engenharia de software magazine

Métodos Ágeis
Estimativas ágeis com planning poker

Planejamento
Conheça o modelo de maturidade para gerenciamento de projetos do PMI, o OPM3

DevMedia group Ano I - Edição 09

ISSN 1983127-7

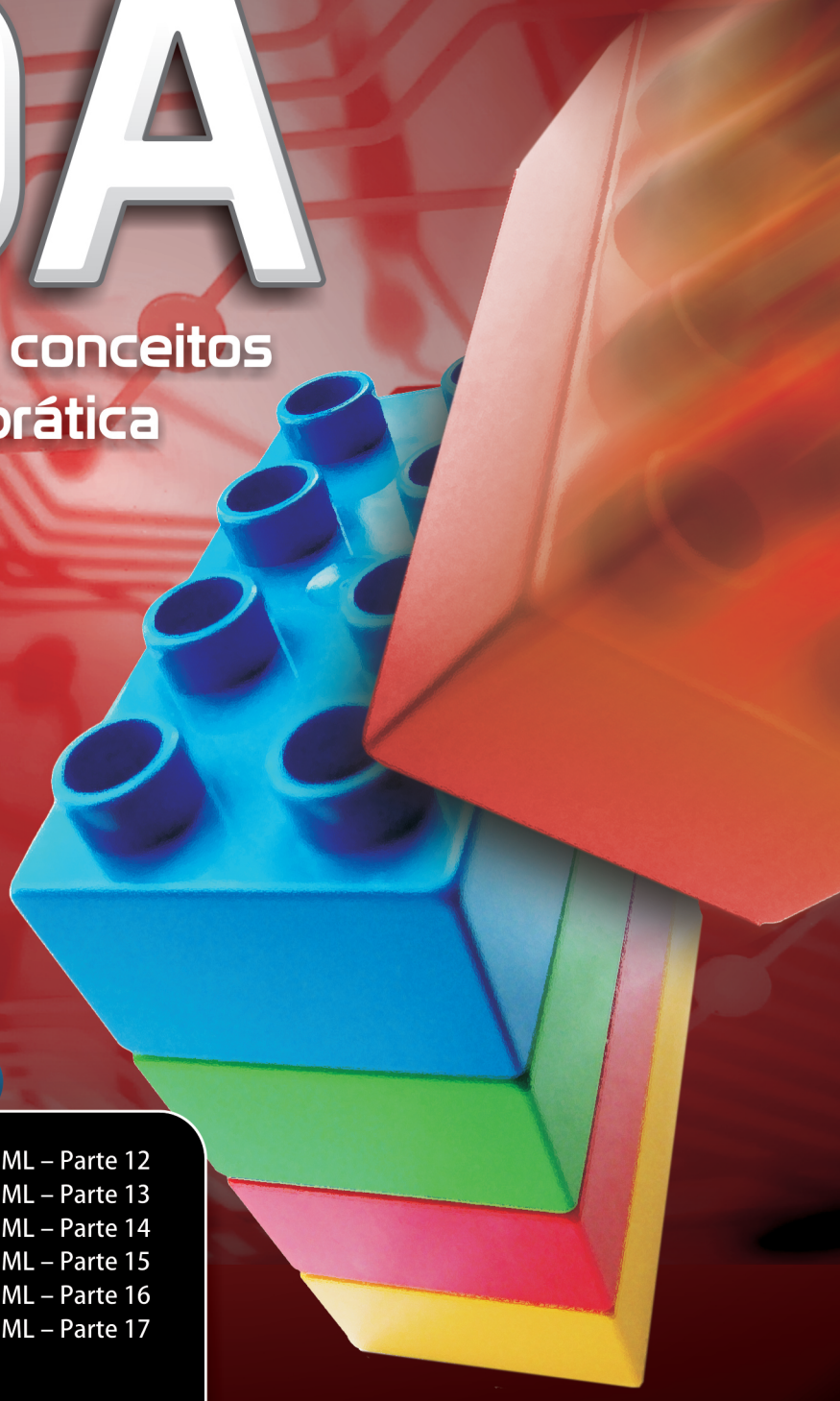
MDA

Conheça os principais conceitos e como aplicá-los na prática

Requisitos
Conheça o diagrama de casos de uso e como detalhar sua descrição

Projeto
Entenda os princípios básicos sobre como avaliar a confiabilidade de um software

Processo
Entenda os conceitos por trás dos sistemas de gestão de qualidade



Aulas desta edição:

- Introdução à Construção de Diagrama de Classes da UML – Parte 12
- Introdução à Construção de Diagrama de Classes da UML – Parte 13
- Introdução à Construção de Diagrama de Classes da UML – Parte 14
- Introdução à Construção de Diagrama de Classes da UML – Parte 15
- Introdução à Construção de Diagrama de Classes da UML – Parte 16
- Introdução à Construção de Diagrama de Classes da UML – Parte 17
- Arquitetura Orientada por Modelos
- Persistência de Objetos com o Framework Hibernate
- Persistência de Objetos com o SGBDOR Caché

Conhecimento faz diferença!

Mais de 60 mil downloads na primeira edição!

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

Faça um *up grade* em sua carreira.

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

DevMedia GROUP

engenharia de software magazine

Ano 1 - 9ª Edição 2009 **Impresso no Brasil**

Atendimento ao Leitor

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

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

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

Publicidade

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

Kaline Dolabella
publicidade@devmedia.com.br

Fale com o Editor!

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

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

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

EDITORIAL

"A MDA (Model Driven Architecture - Arquitetura Orientado por Modelos) propõe separar a lógica de negócio, plataforma e tecnologia, de tal forma que as modificações na plataforma não afetem as aplicações existentes e a lógica de negócio evolua independente da tecnologia. Com isso, os benefícios são evidentes, tais como a redução de custo do ciclo de vida do projeto, a redução do tempo de desenvolvimento para novas aplicações, o aumento da qualidade da aplicação e o aumento do retorno no investimento em tecnologias. Para obter tais benefícios, a MDA faz uso de modelos, os quais podem ser independentes ou específicos de uma plataforma. Na MDA, o Modelo Independente de Plataforma (PIM - Platform Independent Model) será submetido a processos de transformações a fim de se obter como destino um ou mais Modelos Específicos de Plataforma (PSM - Platform Specific Model). Ao final do processo, os PSM poderão ser transformados em código fonte."

Neste contexto, esta edição da Engenharia de Software Magazine traz como tema de capa MDA. Para abordar este tema, destacamos dois artigos:

MDA – Arquitetura Orientada por Modelos

Este artigo trata do uso do framework MDA (Model Driven Architecture) em um estudo de caso prático no qual são aplicados seus principais princípios, tais como a separação das regras de negócio da implementação.

Model Transformation com ATL

Neste artigo são apresentados alguns fundamentos iniciais sobre MMD e MDA, e então, é posto em prática o conceito de transformações de modelos (Model Transformation).

Além destas duas matérias, esta edição traz mais cinco artigos:

- Estimativas Ágeis com Planning Poker;
- Uma Análise do Modelo de Maturidade OPM3;
- UML – Casos de Uso;
- Sistemas de Gestão de Qualidade;
- Confiabilidade de Software: Etapas na Determinação da Confiabilidade de Software.

Desejamos uma ótima leitura!

Equipe Editorial Engenharia de Software Magazine

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

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

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

Caro Leitor,
Para esta edição, temos um conjunto de 9 vídeo aulas. Estas vídeo aulas estão disponíveis para download no Portal da Engenharia de Software Magazine e certamente trarão uma significativa contribuição para seu aprendizado. A lista de aulas publicadas pode ser vista abaixo:

Tipo: Projeto
Título: Arquitetura Orientada por Modelos
Autor: Marco Antônio Pereira Araújo
Mini-Resumo: A MDA (Model Driven Architecture - Arquitetura Orientado por Modelos) propõe a separação dos modelos de domínio daqueles particulares para determinadas plataformas ou tecnologias, de tal forma que modificações técnicas não influenciem no modelo de negócio da aplicação, que pode evoluir independente da tecnologia utilizada. Esta vídeo aulas apresenta de forma prática como proceder com a transformação de modelos seguindo esta abordagem.

Tipo: Projeto
Título: Persistência de Objetos com o Framework Hibernate
Autor: Marco Antônio Pereira Araújo
Mini-Resumo: Camadas de mapeamento objeto-relacionais são alternativas bastante utilizadas quando da utilização de Sistemas de Gerenciamento de Bancos de Dados Relacionais em aplicações orientadas a objetos. Esta vídeo aula apresenta o Framework Hibernate que, de forma simples, possibilita a persistência de objetos em bancos de dados relacionais.

Tipo: Projeto
Título: Persistência de Objetos com o SGBDOR Caché
Autor: Marco Antônio Pereira Araújo
Mini-Resumo: Uma alternativa para a persistência de objetos é através da utilização de Sistemas de Gerenciamento de Bancos de Dados Objeto-Relacionais, também conhecidos como Híbridos ou Universais. Esta vídeo-aula apresenta o Caché, um SGBDOR para persistência de objetos, com o objetivo de demonstrar este tipo de técnica de persistência.

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 12
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade à elaboração do diagrama de classes para o sistema de locadora de veículos. Em particular, são definidos os relacionamentos que existirão entre as classes do modelo.

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 13
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade à elaboração do diagrama de classes para o sistema de locadora de veículos. Em particular, os atributos identificados são atribuídos às suas respectivas classes e acrescentados ao diagrama de classes.

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 14
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade à elaboração do diagrama de classes para o sistema de locadora de veículos. Em particular, são identificadas as operações que deverão estar contempladas no diagrama de classes.

Tipo: Projeto
Título: Introdução à Construção de Diagrama de Classes da UML – Parte 15
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade à elaboração do diagrama de classes para o sistema de locadora de veículos. Em particular, as operações identificadas na aula anterior são atribuídas às suas respectivas classes e são acrescentadas ao diagrama de classes.

Título: Introdução à Construção de Diagrama de Classes da UML – Parte 16
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade ao estudo de caso de um sistema de gestão de festas. Neste estudo de caso, partiremos dos requisitos funcionais do software. A partir deles, será elaborado o diagrama de casos de uso do sistema. Com o diagrama de casos de uso elaborado, serão especificados três casos de uso. Por último, será elaborado o diagrama de classes com base nos requisitos funcionais e casos de uso especificados. Nesta aula, o foco será a especificação do caso de uso configurar festa.

Título: Introdução à Construção de Diagrama de Classes da UML – Parte 17
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta vídeo aula dá continuidade ao estudo de caso de um sistema de gestão de festas. Neste estudo de caso, partiremos dos requisitos funcionais do software. A partir deles, será elaborado o diagrama de casos de uso do sistema. Com o diagrama de casos de uso elaborado, serão especificados três casos de uso. Por último, será elaborado o diagrama de classes com base nos requisitos funcionais e casos de uso especificados. Nesta aula, o foco será a especificação do caso de uso configurar festa – parte 2.

ÍNDICE

Você não está mais sozinho.

Estimativas Ágeis com Planning Poker

Dairton Bassi

dbassi@gmail.com

Mestre em Engenharia de Software com ênfase em Métodos Ágeis pelo IME-USP. Bacharel em Ciência da Computação pelo IME-USP. Membro-fundador da AgilCoop e criador do Encontro Ágil (www.encontroagil.com.br). Especialista em implantação de metodologias ágeis. Ministra cursos e palestras sobre métodos ágeis. Atuou como programador, líder de desenvolvimento e consultor em diversas instituições do setor público e privado.

Criar boas estimativas nunca foi uma tarefa simples para a indústria de software. Ainda hoje, essa é uma de suas principais dificuldades, pois uma estimativa é basicamente uma previsão e os seres humanos tendem a ser otimistas ao fazer previsões. Esse otimismo faz com que os prazos sejam, na maioria das vezes, menores do que o necessário para produzir o software. Na tentativa de entregar rapidamente, é comum menosprezar ou desconsiderar dificuldades técnicas. Além disso, a falta de um vasto conhecimento sobre o que será desenvolvido colabora para que as estimativas não sejam precisas. E pequenas mudanças no escopo do projeto contribuem para que elas sejam pouco duradouras.

Para chegar a estimativas, há diversas técnicas usadas pela indústria de software. Neste artigo veremos uma que é compatível com metodologias ágeis de desenvolvimento de software, o Planning Poker.

Neste artigo veremos

Neste artigo entenderemos as dificuldades da obtenção de estimativas duradouras e apresentaremos uma técnica usada em metodologias ágeis para mensurar o esforço e a duração do desenvolvimento do software, o Planning Poker.

Qual a finalidade

Usando planning poker, a equipe de desenvolvimento chegará a estimativas que dimensionam o volume que trabalho e, a partir delas, terá condições de determinar prazos razoáveis para a entrega do software.

Quais situações utilizam esses recursos?

O uso do Planning Poker permite que todos os envolvidos dêem suas opiniões e participem do planejamento ao mesmo tempo em que tiram suas dúvidas. Além de servir como uma ferramenta para chegar a estimativas, o Planning Poker aumenta a comunicação dentro da equipe e promove o aprendizado a respeito do sistema que será desenvolvido.

Por que estimar?

Ter estimativas confiáveis para o desenvolvimento de software é fundamental para que os profissionais das áreas de negócios possam orientar suas atividades.

A partir da previsão de término do desenvolvimento do software, ações de marketing, comerciais e publicitárias são preparadas. Quando as pessoas acreditam em estimativas pouco realistas, a frustração e o estresse são inevitáveis, pois as expectativas para a entrega do software não são atendidas e isso compromete muito trabalho.

Fatores que dificultam as estimativas

Dar uma estimativa significa fazer uma previsão, e isso não é trivial quando o contexto está sujeito a mudanças. Para software, previsões estão sujeitas à volatilidade de muitas variáveis. O escopo da tarefa estimada e o seu executor são dois exemplos de variáveis difíceis de controlar. Alguns outros fatores que contribuem para invalidar estimativas são:

- **Equipe:** mudanças na equipe podem causar perdas de conhecimento do projeto. Mesmo se houver uma boa documentação, há informações que estão na mente dos envolvidos que só são adquiridas com experiência no projeto e vivência com a equipe;
- **Executor:** indisponibilidades de um especialista atrasam a execução porque o desenvolvimento irá parar ou a velocidade será reduzida devido a uma pessoa menos qualificada para realizar a tarefa;
- **Mercado:** o mercado pode sinalizar novas tendências e forçar mudanças nas funcionalidades ou no escopo da solução;
- **Melhorias:** quando as funcionalidades são testadas por clientes ou usuários, suas sugestões podem mudar o comportamento destas funcionalidades;
- **Concorrência:** o comportamento da concorrência pode forçar mudanças ou até melhorias nas funcionalidades. Por exemplo, se outra empresa atacar o mercado que você está visando, pode ser preciso adaptar a sua estratégia, com impacto nas características do software em desenvolvimento.

Para alguns desses fatores, pode-se definir valores no início do projeto com o preço de abrir mão da flexibilidade caso fatores que estão fora do controle mudem. Por exemplo, a estratégia dos concorrentes, as tendências do mercado e a permanência dos funcionários na equipe são fatores que nem sempre estão sob controle.

Métodos Ágeis e Estimativas

O primeiro passo para resolver um problema é tomar consciência de que ele existe. Metodologias Ágeis aceitam a existência de diversos fatores que dificultam a criação de previsões precisas e duradouras.

Equipes ágeis baseiam-se na comunicação e na transparência. Ao invés de tratar suas estimativas como fatos, admitem que existe uma incerteza associada a ela e evidenciam isso para que o cliente e outros envolvidos também tomem ciência do grau de dificuldade de cada funcionalidade. Estimativas de longo prazo e funcionalidades complexas geralmente possuem graus maiores de incerteza associados. À medida que o tempo passa e o conhecimento sobre o assunto aumenta, as estimativas podem ser melhoradas considerando mais detalhes do projeto e com probabilidades mais altas de sucesso.

Estimativas de tamanho e estimativas de duração

Os inúmeros fatores que contribuem para agregar instabilidade ao projeto também colaboram para invalidar as estimativas de duração do projeto. Em projetos onde há muitos fatores de instabilidade, as estimativas tendem a durar pouco tempo pois mudanças no projeto contribuem para que elas percam rapidamente a sua validade. Para ter as estimativas atualizadas com freqüência, mas evitar que elas tenham que ser refeitas constantemente, é importante separar as estimativas de tamanho das estimativas de duração. As estimativas de tamanho são medidas do volume de trabalho. As estimativas de duração estão associadas à quantidade de tempo necessária para executar esse trabalho.

Estimativas de tamanho requerem a compreensão das características da funcionalidade que será implementada. A grande vantagem dessas estimativas é que uma vez feitas, não se desatualizam, pois o tamanho da tarefa é independente do tempo disponível para realizá-la e da quantidade de pessoas na equipe.

Estimativas de duração são obtidas a partir das estimativas de tamanho e da velocidade da equipe de desenvolvimento. Estimar a duração corresponde precisamente em dizer quanto tempo a equipe levará para concluir o trabalho. Para a mesma tarefa, o tempo para a sua realização pode variar, por exemplo, conforme o tamanho da equipe e a disponibilidade de seus membros, pois esses fatores influem na velocidade de desenvolvimento. Quando a velocidade da equipe muda durante o projeto, as estimativas de duração podem ser facilmente recalculadas com base nas estimativas de tamanho.

A velocidade da equipe é a quantidade de tarefas que ela consegue concluir em um período fixo, por exemplo, a cada semana, a cada quinzena ou a cada iteração do projeto.

A estimativa de duração diz o tempo necessário para a conclusão da tarefa considerando uma determinada velocidade. Para obtê-la basta dividir a estimativa de tamanho pela velocidade da equipe:

```
Estimativa_de_duração = Estimativa_de_tamanho / velocidade
```

Para entender a diferença entre as estimativas de tamanho e de duração, vamos, por exemplo, considerar uma tarefa simples. Suponha que a tarefa é digitar um texto. Para isso, faremos a estimativa de tamanho, depois verificaremos a velocidade e, finalmente, obteremos a duração. Para estimar o tamanho, precisamos considerar o volume de trabalho que a digitação requer. Meu palpite é de que esse texto tem aproximadamente três páginas. Essa foi a minha estimativa de tamanho. Agora poderemos chegar a uma estimativa de duração se considerarmos a minha velocidade de digitação. Não sou o melhor digitador, gasto aproximadamente 15 minutos para digitar cada página. Esta é a minha velocidade: 1 página a cada 15 minutos ou, 4 páginas por hora. Portanto, a estimativa de duração é de 45 minutos para concluir as três páginas. Porém, se enquanto digito, estiver assistindo televisão, minha velocidade cai para 1 página a cada 20 minutos, logo, a estimativa de duração será de 60 minutos para

terminar a tarefa. Se em paralelo à digitação eu tirar dúvidas de companheiros de trabalho em um instant messager, a minha velocidade cairá para 1 página a cada 40 minutos e a estimativa de duração da tarefa será de 120 minutos.

Neste exemplo, a estimativa de tamanho precisou ser feita apenas uma vez. Depois, ela serviu de base para chegar a diversas estimativas de duração, conforme a dedicação dispensada à tarefa. Com essa separação entre volume de trabalho e tempo de execução, a estimativa de tamanho pode ser usada para estimar a duração, mesmo se a tarefa for delegada para outros executores. Por exemplo, para um digitador profissional, que pode fazer o mesmo trabalho muito mais rápido.

Técnicas para Estimar

Existem diversas maneiras de mensurar o desenvolvimento de software. Na prática, ao invés de tentar descobrir ou calcular, as melhores estimativas vêm de alguém que já tenha feito um trabalho igual ou equivalente ao que está sendo estimado. Se esta pessoa domina o assunto, podemos chamá-la de especialista e ela será a melhor pessoa para oferecer opiniões sobre o trabalho. Portanto, delegar a tarefa a um especialista é uma ótima maneira de obter estimativas precisas, pois ele tem experiência no assunto e já sabe como fazer o trabalho.

Outra forma de chegar a estimativas é através da divisão e conquista. Para dimensionar grandes funcionalidades ou sistemas inteiros, dividi-los em partes torna mais fácil o seu dimensionamento. Cada uma delas pode ser estimada por analogia comparando-a com implementações já realizadas. Depois, é importante considerar que as partes precisarão ser integradas e isso também irá requerer esforço de implementação.

Contudo, os sistemas atuais tornaram-se tão complexos que para reunir todas as competências necessárias para criá-los, é preciso fazer uso de vários tipos de especialistas. Programadores que conheçam determinados frameworks, DBAs e designers são algumas das especialidades necessárias em muitos projetos. Porém, nem sempre há especialistas de todas as áreas à disposição do projeto, ou então, quem faz as estimativas não é quem realizará a implementação. Essas situações dificultam a obtenção de planos factíveis e realistas.

A técnica que apresentamos a seguir é o Planning Poker, um modelo de obtenção de estimativas que reúne a opinião dos principais envolvidos com o projeto. Na maioria das vezes eles não são especialistas em todas as áreas, mas serão eles os que farão a implementação, então é importante que isso seja considerado. Para chegar a estimativas realistas e razoáveis, as próprias pessoas que participarão da implementação usam a estratégia de divisão e conquista junto com o dimensionamento por analogia.

Planning Poker

Planning poker foi criado por James Grenning em 2002, mas foi de Mike Cohn a contribuição decisiva para difundir essa técnica entre os praticantes de métodos ágeis. O Planning Poker usa o conhecimento dos desenvolvedores para chegar às estimativas. As suas regras estimulam a interação entre os membros da equipe e permitem que todos expressem as suas opiniões, ao mesmo tempo em que participam do planejamento do projeto e aumentam o seu entendimento sobre o sistema que irão desenvolver. Além de obter estimativas para a implementação, essas características colaboram para que a equipe melhore o seu desempenho e atinja os objetivos do projeto.

Para começar a usar Planning poker, é preciso definir alguns conceitos e fazer alguns preparativos, conforme veremos a seguir.

Defina uma escala

Separando estimativas de tamanho e duração, isolamos a variável tempo e nos concentramos em identificar o volume de trabalho durante o processo de mensuração.

Fazer estimativas de tamanho para software não é trivial, pois ele não é palpável, portanto, para medi-lo também usaremos uma medida de pontos abstratos que representam a quantidade de esforço para produzir o software desejado. Para atribuir esses pontos é importante definir os valores de uma escala. A partir dela, cada funcionalidade receberá uma quantidade de pontos que representará a estimativa do seu tamanho.

Boas escalas podem ser criadas usando números de Fibonacci (1, 2, 3, 5, 8, 13,...), potências de 2 (1, 2, 4, 8, 16, 32,...) ou (1, 5, 10, 20, 40, 80,...), pois são seqüências exponenciais que irão absorver parte do erro associado às estimativas de funcionalidades grandes.

Opcionalmente, a equipe pode incluir o 0 (zero) na escala para estimar tarefas triviais. Depois, muitas tarefas triviais podem ser agrupadas e receber tamanho 1, afinal, mesmo sendo triviais, tomarão algum tempo para serem executadas.

A escala usada para estimar pode utilizar pontos ou dias ideais. Os pontos são medidas abstratas com tamanho definido pela equipe. Um dia ideal é o quanto seria produzido em um dia de trabalho se 100% do tempo fosse dedicado a uma atividade, sem interrupções ou distrações. Para este caso, a equipe precisa determinar a porcentagem de seu tempo que ela dedica ao projeto. Este percentual servirá para fazer a conversão entre dias ideais e dias de trabalho.

Os pontos e os dias ideais fornecem estimativas de tamanho, que são medidas sobre o volume de trabalho. Além dessas, é interessante obter estimativas de duração, como por exemplo, horas de trabalho ou dias de trabalho, que prevêem a quantidade de tempo necessária para cumprir a estimativa de tamanho. Contudo, é importante que medidas de tamanho e duração fiquem separadas, pois a velocidade da equipe pode variar durante o projeto. Isso influencia as estimativas de duração, que são as que os clientes estão mais interessados, e obriga a equipe a refazê-las. Porém, as estimativas de tamanho não mudarão. Estas serão usadas para chegar às novas previsões de duração. Quando a velocidade de desenvolvimento mudar, basta identificar a nova velocidade e derivar as novas estimativas de duração.

Para chegar às estimativas de tamanho, a estratégia da comparação é o melhor caminho, pois a mente humana consegue lidar melhor com medidas relativas do que com medidas absolutas. Isso significa que é muito mais fácil dizer, por exemplo, quantas vezes uma sala é maior do que outra do que determinar a sua área em metros quadrados.

Para estimar desenvolvimento de software por comparação, a equipe identifica uma funcionalidade pequena, cujo esforço

para implementá-la seja conhecido e a define como a sua base para comparações. Usando, por exemplo, a escala {1, 2, 4, 8, 16, 32}, atribui a essa funcionalidade o valor 2. Depois, o valor das próximas funcionalidades é obtido comparando cada uma delas com a funcionalidade base de tamanho 2. Funcionalidades com o dobro do tamanho receberão o valor 4, funcionalidades quatro vezes maior, o valor 8, e assim por diante.

Perceba que nas escalas sugeridas, os valores não são seqüenciais, o que obriga os desenvolvedores a usar apenas os valores da escala. Isso ajuda a fazer com que as estimativas transmitam seu significado real. Quanto maior é a funcionalidade, menor a precisão da estimativa. A fixação de valores evita que a estimativa seja confundida com a realidade. Se você usar números seqüenciais e estimar duas funcionalidades grandes, respectivamente, com os valores 78 e 81, você transmitirá uma falsa sensação de precisão. Agora, se você usa uma escala com os valores 10, 20, 40 e 80 e diz que as duas funcionalidades têm valor 80, fica implícito que o tamanho real pode ser um pouco mais ou um pouco menos do que 80.

Com os valores fixados, os desenvolvedores usarão naturalmente aquele que estiver mais próximo da sua percepção, forçando-os a fazer um arredondamento em suas previsões. Esse comportamento é natural e não representa um problema porque algumas vezes o arredondamento aumentará o valor da estimativa e, em outras, o diminuirá, de forma que quando um conjunto de funcionalidades for considerado, essas diferenças tendem a se equilibrar.

Materiais e preparativos

Para estimar com Planning Poker, é preciso reunir todos os envolvidos com a implementação e preparar o material. Cada um dos participantes deve ter um conjunto de cartas nas quais cada uma contém um dos valores da escala escolhida.

As funcionalidades já devem ter sido identificadas e escritas no formato de cartões de histórias de XP (em XP, cada funcionalidade é descrita com texto em um cartão de papel. Quando necessário, pode-se fazer desenhos ou incluir exemplos para facilitar a compreensão. O tamanho do cartão é variável, em torno de 10cm altura e 15cm de largura, com espaço para algumas linhas de texto) e os participantes já devem tê-las lido e tirado suas dúvidas com o cliente ou o analista que as escreveu.

Quando a história é muito complexa, ela deve ser dividida em histórias menores para que seja possível classificá-la na escala preestabelecida. Ao criar histórias pequenas, o trabalho de dimensionamento fica mais simples, pois é mais fácil estimar várias tarefas pequenas, uma a uma, do que medir apenas uma grande tarefa.

Chegando às Estimativas

Um dos desenvolvedores lê em voz alta cartão a cartão. Após a leitura de cada um, todos têm a chance de tirar novas dúvidas ou fazer observações que ajudem a equipe a compreender os requisitos. Quando todos estiverem certos de que entenderam a funcionalidade, cada participante escolhe em seu conjunto de cartas aquela com o valor de sua estimativa. Quando todos estivem prontos, as cartas escolhidas são reveladas ao grupo.

Como cada um deu a sua opinião, é comum que nem todos apresentem o mesmo valor. Quando isso acontece, os participantes que apresentaram a maior e a menor estimativas explicam porque as fizeram. Baseados nas explicações, todos repensam suas opiniões e apresentam as estimativas revisadas. Se elas ainda não forem iguais, os novos estimadores do maior e do menor valor apresentam seus argumentos para a equipe. Este ciclo se repete até que a equipe chegue a um consenso sobre o valor da estimativa.

As explicações fornecidas pelos que fizeram a maior e a menor estimativas ajudam a equipe a compreender o problema e a encontrar melhores soluções. O desenvolvedor que sugeriu o valor mais alto pode ter percebido alguma dificuldade que os demais não observaram. Ao justificar a sua estimativa, ele ajuda o resto da equipe a refletir sobre o problema sob um novo ponto de vista. Da mesma forma, o desenvolvedor que faz a estimativa mais baixa expõe seus argumentos. Ele pode ter encontrado uma solução simples e eficaz que resolve o problema. Ao explicá-la para a equipe, todos aprenderão uma forma eficiente de implementar aquela funcionalidade.

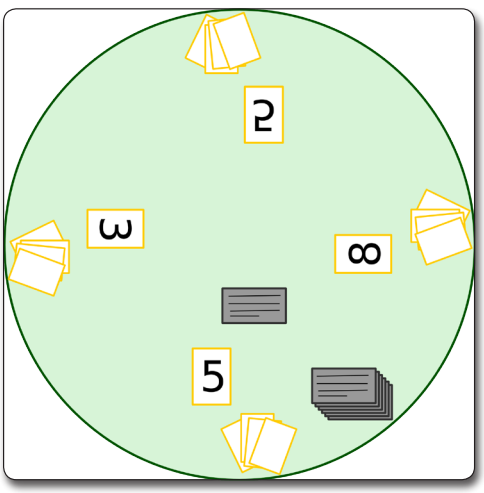


Figura 1. Exemplo com 4 participantes estimando funcionalidades escritas em cartões de história.

