engenharia de software magazine

Edição 17 :: Ano 2

DevMedia group

ISSN 1983127-7

Processo
Veja como o Eclipse Process Framework Composer apóia adaptações de processo de desenvolvimento de software

Gerência de Riscos
Saiba como utilizar mapas conceituas para identificar riscos associados aos requisitos

Engenharia de Software
Aprenda um pouco mais sobre Governança de Tecnologia de Informação com ITIL

Projeto
Aplique MDD na prática

Predição de defeitos em Software

Conheça algumas métricas que podem ser coletadas diretamente de projetos, assim como algumas das técnicas mais utilizadas nos últimos anos para a criação de modelos de predição de defeitos

Projeto
Aprenda a criar protótipos de interfaces com a ferramenta Axure

Projeto
Conheça os Diagramas de Objetos, Máquina de Estados e de Atividades da UML

Aulas desta edição:

- Definições Iniciais sobre Validação, Verificação e Testes – Partes 4 a 8
- Diagrama de Classes na Prática – Partes 1 a 3

Conhecimento faz diferença!

Mais de 60 mil downloads na primeira edição!

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

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

DevMedia GROUP

engenharia de software magazine

Ano 2 - 17ª Edição 2008 **Impresso no Brasil**

EDITORIAL

"A produção de um software de qualidade depende de muitos fatores envolvidos no seu processo de desenvolvimento. Uma das maneiras de contribuir com a qualidade de um produto é incluir atividades de testes de software bem direcionadas e conduzidas durante a sua construção. O processo de testes tem como uma de suas etapas a identificação dos defeitos existentes no produto em análise. Antecipar a localização de defeitos, antes mesmo que ocorram, pode contribuir para o direcionamento das atividades de testes, de maneira que a equipe envolvida nessas atividades possa direcionar seus esforços para áreas do software que apresentam maior propensão a apresentar problemas. Iniciativas neste sentido são muito difundidas na indústria, apresentando diversos casos práticos em empresas como: Microsoft, Ericsson e IBM."

Nesta edição, a Engenharia de Software Magazine destaca uma matéria muito interessante sobre predição de defeitos. "Predição de defeitos é a tentativa de antecipar a localização de defeitos em uma aplicação através do uso de técnicas específicas. Em muitos casos, trata-se da criação de modelos, que analisam dados históricos de um projeto para, a partir daí, tentar adivinhar como se comportarão versões futuras de seu código. Predizer a localização de defeitos em módulos da aplicação aumenta as chances que as atividades de testes têm de encontrá-los. Dessa maneira, ações corretivas podem ser tomadas o quanto antes no desenvolvimento do produto."

Além desta matéria, esta edição traz mais seis artigos:

- Mapas Conceituais: Utilizando-os na Identificação de Riscos de Requisitos
- ITIL - Information Technology Infrastructure Library
- UML – Explorando outros diagramas
- Aplicando MDD na Prática
- Prototipação no Desenvolvimento de Software
- Eclipse Process Framework Composer

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)
Doutor 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 e Coordenador do curso de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora, Professor do curso de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery, Professor e Diretor do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas da Fundação Educacional D. André Arcoverde, Analista de Sistemas da Prefeitura de Juiz de Fora, Colaborador 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 **8 vídeo aulas**. Estas vídeo aulas estão disponíveis para download no Portal da Engenharia de Software Magazine e certamente trarão uma significativa contribuição para seu aprendizado. A lista de aulas publicadas pode ser vista abaixo:

Tipo: Validação, Verificação e Teste
Título: Definições Iniciais sobre Validação, Verificação e Testes – Partes 4 a 8
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Atividades de VV&T são fundamentais ao longo de todo o processo de desenvolvimento de software. Entretanto, ainda é comum o pensamento de que só devemos avaliar a qualidade do software ao final do processo através de atividades de testes. Veremos nesta série de vídeo aulas o porquê desta perspectiva não estar correta e entenderemos os principais conceitos associados às atividades de validação, verificação e testes. Nestas aulas iniciaremos os estudos sobre as atividades de revisões de software.

Tipo: Projeto
Título: Diagrama de Classes na Prática – Parte 1 – Definições Iniciais
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta aula é parte de uma série sobre a construção de diagrama de classes da UML. O objetivo do conjunto de aulas é apresentar de forma prática como elaborar o diagrama de classes a partir de diferentes estudos de caso. Nesta primeira aula, são apresentadas as definições iniciais sobre diagrama de classes. O entendimento delas é fundamental para a continuidade do curso.

Tipo: Projeto
Título: Diagrama de Classes na Prática – Parte 2 – Definições Iniciais
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta aula é parte de uma série sobre a construção de diagrama de classes da UML. O objetivo do conjunto de aulas é apresentar de forma prática como elaborar o diagrama de classes a partir de diferentes estudos de caso. Nesta segunda aula, são apresentadas as últimas definições sobre diagrama de classes. O entendimento delas é fundamental para a continuidade do curso. Além disto, é apresentado o processo que será seguido ao longo do curso para apoiar a construção de diagramas de classes.

Tipo: Projeto
Título: de Classes na Prática – Parte 3 – Estudo de Caso 1
Autor: Rodrigo Oliveira Spínola
Mini-Resumo: Esta aula é parte de uma série sobre a construção de diagrama de classes da UML. O objetivo do conjunto de aulas é apresentar de forma prática como elaborar o diagrama de classes a partir de diferentes estudos de caso. Nesta terceira aula, é dado início à elaboração do diagrama de classes para o primeiro estudo de caso. Nesta etapa inicial, serão identificadas as classes do modelo.

ÍNDICE

08 - Predição de Defeitos em Software
Melissa Barbosa Pontes

14 - Mapas Conceituais
Cristine Gusmão, Kenelly Almeida e Júlio Venâncio

22 - ITIL - Information Technology Infrastructure Library
Monalessa Perini Barcellos e Alex Sandro Barreto Rodrigues

28 - UML – Explorando outros diagramas
Ana Cristina Melo

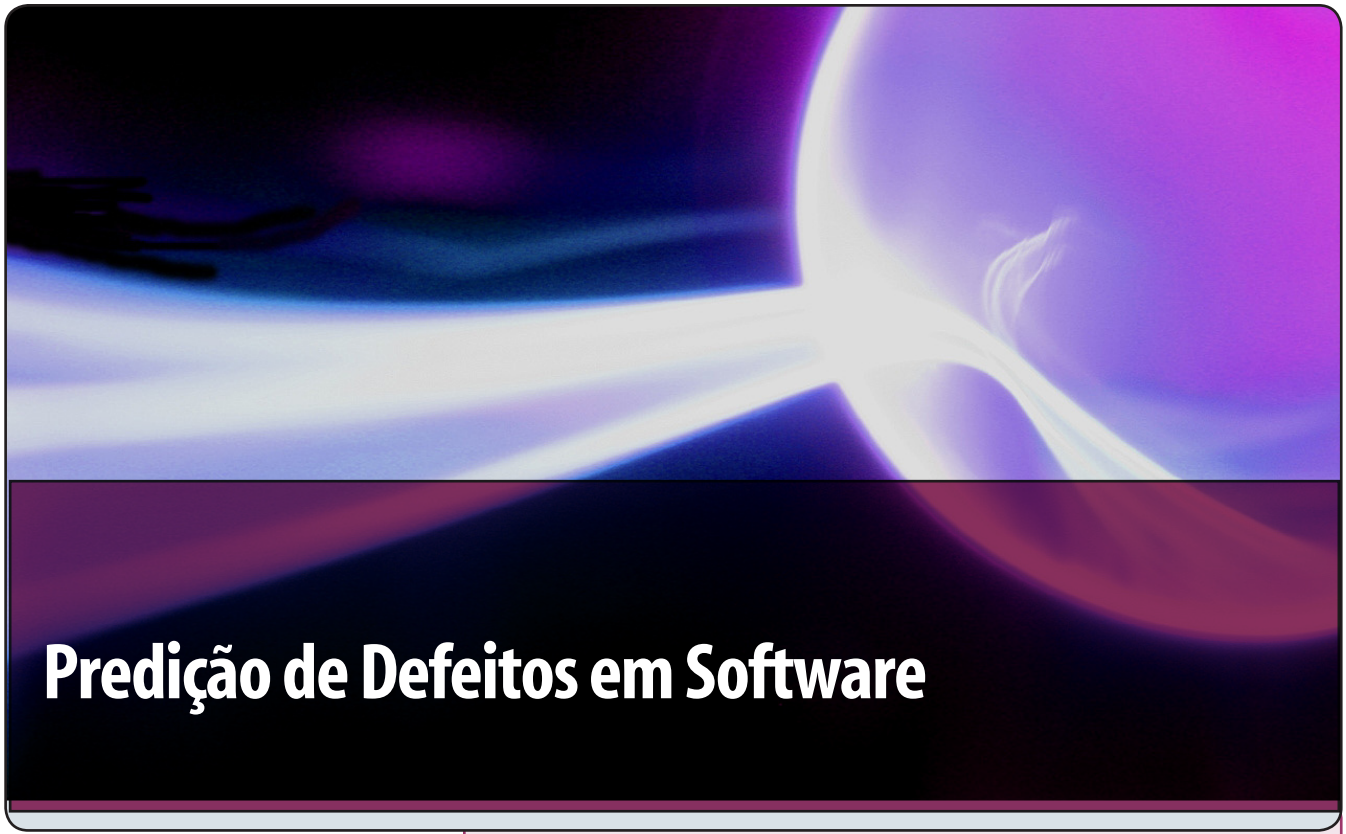
36 - Aplicando MDD na Prática
Denis Silva Loubach

44 - Prototipação no Desenvolvimento de Software
Vinícius Rodrigues de Souza, Ricardo Cunha Vale e Marco Antônio Pereira Araújo

52 - Eclipse Process Framework Composer
Felipe Furtado, Teresa Maria Medeiros Maciel, Andrea Pinto e Elisabeth Morais

59 - Gerenciamento de Configuração
Felipe La Rocca Teixeira e Marco Antônio Pereira Araújo

Você não está mais sozinho.



Predição de Defeitos em Software

De que se trata o artigo?

Este artigo aborda o tema predição de defeitos em software. São apresentadas algumas métricas que podem ser coletadas diretamente de projetos, assim como algumas das técnicas mais utilizadas nos últimos anos para a criação de modelos de predição de defeitos. Apresenta também uma série de dificuldades encontradas na coleta de métricas de projetos reais, que podem inviabilizar a criação de modelos de predição. Por fim, é apresentado um subprocesso que pode ser integrado a processos de desenvolvimento de software, para viabilizar uma boa coleta de métricas.

Para que serve?

Fornecer a gerentes, líderes e demais integrantes de equipes de desenvolvimento e testes de software uma breve introdução sobre o assunto predição de defeitos, tema já bastante discutido e utilizado na academia e na indústria nacional e internacional na atualidade.

