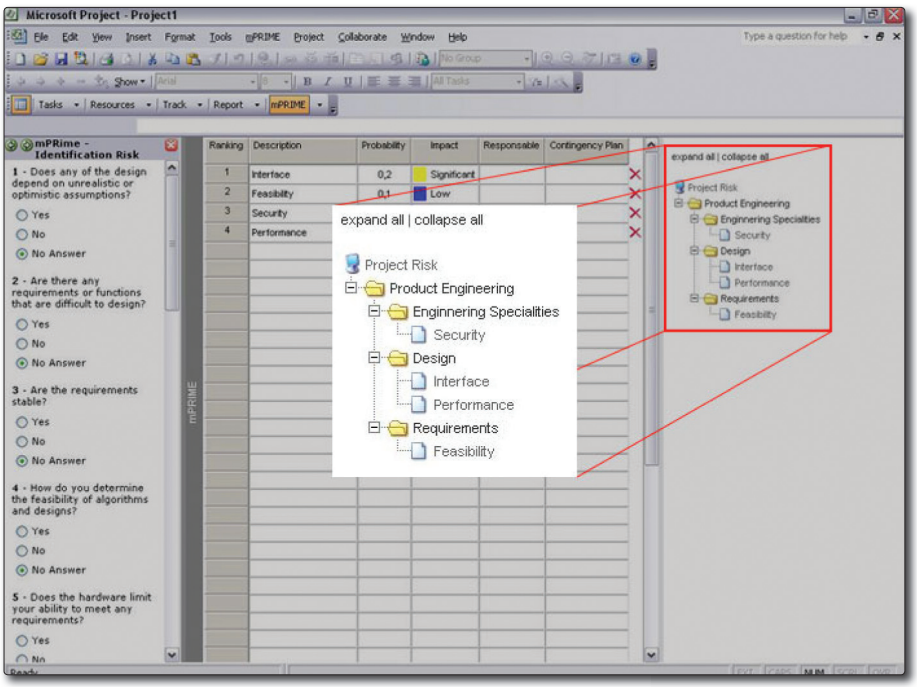


Figura 6. Visão em detalhe da Árvore de Riscos

• **Sugestão de riscos pelas tarefas do projeto:** o usuário pode selecionar esta opção, e o sistema fará uma varredura em todas as palavras presentes na lista de tarefas do projeto, procurando relacioná-las através do componente de Relacionamento "palavra-chave" – Risco, e sugerir os riscos encontrados.

• **Sugestão de riscos pelas tarefas do projeto, com recursão**: com esta opção selecionada, ao solicitar a sugestão de riscos pelas tarefas do projeto, os riscos gerados passarão pelo componente de Relacionamentos Risco – Riscos, podendo gerar uma quantidade maior de riscos.

• **Sugestão de riscos pela lista de riscos do projeto:** quando o usuário seleciona esta opção, o sistema buscará a lista atual de riscos do projeto, e as passará pelo componente de Relacionamentos Risco – Riscos, gerando assim uma nova lista de riscos sugeridos.

• **Sugestão de riscos pela lista de riscos do projeto, com recursão**: após gerar riscos pela lista de riscos, o sistema irá repassar a lista resultante novamente pelo componente de Relacionamentos Risco – Riscos, podendo gerar um número maior de riscos.

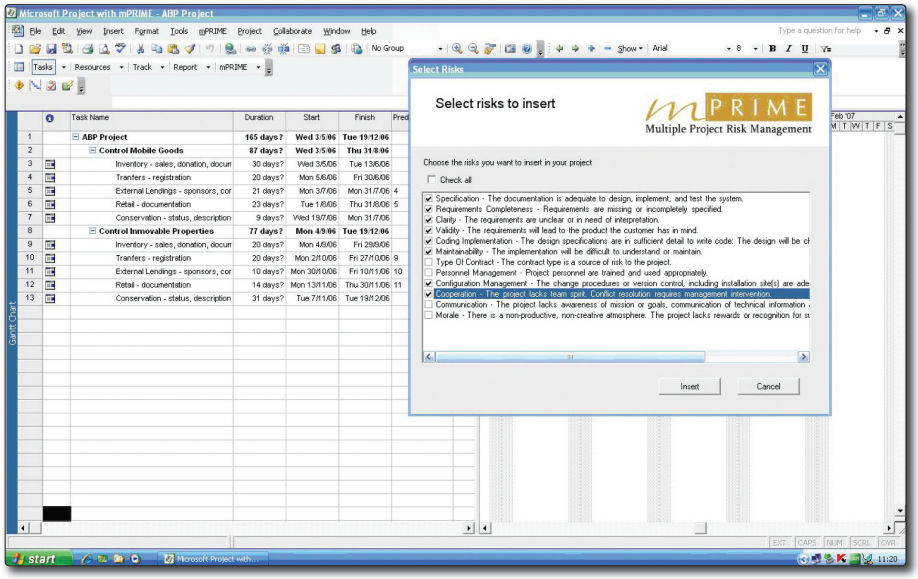


Figura 7. Identificação de Riscos com a *m*PRIME *Tool*

É importante lembrar que esta funcionalidade sugere os riscos ao usuário em forma de lista, ou seja, o usuário poderá selecionar aqueles que ele achar mais convenientes e adequados ao contexto do projeto em análise, e descartar os demais. A **Figura 2** apresenta, do lado direito, a árvore de riscos gerada pela seleção dos respectivos eventos de riscos para o projeto em análise.

3. *Analisar e Priorizar Riscos*

Dando suporte à atividade Análise de Riscos, a *m*PRIME *Tool* permite a análise qualitativa e a priorização dos riscos de acordo com a probabilidade e o impacto de ocorrência do evento, gerando o grau de exposição ao risco do projeto. Essas informações são configuráveis, pois as variáveis devem ser calibradas ao longo do gerenciamento de projetos do ambiente organizacional.

Para cada risco inserido ou sugerido pela *m*PRIME *Tool*, o usuário poderá fazer uma análise inicial e atualizar esta análise de acordo com as mudanças ocorridas no projeto. A ferramenta traz uma série de variáveis que devem ser preenchidas pelo gerente, como pode ser visto na **Figura 4**. Algumas dessas variáveis são: probabilidade de ocorrência do risco, nível de tolerância, impacto e tarefas relacionadas ao risco.

