engenharia de software magazine

Projeto
Integração contínua tornando o trabalho de equipes mais organizado

Projeto
Avalie a usabilidade de suas aplicações web através da avaliação heurística

DevMedia group Ano I - Edição 07

ISSN 1983127-7

MPS.BR

Conheça algumas das práticas mais utilizadas por implementadores para alcançar o nível G

Agilidade
Estimativas utilizando o Ideal Day e Priorização de Atividades do Projeto

Validação, Verificação & Teste
Veja como executar testes de desempenho com o JMeter

Gestão de Conhecimento
Conheça as camadas que geralmente formam um sistema de auxílio à gestão do conhecimento

Agilidade
Conceitos Fundamentais, Vantagens e Desvantagens

Aulas desta edição:

- Introdução à Construção de Diagrama de Classes da UML – Parte 6
- Introdução à Construção de Diagrama de Classes da UML – Parte 7
- Teste de Web Services
- Teste de Desempenho
- Teste de Mutação

Profissional de TI

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

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

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

PRIMEUP

IBM Business Partner

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

Tel: (21) 2512-6005

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

engenharia de software magazine

Ano 1 - 7ª Edição 2008 **Impresso no Brasil**

EDITORIAL

"No inicio da produção de software (1960) não havia se pensado em estruturas formais de desenvolvimento de software. Existia no mercado a necessidade de softwares cada vez maiores e com essas necessidades aumentadas, começaram a se tornar mais constantes os problemas e prejuízos com os atrasos e cancelamentos nos projetos de software. A solução foi a criação da Engenharia de Software que tinha como principais objetivos definir processos, métodos e ferramentas para a produção de software com a qualidade esperada pela indústria. Com o aumento da qualidade foi introduzido um aperfeiçoamento contínuo de métodos, técnicas e artefatos (PRESSMAN, 2006).

Nos dias atuais, com a necessidade de se alcançar maior competitividade e qualidade na construção de software as empresas nacionais sentem-se na obrigação de modificar suas estruturas organizacionais e processos produtivos para que se adaptem ao tamanho e características dos projetos sem perder os padrões de qualidade exigidos tanto nacional, como internacionalmente. Com todas essas necessidades foi criado o MPS.BR , um modelo de maturidade nacional com padrões de qualidade internacionais no qual as empresas se certificam em diferentes níveis de acordo com seu grau de maturidade.

Hoje no Brasil, mais de 100 empresas possuem certificação MPS e estas certificações, mesmo para os níveis menos complexos, são extremamente difíceis."

Neste contexto, a Engenharia de Software Magazine destaca nesta edição uma matéria muito interessante que identifica as práticas mais utilizadas por implementadores MPS para evidenciar as exigências do modelo de maturidade brasileiro, visando auxiliar, orientar e minimizar as dificuldades enfrentadas na sua implementação. Desta forma torna-se possível promover um melhor entendimento por parte das empresas e dos implementadores sobre como evidenciar os resultados exigidos para a certificação do nível G do modelo MPS.

Além desta matéria, esta sétima edição traz mais sete artigos: Metodologias Ágeis para Desenvolvimento de Software; Ideal Day e Priorização: Métodos Ágeis no Planejamento; Ferramentas de Integração Contínua tornando o Trabalho de Equipes mais Organizado; Avaliação Heurística de Web Sites; Teste de Desempenho de Aplicações Web com Apache Jmeter; Introdução à Gestão de Conhecimento – Parte 2; Apoiando a Implementação do Modelo de Maturidade MPS Nível G.

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 quinta edição, temos um conjunto de 5 vídeo aulas. Estas vídeo aulas estão disponíveis para download no Portal da Engenharia de Software Magazine e certamente trarão uma significativa contribuição para seu aprendizado. A lista de aulas publicadas pode ser vista ao lado:

Tipo: Verificação, Validação e Teste
Título: Teste de Web Services
Autor: Marco Antônio Pereira Araújo
Mini-Resumo: Esta vídeo aula apresenta a implementação de Testes de Web Services utilizando a ferramenta SoapUI. Através da aplicação de testes unitários em Web Services, é possível proporcionar maior qualidade neste tipo de tecnologia, cada vez mais utilizada.

Tipo: Verificação, Validação e Teste
Título: Teste de Desempenho
Autor: Marco Antônio Pereira Araújo
Mini-Resumo: Esta vídeo aula apresenta a implementação de Testes de Desempenho utilizando a ferramenta Apache JMeter, proporcionando verificar se o desempenho de aplicações Web está de acordo com o esperado pelos usuários e desenvolvedores.

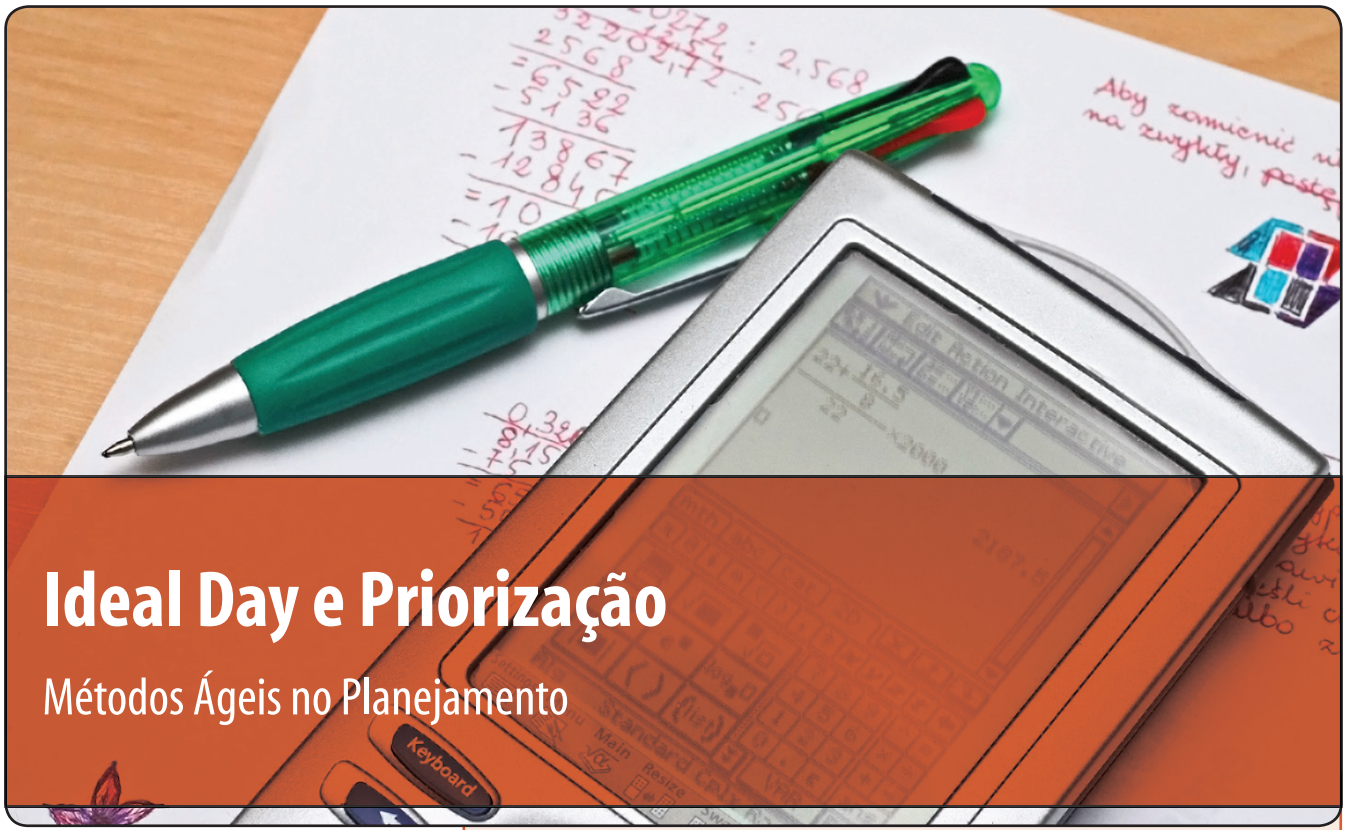
Tipo: Verificação, Validação e Teste
Título: Teste de Mutação
Autor: Marco Antônio Pereira Araújo
Mini-Resumo: Esta vídeo aula apresenta a implementação de Testes de Mutação utilizando a ferramenta MuClipse. Testes de mutação são considerados uma técnica que permite melhorar a qualidade dos casos de testes de uma aplicação.

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 6
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula apresenta o uso da heurística de apoio à elaboração de diagramas de classes na prática através de um estudo de caso para um sistema de locadora de veículos.

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 7
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula complementa a parte 6 do curso introdutório à construção de diagrama de classes da UML. Para isto, é apresentada a segunda parte do uso da heurística de apoio à elaboração de diagramas de classes através de um estudo de caso para um sistema de locadora de veículos.

ÍNDICE

Você não está mais sozinho.



Ideal Day e Priorização

Métodos Ágeis no Planejamento

De que o artigo se trata?

Neste artigo, trataremos sobre o método de estimativa Ideal Day (ID) e uma forma de priorização de trabalho dos requisitos relativos a um projeto (Release), baseando-nos em estudos e aplicação na área de Pesquisa e Desenvolvimento da Powerlogic Consultoria e Sistemas. As estimativas de tamanho de cada funcionalidade/requisito serão apresentadas utilizando o método acima, criando indicadores para seu gerenciamento e acompanhamento e ainda servindo para realimentar ciclos (Sprints) futuros com dados estatísticos.

Para que serve?

Para realizar estimativas de forma Ágil.

Quais situações utilizam esses recursos?

Ao planejar um projeto (release) e seus ciclos (sprints). No mundo ágil, estamos continuamente planejando e não somente em marcos pré-definidos, como início de um projeto, por exemplo. Por isso, este tema se torna um importante aliado para o gerenciamento e acompanhamento mais assertivo de projetos em geral.

Fernanda Alves

fernanda.alves @powerlogic.com.br

Consultora em gerenciamento de projetos (APM e EPM) e processos. Atua em levantamento, análise, desenho e descrição de processos corporativos para modelos de melhoria de processos utilizando metodologias diversas. Integrante da equipe de SEPG (Software Engineering Process Group) da Powerlogic Consultoria e Sistemas.

Márcia Alves

marcia.alves@powerlogic.com.br

Graduanda em Administração de empresas com habilitação em Marketing, pelo Unicentro Newton Paiva, é consultora em Gerenciamento de Processos. Atualmente é Gerente de Projetos para implantação de melhorias - para alcance do nível de maturidade C pelo modelo MPS.BR (Melhoria de Processos do Software Brasileiro) utilizando metodologia SCRUM. Integrante da equipe de SEPG (Software Engineering Process Group) da Powerlogic Consultoria e Sistemas.

Isabella Fonseca

isabella@powerlogic.com.br

Atua desde 1999 como consultora em e-Business para grandes empresas utilizando Java EE. É Certified ScrumMaster e gerente da equipe de desenvolvimento e de projetos do eCompany Portal - primeira solução de EIP (Enterprise Information Portal) do país, e do eCompany Process - solução de definição e controle de Processos Corporativos integrada ao Gerenciamento de Projetos (EPM e APM), de Requisitos e de Produtos (Aplication Lifecycle Management). Ambos são gerenciados com SCRUM e certificados MPS.Br nível F. Integrante da equipe de SEPG (Software Engineering Process Group) da Powerlogic Consultoria e Sistemas.

O que é Ideal Day?

Imagine uma parede de 1mx1m. Independente de quem a concebeu, ela continuará tendo o mesmo tamanho. Por outro lado, seu tempo de entrega, variará de acordo com o executor designado para tal. Ideal Day funciona da mesma maneira.

Um Ideal Day corresponde à quantidade de trabalho que um profissional de nível sênior, com fluência nas tecnologias e ferramentas envolvidas (Ideal Developer) consegue realizar em 08 (oito) horas de trabalho dedicadas (sem interrupções).

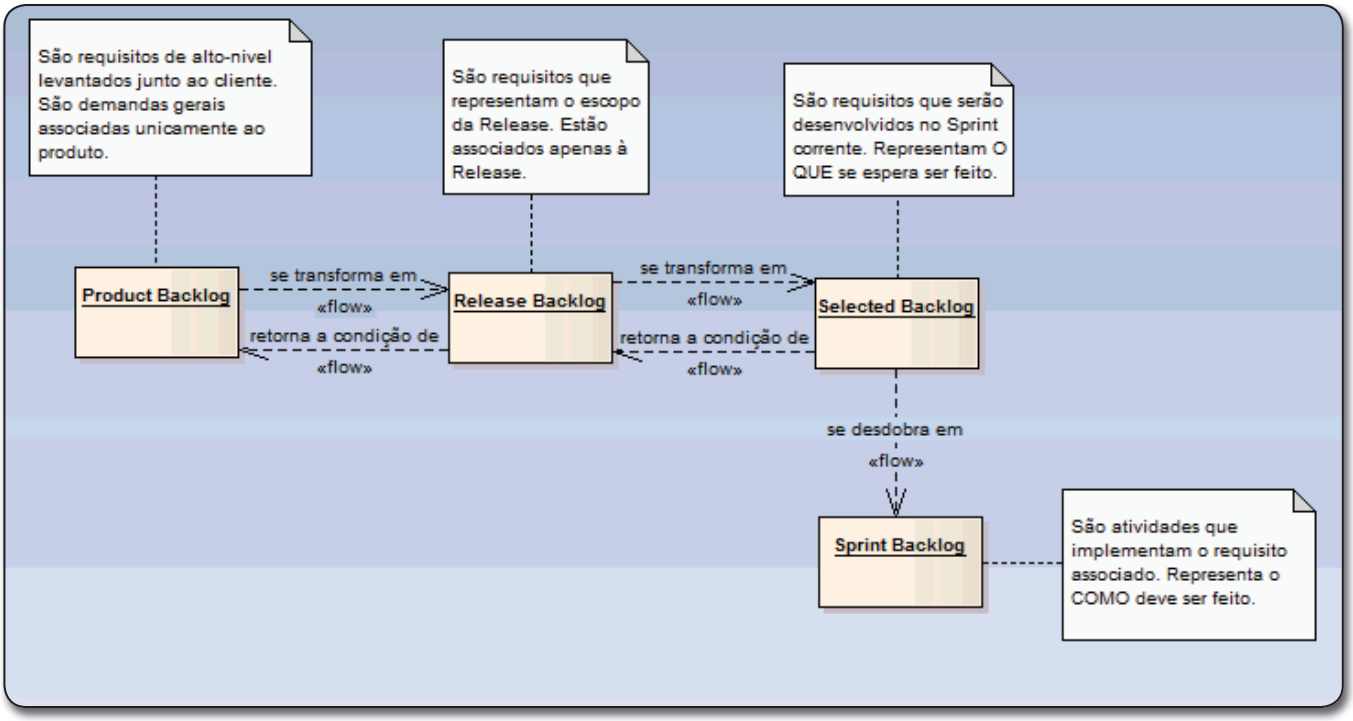


Figura 1. Ciclo de vida.

É importante que se compreenda que o "Dia Ideal", com 08 (oito) horas de trabalho sem interrupções, de um "desenvolvedor ideal", raramente irá ocorrer na prática, e portanto deve ser utilizado unicamente como "moeda" estável para quantificação de tamanho de referência e balizador ideal de produtividade.

É uma estimativa empírica, executad por especialistas ("Expert Judment") para desenvolvimento com base em "exploração adaptativa". Segundo estudos mais recentes da escola ágil, a estimativa empírica é uma maneira sensata de se prever o tamanho de requisitos em uma dinâmica de "requisitos evolucionários", com práticas de "exploração e adaptação", especialmente se acompanhada por:

- Realimentação iterativa da "velocidade", a partir de dados históricos, preferencialmente coletados durante o mesmo projeto para a mesma equipe;
- Previsão sobre uma mesma "ordem de grandeza", neste caso que não ultrapasse o espaço de algumas horas para alguns poucos dias;
- Realização de consenso entre especialistas, com técnicas de comunicação e convergência como a do Pocker Planning;
- Utilização da técnica de PERT (Program Evaluation and Review Technique);
- Utilização de balanceamento como a técnica Cocomo (COnstructive COst Model).

Irão contribuir para que um "Ideal Day" não aconteça, na prática, em um dia típico:

- Natureza humana do desenvolvedor (comer, beber, alongar, socializar, sono, mal-estar eventual, etc);
- Deficiências técnicas do desenvolvedor (desconhecimentos do assunto ou tecnologia específicos);
- Interrupções da empresa (reuniões administrativas, conversa com o 'chefe', ligações de clientes);
- Interrupções pessoais.
- etc...

Dessa forma, a equipe deve ter sua velocidade medida pelo tempo gasto para se implementar um Ideal Day. Quanto menos tempo, maior a velocidade, e maior a produtividade da mesma. Na realidade, não é importante conhecer a velocidade individual e sim a média da equipe. Para se manter uma unidade, não é interessante expor se um integrante executa suas atividades em mais ou menos tempo. É uma dinâmica do grupo! Ele deve aprender como interagir melhor para a busca de entrega de maior retorno de valor para o cliente. Se for necessário utilizar de técnicas como pair-programming para agilizar o desenvolvimento e validação de um requisito, o time deve escolher este caminho. Também se pode utilizar peer-review para verificações e validações, assumir outro papel (trocar de "chapéu" dentro da equipe) para convergir para o objetivo definido.

Ideal Day e SCRUM

O SCRUM é um framework de processo ágil utilizado para gerenciar e controlar o desenvolvimento de um produto de software através de práticas iterativas e incrementais. É composto por um conjunto de boas práticas de gestão que admite ajustes rápidos, acompanhamento e visibilidade constantes e planos realísticos. Por ter ciclos (Sprints) curtos, sua natureza leva à utilização de requisitos de granularidade pequena, aplicando o conceito de pilha – não há alocação prévia de recurso - desconsiderando a velocidade individual e, portanto, favorecendo o uso de métodos de estimativa, como Ideal Day.

Seus artefatos principais são o Product Backlog e Sprint Backlog – artefatos que representam seus requisitos/atividades

além de Burndown charts e impediment backlogs. Para representar o ciclo de vida de um requisito, usa-se derivações do Product Backlog, como Release Backlog e Selected Backlog. A **Figura 1** exemplifica o ciclo mencionado.

O método de estimativa utilizado para o Selected Backlog reúne técnicas modernas voltadas à estimativa de requisitos em projetos cujo índice de imprevisibilidade é notadamente alto, seja por envolverem tecnologias de ponta, inovações que requerem desenvolvimento "iterativo", baseado em "exploração e adaptação", ou mesmo ineditismo do que se está por produzir - ou seja, previsão para 'criação de novo produto', e não para 'manufatura de mais do mesmo produto'.

Realimentação iterativa da "velocidade"

A velocidade deve ser reajustada, a cada final de Sprint, para um ajuste apropriado que maximize a previsibilidade. Quanto maior o tamanho de um requisito a ser estimado, mais aumentam as possibilidades de desvios, tanto relativas (maior percentual de erro), quanto absolutas (maior tamanho do erro, para um mesmo percentual). Por isso, em um processo empírico gerenciado deve-se "decompor" itens do Selected Backlog que ultrapassem 01 Ideal Day em itens do Sprint Backlog, rastreáveis entre si, mas de implementação independente. Estes itens podem ser chamados no XP de "estórias do usuário" ou similares a "quebrar cenários de Caso de Uso" no Processo Unificado.

Portanto, a velocidade é calculada através do número de horas que a equipe como um todo gasta para implementar um trabalho equivalente a 01 (um) Ideal Day.

A reunião de planejamento de uma Release ou Sprint deve contemplar a estimativa dos requisitos por meio da técnica de Poker Planning.

A prática do Poker Planning é a seguinte:

1. Todos devem possuir "cartas" contendo os intervalos discretos de previsão, e mais uma que representa o estouro, contendo "?", para indicar o julgamento do item do "Selected Backlog acima da Ordem de Grandeza", ou seja, caso um membro do time acredite que o item do Selected Backlog em votação possua um valor maior do que aqueles apresentados nas cartas.

2. O Selected Backlog deve ser lido e discutido por todos, e em seguida, sem comentarem seus votos, todos devem apresentar as cartas com a previsão que julgam de maior aproximação

3. Caso não haja uma convergência óbvia (média aproximada entre todos), ou caso alguém apresente o "?", deve-se rediscutir o requisito, principalmente ouvindo-se os argumentos daqueles que votaram com maior desvio.

Em função da discussão, pode-se:

3.1 Melhorar a especificação do item do Selected Backlog.

3.2. Decompor o item do Selected Backlog, mantendo a rastreabilidade.

3.3. Simplesmente prestar-se mais esclarecimentos aos votantes

Por fim, deve-se proceder com uma nova votação, e retornar ao passo 3, até que todos entrem em um consenso sobre o valor.

A previsão deve ser realizada em escalas discretas, entre 1/8, 1/4, 1/2 e 1 Ideal Day. Não há necessidade de se expressar refinamentos nesta escala, pois a margem de erro é considerada maior que um possível refinamento da aproximação.

Outro ajuste importante para aprimoramento de previsão empírica é o uso por todos os envolvidos de previsões dos limites, e não da ideal. Ou seja, a estimativa é realizada a partir das seguintes questões:

1 - Qual o tamanho mínimo em Ideal Days do item do Selected Backlog considerando-se, por exemplo, um baixo impacto ou instabilidade provocados pela sua implementação? – Melhor Caso.

2 - Qual o tamanho máximo em Ideal Days do Selected Backlog considerando-se um maior impacto e instabilidade? – Pior Caso.

Este tipo de análise contribui para a reflexão sobre situações limítrofes para pior ou para melhor, deste modo aprimorando o resultado ideal.

Aliado a isso, técnicas como PERT abordam uma terceira medida, mais provável, que garantem estatisticamente maior assertividade nas estimativas executadas. Portanto, consideramos como melhor prática o alinhamento entre Ideal Day e PERT. Ainda, é aconselhável utilizar matrizes de rastreabilidade de requisitos para que se tenha uma dimensão do impacto (número e artefatos afetados) do Selected Backlog a cada estimativa ou alteração no requisito.

Etapas de priorização da pilha de Product Backlog utilizando Business Value

Business Value (BV) ou Valor de Negócio reflete a importância estratégica de uma funcionalidade do produto para o mercado. O responsável pelo produto deve manter a lista de Product Backlog constantemente atualizada e avaliada com seu business value atribuído, podendo ser revisado a qualquer momento. Esta prática faz parte da primeira etapa de priorização, que permite ao responsável classificar os itens de maior valor para o mercado e obter maior retorno para o negócio sobre o investimento. Ao iniciar uma Release, será possível identificar e selecionar os possíveis itens do Product Backlog que farão parte do escopo a ser desenvolvido. É interessante estipular um valor limite para a distribuição entre os itens da pilha. Cada funcionalidade deve ter seu valor compreendido na escala estabelecida pela organização, por exemplo, de 0 a 100.

Outra variável que deve ser inserida na fórmula de priorização dos itens que irão compor o escopo da Release (Release/Selected Backlog) é a facilidade de implementação do requisito, executando assim a segunda etapa de priorização. Quanto maior a facilidade, menor deve ser seu esforço de implementação. A **Figura 2** exibe um gráfico de quatro quadrantes de itens do escopo que foram classificados em função seu BV e sua facilidade de implementação. Itens que se concentram no quadrante superior direito são os que devem ser priorizados e, seguramente, são os que mais representam a maximização do resultado.

As coordenadas do gráfico seguem os valores de BV determinados pelo responsável do produto e a facilidade de implementação estimada em consenso pela equipe de desenvolvimento. O tamanho de cada bolha representa proporção de valor de negócio de cada item considerando a lista em que ele está inserido. Caso seja necessário, cabe a cada organização determinar um diferente "peso" para esta representação.

A última etapa de refinamento da priorização de requisitos leva em consideração o tamanho do requisito estimado pela equipe de desenvolvimento. Ela organiza a ordem de execução dos requisitos da pilha selecionados na segunda etapa e norteia a equipe de desenvolvimento. Segundo o SCRUM, a própria equipe tem autonomia para tal atividade e esta prática garante o comprometimento dos envolvidos para o alcance dos objetivos.

Dado isso, a fórmula final deve ser a seguinte:

Priorização final da pilha = BV/ Tamanho do requisito

Em caso de empate – dois requisitos com mesmo valor final de priorização – deve-se considerar primeiramente seu BV como critério de desempate e, como segundo critério, a criticidade sendo:

1 – Baixa
3 – Normal
5 – Alta
7 – Urgência
9 – Emergência

A Fórmula de priorização para desempate deve ser representada como (BV/

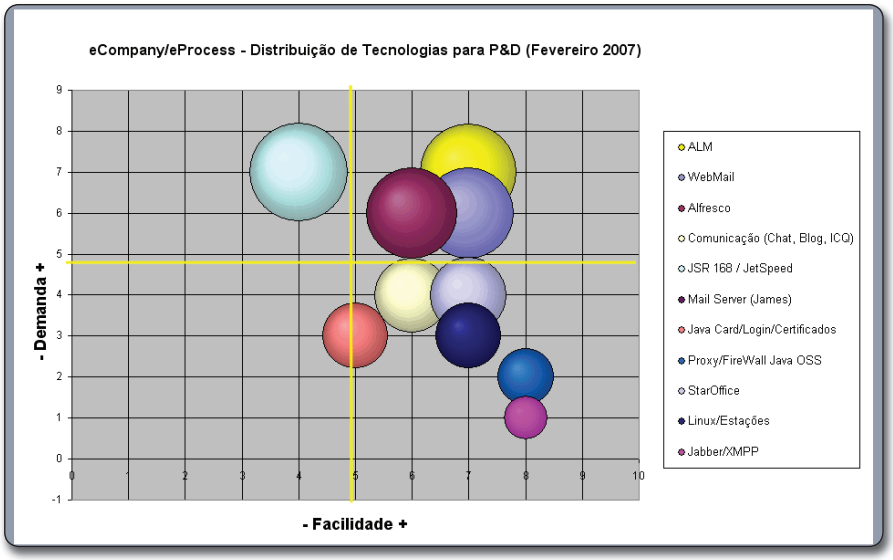


Figura 2. Classificação dos itens de escopo de acordo com seus valores de negócio.

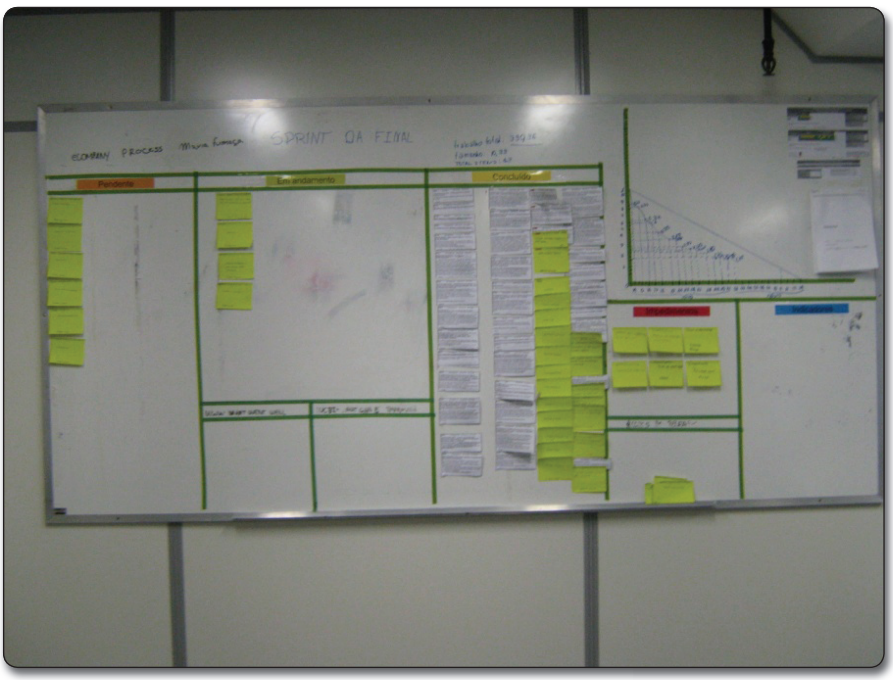


Figura 3. Acompanhamento do projeto.

Requisito	Tamanho Previsto	Recurso	Horas real
RF01	0,5 ID	Recurso 1	4,5 horas
RF02	0,3 ID	Recurso 2	2,5 horas
RF03	0,1 ID	Recurso 1	1,5 horas
RF04	0,4 ID	Recurso 2	3 horas
TOTAL:	1,3 ID		11,5 horas

Tabela 1. Dados apurados.

	Velocidade inicial	ID previsto	ID realizado	Horas real
Recurso 1	10	0,6	0,6	6
Recurso 2	10	0,7	0,7	5,5

Tabela 2. Dados consolidados

Tamanho do requisito) * criticidade.

Outro ponto a ser destacado e comum em projetos de manutenção de produtos é a presença de erros que deverão entrar na pilha do Product Backlog. Pode-se dar um "peso" e modificar a fórmula acima para contemplar este cenário. Uma vez que BV diz respeito a valor de negócio que a inclusão da funcionalidade irá prover para o mercado e erro não acrescenta valor ao produto, sugerimos determinar BV negativo para este caso. Para que este requisito entre corretamente na priorização da pilha, deve-se aplicar seu valor absoluto.

A partir deste resultado, a equipe de desenvolvimento consegue determinar prazos para cada entrega que contempla o escopo acordado. O número de Sprints de implementação necessários é então comunicado ao responsável pelo produto. A partir da definição de sua restrição – escopo, tempo, custo ou qualidade - ele tomará uma decisão e definirá o objetivo maior da Release e do primeiro Sprint. O SCRUM prega o planejamento contínuo, portanto, as demais iterações serão planejadas oportunamente.

Após esta definição, a equipe de desenvolvimento consegue implementar os requisitos seguindo, rigorosamente, do topo para baixo da pilha.

Em sistema enxutos (Lean) de "push", como o Scrum, resolve-se o problema de dependências de "caminho crítico" simplesmente ajustando-se a ordem de execução dos itens de Product Backlog: os que dependem são colocados abaixo de suas dependências.

A **Figura 3** exemplifica um Agile Radiator monitorando um projeto real. Eles garantem visibilidade do projeto a todos os envolvidos. Não há como mascarar o real andamento. O goal fica afixado e os requisitos – através de Post-its - (Selected backlogs) e seus desdobramentos (Sprint backlogs) são posicionados na situação onde se encontram (se ainda não iniciados - planejados, se sendo executados no momento - em andamento e se terminados – 100% concluídos). Eles devem ser posicionados de acordo com a prioridade dos mesmos – Business Value declarado pelo Product Owner. Post-its localizados no topo nos dizem ser de maior prioridade que os posicionados no rodapé do quadro branco.

Para entendermos na prática como aplicar as técnicas apresentadas, será apresentado a partir de agora um caso prático.

Caso prático

Considerando que a jornada de trabalho de sua equipe seja de 4 horas diárias, você deverá estipular seu valor de referência, ou seja, 1 Ideal Day = 4 horas e ainda determinar a velocidade média inicial de sua equipe, por exemplo, ID = 10 horas. É importante entendermos que 4 horas corresponde a horas trabalhadas por um especialista em um "dia perfeito"; e 10 horas a média da equipe estipulada, inicialmente e empiricamente, por especialistas – este valor será reavaliado e as demais medidas serão baseadas em dados históricos.

1) Segue abaixo uma lista resumida de itens do Selected Backlog já estimados via Ideal Day durante o Poker Planning:

a. RF01 – Implementar carrinho de compras – 0,5 ID

b. RF02 - Cadastrar livros – 0,3 ID

c. RF03 – Consultar livros por autor– 0,1 ID

d. RF04 - Implementar recomendação automática de livros – 0,4 ID

2) Pelo conceito de pilha, cada membro deve retirar uma tarefa para executar e apropriar as horas gastas ao final do dia. Este procedimento é necessário para que, ao final do Sprint, seja possível determinar quantos Ideal Days foram concluídos e o seu tempo de execução.

Note que o tamanho de um requisito nunca muda! O que muda é o esforço do mesmo, que depende do recurso alocado - velocidade.

O recurso 1 irá implementar o RF01, que tem 0,5 ID de tamanho. Se é a primeira vez e não tenho dados históricos para determinar o esforço do recurso, vou utilizar a velocidade inicial determinada empiricamente de 10 horas.

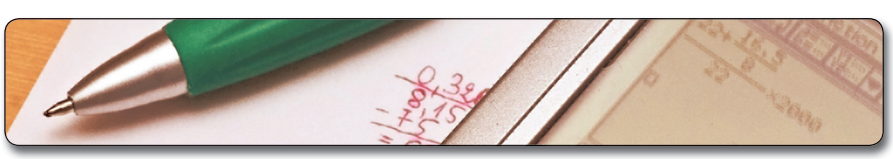
Por isso, o requisito terá duração prevista de 5 horas (5 horas corresponde à metade de um ideal Day cujo valor estimado de velocidade foi de 10 para a equipe):

Esforço = tamanho x velocidade

Esforço = 0,5 x 10 = 5 horas

De acordo com o número de horas que a pessoa trabalha (se ele trabalha somente pela manhã), será necessário "quebrar" este requisito para que seja possível concluí-lo em 1 dia de trabalho e mover o post-it de "em andamento" para "finalizado" evidenciando a evolução do trabalho.

Desse modo, atualizando o gráfico Burndown (representado na **Figura 3** - quadrante superior direito) através da subtração do tamanho (0,5 ID) de Ideal Days realizados, será possível acompanhar desvios entre previsto e realizado de maneira efetiva e visual.



Com a apropriação do Sprint, obtém-se dados para calcular a velocidade padrão do grupo. A **Tabela 1** exemplifica os dados apurados após implementação.

De acordo com a **Tabela 1**, o recurso 1 implementou os requisitos RF01 e RF03, totalizando entrega de 0,6 ID. Já o recurso 2, os RF02 e RF04, totalizando 0,7 ID. Para calcular a nova velocidade, após o Sprint, devemos considerar os dados da **Tabela 2**.

Para concluirmos o cálculo da velocidade é preciso ainda considerar o tempo de retrabalho, ou seja, o tempo gasto em correções que terão um peso maior no cálculo da velocidade da equipe.

Para isso, temos que a Fórmula para cálculo da velocidade é:

horas_realizadas + (horas_retrabalho * 1,3) / ID realizados

Com isso, tempos que:

Recurso 1 = 6 + 0 / 0,6 = 10 horas

Recurso 2 = 5,5 + 0 / 0,7 = 7,8 horas

Média da equipe = 8,9 horas.

No próximo Sprint, a média a ser considerada será a calculada no Sprint anterior (8,9 horas) e não mais empiricamente (10 horas).

A produtividade é extraída da avaliação do número de Ideal Days/Sprint. No primeiro Sprint, a produtividade é igual ao número de Ideal Days entregues.

Produtividade da equipe = 1,3 ID

Considerando a produtividade da equipe, no próximo Sprint será possível alocar itens do Selected Backlog que totalizem o número de Ideal Days entregues no Sprint anterior – no nosso exemplo, 1,3 ID. Ao final do mesmo, deve-se apurar novamente este número e fazer a média entre Sprints, atualizando sempre esta informação.

Você deverá medir novamente e obter mais dados históricos para conhecer a média de produtividade e de velocidade da equipe.

A produtividade da Release pode ser calculada como:

Produtividade da Release = ID Realizados/ Número de Sprints.

De acordo com o exemplo anterior, durante o sprint 1 a equipe manteve a produtividade planejada de 1,3 ID. Caso, a release tenha 3 sprints e a equipe entregar 0,9 ID no segundo e 1,1 ID no terceiro, a produtividade média da release será: 1,3+0,9+1,1/3 = 1,1 ID.