Em que situação o tema é útil?

A realização de predição de defeitos em um projeto de software oferece alternativas para um melhor direcionamento das atividades de testes, que pode acarretar em redução de custos neste projeto.

Melissa Barbosa Pontes

melissa.pontes@cesar.org.br

Mestre em Engenharia de Software do Cesar.EDU. Certificada em testes de software pelo ISTBQ (International Software Testing Qualification Board), atualmente trabalha como engenheira de testes no CESAR (Centro de Estudos e Sistemas Avançados do Recife), onde desempenha atividades de definição de processos, liderança e consultoria. É integrante do GrIT (Grupo Independente de Testes do Cesar) através do qual realiza pesquisas e treinamentos em testes na organização. Possui artigos e trabalhos publicados em eventos internacionais. É membro integrante da comissão de organização de EBTS - Encontro Brasileiro de Testes de Software, que já se encontra na quarta edição. Ministra cursos de Testes de Software em faculdades.

A produção de um software de qualidade depende de muitos fatores envolvidos no seu processo de desenvolvimento. Uma das maneiras de contribuir com a qualidade de um produto é incluir atividades de testes de software bem direcionadas e conduzidas durante a sua construção. O processo de testes tem como uma de suas etapas a identificação dos defeitos existentes no produto em análise. Antecipar a localização de defeitos, antes mesmo que ocorram, pode contribuir para o direcionamento das atividades de testes, de maneira que a equipe envolvida nessas atividades possa direcionar seus esforços para áreas do software que apresentam maior propensão a apresentar problemas. Iniciativas neste sentido são muito difundidas na indústria, apresentando diversos casos práticos em empresas como: Microsoft, Ericsson e IBM.

O que é Predição de Defeitos em Software?

Predição de defeitos é a tentativa de antecipar a localização de defeitos em uma aplicação através do uso de técnicas específicas. Em muitos casos, trata-se da criação de modelos, que analisam dados históricos de um projeto para, a partir daí, tentar adivinhar como se comportarão versões futuras de seu código.

Predizer a localização de defeitos em módulos da aplicação aumenta as chances que as atividades de testes têm de encontrá-los. Dessa maneira, ações corretivas podem ser tomadas o quanto antes no desenvolvimento do produto.

As atividades de testes de software podem ser responsáveis por uma parcela considerável dos custos totais de um projeto. Tosun, em seu artigo *Ensemble of software defect predictors: a case study*, publicado em 2008, relata uma redução de aproximadamente 29% (vinte e nove por cento) no esforço de testes quando as atividades são direcionadas através da predição de defeitos.

Assim, através da aplicação de tais técnicas, pode-se prever em uma aplicação, por exemplo:

• **Se determinado módulo está ou não propenso ao aparecimento de defeitos**. Um módulo pode ser representado por um pacote, um arquivo, uma classe, ou um método, a depender da granularidade escolhida para análise. Neste caso, diz-se que um módulo classificado como SIM está propenso ao aparecimento de defeitos, assim como um módulo classificado como NÃO, não está propenso ao aparecimento de defeitos;

• **Quantidade de defeitos em um módulo**. Em módulos onde é esperado o aparecimento de defeitos, pode-se querer estimar a quantidade desses defeitos em cada módulo;

• **Densidade de defeitos na aplicação**. A densidade de defeitos pode ser representada pela quantidade de defeitos por linha de código;

• **Quais módulos apresentam a maior quantidade de defeitos na aplicação**. Com nessa informação, o time de testes pode priorizar seus esforços em módulos com propensão a maior quantidade de defeitos.

Benefícios da Predição de Defeitos

Diversos são os benefícios reportados na academia e na indústria pela aplicação de técnicas de predição de defeitos, como por exemplo, a geração de informação para:

• Suporte ao planejamento e execução de testes;

• Identificação de pontos de melhoria no código desenvolvido, através do diagnóstico de trechos de código complexos que podem ser simplificados;

• Redução da taxa de inserção de defeitos na manutenção e evolução do software;

• Planejamento do esforço de desenvolvimento de iterações futuras do projeto.

Para que uma equipe possa se beneficiar da aplicação de técnicas de predição, esta deve definir quais informações coletar em um projeto já no inicio do seu desenvolvimento. A seção seguinte explica que tipo de métricas podem ser coletadas em um projeto.

Métricas para Predição de Defeitos

Algumas métricas de projeto devem ser coletadas, de maneira manual ou automática. Em seguida, essas métricas devem ser analisadas, caso a caso, visando à identificação da sua relação com o aparecimento de defeitos em versões futuras do código. Diversos tipos de métricas podem ser utilizadas, de diferentes categorias, por exemplo: métricas de métodos, de classes, de componentes ou processo.

As métricas de projeto podem ser básicas ou derivadas. Cibele Feitosa explica a diferença entre esses dois tipos de métricas em um trabalho publicado em 2004, chamado *Definição de Um Processo de Medição e Análise com Base nos Requisitos do CMMI*. Segundo Cibele, uma métrica básica é "independente de outras medidas". Já a métrica derivada pode ser definida como "função de duas ou mais medições básicas ou derivadas".

Como exemplo de métricas básicas, podemos mencionar as seguintes métricas retiradas do artigo *Use of relative code churn measures to predict system defect density*, publicado por Naggapan, em 2005:

• LOC – quantidade de linhas de código;

• Quantidade de modificações feitas em um arquivo;

• Quantidade de arquivos que foram modificados.

Como exemplo de métricas derivadas, podemos citar as métricas abaixo, retiradas do mesmo artigo:

• LOC modificado / LOC Total – quantidade de linhas de código que foram modificadas em relação à quantidade total de linhas de código existentes na aplicação;

• Arquivos modificados / Total de arquivos – quantidade de arquivos que foram modificados em relação à quantidade total de arquivos existentes na aplicação.

Uma busca em bases eletrônicas por artigos que abordassem o tema predição de defeitos foi realizada pela autora deste artigo com o objetivo de identificar que métricas de projeto geralmente são utilizadas para servir de entrada para a criação de modelos de predição.

Como resultado desta pesquisa, a **Tabela 1** exibe as métricas que tiveram uma quantidade maior de ocorrências dentre os 66 (sessenta e seis) artigos selecionados pelo estudo (Pontes, 2009). A coluna ocorrências indica a quantidade de vezes que a métrica apareceu em relação à quantidade total de artigos analisados.

É importante ressaltar que 1134 (mil cento e trinta e quatro) artigos foram encontrados pelas palavras-chave de pesquisa utilizadas na busca. Em seguida foram definidos critérios de inclusão e exclusão desses artigos nesta revisão da literatura, de maneira que somente artigos recentes, e que abordassem diretamente o tema, fossem incluídos para extração de suas informações.

A seguir, apresentamos uma breve explicação de cada métrica mencionada na **Tabela 1**.

• LOC – Quantidade de linhas existentes em um módulo (NASA, 2009);

• Defeitos – Representa a quantidade de defeitos que são encontrados nos módulos do mesmo projeto, em versões anteriores do código;

Métrica	Descrição	Ocorrências	Categoria
LOC	Linhas de Código	26 / 66	Método
Defeitos	Defeitos encontrados em versões anteriores	21 / 66	Processo
CC	Complexidade Ciclomática	14 / 66	Método
Halstead	Suíte de métricas de complexidade de código	10 / 66	Método
Idade	Idade do arquivo no sistema	9 / 66	Arquivo
Chidamber & Kemerer (C&K) / LP	Suíte de métricas de complexidade de projeto / Linguagem de Programação	8 / 66	Classe

Tabela 1. Métricas para Predição de Defeitos

• Complexidade Ciclomática – Medida de complexidade de um módulo, representada pelo número de caminhos linearmente independentes (NASA, 2009);
• Métricas do Grupo Halstead – Conjunto de métricas de complexidade de código calculadas a partir dos operadores e operandos existentes em um módulo;
• Idade – A idade do arquivo no sistema pode ser definida como a quantidade de dias ou meses desde a criação deste arquivo. Percebe-se que arquivos mais antigos tendem a estabilizar-se e estar menos suscetível ao aparecimento de defeitos do que arquivos mais recentes;
• Métricas do Grupo CK (Chidamber e Kemerer) – Conjunto de métricas orientadas a objeto derivadas das classes da aplicação;
• Linguagem de Programação – Devido à complexidade, desenvolver software em algumas linguagens de programação pode favorecer o aparecimento de defeitos.

Algumas considerações sobre Métricas para Predição

Embora seja mais comum encontrar trabalhos de predição de defeitos que utilizam métricas coletadas no nível de método, Catal, em seu artigo *An Artificial Immune System Approach for Fault Prediction in Object-Oriented Software*, publicado em 2007, sugere que métricas no nível de classes proporcionam melhor precisão aos modelos criados.

Ainda, alguns autores, como Nagappan, afirmam que a utilização de métricas derivadas apresenta melhores resultados para os modelos criados do que a utilização de métricas básicas.

O uso de métricas combinadas também consiste em uma alternativa que pode melhorar o desempenho de modelos de predição, embora ainda represente uma abordagem ainda pouco explorada.

Quanto mais métricas puderem ser coletadas, melhor, pois a análise da relação de cada métrica com o aparecimento de defeitos em código é realizada de maneira automática.

Técnicas para Predição de Defeitos

Diversas técnicas podem ser utilizadas para antecipar a localização de defeitos em um software, sendo as técnicas estatísticas e de aprendizado de máquina as mais comuns.

Dentre as técnicas estatísticas mais encontradas, está a análise de regressão que pode se subdividir (mas não se limita) em:
• Regressão Linear;
• Regressão Logística;
• Regressão Binomial Negativa.

No livro *Introduction to Econometrics*, Maddala define análise de regressão como a "descrição e avaliação da relação entre uma variável, normalmente chamada de variável dependente ou variável resposta, e uma ou mais outras variáveis, normalmente chamadas de variáveis independentes ou preditoras". Trazendo para o contexto deste artigo, podemos exemplificar como variáveis independentes todas as métricas do projeto que podem ser coletadas para analisar seu impacto na ocorrência de defeitos. Podemos exemplificar como variável dependente, o que se quer observar, como a existência ou não de defeitos em determinado módulo da aplicação, quantidade dos defeitos existentes ou densidade.

As técnicas de aprendizado de máquina mais comumente utilizadas são:
• Árvore de decisão;
• Classificadores Bayesianos.

A **Tabela 2** apresenta um resumo das técnicas mais encontradas pela revisão da literatura mencionada anteriormente.