A categorização dos riscos é segmentada em três partes: (i) **Classe** – relaciona a categoria do risco como, por exemplo, Engenharia do Produto; (ii) **Elemento** – relaciona as partes integrantes da Classe como, por exemplo, Requisitos; (iii) **Tipo** (origem) – indica a possível origem do risco dentro do elemento como, por exemplo, Estabilidade. Logo, a instabilidade de requisitos em um projeto de desenvolvimento representa, dentro desta categorização – Engenharia de Produto/Requisitos/Estabilidade, um risco de requisitos instáveis. A partir destas informações sobre possíveis origens de riscos é construída a árvore de riscos (**Figura 2** – lado direito). Todos os relacionamentos utilizados foram definidos na ontologia *m*PRIME *Ontology* (*p*ara conhecer um pouco mais sobre esta funcionalidade acesse o *m*PRIME *Risk Inferrer*, disponível em http://www.dsc.upe.br/~tspc).

Em seguida, informações sobre a probabilidade e impacto do risco para o projeto devem ser avaliadas. A exposição ao risco (grau do risco para o projeto) é gerada a partir das informações da probabilidade e impacto. A tolerância está relacionada aos meios de tratamento do risco. Quanto menor a tolerância, maior é o potencial do risco para a organização. Ao final da análise, com as informações relativas ao grau de exposição do projeto ao risco, o registro é realizado através da atualização dos relacionamentos na árvore de riscos e da geração da matriz, conforme visualizado na **Figura 5**.

A matriz do risco é o registro das informações importantes para representar o risco, já a árvore de riscos representa o relacionamento horizontal entre as várias classes de riscos, seus elementos e tipos.

Cada risco pode ter associada uma cor para representar sua potencialidade. Como exemplo, temos na **Figura 5** as cores vermelho e azul representando graus extremos de severidade, muito alto e muito baixo, respectivamente.

4. *Resolver Riscos*

Esta atividade também é conhecida pelo planejamento dos tratamentos para contingência (eliminação e minimização) dos riscos do projeto. O gerente terá a possibilidade de, para cada risco, criar um plano de contingência, cada um com uma lista de ações a serem seguidas caso a probabilidade daquele risco esteja alta, e um responsável por aquele plano – caso ele precise ser posto em prática, conforme **Figura 4**. O Plano de contingência é um arquivo texto anexado, uma vez que as organizações possuem padrões de informações particularizadas para seus ambientes.

Desenvolvendo um plano de contingência antecipadamente pode-se reduzir enormemente o custo de uma ação quando o risco se concretizar.

5. *Monitorar Riscos*

As atividades de monitoramento incluem o controle das informações levantadas sobre os riscos do projeto, como também a comunicação destas para os membros da equipe do projeto.

A ferramenta disponibiliza através da árvore de riscos uma visão gráfica da situação de cada risco do projeto, como pode ser visto na **Figura 6**. Desta forma, conjunto de riscos são comunicados para as partes interessadas no projeto.

A definição desta lista de riscos e atividades associadas possibilita o registro de critérios subjetivos baseados na experiência da gerência e equipe de projeto, aumentando a memória organizacional de projetos e, conseqüente, conhecimento sobre riscos. O gerente de projeto poderá visualizar e atualizar a matriz de riscos constantemente.

O plano de contingência, elaborado na atividade de resolver riscos, é usado como forma de controle dos riscos. O responsável pelo plano de contingência tem a tarefa de executar as ações definidas no

plano. Riscos que atingem alta probabilidade de ocorrência são destacados através de alertas que sugerem ao gerente a consulta de seus planos de contingência.

Através de relatórios com a matriz de riscos, a ferramenta facilita a atividade de Comunicação dos Riscos dentro da organização. É possível a geração de três tipos de relatórios sobre os riscos: *Risk Ranking*, *Risk Tree* e *Risk Planning*. Estes relatórios apresentam de formas distintas as informações dos riscos avaliados.

A *m*PRIME *Tool* também permite que o usuário defina novos tipos de riscos, além dos definidos na *m*PRIME *Ontology*. Como cada empresa possui uma política interna e um processo de desenvolvimento, muitas vezes exclusivo, a *m*PRIME *Tool* possibilita que o usuário defina novas classes e atributos de riscos de forma a adequar a lista inicial de riscos disponibilizada a estas particularidades.

Situação Atual

A *m*PRIME *Tool* recentemente teve mais uma forma de identificação de riscos incorporada ao seu portfólio de funcionalidades. Nessa nova funcionalidade foi enfatizada a importância do registro e armazenamento das ocorrências de riscos para futuras situações semelhantes.

A experiência do gerente com projetos passados é muito importante para evitar que erros se repitam e tomar decisões corretas frente a um risco recorrente. Assim, é importante conhecer bem projetos passados, seus riscos e ações, para que, ao deparar-se com um cenário similar, os mesmos possam ser considerados e mitigados ou evitados de forma mais eficaz.

O método *CBR Risk* (www.dsc.upe.br/~pma/cbrrisk) possui como premissa fundamental "projetos de software semelhantes têm riscos semelhantes" [Trigo et al 2007]. Dado um novo projeto, o *CBR Risk* tenta identificar projetos anteriores similares numa base de dados. Uma vez identificados, os riscos associados a estes projetos podem ser também associados ao projeto atual. A busca por projetos semelhantes é feita através da técnica Raciocínio Baseado em Casos [Wangenheim e Wangenheim 2003].

Estudos de Caso

Como forma de avaliar as funcionalidades da *m*PRIME *Tool*, em especial as relacionadas a técnicas e métodos de identificação de riscos, foram realizados estudos de caso. Os estudos foram divididos em duas categorias: acadêmicos e indústria.

Para a realização destes estudos de caso acadêmicos, foi definido conteúdo programático aliando teoria e prática para gerenciamento de riscos. O foco das atividades práticas foi definido com base nas funcionalidades disponibilizadas pela *m*PRIME *Tool*.

A *m*PRIME *Tool* é uma ferramenta em evolução sendo utilizada em projetos acadêmicos e como apoio ao aprendizado da disciplina de Gerenciamento e Planejamento de Projetos, especialmente para a fundamentação dos conceitos de riscos, no Centro de Informática da Universidade Federal de Pernambuco (CIn – UFPE) e no Departamento de Sistemas e Computação da Escola Politécnica de Pernambuco (DSC – POLI).

