Engenharia de Requisitos: Saiba como a Abordagem Baseada em Objetivos pode apoiar as atividades de requisitos

engenharia de software magazine

Edição 16 :: Ano 2

DevMedia group

ISSN 1983127-7

Engenharia de Requisitos
Alguns fundamentos e conceitos sobre cenários e casos de uso

Processo
Entenda como aprimorar a gestão da qualidade através do Seis Sigma

Processo
Introdução aos conceitos de interface homem máquina

Motivação + Engenharia de Software

Entenda alguns aspectos associados à motivação em integrantes de equipes de desenvolvimento de software

Validação, Verificação e Teste
Entenda a importância do Perfil Operacional nas atividades de teste de software

Validação, Verificação e Teste
Saiba como executar testes funcionais utilizando o Abbot Framework

Conhecimento faz diferença!

Mais de 60 mil downloads na primeira edição!

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

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

DevMedia GROUP

engenharia de software magazine

Ano 2 - 16ª Edição 2008 **Impresso no Brasil**

Corpo Editorial

Colaboradores
Rodrigo Oliveira Spínola
rodrigo@sqlmagazine.com.br

Marco Antônio Pereira Araújo
Eduardo Oliveira Spínola

Revisão e Supervisão
Thiago Vincenzo Ciancio - thiago.v.ciancio@devmedia.com.br

Coordenação Geral
Daniella Costa - daniella@devmedia.com.br

Diagramação
Janete Feitosa

Capa
Romulo Araujo - romulo@devmedia.com.br

Na Web
www.devmedia.com.br/esmag

Apoio

Parceiros:

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:

Cristiany Queiróz – Atendimento ao Leitor
www.devmedia.com.br/mancad
(21) 2220-5375

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

Publicidade

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

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

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

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

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

EDITORIAL

"Os impactos econômicos resultantes de práticas de gestão do comportamento têm despertado o interesse da indústria, fundamentadas em duas idéias principais: a primeira corresponde à necessidade de criação de um clima organizacional onde os colaboradores possam obter respeito e valorização; e a segunda é a premissa de que pessoas motivadas produzem mais. Por isso, constantemente surgem novas abordagens do tema aplicado à gestão empresarial. Especialmente as práticas motivacionais tornaram-se tão populares dentro das organizações, que atualmente são consideradas indispensáveis para a construção de um diferencial competitivo."

"Mas, de modo geral, o que é motivação? O que a motivação das pessoas pode causar numa organização? A motivação das pessoas é um aspecto relevante para o sucesso de projetos? O que a administração da motivação pode ter a ver com a engenharia de software? E o que a indústria e a academia pensam sobre o tema?"

Nesta edição, a Engenharia de Software Magazine destaca justamente esta temática: motivação de equipes de desenvolvimento de software. No artigo de capa desta edição são abordadas as principais teorias e conceitos relacionados à motivação e ao gerenciamento de equipes em projetos de software. Para isso, será apresentado um pouco do estudo da motivação humana aplicada à Engenharia de Software.

Além desta matéria, esta edição traz mais seis artigos:

- Cenários e Casos de uso: Fundamentos e Conceitos;
- Abordagens Baseadas em Objetivos: Visão Panorâmica;
- Gestão da Qualidade: Introdução ao Seis Sigma;
- Perfil Operacional: Informação Estratégica para Testes de Software;
- Teste funcional utilizando o Abbot Framework;
- Interação Humano-Computador.

Desejamos uma ótima leitura!

Equipe Editorial Engenharia de Software Magazine

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

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

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

Caro Leitor

Para esta edição, temos um conjunto de **6 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:

Tipo: Engenharia de Requisitos
Título: Definições Iniciais sobre Validação, Verificação e Testes – Parte 1
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Atividades de VV&T são fundamentais ao longo de todo o processo de desenvolvimento de software. Entretanto, ainda é comum o pensamento de que só devemos avaliar a qualidade do software ao final do processo através de atividades de testes. Veremos nesta série de vídeo aulas o porquê desta perspectiva não estar correta e entenderemos os principais conceitos associados às atividades de validação, verificação e testes. Nesta primeira aula, serão apresentadas algumas definições inicias que fundamentarão o entendimento do restante desta série de aulas.

Tipo: Engenharia de Requisitos
Título: Definições Iniciais sobre Validação, Verificação e Testes – Parte 2
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Atividades de VV&T são fundamentais ao longo de todo o processo de desenvolvimento de software. Entretanto, ainda é comum o pensamento de que só devemos avaliar a qualidade do software ao final do processo através de atividades de testes. Veremos nesta série de vídeo aulas o porquê desta perspectiva não estar correta e entenderemos os principais conceitos associados às atividades de validação, verificação e testes. Nesta segunda aula, o foco será a definição e os principais tipos de defeito.

Tipo: Engenharia de Requisitos
Título: Definições Iniciais sobre Validação, Verificação e Testes – Parte 3
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Atividades de VV&T são fundamentais ao longo de todo o processo de desenvolvimento de software. Entretanto, ainda é comum o pensamento de que só devemos avaliar a qualidade do software ao final do processo através de atividades de testes. Veremos nesta série de vídeo aulas o porquê desta perspectiva não estar correta e entenderemos os principais conceitos associados às atividades de validação, verificação e testes. Nesta terceira aula, iniciaremos os estudos sobre as atividades de revisões de software.

Tipo: Validação, Verificação e Teste
Título: Teste Funcional de Aplicações Desktop com a ferramenta Abbot
Autor: Marco Antônio Pereira Araújo
Mini-Resumo: Nesta vídeo aula será apresentada a ferramenta Abbot no apoio à elaboração de Testes Funcionais para aplicações Desktop em Java.

Tipo: Validação, Verificação e Teste
Título: Depuração de Código no Eclipse
Autor: Marco Antônio Pereira Araújo
Mini-Resumo: Nesta vídeo aula serão apresentados os principais conceitos de depuração de código fonte e demonstração de um estudo de caso utilizando o IDE Eclipse.

Tipo: Validação, Verificação e Teste
Título: Verificação de aderência a padrões de codificação com Checkstyle
Autor: Marco Antônio Pereira Araújo
Mini-Resumo: Nesta vídeo aula será apresentada a ferramenta Checkstyle para apoiar a construção de código fonte segundo padrões de codificação previamente estabelecidos, contribuindo para a qualidade do código fonte.

ÍNDICE

Você não está mais sozinho.

Cenários e Casos de uso: Fundamentos e Conceitos

De que se trata o artigo?

Este artigo destaca a importância de se considerar as perspectivas de Engenharia de Software (ES) e Interação Humano-Computador (IHC) para desenvolver sistemas interativos úteis e usáveis. Os conceitos de cenários e casos de uso são apresentados e analisados visando a integração das duas áreas.

Para que serve?

Conhecer como desenvolver software capaz de suportar o trabalho de pessoas, tornando este trabalho ainda mais fácil, rápido, simples e flexível. Para isso é necessário entender melhor as tarefas realizadas pelas pessoas e relacioná-las às tarefas desempenhadas no desenvolvimento de software.

Em que situação o tema é útil?

Desenvolver sistemas com melhor usabilidade é um requisito buscado e estudado por diversas empresas. Uma das formas de alcançar este ponto é considerando e combinando estudos de ES e IHC. É neste escopo que direcionamos este artigo.

Roberto Vedoato

Mestre em Ciência da Computação pela UFRGS. Especialista e Bacharel em Ciência da Computação pela UEL. Foi Professor das Universidades Estaduais UDESC, UEL e UENP, e atualmente é Analista de Sistemas do Ministério Público do Trabalho. Tem experiência de 15 anos na área de Computação, com ênfase em Engenharia de Software e Interação Humano Computador.

Freqüentemente, sistemas interativos são desenvolvidos com visões isoladas das áreas de Engenharia de Software (ES) e de Interação Humano-Computador (IHC). ES, tradicionalmente, prioriza aspectos essencialmente funcionais dos sistemas, como eficiência, manutenibilidade e portabilidade, conferindo ao desenvolvimento uma orientação funcional em detrimento da operacional. Já IHC foca principalmente a usabilidade dos sistemas, concentrando atenção aos aspectos de interação e não considerando adequadamente os aspectos enfatizados pela ES [CYB 98]. Essas duas abordagens refletem diferentes perspectivas, uma mais orientada ao sistema (tecnológica) e outra mais orientada ao usuário (humana), sobre a mesma atividade de desenvolvimento.

Recentes trabalhos de pesquisa convergem para idéia central de, para desenvolver sistemas interativos úteis e usáveis, ser necessário considerar as perspectivas de ES e IHC, desde as

primeiras etapas do desenvolvimento do sistema, exigindo a utilização conjunta e integrada de conceitos, técnicas e metodologias de desenvolvimento de ambas as áreas. Porém, essa interseção não é ainda nem natural nem objetiva: cada área considera aspectos diferentes e, muitas vezes, disjuntos do sistema, sem nenhuma correspondência explícita e sistematicamente estabelecida, tendo como conseqüência o fracionamento de requisitos [PIM 2000]. Compor o desenvolvimento de sistemas com grupos multidisciplinares, com diferentes pontos de vista sobre o processo de desenvolvimento, tem como potencial problema promover um entendimento mútuo da mesma tarefa de desenvolvimento e dos objetivos comuns [ABO 94].

Este artigo visa mostrar as possibilidades desta integração, através da combinação dos conceitos de Casos de Uso (*Use Cases*) e Cenários (*Scenarios*), importantes técnicas de modelagem, amplamente utilizadas, respectivamente nas áreas de ES e IHC, em diferentes contextos, com diferentes visões; mas apresentando similaridades valiosas para propiciarem o uso complementar de ambas as técnicas. Ambos são descrições narrativas. Cenários são utilizados em IHC para diversos fins, todavia particularmente úteis para inspecionarem atividades humanas ao usar um, presente ou futuro, artefato [CAR 95] e mediarem a comunicação entre grupos multidisciplinares, enquanto que casos de uso são usualmente utilizados, em ES, para modelarem requisitos funcionais e manipularem modelos de objetos [JAC 94a]. Baseando-se em estudo realizado previamente [VED 2001], tem-se a convicção de serem conceitos valiosos para combinar as diferentes perspectivas das duas áreas citadas, permitindo uma busca integrada da qualidade interna e externa dos sistemas.

Fazendo uso do conceito de níveis de abstração e partindo do enfoque de que cenários são instâncias concretas de casos de uso, podem-se combinar os propósitos e usos complementares de ambas as técnicas.

Acredita-se que a combinação dessas técnicas, aliada a uma boa compreensão das tarefas e do contexto organizacional em que elas são executadas, juntamente com a explicitação dos objetivos, previnam a possibilidade de que especificações se tornem desencontradas das reais necessidades dos usuários e devam ser a base para um processo de desenvolvimento de software interativo.

Contudo, abordagens apontam que casos de uso apresentam problemas conceituais [REG 95a, REG 95b] e a maneira pela qual são comumente usados na Unified Modeling Language (UML) [BOO 99], não considera adequadamente aspectos de usabilidade [CON 2000b]. Além disso, várias pesquisas têm focado, principalmente, as funções e os usos de cenários e casos de uso, no processo de desenvolvimento de sistemas, enquanto que o processo de construção destes, muitas vezes, é negligenciado.

Com o intuito de fundamentar esta possibilidade de integração, este artigo apresenta uma investigação dos conceitos e fundamentos de cenários e casos de uso, sob a ótica, respectivamente, de IHC e ES.

Cenários: Visão de IHC

Segundo definição do dicionário Aurélio [FER 99], o termo "cenário" significa: *"Lugar onde ocorre algum fato, ou onde decorre a ação, ou parte da ação, de uma peça, romance, filme, etc"*. Em IHC, o comum significado para o termo "cenário" é uma instância representativa de uma interação entre usuário e sistema [CAM 92]. Um cenário é uma descrição narrativa informal e concreta a respeito de um uso de um sistema por uma pessoa.

Aplicação

Na literatura de IHC, há um amplo consenso sobre os benefícios da utilização de cenários, no processo de desenvolvimento de software, e, também, que cenários podem ser usados para muitos propósitos diferentes. Em IHC, cenários são particularmente úteis para

- ilustrar como um usuário provavelmente efetua tarefas particulares com um, presente ou futuro, sistema;
- prover uma representação de projeto, formulada em ações e experiências das pessoas que, efetivamente, usarão a tecnologia;
- avaliar a usabilidade de sistemas;
- guiar o projeto de interfaces dos usuários (IUs);
- testar teorias de IHC;
- mediar a comunicação entre grupos multidisciplinares [CAM 92, CAR 2000].

Notação

Cenários geralmente são representados por narrativas textuais, em linguagem natural, que, freqüentemente, são aprimoradas por gráficos, diagramas e imagens [RYS 2000]. Podem também ser representados através de protótipos, *storyboards*, vídeos, maquetes, ou até mesmo por situações físicas planejadas para suportar as atividades de um usuário [CAR 2000]. Um exemplo de cenário pode ser observado abaixo:

- Maria deseja utilizar o caixa eletrônico do banco X para retirar RS 50,00 em dinheiro de sua conta corrente; Ela passa seu cartão no leitor do caixa eletrônico pronto para ser usado. O sistema valida o cartão e, em seguida, requisita a senha a Maria. Pelo teclado, Maria digita sua senha. O sistema valida a senha digitada e mostra na tela o menu de opções de transações. Maria escolhe a opção saque, tocando no botão com esta denominação que está na tela; em seguida escolhe R$ 50,00 no menu de possíveis quantidades de dinheiro. O caixa automático verifica a quantia escolhida, a existência de fundos, aprova a transação e libera a quantia correta e um recibo do saque. Maria pega o dinheiro e o recibo nos locais especificados, o sistema debita a quantia da conta corrente, exibe uma mensagem, pedindo para o cliente certificar-se de que está de posse do seu cartão magnético; e, por fim, mostra novamente o menu de opções de transações.

Estrutura

Uma coleção de cenários não possui uma estrutura definida, é representada através de um conjunto não estruturado.

Uso

Cenários são utilizados tanto de maneira formal quanto informal, variando seu uso de partes constituintes do processo de desenvolvimento; por exemplo, no desenvolvimento baseado em cenários e no desenvolvimento participativo de software; até usos mais espontâneos; por exemplo, para propósitos de comunicação [KLA 93].

Em geral, desenvolvedores parecem não considerar cenários como parte integrante do método de desenvolvimento [ABO 94]. Estudos empíricos demonstram que programadores avaliam projetos, simulando cenários mentalmente [BEN 93]. O uso informal de cenários pode ser observado quando desenvolvedores têm algum ponto de discordância, exploram opções de projeto, procuram esclarecer requisitos do sistema etc. [KLA 93].

Quando utilizados de maneira informal, cenários são construídos *ad hoc*, nenhum método específico é utilizado para gerá-los ou avaliá-los [ABO 94]; seus conteúdos provêem de muitas fontes diferentes, desde experiências dos desenvolvedores e conhecimento do domínio a reais observações dos usuários. Não existem compromissos com o que um cenário deve conter ou o quão representativo deve ser para uma futura situação de uso [KLA 93]. Os usos informais ou implícitos de cenários são predecessores dos usos mais formais ou explícitos.

Segundo Carroll [CAR 95], o desenvolvimento tecnológico tem sido dificultado pela incompleta compreensão da prática. Geralmente, computadores são vistos como artefatos da atividade humana por meio de especificações funcionais, o que gera uma visão idealizada de uso, não de como um usuário ativaria uma função e experimentaria seus efeitos. Computadores são mais do que funcionalidade. Eles inevitavelmente reestruturam atividades humanas, criando novas possibilidades, assim como novas dificuldades [CAR 2000]. Cenários são meios concretos para descreverem situações existentes e para projetarem futuras situações de uso, facilitando a comunicação efetiva entre usuários e desenvolvedores sobre os requisitos do sistema e opções de projeto.

Extrair dos usuários aquilo que desejam e de que necessitam não é uma tarefa fácil. A ciência cognitiva tem demonstrado que a inteligência humana está organizada em *chunks*, que reconhecem e respondem a situações específicas. A resolução de problemas, pelos humanos, tende a ser altamente situada; por exemplo, ao invés de realizarem planos detalhados antecipadamente, pessoas respondem aos detalhes de uma situação, conforme eles aparecem [BEN 93]. Representar o uso de um sistema através de um conjunto de cenários torna o uso explícito, isso pode ajudar a programadores e analistas a focarem atenção na compreensão das pessoas e suas tarefas, aspectos, muitas vezes, implícitos nos sistemas [CAR 2000]. O real uso de um sistema pode ser concretamente descrito por um conjunto de cenários [CAR 95].

Enquanto abordagens tradicionais lidam com a complexidade do processo de desenvolvimento, através de abstração, o desenvolvimento baseado em cenários utiliza concretização. Ao invés de desenvolver software, listando requisitos, funções e módulos de código, o desenvolvedor foca primeiramente as tarefas dos usuários que precisam ser suportadas pelo sistema e, então, através de cenários, realiza descrições dessas para guiar todo o processo de desenvolvimento.

Na abordagem baseada em cenários, estes são construídos, baseados no ciclo tarefa-artefato: as tarefas que as pessoas estão engajadas e aquelas que elas desejam realizar definem os requisitos da futura tecnologia, incluindo novos artefatos de IHC. Os novos artefatos criam novas possibilidades para as tarefas humanas, novas maneiras de executar coisas familiares e atividades completamente novas para fazer; mas, também, criam novas complexidades de aprendizagem e performance, novas interações entre tarefas e, naturalmente, novas dificuldades e novos erros para as pessoas. Essas novidades, eventualmente, tornam-se requisitos dos próximos desenvolvimentos tecnológicos, provocando novas transações [CAR 95]. A **Figura 1** ilustra o ciclo tarefa-artefato.

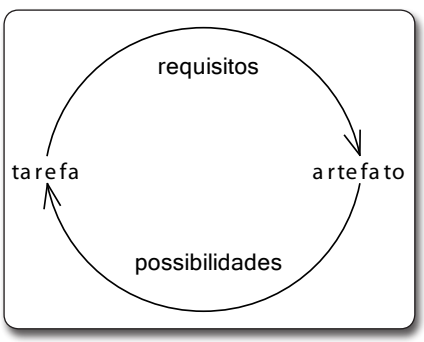


Figura 1. Ciclo tarefa-artefato

Para construir representações de cenários, não se pode contar somente com observações. Caso se deseje que cenários representem um papel pró-ativo, guiando o planejamento e desenvolvimento de um novo sistema, precisa-se ser capaz de criar representações de cenários, antes que qualquer versão do sistema tenha sido desenvolvida. Podem-se construir cenários que representam futuras situações de uso, através do uso coordenado de observações empíricas e de abstrações das teorias da atividade humana [CAR 95].

Portanto, o desenvolvimento baseado em cenários é uma técnica, orientada a tarefa, para fazer uma projeção do uso de um artefato, antes de construí-lo; nele, cenários podem ser usados ao longo de todo o ciclo de vida do software, desde a análise de requisitos e projeto do sistema até a documentação, o treinamento e a avaliação de protótipos.

Casos de Uso: Visão de ES

Casos de uso foram introduzidos por Jacobson como parte da metodologia de desenvolvimento de software OOSE (*Object-Oriented Software Engineering*) [JAC 92]. Sua definição original é *"Um caso de uso é uma maneira específica de usar um sistema usando alguma parte da funcionalidade. Constitui um curso completo de interação que acontece entre um ator e o sistema"*.

Por definição, o termo "caso de uso", introduzido nesta metodologia, não é sinônimo do termo "cenário". Um cenário

deve ser entendido como uma instância específica de um caso de uso [JAC 94a], ao passo que o caso de uso é uma coleção de cenários, representando múltiplos caminhos.

A idéia geral de um caso de uso é representar seqüências de interações entre um sistema, mesmo que ainda não implementado, e o mundo externo ao sistema; ou seja, descreve maneiras de usar um sistema.

Muitas metodologias de desenvolvimento de software possuem alguma noção de casos de uso. Em [REG 96] pode-se encontrar uma visão geral, na forma de uma tabela comparativa, sobre como casos de uso são aplicados às metodologias Orientadas a Objeto (OO): OOSE, *Object Modeling Technique* (OMT) e Booch. A **Tabela 1** é uma transcrição parcial desta visão geral.

As semânticas dos conceitos, relacionados a casos de uso nos diferentes métodos, não são correspondentes e existe uma significante inconsistência entre os métodos com relação a como os conceitos são interpretados [REG 96].

Posteriormente, os autores das três metodologias, Jacobson (OOSE), Booch (Booch) e Rumbaugh (OMT), uniram e estenderam suas abordagens através de uma linguagem de modelagem unificada a *Unified Modeling Language* (UML), que emergiu como um padrão entre desenvolvedores de software OO. Conseqüentemente, neste trabalho, dar-se-á enfoque à UML.

A UML é uma linguagem padrão para elaboração da estrutura de projetos de software, empregada para a visualização, especificação, construção e documentação de artefatos que façam uso de sistemas complexos de software [BOO 99]. Ela integra técnicas de modelagem de diversas metodologias de desenvolvimento. Muitos dos conceitos sobre casos de uso, originalmente associados à metodologia OOSE, foram integrados à especificação da UML.

UML é linguagem de modelagem, não é metodologia, portanto é somente parte do método de desenvolvimento de software. A UML não prescreve explicitamente um processo de utilização; porém, para obter maior proveito da UML é preciso que o processo tenha a característica de ser baseado em casos de uso. Isto significa que casos de uso são utilizados como principal artefato para estabelecer o comportamento desejado do sistema, para verificação e validação da arquitetura do sistema, para a realização de testes e para comunicação entre os envolvidos no desenvolvimento do sistema [BOO 99].

A definição formal de caso de uso, segundo a UML [BOO 99], é *"Caso de uso é uma descrição de um conjunto de seqüências de ações, inclusive variantes, que um sistema realiza para produzir um resultado de valor observável a um ator"*.

Um ator representa um agente externo ao sistema que de alguma forma participa de um caso de uso. É chamado de ator um papel que uma pessoa, um dispositivo de hardware, ou, até mesmo, outro sistema desempenha com o software [BOO 99]. Atores ajudam a delimitar o sistema em desenvolvimento, por representarem agentes externos que interagem com o mesmo.

Existe distinção entre os conceitos de ator e usuário. Usuário é um conceito não formal. Ator representa um papel específico que um usuário pode atuar, enquanto que o usuário é alguém que utiliza o sistema [JAC 94a].

Um caso de uso é uma descrição narrativa de um processo do domínio da aplicação. Ele representa um requisito funcional do sistema [BOO 99] e captura o comportamento pretendido do sistema; sem, no entanto, especificar como este comportamento é implementado, descreve o que o sistema faz e não como é feito. Casos de uso proporcionam uma visão externa do sistema; através deles, o sistema é visto como uma caixa-preta (*black box*).

Aplicação

Caso de uso é uma técnica de modelagem que pode ser aplicada a qualquer paradigma de desenvolvimento de software, estruturada, ou OO. A semântica de um modelo de casos de uso relaciona-se fortemente com a de um modelo de objetos [JAC 94b]; porém, um modelo de casos de uso não substitui um de objetos. O modelo de casos de uso é uma visão externa de um sistema; o modelo de objetos é uma visão interna do mesmo sistema [JAC 94a].

Nos últimos anos, casos de uso têm recebido muita atenção na área de ES, como meio para elicitar, documentar e validar requisitos, no entanto, isto não significa que estão limitados a ER [REG 96, RYS 2000]; eles podem ser usados ao longo de todo o ciclo de vida do desenvolvimento do software.

Casos de uso são utilizados para diversas finalidades, entre as quais:

- Estruturar complexos modelos de objetos, em visões manejáveis;
- Capturar, documentar e validar requisitos funcionais de sistemas;

Metodologia	OOSE	OMT	Booch
Conceitos	Casos de uso, ator, exceção, extensão (extends), inclusão (uses).	Caso de uso, ator, cenário, pré e pós-condição, exceção, adds.	Caso de uso, cenário, iniciador, pré e pós-condições.
Notação na fase de análise de requisitos	Linguagem natural para descrever casos de uso. Diagramas para descrever relações entre casos de uso.	Linguagem natural com algumas recomendações para estruturação.	Linguagem natural.
Função no processo de desenvolvimento	Guia todo processo, é usado para identificar objetos e para criar projetos robustos.	É usado para reforçar a análise de requisitos e para identificar objetos.	É usado para identificar objetos, para concepção e para planejamento de versão.
Metodologia para criação de casos de uso	Algumas heurísticas. Questões para responder a cada ator ajudam a identificar casos de uso.	Lista de ações passo a passo. Cenários são combinados ou generalizados em casos de uso.	Prescreve o planejamento de cenários como uma atividade e provê algumas poucas recomendações.

Tabela 1. Visão geral de casos de uso em métodos OO [REG 96]

• Gerenciar a complexidade de desenvolvimento, focando um aspecto distinto por vez;
• Compreender e estabelecer o comportamento desejado do sistema;
• Ganhar percepção sobre modelos alternativos de software;
• Fornecer uma descrição consistente e clara sobre as responsabilidades a serem cumpridas pelo sistema, funcionando como uma espécie de contrato;
• Facilitar a comunicação entre os envolvidos no desenvolvimento do sistema;
• Envolver usuários no desenvolvimento de software;
• Verificar e validar a arquitetura de sistemas;
• Realizar testes de sistemas;
• Servir como fonte para construção de manuais;
• Derivar modelos conceituais;
• Reengenharia de software.

Notação

Existe uma grande variedade de estilos para descrever o conteúdo narrativo de um caso de uso. A abordagem original de casos de uso, proposta por Jacobson e, posteriormente, adotada na UML, não define precisamente nenhum formato ou *template* específico para descrever o conteúdo de um caso de uso. Originalmente, casos de uso não possuem notação formal; eles são representados na forma de narrativa textual contínua.

Os problemas deste estilo de narrativa são numerosos. Não há nenhuma separação clara entre o lado do usuário e o do sistema. A narrativa mistura exigências internas e externas e salta irregularmente entre perspectivas internas e externas. Os elementos essenciais à natureza do problema são misturados com decisões de projeto, e a falta de estrutura força o leitor a seguir o texto inteiro apenas para obter uma idéia geral do caso de uso [CON 2000b].

Outro estilo comum para representar casos de uso é a seqüência numerada; nele, a narrativa é escrita como uma série de etapas numeradas. O trecho abaixo ilustra um exemplo extraído de [CON 2000b]:

1. O caso de uso inicia, quando o cliente introduz um cartão no caixa automático. O sistema lê e valida a informação do cartão;
2. O sistema aguarda pela senha. O cliente entra com a senha. O sistema valida a senha;
3. O sistema pergunta que operação o cliente deseja executar. O cliente seleciona "saque de dinheiro";
4. O sistema pede a quantia. O cliente entra com a quantia;
5. O sistema pede o tipo. O cliente seleciona o tipo de conta (poupança, conta corrente);
6. O sistema comunica-se com a rede do caixa automático para validar o número da conta, o cartão, a senha e a disponibilidade da quantia pedida;
7. O sistema pergunta ao cliente se ele deseja um recibo. Esta etapa é executada somente se houver papel para a impressão do recibo;
8. O sistema pede que o cliente retire o cartão. O cliente retira o cartão. (Esta é uma medida de segurança para assegurar que os clientes não deixem seus cartões na máquina.);
9. O sistema libera a quantia pedida;
10. O sistema imprime o recibo;
11. O caso de uso termina.

Uma vantagem deste estilo é evidente: a separação em etapas distintas facilita a leitura do caso de uso para a obtenção de uma visão geral. Todavia, sofre muito dos mesmos problemas do estilo narrativo contínuo. Apesar da segmentação em etapas discretas, as etapas individuais mesclam ações do sistema e do usuário.

Ambos os estilos de narrativa, textual contínua e seqüência numerada, apresentam abundância de palavras. Devido ao fato de não possuírem quase nenhuma estrutura, esses estilos de narrativa requerem, para clareza, que a perspectiva seja repetidamente declarada - o sistema faz isto, o usuário escolhe isso, o sistema termina algo mais -, o que contribui para suas loquacidades. Mesmo assim, o limite entre o que está dentro do sistema e o que está fora não está explicito e, somente, poderá ser discernido através de uma leitura cuidadosa da narrativa inteira [CON 2000b].

A solução simples para isto é separar completamente da interação o lado do usuário e o do sistema. Para casos de uso, esta separação foi originalmente sugerida por Wirfs-Brock, sendo criado o estilo de narrativa com partição [CON 2000b]. Neste estilo, a narrativa de casos de uso é dividida em duas colunas: ação do usuário e resposta do sistema. Assim, o limite das perspectivas internas e externas torna-se óbvio e explícito, sem repetição desnecessária. Por exemplo, a **Tabela 2** é uma narrativa com partição do caso de uso, sacar dinheiro, representado nos 11 passos representados acima, como narrativa com seqüência numerada.

Este estilo de narrativa é bem mais legível e apresenta uma menor verbosidade do que os outros estilos apresentados, anteriormente. Naturalmente, nada impede que se numerem as ações e as respostas, tornando-o ainda mais útil.

Casos de uso podem ainda ser representados por pseudo-códigos. Embora tais expressões pareçam oferecer precisão e possam ser confortáveis e familiares aos engenheiros de software, elas, raramente, são compreensíveis aos usuários comuns [CON 2000b].

Estrutura

Em adição aos de casos de uso, Jacobson também introduziu um modo de representá-los graficamente: o diagrama de casos de uso, o qual também é parte integrante da UML.

Um diagrama de caso de uso exibe uma coleção de casos de uso, atores e seus relacionamentos. Essa notação permite visualizar um caso de uso em separado de sua realização e no contexto com outros casos de uso [BOO 99].

Graficamente, um caso de uso é representado por uma elipse com linhas contínuas, incluindo um nome que o diferencia dos demais; um ator é representado por uma figura esquematizada, um "boneco de palitos"; e as associações entre atores e casos de uso são representadas por linhas. O escopo do sistema é explicitado por uma caixa que envolve os casos de uso. A **Figura 2** mostra um exemplo de diagrama de caso de uso.

Ação do usuário	Resposta do sistema
insere o cartão no caixa automático	
	lê o cartão
	solicita a senha
entra com a senha	
	valida a senha
	mostra o menu de opções
seleciona a opção	
	mostra o menu de conta
seleciona a conta	
	solicita a quantia
entra com a quantia	
	mostra a quantia
confirma a quantia	
	retorna o cartão
pega o cartão	
	libera o dinheiro se disponível

Tabela 2. Exemplo de caso de uso na notação de narrativa com partição [CON 2000b]

Diagramas de caso de uso fornecem um modo de descrever a visão externa do sistema e suas interações com o mundo exterior, representando uma visão de alto nível de funcionalidade intencional, mediante o recebimento de um tipo de requisição de usuário [FUR 98]. Esses diagramas definem a total funcionalidade do sistema e são importantes principalmente, para organização e modelagem de comportamento do sistema [JAC 94a, BOO 99].

Relacionamentos

Na UML, a conexão entre um ator e um caso de uso é denominada de associação. Ela indica que o ator e o caso de uso comunicam entre si, cada um com a possibilidade de enviar e receber mensagens.

Podem existir relacionamentos entre casos de uso, aplicados com a finalidade de fatorar comportamentos comuns e variantes, evitando redundâncias e aumentado o reuso. Os relacionamentos existentes na UML são generalização, inclusão e extensão [BOO 99].

Um relacionamento de generalização entre casos de uso indica que um caso de uso pode compartilhar o comportamento definido em um ou mais casos de uso. É um mecanismo usado para identificar comportamentos reutilizáveis.

O relacionamento de inclusão entre casos de uso, identificado pelo estereotipo "inclui" (do inglês "include"), significa que um caso de uso incorpora explicitamente o comportamento de outro caso de uso. O caso de uso, incluído, nunca permanece isolado, apenas é instanciado como parte de alguma

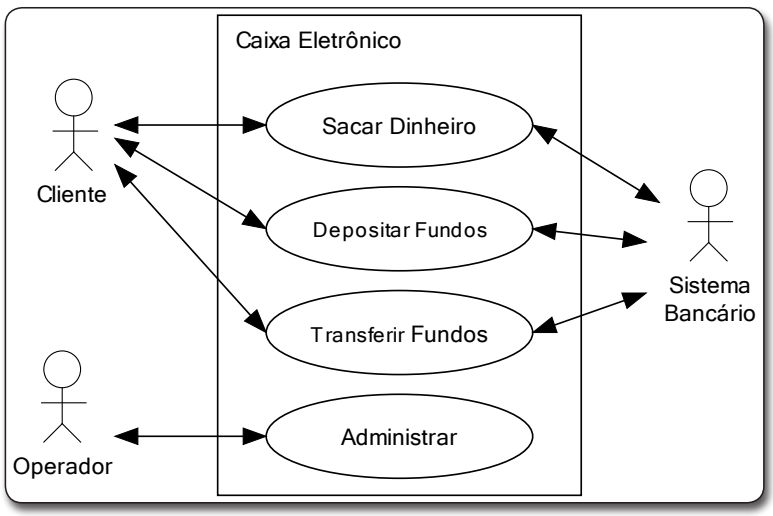


Figura 2. Exemplo de diagrama de casos de uso

base maior que o inclui. Este tipo de relacionamento é utilizado para evitar descrever o mesmo fluxo de eventos várias vezes, fatorando o comportamento comum em um caso de uso próprio. O relacionamento de inclusão é essencialmente um exemplo de delegação [BOO 99]. A **Figura 3** ilustra um exemplo de relacionamento de inclusão.

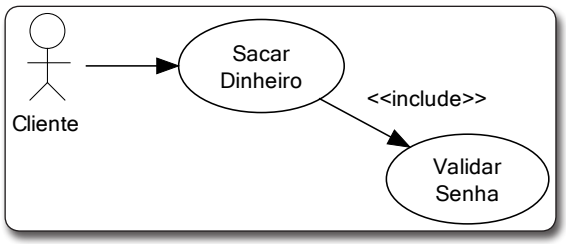


Figura 3. Exemplo de relacionamento de inclusão

Já o relacionamento de extensão entre casos de uso, identificado pelo estereotipo "estende" (do inglês "extend"), significa que um caso de uso incorpora implicitamente o comportamento de um outro caso de uso, em um local especificado indiretamente pelo caso de uso estendido. Este relacionamento é usado para modelar um comportamento que ocorre em situações específicas, de modo a separar os comportamentos opcionais do comportamento obrigatório de um caso de uso. A **Figura 4** é um exemplo deste tipo de relacionamento.