Categoria	Técnica	Ocorrências
Técnica Estatística	Regressão Binomial Negativa	9
Técnica Estatística	Regressão Logística	9
Técnica Estatística	Regressão Linear	3
Total para Técnicas Estatísticas		21
Aprendizado de Máquina	Classificador Bayesiano	7
Aprendizado de Máquina	Árvore de Decisão C4.5	5
Aprendizado de Máquina	Árvore de Decisão J48	5
Total para Aprendizado de Máquina		17
Total de ocorrências		38

Tabela 2. Técnicas para Predição de Defeitos

De maneira geral, não se pode afirmar que uma técnica seja melhor ou pior que outra. Uma técnica pode mostrar melhor desempenho em um projeto do que em outro, a depender do formato em que estão os dados que foram coletados e do que se pretende prever neste projeto.

A qualidade dos dados coletados pode influenciar de maneira positiva ou negativa a qualidade dos modelos de predição criados para um projeto. Assim, é muito importante que estes dados sejam confiáveis.

Dificuldades em Coleta de Informações

Diversos obstáculos ainda na fase de coleta de métricas podem ser encontrados na maioria dos projetos de software, o

que pode acarretar em dificuldades para a criação de modelos de predição. Algumas das dificuldades estão listadas abaixo.

Falta de registro formal dos defeitos encontrados

Apesar de ser um problema que acontece com pouca freqüência em empresas com alto nível de maturidade de seus processos, não podemos deixar de enfatizar a importância do registro de todos os defeitos encontrados na aplicação, em qualquer fase de testes.

Possíveis motivos:

• Falta de conhecimento da importância desta informação;

• Testes de software executados pelos próprios desenvolvedores, que muitas vezes corrigem os defeitos sem devidamente registrá-los;

• Testadores e desenvolvedores trabalhando em conjunto, o que pode acarretar em defeitos encontrados pelos testadores e corrigidos imediatamente pelos desenvolvedores antes dos mesmos serem registrados.

Mitigação:

• Conscientizar a equipe do projeto da importância do registro formal dos defeitos encontrados na aplicação. Esta é uma etapa do desenvolvimento de um software imprescindível para análise da qualidade do produto;

• Alocar testadores independentes, que não sejam os desenvolvedores do projeto, para a realização das atividades de testes.

Isolamento da localização exata do defeito

Em muitos casos, é difícil identificar exatamente em que arquivo ocorreu o defeito.

Possíveis motivos:

• Software desenvolvido sem gerência de configuração;

• Para se corrigir um defeito, muitas vezes mais de um arquivo do sistema é modificado. Se não for registrada a informação de onde exatamente estava o defeito, arquivos não defeituosos podem ser classificados como defeituosos, gerando informações inconsistentes para a criação de modelos de predição.

Mitigação:

• Desenvolver software sob gerência de configuração, para armazenar as informações do projeto de maneira organizada;

• No momento da correção do defeito, recomenda-se que o desenvolvedor registre na ferramenta de rastreamento de defeitos o(s) módulo(s) onde foi(ram) feita(s) correção(ões) do problema.

Falta de acesso às informações dos projetos

Muitas vezes, é difícil obter qualquer informação dos projetos.

Possíveis motivos:

• Essa dificuldade pode aparecer, por exemplo, quando o projeto tem informações confidenciais;

• Pode haver desinteresse da gerência e demais integrantes da equipe nesse tipo de trabalho, talvez por falta de conhecimento dos benefícios que a realização de predição de defeitos de software pode trazer.

Mitigação:

• Realizar apresentações para a equipe do projeto mostrando os benefícios que eles podem ter com a aplicação das técnicas de predição de defeitos.

• Buscar apoio gerencial.

• Mascarar informações coletadas, como renomear arquivos, mudar escala de números, etc.

Além de resolver as dificuldades mencionadas acima, Kitchenham, Hughes e Linkman (2001) ressaltam que para uma boa coleta de métricas em projetos, é preciso um processo que defina quem, quando e como essas medidas devem ser coletadas.

Processo de Coleta de Dados

A criação de um subprocesso de coleta de informações ajuda na obtenção de dados mais confiáveis e facilita a criação de modelos de predição através da análise desses dados. A correta realização das atividades desse subprocesso permite que os dados coletados sejam mais robustos e confiáveis para utilização em aplicação de técnicas de predição de defeitos.

A **Figura 1** ilustra algumas atividades, em cor de destaque, que podem ser integradas a processos de desenvolvimento de projetos.

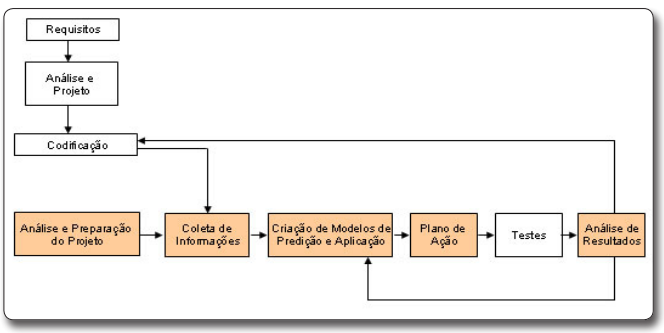


Figura 1. Processo de Coleta de Métricas

O subprocesso de coleta de métricas apresentado tem como base o processo descrito por Catal e Diri (2008), e o trabalho de Kitchenham, Hughes e Linkman (2001). A seguir, serão detalhadas suas atividades.

Análise e Preparação do Projeto

Esta etapa visa identificar características importantes nos projetos e verificar sua adequação para a coleta de métricas. Caso o projeto ainda não esteja adequado, por exemplo, apresentando uma equipe preparada para as atividades, o processo desse projeto deve ser revisto antes que as atividades de coleta tenham início.

Não deve haver, por exemplo, as dificuldades mencionadas anteriormente em relação à coleta de dados.

Alguns itens importantes devem ser verificados:

• A pessoa responsável pela realização da predição deve ser formalmente alocada para esta atividade;

• O projeto deve estar sob controle de versão, para facilitar a aquisição de informações;

• Deve ser escolhida a ferramenta de coleta de métricas de complexidade de código, que depende da linguagem de programação em que o projeto será desenvolvido. Posteriormente, apresentaremos algumas ferramentas que podem ser utilizadas para a coleta de tais métricas;
• Definir a granularidade de coleta (pacote, arquivo, classe ou método), o conjunto de métricas a serem inicialmente coletadas e a unidade de medida;
• Construir métricas derivadas, caso a equipe opte por esse tipo de métrica;
• Definir variável de saída que se espera do modelo criado, em outras palavras, definir o que se pretende observar;
• Estabelecer a versão inicial do software, que servirá de ponto de partida para a predição dos defeitos;
• Definir qual técnica será utilizada para a criação de modelos de predição.

Coleta de Informações

Esta etapa visa coletar as informações do projeto que foram definidas na etapa anterior. Além da coleta das métricas na granularidade definida, devem ser coletados os defeitos encontrados, caso existam.

Nesta fase, deve-se:
• Extrair relatório de métricas do projeto, através das ferramentas, utilizando a versão do software definida na fase anterior;
• Construir métricas derivadas do projeto, a partir das métricas básicas coletadas, caso seja necessário;
• Coletar informações sobre defeitos encontrados em versões anteriores da aplicação, caso existam. Deve ser extraída da ferramenta de rastreamento de defeitos quais módulos da aplicação apresentaram problemas;
• Construir uma planilha de dados, ou *dataset*, composto pelas métricas do projeto e para cada linha da planilha que representa um módulo, associar a informação de defeitos existentes. Um exemplo de planilha de dados é apresentado na **Figura 2**. Esse exemplo é disponibilizado pelo NASA *Independent Verification and Validation Metrics Data Program (MDP)*. Cada coluna dessa planilha apresenta uma métrica de código coletada no projeto em questão.

MODULE	LOC_BLANK	BRANCH_COUNT	CALL_PAIRS	LOC_CODE_AND_COMMENT	LOC_COMMENTS	CONDITION_COUNT
25523	0	5	1	0	0	8
25524	1	3	2	0	0	4
25525	0	1	0	0	0	0
25526	18	19	5	1	58	34
25527	2	3	0	0	9	4
25528	2	1	2	0	7	0
25529	36	13	8	0	42	18
25530	43	39	20	1	35	56
25531	10	3	5	0	2	4
25532	34	23	6	0	31	34
25533	17	15	5	0	7	8
25534	55	40	5	0	55	44
25535	32	57	3	0	39	86
25536	4	1	1	0	3	0
25537	9	11	3	0	10	20
25538	17	24	4	1	16	28
25539	225	236	15	2	40	384

Figura 2. Exemplo de dataset do MDP (Projeto MW1)

Criação de Modelos de Predição e Aplicação

Esta etapa visa a criação ou melhoria de modelos de predição de defeitos a partir das métricas do projeto coletadas na etapa anterior.

Nesta etapa, deve-se:
• Criar modelos de predição de defeitos, aplicando a técnica escolhida ao *dataset* do projeto. A criação de modelos de predição pode acontecer da seguinte maneira: O *dataset* é separado em três partes iguais. 2/3 (dois terços) desses dados podem ser utilizados para treino do modelo de predição. 1/3 (um terço) desses dados podem ser utilizados para testes do modelo criado. Então, temos o que chamamos de dados de treino, e dados de testes. Os dados de treino são avaliados, e o modelo vai analisando em que situações o código apresentou ou não defeito. Então este modelo usa essas informações "aprendidas" e passa a classificar os dados de teste;
• Verificar a qualidade do modelo criado. A qualidade do modelo pode ser verificada através da análise de seus dados de saída, visando verificar o percentual de módulos classificados corretamente como defeituosos ou não defeituosos. A princípio podem-se identificar as seguintes classificações dos resultados (Witten, Frank, 2005), que dão origem a informações como precisão e confiabilidade do modelo criado:
  - VP = Verdadeiro Positivo. Isso significa que o módulo foi classificado pelo modelo como defeituoso, e de fato, havia defeito;
  - VN = Verdadeiro Negativo. Isso significa que o módulo foi classificado pelo modelo como não defeituoso, e de fato, não havia defeito;
  - FP = Falso Positivo. Isso significa que o módulo foi classificado pelo modelo como defeituoso, e de fato, não havia defeito;
  - FN = Falso Negativo. Isso significa que o módulo foi classificado pelo modelo como não defeituoso, e de fato, havia defeito.

A partir dessas classificações, podemos derivar alguns parâmetros, que vão informar sobre a qualidade deste modelo[1]:
  - Precisão - Precisão é uma medida de exatidão ou fidelidade que representa a proporção dos módulos que o classificador previu como propensos à falha e que realmente possuíam algum tipo de defeito;
  - *Recall* - *Recall* é uma medida de completude que representa a proporção dos módulos realmente defeituosos que foram corretamente identificados, sendo considerada a probabilidade de detecção;
  - Falso Alarme - A probabilidade de falso alarme é um parâmetro que representa a proporção de módulos sem defeito incorretamente classificados;
  - Acurácia - A acurácia é um parâmetro que representa a proporção da quantidade total de módulos que foram corretamente classificados.