Com relação à utilização da *m*PRIME *Tool*, em casos práticos da indústria, a seguir será apresentado um conjunto de informações coletadas durante a identificação de riscos.

Identificando Riscos no Projeto ABP

A *m*PRIME *Tool* foi utilizada para apoiar a identificação dos riscos do projeto ABP, em análise em ambiente organizacional de desenvolvimento de produtos de software.

Inicialmente foram transportadas as listas dos riscos já identificados para o projeto ABP (através das listas de verificação). A identificação foi feita de forma manual e pela experiência apresentada pelo gerente de projeto e sua equipe (mais de 5 anos no domínio da aplicação). Os riscos foram digitalizados na *m*PRIME *Tool*.

Depois da inclusão dos riscos inicialmente percebidos para o projeto ABP, foram utilizadas as técnicas de questionário e *m*PRIME *Ontology*, disponibilizadas na *m*PRIME *Tool*, para avaliação dos riscos ora inseridos (disponibilizados) e identificação de outros, até então não percebidos.

A lista inicial de riscos era composta por 7 fatores de riscos (riscos em potencial para o projeto), após a utilização das técnicas disponibilizadas na *m*PRIME *Tool*, mencionadas anteriormente, a lista foi avaliada e passou a conter 15 fatores de risco, conforme **Figura 7**.

Considerações Finais

A execução eficiente do gerenciamento de riscos de projetos é muitas vezes dificultada pela falta de percepção dos gestores - dificuldade na avaliação e controle dos riscos. Para apoiar esse processo, foi desenvolvida uma ferramenta para gestão de riscos em ambientes de múltiplos projetos, a *m*PRIME *Tool*, tendo por base estudo de doutorado do Centro de Informática da UFPE.

Conforme mencionado, a *m*PRIME *Tool* está em constante evolução - sendo um projeto da academia onde novos estudos são realizados com a finalidade de compartilhar informações e garantir a qualidade do processo de desenvolvimento de software, subsidiando gestores na tomada de decisão. ●

Referências

[Costa e Gusmão 2008] Costa, T. S. P.; Gusmão, C. M. G. (2008) Definição de Ontologia para Identificação de Riscos de Projetos de Software. In: I Seminário de Pesquisa de Ontologia no Brasil. Universidade Federal Fluminense. Niterói – RJ.

[Gruber 1995] Gruber, T. R. (1995) Toward principles for the design of ontologies used for knowledge sharing. In: Formal Ontology in Conceptual Analysis and Knowledge Representation. Kluwer Academic Publishers.

[Gusmão 2007] Gusmão, C. M. G. (2007) Um Modelo de Processo de Gestão de Riscos para Ambientes de Múltiplos Projetos de Desenvolvimento de Software. Tese de Doutorado. Universidade Federal de Pernambuco – Recife/PE – Brasil.

[PMI 2004] PMI - Project Management Institute. (2004) A Guide to the Project Management Body of Knowledge – ANSI/PMI 99-01-2004. Project Management Institute. Four Campus Boulevard. Newtown Square. USA.

[SEI 2001] SEI - Software Engineering Institute. (2001) CMMI - Capability Maturity Model Integration version 1.1 Pittsburgh, PA. Software Engineering Institute, Carnegie Mellon University. USA.

[Trigo et al 2007] Trigo, T. R.; Lins, A. V.; Gusmão, C. M. G. (2007).
CBR Risk: Método para Identificação de Riscos em Projetos de Software In: Conferência Ibero-Americana InterTIC 2007, 2007, Porto – Portugal. International Association for The Scientific Knowledge – IASK. 355p.

[Wangenheim e Wangenheim 2003] Wangenheim, C. G e Wangenheim, A. Raciocínio Baseado em Casos. Ed. Manole Ltda, São Paulo, Brasil. 2003

Introdução à Gestão de Conhecimento

De que se trata o artigo:

Nos últimos anos, a gestão de conhecimento surgiu como um dos principais focos de preocupação em grandes organizações. Mais do que a tecnologia, o conhecimento é chave para companhias que pretendem agregar valor a seus produtos e serviços. Neste contexto, este artigo apresentará algumas definições introdutórias à área de gestão de conhecimento.

Para que serve:

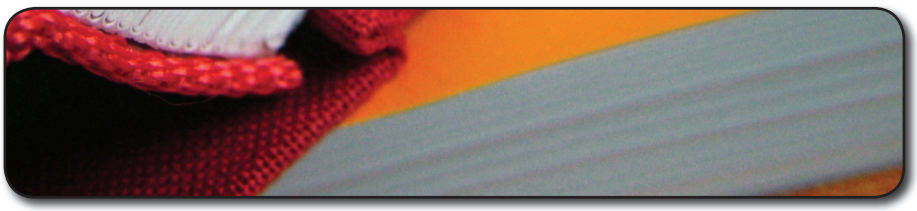
A gestão de conhecimento apóia o compartilhamento do conhecimento nas organizações. Esta é uma realidade que também pode estar presente em empresas desenvolvedoras de software.

Em que situação o tema é útil:

Gestão de conhecimento é um assunto amplamente estudado atualmente e utilizado no apoio à disseminação do conhecimento nas organizações.

Nos últimos anos, a gestão de conhecimento surgiu como um dos principais focos de preocupação em grandes organizações. Mais do que a tecnologia, o conhecimento é chave para companhias que pretendem agregar valor a seus produtos e serviços (SVEIBY, 2000). Companhias de sucesso são hoje caracterizadas por possuírem a capacidade de gerir seu capital intelectual, consistentemente produzindo conhecimento, rapidamente disseminando-o através da organização, e transformando-o em novos produtos e serviços. Existem muitas histórias de sucesso reportadas na literatura, todavia devido ao valor intrínseco associado aos programas de sucesso em gestão de

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 Engenharia de Software Magazine.



conhecimento, pouco material técnico está disponível sobre o assunto. Neste contexto, este artigo apresentará algumas definições introdutórias à área de gestão de conhecimento.