Conclusão

A partir de experiências vividas em nossa área, percebemos que planejar continuamente traz grandes benefícios tanto para o desenvolvimento do produto quanto para seus clientes. A entrega de maior valor de negócio é assegurada e ela acontece mais rapidamente. Além disso, utilizando medidas de tamanho, como Ideal Day, o planejamento se baseia na equipe e não em indivíduos, possibilitando o trabalho organizado em "pilha" de requisitos e atividades. Obviamente, a aplicação deste conceito requer planejamento e preparação da organização além da capacitação de todos os envolvidos para que haja a multidisciplinaridade necessária a este tipo de desenvolvimento.

Você pode estar se perguntando se sua equipe possui este perfil de homogeneidade. Provavelmente ainda não. Para que os resultados sejam positivos, é preciso que a organização apóie e invista neste modelo. Em um primeiro momento, a velocidade da equipe pode ficar comprometida, pois ela ainda não trabalha da mesma forma, tem-se perfis muito diferentes, experiências e conhecimentos diversificados. Mas as iterações vão garantindo compartilhamento global, aprendizado contínuo e tácito, além de evitar o famoso "dono do código" – que no possível desligamento da empresa, leva consigo toda a história do desenvolvimento.

Por tudo isso, o SCRUM não é milagreiro. Quem contribui, em muito, para o sucesso de um projeto são as pessoas envolvidas: sejam desenvolvedores, gerentes, o corpo diretor ou clientes. Todos devem estar cientes do porquê utilizar as técnicas citadas neste artigo: pair programming, peer review, comunicação tácita, compartilhamento de código, entrega de maior valor de negócio, planejamento contínuo, etc. Estas são somente algumas práticas que você pode aplicar para complementar seu dia-a-dia no gerenciamento de projetos. ●

Referências

Processo de Desenvolvimento de Software da Powerlogic Consultoria e Sistemas, versão PDS_P&D_v19.

Agile Estimating and Planning, Prentice Hall, 2006, Cohn , M.

Agile Development Blog / SCRUM, *http://www.targetprocess.com/blog/2004/12/iteration-velocitys-and-user-story.html*, acessado em 01/09/2008.

Mike Cohn's Blog *http://blog.mountaingoatsoftware.com/?p=15*, acessado em 12/09/2008.

Artigo:"Por que SCRUM?",ESM, 4ª. ed. (Agosto 2008) escrito por Fonseca, I. e Campos, A.

Metodologias Ágeis para Desenvolvimento de Software

Michel dos Santos Soares
Mics.soares@gmail.com
Graduado (2000 - UFSCar) e Mestre (2004 - UFU) em Ciência da Computação, atualmente (desde 2006) trabalhando como pesquisador-doutorando (PhD Researcher) na Delft University of Technology, em Delft, na Holanda. Atuou como Analista de Sistemas para sistemas Web e bancários, e como Professor Universitário no Brasil, nas áreas de Programação, Engenharia de Software e Qualidade de Software. Atualmente desenvolve pesquisas na área de sistemas que usam intensamente software (Software Intensive Systems) para o controle de infra-estruturas críticas, aplicando métodos formais e semi-formais de Engenharia de Sistemas e de Software. É co-autor de um livro de Qualidade de Software, pela Novatec, atualmente na 2.a edição, publicou diversos capítulos de livros e artigos científicos em congressos e revistas científicas nacionais e internacionais.

De que se trata o artigo:
Este artigo tem como objetivo descrever as metodologias ágeis em geral e suas práticas comuns, mostrando algumas vantagens, limitaçoes e desvantagens, e comparando com as metodologias tradicionais.

Para que serve:
O artigo compara as metodologias tradicionais e ágeis e sugere quando usar cada diferente tipo de metodologia.

Em que situação o tema é útil:
O tema é útil para empresas e profissionais que não utilizam nenhuma metodologia e têm experimentado problemas no desenvolvimento de software, ou que queiram descobrir qual tipo de metodologia é mais adequado para sua situação.

Este texto é sobre metodologias ágeis para desenvolvimento de software. Mas se existem metodologias ágeis, significa que existem também as consideradas não-ágeis, também conhecidas como pesadas ou tradicionais. Primeiramente, vamos definir o que é uma metodologia de desenvolvimento de software. Posteriormente, serão apresentadas as principais características das metodologias tradicionais e ágeis, além de exemplos das respectivas metodologias.

Metodologia (ou processo) de desenvolvimento de software

Para desenvolver software é necessário um conjunto de atividades e tarefas que resultem, após sua aplicação sistemática, em software. A esse conjunto de tarefas e atividades, mais ou menos organizadas e sistematizadas, damos o nome de metodologia (ou processo) de desenvolvimento de software. Dependendo da

metodologia, existe mais ou menos documentação produzida, maior ou menor iteratividade (desenvolvimento em ciclos) e interatividade com o cliente. Independente da metodologia, existem algumas atividades comuns em desenvolvimento de software [Sommerville, 2008]:

Especificação: definição das funcionalidades, restrições e demais características do produto.

Projeto e implementação: o software é produzido de acordo com as especificações. Nesta fase são propostos modelos por meio de diagramas que serão implementados em alguma linguagem de programação.

Validação: atividades de revisão e testes visando assegurar que os requisitos sejam cumpridos.

Evolução: atividades de manutenção, por exemplo, para adaptar o software a novas necessidades do cliente.

Muitas organizações desenvolvem software sem usar nenhum processo. Geralmente isso ocorre porque os processos tradicionais não são adequados às suas realidades. Em particular, as pequenas e médias organizações não possuem recursos suficientes para adotar o uso de metodologias pesadas e, por essa razão, normalmente não utilizam processos definidos e sistematizados. O resultado dessa falta de sistematização na produção de software é a baixa qualidade do produto final, além de dificultar a entrega do software nos prazos e custos predefinidos e inviabilizar a futura evolução do software. A metodologia Cascata é apresentada brevemente a seguir com o objetivo de compará-la com as metodologias ágeis.

Metodologia (ou processo) de desenvolvimento de software

Um exemplo clássico de metodologias tradicionais é o modelo em Cascata, conhecido também como Clássico ou Waterfall [Royce, 1970]. Curiosamente, atribui-se a Royce um defensor ou introdutor do modelo em Cascata. Este é um equívoco comum. Ele apenas o descreveu, num artigo em 1970, e já naquela época identificou problemas no processo. No artigo, ele caracteriza o processo como "... apresenta riscos e é um convite às falhas".

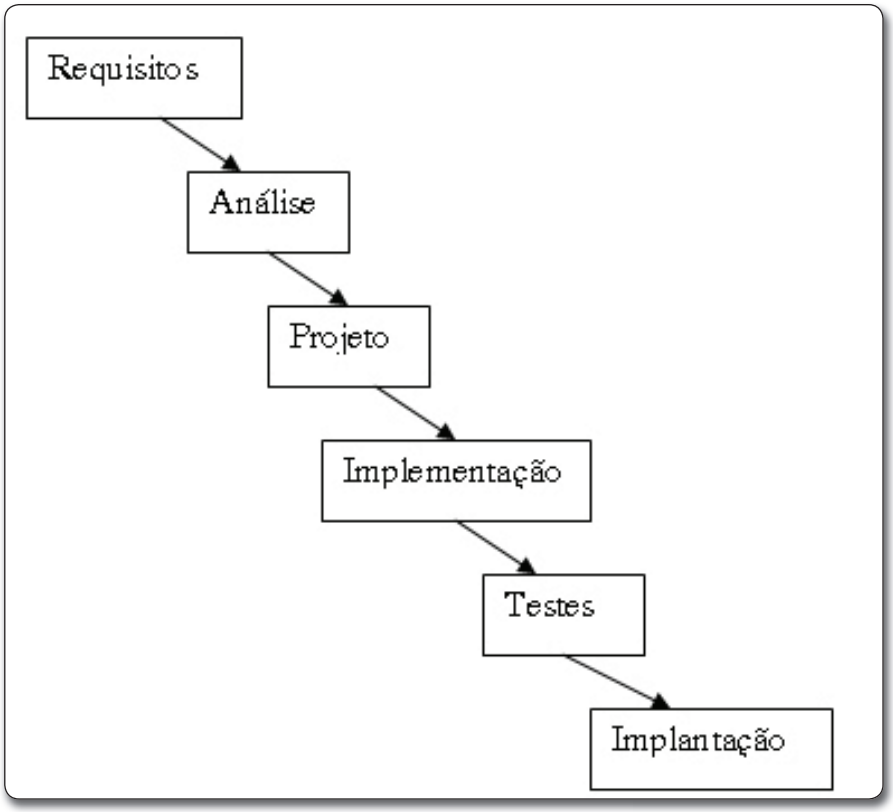


Figura 1. O Modelo em Cascata.

O modelo Cascata (**Figura 1)** é baseado em fases bem definidas, com entradas e saídas para as próximas fases, de forma seqüencial. Cada etapa tem associada ao seu término uma documentação que deve ser aprovada para que a etapa posterior possa ter início. Existem diversas representações semelhantes, mas basicamente o processo consiste nas seguintes fases: Requisitos, Análise, Design (Projeto), Implementação, Testes e Implantação. Simplificadamente, a metodologia funciona da seguinte forma:

1. Requisitos são extraídos dos *stakeholders* (clientes, usuários, gerentes, etc), ou seja, todas as pessoas e entidades que possuem algum interesse no software, seja por que irão efetivamente usá-lo em seu trabalho diário, ou porque estão pagando pelo software, mesmo que não sejam futuros usuários diretos.

2. As especificações de requisitos são transformadas em modelos de software, mas de forma mais abstrata, sem detalhes técnicos. Esta etapa é chamada de análise.

3. Baseado na análise, são gerados modelos mais específicos para futura implementação. Esses modelos representam de forma menos abstrata o problema inicialmente especificado. Esta fase é comumente chamada de design em inglês ou projeto (não confundir com project, e não traduzir como desenho).

4 Os modelos de projeto e análise são implementados, de forma mais ou menos automática, usando linguagens de programação. Muitas pesquisas estão sendo feitas sobre a geração automática de software a partir de modelos de análise e projeto, como por exemplo, MDA (Model Driven Architecture) (http://www.omg.org/mda/).

5. Testes unitários são feitos, e posteriormente testes integrados, contendo todo o software.

6. Na fase de Implantação, o software é colocado em funcionamento. Várias atividades são relacionadas a essa fase, como popular a base de dados, incluindo dados reais, treinamento, instalação, etc. As atividades dependem em muito do tipo de sistema, da empresa, e do ambiente em que o software funcionará.

Contexto

O contexto em que o modelo em Cascata foi usado é muito diferente da época atual. No século passado, principalmente entre as décadas de 50 a 80, a disponibilidade de recursos computacionais era limitada em vários aspectos. Recursos computacionais incluem não apenas o computador, mas também impressoras, memória principal e capacidade de armazenamento de dados. Inicialmente apenas um computador central, normalmente alimentado por cartões perfurados, era usado (geralmente alugado por um alto valor, ou comprado por grandes empresas, governos e institutos de pesquisa). Posteriormente, os minicomputadores e "terminais burros" (sem poder de processamento autônomo) facilitaram a programação, e tornaram-se mais acessíveis, mas ainda assim limitados, inclusive em número e qualidade de ferramentas computacionais que auxiliassem a tarefa de programação.

Nesse ambiente de baixa disponibilidade computacional e alto custo, era necessário evitar-se ao máximo deixar que o computador descobrisse erros simples, como sintaxe errada ou não declaração de variáveis. Todas as funções e procedimentos precisavam ser escritas anteriormente, com suas entradas e saídas, de forma detalhada e executadas primeiramente por pessoas, e depois por máquinas. São os famosos testes de mesa, em que o programador verifica se a lógica de seu algoritmo está correta usando papel e caneta. Primeiramente o software deveria funcionar na teoria, para depois ser executado no computador. Do contrário, simples erros poderiam levar horas para serem descobertos pelo computador e posteriormente corrigidos. Desta forma, o software deveria ser completamente analisado antes de ser implementado no computador. Fazia sentido gastar tempo em documentação detalhada. Eventualmente era necessário fazer alterações nos requisitos, e aproveitar ao máximo o software já existente era muito importante. Desta forma, documentações detalhadas eram lidas e entendidas, para então alterar o software.

Essa realidade foi sendo alterada aos poucos, e já a partir da década de 90 tornou-se comum ter um computador (não um terminal conectado a um mainframe, mas um computador totalmente funcional) para cada funcionário em uma empresa. As ferramentas de desenvolvimento, como editores de texto, depuradores de código e compiladores passaram a ser integradas em um só produto, facilitando a tarefa de programação e testes. Os novos ambientes de desenvolvimento permitiram facilmente executar um software passo a passo e avaliar o conteúdo de suas variáveis. Por exemplo, avaliar quantas vezes um bloco de instruções é executado controlado por uma variável.

De maneira geral, o modelo clássico deve ser usado principalmente quando os requisitos forem estáveis, e em situações em que a documentação é muito importante. Isso pode acontecer devido a vários fatores, como o sistema ser crítico ou muito complexo.

Experiências na indústria

Vários estudos e pesquisadores mostram que modelos seqüenciais não funcionam bem. Por exemplo, Fred Brooks em seu famoso artigo *"No Silver Bullet: Essence and Accidents of Software Engineering"*, descreve que a idéia de especificar totalmente um software antes do início de sua implementação é impossível [Brooks, 1987]. Outro pesquisador, Tom Gilb, desencoraja o uso do modelo em Cascata para software, estimulando o desenvolvimento incremental como um modelo que apresenta menores riscos e maiores possibilidades de sucesso [Gilb, 1988].

Dados de 1995 [Standish Group, 1995] usando como base 8.380 projetos, mostram que apenas 16,2% deles foram entregues respeitando prazos, custos e todas as funcionalidades especificadas. Aproximadamente 31% foram cancelados antes de serem finalizados e 52,7% foram entregues, porém com prazos e custos maiores ou com menos funcionalidades do que o especificado

inicialmente. Mesmo projetos que respeitaram prazos e custos apresentaram problemas. As principais razões dessas falhas estavam relacionadas com metodologias seqüenciais e suas dificuldades em possibilitar alterações nos requisitos do software.

Dados mais recentes [Standish Group, 2000] mostraram melhora significativa, mas ainda não definitiva. Nesta pesquisa, 15% dos projetos terminaram sem mostrar resultados e 66% dos projetos não atenderam às necessidades dos usuários. A média de atrasos caiu para 63% e os projetos custaram em média 45% a mais que o planejado. De apenas 16% de projetos que cumpriram o planejado em 1994 (prazos, custos e funcionalidades), a pesquisa de 2000 mostrou que 28% dos projetos cumpriram o planejado. Entretanto, a pesquisa chamou a atenção para um fato: muitos dos projetos que tiveram êxito foram superestimados (existem casos de as estimativas serem feitas e, posteriormente, serem acrescentadas em 150%).

Foram várias as razões que contribuíram para essa melhoria. Talvez a principal tenha sido o uso de ferramentas computacionais, mais acessíveis e melhores. Por exemplo, apesar do alto custo de algumas ferramentas CASE, há várias que são gratuitas. Um outro fator que contribuiu foi a melhoria da qualidade dos processos de desenvolvimento. Uma das recomendações dessas pesquisas foi que o desenvolvimento de software deveria ser baseado em modelos incrementais, o que poderia evitar muitas das falhas reportadas.

Metodologias Ágeis

O termo metodologias ágeis tornou-se popular em 2001, quando 17 especialistas em processos de desenvolvimento de software estabeleceram princípios comuns compartilhados por várias metodologias. O resultado foi a criação da Aliança Ágil e o estabelecimento do Manifesto Ágil [http://agilemanifesto.org/]. Apesar das metodologias ágeis variarem em termos de práticas e ênfases, elas compartilham algumas características, como desenvolvimento iterativo e incremental, comunicação

e redução de produtos intermediários, como documentação extensiva. Segundo o manifesto ágil, deve-se valorizar:

Indivíduos e interações sobre processos e ferramentas

Os desenvolvedores devem ser ouvidos e ter uma opinião relevante no desenvolvimento de software. Sua forma de trabalho deve ser respeitada, evitando-se obrigatoriedades que freqüentemente não fazem sentido para quem na prática está acostumado a desenvolver software. Alterar os processos, ou mesmo comprar ferramentas caras para auxiliar o trabalho dos programadores devem ser medidas tomadas com cautela. A introdução de ferramentas CASE deve ser planejada e a equipe que irá usá-las deve ser ouvida, antes que as ferramentas sejam compradas.

Software executável sobre documentação

Por melhor que seja a documentação (atualizada, seguindo normas e padrões, itens facilmente identificáveis, para citar alguns exemplos), ainda assim é melhor para os clientes visualizarem o software na prática, executando em uma máquina real, ao invés de somente esquematizado em documentos, gráficos, tabelas e diagramas que normalmente são específicos para software e por isso mesmo podem ser de difícil compreensão para quem não os conhece.

Colaboração com o cliente sobre negociação de contratos

As metodologias ágeis não rejeitam que contratos sejam firmados. Apenas enfatiza que o cliente deve estar satisfeito, então renegociações podem ser necessárias. Por exemplo, o cliente pode querer maior tempo de testes, o que gera maiores custos. Uma forma de tentar resolver o problema seria convidar o cliente a testar partes do software juntamente com a equipe de desenvolvimento, ou até mesmo de forma isolada, o que tem potencial de diminuir custos.

Responder as mudanças sobre seguir um plano.

Supondo que uma pessoa deve percorrer de carro a distância de mil quilômetros entre duas cidades. Faz-se um plano, mesmo que seja informal, sobre que caminhos seguir, onde e quando parar para descansar, qual horário de partida e assim por diante. Caso algo ocorra de forma diferente, é necessário adaptar este plano. Por exemplo, caso uma parte de uma estrada esteja impedida, e haja um caminho alternativo, muda-se o caminho. Posteriormente, ao encontrar um posto de combustível com preços mais acessíveis, pode-se optar por antecipar uma parada de descanso dependendo do custo/benefício. Os dois exemplos mostram que planos podem ser alterados quando necessário. Assim funciona também quando metodologias ágeis são usadas para desenvolvimento

de software. Planos são feitos, mas para auxiliar, não para servirem de referência constante e nunca mudarem.

Princípios que Suportam as Metodologias Ágeis

No manifesto ágil, são listados alguns princípios que devem nortear todas as metodologias ágeis em geral. Por exemplo, um princípio fundamental é que o cliente deve ser satisfeito, o que deve ser feito através de entregas contínuas de partes funcionais do software. As mudanças pedidas pelos clientes são bem vindas em qualquer estágio do desenvolvimento do software, o que também contribui para satisfazer o cliente. Outro princípio é relativo às pessoas que estão desenvolvendo o software, que devem ser continuamente motivadas e ter todo o suporte necessário para auxiliar no desenvolvimento do seu trabalho.

Exemplos

Duas das metodologias mais citadas são a XP (eXtreme Programming) e a Scrum. A XP é uma metodologia ágil para equipes pequenas e médias que desenvolvem software baseado em requisitos vagos e que se modificam rapidamente. O primeiro projeto a usar XP foi o C3, da Chrysler. Após anos de fracasso utilizando metodologias tradicionais, com o uso da XP o projeto ficou pronto em pouco mais de um ano [Highsmith et al., 2000]. A XP baseia-se em 12 práticas, dentre elas a programação em pares, entregas freqüentes, refatoração continua e simplicidade.

Outra metodologia ágil que apresenta uma comunidade grande de usuários é a Scrum [Schwaber e Beedle, 2002]. Seu objetivo é fornecer um processo conveniente para projeto e desenvolvimento orientado a objeto. A Scrum apresenta uma abordagem empírica que aplica algumas idéias da teoria de controle de processos industriais para o desenvolvimento de softwares, reintroduzindo as idéias de flexibilidade, adaptabilidade e produtividade. O foco da metodologia é encontrar uma forma de trabalho dos membros da equipe para produzir o software de forma flexível e em um ambiente em constante mudança. A idéia principal da Scrum é que o desenvolvimento de software envolve muitas variáveis técnicas e do ambiente, como requisitos, recursos e tecnologia, que podem mudar durante o processo. Isto torna o processo de desenvolvimento imprevisível e complexo, requerendo flexibilidade para acompanhar as mudanças. O resultado do processo deve ser um software que é realmente útil para o cliente.

Existem várias outras metodologias ágeis disponíveis, mas que não são tão usadas como a XP e a Scrum, como por exemplo DSDM, Crystal, Agile Unified Process, Essential Unified Process e Feature Driven Development.

Resultados

Estudos empíricos mostram diversos casos de sucesso. Vários casos são registrados em congressos específicos da área, como *International Conference on Agile Processes and eXtreme Programming in Software Engineering e Agile Conference.*

Por exemplo, um artigo [Charette, 2001] comparando metodologias ágeis com as metodologias tradicionais mostrou que os projetos usando metodologias ágeis obtiveram melhores resultados em termos de cumprimento de prazos, de custos e padrões de qualidade. Além disso, o mesmo estudo mostra que o tamanho dos projetos e das equipes que utilizam as metodologias ágeis têm crescido. Apesar de serem propostas idealmente para serem utilizadas por equipes pequenas e médias (até 12 desenvolvedores), aproximadamente 15% dos projetos que usam metodologias ágeis estão sendo desenvolvidos por equipes de 21 a 50 pessoas, e 10% dos projetos são desenvolvidos por equipes com mais de 50 pessoas, considerando um universo de 200 empresas usado no estudo.

Vários estudos foram sistematizados para comparação em [Dybå e Dingsøyr, 2008]. No artigo, diversas pesquisas mostraram várias vantagens das metodologias ágeis em relação às tradicionais. Por exemplo, a prática da programação em pares mostrou-se útil não só como forma de treinar todos os membros da equipe, mas também para aumentar a padronização de código e aumentar a qualidade final do software. A equipe inteira torna-se responsável pelo software como um todo, evitando que partes sejam entendidas por apenas uma pessoa. Em termos de produtividade e qualidade, novamente as metodologias

ágeis apresentam bons resultados. As equipes que usaram XP foram mais produtivas e produziram software com menos erros. Interessante ainda notar que o tamanho do software final, em termos de linhas de código, foi menor em equipes usando XP e Scrum quando comparado com equipes que usaram metodologias tradicionais.

Problemas e Dificuldades

As metodologias ágeis apresentam também alguns problemas [Soares, 2004a]. Muitos acreditam que essas metodologias sejam uma volta ao processo caótico de desenvolvimento de software, conhecido também como "codifica-remenda". Esse modelo existe principalmente em pequenas e médias organizações que não podem suportar os altos custos de desenvolvimento das metodologias tradicionais.

O uso errôneo da XP pode inibir certas práticas positivas de desenvolvimento, como, por exemplo, a análise do problema por meio de diagramas. Obviamente, não se deve projetar diagramas que nunca serão consultados, mas é importante projetar alguns modelos que ajudarão no entendimento do problema.

A informalidade no levantamento de requisitos pode não ser bem vista pelos clientes, que podem sentir-se inseguros [Soares, 2004b]. Situação semelhante pode ocorrer com a refatoração de código, que pode ser interpretada pelos clientes como amadorismo e incompetência. Segundo Beck, o criador da XP, há ainda outros fatores que podem tornar a metodologia inadequada. Por exemplo, profissionais que não se adaptam bem a práticas de equipe, como a programação em duplas, podem ter muita dificuldade em aceitar a XP. A exigência de que a equipe não esteja geograficamente separada cria sérias dificuldades, sobretudo em grandes empresas onde isso é mais comum. Essas empresas apresentam em geral maiores resistências para adotar metodologias ágeis.

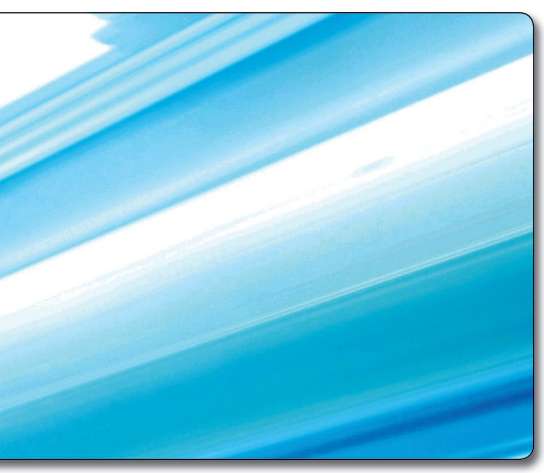
A prática da programação em pares talvez seja uma das mudanças mais radicais na XP. Profissionais mais experientes acham a prática ineficiente. O problema torna-se maior quando existe uma diferença significativa de conhecimento, experiência e mesmo capacidade entre os pares. Neste caso podem haver desentendimentos e queda de produtividade. Ainda em relação a pessoas, uma crítica comum é que as metodologias ágeis seriam adequadas somente para equipes e profissionais experientes, sendo ineficientes para novatos.

Conclusão

Todas metodologias de desenvolvimento de software apresentam vantagens e desvantagens, limites e restrições. Em geral, pode-se assegurar que ter uma metodologia é melhor que não ter e depender de heróis, aqueles profissionais que resolvem facilmente os problemas, que possuem a confiança da organização, mas que na prática não podem sair de férias sem que problemas sejam percebidos.

De forma geral, os resultados iniciais do uso de metodologias ágeis, em termos de qualidade, confiança, prazos de entrega e custo são promissores. Maiores estudos ainda são necessários, em diferentes projetos, tamanhos da equipe e com outras metodologias ágeis. Desta forma, os relatos de experiência podem ser úteis para organizações evitarem erros na adoção de metodologias ágeis.

A empresa ou mesmo uma equipe dentro da empresa precisa conhecer seus próprios problemas, dificuldades, limitações e capacidades para escolher a metodologia que melhor convém. Abordagens híbridas, modificadas e adaptadas para cada empresa, ou mesmo grupo da empresa e até para cada tipo de projeto específico pode ser a melhor escolha. ●



Dê seu feedback sobre esta edição!

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

www.devmedia.com.br/esmag/feedback

Referências

[Sommerville, 2003] Sommerville, I. Engenharia de Software Engineering. 6a. Edição. Addison Wesley, 2003.

[Royce, 1970] Royce, W. Managing the Development of Large Software Systems: Concepts and Techniques. Proc. WESCON, IEEE Computer Society Press, Los Alamitos, CA, 1970.

[Brooks, 1987] Brooks, F. No Silver Bullet: Essence and Accidents of Software Engineering. Proc. IFIP, IEEE CS Press, 1987, pp. 1069-1076.

[Gilb, 1988] Gilb, T., Principles of Software Engineering Management, Addison-Wesley, 1988.

[Standish Group, 1995] CHAOS report, 586 Olde Kings Highway. Dennis, MA 02638, USA, 1995.

[Standish Group, 2000] CHAOS report, USA, 2000.

[Highsmith et al., 2000] Highsmith, J. Orr, K. Cockburn, A. Extreme Programming, E-Business Application Delivery, Feb., (2000), pp. 4-17.

[Schwaber e Beedle, 2002] Schwaber, K. e Beedle, M. Agile Software Development with Scrum, Prentice-Hall, (2002).

[Charette, 2001] Charette, R. Fair Fight? Agile Versus Heavy Methodologies. Cutter Consortium E-project Management Advisory Service, 2, 13, (2001)

[Dybå e Dingsøyr, 2008] Dybå, T e Dingsøyr, T. Empirical studies of agile software development: A systematic review Information and Software Technology Volume 50, Issue 9-10, Agosto 2008, p. 833-859.

[Soares 2004a] Soares, M.S. Metodologias Ágeis Extreme Programming e Scrum para o Desenvolvimento de Software. Ano III, Número 1, Novembro de 2004.

[Soares 2004b] Soares, M.S. Comparação entre Metodologias Ágeis e Tradicionais para o Desenvolvimento de Software. INFOCOMP - Journal of Computer Science. Vol.3, N.2, Novembro, 2004, p.8-13.

Ferramentas de Integração Contínua tornando o Trabalho de Equipes mais Organizado

Conhecendo os Bastidores

Andrew Diniz da Costa

andrew@les.inf.puc-rio.br

Técnico em Informática pelo Instituto Brasileiro de Pesquisa em Informática (IBPI). Possui graduação em Bacharelado de Informática pela Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio). Mestre e aluno de doutorado em Informática da PUC-Rio. Desempenha o papel de pesquisador e líder de tecnologia na área de testes do escritório de qualidade de software do Laboratório de Engenharia de Software (LES). Possui experiência em desenvolvimento Java, Visual Basic, Delphi, ASP e C, além do uso de SGBDs, como, SQL Server e Oracle.

Co-autor: Carlos J. P. de Lucena

Possui graduação em Economia Matemática pela Pontifícia Universidade Católica do Rio de Janeiro (1965), mestrado em Informática pela University of Waterloo (1969), doutorado em School Of Engineering And Applied Sciences pela University Of California At Los Angeles (1974) e pós-doutorado pela IBM (1975). Atualmente é professor titular da Pontifícia Universidade Católica do Rio de Janeiro. Em sua carreira atuou como vice-reitor da Universidade, Decano do Centro Técnico e Científico e por várias vezes Diretor do Departamento Informática. Foi premiado com a insígnia da Classe Grã-Cruz da Ordem do Mérito Científico da Presidência da República do Brasil, com a Medalha Carlos Chagas Filho de Mérito Científico, Diretoria e Conselho Superior da FAPERJ, com o Prêmio Álvaro Alberto de Ciências e Tecnologia do Ministério de Ciência e Tecnologia e com vários prêmios IBM Innovation Award, dentre muitos outros. Publicou mais de 400 trabalhos, além de orientar até 2008, 34 teses de doutorado e 89 dissertações de mestrado.

Co-autor: Arndt Von Staa

Professor do Departamento de Informática da PUC-Rio. É PhD em Ciência da Computação (1974, Engenharia de Software) pela Universidade de Waterloo, Ontário, Canadá. Atua em computação desde 1962. Suas atividades de pesquisa e desenvolvimento mais recentes concentram-se em técnicas de controle da qualidade de software, em métodos e processos de desenvolvimento de software de qualidade assegurada e em ambientes de desenvolvimento de software assistidos por computador. Até julho de 2008 publicou mais de 80 artigos arbitrados completos em veículos nacionais e internacionais, cinco livros e nove capítulos de livros. Tem 14 softwares desenvolvidos sem registro de patente. Orientou 62 dissertações de mestrado e sete teses de doutorado. Coordena diversos projetos de pesquisa e desenvolvimento em parceria com empresas.

Diversas razões definem a presença de grandes grupos de desenvolvedores em uma mesma equipe para a criação de sistemas, tais como, tamanho, prazo de entrega solicitado pelo cliente, etc. Assim, a necessidade de desenvolvedores trabalharem em conjunto de forma organizada está presente em nosso dia a dia. Pensando nisso, como um projeto pode ter uma equipe organizada enquanto que um mesmo código fonte é compartilhado? Existem ferramentas que auxiliem nesse processo? Caso existam, quais seriam?

Para responder todas essas perguntas, podemos citar o conceito de integração contínua, bastante utilizado por processos ágeis, tais como Extreme Programming. A idéia principal consiste em integrar o trabalho realizado por várias pessoas durante diversos momentos do dia, e realizar testes que permitam assegurar que o código continue consistente ao final de cada integração.

A forma ideal para aplicar integração contínua é através do uso de diversas ferramentas, tais como, controle de

versões (ex: CVS, Subversion e Clear Case), sistemas de *build* (ex: Ant, NAnt e Maven), além de notificações para usuários a partir de execuções realizadas (ex: email, MSN e SMS). Execuções, como, por exemplo, *builds* de projetos, são chamados de *Jobs* em diversos pontos no artigo.

Na **Figura 1** ilustramos uma arquitetura que representa a idéia de integração contínua. Observe que nesse exemplo temos uma equipe com três desenvolvedores realizando mudanças em um mesmo arquivo ou em um conjunto de arquivos que constituem um projeto, compartilhado por um repositório localizado em outra máquina. No exemplo, consideramos que uma ferramenta própria para integração contínua instalada em outra estação é utilizada. A partir dela, poderíamos definir uma hora do dia para executar o *build* da última versão do projeto em desenvolvimento. Além disso, dependendo da ferramenta utilizada, podemos definir que uma notificação por e-mail deve ser enviada para toda a equipe de desenvolvimento informando se a execução foi realizada com sucesso ou se falhou.

Nesse artigo, apresentamos inicialmente uma visão geral de sistemas de build e de controle de versão. Em seguida, são apresentadas as principais características de três ferramentas de integração contínua: Cruise Control, Continuum e Hudson, e ao final do artigo é realizada uma comparação entre elas. A partir dessa comparação, será possível identificar quais suas principais vantagens e desvantagens.

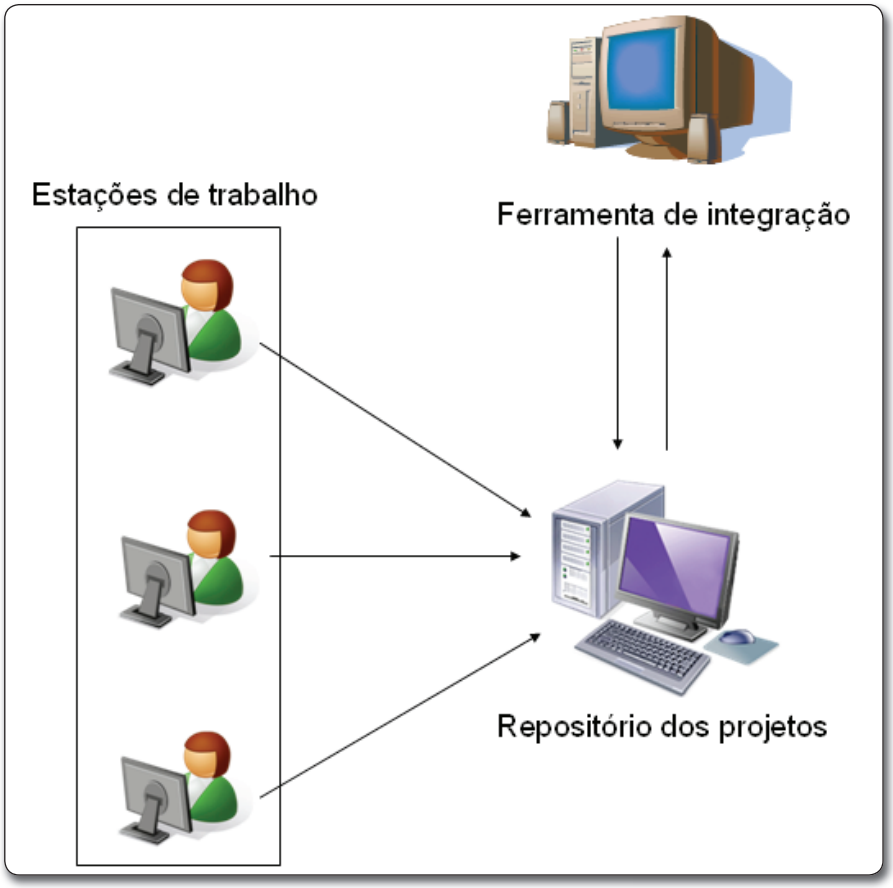


Figura 1. Arquitetura para representar integração Contínua.

Sistemas para Build

Sistemas de *build* são usados para agilizar e automatizar execuções de projetos, através da automatização de tarefas, como: compilação do código do sistema; compilação e execução dos testes unitários e de aceitação; geração de relatórios dos testes, de cobertura e de análise estática do código; e quaisquer outras atividades necessárias. Exemplos de ferramentas de *build* usadas hoje em dia são: Ant, Maven 1, Maven 2, NAnt, entre outras.