1 - Definições retiradas da dissertação de mestrado de Miguel Bezerra, intitulada Detecção de Módulos de Software Propensos a Falhas através de Técnicas de Aprendizagem de Máquina

Uma vez que o modelo apresenta um desempenho satisfatório, poderá ser aplicado no projeto. Devemos buscar a criação de modelos que classifiquem a maioria dos módulos corretamente. Porém, Ma e Cukic, em um artigo chamado *Adequate and Precise Evaluation of Predictive Models in Software Engineering Studies*, publicado em 2007, relatam que quanto maior a acurácia de um modelo, menor a sua taxa de detecção da propensão a defeitos. Isso significa que mais módulos defeituosos podem vir a ser classificados como livres de defeito.

• Aplicar o modelo criado para predizer o comportamento de uma versão futura do código da aplicação.

• Melhorar os modelos de predição criados, à medida que as informações do projeto permitam que se refinem os modelos.

Plano de Ação

Esta etapa visa o direcionamento das atividades do projeto, com base nas informações obtidas com a aplicação do modelo. A equipe do projeto deve fazer uma análise crítica do que pode acontecer em versões futuras do código da aplicação.

É aconselhável nesta fase:

• Identificar pontos de melhoria no código desenvolvido. Estudos relatam que melhoria de código e defeitos têm uma correlação inversa. Ratzinger, Sigmund e Gall, publicaram um artigo no ano de 2008, chamado *On the relation of refactorings and software defect prediction*, que explica que em um período, a taxa de defeitos em código diminui a medida que a quantidade de melhorias nesse mesmo código aumenta.

• Mapear módulos propensos a defeito em funcionalidades do sistema.

• Direcionar o planejamento e execução das atividades do processo de testes.

• Planejar o esforço de desenvolvimento de iterações futuras

Análise de Resultados

Esta etapa visa a avaliação dos resultados fornecidos pelos modelos criados.

Nesta etapa, deve-se:

• Validar os modelos de predição utilizados. Devem-se comparar as informações obtidas através dos testes (que módulos realmente apresentaram defeitos) com as informações esperadas (que módulos o modelo previu que apresentaria defeitos).

• Deve ser avaliada a possibilidade de melhorar o(s) modelo(s) criado(s), ou melhorar os dados utilizados.

Ferramentas para Coleta de Dados

Algumas ferramentas podem ser utilizadas para coletar as métricas de complexidade. Dependendo da linguagem de programação utilizada no projeto, uma ou outra ferramenta será adequada à coleta das informações. A **Tabela 3** apresenta algumas destas ferramentas.

Ferramentas para Mineração de Dados

Na categoria de ferramentas que dão suporte à mineração de dados, algumas ferramentas como a Weka (WEKA, 2009) apresentam implementações de algoritmos de aprendizado de máquina, além de outras implementações, que podem ser utilizadas com eficiência para classificar dados. A **Tabela 4** apresenta algumas dessas ferramentas.

Ferramenta	LP	Site	Licença
CCCC	C e C++	http://sourceforge.net/projects/cccc	Gratuita
DevMetrics	.NET	http://www.anticipatingminds.com	Gratuita
Metrics	Java	http://metrics.sourceforge.net/	Gratuita
StatSVN	Independe	http://www.statsvn.org/	Gratuita

Tabela 3. Ferramentas para auxiliar a coleta de métricas em projetos

Ferramenta	Site	Licença
WEKA	http://www.cs.waikato.ac.nz/~ml/weka	Gratuita
Orange	http://www.ailab.si/orange	Gratuita
Tanagra	http://chirouble.univ-lyon2.fr/~ricco/tanagra	Gratuita

Tabela 4. Ferramentas para mineração de dados

Conclusões

Confiança na qualidade dos dados coletados é fundamental para a realização de um bom trabalho de predição de defeitos. Por isso, a coleta de dados dos projetos deve ser planejada e monitorada.

Deve ficar claro que não se recomenda escolher um conjunto de métricas na literatura e extraí-las do projeto para que sirvam de entrada para modelos de predição de defeitos. Para cada métrica de projeto coletada, deve ser feita uma análise da sua relação com a aparição de defeitos na aplicação, para que métricas insignificantes não interfiram negativamente na qualidade do modelo de predição criado. Esta análise é necessária, pois, uma métrica que representa defeitos em um projeto pode não significar o mesmo em outro projeto. Isso mostra que a realização de predição de defeitos em projetos é algo particular a cada projeto, não sendo recomendado reaproveitar modelos criados em outros projetos sem uma análise e adaptação muito críticas.

A escolha da técnica a ser utilizada para gerar modelos de predição deve levar em consideração que métricas de projeto serão coletadas, para servir de entrada para o modelo criado, considerando o formato desses dados. Além disso, deve-se determinar qual o conhecimento que se deseja obter, ou seja, o que se deseja ter como saída do modelo. Dependendo dos dados coletados, uma ou outra técnica pode se mostrar melhor ou pior na classificação de informação. Não é objetivo deste artigo detalhar as técnicas apresentadas, visto que não há uma técnica que seja ótima para todos os casos.

Por fim, predição de defeitos deve ser encarada como um direcionamento para as atividades do projeto, não levando a conclusão, em hipótese alguma, que somente as áreas indicadas com maior propensão a defeitos é que devem ser testadas. ●

Mapas Conceituais

Utilizando-os na Identificação de Riscos de Requisitos

Cristine Gusmão
cristine@dsc.upe.br
Professora 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). Coordena o Mestrado em Engenharia da Computação e o PROMISE Project Management Improvements in Software Engineering grupo de pesquisa na área de Engenharia de Software. È coordenadora do Curso de Bacharelado em Sistemas de Informação 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. Pesquisadora associada do Núcleo de Telesáude - NUTES HC - UFPE.

Kenelly Almeida
ksra@dsc.upe.br
Graduanda em Engenharia de Computação pela Universidade de Pernambuco (UPE). Pesquisadora financiada pelo PIBIC/CNPq, atuando na área de Engenharia de Software e Gerenciamento de Projetos, com ênfase em Gerenciamento de Riscos em Projetos de Desenvolvimento de Software. Membro do grupo de pesquisa Project Management Improvements in Software Engineering (PROMISE).

Júlio Venâncio
jvmj@dsc.upe.br
Pesquisador do Núcleo de Telesaúde da Universidade Federal de Pernambuco (NUTES - HC – UFPE), atuando na área de Engenharia de Software e Gerência de Projetos. É graduando em Engenharia de Computação pela Universidade de Pernambuco (UPE). Membro do grupo de pesquisa Project Management Improvements in Software Engineering (PROMISE).

De que se trata o artigo?
Este artigo apresenta uma técnica para apoio à identificação de riscos de requisitos em ambientes de desenvolvimento de software utilizando mapas conceituais como ferramenta.

Para que serve?
O método proposto serve para expor os possíveis eventos relacionados aos requisitos do sistema que podem afetar negativamente o projeto, de forma simples e intuitiva, sem que haja a necessidade de conhecimento denso sobre riscos de projetos.

Em que situação o tema é útil?
Nos últimos anos tem-se observado um aumento nos investimentos em conhecimento e práticas de gerenciamento de riscos em projetos de desenvolvimento de software, objetivando alcançar o sucesso nesses projetos. Desta forma, o uso de técnicas para identificação de adversidades que podem afetar o bom andamento do projeto são necessárias para que estes riscos possam ser posteriormente mitigados e controlados.

A importância da utilização de processos, técnicas e ferramentas de Gerenciamento de Riscos é cada dia mais reconhecida nos ambientes de desenvolvimento de software [Gusmão 2007]. Isso se deve, por vezes, pelo fato da associação do insucesso de projetos a uma má gerência de riscos. Neste sentido, o uso da gerência de risco se faz importante ao gerenciamento de qualquer projeto de desenvolvimento de software.

Uma boa gerência de riscos permite: i) dar melhor visibilidade às incertezas inerentes a um projeto de software; ii) gerar

maior proteção e atenção contra incertezas a um custo menor em relação ao que seria gasto caso o risco se materializasse e; iii) minimizar falhas e retrabalhos nos projetos.

No contexto da Engenharia de Software, os requisitos descrevem as funções e restrições que o produto a ser desenvolvido deverá possuir. Durante o ciclo de vida de um projeto de software, o levantamento de requisitos é realizado nas fases iniciais do processo de desenvolvimento.

No entanto, diversas falhas no produto são geradas pelo simples fato da análise dos requisitos, muitas vezes por limite de tempo, não ser realizada de maneira correta, apresentando muitos requisitos ambíguos, instáveis ou até mesmo incompletos. Vale salientar que a compreensão completa dos requisitos do software é fundamental para um desenvolvimento de software bem sucedido [Pressman 1995].

Outro ponto interessante é que quanto mais tarde os defeitos ou falhas forem encontrados no produto do software, mais cara é a sua correção. Desta maneira, é relevante o uso de métodos que visem melhorar a qualidade do processo de análise de requisitos.

A gerência de riscos é uma área que tem, como uma grande vantagem, a possibilidade de ser incorporada em todo o ciclo de vida do desenvolvimento do software, inclusive na etapa de levantamento e análise dos requisitos.

A captura e análise dos requisitos são atividades que incutem um grau de subjetividade, normalmente associada ao conhecimento e experiência do analista responsável por elas. Dentro deste contexto, a utilização de técnicas que promovam a padronização de termos e objetos de gerenciamento tende a proporcionar uma mitigação de prováveis insucessos.

Os Mapas Conceituais são instrumentos utilizados para representar graficamente o conhecimento, ou parte dele, adquirido sobre um determinado domínio. A representação do conhecimento em riscos de requisitos pode ser uma boa alternativa no processo de identificação destes riscos, quando da análise dos requisitos, ao tornar mais clara a percepção e compreensão de eventos adversos ao projeto, pois estrutura o conhecimento de forma semelhante ao processo cognitivo do cérebro humano.

Com base nestas informações iniciais, este artigo apresenta uma abordagem de identificação de riscos de requisitos utilizando mapas conceituais. Para melhor entendimento, o restante do artigo inicialmente apresenta conceitos básicos de gerenciamento de riscos com foco na atividade de identificação, apresentando algumas técnicas. Em seguida o tema Mapas Conceituais é introduzido, exemplificando como eles podem se tornar um elemento facilitador no processo de identificação de riscos de requisitos em projetos de desenvolvimento de software.

Gerenciamento de Riscos – Visão Geral

No contexto de projetos de software, risco é um evento que possui probabilidade de ocorrência, e, caso ocorra, poderá trazer impactos positivos ou negativos.