Dados, Informação, Conhecimento

A sociedade em que vivemos vem passando por várias transformações tecnológicas. Uma delas é a enorme facilidade de acúmulo de dados em repositórios automatizados. Dados são coletados, por exemplo, ao fazermos compras em um supermercado, ao irmos ao médico, ou ao votarmos em uma pesquisa interativa pelo telefone. Acontece que estes dados não têm nenhuma utilidade se não puderem ser efetivamente transformados em conhecimento utilizável por pessoas e instituições. Nesta seção, serão abordados os conceitos de dados, informação e conhecimento e, a importância destes na transformação do conhecimento.

Dado pode ser entendido como um conjunto de determinados fatos ou um registro de uma transação. Algum tempo atrás, tinha-se a ilusão de que quanto mais dados fossem coletados, mais "sábia" seria uma instituição. Porém, a experiência tem-nos mostrado que o trabalho para contextualizar estes dados, ou seja, atribuir um significado a ele, é tão ou mais importante que coletá-los.

Como estes dados, a priori, não nos revelam muita coisa, há a necessidade de transformá-los em **informação**. Para que isto ocorra, estes dados têm que passar por um processo de:

- **Contextualização:** identificar a finalidade dos dados;
- **Condensação:** resumir os dados em uma forma mais concisa;
- **Categorização:** identificar os componentes chave para a análise dos dados;
- **Cálculo:** analisar os dados matematicamente ou estatisticamente;
- **Correção:** remover os erros dos dados.

Depois destas etapas, os dados possuem uma utilidade bem definida. Com a informação pronta para ser utilizada, outros dois problemas surgem: (1) como organizar e distribuir esta informação de tal forma que ela gere conhecimento; e, (2) como gerenciar este conhecimento de tal forma que este seja sempre incrementado e compartilhado.

Conhecimento pode ser definido como informação útil e não trivial sobre um certo domínio de aplicação. Este conhecimento pode ser codificado de forma explícita em documentos ou sistemas de informação. Muitas vezes, todavia, ele existe de forma tácita, residindo apenas na cabeça de pessoas. Além das funcionalidades de um sistema típico de processamento de informações, sistemas computacionais de auxílio à gestão de conhecimento também têm por objetivo: (1) auxiliar as pessoas a externalizar seu conhecimento de forma explícita em documentos ou sistemas; (2) auxiliar as pessoas a internalizar o conhecimento explícito produzido por outras pessoas; e (3) criar ambientes virtuais (ex. chats e grupos de discussão) onde pessoas possam se socializar e compartilhar conhecimento sobre um determinado domínio de aplicação.

Segundo DAVENPORT (1998), também existe um processo de transformação da informação em conhecimento. Este processo consiste de quatro etapas executadas implicitamente e estão descritas a seguir:

- **Comparação:** ato de comparar as informações referentes a um acontecimento com situações diferentes;
- **Conseqüências:** que implicações a informação pode trazer na tomada de decisões;
- **Conexões:** relacionamento com outros conhecimentos;
- **Diálogo:** pensamento de outras pessoas sobre esta informação;

Com estas etapas concluídas, podemos dizer que existe um processo através do qual o dado pode gerar conhecimento. Este está demonstrado na **Figura 1**.

Nesta seção, verificamos as diferenças existentes entre dados, informação e conhecimento e, também, a importância de cada um para as organizações atuais levando em conta o processo de

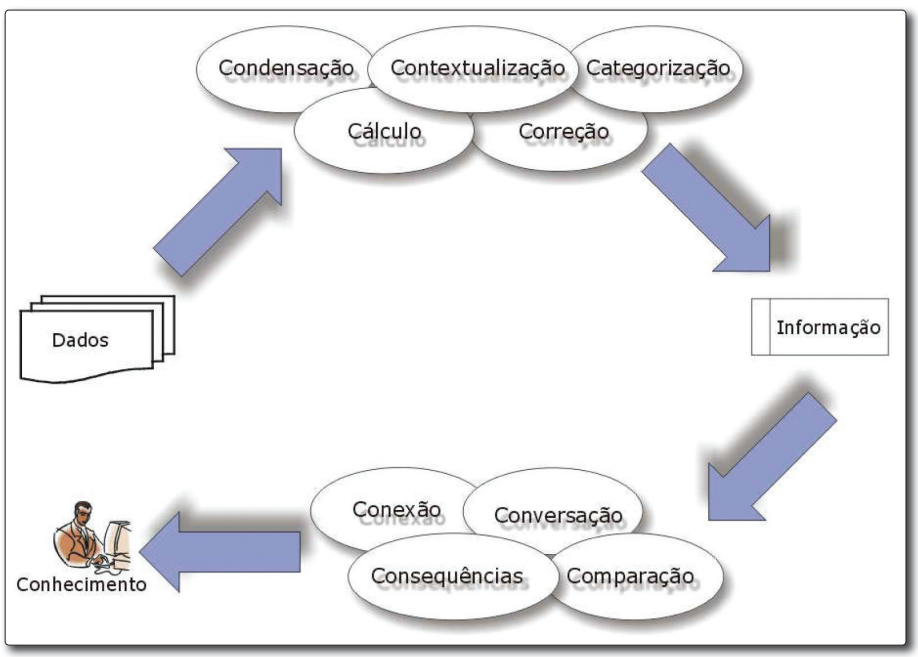


Figura 1. Dados, Informação e Conhecimento. Adaptado do modelo proposto por DAVENPORT (1998).

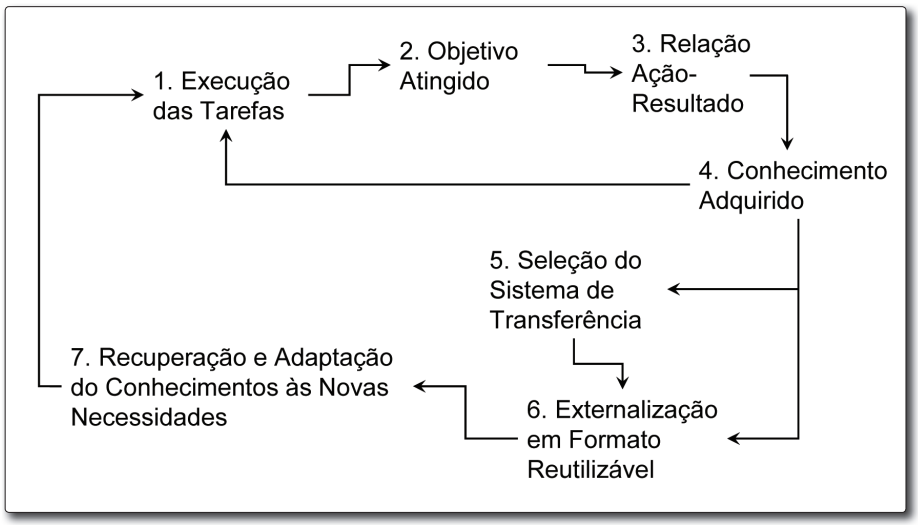


Figura 2. Etapas da Transferência do Conhecimento. Adaptado do modelo proposto por DIXON (2000: 23).