Sistemas para controle de versão

Um sistema de controle de versão é usado para armazenar e manter versões de arquivos, assim como, todo o código fonte de um sistema. Também conhecido por repositório, é responsável por automatizar tarefas como: identificação de mudanças locais; exibição das diferenças entre o código de um arquivo local e uma de suas versões armazenadas no repositório; identificação de quem realizou uma determinada alteração; incorporação de uma mudança em um arquivo local a uma versão do mesmo arquivo armazenada no repositório; sincronizar o código local com uma das versões do sistema ou de um arquivo armazenadas no repositório. Ferramentas muito usadas para desenvolvimento são as seguintes: Subversion, CVS, Clear Case.

Para realizar a integração contínua, é importante utilizar um sistema de controle de versões. Caso não haja um sistema desse tipo, torna-se difícil permitir que equipes com diversos desenvolvedores trabalhem juntos e compartilhem o código fonte de um mesmo projeto.

Imagine dois desenvolvedores alterando um mesmo arquivo. Duas grandes preocupações estariam presentes: a primeira refere-se a garantir que cada desenvolvedor esteja ciente que o mesmo arquivo está sendo alterado por outras pessoas, enquanto que a segunda seria entender quais mudanças foram feitas por cada desenvolvedor para que uma versão integrada possa ser gerada. A partir de sistemas que realizam controle de versão, tais preocupações podem ser mais facilmente gerenciadas.

Cruise Control

Essa é uma das aplicações mais famosas para integração contínua, mantida e desenvolvida por voluntários dispostos a melhorar e disponibilizar novas funcionalidades. A aplicação permite que *builds* sejam executados a partir de diferentes ferramentas, a

Instalação

Pré-requisito: Considerando a linguagem Java, instale o JDK 5.x ou superior.

Acesse o link http://cruisecontrol.sourceforge.net/download.html, e a seguir escolha a versão que deseja. Atualmente, dois formatos de instalação são oferecidos: ".exe" e ".zip".

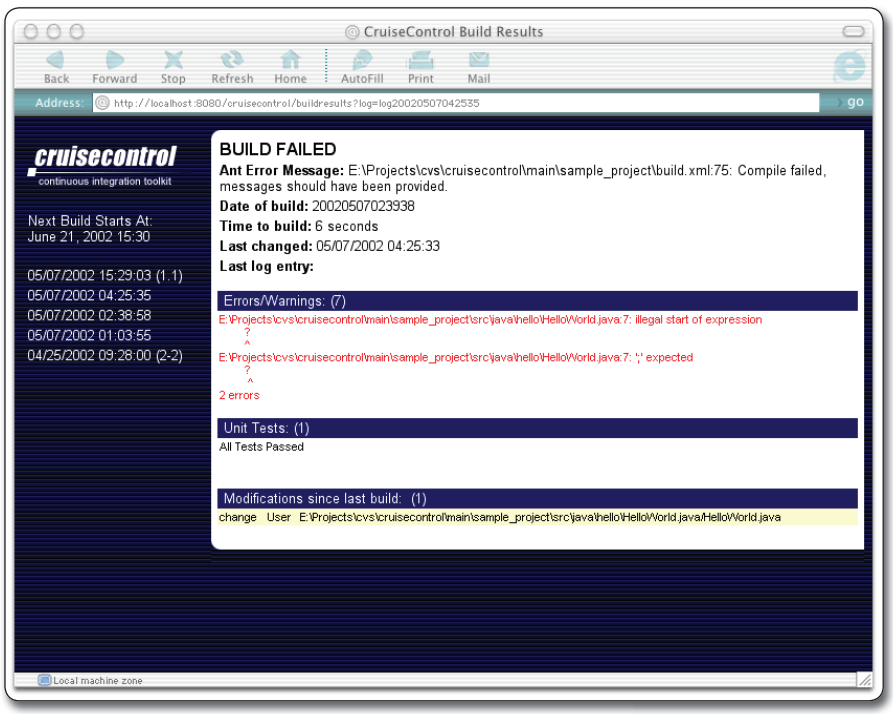


Figura 2. Página JSP para notificação.

integração com diferentes controles de versão, além de realizar notificações de diferentes formas, quando, por exemplo, alguma falha acontece durante a execução de um *job*.

Cruise Control é dividido em três partes principais: (i) configuração do build dos projetos em um arquivo xml chamado "config.xml", (ii) geração de relatórios que descrevem o que aconteceu na execução de *builds*, (iii) além de permitir o acompanhamento do status dos *builds* de diferentes projetos a partir de uma página web (*dashboard*).

Quando algum desenvolvedor deseja executar o *build* de um projeto, o primeiro passo é configurar a sua execução no arquivo config.xml. Diversos tipos de *tags* são oferecidos para que seja possível definir qual sistema de *build* será usado (Ant, Maven, Maven2, NAnt ou Phing) e qual controle de versão (AccuRev, AlienBrain, ClearCase, CVS, Perforce, PVCS, StarTeam, Subversion ou Visual Source Safe). Além disso, a ferramenta permite que caso o projeto tenha sofrido mudanças no repositório, a última versão seja utilizada para a execução do seu build. Caso aconteçam falhas durante um *build*, o Cruise Control permite que seja definido se sua execução deve continuar ou se deve ser cancelada.

Outras funcionalidades interessantes que podem ser definidas é a possibilidade de excluir arquivos gerados em execuções anteriores de um *build* (exemplo: arquivos de *log*), assim como especificar qual será o diretório que os novos artefatos a serem gerados serão armazenados em futuras execuções. Podem ser guardados tanto localmente, como também em diretórios remotos (ex: FTP). Uma funcionalidade diferenciada é a possibilidade de realizar um merge de um *log* gerado pelo próprio Cruise Control com os resultados obtidos de um build que realiza testes unitários com JUnit utilizando o sistema Ant. A utilidade desse merge é a oportunidade de analisar de forma unificada como foi a execução dos builds e em que pontos foram realizados tais testes.

O exemplo a seguir apresenta parte de um arquivo config.xml responsável por definir um projeto chamado "project1" que terá seu *build* executado por um ant (<ant>). Nesse exemplo são definidos três pontos: (i) o arquivo de *log* gerado pelo Cruise Control faz um merge com os testes feitos em junit (<log>), (ii) toda vez que o arquivo mof_file.txt for modificado o cvs é acessado (<modificationset>), (iii) e as notificações das execuções serão realizadas via e-mail. Toda vez que for executado um *build*, o e-mail *desenvolvimento@gmail.com* recebe as informações do *build* realizado, já o e-mail *suporteles@gmail.com* recebe e-mail somente quando um *build* gera alguma falha.

```
<cruisecontrol>
          ...
  <property name="projectdir"
     value="${env.CCDIR}/checkout/
      ${project.name}"/>
  <property name="testdir"
     value="${projectdir}/build/
      junit-reports"/>
                ...
  <project name="project1"/>
        ...
      <log>
        <merge dir="${testdir}">
      </log>

      <modificationset quietperiod="1" >
        <compound
          includeTriggerChanges="false">
        <triggers>
          <filesystem
            folder="./mod_file.txt" />
        </triggers>
        <targets>
           <cvs cvsroot=":pserver:
             user@cvs_repo.com:/cvs" />
        </targets>
        </compound>
      </modificationset>

     <schedule>
       <ant
          antscript="C:\Java\
           apache-ant-1.6.1\bin\ant.bat"
          antworkingdir="D:\
           workspace\MyProject"
          buildfile="MyProject-build.xml"
          uselogger="true"
          usedebug="false"/>
     </schedule>

     <publishers>
       <htmlemail>
         <always
           address="desenvolvimento@gmail.com">
         <failure
           address="suporteles@gmail.com">
       </htmlemail>
     </publishers>
  </project>

</cruisecontrol>
```

Outro ponto importante oferecido é o serviço de notificação. Toda vez que acontece uma execução de um *job*,

diferentes formas de notificações podem ser realizadas, como: e-mail, weblog, Yahoo IM Message, geração de uma página em html ou jsp, assim como ilustrado na **Figura 2**. Perceba que a notificação desempenha o papel de um relatório responsável por apresentar o que aconteceu durante alguma execução.

Para que seja possível acompanhar o status dos projetos configurados no config.xml, é oferecido o Cruise Control Dashboard que vem a ser uma página web que permite acompanhar a execução dos *jobs*. O resultado de cada *job* é representado por ícones que representam o status: pausado, em fila ou em execução. Na **Figura 3** é apresentado um exemplo de projeto que teve seu build executado com sucesso e que a seguir foi armazenado (*queued*) para manter o histórico de execuções do projeto. Esse histórico pode ser acessado a partir da opção *Builds* apresentada na **Figura 4**. Perceba que quando um projeto já foi executado diversas vezes, seu historio é guardado (*Latest Builds*), permitindo que o usuário possa clicar em qualquer *build* e analisar o que aconteceu em cada execução, como, por exemplo, se houve sucesso ou falha e quais modificações aconteceram em relação à execução anterior.

O Cruise Control oferece suporte para três linguagens de programação: Java, Ruby/Rails e .NET. Além disso, a ferramenta pode ser executada tanto no Windows como no Unix.

Como a configuração dos projetos é toda realizada em XML, a curva de aprendizado pode ser grande em comparação a ferramentas de integração contínua que permitem tais configurações a partir de páginas web. Outra desvantagem é em relação ao dashboard (página web) apresentado, pois caso o arquivo config.xml seja alterado em algum instante, a página web não é alterada automaticamente, ou seja, há a necessidade de atualizar a página (clicar no botão de refresh do browser), para que tais mudanças sejam percebidas.

Instalação

Pré-requisito: Considerando a linguagem Java, instale o JDK 5.x ou superior.

Acessar o link http://continuum.apache.org/download.html, realizar o download da versão desejada, descompactá-la em um diretório local, e a seguir executar a aplicação. Caso esteja usando o Windows, o arquivo que deve ser executado é o "...\continuum-1.1\bin\windows-x86-32\run.bat".

Figura 3. Cruise Control Dashboard.

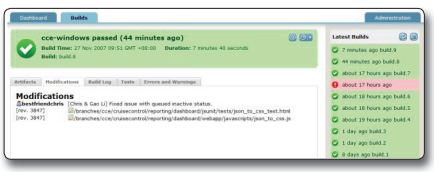


Figura 4. Detalhe dos builds.

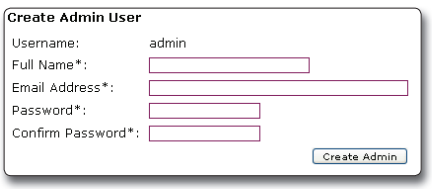


Figura 5. Criar usuário administrador.

Continuum

Outra ferramenta de bastante sucesso e que tem sido utilizada por diversas equipes de desenvolvimento é o Continuum. Assim como o Cruise Control, o Continuum funciona como um servidor para integração contínua. No entanto, foi desenvolvida pela equipe da ferramenta Maven, sendo considerada a ferramenta default para projetos que trabalhem com Maven1 e Maven2.

Para facilitar a gerência dos projetos que terão seus *builds* executados, é oferecida uma página web bastante amigável, que pode ser executada tanto em Windows, Debian, Fedora Core, Mac OS X e Solaris. A partir dela podem ser usados diferentes sistemas de *build* (Maven2, Maven, Ant e Shell Script), assim como diferentes controles de versão (Subversion, CVS, Starteam, Clearcase, Perforce, bazaar e Visual Source Safe). Outra funcionalidade importante são as formas de notificar pessoas sobre as

Instalação
Pré-requisito: Considerando a linguagem Java, instale o JDK 5.x ou superior.
Acessar o link https://hudson.dev.java.net/servlets/ProjectDocumentList, realizar o download da versão desejada, e descompactá-la em um servidor web, como, por exemplo, Tomcat 5.x. Quando o servidor for executado, acesse, por exemplo, a página a partir do link http://localhost:8080/hudson.

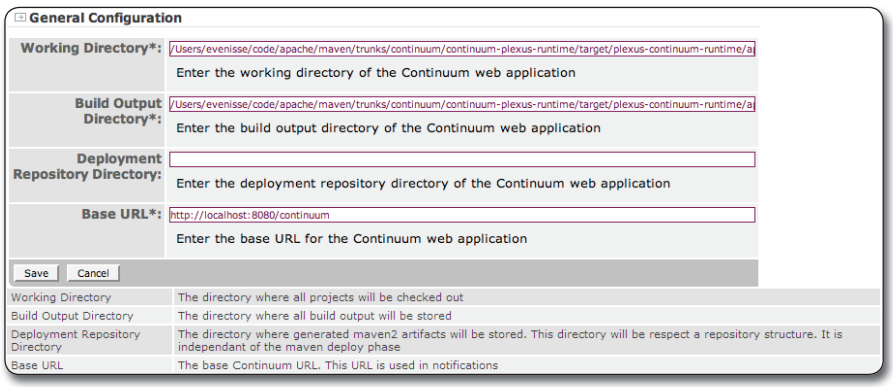


Figura 6. Configurações gerais dos projetos.

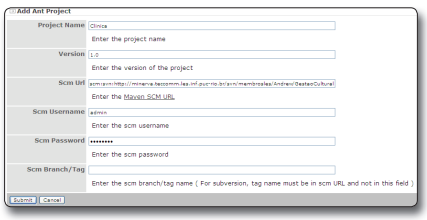


Figura 7. Adicionar Projeto do tipo Ant.

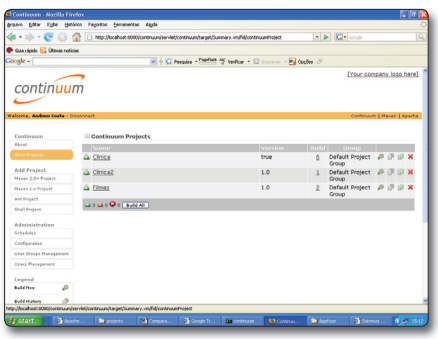


Figura 8. Apresentação dos projetos em acompanhamento.

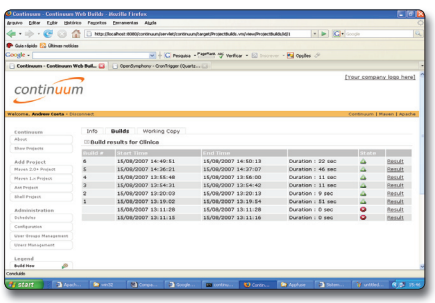


Figura 9. Histórico dos builds realizados em um projeto.

execuções dos *jobs*: e-mail, IRC (Internet Relay Chat), Jabber (protocolo aberto baseado em XML) e MSN.

Assim que a instalação do Continuum é realizada e sua execução é feita pela primeira vez, a primeira etapa é criar uma conta admin, como ilustrado na **Figura 5**. Após a criação do *login*, será possível realizar as configurações gerais dos projetos que terão seus *jobs* executados. Tais informações, que serão usadas como default para todos os outros projetos a serem incluídos, devem ser fornecidas no formulário apresentado na **Figura 6**.

A partir das configurações realizadas, projetos usando diferentes ferramentas de *build* podem ser adicionados, como, por exemplo, projetos do tipo Ant, Maven 2, etc. Veja um exemplo na **Figura 7.**

Quando os projetos já estiverem adicionados, o usuário poderá executá-los e acompanhá-los quantas vezes quiser. Na **Figura 8** é apresentada uma página que permite acompanhar os *jobs* que estão sendo gerenciados, enquanto que na **Figura 9** o histórico dos *jobs* executados por projeto é apresentado. Já na **Figura 10** todos os arquivos que fizeram parte do último *job* são apresentados e disponibilizados para que os usuários possam acessá-los a partir de um clique.

Para permitir um maior controle e organização no acompanhamento dos projetos, o Continuum oferece a possibilidade de criar grupos de acesso ao sistema. Dessa forma, podemos especificar quais permissões de acesso cada usuário terá, como:

- Adicionar projetos.
- Editar projetos.
- Excluir projetos.
- Executar *builds*.
- Adicionar definições de *job* (ex: *build*), isto é, configurações relacionadas, como, por exemplo, linha de comandos e argumentos necessários para sua execução.
- Excluir definições de *job*.
- Adicionar notificações (avisar por email, por exemplo, um grupo de pessoas sobre a execução de um *build*).
- Editar notificações.
- Excluir notificações.
- Gerenciar escalonadores, isto é, definir de quanto em quanto tempo um *job* deve ser executado, ou talvez definir a hora, dia e ano que deseja realizar sua execução.
- Permitir a gerencia de grupos de acesso para usuários.

Referente a testes, o Continuum disponibiliza relatórios de testes unitários gerados a partir das ferramentas Maven 1 ou Maven 2. Essa é uma das características não presente no Cruise Control.

Hudson

Hudson, criado pelo engenheiro da Sun Microsystems Kohsuke Kawaguchi, é uma ferramenta para integração contínua que monitora a execução de trabalhos repetidos, como o *build* de um projeto, assim como o Cruise Control e o Continuum. Através de uma interface amigável, torna-se fácil acompanhar o estado de cada *build*. Além disso, permite o trabalho colaborativo com diversos tipos de sistemas de controle de versão (CVS e Subversion), e sistemas de *build* (Ant, Maven 1, Maven 2, Shell Script e Batch command do Windows). Assim como as outras ferramentas de integração, o

Hudson também provê suporte para notificações das execuções dos *jobs* através de e-mail, RSS e IM Integration.

A página que permite a gerência da execução dos *jobs* é de fácil entendimento e de fácil instalação. Na **Figura 11** é apresentada a página inicial para a criação de um novo *job*. Percebe-se cinco diferentes opções, dentre elas a criação de um *job* para um projeto que utilize Maven 2.

Para exemplificar uma criação de *job*, selecionamos a opção "*Build a free-style software project*" e informamos o nome "*gestão*". A seguir, diversas opções são apresentadas, assim como apresentado na **Figura 12**. Observe que podemos especificar em qual repositório está o projeto desejado. Com isso, a ferramenta procura o "build.xml" do projeto e o executa de forma automática. Outras opções interessantes, como, por exemplo, gerar um link que permita acesso ao javadoc do (opção "*Publish Javadoc*") são apresentadas. Para cada opção perceba que há um ponto de interrogação ao lado responsável por explicar qual sua finalidade. Assim que as configurações são salvas, os usuários poderão executar o *job* e visualizar seu histórico de execuções quantas vezes quiserem. Um dos poucos pontos que a ferramenta não trata é a ausência de um controle de acesso por usuário, assim como é oferecido pelo Continuum.

Observe que na **Figura 13** é apresentada uma página com todos os projetos cadastrados no sistema, além de informações úteis, como, por exemplo, quando foi sua última execução e quando aconteceu sua última falha. A interface torna-se ainda mais atrativa, pois dependendo da quantidade de sucessos ou falhas presentes nas execuções de cada projeto, uma imagem condizente é apresentada ao lado (ex: tempo chuvoso, sol, etc).

Outro ponto interessante apresentado na **Figura 14** é a possibilidade de acompanhar em tempo real a execução dos *builds*. Ao final da execução, caso deseje gerar um arquivo de texto com as informações apresentadas durante o *build*, basta selecionar a opção "*View as plain text*".

Em relação a testes, o Hudson oferece suporte para geração de relatórios usando JUnit e TestNG. Tais relatórios podem ser tabulados, sumarizados e indicados com informações de histórico, como o início da execução do *job* correspondente. Outra funcionalidade interessante é a criação de um gráfico descrevendo o percentual de *builds* executados com sucesso e falha em relação a algum projeto. Na **Figura 15** é apresentado um exemplo com dois *builds* ("*count*" igual a 2) executados com

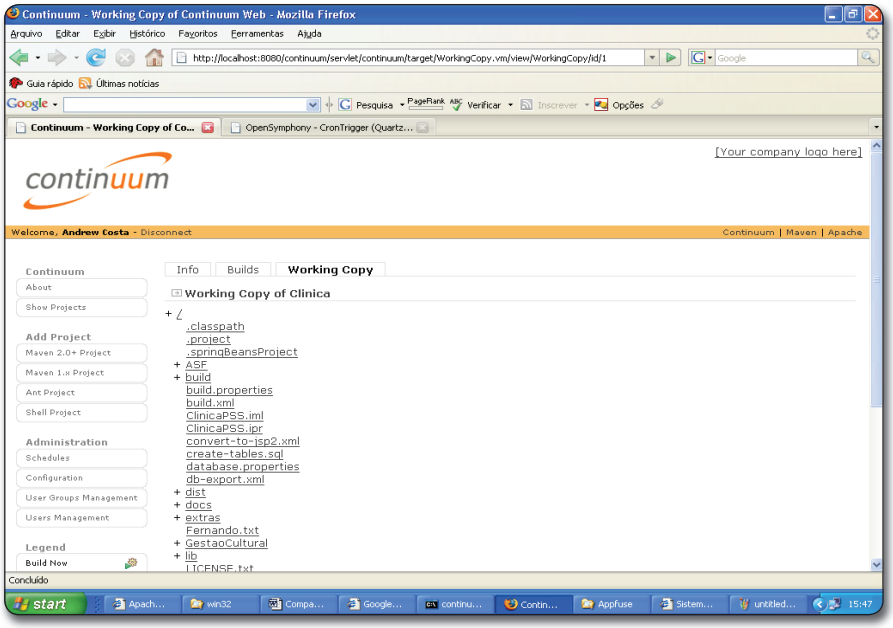


Figura 10. Arquivos usados na última execução de um *build*.

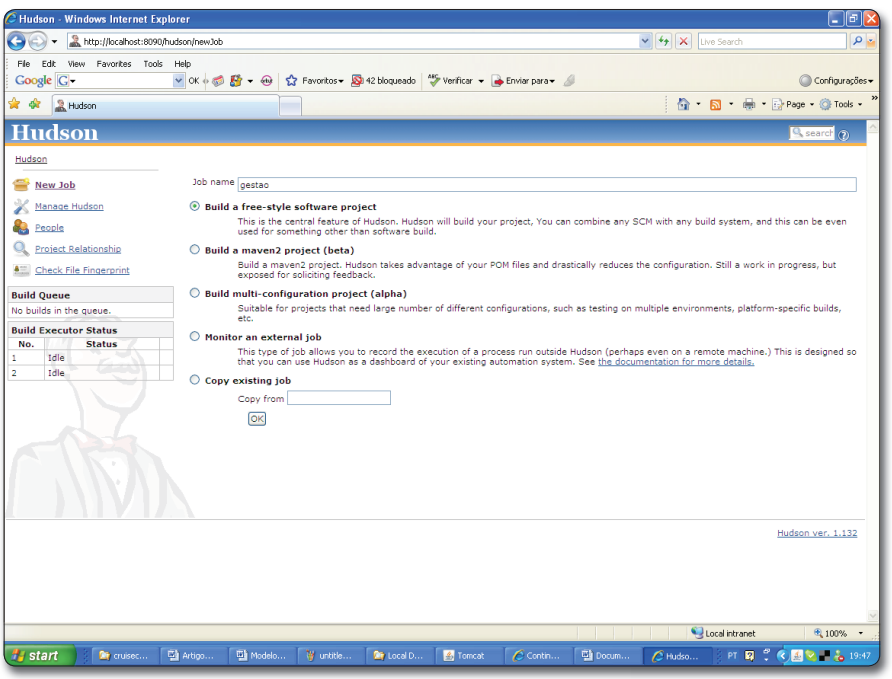


Figura 11. Tipos de jobs que podem ser criados.

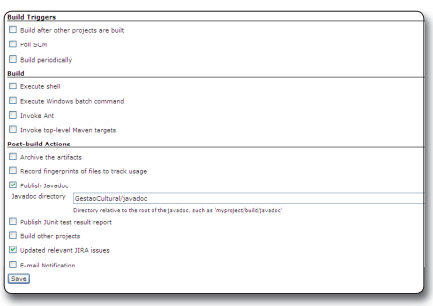


Figura 12. Formulário de criação de um job.

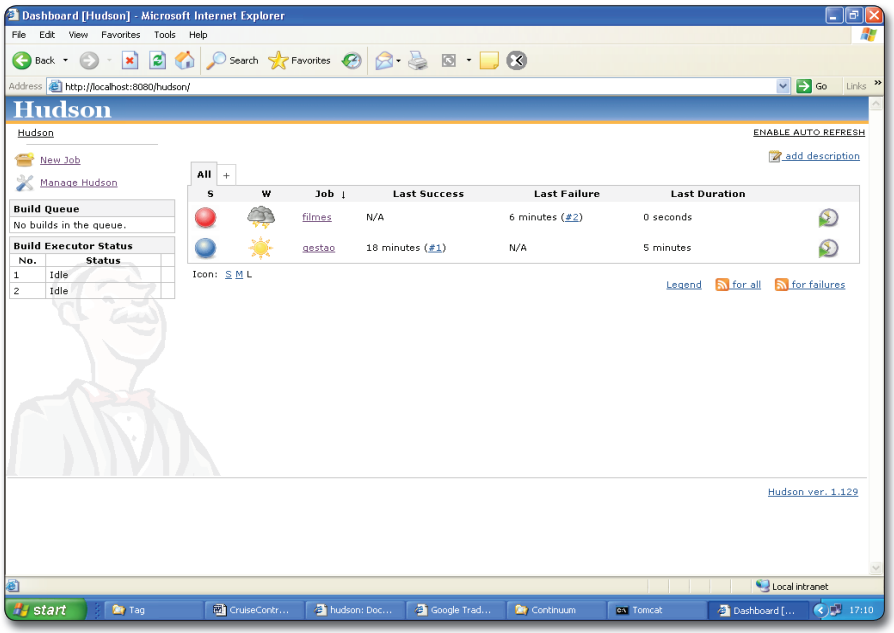


Figura 13. Jobs cadastrados e seus status atuais.

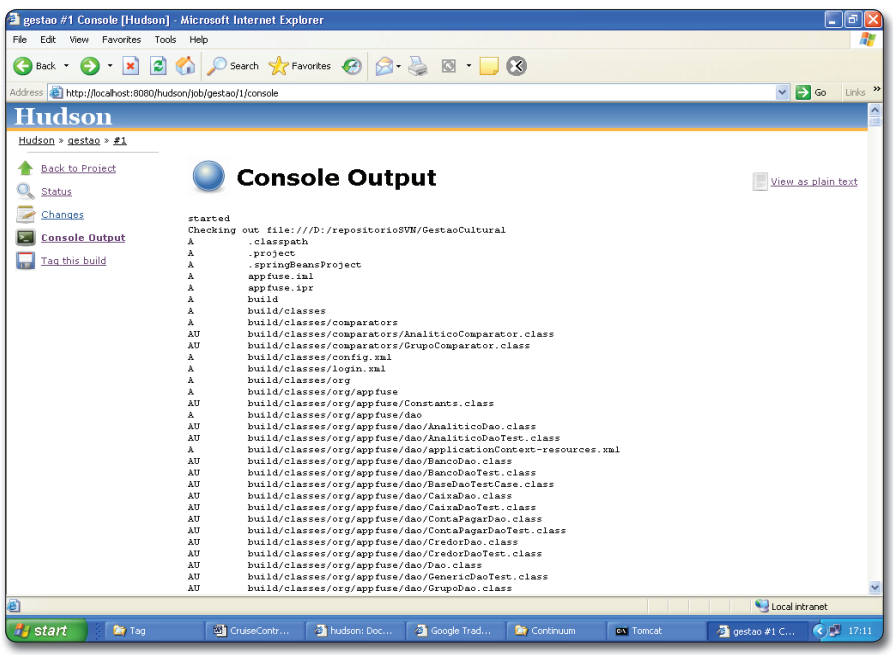


Figura 14. Console de acompanhamento de um build.

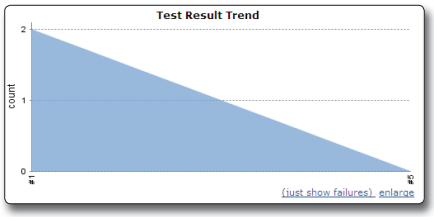


Figura 15. Gráfico de erros de um projeto.

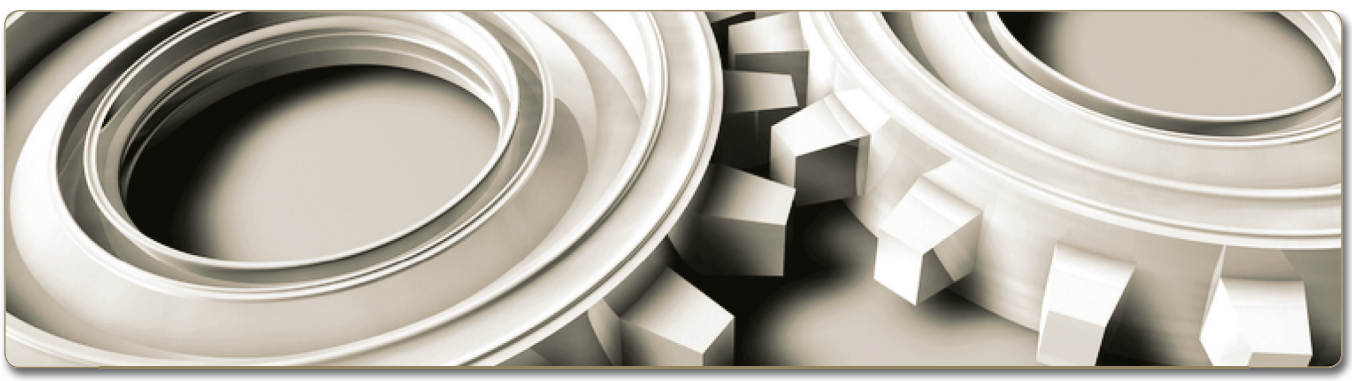
sucesso. As identificações dos *builds*, #1 e #5, foram geradas pelo próprio Hudson. Observe que no gráfico, dois testes são executados no build #1 enquanto que no build #5 nenhum deles é executado. Isso aconteceu devido o código fonte não sofrer alterações.

Além das vantagens mencionadas, o Hudson permite sua extensão a partir de *plug-ins*, como, por exemplo, o uso do JIRA *plug-in*. JIRA é um programa de gestão de projetos e acompanhamento de tarefas e erros. A partir dele, *issues* descrevendo falhas em execução podem ser criadas. Pensando nisso, o JIRA *plug-in* foi criado para permitir que novas *issues* sejam criadas quando falhas acontecerem em algum *build*. Essa característica de extensão da ferramenta de integração contínua a partir de *plug-ins* também está presente no Cruise Control.

Devido às facilidades citadas, o Hudson torna o trabalho de gerenciamento dos projetos fácil e particularmente agradável. Para maiores detalhes visite o site da ferramenta apresentado no final do artigo.

Comparação das Ferramentas

Visando facilitar a comparação das ferramentas de integração contínua mencionadas, na **Tabela 1** é apresentado um resumo das suas principais características. A partir do estudo realizado percebeu-se que o Cruise



Control é dentre as três, aquela que oferece mais funcionalidades, enquanto que o Continuum oferece maior suporte para projetos que trabalham com Maven 1 e Maven 2, e o Hudson apresenta a interface web mais amigável para uso, além de oferecer diversas funcionalidades interessantes, como, por exemplo, suporte para testes utilizando JUnit e TestNG.

Conclusão

Nesse artigo foram apresentadas três ferramentas de integração contínua amplamente usadas no mercado e que ajudam a organizar trabalhos de equipes compostas por desenvolvedores. Tais ferramentas permitem que projetos tenham seus *builds* executados, realize controles de versão, acompanhe testes executados, realize notificações sobre as execuções, especifique data e horário que um *build* deve ser executado automaticamente, etc.

Em diversas situações torna-se uma dor de cabeça para equipes de desenvolvimento a integração dessas funcionalidades para a gerência de um ou mais projetos. Assim, uma ótima solução é o uso das ferramentas de integração contínua. ●

Links

Cruise Control Site
http://cruisecontrol.sourceforge.net/

Apache Continuum: Continuous integration and Build Server Site
http://continuum.apache.org/

Hudson: an Extensible Continuous Integration Engine Site
https://hudson.dev.java.net/

Artigo "Integração Contínua", escrito por Vinícius Manhães Teles
http://www.improveit.com.br/xp/praticas/integracao

Integração Contínua: Desenvolvimento Ágil
http://devagil.wordpress.com/2007/04/14/4611-integracao-continua/

JUnit
http://www.junit.org/

Apache Maven Project Site
http://maven.apache.org/

The Apache Ant Project Site
http://ant.apache.org/

Apache Maven Project Site
http://maven.apache.org/

CVS – Concurrent Versions System
http://www.nongnu.org/cvs/

Subversion
http://subversion.tigris.org/

Extreme Programming> A Gentle Introduction
http://extremeprogramming.org/

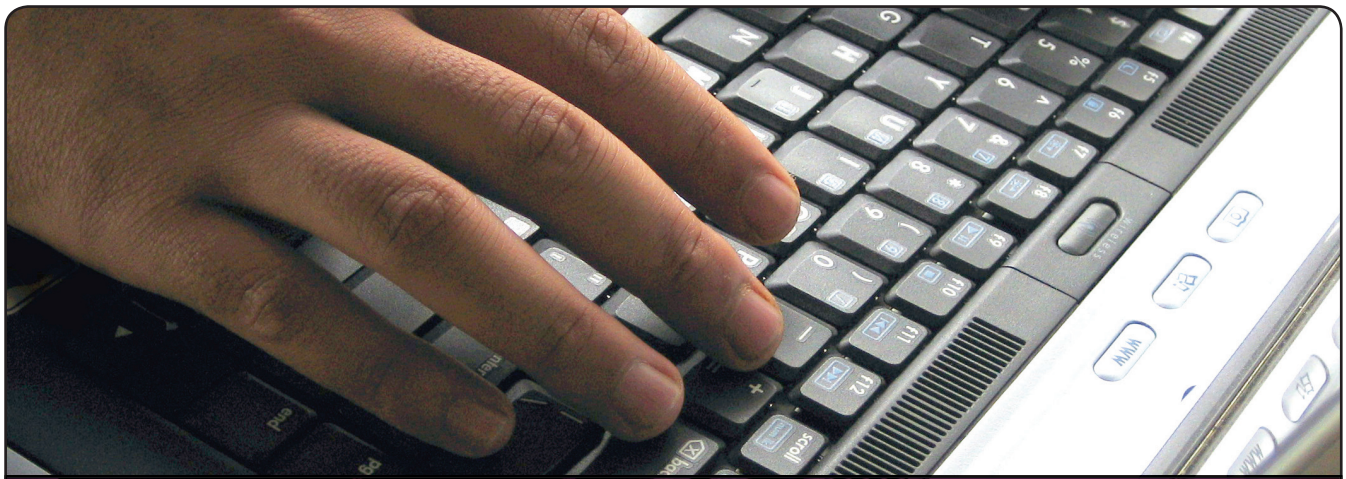
Dê seu feedback sobre esta edição!

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

	Cruise Control	Continuum	Hudson
Linguagens de programação que oferece suporte.	Java, Ruby/Rails e .Net	Java	Java
Sistemas Operacionais.	Windows e Unix	Windows e Unix	Windows e Unix
Ferramentas de build.	Ant, NAnt, Maven 1, Maven 2, Shell Script	Maven 2, Maven 1, Ant e Shell Script.	Maven 2, Maven 1, Ant, Shell Script e Batch command do Windows.
Suporte para testes.	Realiza um merge dos testes gerados pelo JUnit de um Ant com um arquivo de log gerado pelo Cruise Control.	Gera documentação de testes para projetos que usam Maven 1 e Maven 2.	(i) Oferece suporte para JUnit e TestNG; (ii) gera relatórios de testes, que podem ser tabulados, sumarizados e indicados com informações de histórico; (iii) criação de um gráfico contando detalhes de cada teste na documentação.
Funcionalidades interessantes.	(i) Especificar se ao encontrar uma falha no build, continua ou para a execução; (ii) atualizar o arquivo de build de um projeto através de um repositório, antes de sua execução; (iii)	(i) Disponibilizar relatórios de testes unitários; (ii) criação de grupos de acesso; (iii) acompanhamento em tempo real da execução de um build através de um console, etc.	(i) Integração com geração de Javadoc; (ii) definir onde os arquivos empacotados (ex: .war, .jar), criados a partir da execução de algum build, devem ser armazenados, (iii) acompanhamento em tempo real da execução de um build através de um console, etc.
Configuração dos projetos.	Configuração realizada em um arquivo xml. Curva de aprendizado inicial pode ser maior que o normal, no entanto, depois a configuração se torna-se fácil.	Configuração realizada em um arquivo xml. Curva de aprendizado inicial pode ser maior que o normal, no entanto, depois a configuração se torna-se fácil.	Configuração realizada em uma interface amigável.
Ferramentas de controle de versão.	AccuRev, AlienBrain, ClearCase, CVS, Perforce, PVCS, StarTeam, Subversion, Visual Source Safe.	Clearcase, CVS, Local, Perforce, Starteam, Subversion, Visual Source Safe.	CVS e Subversion.
Formas de extensão.	Código fonte fornecido para extensões. Possibilidade de utilizar plugins.	-	Código fonte fornecido para extensões Possibilidade de utilizar plugins.
Formas de notificação.	Email, Weblog, Yahoo IM message, Html, JSP.	Email, IM (IRC, Jabber, MSN).	Email, RSS, IM Integration.