No exemplo da Figura 1, quatro participantes estimam funcionalidades escritas em cartões de história de XP (em cinza). Cada participante possui o seu conjunto de cartas com os valores da escala escolhida (em amarelo). Neste exemplo, usamos a escala com os valores 1, 2, 3, 5, 8, 13. Na parte de baixo da mesa está a pilha de cartões com as funcionalidades que serão dimensionadas e, um pouco acima, próximo ao centro da mesa, o cartão que está sendo estimado. Um dos participantes leu o cartão, eventuais dúvidas foram esclarecidas e todos os participantes apresentaram as suas estimativas para a funcionalidade.

Neste exemplo, dois dos participantes estimaram tamanho 5, um deles tamanho 3 e o outro tamanho 8. Como não houve consenso sobre o valor, os participantes que sugeriram 3 e 8 irão justificar suas estimativas. Em seguida, todos revêem as suas previsões e apresentam o novo valor.

A experiência com esse tipo de técnica tem mostrado que o mais comum é ter apenas um turno de explicações e que

raramente é preciso mais do que duas revisões nas estimativas para que todos os valores convirjam.

Quando a equipe concorda com um tamanho para a funcionalidade, o valor é escrito no cartão para ser usado mais tarde na montagem das releases e iterações do projeto e para calcular a velocidade da equipe e as estimativas de duração.

Se a equipe tiver à sua disposição uma ampulheta de aproximadamente dois minutos, ela poderá ser usada para controlar o tempo de argumentação. Idealmente a obtenção da estimativa de cada cartão não deve durar mais do que o um giro da ampulheta. Isso ajuda a equipe a concentrar-se em estimativas e evita que ela perca o foco com discussões sobre detalhes específicos da implementação.

Conclusões

O Planning Poker é uma técnica para obtenção de estimativas compatível com Metodologias Ágeis de desenvolvimento de software. Ele é um processo rápido para chegar a estimativas que pode ser usado em diferentes momentos do projeto. No seu início, quando ainda há pouca informação sobre o que será desenvolvido, para obter dimensões superficiais e, durante o projeto, com mais conhecimento e detalhes sobre as funcionalidades, para conseguir estimativas de maior precisão.

Com Planning Poker as estimativas são obtidas rapidamente sem despender muitos esforços. Ele considera o conhecimento e as opiniões dos desenvolvedores e permite que eles participem do planejamento do projeto enquanto aumentam seu conhecimento sobre o que irão implementar. Essas características ajudam a equipe a aprender com mais velocidade e a difundir o conhecimento entre todos os envolvidos. ●

Referências

James Grenning. Planning Poker, 2002 - http://renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf

Kent Beck. Extreme Programming Explained: Embrace Change. Addison-Wesley, 1999.

Albert L. Lederer and Jayesh Prasad. Nine management guidelines for better cost estimating. Commun. ACM, 35(2):51–59, 1992.

Dairton Bassi. Experiências com desenvolvimento Ágil. Master's thesis,

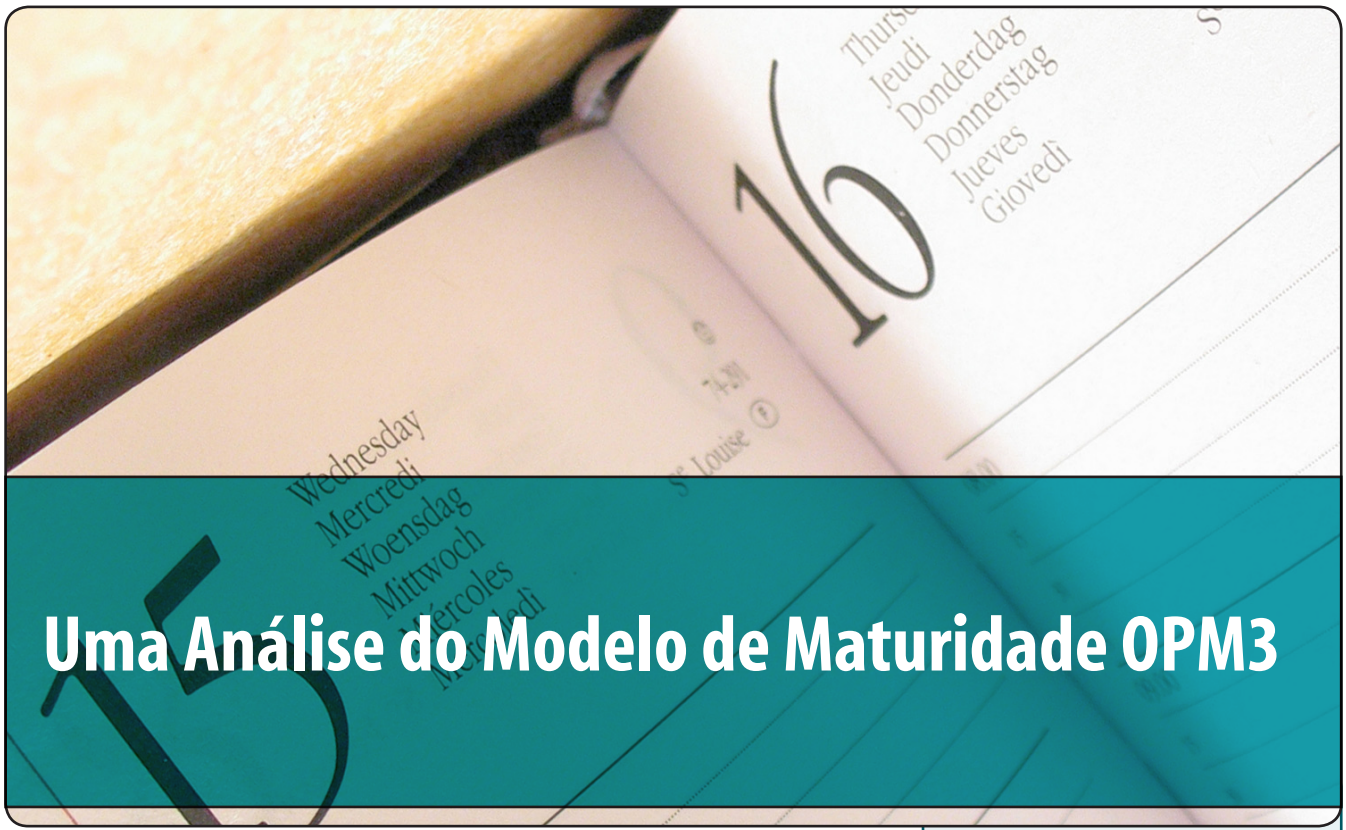
Instituto de Matemática e Estatística da Universidade de São Paulo - IME/USP, 2008.

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

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



Uma Análise do Modelo de Maturidade OPM3

Luciana Leal
lql@cin.ufpe.br
Doutoranda e Mestre em Ciência da Computação pela Universidade Federal de Pernambuco. Graduada em Ciência da Computação pela Universidade Federal da Paraíba e tecnóloga em Telemática pelo Centro Federal de Educação Tecnológica da Paraíba.

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

Hermano Perrelli
hermano@cin.ufpe.br
Atualmente é professor Adjunto e Vice-Diretor do Centro de Informática da Universidade Federal de Pernambuco. Consultor e instrutor da Qualiti Software Processes. Certificado PMP - Project Management Professional pelo PMI. PhD In Computing Science pela University of Glasgow, mestre em Informática pela Universidade Federal de Pernambuco e graduado em Engenharia Eletrônica pela Universidade Federal de Pernambuco.

Devido às modificações que as organizações vêm enfrentando no que diz respeito ao desenvolvimento de seus projetos, fica evidente a necessidade de se estabelecer metodologias que conduzam ao sucesso e que proporcionem o aumento da satisfação dos clientes. Neste sentido, é preciso estabelecer conhecimentos, ferramentas e técnicas que serão utilizadas na condução destes projetos e, para tal, a escolha de um modelo de maturidade pode ser considerado um dos primeiros passos ser dado.

Tendo como motivação o crescente interesse em qualidade, vários padrões e metodologias foram estabelecidos na década de 90 e vêm sofrendo melhorias desde então. Órgãos internacionais como a ISO e o IEC (ISO/IEC 15504), o PMI (OPM3), SEI (CMM/CMMI), e iniciativas nacionais (MPS.BR, Prado-MMGP) comprovam esta constatação e a importância que é atribuída ao aumento da qualidade, traduzida na maturidade que as organizações esperam alcançar.

Atualmente, a maturidade em gerenciamento de projetos tem sido bastante

Neste artigo veremos

Apresentação de uma visão geral e de uma análise do modelo de maturidade organizacional OPM3.

Qual a finalidade

Fornecer uma motivação para o uso de modelos de maturidade, apresentar o OPM3 e alguns dos seus pontos fortes e fracos.

Quais situações utilizam esses recursos?

O interesse em maturidade em gerenciamento tem aumentado nos últimos tempos. Vários modelos estão sendo adotados e, dentre eles o OPM3, que se destaca pela sua grande aceitação. Motivado por esta tendência, este artigo apresenta um estudo analítico sobre o OPM3 e pontos passíveis de melhoria para as organizações que estão implementando este modelo ou desejam adotá-lo.

discutida. Dentro deste contexto, destaca-se a preferência pelo o uso do OPM3, que, em 2007, foi o modelo mais utilizado no Brasil. De acordo com esta tendência e com vistas num melhor aproveitamento do OPM3, este artigo apresenta uma motivação para o uso de modelos de maturidade, uma visão geral do OPM3 e uma análise deste modelo.

Por que Utilizar um Modelo de Maturidade?

Nos últimos anos, as organizações estão procurando levar a sério o gerenciamento que fazem de seus projetos, por acreditar que esta é uma estratégia para o aumento do sucesso. Para auxiliar nesta tarefa, um grande conjunto de produtos e serviços está sendo adotado. O objetivo é a obtenção de retornos previsíveis, o aumento da qualidade dos produtos e serviços e, principalmente, a satisfação dos envolvidos no processo.

Várias razões levam as organizações à implantação de um modelo de maturidade em gerenciamento de projetos. Algumas delas são: o fato de a estratégia passar a ter um papel fundamental na sobrevivência das organizações, o aumento da complexidade dos projetos empresariais e o uso crescente de práticas de gerenciamento.

Uma preocupação constante no cenário gerencial é obter o resultado desejado na execução dos projetos, o que raramente acontece. Levando isto em consideração, modelos de maturidade são uma boa opção para descobrir oportunidades para a melhoria no gerenciamento de projetos e para identificar pontos fortes e fracos desta atividade.

O uso de modelos de maturidade faz com que as organizações verifiquem quais mudanças são necessárias e os resultados que podem servir como base para o desenvolvimento das estratégias relacionadas ao gerenciamento de projetos. Deste modo, as organizações esperam reduzir custos, aumentar a produtividade das equipes, diminuir os prazos de entrega, minimizar os riscos nos projetos, aumentar o retorno sobre o investimento (ROI), além de outros benefícios obtidos pela execução eficaz do gerenciamento de projetos.

Os Modelos Mais Usados

Os modelos de maturidade atuais possuem forte fundamentação em conceitos de gerenciamento da qualidade. Isto fica claro quando observamos que a grande maioria destes modelos é baseada em cinco níveis incrementais de maturidade ou ainda em práticas para o melhoramento contínuo dos processos de gerenciamento, como sugere o CMM (Capability Maturity Model).

Atualmente existem mais de 30 modelos com o objetivo de aumentar a maturidade das organizações. Estes modelos procuram fazer com que as organizações evoluam através de processos, práticas e ferramentas. Dentre os existentes podem ser destacados: Kerzner Project Management Maturity Model (KPMMM); Project, Programme & Portfólio Management Maturity Model (P3M3); Modelos de Maturidade em Gestão de Projetos (modelos Prado-MMGP) e Organizational Project Management Maturity Model (OPM3).

O modelo Kerzner Project Management Maturity Model (KPMMM), proposto por Harold Kerzner, é composto por cinco níveis: Linguagem Comum, Processos Comuns, Metodologia Única, Benchmarking e Melhoria Contínua [Kerzner 2001]. Possui um questionário com 183 questões de múltipla escolha que avaliam e enquadram as organizações em um dos cinco níveis de maturidade. Além de fornecer o nível de maturidade de acordo com o KPMMM, a avaliação apresenta um relatório com um plano de ação a ser executado pela organização.

O modelo Project, Programme & Portfólio Management Maturity Model (P3M3) foi lançado em 2006 pelo OGC (Office of Government Commerce) [OGC 2008]. É um modelo que trata nos domínios de portfólio, programas e projetos, e possui cinco níveis de maturidade para cada um destes domínios. Uma grande diferença entre o P3M3 os demais modelos apresentados neste artigo são as sete perspectivas que agrupam diversas características-chave e os cinco atributos comuns, presentes em todas as perspectivas.

O modelo Prado-MMGP Modelo de Maturidade em Gestão de Projetos foi desenvolvido entre 1999 e 2002 por Darci Prado [Prado 2008]. É um modelo brasileiro que possui as versões setorial e organizacional. Este modelo, assim como os anteriores, possui cinco níveis de maturidade: Inicial, Conhecido, Padronizado, Gerenciado e Otimizado. A maturidade de uma organização é avaliada dentro destes níveis e a sua relação com as seis dimensões existentes no modelo: Conhecimentos de Gerenciamento, Uso de Metodologia, Informatização, Relacionamentos Humanos, Estrutura Organizacional.

Em 2004, o PMI lançou o modelo de maturidade Organizational Project Management Maturity Model (OPM3) [PMI 2003]. O modelo é baseado em melhores práticas em gerenciamento de projetos. Estas melhores práticas são utilizadas para que as organizações possuam capacidades, que são verificadas através de resultados e indicadores de desempenho. O OPM3 atua em três domínios: projeto, programa e portfólio. Neste modelo, a maturidade de uma organização é indicada através de um valor percentual ou contínuo de maturidade, ao invés de um nível. De acordo com [PMI 2008], o OPM3 é adotado por 39% das organizações brasileiras que responderam ao estudo de benchmarking do PMI, que foi realizado no Brasil no ano de 2007.

Os benchmarkings realizados pelo PMI, nos anos de 2003 a 2007, mostram que o interesse pelo estudo e aplicação de modelos de maturidade vem aumentando. Com relação ao nível de maturidade em gerenciamento de projetos, as organizações entrevistadas até 2005 apresentaram-se entre os níveis 1 e 3 (estes níveis foram definidos pelo PMI, de acordo com características observadas na análise dos dados).

No benchmarking de 2006, as organizações afirmam conhecer os modelos de maturidade em gerenciamento de projetos OPM3 (63%), MMGP (7%) e Kerzner (2%). Esses modelos são citados como utilizados no benchmarking do ano seguinte: OPM3 (39%), MMGP (10%) e Kerzner (3%). Isto indica que o interesse em maturidade organizacional aumentou e, em decorrência disto, o uso dos modelos já é realizado.

Em 2006, 46% das organizações relataram estar nos níveis mais altos de maturidade em Gerenciamento de Projetos (3, 4 e 5) e, em 2007, esse percentual evoluiu para 51%, o que pode significar um aumento natural da maturidade das organizações devido aos investimentos realizados no ano anterior [PMI 2008].

A Tabela 1 mostra uma parte do estudo realizado pelo PMI por Setor da Economia [PMI 2008]. De acordo com os resultados apresentados,

	Adm. Pública	Engenharia	Indústria	Serviços	TI	Telecom
Não conhecem	13%	13%	29%	14%	16%	20%
Outros	25%	13%	0%	0%	5%	0%
Kerzner	0%	7%	0%	0%	1%	10%
CMM	0%	0%	14%	0%	0%	10%
CMMI	13%	0%	14%	14%	11%	20%
MMGP	13%	13%	0%	14%	18%	0%
OPM3	38%	53%	43%	57%	49%	40%

Tabela 1. Estudo de Benchmarking em Gerenciamento de Projetos Brasil

o OPM3 é o modelo mais conhecido pelas organizações brasileiras e também o mais utilizado (destaque para os setores de Engenharia, Serviços e TI). Esta tendência justifica o estudo deste modelo de forma mais detalhada, que será apresentado nas próximas seções.

O Modelo OPM3

Um modelo de maturidade de gerenciamento de projetos é uma estrutura composta por processos através dos quais uma organização se desenvolve em busca de um estado futuro que ela deseja. Este estado futuro é visto por muitas organizações como o tão desejado sucesso, que deve ser um indicativo de que estão obtendo resultados melhores.

O OPM3 é um modelo que procura oferecer essa estrutura conceitual para as organizações. Ele foi lançado pelo PMI em 2004 com o objetivo de descrever um processo no qual uma organização possa desenvolver ou atingir um estado desejado.

Foi desenvolvido sem um sistema global de níveis de maturidade, como mostra a Figura 1. Optou-se por fazer com que o modelo represente o amadurecimento da organização de forma multidimensional. Assim, o OPM3 possui uma dimensão que visualiza Melhores Práticas de acordo com suas associações com os estágios progressivos de melhoria no processo (Padronização, Medição, Controle e Melhoria). Outra dimensão envolve o progresso das Melhores Práticas associado aos domínios de gerenciamento: Projeto, Programa e Portfólio. Estas progressões representam um contínuo de maturidade e as organizações devem evoluir tomando-o como guia.

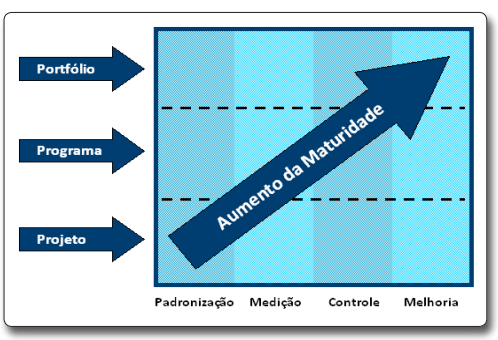


Figura 1. Dimensões de maturidade no modelo OPM3

Dentro dessas duas dimensões está a progressão das Capacidades levando a cada Melhor Prática. Cada uma das dimensões apresentadas possui associações com os cinco grupos de processo de gerenciamento de projetos (Iniciação, Planejamento, Execução, Monitoramento e Controle, e Conclusão). Estes processos podem evoluir com relação aos quatro estágios e dentro de cada domínio de gerenciamento, numa quarta dimensão de maturidade.

Segundo o PMI (2003), a vantagem de se trabalhar com múltiplas perspectivas para avaliar a maturidade é que elas fazem com que a aplicação do modelo seja flexível de acordo com as necessidades da organização. Além disso, um modelo multidimensional é capaz de fornecer maiores detalhes sobre o suporte a decisões e planos de melhoria.

Composição do Modelo

O conjunto de conhecimentos do modelo OPM3 está disposto em três diretórios: (i) Diretório de Melhores Práticas, (ii) Diretório de Capacidades e (iii) Diretório de Planejamento de Melhorias.

Os três diretórios são necessários para se utilizar o modelo por completo e cada um deles possui um único propósito. O conteúdo destes diretórios pode ser desdobrado em partes menores que são: Melhores Práticas, Capacidades, Resultados, Indicadores-chave e Dependências.

Uma Melhor Prática (Best Practice) é uma maneira otimizada para se atingir um objetivo definido. No OPM3, a maturidade de gerenciamento de projeto organizacional é descrita através da existência das Melhores Práticas. Uma Melhor Prática inclui a habilidade de entregar projetos em tempo previsto, de forma consciente e com sucesso.

A Tabela 2 mostra a disposição de algumas Melhores Práticas no diretório que as contêm. O Diretório de Melhores Práticas identifica cada uma das práticas através de um número que a precede (ID) e traz para cada uma delas um título e uma breve descrição. Além disso, apresenta o mapeamento de cada prática para o domínio de gerenciamento de projeto organizacional e para os quatro estágios de melhoramento de processo.

ID	Título	Descrição	Projeto	Programa	Portfólio	Padronização	Medição	Controle	Melhoria
1000	Estabelecer políticas de gerenciamento	A organização tem políticas descrevendo a padronização, medição, controle e melhoria contínua dos processos de gerenciamento de projetos organizacional.	x	x	x	x	x	x	x
1010	Padronização do processo de iniciação do projeto	Padrões de processo de iniciação de projeto são estabelecidos.	x			x			
1020	Padronização do processo de desenvolvimento de plano de projeto	Padrões de processo de desenvolvimento de plano de projeto são estabelecidos.	x			x			
1030	Padronização do processo de planejamento de escopo de projeto	Padrões de processo de planejamento de escopo de projeto são estabelecidos.	x			x			
1040	Padronização do processo de definição de escopo de projeto	Padrões de processo de definição de escopo de projeto são estabelecidos.	x			x			
1050	Padronização do processo de definição de atividades de projeto	Padrões de processo de definição de atividades de projeto são estabelecidos.	x			x			

Tabela 2. Exemplo do Diretório de Melhores Práticas do OPM3

Um ponto importante a ser considerado nas Melhores Práticas é o seu dinamismo, já que elas são desenvolvidas gradualmente. Isto acontece porque com o passar do tempo, novas e melhores abordagens são desenvolvidas para se atingir as metas definidas. A principal vantagem de se utilizar Melhores Práticas é que elas aumentam as chances de um objetivo ser atingido. Cada Melhor Prática é composta por uma ou mais Capacidades.

Uma Capacidade (Capability) consiste em uma competência específica que deve existir em uma organização a fim de que ela execute processos e entregue produtos e serviços de gerenciamento de projetos. A existência de uma Capacidade é demonstrada através de um ou mais Resultados (Outcomes) correspondentes. Os Resultados da aplicação de uma Capacidade podem ser tangíveis ou intangíveis.

Dentro deste contexto, um Indicador Chave de Desempenho, ou Key Performance Indicator (KPI), consiste num critério através do qual uma organização pode determinar, quantitativa e qualitativamente, se o Resultado associado à Capacidade existe e o grau para o qual ele existe. Um KPI pode ser obtido por uma medição direta ou através de uma avaliação mais especializada. KPIs podem ser tangíveis, como uma contagem de erros, ou intangíveis, como a satisfação do cliente.

Para se certificar da existência de uma Melhor Prática, a organização precisa entender o que são dependências entre Melhores Práticas e Capacidades. Dependências são representadas por um conjunto de Capacidades que resultam em uma única Melhor Prática. Ao dividir cada Melhor Prática em suas Capacidades constituintes, e mostrando as dependências entre elas, é revelada uma seqüência que permite uma avaliação detalhada e organizada, além de prover uma base para decisões posteriores relacionadas ao melhoramento.

Domínios e Estágios

Melhores Práticas e Capacidades no Padrão OPM3 estão relacionadas a dois diferentes fatores-chave: domínio e estágio.

Um domínio de abrangência se refere ao Gerenciamento de Projeto, de Programa e de Portfólio. Cada Melhor Prática (e suas Capacidade associadas) faz parte de um domínio de gerenciamento organizacional. Sobre estes domínios se desenvolvem indicações e recomendações de amadurecimento.

Um estágio está ligado à progressão na melhoria do processo.

O conceito de melhoria tornou-se amplamente adotado como um resultado do trabalho de W. Edwards Deming e Walter Shewart em 1920. De acordo com este trabalho, os estágios de melhoramento são Padronizar, Medir, Controlar e Melhorar. Essa seqüência implica em um relacionamento de pré-requisito entre os estágios, no qual o estágio mais avançado, o melhoramento contínuo, é dependente do estágio de controle, que é, por sua vez, dependente da medição, que é dependente da padronização. Cada Melhor Prática e cada Capacidade no OPM3 estão associadas a um ou mais desses estágios de melhoramento de processo.

Em adição a essas categorizações, as Capacidades no OPM3 são também mapeadas nos cinco grupos de processo de gerenciamento de projeto (Iniciação, Planejamento, Execução, Monitoramento e Controle e Fechamento) especificados no PMBOK Guide. Isso ajuda a identificar as Capacidades que permitirão às organizações implementar esses processos com sucesso, dentro de cada um dos três domínios, ou a cada estágio de melhoria de processo.

Avaliação e melhoria: O Ciclo OPM3

Para começar o uso do OPM3, é ideal que a organização interessada realize uma avaliação inicial, conhecida como levantamento de alto nível ou preliminar. O levantamento de alto nível é realizado através de 151 questões que fazem parte do modelo. A maturidade da organização é visualizada através de quatro gráficos gerados com as respostas deste questionário.

A partir deste resultado, se a organização desejar o aumento da sua maturidade, ela deve realizar os cinco passos descritos abaixo e resumidos na Figura 2:

1. Preparar-se para a avaliação: O primeiro passo é a organização se preparar para o processo de avaliar sua maturidade no gerenciamento de projeto em relação ao OPM3. Isso envolve entender os conteúdos do modelo e como usá-los.
2. Avaliar: O passo seguinte é avaliar o grau de maturidade, onde a organização deve comparar as características de seu estado atual de maturidade com as características descritas pelo OPM3.
3. Planejar melhorias: Os resultados do passo Avaliar apresentam um valor percentual, também chamado contínuo de maturidade, e formarão a base para um plano de melhoria. O OPM3 fornece um diretório que apóia a realização desta atividade.
4. Implementar as melhorias: A organização terá que implementar o plano de melhoria com o objetivo de alcançar as Capacidades necessárias para implementar uma ou mais Melhores Práticas.
5. Repetir o processo: Ao completar uma atividade de melhoria, a organização poderá refazer a auto-avaliação a fim de verificar onde ela está no contínuo da maturidade, que é o recomendado pelo OPM3. Ainda existe a opção de realizar o planejamento de outras Melhores Práticas identificadas numa avaliação anterior e seguir o ciclo novamente.

Estes passos conduzem a organização às capacidades necessárias para avançar no caminho para aumentar sua maturidade. Ainda, sugerem que o processo seja repetido, incentivando a melhoria contínua.

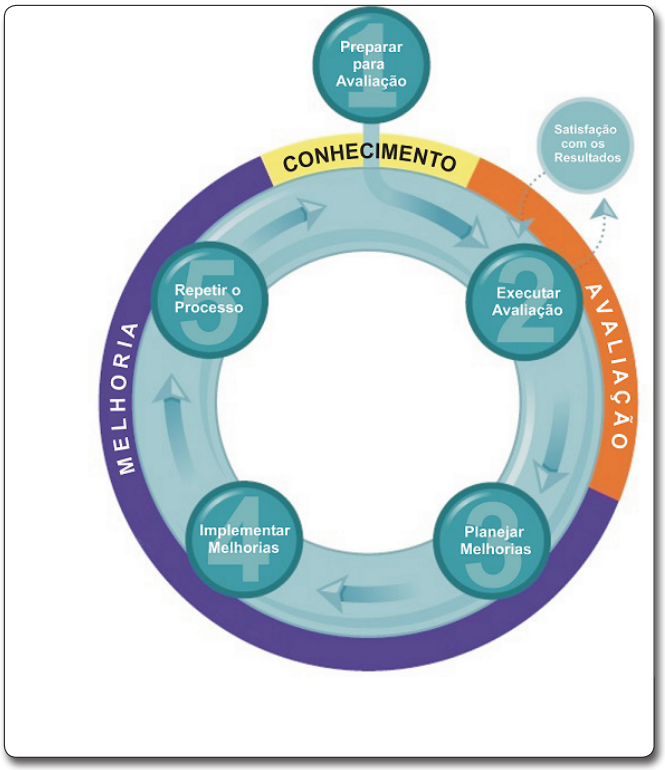


Figura 2. O ciclo OPM3

Análise do Modelo

O OPM3 é um dos modelos de maturidade mais robustos existentes. Foi desenvolvido pelo PMI através de uma pesquisa realizada em 27 modelos de qualidade e maturidade. Gerentes de projetos do mundo todo, representados por uma equipe de mais de 800 colaboradores de diversos segmentos da indústria em 35 países, desenvolveram o modelo.

Da forma como foi construído, é considerado um modelo universal, apto a ser aplicado a diversos tipos de organização e em qualquer parte do mundo. Pode fazer com que a organização promova os projetos certos, alinhados estrategicamente. Ainda, possui a vantagem de permitir a associação do sucesso da organização ao gerenciamento de projetos que ela realiza.

Contudo, apesar de ser um modelo bastante elaborado, o OPM3 apresenta alguns pontos sujeitos a melhoria. Esses pontos foram agrupados nas categorias: Tamanho, Complexidade, Redundância, Estrutura e Conteúdo, que serão descritas a seguir.

Tamanho

A extensão do modelo OPM3 e a sua estrutura complexa são grandes dificultadores do seu uso. A quantidade de melhores práticas é questionada, são ao todo 586. Seu questionário de avaliação preliminar, que também é considerando grande, possui 151 questões.

Complexidade

Por ser um modelo multidimensional, a compreensão de cada uma das dimensões e da relação entre elas torna-se difícil. É possível que o contínuo de maturidade obtido ao final de cada avaliação não consiga expressar a idéia real de amadurecimento. Um valor percentual pode não ser claro o suficiente para indicar quais as atividades que devem ser realizadas para melhoria ou os pontos fortes em relação ao gerenciamento da organização. Também pode dificultar a comunicação interna dos resultados e o estabelecimento de metas. Por não classificar a organização em um nível, faz com que os envolvidos não se sintam estimulados a alcançar maiores percentuais de maturidade.