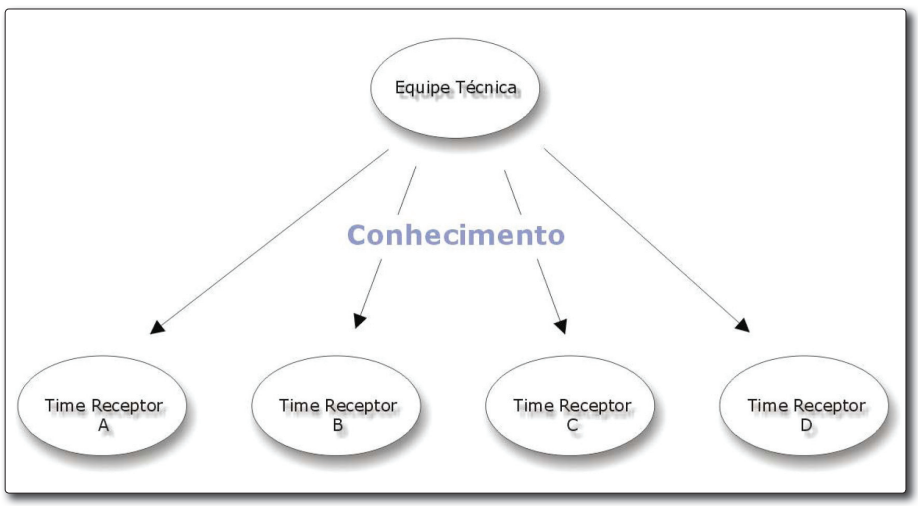


Figura 3. Modelo Centralizado (DIXON: 150)

criação do conhecimento. Tendo entendido o processo de transformação dos dados em informação e desta em conhecimento, será discutido na próxima seção diversos processos de transferência do conhecimento.

Transferência de Conhecimento

O conhecimento como bem não tangível, é altamente mutável e possui características peculiares em seus diversos tipos de ocorrência. Segundo DIXON (2000), existem formas diferentes de transferência do conhecimento para diferentes tipos de conhecimento a serem socializados.

Segundo DIXON (2000), as cinco maneiras de transferência do conhecimento são: (1) Transferência em Série; (2) Transferência Próxima; (3) Transferência Distante; (4) Transferência Estratégica e; (5) Transferência Técnica. Cada um desses tipos possui suas características próprias que tornam o processo de socializar o conhecimento complexo. Para distinguir essas categorias, existem três pontos a serem discutidos:

- O grau de similaridade das tarefas e contexto do conhecimento para o receptor;
- A freqüência na qual a tarefa ocorre;
- O tipo de conhecimento a ser transferido.

Estes três pontos influenciam diretamente o processo de transferência do

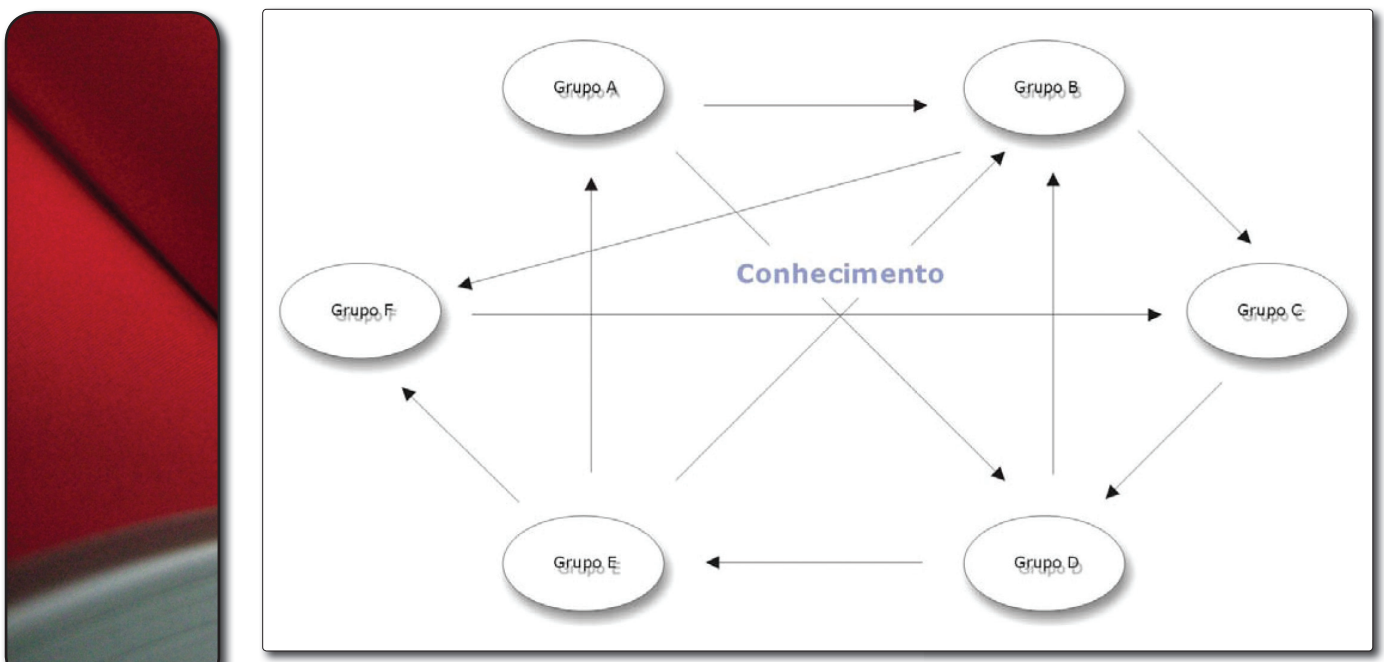


Figura 4. Modelo Distribuído (DIXON: 151)

conhecimento. De acordo com DIXON (2000), esta atividade possui algumas etapas que se repetem continuamente. (1) Uma equipe realiza uma tarefa (2) atingindo um objetivo. (3) Outra equipe analisa o relacionamento entre as ações tomadas e os objetivos atingidos. Desta forma, (4) o conhecimento é adquirido pela equipe. Depois que (5) o sistema para transferência do conhecimento é selecionado, o mesmo (6) é externalizado de forma útil para outras equipes. Estas (7) recebem o conhecimento e adaptam às suas necessidades. A partir daí, o ciclo recomeça. Essas etapas podem ser visualizadas na **Figura 2**.