São várias as classificações de riscos de software encontradas na literatura. Umas das mais utilizadas e conhecidas é a Taxonomia de Riscos definida pelo Instituto de Engenharia de Software (Software Engineering Institute – SEI), no qual os riscos são classificados em três categorias [Carr et al 1993], como pode ser visualizado através da **Figura 1**:

• **Engenharia do Produto** – Conhecidos como risco técnicos, compreende os eventos que podem afetar na construção do produto que está sendo desenvolvido. Normalmente este tipo de risco está relacionado às seguintes atividades: Análise de Requisitos, *Design*, Desenvolvimento e Testes.

• **Ambientes de Desenvolvimento** – Foca nos eventos que afetam o ambiente de desenvolvimento de software, principalmente os riscos relacionados ao processo e à metodologia adotados para a construção do software, bem como o próprio ambiente organizacional.

• **Restrições dos Programas** – São os riscos ligados a fatores externos ao projeto, mas que podem afetá-lo direta ou indiretamente no seu andamento. Estes fatores externos podem impactar, por exemplo, nos recursos, nos contratos e nos envolvidos no projeto.

A Gerência de Riscos de projetos de software é essencial para o sucesso de projetos, pois permite auxiliar na minimização da ocorrência de fatores que podem impactar de maneira negativa o projeto, bem como na identificação de oportunidades ao assumir certos riscos. Além de o risco ser parte inerente de todo projeto, o seu gerenciamento disponibiliza visões sobre a relação causa-efeito resultantes das decisões tomadas, orientando os envolvidos no projeto. Assim, a gerência de riscos pode ser definida como o emprego de habilidades e competências, aliadas ao conhecimento adquirido, através do uso de processos com a utilização de técnicas e métodos para identificação, análise e controle dos riscos [Gusmão 2007].

Com relação às atividades que compõem o processo de Gerenciamento de Riscos na Engenharia de Software, parece haver um consenso. São elas:

• **Planejar a Gerência de Risco** – É onde são definidas as políticas, responsabilidades, cronograma, metodologia, recursos e estratégias para a realização da gerência de riscos do projeto.

• **Identificar Riscos** – Etapa inicial de um projeto de software, a identificação de riscos determina e documenta todos os riscos que podem afetar o sucesso do projeto. É um processo iterativo e incremental que acompanha o projeto também durante todo seu ciclo de vida. Existem diversas técnicas de identificação de riscos, tais como Listas de Verificação, Comparação por Analogia e *Brainstorming*, dentre outras.

• **Analisar Riscos** – Nesta etapa, os riscos são priorizados e categorizados segundo algum critério específico estabelecido, para tornar a gerência concentrada nos riscos prioritários. Normalmente existem duas maneiras de realizar a análise de riscos:

• Análise qualitativa: Utiliza escalas de probabilidade e impacto para o cálculo da exposição ao risco. Essas escalas são definidas durante o planejamento da gerência de risco. Seu objetivo primordial é priorizar os riscos identificados.

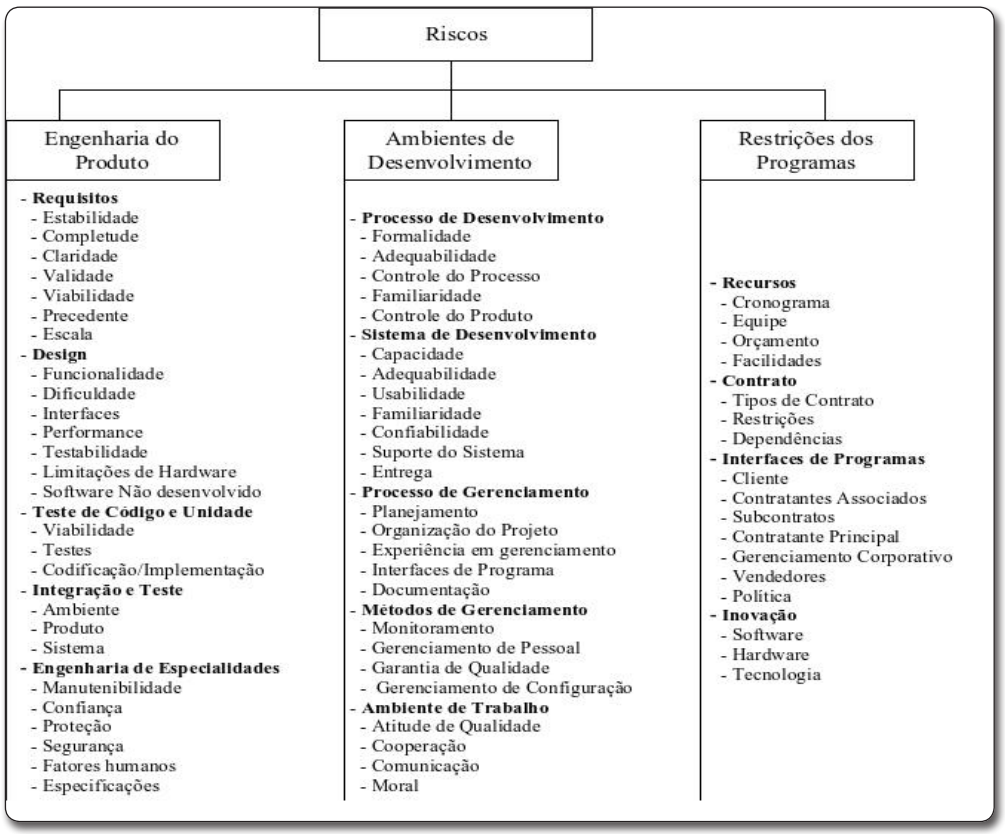


Figura 1. Taxonomia de Riscos de Projetos de Software

• Análise quantitativa: Normalmente essa atividade é realizada após a análise qualitativa nos riscos que foram priorizados. Utiliza técnicas baseadas em análise de dados estatísticos, fornecendo como resultado, por exemplo, estimativas mais precisas de custo e tempo ocasionados pelos riscos identificados.

• **Planejar Respostas aos Riscos** – Esta atividade diz respeito à definição de estratégias para tratamento dos riscos identificados. Ou seja, planos de ação são criados para os riscos identificados e planos de contigência são criados para os riscos que se encontram além das capacidades de mitigação.

• **Controlar e Monitorar os Riscos** – O controle avalia a situação corrente para determinar eventuais desvios do planejado, enquanto que o monitoramento é a observação da efetividade dos planos de ação na execução do desenvolvimento do projeto de software [Gusmão 2007].

Um ponto importante na gerência de riscos, principalmente em ambientes de desenvolvimento de software, é a questão da comunicação. Pouco adianta se todos os *stakeholders* do projeto não tiverem certa noção sobre os riscos, principalmente nos ambientes de gestão ágil de projetos. O próprio SEI propõe um formato de gerenciamento de riscos fortemente baseado na comunicação [Carr et al 1993]. Desta maneira, a comunicação dos riscos favorece não só o compartilhamento de informações deixando os envolvidos alertas, como também o esforço coletivo de todos no processo de gerenciamento dos riscos.

Após esta visão geral sobre o Gerenciamento de Riscos, o foco deste artigo nas seções seguintes será direcionado para a atividade de Identificação de Riscos, mais especificamente os riscos de requisitos. A seguir serão apresentadas algumas técnicas de identificação de riscos mais utilizadas em ambientes de desenvolvimento de software.

Técnicas de Identificação de Riscos

São várias as técnicas de identificação de riscos disponíveis na literatura. Algumas destas técnicas, inclusive recomendadas pelo Guia PMBOK (*Project Management Body of Knowledge)* [PMI 2004], são: listas de verificação (*checklists*), *brainstorming*, comparação por analogia, questionários, técnica Delphi, análise SWOT, entre outras. A seguir são apresentadas sucintamente algumas destas técnicas mais comuns.

Listas de Verificação

Também conhecida como *checklists*, trata-se de uma técnica onde existe uma lista pré-definida de riscos, normalmente agrupadas por categorias, no qual são verificados que riscos da lista estão presentes em um determinado projeto. Essas listas normalmente são criadas com base em projetos passados e/ou similares, sendo bastante utilizadas em combinação com outras técnicas de identificação de riscos.

Sua grande vantagem reside na facilidade de utilização e maior rapidez para identificar os riscos, pelo fato de apenas haver uma comparação do que está na lista com a realidade do projeto. No entanto, ficar preso a uma lista pode fazer com que outros riscos passem despercebidos. Por isto que essa técnica é normalmente utilizada em combinação com outra.

Brainstorming

Visa obter uma lista abrangente dos riscos de projetos. É uma das técnicas mais utilizadas, e a sessão é normalmente composta pelos membros do projeto e consultores que não fazem parte da equipe. É dividida em dois momentos: o primeiro consiste no levantamento de todos os riscos e o segundo é o processo de classificação dos riscos identificados. São diversas as maneiras de classificar os riscos. Por exemplo, uma das formas de classificação de riscos é realizada através de uma Estrutura Analítica de Riscos (*Risk Breakdown Structure* – RBS).

A vantagem do uso do *brainstorming* é que permite a geração de várias idéias novas rapidamente, envolve múltiplas visões e promove a participação de todos da equipe. A principal desvantagem do uso dessa técnica é que muitas vezes não há tempo suficiente para o levantamento completo dos riscos pela não-exploração correta das idéias. Outro problema é que os membros da equipe podem se esbarrar no pouco conhecimento sobre riscos, gerando ambigüidades ou não-entendimento correto sobre os riscos [Hossenloop e Bass 2007].

Questionário baseado em Taxonomia de Riscos

A Taxonomia de Riscos proposta pelo SEI organiza os riscos de desenvolvimento de software em três níveis: Classe, Elemento e Atributo. O Questionário baseado em Taxonomia de Riscos (TBQ – *Taxonomy Based Questionnaire*) consiste de perguntas pré-definidas sobre cada atributo, disponíveis em [Carr et al 1993].

A vantagem do uso desta técnica é que não é exigido o conhecimento aprofundado em riscos por parte de quem responde o questionário. No entanto, muitas vezes o questionário pode se tornar bastante extenso, tornando a atividade de identificação de riscos bastante repetitiva e cansativa.

Comparação por Analogia

Nessa técnica, os riscos são identificados com base em projetos similares. Logo, parte-se da premissa que projetos similares possuem riscos similares.

Essa técnica é bastante simples, porém o método para identificação de similaridades entre os projetos pode ser bastante subjetivo. Além do mais, o acesso às informações sobre o histórico de projetos passados pode ser dispendioso pelo nível de detalhamento destes projetos. Neste sentido, existem técnicas de Inteligência Artificial que auxiliam na identificação de riscos com base em projetos passados. Um deles é o método CBR *Risk*[1] que utiliza uma técnica de Inteligência Artificial chamada Raciocínio Baseado em Casos para identificação de riscos com base na similaridade de projetos [Trigo et al 2008].