Redundância

As questões que compõem o questionário de avaliação são burocráticas e de certa forma repetitivas. Existem questões muito parecidas que estão relacionadas diretamente com os grupos de processos do PMBOK Guide, com os três domínios que o modelo abrange e com os quatro estágios de maturidade. Estas perguntas similares, que possuem variação dentro dos grupos de processo, domínios e estágios de maturidade, poderiam ser condensadas num número menor.

Assim como o questionário de avaliação preliminar, as Melhores Práticas apresentam-se de forma repetitiva em grande parte do diretório que as contêm. Levando em conta esta constatação, uma junção de algumas destas Melhores Práticas poderia diminuir a redundância do modelo, sem perdas no conteúdo geral.

Estrutura

A estrutura geral do documento inicialmente parece confusa e, segundo [Alleman 2005], pode ser modificada. O autor sugere uma separação do OPM3 em três partes pela natureza distinta de suas atividades:

1. Organizacional: conteúdo com a finalidade de definir, direcionar e avaliar a estratégia do negócio.
2. Gerenciamento de projetos: parte que deve incentivar a compreensão de como gerenciar projetos e adotar um guia para esta atividade (o PMBOK Guide, por exemplo).
3. Modelo de maturidade: deve conter as práticas necessárias ao amadurecimento da organização.

Conteúdo

O modelo traz muito sutilmente, nos Diretórios de Melhores Práticas e Capacidades, a idéia de obtenção de perfis de acordo com as habilidades profissionais. Entretanto, não se aprofunda a respeito da capacitação dos profissionais em gerenciamento de projetos, não menciona qual é o tipo e abrangência do treinamento, quem deve ser treinado, e qual a quantidade de pessoas que deve ser treinada para suprir as necessidades de um determinado nível de maturidade.

Dentro dos diretórios, o uso de uma metodologia única de gerenciamento de projetos é pouco citado e pouco estimulado. O OPM3 não deixa explícito o quanto esta metodologia deve ser utilizada em cada estágio e como deve ser elaborada de acordo com os domínios.

Em relação ao planejamento de melhorias, o OPM3 apenas identifica possíveis ações a serem realizadas. No modelo não são encontradas orientações para o desenvolvimento do plano de ação.

Considerações

A busca de melhores resultados no gerenciamento de projetos tem feito as organizações adotarem modelos que possam descrever o seu grau maturidade. Estes modelos fornecem as diretrizes para a avaliação contínua da organização e estruturam planos estratégicos de melhoria.

Dentre os modelos mais utilizados, encontra-se o OPM3 que apresenta um grande escopo e grande flexibilidade na sua aplicação. Também possui algumas redundâncias em seu conteúdo e não cita de forma satisfatória algumas orientações para sua implementação, o que pode trazer vantagens e desvantagens no seu uso.

É um modelo recente, ainda em sua primeira versão, e possui vários pontos sujeitos a melhoria. Por este motivo, uma nova versão está sendo desenvolvida e é esperada para o ano de 2009. ●

Referências

[Alleman 2005] Alleman, G. (2005) "Some Thoughts on Project Management Organizational Maturity Models". PM FORUM, PM TODAY, 2005. Disponível: http://www.pmforum.org/viewpoints/2005/0102stopmomm.htm

[OGC 2008] OGC (2008) "Portfolio, Programme & Project Management Maturity Model (P3M3)". P3M3 Public Consutation Draft Versão 1.0, Junho.

[PMI 2003] PMI – Project Management Institute - USA (2003) "Organizational Project Management Maturity Model (OPM3)", Knowledge Foundation.

[PMI 2008] PMI – Project Management Institute – Brasil (2008) "Estudo de Benchmarking em Gerenciamento de Projetos Brasil 2007", http://ww.pmi.org.br, Agosto.

[Prado 2008] Prado, D. (2008) "Maturidade em Gerenciamento de Projetos" – Série Gerência de Projetos – Volume 7 INDG Tecs.

UML – Casos de Uso

Modelando os casos de uso como contrato entre usuários e desenvolvedores

Neste artigo veremos

O artigo trata de conceitos fundamentais de modelagem de dados, importantes para todos os Arquitetos de Dados.

Qual a finalidade

Este artigo serve de base para a construção de projetos de banco de dados.

Quais situações utilizam esses recursos?

O conhecimento das bases fundamentais de modelagem se mostram muito úteis em todas as situações do cotidiano de um Arquiteto de Dados.

Este artigo apresentará uma das principais ferramentas oferecidas pela UML — o caso de uso, demonstrando sua utilização como uma técnica para compreender e descrever requisitos, além de um contrato poderoso entre usuários e analistas, e entre analistas e programadores. Serão apresentadas todas as seções de um caso de uso, explicando as melhores práticas em cada uma.

O que é um caso de uso?

O caso de uso é utilizado para capturar os requisitos do sistema, ou seja, o que o *software* deve fazer a fim de atender as necessidades das partes interessadas pelo mesmo, partes estas que são conhecidas na área de projetos como *stakeholders*.

A UML apresenta o diagrama de casos de uso, que permite ao analista agrupar o comportamento esperado do sistema em rotinas de limites muito bem definidos, que farão a interação com os usuários. Contudo,

Ana Cristina Melo

informatica@anacristinamelo.com.br

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

além do diagrama, Ivar Jacobson, o idealizador do conceito de caso de uso (*use case*), nos oferece a especificação dos requisitos na forma textual. Esse formato, na UML, é chamado de descrições informais, que nada mais são do que os cenários de cada caso de uso. As práticas aconselhadas para a escrita desses cenários foram delineadas por metodologistas que vêm trabalhando para aperfeiçoar essas técnicas de escrita. Entre eles estão: Kurt Bittner, Alistair Cockburn, Gunnar Overgaard e Geri Schneider.

Os principais conceitos associados ao modelo de caso de uso são: **atores** e **casos de uso.**

Um **ator** (*actor*) representa um papel executado por um usuário ou por outro sistema que interaja com o sistema modelado. O ator não vai representar a pessoa e sim o papel que essa pessoa encena. Dessa forma, pode ser que a secretária Maria possa interagir com o sistema com dois ou mais papéis, o de secretária e o de vendedora.

Um ator é representado visualmente por um ícone conhecido como *stick man* (homem palito), com o nome do ator abaixo ou acima do desenho. O mais comum é modelarmos abaixo. Outra representação para um ator é mostrá-lo como uma classe estereotipada «actor». Por fim, podemos associar o ator a um ícone específico que indique o tipo do mesmo, como por exemplo a figura de um computador para representar um outro sistema que interaja com o sistema modelado. A vantagem dessa representação é separarmos a identificação visual de atores humanos dos atores não-humanos. Veja esses exemplos na **Figura 1.**

Figura 1. Representações visuais possíveis para um ator

Um **caso de uso** (*use case*) é a especificação de um conjunto de ações executadas por um sistema, produzindo um resultado observável por um ou mais atores, e que são representadas por um cenário principal e cenários alternativos.

O principal objetivo de um caso de uso é mostrar o comportamento oferecido por um sistema (ou parte dele), sem fazer referência a sua estrutura interna. Dessa forma, podemos deduzir que é interessante para o usuário saber que seu cadastro de clientes será atualizado (ou até mesmo gravado), mas não é objetivo do caso de uso dizer que o sistema está gravando a tabela TabClientes.

Sua representação é feita por meio de uma elipse, com os títulos dos casos de uso no seu interior, ou abaixo dele. Veja exemplos na **Figura 2**.

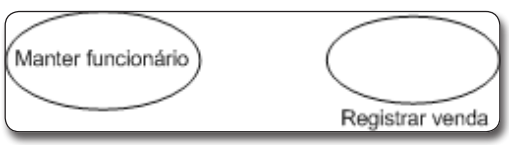


Figura 2. Representações para o caso de uso

Classificações e níveis de detalhamento

Um caso de uso pode documentar processos de negócio ou o comportamento do sistema. Quando modelamos **casos de uso de negócio** (*business use cases*), os atores são enxergados como clientes da empresa ou até mesmo seus fornecedores, pois os casos de uso irão representar o funcionamento da empresa, suas operações com o mercado. Quando modelamos os **casos de uso de sistema** (*system use cases*), os atores são os usuários que irão efetivamente utilizar o sistema. Na prática, é quem estará diante da interface interagindo com o mesmo, "botando a mão no teclado". O objetivo desses casos de uso é identificar e modelar os requisitos do sistema.

Alistair Cockburn, no seu livro *Writting Effective Use Cases* (*Escrevendo Casos de uso Eficazes*) identifica três níveis de detalhamento na escrita de casos de uso:

- **Resumido** (*Brief use case*) – consiste de duas ou quatro sentenças resumindo o caso de uso.
- **Casual** (*Casual use case*) – consiste de poucos parágrafos de texto, resumindo o caso de uso.
- **Completo** (*Fully dressed use case*) – é uma documentação formal baseada em um *template* detalhado com campos para cada seção.

Na **Tabela 1**, vemos um exemplo da descrição do tipo "resumido" e "casual". O exemplo de uma descrição completa veremos no tópico "Escrevendo casos de uso".

Formato resumido
Um cliente chega em um ponto de pagamento com itens que deseja adquirir. O caixa usa o sistema PDV para registrar cada item comprado. O sistema, a cada item registrado, apresenta um total parcial e uma linha de detalhes. O caixa entra com os dados de pagamento que são validados e, em seguida, registrados pelo sistema. O sistema atualiza o estoque. O cliente recebe um recibo de compra.
Formato casual
Cenário principal: um cliente chega em um ponto de pagamento com itens que deseja adquirir. O caixa usa o sistema PDV para registrar cada item comprado. O caixa valida e registra os dados de pagamento. O cliente recebe um recibo de compra. **Cenários alternativos:** Se o cliente pagou os itens com cartão de crédito, o sistema deve validar o cartão no Sistema da Administradora de Cartões. Se o identificador do produto não for encontrado no sistema, este notifica o caixa e sugere que entre manualmente o código do produto.

Tabela 1. Formato "resumido" e "casual" de um mesmo caso de uso "Registrar venda em PDV"

O que minha experiência de vinte anos diz? Dê sempre preferência à documentação completa. Há de se ouvir muitas vozes que não aprovam sequer a utilização do caso de uso, associando-o à burocracia. Contudo, a experiência com grandes projetos nos leva à realidade de que o caso de uso é um contrato poderoso não só entre cliente e analista, como entre analista e programador. Não há como determinar o que deve ser feito num sistema, sem

que se ofereça a um programador todas as regras e as exceções que precisam ser tratadas. Oferecer a um programador apenas um protótipo e uma descrição resumida de uma rotina, para que a partir dali ele desenvolva um produto é temeroso.

Assim, mais adiante demonstrarei como escrever essa descrição completa.

Desenhando diagramas de casos de uso

São incluídos num diagrama de casos de uso os atores que interagem com o sistema, os casos de uso que representam os requisitos, e os relacionamentos existentes.

Os casos de uso representam conjuntos bem definidos de funcionalidades que não podem trabalhar sozinhas no contexto do sistema. Portanto, esses casos de uso precisam se relacionar com atores que enviarão e receberão mensagens deste, além de se relacionarem com os outros casos de uso do modelo.

Vejamos os relacionamentos existentes entre os elementos de um modelo de casos de uso:

• Para *relacionamentos de casos de uso* entre si temos: **generalização, extensão** e **inclusão**;

• Para *relacionamentos de atores* entre si temos um único tipo: **generalização**;

• Para *relacionamentos entre atores e casos de uso* temos apenas a **associação.**

No relacionamento de *associação* (que só existe entre atores e casos de uso) teremos um ator fazendo a chamada de uma instância do caso de uso.

Para identificarmos que um ator pode chamar, em paralelo, múltiplas instâncias de um mesmo caso de uso, basta associarmos o ator ao caso de uso com uma multiplicidade maior do que 1. Isso significa, por exemplo, que se tenho um caso de uso "Registrar venda", posso querer que o usuário que irá interagir com essa rotina possa registrar mais de uma venda ao mesmo tempo, em rotinas (janelas) separadas. Da mesma forma, posso querer especificar que a rotina de "Gerar folha de pagamento" tenha que ser executada de forma exclusiva, não permitindo múltiplo processamento.

Normalmente nenhuma multiplicidade é colocada na associação. Quando isso acontece, significa que se trata do valor default da multiplicidade entre atores e casos de uso que é 0..1. Isto indica que um ator chama por vez nenhuma ou uma única instância do caso de uso.

Veja na **Figura 3** dois exemplos dessa associação. No primeiro caso, a multiplicidade da associação está informando que o ator *Vendedor* pode chamar várias instâncias em paralelo do caso de uso "Registrar pré-venda". Para entender esse exemplo, tome por base que estejamos falando de um sistema de vendas de uma loja, na qual, há terminais dentro da mesma, para que os vendedores realizem a pré-venda da cesta de produtos; depois encaminhando o cliente ao caixa, apenas para pagamento. Imagine que esse vendedor esteja realizando uma grande venda de vários itens, e descubra que um determinado produto não está cadastrado. Enquanto aguarda a regularização, outros vendedores podem querer usar o terminal, ou ele mesmo pode querer abrir outra instância de registro de venda, para atender ao próximo cliente. No segundo caso, o ator *Estoquista* deve proceder ao fechamento de balanço mensal. Essa rotina fará alguns cálculos e registrará alguns valores. Isso significa que não é aconselhável que a mesma operação esteja sendo feita duas vezes ao mesmo tempo.

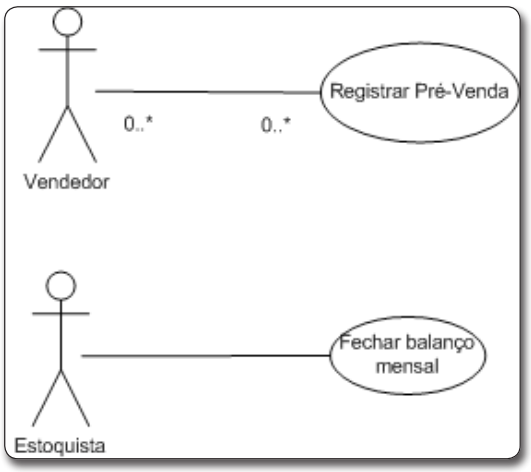


Figura 3. Exemplos de associação entre atores e casos de uso

O relacionamento de *generalização* entre atores demonstra que há um compartilhamento de metas e propósitos. Isso permite, por exemplo, que possamos ter um determinado papel que herde todas as responsabilidades de um outro, acrescentando apenas as que lhe são inerentes. Por exemplo, na **Figura 4**, temos o contexto de um sistema de vendas para um supermercado. Vemos que o ator *Supervisor* está herdando as responsabilidades do ator *Caixa*. Temos, portanto, a seguinte situação: um Supervisor tem permissão de executar todas as funções que são executadas por um Caixa. Contudo, somente ele pode fazer a sangria do caixa.

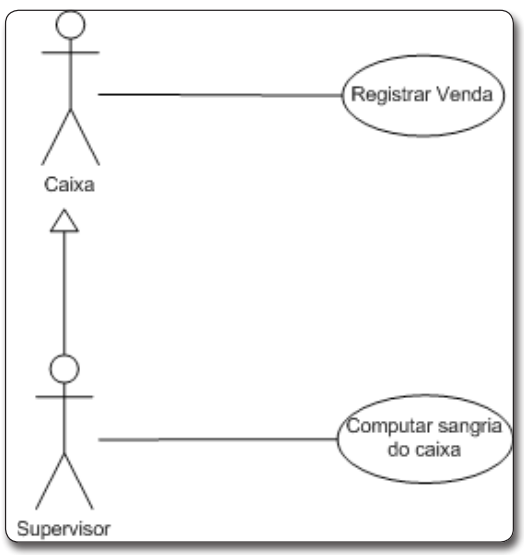


Figura 4. Exemplo de generalização entre atores

Já na *generalização* entre casos de uso, buscamos identificar o comportamento similar entre dois ou mais casos de uso, de forma a aproveitar as regras e soluções aplicadas para o primeiro caso. Isso também significa que um caso de uso filho herdará todos os relacionamentos existentes no caso de uso pai. Vejamos

o exemplo da **Figura 5**, no qual temos um caso de uso "Manter funcionário". Imagine que tenhamos um caso especial dessa funcionalidade que seja o "Manter professor". Neste cadastramento, imaginamos precisar de tudo que se tem em "Manter funcionário", porém acrescentando controles e informações que são inerentes apenas ao professor. Assim, ao criarmos uma generalização entre esses casos de uso, obtemos a reutilização de cenários, com toda a sua complexidade e tratamento, mantendo no caso de uso mais específico, somente o que lhe é inerente.

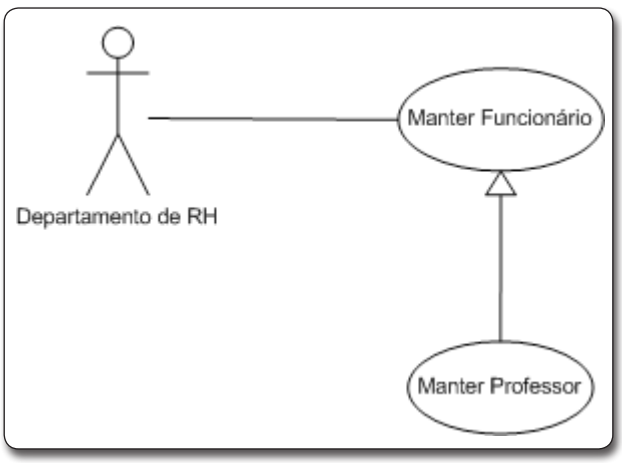


Figura 5. Exemplo de generalização entre casos de uso

Entre casos de uso podemos ter relacionamentos de dependência, estereotipados como relacionamento de inclusão (*include*) ou extensão (*extends*).

O relacionamento *extends* define um conjunto de comportamentos que incrementará a execução do caso de uso base. Contudo, o caso de uso base é definido independentemente do caso de uso de extensão. Assim, o caso de uso de extensão define um conjunto de comportamentos que incrementará a execução do caso de uso base de acordo com condições específicas. A extensão ocorre em um ou mais pontos de extensão específicos no caso de uso base.

O mesmo caso de uso de extensão pode estender mais do que um caso de uso.

De acordo com a documentação da UML, esse tipo de relacionamento é aconselhável a ser usado quando há algum comportamento adicional que deve ser acrescentado (possivelmente de forma condicional) ao comportamento de outro caso de uso. Não é raro eu ouvir o questionamento (quase afirmativo) de que o relacionamento de extensão é usado apenas para a modelagem de parte de um caso de uso que se considere como opcional. Não é verdade. É apenas uma das possibilidades. O relacionamento de extensão também pode ser utilizado para a modelagem de um subfluxo separado, que é executado somente sob determinadas condições. Isso significa que ele é condicional, não opcional.

Visualmente esse relacionamento é representado por uma seta com linha tracejada, cuja origem sai do caso de uso de extensão em direção ao caso de uso base. Junto à linha é colocado o estereótipo «extend». É possível ainda representar visualmente a condição para acionamento do relacionamento, as referências aos pontos de extensão. Essa representação é feita com uma nota ligada ao relacionamento *extend*. O exemplo da **Figura 6** mostra um caso de uso "Registrar venda" que possui um caso de uso de extensão "Lançar autorização manual de cartão de crédito". Esse caso de uso é descrito no cenário, no ponto de extensão "Validação do cartão de crédito", e só é acionado se houver perda de conexão com o sistema de autorização da administradora do cartão de crédito.

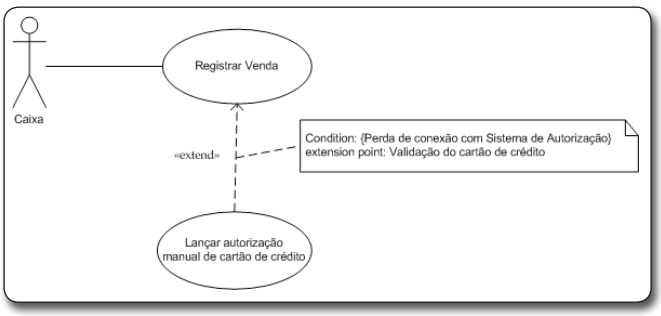


Figura 6. Exemplo de relacionamento de extensão

O relacionamento *include* define que um caso de uso contém o comportamento definido em outro caso de uso.

O caso de uso incluído não é opcional, e é sempre requerido para que o caso de uso base execute corretamente. Esse tipo de relacionamento é utilizado quando há partes comuns de comportamento em dois ou mais casos de uso. Essa parte comum é então extraída para um caso de uso separado, para ser incluído por todos os casos de uso base que tenham essa parte em comum.

A execução de um caso de uso de inclusão é similar à chamada de sub-rotinas. Pode ser utilizado, portanto, para separar em sub-funções um caso de uso muito complexo e longo, que passa a controlar mais do que seu objetivo definido.

Visualmente esse relacionamento é representado por uma seta com linha tracejada, cuja origem sai do caso de uso base em direção ao caso de uso a ser incluído. Junto à linha é colocado o estereótipo «include».

O exemplo da **Figura 7** mostra dois casos de uso ("Emitir Histórico" e "Registrar nota"), no contexto de um sistema acadêmico. É de se esperar que em ambos os casos, o usuário precise informar a matrícula do aluno. Mas também é de se esperar que o aluno ao chegar à Secretaria e pedir seu histórico, nunca saiba de cor sua matrícula; e que o professor ao lançar a nota do aluno, com base na prova que está em mãos, se depare com o campo matrícula em branco (eu que sou professora passo por isso, com freqüência). Sendo assim, o correto é oferecer uma rotina que possibilite a pesquisa desse aluno por outros meios, como nome e cpf. Se essa rotina é reutilizada pelos casos de uso citados, o aconselhável é fazer sua modelagem como um caso de uso separado, que se relacionará com os outros por meio da inclusão.

Como chegar à modelagem de um caso de uso

A modelagem de um caso de uso partirá sempre do levantamento de requisitos com o usuário. A partir destes requisitos, será possível determinar o contexto do sistema e os atores que

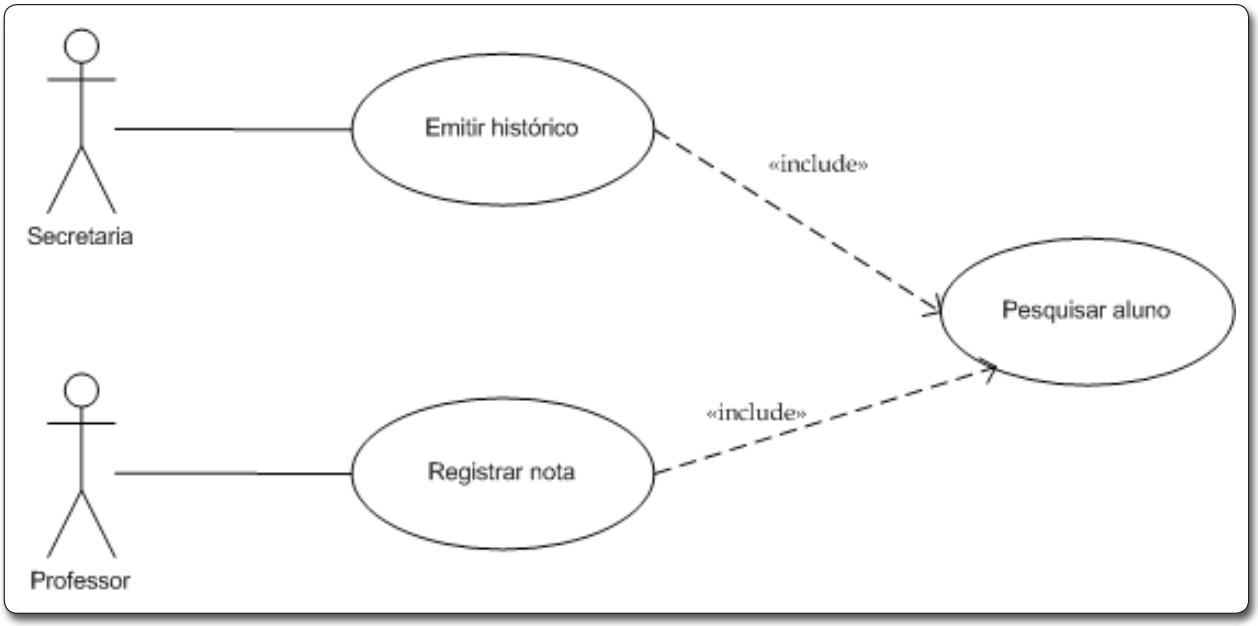


Figura 7. Exemplo de relacionamento de inclusão

irão interagir com o mesmo.

Para cada ator identificado, buscam-se suas responsabilidades e o que cada um espera de comportamento do sistema. Esses comportamentos são nomeados como casos de uso.

A partir de uma primeira versão, é possível refinar esse modelo, estabelecendo os relacionamentos de generalização, inclusão e extensão.

Escrevendo casos de uso

A descrição de um caso de uso na realidade é uma coleção de cenários de sucesso (cenário principal) e fracasso (cenários alternativos) que vão descrever os passos de um ator na utilização do sistema, e a reação desse sistema a cada um desses passos, até que se atinja o objetivo principal do caso de uso.

Ao se escrever os cenários de um caso de uso, busca-se o conceito de *caixa preta* existente no desenvolvimento de sistemas. Isto é, busca-se dizer *o que* o sistema deve fazer em detrimento de *como* ele o fará. Focar no "o quê" significa focar nos requisitos funcionais, em seu comportamento para atender esses requisitos. Descrever o "como", significa estabelecer uma modelagem na fase de projeto. Ora, se o caso de uso vem a ser um contrato entre desenvolvedores e clientes, não é possível se estabelecer nessa fase inicial detalhes de projeto.

Por mais de uma vez encontrei modelos de casos de uso com citações à componentes da interface, bem como à características da arquitetura utilizada. Algo como dizer: "o usuário clica no link X". Ao modelarmos os requisitos de um sistema ainda não sabemos qual será a solução de projeto que será adotada. Sendo assim, o correto é escrevermos:

O sistema registra a venda

Em vez de:

O sistema grava a venda na tabela TabVendas.

ou

O sistema gera uma instrução INSERT INTO TabVendas ..."

O texto de um caso de uso é dividido em várias seções, sendo algumas opcionais. Das mais utilizadas, cito:

- **Descrição**
- **Lista de atores**
- **Pré-condições**
- **Cenário principal**
- **Cenários alternativos**
- **Pós-condições**
- **Regras de negócio**

Vejamos um pouco sobre cada seção.

Descrição. Normalmente é escrita com uma ou duas frases que identificam o objetivo do caso de uso, ou seja, um resumo do mesmo. Não são citados detalhes, pois estes entrarão no corpo do caso de uso.

Lista de atores. Neste item relaciona-se o nome de todos os atores que interagem com o caso de uso.

Pré-condições. Indica tudo que deve ser verdadeiro para que o caso de uso tenha início. Quem garantirá essa pré-condição é o sistema, que não permitirá que o caso de uso se inicie sem que a mesma seja verdadeira. Quando estamos relacionando um caso de uso a outro, muitas vezes é necessário estabelecer o que de verdadeiro precisa ser cumprido pelo caso de uso chamador para que a execução do caso de uso chamado seja feita de forma correta. Por exemplo: um caso de uso que valide o CPF, tem como pré-condição receber esse CPF.

É comum encontrarmos como exemplos de pré-condições algo como: "O ator é identificado e validado" ou ainda "O usuário fez login no sistema". Contudo, é uma preferência minha omitir esse tipo de pré-condição. E a justificativa é lógica. Ora, se um sistema pressupõe que para se ter acesso ao mesmo é preciso logar-se, nada mais lógico de que nenhuma rotina (caso de uso) possa ser executada se o usuário não estiver logado. A não ser que estejamos tratando de um sistema que mescle rotinas com *login* e sem *login*. Nesse caso, aprovo a colocação

desse tipo de pré-condição.

Assim, opto por trabalhar nessa seção apenas características relevantes e que devem ser indispensavelmente analisadas antes da execução do caso de uso. Por exemplo: suponha um caso de uso "Lançar avaliação" de um Sistema Acadêmico. Pressupondo que haja um período geral para que todos os professores façam os lançamentos das notas de seus alunos, uma pré-condição poderia ser: *A data atual deve estar contida no período disponível para lançamento de avaliação.*

Cenário principal. Descreve uma seqüência de ações que serão executadas considerando que nada de errado ocorrerá durante a execução. Isso significa que se traça uma espinha dorsal, na qual estão elencados todos os itens necessários para se alcançar o objetivo do caso de uso. Feito isso, num segundo momento, voltamos a cada item e questionamos que tipo de situação de erro pode ocorrer, ou que tipo de alternativa podemos ter àquela solução. Dessas respostas nascem os cenários alternativos.

A forma mais recomendada de se escrever cada sentença é usar a combinação:

sujeito + predicado + objeto

Ou seja, indicar quem faz a ação, a ação em si, e se necessário, uma informação complementar sobre essa ação.

Por exemplo:

O atendente informa telefone.
O atendente informa sexo, selecionado entre as opções: feminino ou masculino.
O sistema pesquisa o aluno e exibe o nome e o telefone.

É imprescindível que o "sujeito" da ação seja citado, no caso o ator ou o sistema, pois é necessário indicar quem está controlando essa ação. A simplicidade e a clareza de um caso de uso estão diretamente relacionadas à forma como ele é escrito.