A evolução destas etapas envolvidas no processo de compartilhamento do conhecimento foi bastante influenciada pelo novo entendimento de como a experiência estava distribuída nas organizações. Há um tempo atrás, as empresas definiam as pessoas que seriam as possíveis detentoras da informação. Com isso, o processo decisório e de socialização do conhecimento estava amarrado a um selecionado grupo de peritos em determinado assunto. Esta abordagem, conhecida como modelo centralizado (DIXON, 2000), mesmo que ultrapassada ainda está presente em diversas instituições. A **Figura 3** demonstra o fluxo do conhecimento neste domínio.

Por outro lado, análises recentes comprovaram que uma abordagem distribuída (DIXON, 2000) seria mais adequada ao contexto socioeconômico atual. Neste modelo, o conhecimento está distribuído por toda a organização não havendo um grupo de peritos. Com isso, o fluxo de compartilhamento do conhecimento aumenta consideravelmente e contribui com uma grande vantagem competitiva. Este modelo pode ser visualizado na **Figura 4**.

Com este último modelo em mente, analisaremos agora cinco tipos de transferência do conhecimento. Cada um destes possui sua particularidade, mas a presença de um não impede a utilização de um outro qualquer.

Transferência em Série

Muitas vezes, tarefas são concluídas repetidamente sem que haja um ganho de eficiência durante as inúmeras vezes em que elas se repetem. Para que o processo envolvido na execução destas tarefas se desenvolva, é necessária a análise das ações envolvidas e resultados obtidos para que estes possam auxiliar o desenvolvimento de melhorias. Entretanto, para que isto ocorra, é necessário que todos os integrantes da equipe em questão estejam dispostos a contribuir com sua experiência. Desta forma, podemos definir a transferência em série como um processo que move o conhecimento adquirido individualmente, de forma que este conhecimento possa ser integrado e aprovado pela equipe como um todo.

Sistemas que auxiliam a transferência de conhecimento em série possuem as seguintes características:

- A equipe à qual o conhecimento se destina deve trabalhar com tarefas similares porém, não há a necessidade de que o contexto seja o mesmo;
- A equipe destino deve possuir capacidade de absorver o conhecimento produzido por outro grupo de pessoas;
- O processo deve ser executado freqüentemente;
- O tipo de conhecimento transferido pode ser tanto tácito como explícito.

Neste tipo de sistema, existem alguns procedimentos que devem ser tomados para o sucesso da transferência. O principal deles é as reuniões regulares e rápidas com participação das pessoas envolvidas na criação do conhecimento. A razão pela qual os encontros devem ser curtos é a indisposição que, geralmente, os participantes têm em dispor seu tempo para reuniões.

Transferência Próxima

Bastante semelhante ao tipo de transferência analisado anteriormente onde constatamos a necessidade de haver tarefas similares a serem efetuadas pelas equipes geradoras e receptoras do conhecimento, a transferência próxima é fundamentada no fato de que novas formas mais eficientes de executar determinadas rotinas são descobertas e devem ser compartilhadas.

Apesar do nome insinuar a necessidade de que fonte e destino do conhecimento devem estar localizados próximos um ao outro, seu verdadeiro significado não é este. Neste contexto, a palavra próximo significa semelhança entre as atividades exercidas.

Sistemas que auxiliam a transferência próxima de conhecimento possuem as seguintes características:

- A equipe à qual o conhecimento se destina deve trabalhar com tarefas bastante parecidas;