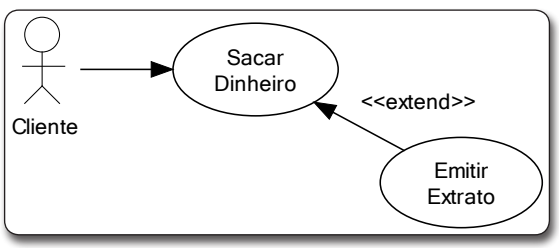


Figura 4. Exemplo de relacionamento de extensão

Organizar os casos de uso, extraindo o comportamento comum, por meio de relacionamentos de inclusão, e diferenciando as variantes, através de relacionamentos estendidos, é uma parte importante para a criação de um conjunto de casos de uso simples, equilibrado e compreensível, para o sistema [BOO 99].

Discussão sobre Cenários e Casos de Uso

Existem ainda várias discussões ao redor de cenários e casos de uso. Os termos "cenário" e "caso de uso" significam diferentes coisas para diferentes pessoas: suas definições, muitas vezes, não são claras, ora sendo utilizados como sinônimos, ora como conceitos distintos.

Em [COC 97a, COC 97b], encontra-se um estudo onde é apontado que as diversas definições de casos de uso diferem sobre quatro dimensões:

- Propósito: casos de uso podem ter o propósito de descrever histórias dos usuários ou especificar requisitos;
- Conteúdo: o conteúdo de um caso de uso pode ser contraditório ou consistente; quando consistente, pode estar na forma de narrativa textual ou notação formal;
- Pluralidade: casos de uso podem ser apenas um cenário ou conter múltiplos cenários;
- Estrutura: uma coleção de casos de uso pode ser representada através de uma estrutura formal, uma estrutura informal, ou, simplesmente, através de um conjunto não estruturado.

A definição original de casos de uso proposta por Jacobson [JAC 92] e, posteriormente, adotada na UML, tem o propósito de especificar requisitos, com o conteúdo na forma de uma narrativa textual consistente, contendo múltiplos cenários e possuindo uma estrutura semiformal.

Entretanto, o conceito de casos de uso ainda é vago e confuso, e existem vários aspectos problemáticos que precisam ser tratados. Abordagens apontam que casos de uso apresentam problemas conceituais, alguns deles são destacados em [REG 95a, REG 95b, CON 2000b] e mostrados a seguir:

- Notação e conteúdo: O que exatamente casos de uso contém? Como são organizados os conteúdos? O que significam os conteúdos?
- Expressões idiomáticas: Qual linguagem é usada para expressar os conteúdos que definem um caso de uso?
- Granularidade: O tamanho e o nível de detalhamento de cada caso de uso é escolhido arbitrariamente?
- Cobertura: Casos de uso podem garantir apenas uma cobertura parcial de todos possíveis usos do sistema?
- Contexto: Casos de uso ocorrem sobre quaisquer situações ou sobre condições específicas?
- Interseção: Casos de uso são independentes ou podem sobrepor-se total ou parcialmente? Como modelar os relacionamentos?
- Concorrência: Como modelar as concorrências internas e entre casos de uso?

Outro problema pertinente é que, na UML, casos de uso geralmente são referentes à interação sob a ótica do sistema, não considerando adequadamente aspectos de usabilidade. Conforme já destacado em [CON 2000a], observa-se que, na UML, a corrente definição de casos de uso, desviou-se da ênfase original de Jacobson, no uso: *"Um caso de uso é uma maneira específica de usar um sistema..."*; para um ponto de vista mais centrado no sistema: *"Caso de uso é uma descrição de um conjunto de seqüências de ações... que um sistema realiza..."*. O foco está no que o sistema executa, não no que o usuário faz ou deseja.

Casos de uso têm sido usados na ES para ganhar percepção em possíveis modelos de software, não focando as tarefas dos usuários, enquanto que cenários, em IHC, capturam primeiramente o comportamento dos usuários. Deve-se atentar para não se modelar apenas a perspectiva do sistema, como apresentado abaixo:

Consultar Extrato
1. O caixa eletrônico captura a identificação;
2. O caixa eletrônico valida a identificação;
3. O caixa eletrônico coleta a transação extrato;
4. O caixa eletrônico coleta o período;
5. O caixa eletrônico valida o período;
6. O caixa eletrônico emite o recibo do extrato;
7. O caixa eletrônico fica pronto para nova transação.

Assim como deve-se atentar para não se modelar apenas a perspectiva do usuário:

Conceito	Cenário	Caso de Uso
Aplicação	Comumente utilizado em IHC para inspecionar atividades humanas ao usar um, presente ou futuro, artefato. Concentra-se em aspectos de usabilidade.	Comumente utilizado em ES para modelar requisitos funcionais e manipular modelos de objetos. Concentra-se em aspectos funcionais.
Pluralidade	Uma seqüência específica de ações, um único caminho.	Conjunto de seqüências de ações, vários caminhos.
Nível de abstração	Concreto.	Diferentes níveis de abstração, mas tipicamente mais abstratos que cenários.
Notação	Descrito em linguagem natural.	Descrito em linguagem natural, semiformal ou formal.
Estrutura	Não há estrutura para representar um conjunto de cenários.	Uma coleção de casos de uso possui uma estrutura. Na UML, são os diagramas de casos de uso.
Abordagem	Provê uma abordagem bottom-up. Cenários são generalizados e sintetizados em casos de uso.	Provê uma abordagem top-down. Casos de uso são refinados respeitando suas propriedades estruturais.

Tabela 3. Quadro sintético, destacando diferenças entre cenários e casos de uso [VED 2001]

Consultar Extrato
1. O cliente fornece sua identificação;
2. O cliente pede o extrato;
3. O cliente escolhe o período;
4. O cliente pega o recibo do extrato;
5. O cliente sai.

É necessário estar centrado nas tarefas dos usuários e nas responsabilidades do sistema para suportá-las, ou seja, no uso do sistema.

Como visto anteriormente, casos de uso e cenários são importantes técnicas de modelagem, amplamente utilizadas respectivamente em IHC e ES, em diferentes contextos, com diferentes visões; mas apresentando muitas similaridades. Embora ambos sejam descrições narrativas sobre interações, existem diferenças entre estes conceitos. A **Tabela 3** destaca resumidamente algumas delas.

Contudo, tem-se a convicção de serem conceitos valiosos para se combinarem as diferentes perspectivas das áreas de IHC e ES.

Fazendo uso do conceito de níveis de abstração e partindo do enfoque de cenários serem instâncias concretas de casos de uso, podem-se combinar os propósitos e usos complementares de ambas as técnicas. De fato, cenários têm sido utilizados com uma base comum entre diferentes tipos de modelagem. Um método de construção que integrasse as duas técnicas seria útil; pois permitiria uma busca integrada da qualidade interna e externa dos sistemas e, conseqüentemente, o desenvolvimento de sistemas interativos úteis e usáveis.

A integração destes conceitos às metodologias de desenvolvimento de software é também um importante aspecto a ser considerado. Em alguns casos, os conceitos são usados informalmente; em outros, fazem parte do processo de desenvolvimento; e, num ponto de destaque, quando suas potencialidades são exploradas mais efetivamente, são eles utilizados ao longo de todo o ciclo de vida do software, guiando o processo de desenvolvimento.

Todavia, não se podem representar através de casos de uso todos os requisitos de um sistema. Casos de uso são mais adequados para representar os requisitos comportamentais e os requisitos funcionais. Demais requisitos não funcionais (RNFs), como formato de dados, requisitos de performance, regras de negócio e fórmulas complexas, devem ser representados em outros formatos. Porém, casos de uso funcionam como uma estrutura central capaz de conectar vários tipos de requisitos, podendo ser ligadas a casos de uso informações associativas. Utilizando a metáfora da roda (**Figura 5**), considera-se casos de uso como o eixo de uma roda que conecta outras informações, associadas aos raios, levando a diferentes direções [COC 2000]. Esta é uma das razões pelas quais muitos consideram casos de uso como o elemento central na ER e no processo de desenvolvimento, como um todo.

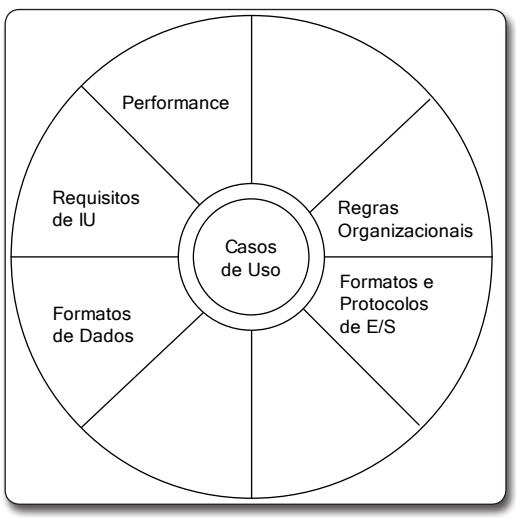


Figura 5. Modelo de requisitos "eixo e raios" [COC 2000]

Segundo [CYS 2001] os RNFs devem ser tratados desde as primeiras fases do desenvolvimento de software, trazendo para os casos de uso as informações e ações necessárias para satisfazerem o conjunto de RNFs elicitados. Os ciclos de requisitos funcionais (visão funcional) e de RNFs (visão não funcional) devem ser integrados através do uso de pontos convergentes [CYS 2001]. De fato, os RNFs são modelados através de outra notação, como, por exemplo, RNFs Framework [CYS 2001]),

não através de casos de uso; embora estes permitam ligar as duas visões. Casos de uso ajudam na clarificação do inter-relacionamento entre requisitos funcionais e RNFs.

Considerações Finais

Várias pesquisas têm focado principalmente as funções e os usos de cenários e casos de uso no processo de desenvolvimento de sistemas, enquanto que o processo de construção destes, muitas vezes, é negligenciado. Desenvolvedores, freqüentemente, não sabem como identificar casos de uso, o que incluir neles, qual o nível de detalhamento, como representá-los e como estruturá-los. A abordagem original de casos de uso [JAC 92, JAC 94a, JAC 94b, JAC 94c], posteriormente adotada na UML [BOO 99], não define precisamente nenhum formato específico para descrever seus conteúdos, nem um processo sistemático para construí-los. A falta de guias para a construção de casos de uso é certamente uma das desvantagens das abordagens baseadas em casos de uso para ER. Caso não sejam devidamente construídos, ou, ainda, sejam ambíguos, incompletos ou inconsistentes, as interações que os casos de uso representam também herdarão essas propriedades [ROL 98a]. ●

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

Referências

[ABO 94] ABOULAFIA, A.; KLAUSEN, T.; JORGENSEN, A. H. Towards a Theoretical Underpinning fo Scenarios. In: INFORMATION SYSTEMS RESEARCH SEMINAR, IRIS, 17., 1994, Finland. **Proceedings**... [S.l.:s.n.], 1994. p.561-571.

[BEN 93] BENNER, M. K. et al. Utilizing Scenarios in the Software Development Process. In: Information System Development Process, IFIP, 1993, Como, Italy. **IFIP WG8.1 Working Conference on Information System Development Process:** proceedings. [S.l.:s.n.], 1993

[BOO 99] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language User Guide**. Reading, MA: Addison-Wesley, 1999. 482 p.

[CAM 92] CAMPBELL, R. L. Will the real scenario please stand up? **SIGCHI Bulletin**, [S.l.], v. 24, n. 2, p.6-8, 1992.

[CAR 95] CARROLL, J. M. Artifacts and Scenarios: An Engineering Approach. In: MONK, A.; GILBERT, N. (Ed.). **Perspectives on HCI:** Diverse Approaches. London: Academic Press, 1995. chap.6, p.121-144.

[CAR 2000] CARROLL, J. M. Five Reasons for Scenario-Based Design. **Interacting with Computers**, [S.l.], v. 13, n. 1, p.61-75, 2000.

[COC 2000] COCKBURN, A. **Writing Effective Use Cases**. Boston: Addison-Wesley, 2000.

[CON 2000a] CONSTANTINE, L. et al. **Structure and Style in Use Cases for User Interface Design**. 2000. Disponível em: <http://www.foruse.com/ Presentations/ structurestyle2.pdf>. Acesso em: 20 fev. 2002.

[CON 2000b] CONSTANTINE, L.; LOCKWOOD, L. **The Usability Challenge**: Can UML and the Unified Process Meet It? Australia, Nov. 2000. Disponível em: <http://www.foruse.com/Presentations/tools.pdf>. Acesso em: 03 mar. 2002.

[CYS 2001] CYSNEIROS, L. M.; LEITE, J. C. S. P. Driving Non-Functional Requirements to Use Cases and Scenarios. In: SIMPÓSIO BRASILEIRO DE ENGENHARIA DE SOFTWARE, SBES, 15., 2001, Rio de Janeiro. **Anais**... Rio de Janeiro: UFRJ, 2001.

[FER 99] FERREIRA, Aurélio B. H. **Novo Dicionário Aurélio – Século XXI**. [S.l.]: Nova Fronteira, 1999.

[FUR 98] FURLAN, J. D. **Modelagem de Objetos através da UML**. São Paulo: Makron Books, 1998.

[JAC 92] JACOBSON, I. et al. **Object Oriented Software Engineering:** A Use Case Driven Approach. Workingham: Addison-Wesley, 1992.

[JAC 94a] JACOBSON, I. Basic Use-Case Modeling. Report on Object Analysis & Design, [S.l.], v. 1, n. 2, Aug. 1994.

[JAC 94b] JACOBSON, I. Use Cases an Objects. **Report on Object Analysis & Design,** [S.l.], v. 1, n. 4, Dec. 1994.

[KLA 93] KLAUSEN, T.; BERSEN, N.O. COSITUE: Towards a Methodology for Constructing Scenarios. In: COGNITIVE SCIENCE APPROACHES TO PROCESS CONTROL, CSAPC, 1993, Copenhagen. **Designing for Simplicity:** proceedings. [S.l.:s.n.], 1993, p.1-16.

[REG 96] REGNELL, B.; ANDERSSON, M.; BERGSTRAND, J. A Hierarchical Use Case Model with Graphical Representation. In: IEEE International Symposium and Workshop on Engineering of Computer Based Systems, Friedrichshafen. **Proceedings**... Los Alamitos: IEEE Computer Society, 1996.

[ROL 98a] ROLLAND, C.; ACHOUR, C. Ben. Guiding the Construction of Textual Use Case Specifications. **Data & Knowledge Engineering Journal**, Amsterdam, v. 25, n. 1-2, p.125-160, Mar. 1998.

[RYS 2000] RYSER, J.; GLINZ, M. **SCENT**: A Method Employing Scenarios to Systematically Derive Test Cases for System Test. Zrich: Universitt Zrich, Institut fr Informatik, Berichte des Instituts fr Informatik, 2000.

[VED 2001] VEDOATO, Roberto. **Uma Revisão Bibliográfica sobre Cenários e Casos de Uso:** Rumo à Integração de Engenharia de Software e Interação Humano-Computador no Desenvolvimento de Sistemas Interativos. 2001. Trabalho Individual (Mestrado em Ciência da Computação) – Instituto de Informática, UFRGS, Porto Alegre. (TI-999).

Modéstia à parte, sua melhor opção para se destacar no mercado!

A Escola Superior da Tecnologia da Informação oferece as melhores opções em cursos, formações, graduações e pós-graduações para profissionais de desenvolvimento e programação.

São programas voltados para a formação de profissionais de elite, com **aulas 100% práticas, corpo docente atuante no mercado**, acesso à mais atualizada biblioteca de TI do Rio, laboratórios equipados com tecnologia de ponta, salas de estudo e exames.

PÓS-GRADUAÇÃO

- **Engenharia de Software: Desenvolvimento Java**

GRADUAÇÃO

- **Análise e Desenvolvimento de Sistemas**

FORMAÇÕES

- **Desenvolvedor Java**
- **Desenvolvedor Java: Sistemas Distribuídos**
- **Gestor de TI**
- **Desenvolvedor Web .NET 2008**
- **MCITP Server Administrator**
- **SQL Server 2008**

Acesse nosso site e conheça todos os nossos programas: **www.infnet.edu.br/esti**

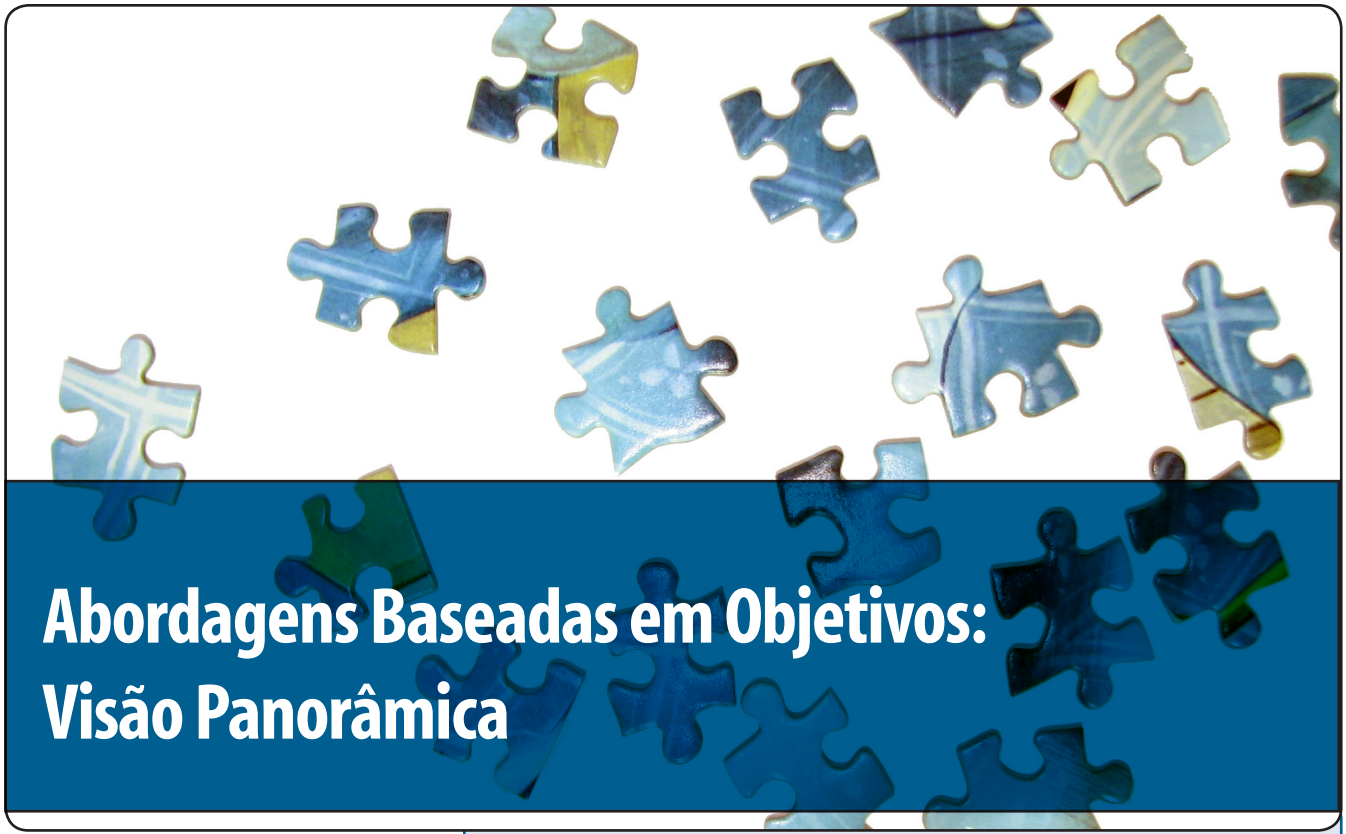
TURMAS NO RIO DE JANEIRO

infnet INSTITUTO

ESCOLA SUPERIOR DA TECNOLOGIA DA INFORMAÇÃO

www.infnet.edu.br - cursos@infnet.edu.br - Central de Atendimento: (21) 2122-8800

EDUCAÇÃO SUPERIOR ORIENTADA AO MERCADO



Abordagens Baseadas em Objetivos: Visão Panorâmica

De que se trata o artigo?

Apresentar a abordagem baseada em objetivos. Diferente das análises de sistemas tradicionais que determinam quais características o sistema deve suportar, a abordagem baseada em objetivos focam o porquê os sistemas são construídos.

Para que serve?

Este tipo de abordagem possibilita uma análise mais ampla do contexto onde o sistema será implementado. Focar objetivos, ao invés de requisitos específicos permite a comunicação entre analistas e stakeholders, através de uma linguagem baseada em conceitos que ambos têm conhecimento.

Em que situação o tema é útil?

Quando se deseja modificar a forma de definição dos requisitos de um sistema empregando uma abordagem que facilita a comunicação entre analistas e stakeholders. Para isso três abordagens baseadas em objetivos para construção de casos de uso e/ou cenários serão explicadas.

Roberto Vedoato

Mestre em Ciência da Computação pela UFRGS. Especialista e Bacharel em Ciência da Computação pela UEL. Foi Professor das Universidades Estaduais UDESC, UEL e UENP, e atualmente é Analista de Sistemas do Ministério Público do Trabalho. Tem experiência de 15 anos na área de Computação, com ênfase em Engenharia de Software e Interação Humano Computador.

Ao se criar um software, não se planeja e não se desenvolve apenas um artefato, criam-se novas possibilidades para ações e interações humanas [CAR 95]. Todo software é uma ferramenta. Como boa ferramenta, deve suportar o trabalho de alguém, tornar o trabalho mais fácil, mais rápido, mais simples e mais flexível. Para desenvolverem-se sistemas que suportam adequadamente o uso, trabalhos de pesquisa enfatizam que se precisa entender melhor as tarefas realizadas pelas pessoas e aplicar de maneira mais eficiente a compreensão das tarefas no processo de desenvolvimento de software.

As atividades de pessoas, no trabalho, podem ser descritas como "tarefas". Segundo [STO 95] o termo "tarefa" é definido como: *"uma tarefa é um objetivo junto com conjuntos ordenados de ações que o satisfariam em contextos apropriados"*. Baseando-se na teoria da ação, sabe-se que, antes de executar uma seqüência de ações, elaboram-se planos, e o ponto inicial de um plano é a formulação de um objetivo [CAR 95]. Seguindo esta perspectiva, está claro que, para desenvolver um sistema interativo, é necessário, em primeiro lugar, conhecer os objetivos dos usuários para, então, propor a especificação dos mecanismos que sustentarão as tarefas necessárias para alcançá-los.

Enquanto análises de sistema tradicionais focam "quais" características um sistema irá suportar, abordagens baseadas em objetivos focam "por que" sistemas são construídos, provendo motivação e argumento para justificar os requisitos de software [ANT 96]. Possibilitam assim uma análise mais ampla do contexto no qual o sistema irá operar. Focar objetivos, ao invés de requisitos específicos, permite analistas comunicarem com *stakeholders* - pessoas envolvidas no sistema em desenvolvimento -, usando uma linguagem baseada em conceitos que ambos têm familiaridade [ANT 96]. Todavia, especificar os requisitos de um sistema, a partir dos objetivos dos usuários, não é uma tarefa trivial. **Objetivos** são estados finais desejáveis de serem alcançados, não ações a serem realizadas [ANT 96]. O refinamento de objetivos em soluções de projeto é um processo de múltiplos estágios; vagos objetivos de alto nível devem ser refinados em concretos objetivos formais [KAV 96]. Isto é necessário, pois somente objetivos primitivos podem ser operacionalizados - traduzidos em ações atômicas do usuário, do hardware ou do sistema – para, então, se tornarem requisitos operacionais na especificação final de requisitos [POT 95].

Um dos fatores positivos de se enfatizarem objetivos, como fontes para o desenvolvimento, é que eles são consideravelmente estáveis [ANT 96]. Vários sistemas podem ser propostos para serem realizados os objetivos dos usuários: eles diferirão basicamente em como os objetivos são operacionalizados, ou seja, quais ações serão executadas para que os objetivos sejam alcançados e quem ou o que as realizará. Isto implica que é possível, a partir dos objetivos, especificar as futuras tarefas com o sistema, ou seja, as tarefas interativas [PIM 97].

Deve-se considerar que a operacionalização dos objetivos não deve ser predominantemente *top-down*, por ser altamente interativa [KAV 96]. Derivação de casos de uso e refinamento de objetivos são atividades do desenvolvimento de software que se apóiam mutuamente [POT 95, ROL 98b]. O conhecimento teleológico preocupa-se com os propósitos específicos para os quais o sistema foi projetado, enquanto a análise de casos de uso é dedicada a preencher a lacuna entre tais propósitos abstratos e a real estrutura e comportamento do sistema [KAV 96].

Casos de uso especificam um modo de operacionalizar um objetivo. Assim, é possível obter um conjunto de casos de uso, analisando objetivos, alocação de objetivos etc. Reciprocamente, é possível retornar e elaborar objetivos, quando se caminha através de um caso de uso, já que ele é uma forma natural para se descreverem as circunstâncias nas quais um objetivo pode falhar ou ser bloqueado, facilitando a descoberta de novos objetivos e a consideração de alternativas para a operacionalização dos mesmos. A análise do caso de uso pode fornecer percepções concretas sobre o comportamento do macrosistema - ambiente geral no qual o software é desenvolvido e deve operar - e as razões para tal, possibilitando a identificação de soluções mais plausíveis [POT 95, ANT 98a].

Trabalhos de pesquisa já reconhecem a importância da relação entre objetivos e casos uso. Segundo Kaindl, somente se podem entender as interações descritas em um caso de uso, quando se conhece seu objetivo [KAI 95]. Quando se está familiarizado com um sistema, os objetivos das interações parecem óbvios; todavia, no processo de elicitação de requisitos e modelagem de tarefas de um novo sistema, ainda desconhecido, saber o objetivo é uma informação crucial para a compreensão de cada caso de uso. De acordo com a abordagem [KAI 95] os propósitos da utilização de casos de uso são claramente definidos, já os propósitos das interações descritas nestes, geralmente são ignorados ou deixados implícitos. Faltam representações adequadas de objetivos em casos de uso.

Na realidade, outras abordagens também consideram serem os objetivos fundamentais para construção de cenários e/ou casos de uso [KLA 93, POT 95, KAV 96, REG 96, COC 97a, COC 97b, ROL 98b, CON 2000a]; porém poucos oferecem um modo de se utilizarem complementarmente as duas técnicas para a determinação dos requisitos. Na maioria das abordagens, ou não existe uma notação para objetivos, ou não existe um processo sistemático de construção e validação de casos de uso, a partir dos objetivos, ou, até mesmo não, existem ambos.

Algumas Propostas

Nesta seção, são apresentadas sucintamente três abordagens baseadas em objetivos para construção de casos de uso e/ou cenários. Notifica-se que se utilizará, ao longo deste artigo, os termos "cenário" e "casos de uso", de acordo com o entendimento apresentado no artigo Cenários e Casos de uso: Fundamentos e Conceitos, também publicado nesta edição da Engenharia de Software Magazine.

Abordagem de Potts [POT 94, POT 95]

Em [POT 94, POT 95], é apresentada a proposta de Colin Potts para a construção de casos de uso. Nesta abordagem, conceitos teleológicos são utilizados para gerarem casos de uso esquemáticos, utilizados para compreender as necessidades dos usuários.

É proposto um esquema de caso de uso com estrutura teleológica (**Figura 1**) que, utilizado em conjunto com uma estratégia de refinamento de objetivos, guia a produção de casos de uso. O princípio é que a seção narrativa de um caso de uso consiste em uma seqüência de episódios, cada um correspondente a algum objetivo ou obstáculo. É **obstáculo** definido como comportamento ou, outro objetivo que bloqueia a realização de um dado objetivo e **episódio** é definido como um conjunto particular de ações (plano), executadas sob condições, que tornam possível a descrição operacional de um objetivo do domínio do problema [POT 95].

Em [POT 95], é apontado que, diferentemente das intuições dos usuários, a hierarquia dos objetivos ajuda a delimitar a cobertura dos casos de uso. Porém, um potencial número ilimitado de casos de uso pode ser derivado a partir dessa hierarquia. Heurísticas são utilizadas para guiarem a obtenção do conjunto de casos de uso "salientes". Um caso de uso saliente tem um propósito, não é redundante e auxilia os usuários e desenvolvedores a levantarem e entenderem algum aspecto do sistema proposto que deve ser resolvido antes de poder ser dito que o sistema atende as necessidades dos usuários.

De acordo com a abordagem, várias atividades são necessárias antes que os casos de uso esquemáticos possam ser produzidos. Os objetivos para o domínio devem ser identificados e decompostos; obstáculos devem ser identificados e analisados para decidir quais merecem futura atenção; e, um ou mais sistemas que cumprem os objetivos identificados devem ser propostos. Com respeito à decomposição de objetivos, apenas é observada a similaridade com a análise hierárquica de tarefas [POT 95] e, para a identificação de obstáculos, é sugerida a realização de um questionamento sistemático [POT 94]. Os relacionamentos entre os objetivos e atores, assim como os relacionamentos entre objetivos, obstáculos e objetivos defensivos ou de suavização são representados por tabelas.

Segundo Potts [POT 95], casos de uso esquemáticos têm a vantagem de poder esclarecer antes da implementação como o sistema proposto, ou propostas alternativas irão afetar os reais objetivos dos usuários em situações atípicas de uso; mas significantes.

Abordagem de Cockburn [COC 97a, COC 97b, COC 98, COC 2000]

Em [COC 97a, COC 97b, COC 98, COC 2000] é proposta, por Alistair Cockburn, uma teoria que utiliza objetivos como o elemento chave dos casos de uso. A abordagem considera que os objetivos podem resumir as funções do sistema em termos compreensíveis e verificáveis.

O foco principal da abordagem é a escrita de casos de uso efetivos, sendo apresentadas técnicas de como pensar, como redigir sentenças e em que seqüência trabalhar. Os modelos e as técnicas apresentados na abordagem têm sido aplicados e avaliados em projetos reais. Os trabalhos [COC 97a, COC 97b, COC 2000] são descritos como resultados destas experiências.

Scenario → Setting Narrative Evaluation
Setting → Background Roles
Roles → Role*
Narrative → Episode+
Episode → Goal Action Outcome

Legend: Asterisks represent zero or more repetitions, braces represent optional categories, and the vertical bar represents alternatives.

Figura 1. Esquema de casos de uso [POT 95]

Na abordagem de Cockburn, é proposto um modelo para descrever os casos de uso que utiliza conceitos teleológicos; mas que, diferentemente, da abordagem apresentada anteriormente, não possui uma estrutura episódica. O modelo pode ser usado tanto na forma textual quanto na tabular; em ambos os casos, a descrição dos casos de uso é divida em seções [COC 98]:

- Nome do caso de uso: uma pequena frase verbal descrevendo o objetivo;
- Objetivo no contexto: uma descrição mais detalhada a respeito do objetivo, caso necessário;
- Escopo: qual sistema deve ser considerado uma "caixa-preta" sobre o ponto de vista do projeto;
- Nível: o caso de uso pode ser de algum dos três tipos: resumo, tarefa primária ou subfunção;
- Pré-condição: as condições esperadas do contexto, antes de iniciar o caso de uso;
- Condição de termino bem sucedido: as condições esperadas do contexto, após a realização completa do caso de uso;
- Condição de termino com falha: as condições esperadas do contexto, após o abandono do caso de uso;
- Ator principal: o nome ou a descrição do ator principal;
- Gatilho (*Trigger*): a ação sobre o sistema que inicia o caso de uso;
- Curso principal: seqüência de passos do curso normal, descrevendo ações, a partir da ativação do caso de uso;
- Extensões: lista de condições, cada uma se referindo ao passo do curso principal;
- Subvariações: as subvariações que, eventualmente, possam causar bifurcação no caso de uso;
- Informações adicionais: outros dados importantes para o caso de uso, como prioridade, performance, freqüência, atores secundários e casos de uso relacionados.

A abordagem de Cockburn [COC 97a, COC 97b, COC 2000] faz uso de metáforas para representar os objetivos e os casos de uso. Os objetivos em seus diversos níveis são estruturados hierarquicamente, formando um gráfico que se assemelha a um "veleiro" (do inglês "*sailing ship*"), ilustrado na **Figura 2**. Já casos de uso são associados a

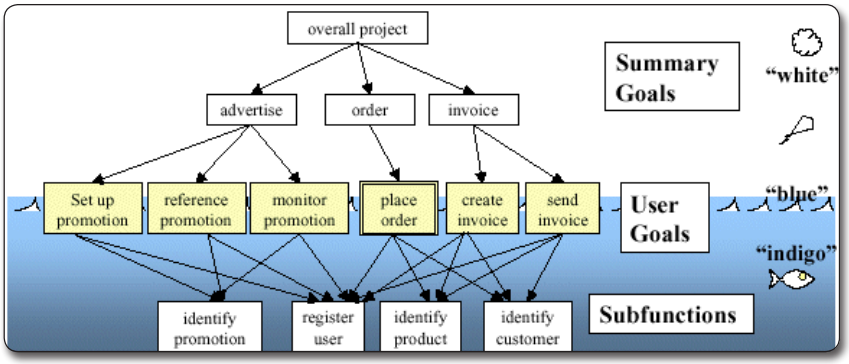


Figura 2. Metáfora do "veleiro" (*sailing ship*) [COC 2000]

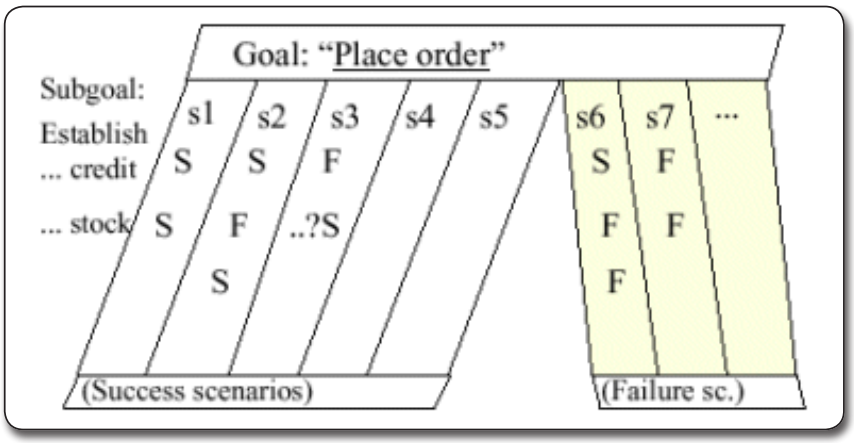


Figura 3. Metáfora da "calça-listrada" (*striped trousers*) [COC 2000]

"calças listradas" (do inglês "*striped trousers*") com pernas de sucesso e falha (**Figura 3**). O cinto da calça é o objetivo que aglutina todos os cenários, cada listra um cenário, e cada linha, em um cenário, uma ação primitiva ou um objetivo de um caso de uso subordinado. Essa analogia provê uma visão sintetizada da coleção dos caminhos de um caso de uso.