Para que não haja repetição do nome do ator em todo o caso de uso, o mesmo pode ser representado pelo termo "usuário" (exemplo: *o usuário informa telefone*). Contudo, se algum passo só puder ser executado por um dos atores, isso deve vir explícito no caso de uso.

Algumas *boas práticas* que devem ser adotadas ao se escrever os cenários principal e alternativos:

• Escreva sempre "o que" acontece e não "como" acontece;

• Escreva um texto com terminologia mais próxima do usuário. Não use termos de sistema.

Exemplo: para o usuário completamente leigo é mais fácil entender "O sistema atualiza o cadastro de vendas" do que "O sistema grava a venda". Gravar é um verbo próximo ao mundo computacional.

• Escreva focando a intenção dos atores, não os seus movimentos. Descrever os movimentos do usuário operando a interface do sistema é um dos erros mais comuns e graves na escrita dos casos de uso. Exemplo:

Em vez de escrever *O usuário clica no botão "Incluir"*;
escreva *O usuário seleciona a opção "Incluir"*

• Você pode numerar ou não os itens de um cenário principal. Basta que você tenha, por exemplo, um editor de textos que lhe facilite esse trabalho. Lembre-se que a numeração é útil quando de referências dentro do próprio cenário principal ou nos cenários alternativos.

Cenários alternativos. Relacionam tudo o que se espera do sistema quando da ocorrência de *exceções* nos itens do cenário principal, ou ainda, a relação de *alternativas* que podem ser criadas para o referido cenário.

Nessa seção teremos vários pequenos cenários, que abrangem tanto exceções quanto alternativas. Para cada um, cabe usar um formato de melhor entendimento:

Título do cenário

Número-cenário-principal.letra-índice. Texto indicando o problema e a solução apresentada.

Exemplo: Vamos supor que temos o seguinte item no cenário principal

3. O usuário informa o valor de saque.

O mundo perfeito espera que o usuário saiba quanto quer sacar, digite o valor de forma correta, que haja saldo na conta, e que haja notas na máquina que atendam ao valor pedido. Mas como nem tudo é perfeito, precisamos prever todos os casos de exceção ou alternativas para a situação acima.

Valor de saque excede limite

3.a. Se o valor de saque exceder o limite máximo de saque diário, o sistema deve notificar o usuário e solicitar novo valor.

Saldo insuficiente

3.b. Se não houver saldo na conta do cliente suficiente para cobrir o valor de saque, o sistema deve notificar o usuário e encerrar o caso de uso.

Valores de sugestão

3.c. O sistema deve sugerir valores para o usuário selecionar, que contemplem quantias de maior freqüência de saque, e estejam de acordo com as notas disponíveis na máquina.

Repare que cada cenário alternativo recebeu um título. É necessário? Respondo essa pergunta com outra. Numa seção na qual constem mais de 20 cenários alternativos, em qual é mais fácil de localizar um desses cenários para manutenção: em uma seção que apresenta apenas os textos de cada cenário, ou numa seção que apresenta um título e em seguida o texto?

O texto deve deixar claro o evento que motivou o cenário, e o tratamento dado ao mesmo. Veja que na primeira exceção a solução adotada fornece ao usuário uma nova chance. Já na segunda exceção, por uma questão de segurança, a rotina é abortada. Essas informações são importantes para que tanto o cliente dê o seu aval para o tratamento adotado, quanto para o programador que não terá que inventar a sua forma de sair dessas situações.

Já no terceiro cenário, o que temos não é uma exceção e sim uma alternativa. O usuário poderá informar o valor a sacar ou selecionar um dos valores que o sistema irá automaticamente lhe sugerir. Veja que não há informações no caso de uso de como esses valores serão exibidos. Por exemplo: se serão exibidos em botões. Nesse momento, o importante é demonstrar os requisitos. Se o cliente desejar determinar a forma como esses valores devem ser apresentados, essa informação será escrita numa seção específica, explicada mais à frente.

Pós-condições. Indica tudo que deve ser verdadeiro para que o caso de uso seja considerado completo com sucesso.

Da mesma forma como citei na seção de pré-condições, a prática me habilita a questionar algumas indicações de uso nesse caso. É comum encontrarmos essa seção sendo chamada de "Garantias de sucesso" e seu conteúdo sendo descrito com algo como "a venda é salva". Requisitando o objetivo da Engenharia de Software, o que se espera de um sistema é sempre a sua qualidade. Desta forma, se o caso de uso apresenta em seu cenário principal uma última frase como "O sistema registra a venda", o que se espera de um sistema é nada mais do que o sucesso nessa operação. Considero como pós-condições tudo aquilo que esteja implícito, mas que seja de suma importância na conclusão do caso de uso. Por exemplo, um caso de uso que cuide de registrar venda, uma pós-condição pode ser "o estoque será atualizado".

Regras de negócio. Indica as regras de negócio que estejam diretamente relacionadas ao conteúdo do caso de uso.

Como boa prática, a fim de evitar repetição de regras, e sua difícil manutenção, evito colocar diretamente as regras no caso de uso. Em vez disso, preparo um documento com todas essas regras, e acrescento na seção do caso de uso apenas seu título, com um link para o texto da referida regra de negócio.

Há uma outra seção que particularmente acrescento no final do documento de caso de uso, de forma a deixá-lo completo.

Requisitos de Implementação. Indica quaisquer requisitos não-funcionais, imprescindíveis à satisfação do cliente, que devem ser observados quando da implementação do caso de uso e/ou na fase de projeto. Você pode encontrar referências a essa seção com os títulos "Lista de variações tecnológicas" ou "Requisitos especiais".

Exemplo: Imagine que você esteja modelando um sistema de senhas para atendimento ao cliente. O "dono do sistema" por experiência de outros *softwares*, determina que o painel de próxima senha exiba o número em preto com fundo branco. Esse é um requisito que precisa ser documentado, mas não no cenário principal. Sendo assim, você pode incluí-lo sem culpa na seção de "Requisitos de Implementação". O prototipador agradecerá.

Outros exemplos de requisitos que entrariam nessa seção:

- A interface do usuário deve ser do tipo tela sensível ao toque (*touchscreen*);
- A resposta da pesquisa de contribuintes deve ocorrer em no máximo 10 segundos;
- As operações de inclusão, alteração e exclusão devem ser auditadas.

Para se escrever os cenários de um caso de uso, adota-se também no mercado o formato de duas colunas, no qual se separa as ações do(s) ator(es) da ação de resposta do sistema. Particularmente, não aprecio esse formato, por ser de difícil manutenção.

Seções podem ser suprimidas, outras incluídas, criando-se um padrão próprio para cada empresa. Os nomes das seções podem ser escritos de forma diferente (cenário principal x fluxo principal x cenário perfeito etc). O importante é não se perder o foco da simplicidade, contudo completude que um caso de uso deve expressar. Sua importância não se limita apenas a modelar os requisitos, mas a ser uma base para todo o projeto de desenvolvimento de um sistema. Não é à toa que a partir dele podemos gerar: modelo de classes, diagrama de seqüências e casos de teste.

Para finalizar, veja na **Listagem 1** o exemplo de um cenário de caso de uso no contexto de um sistema de controle de estacionamento.

Listagem 1. Exemplo de um caso de uso

```
UC Registrar Entrada de veículo

Descrição: Este caso de uso tem por objetivo registrar os dados
do veículo que esteja entrando no estacionamento.

Atores: Atendente

Pré-condição: não se aplica

Cenário principal:
   1.O sistema prepara uma lista de modelos de veículo, exibindo
para cada um: modelo e fabricante.
   2.O usuário informa:
      2.1.placa do carro
      2.2.modelo, selecionado de uma lista pré-existente
      2.3.cor
   3.O sistema obtém a data e hora atual, registrando como
   início do estacionamento.
   4.O sistema atualiza o cadastro do estacionamento.
      4.1.O sistema imprime o ticket de estacionamento. Include
UC Imprimir Ticket

Cenários alternativos:

   Veículo já cadastrado
   2.a. Se a placa do veículo já estiver cadastrada, o sistema
   deve carregar
   automaticamente o modelo e a cor do veículo.

   Novo modelo
   2.b. Se não existir o modelo do veículo que estiver
   estacionando, o usuário poderá
   cadastrar diretamente o modelo, sem prejuízo do cadastro.
Include UC Manter modelo de veículo.

   Segunda via do ticket
   4.a. Caso o ticket do estacionamento não seja impresso, o
   usuário poderá solicitar segunda via do mesmo.
```

Referências

Alistair Cockburn Site
http://alistair.cockburn.us
Artigo "Documenting a use case", escrito por Scott W. Ambler, acessado em http://www.ibm.com/developerworks/webservices/library/ws-tip-docusecase.html

UML Site
www.uml.org
COCKBURN, Alistair. Escrevendo casos de uso eficazes. Porto Alegre: Bookman, 2005.

BOOCH, Grady et al. UML: guia do usuário. Rio de Janeiro: Campus, 2000.

LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e ao Processo Unificado. 2ª edição. Porto Alegre: Bookman: 2004.

Cursos Online

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

Curso de .net em destaque

Aplicação completa de orçamento Doméstico no Visual Studio 2005

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

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

Curso de Java em destaque

Introdução ao desenvolvimento para celulares com J2ME

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

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

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

A sua melhor opção de aprendizagem!

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

DevMedia GROUP **Mais Informações: www.devmedia.com.br/central - Tel.: 21 2220-5375 / 2220-5435**

MDA – Arquitetura Orientada por Modelos

Um Exemplo Prático

Italo Fernando Comini Souza
italocomini@msn.com
Bacharelado em Sistemas de Informação pelo Centro de Ensino Superior de Juiz de Fora. Atualmente é Analista de Sistemas da SOLUCIONAR Informática & Sistemas Ltda.

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

A MDA (Model Driven Architecture - Arquitetura Orientado por Modelos) propõe separar a lógica de negócio, plataforma e tecnologia, de tal forma que as modificações na plataforma não afetem as aplicações existentes e a lógica de negócio evolua independente da tecnologia. Com isso, os

Neste artigo veremos

Este artigo trata do uso do framework MDA (Model Driven Architecture) em um estudo de caso prático no qual são aplicados seus principais princípios, tais como a separação das regras de negócio da implementação.

Qual a finalidade

Tais modelos irão ajudar no desenvolvimento de sistemas complexos devido à separação das regras de negócio da implementação. Na MDA os modelos são elementos ativos e participantes de todo ciclo de desenvolvimento e não apenas um meio de comunicação entre os participantes do projeto.

Quais situações utilizam esses recursos?

No processo de desenvolvimento de software, os modelos podem ser usados na geração de programas, scripts de banco, documentação de usuário, configurações e quaisquer outros elementos que façam parte do processo de desenvolvimento.

benefícios são evidentes, tais como a redução de custo do ciclo de vida do projeto, a redução do tempo de desenvolvimento para novas aplicações, o aumento da qualidade da aplicação e o aumento do retorno no investimento em tecnologias.

Para obter tais benefícios, a MDA faz uso de modelos, os quais podem ser independentes ou específicos de uma plataforma. Na MDA, o Modelo Independente de Plataforma (PIM - Platform Independent Model) será submetido a processos de transformações a fim de se obter como destino um ou mais Modelos Específicos de Plataforma (PSM - Platform Specific Model). Ao final do processo, os PSM poderão ser transformados em código fonte.

Arquitetura Orientada por Modelos

A Arquitetura Orientada por Modelos é um framework definido pelo Object Management Group (OMG), que é responsável por diversos padrões de computação para especificação de sistemas e interoperabilidade, sendo alguns deles influentes e conhecidos, tais como Unified Modeling Language (UML), Meta-Object Facility (MOF), XML Metadata Interchange (XMI) e Common Warehouse Meta-model (CWM).

O ciclo de desenvolvimento através da MDA não é muito diferente do ciclo de desenvolvimento tradicional. A grande diferença está nos modelos gerados, sendo Modelo Independente de Plataforma (PIM) e Modelo Específico de Plataforma (PSM) durante o processo de desenvolvimento.

Modelo Independente de Plataforma

É um modelo com um alto nível de abstração com foco nas regras de negócio do sistema. Através do PIM é possível ter uma visão do software de forma independente de plataforma específica e, a partir da transformação do PIM, gerar Modelos Específicos de Plataforma para diferentes plataformas, conforme ilustrado na Figura 1.

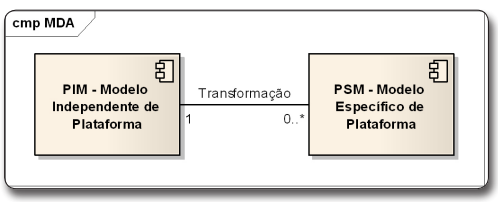


Figura 1. Transformação de um PIM em diversos PSM

Modelo Específico de Plataforma

É um modelo de um PIM com características específicas de uma determinada plataforma. Através do PSM é possível ter uma visão do sistema com detalhes específicos de um tipo particular de plataforma. Ele recebe detalhes (ver Figura 2) da construção de um sistema baseado na tecnologia selecionada. Este é o modelo que está mais próximo da linguagem de programação, com nível de abstração mais baixo. Com o PSM, através do processo de transformação, é possível gerar o código fonte.

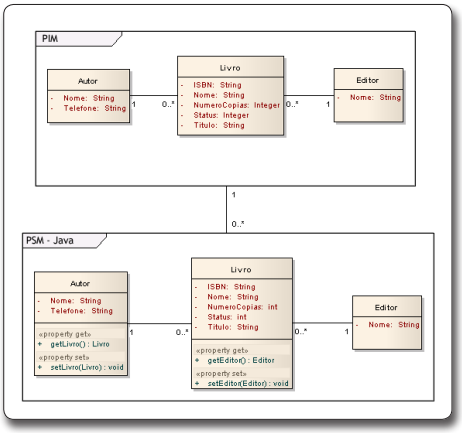


Figura 2. Diferença entre PIM e PSM

Estudo de Caso

Neste artigo será apresentado um estudo de caso demonstrando de forma prática o uso do framework MDA na construção de uma pequena aplicação. O processo será apoiado pela ferramenta Enterprise Architect (EA), onde são criados os modelos PIM e submetidos ao processo de transformação para obter o PSM e o código fonte.

A ferramenta Enterprise Architect é desenvolvida pela Sparx com suporte a construção de modelos, sendo possível modelar e transformar modelos para diversas plataformas. Para este estudo de caso, foi utilizada a versão 7.1.833 da ferramenta que pode ser obtido uma versão de avaliação no site do fabricante (ver seção Links).

O estudo de caso será dividido em três partes. Na primeira será demonstrada como configurar o ambiente para a modelagem do sistema proposto, um fragmento de um Sistema de Controle Acadêmico (SCA). A segunda será a construção do PIM, configurar a transformação e gerar o PSM. Na última parte será demonstrada como gerar código fonte através do PSM.

Configuração do Ambiente

Para iniciar o estudo de caso, com o Enterprise Architect aberto, selecione New Project no menu File. Será exibida uma janela solicitando o destino de armazenamento e o nome do arquivo do projeto. Após salvar o projeto, será exibida a janela onde podem ser selecionados os modelos que serão utilizados, conforme a Figura 3.

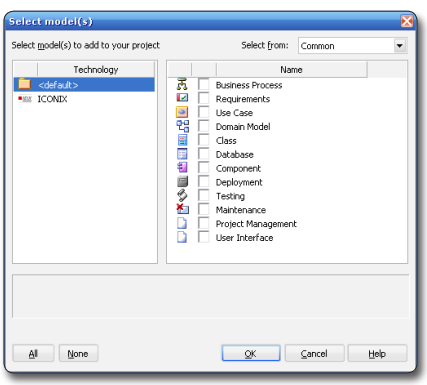


Figura 3. Seleção de Modelos

Para criar o repositório para o diagrama de classes, será necessário selecionar no "Project Browser" o nó principal do projeto e clicar em New Package (área em destaque na Figura 4).

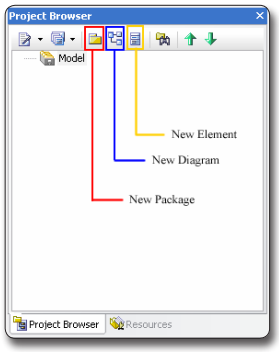


Figura 4. Project Browser

A janela exibida na Figura 5 é responsável por indicar o nome do repositório e o tipo de diagrama a ser criado.

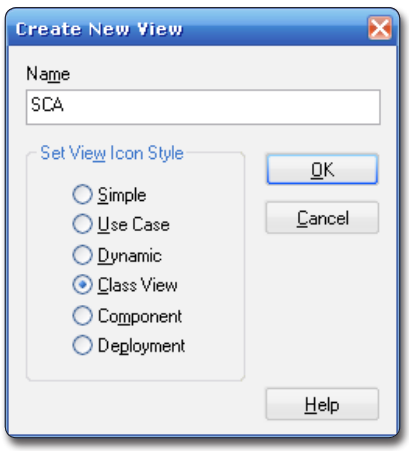


Figura 5. Criação de um novo Package

Criado o repositório, será necessário adicionar um diagrama de classes através do botão New Diagram (Figura 4) no Project Browser. O diagrama de classes está localizado dentro de UML Structural (Figura 6).

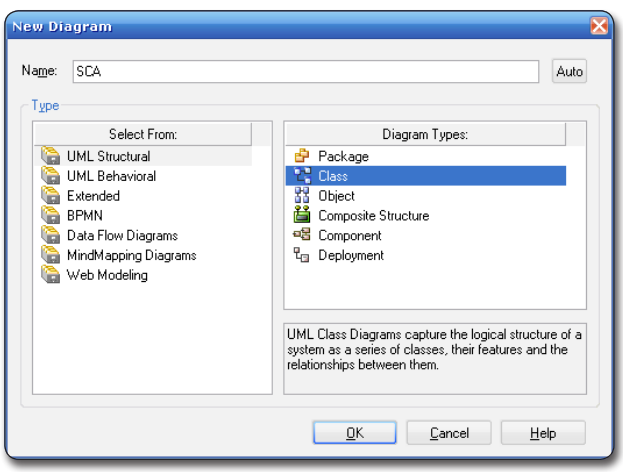


Figura 6. Criação de diagrama de classes

Após a criação do diagrama, será necessário adicionar dois Packages no mesmo, um chamado PIM e outro PSM, disponível na Toolbox. Na janela de propriedades que será exibida para configurar o Package, é necessário marcar a propriedade Language como <none>, conforme ilustrado na Figura 7, para que o mesmo seja independente de plataforma.

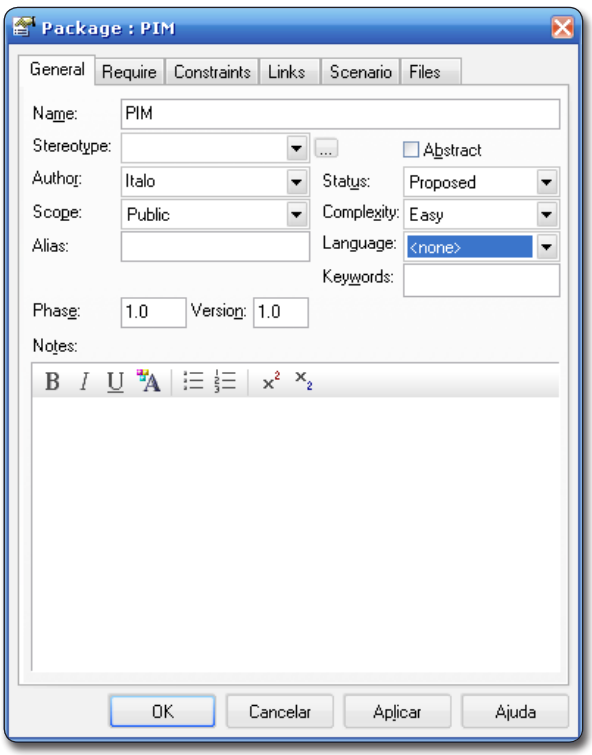


Figura 7. Propriedades do Package

Após estas configurações, o Project Browser deverá ficar conforme a Figura 8, de forma que possa ser modelado o PIM e, após a modelagem, gerar os modelos PSM.

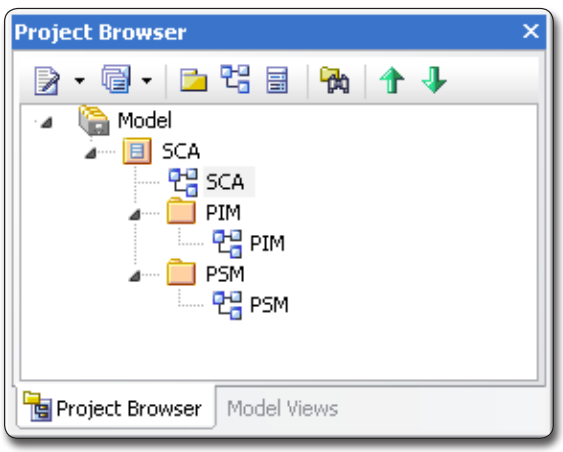


Figura 8. Project Browser com pacotes PIM e PSM

Modelagem do PIM e Transformação para PSM

Nesta seção será visto como modelar o PIM através do diagrama de classes, além de efetuar a transformação do mesmo

para um PSM. Esta é a principal parte deste estudo de caso, onde serão abordadas as principais características da MDA, gerando a partir de um Modelo Independente de Plataforma um Modelo Específico de Plataforma.

O primeiro passo dessa seção será modelar o PIM no diagrama de classes que está contido dentro do pacote (Package) PIM. O modelo deverá ser construído conforme a Figura 9.

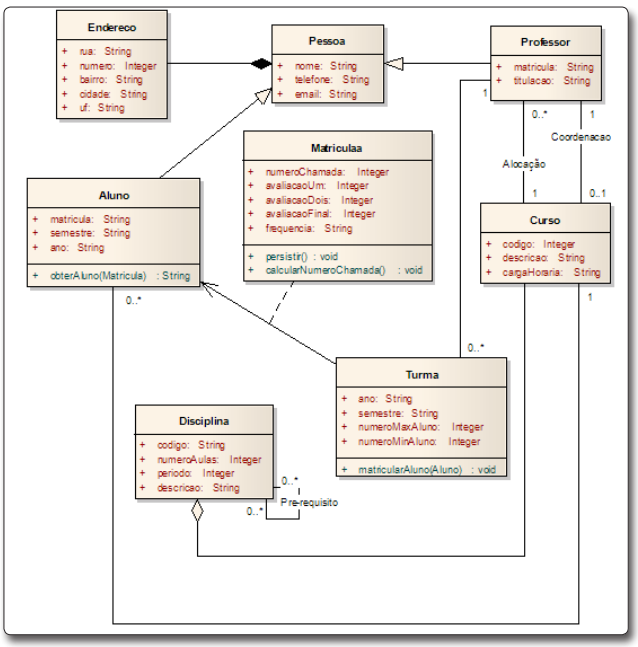


Figura 9. PIM do Sistema de Controle Acadêmico

No momento da modelagem do diagrama de classes, é importante selecionar a opção <none>, para que a classe a ser construída não esteja relacionada a nenhuma plataforma destino (Figura 10).

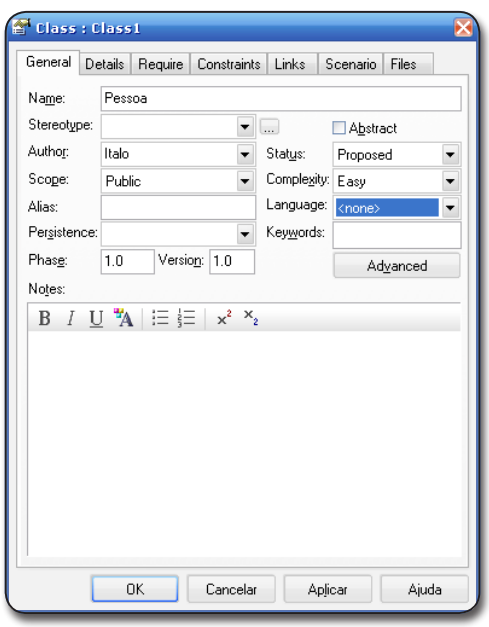


Figura 10. Propriedades de classe

Um ponto muito importante na construção do PIM é utilizar tipos de dados simples, tais como: Integer, String e Boolean. Estes tipos simples de dados serão mapeados e convertidos no processo de transformação para a plataforma destino através da macro CONVERT_TYPE.

Para automatizar tipos de dados mais complexos como Date (Data), é necessária a customização do template de transformação. O mesmo é necessário para atributos com multiplicidade superior a 1 (um) ou coleções, bastando configurar as classes responsáveis pelas coleções conforme ilustrado na Figura 11. A janela de configuração pode ser acessada através de Tools / Options / Source Code Engineering, selecionando a plataforma desejada, clicando no botão Collection Classes.

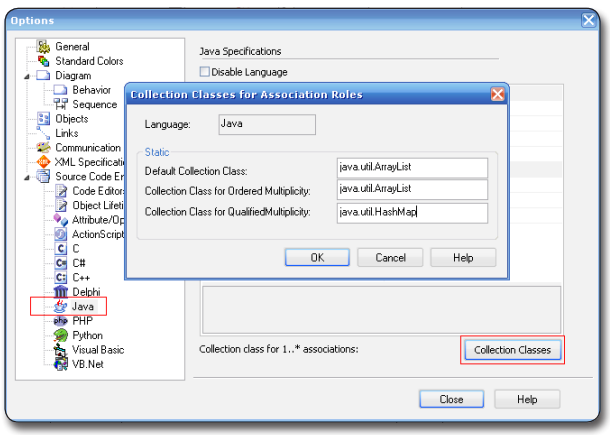


Figura 11. Configuração de coleções em Java

No processo de transformação, para linguagens como C# e Java, os atributos com escopo público (public) serão convertidos para privado (private) e serão criados métodos get e set para acessar os atributos.

Ainda em Options (Figura 12), deve-se selecionar a opção Capitalized Attribute Name for Properties e configurar o campo Remove prefixes when generating Get/Set properties. Estas configurações são importantes para não ocorrer conflitos e gerar prefixos automáticos no código fonte.

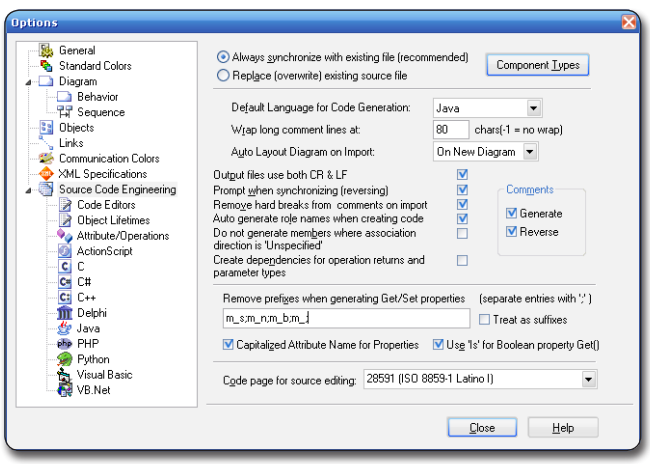


Figura 12. Marcações para transformação

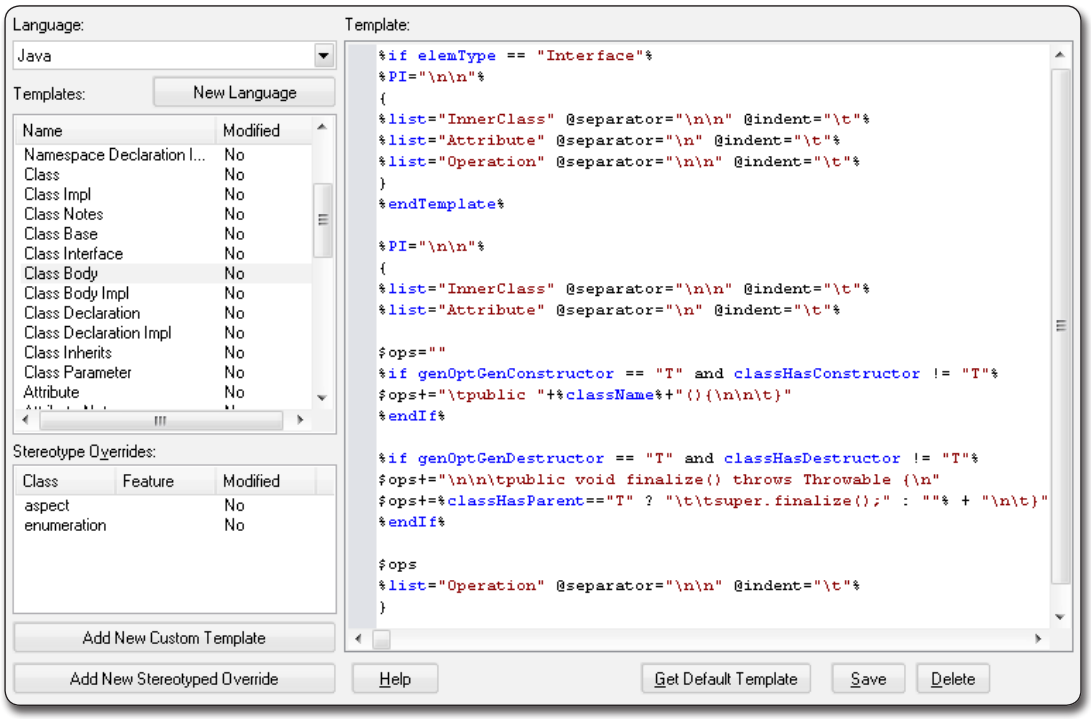


Figura 13. Modelo de mapeamento padrão

Antes de realizar a transformação, é necessário criar o mapeamento para a plataforma destino (PSM). Para editar do template de mapeamento, deve-se acessar Settings / Code Generation Templates ou Ctrl+Shift+P, importante para customização da transformação.