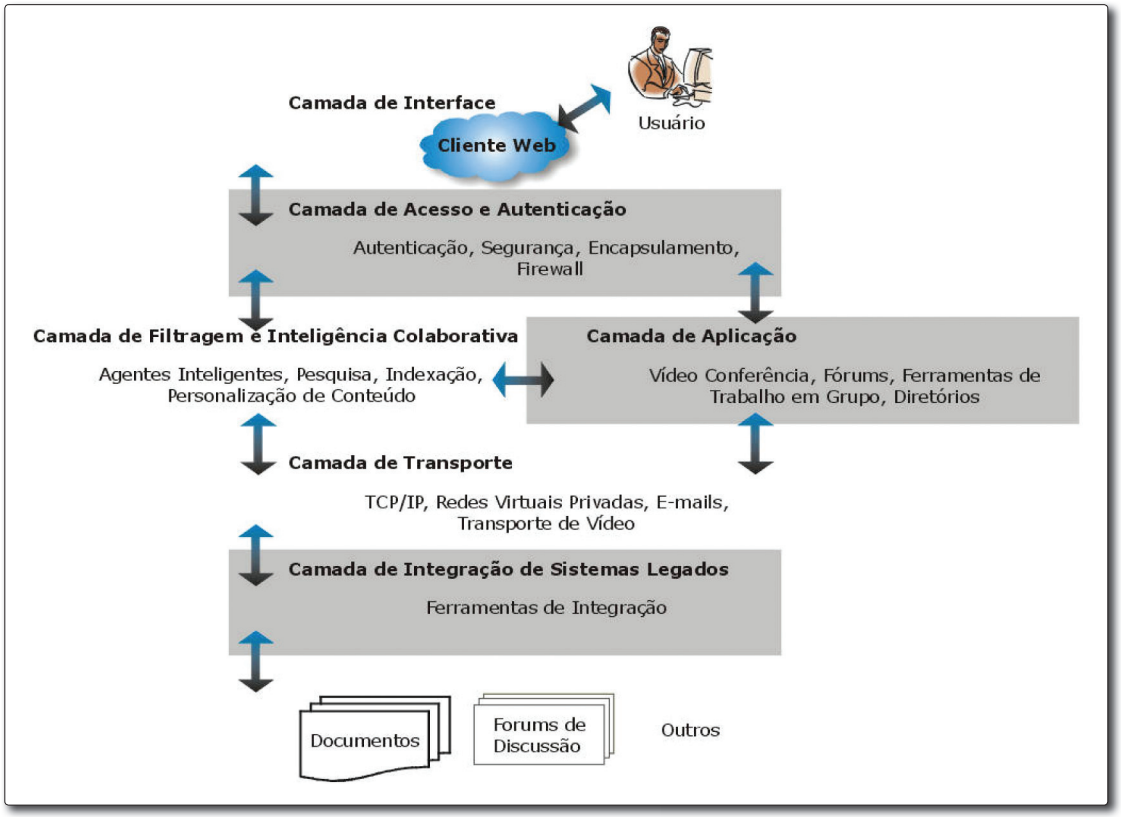


Figura 5. Camadas de um Sistema de Auxílio à Gestão de Conhecimento.

• A equipe destino deve possuir capacidade de absorver o conhecimento produzido por outro grupo de pessoas;

• O processo deve ser executado freqüentemente;

• O tipo de conhecimento transferido é principalmente o explícito sendo a participação do tácito limitada.

Existem algumas linhas mestras que podem auxiliar neste modelo de socialização. A principal delas é a utilização de meios eletrônicos para difusão do conhecimento. Por ser este em sua grande parte explícito, a utilização de servidores de informação ganha importância. Assim, os usuários do sistema podem especificar em qual tipo de conteúdo estão interessados assim como contribuir externalizando sua experiência. Os problemas que podem surgir deste modelo baseiam-se no fato de que pode haver quantidade muito grande de informação dificultando a recuperação destas e aumentando também quantidade de tempo necessária no processo de internalização do conhecimento.

Transferência Distante

Em diversas organizações, a pesquisa por novas metodologias cujo resultado ajude no desenvolvimento da instituição é constante. Daí, novos métodos em diferentes contextos são criados constantemente. Estes são um grande diferencial competitivo que cada empresa possui em relação às suas concorrentes. Porém, o processo de disseminar este conhecimento contextual pela instituição como um todo ou ao menos aos interessados é uma tarefa difícil. Um exemplo que se encaixa perfeitamente neste caso é a exploração petrolífera. Para esta, existem alguns procedimentos a serem seguidos, entretanto a forma como são utilizados depende muito das condições ambientais e também intelectuais dos técnicos envolvidos. Tendo analisado este exemplo, podemos conceituar transferência distante como aquela em que o tipo de tarefa executada pela equipe fonte e destino do conhecimento são as mesmas, porém com características peculiares que exigem adaptação do conhecimento adquirido a cada realidade.

Sistemas que auxiliam a transferência distante do conhecimento possuem as seguintes características:

• A equipe à qual o conhecimento se destina deve trabalhar com tarefas similares, porém o contexto varia bastante causando a necessidade de adaptação do conhecimento adquirido;

• A equipe destino deve possuir capacidade de absorver o conhecimento produzido por outro grupo de pessoas, mas essa capacidade pode variar bastante;

• O processo é executado freqüentemente;

• O tipo de conhecimento transferido é em sua maior parte implícito. Contudo, pode ser que haja uma pequena quantidade de forma explícita.

Transferência Estratégica

Em muitas organizações, a velocidade com que o mercado vem estabelecendo novos paradigmas e provocando reestruturações internas nas empresas faz com que haja a necessidade de ações pouco freqüentes que muitas vezes fogem ao domínio de conhecimento dos envolvidos. Para que não ocorram esforços desnecessários, caso algum outro grupo já tenha desempenhado atividade semelhante, esta experiência deve ser compartilhada. Neste contexto se enquadra a transferência estratégica. Este tipo de transferência se aplica quando:

A equipe à qual o conhecimento se destina deve trabalhar com tarefas similares, porém o contexto varia bastante causando a necessidade de adaptação do conhecimento adquirido;

• A equipe destino possui baixa capacidade de absorver o conhecimento produzido por outro grupo de pessoas uma vez que nunca havia participado anteriormente de um processo de reestruturação;

• O processo não é executado freqüentemente;

• O tipo de conhecimento transferido pode ser tanto tácito como explícito, sendo os dois de grande importância para o sucesso da socialização.

Como este processo de transferência está diretamente envolvido com tomadas de decisão da instituição, a definição do conhecimento a ser internalizado parte da gerência da empresa. Tendo esta definição, outro passo importante é a seleção de uma equipe de especialistas que possam coletar e interpretar o conhecimento colocando-o numa forma utilizável. Mas, o fato de ter que disponibilizar uma equipe de peritos em um determinado assunto encarece bastante esta forma de compartilhamento.

Transferência Técnica

De todos os tipos de transferência estudados até agora, este é o mais simples de ser implementado eletronicamente. As informações aqui compartilhadas são de natureza técnica e podem ser facilmente explicitadas. Este tipo de socialização ganha importância em organizações onde ocorra a utilização de equipamentos tecnológicos e outros processos técnicos. Isso não significa que o conhecimento esteja concentrado em manuais e/ou relatórios. Sabemos das necessidades de peritos em determinadas áreas cujos problemas incomuns possam acontecer.

Esse tipo de transferência se aplica quando:

• A equipe à qual o conhecimento se destina trabalha com tarefas diferentes das desenvolvidas pelo grupo de origem sendo, porém, o contexto igual;

• A equipe destino deve possuir alta capacidade de absorver o conhecimento produzido pelo outro grupo uma vez que a linguagem técnica utilizada é a mesma;

• O processo é executado freqüentemente;

• O tipo de conhecimento transferido é extremamente explícito.

Como citado anteriormente, nesta categoria de transferência a utilização de meios eletrônicos é bastante tendenciosa. Aplicações como fóruns de discussão, chats, repositórios estruturados de documentos e ferramentas de busca estão presentes.