Em [COC 97a, COC 97b, COC 2000], a explosão do número de casos de uso é evitada, utilizando três técnicas: casos de uso subordinados, extensões e variações. Para ligar casos de uso a requisitos de interface, de performance e aos atores, é sugerido o uso de tabelas auxiliares.

Abordagem de Rolland [ROL 98a, ROL 98b, ROL 99]

A abordagem CREWS-*L'Ecritoire* de Colette Rolland é parte integrante do projeto CREWS (*Cooperative Requirements Engineering With Scenarios*), um longo projeto de pesquisa, iniciado em 1996, pela Comunidade Européia. A abordagem visa elicitar requisitos, explorando o relacionamento bidirecional entre a autoria de cenários textuais e a descoberta de objetivos. Para cada objetivo descoberto, é escrito um cenário que, posteriormente, é analisado para descobrir novos objetivos, o que leva à descrição de novos cenários e assim sucessivamente. Os requisitos elicitados são expressos na forma de uma hierarquia de objetivos e cenários.

A idéia central da abordagem CREWS-*L'Ecritoire* é o conceito de *Requirements Chunks* (RC) definido como um par de objetivo e cenário. Um objetivo é de natureza intencional; e um cenário, de natureza operacional; um RC é uma possível maneira de alcançar um objetivo. Os RCs são inter-relacionados por composição, alternativa e refinamento, constituindo um sistema de conceitos, denominado *Requirement Chunck Model* (RC *Model*).

Os RCs são organizados hierarquicamente em três níveis de abstração: contextual, funcional e físico. O nível contextual tem o propósito de identificar os serviços que um sistema deve prover para uma organização. O foco do nível funcional é nas interações, entre o sistema e seus usuários, necessárias para alcançarem os serviços atribuídos ao sistema ao nível contextual. O nível físico foca quais ações internas o sistema precisa para executar as interações selecionadas ao nível funcional.

Uma ênfase particular é dada na exploração de cenários textuais, descritos em linguagem natural pelos *stakeholders*. São propostas guias para autoria de cenários e elicitação de objetivos.

A elicitação de objetivos é baseada em análise lingüística de declarações de objetivos. O processo guiado de autoria de cenários é dividido em dois estágios principais: a escrita dos cenários e a correção dos mesmos. Para guiar a escrita dos cenários são propostas guias de estilo e conteúdo referentes a um modelo conceitual e um modelo lingüístico de cenários. Para guiar a correção dos cenários é proposto um conjunto de regras. Cenários escritos, em prosa informal, são progressivamente

Abordagem Critério	Potts	Cockburn	Rolland
Investigação das singularidades	Um questionamento sistemático e a criação de ações defensivas e corretivas	Um questionamento sistemático e algumas guias.	Algumas guias
Delimitação do conjunto de casos de uso	Algumas heurísticas para determinar a saliência dos casos de uso. Uso do conceito de episódios	Uso de técnicas de subordinação, extensão e variação	Sugere que os objetivos delimitam o número de casos de uso. Uso do conceito de episódios
Uso complementar dos conceitos de cenários e casos de uso	Apenas é sugerido, sem especificar, que cenários podem ser instanciados para serem avaliados pelos usuários	Cenários não são usados para experimentação com os usuários	Cenários são usados apenas num primeiro momento para elicitar objetivos e posteriormente são integrados em casos de uso. Não são, porém, usados para experimentação com os usuários
Avaliação da usabilidade com os usuários	Não	Não	Não
Diferentes níveis de abstração	Não	Sumário, objetivos do usuário e subfunções	Contextual, funcional e físico
Nível de abstração que estabelece ligação com abordagens OO	Não	Não	Não
Notações definidas	Sim	Sim	Sim
Guias para a descrição textual dos casos de uso	Não	Sim	Sim
Relações temporais nos casos de uso	Não	Não	Não
Fluxo dos eventos	Seqüencial	Seqüencial	Seqüencial
Consideração de eventos assíncronos	Não	Sugere tratá-los em uma seção separada, usando asterisco (*) para identificá-los	Não
Representação de Objetivos	Apenas é sugerida uma hierarquia de objetivos similar a modelos de tarefas	Metáfora do veleiro	Requirements Chunks organizados hierarquicamente
Representação da ontologia	Não	Não	Não

Tabela 1. Comparação entre abordagens baseadas em objetivos

transformados em textos estruturados e não ambíguos, integrados ao RC *Model*. Através de estratégias de composição e alternativa, diferentes cenários são integrados em um caso de uso.

Análise Crítica das Abordagens

A análise destes trabalhos permitiu a avaliação da importância de relacionar objetivos com casos de uso. A abordagem de Potts [POT 94, POT 95], em especial, clarifica a importância da consideração de obstáculos. A abordagem de Cockburn [COC 97a, COC 97b, COC 98, COC 2000] é um relato prático de como casos de uso têm sido utilizados em projetos e possui valiosas guias para a construção e descrição de casos de uso, assim como sugere o uso de uma simples, mas eficiente, lista para relacionarmos atores a objetivos. Diferentemente das duas abordagens anteriores, a CREWS-*L'Ecritoire* de Rolland [ROL 98a, ROL 98b, ROL 99] utiliza primeiramente cenários e propõe análises sintáticas e semânticas destes para a elicitação de objetivos e obstáculos e, também, para a integração dos cenários em casos de uso. As análises lingüísticas, sintáticas e semânticas, são relativamente complexas. Deste modo, precisa-se de ferramentas especializadas para utilizar a abordagem CREWS-*L'Ecritoire*, mesmo para pequenos projetos. De fato, em [ROL 98b] já é apresentada uma ferramenta para a aplicação da abordagem.

Uma análise comparativa das abordagens é apresentada sinteticamente através da **Tabela 1.** ●

Referências

[ANT 96] ANTÓN, A. I. Goal-Based Requirements Analysis. In: INTERNATIONAL CONFERENCE ON REQUIREMENTS ENGINEERING, ICRE, 1996, Colorado, USA. Proceedings... Colorado Springs: ICRE, 1996. p.136-144.

[ANT 98a] ANTÓN, A. I.; POTTS, C. A Representational Framework for Scenarios of Systems Use. Requirements Engineering Journal, [S.l.], v.3, n.3-4, p.219-241, 1998.

[CAR 95] CARROLL, J. M. Artifacts and Scenarios: An Engineering Approach. In: MONK, A.; GILBERT, N. (Ed.). Perspectives on HCI: Diverse Approaches. London: Academic Press, 1995. chap.6, p.121-144.

[CAR 2000] CARROLL, J. M. Five Reasons for Scenario-Based Design. Interacting with Computers, [S.l.], v. 13, n. 1, p.61-75, 2000.

[COC 97a] COCKBURN, A. Structuring Use Cases with Goals (Part 1). Journal of Object Oriented Programming, [S.l.], v. 9, n. 5, p.35-40, Sept./Oct. 1997.

[COC 97b] COCKBURN, A. Structuring Use Cases with Goals (Part 2). Journal of Object Oriented Programming, [S.l.], v. 9, n. 6, p.56-62, Nov./Dec. 1997.

[COC 98] COCKBURN, A. Basic Use Case Template. Oct. 1998. Disponível em: <http://www.members.aol.com/acockburn/papers/uctempla.htm>. Acesso em: 08 jan. 2002.

[COC 2000] COCKBURN, A. Writing Effective Use Cases. Boston: Addison-Wesley, 2000.

[KAI 95] KAINDL, H. An Integration of Scenarios with Their Purposes in Task Modelling. In: SYMPOSIUM ON DESIGNING INTERATIVE SYSTEMS, DIS, 1995, Ann Arbor, USA. Processes, Practices, Methods & Techniques: proceedings. [Ann Arbor: ACM], 1995. p.227-235.

[KLA 93] KLAUSEN, T.; BERSEN, N.O. COSITUE: Towards a Methodology for Constructing Scenarios. In: COGNITIVE SCIENCE APPROACHES TO PROCESS CONTROL, CSAPC, 1993, Copenhagen. Designing for Simplicity: proceedings. [S.l.:s.n.], 1993, p.1-16.

[KAV 96] KAVAKLI, E.; LOCOPOULOS, P.; FILIPPIDOU, D. Using Scenarios to Systematically Support Goal-Direct Elaboration for Information System Requirements. In: IEEE Symposium and Workshop on Engineering of Computer-Based System, ECBS, 1996, Friedrichshafen, Germany. Proceedings... [Los Alamitos: IEEE Computer Society], 1996. p.308-314.

[PIM 97] PIMENTA, Marcelo S. TAREFA: Une Approche pour l'Ingénierie des Besoins des Systèmes Interactifs. 1997. 285p. Tese (Doutorado) - Université Toulouse, Toulouse.

[POT 94] POTTS, C.; TAKAHASHI, K.; ANTÓN, A. Inquiry-Based Scenario Analysis of Systm Requirements. [S.l.]: Georgia Tech College of Computing, Jan. 1994. Technical Report.

[POT 95] POTTS, C. Using Schematic Scenarios to Understand User Needs. In: SYMPOSIUM ON DESIGNING INTERATIVE SYSTEMS, DIS, 1995, Ann Arbor, USA. Processes, Practices, Methods & Techniques: proceedings. [Ann Arbor: ACM], 1995. p.247-256.

[REG 96] REGNELL, B.; ANDERSSON, M.; BERGSTRAND, J. A Hierarchical Use Case Model with Graphical Representation. In: IEEE International Symposium and Workshop on Engineering of Computer Based Systems, Friedrichshafen. Proceedings... Los Alamitos: IEEE Computer Society, 1996.

[ROL 98a] ROLLAND, C.; ACHOUR, C. Ben. Guiding the Construction of Textual Use Case Specifications. Data & Knowledge Engineering Journal, Amsterdam, v. 25, n. 1-2, p.125-160, Mar. 1998.

[ROL 98b] ROLLAND, C.; SOUVEYET, C.; ACHOUR, C. Ben. Guiding Goal Modeling Using Scenarios. IEEE Transactions on Software Engineering, New York, v. 24, n. 12, p.1055-1071, Dec. 1998 .

[ROL 99] ROLLAND, C. et al. Experience With Goal-Scenario Coupling In Requirements Engineering. In: IEEE INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING, RE, 4., 1999, Limerick, Ireland. Proceedings... [S.l.]: IEEE Computer Society, 1999.

[STO 95] STORRS, Graham. The Notion of Task in Human-Computer Interaction. In: HUMAN COMPUTER INTERACTION, HCI, 1995, Huddersfield. Preceedings... Huddersfield: Cambridge University Press, 1995. p.357-365.

Motivação em Integrantes de Equipes de Desenvolvimento de Software

César C. França
cesarfranca@gmail.com
É mestre em Ciência da Computação (UFPE), com foco em Engenharia de Software experimental. Autor do livro "UM ESTUDO SOBRE MOTIVAÇÃO EM INTEGRANTES DE EQUIPES DE DESENVOLVIMENTO DE SOFTWARE" (Ed. Universitária da UFPE, 2009). Escritor de artigos científicos e palestrante em congressos e conferências de Tecnologia. Consultor independente e sócio da Experience IT, atuando nas áreas de capital humano para TI, planejamento e gerenciamento de projetos e negócios online. Editor do blog QueroFicarRico.com, onde escreve sobre educação financeira, gerenciamento financeiro pessoal, mercado de capitais, investimento e negócios.

De que se trata o artigo?
O artigo aborda as principais teorias e conceitos relacionados à motivação e ao gerenciamento de equipes em projetos de software. Para isso apresentaremos um pouco do estudo da motivação humana aplicada à Engenharia de Software.

Para que serve?
Este artigo serve para entendermos a importância da motivação humana, tendo o objetivo de saber como motivar as equipes de desenvolvimento de software e elevar os índices de produtividade.

Em que situação o tema é útil?
Este tema é útil para todos os gerentes de projeto e pessoas que pretendem um dia liderar equipes de desenvolvimento de software. A partir deste estudo você saberá como motivar suas equipes e obter melhores resultados nos projetos.

Os impactos econômicos resultantes de práticas de gestão do comportamento têm despertado o interesse da indústria, fundamentadas em duas ideias principais: a primeira corresponde à necessidade de criação de um clima organizacional onde os colaboradores possam obter respeito e valorização; e a segunda é a premissa de que pessoas motivadas produzem mais. Por isso, constantemente surgem novas abordagens do tema aplicado à gestão empresarial. Especialmente as práticas motivacionais tornaram-se tão populares dentro das organizações, que atualmente são consideradas indispensáveis para a construção de um diferencial competitivo.

Mas, de modo geral, o que é motivação? O que a motivação das pessoas pode causar numa organização? A motivação das pessoas é um aspecto relevante para o sucesso de projetos? O que a administração da motivação pode ter a ver com a engenharia de software? E o que a indústria e a academia pensam sobre o tema? Estas são as perguntas centrais que nortearam o desenvolvimento deste trabalho.

Um conjunto de enquetes promovido pela SkillSoft[1] em 2006, envolvendo 2.800 profissionais revelou o setor de TI como o de maior número de profissionais estressados, ficando a frente de outros setores considerados críticos, como medicina, finanças e vendas. Esta pesquisa levantou números indicando que 28% dos profissionais de TI estariam insatisfeitos com o atual trabalho (JAGGS, 2006a). Mais além, 75% deles trocariam de emprego na primeira oportunidade (JAGGS, 2006b). Outro relatório técnico recente cita ainda que 41% dos profissionais espalhados pelo mundo sentem-se desmotivados no emprego (SKILLSOFT, 2008).

A indústria de software é composta em sua maioria por trabalhadores do conhecimento[2]. Por isso, os números apresentados sobre a desmotivação dos engenheiros de software[3] representam um problema suficientemente grande a ser resolvido. Em termos práticos, entre os fatores que são determinantes para o sucesso dos projetos de software, citados pelo CHAOS Report (STANDISH, 1995), dois deles envolvem aspectos pessoais: equipe competente e focada, trabalho duro.

O comportamento de equipes de desenvolvimento de software, em particular, tem sido estudado desde os primórdios da engenharia de software. Inclusive, a quantidade de pesquisas relacionadas com o tema vem crescendo nos últimos anos. Estas pesquisas focam principalmente a identificação de fatores que caracterizam a efetividade e em métodos de montagem de equipes de sucesso. Este aumento é justificado pelo reconhecimento, tanto da indústria quanto da academia, de que diversos desafios inerentes a um projeto de software problemático estão relacionados a uma gestão ineficaz de recursos humanos.

Todavia, são escassas as pesquisas que abordam especificamente a *motivação* como uma alternativa para alavancar o sucesso em projetos de software. Esta escassez pode ser justificada pela complexidade da mensuração dos aspectos humanos envolvidos neste tipo de estudo, o que dificulta a obtenção de resultados práticos. Apesar das disciplinas *administração do comportamento organizacional* e *psicologia organizacional* realizarem pesquisas sobre motivação há pelo menos 60 anos, na engenharia de software estes estudos ainda apresentam um caráter essencialmente teórico e com pouca aplicabilidade prática.

Uma vez ressaltado que grande parte dos problemas em projetos de software poderia ser combatida através da adoção de uma sistematização nos processos de gestão dos aspectos humanos – inclusive da motivação – a seguinte **problemática** é levantada: como motivar equipes de software?

O **objetivo central** deste artigo é apresentar as principais teorias e conceitos relacionados à *motivação* e ao gerenciamento de equipes em projetos de software. Para isto, o artigo está estruturado da seguinte forma: inicialmente é apresentada a origem do estudo da motivação humana, bem como suas principais teorias. Em seguida, são apresentados trabalhos teóricos sobre equipes e sobre o estudo do trabalho em equipe na engenharia de software. Por fim, são apresentadas, analisadas e discutidas as pesquisas relacionadas à administração da motivação aplicada à engenharia de software.

Referencial Teórico

Este artigo reúne as principais teorias e conceitos relacionados à *motivação* e ao gerenciamento de equipes em projetos de software. Estes conceitos, necessários e suficientes para o entendimento e a correta interpretação dos resultados apresentados neste artigo são resultantes de uma ampla revisão bibliográfica. A Seção Estudo da Motivação Humanda apresenta um estudo sobre a motivação humana e a sua aplicação na administração de empresas. A Seção Equipes e Engenharia de Software apresenta referências sobre o gerenciamento de equipes em projetos de software. Por fim, a Seção O Estudo da Motivação em Equipes de Software apresenta pesquisas sobre motivação no campo da engenharia de software.

O Estudo da Motivação Humana

A motivação humana tem sido estudada desde o início do século XX. Teve suas bases originadas na psicoterapia, na psicometria[4] e nas teorias da aprendizagem (TODOROV e MOREIRA, 2005). Apenas a partir de 1930 o tema difundiu-se nas escolas de administração do relacionamento humano. Desde aquela época, então, a definição do termo *motivação* tem levantado polêmica e estimulado pesquisadores e administradores. Isto resultou no surgimento de uma infinidade de teorias sobre o assunto (BUENO, 2002; SALGADO, 2005). Atualmente é possível encontrar diversas definições para o termo, como exemplificado na **Tabela 1**.

A palavra *motivação* é derivada do Latim *motus*, uma variação do verbo *movere*, que também originou outras palavras para o português, como *motor* e *motivo*. Segundo Golembiewski (2005), ainda é possível encontrar outras 140 definições acadêmicas para o termo *motivação*. Mas partindo da análise das diversas definições apresentadas na **Tabela 1**, é possível inferir que elas convergem para um conjunto de características comuns: a *motivação* é interna ao indivíduo; varia de acordo com o objetivo; apresenta intensidade, força e duração; e determina o comportamento. Estas características são resumidas na **Figura 1**.

1 - A SkillSoft Corporation é uma empresa estadunidense que oferece serviços de educação à distância e produtos de software para negócios e profissionais de TI. Desde1999 é PMI® *Registered Education Provider* (Provedor oficial de serviços educacionais) (PMI, 2008).

2 - Trabalhadores do conhecimento são aqueles que constroem seu conhecimento com base na experiência prática. Eles atuam diretamente em tarefas intelectuais, usam da comunicação intensivamente, têm o seu tempo autogerenciado e frequentemente desafiam as estruturas sociais. O termo original "Tecnólogos do Conhecimento" foi cunhado por Peter Drucker, no livro *Management Challenges for de 21st Century* (Desafios do Gerenciamento no Século 21) (DRUCKER, 1999)

3 - Segundo Hall et. al. (2007) a forma como o termo "Engenheiro de Software" tem sido utilizado nos últimos 26 anos evoluiu consideravelmente. No entanto, neste artigo o termo "Engenheiro de Software" referir-se-á a analistas, desenvolvedores, designers e quaisquer outros papéis que compõem o campo da Engenharia de Software.

4 - Psicoterapia é a área da psicologia que trata de perturbações mentais e comportamentais. Psicometria é a área da psicologia que busca identificar padrões de personalidade e de comportamento (PRIBERAM).

Ano	Autor(es)	O que é motivação?
1959	Krench e Crutchfield	Um motivo é uma necessidade ou desejo acoplado com a intenção de atingir um objetivo apropriado.
1961	Young	Uma busca dos determinantes (todos os determinantes) da atividade humana e animal.
1964	Atkinson	Pode-se falar em uma teoria da motivação e significar uma concepção coerente dos determinantes contemporâneos da direção, do vigor e da persistência da ação.
1967	Hilgard e Atkinson	Entendemos por "motivo" algo que incita o organismo à ação ou que sustenta ou dá direção à ação quando o organismo foi ativado.
1977	Arkes e Garske	O estudo da motivação é a investigação das influências sobre a ativação, força e direção do comportamento.
1997	Rogers, Ludington e Graham	Motivação é um sentimento interno é um impulso que alguém tem de fazer alguma coisa.
2000	Lieury e Fenouillet	... a motivação é o conjunto de mecanismos biológicos e psicológicos que possibilitam o desencadear da ação, da orientação (para uma meta ou, ao contrário, para se afastar dela) e, enfim, da intensidade e da persistência: quanto mais motivada a pessoa está, mais persistente e maior é a atividade.
2001	Penna	é o conjunto de relações entre as operações de estimulação ou privação e as modificações observadas no comportamento que se processa após as citadas operações.
2004	Bzuneck	A motivação tem sido entendida ora como um fator psicológico, ou conjunto de fatores, ora como um processo. Existe um consenso generalizado entre os autores quanto à dinâmica desses fatores psicológicos ou do processo, em qualquer atividade humana. Eles levam a uma escolha, instigam, fazem iniciar um comportamento direcionado a um objetivo.

Tabela 1. Evolução do Conceito de Motivação (*Fonte: Elaborado a partir de Todorov e Moreira (2005)*)

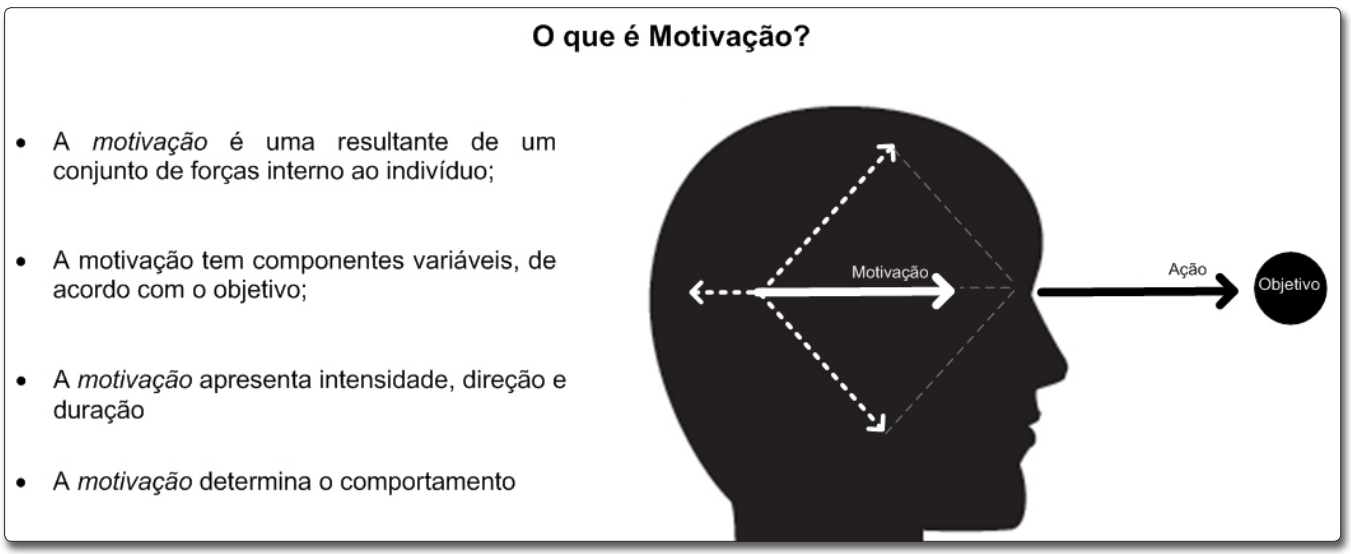


Figura 1. O que é Motivação? (*Fonte: Elaboração Própria*)

Faz-se importante não confundir o constructo *motivação* com outros constructos popularmente semelhantes como, por exemplo, o *condicionamento*. O que difere fundamentalmente a *motivação* do *condicionamento* é o seu caráter intrínseco. O *condicionamento* incentiva a ação a partir de recompensas extrínsecas e ignora a existência de sentimentos, atitudes e expectativas (**Figura 2**). Segundo Kjerulf (2006), apenas através da *motivação* é possível atingir um comportamento sustentável. No entanto, as empresas insistem em utilizar o *condicionamento* para tentar extrair um melhor desempenho dos seus colaboradores.

Entusiasmo, satisfação, conforto, alegria, necessidade, desejo, atitude, vontade, instinto e *fé* são exemplos de outras palavras frequentemente confundidas com *motivação*. Não é o intuito deste trabalho discutir a definição, a distinção, nem a relação delas com o termo *motivação* ou entre si. Estas palavras serão apresentadas apenas quando for conveniente para o entendimento de alguma teoria específica.

Segundo Pritchard e Ashwood (2008) e Sommerville (2007), o estudo da motivação está intimamente ligado ao gerenciamento em empresas. A função do gerente é direcionar o comportamento das pessoas a fim de alcançar um determinado objetivo. Mas para isto é essencial entender o funcionamento da motivação. Manter alta a motivação das pessoas não só colabora para que exerçam o seu melhor desempenho[5], mas também gera responsabilidade, confiança e satisfação nos colaboradores, que acabam comprometendo-se com os resultados organizacionais (BOWDITCH E BUONO, 2000).

As teorias de motivação são classificadas em três grupos. As **teorias de conteúdo estático** são aquelas que observam os

5 - Nenhuma das obras referenciadas cita experimentos que comprovem a correlação existente entre motivação e desempenho. O entendimento de que um indivíduo, quando está fortemente motivado, apresenta um desempenho superior ao que apresentaria se estivesse fracamente motivado, consta como senso comum na literatura.

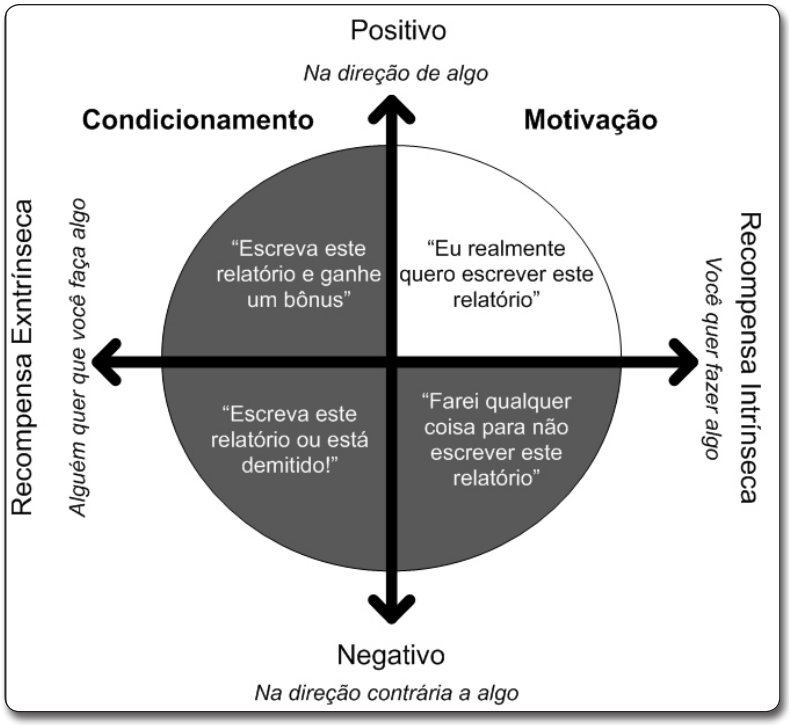


Figura 2. Condicionamento x Motivação (*Fonte: Adaptado de Kjerulf (2006)*)

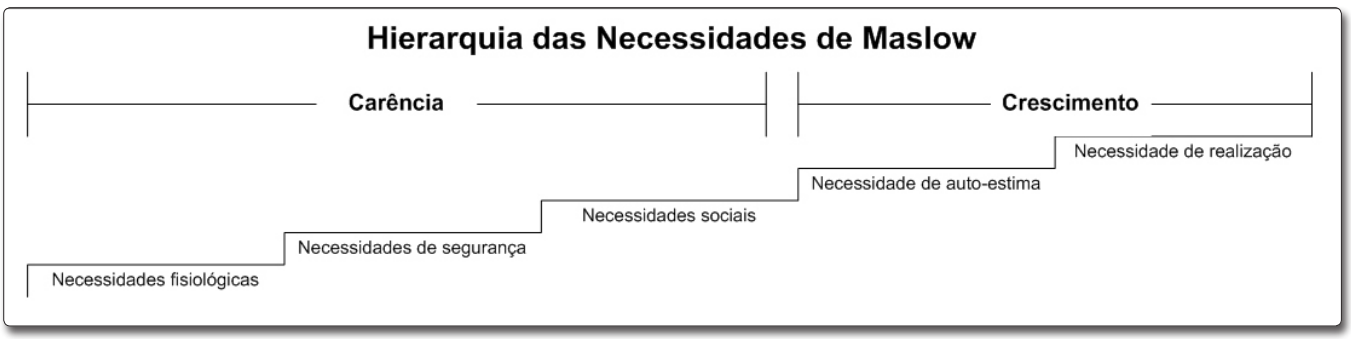


Figura 3. Hierarquia de Necessidades de Maslow[7] (*Fonte: Adaptado de Bowditch e Buono (2000)*)

fatores que influenciam na motivação humana. As **teorias de processos**, por sua vez, observam a dinâmica destes fatores. Já as **teorias baseadas no ambiente** enfocam genericamente o equilíbrio da motivação ao longo do tempo. Nenhuma delas, isoladamente, é suficiente para explicar o comportamento em qualquer situação. É exatamente por isso que existem tantas teorias (BOWDITCH e BUONO, 2000). A seguir serão explicadas algumas das teorias existentes.

Teoria da Hierarquia das Necessidades

O primeiro teórico a estabelecer uma teoria completa sobre motivação foi Abraham Maslow, em 1943, na obra *Motivation and Personality*[6]. A sua Teoria da Hierarquia das Necessidades acabou dando origem a diversas outras teorias de conteúdo estático existentes atualmente. Maslow iniciou seus estudos observando o comportamento de primatas e depois passou a estudar o comportamento de personalidades como Abraham Lincoln, Benjamin Franklin, Albert Einstein, Mahatma Gandhi, entre outros. Isto, de certa forma, explica a simplicidade com que ele trata o comportamento humano (BUENO, 2002; TODOROV E MOREIRA, 2005).

Esta teoria parte da premissa de que a motivação é determinada por um impulso genérico no sentido de satisfazer necessidades. Se um organismo está com sede, ele bebe, se está com fome, come, e assim por diante (BUENO, 2002). Então Maslow identificou uma hierarquia de necessidades (**Figura 3**). À medida que uma necessidade é satisfeita, a imediatamente superior passa a determinar o comportamento do indivíduo. Nos degraus

6 - MASLOW, Abraham H. Motivation and Personality. HarperCollins Publishers, 3ª. Ed 1987. ISBN: 0060419873.

7 - A obra original de Maslow apresenta a hierarquia de necessidades como uma pirâmide, pois, segundo ele, apenas uma minoria da sociedade (estimada arbitrariamente em 10%) seria capaz de atingir o nível superior durante a vida, enquanto a grande maioria ficaria estagnada nas bases da pirâmide. Porém, para facilitar o entendimento, optou-se por utilizar a ilustração da escada, pois dá uma noção mais adequada de sequenciamento entre as necessidades (BOWDITCH E BUONO, 2000).

iniciais estariam as necessidades da **carência**, que fazem com que o indivíduo se sinta saudável. As superiores são caracterizadas como necessidades de **crescimento** (SALGADO, 2005).

Segundo Maslow, as necessidades humanas seriam (ANDRADE, 2006):

- **Necessidades biológicas, ou fisiológicas:** são as necessidades inatas, como fome, sede, abrigo, reprodução, entre outras;
- **Necessidades de segurança:** refere-se à busca de proteção contra os perigos reais ou imaginários, relacionadas à estabilidade e sobrevivência;
- **Necessidades de afiliação, ou sociais:** são as necessidades de associação, aceitação por parte de um grupo, troca de amizade, afeto, amor;
- **Necessidade do ego, ou de auto-estima:** são relacionadas à forma como o indivíduo se auto-avalia. Envolve auto-apreço, autoconfiança, etc.;
- **Necessidade de realização pessoal:** são as necessidades mais elevadas. São o que levam o indivíduo a querer realizar todo o seu potencial.

A Teoria da Hierarquia das Necessidades, de Maslow, foi duramente criticada no curso da história[8]. Clayton Alderfer (apud. SALGADO, 2005), por exemplo, sugeriu uma simplificação para três necessidades, que denominou de teoria ERC[9]: existência, relacionamento e crescimento. Henry Murray (apud. SALGADO, 2005) identificou mais de 20 necessidades e sugeriu que entre elas não existiria hierarquia. David McLelland (apud. SALGADO, 2005), propôs que as necessidades poderiam variar de acordo com a cultura de origem do indivíduo. Mas mesmo com tantas críticas, a teoria de Maslow, ainda hoje é a mais reconhecida.

Mais tarde, administradores tentaram relacionar as necessidades humanas às necessidades de um colaborador dentro de uma empresa. Como Tenório (1997) e Maitland (2000) exemplificam, as necessidades fisiológicas seriam ligadas ao salário e benefícios. As necessidades de segurança estariam relacionadas à segurança no trabalho, planos de saúde, etc. As necessidades de afiliação estariam relacionadas ao trabalho em equipe e à amizade. As necessidades de auto-estima seriam satisfeitas pelo elogio ou por promoções. Por fim, a autorealização seria atingida quando se proporcionam trabalhos gratificantes (**Tabela 2**).

Necessidades de Maslow	Necessidades dos Colaboradores
Necessidades fisiológicas	Salário, benefícios, e condições físicas do ambiente.
Necessidades de segurança	Aposentadoria, plano de saúde, auxílio-educação.
Necessidades de afiliação	Participação em movimentos de classe.
Necessidade de auto-estima	Reconhecimento e desenvolvimento profissional.
Necessidade de realização pessoal	Identificação com o trabalho e com a organização.