A ferramenta oferece vários templates de mapeamento padrão para as linguagens mais populares, porém, aquelas que a ferramenta não dispõe dos modelos, podem ser criados e os existentes podem customizados. A Figura 13 ilustra um template de mapeamento que tem como plataforma destino a linguagem Java.

A transformação será realizada de forma automática pela ferramenta. Para iniciar o processo de transformação é necessário selecionar o modelo de origem (Package PIM) no Project Browser e acessar o menu Project / Transformations / Transform Current Package.

A janela Model Transformation (Figura 14) é responsável pela seleção dos elementos a serem transformados e a plataforma do modelo de destino com seu respectivo pacote.

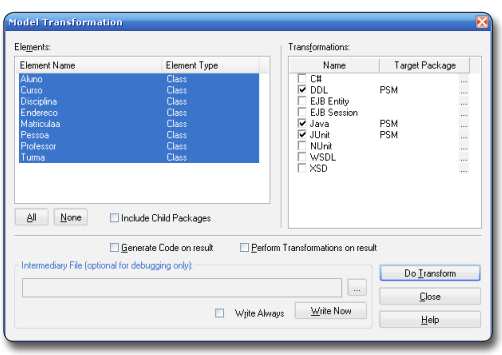


Figura 14. Transformação de Modelos

Com a confirmação através do botão Do Transform, o processo de transformação será iniciado (Figura 15), gerando os Modelos Específicos de Plataforma conforme selecionada da janela de transformação de modelos (Model Transformation).

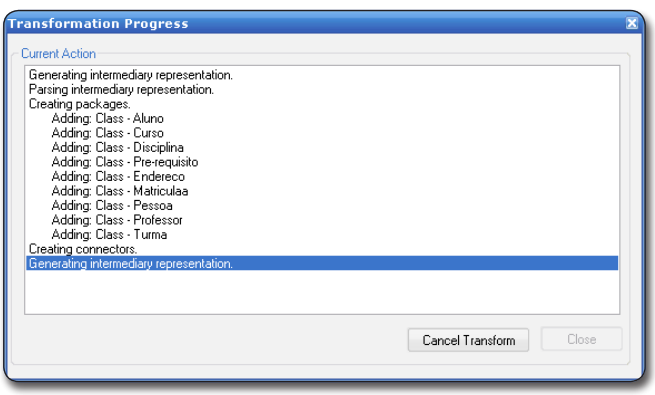


Figura 15. Processo de Transformação

O resultado deste processo de transformação foram Modelos Específicos de Plataformas, sendo para a plataforma Java (incluindo código para testes unitários com JUnit) e DDL, responsável pela criação das tabelas do banco de dados relacional.

O PSM Java irá passar por mais processo de transformação para gerar o PSM JUnit, para os casos de testes das classes. Para realizar a transformação, basta selecionar o modelo PSM em Java e realizar o mesmo processo que foi aplicado anteriormente no PIM. Após a transformação do PSM JUnit, o Project Browser terá 1 (um) PIM e 3 (três) PSMs. O exemplo das classes PSM geradas está ilustrada na Figura 16, onde se tem PSMs de plataformas

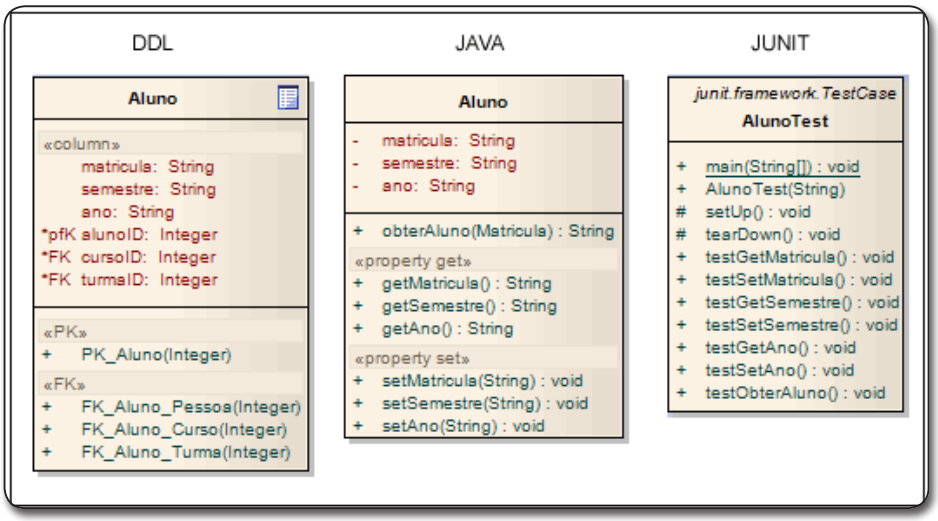


Figura 16. Classes PSM geradas de uma classe PIM

diferentes, sendo geradas a partir da classe Aluno do PIM.

O foco do processo de desenvolvimento baseado na MDA deve ser o PIM, a partir do qual foram gerados diversos Modelos Específicos de Plataforma para diferentes plataformas.

Transformação de PSM para código fonte

Após a geração dos PSMs, o próximo passo no estudo de caso é a geração do código fonte. Vale ressaltar que o processo de transformação PSM para código fonte é mais simples do que a transformação de PIM para PSM, devido ao PSM já estar bem próximo do código.

Para iniciar o processo de transformação para a geração da base de dados, basta selecionar o PSM DDL e, através do menu Project / Database Engineering / Generate Package DDL, solicitar a transformação.

A janela Generate Package DDL (Figura 17) será aberta, possibilitando a configuração de diversos parâmetros relacionados à DDL e também a seleção das tabelas a serem geradas na transformação. Para este estudo de caso foi selecionada a opção Create Primay/Foreign Key Constraints e, após selecionar o local de armazenamento do código a ser gerado, basta clicar no botão Generate e a transformação será iniciada.

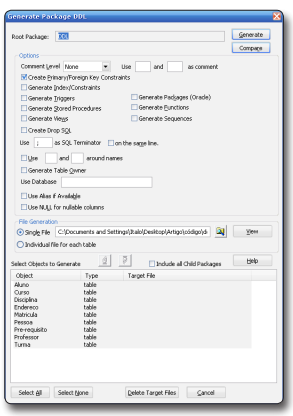


Figura 17. Geração de script de banco de dados

Um pequeno exemplo do resultado da transformação pode ser observado na Figura 18.

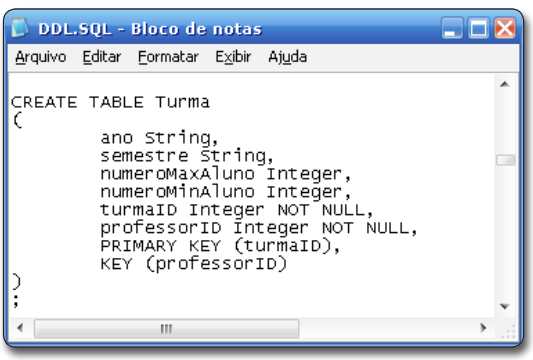


Figura 18. Código fonte DDL

Para realizar a transformação do PSM Java para código fonte, não há muita diferença do processo de geração de DDL. No menu Project / Source Code Engineering / Generate Package Source Code será solicitado a janela de configuração da transformação. Nesta janela (Figura 19) serão selecionados todos os objetos a serem gerados, podendo então ser acionada a transformação através do botão Generate.

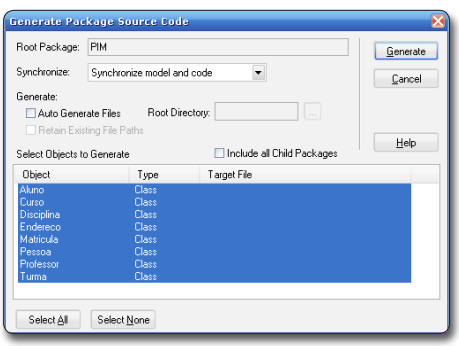


Figura 19. Janela de configuração para geração de código fonte em Java

Após o processo de transformação, é gerado o código referente a todas as classes selecionadas na janela de configuração de transformação. Parte do código gerado está apresentado na Figura 20.

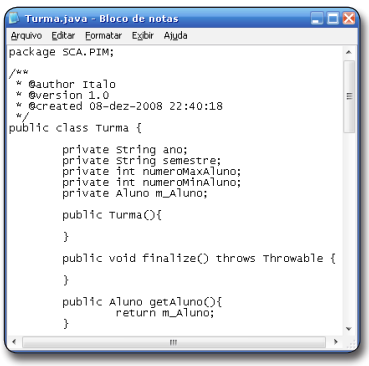


Figura 20. Código fonte Java gerado na transformação PSM para Código Fonte

O mesmo processo de transformação realizado para gerar código fonte para plataforma Java pode ser aplicado para JUnit (Figura 21).

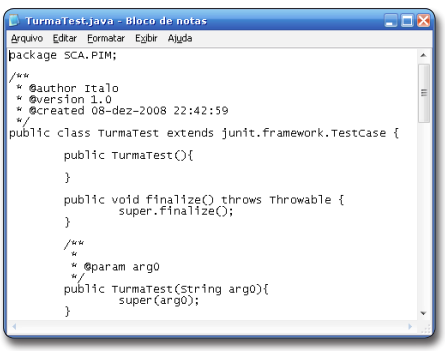


Figura 21. Código fonte JUnit

Chegamos então ao fim do estudo de caso, aplicando de forma prática os alguns dos principais princípios da MDA. Foi demonstrado como realizar a transformação do PIM para PSM e PSM para código fonte para diferentes plataformas.

Conclusão

Ao longo do desenvolvimento deste artigo foi possível demonstrar que através da MDA, mesmo esta sendo uma abordagem relativamente recente, é possível ter um aumento significativo na produtividade, garantir a documentação do sistema em vários níveis de abstração e a interoperabilidade entre os modelos, com a utilização de ferramentas apropriadas, as quais vêm surgindo com bastante força no mercado.

No estudo de caso, a produtividade foi adquirida através da modelagem do PIM, pelo qual foi possível gerar modelos e código fonte para diversas plataformas, necessários para o desenvolvimento de um sistema, como a geração das classes em Java e scripts para geração das tabelas no banco de dados. Uma vez modelado o PIM, este poderá ser reutilizado para desenvolver diversos outros sistemas, em diferentes plataformas. ●

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

Cursos Online

Assinatura

.net magazine PLUS

Mais conteúdo .NET por muito menos!

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

Curso em destaque

Crie uma loja virtual completa

Confira neste curso Online como construir uma loja virtual completa no Visual studio 2005. Aprenda como criar o banco de dados, carrinho de compras, profile, página de erros e muito mais .

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

A sua melhor opção de aprendizagem!
Assine a **.net Magazine** e comece já seu treinamento!
www.devmedia.com.br/assine

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

- Construindo relatórios com Crystal Reports e Visual Studio 2005
- Criando uma aplicação Web Completa
- Criando uma aplicação client/server no Visual Studio 2005
- Aprenda a criar um blog com ASP.NET
- Curso de C#
  * Curso em andamento

DevMedia GROUP

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

Model Transformation com ATL

Pondo em prática a M2M Model Transformation

Josias Paes

josiaspaesjr@gmail.com

Possui grau de Bacharelado em Ciências da Computação pelo Centro Universitário de João Pessoa. Tem experiência na área de desenvolvimento com ênfase para dispositivos móveis. Segue hoje a vertente acadêmica iniciando atualmente mestrado em Engenharia de Requisitos na UFPE e cursando especialização em Desenvolvimento de Software para Dispositivos Móveis na Fatec-PB.

Model Driven Development (MDD) é uma nova abordagem de desenvolvimento de software que está alcançando grandes proporções no mundo da engenharia de software. É possível encontrar inúmeras matérias, artigos e trabalhos sobre o assunto, porém, o que realmente pesa na mente de um iniciante nesta nova abordagem é como colocar os conceitos vistos na teoria em prática. Neste artigo vamos apresentar alguns fundamentos iniciais sobre MMD e MDA, e então, colocar em prática o conceito de transformações de modelos (Model Transformation) (ler Nota 1).

Neste artigo veremos

O artigo tem como objetivo apresentar a criação de um pequeno projeto baseado nos conceitos iniciais da MDD e MDA, com foco principal nas transformações de modelos do tipo Model-to-Model (M2M) utilizando a linguagem ATL.

Qual a finalidade

É muito comum os iniciantes desta nova abordagem de desenvolvimento de software terem a curiosidade sobre como aplicar os conceitos vistos na teoria sobre MDA na prática. Sendo assim, fornecemos este artigo para que seja possível por em prática tais conceitos.

Quais situações utilizam esses recursos?

Quando se trabalha com o desenvolvimento de software com MDA é necessário realizar transformações de modelos para geração automática/semi-automática de novos modelos ou código-fonte. Este artigo lhe oferece os conceitos básicos para realização destas etapas mostrando o passo a passo de como construir transformações de modelos através das linguagens ATL, KM3, Ecore e XMI.

Nota 1. Definições

Ecore: É um padrão introduzido junto com o Eclipse Modeling Framework (EMF) que define a linguagem utilizada para definir modelos padronizados, ou seja, linguagem utilizada para criação de metamodelos.

KM3: A finalidade do KM3 é dar uma solução relativamente simples para a definição de um meta-modelo, ou seja, ele é uma Domain Specific Language (DSL) utilizada para definição de meta-modelos. Para mais informações acesse http://wiki.eclipse.org/index.php/KM3.

DSL: Domain-Specific Language foi criado para solucionar problemas em um determinado domínio específico e não se destina a resolver problemas fora do contexto que foi aplicado.

OCL: É uma linguagem de expressões utilizada para especificar regras sobre modelos. As expressões OCL são utilizadas para definir condições às classes representadas em um modelo e também são utilizadas para especificar as pré e pós-condições em operações aplicadas às classes deste modelo.

XMI: É um formato padrão baseado no XML definido pela OMG, que tem como objetivo o intercâmbio de dados para o compartilhamento de modelos e ferramentas de modelagem diferentes. Os seus maiores benefícios são a consistência e compatibilidade em sistemas criados em ambientes distintos.

MDD/MDA

MDD é uma abordagem de desenvolvimento de software que se concentra na criação dos modelos do sistema. O foco principal é abstrair todas as funcionalidades e recursos do sistema expressando-as em um conjunto de modelos, por exemplo, UML. O empenho na construção de um sistema não fica mais a cargo da criação de código-fonte e sim na modelagem do sistema. Modelar reduz o tempo e os custos durante o ciclo de desenvolvimento do sistema, proporcionando aos desenvolvedores menor preocupação com o código-fonte por ser gerado automaticamente [1].

Model Driven Architecture (MDA) é outra abordagem padronizada destinada a dar suporte ao MDD definida pela Object Management Group (OMG). É utilizada na definição do sistema com foco na separação da especificação das funcionalidades do sistema e a especificação da plataforma tecnológica utilizada no desenvolvimento. Ou seja, os modelos criados no início do desenvolvimento não terão nenhuma dependência quanto às plataformas tecnológicas, ao invés disso, terão como escopo apenas a lógica funcional do sistema. Em um segundo momento ocorrerá a geração de modelos específicos a plataformas tecnológicas através do conceito de transformação de modelos [1].

O que o MDA nos traz em termos de melhorias no processo de desenvolvimento de software são:

• Produtividade – Transformações automáticas de modelos independentes de plataforma (PIM) em modelos dependentes de plataforma tecnológica (PSM) assim como a geração automática de código-fonte;

• Portabilidade – Menor esforço para portar um sistema construído em uma determinada plataforma para uma plataforma diferente;

• Qualidade – Modelagem independente de plataforma tecnológica. O foco principal da construção do sistema fica apenas nas funcionalidades e recursos do mesmo;

• Redução nos custos de desenvolvimento – Mudanças no sistema são facilmente realizadas. Como o foco do desenvolvimento está nos modelos, as alterações são feitas em nível de abstração (nos modelos) e não em código. Portanto, ao finalizar todas as mudanças, basta gerar novamente o código-fonte. Observe a Figura 1.

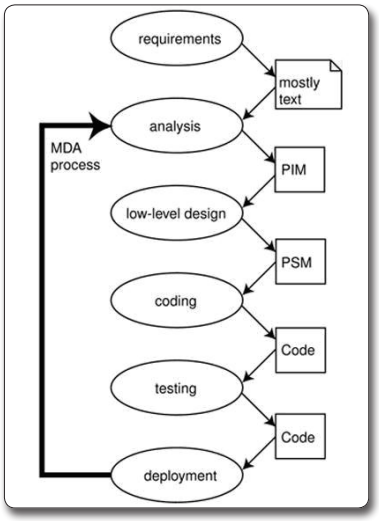


Figura 1. Ciclo de vida do desenvolvimento MDA

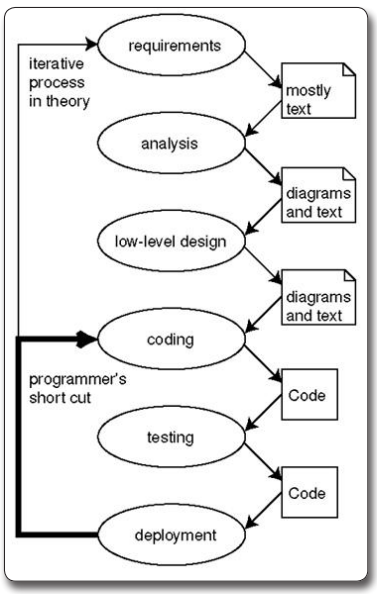


Figura 2. Ciclo de vida do desenvolvimento tradicional

As etapas do ciclo de vida do desenvolvimento MDA são idênticas as de um ciclo de desenvolvimento tradicional. A diferença está nos artefatos que são criados durante o processo. Em MDA estes artefatos são modelos formais.

Geralmente, na prática, os documentos e os diagramas gerados nas três primeiras etapas do ciclo de vida do desenvolvimento tradicional (Figura 2) desvalorizam-se assim que a codificação é iniciada. Quando ocorrem mudanças ao longo do desenvolvimento, amplia-se cada vez mais a distância entre estes. Isso acontece pelo fato de que as mudanças são feitas apenas em código, pois, muitas vezes não existe tempo disponível

para atualização dos documentos de alto nível [1].

No ciclo de vida MDA isto não acontece, todo processo é voltado para construção dos modelos e definição de regras de transformações para geração automática de novos modelos e código-fonte, minimizando o problema encontrado na prática no desenvolvimento tradicional (Figura 1) [1].

O que é um modelo?

Modelo é a representação de uma parte de um sistema, ou seja, ele representa funcionalidades ou recursos desse sistema. Dentre os vários tipos de modelos e seus níveis de abstração, todos são úteis para auxiliar a tomada de decisões sobre quais as melhores ações devem ser aplicadas para atingir as metas do sistema. Uma das linguagens mais conhecidas para modelagem de sistemas é a Unified Modeling Language (UML) [1].

O que é uma transformação?

Existem duas categorias de transformações: a primeira é a geração automática/semi-automática de um modelo de destino a partir de um modelo de origem (ambos os modelos são definidos através de um meta-modelo). Esta transformação é denominada Model-to-Model (M2M); a segunda é a geração automática/semi-automática de código-fonte a partir de um modelo de entrada. Esta transformação é denominada Model-to-Code (M2C). As transformações são definidas por um conjunto de regras que juntas descrevem como um modelo pode ser transformado em um ou mais modelos. O exemplo de transformação que será apresentado em breve será baseado na categoria M2M [2].

As transformações podem ser classificadas considerando duas características: quanto aos formalismos dos modelos de entrada e saída, e quanto ao nível de abstração [2].

As características quanto ao formalismo são:

- Exógena – São transformações entre modelos expressos em linguagens diferentes, ou seja, transforma um modelo para um formalismo diferente. Por exemplo:
  - Síntese – Transformação de um modelo de mais alto nível em um de mais baixo nível. O exemplo típico é a geração de bytecode a partir de um código-fonte Java ou geração de código-fonte a partir de um modelo.
  - Engenharia Reversa – É o inverso da síntese. Extrai o nível superior a partir de uma especificação em um nível inferior.
  - Migração – Transforma um programa escrito em uma linguagem para outra mantendo o mesmo nível de abstração.
- Endógena – São transformações entre modelos expressos na mesma linguagem. Por exemplo:
  - Otimização – Transformação destinada a melhorar certas qualidades operacionais (desempenho) do sistema, sempre preservando a sua semântica.
  - Refactoring – Mudança da estrutura interna do software para melhorar a qualidade do software (reutilização, modularidade, adaptabilidade, etc.) sem alterar o seu comportamento.

As características quanto ao nível de abstração podem ser:

- Horizontal – Transformação onde os modelos de destino e origem estão no mesmo nível de abstração. Os exemplos típicos são refactoring e migração;
- Vertical – Transformações onde os modelos de destino e origem estão em diferentes níveis de abstração. Os exemplos típicos são geração de código e engenharia reversa.

O que é um meta-modelo?

Um meta-modelo é um modelo de modelos que introduz um novo nível de abstração. Como dito, um modelo é apenas uma representação do sistema que pode ser construída em diversas representações sem sair do mesmo nível de abstração. Um meta-modelo descreve a possível estrutura dos modelos, define uma linguagem de modelagem e as suas relações, assim como restrições e regras de modelação [1].

Enfim, os meta-modelos definem a sintaxe abstrata e semântica de uma determinada linguagem de modelagem. Os meta-modelos são definidos por outra linguagem de meta-modelagem. Observe a Figura 3.

O M3 (Metametamodelo) define a semântica dos meta-modelos. O M2 (Meta-modelo) deve estar conforme o descrito em M3 e define a semântica dos modelos. O M1 (Modelo) deve estar em conformidade com o descrito em M2 e representa o mundo real, o sistema. E ao nível M0 temos as instâncias dos modelos [1].

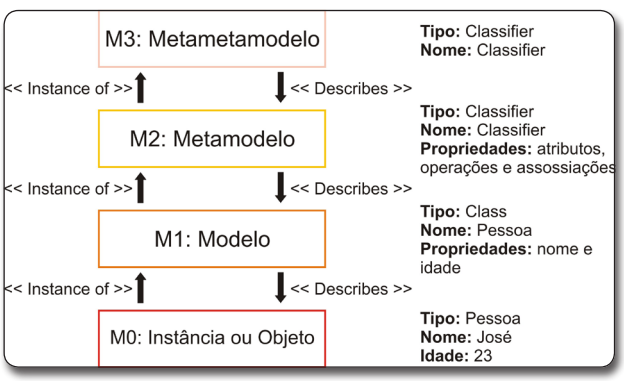


Figura 3. Diagrama representativo dos quatro níveis de abstração propostos pela OMG

O processo MDA

O processo de desenvolvimento do MDA se resume em três etapas [1] (Figura 4):

1. Construir um modelo com um alto nível de abstração independente de qualquer tecnologia de implementação. Este modelo é denominado Platform Independent Model (PIM);
2. Transforma um PIM em um ou mais modelos que tem como características os aspectos para execução em uma tecnologia específica. Este modelo é denominado Platform Specific Model (PSM);
3. Transformar PSM em código.

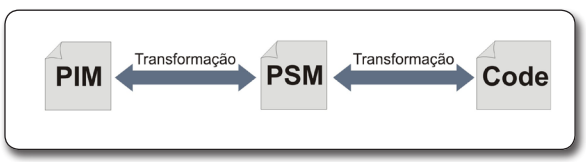


Figura 4. As três etapas do processo de desenvolvimento MDA

PIM

O PIM descreve o sistema completo baseado apenas na lógica funcional do sistema. Na criação de um PIM não deve existir preocupação com a tecnologia que será empregada no sistema. A preocupação que os analistas, engenheiros e desenvolvedores devem ter na fase de criação de um PIM é descrever fielmente as necessidade do cliente através de artefatos como modelos de classes, seqüência, comportamental, etc. [1].

PIM para PSM

Ao fim dos testes e validação do PIM, a próxima etapa é transformá-lo em outras formas de representação na qual apresentam características sobre a tecnologia que deve estar empregada no sistema, ou seja, um modelo PSM. É muito difícil alcançar um nível de transformação que seja totalmente automático. Isto significa que, após a geração dos PSM é muito provável que os membros da equipe necessitem refinar as informações geradas para garantir que o próximo processo seja realizado com sucesso [1].

Geração de código

Uma nova transformação é realizada para geração de código-fonte a partir dos PSM gerados anteriormente [1].

ATL

A Atlas Transformation Language é uma linguagem de transformação de modelos muito utilizada na MDA. Ela permite produzir modelos a partir de outros modelos. Observe a Figura 5 que apresenta uma visão geral da transformação de modelos utilizando as tecnologias ATL e Ecore.

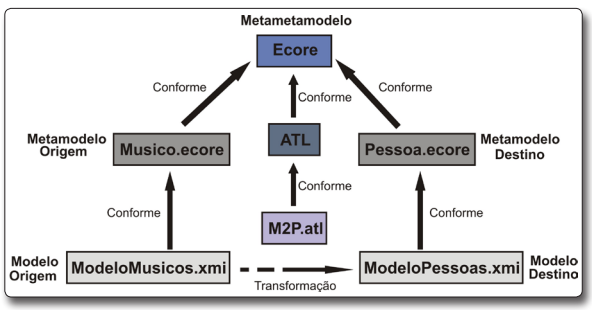


Figura 5. Overview da transformações de modelo com ATL

Em nosso exemplo iremos utilizar os conceitos da M2M model transformation para mostrar na prática como funciona a transformação de modelo de origem em um modelo de destino de acordo com os conceitos vistos anteriormente.

Os arquivos ModeloMusicos.xmi (Listagem 3) e ModeloPessoas.xmi (Listagem 5) são os modelos do nosso exemplo. Em transformações M2M o modelo de destino é gerado de forma automática/semi-automática a partir de um modelo de origem, isto significa que o arquivo ModeloPessoas.xmi será gerado automaticamente a partir do arquivo ModeloMusicos.xmi, como pode ser observado na Figura 5.

Os arquivos Musico.ecore e Pessoa.ecore são os meta-modelos do nosso projeto. Cada um destes serão construídos a partir dos arquivos Músicos.km3 e Pessoas.km3 (Listagem 1 e 2) como pode ser observado na seção "Criando os meta-modelos".

Por último, temos o arquivo M2P.atl (Listagem 4) que define as regras de transformações que tratam o processo de concepção de uma Pessoa a partir de um Músico. Este arquivo deve estar em conformidade com a linguagem ATL.

ATL units

A linguagem ATL oferece aos desenvolvedores diferentes tipos de unidade ATL denominados ATL units. Uma ATL units qualquer que seja é definida pelo arquivo .atl. Iremos definir neste arquivo as regras de transformações da nossa aplicação [3].

Uma das ATL units é o ATL modules. Esta units é responsável pela definição de operação para especificação de regras de transformação. Além dos modules, em ATL é possível criar um modelo para tipos de dados primitivos que são denominadas ATL queries. Por fim, também é possível criar ATL libraries que podem ser importadas pelas diferentes ATL units. Libraries são maneiras convenientes de fatorar código ATL.

Em nosso exemplo utilizaremos apenas ATL modules. Para mais informações sobre as outras units e operações ATL acesse http://www.eclipse.org/m2m/atl/.

Modules

Um module define um modelo para modelar transformações. Ele é composto dos seguintes elementos:

- Um header section que define alguns atributos que são relativos ao transformation module;
- Uma seção opcional de import que permite importar algumas ATL libraries existentes;
- Uma série de helpers que podem ser considerados equivalentes a ATL Java métodos;
- Um conjunto de rules que definem a forma como os modelos destino são gerados a partir de modelos de origem.

Header Section

O header section define o nome do transformation module e o nome das variáveis correspondentes aos modelos de origem e destino, como pode ser observado abaixo e na Listagem 4.

```
Nome do transformation module
module M2P;
Nome das variáveis correspondentes aos
modelos de origem e destino
create OUT : Pessoas from IN : Musicos;
```

A palavra chave from é utilizada para indicar que acontecerá a criação de um modelo destino. Ao invés do from poderia ser usada a palavra chave refines para refinar a transformação de um modelo destino.

Helpers

Utilizada para fatoração do código interno de um ATL modules. São trechos de códigos que podem ser chamados em diferentes pontos de uma transformação ATL (como se fossem métodos da linguagem Java).