Técnica Delphi

É uma técnica utilizada quando há a necessidade de um consenso na opinião de um grupo de especialistas a respeito de eventos futuros. Um elemento facilitador usa um questionário para levantar idéias sobre os riscos mais importantes de um projeto. Essas respostas são resumidas e redistribuídas para os especialistas para comentários adicionais, caso necessário. O consenso é obtido após algumas rodadas do processo descrito anteriormente.

Esta técnica baseia-se no uso estruturado do conhecimento, da experiência e da criatividade de um painel de especialistas, pressupondo-se que o julgamento coletivo, quando organizado adequadamente, é melhor que a opinião de um só indivíduo. A vantagem desta técnica é que aumenta a imparcialidade dos resultados por não haver influência entre os especialistas na resolução dos questionários. A desvantagem é que a troca de idéias entre os participantes é bastante escassa e há forte dependência dos questionários.

Análise Causal

Nessa técnica é realizada uma análise das causas com base nos efeitos com o objetivo de identificar os riscos pelas causas. Parte-se da premissa que é possível desenvolver respostas eficazes aos riscos através das causas abordadas.

Entre os métodos que empregam a análise causal estão: o Diagrama de Causa e Efeito, também conhecido com Espinha de Peixe (*fishbone*), e a técnica dos 6 W´s (*Who, Why, What, Whichway, Wherewithal, When*) [Boehm e De Marco 1997].

Utilizando Mapas Conceituais para Identificação de Riscos de Requisitos

Mapas conceituais são representações gráficas semelhantes a diagramas que relacionam conceitos sobre um determinado conteúdo. Os conceitos e suas relações são dispostos na forma de um mapa, onde os nós representam os conceitos, e as conexões entre dois conceitos são nominativas, formando proposições.

Mapas conceituais como estruturador do conhecimento

A teoria de Ausubel afirma que o ser humano aprende mais facilmente quando os conceitos mais gerais e mais inclusivos são apresentados primeiro, e posteriormente os mais específicos e menos inclusivos. Essa teoria caracteriza uma **diferenciação progressiva**, onde conceitos visivelmente semelhantes e mais abrangentes se desdobram em conceitos menos inclusivos, diferenciando-se entre si [Ausubel *et al* 1980]. Um mapa conceitual hierárquico se ramifica em vários ramos a partir de um conceito central. Um conceito de um ramo da raiz pode se relacionar com um conceito de outro ramo de modo a permitir uma relação entre conceitos aparentemente díspares, indicando uma **reconciliação integrativa**, o que explicita uma conexão aparentemente imperceptível.

Apoiados fortemente na teoria da Aprendizagem Significativa de David Asusubel, Novak e Gowin propuseram a construção de mapas conceituais em um modelo hierárquico, numa necessidade de representar os conceitos e relações presentes na estrutura cognitiva de uma determinada pessoa, tanto por uma diferenciação progressiva, quanto por uma reconciliação integrativa, contribuindo de forma mais eficiente para a aquisição e construção de conhecimento do indivíduo [Novak e Gowin 1999].

Quanto a sua dimensão, os mapas conceituais podem ter uma, duas, três ou mais dimensões. Os unidimensionais resultariam

1 - CBRRisk na Web: pma.dsc.upe.br/cbrrisk

em uma lista de conceitos; bidimensionais, se apresentam como uma boa alternativa para estruturar o conhecimento incluindo vantagens do ponto de vista instrucional; os tridimensionais possuem várias possibilidades de conexões entre os conceitos, o que tornaria sua construção complexa; e, mais de três dimensões tornaria o mapa conceitual demasiadamente abstrato, não apresentando grandes utilidades práticas.

Existem inúmeros tipos de mapas conceituais, construídos para diversas finalidades. Alguns são simples de elaborar, como o mapa conceitual do tipo Teia de Aranha, onde todos os conceitos estão conectados a um tema central e poucas relações são feitas entre os conceitos. O mapa do tipo Fluxograma normalmente é utilizado para otimizar algum processo com os conceitos apresentados de forma lógica e sequencial, enquanto o mapa conceitual do tipo Hierárquico apresenta as informações em ordem descendente de importância, fornecendo primeiro os conceitos mais globais e estendendo-se com conceitos auxiliares.

No entanto, o único modelo de mapa conceitual que é baseado na teoria da aprendizagem significativa é o mapa conceitual do tipo Hierárquico proposto por Novak e seus colaboradores.

Como o mapeamento conceitual é uma técnica subjetiva, não existe um mapa correto ou incorreto. No entanto, a partir de conceitos aceitos pela comunidade científica sobre determinado conteúdo, pode-se construir um bom mapa conceitual, com inúmeras relações entre os conceitos, ou um mau mapa conceitual, com as relações linearizadas.

O processo de construção de um mapa conceitual hierárquico é simples:

1. Primeiramente identificam-se palavras-chave inclusas no conteúdo que é desejado mapear, elaborando uma lista de conceitos sobre um determinado assunto ou tema;
2. Ordenam-se os conceitos, colocando o mais geral ou mais inclusivo no topo do mapa, desdobrando progressivamente a partir deste os mais específicos até completar o mapa, de acordo com o princípio da diferenciação progressiva;
3. Após a disposição dos conceitos no mapa, verificam-se as possíveis relações entre conceitos de ramos diferentes, de acordo com a reconciliação integrativa. É necessário rotular as linhas das relações entre dois conceitos com palavras de ligação, a fim de formar uma proposição para expressar o significado da relação.

A **Figura 2** apresenta um mapa conceitual conciso sobre mapas conceituais tanto para elucidar alguns conceitos já expostos, como para prover um exemplo de mapa conceitual.

Existem diversas ferramentas utilizadas para a construção de mapas conceituais, dentre elas a ferramenta CMAP *Tools*, desenvolvida pelo *Institute for Human and Machine Cognition* (IHMC), considerada pela comunidade acadêmica como um dos principais instrumentos para a elaboração de mapas conceituais. Além de distribuída gratuitamente, CMAP *Tools* permite a elaboração de mapas conceituais compartilhados através da internet, facilitando o trabalho colaborativo.

Mapas conceituais são fortemente utilizados no âmbito

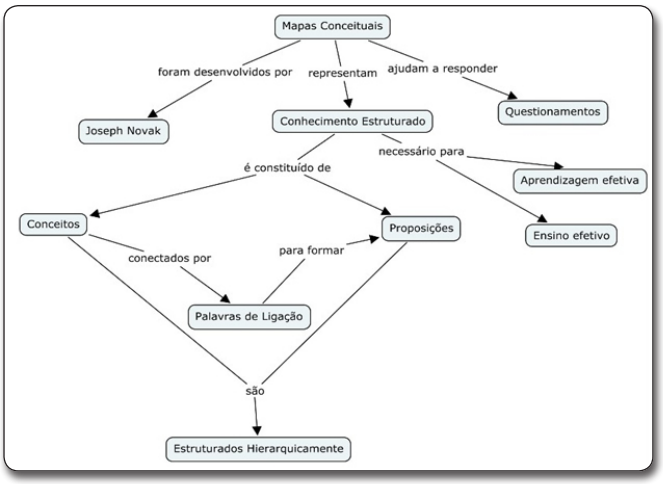


Figura 2. Mapa conceitual sobre mapas conceituais, modelo adaptado do Institute for Human and Machine Cognition (IHCM).

educacional, propostos como meios instrucionais que podem ser aplicados tanto na estruturação e análise do conteúdo, como no ensino e na verificação de aprendizagem, cuja maior vantagem pode estar no fato de enfatizarem o ensino e aprendizagem de conceitos sobre um determinado tema.

Há algumas aplicações interessantes do ponto de vista tecnológico da técnica de mapeamento de conceitos:

- Na Engenharia de Software, mais especificamente na especificação de Requisitos de Software, mapas conceituais podem ser utilizados durante o levantamento dos requisitos do sistema possibilitando a criação de um canal de comunicação entre os conceitos que compõem o domínio do sistema e os elementos que irão compor a solução no nível operacional [Rolim e Oliveira 2006];
- Em ambientes de Desenvolvimento Distribuído de Software, organizando os conceitos do software de modo a auxiliar o gerente de projetos a visualizar os aspectos mais importantes existentes em determinado projeto [Prikladnicki 2003];
- Na Gestão da Informação e Comunicação, enquanto instrumentos de apoio no desenvolvimento de habilidades de acesso e uso da informação [Belluzzo 2005].

Embora os mapas conceituais consigam transmitir uma informação tão bem quanto um texto, as representações gráficas são bem mais efetivas, pois facilitam a visualização de relações entre os conceitos relevantes, melhorando a acessibilidade e usabilidade, pois apresentam maior facilidade na localização de informações.

Um mapa conceitual sobre um determinado conteúdo é apenas uma das possíveis representações de uma estrutura conceitual, o que faz dele um instrumento simples e flexível, podendo ser aplicado a uma variedade de situações e com diferentes finalidades.

Mapa conceitual para a identificação de riscos de requisitos

Risco é um evento inerente a todo projeto de desenvolvimento de Software. Portanto, é necessário que essas adversidades sejam gerenciadas de modo a alcançar o sucesso do projeto e a qualidade desejada do produto final. Para gerenciar riscos

de modo eficaz, é importante aplicar as atividades de riscos descritas na abordagem de gerenciamento de riscos escolhida para o projeto em questão.

Diante de todas as atividades do processo de gestão de riscos, uma eficiente identificação de riscos é essencial para a continuidade bem sucedida do processo, que deve ser realizado de forma repetitiva e cíclica durante todo o ciclo de vida do projeto, pois não se pode gerenciar aquilo que não se conhece.

A taxonomia de riscos de desenvolvimento de software disponibilizada pelo SEI, classifica os riscos em três grandes classes apresentadas anteriormente: Engenharia do Produto, Ambiente de Desenvolvimento e Restrições do Programa. As classes taxonômicas são divididas em elementos, estes são constituídos de atributos. Dentre os elementos inclusos na classe Engenharia do Produto, encontra-se o elemento Requisitos. Alguns atributos caracterizam o elemento Requisitos: Estabilidade, Completude, Clareza, Validação, entre outros.

A taxonomia de riscos proposta pelo SEI provê um *framework* para organização e estudo abrangente das questões envolvendo o desenvolvimento de software [Carr et al 1993]. Além disto, serve como base para a estruturação de todos os possíveis riscos de desenvolvimento de software, ambos técnicos ou não técnicos.

Baseado neste pressuposto, um método de identificação de riscos através de Mapas Conceituais foi elaborado, baseado na taxonomia oferecida pelo SEI. O objetivo desse método é expor possíveis riscos de desenvolvimento de software que podem afetar negativamente o projeto, impactando nos custos, qualidade e cronograma do projeto.