Tabela 2. Hierarquia de Necessidades aplicada ao gerenciamento em empresas (Fonte: Adaptado de Tenório (1997))

Teoria da Expectativa

A Teoria da Expectativa[10], conhecida também como Teoria Contingencial, é a mais importante teoria de processo de motivação. Foi formulada pelo psicólogo canadense Victor Vroom, e publicada em 1964. Vroom não concordara com as teorias de conteúdo até então existentes, pois para ele o processo motivacional "não depende apenas dos objetivos individuais, mas também do contexto de trabalho em que o indivíduo está inserido" (BUENO, 2002). Por isso optou por uma abordagem inovadora sobre o assunto, definindo a motivação como "o processo de governança de escolhas dentre as formas possíveis de ação voluntária" (LATHAM, 2006).

Vroom, então, propôs-se a descrever a dinâmica da motivação. Segundo ele, a natureza do indivíduo é de agir em busca da satisfação. A satisfação, por sua vez, é obtida a partir da avaliação que o indivíduo dá ao resultado de alguma ação. E a ação, por fim, é determinada pela motivação (**Figura 4**). Dessa forma, a motivação não estaria atrelada a um conjunto fechado de "necessidades", mas seria uma combinação de fatores motivacionais presentes no ambiente, da percepção, da crença e da interpretação destes fatores pelo indivíduo. (GREEN, 1992; MAITLAND, 2000; BOWDITCH E BUONO, 2000;BUENO, 2002; Latham, 2006).

Então, para entender como se dá o processo de motivação, Vroom decompôs o que ele chamou de "Força Motivacional" em três vetores, nomeados Valia[11], Expectativa e Instrumentalidade. Sinteticamente, o primeiro vetor representa o valor que uma pessoa atribui a alguma recompensa; o segundo representa a crença de que a recompensa é possível; e o terceiro representa a relação que uma ação escolhida tem com a recompensa desejada. Vroom, então, assumiu a motivação como um processo dinâmico relativamente consciente[12]. A seguir, estes fatores são apresentados detalhadamente.

Valia (V)

Corresponde à importância que o indivíduo atribui ao resultado de alguma ação (SOTO E MARRAS, 2002). Representa as

8 - Segundo Oliveira (2002), inclusive o próprio Abraham Maslow publicou diversos trabalhos apontando que na Teoria da Hierarquia das Necessidades existia ainda "muito espaço para dúvidas" (sic).

9 - Do inglês ERG: existence, relationship e growth.

10 - Do inglês Expectancy Theory. Frequentemente ela é citada na literatura em língua portuguesa como Teoria da Expectância ou Teoria da Expectação. No entanto, nenhum dos dois verbetes ("Expectância" e "Expectação") são citados nos dicionários de língua portuguesa. Por isso, optou-se por utilizar a nomenclatura "Teoria da Expectativa".

11 - Do inglês Valency, cuja tradução ao pé da letra seria "Valência". No entanto, em português, a palavra valência é utilizada vulgarmente na literatura popular com um sentido de "sorte" ou "proteção". No campo da química, a valência de um átomo significa o poder de combinação de um elemento químico. Neste artigo, optou-se por utilizar a tradução "Valia", que em português significa "valor estimado" e comunica com fidelidade a idéia de Vroom. Esta liberdade de certa forma é permitida, pois na literatura especializada há uma absoluta desorganização, descuidado e desatenção com a tradução do termo, que pode ser encontrado em artigos científicos com os nomes de "Valência", "Valor", "Valoração", e outros.

12 - Em alguns casos a Teoria da Expectativa também é referenciada como Teoria VIE, acrônimo para Valia, Instrumentalidade e Expectativa (Soto e Marras, 2002).

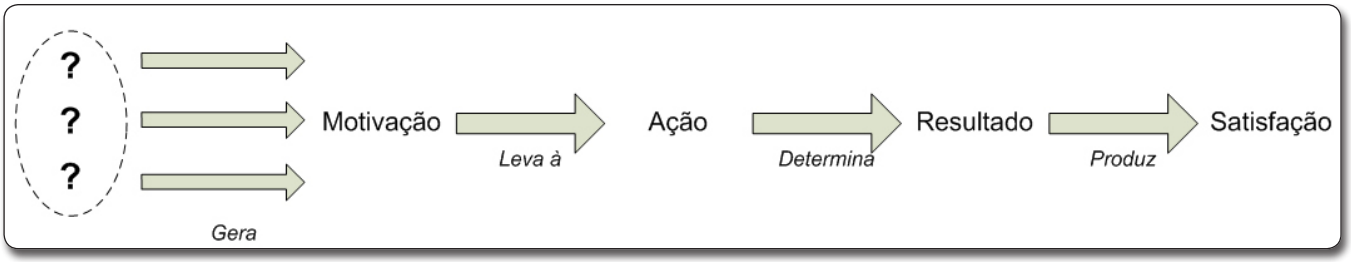


Figura 4. Dinâmica da motivação (Fonte: Adaptado de Maitland (2000))

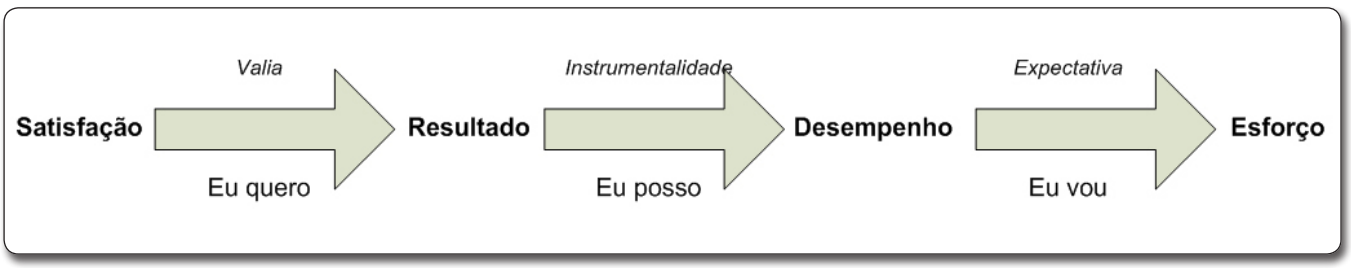


Figura 5. Processo de motivação através da Teoria da Expectativa (Fonte: Adaptado de Green (1992))

suas preferências (BORGES E ALVES FILHO, 2001). Um alto nível de valia indica um desejo forte por algo (MAITLAND, 2000). Quanto maior a valia, maiores são as chances de o indivíduo exercer um esforço na direção da ação que gerará o resultado esperado, ou seja, motivar-se (LATHAM, 2006). Isso explica porque um determinado sistema de recompensas implantado por uma empresa pode não motivar um grupo de colaboradores, mas ser eficaz para um grupo distinto (BUENO, 2002).

Dado a existência de um conjunto de *n* fatores que resultam de uma determinada ação, a valia (*V*) de um indivíduo para realizar esta ação é dada pela soma das valias atribuídas a cada um dos fatores (V_n), como representado em (1) (BORGES E ALVES FILHO, 2001).

$$V = \sum_{1}^{n} (V_i), \text{ onde } 0 \leq V_i \leq 1 \quad (1)$$

Expectativa (E)

DuBrin (2003) define a Expectativa (E) como sendo "o palpite subjetivo de uma pessoa sobre as chances de que um esforço adicional o levará a um desempenho ótimo". Em outras palavras, é a percepção que o indivíduo tem da sua própria capacidade realizar uma determinada ação (BUENO, 2002). Por exemplo, algumas pessoas podem não se motivar a andar de bicicleta porque duvidam que sejam capazes de controlar o guidão com as mãos, para manter o equilíbrio, enquanto estão tentando pedalar com os pés. "Assim, se esta pessoa não acredita que tenha habilidade para realizar uma tarefa, poderá nem tentar" (DuBRIN, 2003).

Da mesma forma como funciona a valia, dado um conjunto de *n* fatores de motivação, a expectativa (*E*) de um indivíduo para um determinado resultado é o somatório da sua expectativa (E_n) para cada um das ações que resultam no objeto desejado, vide equação (2) (BORGES E ALVES FILHO, 2001).

$$E = \sum_{1}^{n} (E_i), \text{ onde } 0 \leq E_i \leq 1 \quad (2)$$

Instrumentalidade (I)

Borges e Alves Filho (2001) interpretam a instrumentalidade como o grau percebido de relação entre uma ação e um resultado. É a probabilidade de que o esforço conduza a algum resultado (SOTO E MARRAS,2002). Assim para motivar-se a realizar alguma ação, o indivíduo tem que acreditar que é capaz de realizá-la; que o resultado produzido será o esperado; e que este resultado esperado é importante para si. Segundo Bueno, (2002), uma organização pode extrair o melhor do seu colaborador quando conseguir "mostrar que o caminho para a satisfação dos seus objetivos individuais está ao alcance do desempenho desejado".

A equação que determina a instrumentalidade (*I*) de um indivíduo é dada em (3).

$$I = \sum_{1}^{n} (I_i), \text{ onde } 0 \leq I_i \leq 1 \quad (3)$$

Força motivacional (Fm)

Para a obtenção da força motivacional, os três elementos são igualmente importantes. Quando os três elementos estão presentes, a motivação é máxima. Quando um dos elementos está ausente, a motivação é fraca. Quando os três elementos estão ausentes, a motivação é mínima. Desta forma, a força motivacional (Fm) é dada pela sentença em (4) (BORGES E ALVES FILHO, 2001; BUENO, 2002).

$$Fm = E \times I \times V, \text{ onde } 0 \leq Fm \leq 1 \quad (4)$$

Numa abordagem didática amigável, Green (1992) apresenta a Teoria da Expectativa, como ilustrado na **Figura 5**.

Em comparação às teorias existentes até então, a inovação de Vroom deu-se primeiro pela consideração da motivação como um processo consciente de escolha. Segundo, pela inclusão do fator de julgamento (consciente ou inconsciente) do próprio indivíduo sobre a sua capacidade, competência e importância dos resultados esperados. Para Bowditch e Buono, (2000) a Teoria da Expectativa é bastante útil para avaliar as opções com a qual uma pessoa é confrontada, pois a escolha que o indivíduo fará é uma função direta da valia, da expectativa e da instrumentalidade envolvidas.

Teoria da Motivação-Higiene

Frederick Herzberg, psicólogo, consultor e professor universitário estadunidense, desenvolveu uma teoria de conteúdo, conhecida como Teoria da Motivação-Higiene, ou Teoria dos Dois Fatores, ou ainda Teoria Bifatorial (BUENO, 2002). Divulgada em 1969, a teoria foi formulada a partir de uma pesquisa na qual se pedia a trabalhadores para descreverem situações em que se sentiam bem e mal no trabalho. Até então, os teóricos consideravam que o trabalho era uma atividade essencialmente desagradável, mas com a sua teoria, Herzberg quebrou este paradigma. (MAITLAND, 2000; SALGADO, 2005; LISBOA,2005)

Basicamente, Herzberg classificou os fatores que influenciam na motivação de um indivíduo em dois grupos: os **fatores motivacionais**, que são aqueles capazes de aumentar a motivação e dirigir o comportamento de um indivíduo; e os **fatores higiênicos** (ou não-satisfatórios), que não são capazes de motivar, porém são capazes de repelir a ação se não forem devidamente tratados. A presença exclusiva de fatores higiênicos não é capaz de motivar o indivíduo. Já a atuação exclusiva de fatores motivacionais não se reflete necessariamente num comportamento motivado. Estes fatores estão exemplificados na **Tabela 3**.

Fatores Higiênicos (Prevenção da Insatisfação)	Fatores Motivacionais
• Condições de trabalho	• O trabalho em si
• Pagamento	• Responsabilidade
• Segurança no trabalho	• Senso de realização
• Relações no trabalho	• Reconhecimento
• Práticas de supervisão e administração	• Perspectivas de evolução
• Política e administração da empresa	

Tabela 3. Fatores motivacionais e fatores higiênicos[13] (Fonte: Adaptado de Maitland (2000))

Discussão

Motivação é o conjunto de forças interno a um indivíduo que determina o seu comportamento. Este conjunto de forças apresenta direção, intensidade, duração e pode variar de acordo com o objetivo. A premissa de que pessoas motivadas atingem um melhor desempenho fez com que a *motivação* no trabalho tenha tornado-se um dos tópicos centrais nas ciências sociais. O estudo da *motivação* no trabalho atualmente dá bases para disciplinas como *comportamento organizacional* e *psicologia organizacional*.

Segundo Grohman (1999), cerca de 85% das empresas reconhecem termos como *Empowerement*, *Coaching*, e outros, porém os índices de utilização prática destas técnicas ainda são tímidos, pois giram em torno de apenas 16%. Todorov e Moreira (2005) e Bowditch e Buono (2000) explicam que isto acontece por que há uma confusão entre os diversos constructos relacionados à *motivação* e a diversidade de modelos e teorias existentes ainda dificulta a escolha e, consequentemente, a aplicação adequada destes modelos e teorias.

Chen & Kanfer (2006), descrevem que o processo de motivação de uma equipe é mais complexo do que simplesmente desenvolver a motivação individual em todos os componentes da equipe. O processo é semelhante ao de motivação individual, mas é possível distinguir os fatores que agem no âmbito da equipe (*Team-Level*) e no âmbito do indivíduo (*Individual-Level*). Um bom clima de liderança, por exemplo, é capaz de influenciar na motivação da equipe como um todo. Este aspecto também deve ser considerado ao trabalhar com motivação de equipes.

Por fim, diversas outras teorias de motivação não foram abordadas por não apresentarem uma importância fundamental para o entendimento deste artigo. Algumas delas seriam: Teoria de Motivação pelo Caminho-Meta, Teoria da Fixação de Metas, a Teoria do Condicionamento e Reforço Operantes, entre outras (Bowditch e Buono, 2000). É importante citar que isso não reduz a importância de qualquer uma destas outras teorias, no entanto as teorias aqui citadas decorrem como as mais citadas e reconhecidas em trabalhos científicos.

Equipes e Engenharia de Software

Batituci (2002) descreve o estudo do *trabalho em equipe*[14] como sendo "a complexa busca do equilíbrio entre as porções individual e coletiva do homem". Ao longo da história é possível citar nomes de personalidades excepcionais, que conquistaram o mundo através de seu carisma e/ou do seu trabalho individual. Alguns exemplos são Buda, Confúcio, Ramsés, Átila, Jesus Cristo, Napoleão, Gandhi, Hitler, entre outros. Mas nas organizações e empresas atuais, os indivíduos, setores e iniciativas, quando isolados, não conseguem sequer sobreviver. Isto retrata a importância do trabalho em equipe (BATITUCI, 2002).

13 - Alguns pesquisadores sugerem uma correspondência entre os fatores higiênicos da Teoria de Herzberg e as necessidades de carência, da Teoria de Maslow, bem como entre os fatores motivacionais de Herzberg e as necessidades de crescimento de Maslow. Nenhuma das referências consultadas comprova esta correspondência. Herzberg também não sugere interdependência nem hierarquia entre os fatores higiênicos e motivacionais identificados.

14 - Nos títulos em língua inglesa, o termo é referenciado como "Team". A tradução ao pé da letra seria "Time". No entanto, optou-se por utilizar a palavra "Equipe". Apesar de serem sinônimos, a palavra "Time" no português brasileiro tem uma conotação esportiva muito popular, enquanto que na língua portuguesa europeia a palavra "Time" nem sequer existe (PRIBERAM).

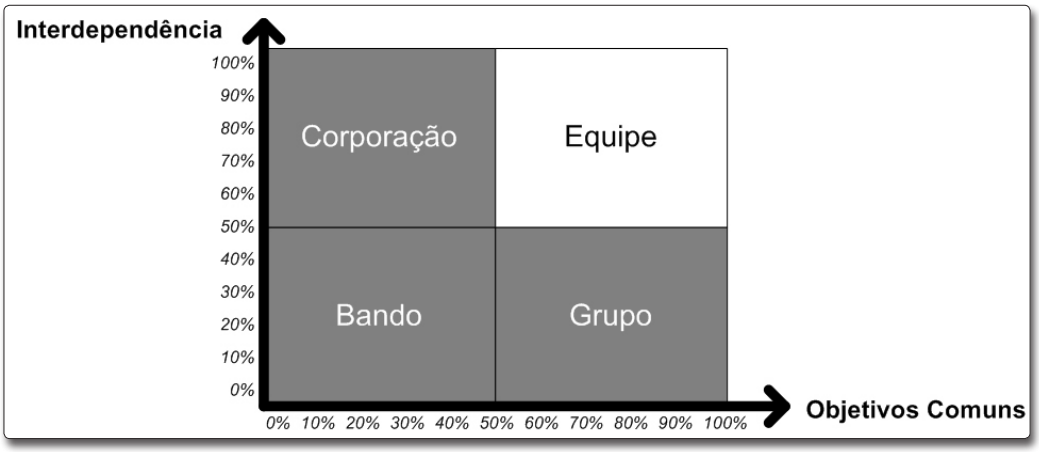


Figura 6. Configurações de agrupamentos de pessoas (Fonte: Adaptado de Batituci (2002))

Segundo Katzenbach e Smith (1993) a forma como o termo *"trabalho em equipe"* acabou tornando-se popular não corresponde exatamente ao seu real sentido. Mas o que uma *equipe* de fato é? Segundo Batituci (2002), uma equipe deve ser definida como "um conjunto de pessoas que trabalham de forma interdependente para um propósito comum". A **Figura 6** ilustra a diferença entre o conceito de *equipe* e outros conceitos que confundem os executivos. Diversos outros autores concordam com esta definição, como Katzenbach e Smith (1993), Scholtes, Joiner e Streibel (2003) e Harvard (2006).

Para Harrington-Mackin (1994), os maiores benefícios que o trabalho em equipe pode trazer para uma empresa são:

- Ambiente altamente motivado[15];
- Um melhor clima de trabalho;
- Propriedade e responsabilidade compartilhada;
- Respostas mais rápidas às mudanças tecnológicas;
- Distribuição equilibrada da carga de trabalho;
- Flexibilidade na atribuição de tarefas;
- Comprometimento comum com os valores e objetivos da organização;
- Abordagem pró-ativa para resolução de problemas, inovadora e efetiva;
- Alto nível de comunicação;
- Valor próprio bem desenvolvido nos componentes;
- As melhores decisões serão tomadas;
- Agilidade para alertar sobre problemas;
- Desenvolvimento conjunto das habilidades das pessoas.
- Por outro lado, trabalhar em equipe também tem seus riscos e custos (HARRINGTON-MACKIN, 1994):
- A equipe pode aparentar ser confusa, desordenada e fora de controle;
- Pode causar confusão com relação a papéis e funções;
- Requer um longo tempo para começar a produzir os resultados;
- Requer uma mudança de paradigma nas pessoas.

15 - Em suas próprias palavras, sem estabelecer necessariamente uma relação com as teorias de motivação vistas no tópico anterior.

Tipologia de Equipes

Os teóricos do assunto, em geral, alertam para o fato de que o trabalho em equipe deve ser implantado com cautela dentro de uma empresa. Principalmente os seguintes fatores devem ser observados para a escolha de um tipo de equipe (FAA HUMAN FACTORS):

- **Composição Funcional:** refere-se ao conjunto de papéis funcionais diferentes que devem ser agrupados para compor a equipe;
- **Propósito:** refere-se ao foco da equipe: resolver problemas, desenvolver produtos específicos ou executar atividades rotineiras;
- **Duração:** refere-se ao tempo que as pessoas permanecerão na equipe, e como será a composição dela em relação aos membros fixos e temporários.

Diversos autores apresentam categorizações de equipes, diferenciando-as pelo propósito (LARSON, 1989), pela composição funcional (HARRINGTON-MACKIN, 1994; HARVARD, 2006), pelo grau de autonomia (SWAMIDASS, 2002), pela composição de diversos destes critérios (SCHOLTES, JOINER E STREIBEL, 2003; Grazier) ou mesmo pela estabilidade dos membros (CLUTTERBUCK, 2007). A **Tabela 4** apresenta a classificação de Larson (1989), de acordo com o propósito da equipe.

Propósito	Característica Dominante	Ênfase
1. Resolução de Problemas	Confiança	Foco nos problemas
2. Criativo	Autonomia	Explorar possibilidades e alternativas
3. Tático	Direção	Clareza, papéis bem definidos, foco na tarefa.

Tabela 4. Diferentes tipos de equipes de acordo com o propósito (Fonte: Traduzido e adaptado de Larson (1989))

A classificação de Larson (1989) pode ser aprofundada como segue:

- ***Equipes táticas:*** São aquelas que têm foco na execução de tarefas técnicas claramente definidas, padronizadas e cujos resultados são facilmente mensuráveis. Estas equipes são geralmente rígidas com relação a seus prazos, qualidade, custos,

etc. As pessoas que compõem este tipo de equipe devem ser responsáveis, comprometidas, orientadas à ação e possuir senso de urgência. Para incentivar o desempenho deste tipo de equipe, as metas determinadas devem ser progressivamente desafiadoras, porém sempre alcançáveis. O trabalho cooperativo também é uma característica deste tipo de equipe;

• ***Equipes criativas:*** São aquelas que têm foco na criação de produtos/serviços novos, inovadores ou exclusivos. Na indústria, Larson (1989) cita exemplos de produtos como o do IBM PC e do projeto do avião Boeing 747. Estas equipes são compostas geralmente por pessoas inteligentes, independentes, pró-ativas, autoconfiantes, teimosas, analíticas e capazes de quebrar paradigmas para explorar alternativas que vão além das soluções tradicionais. Os indivíduos que compõem este tipo de equipe têm interesses pessoais que casam com os objetivos da equipe e, por isso, muitas vezes não relutam em trabalhar durante finais de semana;

• ***Equipes de resolução de problemas:*** São aquelas que têm foco na resolução de problemas críticos. Exemplos seriam: uma central de atendimento ao cliente, ou um gabinete presidencial – onde os problemas precisam ser resolvidos rapidamente. Este tipo de equipe trabalha constantemente sob pressão, por isso, demanda confiança mútua entre os componentes. As pessoas que compõem esta equipe devem ser inteligentes, íntegras, sociáveis, e possuírem alto grau de pensamento analítico, capazes de reduzir problemas complexos em tarefas objetivas, além de demandarem capacidade de concentração para levar as atividades até o fim.

Equipes na Engenharia de Software

O gerenciamento de pessoas na engenharia de software não é um tópico novo, nem tampouco inovador. Fred Brooks, no clássico *The Mythical Man-Month*[16], de 1975, trata o gerenciamento da produtividade em projetos de software com foco nas pessoas. Ele exemplifica o problema citando que "se um pedreiro constrói um muro em quatro dias, cem pedreiros não o construirão em 10 minutos". O raciocínio que parece absurdo no exemplo era o aplicado, até então, para resolver os problemas de prazo em projetos de software (KOSCIANSKI E SOARES, 2007). Ou seja, há mais variáveis por trás da produtividade do que apenas a quantidade de pessoas[17].

Uma década mais tarde, DeMarco e Lister publicaram *Peopleware: Productive Projects and Teams*[18]. Neste outro clássico sobre o gerenciamento de equipes de projetos de software, eles trazem à tona uma discussão sobre como os diversos itens, alguns dos quais apontados por Brooks, poderiam ser gerenciados para gerar produtividade para uma equipe. São considerados itens como a personalidade dos componentes, o ambiente de trabalho, a composição e a organização da equipe. Para eles, criar uma equipe eficiente é uma questão de *semear* (como na agricultura) e não de *construir* (como na engenharia) (DeMARCO E LISTER, 1999).

De acordo com Lettice e McCracken (2007), a quantidade de pesquisas relacionadas com o gerenciamento de equipes em projetos de software vem crescendo ao longo do tempo[19], e focam principalmente nos dois seguintes tópicos:

• Identificação dos fatores que caracterizam a eficácia e alto desempenho da equipe e o sucesso do projeto, a exemplo de Becker, Burns-Howel e Kyriakides (2000), Tarricone and Luca (2002), Trent (2003), Senior and Swailes (2004) e França *et al*. (2008);

• Como a combinação de diferentes perfis de personalidade pode ajudar a construir times mais produtivos, a exemplo de Bradley & Herbert (1997), Stevens (1998), Neuman, Wagner and Christiansen (1999), Capretz (2003), Fürst (2004), Karn and Cowling (2006) e França e da Silva (2007).

De acordo com dados do SEI[20], os recursos humanos constituem em média 70% dos custos de um projeto de engenharia de software. Por isso, é possível afirmar que diversos problemas que caracterizam o fracasso de um projeto, por exemplo, a alta quantidade de falhas em um produto de software, a variabilidade do escopo de um projeto de software, entre outros, estão relacionados com uma gestão de pessoas ineficaz nos projetos de software. Segundo França *et al*. (2008) grande parte destes problemas poderia ser evitada apenas adotando-se um gerenciamento adequado da equipe de desenvolvimento (**Figura 7**)[21].

Os modelos de processos e de qualidade de software demonstram certo cuidado com a administração de equipes (**Tabela 5**). O UP[22] sugere as características pessoais adequadas para cada papel funcional de acordo com as suas responsabilidades (KROLL E KRUTCHEN, 2003). O SEI também desenvolveu processos especificamente para equipes de software: o *People* CMM (ou p-CMM) (SEI, 2001) e o *Team Software Process (SEI, 2000)*. O PMBOK

16 - BROOKS, Fred. The Mythical Man-Month. Addison-Wesley. 1975. ISBN: 201-00650-2

17 - Brooks cunhou o termo "Lei de Brooks", que determina que "adicionar força-humana a um projeto de software atrasado o tornará ainda mais atrasado" (Koscianski e Soares, 2007).

18 - DeMARCO, Tom; LISTER, Timothy. Peopleware: Productive Projects and Teams. Dorset House Pub. 1987. ISBN 0932633439. Peopleware é uma palavra que não possui tradução para a língua portuguesa. Aparentemente é uma alusão à software e hardware, para indicar que por trás da engenharia há também o fator humano.

19 - De 1995 a 2006, a quantidade de artigos sobre o assunto mais do que duplicou. Neste período, os Estados Unidos se manteve sempre como o maior produtor de conhecimento na área.

20 - O SEI – Software Engineering Institute (Instituto de Engenharia de Software), pertencente à Carnegie Melon University, ficou famoso pela publicação do modelo de capacidade e maturidade integrado (CMMi), o modelo de qualidade mais amplamente respeitado na engenharia de software.

21 - França et. al. (2008) desenvolveram uma pesquisa de campo qualitativa a fim de levantar suposições entre práticas de gerenciamento de equipes de software e o sucesso/fracasso de projetos. A Figura 7 foi construída a partir de dados de entrevistas realizadas com seis gerentes de projetos de empresas de desenvolvimento de software em Recife-PE.

22 - UP –Unified Process (ou Processo Unificado), como passou a ser conhecido o antigo RUP (Rational Unified Process), após a aquisição da Rational pela IBM.

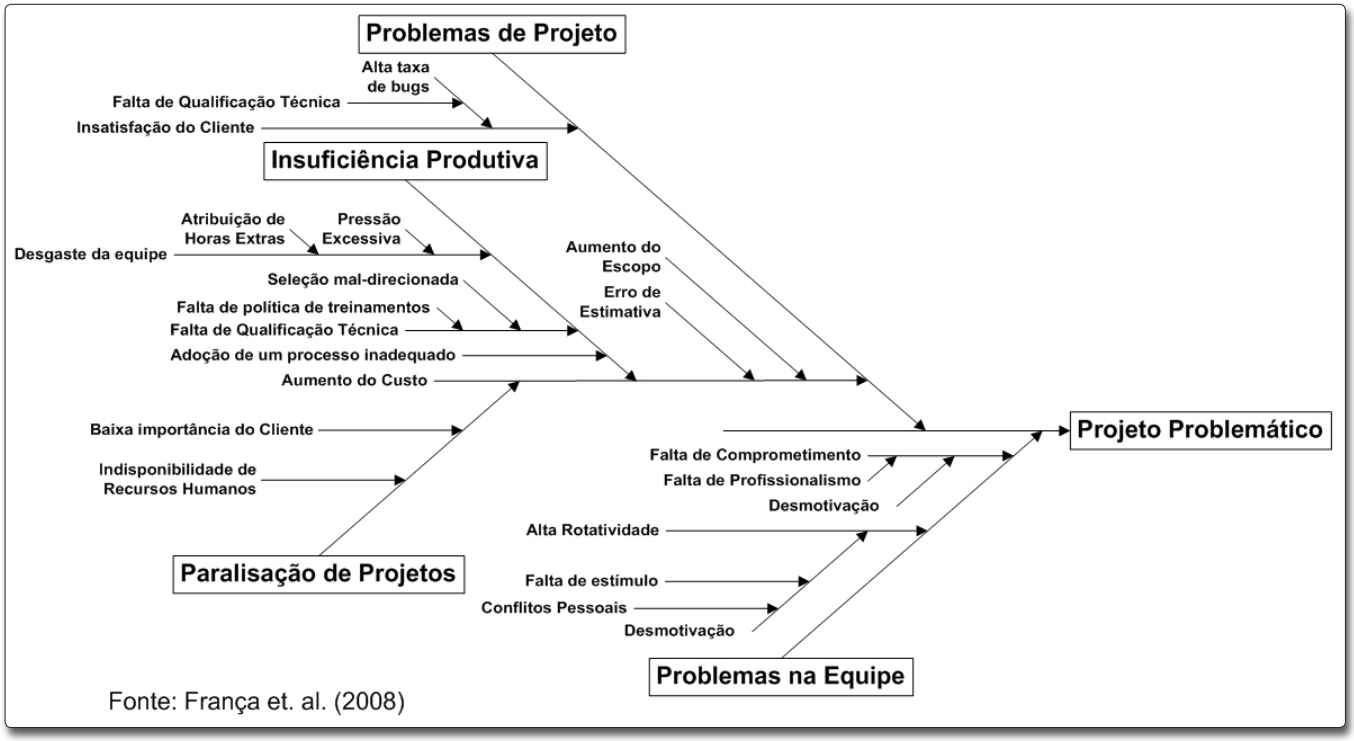


Figura 7. Fatores pessoais que influenciam no fracasso de um projeto

Modelo de Processos	Características
Unified Process (Processo Unificado)	Trata-se de um modelo de processo para desenvolvimento de software. Determina fases, iterações, disciplinas, procedimentos, atividades, tarefas, artefatos e responsabilidades (papéis). Para cada papel funcional definido, o UP sugere um conjunto de características pessoais, por exemplo: o gerente do projeto deve ser comunicativo, o arquiteto deve ser criativo, etc.
Team Software Process	O TSP é um modelo de processos para desenvolvimento de software em equipe, desenvolvido baseado nas características de efetividade de equipes (vide Seção 2.2.3). O TSP determina procedimentos para construção e gerenciamento de equipes (planejamento de metas, distribuição de papéis, controle de mudanças, custos, comunicação e qualidade, etc.).
People CMM	O p-CMM é um modelo de implementação gradativa de boas práticas de gerenciamento de equipes. Classifica um conjunto de práticas de gerenciamento de pessoas em cinco níveis de maturidade: gerenciamento inconsistente (nível inicial); gerenciamento de pessoas (nível gerenciado); gerenciamento de competências pessoais (nível definido); gerenciamento de capacidades (nível previsível) e gerenciamento de mudanças (nível otimizado).
PMBOK	É um conjunto de boas práticas de gerenciamento de projetos. Não é específico para projetos de software, mas é amplamente utilizado na engenharia de software. Define que um projeto deve ter os seguintes processos: planejamento de recursos humanos; contratação e mobilização de equipes; desenvolvimento da equipe do projeto e gerência do desempenho da equipe. Descreve em detalhes como cada um destes processos deve ser executado.

Tabela 5. Gerenciamento de equipes na Engenharia de Software (Fonte: Adaptado de Kroll e Krutchen (2003), SEI (2000), SEI (2001) e PMI (2005))

do PMI[23], largamente utilizado na engenharia de software, define a gerência de recursos humanos como uma área-chave para o sucesso do projeto (PMI, 2005).

Critérios de efetividade para equipes de software

Como citado anteriormente, apenas entender o conceito de trabalho em equipe não é suficiente para garantir um impacto significativo na produtividade da empresa. Pesquisadores buscam identificar exatamente os fatores que influenciam na efetividade de uma equipe de software (BECKER, 2000; TARRICONE E LUCA 2002; TRENT; 2003; SENIOR E SWAILES; 2004; FRANÇA *et al*., 2008). Segundo Katzenbach e Smith (1993) a efetividade é o que leva uma equipe a transpassar a sua barreira de limite de desempenho e a atingir o estado que ele denominou de "equipes de alto desempenho" (**Figura 8**).

Becker, Burns-Howel e Kyriakides (2000), em um trabalho específico com equipes de engenharia de software, condensaram os diversos fatores de efetividade em dezessete grupos, exibidos na **Tabela 6**. Eles não sugeriram nenhuma ordenação específica ou relação entre estes fatores. No entanto, a *motivação* como explicada nos tópicos anteriores, é claramente influenciada por diversos dos outros fatores presentes nesta lista. Apesar de isso gerar certa confusão, acaba por estabelecer um vínculo bem apropriado entre as teorias de motivação e administração da efetividade de equipes abordadas neste artigo.

23 - PMI – Project Management Institute (Instituto de Gerenciamento de Projetos) é a maior associação de profissionais de gerenciamento de projetos do mundo. Está presente em mais de 160 países e conta com mais de 240.000 registros de associados (ANSI).