Tabela 1. Tabela comparativa das ferramentas de integração contínua.



Avaliação Heurística de Web Sites

Identificação de Problemas de Usabilidade

De que se trata o artigo:
Identificação de problemas da usabilidade em Web sites utilizando o método de avaliação heurística (visto na edição anterior) vinculando esses problemas às heurísticas e sugerindo correções no projeto.

Para que serve:
Prover uma técnica de baixo custo para avaliação da usabilidade de Web sites para identificar problemas de usabilidade no produto de software durante e ao final do processo de desenvolvimento..

Em que situação o tema é útil:
Além de ser considerada uma boa prática de avaliação, permite ao engenheiro de software, ou avaliador, identificar problemas de usabilidade antes de apresentar o produto (Web site) aos usuários. camente todos os portais de informação, entretenimento etc. possuem seus links RSS para leitura rápida.

Antonio Mendes da Silva Filho
antoniom.silvafilho@gmail.com
Professor e consultor em área de tecnologia da informação e comunicação com mais de 20 anos de experiência profissional, é autor do livros Arquitetura de Software e Programando com XML, ambos pela Editora Campus/Elsevier, tem mais de 30 artigos publicados em eventos nacionais e internacionais, colunista para Ciência e Tecnologia pela Revista Espaço Acadêmico com mais de 60 artigos publicados, tendo feitos palestras em eventos nacionais e exterior. Foi Professor Visitante da University of Texas at Dallas e da University of Ottawa. Formado em Engenharia Elétrica pela Universidade de Pernambuco, com Mestrado em Engenharia Elétrica pela Universidade Federal da Paraíba (Campina Grande), Mestrado em Engenharia da Computação pela University of Waterloo e Doutor em Ciência da Computação pela Univesidade Federal de Pernambuco.

Um requisito essencial no projeto da interface de usuário de sistemas de software e, especificamente, em web sites é prover usuários com facilidade de uso de modo que o usuário possa rapidamente encontrar a informação buscada ou realizar uma tarefa. Fazer isso implica em tornar a interface intuitiva oferecendo suporte à usabilidade. Este artigo explora o uso da avaliação heurística como método de inspeção da usabilidade de web sites. O objetivo do artigo é usar um conjunto de heurísticas para identificar problemas de usabilidade que, geralmente, comprometem o bom uso e desempenho de usuários.

Usabilidade de Web Sites

Os projetistas de interface consideram recomendações de projeto de interfaces e heurísticas de usabilidade durante o projeto de sistemas de software e, em específico, de web sites. Diferentes perfis de usuários impõem dificuldade adicional no projeto de interfaces devido à necessidade de tornar a interface intuitiva

para a ampla variedade de usuários.

Os seres humanos possuem uma ampla variedade de habilidades que os diferenciam uns dos outros. Essas diferenças podem ser categorizadas em termos culturais, idade, gênero, personalidade, habilidades cognitivas, entre outros. Quando se fala em prover usabilidade, o foco recai, principalmente, sobre as habilidades cognitivas e sensoriais dos usuários, isto é o que torna a realização de uma tarefa ser algo simples e intuitivo.

Note que a usabilidade é uma característica através da qual o usuário percebe quão intuitivo e fácil de usar é um produto como, por exemplo, um web site, e expressa sua satisfação no uso deste site. Usabilidade resulta em simplicidade e agilidade.

Projetistas de interface e, especificamente, de web sites consideram recomendações e heurísticas de

• **Visibilidade do estado do sistema** – Prover o usuário de feedback apropriado.

• **Casamento do sistema com o mundo real** – Utilizar termos, objetos e conceitos familiares à linguagem do usuário.

• **Controle e liberdade de escolha do usuário** – Oferecer recursos como Undo que permita o usuário desfazer ações realizadas e retornar, por exemplo, a revisão anterior de um documento.

• **Consistência e aderência a padrões** – Seguir recomendações dadas em guidelines e guias de estilo, visando prover suporte à consistência.

• **Prevenção de erros** – Identificar e eliminar situações que possam levar a erros do usuário.

• **Flexibilidade e eficiência de uso** – Considerar a diversidade de usuários (novatos e experientes), provendo mecanismos apropriados, como o uso de teclas de atalho, ícones e menus.

• **Estética e projeto minimalista** - Não adicionar informações desnecessárias ou raramente utilizadas, que competem com informações relevantes.

• **Prover ajuda aos usuários de reconhecimento, diagnóstico e recuperação de erros** – Mensagens de erro devem ser construtivas, sugerindo solução para o usuário.

• **Ajuda e documentação** – Prover documentação e recursos de ajuda, facilitando a busca e com foco nas tarefas do usuário.

Figura 1. Heurísticas de Usabilidade propostas por Jakob Nielsen.

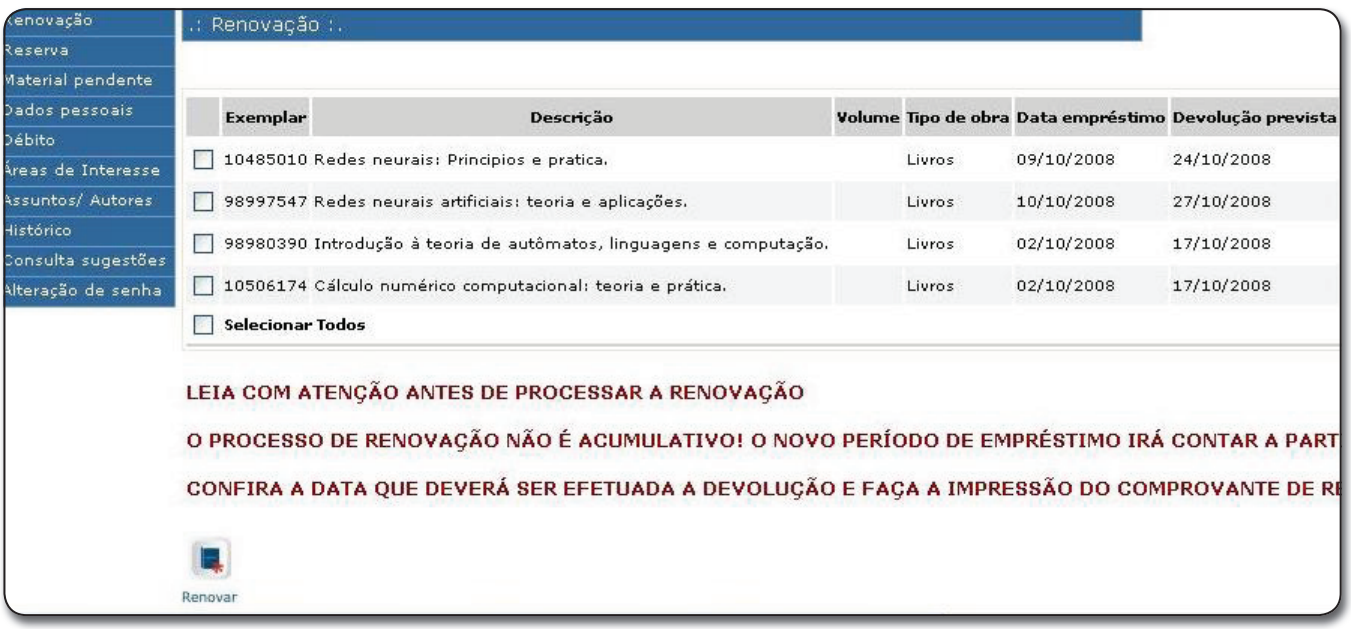
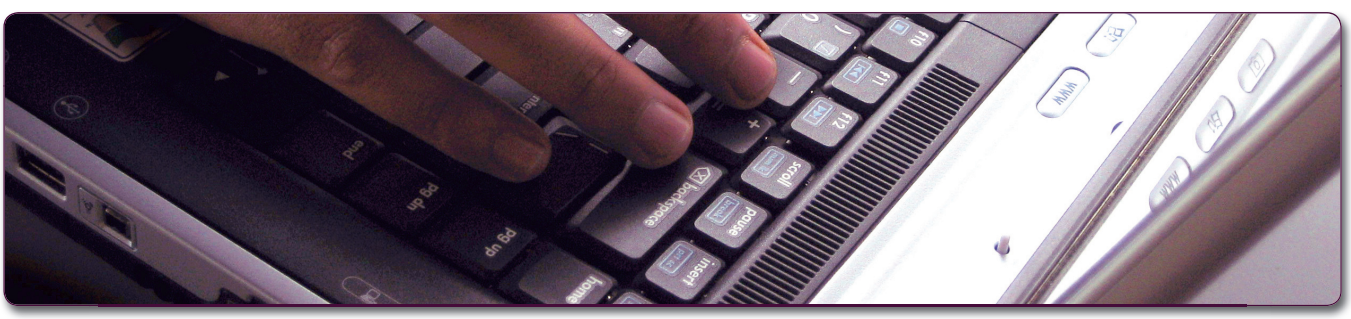


Figura 2. Exemplo de renovação de empréstimo em aplicação Web.



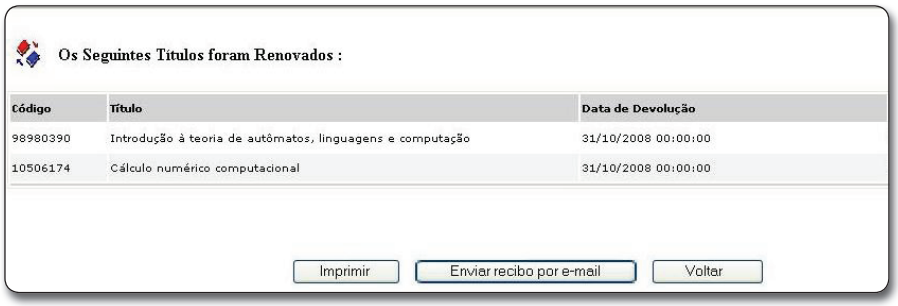


Figura 3. Tela da aplicação após renovação de empréstimo.

Figura 4. Exemplo de feedback não adequado.

Figura 5. Exemplo de casamento não adequado com o mundo real

Figura 6. Resposta a ação de buscar

usabilidade durante o projeto. As heurísticas compreendem conjunto de diretrizes, geralmente, que formam o conhecimento sobre para o tratamento de determinado problema. Essas heurísticas servem de critério na avaliação da usabilidade de um web site, como discutido neste artigo.

Heurísticas de Usabilidade

Heurísticas de usabilidade são princípios resultantes da experiência e, geralmente, aceitos que são aplicados no desenvolvimento de sistemas interativos. Elas também são empregadas durante a avaliação de produtos como web sites.

Este artigo faz uso de um conjunto de dez heurísticas, referenciados no quadro de links, propostas por Jakob Nielsen para o projeto de interface de usuário. Essas dez heurísticas de usabilidade, consideradas como princípios para o projeto de interface de usuário, são apresentadas na **Figura 1**.

As heurísticas apresentadas na **Figura 1** servem a uma ampla variedade de sistemas de software. A seção seguinte faz uso dessas heurísticas para identificar problemas de usabilidade em sites. O objetivo aqui é apenas o de identificar problema de usabilidade (que pode comprometer a facilidade de uso e desempenho de usuários) quando usando um site. O leitor é referenciado ao artigo que trata do método de avaliação heurística, publicado na edição anterior dessa revista.

Avaliação Heurística de Web Sites

A avaliação heurística é um método de avaliação de usabilidade de sistemas e produtos como, por exemplo, web sites que permite o avaliador detectar problemas de usabilidade. Esses problemas, em geral, estão relacionados com alguma das dez heurísticas de usabilidade apresentadas anteriormente ou outra recomendação de projeto de empresas

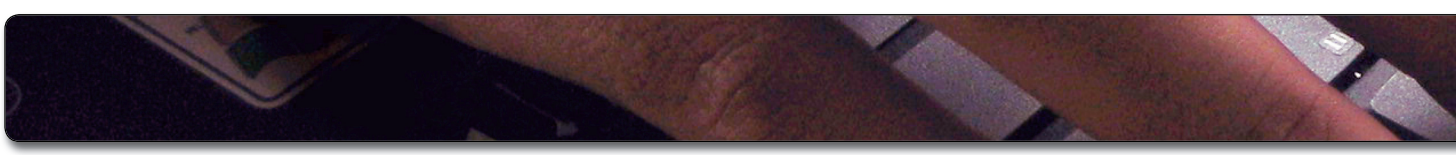


Figura 7. Opções de navegação em site de busca.

(Apple, Microsoft e Sun Microsystems) destacados em links deste artigo. Problemas vinculados às heurísticas destacadas na **Figura 1** são apresentados a seguir.

Visibilidade do estado do sistema

Qualquer usuário quando se encontra utilizando um Web site precisa ter um *feedback* adequado do que está acontecendo (isto é, da tarefa que está executando), pois do contrário este usuário ficará confuso sem saber se a tarefa que estava realizando foi terminada com sucesso ou não. Mesmo quando a tarefa não é encerrada com sucesso, ainda assim o usuário precisa de um *feedback*. Considere a **Figura 2** que ilustra parte de uma aplicação Web que permite ao usuário renovar remotamente livros emprestados de um biblioteca. O lado esquerdo da figura destaca duas caixas do tipo *checkbox* que permite o usuário selecionar para renovação de empréstimo. Além disso, no canto inferior esquerdo, há um botão no qual o usuário clica para efetuar a renovação.

Após a renovação dos livros, a aplicação mostra o conteúdo exibido na **Figura 3**, onde o usuário é informado que os títulos foram renovados e a nova data de devolução.

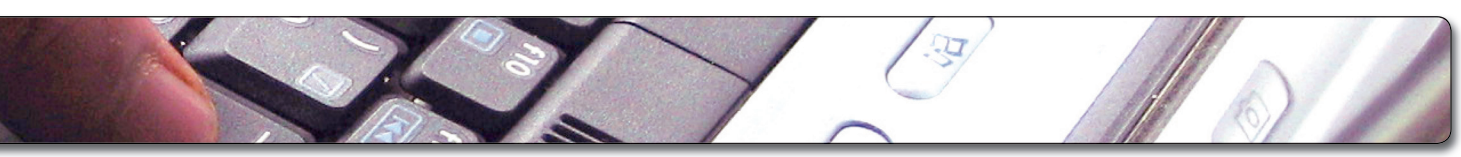
Observe que o usuário tem a opção de imprimir um comprovante, ter um comprovante enviado para seu e-mail cadastrado ou voltar à tela anterior. Entretanto, se o usuário optar pelo envio de e-mail o botão é desabilitado, como ilustrado na **Figura 4**, mas nenhuma informação adicional é dada ao usuário, informando-o de que o e-mail foi enviado. O usuário apenas saberá se um comprovante foi enviado se ele acessar seu cliente de e-mail para checar pela referida mensagem. Portanto, a aplicação não fornece qualquer feedback ao usuário de modo que ele possa saber (com certeza) de um e-mail contendo comprovante do recibo tenha sido enviado a ele. Um único feedback dado ao usuário é a desabilitação do botão "Enviar recibo por e-mail".

Figura 8. Uso inadequado do "Clique aqui".

Figura 9. Uso da janela de "Busca" em site de conteúdo.

Casamento do sistema com o mundo real

No projeto de interfaces e de sites, o projetista deve fazer uso de termos,



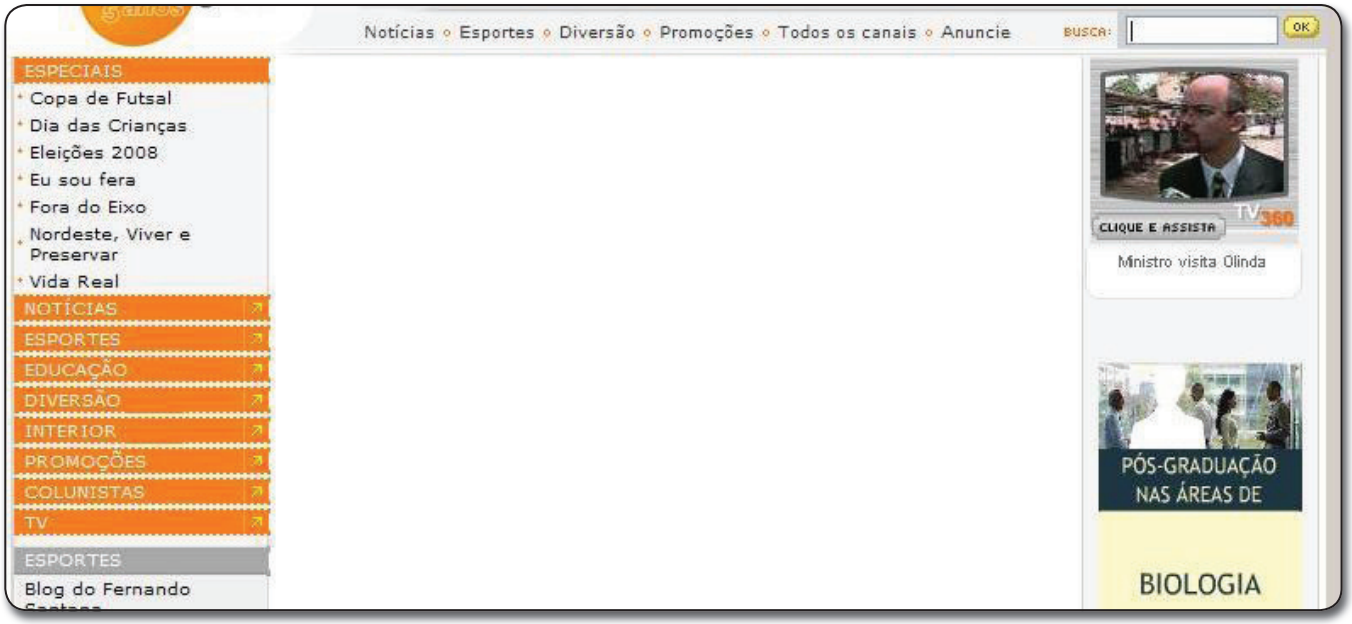
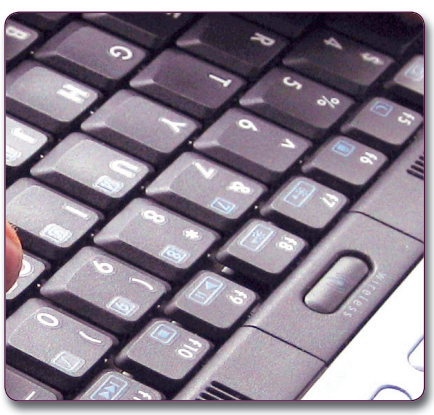


Figura 10. Resultado exibido na busca do site de conteúdo.

Figura 11. Localização inadequada do mapa do site.



conceitos, objetos (como, por exemplo, ícones) familiares à linguagem do usuário de modo a tornar mais intuitivo o significado das palavras e figuras. Isto permite ao usuário identificar mais facilmente como e onde realizar a tarefa desejada. Considere a **Figura 5** que ilustra parte de um site de uma imobiliária. Esta parte destacada captura uma das funcionalidades do site que permite a busca de imóveis.

O ícone da lupa, destacado na **Figura 5**, não permite qualquer ação do usuário, já que esse ícone está desabilitado. Usuários tentam clicar no referido ícone sem terem sucesso. Além disso, o botão "DETALHAR" leva o usuário para outra tela na qual ele pode detalhar uma busca, enquanto que o botão "BUSCAR" traz como resultado um conjunto de imóveis, mesmo que o usuário não tenha especificado qualquer coisa, como mostra a **Figura 6**.

O problema destacado neste site é o uso de objetos (ícones) desabilitados que não permite ação do usuário. Note que o usuário poderia ter duas opções (busca rápida ou busca detalhada). Esse não casamento com o mundo real, ou seja, aquilo que seria esperado pelo usuário (de modo mais intuitivo), força-o a descobrir como poderia obter a informação desejada.

Controle e liberdade de escolha do usuário

As interfaces de usuário de sistemas de software devem possibilitar o usuário desfazer ações que tenham realizado sempre que necessário. No caso de Web sites, por exemplo, o usuário deve ter a liberdade de escolha ou de navegação no site, sem que a ele seja imposta qualquer restrição de sair do site. Essa não é uma boa prática de projeto, pois isso muitas vezes irrita o usuário, que tenta voltar ou retornar ao

Figura 12. Mapa do site sem texto explicativo.

site anterior que o redirecionou, mas sem sucesso.

Adicionalmente, o projetista pode adicionar recursos (botões na interface) que possibilita o usuário mais facilmente navegar numa página ou num conjunto de páginas resultantes de um processo de busca como ilustrado na **Figura 7** que ilustra parte de um site de busca. O mesmo pode ser feito em site de conteúdo que contém ícones ou botões que permitem o usuário retornar para a página principal.

Consistência e aderência a padrões

Uma boa prática de projeto de interface de usuário é seguir as recomendações de projeto, também conhecidas como

Figura 13. Mapa do site sem texto explicativo.

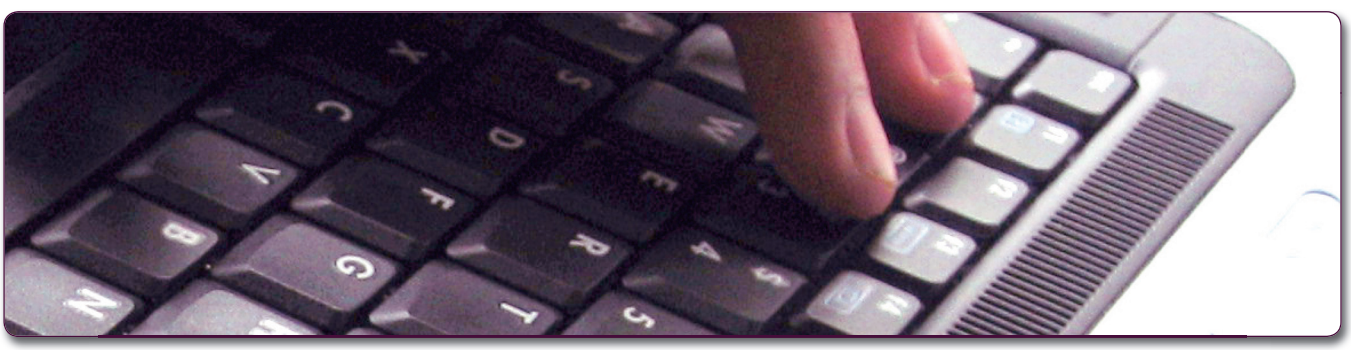


Figura 14. Falta de mapa do site como recurso de ajuda.

user interface guidelines. Observe que é natural aos usuários ficarem confusos quando eles se deparam com situações de inconsistências no projeto da interface. Portanto, isso deve ser evitado. Usuários retornam ou recomendam um Web site quando eles são capazes de facilmente localizar as informações procuradas e quando há consistência na forma em que o conteúdo do site é apresentado aos usuários.

Considere, como exemplo, uma recomendação conhecida que sugere o projetista evitar fazer uso do termo "Clique aqui" como âncora para links. Esse é um problema menor de usabilidade, já que apenas um texto curto representativo do conteúdo deveria ser usado como âncora para um link. Um exemplo desse problema é encontrado na página de uma imobiliária mostrada na **Figura 8**. Para este exemplo, o projetista poderia ter usado apenas o texto "Consulte imóveis para aluguel". No entanto, ele faz o uso não recomendado do termo "Clique aqui e consulte imóveis para aluguel".

Prevenção de erros

Os projetistas de interface devem prevenir erros de usuários. Para tanto, eles procuram identificar e eliminar situações que possam levar os usuários a cometerem erros. Considere, por exemplo, a **Figura 9** que ilustra parte de um site de conteúdo com diversas informações. O site contém uma janela que permite ao usuário efetuar uma busca como destacado na figura.

Contudo, ao digitar qualquer palavra(s) para consulta, nada é retornado ao usuário, como ilustrado na **Figura 10**. E, se quaisquer outras palavras, mesmo utilizando-se de palavras de conteúdo já apresentado no próprio site, ainda assim nada é retornado. Apenas será exibido um grande espaço em branco sem qualquer conteúdo, como mostrado na figura.

Cabe destacar que nem sequer é dada a informação ao usuário de que o site não tem qualquer informação daquele conteúdo buscado. Se essa funcionalidade não está disponível, o usuário deveria ser informado, visando evitar esse tipo de situação.

Flexibilidade e eficiência de uso

Os projetistas de Web sites devem considerar a diversidade de usuários que compreendem novatos e experientes. Dessa forma, recomenda-se prover os usuários com maior flexibilidade nas formas de acesso às informações e

funcionalidades. Pode-se destacar, em locais mais proeminentes da interface, a localização de objetos de interação como ícones e botões de acesso a funcionalidades. A **Figura 11** ilustra parte de um site de uma instituição de ensino na qual o usuário não encontra o mapa do site facilmente. Geralmente, o mapa do site é posicionado no canto superior direito da interface. Após observar todo o site, o usuário poderá encontrar o mapa do site no canto inferior esquerdo em local não proeminente, como destacado na figura.

Vale ressaltar que o projetista colocou um ícone e ao lado o título indicativo da funcionalidade, visando facilitar o entendimento do significado do ícone, que nem sempre é intuitivo.

Isso, contudo, não foi considerado pelo projetista do site, mostrado na **Figura 12**. Este Web site de outra instituição não apresenta qualquer rótulo ou pequeno texto explicativo para o referido ícone, indicado na **Figura 12**. Se um rótulo fosse adicionado ou se uma janela com texto explicativo fosse apresentada ao usuário quando ele posicionasse o mouse sobre o referido ícone, ele teria um melhor compreensão do Web site e poderia mais facilmente localizar a informação procurada.

Estética e projeto minimalista

Simplicidade é uma palavra chave no projeto de interface de usuário e, para isso, os projetistas devem considerar o projeto minimalista, no qual ele coloca apenas no primeiro plano (isto é, na página principal) as informações mais relevantes. Essa prática é recomendada porque tudo o que é colocado em primeiro plano, ou na página principal, compete pela atenção do usuário. Portanto, recomenda-se não adicionar informações desnecessárias ou raramente utilizadas ou consultadas pelos usuários, pois estas competem com informações relevantes.

Um exemplo disso é mostrado na **Figura 13** que ilustra parte de um site no qual há um excesso de informações e essas são distribuídas de maneira desordenada, além de uso excessivo de cores. E, observe que a figura mostra apenas parte do conteúdo do site. Isso é um exemplo do que não se deve fazer em termos de uso cores, da enorme quantidade de informações (competindo pela atenção do usuário) e uso de ícones nada representativos.

Prover ajuda aos usuários para reconhecimento, diagnóstico e recuperação de erros

Usuários precisam ser informados de maneira adequada em situações de erro e indisponibilidade de serviços. Esse tipo de mensagem deve ser fornecida ao usuário visando orientá-lo de como proceder numa situação de erro ou notificando-o quando da não disponibilidade de serviço. Um exemplo disso foi mostrado na **Figura 10** onde nenhum conteúdo da busca é retornado e qualquer outra mensagem é dada ao usuário informando-o da indisponibilidade do serviço de busca.

Ajuda e documentação

Prover recursos de ajuda que facilitem o acesso a informações e funcionalidades em uma aplicação ou Web site é uma necessidade. Isso pode ser na forma de documentação de ajuda como em "respostas a perguntas freqüentes", também conhecida com FAQ (*Frequently Asked Questions*), ou em mapas de sites. É uma boa prática manter um foco nas tarefas que o usuário pode precisar e, portanto, prover esses recursos de apoio.

A **Figura 14** mostra parte de um site de uma instituição de ensino que não fornece o recurso de mapa do site, essencial a sites que contêm grande quantidade de informações. Embora o usuário pudesse pensar e checar o ícone contendo um sinal de + no canto superior direito do site, como mostrado na figura. Esse ícone é usado para busca avançada.

Comentários Finais

As heurísticas de usabilidade apresentadas neste artigo tratam de problemas de usabilidade que podem ser encontrados em Web sites, como em outros produtos. Essas heurísticas servem como recomendações e, portanto, requer a atenção do projetista no desenvolvimento de interface de um Web site ou outro produto. As heurísticas estão diretamente relacionadas a necessidades do usuário quando interagindo com uma aplicação. Considerá-las no projeto é manter o foco no usuário. Utilizar essas heurísticas na avaliação é dar oportunidade ao avaliador de checar se a usabilidade está sendo suportada e também fornecer a oportunidade de corrigir problemas identificados. ●

Links

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

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

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

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

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

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

Teste de Desempenho de Aplicações Web com Apache JMeter

Vinícius Rodrigues de Souza
vrsouzainfo@gmail.com
É graduando em Sistemas de Informação pela 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
É graduando em Sistemas de Informação pela Faculdade Metodista Granbery e estagiário na Prefeitura de Juiz de Fora na área de desenvolvimento e testes de software.

Marco Antônio Pereira Araújo
maraujo@granbery.edu.br
É Doutorando e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Matemática com Habilitação em Informática pela UFJF, Professor do Curso de Bacharelado em Sistemas de Informação 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 configuração e utilização da ferramenta Apache JMeter, capaz de executar testes de desempenho em sistemas baseados na Web, a fim de se antecipar a possíveis problemas de sobrecarga na utilização do software.

Para que serve:
Apache JMeter é uma aplicação desenvolvida totalmente em Java que auxilia na geração de testes de desempenho para aplicações Web. Ela é capaz de simular acessos simultâneos na aplicação e possibilita a visualização dos resultados para avaliação do desempenho por meio de gráficos e tabelas.

Em que situação o tema é útil:
O intuito desse processo é assegurar que a arquitetura desenvolvida para atender a uma solução realmente consiga suportar a quantidade de usuários previstos para acessar o aplicativo, sendo possível mensurar alguns atributos determinantes para um bom funcionamento do sistema, tais como consumo de memória e uso de CPU dos servidores, nível de tráfego na rede e tempo de resposta.

Atualmente, há uma exigência cada vez maior quanto à boa qualidade e conseqüente confiabilidade dos softwares produzidos. Na busca dessas características necessárias, existem etapas importantes no ciclo de desenvolvimento de software que devem ser observadas de perto, dentre elas, a fase de testes. Existem vários tipos de teste de software, que abrangem desde o levantamento de requisitos até o fim da fase de implantação do sistema.

Nesse artigo faremos um estudo de caso a fim de demonstrar um dos tipos de teste de software, o teste de desempenho, através da configuração e utilização da ferramenta Apache JMeter. A versão da ferramenta a ser abordada neste artigo será a 2.3.2, e pode ser encontrada para download no site http://jakarta.apache.org/jmeter. Para utilizar a JMeter, deve-se observar

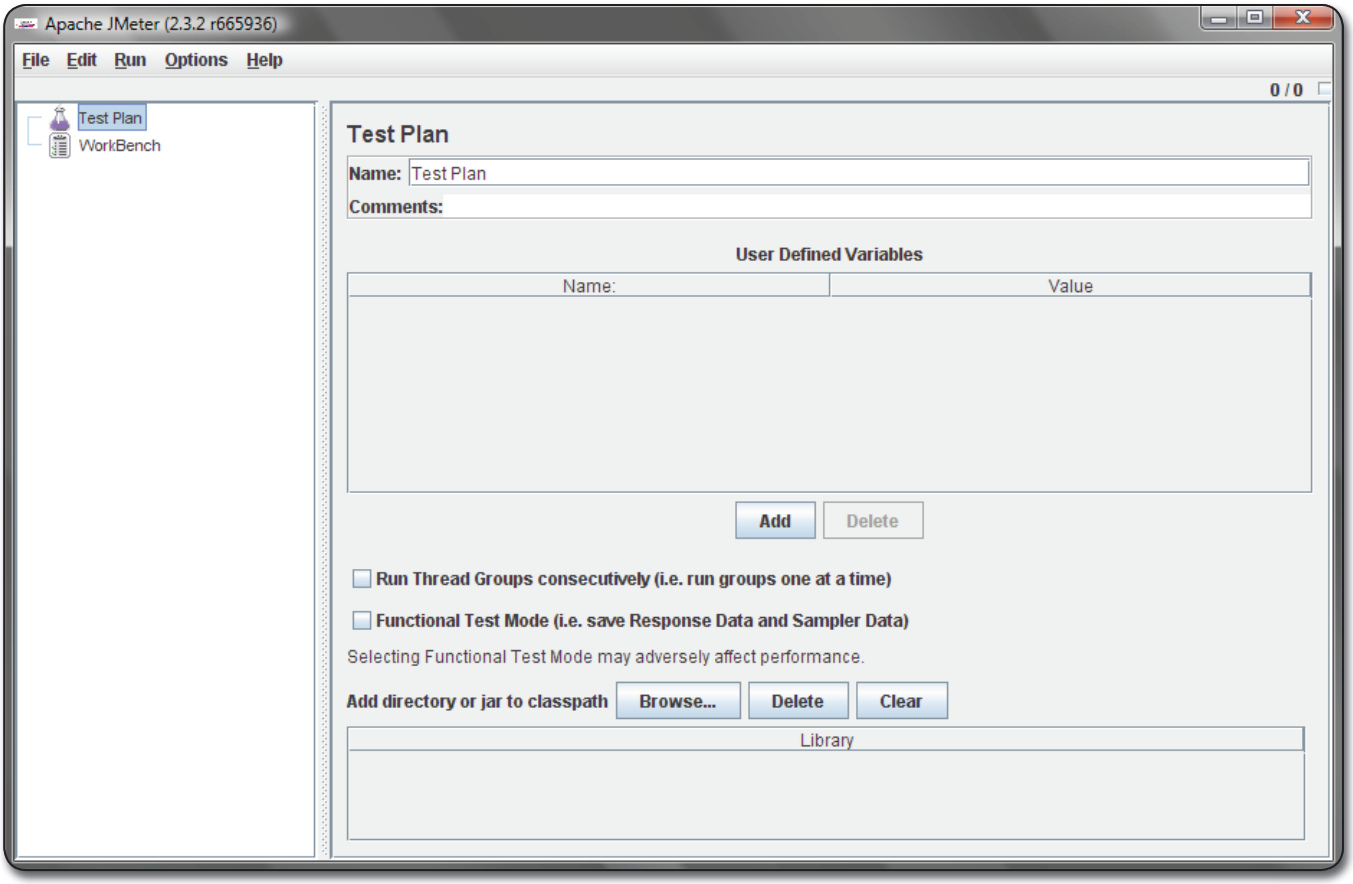


Figura 1. Visualização da tela inicial do sistema.

os requisitos mínimos, devendo estar instalado a JVM (Java Virtual Machine). Essa versão da JMeter suporta testes de desempenho em aplicações Web (HTTP/HTTPS), FTP, JDBC, LDAP, Java e JUnit.

Após o download do arquivo Zip, basta descompactar e abrir a pasta bin, clicando no executável chamado ApacheJMeter.jar. A janela inicial da ferramenta, além do menu superior, já conta com uma árvore com dois elementos principais: *Teste Plan* e *WorkBench* (ver **Figura1**).