Quando os requisitos estão especificados incompletamente, o desenvolvimento do software pode ser baseado em hipóteses ou suposições por parte da empresa fornecedora do software, podendo não atender às expectativas e necessidades do cliente. No caso da omissão de alguns requisitos, pode implicar em alterações no custo e cronograma do projeto, pois os requisitos não foram completamente documentados.

A **Figura 3** apresenta uma pequena parte do mapa conceitual utilizado para identificação de riscos de requisitos, mais especificamente no contexto de Riscos de Completude. Quando um requisito não foi completamente definido ou especificado, não atende aos critérios de aceitação dos desenvolvedores, ou até mesmo omitido, então está caracterizado um Risco de Completude.

A utilização do método é bastante simples e intuitiva, iniciando a partir do conceito raiz, deve-se percorrer o caminho pelo ramo que se encontra, associado com a situação do requisito que se deseja verificar. Ao final de cada ramo pode-se chegar a algum atributo de risco de requisito, como apresentado na **Figura 3**, um risco de completude do requisito. Dado um requisito, existem duas possibilidades de finalizar o processo de identificação de riscos, através do mapa conceitual exposto na **Figura 3**:

1. Atingindo o conceito Riscos de Completude, pode-se concluir, dependendo do caminho percorrido, que a especificação do requisito utilizado para percorrer o mapa está incompleta ou que o requisito está omisso, podendo implicar em um possível evento negativo para o projeto.
2. Caso os ramos sejam seguidos corretamente e o resultado no final não implicou na indicação de um possível risco, então existe um indício de que a especificação do requisito avaliado esteja adequada, tornando pouco provável a ocorrência de um risco de completude causado pelo mesmo.

Para facilitar o entendimento do uso de mapas conceituais na identificação de riscos de requisitos, o Quadro 1 apresenta um projeto totalmente fictício chamado TEX (Test eXpert) e o estado de dois requisitos desse projeto.

Com base na especificação dos requisitos do projeto TEX, pode-se analisar se há a possibilidade de algum risco afetar negativamente o desenvolvimento do projeto de software utilizando o método de identificação de riscos através de mapas conceituais.

De posse destas informações sobre o projeto TEX, na **Figura 4** é apresentada a sequência de ações, com o uso do mapa conceitual, referente à

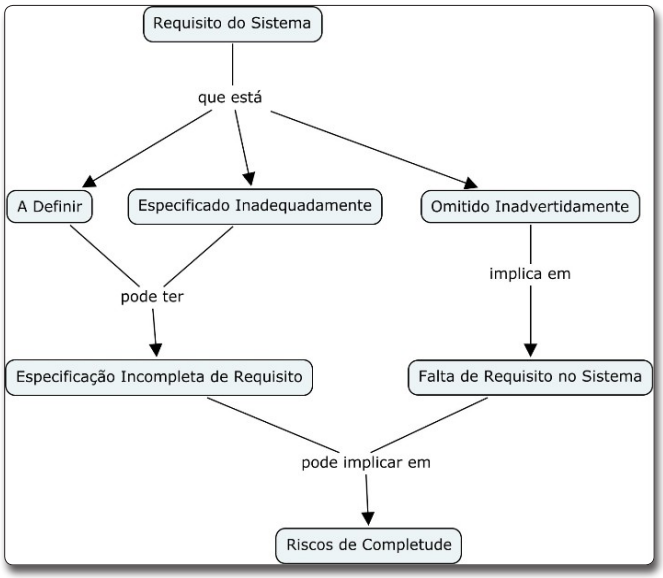


Figura 3. Mapa Conceitual genérico para identificação de Riscos de Completude de Requisitos em um projeto de desenvolvimento de software.

análise do requisito do sistema **Gerar Relatórios de Execução dos Testes** para levantamento de possíveis riscos associados a esse requisito. O mesmo está no topo do mapa conceitual indicando que será o requisito examinado. De acordo com a especificação desse requisito, o responsável pela identificação de riscos pode verificar se o requisito está Especificado Inadequadamente, Omitido, ou com algum ponto A Definir. Como alguns parâmetros não foram definidos ainda para este requisito, então se segue através do ramo **A Definir**. A partir desta definição, pode-se concluir que a há uma **Especificação Incompleta do Requisito** e, consequentemente, pode implicar em um **Risco de Completude,** no qual as métricas e a forma de interação com o usuário para visualização do relatório não foram definidas para aquele requisito. Esse tipo de risco é apenas uma classificação adotada pelo SEI, para melhor entendimento da sua natureza.

Analisando o segundo requisito **Criar Planos de Teste**, observa-se que a especificação não satisfaz a nenhuma das opções de conceitos que segue o Requisito do Sistema, finalizando a busca por riscos de completude relacionados a esse requisito. Pode-se concluir então, que é pouco provável que este requisito implicará em algum risco de completude para o desenvolvimento do projeto.

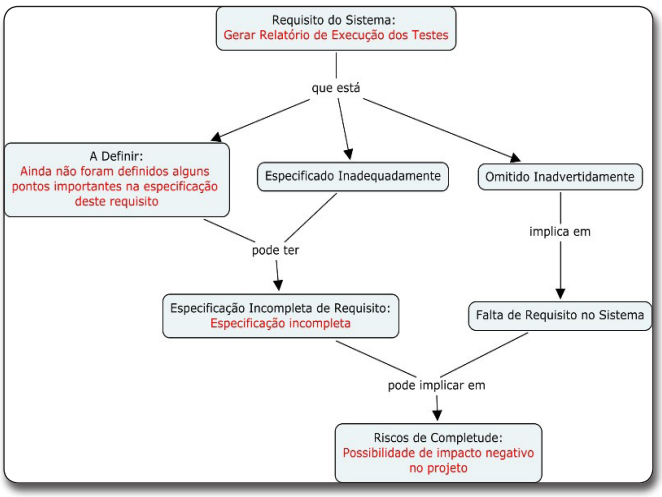


Figura 4 . Mapa Conceitual para análise do requisito "Gerar Relatório de Execução dos Testes".

É importante elucidar que a técnica de identificação de riscos através de mapas conceituais pode ser utilizada em conjunto com qualquer abordagem de gerenciamento de riscos que apresente a atividade de identificação. O método tem apenas o objetivo de expor os possíveis riscos de requisitos em projetos de desenvolvimento de software que podem afetar negativamente o projeto. Um conjunto de soluções para os riscos identificados são elaborados em outra etapa do processo de gerenciamento de riscos após as atividades de planejamento, análise e priorização dos mesmos.

Considerações Finais

A finalidade deste artigo foi apresentar uma abordagem no processo de identificação de riscos de requisitos em projetos de desenvolvimento de software através da utilização de mapas conceituais, auxiliando o aprendizado dos profissionais da

O projeto TEX tem como objetivo criar uma ferramenta web de gerenciamento de casos de teste de requisitos para uma empresa especializada em avaliação de produtos de software.

Um requisito desse software, considerado essencial na especificação, é "Gerar Relatórios de Execução dos Testes". Segundo a documentação, o objetivo deste requisito é mostrar ao usuário uma visão geral dos resultados das execuções de testes e valores de métricas pré-definidas para controle e monitoramento dos testes. No entanto, ainda não foram definidas que métricas serão utilizadas, tampouco é mencionada a forma como o relatório será visualizado.

Existe outro requisito, também considerado essencial, chamado "Criar Planos de Teste", que permite a inclusão de planos de teste de um projeto cadastrado no sistema. Os campos para preenchimento na tela do usuário foram definidos de acordo com um conhecido padrão para documentação de teste de software chamado IEEE 829-1998. Ele vai receber como entrada uma listagem dos requisitos, o plano do projeto e o cronograma. Esses três itens já foram especificados e implantados anteriormente. As pré-condições, pós-condições e cenários principal, alternativos e de exceção desse requisito foram definidos de maneira clara e toda a equipe do projeto está ciente do que se trata esse requisito e como os dados do plano de teste serão armazenados.

Quadro 1. Exemplo do Projeto TEX

área de tecnologia da informação (TI). É fato que este tema é bastante novo e requer amadurecimento e aplicações práticas para uma avaliação efetiva.

No entanto, mapas conceituais são uma forma de representação gráfica que se mostra como uma alternativa de estruturação do conhecimento em um dado domínio. Seu uso vem sendo bastante recomendado para avaliação do processo de aprendizado. Nesta ótica, o uso de mapas conceituais na identificação de riscos de requisitos pode trazer diversos benefícios, entre os quais podemos destacar:

1. Aprimoração do conhecimento em riscos pela equipe de desenvolvimento de software. Uma vez que o risco muitas vezes é considerado algo abstrato, um melhor conhecimento em riscos pode ajudar a reduzir este nível de abstração.
2. Consequentemente, o nível de compreensão da especificação de cada requisito torna-se melhor, reduzindo a incidência de defeitos e falhas no produto desenvolvido.
3. Diversos requisitos apresentam semelhanças, o que pode acarretar em riscos semelhantes. Isto pode permitir ao reuso de mapas, agilizando e amadurecendo mais ainda a atividade de identificação de riscos.
4. Em relação à comunicação, mapas conceituais se apresentam como uma estrutura que permite maior visualização e acessibilidade das adversidades envolvidas no projeto. Portanto, após a identificação, pode ser utilizado também na atividade de comunicação dos riscos.

Desta forma, espera-se que os mapas conceituais também ajudem a aumentar as chances de sucessos de projetos ao se incorporarem na gerência de riscos de projetos de software. ●

Referências

[Ausubel et al 1980] Ausubel, D. P.; Novak, J. D.; Hanesian H. Psicologia Educacional. 2ª Ed. Editora Interamericana. Rio de Janeiro, 1980.

[Belluzzo 2005] Belluzzo, R. C. B. O Uso de Mapas Conceituais para o Desenvolvimento de da Competência em Informação: Um Exercício de Criatividade. 2ª Ed. Competência em Informação na Sociedade da Aprendizagem. Editora Kairós. São Paulo. 2005.

[Boehm e De Marco 1997] Boehm, B. W.; De Marco, T. (1997) Software Risk Management. IEEE Software. IEEE Computer Society Press. pp 17-19.

[Car et al 1993] Carr, M. J., Konda, S.L., Monarch, I., Ulrich, F. C., Walker, C. F. Taxonomy Based Risk Identification. Software Engineering Institute, Carnegie Mellon University, EUA, 1993.

[Gusmão 2007] Gusmão, C. Um Modelo de Processo de Gestão de Riscos para Ambientes de Múltiplos Projetos de Desenvolvimento de Software. Tese de Doutorado, Universidade Federal de Pernambuco/Brasil, 2007.

[Hossenloop e Bass 2007] Hossenloop, R., Bass K. Unearthing Business Requirements: Elicitation Tools and Techniques, Management Concepts, Incorporated, 2007.