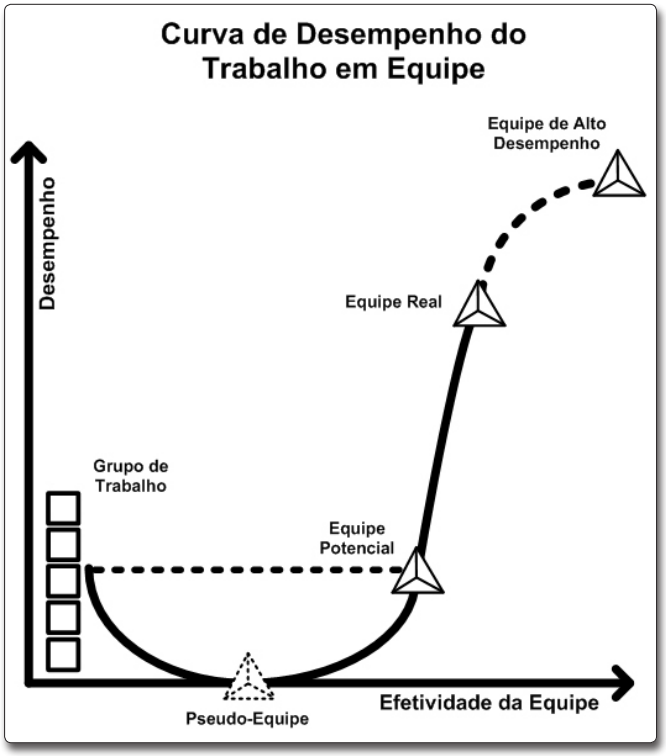


Figura 8. Curva de desempenho do trabalho em equipe (Fonte: Traduzido de Katzenbach e Smith (1993))

Fatores de efetividade para equipes de software	
Propósito e objetivos claros	Gerenciamento de conflitos
Comunicação aberta	Aprendizagem
Identidade e Abordagem de trabalho	Diversão
Papéis bem definidos	Qualidade e desempenho
Flexibilidade	Empowerement
Individualidade	Confiança e responsabilidade mútua
Gerenciamento do relacionamento externo	Recompensas financeiras apropriadas
Comprometimento	Motivação[24]
Heterogeneidade de habilidades	

Tabela 6. Fatores de efetividade para equipes de software (Fonte: Elaborado a partir de Becker, Burns-Howel e Kyriakides (2000))

O Estudo da Motivação em Equipes de Software

Como tratado nos tópicos anteriores, a *motivação* é um tema trabalhado nas empresas como uma ferramenta indispensável para atingir o diferencial competitivo. No entanto, poucas são as pesquisas que abordam a *motivação* como uma alternativa para prover produtividade em equipes de software (FREITAS E BELCHIOR, 2006; BEECHAM *et al*., 2007; HALL *et al*., 2007; FRANÇA *et al*., 2008; entre outros). Mas a escassez de estudos nesta área pode ser justificada tanto pela dificuldade de mensuração dos aspectos humanos e sociais envolvidos, quanto pela a dificuldade de se obter resultados práticos nesta área.

Freitas e Belchior (2006) utilizaram como base as práticas do p-CMM para avaliar quais delas seriam as mais e menos motivadoras para engenheiros de software, e como estas práticas eram tratadas em empresas de desenvolvimento[25]. A pesquisa foi realizada utilizando uma simplificação da Teoria da Expectativa para mensurar a motivação[26] dos profissionais. Mesmo tendo adotado uma abordagem indutiva, eles chegaram a um resultado curioso: as práticas motivacionais mais valorizadas pelos profissionais eram exatamente as menos utilizadas pelas empresas, enquanto as mais utilizadas pelas empresas foram as de menor valor.

Este resultado tem duas interpretações possíveis: a primeira é a de que as empresas estão de fato valorizando as práticas erradas em relação à motivação dos seus colaboradores. A segunda é que os profissionais têm a tendência de supervalorizar aquilo que não possuem, enquanto por outro lado sub-valorizam o que já possuem. Esta segunda interpretação, inclusive, faria mais sentido do ponto de vista das teorias de conteúdo, que predizem que as necessidades não-satisfeitas determinam o comportamento. No entanto, por se tratar de uma pesquisa qualitativa com amostra reduzida, nada se pode concluir, de fato, a partir dela.

Outra equipe, de duas universidades do Reino Unido[27], adotou uma abordagem diferenciada para tratar a motivação na engenharia de software. O seu objetivo foi desenvolver um modelo conceitual de motivação específico para engenheiros de software[28]. Para tanto, eles realizaram uma revisão sistemática da literatura de 20 anos sobre *motivação* aplicada à engenharia de software, envolvendo mais de 2.000 documentos disponíveis na internet em portais confiáveis, a fim de responder as seguintes cinco questões (BEECHAM *et al*., 2007; BADOO *et al*., 2007; SHARP *et al*., 2007; HALL *et al*., 2007):

- Quais são as características dos engenheiros de software?
- O que (des)motiva engenheiros de software a serem mais/menos produtivos?
- Quais são os benefícios de ter uma equipe de engenheiros motivada?

24 - O fator motivação, ao ser citado nesta, ajuda a estabelecer o vínculo apropriado entre todas as teorias de motivação e gerenciamento de equipes descritas neste artigo.

25 - A pesquisa envolveu apenas quinze engenheiros de software, com em média cinco anos de experiência, de três empresas do ramo de desenvolvimento de software, em Fortaleza.

26 - A simplificação sugeria que os respondentes deveriam atribuir um valor de 0 a 5 para representar o seu grau de motivação real, e outro para a sua motivação ideal. Motivação ideal seria o julgamento da importância dada à prática. A motivação real correspondeu à aplicação da prática pela empresa.

27 - School of Computer Science, University of Hertfordshire, College Lane Campus, Hatfield, Hertfordshire e Department of Computing, Faculty of Mathematics and Computing, The Open University, Walton Hall, Milton Keynes.

28 - A importância de desenvolver um modelo de motivação é que este ajuda a entender a sistemática da realidade (Badoo et al., 2007).

• Quais aspectos intrínsecos à engenharia de software (des) motivam os engenheiros de software?
• Quais são os modelos de motivação existentes na engenharia de software?

Com os resultados obtidos, eles elencaram um conjunto de fatores, anteriormente estudados por outros pesquisadores, que são capazes de motivar ou desmotivar engenheiros de software. Estes fatores estão expostos na **Tabela 7**. A identificação e a análise destes fatores motivacionais e o reconhecimento dos modelos já existentes permitiram a elaboração de um novo modelo de motivação. Os principais benefícios de se ter uma equipe motivada, segundo eles, seriam basicamente: retenção da equipe (combate à rotatividade), entrega do produto no prazo e aumento da produtividade (BEECHAM *et al*., 2008).

A **Figura 9** exibe as dimensões de atuação dos fatores motivacionais (SHARP *et al*., 2007). Os fatores individuais são aqueles que dependem unicamente de aspectos pessoais (como personalidade, cultura, preferências, etc.). Os fatores da equipe envolvem: o bom relacionamento entre os colegas, confiança, respeito, comunicação, entre outros. Os fatores da empresa envolvem a disponibilidade de recursos e os outros fatores controlados pela organização. E os fatores globais envolvem aqueles externos à organização (por exemplo, o reconhecimento como uma empresa de sucesso, por exemplo) (SHARP *et al*., 2007).

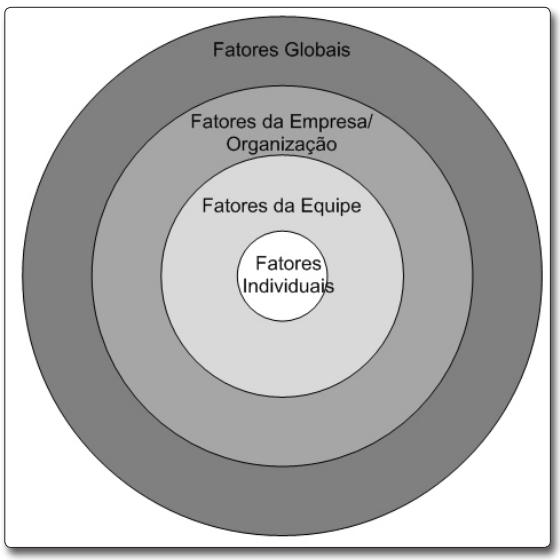


Figura 9. Dimensões dos fatores motivacionais (Fonte: Adaptado de Sharp et al. (2007))

Motivadores[29]	Desmotivadores
• Benefícios, recompensas e incentivos financeiros; • Desenvolvimento pessoal (oportunidades de treinamento, especialização, etc.); • Variedade de trabalhos (fazer um largo uso das suas habilidades); • Perspectiva de carreira (promoções ou planos de carreira); • Empowerement (onde responsabilidades são atribuídas às pessoas, ao invés de tarefas); • Bom gerenciamento (boa comunicação e gerente experiente); • Sensação de pertencimento (relações amigáveis com os colegas de trabalho); • Balanceamento do trabalho com a vida pessoal; • Trabalhar em empresas de sucesso (estabilidade financeira); • Participação, envolvimento, trabalho em equipe; • Receber Feedback (emitir informações e opiniões sobre o desempenho do colaborador); • Reconhecimento pessoal (não necessariamente financeiro); • Equidade (tratamento igualitário da equipe por parte da gerência); • Confiança/Respeito; • Trabalho desafiador; • Segurança no trabalho/Ambiente estável; • Identificação com a tarefa (interesses pessoas no trabalho); • Autonomia (liberdade para gerenciar suas próprias tarefas); • Condições de trabalho apropriadas; • Significado do trabalho (fazer contribuições significativas para a vida de outras pessoas); • Recursos suficientes; • Resolução de problemas; • Mudança constante; • Experimentar coisas novas; • Aplicação de boas práticas de engenharia de software; • Participação de todo o ciclo de vida de um projeto. • Exercitar a criatividade	• Risco; • Estresse; • Inequidade (reconhecimento baseado em fatores subjetivos); • Terceirização do trabalho mais interessante; • Sistema de recompensas injusto; • Falta de oportunidades de crescimento • Falta de identificação com o trabalho • Falta de comunicação; • Salário abaixo do mercado; • Metas e objetivos não-realistas, prazos mentirosos; • Mau relacionamento com usuários ou colegas de trabalho • Ambiente de trabalho pobre, falta de investimentos e recursos; • Gerenciamento ruim, reuniões que são perda de tempo; • Má qualidade do produto; • Falta de influência nas tomadas de decisão.

Tabela 7. Fatores que motivam e desmotivam engenheiros de software (Fonte: Traduzido e adaptado de Beecham et al. (2007))

29 - Aqui foram agrupados os fatores motivacionais do engenheiro de software e os fatores motivacionais intrínsecos à engenharia de software.

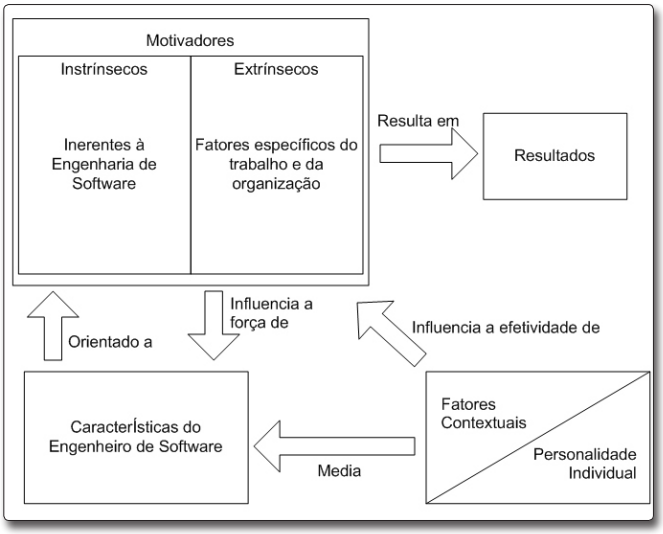


Figura 10. Modelo de motivação de engenheiros de software (*Fonte: Traduzido de Sharp et al. (2007)*)

Por fim, o modelo de motivação elaborado pelos britânicos ainda considerou um conjunto de outros modelos de motivação previamente existentes: o *Job Characteristics Model*, modelos de liderança, modelos de motivação para o desenvolvimento *Open Source*, inclusive modelos desenhados baseando-se nas Teoria da Expectativa, além de outros. Para completar o modelo, eles adicionaram ainda os elementos de controle: que são relacionados à personalidade e ao contexto. Estes elementos mediam as características dos engenheiros e influenciam a efetividade dos fatores motivacionais (**Figura 10**) (SHARP *et al*., 2007).

Apesar do cuidado na sua elaboração e do seu rigor científico o modelo, denominado MOCC[30] pelos ingleses, ainda carece de muitos esclarecimentos relacionados principalmente à sua aplicação prática. Ele é um modelo criado recentemente, por isso, é possível acreditar que uma evolução futura possa povoá-lo de técnicas e práticas de administração, a fim de ajudar os gerentes de projeto a desenvolverem engenheiros motivados e com melhor desempenho. No geral, o seu processo de construção já trouxe importantes evoluções para o campo de estudo da *motivação* aplicada ao gerenciamento de equipes de software.

O fato de ser um modelo focado essencialmente na motivação individual acrescenta algumas fraquezas ao MOCC. Primeiro porque, dada uma equipe com propósito específico, ele não orienta os gerentes sobre como motivar os seus subordinados. Ele atua unicamente explicando o porquê dos colaboradores estarem motivados ou desmotivados. E segundo porque, de acordo com Chen e Kanfer (2006), a motivação de equipes não deve ser tratada unicamente como a resultante da motivação dos seus membros. Devem ser considerados fatores que motivam/desmotivam a equipe como uma entidade distinta.

Considerações Finais

O propósito deste artigo foi reunir as principais teorias, conceitos e constructos sobre motivação e gerenciamento de equipes em projetos de software. Discutiu-se o que se entende por motivação, por que é importante manter os profissionais motivados dentro de uma empresa, porque acreditar que equipes motivadas conduzem à produtividade e foram apresentadas algumas pesquisas sobre como motivar engenheiros de software.

No primeiro tópico foram apresentados os significados atribuídos ao termo *motivação*. A motivação atrai a atenção de setores como psicologia, administração e sociologia. Mas a grande quantidade de pesquisas e a falta de coordenação entre as diversas áreas acabam gerando uma enorme confusão. Esta confusão foi esclarecida através da apresentação das principais teorias de motivação: a Teoria da Hierarquia das Necessidades (de Maslow), a Teoria da Motivação-Higiene (de Hezberg) e a Teoria da Expectativa (de Vroom).

Em seguida, foi apresentado o significado do termo *trabalho em equipe*, seus benefícios reais para uma empresa, e as diferentes classificações existentes de equipes. Esta Seção trouxe um histórico sobre o estudo do trabalho em equipe na engenharia de software e apresentou um conjunto de critérios importantes para gerar efetividade numa equipe de software. A definição do constructo de *efetividade* é um ponto que merece especial atenção neste tópico, pois apenas através da efetividade, é possível conduzir equipes ao estado de "equipe de alto desempenho".

Por fim, foram apresentadas as pesquisas mais recentes sobre a administração da motivação como ferramenta para desenvolvimento de equipes produtivas na engenharia de software. Apesar da escassez de pesquisas neste tópico, foi possível referenciar importantes resultados como, por exemplo, a elaboração do modelo MOCC, discutindo inclusive os méritos das suas principais contribuições e eventuais limitações. ●

30 - MOCC é acrônimo de "Motivators, Outcomes, Characteristics, Context".

Referências

• ANDRADE, Roberto L. Os Aspectos Não-Financeiros da Motivação: Estudo de caso da Líder Comércio de Lubrificantes LTDA. Dissertação apresentada à Escola de Administração da Universidade Federal da Bahia para o curso de People Management Especialization, 2005/01.

• ANSI. PMI: Project Management Institute, Disponível em: <http://webstore.ansi.org/FindStandards.aspx?Action=displaydept&DeptID=3158&Acro=PMI&DpName=Proje ct%20Management%20Institute>. Acessado em: 28 de Dez de 2008.

• BADOO, Nathan et al. Motivating Software Engineers: A theoretically reflective model. Paper supported by the UK's Engineering & Physical Sciences Research Council. UK, 2007.

• BATITUCCI, Márcio D. Equipes 100%: o Novo Modelo do Trabalho Cooperativo no 3º Milênio. São Paulo: Pearson Education do Brasil, 2002. ISBN: 8534614407.

• BECKER, Melanie; BURNS-HOWEL, James; KYRIAKIDES, John. IS Team Effectiveness Factors and Performance. Empirical Research Reports, University of Cape Town, Faculty of Comerce, Information Systems Research. 2000.

• BEECHAM, Sarah et al. Motivation in Software Engineering: A systematic literature review. Information and Software Technology: Elsevier, v. 50, n. 860-878, 2007.

• _____. The Motivation of Software Engineers: Developing a Rigorous and Usable Model. School of Computer Science. Faculty of Engineering and Information Sciences. University of Hertfordshire . Technical Report No: 458, 2007.

• Belbin, M.R. (1981) Management Teams: Why they succeed or Fail. Butterworth-Heinemann Ltd.

• BORGES, Livia de O.; ALVES FILHO, Antônio. A mensuração da motivação e do significado do trabalho. Estudos de Psicologia 2001, Vol 6, Num. 2, pp.177-194.

• BOWDITCH, James L.; BUONO, Anthony F. Elementos de Comportamento Organizacional. Cengage Learning Editores, 2000. ISBN 8522101426.

• BRADLEY, John H.; HERBERT, Frederic J. The effect of personality type on team performance, Journal of Management Development, Vol. 16, No. 5, 1997, pp. 337-353, MCB University Press.

• BUENO, Marcos. As Teorias de Motivação Humana e sua Contribuição para a Empresa Humanizada: um tributo a Abraham Maslow. Revista do Centro de Ensino Superior de Catalão - CESUC - Ano IV – 2002 – nº 06 - 1º Semestre.

• CAPRETZ, Luiz F. Personality types in software engineering. Int. J. Human-Computer Studies 58, 2003. 207-214.

• CHEN, Gilad; KANFER, Ruth. Toward A Systems Theory Of Motivated Behavior In Work Teams. Research in Organizational Behavior: Elsevier, 2006 Vol. 27, 223–267.

• CLUTTERBUCK, David. Coaching the Team at Work. Nicholas Brealey Publishing, 2007. ISBN: 1904838081.

• DeMARCO, Thom; LISTER, Timothy. Peopleware: Productive Projects and Teams. New York: Dorset House Publishing Co, ed. 2, 1999.

• DUBRIN, Andrew J. Fundamentos do comportamento organizacional. Cengage Learning Editores, 2003. ISBN: 8522103321.

• DRUCKER, Peter F. Management Challenges for the 21st Century. Butterworth-Heinemann, 1999.

• FAA HUMAN FACTORS. Types of Teams. Team Performance Module. Disponível em: <http://www.hf.faa.gov/webtraining/TeamPerform/Team012.htm>. Acessado em: 28 de Dez de 2008.

• FERNANDES, Aguinaldo A.; TEIXEIRA, Descartes de S. Fábrica de Software: implantação, gestão e operações. São Paulo: Atlas, 2004. ISBN: 8522436908.

• FRANÇA, A. César C. et al. A Qualitative Research on Software Projects Team Building. In: CONTECSI – 5ª International Conference on Technology and Information Systems, 2008, São Paulo.

• FRANÇA, A. César F.; da SILVA, Fabio Q. B. Um estudo sobre Relações entre Papéis Funcionais do RUP e o Comportamento Pessoal no Trabalho em Equipe em Fábricas de Software. In: III Workshop Um Olhar Sociotécnico sobre a Engenharia de Software – WOSES, 2007, pp25-36.

• FREITAS, Sergiana; BELCHIOR, Arnaldo. Análise de Aspectos Motivacionais que podem Influenciar Atores no Processo de Software. In: II Workshop Um Olhar Sociotécnico sobre a Engenharia de Software – WOSES, 2006. Disponível em: <http://www.cos.ufrj.br/~handrade/woses/woses2006/pdfs/09-Artigo09WOSES-2006.pdf>. Acessado em: 28 de Dez de 2008.

• FÜRST, Sandra. Determining the Relationships Between Team Performance, Soft skills and Personality Types in HighTech Industry. Tese apresentada à Faculty of Business School, The human side of leading high-tech teams. 2004.

• GOLEMBIEWSKI, Robert T. Handbook of Organizational Behavior. Marcel Dekker: ed. 2, 2000, ISBN: 0824703936.

• GRAZIER, Peter. Team Basics: Types of Team. Team Problem Solver. Disponível em: < http://www.teambuildinginc.com/tps/020a.htm>. Acessado em: 28 de Dez de 2008.

• GREEN, Thad B. Performance and Motivation Strategies for Today's Workforce: A Guide to Expectancy Theory Applications. Greenwood Publishing Group, 1992. ISBN: 0899306780.

• GROHMAN, Márcia Z. Novas Abordagens de Motivação no Trabalho: Identificação do Nível de Conhecimento e Utilização. Universidade Federal de Santa Maria, Santa Maria - RS. Projeto do Posto e de Sistema de Organização do Trabalho - Gerência de Pessoas, 1999.

• HALL, Tracy et al. Motivation Theory in Software Engineering. UK. ScholarOne Manuscript Central. 2007.

• HARRINGTON-MACKIN, Deborah. The Team Building Tool Kit: Tips, Tactics, and Rules for Effective Workplace Teams. AMACOM Div American Mgmt Assn, 1994. ISBN: 0814478263.

• HARVARD BUSINESS PRESS. Leading Teams: Expert Solutions to Everyday Challenges. Harvard Business Press, 2006. ISBN: 1422101843.

• HUGHES, Martin; Managing Cultural Diversity. Dissertação apresentada à Maastricht School Of Management para o curso de Mba Outreach Program In Egypt. Cairo: 2008.

• JAGGS, Louise. 75 per cent of IT professionals are considering changing jobs claims new survey. Skillsoft, 2006. Disponível em: <http://www.skillsoft.com/EMEA/news/15-Aug-06.asp >. Acessado em: 03 de Dez de 2008.

• _____. IT pros more likely to suffer from stress, says new survey. Skillsoft, 2006. Disponível em: <http://www.skillsoft.com/EMEA/news/19-May-06.asp>. Acessado em: 03 de Dez de 2008.

• KARN, John; COWLING, Tony. A Follow up Study of the Effect of Personality on the Performance of Software Engineering Teams. Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE'06), Rio de Janeiro, Brazil, 2006 pp. 232-241,.

• KATZENBACH, Jon R.; SMITH Douglas K. The Wisdom of Teams: Creating the High-performance Organization. Harvard Business Press, 1993. ISBN: 0875843670.

• KJERULF, Alexander. Why "Motivation by Pizza" Doesn't Work. Chief Hapiness Office. Disponível em: < http://positivesharing.com/2006/12/why-motivation-by-pizza-doesnt-work/>. Acessado em: 28 de Dez de 2008.

• KOSCIANSKI, André; SOARES, Michel dos S. Qualidade de Software. Novatec, Ed. 2, 2007.

• KROLL, Per; KRUCHTEN, Philippe; The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP. Addison-Wesley, 2003. ISBN: 0321166094.

• LARSON, Carl E. Teamwork: What Must Go Right, what Can Go Wrong. Sage Publications, 1989. ISBN: 0803932901.

• LATHAM, Gary P.. Work Motivation: History, Theory, Research, and Practice. Foundations For Organizational Science. Sage Publications, 2006. ISBN: 0761920188.

Referências

• LETTICE, Fiona; McCRACKEN, Martin. Team performance management: a review and look forward. Team Performance Management Vol. 13 No. 5/6, 2007 pp. 148-159. Emerald Group Publishing Limited 1352-7592. DOI 10.1108/13527590710831855.

• LISBOA, Teresinha C. Gestão em Recursos Humanos. In: Gestão Estratégica para a Liderança em Empresas de Serviço Privadas e Públicas. Nobel, 2005. ISBN: 8521313063.

• LUCENA, Evisson F. Um Estudo sobre Critérios de Formação de Equipes e seus Impactos no Desempenho dos Projetos de Software. Dissertação de mestrado apresentada ao Centro de Informática da UFPE, 2009.

• MAITLAND, Iain. Como motivar pessoas; tradução Pedro Marcelo Sá de Oliveira e Giorgio Cappelli — São Paulo: Nobel, 2000. Título original: Motivating people (1995). ISBN 85-213-0968-6.

• MARCONI, Marina de A.; LAKATOS, Eva M. Metodologia Científica. 3ª ed. Atlas, 2004. ISBN: 8522437998.

• NEUMAN, George A.; WAGNER, Stephen H.; CHRISTIANSEN, Neil D. The Relationship Between Work-Team Personality Composition and the Job Performance of Teams. Group & Organization Management, Vol. 24 No. 1, March 1999 28-45.

• OLIVEIRA, Silvio L. de. Sociologia das organizações: uma análise do homem e das empresas no ambiente competitivo. Cengage Learning Editores, Ed 3, 2002. ISBN 8522101760.

• OREG, Shaul; NOV, Oded. Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. Computer in Human Behavior: Elsevier, v. 24, n. 2055-2073, 2008.

• PMI, Um Guia Do Conjunto De Conhecimentos em Gerenciamento De Projetos (Guia PMBOK), Project Management Institute, 2005, ISBN 1930699743, 388p.

• PMI. Provider Profile for SkillSoft Corporation. Disponível em: < http://sparky.occe.ou.edu/pmi/php/profilebox2.php?pass=1008&id=1>. Acessado em: 28 de Dez de 2008.

• PRIBERAM Informática. Dicionário UNIVERSAL da Língua Portuguesa On-Line. Disponível em: <http://www.priberam.pt/dlpo/dlpo.aspx>. Acessado em: 28 de Dez de 2008.

• PRITCHARD, Robert; ASHWOOD, Elissa. Managing Motivation: A Manager's Guide to Diagnosing and Improving Motivation. CRC Press, 2008. ISBN: 1841697893.

• PORTO DIGITAL. Disponível em: <http://www.portodigital.org/>. Acessado em: 28 de Dez de 2008.

• REA, Louis M; PARKER, Richard A. Metodologia de pesquisa: do planejamento à execução. Cengage Learning Editores, 2002. ISBN: 8522102163.

• SALGADO, Léo. Motivação no Trabalho. Rio de Janeiro: Qualitymark 2005.

• SCHOLTES, Peter R.; JOINER, Brian L.; STREIBEL, Barbara J. The Team Handbook. Oriel Incorporated, 2003. ISBN: 1884731260.

• SEI. People Capability Maturity Model® (P-CMM®) Version 2.0. Technical Report, 2001. Disponível em: <http://www.sei.cmu.edu/cmm-p/version2/ >. Acessado em: 28 de Dez de 2008.

• _____. The Team Software ProcessSM (TSPSM). Technical Report, 2000. Disponível em: <http://www.sei.cmu.edu/publications/documents/00reports/ 00tr022.html>. Acessado em: 28 de Dez de 2008.

• _____. Overview of Team Software Process and Personal Software Process: What Is Personal Software Process (PSP)? Disponível em: <http://www.sei.cmu.edu/tsp/index.html>. Acessado em: 28 de Dez de 2008.

• SELLTIZ; WRIGHTSMAN; COOK. Métodos de Pesquisa nas Relações Sociais: Medidas na Pesquisa Social. Org. Louise Kidder; José Roberto Malufe; Bernardete A. Gatti. Vol. 2, 4ª ed. São Paulo: Editora Pedagógica Universitária, 1987.

• _____. Métodos de Pesquisa nas Relações Sociais: Análise de Resultados. Org. Louise Kidder; José Roberto Malufe; Bernardete A. Gatti. Vol. 3, 4ª ed. São Paulo: Editora Pedagógica Universitária, 1987.

• SENIOR, Barbara; SWAILES, Stephen. The dimensions of management team performance: a repertory grid study. International Journal of Productivity and Performance Management Vol. 53 No. 4, 2004, pp. 317-333. Emerald Group Publishing Limited 1741-0401. DOI 10.1108/17410400410533908.

• SHARP, Hellen et al. Models of motivation in software engineering. Information and Software Technology: Elsevier, 2008.

• SKILLSOFT. Why Talent Walks Out the Door; Nigel Paine. Disponível em: < http://www.skillsoft.com/online/skillsoft/form.asp?docnm=Why%20Talent%20Walks%20Out%20the%20Door#URL=http://www.skillsoft.com/infocenter/whitepapers/documents/Why_Talent_Walks_Out_the_Door.pdf>. Acessado em: 28 de Dez de 2008.

• SOMMERVILLE, Ian. Engenharia de Software. ed 8. Tradução: Selma Shin Shimizu Melkinoff, Reginaldo Arakaki, Edilson de Andrade Barbosa. São Paulo: Pearson Addison-Wesley, 2007.

• SOTO, Eduardo; MARRAS, Jean Pierre. Comportamento organizacional: o impacto das emoções. Traduzido de Comportamiento organizacional: impacto de las emociones. Cengage Learning Editores, 2002. ISBN: 8522102732.

• STANDISH Group, The. CHAOS Report. 1995.

• STEVENS, Todd K. The Effects of Roles and Personality Characteristics on Software Development Team Effectiveness. Dissertação submetida à Faculty of Virginia Polytechnic Institute and State University (Doutorado em Ciência da Computação). Blacksburg, Virginia 1998.

• SWAMIDASS, Paul M.. Innovations in Competitive Manufacturing. AMACOM Div American Mgmt Assn, 2002. ISBN: 0814471404.

• TARRICONE, Pina; LUCA, Joe. Successful teamwork: A case study. Higher Education Research and Development Society of Australasia - HERDSA conference 2002 proceedings pp640-646.

• TENÓRIO, Fernando Guilherme. Gestão de ONGs: principais funções gerenciais. Fundação Getúlio Vargas. FGV Editora, 1997. ISBN 8522502234.

• TODOROV, João C.; MOREIRA, Márcio B. O Conceito de Motivação na Psicologia. Revista Brasileira de Terapia Comportamental e Cognitiva. 2005, Vol. VII, nº 1, 119-132. ISSN 1517-5545.

• TRENT, Robert F. Planning to use work teams effectively. Team Performance Management: an international journal Vol. 9 No. ¾, 2003 pp. 50-58. MCB University Press Limited 1352-7592.

• WAINER, Jacques. Métodos de Pesquisa Quantitativa e Qualitativa para a Ciência da Computação. In: Atualização em Informática. Org: Tomasz Kowaltowski; Karin Breitman. Rio de Janeiro: Ed. PUC-Rio; Porto Alegre: Sociedade Brasileira de Computação, 2007, pp221-262.

• WU, Chorng-Guang; GERLACH, James; YOUNG, Clifford. An empirical analysis of open source software developers' motivations and continuance intentions. Information and Management: Elsevier, v. 44, n. 253-262, 2007.

Gestão da Qualidade

Introdução ao Seis Sigma

De que se trata o artigo?

Neste artigo veremos o conceito sobre o que é Seis Sigma, seus objetivos, vantagens, importância e conceitos estatísticos.

Para que serve?

O Seis Sigma significa, para uma organização, oferecer produtos de excelente qualidade, ao mesmo tempo em que elimina todas as ineficiências internas. Em processos de fabricação, Seis Sigma significa não somente oferecer um produto sem defeitos, mas também oferecer produtos de excelente qualidade e aumentar a produtividade, mantendo um rendimento do processo de 99,9999998%, taxas de defeituosos abaixo de 0,002 ppm (parte por milhão) e praticamente erradicando defeitos, reparos no produto e sucata.

Em que situação o tema é útil?

O Seis Sigma proporciona a tomada de decisão com base em fatos e dados ao invés da intuição, uma vez que as variáveis do processo passam a ser mapeadas e mensuradas. Outras características benéficas de processos Seis Sigma são:

- Processos são simplificados, reduzindo o número de passos e tornando-os mais rápidos e eficientes;
- Ciclos de processos são otimizados, reduzindo a possibilidade de geração de defeitos e de oportunidades de erros;
- A produtividade e a lucratividade aumentam, uma vez que defeitos, falhas ou erros aumentam os custos, tornam os produtos mais caros e deixam os clientes insatisfeitos.

Augusta Lopes
augustalopes@gmail.com
Atua no ramo de qualidade e teste de software há mais de 8 anos, é bacharel em Sistema da Informação pela FIR-PE, Especialista em Engenharia de Software - ênfase em Teste de Software pela UFPE e Especialista em Administração da Qualidade - Six Sigma pela FAE-PR. Trabalhou na Motorola no projeto BTC - Brazil Test Center como analista de teste e na Aton Tecnology como Coordenadora de Teste. Atualmente é Analista de Teste na WIPRO Technology. Possui certificação CBTS (Certificação Brasileira de Teste de Software). Palestrante das disciplinas de Teste de Software, Processo de software e Six Sigma, com o objetivo de disseminar a área de Qualidade e Teste de Software no Brasil.

O Seis Sigma foi criado pela Motorola com o objetivo de aproximar todas as áreas da empresa (serviços, produtos, processos e administração) ao padrão de zero defeito. É uma metodologia estruturada para fornecimento de produtos e serviços melhores, mais rápidos com custos mais baixos, com uma forte base em conhecimento de processos e através da redução da variabilidade dos processos. Esta metodologia foca na redução do tempo de ciclo, redução drástica de defeitos e satisfação dos clientes.

Abrantes (2001, p.162) diz que "o Seis Sigma é muito mais do que um método de melhoria de qualidade e aumento de produtividade baseado na estatística.

Ele em verdade causa uma verdadeira revolução e mudanças em toda a empresa, atingindo desde o topo da alta administração até a base. Mais do que quebrar paradigmas, ele mexe com mudanças de hábitos e comportamentos, exigindo total comprometimento e não apenas 'envolvimento'. Exatamente por isso é que se torna um desafio sua implantação."

O Seis Sigma proporciona a tomada de decisão com base em fatos e dados ao invés da intuição, uma vez que as variáveis do processo passam a ser mapeadas e mensuradas. Tem como meta de desempenho, calculada estatisticamente, operar com apenas 3,4 defeitos para cada milhão de atividades ou oportunidades. É uma meta que poucas empresas ou processos podem afirmar ter conseguido.

Onde Aplicar o Seis Sigma e quanto tempo para alcançá-lo?

O Seis Sigma pode ser aplicado tanto em processos de fabricação quanto em processos administrativos, tanto em produtos quanto em serviços. Empresas como a General Electric, Motorola, Allied-Signal, Asea Brown Boveri já conseguiram sucesso com aplicação do Seis Sigma e obtiveram ganhos de bilhões de dólares.

A aplicação do Seis Sigma exige uma transformação cultural para lidar com fatos, dados e pessoas. Para algumas empresas esta transformação exige mudanças radicais em suas práticas de gestão. Por esta razão, é natural que surjam resistências.

Na escolha de projetos para aplicar o Seis Sigma, é importante ressaltar que tem que ser verificado se o projeto atende aos objetivos estratégicos da empresa. O projeto tem que ter uma forte contribuição para o alcance das metas estratégicas da empresa e para o aumento da satisfação dos clientes. Caso não seja assim, deve ser abortado, pois coloca em risco não apenas o projeto, como também o programa de qualidade. Mesmo que o projeto seja bem-sucedido, ao não causar nenhum impacto importante nos negócios, gera o ceticismo de que o programa de qualidade é só mais uma tarefa.