O *WorkBench* é uma área onde os itens existentes não são considerados como parte de um plano de testes em particular. São armazenados temporariamente, e não são salvos no momento que o plano de teste é salvo, tendo função de apoio à elaboração de planos de testes.

O *TestPlan* é onde são definidos todos os testes que irão ser executados. Agrupa todos os elementos possíveis de configuração dos *samplers*, tais como: *controladores*, *listeners*, *assertions*, dentre outros, que serão abordados ao longo do artigo. O *TestPlan* pode ser configurado para que as *threads* sejam executadas de maneira seqüencial ao invés de simultaneamente, selecionando "Run Thread Groups consecutively", e é possível a configuração da ferramenta para testes de caixa preta selecionando "Functional Test Mode".

Para o início da configuração dos testes, deve-se incluir elementos ao *TestPlan*. São eles:

Thread Group: simulará os usuários que irão executar os testes. Nele podem-se configurar quantos usuários irão fazer as requisições, o tempo total do grupo de requisições e quantas vezes o teste será executado. Ao selecionar a opção "Scheduler" pode-se ainda indicar a hora que irá iniciar e terminar o teste. É no Thread Group que podem ser incluídos os elementos **Sampler** que farão as requisições físicas de um determinado servidor. Podem–se ainda

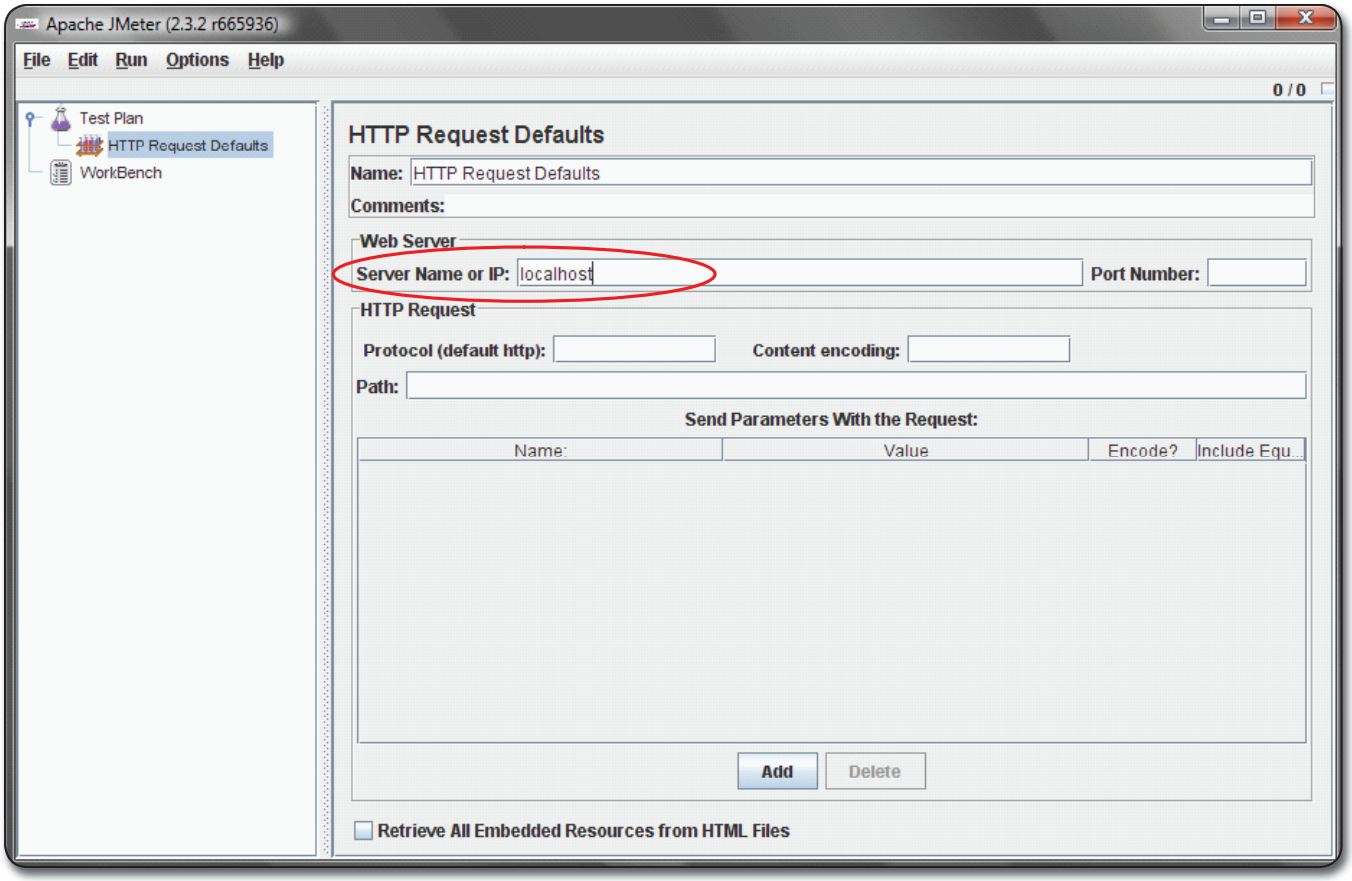


Figura 2. Janela de Configuração do HTTP Request Defaults.

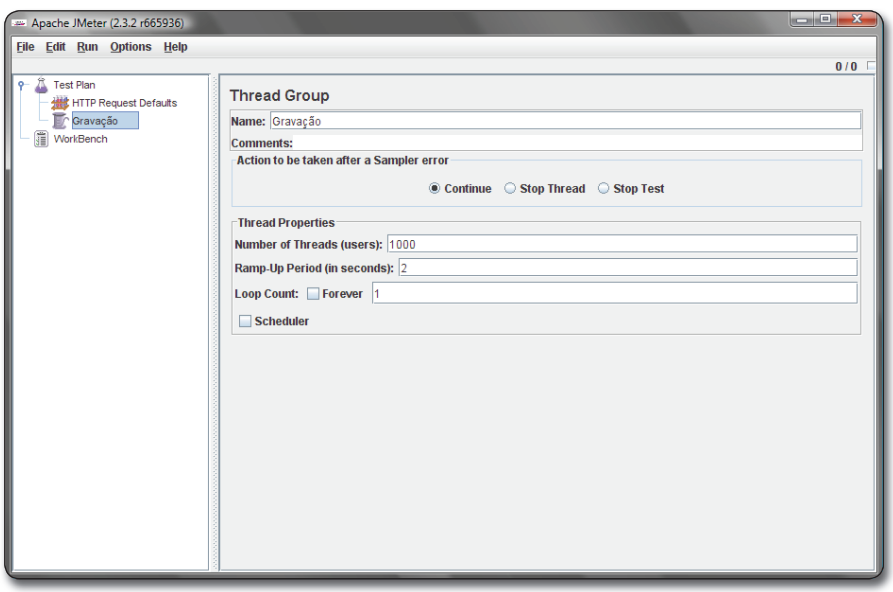


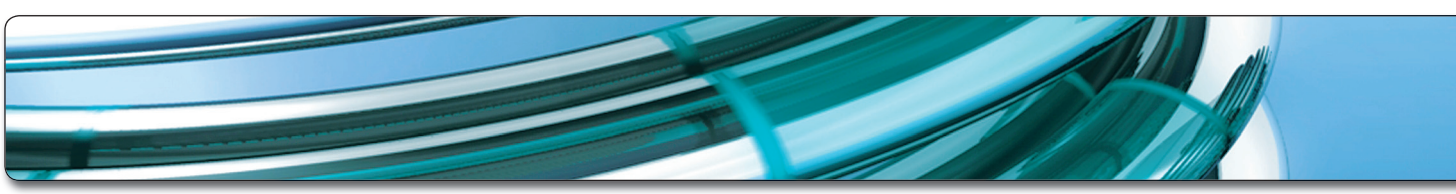
Figura 3. Janela de Configuração do Thread Group.

incluir ao Thread Group os elementos **Logic Controller** que permitem testes customizados, como usados para fazer randomização ou criar laços.

Config Element: categoria de elementos de configuração do plano de testes, podendo configurar variáveis que posteriormente serão utilizadas pelo Sampler.

Timer: responsáveis pelo controle mais preciso no tempo de execução dos testes, estabelecem um intervalo padrão ou aleatório entre as threads ou até delays entre as threads.

Pre Processors: são elementos que processam um dado antes de acionar um Sampler. São usados para modificar um Sampler do mesmo escopo. Podem também ser usados para gerar dados dinâmicos.



Assertions: aplicados para conferir se os dados estão de acordo com o previsto pelo Sampler, como encontrar determinado texto, ou se as requisições têm que retornar em determinado tempo especificados em um elemento Assertion.

Post Processors: aplicados após um Sampler e servem para extrair dados de resposta de uma requisição.

Listener: capturam os resultados gerados pelo plano de testes e apresenta-os em um determinado formato escolhido. Os dados podem ser visualizados por meio de gráficos ou por meio de tabelas que informam o que ocorreu durante o teste, indicando o tempo gasto para a requisição ser feita, ou se ocorreu algum erro durante o teste.

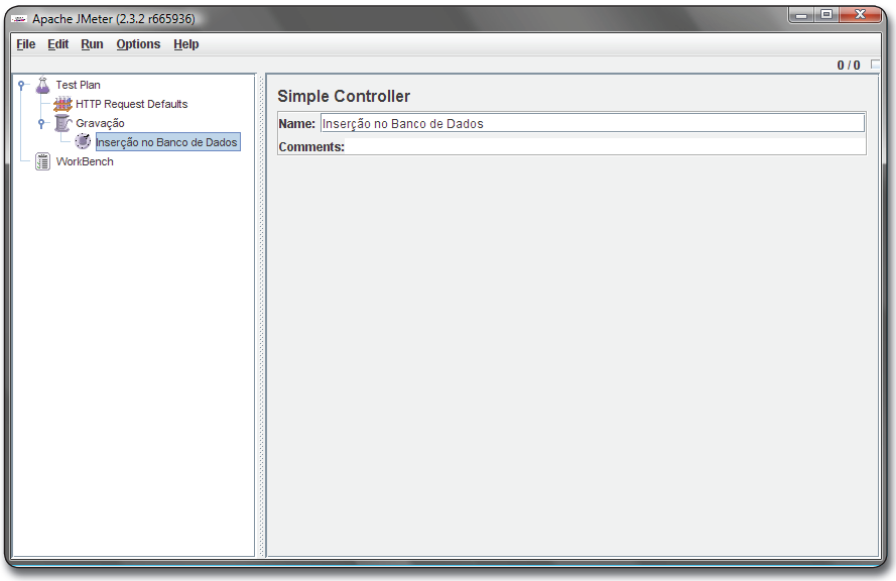


Figura 4. Tela do Simple Controller.

Criando um teste

Para exemplificar a utilização da ferramenta, será utilizado um aplicativo Web previamente construído com acesso a um banco de dados. Primeiramente, serão configurados testes para serem aplicados neste sistema que acessa o banco de dados e grava os dados de um formulário contendo nome e email.

Inicialmente, clicando com o botão direito sobre *TestPlan*, adiciona-se um *HTTP Request Defaults*, localizado dentro do grupo *Config Element*. Este elemento serve para evitar que se adicionem vários *HTTP Request* com as mesmas configurações de servidor. Todas as requisições dos testes serão feitas a um mesmo servidor já configurado neste elemento. Como estamos utilizando um servidor local, configura-se o campo "Server name or ip" com "localhost" (ver **Figura 2**).

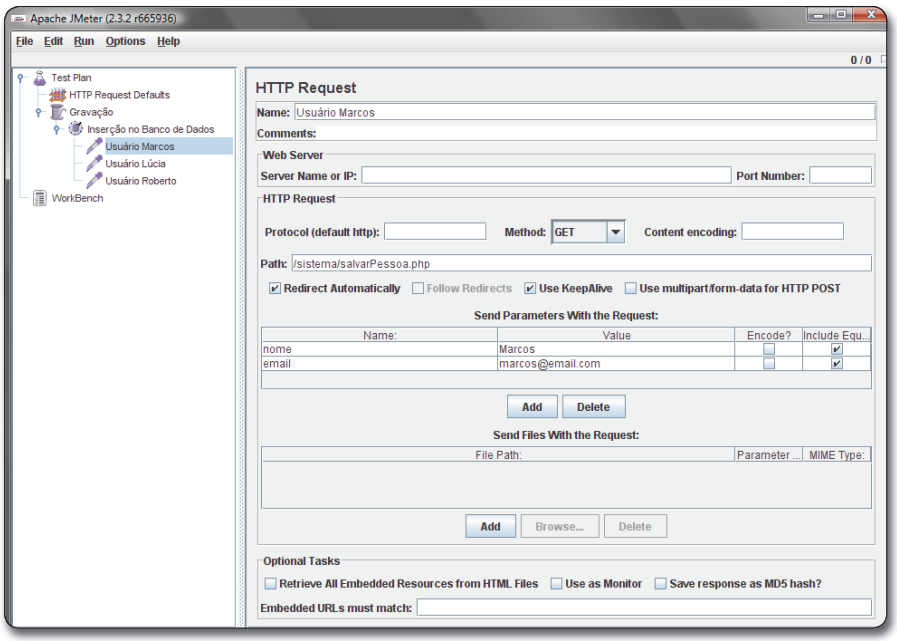


Figura 5. Configuração do HTTP Request *Usuário Marcos*.

Em seguida adicionamos um *Thread Group*. Na janela que se abrirá, definimos o nome desse grupo de teste para "Gravação", a quantidade de usuários que farão as requisições em 1000, e o tempo total dessas requisições para 2 segundos. Não serão configurados os horários de início e término do teste, que será disparado manualmente (ver **Figura 3**).

Neste teste, serão utilizados três *HTTP Request* para simular o cadastro de três registros diferentes, e como característica comum, somente o campo "Path" com "/sistema/salvarPessoa.php", que representa a página da aplicação a ter seu desempenho testado. Este elemento é responsável por

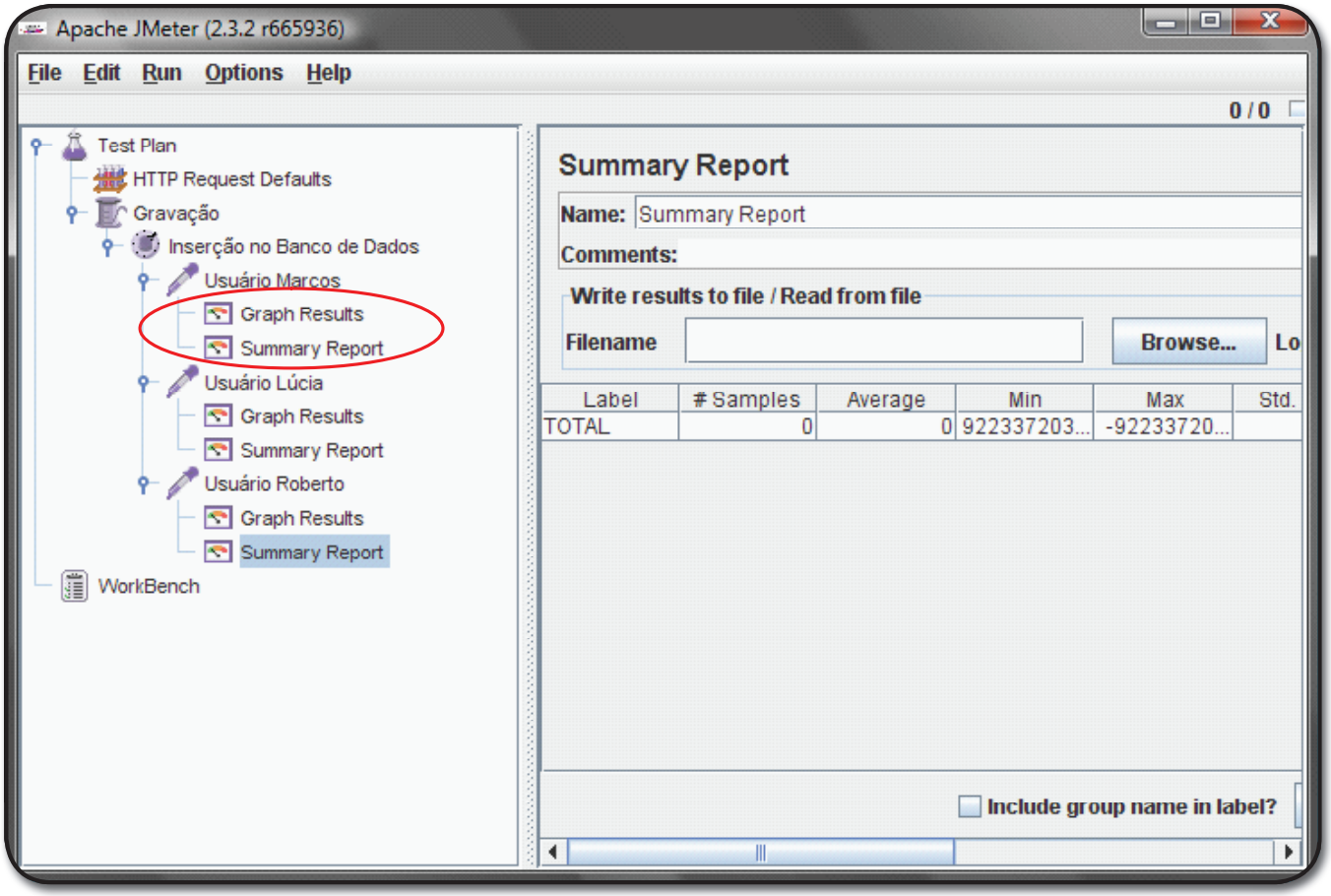


Figura 6. Listeners individuais.

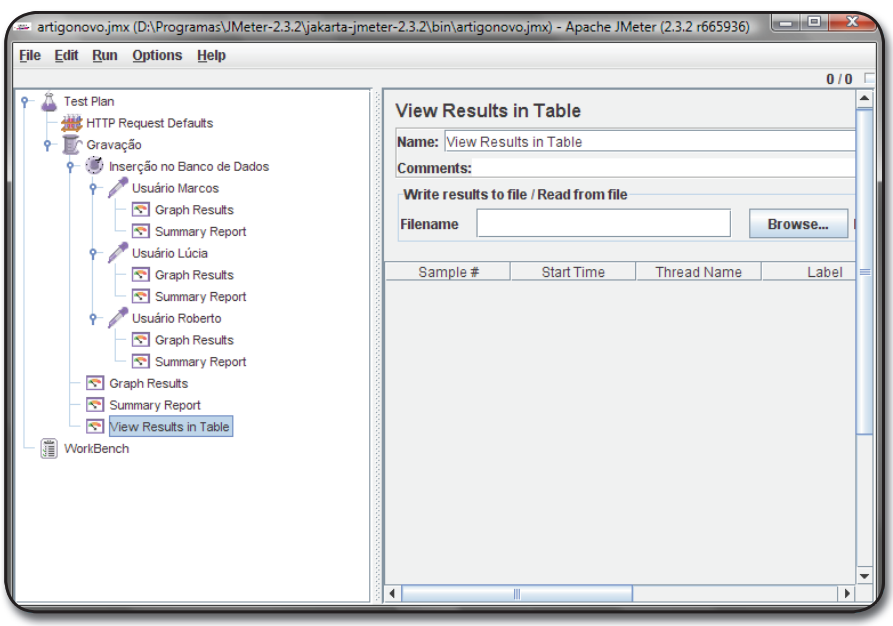


Figura 7. *Listener* de resultados em conjunto.

configurar características de cada tipo de requisição, incluindo o caminho de cada diretório do arquivo específico a ser testado. O grupo *Logic Controller* tem como função controlar as execuções das requisições dentro de um *Thread Group* de forma personalizada. Possui controladores lógicos que permitem a criação de laços, modularização e randomização. Para agrupar esses elementos HTTP Request citados, utilizaremos o *Simple Controller*, localizado dentro do grupo *Logic Controller*, com o nome de "Inserção no Banco de Dados". Ele tem simplesmente a função organizacional. Cada requisição vai conter seu rótulo correspondente e seus parâmetros específicos para serem passados para a página em questão, definidos na janela com o título "Send Parameters With The Request". Simularemos o cadastro de três usuários quaisquer, com os nomes definidos aleatoriamente por Marcos, Lúcia e Roberto. No caso, todos os parâmetros

serão passados por método "GET", que também deve ser definido na caixa de seleção (ver **Figuras 4** e **5**).

Para cada *HTTP Request* configurado, utilizaremos como visualizadores os elementos *Graph Results* e *Summary Report*. O *Graph Results* contém um gráfico Execuções X Tempo, exibindo os resultados, a média, a mediana, os sucessos e os erros nas requisições. O *Summary Report* exibe os mesmos resultados, mas de maneira mais específica, apresentando os valores coletados no teste (ver **Figura 6**).

Todos os *listeners* adicionados anteriormente estão configurados para funcionar de maneira individual, mas o JMeter permite também a observação de resultados em conjunto, por isso, como filhos do elemento *ThreadGroup*, mais três listeners são adicionados: *Graph Results*, *Summary Report* e *View Results in Table*, que listam numa tabela, os resultados de todas as requisições individualmente. Isso permite saber em que ponto exato ocorreu uma possível falha e obter suas características (ver **Figura 7**).

Depois de feito esse processo, a parte de configuração do teste está concluída. Partimos agora para a execução dos testes. No menu superior, há o item "Run", dentro dele a opção "Start". Ao clicar, os testes irão iniciar e durar o tempo previamente determinado, no caso, dois segundos.

Durante o período de execução, os resultados são mostrados em tempo real, e é possível acompanhar todo o processo.

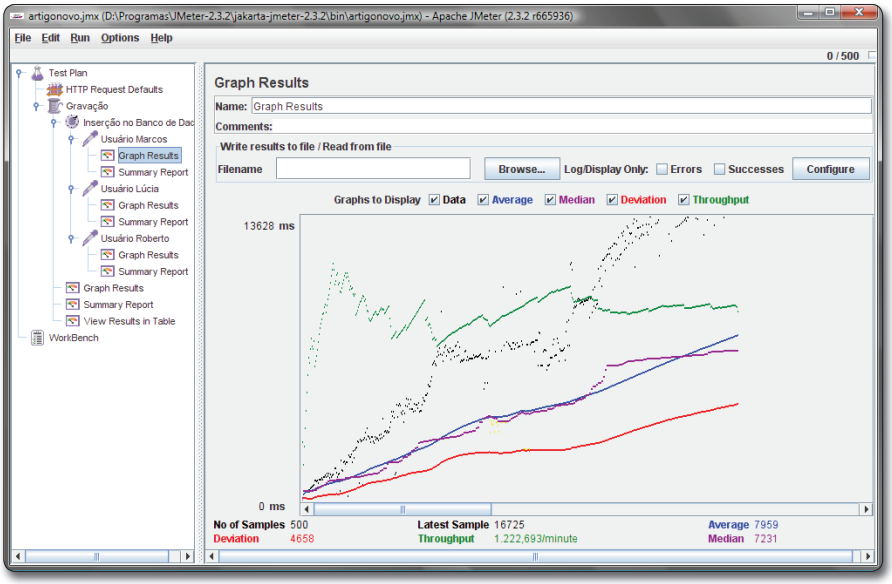


Figura 8. Graph Results referente ao HTTP Request *Usuário Marcos*.

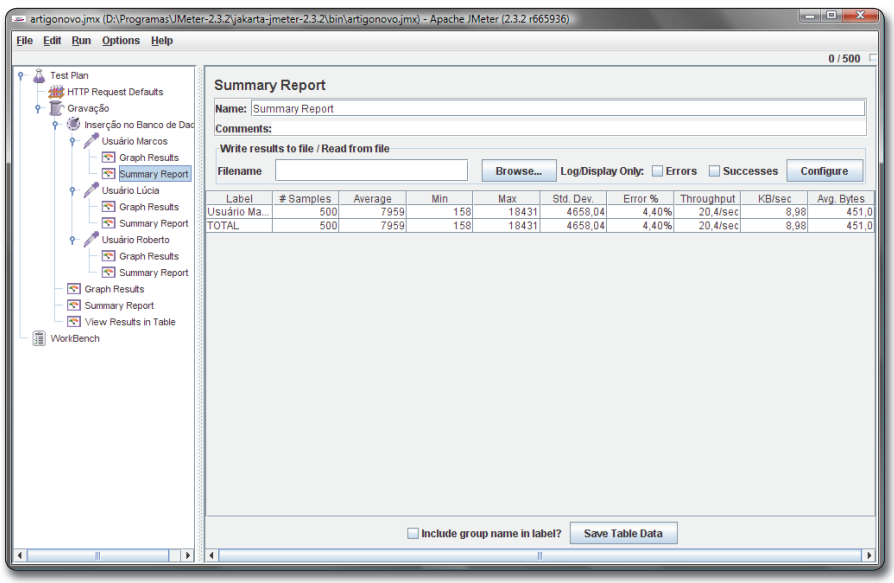


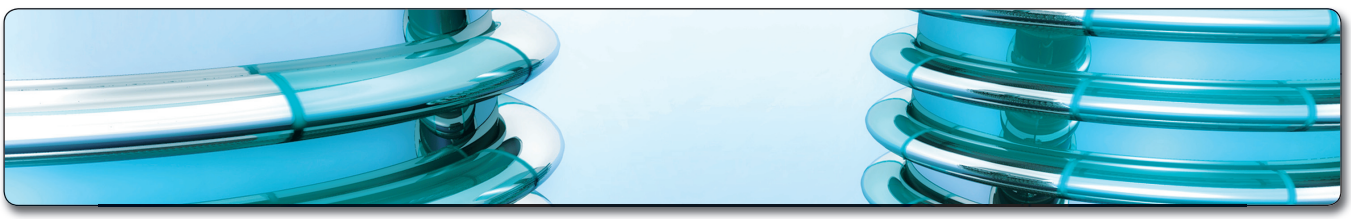
Figura 9. Summary Report referente ao *HTTP Request Usuário Marcos*.

Avaliando os Resultados

Após o teste ter sido realizado, os elementos *Listener* irão exibir os dados correspondentes para todos os *HTTP Request* configurados, e ainda os resultados em conjunto. Analisaremos os *Listeners* correspondentes às requisições feitas ao banco.

Para as requisições feitas ao banco, analisaremos primeiro os elementos *Graph Results* e *Summary Report* (ver **Figuras 8** e **9**). Esses elementos exibirão os dados correspondentes somente ao *HTTP Request Usuário Marcos*. Note que no *Graph Results* existem várias linhas de diferentes cores. Essas linhas representam dados como média (Average), mediana (Median), desvio padrão (Deviation) e o mínimo e máximo do tempo de resposta das requisições, sendo possível então analisar visualmente o comportamento da aplicação em relação ao desempenho apresentado.

O *Summary Report* nos mostra de forma mais precisa o que ocorreu durante as requisições do *HTTP Request Cadastro*



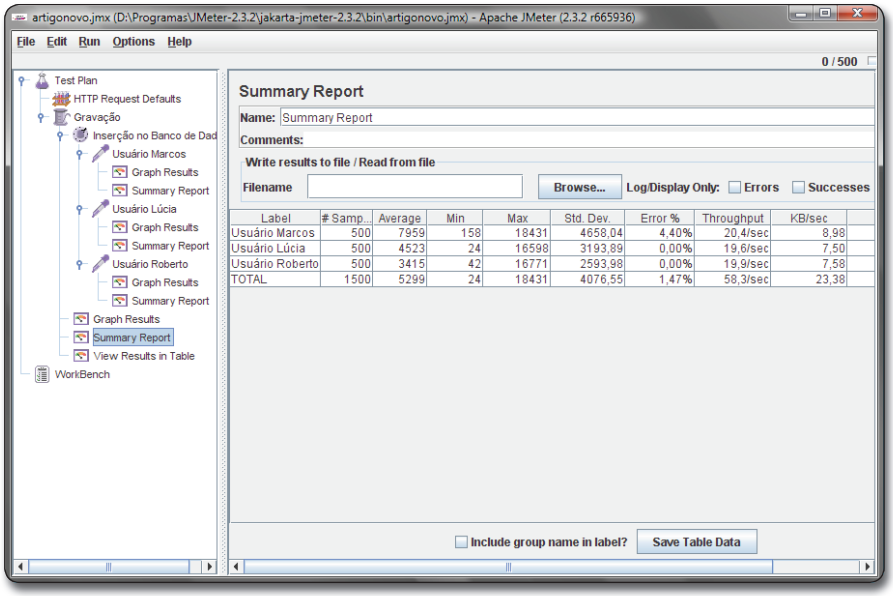


Figura 10. *Summary Report* referente a todos os *HTTP Request* do teste.

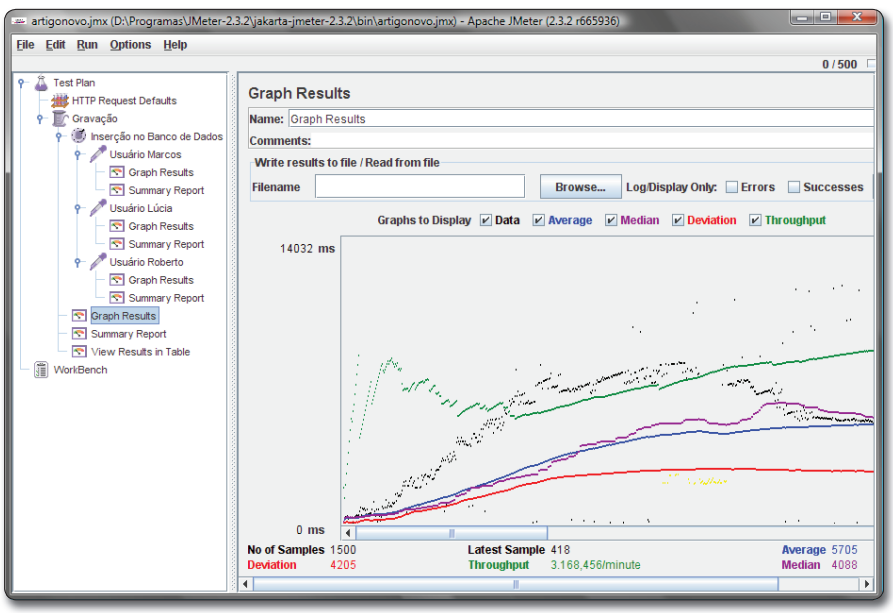


Figura 11. *Graph Result* referente a todos os *HTTP Request* usados.

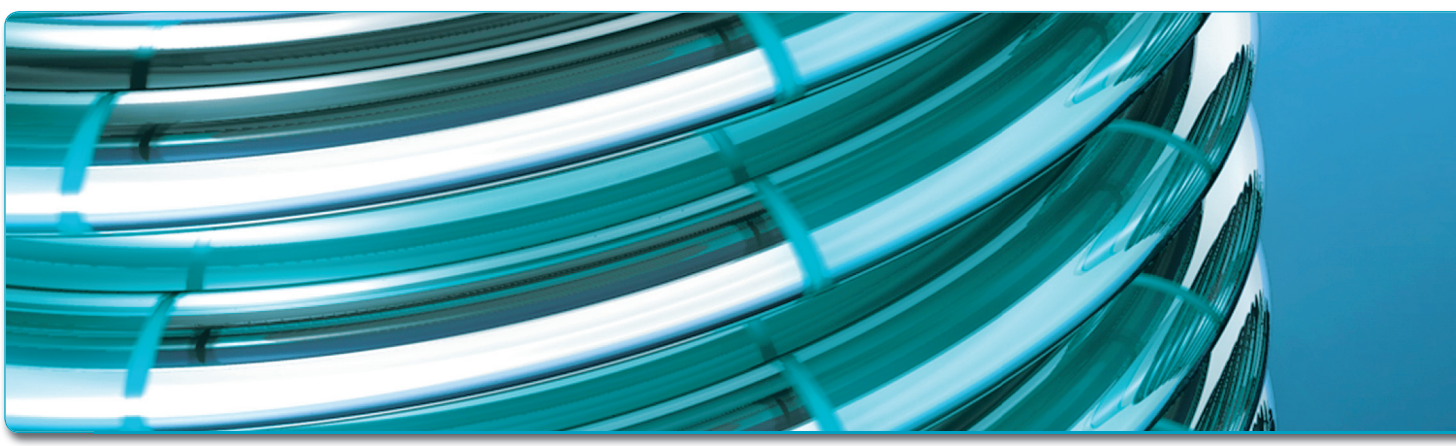
Joao. Nesse exemplo ele mostra que a média foi de 7959 milissegundos e o desvio padrão de 4658 milissegundos. O tempo, medido em milissegundos, gasto foi no mínimo de 158 e no máximo de 18431. Exibe também que houve 4,4% de erro nas execuções das requisições realizadas pelo teste, que pode ser, inclusive, da incapacidade do servidor de processar este número de requisições dentro da faixa de tempo estipulada.

Tomando como base os dados referentes dos dois elementos *Listeners*, podemos concluir que foram feitas muitas gravações em período muito curto de tempo, mas o teste não ficou isento de erros. Isso pode ter sido causado por vários fatores, dentre eles: configuração de máximo de conexões simultâneas no banco de dados, demora no processamento por parte do servidor, além de problemas físicos.

Agora iremos analisar os *listeners* adicionados como filhos do *Thread Group*. Esses elementos irão exibir os dados correspondentes a todos os *HTTP Request utilizados* durante o teste, os referentes ao cadastro dos usuários Marcos, Lúcia e Roberto. Aqui colocamos três *listeners*: *Graph Results*, *Summary Report* e *View Results in Table*.

Analisaremos primeiramente o *Summary Report*. Esse elemento exibirá os dados de todos os *HTTP Request* usados, além de exibir um Total dos *HTTP Request* (ver **Figura 10**).

Em nosso teste, por exemplo, o cadastro do Usuário Marcos possui tempo mínimo de 158 milissegundos, enquanto o cadastro do Usuário Lucia



é de 24 milissegundos e o do Usuário Roberto de 48 milissegundos. Já o tempo máximo do cadastro do Usuário Marcos é 18431 milissegundos, enquanto o do cadastro do Usuário Lúcia é de 16598 milissegundos e o do Usuário Roberto 16771 milissegundos. Com isso é possível comparar diferentes comportamentos de diferentes requisições, todos efetuando a mesma ação. Numa aplicação real, pode-se avaliar o comportamento de diferentes páginas de uma aplicação Web, não necessariamente à mesma página como apresentado neste estudo de caso.

Iremos agora demonstrar o *Graph Result* (ver **Figura 11**). Ele mostra as linhas dos três grupos de requisições em diferentes posições. Por isso é bom que ele seja colocado também como filho de um *HTTP Request*, já que aqui são mostrados dados correspondentes a todos os Http Request usados.

Este tipo de gráfico apresenta uma forma bastante útil de análise dos dados. O eixo vertical, por apresentar o tempo gasto pelas requisições, pode ser utilizado como balizador do desempenho da aplicação, possibilitando definir o que é considerado aceitável para o tempo de resposta, confrontado com o comportamento apresentado pelas linhas do gráfico.

Iremos agora analisar outro tipo de *Listener*, chamado de *View Results in Table* (ver **Figura 12**).

Com o *View Results in Table*, é possível observar o estado de cada requisição feita. Ele exibirá as *threads* com seus respectivos nomes, o tempo da requisição e o estado da requisição. Observa-se os estados das requisições das *threads*, indicando sucesso ou erro, sendo apresentadas pelo símbolo exibido na coluna status. Caso algum erro ocorra durante o teste, esse também seria exibido no *Summary Report* por meio de porcentagem, ou seja, não seria possível saber qual *thread* falhou. Com o *View Results in Table* pode-se saber qual *thread* conseguiu fazer a requisição ou não, além de exibir a média e o desvio padrão total do teste.