Um helper pode ser definido como apresentado a seguir, e observado na Listagem 4.

```
helper context Musicos!Musico def:
idade(anoNasc: Integer, anoAtual: Integer) :
Integer =
anoAtual - anoNasc;
```

A palavra chave helper é o que define a criação de um helper, o context define o tipo do elemento no qual o helper se aplica, ou seja, o tipo dos elementos que o helper poderá invocar. O def define o nome do helper, e no seu interior é possível definir parâmetros assim como nos métodos Java, porém não são obrigatórios.

```
-- Helper com parâmetro
def: idade(anoNasc: Integer, anoAtual:
Integer) : Integer
-- Helper sem parâmetro
def: idade() : Integer
```

Logo após o def é definido o tipo de retorno do helper. Pode ser utilizado qualquer tipo de retorno. Note que o tipo de retorno é seguido pelo caractere =, isto significa que devemos atribuir a saída de um helper a este retorno. Neste corpo podemos utilizar quaisquer operações que retornem um valor, como por exemplo, uma expressão if ou operações OCL. Isso mesmo, ATL inclui operações OCL para auxiliar a definição das regras de transformações.

```
helper context Musicos!Musico def:
isFeminino(b: Boolean) : Boolean =
    if self.  then   true
    else    false
    endif;
```

Rules

A verdadeira criação do modelo destino acontece dentro das rules. Existem dois tipos de rules:

Matched Rules – São destinadas a especificar quais os tipos de elementos de um modelo de destino serão gerados a partir de elementos do modelo de origem, e o modo como os elementos do modelo de destino serão inicializados.

Na implementação destas regras devem ser realizadas as declarações do source-pattern e do target pattern. O source-pattern é a definição dos elementos de um modelo de origem correspondentes aos elementos do modelo de destino, enquanto o target pattern destina-se a especificar os elementos que devem ser gerados a partir dos elementos definidos no source-pattern. A sintaxe é apresentada abaixo:

```
rule Feminino { from
     -- Source-Pattern
   to
     -- Target-Pattern     }
```

• **Called Rules** – Segue a mesma estrutura de uma matched rules. A diferença é que as matched rules são executadas automaticamente no momento da transformação. Enquanto as called rules como o próprio nome diz, devem ser chamadas em alguma seção de código imperativo do documento de transformação para serem executadas, e dentro do seu corpo podem ser declarados parâmetros.

```
rule Feminino (var: Tipo) { from
     -- Source-Pattern
   to
     -- Target-Pattern     }
```

ATL na prática

A idéia do nosso projeto é realizar uma transformação que gera um modelo Pessoa a partir de um modelo Músico. O modelo Músico nos traz algumas informações como: nome do músico, sobrenome, ano de nascimento, sexo e o tempo que ele exerce a profissão. A partir destas informações vamos obter um modelo Pessoa que tem um nome completo, idade e a sua categoria profissional (iniciante, amador ou profissional). Para isso, devemos definir regras de transformação para que a geração do modelo destino (Pessoas.xmi) seja realizada com sucesso.

As regras utilizarão as informações contidas no modelo Músico para gerar o modelo Pessoa, por exemplo, através do ano de nascimento do músico podemos gerar a idade da pessoa, através do tempo de profissão podemos dizer qual categoria profissional a pessoa se encaixa. Tudo isso será definido conforme o especificado no meta-modelo do modelo Pessoas.

Então vamos iniciar o nosso projeto.

Criando um projeto ATL

A ferramenta utilizada para construção de um projeto para transformações de modelos pode ser encontrada no Quadro "Tecnologias utilizadas". Os principais componentes que utilizaremos serão o Eclipse Modeling Framework (EMF) e o ATL. O EMF é um projeto de modelagem e geração de códigos que tem sua execução sob a plataforma Eclipse. Os modelos e códigos-fonte são gerados a partir de um modelo descrito em XMI.

Com o Eclipse Modeling Tools instalado podemos iniciar a construção do projeto ATL. O primeiro passo é habilitar a perspectiva ATL (ATL perspective). Para isso, acesse o menu Window/Open Perspective/Other/ATL. Feito isso, é hora de criar o nosso projeto, abra o menu File/New/ATL Projects, como pode ser observado na Figura 6. Nomeie o projeto como Musico2Pessoa.

Tecnologias utilizadas

Utilizaremos os frameworks EMF para o Eclipse. Para isto faça o download do Eclipse Modeling Tools através do link http://www.eclipse.org/downloads/. Este pacote contém uma coleção de componentes para modelagem de sistemas como EMF, GMT, ATL, OCL, UML 2.0 dentre vários outros.

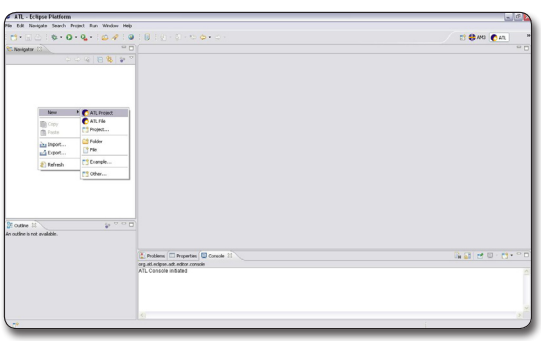


Figura 6. Criando um projeto ATL

Criando os meta-modelos

Faz-se necessário criar os meta-modelos tanto para os modelos origem quanto para os modelos destino. Para isto usaremos o formato de arquivo fornecido pela EMF, o .ecore. Esta linguagem não é muito amigável, e a fim de tornar mais fácil a codificação de um meta-modelo, o EMF inclui uma simples notação textual denominada Kernel Metametamodel 3 (KM3). O KM3 facilita muito a criação de um meta-modelo, sendo possível ao fim de sua construção injetá-lo no formato Ecore utilizando as ferramentas do EMF.

Observe as Listagens 1 e 2 e perceba o quão a linguagem KM3 assemelha-se com a linguagem Java. Segue algumas notações que utilizaremos neste artigo sobre a criação de arquivos .km3:

Como dito, KM3 assemelha-se bastante com a linguagem Java, portanto nela existem os conceitos de pacotes, classes, atributos, associações, etc.

Em nossos arquivos devemos sempre criar o pacote, como pode ser observados nas Listagens 1 e 2. Estes pacotes devem conter o mesmo nome do arquivo .km3. Os pacotes são criados através da palavra chave package.

```
package Musicos{
}
```

Os atributos sempre são definidos com a palavra chave attribute seguida pelo nome do atributo e o tipo do atributo.

```
attribute nome: String;
```

Existe uma parte de código que é obrigatório para todo e qualquer arquivo .km3. É a criação do pacote PrimitiveTypes. Esta importação serve para definir os tipos de atributos que serão utilizados na criação dos atributos.

```
package PrimitiveTypes {
  datatype String;
  datatype Integer;
  datatype Boolean;
}
```

Primeiramente vamos criar o meta-modelo Musicos.km3 (Listagem 1). Dessa forma, acesse o menu File/New/File e crie o arquivo .km3 (Figura 7). Ao criá-lo, clique com o botão direito do seu mouse no arquivo e então selecione a opção Inject KM3 to Ecore metamodel para transformá-lo em um arquivo .ecore (Figura 8). Repita o processo para o arquivo Pessoas.km3 (Listagem 2).

O meta-modelo Musicos.km3 define alguns elementos que são utilizados na inicialização dos elementos do modelo Pessoas.km3.

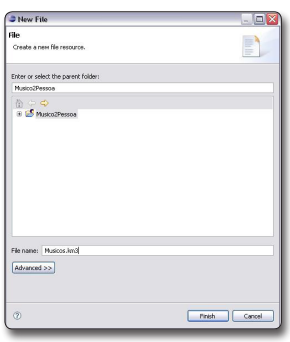


Figura 7. Criando um novo arquivo

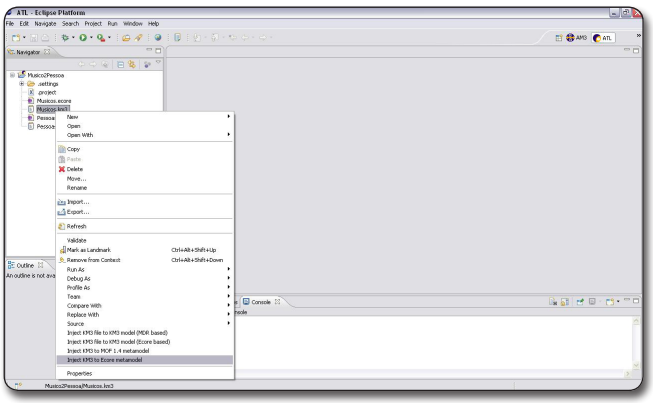


Figura 8. Transformando um arquivo KM3 em Ecore

Listagem 1. Musicos.km3

```
package Musicos{

class Musico{
    attribute nome : String;
    attribute sobrenome : String;
    attribute anoNascimento: Integer;
    attribute sexo : Boolean;
    attribute tempoProfissao: Integer;
   }
}

package PrimitiveTypes {
datatype String;
datatype Integer;
datatype Boolean;
}
```

Listagem 2. Pessoas.km3

```
package Pessoas{

abstract class Pessoa{
    attribute nomeCompleto : String;
    attribute idade : Integer;
    attribute categoriaProfissional : String;

   }

class Masculino extends Pessoa{ }

class Feminino extends Pessoa{ }

}

package PrimitiveTypes {
datatype String;
datatype Integer;
}
```

Criando o modelo de entrada

Com o meta-modelo definido, podemos criar o modelo de entrada ou origem. Estes contêm informações baseadas no meta-modelo construído anteriormente, para que através deste possamos gerar o nosso modelo destino. Descreveremos nosso modelo utilizando a linguagem XMI, na qual podemos instanciar os elementos definidos no meta-modelo. Para criar um arquivo XMI, acesse o menu File/New/File e então crie o arquivo ModeloMusicos.xmi (Listagem 3).

Inicialmente defina o namespace da tag xmi com o nome do pacote descrito no meta-modelo.

```
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://
```

```
www.omg.org/XMI" xmlns="Musicos">
-- Musicos é o nome do pacote definido na Listagem
1
```

As tags do documento são as classes definidas no meta-modelo. No nosso exemplo criamos apenas uma classe, Musico. A cada instância criada devem-se inicializar os elementos (atributos) dessa tag. Na classe Musico criamos os atributos nome, sobrenome, sexo, ano de nascimento e o tempo de profissão.

```
 <Musico nome="Sanmara" sobrenome="Alves"
sexo="true" anoNascimento="1985"
tempoProfissao="1"/>
 <Musico nome="Victor" sobrenome="Magliano"
sexo="false" anoNascimento="1986"
tempoProfissao="9"/>
```

Listagem 3. ModeloMusicos.xmi

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
xmlns="Musicos">
  <Musico nome="Marcos" sobrenome=" Filho" sexo="false"
anoNascimento="1980" tempoProfissao="4"/>
  <Musico nome="Sanmara" sobrenome="Alves" sexo="true"
anoNascimento="1985" tempoProfissao="1"/>
  <Musico nome="Victor" sobrenome="Magliano" sexo="false"
anoNascimento="1986" tempoProfissao="9"/>
  <Musico nome="Diego" sobrenome="Santos" sexo="false"
anoNascimento="1983"  tempoProfissao="12"/>
</xmi:XMI>
```

Definindo as regras da transformação

Neste momento vamos começar a criar a lógica de toda a transformação através da definição das regras de transformação. Para isto, criamos um arquivo .atl onde podemos construir as estruturas descritas anteriormente (ATL units). Para criar o arquivo acesse o menu File/New/ATL File e uma janela irá surgir (Figura 9). Nesta janela podemos construir o nosso ATL module, onde inicialmente definiremos o nosso header section. Defina o nome do ATL module e logo após especifique os meta-modelos de entrada (IN) e saída (OUT) e clique no botão ADD. Em outros casos também é possível especificar uma ou várias ATL libraries. Ao finalizar teremos o arquivo M2P.atl e em seu interior a header section.

```
module M2P;
create OUT : Pessoas from IN : Musicos;
```

Feito isso, podemos então construir toda a lógica de transformação para geração do modelo ModeloPessoas.xmi.

Iremos utilizar os atributos definidos em Musico para inicializar os elementos definidos em Pessoa. As regras que vamos construir são do tipo matched rule chamadas Feminino e Masculino, onde ambas tem o mesmo tratamento (Listagem 4). Como dito anteriormente, a primeira tarefa é declarar o source-pattern e logo após o target-pattern.

```
rule Feminino { from   m : Musicos!Musico
(s.isFeminino(s.sexo))
    to
t : Pessoas!Feminino (
    nomeCompleto <- m.nome + ' ' +
    m.sobrenome, (. . .) ) }
```

Note que na declaração do source-pattern foi realizada uma chamada a um helper denominado isFeminino. Este helper (Listagem 4) testa o sexo do músico de entrada para saber se é do sexo feminino ou masculino. Para a regra Feminina, se o retorno do helper for true a regra será aplicada para a instância do músico em atividade, caso o retorno seja false a regra não será aplicada à instância do músico em atividade. A regra Masculina é o inverso da regra Feminina.

```
rule Masculino {  from
   m : Musicos!Musico (not m.isFeminino(m.sexo))
 to
   (. . .)   }
```

Perceba também que a chamada do helper foi realizada utilizando a variável local m. A variável m representa a instância do músico em atividade. Esta variável é utilizada para chamadas de helpers e os elementos de músico. Isto faz com que a regra definida no arquivo .atl seja aplicada para cada instância de Musico no arquivo .xmi. Neste helper faz-se necessário passar um parâmetro booleano, ou seja, o sexo do músico em atividade.

Na declaração do target-pattern será realizada a inicialização dos elementos do modelo destino. Observe a Listagem 2 e veja que a classe Pessoa é abstrata e possui mais duas classes (Feminino e Masculino) que estendem a classe Pessoa. Então na declaração do target-pattern utilizaremos uma dessas duas classes, ou seja, para a regra Feminino usaremos a classes Feminino, assim como para regra Masculino usaremos a classe Masculino (Listagem 4).

Isto significa que a regra inicializa apenas os elementos da classe que for declarada no target-pattern. Quando geramos o modelo destino, como dito, as tags do arquivo .xmi serão criadas de acordo com as classes declaradas no meta-modelo. Ou seja, as tags do arquivo ModeloPessoas.xmi são Feminino e Masculino, como pode ser observado na Listagem 5.

Também foram definidos outros helpers. O helper idade, que tem como objetivo calcular a idade de uma pessoa baseado no ano de nascimento de um Músico. O outro helper é denominado categoria, que especifica a categoria profissional de uma pessoa baseado no tempo de profissão de um determinado Músico. Estes helpers são chamados na inicialização dos elementos do modelo destino, ou seja, no target-pattern das regras descritas na Listagem 4.

Para inicializar o elemento idade foi utilizado o valor de retorno do helper idade. Passamos por parâmetro o ano de nascimento de um músico e o ano atual para retornar a idade de uma pessoa. Na inicialização do elemento categoria profissional chamamos o helper categoria, recebendo como parâmetro o tempo de profissão de um determinado músico para determinar a categoria profissional da pessoa. Veja o código:

```
t : Pessoas!Feminino (
    (...)
    idade <- m.idade(m.anoNascimento, 2009),
    categoriaProfissional <- m.categoria(m.
    tempoProfissao)  )
```

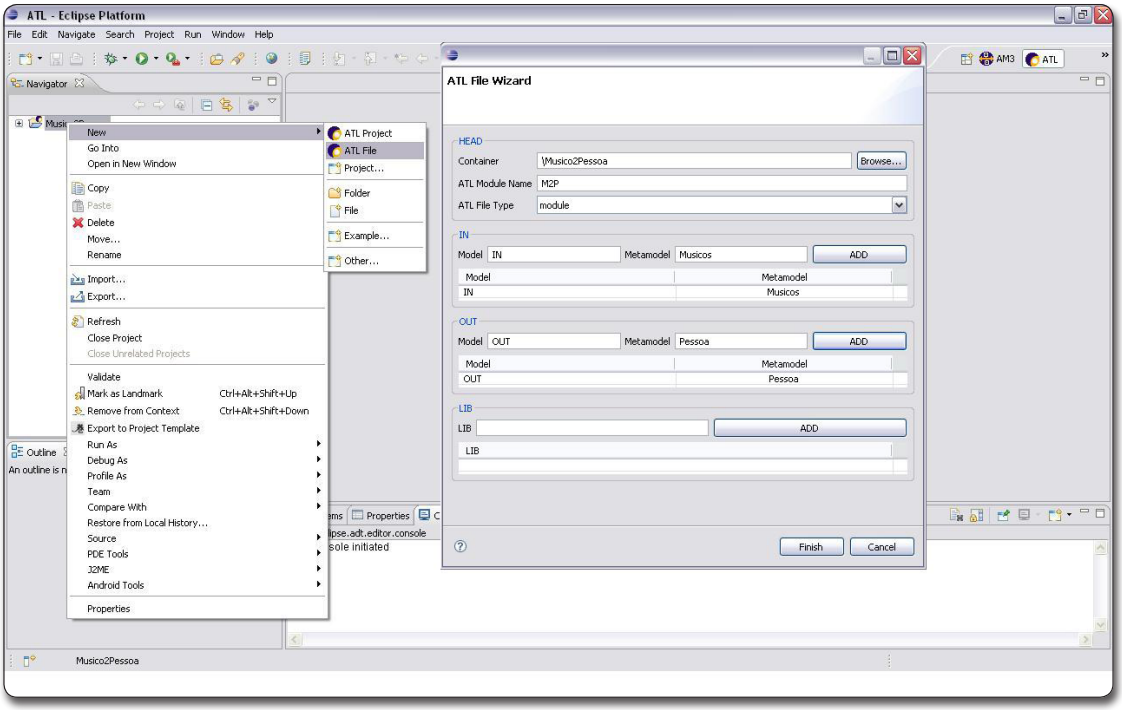


Figura 9. Criando um arquivo ATL

Listagem 4. M2P.atl

```
module M2P; -- Module Template
create OUT : Pessoas from IN : Musicos;

helper context Musicos!Musico def: isFeminino(b: Boolean) :
Boolean =
if b  then
    true
else
    false
endif;

helper context Musicos!Musico def: idade(anoNasc: Integer,
anoAtual: Integer ) : Integer =
anoAtual - anoNasc;

helper context Musicos!Musico def: categoria(tempo: Integer):
String =
if tempo<=1  then
    'Estudante'
else
    if tempo>1 and tempo<5  then
        'Amador'
    else
        'Profissional'
    endif
endif;

rule Feminino {
from
    m : Musicos!Musico (m.isFeminino(m.sexo))
  to
    f : Pessoas! Feminino (
        nomeCompleto <- m.nome + ' ' + m.sobrenome,
        idade <- m.idade(m.anoNascimento, 2008),
        categoriaProfissional <- categoria
(m.tempoProfissao)
      )
}

rule Masculino {
from
    m : Musicos!Musico (not m.isFeminino(m.sexo))
  to
    f : Pessoas! Masculino (
        nomeCompleto <- m.nome + ' ' + m.sobrenome,
        idade <- m.idade(m.anoNascimento, 2008),
        categoriaProfissional <- categoria
(m.tempoProfissao)
      )
}
```

Executando

Com a definição dos meta-modelos de Músico e Pessoa, a criação do modelo de origem (Musico) e o arquivo ATL, podemos então realizar a nossa transformação de modelos. Para isto acesse o menu Run/Open Run Dialog para abrir a janela de configuração de execução.

Em ATL Transformation clique com o botão direito do mouse e selecione new para criar uma nova configuração de execução ATL (Figura 10). Ao criar, nomeie a configuração como desejar e selecione o arquivo .atl, em ATL file, o qual contém as regras de transformações que serão utilizados nas transformações de modelos.

Logo após informe o local dos meta-modelos construídos utilizando o botão Workspace. Não se esqueça de importar os dois meta-modelos. E por último, vamos definir os modelos de origem e destino. No modelo de origem informe o Workspace do arquivo ModeloMusicos.xmi. O modelo de destino não existe, é justamente o modelo que iremos gerar. Então apenas informe o local que o modelo deve ser criado e defina o nome do modelo como sendo ModeloPessoas.xmi. Feito isso, clique no botão Run e assim o arquivo ModeloPessoas.xmi será criado de acordo com as regras e o seu meta-modelo (Listagem 5).

Listagem 5. ModeloPessoas.xmi

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMI xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI"
 xmlns="Pessoas">
  <Feminino nomeCompleto="Sanmara Alves" idade="23" categoriaPro
 fissional="Estudante"/>
  <Masculino nomeCompleto="Marcos  Filho" idade="28" categoriaPro
 fissional="Amador"/>
  <Masculino nomeCompleto="Victor Magliano" idade="26" categoria
 Profissional="Profissional"/>
  <Masculino nomeCompleto="Diego Santos" idade="25" categoriaPro
 fissional="Profissional"/>
</xmi:XMI>
```

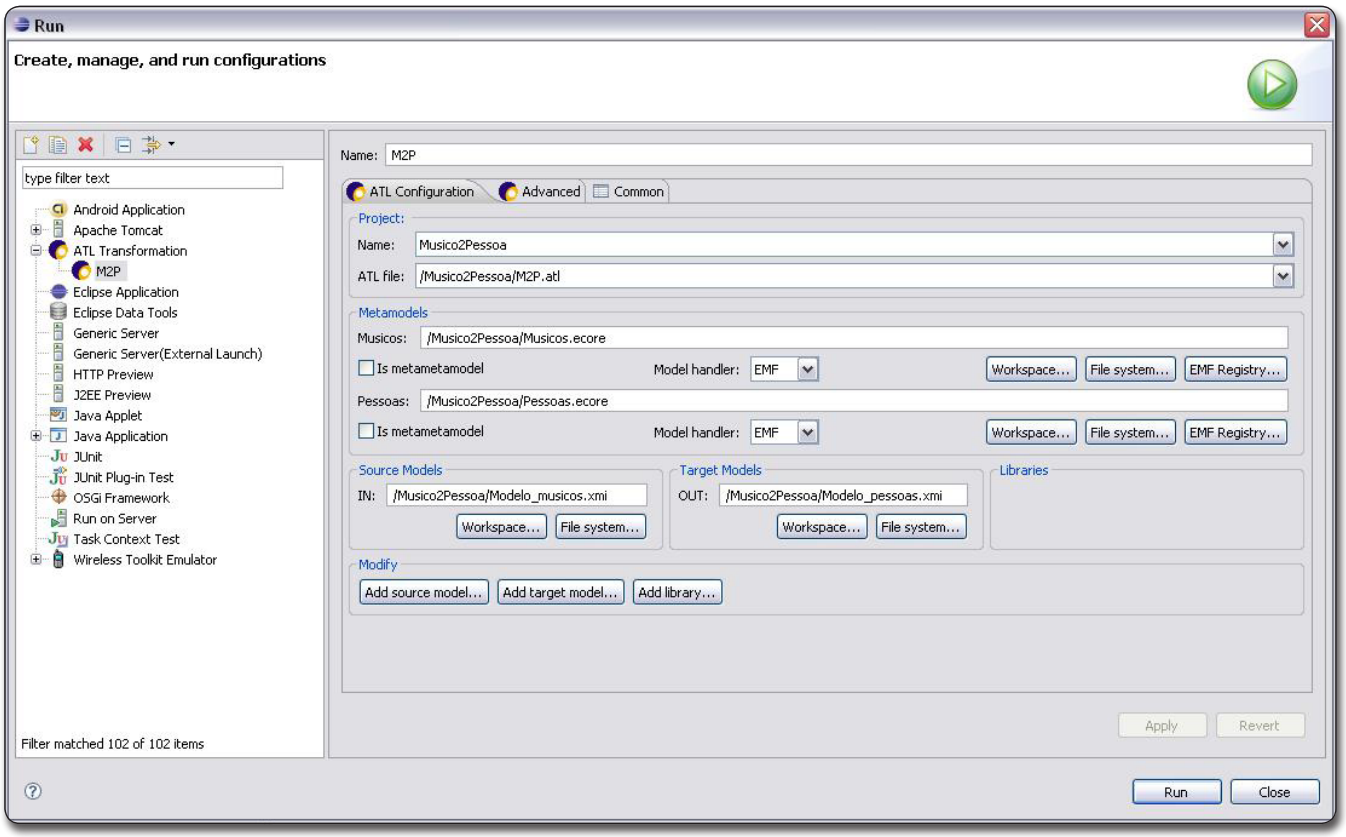


Figura 10. Executando a transformação

Conclusões

Neste artigo aprendemos como realizar transformações de modelos (M2M) utilizando as linguagens ATL, KM3, Ecore e XMI sob o conceito do desenvolvimento de software com MDA. Para isto foi realizada uma contextualização sobre MDD e MDA de forma a apresentar os conceitos para realização dos objetivos deste artigo.

Em outra parte, abordamos na prática os conceitos vistos sobre transformações de modelos. Utilizamos a linguagem ATL para definição das regras de transformações, KM3 para construção dos meta-modelos e XMI para definição dos modelos. Todas estas linguagens foram utilizadas em conjunto com a ferramenta EMF do Eclipse.

Em nosso exemplo realizamos uma transformação de modelos consideravelmente simples, com intuito de apresentar os primeiros passos para utilização das linguagens necessárias e amostragem de como transformações de modelos são realizadas na prática. Inicialmente construímos os meta-modelos de Músicos e Pessoas, logo após definimos o modelo de Músico (modelo de origem) e então construímos as regras de transformação utilizando os meta-modelos e o modelo definido para geração do modelo de destino.

Com isso esperamos que os iniciantes em MDA possam ter um embasamento simples sobre como aplicar os conceitos vistos na literatura e colocá-los em prática. ●

Referências

Kleppe, Anneke et al. MDA Explained: The Model Driven Architecture: Practice and Promise. Addison Wesley. 2003.

Mens, Tom et al. A taxonomy of Model Transformation. Elsevier. 2006.

ATLAS Group. ATL User Manual. 2006.

Jouault, F et al. KM3: A DSL for Metamodel Specification. Proceedings of 8th IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems. 2006.

Links

OMG: Site Oficial
www.omg.org
ATL: Site Oficial
www.eclipse.org/m2m/atl
KM3: Site do Eclipse com referências sobre o KM3
http://wiki.eclipse.org/index.php/KM3

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

Sistemas de Gestão de Qualidade

Janaína Bedani Dixon Moraes
jana_dixon2001@yahoo.com.br
Especialista em Concepção e Gerência de Sistemas Orientado a Objeto pela UNIRONDON, obteve a certificação de Auditor Interno ISO 9001:2000. Tem exercido a atividade de analista de sistemas desde 2002 e prestado serviço a diversos órgãos do Estado de Mato Grosso. Trabalha atualmente nos projetos SEFAZ-MT como analista de requisitos e gerente de projeto. Presta serviço de consultoria para a implantação do Sistema de Gestão de Qualidade da empresa onde trabalha.

Sistema de Gestão da Qualidade (SGQ) diz respeito a um conjunto de regras mínimas, implementado de forma adequada, com o objetivo de orientar cada parte da empresa para que execute de maneira correta e no tempo devido a sua tarefa, em harmonia com outras, para atingir o objetivo principal da empresa: ser competitiva.

Para a empresa ser competitiva é necessário que esta tenha qualidade (satisfação dos clientes) e produtividade (fazer cada vez mais com menos recursos). Com isso a empresa que tem qualidade e produtividade adquire competitividade, o que lhe garante o lucro sustentado. Desta forma, o SGQ leva a organização a analisar requisitos, criar e controlar processos que tornem possível a obtenção de produtos de qualidade.

Neste artigo veremos

O objetivo desse artigo é fornecer ao leitor explicação do que são, para que servem e quais são os procedimentos necessários para criar um Sistema de Gestão da Qualidade. Detalhamos o Sistema de Gestão da Qualidade ISO 9001.

Qual a finalidade

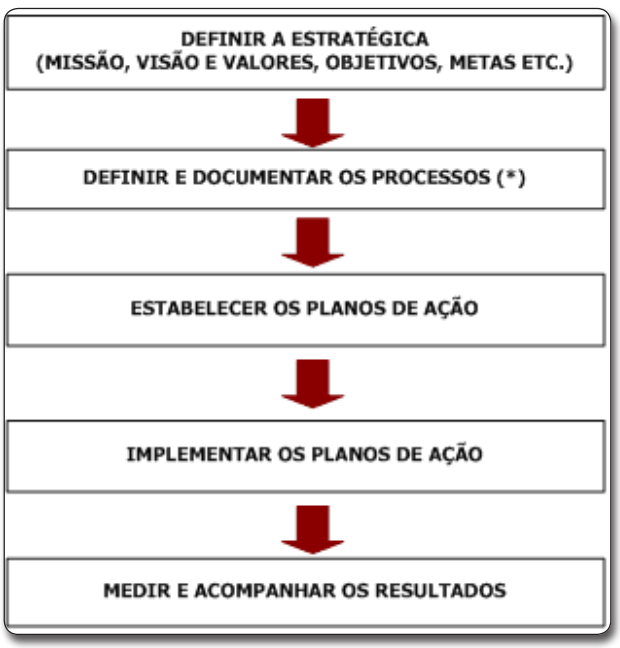
A qualidade deixou de ser um diferencial das organizações e tornou-se um pré-requisito para o mercado. O SGQ leva a organização a analisar requisitos, criar e controlar processos que tornem possível a obtenção de produtos de qualidade.

Quais situações utilizam esses recursos?

Esse tema é útil para profissionais que procuram conhecer quais são os procedimentos necessários para a elaboração do Sistema de Gestão da Qualidade com base na Norma ISO 9001.

O SGQ fornece uma estrutura para melhoria contínua com objetivo de aumentar a probabilidade de ampliar a satisfação de clientes ou partes interessadas. Avaliações do SGQ são importantes pois se não forem revisadas, tornam-se desatualizadas e não agregam valor. Para obter as informações sobre a necessidade de revisão do SGQ são realizadas auditorias. Nas auditorias, para cada processo, é possível verificar se: o processo está adequadamente definido; as responsabilidades estão atribuidas; os procedimentos estão implementados e mantidos e se o processo é eficaz para atender aos resultados requeridos pelo cliente.