Sistemas de Auxílio à Gestão de Conhecimento

Conceitualmente, um sistema computacional de auxílio à gestão de conhecimento pode ser dividido em sete camadas (TIWANA, 2000): interface, acesso e autenticação, inteligência colaborativa e filtragem, camada de aplicação, transporte, integração e por fim, os repositórios de dados. Esta estrutura em camadas está mostrada na **Figura 5**.

Como podemos perceber, cada camada possui seu próprio aparato tecnológico para realizar suas funções e que alguns destes meios já estão bastante disseminados entre empresas e instituições em geral. O que se necessita é uma efetiva integração destas tecnologias e a adição de alguns outros componentes para o desenvolvimento de um bom sistema de gestão de conhecimento.

Antes de discutirmos as sete camadas que compõem um sistema de auxílio à GC, é importante entender os dois importantes recursos que tornam possível seu desenvolvimento: comunicação e o armazenamento de dados.

A comunicação de dados tem grande importância no contexto da gestão de conhecimento uma vez que a mesma permite que o conhecimento implícito e explícito seja compartilhado de várias maneiras. As redes de computadores permitem a transferência de conhecimento e a colaboração entre integrantes das organizações que desejam gerir seu conhecimento. Utilização de vídeo conferência e e-mails são alguns exemplos que já estão muito difundidos na sociedade atual.

O armazenamento de dados é o outro mecanismo de grande importância para a gestão de conhecimento. É o armazenamento que permite a criação de uma memória organizacional onde o que é produzido é guardado em memória persistente e facilmente recuperável. Para isso, sistemas modernos de armazenamento possuem mecanismos para a qualificação da informação, armazenagem distribuída de dados, acesso remoto, controle de acesso, e segurança.

Como mencionamos anteriormente, nosso modelo conceitual definido para os sistemas de auxílio à gestão do conhecimento é dividido em sete camadas:

1. Interface: Esta camada é a única camada com a qual o usuário interage diretamente. É de fundamental importância uma boa concepção da mesma, caso contrário o sistema como um todo poderá fracassar. Além de estar comprometida com o usuário, a plataforma a qual esta camada esta associada deve também suprir os seguintes requisitos básicos:

protocolos eficientes, portabilidade, escalabilidade, segurança, integração com sistemas existentes e flexibilidade.

2. Acesso e autenticação: Esta camada tem como principal função autenticar usuários válidos, restringir e prover segurança para o acesso às outras camadas. Como as redes de comunicação para o compartilhamento do conhecimento não têm sido limitadas apenas às intranets, a importância dada à segurança está em crescimento.

3. Inteligência colaborativa e filtragem: A idéia básica desta camada é prover a estrutura funcional para que se consiga fazer pesquisas, resumos, interpretações e a análise de grandes volumes de dados habilitando os usuários do sistema de gestão de conhecimento a contextualizá-los de forma efetiva e eficiente. Para este fim, existe um gama de possibilidades de combinação de tecnologias: ferramentas de inteligência artificial, redes neurais, agentes inteligentes, pesquisa por conteúdo e pesquisa por atributo, dentre outros. Por ser um sistema que sofre constantes interações por parte dos usuários e por se tratar de um sistema com a infra-estrutura lógica (protocolos de comunicação) e física da Internet, é importante também que sua estrutura de funcionamento interno não tenha como base estruturas estáticas já bastante difundidas com o conceito de apontadores, mas sim, dinâmicas que automaticamente se adaptem a modificações na localização das informações. Neste caso, os ponteiros criados para outros documentos não se perdem tornado a navegação pela informação menos incômoda e menos frustrante.

4. Aplicação: Esta camada engloba as ferramentas de integração usuário/máquina que provêem boa parte da funcionalidade de um sistema de gestão de conhecimento.

5. Transporte: Tendo decidido a plataforma a ser utilizada, é preciso conhecer a maneira como os dados serão transportados pelas redes de comunicação. A forma como os dados são transportados depende de quem solicita o envio dos mesmos e das necessidades de serviço que cada tipo de dado tem na sua transferência.

6. Camada de integração de sistemas legados: Esta camada é necessária quando se quer integrar plataformas diferentes de um ambiente heterogêneo em uma determinada empresa. É comum ver companhias reestruturando sua infra-estrutura e "empacotando" seus sistemas legados com camadas de software que permitem a adoção de padrões mais modernos de comunicação e acesso a dados.

7. Armazenamento: Nesta camada os dados, informação e "conhecimento" são armazenados para posterior consulta, alteração, e deleção. A forma como a informação é armazenada difere a depender do tipo da mesma (imagem, som, animações, documentos). Existe neste nível a necessidade de se utilizar diversos tipos de repositórios que possam ser integrados de forma a prover uma estrutura coesa de acesso à informação.

Considerações Finais

Este artigo apresentou alguns conceitos básicos relacionados à gestão de conhecimento. Esta é uma importante área do conhecimento e tem sido estudada em diferentes contextos. Um exemplo é sua aplicação em ambientes de desenvolvimento de software e na captura de decisões arquiteturais durante a fase de projeto. É um assunto que está longe de ser esgotado e que certamente voltaremos a ter outras matérias mais específicas em edições futuras. ●

Referências

TIWANA, Amrit. The Knowledge Management Toolkit: Practical Techniques for Buildings a Knowledge Management System. Prentice Hall, 2000.

SVEIBY, Karl; STORK, John; HILL, Patricia, et al. Gestão do Conhecimento: Um Novo Caminho. HSM Management, setembro-outubro, 2000.

DAVENPORT, Thomas; PRUSAK, Laurence. Working Knowledge: How Organizations Manage what They Know. Harvard Business School Press, 1998.

DIXON, Nancy M. Common Knowledge: How Companies Thrive by Sharing what They Know. Harvard Business School Press, 2000.

Chegou o Sistema de Créditos DevMedia

Agora o **conteúdo completo** do nosso site está **ao seu alcance!**

A partir de agora todas as **2000 vídeo-aulas** do site DevMedia podem ser compradas individualmente.

Economia - Você não precisa mais assinar oito produtos diferentes para ter acesso ao conteúdo completo do site!

Todas as vídeos que você sempre quis ver a partir **R$ 0,75!**

Saiba mais sobre o Sistema de Créditos!

DevMedia GROUP

www.devmedia.com.br/creditos