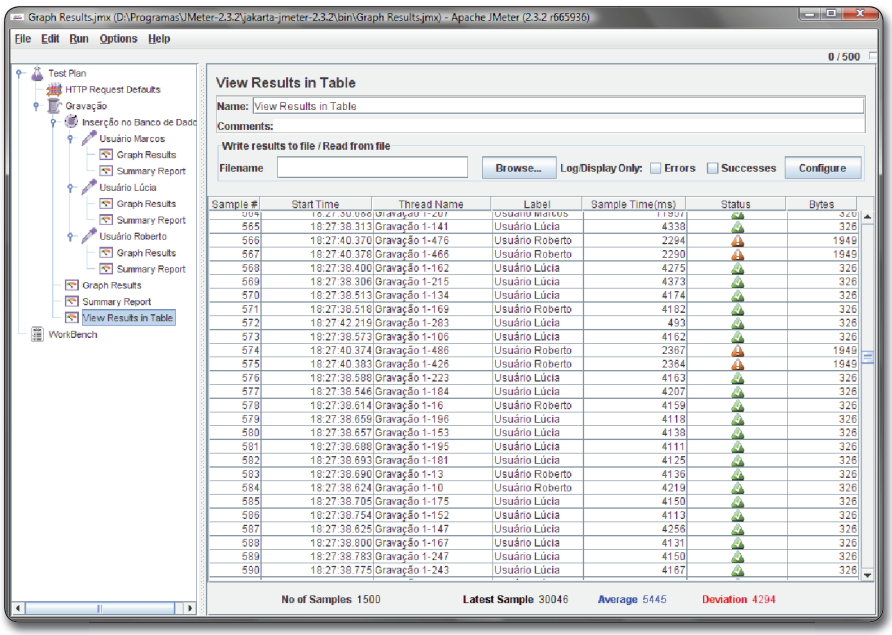


Figura 12. View Results in Table.

Conclusão

Vimos neste artigo que a ferramenta JMeter auxilia o desenvolvedor a testar se sua aplicação possui o desempenho esperado, possibilitando saber se o sistema irá suportar o número de requisições que os usuários poderão fazer. Para isso, é necessário elaborar os testes de maneira que retratem a realidade do uso da aplicação, sendo um importante mecanismo de prevenção de falha por desempenho ruim ou insatisfação do usuário, possibilitando que a aplicação seja melhorada. ●

Introdução à Gestão de Conhecimento - Parte 2

De que se trata o artigo:
Nos últimos anos, a gestão de conhecimento surgiu como um dos principais focos de preocupação em grandes organizações. Mais do que a tecnologia, o conhecimento é chave para companhias que pretendem agregar valor a seus produtos e serviços. Neste contexto, este artigo as setes camadas que normalmente compõem sistemas de auxílio à gestão do conhecimento.

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

Em que situação o tema é útil:
Gestão de conhecimento é um assunto amplamente estudado atualmente e utilizado no apoio à disseminação do conhecimento nas organizações.

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

Vimos no artigo sobre gestão de conhecimento da edição 6 da Engenharia de Software Magazine que nos últimos anos, a gestão de conhecimento surgiu como um dos principais focos de preocupação em grandes organizações. Isto por que, mais do que a tecnologia, o conhecimento é chave para companhias que pretendem agregar valor a seus produtos e serviços. Neste contexto, o artigo apresentou algumas definições introdutórias à área de gestão de conhecimento. Nesta edição, daremos continuidade ao assunto e o foco será na discussão sobre as sete camadas de um sistema de gestão de conhecimento.

Conceitualmente um Sistema de Auxílio à Gestão de Conhecimento pode ser dividido em sete camadas: interface, acesso e autenticação, inteligência colaborativa e filtragem, camada de aplicação, transporte, integração e por fim, os repositórios de dados. Esta estrutura em camadas está mostrada na **Figura 1**.

Cada camada possui seu próprio aparato tecnológico para realizar suas funções e alguns destes meios já estão bastante

disseminados entre empresas e instituições em geral. O que se necessita para o desenvolvimento de um bom sistema de gestão de conhecimento é uma efetiva integração destas tecnologias e a adição de alguns outros componentes.

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

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

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

As sete camadas de um sistema de gestão de conhecimento

Como mencionamos anteriormente, sistemas de apoio à gestão do conhecimento são compostos em geral por sete camadas.

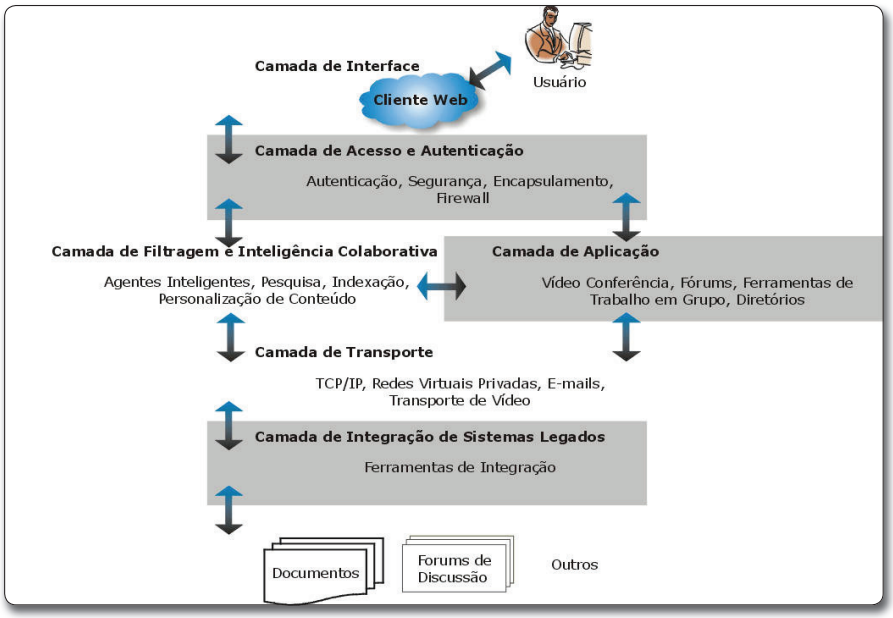
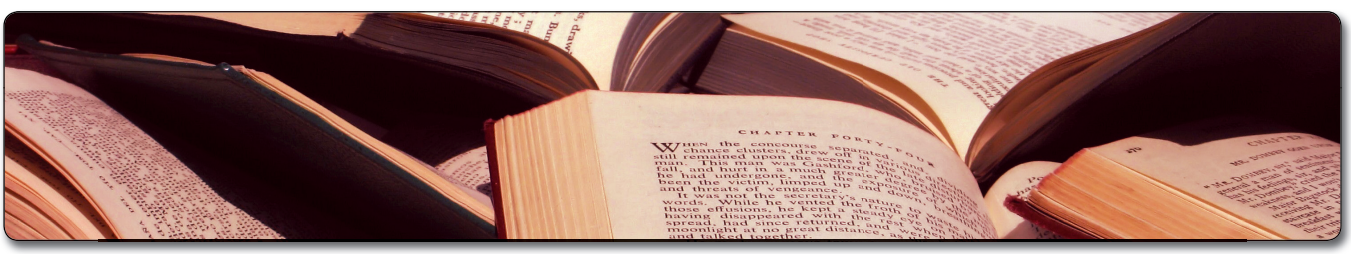


Figura 1. Camadas de um Sistema de Gestão de Conhecimento. Adaptado do modelo proposto por Tiwana (2000).

Interface

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

- Protocolos eficientes: protocolos de rede que permitam a utilização de recursos necessários para permitir colaboração, segurança e compartilhamento rápido de recursos;
- Portabilidade: é sabido que parte das empresas têm sua estrutura tecnológica baseada em diversas plataformas e diferentes sistemas operacionais. Desta forma, o sistema de gestão de conhecimento deve operar entre essas plataformas. Ganha então força o uso da Internet e o protocolo HTTP. Este, através dos browsers é suportado pelas plataformas mais comuns em uso nas empresas modernas;
- Escalabilidade: por se tratar de um sistema cuja probabilidade de crescimento no número de usuários é real, a plataforma na qual o mesmo seja implantado deve permitir o aumento de usuários sem degradar o desempenho;
- Segurança: este é um requisito indispensável. Como o "conhecimento" da empresa estará sendo disponibilizado para vários atores, existe a possibilidade de invasões indesejadas que terminem acessando informações estratégicas e/ou alterando documentos privados;
- Integração com sistemas existentes;
- Flexibilidade: os usuários devem ter a possibilidade de filtrar as informações que lhes convêm. Assim, é desejável um bom grau de facilidade de configuração e flexibilidade no que diz respeito ao que o usuário tem acesso e ao que ele realmente quer acessar.



Aplicação	Protocolo
Correio eletrônico	TCP
Web	TCP
Transferência de arquivos	TCP
Fluxo Multimídia (ex.: filme)	UDP
Telefonia pela Internet	UDP

Tabela 1. Aplicações e Protocolos

Baseados nestes princípios, percebemos que a Web é uma das plataformas mais adequadas às necessidades de um sistema de gestão de conhecimento. Entretanto, apesar das tecnologias associadas à Web preencherem boa parte dos requisitos listados acima, a camada de interface deve ainda prover funcionalidade para que as pessoas usem de forma efetiva a infra-estrutura criada com a tecnologia da informação. Necessita-se de bons recursos funcionais para criar, explicar, utilizar, e compartilhar conhecimento. Este é o principal conjunto de requisitos imposto sobre a camada de interface. Neste ponto, foca-se a principal dificuldade de projeto de um sistema de gestão de conhecimento.

Acesso e autenticação

Esta camada tem como principal função autenticar usuários válidos, restringir e prover segurança para o acesso às outras camadas. Como as redes de comunicação para o compartilhamento do conhecimento não têm sido limitadas apenas às intranets, a importância dada à segurança cresce. A Internet vem sendo cada vez mais utilizada para prover acesso a usuários que por algum motivo está em um lugar remoto. Neste ponto, podemos destacar a importância das Redes Virtuais Privadas (em inglês, Virtual Private Network), as quais possibilitam transmissão de dados de forma mais segura sob Internet.

Inteligência colaborativa e filtragem

A finalidade básica desta camada é prover a estrutura funcional para que se consiga fazer pesquisas, resumos, interpretações e a análise de grandes volumes de dados habilitando os usuários do sistema de gestão de conhecimento a contextualizá-los de forma efetiva e eficiente. Para este fim, existe um gama de possibilidades de combinação de tecnologias: ferramentas de inteligência artificial, redes neurais, agentes inteligentes, pesquisa por conteúdo e pesquisa por atributo, dentre outros.

O uso das tecnologias de pesquisa e interpretação depende de como está estruturado o sistema como um todo. Assim, é preciso entender como tais tecnologias funcionam para se optar pela ferramenta mais correta. As ferramentas de pesquisa trabalham basicamente através da busca e comparações de certos atributos de um determinado tipo de objeto. Por exemplo, arquivos criados em editores de texto podem possuir atributos como autor, data de criação, última modificação e assunto, dentre outros. Este leque de opções que envolvem a pesquisa, torna-a extremamente importante em um sistema de gestão de conhecimento.

Por ser um sistema que sofre constantes interações por parte dos usuários e por se tratar de um sistema com a infra-estrutura lógica (protocolos de comunicação) e física da Internet, é importante que sua estrutura de funcionamento interno não tenha como base estruturas estáticas - já bastante difundidas com o conceito de links - mas sim, dinâmicas, que automaticamente se adaptem a modificações na localização das informações. Neste caso, os ponteiros criados para outros documentos não se perdem, tornado a navegação pela informação menos incomoda e menos frustrante.

Aplicação

Esta camada engloba as ferramentas de integração usuário/máquina que provêm boa parte da funcionalidade de um sistema de gestão de conhecimento. Softwares e hardwares para vídeo conferência, fóruns de discussão e ferramentas de suporte a decisão são algumas aplicações que podem ser disponibilizadas aos usuários de um sistema de gestão de conhecimento.

Transporte

Tendo decidido a plataforma a ser utilizada, é preciso conhecer a maneira como os dados serão transportados pelas redes de comunicação e que serão os servidores e clientes utilizados na comunicação. Para o papel de cliente deve existir o software cliente que no caso da web, deve prover acesso à Intranet/Internet. Já para o papel de servidor, deve existir um software que funcione sobre o(s) repositório(s) de informações, disponibilizando acesso ao conteúdo do sistema.

A forma como os dados são transportados depende de quem solicita o envio dos mesmos e das necessidades que cada tipo de dado tem na sua transferência. Estas necessidades se resumem

basicamente ao fato de o serviço ser confiável ou não. Existem dois protocolos bastante utilizados: o TCP (do inglês, transmission control protocol) e o UDP (do inglês, user datagram protocol). O TCP provê transmissão de dados orientada à conexão, comunicação confiável para aplicações que tipicamente transferem grandes quantidades de dados e que requerem um reconhecimento da entrega dos dados. O TCP garante a entrega dos dados assegurando que os mesmos chegarão ordenados à aplicação destino e ainda possui mecanismos para identificar se o pacote transmitido sofreu alguma alteração.

Por outro lado, o UDP provê comunicação não orientada à conexão e não garante que os pacotes serão entregues. O seu ponto positivo é a velocidade com que os dados são entregues uma vez que o mesmo não possui algoritmos de tratamento de erros terminam tomando tempo durante o processo de transmissão. Desta forma, os pacotes estão sempre sendo enviados mesmo que esteja havendo perda dos mesmos. Este protocolo é utilizado por serviços cuja confiabilidade na entrega dos dados não é primordial. A tabela a seguir exemplifica algumas aplicações e o protocolo utilizado na comunicação das mesmas.

Camada de integração de sistemas legados

Esta camada é necessária quando se quer integrar plataformas diferentes de um ambiente heterogêneo em uma determinada empresa. Como um dos pontos chaves no processo de implantação de um sistema de gestão de conhecimento é a incorporação de estruturas já existentes, esta camada pode assumir grande importância. A integração entre sistemas que normalmente não se comunicam é um ponto chave em empresas que, por terem já um bom tempo de mercado, fazem uso de tecnologia de informação mais tradicional possuindo, por exemplo, sistemas baseados em mainframes. É comum ver estas companhias reestruturando sua infra-estrutura e "empacotando" seus sistemas legados com camadas de software que permitem a adoção de padrões mais modernos de comunicação e acesso a dados.

Armazenamento

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

Alguns fatores importantes para o sucesso de um sistema de auxílio à gestão do conhecimento

A forma de mensurar o sucesso de um sistema de auxílio à gestão do conhecimento é um assunto bastante discutido e ainda inacabado. Entretanto, assim como qualquer outro processo de reestruturação organizacional, existe alguns pontos sobre os quais podemos fazer uma análise:

- Crescimento de investimentos por par te da gerência no desenvolvimento do projeto com o passar do tempo;
- Crescimento no volume de conteúdo disponibilizado e na utilização deste;
- Não haver resistência na organização com os conceitos de conhecimento e gerência do conhecimento;
- Alguma percepção que de alguma forma esta havendo retorno financeiro.

A partir do momento que os pontos citados acima são realidade em um projeto de gestão do conhecimento, podemos dizer que este está tendo sucesso. Segundo Laurence Prusak, o sucesso ainda pode ser dividido em dois níveis: (1) quando o sistema provoca mudanças em toda a empresa e, (2) quando o sistema afeta apenas alguns setores específicos da organização.

Estas métricas mesmo que um pouco abstratas, são de grande valor no momento de justificar e analisar os resultados do sistema implantado. Porém, mais importante que verificar se o sistema obteve sucesso ou não é a tarefa de tentar desenvolvê-lo de modo que atinja seus objetivos e para isso, existe uma série de variáveis que devem ser levadas em conta.

Cultura orientada ao conhecimento

Este é, seguramente, um dos mais importantes fatores para o sucesso do projeto. Organizações em que essa cultura está bem fundamentada estão em vantagem pois é muito difícil iniciar um projeto tendo antes que modificar a forma de pensar das pessoas. Os dois extremos verificados quanto à presença desta variável são: (1) Orientação ao conhecimento valorizado onde os funcionários têm liberdade para incrementar e compartilhar suas experiências e; (2) Inibição à cultura do conhecimento onde as pessoas sentem-se ressentidas pela empresa não estimular a troca de experiências.

Parece difícil acreditar, mas ainda existem empresas que não perceberam que o crescimento intelectual de seus funcionários é fator chave para o sucesso. Organizações onde as pessoas não compartilham seu conhecimento pensando que desta forma se tornarão especiais para a empresa por ser o único perito em determinada área certamente ainda não possuem a cultura do conhecimento difundida.

Infra-estrutura organizacional e técnica

Processos que possam provocar uma reestruturação da empresa necessitam de certos requisitos. Particularmente,

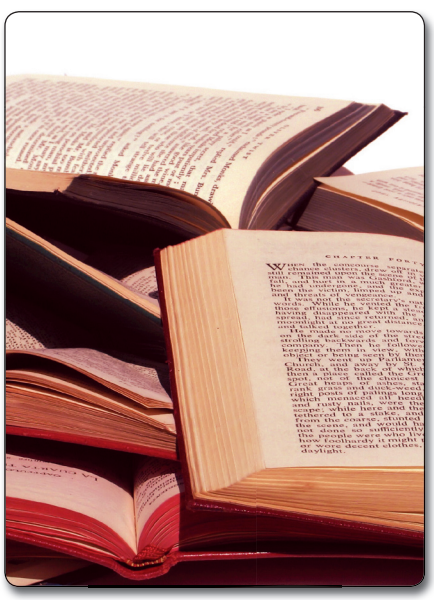
projetos envolvendo o conhecimento possuem duas necessidades básicas com relação à infra-estrutura da empresa. Primeiramente, uma boa base tecnológica que possa atender os requisitos de um sistema de gestão do conhecimento. Em segundo lugar, uma estrutura organizacional interna que torne possível a fixação da cultura do conhecimento.

Caso a empresa não possua a infra-estrutura citada acima, a organização deverá dispor de recursos financeiros uma vez que para alcançar esses dois objetivos, os investimentos em tecnologia e a criação de novas regras organizacionais é inevitável.

Apoio da alta gerência

Programas de gestão do conhecimento também se beneficiam do suporte dado pela alta gerência. Este se torna particularmente importante a partir do momento que estes tipos de sistema necessitam que a cultura organizacional esteja voltada para o conhecimento. O apoio pode ser dado de várias formas como:

- Criar uma infra-estrutura favorável para a implantação do sistema;
- Estimular as pessoas a compartilharem seu conhecimento e investirem em aprendizagem argumentando a importância dessas ações para o sucesso da companhia;
- Pesquisar e definir que tipo de conhecimento é mais importante para a empresa.



Relacionamento com valores econômicos e/ou industriais

Por serem sistemas que requerem investimentos geralmente altos, a necessidade de benefícios econômicos ou diferencial na área de atuação da organização são extremamente importantes. Porém, muitas vezes é difícil mensurar os benefícios advindos da implantação de um sistema de gerenciamento do conhecimento. Por isso, uma parte dos benefícios pode ser calculada indiretamente através de resultados na diminuição de despesas, nível de satisfação dos clientes e redução no tempo de produção.

Estímulo a ações não triviais

Um dos requisitos básicos para o sucesso de sistemas de gestão do conhecimento é o envolvimento dos integrantes do escopo do projeto. Este envolvimento significa criar, externalizar, internalizar e utilizar o conhecimento. Estas são tarefas difíceis de serem cumpridas e que muitas vezes somente são executadas mediante o uso de técnicas motivacionais. É preciso que haja alguma forma de premiar aqueles que mais contribuírem participando do processo de socialização do conhecimento. Para isto, algumas técnicas estudadas na administração podem ser utilizadas como:

- Desenvolver a autoconfiança;
- Relacionar recompensas com o desempenho;
- Utilizar uma variedade de recompensas;
- A alta gerência deve ser positiva e esperançosa;
- Fazer com que os envolvidos no sistema sintam-se parte integrante do processo como um todo.

A partir do momento que a atividade de criar, compartilhar e utilizar o conhecimento torna-se parte da cultura das pessoas, a organização conseguiu um grande avanço para fazer com que o sistema tenha sucesso.

Múltiplos meios para a transferência do conhecimento

Por se tratar de um bem intangível e altamente mutável, o conhecimento pode ser compartilhado de diversas formas. Em seu caráter explícito, o conhecimento pode ser socializado por exemplo, através de sistemas gerenciadores de informação como é o caso de ferramentas baseadas em armazenamento estruturado de dados. Estes meios de transferência do conhecimento já possuem um bom grau de desenvolvimento mas, quando o conhecimento é tácito, ainda há muita dificuldade em compartilhá-lo de forma adequada. Por isso, há a necessidade de múltiplos meios para a transferência do conhecimento. Estes meios vão desde conversas via telefone, sistemas de vídeo conferência até mesmo às reuniões presenciais.

Considerações Finais

Este artigo apresentou as sete camadas que normalmente são encontradas em sistemas de apoio à gestão de conhecimento. Muitos estudos e pesquisas têm sido realizadas no âmbito de cada uma destas camadas e certamente serão discutidos em artigos futuros da Engenharia de Software Magazine. ●

Dê seu feedback sobre esta edição!

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

www.devmedia.com.br/esmag/feedback

Referências

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

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

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

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

Apoiando a Implementação do Modelo de Maturidade MPS Nível G

Augusto César Brauns Munhão
augusto.munhao@promon.com.br
Cursando a Graduação em Ciência da Computação pelo Instituto Metodista Bennett do Rio de Janeiro. (Graduação em 2008). Graduado em Tecnólogo em Analise de Sistemas pelo Instituto Metodista Bennett do Rio de Janeiro. Executa Customizações a softwares de CAD-CAE e Plant Design com suporte a usuários e gerenciamento de Backups. Atua em Projetos nas Áreas de Mineração, Geração de Energia, Petroquímica, Siderurgia e Indústria Alimentícia.

Adriano Cláudio Costa Campos
adriano.campos10@gmail.com
Cursando o 8 período da Graduação em Ciência da Computação pelo Instituto Metodista Bennett do Rio de Janeiro. Graduado em Tecnólogo em Analise de Sistemas pelo Instituto Metodista Bennett do Rio de Janeiro. Presta suporte na área de plataformas de gerência de redes de telecomunicações (IP, ATM, TDM, Frame-Relay, X-25, XDSL) baseadas em sistemas operacionais Unix e protocolos de gerência SNMP e TL1. Atua em projetos nas áreas de Segurança da Informação, Gerenciamento de Redes de Telecomunicações e Sistemas de Apoio à Gestão.

Marcos Kalinowski
mk@kalisoftware.com
Doutorando e mestre em Engenharia de Software da COPPE/UFRJ. Bacharel em Ciência da Computação pela UFRJ. Diretor executivo da Kali Software (www.kalisoftware.com), empresa de consultoria e treinamento em Engenharia de Software. Coordenador e professor dos cursos Ciência da Computação e Análise e Desenvolvimento de Sistemas do Centro Universitário Metodista Bennett. Professor da pós-graduação e-IS Expert da UFRJ. Consultor de implementação, instrutor, avaliador e membro da equipe técnica do MPS.BR.

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

De que se trata o artigo:
O objetivo deste artigo é apresentar os resultados obtidos com uma pesquisa de campo (survey) para apontar as práticas mais adotadas pelos implementadores do modelo MPS ao evidenciar as exigências do nível G deste modelo.

Para que serve:
As informações geradas por este trabalho, juntamente com os guias disponibilizados pela SOFTEX, representam para as empresas, uma oportunidade a mais de conhecer boas práticas relacionadas à coleta de evidências diretas e indiretas para evidenciar as exigências do nível G, quando em busca de uma certificação. Por conseqüência, as informações resultantes deste trabalho, podem auxiliar na redução de custos de implementação e tempo gasto na preparação para o processo de certificação.

Em que situação o tema é útil:
Empresas que tenham interesse em adotar práticas de melhoria de processo.

No início da produção de software (1960) não havia se pensado em estruturas formais de desenvolvimento de software. Existia no mercado a necessidade de softwares cada vez maiores e com essas necessidades aumentadas, começaram a se tornar mais constantes os problemas e prejuízos com os atrasos e cancelamentos nos projetos de software. A solução foi a criação da Engenharia de Software que tinha como principais objetivos definir processos, métodos e ferramentas para a produção de software com a qualidade esperada pela indústria. Com o aumento da qualidade foi introduzido um aperfeiçoamento contínuo de métodos, técnicas e artefatos (PRESSMAN, 2006).

Nos dias atuais, com a necessidade de se alcançar maior competitividade e qualidade na construção de software as empresas nacionais sentem-se na obrigação de modificar suas estruturas organizacionais e processos produtivos para que se adaptem ao tamanho e características dos projetos sem perder os padrões de qualidade exigidos tanto nacional, como internacionalmente. Com todas essas necessidades foi criado o MPS.BR , um modelo de maturidade nacional com padrões de qualidade internacionais no qual as empresas se certificam em diferentes níveis de acordo com seu grau de maturidade.

Hoje no Brasil, mais de 100 empresas possuem certificação MPS e estas certificações, mesmo para os níveis menos complexos, são extremamente difíceis.

Dentro deste contexto, percebeu-se a oportunidade de realizar um trabalho que identificasse as práticas mais utilizadas por implementadores MPS para evidenciar as exigências do modelo de maturidade brasileiro, visando auxiliar, orientar e minimizar as dificuldades enfrentadas na sua implementação. Desta forma torna-se possível promover um melhor entendimento por parte das empresas e dos implementadores sobre como evidenciar os resultados exigidos para a certificação do nível G do modelo MPS.

Desta forma, o objetivo deste artigo é apresentar os resultados obtidos com uma pesquisa de campo (survey) para apontar as práticas mais adotadas pelos implementadores do modelo MPS ao evidenciar as exigências do nível G deste modelo.

Além disso, o trabalho envolve uma pesquisa envolvendo o Guia Geral e o Guia de Implementação do MPS, com o intuito de abordar as características e diretrizes do modelo de Maturidade MPS no tratamento e na evidência dos resultados exigidos pelo modelo, quando da obtenção da certificação do nível G.

Como resultado, pretende-se que as informações geradas por este trabalho, juntamente com os guias disponibilizados pela SOFTEX, representem para as empresas, uma oportunidade a mais de conhecer boas práticas relacionadas à coleta de evidências diretas e indiretas para evidenciar as exigências do nível G, quando em busca de uma certificação. Por conseqüência, as informações resultantes deste trabalho, podem auxiliar na redução de custos de implementação e tempo gasto na preparação para o processo de certificação.

Métodos Orientados a Plano

Antes de falar do MPS é importante explicar qual a metodologia utilizada por este modelo. O MPS é um modelo orientado a plano. Mas o que significa dizer que um processo é orientado a plano? Métodos orientados a planos são muitas vezes referenciados como "a forma tradicional" de se desenvolver software. Eles se caracterizam por possuir processos bem definidos que são constantemente aperfeiçoados nas organizações.

A origem dos métodos orientados a planos deu-se devido à necessidade de algumas organizações adotarem processos que garantissem a qualidade de seus produtos. Para atingir este objetivo era preciso delinear muito bem o escopo do trabalho e as diretivas a serem seguidas. Vários motivos contribuíram para a adoção desta nova forma de tratamento do produto de software. Dentre eles, podemos citar:

- Risco em caso de falhas;
- Necessidade de controle de alterações;
- Controle de prazos e custos, entre outros.

Entre os ramos da indústria que se anteciparam em desenvolver estes processos temos o Aeroespacial, Mísseis Balísticos, Satélites (BOEHM e TURNER, 2004).

A adoção de processos voltados à construção de software apresentou algumas deficiências em suas primeiras experiências de implementação por que, sem métricas, os processos ficavam sujeitos a falhas nos orçamentos, gerenciamento, etc. Porém, ao longo do tempo, estes processos sofreram significativo amadurecimento e atualmente são recomendados à atividade de desenvolvimento de software.

Principais Características dos Métodos Orientados a Planos

Como principais características dos métodos orientados a planos, podemos citar a documentação bem estruturada e concisa, a rastreabilidade entre requisitos, projetos e códigos. Outras características importantes são a definição e gerenciamento de processos que são aperfeiçoados e normalizados. Todos os processos devem possuir atividades detalhadas, fluxo de trabalho, responsabilidades definidas. Além disso, é comum o emprego de Gerentes de Projetos que monitoram, verificam e disseminam a informação.

De acordo com (BOEHM e TURNER, 2004) os principais conceitos dos métodos orientados a planos são:

- Aperfeiçoamento de processos;
- Processo de capacitação;
- Maturidade organizacional;
- Gerenciamento de risco;
- Verificação;
- Validação;
- Arquitetura dos sistemas de software.

Exemplos de Métodos Orientados a Planos

Ao longo dos anos, diversos métodos orientados a planos foram desenvolvidos e modelos de maturidade para tais métodos foram criados. De acordo com (BOEHM e TURNER, 2004) entre estes métodos destacam-se:

- CMMI – SEI (Modelo de Maturidade) (SEI, 2007);
- Military Standards – Departamento de Defesa dos USA;
- General Process Standards – ISO , EIA , IEEE;
- Software Favorities – Hitachi, General Electric;

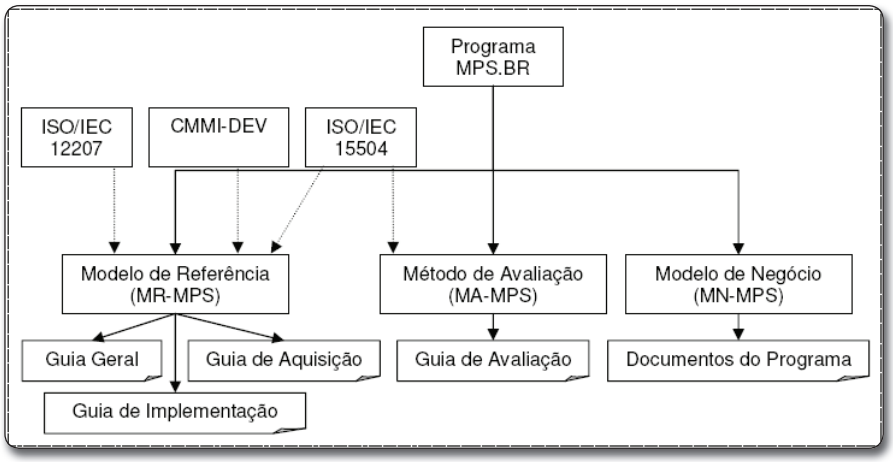


Figura 1. MPS.BR (SOFTEX,2007a)

- Cleanroom – IBM;
- Capability Maturity Model Software (SW – CMM) – SEI;
- Personal Software Process (PSP) – SEI;
- Team Software Process (TSP) – SEI.

A partir de agora, conheceremos o MPS BR. Os conceitos apresentados visam fornecer um detalhamento sobre este modelo como forma de prover uma melhor compreensão da pesquisa desenvolvida pelos autores deste artigo.

Introdução ao MPS

As mudanças que estão ocorrendo nos ambientes de negócios têm motivado as empresas a modificar suas estruturas organizacionais e processos produtivos, saindo da visão tradicional baseada em áreas funcionais em direção a redes de processos centrados no cliente. A competitividade depende, cada vez mais, do estabelecimento de conexões entre estes processos, criando elos essenciais nas cadeias produtivas. Alcançar competitividade pela qualidade, para as empresas de software, implica tanto na melhoria da qualidade dos produtos de software e serviços correlatos, como dos processos de produção e distribuição de software.

Desta forma, assim como para outros setores, qualidade é fator crítico de sucesso para a indústria de software. Para que o Brasil possua um setor de software competitivo, nacional e internacionalmente, é essencial que os empreendedores do setor coloquem a eficiência e a eficácia de seus processos em foco nas suas empresas, visando à oferta de produtos de software e dos serviços oferecidos por estes, de acordo com padrões internacionais de qualidade.

O foco principal, embora não exclusivo, do MPS, está no grupo de empresas formado, em geral, pela grande massa de micro, pequenas e médias empresas de software brasileiras, com poucos recursos e que necessitam melhorar radicalmente seus processos de software em 1 ou 2 anos (SOFTEX, 2007a). O intuito é que o modelo seja adequado ao perfil de empresas com diferentes tamanhos e características, sejam elas públicas ou privadas, de portes variados e que seja compatível com os padrões de qualidade aceitos internacionalmente.

Além disso, tem como pressuposto o aproveitamento de toda a competência existente nos padrões e modelos de melhoria de processo já disponíveis. Dessa forma, ele tem como base os requisitos definidos nos modelos de melhoria de processo.

O MPS visa atender a necessidade de implantar os princípios de Engenharia de Software de forma adequada ao contexto das empresas brasileiras, sendo compatível com as principais abordagens internacionais para definição, avaliação e melhoria de processos de software, melhoria da qualidade e construção de produtos de software e serviços correlatos, baseando-se nos conceitos de maturidade e capacidade destes processos.

Dentro desse contexto, o MPS possui três componentes (SOFTEX, 2007a):

- Modelo de Referência (MR-MPS);
- Método de Avaliação (MA-MPS);
- Modelo de Negócio (MN-MPS).

Descrição Geral do Modelo MPS

O MPS tem como objetivo definir um modelo de melhoria e avaliação de processo de software, preferencialmente para as micro, pequenas e médias empresas, de forma a atender às necessidades de negócio e, além disso, ser reconhecido nacional e internacionalmente como um modelo aplicável à indústria de software. Este é o motivo pelo qual ele está aderente a modelos e normas internacionais. O MPS também define regras para sua implementação e avaliação, dando sustentação e garantia de que está sendo empregado de forma coerente com as suas definições (SOFTEX, 2007a).

Conforme descrito em (SOFTEX, 2007a), a base técnica utilizada para a construção do MPS é composta pelas normas (ABNT, 1998) – Processo de Ciclo de Vida de Software e suas emendas 1 e 2 (ISO/IEC 15504-1,2004) e (ISO/IEC 15504-2,2003) – Avaliação de Processo (também conhecida por SPICE: Software Process Improvement and Capability Etermination e seu Modelo de Avaliação de Processo de Software (ISO/IEC 15504-5,2006). O modelo está totalmente aderente a essas normas. O MPS também cobre o conteúdo do CMMI-SE/SWSM, através da inclusão de processos e seus resultados de acordo com as Normas (ABNT, 1998), conforme **Figura 1**.

Cada componente do modelo é descrito através de guias e dos documentos do projeto. Um destes documentos é o Modelo de Referência de Melhoria de Processo de Software (MR-MPS), que contém os requisitos que as organizações deverão atender para estar em conformidade com o modelo de maturidade em referência. Ele contém as definições dos níveis de maturidade e aborda também a capacidade de realização dos processos dentro de cada um dos níveis do modelo. Ele foi baseado nas normas (ABNT,1998) e suas emendas 1 e 2, ISO/IEC 15504 e adequado ao CMMI-SE/SWSM (SOFTEX,2007a).

Nível	Processos	Atributos de Processo
A	Análise de Causas de Problemas e Resolução – ACP	AP 1.1, AP 2.1, AP 2.2, AP 3.1, AP3.2, AP 4.1, AP 4.2 , AP 5.1 e AP 5.2
B	Gerência de Projetos – GPR (evolução)	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP3.2, AP 4.1 e AP 4.2
C	Gerência de Riscos – GRI	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP3.2
	Desenvolvimento para Reutilização – DRU	
	Análise de Decisão e Resolução – ADR	
	Gerência de Reutilização – GRU (evolução)	
D	Verificação – VER	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP3.2
	Validação – VAL	
	Projeto e Construção do Produto – PCP	
	Integração do Produto – ITP	
	Desenvolvimento de Requisitos – DRE	
E	Gerência de Projetos – GPR (evolução)	AP 1.1, AP 2.1, AP 2.2, AP 3.1 e AP3.2
	Gerência de Reutilização – GRU	
	Gerência de Recursos Humanos – GRH	
	Definição do Processo Organizacional – DFP	
	Avaliação e Melhoria do Processo Organizacional – AMP	
F	Medição – MED	AP 1.1, AP 2.1 e AP 2.2
	Garantia da Qualidade – GQA	
	Gerência de Configuração – GCO	
	Aquisição – AQU	
G	Gerência G de Requisitos – GRE	AP 1.1 e AP 2.1
	Gerência de Projetos – GPR	

Tabela 1. Níveis de Maturidade (SOFTEX,2007a)

Descrição do MR-MPS

O Modelo de Referência MR-MPS define níveis de maturidade que são uma combinação entre processos e capacidade de implementação de cada processo.