A Figura 1 apresenta resumidamente o fluxo necessário para a construção de um SGQ.



* Os processos devem incluir a sistematização dos objetivos e metas.

Figura 1. Fluxo para construir um SGQ (ISO 9001:2000 – Item 2.3)

O primeiro passo apresentado é "Definir Estratégia". Estratégia é o método pelo qual a empresa define a mobilização de seus recursos para alcançar os objetivos propostos. É um planejamento global a curto, médio e longo prazo.

O planejamento estratégico procura responder a questões básicas, como: Por que a organização existe? O que e como ela faz? Onde ela quer chegar?

A elaboração do Planejamento Estratégico envolve definir a missão, visão e valores, objetivos, metas, que servem de referência e guia para a ação organizacional.

A declaração de missão deve refletir a razão de ser da empresa, geralmente é uma declaração curta, que destaca as atividades da empresa. A visão da empresa é a definição de onde a empresa quer estar em um determinado período de tempo.

A definição dos valores da organização são entendimentos e expectativas que descrevem como os profissionais da organização se comportam e sobre os quais todas as relações organizacionais estão baseadas.

Análise de ambiente interno e externo é fundamental para a definição das metas e estratégias, pois é da análise de ambiente que as estratégias são formuladas. A análise de ambiente é a definição das forças, fraquezas, ameaças e oportunidades da empresa que afetam a empresa no cumprimento da sua missão.

As estratégias são escritas com base na análise de ambiente levantada, após uma priorização dos principais objetivos e agrupamento por temas. A estratégia precisa estar voltada para o futuro da organização, porém para ser bem descrita necessita estar de acordo com as etapas anteriores (missão, visão, negócio e ambiente).

O Plano de Ação envolve definir claramente quem será o responsável pela execução de determinada ação, como e quando será implementada, qual será o cronograma a ser seguido e qual será o custo. Esta etapa garante a execução de tudo o que foi levantado e priorizado. Não adianta definir um desafiador planejamento se não houver implementação e acompanhamento e para isto o Plano de Ação precisa ser apresentado e seguido.

Um outro aspecto importante para o sucesso da implementação do planejamento estratégico está em ficar atento às mudanças do mercado. O planejamento estratégico não pode ser fixo, o processo precisa ser "dinâmico" e se for bem utilizado trará bons resultados para uma organização.

Informação sobre as normas ISO série 9000:2000

A NBR ISO 9000 define Sistema de Gestão de Qualidade como "conjunto de elementos inter-relacionados ou interligados para estabelecer política e objetivos e para atingir esses objetivos, com o fim de controlar uma organização no que diz respeito à Qualidade".

Existem vários possíveis modelos de Sistemas de Gestão da Qualidade exemplo: o Controle da Qualidade Total (TQC - Total Quality Control), o Gerenciamento pela Qualidade Total (TQM – Total Quality Management), etc. O modelo proposto pela ISO 9000 é adotado no mundo inteiro por possuir a particularidade de ser simples, eficiente e eficaz.

A sigla ISO é formada pelas letras iniciais de International Organization for Standardization (Organização Internacional para Normalização Técnica), com sede em Genebra Suíça. No Brasil a norma internacional ISO 9000 é registrada sob o código NBR ISO 9001, nos dois órgãos que tratam do assunto de "Normalização Técnica": INMETRO e ABNT.

A ISO série 9000 (uma pequena parte da série completa ISO) é um conjunto de Normas Técnicas que trata exclusivamente do assunto Gestão da Qualidade.

Historicamente, a ISO série 9000 é resultado da evolução de normas instituídas em duas fases: segurança das instalações nucleares e confiabilidade de artefatos militares e aeroespaciais.

A Figura 2 apresenta a visão histórica da ISO 9000.

A partir da versão de 1994, a série "ISO 9000" passou a ser conhecida como "família ISO 9000", constituída levando em conta as situações contratual e não-contratual das organizações.

Para se compreender a filosofia ISO 9000, é necessário antes

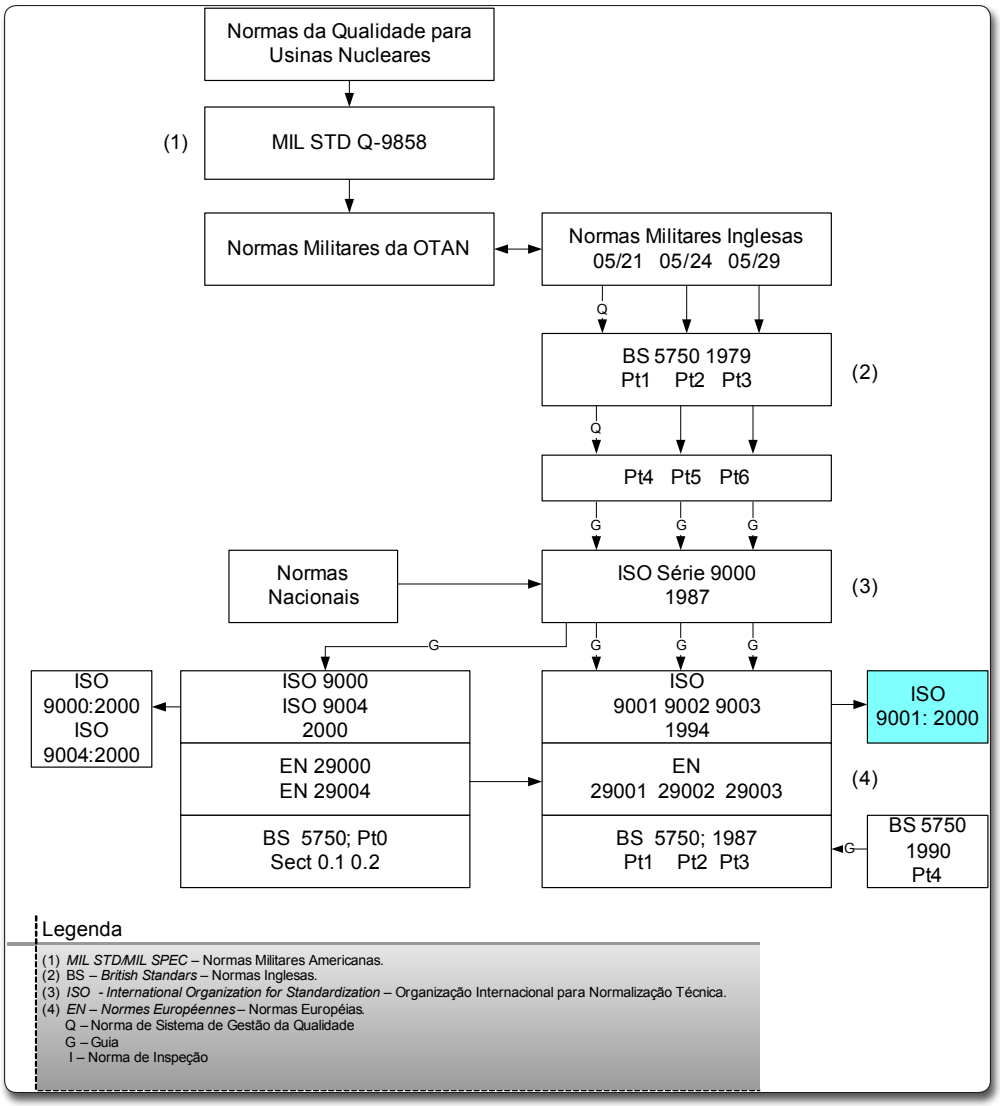


Figura 2. Visão Histórica da ISO 9000 (MARANHÃO, MAURITI, 2006)

definir o que é uma "situação contratual". Nas relações envolvendo qualidade temos dois interessados: o cliente, para o qual é destinado o produto, e o fornecedor, que fabrica, intermedia ou vende o produto, na relação imediata com o cliente. A terminologia da versão 2000 define o "fornecedor" de "organização".

Na situação contratual é contemplada a relação da organização com o cliente externo, envolvem compromissos contratuais externos à organização. Em um simples exemplo de cliente e organização temos: quando se compra um remédio em uma farmácia, se está diante de uma situação contratual, pois neste caso existe o "cliente" (o comprador), a "organização" (o representante da farmácia) e obrigações bilaterais de contrato (remédio nas condições combinadas contra dinheiro).

Alternativamente às situações contratuais, é possível considerar apenas o que existe dentro da fronteira do Sistema de Gestão da Qualidade. A satisfação do cliente no final da cadeia somente é possível quando os passos anteriores, isto é, as atividades dentro do ambiente da organização forem construtivas. Sob esta alternativa, podemos estender o conceito de cliente e organização para tratar todo e qualquer processo interno à organização, definindo o que se costuma chamar "cadeia de clientes e fornecedores internos".

A situação não-contratual acontece quando o foco da organização é a preocupação interna para produzir qualidade, ou seja, são diretrizes válidas apenas dentro da fronteira do SGQ estabelecido e utilizado pelas pessoas de dentro da organização.

As normas da série ISO 9000 não tratam diretamente da qualidade de produtos, asseguram a estabilidade do seu processo de produção. Por exemplo, suponhamos que uma empresa "A" possui o certificado de ISO 9000 e forneça como produto aos seus clientes o refrigerante "D". O fato da empresa "A" possuir o certificado ISO 9001:2000 não prova que o refrigerante que ela fornece é o melhor do mercado. Ter um certificado ISO 9001:2000 indica que a empresa desenvolve seu negócio de uma forma mais sistêmica e tem trabalho constante para se atingir e manter as metas estabelecidas.

A ISO 9000 na gestão da qualidade estabelece oito princípios de qualidade para que a organização conduza e opere suas atividades visando melhorar continuamente seu desempenho através do foco no cliente: Foco no cliente, Liderança, Engajamento das pessoas, Abordagem de processos, Abordagem sistêmica para a gestão, Melhoria contínua, Abordagem factual para a tomada de decisão e Benefícios mútuos nas relações com os fornecedores.

No principio de qualidade, a abordagem de processo utiliza os princípios do ciclo PDCA. O PDCA foi idealizado por Walter Shewart na década de 30 e difundido por Deming na década de 50 no Japão. As atividades de Planejamento (Plan - P), realização (Do - D), verificação (Check - C) e ação (Act - A) devem se inter-relacionar de forma que o ciclo seja realimentado com ações não só corretivas, mas também pró-ativas, promovendo melhorias na gestão.

Dentro do contexto de um sistema de gestão da qualidade, o PDCA é um ciclo dinâmico que pode ser desdobrado dentro de cada processo da organização e para o sistema de processos em sua totalidade. Ele está associado com planejamento, implementação, controle e melhoria contínua dos processos de realização do produto e do sistema de gestão da qualidade.

A Série de Normas ISO 9000		
Número	Título	Finalidade
NBR ISO 9000	Sistemas de gestão da Qualidade—Fundamentos e vocabulário.	Estabelecer os fundamentos e o vocabulário da Qualidade. Entende-se fundamentos como sendo o conjunto de proposições e idéias mais gerais, de onde se pode deduzir os conhecimentos sobre o assunto Qualidade.
NBR ISO 9001	Sistemas de gestão da Qualidade Requisitos	Especificar os requisitos de sistemas de gestão da Qualidade para a organização produzir produtos conformes e obter satisfação dos clientes. É a única norma de natureza contratual da série 9000.
NBR ISO 9004	Sistemas de gestão da Qualidade Diretrizes para melhorias de desempenho.	Prover guias para sistemas de gestão da Qualidade, incluindo melhorias contínuas, para satisfação dos clientes e de outras partes interessadas.
NBR ISO 19011	Diretrizes para auditoria de sistemas de gestão da Qualidade e gestão ambiental.	Prover requisitos e diretrizes para processos de auditorias (SGQ- sistema de gestão de qualidade/SGA- sistema de gestão ambiental).

Tabela1. Série de Normas ISO 9000 (MARANHÃO, MAURITI, 2006))

A Norma ISO 9000 estabelece as bases para construção de SGQs: fundamentos e vocabulário. A terminologia apresentada por ela tem por objetivo limitar a variedade de interpretações.

Para a situação contratual é indicada a aplicação da norma ISO 9001. Já na situação não contratual é aplicada a norma ISO 9004. As normas ISO 9001 e ISO 9004 formam o chamado "par consistente", sugerindo a aplicação conjunta dessas normas. A ISO 19011 que trata de auditorias apresenta normas, para ambas as situações - contratual e não contratual.

Para auxiliar a compreensão dos benefícios de implementação de Sistemas de Gestão da Qualidade, é interessante que a organização inicie o seu projeto de qualidade pela aplicação da norma ISO 9004, de organização interna (situação não-contratual).

Documentação típica de um Sistema de Gestão da Qualidade ISO 9000

Existem passos comuns para a implementação de um SGQ. Alguns deles são apresentados na Tabela 2.

Abordaremos neste artigo somente o passo "Elaboração dos documentos que irão compor o SGQ".

A organização normativa de um Sistema de Gestão da Qualidade pode ser ilustrada de forma triangular. A forma triangular usualmente representa a organização, simbolizando os níveis hierárquicos (estratégico, tático e operacional) e a correspondente distribuição de pessoas, isto é, poucas pessoas no topo, com mais poder, e mais pessoas na base da pirâmide com poderes menores.

Ao observar a Figura 3 verifique que existe correspondência entre todas as partes apresentadas através do triângulo.

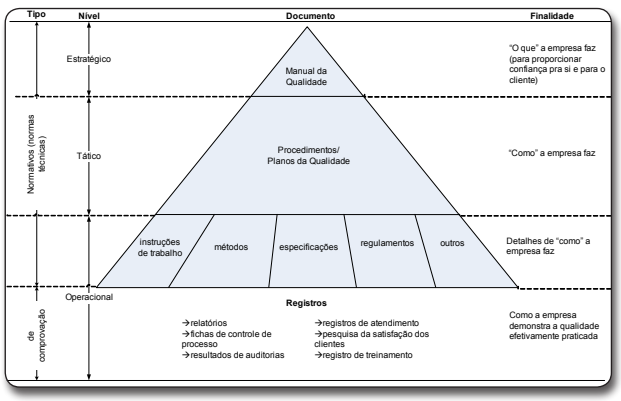


Figura 3. Triângulo da Documentação de um SGQ (MARANHÃO, MAURITI, 2006)

No aspecto geral podemos classificar a documentação de um SGQ nos seguintes tipos: Normativos, Comprobatórios e Documentos não-normativos internos ou externos.

- Normativos: são aqueles que definem como as atividades são executadas. É o planejamento, a previsão de como os processos devem ser realizados. Pode ser dividido em:
  - Normativos internos: normas geradas e aprovadas internamente à organização;
  - Normativos externos: normas geradas e aprovadas externamente ao escopo do SGQ e de uso obrigatório, como por exemplo, Leis. Essas normas têm que ser controladas internamente, mas a organização não tem autoridade para modificá-las.
- Comprobatório: são registros ou documentos não-normativos destinados à comprovação da realização das atividades. Os registros formam toda base documental ou evidências de comprovação para qualquer finalidade;
- Documentos não-normativos internos ou externos: planos, projetos, atas, relatórios, comunicação interna e externa, correspondência, etc. Esses documentos podem se transformar

Convencimento da direção	Refere-se ao envolvimento da alta administração perante o Sistema de Qualidade. Como principais exigências para o cumprimento desse item, a empresa deve definir um membro da direção que irá responder pelo Sistema da Qualidade e por uma Política de Qualidade, que representará formalmente o compromisso da empresa com a qualidade.
Avaliação da situação atual da empresa	Tem por objetivo comparar as situações atual e desejada, de forma a gerar tensão estrutural que impulsionará a mudança organizacional. Torna-se necessário mapear a organização como ela é (As-Is), identificando qual é o problema do processo para modelar como ela deverá ser (To-Be), para apresentar um mapa de "Como" o problema será resolvido ou da implantação do novo processo. Desta maneira, a realização de uma mudança organizacional significativa precisa de um profundo conhecimento das atividades que constituem os processos essenciais de uma organização e os processos que os apóiam, em termos de seus objetivos, pontos de início, entradas, saídas e influências limitadoras. Este entendimento pode ser melhor alcançado pelo "mapeamento","modelagem" e medida dos processos.
Gerenciamento do tempo	Essa área abrange a definição, prosseguimento, estimativa de duração, desenvolvimento de cronograma e controle de atividades. Tempo é elemento essencial ao projeto e ao seu sucesso.
Elaborar o planejamento estratégico	Detalhado na introdução desse artigo.
Unificação conceitual	Tem por objetivo a disseminação da informação para cada nível hierárquico, esclarecendo sobre a intenção da empresa implementar um SGQ ISO 9001.
Mapear e modelar os processos	Busca entender os processos de negócios existentes e futuros para criar melhor satisfação do cliente e melhor desempenho de negócios.
Formar e implementar os grupos de trabalho	Na implantação do programa da ISO 9000 a liderança é fundamental, mas a participação de todos é indispensável. É necessário que não apenas o diretor convença a si mesmo que as mudanças são necessárias, mas que convença toda a empresa a aderir a essa idéia, o que é um trabalho muito árduo. Para envolver os colaboradores podem ser criados grupos de trabalho, com um grupo de coordenação onde participam pessoas chave da empresa e grupos de trabalho com alguns membros de cada departamento. Podem ser elaborado planos de ação para os grupos e organizar as tarefas a serem realizadas de forma simples e objetiva.
Realizar o Housekeeping	Ter padrões aceitáveis de organização: higiene e limpeza (housekeeping) são pré-requisitos desejados para iniciar a qualidade em uma empresa. É adequado ter o mínimo de condições ambientais para se falar em qualidade.
Elaborar o manual da qualidade	Detalhado abaixo.
Elaboração dos documentos que irão compor o SGQ	Detalhado abaixo.
Realizar auditorias internas	Tem por objetivo melhorar o SGQ mediante a identificação de oportunidades de melhorias.
Realizar treinamento de suporte	A empresa deve identificar e providenciar treinamentos específicos para suprir as dificuldades técnicas específicas das atividades mais complexas
Implementação de um processo de análise e melhoria	Estabelecido na fase "C" e "D" do ciclo PDCA. Ciclo de Demimg, indicando as iniciais das palavras inglesas Plan (planejar), Do (fazer), Check (verificar) e Action (fazer). A seção 8 da norma ISO 9001:2000 estabelece a base para os requisitos da fase "C" e "D" do ciclo PDCA.
Atividades que envolvem a auditoria externa.	Contrato com Organismo Certificador Credenciado – OOC. Realizar a pré-auditoria. Executar as ações corretivas e ações preventivas. Realizar auditoria de certificação. Realizar a manutenção do SGQ.

Tabela 2. Passos comuns para a implementação de um SGQ.

em registros, caso venham a ser utilizados para comprovar a realização de uma determinada atividade.

O Manual da Qualidade documenta como a organização funciona e o que ela se propõe a fazer quanto à qualidade, ou seja, o que contratualmente pode ser exigido pelo cliente ou pelos auditores independentes. O ideal é elaborar o manual de forma objetiva e eficaz.

É recomendado que a elaboração do Manual da Qualidade siga a mesma seqüência da NBR ISO 9001.

No meio da pirâmide é destacada a documentação de nível tático, que é composto pelos procedimentos e planos da qualidade. Os procedimentos são documentos que detalham os requisitos do Manual ou da ISO, e explanam como a empresa executa suas atividades para obter a qualidade desejada tanto a nível tático como operacional.

A Figura 4 mostra quais são os procedimentos documentados exigidos pela NBR ISO 9001 que a empresa deve possuir. Além desses procedimentos obrigatórios, a empresa pode elaborar outros de forma a adequar os problemas da necessidade de padronização para seus processos de trabalho, exemplo: comercial, operacional, logística e outros. Em geral, a abrangência e a forma da documentação devem levar em conta:

- O tamanho da organização e suas atividades;
- A complexidade dos processos e suas interações e dos produtos;
- Os requisitos do cliente e de regulamentos aplicáveis a empresa.

Não há regras fixas para a criação de um Sistema de Gestão da Qualidade. Cada empresa deve criar seu próprio sistema para disciplinar e facilitar a vida. Em geral, para empresas menores os procedimentos documentados de segundo nível, que documentam as atividades juntamente com o Manual

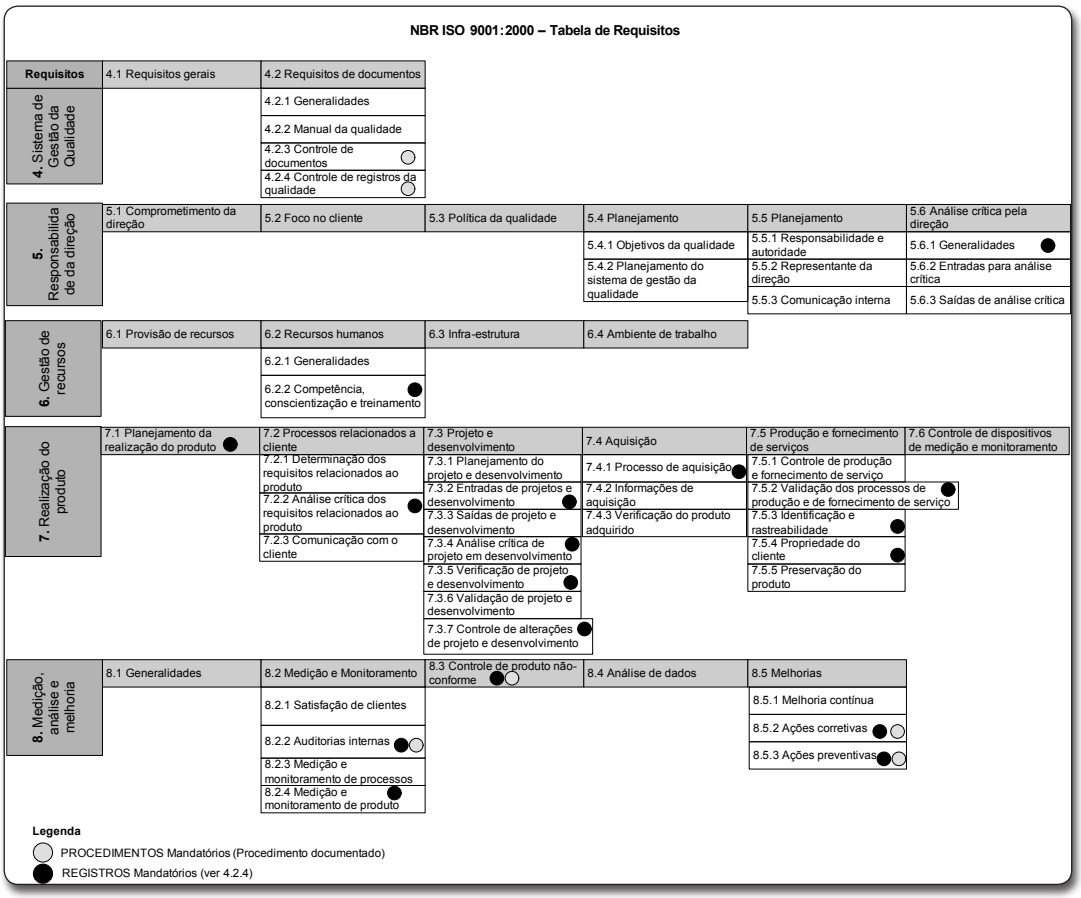


Figura 4. Tabela de requisitos NBR ISO 9001:2000

da Qualidade, são suficientes para compor a documentação do SGQ.

Para empresas maiores e mais complexas, é conveniente detalhar os procedimentos em documentos normativos de terceiro nível, chamados operacionais, também conhecidos como: Instruções de trabalho, Rotina, Métodos, Regulamentos internos, etc.

Para concluir a documentação necessária para o Sistema de Gestão de Qualidade temos os Registros da Qualidade, que são os documentos comprobatórios oriundos de qualquer nível hierárquico da organização. Esses registros englobam todos os documentos que demonstram, por informações e dados, a qualidade praticada na empresa. Também servem como histórico do que efetivamente foi realizado.

Como controlar o Sistema de Gestão da Qualidade

Para manter o Sistema de Gestão da Qualidade funcionando, utilizam-se sensores e atuadores com o objetivo de subsidiar o ciclo de melhorias contínuas.

Os sensores representam as medidas e verificações que a organização faz sobre seus processos. Sem medidas é pouco provável que possa haver controle das ações, com isso a empresa fica incapaz de perceber os riscos e ameaças como, por exemplo, o baixo grau de qualidade dos produtos que gera insatisfação dos clientes. Os atuadores são mecanismos que vão permitir à empresa reagir tão logo ela perceba que algo não vai bem, indicado pelos sensores.

Como sensor do SGQ pode ser utilizado as auditorias internas da qualidade. Os atuadores são as ações corretivas, ações preventivas e oportunidades de melhoria executadas de forma que haja evidência objetiva que foram eliminadas as causas das não-conformidades.

As auditorias internas devem acontecer em datas especificas, não ultrapassando 6 meses. É indicado criar um calendário anual, e o mesmo deve seguir a risca os fundamentos dos procedimentos internos e ter como base o atendimento a Norma ISO 9001.

Ao ser realizada a auditoria interna deverá ser considerado como não conformidade critérios que não atendam as especificações do produto e ou serviço, e ou não atendam os procedimentos internos da empresa, baseado na Norma ISO.

As auditorias internas são documentadas por relatórios, dos quais devem ser derivadas as ações corretivas para as não-conformidades relevantes.

Todas as não conformidades devem ser registradas em documento padrão estabelecido no SGQ da empresa, porém devem ser avaliadas junto ao setor ou responsável, verificando realmente a procedência.

Depois de concluído esta etapa deve ser definida uma data específica para ação corretiva. Na data agendada verifica-se da

ação corretiva: a eficácia e implantação, e após esta se conclui a não conformidade. Por isso podemos dizer que as ações corretivas têm que ter resultados práticos no prazo estipulado.

No caso das ações corretivas surgirem sem auditorias internas, as ações corretivas funcionam para eliminar causas de não-conformidades ou situações indesejáveis detectadas na organização.

Quando o documento já tem algum tempo de aplicação, ele requer revisões, cujas frequências são proporcionais à capacidade de inovação da empresa. Em tese, tais revisões refletem que o Sistema de Gestão da Qualidade funciona e também que a empresa é dinâmica, evolui e introduz melhorias.

Conclusão

A implantação de um Sistema de Gestão da Qualidade começa pela definição e implantação de um conjunto de processos da organização. Os processos devem ser documentados, compreendidos e seguidos para que tornem possível a obtenção de produtos de qualidade.

As normas ISO 9000 surgiram para criar uma linguagem comum no que diz respeito a sistemas de gestão da qualidade. A série ISO 9000 é um conjunto de cinco normas que podem ser divididas em diretrizes e normas contratuais. Estas normas garantem que os produtos fabricados por um processo certificado tenham sempre a mesma qualidade. O fato de o processo ser certificado segundo as normas ISO 9000 não acrescenta qualidade aos produtos.

A principal vantagem das normas é a documentação do sistema da qualidade. Esta documentação pode ser dividida nos documentos da qualidade que descrevem o processo e nos registros da qualidade que registram os resultados do processo. É também nesta documentação que são detectadas a maior parte das não-conformidades às normas.

Auditoria interna é a base para o desenvolvimento da melhoria contínua da qualidade. Cada auditor deve ter curso de auditor interno e conhecimentos básicos nos processos em que o mesmo irá atuar.

Uma empresa só pode ser certificada em relação às normas contratuais (9001, 9002 e 9003). A certificação serve para demonstrar que a empresa possui capacidade de fornecer produtos que atendam aos requisitos do cliente e aos requisitos regulamentares. Porém, fazer da certificação o grande objetivo da implementação do SGQ normalmente conduz a um desastre a longo prazo.

O alvo para a implementação do Sistema de Gestão da Qualidade deve ser a melhoria da qualidade e da competitividade. O resultado de um bom Sistema de Gestão da Qualidade é o aumento da lucratividade e abrir perspectivas para o crescimento sustentado da organização. É por esse motivo que a certificação deve ser uma conseqüência natural da melhoria da empresa e não o foco do projeto. ●

Referências

NBR ISO 9000:2000, Sistema de gestão da qualidade – Fundamentos e vocabulário.

NBR ISO 9001:2000, Sistema de gestão da qualidade – Requisitos.

NBR ISO 9004:2000, Sistema de gestão da qualidade – Diretrizes para melhorias de desempenho.

Maranhão, Mauriti. ISSO série 9000: versão 2000: Manual de implementação: O passo a passo para solucionar o quebra cabeça da gestão. 8° ed – Rio de Janeiro : Qualitymark, Ed., 2006.

Confiabilidade de Software

Etapas na Determinação da Confiabilidade de Software

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

Incerteza é um estado com o qual as pessoas se deparam no cotidiano. Se considerarmos um projeto de software, encontramos incerteza quando se verifica o prazo e/ou custo para concluir o projeto. Também há incerteza quanto ao número de faltas ou defeitos remanescentes no software. Por exemplo, um produto resultante do projeto de um sistema de software pode ter uma confiabilidade de 0.999, que pode ser considerado como um nível satisfatório de confiabilidade (isto é, que um sistema estaria operacional e funcionando corretamente durante 99,9% do tempo). Entretanto, ainda assim, não há a certeza de que o produto não apresentará falhas. E, caso o software não tenha sido exaustivamente testado, o que se pode obter é um indicador da confiabilidade do software, geralmente, definido em termos de uma medida estatística da operação de um software sem a ocorrência de falhas.