Para alcançar o completo desenvolvimento do Seis Sigma pode levar alguns anos, pois depende de um processo de treinamento e desenvolvimento intensivo das pessoas de toda a organização para utilizar as ferramentas, descrever e otimizar seus processos e adotar uma mentalidade Seis Sigma. Na Motorola foi estabelecido um prazo de cinco anos para alcançar o Seis Sigma. Quando a General Electric decidiu se tornar uma empresa de qualidade Seis Sigma, em 1996, estabeleceu que o faria até o ano de 2000.

Na busca do sucesso na implantação do programa Seis Sigma é muito importante o comprometimento da alta administração da empresa como líderes do processo dessa mudança cultural, a quem os demais seguirão. No seu livro "Criando a cultura seis sigma" a autora Werkema (2002, p.52-55) relaciona alguns pontos críticos para o êxito do programa de qualidade Seis Sigma:

- Garantir recursos suficientes;
- Criar planos de reconhecimento e recompensa, atrelados aos resultados obtidos no âmbito do Seis Sigma;
- Promover a expansão do Seis Sigma para as demais áreas da empresa, fornecedores e clientes;
- Informar e explicar a necessidade de mudança para as pessoas da empresa;
- Modificar gradualmente os sistemas e estruturas da empresa, como atrelar uma parte da remuneração variável aos resultados do programa;
- Traduzir os resultados dos projetos em linguagem financeira;
- Associar projetos Seis Sigma aos objetivos estratégicos da empresa;
- Conseguir alguns resultados no curto prazo (04 a 06 meses);
- Integrar o Seis Sigma aos outros programas de qualidade existentes na empresa.

Quem participa de um projeto Seis Sigma?

Como já comentado antes, a implementação do Seis Sigma depende do comprometimento da alta administração da empresa como líderes do processo dessa mudança cultural. O Seis Sigma deve ser implementado de cima para baixo:

- Sponsor: É o "número um" da empresa, responsável por promover e definir as diretrizes para a implementação do Seis Sigma. O CEO ou Presidente da empresa é a liderança definitiva e deve ser o patrocinador do Seis Sigma;
- Champions: São gerentes de diferentes níveis da empresa que definem os estudos ou projetos, mas sem papel ativo na equipe. Sua função é de manter-se informado sobre o progresso da equipe, remover possíveis barreiras ao seu desenvolvimento, assegurando-se que mudanças, melhorias ou soluções sejam implementadas;
- Master Black Belts: São profissionais que assessoram os Patrocinadores e os Campeões, exercendo posições de tempo integral, 100% dedicadas a oferecer suporte às equipes e aos líderes de Equipe ou Faixas Pretas. Eles atuam como fontes de experiência para todas as suas equipes e fornecem *coaching*, treinamento *just in time* e especialização estatística. Trabalhando com os Campeões, atuam para eliminar os obstáculos que impedem o sucesso de uma equipe. Eles formalizam os estudos e projetos e oferecem à liderança gerencial relatórios de progresso;
- Black Belts: Ainda que 100% dedicados a ajudar suas equipes, os Faixas Pretas não precisam ficar em suas posições em tempo integral. São responsáveis por implementar a metodologia que vai alcançar o Seis Sigma nos seus estudos ou projetos. São membros ativos da equipe e estão encarregados da coordenação geral das atividades e progresso da equipe; designam responsabilidades a todos os Membros da Equipe, acompanhando as metas e planos da equipe e gerenciando os programas e obrigações administrativas da equipe;
- Green Belts: Os Faixas Verdes fazem parte da equipe liderada pelos Faixas Pretas. São funcionários que mantêm seus empregos regulares, mas são designados a uma ou mais equipes de acordo com seu *know how* ou histórico em estudos e projetos selecionados. Eles têm total responsabilidade como um membro do projeto, assim como o líder de equipe.

Conceito estatístico do Seis Sigma

O termo Sigma (letra grega) significa desvio-padrão, que mede a dispersão dos dados de uma amostra ou o grau de

variabilidade, ou seja, indica falhas na saída de um processo, e nos ajuda a compreender até que ponto o processo afasta-se da perfeição.

A média(μ) e o desvio-padrão caracterizam na estatística a denominada *curva de distribuição normal*, que é uma curva em forma de sino que se prolonga indefinidamente em ambas as direções. Existe apenas uma curva de distribuição normal, dados uma média e um desvio-padrão.

A curva de distribuição normal relaciona os valores da média e do desvio padrão com suas probabilidades de ocorrência, possibilitando a previsão da ocorrência de eventos, através do cálculo da área sob a curva. A **Figura 1** mostra o nível de sigmas correspondente à área na curva normal.

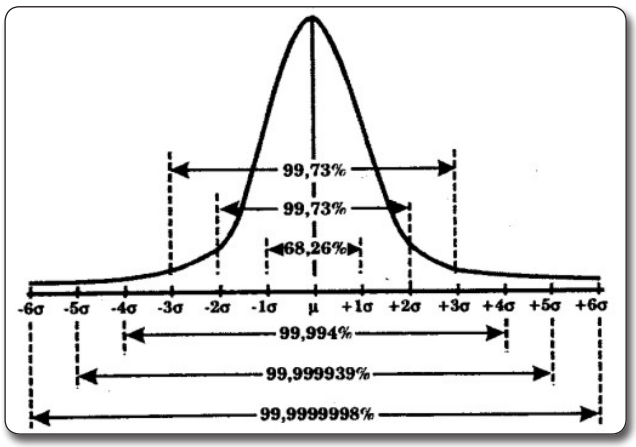


Figura 1. Áreas abaixo da curva normal

Reduzir a variação de resultados é o objetivo principal de um programa Seis Sigma, equivale a taxa de 3,4 defeitos por milhão ou 99,99966% de perfeição. O cálculo do Sigma está em multiplicar o número de unidades processadas pelo número de potenciais defeitos por unidade, dividindo pelo número de defeitos atualmente produzidos, multiplicando tudo por um milhão.

Os sistemas de melhoria, e os processos que ocorrem dentro deles, estão relacionados com a maneira de reduzir a variação do processo e o valor do Sigma (desvio-padrão) de tal forma a ajustar 12 desvios-padrão dentro dos limites de especificação. Não é uma forma de deixar que a média do processo varie mais ou menos 1,5 sigma, produzindo falhas, erros e defeitos em um nível de 3,4 defeitos por milhão, ao contrário, é uma maneira de não deixar que a média do processo varie e produza falhas, erros e defeitos em um nível de 0,002 defeitos por milhão, atingindo praticamente o zero defeito (CAMPOS, 2000, p.30).

O valor do Seis Sigma é composto, derivado da multiplicação de 12 vezes um dado valor de sigma, assumindo seis vezes o valor do sigma dentro do Limite Inferior de Especificação (LIE) da média, e seis vezes o valor do sigma dentro do Limite Superior de Especificação da média (LSE) em uma distribuição normal. Ou seja, ± 6 □ = 12 □.

Na implantação do programa Seis Sigma, um processo é medido pelos índices, relacionados abaixo:

Coeficientes de capacidade do processo: Os principais índices utilizados para medir o desempenho de um processo em relação às exigências da especificação são o Cp e o Cpk, que serão apresentados a seguir. A **Tabela 1** mostra a correspondência entre Cp, Cpk e nível de sigma.

Nível de Sigma	Cp	Cpk
±1 sigma	0,33	0,33
±2 sigmas	0,67	0,67
±3 sigmas	1,0	1,0
±4 sigmas	1,33	1,33
±4,5 sigmas	1,50	1,50
±5 sigmas	1,67	1,67
±6 sigmas	2,0	2,0

Tabela 1. Correspondência entre o nível de SIGMA, CP e cpk.
FONTE: PEREZ-WILSON, 2000, p.175.

- Capabilidade dos Processos (CP): é a razão entre a exigência do cliente, compreendida entre os limites de especificação superior e inferior (LSE e LIE respectivamente), e a variação do processo (sigma). A quantidade de sigmas no denominador corresponde ao total de sigmas em cada lado da curva normal.

A fórmula geral do cálculo do Cp é a seguinte:

$$Cp = \frac{LSE - LIE}{6Sigmas}$$

Se Cp > 1,0, o processo atende as especificações com folga (**Figura 2**).

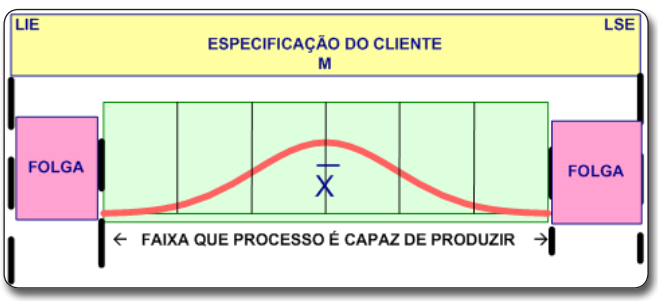


Figura 2. Cp > 1,0. FONTE: BLAUTH, 2006a, p. 54.

Se Cp = 1,0, isto significa que o processo está "justo" e atende as especificações sem folga (**Figura 3**).

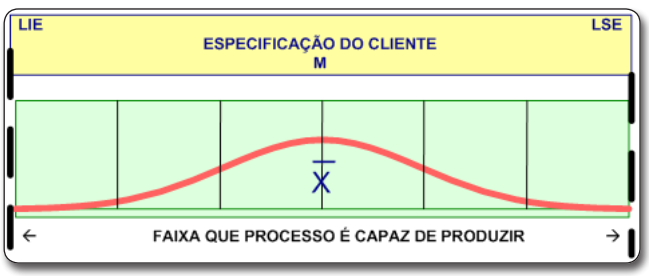


Figura 3. Cp = 1,0. FONTE: BLAUTH, p. 54.

Cp < 1,0 significa que o processo não atende às especificações (**Figura 4**).

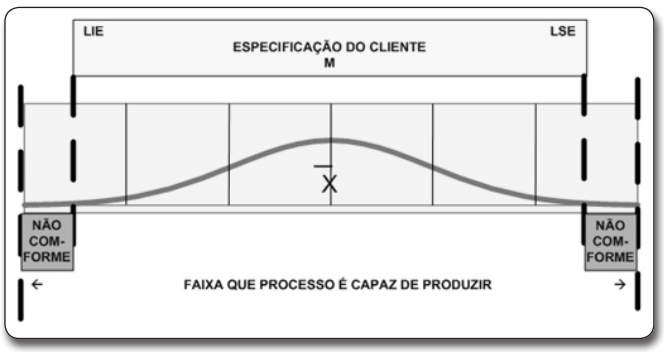


Figura 4. Cp < 1,0. FONTE: BLAUTH, p. 54.

- Capabilidade ajustada dos Processos (CPk)

"As variações aleatórias dos elementos que compõem os processos têm como conseqüência variações no valor médio, que tipicamente flutua em mais ou menos 1,5 desvio padrão" (BLAUTH, 2006b, p. 28). Isso significa que um processo considerado capaz, com Cp >= 1,0 pode não atender as especificações do cliente.

O Cpk considera estas variações, "penalizando" o valor do Cp à medida que a média do processo se afasta da média da especificação. A **Figura 5** mostra a média do processo se afastando da média da especificação fazendo com que o processo deixe de ser considerado capaz, pois excede um dos Limites de Especificação (LSE).

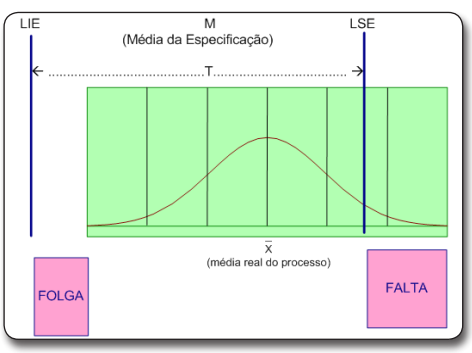


Figura 5. Processo incapaz. FONTE: BLAUTH, 2006b, p. 28.

Se Cpk < 1,0, o processo é considerado incapacitado.

Se Cpk >= 1,0, o processo é considerado capacitado.

A **Figura 6** representa graficamente a situação Cp = 1,0 e Cpk = 0,5. Neste caso, o processo está "justo", atendendo as especificações do cliente sem folga, (Cp=1,0). Quando a média do processo se afastou da média de especificação, a curva se deslocou e ultrapassou um dos limites especificados, o LSE, tornando-o incapacitado.

A **Figura 7** representa graficamente a situação Cp = 2,0 e Cpk = 1,5. O processo atende o nível Seis Sigma (Cp=2,0) de

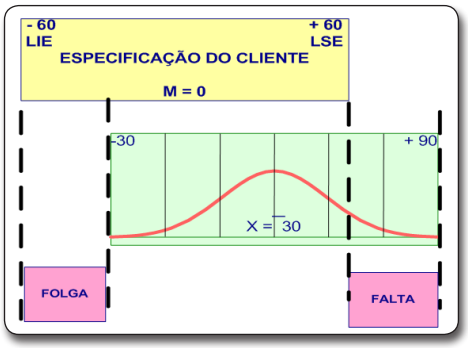


Figura 6. CP = 1,0 e CPK = 0,5 (BLAUTH, 2006b, p. 30)

conformidade e o índice Cpk acusa afastamento da média do processo da média especificada. Mesmo assim, conforme podemos observar, a curva do processo continua compreendida entre os limites especificados e o processo continua capaz.

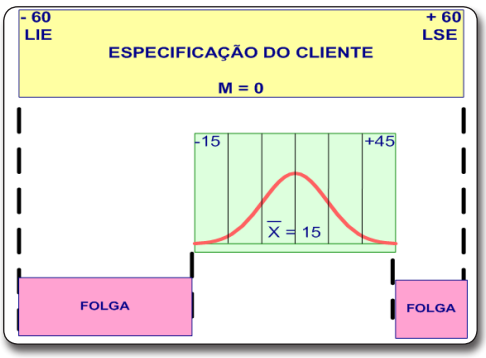


Figura 7. CP = 2,0 e CPK = 1,5. (BLAUTH, 2006b, p. 30)

Além do CP e CPK, na implantação do programa Seis Sigma, pode-se medir o processo pelos índices DPU e DPMO.

Defeitos por Unidade (DPU): é a fração de amostra com defeitos em relação à quantidade total de amostras num determinado ponto de avaliação. A Unidade de representação do DPU é fração, percentual ou PPM – Partes por Milhão.

Fórmula:

$$DPU = \frac{Quantidade.de.Defeitos}{Quantidade.de.amostras}$$

Exemplo 1: Na fabricação de pneus, 20 pneus num conjunto de 1000 fabricados foram rejeitados por apresentarem defeitos. Na **Tabela 2** é apresentado o valor do DPU em fração, percentual e partes por milhão.

Defeitos por Milhão de Oportunidades (DPMO): Número total de defeitos que é encontrado quando se produz um milhão de unidades. É calculado multiplicando-se o DPO – Defeitos por unidade de oportunidade (número de defeitos por unidade (DPU) dividido pelo total de oportunidades) por um milhão. É uma medida direta do nível sigma.

Defeitos	Amostra total	DPU Fração	%	PPM (parte por milhão) = (DPU*Amostra Total)
20	1000	0,02	2	20.000 PPM

Tabela 2. Cálculo do DPU e suas representações por fração, percentual e partes por milhão

Fórmula:

$$DPMO = DPU \times \frac{1.000.000}{Oportunidade.de.defeitos}$$

Exemplo 2: Na **Tabela 3** é apresentado o cálculo do DPMO calculado para o Exemplo 1.

* ***R - Rendimento:*** É a fração ou percentual de amostras que foram consideradas sem defeitos.

$$R = 1 - \frac{Quantidade.de.Defeitos}{Quantidade.de.amostras}$$

** ***Oportunidades de Defeitos:*** é a quantidade de oportunidades que podem gerar um defeito no processo.

Com o DPMO pode-se calcular o valor do sigma do processo através da aplicação da seguinte fórmula:

$$SIGMA = 0{,}8406 + \sqrt{29{,}32 - 2{,}22 \ln(DPMO)}$$

onde ln = logaritmo neperiano.

Método para implantar Seis Sigma

De acordo Werkema (2002), o Seis Sigma pode ser implantado com base nos métodos ***DMAIC*** (***D****efine*, ***M****easure*, ***A****nalyze*, ***I****mprove*, ***C****ontrol*) e ***DMADV*** (***D***efine, ***M***easure, ***A***nalyze, ***D***esign, ***V***erify). O método DMAIC é usado para a melhoria do desempenho de produtos e processos já existentes. Já o *DMADV* é o método para implantação do ***Design for Six Sigma*** **(DFSS)**, sendo utilizado em projetos cujo escopo é o desenvolvimento de novos produtos e processos. Neste artigo será abordado o método DMAIC.

O DMAIC é composto de cinco etapas, como mostra a **Figura 8**.

Definir: O objetivo desta etapa é definir claramente o escopo e a meta do projeto, com base no Business Case do projeto. Nesta etapa, também devem ser levantados o impacto econômico do projeto e quais os clientes afetados, bem como as necessidades, percepções e expectativas percepções destes quanto aos produtos da empresa. No caso de projetos de software, as metas poderiam ser a redução do nível de defeitos encontrados na fase de homologação do sistema.

Medir: O objetivo desta etapa é determinar a localização ou foco do problema, é identificar o que precisa ser medido, definir um plano de coleta de dados e executá-lo. Duas questões devem ser respondidas:

- Que resultados devem ser medidos para a obtenção de dados úteis à focalização do problema?
- Quais são os focos prioritários do problema?

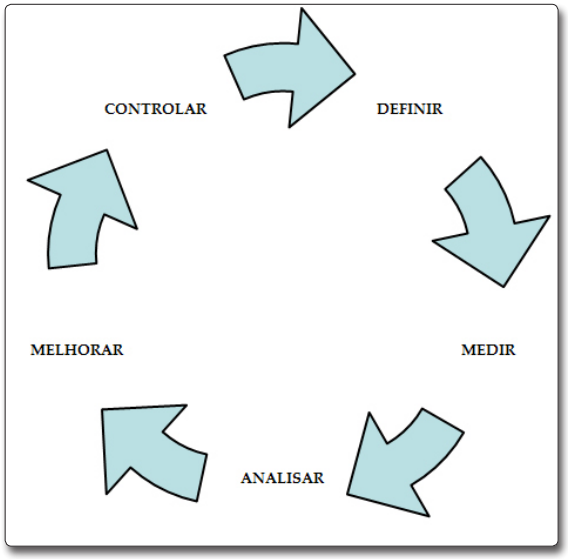


Figura 8. Método DMAIC

É nesta etapa que a equipe deve decidir se os dados já existentes na empresa são suficientes para determinar o problema. Freqüentemente, os dados já existentes não são confiáveis, o que implica na necessidade de coleta de novos dados.

Analisar: O objetivo desta etapa é determinar as causas fundamentais do problema prioritário que precisa de melhoria. É importante uma análise minuciosa nos dados coletados para descobrir quais são os fatores que introduzem variações nos resultados associados ao problema e como essas variações se apresentam.

Melhorar: O objetivo desta etapa é propor, avaliar e implementar soluções para o problema prioritário, gerando idéias sobre soluções potenciais para a eliminação das causas fundamentais do problema detectadas na etapa Analisar. As soluções levantadas e escolhidas pela equipe devem ser testadas em pequena escala. A partir dos resultados obtidos, devem ser identificados e implementados possíveis ajustes ou melhorias nas soluções selecionadas. Após a implementação dos possíveis ajustes, a equipe deve avaliar se as soluções selecionadas tiveram potencial suficiente para levar ao alcance da meta e se não produziram efeitos correlatos indesejáveis. Caso o resultado dessa avaliação seja desfavorável, a equipe deverá retornar à etapa M do DMAIC. Se for favorável (meta atingida), o próximo passo consistirá na elaboração e execução de um plano para implementação das soluções em larga escala.

Controlar: a primeira fase desta etapa consiste em garantir que o alcance da meta seja mantido a longo prazo. Os resultados obtidos após a ampla implementação das soluções

Processo	Defeitos	Unidades	Cálculo DPU	Calculo do Rendimento*	Oportunidade de defeitos**	Cálculo DPMO
Fabricação de Pneus	20	1000	0,02	0,9800	4	5000,00

Tabela 3. Cálculo do DPMO, considerando 4 (quatro) oprtunidades de gerar um defeito no processo de fabricação de pneus.

devem ser monitorados para a confirmação do alcance do sucesso. Caso se verifique que a meta em larga escala não foi alcançada, deve-se retornar à etapa M do DMAIC. Caso tenha sido alcançada, deve-se prosseguir com a padronização das alterações realizadas decorrentes das soluções adotadas. Nesse sentido, novos procedimentos operacionais padrão devem ser elaborados ou os procedimentos antigos devem ser revisados. É importante divulgar os novos padrões para todos os envolvidos por meio de manuais de treinamento, realização de palestras, reuniões e treinamento no trabalho. A próxima fase desta etapa consiste em definir e implementar um plano para monitoramento da performance do processo e o alcance da meta para impedir que o problema já resolvido ocorra novamente no futuro.

Finalmente, todas as atividades realizadas devem ser recapituladas, para que seja feita uma reflexão sobre a forma de condução do projeto e também para que sejam levantados os pontos não abordados no trabalho, assim, sumarizando o que foi aprendido e fazer recomendações para trabalhos futuros.

Conclusão

Este artigo teve como objetivo fazer uma introdução aos conceitos Seis Sigma, apresentando sua definição, objetivo, métodos e conceitos estatísticos. O Seis Sigma está diretamente relacionado com a excelência da qualidade e as empresas que aplicam o Seis Sigma com sucesso têm diretamente um retorno financeiro positivo. No entanto, para o sucesso absoluto do programa Seis Sigma, é necessário todo o comprometimento da alta administração da empresa e de todos os envolvidos no programa.

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

Referências

ABRANTES, J. Programa 8S: da alta administração à linha de produção: o que fazer para aumentar o lucro? A base da filosofia seis sigma. Rio de janeiro: Interciência, 2001.

BLAUTH, R. Estratégia qualidade e produtividade seis sigma. Curitiba, 2006. Apostila (Pós-graduação) – UNIFAE. (Medir)

CAMPOS, M. S. Seis Sigma: sonho ou realidade em atingir o defeito zero. Revista Banas Qualidade nº 96 - Mai/2000 - Pág. 30

PEREZ-WILSON, M. Seis Sigma. Compreendendo o conceito, as implicações e os desafios. Tradução de Bazán Tecnologia e Lingüística. Rio de Janeiro: Qualitymark, 2000.

WERKEMA, M. C. C. Criando a cultura seis sigma. Rio de Janeiro: Qualitymark, 2002.

Perfil Operacional

Informação Estratégica para Testes de Software

Antonio Mendes da Silva Filho

antoniom.silvafilho@gmail.com

Professor e consultor em área de tecnologia da informação e comunicação com mais de 20 anos de experiência profissional, é autor do livros Arquitetura de Software e Programando com XML, ambos pela Editora Campus/Elsevier, possui diversos artigos publicados em eventos nacionais e internacionais, colunista para Ciência e Tecnologia pela Revista Espaço Acadêmico com mais de 90 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.

De que se trata o artigo?

Apresenta o que é perfil operacional e mostra o passo a passo de como construir o perfil operacional de um sistema de software, que constitui informação essencial para as atividades de teste de software.

Para que serve?

Informar como e quando considerar o perfil operacional de um sistema no desenvolvimento de software, visando tornar mais eficientes os testes de software.

Em que situação o tema é útil?

Trata-se de uma prática de engenharia de software considerar o perfil operacional de um sistema de modo a melhor alocar recursos para atividades de teste de software, objetivando aumentar a confiabilidade de um software.

Teste de software é uma das atividades do processo de desenvolvimento de sistema de software que visa executar um programa de modo sistemático com o objetivo de encontrar falhas. Perceba que teste de software não é a última atividade do processo de desenvolvimento de software, conforme uma visão geral do processo RUP (*Rational Unified Process*) mostrado na **Figura 1**. Observe que as atividades de teste ocorrem durante todo o processo.

Agora, como o objetivo básico do teste do software é encontrar falhas, é necessário definir como o software será testado e quanto tempo será gasto com essa atividade. Em outras palavras, torna-se necessário estabelecer quanto

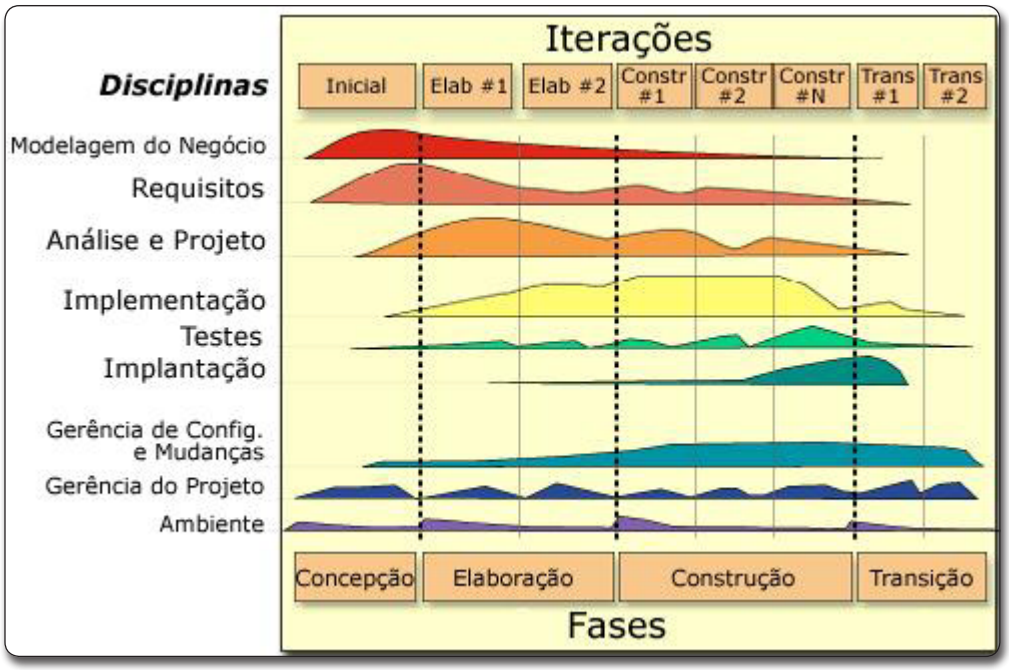


Figura 1. Visão geral do RUP.

de esforço será dedicado aos testes de software de modo a ter um resultado satisfatório.

Perceba aqui que é preciso ter uma estratégia que torne a atividade de teste de software mais eficiente. Nesse sentido, este artigo apresenta o perfil operacional (ou *operational profile*) como uma informação estratégica que objetiva auxiliar a atividade de teste de software.

Teste de Software

Software permeia nosso cotidiano e assegurar que ele funcione corretamente, entregando as funcionalidades (para as quais ele foi projetado) de modo confiável é função do engenheiro de software. No entanto, do ponto de vista prático, as questões abaixo surgem:

1 - É possível ter um software 'perfeito'?

a. Se sim, como?

b . Se não, quando parar de testar?

Antes de responder às questões acima, deve-se lembrar que o componente hardware nos sistemas computacionais é considerado como confiável atualmente. Entretanto, não se pode dizer o mesmo do componente software. A confiabilidade de software tem papel relevante nos sistemas atuais e de suma importância nos sistemas críticos.

Ter o software como elemento essencial dos sistemas coloca sobre ele a necessidade de assegurar qualidade e, mais especificamente, sua capacidade de não apresentar falhas quando em uso, isto é, o software tem de ser confiável. A **confiabilidade de software** é definida como a probabilidade do software operar sem ocorrência de falhas durante um período especificado de tempo em um determinado ambiente.

Com esses conceitos em mente, já podemos responder às questões acima. A resposta para primeira pergunta é não. Para entender isso, você precisa lembrar que faltas podem ser introduzidas em qualquer uma das fases do desenvolvimento de software, ou seja, especificação, projeto, codificação, testes e manutenção. Vale lembrar que se diz que um sistema possui uma **falta** se para algum dado de entrada, o comportamento do sistema está incorreto, isto é, seu comportamento é diferente daquele incluído na especificação do software. Portanto, uma falta é uma causa identificada da falha de software ou erro interno do sistema, o qual é comumente denominado de **bug**. Entretanto, quando a distinção entre falta (causa da falha) e falha (o efeito observável) não é critica, utiliza-se o termo genérico **defeito** para se referir à falta ou a falha.

Além disso, você também deve lembrar que os requisitos de software evoluem, ou seja, eles estão sujeitos a mudanças. Outro aspecto, que não pode ser esquecido, é que software em sistemas de médio a grande porte possui complexidade (inerente) em termos de código, arquitetura e de funcionalidades.

Perceba que essa dificuldade de se obter um 'software perfeito' deve-se essencialmente à natureza do software.

Respondida a primeira questão, resta a segunda: quando parar de testar?

Aqui, temos mais de uma resposta: podemos parar de testar quando não há mais recursos ou não há mais tempo. Outra razão para encerrar os testes é quando a meta de confiabilidade foi atingida ou uma determinada quantidade de falhas foi encontrada.

Cabe salientar aqui que, além de encontrar falhas, testes objetivam aumentar a confiabilidade de um sistema de software, isto é, aumentar a probabilidade de que um sistema continuará funcionando sem falhas durante um período de

tempo. Nesse sentido, para que os testes sejam realizados com mais eficiência, isto é, revelem um maior número de falhas (restando um número muito pequeno de falhas) e, portanto, aumentem o nível de confiabilidade de software até a meta desejada, pode-se utilizar o perfil operacional.

Embora seja desejável testar um sistema por completo, deve-se ter em mente que a atividade de teste assegura apenas encontrar falhas se ela(s) existir(em), mas não asseguram sua ausência. Portanto, as atividades de teste devem ser disciplinadas a fim de identificar a maioria dos erros existentes. Para tanto, um informação essencial a ser considerado é o perfil operacional do sistema, o qual é tratado na seção seguinte.

Perfil Operacional

A atividade de teste de software demanda tempo e recursos ao longo do ciclo de vida do software. Realizá-la de maneira eficiente implica numa redução de custos e tempo. Uma estratégia que pode contribuir para alcançar esta meta é utilizar informações do perfil operacional do sistema de software considerado.

Teste de software tem duas metas: encontrar falhas (se elas existirem) e, consequentemente, aumentar a confiabilidade do software. Uma atividade que pode anteceder a atividade de teste de software é desenvolver o perfil operacional do sistema em desenvolvimento. O processo de determinação e avaliação da confiabilidade de software é ilustrado na **Figura 2**.

Lembre-se que um dos motivos para encerrar os testes de software é alcançar a meta de confiabilidade. Assim, para determinar a confiabilidade, você deve verificar se a confiabilidade alcançada satisfaz ou não a meta definida de confiabilidade. Se o nível de confiabilidade é atendido, então o sistema está pronto para entrar em uso. Caso contrário, mais faltas devem ser removidas, como ilustrado no processo acima (para mais detalhes, você pode consultar o artigo sobre confiabilidade de software da edição de número 9 da Engenharia de Software Magazine).

Observe que uma atividade essencial desse processo, a qual oferece informações às atividades de testes, é o desenvolvimento de um perfil operacional do sistema. Um perfil operacional é um conjunto completo de operações com suas respectivas probabilidades de ocorrência. Aqui, uma operação representa uma importante tarefa que é inicializada por algum ator a qual terá o controle retornado quando a tarefa tiver sido concluída. Note que a tarefa aqui se trata de uma funcionalidade do sistema

Por exemplo, uma probabilidade de ocorrência de 0,20 para o encaminhamento de uma ligação telefônica significa que 20 de cada 100 operações telefônicas serão para encaminhamento de ligação telefônica.

Você deve utilizar a informação do perfil operacional para guiar as atividades de testes. Para tanto, cinco etapas são necessárias:

1. Identificar atores das operações
2. Criar uma lista de operações
3. Revisar a lista operações
4. Determinar as taxas de ocorrência
5. Determinar as probabilidades de ocorrência

Essas cinco etapas acontecem no início do desenvolvimento do sistema de software e são detalhadas e revisadas durante as demais fases do desenvolvimento.

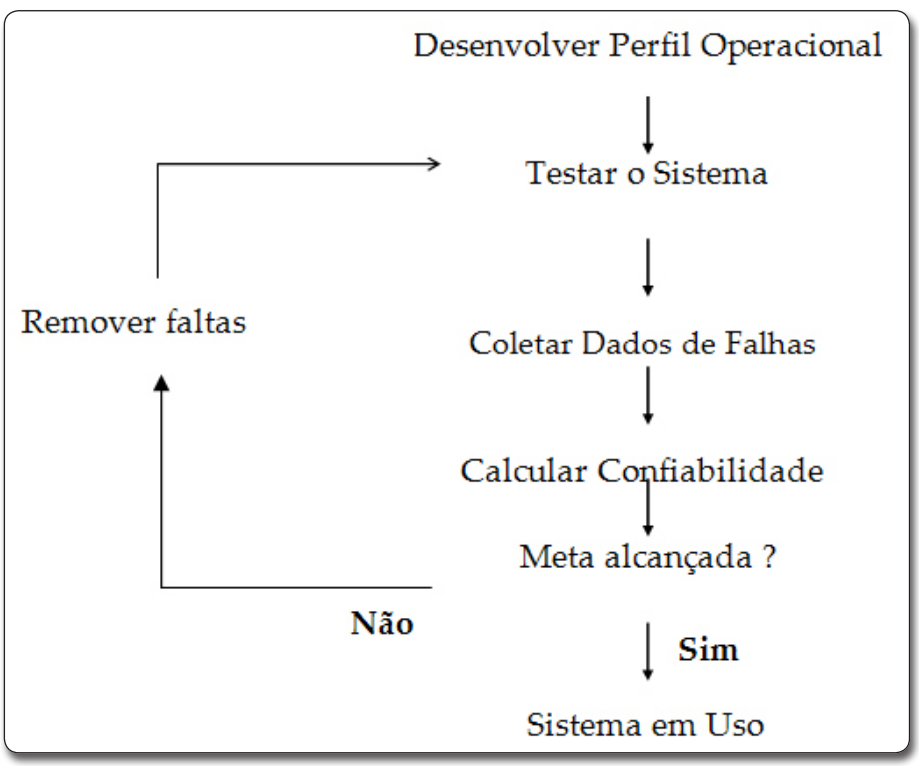


Figura 2. Processo de determinação e avaliação da confiabilidade de software.