A capacidade de processo é sua habilidade para alcançar os objetivos de negócio, atuais e futuros. Ela está relacionada com o atendimento dos atributos de cada processo dentro de cada nível de maturidade.

Níveis de maturidade - Descrição Geral

Os níveis de maturidade estabelecem patamares de evolução de processos, caracterizando estágios de melhoria de implementação de processos na organização. O nível de maturidade em que se encontra uma organização permite prever seu desempenho futuro em uma ou mais disciplinas. O MR-MPS define sete níveis de maturidade:

- A (Em Otimização)
- B (Gerenciado Quantitativamente)
- C (Definido)
- D (Largamente Definido)
- E (Parcialmente Definido)
- F (Gerenciado)
- G (Parcialmente Gerenciado).

A escala de maturidade se inicia no nível G e progride até o nível A. Para cada um destes sete níveis de maturidade foi atribuído um perfil de processos e de capacidade destes processos que indicam onde a organização tem que colocar esforço para melhoria de forma a atender os objetivos de negócio, conforme **Tabela 1**.

O progresso e o nível de maturidade são obtidos quando são atendidos todos os resultados e propósito do processo, além dos atributos de processo relacionados àquele nível e aos anteriores.

A divisão em estágios, embora baseada nos níveis de maturidade do CMMISE/SWSM tem uma graduação diferente, com o objetivo de possibilitar uma implementação e avaliação mais gradual e adequada às pequenas e médias empresas. A possibilidade de se realizar avaliações considerando mais níveis permite uma visibilidade dos resultados de melhoria de processos com prazos mais curtos.

Capacidade do Processo - Descrição Geral

A capacidade / adequação do processo ao MPS é representada por um conjunto de atributos descritos em termos de resultados esperados. Ela representa o nível de refinamento e institucionalização com que o processo é executado na organização. No MPS, à medida que a organização evolui nos níveis de maturidade, um maior nível de capacidade para desempenhar o processo deve ser atingido pela organização.

O atendimento aos atributos do processo (AP) será atingido através da geração dos resultados esperados dos atributos do processo (RAP), que são requeridos para todos os processos no nível correspondente ao nível de maturidade, embora eles não sejam detalhados dentro de cada processo. Os níveis são acumulativos, ou seja, se a organização está no nível F, esta possui o nível de capacidade do nível F que inclui os atributos de processo dos níveis G e F para todos os processos relacionados no nível de maturidade F. Isto significa que, ao passar do nível G para o nível F, os processos do nível G passam a ser executados no nível de capacidade correspondente ao nível F.

A capacidade do processo (SOFTEX, 2007a) no MPS possui nove (9) atributos de processos (AP). São eles:

- AP 1.1 – Processo executado (Necessário para certificação Nível G);
- AP 2.1 – Processo gerenciado (Necessário para certificação Nível G);
- AP 2.2 – Produtos de trabalho do processo são gerenciados;
- AP 3.1 – Processo é definido;
- AP 3.2 – Processo está implementado;
- AP 4.1 O processo é medido;
- AP 4.2 O processo é controlado;
- AP 5.1 O processo é objeto de inovações;
- AP 5.2 O processo é otimizado continuamente.

A seguir estão descritos os atributos de processo (AP) necessários para a certificação MPS do nível G, uma vez que esse é o foco da pesquisa desenvolvida para este trabalho. Cada AP está detalhado em termos de resultados esperados do atributo de processo (RAP) para alcance completo do atributo de processo.

AP1.1 Processo é executado

Este atributo é uma medida da extensão na qual o processo atinge o seu propósito. Resultado esperado:

- RAP 1 - O processo atinge seus resultados definidos.

AP 2.1 Processo é gerenciado

Este atributo é uma medida da extensão na qual a execução do processo é gerenciada. Resultados esperados:

- RAP 2 - Existe uma política organizacional estabelecida e mantida para o processo;
- RAP 3 - A execução do processo é planejada;
- RAP 4 (Para o Nível G) - A execução do processo é monitorada e ajustes são realizados para atender aos planos;
- RAP 4 (A partir do Nível F) - Medidas são planejadas e coletadas para monitoração da execução do processo;
- RAP 5 - Os recursos necessários para a execução do processo são identificados e disponibilizados;
- RAP 6 - As pessoas que executam o processo são competentes em termos de formação, treinamento e experiência;
- RAP 7 - A comunicação entre as partes interessadas no processo é gerenciada de forma a garantir o seu envolvimento no projeto;
- RAP 8 - Métodos adequados para monitorar a eficácia e adequação do processo são determinados.

Processos

Os processos são agrupados de acordo com a sua natureza, o seu objetivo principal no ciclo de vida de software. Esse agrupamento resultou em 3 diferentes classes de processos, que são:

- Fundamental: Atendem o início e a execução do desenvolvimento, operação ou manutenção dos produtos de software e serviços correlatos durante o ciclo de vida de software;
- De Apoio: auxiliam um outro processo e contribuem para o sucesso e qualidade do projeto de software;
- Organizacional: Uma organização pode empregar estes processos em nível corporativo para estabelecer, implementar e melhorar um processo do ciclo de vida.

Descrição de Processos do Nível G do MPS.BR

Segundo (SOFTEX, 2007a), temos descritos para o nível G do MPS.BR os seguintes processos:

- O processo de Gerência de Projetos tem como propósito, identificar, estabelecer, coordenar e monitorar as atividades, tarefas e recursos que um projeto necessita para produzir um produto e/ou serviço, no contexto dos requisitos e restrições do projeto.
- O processo de Gerência de Requisitos tem como propósito gerenciar os requisitos dos produtos e componentes do produto do projeto e identificar inconsistências entre esses requisitos e os planos e produtos de trabalho do projeto.

Implementação do Nível G do Modelo MPS

A finalidade do Guia de Implementação é fornecer diretrizes para a implementação dos níveis de maturidade descritos no Modelo de Referência MR-MPS em empresas que desejam aumentar seu grau de competitividade e qualidade nos seus processos de construção de software. Este guia detalha os processos requeridos para cada nível de maturidade e os seus resultados esperados com a implementação dos processos. Este documento é destinado, mas não está limitado, a organizações interessadas em utilizar o MR-MPS para melhoria de seus processos de software e instituições implementadoras.

O Guia de implementação está subdividido em sete partes.

- Parte 1: nível G (que é o foco do trabalho);
- Parte 2: nível F;
- Parte 3: nível E;
- Parte 4: nível D;
- Parte 5: nível C;
- Parte 6: nível B; e
- Parte 7: nível A;

Como o foco do presente trabalho será tentar mostrar os artefatos mais utilizados segundo a visão dos implementadores para que seja conseguida

a certificação nível G do modelo MPS, o trabalho irá se ater ao Guia de Implementação parte 1 – Nível G (SOFTEX, 2007b).

Implementação do Nível G

No Guia de Implementação parte 1 – Nível G (SOFTEX, 2007b) o nível G é o primeiro nível de maturidade do modelo MR-MPS. Deve-se tomar cuidado redobrado para que sejam formadas estruturas sólidas para a implantação de melhoria dos processos de software na organização. Ao final da implantação deste nível de maturidade a organização será capaz de gerenciar parcialmente seus processos de construção de software.

Deve haver uma mudança radical na cultura da empresa orientando-se a melhoria dos processos de construção e também a definição do conceito acerca do que é "projeto" para a organização, isto é, redefinir operações já em andamento estabelecendo objetivos, prazos e escopo para sua execução.

Para que seja conseguida a certificação o nível G do modelo MPS, as organizações devem fazer a Gerência de Projeto e Gerência de Requisitos. Que serão descritos a seguir.

Gerência de Projetos

Propósito

No Guia de Implementação parte 1 – Nível G (SOFTEX, 2007b) a Gerência de Projetos tem como propósito estabelecer e manter planos que definem as atividades, recursos e responsabilidades do projeto, prover informações sobre o andamento do projeto que permitam correções quando houver desvios significativos no desempenho do projeto. À medida que o nível de maturidade da empresa evolui, este propósito também evolui. Como exemplo, a partir do nível E alguns processos evoluem e outros surgem. No nível B a Gerência de projetos passa a ter enfoque quantitativo.

Fundamentação Teórica

O Guia de Implementação parte 1 – Nível G (SOFTEX, 2007b) busca fundamentação teórica no PMBOK (Project Management Body of Knowledge) que é um guia em gerência de projetos tendo como responsável por sua publicação o PMI (Project Management Institute). O PMI é um dos mais conceituados institutos na área de gerência de projetos. O PMBOK agrupa as boas práticas reconhecidas em gerenciamento de projeto.

Quando se fala de Gerência de Projetos, deve-se ter o conceito de projeto bem claro para que possamos entender melhor sua gerência. O PMBOK (PMBOK, 2004) apresenta uma das definições de projeto mais reconhecidas atualmente: "Projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo". Temporário por que o projeto deve possuir um inicio e um fim.

No termo "produto, serviço ou resultado exclusivo", a exclusividade e a elaboração progressiva são características importantes a serem observadas nas entregas do projeto. Outra característica importante de projeto é a elaboração progressiva que integra os conceitos de temporalidade e exclusividade. Elaboração progressiva significa desenvolver em etapas e por incrementos. Por exemplo, o escopo do projeto será identificado de maneira geral no início do projeto e se tornará mais claro e refinado à medida que a equipe do projeto desenvolve um entendimento mais completo dos objetivos e das entregas. Outro conceito são as operações que fazem parte da execução de um trabalho para que seja atingido um objetivo. Estas operações devem ser planejadas, executadas e controladas por pessoas e têm restrições de recursos.

Como descrito no Guia de Implementação parte 1 – Nível G (SOFTEX, 2007b) e no PMBOK (Project Management Body of Knowledge) (PMBOK, 2004), o gerenciamento de projeto é a aplicação de conhecimento, habilidades, ferramentas e técnicas às atividades do projeto, a fim de atender aos seus requisitos, estabelecer prioridades, gerenciar conflitos, identificar necessidades para que haja aumento de qualidade, redução de custos e prazos na produção de artefatos de acordo com requisitos especificados.

Conforme o IEEE (Institute of Electrical and Electronics Engineers), em seu Glossário Padrão de Terminologias da Engenharia de Software (IEEE Std 610.12, 1990), a gerência de projetos de software pode ser definida como a aplicação de planejamento, coordenação, medição, monitoramento, controle e divulgação de relatórios, com o intuito de garantir que o desenvolvimento e a manutenção de software sejam sistemáticos, disciplinados e qualificados.

Gerência de Requisitos

Propósito

No Guia de Implementação parte 1 – Nível G (SOFTEX, 2007b) na Gerência Requisitos de produtos, a identificação de inconsistências é o propósito principal do processo de Gerência de Requisitos e tem como objetivo o controle da evolução dos requisitos tanto funcionais como não-funcionais. Também fornece apoio ao planejamento definindo um conjunto de passos que serão seguidos pela organização. Com a Gerência de Requisitos, alterações de requisitos de projeto são

melhor administradas, diminuindo os impactos que estas possam causar aos custos, prazos e qualidade do projeto. Documentar as mudanças nos requisitos e suas justificativas mantém a rastreabilidade bidirecional entre os requisitos e produtos de trabalho e identificam inconsistências entre os requisitos, os planos do projeto e os produtos de trabalho do projeto.

Fundamentação Teórica

O Guia de Implementação parte 1 – Nível G (SOFTEX, 2007b) busca fundamentação teórica em Dorfmann e Thayer (1990) que diz que requisito de software representa a capacidade que deve ser encontrada ou possuída por um determinado produto ou componente de produto para satisfazer a um contrato, a um padrão, a uma especificação ou a outros documentos formalmente impostos. Os requisitos indicam a capacidade do software requerida pelo usuário para resolver um problema ou alcançar um objetivo. A gerência de requisitos envolve estabelecer e manter um acordo entre o cliente e a equipe de projeto sobre os requisitos estabelecidos e sobre qualquer mudança ocorrida neles. Para apoiar esse processo de mudança, é fundamental definir e manter a rastreabilidade dos requisitos. Rastreabilidade é o grau de relacionamento que pode ser estabelecido entre dois ou mais produtos.

No Guia de Implementação parte 1 – Nível G (SOFTEX, 2007b) a gerência de requisitos é implementada com a identificação, controle e rastreamento dos requisitos, bem como com o tratamento das mudanças nos requisitos mantendo-se a rastreabilidade bidirecional dos requisitos. A rastreabilidade pode ser estabelecida, desde um requisito alto nível até seus requisitos de mais baixo nível e destes até o seu requisito de alto nível. A rastreabilidade bidirecional auxilia a determinar se todos os requisitos de alto nível foram completamente tratados e se todos os requisitos de mais baixo nível podem ser rastreados.

Atributos de processo no nível G

De acordo com o Guia Geral do MR-MPS: "a capacidade do processo é representada por um conjunto de atributos de processo descrito em termos de resultados esperados." Porém, sua análise não está no escopo deste artigo.

Segundo o Guia de Implementação parte 1 – Nível G (SOFTEX, 2007b), existem dois atributos de processo que são:

- AP 1.1 - O processo é executado;
- AP 2.1 - O processo é gerenciado.

Como este não é o foco deste artigo, foi feita apensa uma menção a estes atributos.

Planejamento do Survey

Justificativa para o Survey

Um número cada vez maior de empresas de desenvolvimento de software brasileiras tem optado pela certificação MPS (modelo de maturidade nacional), isto por que este modelo aprimora de forma simples e efetiva os processos de software destas empresas. Conforme já citado anteriormente, o MPS tem como principal objetivo a certificação de pequena e médias empresas. Isto é comprovado pelos seguintes dados: atualmente em mais de 100 avaliações oficiais MPS, 91,4% foram para os níveis F e G.

Para que os custos de implantação e avaliação diminuíssem, o MPS foi dividido em níveis: Nível G – Parcialmente Gerenciado, F – Gerenciado, E – Parcialmente Definido, D – Largamente Definido, C – Definido, B – Gerenciado Quantitativamente e A – Em Otimização. Existem projeções oficiais que 180 empresas serão certificadas até dezembro de 2009. O MPS vem sendo apoiado diretamente pelo governo brasileiro (MCT e FINEP) e BID.

Dentro deste contexto, identifica-se a oportunidade de apoiar as empresas que optam por implementar as normativas que compõem o modelo MPS, como forma de padronizar seus processos, gerenciá-los e controlá-los, no intuito de trazer ganhos, sobretudo em qualidade e redução de custos.

Baseado nesta oportunidade identificada, pode-se dizer que um trabalho de survey, desenvolvido e fundamentado em práticas reais de implementação, poderia orientar novos implementadores e empresas interessadas em implementar o modelo, com as formas utilizadas pelos atuais implementadores para evidenciar as exigências do MPS, assim como dar ciência a estes atuais implementadores do modelo sobre quais são os artefatos mais utilizados por aqueles que contribuíram com a pesquisa. Este trabalho poderia conter ainda dados percentuais indicativos de quão utilizados são pelos implementadores do MPS.

Segundo (Wohlin, C., et al, 2000), "em um survey, um questionário pode ser construído para que se obtenha

informações necessárias à pesquisa. As informações coletadas são então arranjadas de forma que possam ser tratadas de maneira quantitativa ou qualitativa".

Neste sentido este artigo foi desenvolvido com base em um survey, com intuito de levantar dados para a geração de conhecimento, através de questionamentos direcionados sobre o assunto, para posterior análise com a aplicação de métricas que proporcionem uma mensuração de forma uniforme e consistente dos dados colhidos, resultando na geração de informações úteis à comunidade implementadora e empresas interessadas na certificação do primeiro nível do modelo (nível G).

Definição da Metodologia

A geração de informações acuradas a partir do trabalho de levantamento e da avaliação dos dados reunidos necessita de uma metodologia bem definida, com objetivos claros e que ainda forneça mecanismos que permitam medir de forma padronizada os dados fornecidos pelos implementadores, para que as informações geradas possam estar consistentes e bem estruturadas.

Para a elaboração do questionário de Levantamento e Avaliação foi adotada a Metodologia GQM - Goal-Question-Metric - (SOFTEX, 2008), devido à necessidade, em um primeiro momento, de definir um padrão que permitisse gerar questionamentos consistentes e mensuráveis, aos implementadores do nível G do modelo MPS. No sentido oposto do processo, era preciso interpretar de forma uniforme as respostas emitidas para cada questionamento relacionado aos diversos resultados esperados pelo modelo MPS, nível G. Pelo fato de ser baseado em métricas, o GQM atendeu bem as necessidades identificadas para a concepção do questionário, assim como para a avaliação das respostas.

GQM (Goal – Question – Metrics)

Definição dos Objetivos

Objetivo Global

Planejar um estudo experimental (survey) para indicar as formas adotadas por implementadores, para evidenciar a implementação dos requisitos básicos para a certificação no nível G de maturidade e capacidade, do modelo de referência MPS.

Objetivos da Medição

I. Pesquisar como os implementadores do MPS evidenciam os requisitos necessários à avaliação do nível G.

II. Apresentar as evidências mais utilizadas pelos implementadores para atender às exigências do nível G do modelo MPS.

Objetivo do Estudo

Analisar os dados fornecidos pelos implementadores do MPS

Com o propósito de caracterizar, identificar e indicar

Com respeito às formas de evidenciar os requisitos necessários à avaliação do nível G

Do ponto de vista dos implementadores do MPS

No contexto do nível G do modelo MPS.

Analisar os dados fornecidos pelos implementadores do MPS

Com o propósito de compreender

Com respeito às evidências mais utilizadas pelos implementadores do MPS para atender às exigências do nível G do modelo MPS.

Do ponto de vista dos pesquisadores

No contexto do processo de certificação do modelo MPS nível G.

Questões e Métricas

I. Analisar os dados fornecidos pelos implementadores do MPS com o propósito de caracterizar, identificar e indicar com respeito às formas de evidenciar os requisitos necessários à avaliação do nível G, do ponto de vista dos implementadores do modelo MPS no contexto do nível G.

Questão 1: GPR1 - Definição do escopo do trabalho. Como você evidencia a definição do escopo de um projeto de software?

Questão 2: GPR2 – Dimensionamento do projeto de software. Como você procura evidenciar os critérios de dimensionamento das tarefas e os produtos de trabalho do projeto?

Questão 3: GPR3 – Definição do modelo e ciclo de vida do projeto de software. Como você evidencia o modelo adotado e as fases do ciclo de vida de um projeto de software?

Questão 4: GPR4 – Estimativa de esforço e custo para a execução das tarefas e dos produtos de trabalho. Qual o método usado para estimar o esforço e o custo para a execução das atividades e geração dos produtos de trabalho?

Questão 5: GPR5 - Estabelecimento de orçamento, cronograma, marcos e/ou pontos de controle do projeto. Como você prefere evidenciar a forma com que são estabelecidos e mantidos estes itens?

Questão 6: GPR6 – Identificação dos riscos do projeto. Como você procura evidenciar os riscos do projeto, seus impactos, probabilidade de ocorrência e prioridades de tratamento?

Questão 7: GPR7 – Definição dos recursos humanos necessários para a execução do projeto. Como você procura evidenciar o planejamento de alocação de recursos humanos dentro do projeto, considerando os perfis e conhecimentos necessários?

Questão 8: GPR8 - Planejamento das tarefas, os recursos e o ambiente de trabalho necessário para executar o projeto. Qual método você utiliza para evidenciar o planejamento dos itens citados acima?

Questão 9: GPR9 - Os dados relevantes do projeto são identificados e planejados quanto à forma de coleta, armazenamento e distribuição. Um

mecanismo é estabelecido para acessá-los, incluindo, se pertinente, questões de privacidade e segurança. Como você evidencia o cumprimento destes requisitos do nível G?

Questão 10: GPR10 - Estabelecimento dos planos de execução do projeto. Como você procura evidenciar a forma de estabelecimento dos planos de execução e como estes são reunidos?

Questão 11: GPR11 - Avaliação da viabilidade de atingir as metas do projeto, considerando as restrições e os recursos disponíveis. Como você evidencia esta análise?

Questão 12: GPR12 - Revisão do Plano do Projeto com todos os interessados. Como você evidencia a revisão do Plano de Projeto e a obtenção do comprometimento de todos os envolvidos?

Questão 13: GPR13 - Monitoramento do progresso do projeto e documentação dos resultados obtidos. Como você evidencia o progresso do projeto, a monitoração efetuada e a documentação dos resultados?

Questão 14: GPR14 - Gerenciamento do envolvimento das partes interessadas no projeto. Como você procura evidenciar o envolvimento dos interessados no projeto?

Questão 15: GPR15 - Revisões em marcos do projeto e conforme estabelecido no planejamento. Como você evidencia a revisões necessárias no projeto?

Questão 16: GPR16 - Registros de problemas identificados e o resultado da análise de questões pertinentes. Como você evidencia o estabelecimento, o registro e o tratamento dos problemas identificados e o resultado da análise de questões pertinentes, incluindo as dependências críticas com as partes interessadas?

Questão 17: GPR17 - Ações para corrigir desvios em relação ao planejado e para prevenir a repetição dos problemas identificados são estabelecidas, implementadas e acompanhadas até a sua conclusão. Quais os métodos utilizados por você para atender a estes requisitos?

Questão 18: GRE1 - Obter o entendimento dos requisitos junto aos fornecedores de requisitos. Como você procura evidenciar o registro do entendimento obtido?

Questão 19: GRE2 - Obter a aprovação dos requisitos de software utilizando critérios objetivos. Como você busca evidenciar a aprovação dos requisitos de software e dos critérios utilizados nesta aprovação?

Questão 20: GRE3 - A rastreabilidade bidirecional entre os requisitos e os produtos de trabalho deve ser estabelecida e mantida. Como você evidencia a criação e manutenção da rastreabilidade bidirecional entre os requisitos e os produtos de trabalho?

Questão 21: GRE4 - Revisões em planos e produtos de trabalho do projeto devem ser realizadas visando identificar e corrigir inconsistências em relação aos requisitos. Como você procura evidenciar a realização destas revisões?

Questão 22: GRE5 - Gerenciamento de mudanças nos requisitos ao longo do projeto. Como você evidencia as mudanças nos requisitos ocorridas no decorrer do projeto?

II. Analisar os dados fornecidos pelos implementadores do MPS com o propósito de compreender, com respeito às evidências mais utilizadas pelos implementadores do MPS, para atender às exigências do nível G do modelo MPS do ponto de vista dos pesquisadores, no contexto do processo de certificação do modelo MPS nível G.

Métrica 1: Dado um conjunto de evidências indicadas como itens de resposta para cada resultado esperado do Nível G do modelo MPS, some o número de vezes que cada uma delas foi selecionada pelos implementadores participantes da pesquisa.

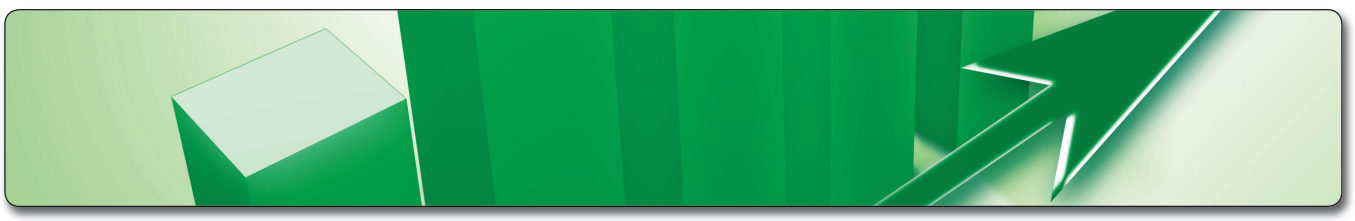
Métrica 2: Dado o total obtido na métrica 1, divida pelo número de implementadores participantes da pesquisa, obtendo o percentual de implementadores que utilizam a prática ou artefato indicado pela opção.

Planejamento

Nas seções a seguir, alguns aspectos importantes serão discutidos para prover uma visão mais clara de todo o processo do survey.

Descrição da Instrumentação

Para analisarmos as práticas mais utilizadas pelos implementadores do modelo MR-MPS ao evidenciarem os resultados esperados no escopo do nível G, foi preciso criar um instrumento que permitisse levantar estes dados de uma maneira clara, objetiva e que pudesse registrá-los para uma posterior análise e geração de informação/conhecimento. A opção adotada foi a criação de um questionário, em formato de planilha, para entrevistar os implementadores, visando obter respostas às perguntas identificadas pela aplicação da metodologia GQM. Esta escolha baseou-se no fato de que um questionário de perguntas diretas seria uma maneira rápida e objetiva de abordar de forma remota o público alvo deste trabalho, além de proporcionar



maior comodidade ao entrevistado e ainda viabilizar a conclusão do processo de pesquisa da forma mais eficiente, suave e menos incisiva, quanto possível.

Os itens sugeridos como possíveis respostas aos questionamentos inseridos no questionário distribuído foram elaborados através de pesquisa aos documentos Guia Geral (SOFTEX, 2007a) e Guia de Implementação (SOFTEX, 2007b), além de consultas ao material do Curso Oficial de Formação de Implementadores MPS.BR (SOFTEX, 2008).

Foi feito um projeto piloto para que avaliássemos os seguintes pontos:

- A compreensão do intuito da pesquisa;
- O entendimento das perguntas;
- O tempo médio de resposta;
- Percentual de resposta a pesquisa; e
- A aceitação do trabalho, críticas e sugestões.

Na execução deste projeto piloto, o questionário mostrou-se adequado, sendo necessários apenas pequenos ajustes na escrita de algumas questões.

Procedimentos de Análise

Os dados coletados para evidenciar os artefatos mais indicados pelos implementadores do modelo MPS participantes da pesquisa, utilizados atender as exigências do nível G do modelo MPS, foram analisados em apenas um lote. Gráficos foram gerados para ilustrar a variação entre as diferentes práticas adotadas em relação às opções existentes.

É importante ressaltar que durante o procedimento de análise foi observado que alguns implementadores utilizaram o campo "Descrição", porém não assinalaram a opção "Outros". Este fato gerou algumas inconsistências nos gráficos, a exemplo do GPR16 e GRE4. Apesar de identificado o problema, os dados gerados pelos implementadores não foram alterados, no intuito de manter a integridade dos dados originais.

Seleção do Contexto

O estudo será conduzido de forma off-line, ou seja, o questionário foi entregue aos participantes e não foi acompanhado.

Ameaças à Validade Interna	
Ameaça	**Tratamento Adotado**
Má instrumentação	Revisão e avaliação do questionário quanto ao formato e à formulação das perguntas.
Rivalidade compensatória.	Como os pesquisados são implementadores confiamos no conhecimento prévio do assunto.
Seleção	Os pesquisadores selecionaram a instituição a ser pesquisada. Os implementadores da COPPE-RJ participantes foram voluntários.
Maturação	Todos os dados foram coletados dentro do mesmo intervalo de tempo e foram analisados todos em lote único.

Tabela 2. Tratamento das Ameaças à Validade da Pesquisa

Ameaças à Validade da Conclusão	
Ameaça	**Tratamento Adotado**
Busca de um resultado específico	Elaborar perguntas que levem os participantes a responder de forma imparcial.
Confiabilidade das medições	Buscou-se utilizar questões objetivas, em detrimento de questões discursivas, na tentativa de evitar a necessidade de interpretação das respostas.
Confiabilidade no tratamento da implementação	Os dados foram coletados com os participantes no mesmo período. Todos foram submetidos ao mesmo questionário e em iguais condições para a resposta.
Confiabilidade do tratamento.	Avaliação do questionário a ser utilizado através de um estudo piloto.
Heterogeneidade dos participantes.	A pesquisa busca conhecer as práticas adotadas pelos implementadores somente para os itens de Gerência de Projetos e Gerência de Requisitos. Além disso, foi escolhida apenas uma instituição implementadora.

Tabela 3. Tratamento das Ameaças à Validade da Pesquisa

Ameaças à Validade da Construção	
Ameaça	**Tratamento Adotado**
Explicação inadequada das construções	Revisão do planejamento, verificando se todas as perguntas dos questionários de caracterização estão claras e bem definidas.
Preocupação dos participantes em relação ao uso dos dados fornecidos	Explicação dos objetivos da pesquisa e da confidencialidade das respostas aos participantes da pesquisa.
Influência da expectativa do pesquisador nos resultados	Revisão do questionário, observando a formulação das perguntas e as possibilidades de resposta fornecidas aos participantes. Buscou-se utilizar questões objetivas, em detrimento de questões discursivas, na tentativa de evitar a necessidade de interpretação das respostas.

Tabela 4. Tratamento das Ameaças à Validade da Pesquisa

Ameaças à Validade Externa	
Ameaça	**Tratamento Adotado**
Interação da seleção e tratamento.	A população utilizada é representativa e inserida no contexto do tratamento. Entretanto, ressalta-se que a generalização para as demais instituições não é o objetivo do presente trabalho.
Interação entre histórico e tratamento	A pesquisa foi realizada logo após a revisão do modelo MPS para que o trabalho pudesse agregar valor referente à atualização o modelo.

Tabela 5. Tratamento das Ameaças à Validade Externa da Pesquisa

Foi realizado em etapa única. Somente um questionário de caracterização de práticas mais utilizadas foi preenchido por cada implementador. Cada participante teve o seu tempo e ambiente para preencher o questionário, colaborando com o estudo.

Seleção dos Participantes

Foi escolhida somente uma instituição implementadora, a COPPE-UFRJ, como universo de amostragem. O intuito foi delimitar um universo de pesquisa para comparação com outras instituições implementadoras em trabalhos

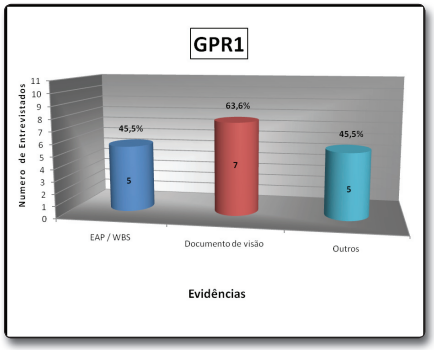


Figura 2. Definição de Escopo de Trabalho (1)

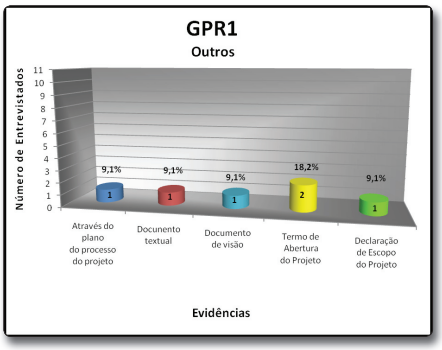


Figura 3. Definição de Escopo de Trabalho (2)

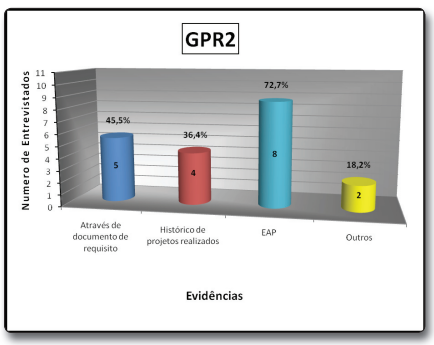


Figura 4. Dimensionamento do projeto de software (1)

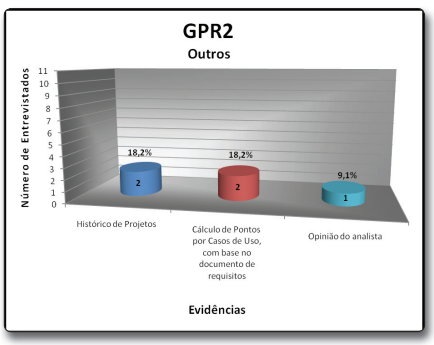


Figura 5. Dimensionamento do projeto de software (2)

Figura 6. Ciclo de vida do projeto de software (1)

Figura 7. Ciclo de vida do projeto de software (2)

futuros. Além disso, a quantidade de implementadores pertencentes à instituição escolhida permitia atingir um número expressivo de respostas, aumentando nossa amostra e conferindo maior confiabilidade ao resultado da pesquisa. Outro ponto importante que influenciou a opção pela COPPE-RJ, foi a proximidade geográfica, pois em caso de um baixo índice de respostas remotas, seria viável uma tentativa de realizar a pesquisa pessoalmente com os implementadores. Portanto, a população atingida pelo survey consiste de implementadores do nível G do modelo MPS certificados junto à SOFTEX, integrantes da instituição COPPE-RJ.

Ameaças à Validade

Validade interna

As ameaças à validade interna estão listadas na **Tabela 2**.

Validade de conclusão

As ameaças à validade de conclusão estão listadas na **Tabela 3**.

Validade de construção

As ameaças à validade de construção estão listadas na **Tabela 4**.

Validade externa

As ameaças à validade externa estão listadas na **Tabela 5**.

Execução da Pesquisa e Análise dos Resultados

A idéia inicial do survey era aplicar o questionário a toda comunidade implementadora do modelo MPS. Porém, ao estudar o universo resultante do que estava sendo pretendido, optou-se por um universo menor e mais controlado, como forma de avaliar a metodologia aplicada e o resultado final do estudo.

Desta maneira o survey foi aplicado a implementadores do modelo MPS nível G, certificados junto à SOFTEX, pertencentes à instituição COPPE-RJ. Eles responderam a perguntas constantes do questionário, tendo em vista os artefatos mais adotados para evidenciar as exigências do nível G do modelo MPS.

O questionário foi distribuído através de correio eletrônico (e-mail), visando primeiramente dar uma maior liberdade aos pesquisados em responder à pesquisa e diminuir o tempo de resposta. Além disso, também se pensou em reduzir o tempo com deslocamentos dos pesquisadores para entrevistar os implementadores. Por este motivo, optou-se por um questionário em meio eletrônico.

As respostas constantes nos documentos recebidos foram catalogadas e agrupadas por resultado esperado (GPR's e GRE's) em planilha Excel e a partir de fórmulas inseridas no aplicativo, foram contabilizadas para que fossem obtidos os índices de preferência entre os implementadores participantes.

É importante frisar que as amostras coletadas neste trabalho foram retiradas de um universo limitado e específico: os implementadores da COPPE-RJ. Ressalta-se ainda que do universo total de possibilidades (26 implementadores da instituição, encontrados no site do

SOFTEX (disponível em www.softex.br)), foram alvo da pesquisa 19 implementadores (73%), devido a problemas encontrados na entrega eletrônica do questionário aos mesmos. Destes, 11 implementadores (58%) responderam ao questionário, com suas opções e considerações a respeito dos itens abordados, o que deu origem à base de dados utilizada neste estudo experimental.

Resultados do Survey para o Processo de Gerência de Projetos

Neste item será mostrado por meio de gráficos como os implementadores participantes da pesquisa realizada evidenciam cada um dos resultados esperados para o processo Gerencia de Projetos (GPR). Estes dados estão explicitados nos gráficos em valores absolutos e relativos.

O primeiro gráfico de cada GPR evidencia as opções de resposta segundo a visão dos pesquisadores para o GPR. Os implementadores tiveram a possibilidade de escolher uma ou mais alternativas sugeridas para cada GPR.

O segundo gráfico de cada GPR evidencia a opção "Outros", onde os implementadores participantes puderam mostrar formas alternativas e particulares de evidenciar um determinado GPR, que não constavam da lista de opções apresentada pelo questionário.