Neste artigo veremos

Apresenta a confiabilidade de software, destacando-a como principal atributo da qualidade de sistemas críticos. Mostra como considerar e determinar a confiabilidade de um sistema de software.

Qual a finalidade

Informar quando considerar a confiabilidade de software no desenvolvimento de sistemas críticos que exigem elevado nível de confiabilidade.

Quais situações utilizam esses recursos?

Trata-se de uma prática de engenharia de software considerar a confiabilidade de software durante e após o desenvolvimento de um sistema de software. Permite ao engenheiro de software avaliar o nível de confiabilidade de um software.

A confiabilidade de software é um dos atributos da qualidade de software. Além dela, outros atributos da qualidade são desempenho, portabilidade, reusabilidade, dentre outros. Desconsiderar a confiabilidade de software durante o desenvolvimento de um sistema de software implica na possível ocorrência de falhas durante a fase operacional do sistema. Este artigo trata da importância da confiabilidade de software para os sistemas.

Confiabilidade de Software em Sistemas Computacionais

O ciclo de vida de um sistema de software consiste de duas grandes fases: desenvolvimento e operação. A fase de desenvolvimento compreende engenharia de sistemas, especificação de requisitos, projeto, codificação e testes. A fase operacional refere-se à fase na qual o sistema de software está em uso, oferecendo funcionalidades aos usuários.

Tradicionalmente, a maioria dos esforços da engenharia de software em produzir software confiável tem sido concentrada na fase de desenvolvimento. A razão para isto recai no fato da natureza inerente do software de não sofrer qualquer desgaste, isto é, por ser um produto intangível. Diferentemente do hardware que sofre desgaste, o software não é susceptível a esse tipo de problema.

Todavia, software é susceptível a erros que podem provocar a ocorrência de falhas. Portanto, é de suma importância considerar a confiabilidade de software no desenvolvimento de sistemas. A confiabilidade de software é definida como a probabilidade do sistema funcionar sem ocorrência de falhas num período e ambiente especificados.

Mas, o que vem a ser uma falha?

Uma falha é uma ocorrência quando um serviço ou funcionalidade entregue por um software não está mais em conformidade com a especificação.

Uma falha também é caracterizada pela incapacidade do sistema ou componente deste em prover funcionalidade desejada em determinado ambiente, num período de tempo especificado. Isto é geralmente considerado como um desvio entre o comportamento desejado e comportamento especificado. As falhas (observadas por usuários) são causadas por faltas. Uma falta é comumente referida como um bug ou defeito no software. Uma falta existe quando, para algum dado de entrada, o comportamento do sistema está incorreto, isto é, seu comportamento é diferente daquele descrito na especificação do software.

As faltas podem ser categorizadas em termos de onde, quando e o que a provoca. Perceba que uma falta pode ser interna ao software ocorrendo durante o ciclo de vida do mesmo. Assim, ela pode ocorrer durante a especificação, projeto, implementação ou testes. Além disso, uma falta pode acontecer devido a um erro humano durante a especificação ou projeto de software, como também pode ser causada pelo ambiente no qual o software é produzido. Neste último caso, acontecendo se o código gerado pelo compilador não está em conformidade com o projeto.

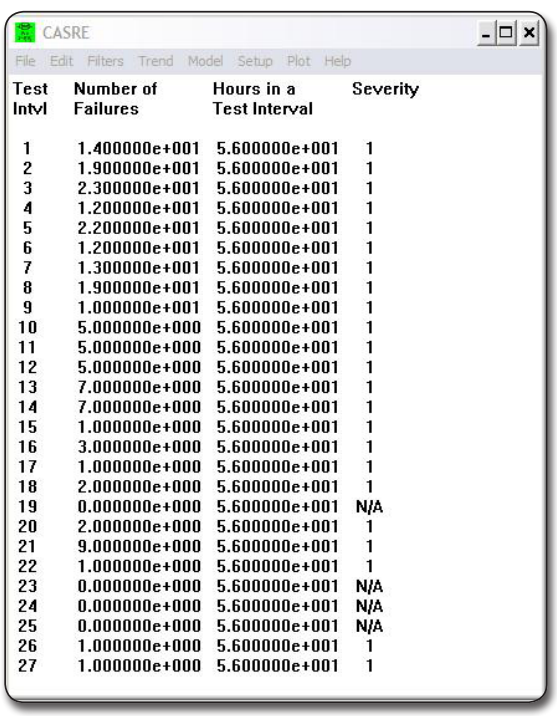
A importância da confiabilidade de software em sistemas computacionais se deve ao fato dela ser um atributo de qualidade essencial e determinante do sucesso do produto. Em sistemas críticos como, por exemplo, sistemas de controle de tráfego aéreo e sistemas de controle de metrô, determinadas falhas podem ser fatais. Em razão deste fato, tais sistemas exigem um nível elevado de confiabilidade, o que implica em custos elevados. Mecanismos para elevar a confiabilidade de software em tais sistemas englobam testes intensivos durante o desenvolvimento, alem da redundância de código e de hardware. Adicionalmente, outras técnicas como recovery blocks e N-version programming podem ser empregadas para aumentar a confiabilidade de sistemas.

Note que, para sistemas não críticos, como uma central telefônica, a ocorrência de poucas falhas é geralmente tolerável, pois o interesse maior está na qualidade de serviço. Aqui, pode-se questionar qual seria a taxa de falhas aceitável para os clientes. Se um cliente tenta efetuar uma ligação e não obtém o tom de discar em 10 dentre 10000 chamadas, isto pode ser tolerável.

Há quatro formas de categorizar as ocorrências de falhas de um sistema ao longo do tempo:

- Tempo de falha;
- Intervalo de tempo entre falhas;
- Falhas cumulativas observadas em determinado período de tempo;
- Falhas observadas num intervalo de tempo.

Considere, por exemplo, os dados apresentados na Figura 1 que contém dados de falhas para um conjunto de intervalos de teste onde é contabilizado o número de falhas e quantitativo de horas no intervalo considerado. Estes dados foram apresentados fazendo uso da ferramenta CASRE (Computer Aided Software Engineering Estimation), cujo link encontra-se no quadro final deste artigo. Esta ferramenta serve de apoio à medição da confiabilidade de software.



Test Intvl	Number of Failures	Hours in a Test Interval	Severity
1	1.400000e+001	5.600000e+001	1
2	1.900000e+001	5.600000e+001	1
3	2.300000e+001	5.600000e+001	1
4	1.200000e+001	5.600000e+001	1
5	2.200000e+001	5.600000e+001	1
6	1.200000e+001	5.600000e+001	1
7	1.300000e+001	5.600000e+001	1
8	1.900000e+001	5.600000e+001	1
9	1.000000e+001	5.600000e+001	1
10	5.000000e+000	5.600000e+001	1
11	5.000000e+000	5.600000e+001	1
12	5.000000e+000	5.600000e+001	1
13	7.000000e+000	5.600000e+001	1
14	7.000000e+000	5.600000e+001	1
15	1.000000e+000	5.600000e+001	1
16	3.000000e+000	5.600000e+001	1
17	1.000000e+000	5.600000e+001	1
18	2.000000e+000	5.600000e+001	1
19	0.000000e+000	5.600000e+001	N/A
20	2.000000e+000	5.600000e+001	1
21	9.000000e+000	5.600000e+001	1
22	1.000000e+000	5.600000e+001	1
23	0.000000e+000	5.600000e+001	N/A
24	0.000000e+000	5.600000e+001	N/A
25	0.000000e+000	5.600000e+001	N/A
26	1.000000e+000	5.600000e+001	1
27	1.000000e+000	5.600000e+001	1

Figura 1. Dados de contagem de falhas.

Para os dados apresentados na Figura 1, as quatro colunas mostram o número seqüencial de intervalo de teste, o número de falhas observadas no intervalo de teste especificado, período de tempo do intervalo de teste considerado e grau de severidade do intervalo de teste em questão, respectivamente.

Vale observar que a ferramenta CASRE permite que os dados mostrados na Figura 1 sejam exibidos na forma gráfica, como ilustrado na Figura 2, que retrata o quantitativo de falhas para o conjunto de intervalos de teste considerados.

Por outro lado, se desejarmos obter o gráfico da quantidade cumulativa de falhas (isto é, somando as quantidades de falhas observadas em cada intervalo de teste para os dados apresentados na Figura 1), selecionamos a referida opção na ferramenta e o resultado é exibido na Figura 3.

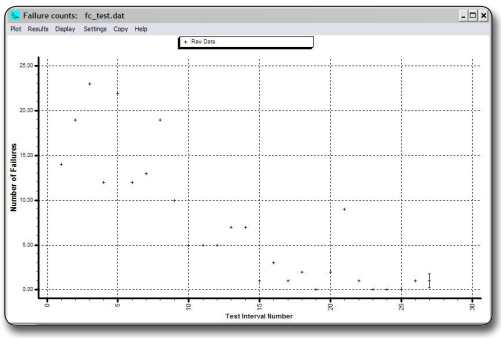


Figura 2. Número de falhas em função dos intervalos de testes.

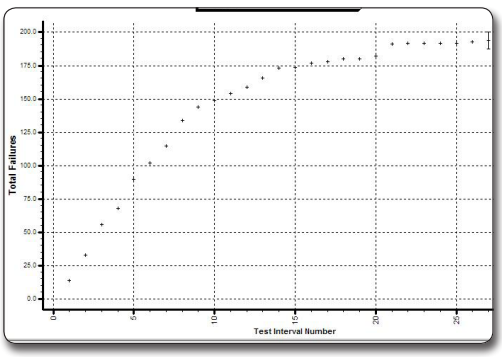


Figura 3. Número cumulativo de falhas em função dos intervalos de testes.

Determinando a Confiabilidade de Sistemas

A confiabilidade é comumente definida em termos de uma medida estatística de um sistema de software na fase operacional quando não ocorre qualquer falha. Em outras palavras, ela é uma medida da probabilidade de ocorrência de falha de software. Intuitivamente, podemos tentar expressar a confiabilidade como:

$$(1) \quad R = 1 - \frac{T_i}{T_t}$$

Onde R é confiabilidade, Ti compreende o valor cumulativo de intervalos de tempo de um sistema qualquer considerado e Tt o registro total de tempo de execução de um sistema.

Agora, considere que as entradas para um determinado software acontecem de forma contínua ao longo do tempo. Assim, se denominarmos p como a probabilidade da ocorrência da primeira falha após n unidades de tempo, então a confiabilidade de um sistema no tempo (Rd(k)) pode ser expressa como a probabilidade de nenhuma falha ocorrer durante a execução que compreende k entradas ou

$$(2) \quad R_d(k) = (1-p)^k$$

Se chamarmos de te o período de tempo de execução para uma dada entrada e considerarmos que esse tempo seja o mesmo para qualquer entrada, então podemos dizer que t = kte. Supondo que existe um limite finito para p/te, onde te tende a 0 (zero), então a intensidade de falhas λ é dada por:

$$(3) \quad \lambda = \lim_{t_e \to 0} (p/t_e)$$

O processo de determinação da confiabilidade considera que as entradas de um determinado sistema acontecem em tempos aleatórios, onde algumas entradas podem provocar falhas enquanto outras não.

Todavia, quando as entradas para um software acontecem de modo contínuo ao longo do tempo, então a probabilidade da ocorrência da primeira falha após n unidades de tempo pode ser expressa através de uma distribuição exponencial. Isso resulta na confiabilidade de software contínua no tempo expressa como:

$$(4) \quad R(t) = \lim_{t_e \to 0} R_d(k)$$

ou

$$(5) \quad R(\lambda) = e^{-\tau\lambda}$$ se considerarmos que a intensidade de falhas λ como constante.

Cabe salientar que τ, na expressão (5), corresponde ao tempo de execução enquanto que **t**, na expressão (4), corresponde ao tempo de calendário.

Note que a intensidade de falha é um número que expressa a quantidade de falhas observadas numa determinada unidade de tempo. Por exemplo, a intensidade de falhas poderia ser de 0.5 falhas/hora. A Figura 4 ilustra a intensidade de falhas para os dados apresentados na Figura 1. Perceba nos dados mostrados que o quantitativo de falhas tende a diminuir, exceto no ponto relativo ao intervalo de teste 21, conforme ilustrado na Figura 4.

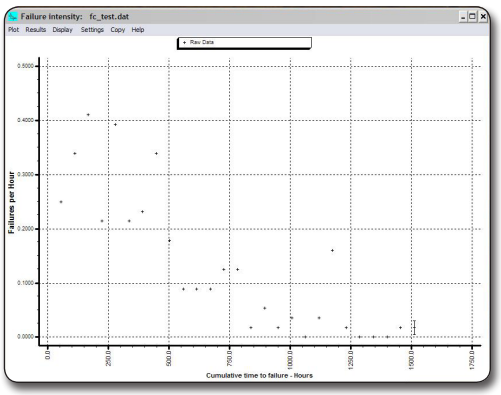


Figura 4. Intensidade de falhas.

Seja μ (t) o número total médio de falhas observadas em função do tempo de execução τ. Aqui, a média é tomada, pois se considera n instâncias independentes de um mesmo software.

Vale lembra que a intensidade de falhas λ(t) compreende o número de falhas por unidade de tempo, a qual pode ser obtida pela derivada de μ (t). O tempo médio entre falhas, também conhecido como MTTF (Mean Time To Failure) é então obtido através da expressão:

$$(6) \quad MTTF = \frac{1}{\lambda(t)}$$

Agora, vamos supor que um sistema de software é composto por quatro componentes, conforme mostrado na Figura 5.

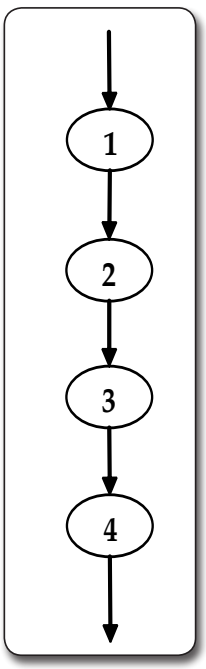


Figura 5. Exemplo de sistema composto de 4 componentes em série.

Como os componentes estão agrupados em série, então a probabilidade de que o sistema funcione sem falhas pode ser, intuitivamente, obtida como o componente 1 está funcionando e o componente 2 está funcionando e o componente 3 está funcionando e o componente 4 está funcionando.

Adicionalmente, se for feita a suposição de que eventuais faltas (que causam falhas) sejam eventos independentes em quaisquer dos componentes, então a confiabilidade do sistema pode ser expressa como:

$$(7) \quad R(t) = \prod_{i=1}^{n} Ri(t)$$

Se, por exemplo, considerarmos que cada componente tem confiabilidade de 0,999, então a confiabilidade resultante será:

$$R = 0{,}999 \times 0{,}999 \times 0{,}999 \times 0{,}999 \cong 0{,}9960$$

Entretanto, se um dos componentes do referido sistema é composto de dois outros componentes em paralelo, como ilustrado na Figura 6, então a confiabilidade é obtida através da expressão:

$$(8) \quad R(t) = 1 - \prod_{n} (1 - Ri(t))$$

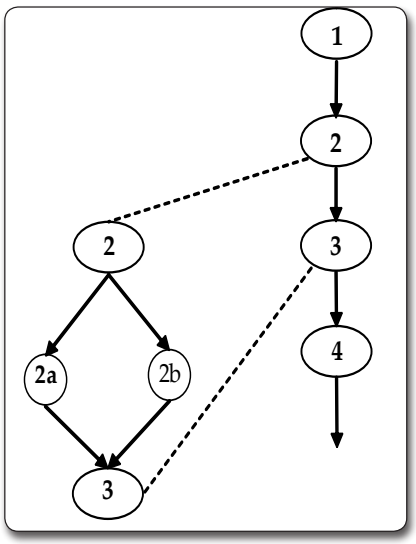


Figura 6. Exemplo de sistema composto de componentes em série e paralelo.

Aqui também é feita a suposição que há a independência de falhas entre os componentes do sistema. Desse modo, para determinarmos a confiabilidade do componente 2, vamos supor que a confiabilidade do componente 2a é 0,99 e que a do componente 2b é 0,90. Utilizando a expressão (8), obtemos:

$$R2 = 1 - (1 - 0{,}90) \times (1 - 0{,}90) = 0{,}99$$

Com isso, a confiabilidade do componente 2 obtida é 0,99 e não mais 0,999 como no exemplo anterior. Assim, a confiabilidade resultante do sistema mostrado na Figura 6 resulta em:

$$R = 0{,}999 \times 0{,}99 \times 0{,}999 \times 0{,}999 \cong 0{,}987033$$

Modelos de Crescimento da Confiabilidade de Software

A confiabilidade de software é um atributo essencial da qualidade de software de sistemas críticos e de sistemas que exigem elevado grau de disponibilidade, na qual as ocorrências de falhas podem ocasionar perdas de vidas, prejuízo financeiro ou indisponibilidade essencial de serviços.

Com o objetivo de determinar se um determinado nível de confiabilidade foi alcançado, podem-se utilizar modelos de crescimento da confiabilidade de software que visam prever a confiabilidade de software. Em geral, esses modelos consideram a observação da ocorrência de falhas e a partir dessa informação, tentam prever o comportamento futuro da ocorrência de falhas. Dois modelos conhecidos são o modelo de Musa e o modelo Musa/Okumoto.

O modelo de Musa faz suposições de que:

1. As faltas são independentes e distribuídas de maneira uniforme;

2. O conjunto de entradas para cada execução é selecionado de maneira aleatória;
3. Todas faltas que provocam falhas são imediatamente corrigidas; dessa forma, a re-ocorrência de falhas não é considerada.
4. O decremento da função de intensidade de falhas é constante.

Como resultado, tem-se que a intensidade de falha é função do número médio de falhas observadas num determinado tempo, sendo dada pela expressão:

$$(9) \qquad \lambda(\mu) = \lambda_0(1 - (\mu \ / \ v_0))$$

Onde:
$\lambda(\mu)$ – intensidade de falhas
λ_0 – intensidade inicial de falhas (no início da execução)
μ – número médio de falhas em determinado instante de tempo
v_0 – número total de falhas ao longo do tempo

Considere, por exemplo, o conjunto de dados apresentados na Figura 7. Esses dados de exemplo foram fornecidos junto com a ferramenta CASRE (arquivo tbe_nhpp.dat).

CASRE

File Edit Filters Trend Model Setup Plot Help

Error No.	Hours Since Last Failure	Severity
1	6.672230e-001	1
2	1.337794e+000	1
3	1.342283e+000	1
4	1.346803e+000	1
5	1.351353e+000	1
6	1.355934e+000	1
7	1.360546e+000	1
8	1.365189e+000	1
9	1.369864e+000	1
10	1.374572e+000	1
11	1.379312e+000	1
12	1.384084e+000	1
13	1.388890e+000	1
14	1.393730e+000	1
15	1.398603e+000	1
16	1.403510e+000	1
17	1.408452e+000	1
18	1.413429e+000	1
19	1.418441e+000	1
20	1.423489e+000	1
21	1.428573e+000	1
22	1.433693e+000	1
23	1.438850e+000	1
24	1.444045e+000	1
25	1.449277e+000	1
26	1.454547e+000	1
27	1.459856e+000	1

Figura 7. Dados de falhas (arquivo tbe_nhpp.dat da ferramenta CASRE).

As Figuras 8, 9 e 10 ilustram o tempo entre falhas, a intensidade de falhas e a confiabilidade estimada usando o modelo de Musa, respectivamente. Esses gráficos foram obtidos a partir dos dados de falhas apresentados na Figura 7. Perceba que a Figura 7 mostra em sua segunda coluna o quantitativo de horas desde a última falha observada. Note que esse número de horas (desde a última falha observada) tem uma variação mínima e apenas nas últimas observações torna-se um pouco crescente, como retratado na Figura 8 que ilustra o tempo entre falhas da Figura 7. Em conseqüência, o quantitativo de falhas ao longo do tempo (ou seja, a intensidade de falhas) começa a decrescer e a confiabilidade aumenta, como mostrado, respectivamente, nas Figura 9 e 10.

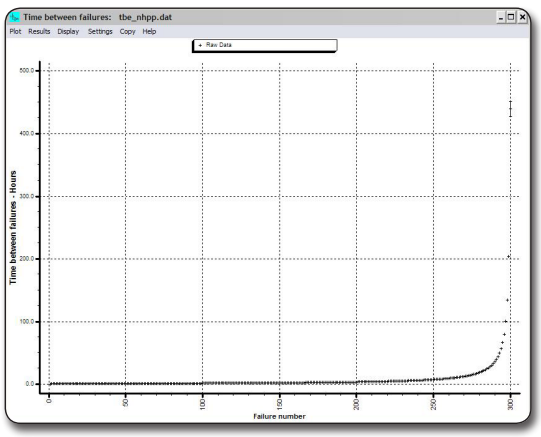


Figura 8. Tempo entre falhas para dados da Figura 7.

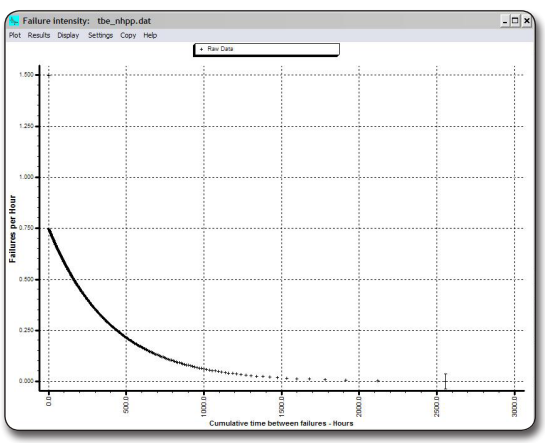


Figura 9. Intensidade de falhas para dados da Figura 7.

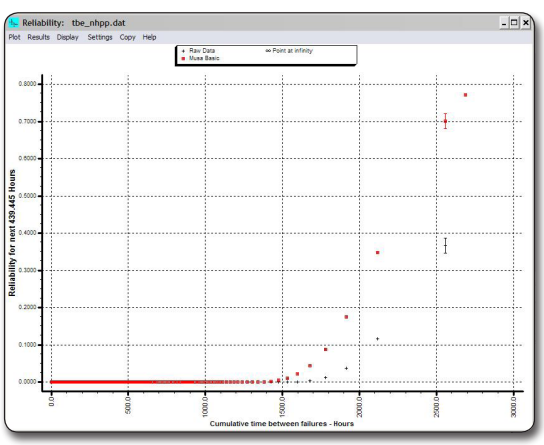


Figura 10. Confiabilidade usando modelo de Musa para os dados da Figura 7.

Os dados apresentados na Figura 7 apontam para uma redução na quantidade de falhas, como observado na Figura 9, que retrata a intensidade de falhas, e na melhoria da confiabilidade conforme apresentado na Figura 10.

Por outro lado, o modelo de Musa/Okumoto pode ser considerado como extensão do modelo de Musa e considera um parâmetro de decaimento θ da intensidade de falha, o qual implica num decréscimo sobre a quantidade de falhas encontradas. Como resultado, tem-se que a intensidade de falha é também em função desse parâmetro de decaimento θ, sendo dada pela expressão:

(10) $$\lambda(\mu) = \lambda_0 \, e^{-\theta\mu}$$

Onde:

$\lambda(\mu)$ – intensidade de falhas

λ_0 – intensidade inicial de falhas (no início da execução)

μ – número médio de falhas em determinado instante de tempo

θ – parâmetro de decaimento da intensidade de falha

Usando a ferramenta CASRE, é possível obter a confiabilidade estimada, utilizando-se o modelo Musa/Okumoto, conforme ilustrado na Figura 11. Observe que o gráfico do modelo de Musa retrata bem os dados apresentados na Figura 7, pois há um decremento na intensidade de falhas que ocorre de forma contínua e que está em conformidade com o modelo.

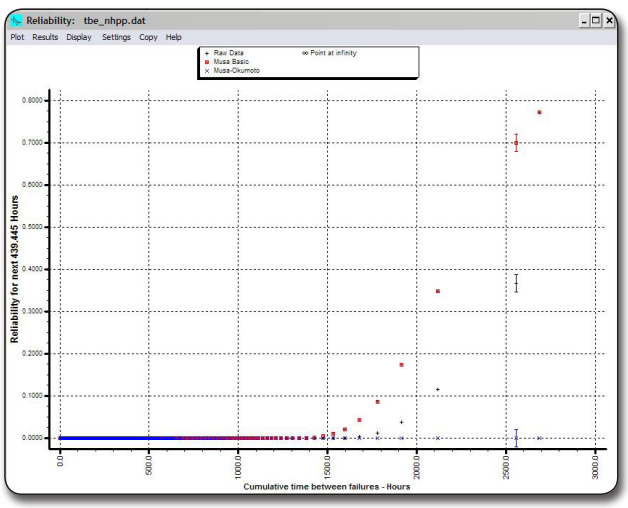


Figura 11. Confiabilidade estimada usando modelo de Musa/Okumoto para dados da Figura 7.

Após determinar a confiabilidade, deve-se verificar se a confiabilidade alcançada satisfaz ou não a meta definida de confiabilidade. Se o nível de confiabilidade é atendido, então o sistema está pronto para entrar em uso. Caso contrário, mais faltas devem ser removidas. Esse processo de determinação e avaliação da confiabilidade de software é ilustrado na Figura 12.

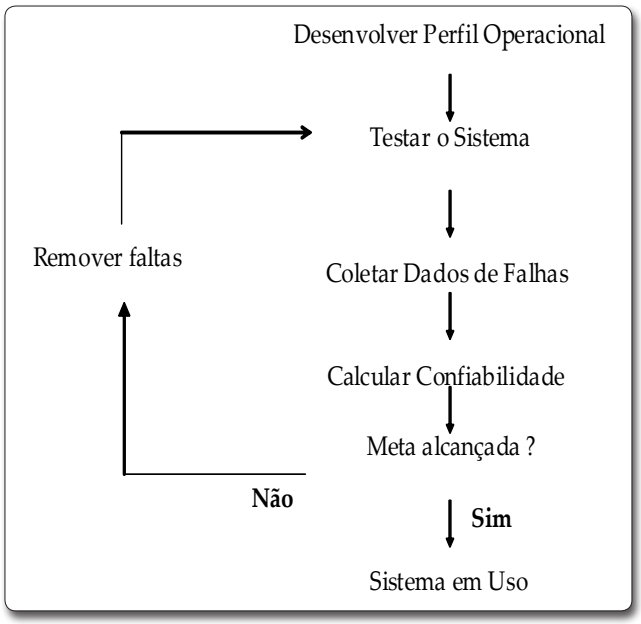


Figura 12. Processo de determinação e avaliação da confiabilidade de software.

Observe que, após usar um modelo de estimativa de crescimento da confiabilidade como, por exemplo, o modelo de Musa, é possível verificar se o sistema está pronto ou não para entrar em operação. Entretanto, para usar quaisquer desses modelos, torna-se necessário a coleta de dados de falhas, conforme ilustrado na Figura 7. Esses dados são produzidos com os testes realizados num sistema em desenvolvimento (já que esta é a ultima fase do desenvolvimento antes da implantação).

Outra atividade desse processo que oferece informação às atividades de testes é o desenvolvimento de um perfil operacional do sistema. Um perfil operacional é um conjunto completo de operações com suas respectivas probabilidades de ocorrência. Por exemplo, uma probabilidade de ocorrência de 0,20 para o encaminhamento de uma ligação telefônica significa que 20 dentre 100 chamadas de operações serão encaminhamento de ligação telefônica.

Considere a Figura 13 que ilustra um exemplo de dados de perfil operacional para um sistema qualquer com três funções F1, F2 e F3. Perceba que para essas funções, existe um conjunto de operações associadas a elas, denominadas de O11, O12 e O13 para uma função F1, O21 e O22 para uma função F2, O31 e O33 para uma função F3. Para cada uma dessas operações, a última coluna informa sua respectiva probabilidade de ocorrência, conforme apresentado na Figura 13.

Cabe destacar que a probabilidade informada na Figura 13 tem apenas a finalidade de exemplificar como poderia ser especificado um perfil operacional de um sistema. Considere, por exemplo, uma probabilidade de ocorrência de 0,15 da operação O12 a qual indica que, para esta operação, 15 dentre 100 chamadas desta operação serão adequadamente tratadas.

Modo	Função	Operação	Probabilidade de ocorrência
Normal	F1	O11	0.3
		O12	0.15
		O13	0.1
	F2	O21	0.20
		O22	0.15
	F3	O31	0.05
		O33	0.03

Figura 13. Exemplo de perfil operacional.

Comentários Finais

Confiabilidade de software é um atributo essencial e determinante de sucesso de sistemas críticos, além de ser qualidade observada pelos usuários que são afetados quando os sistemas não são confiáveis. Não considerar a confiabilidade no processo de desenvolvimento de sistemas, principalmente de natureza crítica, pode implicar em situações indesejáveis onde se pode ter perdas econômicas grandes e até perdas de vidas humanas. Este artigo caracterizou a confiabilidade e, adicionalmente, mostrou como determiná-la em alguns exemplos, destacando dois modelos de crescimento da confiabilidade de software.

Links

História da Indústria de Software
www.softwarehistory.org

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

Computer Aided Software Reliability Estimation (CASRE) http://www.openchannelsoftware.com/projects/CASRE_3.0

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

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

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

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