A primeira etapa consiste em **identificar os possíveis atores responsáveis por inicializar operações**. Esses atores compreendem:

• Usuários que fazem uso do sistema (causando o início de uma operação). Esses usuários podem ser instituições ou pessoas que utilizam o sistema no seu cotidiano.

• Sistemas externos (ou componentes desses sistemas) que iniciam operações do sistema em desenvolvimento.

A segunda etapa exige você **criar uma lista de operações** e, para tanto, inicialmente, você deve criar uma lista de operações para cada tipo de ator. Para fazer isso, você deve consultar toda documentação do projeto, tais como requisitos do sistema, manual do usuário, eventuais protótipos (se existirem) e diagramas do projeto. Com o objetivo de entender melhor as operações, você pode também se reunir com partes interessadas no projeto (isto é, os *stakeholders*), tais como o engenheiro de software, o projetista de interface e os possíveis usuários, dentre outros.

Observe que você deve entender e definir cada operação do ponto de vista do ator (ou seja, aquele que a inicia), sem considerar como ela será implementada. Cabe destacar que até mesmo operações de inicialização de sistema devem ser consideradas.

Cada operação deve ter um caso de teste associado a ela de modo que cada operação seja executada pelo menos uma vez. O propósito de se identificar operações considera que cada operação deveria ter processamento distinto, o que pode motivar um comportamento com falha. Quando você vai realizar testes num software, você tem o objetivo de fazê-lo de forma eficiente e, portanto, de maximizar as chances que seus casos de testes possam revelar faltas.

Uma informação empírica levantada e utilizada pela comunidade de Engenharia de Confiabilidade de Software diz que há uma probabilidade de aproximadamente de 0.7 que duas operações terão faltas distintas se elas variam em cerca de 100 linhas de código. Nesse sentido, você deve procurar identificar operações que tenham processamentos significativamente diferentes (isto é, em cerca de 70%), o que poderia motivar comportamento inválido (contendo uma falha). Lembre-se que em teste de software, a meta é que cada operação seja executada pelo menos uma vez, a menos que ela não seja crítica (ou seja, que tenha baixa probabilidade de ocorrência) e, portanto, não sendo muito utilizada pelos usuários.

Uma vez que você tenha obtido um conjunto de operações, então você deve **revisar a lista de operações**, o que constitui a terceira etapa no desenvolvimento de um sistema operacional. Para tanto, você não deve se preocupar se sua lista está completa ou não. Observe que cada operação será inicializada por um ator (que pode ser um usuário, componente do sistema ou algum sistema externo). Em tal situação, é uma boa prática ter um especialista para cada tipo de operação.

Considere, por exemplo, que você é parte de uma equipe que está desenvolvendo um software para uma central telefônica. Num sistema como esse, que requer que inúmeros testes sejam realizados para assegurar a confiabilidade de software, você precisará de especialistas para parte da central encarregada de tratar o processamento das ligações telefônicas e, possivelmente, outra pessoa especialista na administração de sistema que tem operações sobre adição e remoção de assinantes.

Adicionalmente, você pode ter outras operações tais como redirecionamento de chamadas telefônicas (também conhecido como *call forward*) ou de tarifação e bilhetagem automática (responsável pela observação de dados de tráfego) que possibilite a medição e registros diários, em forma de relatórios específicos para determinação de custos, ocupação dos troncos e ramais, duração de chamadas, avaliação da carga de serviço em períodos específicos.

Perceba que se trata de operações diferentes e, para tanto, é (quase) imprescindível a participação de especialistas para fazer a revisão da lista de operações. A quantidade de operações dependerá do tamanho do sistema e poderá conter de poucas dezenas até centenas de operações.

Outro aspecto a destacar é que o perfil operacional de um sistema evolui à medida que o sistema vai sendo desenvolvido, pois mais informações são obtidas, o que permite seu refinamento. Um exemplo de um conjunto de operações de um software de central telefônica com respectivos atores (responsáveis por inicializá-las) é apresentado na **Tabela 1**.

Ator	Operação
Controlador de ligação telefônica	Processar ligação telefônica completada
Controlador de ligação telefônica	Processar ligação telefônica não completada
Gerenciador de recursos	Controlar recursos p/ completar ligação telefônica
Assinante	Iniciar ligação telefônica
Assinante	Cancelar ligação telefônica
Assinante	Redirecionar ligação telefônica
Administrador de sistema	Adicionar assinante
Administrador de sistema	Alterar dados de assinante
Administrador de sistema	Remover assinante
Gerenciador da base de dados	Validar a base de dados de assinantes

Tabela 1. Exemplo de lista de operações para uma central telefônica.

Observe na tabela que as operações listadas compreendem funcionalidades de um sistema exemplo (central telefônica). Embora isso tenha sido utilizado aqui para sistema telefônico, o mesmo procedimento pode ser feito em outros sistemas como um caixa eletrônico, um editor de texto ou mesmo um sistema de informação.

Em um caixa eletrônico, exemplos de operações são *sacar* e *consultar saldo*, enquanto num sistema acadêmico de uma universidade pode-se ter *matricular aluno* como exemplo de uma operação. De um modo geral, em sistemas orientados a objetos podem-se ter cada operação associada a um ou mais casos de uso (existente na modelagem de sistemas).

Depois de revisar a lista de operações, o próximo passo é **determinar a taxa de ocorrência** para o conjunto de operações levantado. A taxa de ocorrência de uma operação compreende a quantidade de ocorrências da operação dividida pelo tempo que o conjunto de operações está sendo executado. Determinar

a taxa de ocorrência de operações é essencial para obtenção do perfil operacional de um sistema de software. No entanto, trata-se de uma tarefa que requer habilidade do engenheiro de software em buscar dados e, para tanto, pode-se:

- Primeiramente, buscar por dados numa versão anterior do mesmo sistema (se existir) ou em sistemas similares.
- Buscar por dados no *log* de sistema (se existir).
- Conversar com pessoal da área de Marketing ou quaisquer outras informações relativas às regras de negócio do produto (sistema de software).
- Registrar dados de uso (isto é, dados de campo) do sistema considerado.
- Fazer simulações do sistema e determinar a taxa de ocorrência de eventos que estimulam a execução de operações.
- Fazer estimativas (caso não exista quaisquer informações iniciais) utilizando-se do conhecimento de profissionais experientes (engenheiro de software e/ou engenheiro de sistema) da área, ou então aplicar o método Delphi no qual um grupo de especialistas se reúne fazendo estimativas, discutindo e refinando essas estimativas e, por fim, fazendo estimativas refinadas.

Para o conjunto de operações mostrado na **Tabela 1**, apresentam-se na **Tabela 2** exemplos de taxa de ocorrência dessas operações.

Operação	Taxa de ocorrência (operações/hora)
Processar ligação telefônica completada	10.000
Processar ligação telefônica não completada	5.000
Controlar recursos p/ completar ligação telefônica	8.000
Iniciar ligação telefônica	8.000
Cancelar ligação telefônica	5.000
Redirecionar ligação telefônica	2.800
Adicionar assinante	180
Alterar dados de assinante	10
Remover assinante	10
Validar a base de dados de assinantes	1.000
Total	40.000

Tabela 2. Exemplo de taxas de ocorrências para uma central telefônica.

As taxas de ocorrências apresentadas na **Tabela 2** visam informar a quantidade de vezes que cada operação é executada no período de uma hora. Em outras palavras, ela nos diz quais operações (ou funcionalidades) são mais requisitadas pelos usuários ou por outros atores do sistema.

É importante também ressaltar que as dez operações apresentadas na **Tabela 2** não contemplam todas as operações (ou conjunto de funcionalidades) de um software de central telefônica. Nesse sentido, o desenvolvimento de um perfil operacional de um sistema consiste em criar e utilizar uma organização (ou arquitetura) baseada na operação, ao invés de ser baseada em componentes. Perceba que as operações (diferentemente de componentes) de software são partes de um produto e, portanto, as operações podem ser utilizadas para planejar a entrega do produto com diferentes *releases* (ou versões) que suportam operações específicas.

Outro aspecto a destacar é o uso do *Princípio de Pareto* utilizado no desenvolvimento do perfil operacional de um sistema de software. Tipicamente, num software, cerca de 5% de suas operações são responsáveis por oferecer 80% das funcionalidades desejadas pelos usuários.

Uma vez que você tenha determinado a taxa de ocorrência do conjunto de operações do sistema de software, o próximo passo é **determinar as probabilidades de ocorrência**.

Para tanto, você deve dividir a taxa de ocorrência de cada operação pelo somatório (ou total) da taxa de ocorrência das operações do sistema. Em seguida, você deve ordenar as probabilidades obtidas em ordem descendente, geralmente, com o objetivo de destacar aquelas que possuem uma maior probabilidade de ocorrência. Realizando este procedimento na **Tabela 2**, você irá obter a **Tabela 3** das probabilidades de ocorrência do sistema exemplo.

Observe que, com as informações apresentadas na **Tabela 3**, você poderá distribuir a quantidade e esforço das atividades de teste de software de maneira proporcional à probabilidade de ocorrência apresentada no quadro de perfil operacional do sistema, que destaca um conjunto de operações que têm maior probabilidade de ocorrência. Perceba ainda que essa informação pode até mesmo ser utilizada nos estágios iniciais de desenvolvimento de software a fim de priorizar quais operações são consideradas como *críticas* e que devem ser tratadas prioritariamente nos primeiros *releases* (versões) do produto.

Operação	Probabilidade de ocorrência
Processar ligação telefônica completada	0.25
Controlar recursos p/ completar ligação telefônica	0.20
Iniciar ligação telefônica	0.20
Cancelar ligação telefônica	0.125
Processar ligação telefônica não completada	0.125
Redirecionar ligação telefônica	0.07
Validar a base de dados de assinantes	0.025
Adicionar assinante	0.0045
Alterar dados de assinante	0.00025
Remover assinante	0.00025
Total	1.0

Tabela 3. Exemplo das probabilidades de ocorrência para uma central telefônica.

Se você levar em conta o *Princípio de Pareto* e utilizar as informações obtidas no perfil operacional de um sistema de software, você poderá planejar a realização de testes e entrega de versões em três etapas:

1. A primeira versão do software deveria entregar 80% das funcionalidades desejadas pelo cliente, o que corresponde a aproximadamente 5% das operações do software (pelo *Princípio de Pareto*).

2. A segunda versão, você poderia planejar entregar outros 15% das funcionalidades, o que poderia corresponder a cerca de 25% das operações.
3. Por fim, a terceira versão iria englobar os 5% restantes das funcionalidades, ou seja, aos 70% de operações menos utilizadas.

Se você participa ou está no comando do desenvolvimento do software de um sistema, você deve ter iniciativas que objetivam tornar eficiente a realização das atividades. Torná-las eficientes significa entregar o produto no prazo, com qualidade e custo reduzido. Isto requer uso eficiente dos recursos e, para tanto, você deve alocar recursos de desenvolvimento para melhor atender às necessidades do cliente o mais cedo possível. Portanto, adotar um planejamento de entregas em conformidade com o apresentado acima é uma boa prática.

Neste momento, é importante destacar os pontos essenciais de tudo o que foi apresentado. Você recorda a definição de confiabilidade de software?

Confiabilidade de software é a probabilidade de um sistema de software operar sem a ocorrência de falhas em um determinado ambiente operacional durante um período de tempo. Mas, o que compreende esse ambiente operacional? Hardware (computadores e infra-estrutura de apoio), software (sistema operacional, aplicativos e outros utilitários) e o grau de utilização (isto é, o perfil operacional) do sistema. Note que grau de utilização é uma caracterização quantitativa de como o sistema será utilizado.

Essa informação obtida com o perfil operacional de um software pode ser utilizada de duas maneiras:

1. Aumentar a eficiência dos testes a serem realizados;
2. Distribuir, de modo mais eficiente, o esforço de desenvolvimento de um sistema.

De tudo o que foi apresentado acima, uma lição deve ficar em mente: o engenheiro de software e o responsável pelas atividades de teste de software devem estar envolvidos durante todo o processo de desenvolvimento de software, pois isto permite:

- Ter um entendimento maior das necessidades do cliente (de como e quanto às funcionalidades serão utilizadas);
- Planejar melhor a alocação de esforço e recursos no desenvolvimento do sistema (visando priorizar aquelas operações de maior ocorrência).

Para finalizar, vale ressaltar que as informações oferecidas num perfil operacional servem de apoio às decisões do gerente de projetos tanto sob a perspectiva de redução de custos quanto de identificação de faltas (causadoras de falhas de software) nas funcionalidades mais usadas.

Conclusão

Teste de software é uma atividade essencial no desenvolvimento de um sistema de software que tem o objetivo de sistematicamente encontrar falhas. Uma das motivações para encerrar os testes ocorre quando a meta de confiabilidade de software é alcançada. Uma estratégia que pode ser empregada para tornar o teste de software mais eficiente é desenvolver e aplicar o perfil operacional do sistema considerado, o que contribui para atender as metas de qualidade, cronograma e custos de um projeto. ●

Links

Software Reliability Engineering: A Roadmap
http://csse.usc.edu/classes/cs589_2007/Reliability.pdf

Software Metrics and Reliability
http://satc.gsfc.nasa.gov/support/ISSRE_NOV98/software_metrics_and_reliability.html

Reliability Engineering
http://en.wikipedia.org/wiki/Reliable_system_design

Reliability Growth Reference On-Line Help
http://www.weibull.com/RelGrowthWeb/reliabilitygrowth.htm

Teste funcional utilizando o Abbot Framework

Automatizando testes em aplicações Java Desktop

Vinícius Rodrigues de Souza
vrsouzainfo@gmail.com
Cursa Sistemas de Informação na Faculdade Metodista Granbery, graduando em Engenharia Civil pela Universidade Federal de Juiz de Fora e estagiário na Prefeitura de Juiz de Fora na área de desenvolvimento e testes de software.

Ricardo Cunha Vale
valericc@gmail.com
Cursa Sistemas de Informação pela Faculdade Metodista Granbery, estagiário na Prefeitura de Juiz de Fora na área de desenvolvimento e testes de software, estagiário na área de desenvolvimento de projetos na Granbery Consultoria Júnior, em parceria com a Fundação COPPETEC.

Marco Antônio Pereira Araújo
maraujo@granbery.edu.br
Doutor 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 da Faculdade Metodista Granbery, Analista de Sistemas da Prefeitura de Juiz de Fora, Editor da Engenharia de Software Magazine.

De que se trata o artigo?
Esse artigo apresenta a utilização do framework Abbot Java GUI Test Framework, capaz de executar testes funcionais baseado em eventos a fim de antecipar possíveis problemas durante a utilização de um software. O Abbot framework é uma biblioteca Java para GUI (Graphic User Interface), que fornece métodos para reproduzir ações do usuário e examinar o estado dos componentes GUI. O framework pode ser utilizado diretamente a partir do código Java inserido na aplicação ou acessados sem programação através da utilização de scripts através do Costello Script para Abbot.

Para que serve?
O Costello é uma aplicação desenvolvida totalmente em Java, que auxilia na geração de testes Funcionais para aplicações. A ferramenta auxilia na criação de testes de Interface com o Usuário (Java UI), sendo capaz de simular comportamentos do software e avaliar o resultado encontrado.

Em que situação o tema é útil?
Assim como outras ferramentas de teste funcional, como a Selenium IDE, o intuito desse processo é prever o comportamento do software e assegurar que o resultado obtido na simulação é realmente o esperado.

O Teste de Software é uma importante etapa na construção de sistemas de qualidade, onde cada vez mais as empresas, visando obter qualidade em seus produtos, têm optado por investir recursos nessa etapa, essencial para prevenir gastos com manutenções futuras.

Os benefícios dos testes de software são notados não apenas pela empresa desenvolvedora, pois primando pela qualidade do software, além de evitar gastos com manutenções desnecessárias, o cliente pode ter em suas mãos um software mais estável e menos sujeito a falhas.

Os benefícios trazidos por uma boa equipe de testes, com sua própria estrutura e metodologia são inúmeros. Empresas que se adéquam a esta tendência obtém melhores resultados e investem cada vez mais nesta atividade, primando sempre pela qualidade de seus produtos.

Devido à maior interatividade dos sistemas com os usuários, a GUI (Graphic User Interface) está se tornando uma porção cada vez maior de um sistema, necessitando testar estes componentes cada vez mais. Dos processos de automação de testes, o de interface se mostra como um grande desafio, já que requer um envolvimento constante entre o usuário e o desenvolvedor.

As ferramentas de automação de testes ainda são complexas e caras, e não substituem o papel da pessoa responsável por realizar os testes no sistema. Além disso, cabe ao desenvolvedor estabelecer o que deve ou não ser automatizado, pois é impossível uma cobertura de todas as tarefas de um sistema.

Neste artigo faremos um estudo de caso a fim de mostrar um dos tipos de teste de software, o teste funcional, através da configuração e utilização do framework Abbot. Esta visão geral do Abbot trará uma idéia do tratamento de diferentes componentes do sistema e como eles podem ser utilizados. A versão da ferramenta a ser abordada neste artigo será a 1.0.2, e pode ser encontrada para download no site http://abbot.sourceforge.net/doc/download.shtml.

Em geral, a utilização do Abbot consiste em obter referências de componentes GUI, realizar ações do usuário sobre os componentes e fazer algumas afirmações sobre o seu estado. Estas operações podem ser feitas tanto de um script de alto nível ou diretamente a partir de código Java (por exemplo, em um método JUnit TestCase).

O Abbot Framework pode ser utilizado de duas formas. Pode-se criar uma classe dentro da aplicação que contenha os testes, ou utilizar um editor de script de testes. No artigo em questão será abordado o Costello Editor de Script para o Abbot. Este editor é fornecido para facilitar a criação e manutenção dos Scripts do Abbot. O editor suporta gravação de eventos para facilitar o usuário escrever scripts, e também pode reproduzi-los para depuração.

Os scripts são uma forma conveniente de organizar um conjunto de ações e afirmações que compõem um caso de teste, salvos no disco no formato XML. São as unidades básicas de teste do Abbot e, em geral, consistem em guardar informações sobre componentes, suas ações e afirmações sobre o seu estado. Comparados com scripts para testes unitários, os de testes funcionais podem ser muito mais longos e complexos. Ao gerar um script, todas as informações necessárias para executar o teste são encapsuladas dentro dele, fornecendo independência na utilização do mesmo em qualquer ambiente.

Scripts são essencialmente compostos de actions, assertions, e referências de um componente. As actions são comandos geralmente realizados pelos usuários em um componente GUI, como pressionar um botão, selecionar um item de menu, ou digitar texto.

Assertions adicionados no script permitem que seja verificado o estado da GUI. Normalmente, durante a execução do teste, o script irá parar de executar e relatar um erro se algum assertion falhar. Estes assertions servem também para aguardar passos, onde o script em execução pára até que uma condição se torne verdadeira.

O Costello Editor de Script possui registradores que possibilitam a captação das informações mais básicas, como um movimento de mouse, além dos principais eventos ocorridos no sistema em teste. O editor possui registradores para a maioria dos componentes Swing padrão e permite o acréscimo de registradores de apoio para componentes adicionais.

Alguns scripts registram da GUI informações como coordenadas para identificar os componentes. Esses scripts são muito frágeis, porque qualquer alteração no posicionamento dos atributos (mudanças no layout ou executando o script em uma plataforma diferente), inviabiliza a execução dos scripts. O Abbot utiliza uma série de atributos para identificar o componente dinamicamente, independente do posicionamento de seus atributos.

Através dessas características, podem-se destacar alguns pontos relevantes na utilização do framework:

- Reprodução confiável de entrada do usuário: as GUI geralmente não são muito testadas, pois não é fácil para o programador simular as entradas dos usuários. Com este framework este problema é reduzido consideravelmente.
- Testes em formato de script: as informações contidas no script são interpretadas, não havendo a necessidade de se criar um código que exige compilação.
- Alto nível nos registros das ações: o Abbot constrói um nível de abstração em cima da classe java.awt.Robot (esta classe é utilizada para gerar eventos de sistema para fins de teste de automação, executar aplicações onde o controle do mouse e do teclado são necessários, tendo por principal objetivo facilitar testes automatizados). Neste nível os testes são mais fáceis de manter, pois é possível identificar e deduzir o que um conjunto de instruções irá fazer como, por exemplo, "Move to 110, 110" e "Press down key".
- Suporte a gravação de eventos: apesar do XML poder ser escrito sempre a mão editando cada passo do Script, é mais interessante que seja usado um recurso do Costello Editor que grava automaticamente uma sequência de ações do usuário e as guarda em um script.

Conhecendo a ferramenta

Após o download do arquivo Zip, basta descompactar e abrir o arquivo, clicando em "abbot.bat". A janela inicial da ferramenta, além da barra menu superior, já conta com algumas abas para configuração (ver **Figura 1**).

Serão apresentados aqui alguns dos itens de menu considerados mais relevantes na utilização do Abbot:

O menu "File" possui os seguintes itens de menu:

- New Script: o framework inicia sem qualquer script para ser configurado, então obrigatoriamente deve-se acionar este item para que seja criado um script com um comando para a inicialização do aplicativo a ser testado.
- Save: Comando padrão para muitos sistemas. Depois de criado um script de teste, este item o salva em formato XML.
- Rename: Item que permite renomear um script já salvo.
- Close Script: Fecha o script aberto e volta com o framework para o seu estado inicial.
- Delete Script: Fecha o script e deleta o arquivo no diretório salvo.

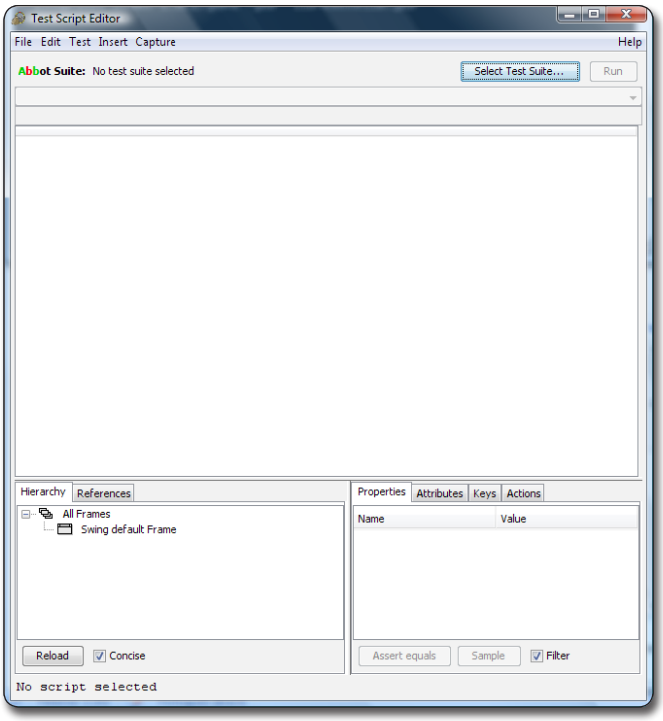


Figura 1. Visualização da tela inicial do Abbot.

O menu "Edit" é responsável por contribuir na edição do script de texto e possui os seguintes itens de menu:

- Move Up: transporta para a linha superior qualquer linha selecionada que tenha sido inserida ou gerada no script.
- Move Down: transporta para a linha inferior qualquer linha selecionada que tenha sido inserida ou gerada no script.
- Clear Script: limpa as linhas geradas ou inseridas no script.

O menu "Test" possui como itens de menu opções para controlar o teste a ser executado, dentre elas estão:

- Run: "roda" o teste executando cada linha do script criado.
- Run to Selection: o teste será executado até uma linha de comando específica previamente selecionada.
- Launch: comando responsável por inicializar e abrir a janela principal do sistema a ser testado pelo script.
- Terminate: finaliza e fecha a janela do sistema a ser testado pelo script.
- Stop on Failure: selecionando esta opção, a execução do teste irá parar em um ponto específico onde houve alguma falha.
- Stop on Error: selecionando esta opção, a execução do teste irá parar em um ponto específico onde houve algum erro que impedisse o prosseguimento do teste.
- Slow Playback: comando muito utilizado que executa as linhas do script de maneira mais lenta, sendo possível visualizar as ações de cada comando contido no script.

Todos os itens do menu "Insert" são comandos que podem ser acrescentados ao script do teste. Dentre eles estão:

- Annotation: semelhante ao item Comment, porém está associado a um componente.
- Call: inclui uma chamada de um método.
- Comment: acrescenta um comentário ao script na posição desejada.
- Expression: insere um código Java script arbitrário.
- Launch: comando utilizado para carregar uma aplicação a ser estada, sendo criado quando é utilizada a opção New Script.
- Sample: salva o resultado de um método anteriormente chamado.
- Script: insere um script salvo com todos os seus comandos dentro do novo script que está sendo feito.
- Terminate: Comando utilizado para fechar a aplicação e terminar o script. Já é criado quando utilizada a opção New Script.
    - Action: possui todos os comandos que testam ações feitas na aplicação, como pressionar um botão ou soltá-lo, ou movimento do mouse.
    - Assert: possui opções de comparação a serem feitas na aplicação, que podem ser inseridas manualmente.
    - Wait for: possui comandos a serem feitos na aplicação que serão realizados a partir de determinado tempo, ou após o uso de algum componente.

O menu "Capture" define quais atividades impostas ao sistema em teste serão capturadas:

- All Actions: com as configurações devidamente realizadas no framework, este comando irá abrir a aplicação e capturar as ações impostas ao sistema a ser testado.
- All Actions (including motion): este comando é semelhante ao All Actions citado anteriormente, porém ele captura também, durante a utilização da aplicação, outras ações como movimentos da janela da aplicação e do cursor.
- Component Image: serve para localizar a imagem do componente desejado (na aba Hierarchy) e assim obter suas propriedades.
- Component Reference: semelhante ao Component Image, sendo possível ter todas as propriedades relacionadas ao componente, bastando apenas selecioná-lo e, em seguida, utilizar as teclas de atalho referente ao comando.

Criando o primeiro teste

Para exemplificar a utilização da ferramenta, será utilizado um aplicativo desenvolvido em Java para Desktop através da IDE NetBeans (ver exemplo na **Figura 9**). O sistema trata da verificação da situação de um aluno que, dadas suas notas e frequência, verifica sua situação quanto à aprovação ou reprovação. De acordo com o que foi estabelecido, para aprovação do aluno, ele deve possuir frequência maior ou igual a 75%, a média das notas maiores que 70 pontos para aprovação direta, e maior que 50 pontos para aprovação com prova final.

Na **Figura 2** é apresentado o exemplo do código que faz esta verificação com as condições descritas. Foi criada uma classe aluno que possui como atributos nota1, nota2, notaFinal e freqüência, com os seus devidos métodos de acesso.

Inicia-se verificando se frequência do aluno é menor do que 75, reprovando-o nesse caso. Senão, se a média das duas notas for menor do que 30, o aluno também está reprovado diretamente. Se a sua media é igual ou maior que 70, ele está aprovado. Senão, faz-se uma média da média anterior somada

com a nota da sua prova final, e verifica se esse valor é maior ou igual a 50, sendo o aluno aprovado em caso verdadeiro (ver **Figura 3**).

```
public class Aluno {

    private float nota1;
    private float nota2;
    private float notaFinal;
    private int frequencia;

    public float getNota1() {...}

    public void setNota1(float nota1) {...}

    public float getNota2() {...}

    public void setNota2(float nota2) {...}

    public float getNotaFinal() {...}

    public void setNotaFinal(float notaFinal) {...}

    public int getFrequencia() {...}

    public void setFrequencia(int frequencia) {...}
}
```

Figura 2. Classe Aluno

```
media = ((aluno.getNota1() + aluno.getNota2()) / 2);

if (aluno.getFrequencia() < 75) {
    jTextArea1.append("Aluno Reprovado");
} else {
    if (media < 30) {
        jTextArea1.append("Aluno Reprovado");
    } else {
        if (media >= 70) {
            jTextArea1.append("Aluno Aprovado");
        } else {
            if (((media + aluno.getNotaFinal()) / 2) < 50) {
                jTextArea1.append("Aluno Reprovado");
            } else {
                jTextArea1.append("Aluno Aprovado");
            }
        }
    }
}
```

Figura 3. Código que verifica aprovação do aluno.

Inicialmente, na tela principal do Costello, seleciona-se File na barra de menu e cria-se um novo Script na opção New Script. Após isso, o programa cria automaticamente (por padrão) dois passos a serem executados, o Launch e o Terminate. O primeiro define a classe e o método a serem carregados, e o Terminate fecha a aplicação e termina o script.

Em seguida pode-se nomear o novo script no campo onde está escrito Untitled, onde será utilizado o título "TesteAluno_Aprovado", já que no primeiro teste será verificada a aprovação do aluno. Com o nome já atribuído ao script, o próximo passo é ir à linha Launch que já foi criada e, após isso, modificar sua propriedade Classpath indicando o caminho do aplicativo ou da classe a ser testada. Depois se modifico o Target Class Name, adicionando o nome da classe específica a ser testada. Com isso, o sistema automaticamente localiza e disponibiliza os métodos contidos nessa classe. Será testado um Form JFrame e, por isso, o método a ser testado é o main[] (ver **Figura 4**).

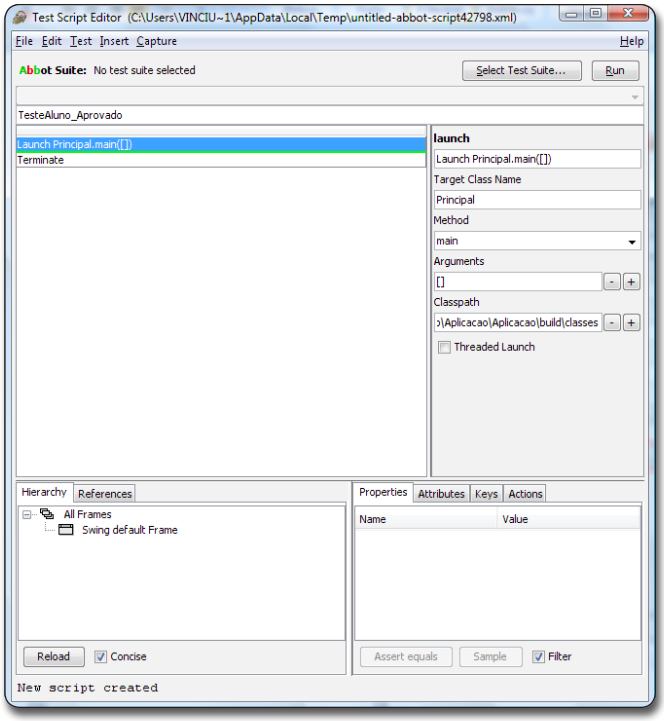


Figura 4. Script configurado

Com as configurações iniciais feitas, serão capturadas as ações do sistema, que receberá dados que qualificarão o aluno como aprovado. Este processo é feito através da seleção do Capture na barra de menu selecionando All Actions. Esta opção irá capturar somente os estados fundamentais do sistema. Logo em seguida, desmarca-se a caixa de seleção Concise, pois com esta opção selecionada, alguns componentes não ficaram expostos para configurações futuras (ver **Figura 5**).

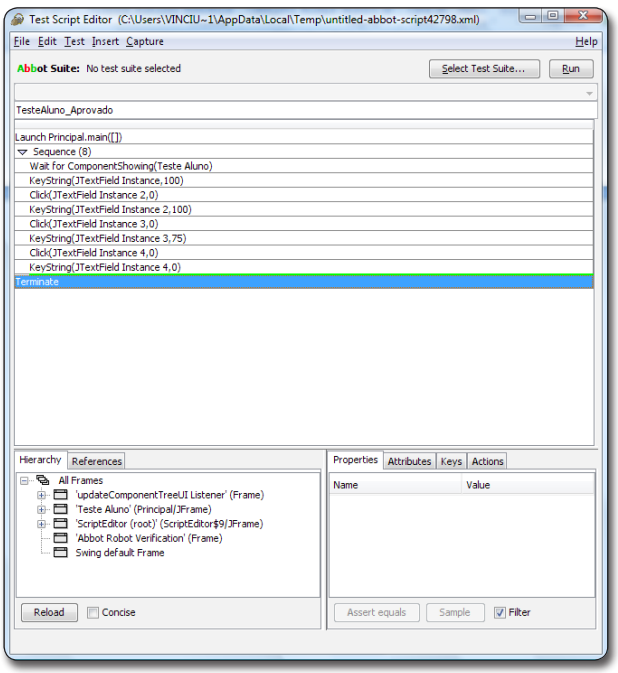


Figura 5. Captura concluída.

Atribuiremos para a nota 1 do aluno o valor de 100 pontos e, em seguida com o clique do mouse, seleciona-se o segundo campo correspondente à nota 2 do aluno, que também atribuiremos 100 pontos. Partiremos agora para a frequência, e diferente do que foi feito anteriormente, será utilizada a tecla TAB para atingir o próximo campo. Foi atribuído 75% de frequência para o aluno e, para completar, é atribuído o valor 0 para nota final, já que devido aos dados informados anteriormente, o aluno será aprovado direto sem prova final.

Com os dados preenchidos, é hora de fazer a verificação apertando o botão "Verificar". É esperada uma mensagem do sistema informando que o aluno está aprovado. O componente utilizado para a mensagem é um JTextArea. Após acionado o botão "Sair", a gravação para e o teste termina.

Agora partiremos para o processo onde se faz as comparações necessárias para a validação do sistema.

A comparação que será feita é que o aluno deve estar obrigatoriamente aprovado, então o que será comparado é a mensagem que o sistema informará sobre a situação do aluno. Na aba References pode ser localizado o componente a ser testado, caso deseje ter certeza do componente, deve-se pressionar Ctrl + L, caso você já tenha fechado o programa. Em seguida, posicione o ponteiro do mouse no componente a ser comparado e pressione Shift + F1. A aba Hierarchy apontará o componente e, ao lado, estarão listadas suas propriedades (ver **Figura 6**).