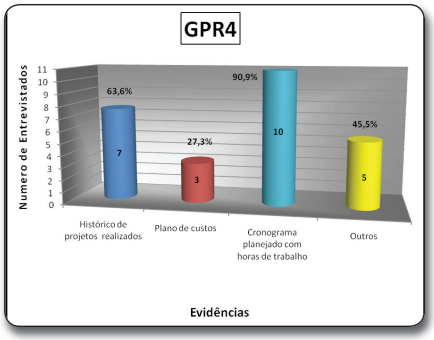


Figura 8. Estimar o esforço e o custo (1)

Figura 9. Estimar o esforço e o custo (2)

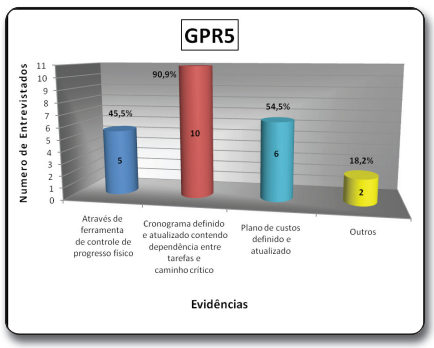


Figura 10. Estabelecimento de orçamento (1)

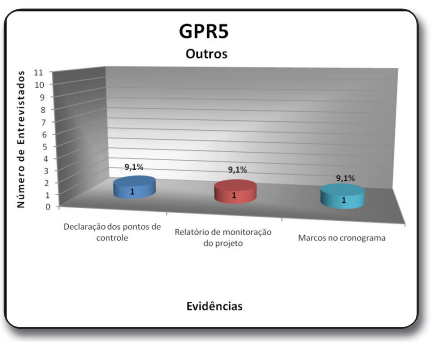


Figura 11. Estabelecimento de orçamento (2)

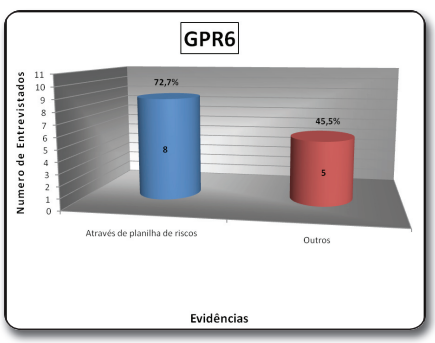


Figura 12. Identificação dos riscos (1)

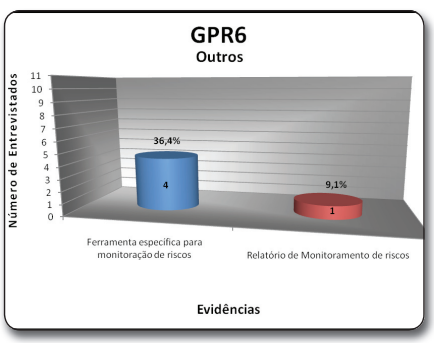


Figura 13. Identificação dos riscos (2)

GPR1 - Definição do escopo do trabalho.

Pergunta: Como você evidencia a definição do escopo de um projeto de software?

A **Figura 2** mostra que a opção Documento de Visão recebendo 7 indicações acrescidas de 2 indicações na opção "Outros" (**Figura 3**) que colocaram o Documento de Visão como forma de evidenciar o GPR1. Porém, aproximadamente 30% dos implementadores selecionaram as opções A e B, assim entende-se que EAP/WBS e o Documento de Visão são complementares como forma de evidenciar o GPR1.

GPR2 - Dimensionamento do projeto de software.

Pergunta: Como você procura evidenciar os critérios de dimensionamento das tarefas e os produtos de trabalho do projeto?

As **Figuras 4 e 5** mostram que mesmo que EAP tenha sido referenciada 8 vezes, o Documento de Requisitos (5) e

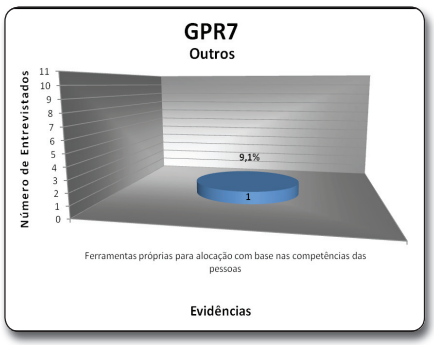


Figura 14. Definição dos recursos humanos (1)

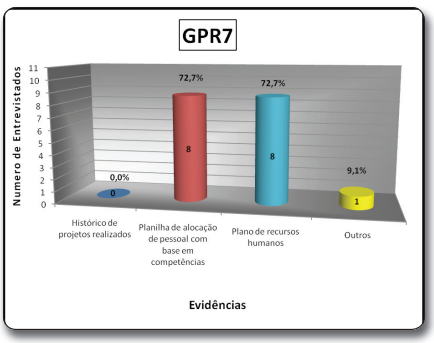


Figura 15. Definição dos recursos humanos (2)

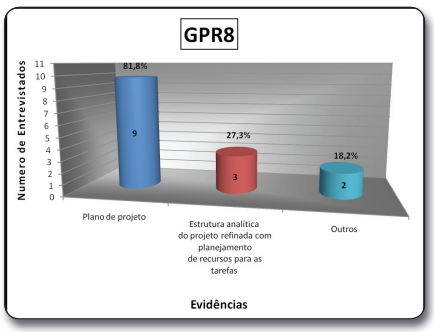


Figura 16. Planejamento das tarefas (1)

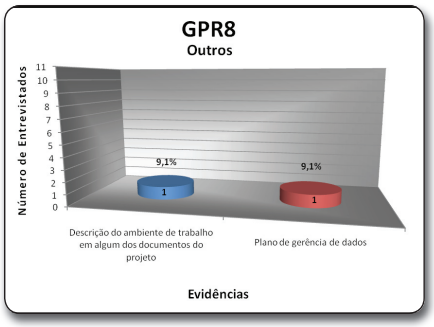


Figura 17. Planejamento das tarefas (2)

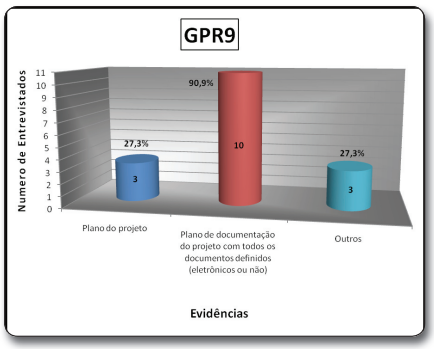


Figura 18. Dados relevantes do projeto (1)

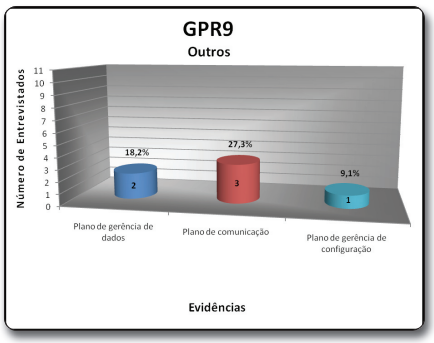


Figura 19. Dados relevantes do projeto (2)

Históricos de Projetos (4+1 na figura 5.4) são formas importantes para a evidenciação do GPR2, haja visto que 46% dos participantes selecionaram estas duas formas de evidenciá-lo.

GPR3 – Definição do modelo e ciclo de vida do projeto de software.

Pergunta: Como você evidência o modelo adotado e as fases do ciclo de vida de um projeto de software?

Com as **Figuras 6 e 7**, claramente mostra-se que o GPR3 é evidenciado pela Descrição das atividades caracterizando o ciclo de vida adotado.

GPR4 – Estimativa de esforço e custo para a execução das tarefas e dos produtos de trabalho.

Pergunta: Qual o método usado para estimar o esforço e o custo para a execução das atividades e geração dos produtos de trabalho?

Para as **Figuras 8 e 9**, a opção "Cronograma planejado com horas de trabalho" (com 10 indicações) poderia ser considerada a evidência que caracteriza melhor o GPR4, mas 64% dos participantes selecionaram a opção "Histórico de Projetos realizados" (7) juntamente com a opção descrita acima. Com isto, entendemos que o conjunto das duas opções evidencia o GPR4.

GPR5 - Estabelecimento de orçamento, cronograma, marcos e/ou pontos de controle do projeto.

Pergunta: Como você prefere evidenciar a forma com que são estabelecidos e mantidos estes itens?

Para as **Figuras 10 e 11**, apesar do item "Cronograma definido e atualizado contendo dependência entre tarefas e o caminho crítico" ter recebido 10 votos, a opção "Plano de custos definido e atualizado" com 6 e o item "Através de ferramentas de controle de progresso físico" com 5 votos sempre estão em conjunto com o primeiro item descrito. Podemos dizer que o "Cronograma definido e atualizado contendo dependência entre tarefas e o caminho crítico" é um item complementar aos artefatos que evidenciam o GPR5.

GPR6 – Identificação dos riscos do projeto.

Pergunta: Como você procura evidenciar os riscos do projeto, seus impactos, probabilidade de ocorrência e prioridades de tratamento?

As **Figuras 12 e 13** possuem indicadores que mostram que 72.7% identificam os riscos através de "Planilha de Riscos". Além disso, 50% dos implementadores que utilizam a planilha de riscos também usam ferramentas próprias para a monitoração dos riscos.

GPR7 – Definição dos recursos humanos necessários para a execução do projeto.

Pergunta: Como você procura evidenciar o planejamento de alocação de recursos humanos dentro do projeto, considerando os perfis e conhecimentos necessários?

Com as **Figuras 14 e 15** podemos dizer

que o "Plano de recursos humanos" e a "Planilha de alocação de pessoal com bases em competências", com 8 votos cada, são ferramentas que em conjunto evidenciam o GPR 7. Em outra análise, podemos dizer também que 46% dos participantes usam as duas ferramentas em conjunto.

GPR8 - Planejamento das tarefas, os recursos e o ambiente de trabalho necessário para executar o projeto.

Pergunta: Qual método você utiliza para evidenciar o planejamento dos itens citados acima?

As **Figuras 16 e 17** evidenciam que o "Plano de Projeto" é o principal artefato para caracterizar o GPR8, pois 50% implementadores que optaram pela "Estrutura analítica do projeto refinada com planejamento de recursos para as tarefas" ou o "Plano de gerencia de projetos" também usam o item mais votado.

GPR9 - Os dados relevantes do projeto são identificados e planejados quanto à forma de coleta, armazenamento e distribuição. Um mecanismo é estabelecido para acessá-los, incluindo, se pertinente, questões de privacidade e segurança.

Pergunta: Como você evidencia o cumprimento destes requisitos do nível G?

As **Figuras 18 e 19** mostram que o "Plano de documentação do projeto com todos os documentos definidos" deve ser a ferramenta para se evidenciar o GPR9, isto por que, todos os outros artefatos utilizam o item acima em conjunto para evidenciar o GPR9.

GPR10 - Estabelecimento dos planos de execução do projeto.

Pergunta: Como você procura evidenciar a forma de estabelecimento dos planos de execução e como estes são reunidos?

A Figura 20 mostra que 100% dos implementadores pesquisados usam o "Plano do projeto integrado" para evidenciar o GPR 10.

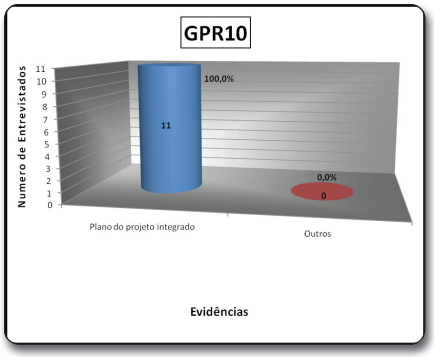


Figura 20. Planos de execução

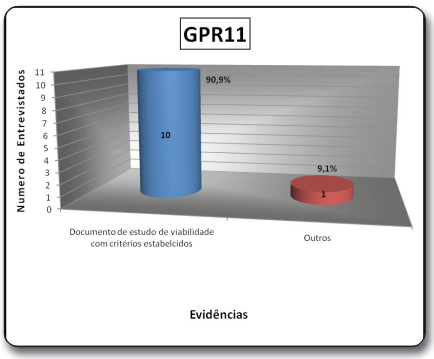


Figura 21. Avaliação da viabilidade (1)

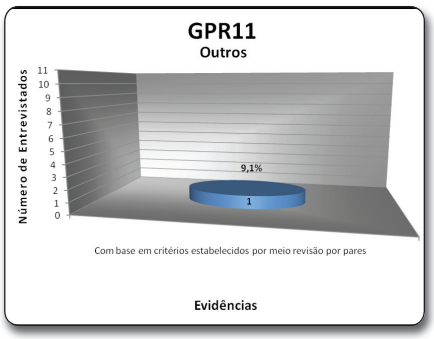


Figura 22. Avaliação da viabilidade (2)

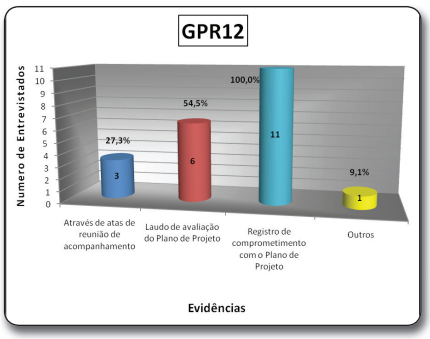


Figura 23. Revisão do Plano do Projeto (1)

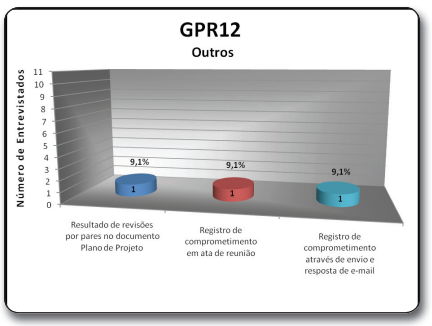


Figura 24. Revisão do Plano do Projeto (2)

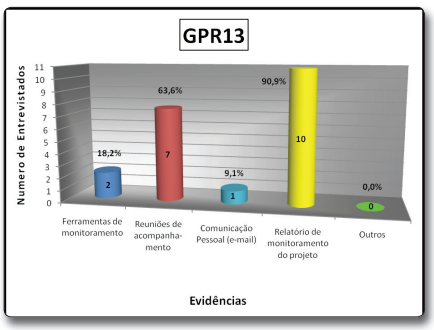


Figura 25. Monitoramento do progresso.

Para o GPR10 a opção "Outros" não foi utilizada.

GPR11 - Avaliação da viabilidade de atingir as metas do projeto, considerando as restrições e os recursos disponíveis.

Pergunta: Como você evidencia esta análise?

As **Figuras 21 e 22** evidenciam que aproximadamente 91% dos implementadores pesquisados utilizam "Documento de estudo de viabilidade, com critérios estabelecidos".

GPR12 - Revisão do Plano do Projeto com todos os interessados.

Pergunta: Como você evidencia a revisão do Plano de Projeto e a obtenção de comprometimento de todos os envolvidos?

As **Figuras 23 e 24** mostram que 100% dos participantes pesquisados optaram pelo "Registro de comprometimento com o plano do projeto". Verificou-se que os 54,5% que optaram pelo "Laudo de avaliação do plano de projeto" também usam o "Registro de comprometimento com o plano do projeto" para evidenciar o GPR12.

GPR13 - Monitoramento do progresso do projeto e documentação dos resultados obtidos.

Pergunta: Como você evidencia o progresso do projeto, a monitoração efetuada e a documentação dos resultados?

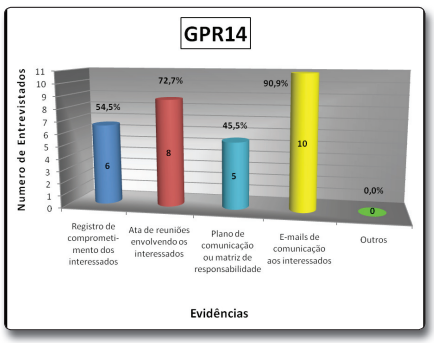


Figura 26. Envolvimento das partes

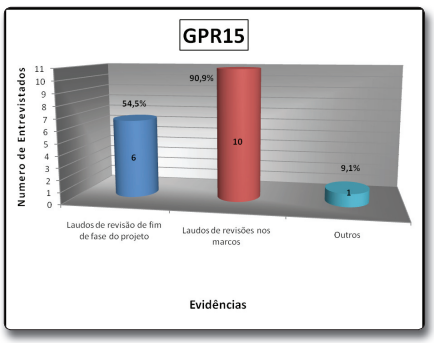


Figura 27. Revisões em marcos (1)

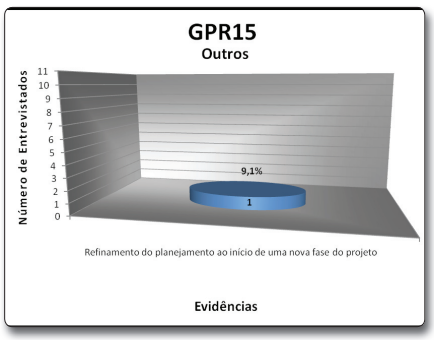


Figura 28. Revisões em marcos (2)

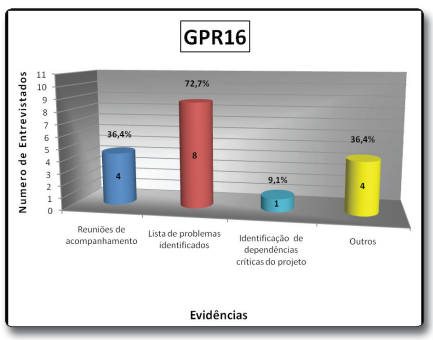


Figura 29. Registros de problemas (1)

Figura 30. Registros de problemas (2)

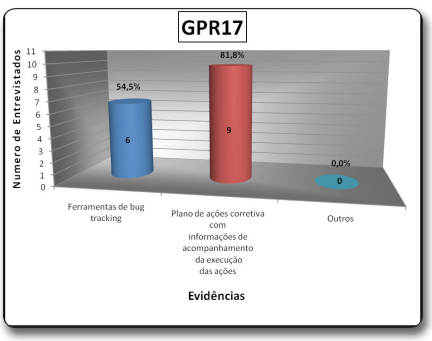


Figura 31. Ações para corrigir desvios

A Figura 25 evidencia que 91% dos pesquisados usam o "Relatório de monitoração do projeto". Verificou-se também que 86% dos pesquisados que usam as "Reuniões de acompanhamento" também utilizam o "Relatório de monitoração do projeto" para complementar a evidenciação do GPR13.

Para o GPR13 a opção "Outros" não foi utilizada.

GPR14 - Gerenciamento do envolvimento das partes interessadas no projeto.

Pergunta: Como você procura evidenciar o envolvimento dos interessados no projeto?

A **Figura 26** possui uma grande variedade de opiniões para a forma de se evidenciar o GPR14. Caso fossemos analisar de forma simplista poderíamos dizer que "Emails de comunicação aos interessados" com 91% das opções assinaladas é simplesmente o artefato mais utilizado, mas 81% usam mais do que um artefato para evidenciar o GPR e 64% usam pelo menos três artefatos para evidenciar o GPR14.

Para o GPR14 a opção "Outros" não foi utilizada.

GPR15 - Revisões em marcos do projeto e conforme estabelecido no planejamento.

Pergunta: Como você evidencia as revisões necessárias no projeto?

Além do que está evidenciado pelas **Figuras 27 e 28**, verificou-se que 83% dos participantes que marcaram a opção "Laudos de revisão de fim de fase de projeto" também selecionaram a opção "Laudos de revisões nos marcos" que recebeu 91% de aceitação.

GPR16 - Registros de problemas identificados e o resultado da análise de questões pertinentes.

Pergunta: Como você evidencia o estabelecimento, o registro e o tratamento dos problemas identificados e o resultado da análise de questões pertinentes, incluindo as dependências críticas com as partes interessadas?

As **Figuras 29 e 30** evidenciam que 72,7% dos participantes optaram por "Lista de problemas identificados". Todos os que usam "Ferramentas de bug-tracking não apenas para os defeitos" (27,2%) usam também a opção mais votada e 75% dos que usam "Reuniões de acompanhamento" também usam a mais selecionada.

GPR17 - Ações para corrigir desvios em relação ao planejado e para prevenir a repetição dos problemas identificados são estabelecidas, implementadas e acompanhadas até a sua conclusão.

Pergunta: Quais os métodos utilizados por você para atender a estes requisitos?

A **Figura 31** nos mostra que o "Plano de ação corretiva com informações de acompanhamento da execução das ações" corresponde a 82% das respostas. Verificou-se que 67% dos que escolheram "Ferramentas de bug-tracking" também usam o "Plano de ação corretiva com informações de acompanhamento da execução das ações".

Resultados do Survey para o Processo de Gerencia de Requisitos

Neste item mostraremos em forma de gráficos como os implementadores participantes da pesquisa realizada evidenciam cada um dos resultados esperados para o processo Gerencia de Requisitos (GRE). Estes dados estão explicitados nos gráficos em valores absolutos e relativos.

O primeiro gráfico de cada GRE mostra as opções de resposta segundo a visão dos pesquisadores para o GRE. Os

implementadores tiveram a possibilidade de escolher uma ou mais alternativas sugeridas para cada GRE.

O segundo gráfico de cada GRE retrata a opção "Outros", onde os implementadores participantes puderam mostrar formas alternativas e particulares de evidenciar um determinado GRE que não constavam da lista de opções apresentada pelo questionário.

GRE1 - Obter o entendimento dos requisitos junto aos fornecedores de requisitos.

Pergunta: Como você procura evidenciar o registro do entendimento obtido?

Com as **Figuras 32 e 33** verificam-se que "Especificação de requisitos do sistema" e "Atas de reuniões ou entrevistas com os stakeholders" são usados de forma conjunta para que seja evidenciado o GRE1.

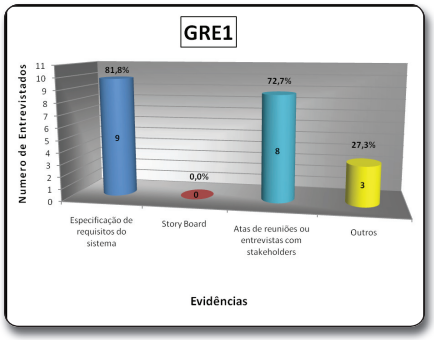


Figura 32. Obter o entendimento dos requisitos (1)

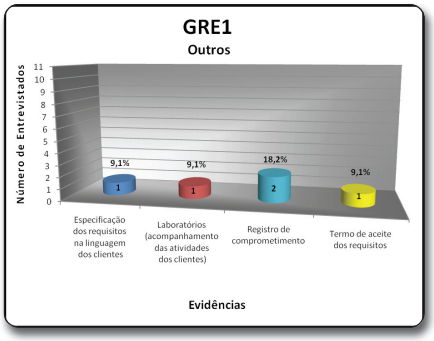


Figura 33. Obter o entendimento dos requisitos (2)

GRE2 - Obter a aprovação dos requisitos de software utilizando critérios objetivo.

Pergunta: Como você busca evidenciar a aprovação dos requisitos de software e dos critérios de utilizados nesta aprovação?

As **Figuras 34 e 35** identificam que "Laudo de avaliação dos requisitos com base" e "Registro de comprometimento com os requisitos estabelecidos, revisto quando há mudanças." são usados de forma conjunta para que seja evidenciado o GRE2. Verificou-se ainda que 89% dos implementadores pesquisados que marcaram a opção "Registro de comprometimento com os requisitos estabelecidos, revisto quando há mudanças" também registraram a "Laudo de avaliação dos requisitos com base".

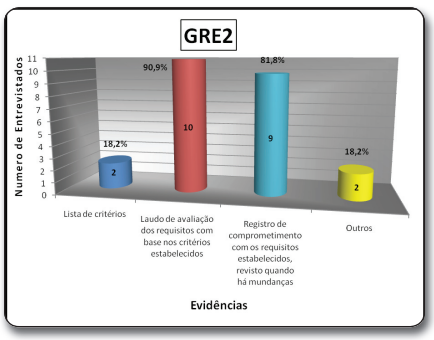


Figura 34. Aprovação dos requisitos (1)

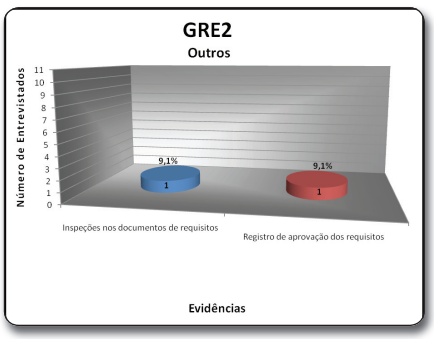


Figura 35. Aprovação dos requisitos (2)

GRE3 - A rastreabilidade bidirecional entre os requisitos e os produtos de trabalho deve ser estabelecida e mantida.

Pergunta: Como você evidencia a criação e manutenção da rastreabilidade bidirecional entre os requisitos e os produtos de trabalho?

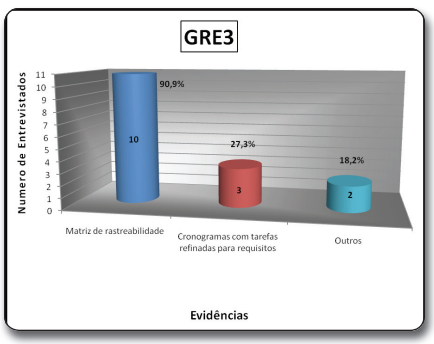


Figura 36. Rastreabilidade bidirecional (1)

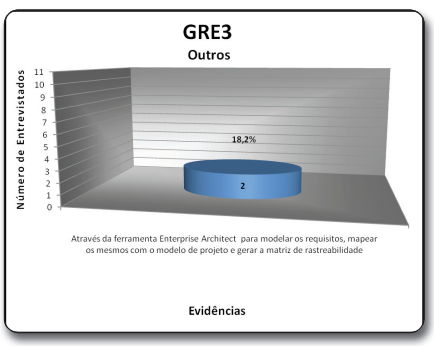


Figura 37. Rastreabilidade bidirecional (2)

As **Figuras 36 e 37** retratam que a "Matriz de rastreabilidade" é o instrumento mais utilizado (91%). O "Cronograma com tarefas refinadas para requisitos" e a solução de mercado "Enterprise Architect" são também mencionadas.

GRE4 - Revisões em planos e produtos de trabalho do projeto devem ser realizadas visando identificar e corrigir inconsistências em relação aos requisitos.

Pergunta: Como você procura evidenciar a realização destas revisões?

As **Figuras 38 e 39** mostram que a opção "Laudo de revisões realizadas" com 91% foi a mais selecionada, mas 75% dos participantes que marcaram a opção "Plano de ação corretiva para inconsistências identificadas" também usaram a primeira resposta. Com isto, acredita-se que para quem utiliza o "Plano de ação corretiva para inconsistências identificadas", os "Laudos

Tabela Resumo da Pesquisa			
Resultados Esperados	Prática mais indicada pelos implementadores participantes da pesquisa, através das opções pré-estabelecidas	Número de Implementadores que indicaram a prática	% de Preferência
GPR1	Documento de Visão	7	63,6%
GPR2	EAP (Estrutura Analítica do Projeto)	8	72,7%
GPR3	Através da descrição das atividades caracterizando o ciclo de vida adotado	10	90,9%
GPR4	Cronograma planejado com horas de trabalho	10	90,9%
GPR5	Cronograma definido e atualizado contendo dependência entre tarefas e caminho crítico	10	90,9%
GPR6	Através de planilha de riscos	8	72,7%
GPR7	Histórico de Projetos Realizados	10	90,9%
GPR8	Plano de Projeto	9	81,8%
GPR9	Plano de documentação do projeto com todos os documentos definidos (eletrônicos ou não)	10	90,9%
GPR10	Plano do projeto integrado	11	100,0%
GPR11	Documento de estudo de viabilidade com critérios estabelecidos	10	90,9%
GPR12	Registro de comprometimento com o plano de projeto	11	100,0%
GPR13	Relatório de monitoramento do projeto	10	90,9%
GPR14	E-mail de comunicação aos interessados	10	90,9%
GPR15	Laudo de revisões nos marcos	10	90,9%
GPR16	Lista de problemas identificados	8	72,7%
GPR17	Plano de ações corretivas com informações de acompanhamento da execução das ações	9	81,8%
GRE1	Especificação de requisitos do sistema	9	81,8%
GRE2	Laudo de avaliação dos requisitos com base nos critérios estabelecidos	10	90,9%
GRE3	Matriz de rastreabilidade	10	90,9%
GRE4	Laudo de revisões realizadas	10	90,9%
GRE5	Registro de análise de impacto realizada	9	81,8%

Tabela 6. Resumo dos resultados.

de revisões realizadas" também são necessários para evidenciar o GRE4.

GRE5 - Gerenciamento de mudanças nos requisitos ao longo do projeto.

Pergunta: Como você evidencia as mudanças nos requisitos ocorridas no decorrer do projeto?

As **Figuras 40 e 41** mostram que o instrumento "Registro de análises de impacto realizadas" é o mais utilizado com 82%. Verificou-se também que praticamente 75% dos implementadores utilizam outros artefatos em conjunto para evidenciar o GRE5.

Conclusões

Neste artigo, foram apresentados todos os resultados obtidos na pesquisa, estratificados por resultado esperado, tanto para o processo de Gerência de Projetos (GPR), quanto para o processo de Gerência de Requisitos (GRE). A **Tabela 6** apresenta um resumo com os artefatos mais indicados pelo grupo de implementadores participantes da pesquisa. Cada linha da tabela traz um dos resultados esperados pelos processos de Gerência de Projetos ou Gerência de Requisitos e a evidência mais indicada na pesquisa pelos implementadores.

Entre as contribuições deste trabalho destacamos as seguintes:

- Uma revisão bibliográfica sobre o modelo MR-MPS e sua versão atual, que entrou em vigor em janeiro de 2008, tendo em vista que o trabalho foi planejado e executado sobre a nova versão do modelo;
- A utilização da metodologia GQM para a concepção de questionários do survey. Devido ao alto número de respostas, 11 respostas de 19 possíveis (~58%), acreditamos que o questionário resultante seja adequado para este tipo de pesquisa. Assim, o planejamento do survey pode servir como referência para o planejamento de novos surveys seguindo a metodologia GQM, mesmo que com objetivos diferentes;
- As informações resultantes da análise dos dados. Espera-se que estas informações possam ser proveitosas para as instituições implementadoras e avaliadoras do modelo MPS, como forma de conhecer quais os artefatos mais utilizados por parte significativa de uma das instituições que realizam

esta atividade no Brasil, a COPPE-RJ, ao evidenciar as exigências do nível G do modelo MPS. Acredita-se que estas informações são a maior contribuição deste trabalho, pois podem ser utilizadas para auxiliar e orientar empresas e implementadores em futuras implementações do modelo MPS, nível G.

Limitações

• A pesquisa foi baseada no universo de implementadores de uma só instituição implementadora, a COPPE/UFRJ - FUNDAÇÃO COPPETEC. Assim, o número de implementadores que responderam a pesquisa não retrata uma amostragem válida para que pudéssemos afirmar que as práticas relacionadas em nossa pesquisa são as mais utilizadas em um contexto mais amplo e geral;

• A elevada utilização da opção "Outros" na resposta a algumas perguntas do questionário sugere a possibilidade de revisão e melhoria na elaboração das alternativas inicialmente apresentadas;

• A estratificação das respostas contidas no campo "Descrição", quando a opção "Outros" foi selecionada pelos participantes, precisou ser interpretada para que pudesse ser agrupada em categorias, pois muitas vezes, uma mesma forma de evidenciar uma exigência foi descrita de várias formas por diferentes participantes. Desta forma, as respostas relacionadas à opção "Outros" e conseqüentemente ao campo "Descrição", durante o processo de análise, podem ter absorvido alguma influência dos pesquisadores. ●

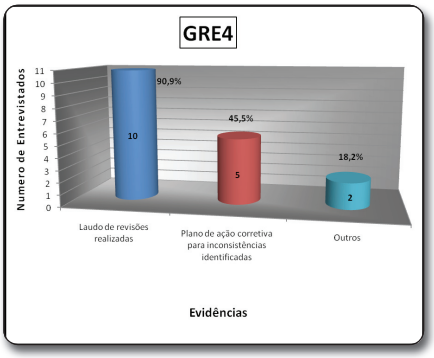


Figura 38. Revisões em planos (1)

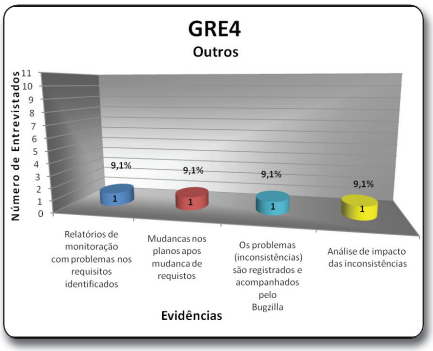


Figura 39. Revisões em planos (2)

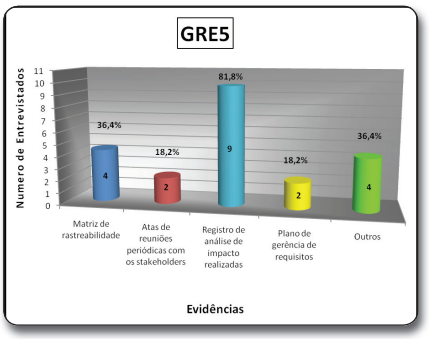


Figura 40. Gerenciamento de mudanças (1)

Figura 41. Gerenciamento de mudanças (2)

Referências

Dorfmann, M. e Thayer, R., Standards, Guidelines, and Examples of System and Software Requirements Engineering. Los Alamitos, CA: IEEE Computer Society Press, 1990.

Fenton, N., Pfleeger, S. L., Software Metrics: A Rigorous and Practical Approach, International Thomson Computer Press, London, UK, 1997, 2a Ed.

IEEE Std 610.12 - IEEE Standard Glossary of Software Engineering Terminology, Institute of Electrical and Electronics Engineers, 1990.

ISO/IEC 15504-1, The International Organization for Standardization and the International Electrotechnical Commission. ISO/IEC 15504-1: Information Technology - Process Assessment – Part 1 - Concepts and Vocabulary, Geneve: ISO, 2004.

ISO/IEC 15504-2, The International Organization for Standardization and the International Electrotechnical Commission. ISO/IEC 15504-2: Information Technology - Process Assessment – Part 2 - Performing an Assessment, Geneve: ISO, 2003.

ISO/IEC 15504-5, The International Organization for Standardization and the International Electrotechnical Commission. ISO/IEC 15504-5: Information Technology - Process Assessment - Part 5: An exemplar Process Assessment Model, Geneve: ISO, 2006.

Pressman, R. S., Engenharia de Software. 6 ed. São Paulo: McGraw-Hill, 2006.

Project Management Institute – PMI. A Guide to the Project Management Body of Knowledge - PMBOK™, Syba: PMI Publishing Division, 2004. Disponível em: <www.pmi.org>.

SOFTEX. MPS.BR – Guia Geral, versão 1.2, Disponível em: http://www.softex.br, junho 2007a.

SOFTEX. MPS.BR – Guia de Implementação – Parte 1 Nível G, versão 1.1, junho 2007b. Disponível em: www.softex.br.

SOFTEX, Apostila do Curso Oficial de Formação de Implementadores MPS.BR - versão 1.2, 2008.

Wohlin, C., Runeson, P., Martin, H., Ohlsson, M. C., Regnell, B., Wesslén, A., Experimentation in Software Engineering: An Introducion. Massachussets: Kluwer Academic Publishers, 2000.

Chegou o Sistema de Créditos DevMedia

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

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

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

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

Saiba mais sobre o Sistema de Créditos!

DevMedia GROUP

www.devmedia.com.br/creditos