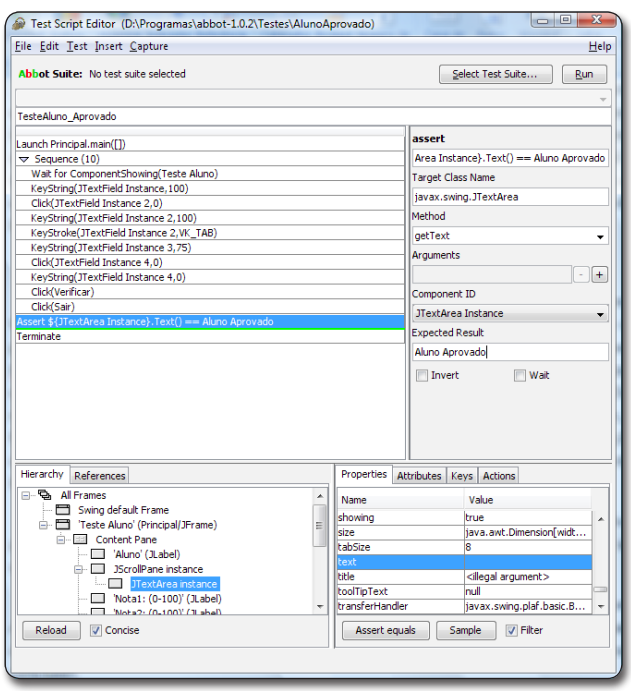


Figura 6. Comparações inseridas no script

Pode-se perceber que todas as propriedades dos componentes podem ser observadas e testadas, bastando selecioná-las e pressionar o botão "Assert equals", logo abaixo da caixa de propriedades. Neste caso específico, escolhemos o componente textArea e a sua propriedade text. Feito isso, percebe-se que foi adicionada uma linha na janela do script, linha essa que conterá o valor a ser comparado. Após selecioná-la, são exibidas (à direita), as configurações referentes àquela linha. No primeiro textfield (assert) estão os dados do componente, a função a ser comparada e o valor encontrado. Essa linha nada mais é do que uma composição de todas as linhas abaixo dela, e que o software se encarrega de montar automaticamente. Note as linhas Expected Result, Component ID e Method.

Logo abaixo é possível ver uma caixa de seleção com o nome invert, sendo responsável por inverter o sentido da comparação, de == (igual) para != (diferente), que também pode ser selecionada, facilitando o processo de teste.

No caso de teste em questão, determinou-se que qualquer valor encontrado diferente de "Aluno Aprovado" seria considerado algum tipo de defeito no sistema. Como colocamos condições para que o aluno seja aprovado de acordo com as regras já citadas, basta posicionar a linha de comparação ao local correto, e o script de teste já estará concluído, devendo estar antes do evento Click do botão "Sair" que encerra a aplicação (ver **Figura 7**).

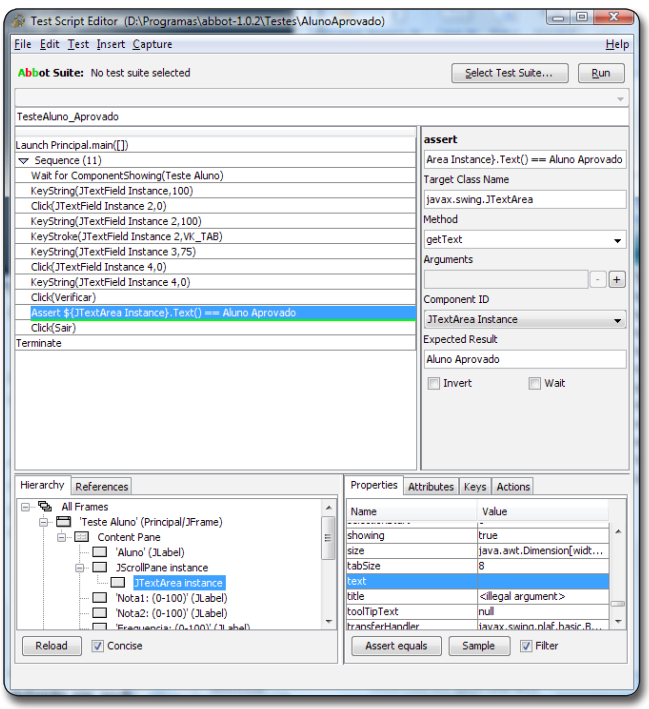


Figura 7. Assert incluído

Este teste foi construído para ser usado em diversas situações, umas delas é se, por acaso, alguém der manutenção no sistema em questão e mudar a regra de negócio. Com o teste definido, ele pode ser utilizado logo após esta manutenção, verificando a integridade das informações após as modificações realizadas.

Para demonstrar a utilidade do script, realizaremos uma simulação desta situação. O código será modificado e em seguida o teste será executado novamente. Na **Figura 8** é

apresentado o código original e o código modificado. Repare que a mudança foi muito sutil, mas que é determinante para a resposta correta do sistema.

```
if (aluno.getFrequencia() < 75){
        jTextArea1.append("Aluno Reprovado");
                        :
                        :

if (aluno.getFrequencia() <= 75){
        jTextArea1.append("Aluno Reprovado");
                        :
                        :
```

Figura 8. Modificação da condição

Após a modificação, o próximo passo será executar o script de teste no sistema e verificar o resultado (ver **Figura 9**). Para uma melhor visualização de cada passo que está sendo executado, no menu Test, seleciona-se a opção Slow Playback. Esta opção retarda a execução de cada linha, podendo ser mais fácil a observação de todo o processo.

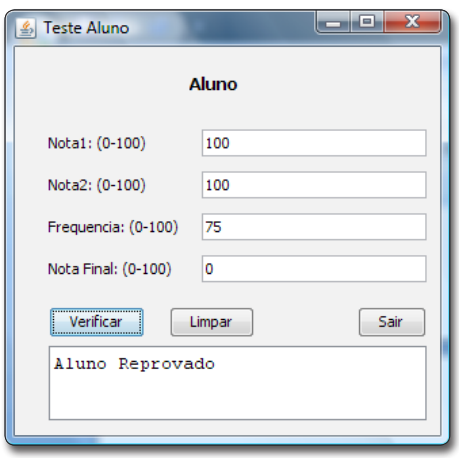


Figura 9. Tela do sistema sendo executado automaticamente.

A imagem mostra que, de acordo com os dados informados, o aluno está Reprovado, contradizendo as regras de negócio citadas anteriormente.

Como já era esperado, após a execução do script foi encontrado um erro, referente à linha de comparação que foi modificada (ver **Figura 10**).

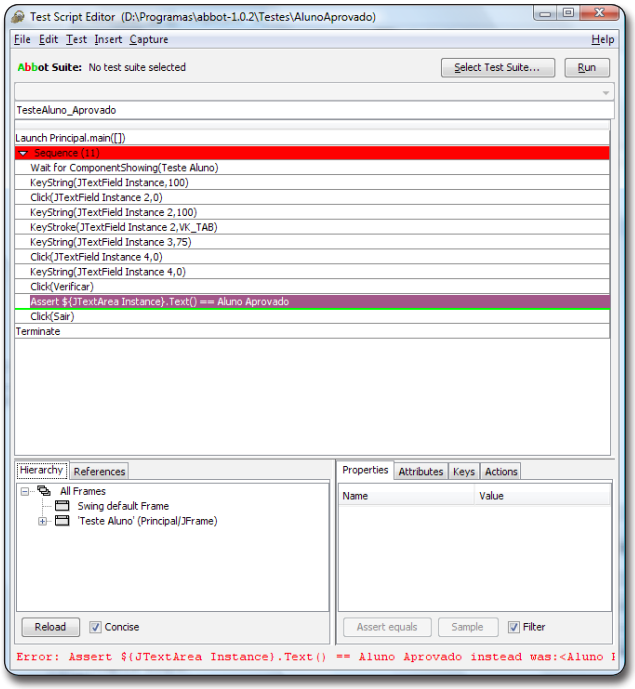


Figura 10. Erro no script

Conclusão

O Abbot Framework se mostra bastante eficaz no que se propõe a fazer, teste funcional de aplicações Java Desktop, sendo de fácil utilização e com muitas opções de configuração. Rapidamente o usuário está familiarizado com a ferramenta e conseguirá personalizar seus testes de acordo com as funcionalidades da aplicação. A realização destes testes aproxima o programador do usuário, na medida em que ele pode prever algumas de suas ações, minimizando falhas e atingindo o objetivo principal da interface que é a interatividade do sistema com quem o utiliza. ●

Interação Humano-Computador

Antonio Alberto Moreira
anjonyck@gmail.com
É Bacharel em Sistemas de Informação pela Faculdade Metodista Granbery.

Eduardo Pagani Julio
epagani@gmail.com
É Mestre em Computação pela Universidade Federal Fluminense e professor dos cursos de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery e do Centro de Ensino Superior de Juiz de Fora.

Alessandreia Marta de Oliveira Julio
alessandreia@gmail.com
É Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ e professora dos cursos de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery e do Centro de Ensino Superior de Juiz de Fora.

De que se trata o artigo?
Este artigo apresenta a Interação Humano-Computador (IHC) no desenvolvimento de sistemas interativos.

Para que serve?
O objetivo é apresentar os conceitos da IHC. Para isto são mostradas algumas definições referentes à interação, à interface e aos usuários, bem como alguns conceitos teóricos como engenharia cognitiva e engenharia semiótica.

Em que situação o tema é útil?
O assunto é importante no desenvolvimento de sistemas interativos, principalmente no que diz respeito aos aspectos relacionados aos fatores humanos.

A necessidade do ser humano em interagir com as máquinas remonta da própria história da humanidade, incentivando-o a desenvolver interfaces cada vez mais sofisticadas para uma melhor facilidade de uso das mesmas.

O surgimento da informática veio fortalecer esta premissa, onde o avanço tecnológico cada vez mais rápido apresenta máquinas que visam satisfazer as necessidades humanas de forma amigável. No entanto, nos primórdios da informática, a complexidade dos computadores e a imaturidade tecnológica impossibilitaram que a comunicação entre o homem e o computador sucedesse de forma fácil.

A redução nos preços dos componentes, em função do amadurecimento tecnológico, foi de extrema relevância para a disseminação da informática, que passou a contar com uma gama de novos usuários com diversos perfis. Neste cenário, surgiu a necessidade de proporcionar a estes usuários uma interação com os computadores que proporcionasse maior facilidade de uso sem muito esforço cognitivo.

O aumento no poder de processamento dos computadores condicionado a evolução tecnológica refletiu no desenvolvimento de interfaces cada vez mais sofisticadas.

O conceito de interface, até então era tido como o conjunto de hardware e software que eram utilizados para se comunicar com o computador. Não obstante, hoje, esta abordagem está obsoleta, e inclui aspectos como processamento perceptual, motor, viso-motor e cognitivo do usuário.

É neste cenário que está inserida a Interação Humano-Computador (IHC), tratada neste artigo e que apresenta estudos sistemáticos e metodológicos e propõe uma abordagem tanto do perfil humano quanto do sistema computacional, e a relação que os interligam.

A IHC é uma subárea da Engenharia de Software (ES) que também propõe modelos de processos, métodos, técnicas e ferramentas para o desenvolvimento de sistemas interativos. Contudo, enquanto o foco da ES é o produto de software e seu processo, a IHC se preocupa mais com os aspectos de interação do homem com a máquina. No entanto, existe a necessidade de se integrar as duas visões no desenvolvimento de sistemas interativos de modo que estes abordem e tratem, de maneira clara e definida, o projeto da interação e da implementação conjuntamente.

Conceituando a Interação Humano-Computador

O processo de interação transpõe o limite da simples percepção física de um sistema, denotando uma atividade dinâmica que ocorre entre duas entidades distintas através da comunicação entre ambas com a finalidade de permutarem informações pertinentes ao contexto a qual estão inseridos.

A **Figura 1** apresenta o contexto da interação humano-computador, que trata de um processo que se estabelece entre os seres humanos e os sistemas computacionais, onde são utilizadas interfaces como agentes que promovem a comunicação entre ambos.

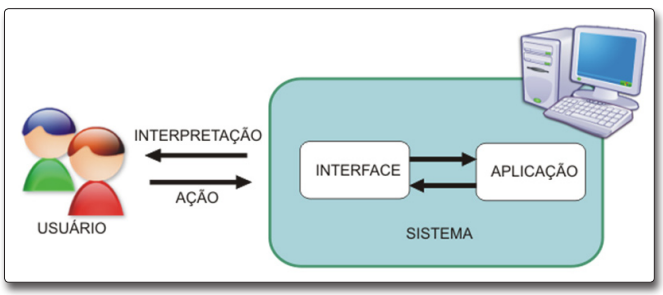


Figura 1. Interação humano-computador

A comunicação entre o homem e o computador é extremamente complexa. Neste aspecto, observa-se que a disciplina IHC inclui o projeto, a avaliação e a implementação de sistemas interativos para uso humano e as operações que os envolvem.

Muitas das vezes o estudo da interação humano-computador tem sido negligenciado por analistas e desenvolvedores de sistemas, tratando-o como uma disciplina que define apenas o layout e a estética das telas dos sistemas computacionais.

Entretanto, de acordo com as diretrizes do MEC, o estudo da IHC abrange não somente os aspectos referentes à interface e a interação, mas também as teorias e as técnicas do projeto de sistemas interativos. Teorias estas que se fundamentam no estudo dos usuários, da tecnologia e a influência que um exerce sobre o outro, dentro do contexto em que está sendo utilizada esta tecnologia. O estudo da interação humano-computador deve procurar conhecer os aspectos relacionados aos fatores humanos, tanto sensoriais quanto cognitivos.

Multidisciplinaridade

O estudo da interação humano-computador apresenta um contexto de multidisciplinaridade. Isto se traduz na condição de que a IHC é alicerçada por uma gama de disciplinas, cada qual fornecendo conhecimentos inerentes à sua área de atuação e auxiliando na composição do projeto e do desenvolvimento do sistema interativo baseado em computador.

Conforme mostra a **Figura 2**, dentre as diversas disciplinas que estão envolvidas com o projeto de desenvolvimento de sistemas interativos, pode-se destacar: psicologia cognitiva, psicologia social e organizacional, ergonomia ou fatores humanos, ciência da computação, inteligência artificial, linguística, filosofia, sociologia, antropologia, engenharia e design.

Figura 2. Disciplinas que contribuem com IHC

Além dos conhecimentos proporcionados pelo inter-relacionamento entre estas disciplinas e a IHC, é imprescindível para a formulação do projeto do sistema interativo considerar o estudo de quatro elementos: o sistema, os usuários, os desenvolvedores e o ambiente de uso, que fazem parte de dois processos importantes: o relacionamento entre usuário-sistema e o desenvolvimento do sistema.

Interface

A interface pode ser definida como parte do aplicativo que o usuário percebe e manipula com o intuito de realizar as tarefas pertinentes ao seu contexto de trabalho. Ou ainda, pode se considerar que a interface é um artefato onde um usuário utiliza os seus recursos para poder se comunicar com o sistema através do contato físico, perceptivo e conceitual.

Contudo, é indispensável observar que o conceito de interface não se restringe somente à parte do aplicativo que viabiliza a comunicação entre o usuário e o sistema. A **Figura 3** apresenta como um conjunto de hardware e software, integrados no ambiente de trabalho, proporciona a interação entre o usuário e o sistema.

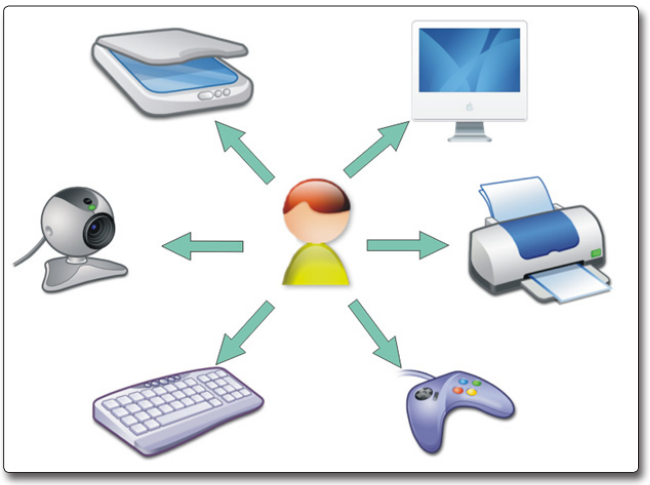


Figura 3. Exemplos de Interfaces

Alguns consideram que a interface é a embalagem do software. Se ela for fácil de aprender, simples de usar, direta e amigável, o usuário estará tendencioso em utilizá-la bem.

Interfaces que apresentam um alto grau de dificuldade na sua forma de utilização tendem a confundir o usuário, conduzindo-o a executar uma série de ações que favorecem as condições que podem corromper os dados ou causarem falhas, degradando o sistema. Este aspecto pode causar um impacto negativo sobre o usuário, considerando que diversos erros podem ser cometidos durante a interação com o sistema, ou até mesmo causar uma recusa ou rejeição na utilização do sistema, independente de sua funcionalidade.

Em algumas aplicações, a satisfação subjetiva do usuário pode ser a chave determinante de sucesso, porém, em outro aspecto, aprendizagens rápidas ou um alto grau de desempenho podem ser extremamente relevantes.

É importante observar que a interface de um sistema interativo, quando bem projetada, define a qualidade e a facilidade de uso do sistema.

Um aspecto importante que deve ser evidenciado é que o projetista de interface não deve se adequar a todas as pessoas de um modo geral, mas sim a um determinado grupo alvo, objetivando assim atingir uma funcionalidade que satisfaça as expectativas e necessidades deste grupo. Não obstante, o que deve impreterivelmente ser entendido, é que aumentar a funcionalidade do sistema não pode ser desculpa para o desenvolvimento de um design pobre.

Metáforas

Existem situações em que os usuários não compreendem claramente o correto funcionamento do software. Para facilitar o entendimento do sistema, sugere-se então o uso das metáforas como forma de indicar ao usuário como a interface se comporta.

As metáforas são analogias estabelecidas com o dia a dia dos usuários, relacionando os artefatos da interface com o ambiente real. Desta forma, torna-se mais fácil e simples para os usuários criarem um modelo mental do sistema, ou seja, o modelo onde ele planeja realizar as tarefas e interpretar as respostas utilizando o sistema.

Neste aspecto, o uso das metáforas tende a favorecer aos usuários no tocante à aplicação dos conhecimentos anteriormente adquiridos, para efetuar uma melhor compreensão do novo domínio. Isto pode ser observado na **Figura 4**, onde é mostrada uma agenda eletrônica que faz alusão a uma agenda de papel.

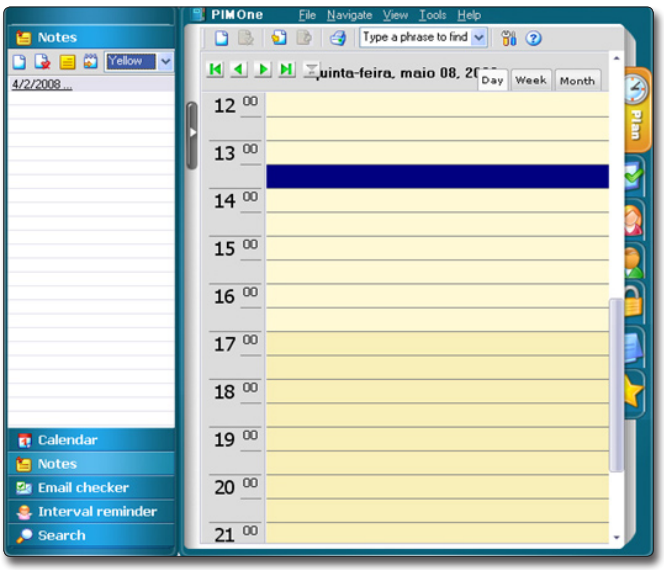


Figura 4. Exemplo de metáfora da agenda PIMone

A utilização de metáforas nas interfaces de sistemas interativos tem provado ser bem sucedida, porém, é imprescindível para o projetista o perfeito entendimento do conceito de metáfora, segundo o qual ele possa projetar e desenvolver uma interface que venha atingir o objetivo desejado.

Entretanto, deve-se enfatizar que é prudente ter cautela na utilização de metáforas, ressaltando que um modelo incoerente causa dificuldades para os usuários e pode ter consequências desagradáveis.

A utilização de metáforas sempre incorre em riscos, e pode resultar em projetos mal planejados, conduzindo os usuários a uma percepção incorreta do sistema. No entanto, é relevante destacar que projetistas prevenidos buscam desenvolver metáforas que combinem o conhecimento familiar com o novo domínio, evitando de forma astuta os problemas que podem ser gerados por uma má formulação de metáforas.

Affordance

Um conceito importante que deve ser considerado no projeto de desenvolvimento de interfaces é a affordance do sistema, e

que se refere às propriedades reais que os usuários percebem, de como um determinado artefato evidencia a sua forma de utilização. Neste contexto, se a affordance de um artefato é percentualmente visível, torna-se simples para o usuário saber utilizar este artefato.

A affordance tem sido muito utilizada no desenvolvimento de sistemas com o intuito de descrever como os artefatos da interface devem ser projetados. Elementos gráficos como botões, ícones, links e barra de rolagens devem ser projetados para parecer evidente a sua forma de utilização.

Na **Figura 5** pode ser observada a affordance de botões de comando, que evidencia a sua forma de utilização.

Em outro aspecto, é imprescindível salientar que uma falsa affordance proporciona efeitos negativos. Isto pode ser observado em objetos que apresentam um determinado comportamento, quando na realidade se comportam de outra maneira.

Um equívoco que os projetistas de interface estão propensos a cometer é o de considerar que o comportamento dos objetos virtuais seja idêntico ao dos objetos físicos. Objetos virtuais não possuem as mesmas propriedades que os objetos físicos, e sendo assim, é necessário que os usuários tenham inicialmente que compreender as convenções estabelecidas de como utilizá-los.

A **Figura 6** exibe figuras em 3D (três dimensões) que, se forem mal utilizadas, deixam a interface saturada, tornando-se um ambiente confuso e complicado, ao passo que o uso de figuras em 2D (duas dimensões) podem assegurar a mesma funcionalidade, além de proporcionarem uma interface limpa e facilmente percebidas pelos usuários.

Entendendo os Usuários

O termo usuário, no contexto de um sistema computacional, adota uma conotação mais ampla, o que se traduz em um conjunto de indivíduos com necessidades, restrições e preferências subjetivas.

Esta característica retrata o usuário como uma abstração, o que faz com que as interações a serem projetadas devam buscar os objetivos fundamentais de suas experiências.

Neste aspecto, é relevante focar o usuário como um elemento referencial no projeto de interação humano-computador de forma que o sistema satisfaça as tarefas a serem realizadas nos papéis que exercem dentro do seu domínio.

Percepção Humana

A percepção humana pode ser considerada não apenas como o ato de ver, mas também a união das informações disponíveis através dos sentidos, juntamente com os conhecimentos retidos na memória. Neste aspecto, pode-se observar que, quando bem alimentada, ela contribui de forma a conduzir os indivíduos a verem claramente as coisas que acontecem no seu cotidiano.

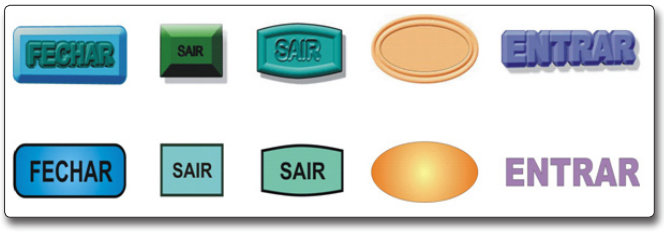


Figura 6. Botões em três e em duas dimensões

Os usuários de sistemas interativos percebem, armazenam e processam as informações através dos sentidos visual, auditivo e tato, utilizando o raciocínio dedutivo ou indutivo. Neste cenário, convém ressaltar que um intenso volume de informações são apresentadas simultaneamente para o usuário absorver, o que evidencia a necessidade de definir uma especificação apropriada de comunicação visual para conduzir a uma interface amigável.

Qualquer som oriundo do computador pode, por exemplo, causar um efeito de distração no usuário, mesmo que este ruído não tenha a pretensão de chamar a atenção.

Memória Humana

A memória humana pode ser traduzida como uma combinação de buffers onde as informações são processadas, e é composta pela memória de curta duração e pela memória de longa duração. As informações captadas pelos sentidos vão para a memória de curta duração (memória de trabalho). Um sistema gerencial define quais informações processadas na memória de curta duração, são armazenadas na memória de longa duração.

Fundamentos Teóricos

As teorias cognitivas sobre IHC propõem uma visão do computador como um instrumento cognitivo que oferece ao ser humano a capacidade de entendimento, memorização e tomada de decisão. Na abordagem semiótica, o computador é visto como uma mídia. Apesar destas abordagens não serem antagônicas, elas representam diferentes visões para a interação humano-computador.

Engenharia Cognitiva

A IHC tem uma abordagem de caráter cognitivo, apresentando pontos em comum com as áreas de psicologia cognitiva,

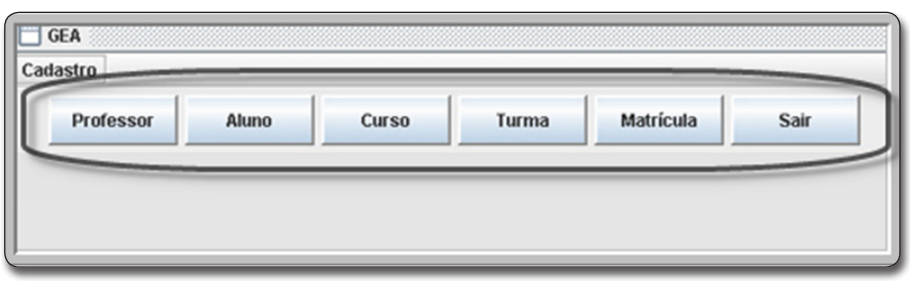


Figura 5. Affordance de um sistema

ciência cognitiva e inteligência artificial, que estudam o processo de aquisição do conhecimento.

Esta abordagem estuda como os seres humanos alcançam seus objetivos através da realização de tarefas cognitivas envolvendo o processamento de informação.

Na visão da engenharia cognitiva, o projetista da interface cria o seu modelo mental, denominado modelo de design. Este modelo tem como referência outros dois modelos denominados: de tarefas e do usuário. O primeiro representa quais as tarefas que os usuários realizam utilizando o sistema, e o segundo representa as características e as necessidades dos usuários. A implementação do modelo de design define a imagem do sistema com a qual o usuário interage. Ao interagir com o sistema, o usuário constrói o seu modelo mental denominado modelo de uso, conforme ilustra a **Figura 7**, e baseado neste modelo, planeja e realiza as tarefas utilizando e interpretando as informações geradas pelo sistema.

A engenharia cognitiva propõe que o projetista de interface desenvolva a imagem do sistema de forma que o seu modelo de design seja o mais compatível possível com o modelo de uso concebido pelo usuário.

Engenharia Semiótica

A engenharia semiótica considera a interface do sistema como uma mensagem que o projetista tenciona enviar ao usuário, representando a maneira de como ele a projetou e para qual finalidade ela foi construída.

A Semiótica Computacional interpretou a interface como um conjunto de signos baseados no computador, ou seja, são todos os artefatos do software que interagem com o usuário.

Do ponto de vista da engenharia semiótica, durante o processo de interação, o usuário e o projetista desempenham o mesmo papel, o de interlocutores recíprocos onde as interfaces são consideradas como objetos de metacomunicação, como mostra a **Figura 8**, que se dividem em dois níveis. No primeiro nível a comunicação é unilateral e integral, do projetista para o usuário, definindo como e porque a interface foi desenvolvida, baseado nas necessidades e expectativas do usuário, seu contexto, gostos, preferências, capacidades e valores. Já no segundo nível, a comunicação se faz do usuário para o sistema. Neste nível, durante o processo de interação, o sistema representa o projetista que transmite a mensagem unilateral e integral para o usuário através da interação entre usuário e sistema.

Desenvolvimento da Interface

O projeto de desenvolvimento da interface engloba as seguintes etapas: análise, design, prototipação e avaliação, sendo que as três últimas fazem parte de um processo iterativo onde, a partir da avaliação, o design é corrigido ou evolui para novas etapas gerando, a cada interação, novos protótipos, conforme pode ser observado na **Figura 9**.

Fatores Humanos ou Ergonomia

O desenvolvimento de um sistema interativo se fundamenta na necessidade do usuário em solucionar questões pertinentes ao seu contexto de trabalho.

No entanto, o desconhecimento por parte do projetista do software, no tocante à tarefa, o modo operatório e a estratégia de resolução de problemas do ser humano,

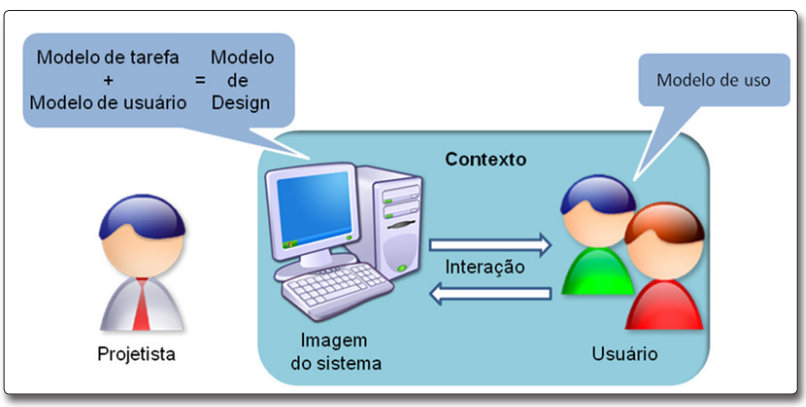


Figura 7. Modelo de Interação na Engenharia Cognitiva

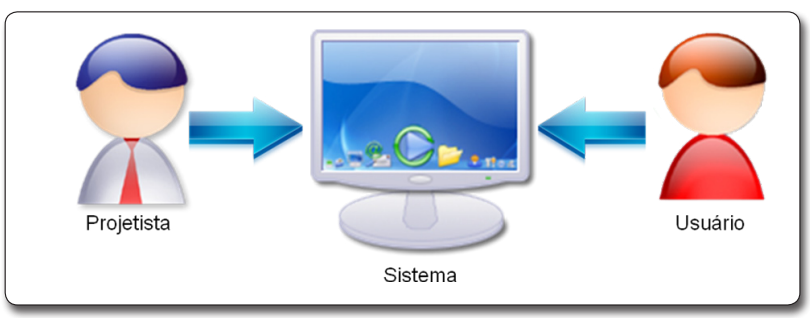


Figura 8. Modelo de Metacomunicação

Figura 9. Desenvolvimento da Interface

geram incompatibilidades no processo de interação humano-computador.

O estudo da ergonomia de software se enquadra neste cenário com o propósito de oferecer ao usuário melhores condições de trabalho, adaptando o sistema a ele, como mostra a **Figura 10**.

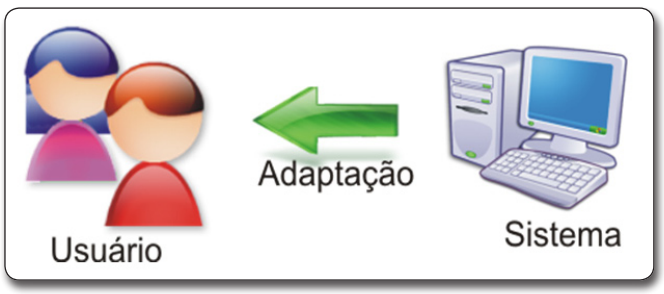


Figura 10. Adaptação do sistema ao Usuário

É importante atentar para a necessidade de analisar os fatores humanos que podem influenciar nas diretrizes que norteiam o projeto de desenvolvimento da interface.

Deve-se considerar a adoção de alguns itens para o desenvolvimento de sistemas interativos como:

• aspectos fundamentais da percepção humana: os sentidos que predominam numa interface humano-computador é o visual, o tátil e o auditivo, que possibilitam ao usuário perceber as informações e armazená-las na memória usando o raciocínio indutivo ou o raciocínio dedutivo;

• nível de habilidade e comportamento humano: cada ser humano é dotado de uma personalidade ímpar. É importante considerar o nível de habilidade individual e as diversas personalidades pertencentes a cada usuário do sistema, buscando conscientizar-se de que nem tudo que é bom para um determinado usuário pode satisfazer e ter o mesmo sentido para outro;

• tarefas e fatores humanos: um sistema raramente permite ao usuário fazer algo inteiramente novo ao contexto do trabalho. Normalmente o sistema automatiza as atividades que até então eram executadas manualmente.

Conclusão

A utilização da IHC no desenvolvimento de sistemas interativos tem sido de tamanha importância para a obtenção de qualidade na interação de forma a satisfazer as necessidades dos usuários proporcionando uma boa usabilidade. Neste contexto, é fundamental absorver os conhecimentos oriundos da IHC para poder aplicá-los ao projeto do sistema interativo. O objetivo deste artigo foi apresentar os conceitos da IHC, abordando questões pertinentes ao relacionamento que ocorre entre o ser humano e o sistema computacional. Para isto foram mostradas algumas definições referentes à interação, à interface e aos usuários. Também foram apresentados alguns conceitos teóricos como engenharia cognitiva e engenharia semiótica que fundamentam o desenvolvimento dos sistemas interativos e as considerações acerca dos fatores humanos, que estabelecem as diretrizes do desenvolvimento dos sistemas interativos. ●

Referências Bibliográficas

OLIVEIRA NETTO, Alvim Antônio de. IHC – Modelagem e Gerência de Interfaces com o Usuário. 2. ed. – Florianópolis: VisualBooks, 2004.

ORTH, Afonso Inácio, Interface Homem-Máquina – Porto Alegre: AIO, 2005.

PREECE, Jennifer; ROGERS, Yvonne; SHARP, Helen. Design de Interação: Além da Interação Homem-Computador – Porto Alegre: Bookman, 2005.

PRESSMAN, Roger S. Engenharia de Software. 6ed., São Paulo: Pearson Education do Brasil, 2006.

ROCHA, Heloísa Vieira da; BARANAUSKAS, Maria Cecília Calani, Design e Avaliação de Interfaces Humano-Computador – Campinas: NIED/UNICAMP, 2003.

SHNEIDERMAN, Ben; PLAISANT, Catherine. Designing The User Interface. 4ed., Addison-Wesley, 2005.

SOMMERVILLE, Ian. Engenharia de Software. 8ed., São Paulo: Addison Wesley, 2007.

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