[Novak e Gowin 1999] Novak, J. D.; Gowin D. B. Aprender a Aprender. 2ª Ed. Edições Técnicas. Lisboa: Plátano, 1999.

[Pressman 1995] Pressman, R. Engenharia de Software. 1ª Ed. Makron Books. São Paulo, 1995.

[PMI 2004] PMI – Project Management Institute. A Guide to the Project Management Body of Knowledge. ANSI/PMI 99-01-2004. Project Management Institute, 2004.

[Prikladnicki 2003] Prikladnicki, R. Uma Proposta de Utilização de Mapas Conceituais em um Contexto de Desenvolvimento Distribuído de Software. Relatório, Programa de Pós-Graduação em Ciências da Computação – PUCRS. Rio Grande do Sul. Brasil. 2003.

[Rolim e Oliveira 2006] Rolim, L. H. M. L.; Oliveira, J. M. P. Utilização de Mapas Conceituais em Engenharia de Software: Projetando uma Ferramenta Case. Anais no XII ENCITA – Encontro de Iniciação Científica e Pós-Graduação do ITA. São Paulo. Brasil. 2006.

[Trigo et al 2008] Trigo, T. R; Gusmão, C. M. G.; Lins, A. V. CBR Risk – Risk Identification Method Using Case Based Reasoning in 5°CONTECSI - International Conference on Information Systems and Technology Management. São Paulo. Brazil. 2008.

ITIL - Information Technology Infrastructure Library

Aprendendo um pouco mais sobre Governança de Tecnologia de Informação

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

Alex Sandro Barreto Rodrigues
asbrodrigues@yahoo.com.br
É profissional da área de Governança de TI, Master Business Administrator em Gerência de Projetos (FGV), Foundation Certificate in IT Service Management - ITIL, Professor da graduação e pós-graduação das Unidades de Negócio e de Sistemas da FAESA (Vitória - ES), Professor de pós-graduação da UCL (Faculdade do Centro Leste) (Vitória - ES), Criador e Coordenador do LIG (ITSMF) do Espírito Santo. Atuante desde 1997 em projetos, consultorias e treinamentos relacionados à Tecnologia da Informação.

No artigo "Governança de Tecnologia de Informação: Uma Visão Integrada à Engenharia de Software", publicado na 15ª edição da Engenharia de Software Magazine, foram apresentados os conceitos da Governança de TI e discutiu-se sobre a importância da Tecnologia da Informação estar alinhada aos negócios para apoiar as organizações no alcance de seus objetivos estratégicos.

Dando continuidade ao tema Governança de TI, este artigo trata a Gestão de Serviços de TI, destacando como a ITIL (Information Technology Infrastructure Library) pode auxiliar os evolvidos em projetos, manutenção, suporte e operação dos serviços da Tecnologia da Informação.

O termo Tecnologia da Informação é bastante amplo e, para este artigo, deve ser entendido como os recursos tecnológicos e computacionais para armazenamento, gerência e uso de dados para a obtenção de informação e conhecimento. Sendo assim, consideram-se como componentes da Tecnologia de Informação: hardware (incluindo-se dispositivos e periféricos), software, redes de telecomunicações e mecanismos de armazenamento e gerência de dados.

De que se trata o artigo?
Este artigo é complementar ao artigo "Governança de Tecnologia de Informação: Uma Visão Integrada à Engenharia de Software", publicado na 15ª Edição da Engenharia de Software Magazine e apresenta os principais aspectos relacionados à ITIL (Information Technology Infrastructure Library).

Para que serve?
Fornecer conhecimento essencial sobre Gestão de Serviços de TI e sobre a ITIL para organizações, pesquisadores, estudantes e profissionais de software.

Em que situação o tema é útil?
Organizações que realizam ou desejam realizar Governança de TI, especialmente na Gestão de Serviços de TI. Pesquisadores, estudantes e profissionais de TI que buscam entendimento sobre o tema.

Do ponto de vista organizacional, tradicionalmente, a Tecnologia da Informação se materializa por meio de uma área funcional. Ou seja, a Tecnologia da Informação é representada como uma 'caixinha' no organograma da organização, juntamente com áreas funcionais como Recursos Humanos, Produção, Marketing e Finanças. Essa forma de se estruturar, adotada pela maior parte das organizações, torna possível a percepção de que elas se baseiam em modelos hierárquicos para definir e representar suas estruturas de poder e responsabilidades.

Esse tipo de abordagem, que privilegia a especialização, acaba por levar as organizações a segregarem a Tecnologia da Informação em áreas menores como, por exemplo, Desenvolvimento, Suporte, Redes, Telecomunicações, Segurança e Banco de Dados. Para cada uma dessas áreas normalmente são definidas equipe e gerência específicas que, dada a limitação das interfaces entre as áreas e o baixo foco estratégico, comumente tratam as demandas da Tecnologia da Informação analisando apenas o próprio ponto de vista.

Por exemplo, não é raro uma organização adquirir um software corporativo e implantá-lo sem que a equipe de Suporte tenha conhecimento dos procedimentos que devem ser adotados para solução de erros e recuperação de falhas, caso os usuários relatem esses tipos de ocorrências. Em casos como esse, o conhecimento da equipe de Suporte terá que ser construído com o software já em operação, o que pode causar uma série de transtornos aos usuários e ainda gerar descrédito para as áreas de Desenvolvimento e Suporte.

Outra ação comum é a área de Desenvolvimento disponibilizar um software para utilização pela organização sem considerar as variáveis envolvidas no ambiente real de operação. Surpresas como lentidão, acesso negado e outros problemas podem prejudicar a utilização do software pelos usuários.

Para a Gestão de Serviços de TI, esses transtornos podem ser resultados da falta de alinhamento da Tecnologia da Informação ao negócio.

Mas, o que fazer para aumentar o alinhamento entre Tecnologia da Informação e negócio?

Buscando responder a essa questão e apoiar as ações necessárias para a gestão da Tecnologia da Informação foi definida a Gestão de Serviços de TI.

A Gestão de Serviços de TI é uma abordagem que trata pessoas, processos, tecnologia e parceiros de TI como componentes de um Serviço de TI oportunizando, portanto, o tratamento do serviço como um todo e não de apenas um de seus componentes.

Nesse contexto, a ITIL – tema central deste artigo – fornece um conjunto de orientações e melhores práticas para tratar o ciclo de vida dos serviços de TI.

Gestão de Serviços de TI

As organizações são criadas para produzir e realizar a entrega de produtos e serviços à sociedade. Um serviço pode ser qualquer atividade ou benefício que uma parte possa oferecer a outra, que seja essencialmente intangível. Sua produção pode ou não estar ligada a um produto físico. Exemplos de serviços são: o processo de atendimento aos clientes de uma organização, o sistema de correio eletrônico, o gerenciamento da rede de computadores, o sistema de telefonia e o desenvolvimento ou instalação de um software.

As características dos serviços são diferentes daquelas encontradas nos produtos. Por exemplo, é possível ver, apalpar e até mesmo cheirar um computador antes de comprá-lo. No entanto, não é possível ter as mesmas sensações na contratação de um serviço: como apalpar um projeto de desenvolvimento de software? Sendo assim, as principais características que diferenciam os serviços dos produtos são a intangibilidade, a indivisibilidade, a variabilidade e a perecibilidade.

A intangibilidade define que os serviços não podem ser observados, provados, apalpados, ouvidos ou cheirados antes de serem adquiridos. Em outras palavras, serviços são abstratos. A intangibilidade dificulta a identificação de falhas que poderiam ser levantadas antes da aquisição do serviço, uma vez que, enquanto o serviço não estiver em uso, apenas existirão expectativas de que ele atenderá ao conjunto de requisitos solicitado.

Uma maneira de minimizar os riscos devido à intangibilidade do serviço é, antes de adquiri-lo, procurar identificar sinais que evidenciem sua qualidade. Esses sinais podem ser identificados através de conclusões obtidas a partir de comunicações realizadas e de evidências concretas fornecidas pelos participantes dos processos utilizados e das tecnologias empregadas no serviço que está sendo avaliado.

O conceito de indivisibilidade explica que os serviços não podem ser separados do seu prestador e da maneira como este é percebido. Assim, torna-se importante avaliar o profissionalismo, a competência, o comprometimento, a aparência e a conduta do potencial prestador do serviço. Isso pode ser realizado através da análise do portfólio de seus clientes e de feedbacks que estes podem fornecer.

A variabilidade explica que os serviços são altamente variáveis, pois dependem diretamente de quem os executa, de quando e onde são executados. Desse modo, a qualidade de um serviço não pode ser separada da qualidade das pessoas que o executam e do ambiente onde será operacionalizado, o que deve ser considerado para a aquisição de um serviço. Por exemplo, uma empresa pode ter contratos para gerenciar redes de computadores em diferentes organizações, porém, cada contrato tem um desempenho próprio, característico da equipe a ele alocada. Sendo assim, uma vez que são as pessoas que realizam as atividades que resultam nos serviços, e que as pessoas estão sujeitas a erros e variações, é importante identificar onde e quando existem maiores chances de falhas e elaborar mecanismos para corrigi-las ou evitá-las quando necessário.

A perecibilidade define que os serviços não podem ser armazenados para venda ou utilização posterior. Não é possível, por exemplo, consultar sistemas de informação para verificar a disponibilidade em estoque de um serviço.

Conhecidas as principais características dos serviços é preciso entender como a abordagem de Gestão de Serviços de TI é na prática.

Gestão de Serviços de TI pode ser definida, de forma bem simples, como a integração entre pessoas, processos e tecnologias para entrega de Serviços de TI alinhados aos requisitos de negócio da organização. A utilização dessa abordagem permite a definição de atributos para medir o valor do serviço entregue em termos de qualidade, custo e alinhamento estratégico.

A **Figura 1** mostra as três dimensões consideradas pela gestão de serviços de TI: pessoas, processos e tecnologias.

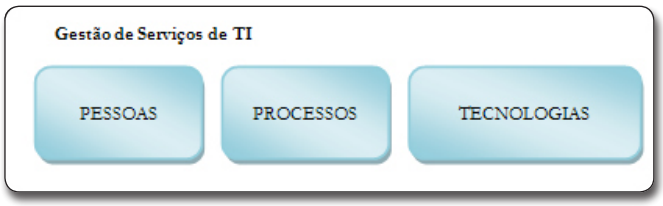


Figura 1. Dimensões da Gestão de Serviços de TI.

Pessoas possuem as competências necessárias para o planejamento, execução, monitoramento e melhoria dos serviços. Processos definem a abordagem a ser seguida para colocar em prática a Gestão dos Serviços de TI. E, por fim, Tecnologias são os elementos tecnológicos - hardware, software, mecanismos de armazenamento e gerência de dados e redes de telecomunicações - necessários para manter o serviço disponível (por exemplo: servidores, dispositivos de rede, sistemas operacionais e sistemas de monitoramento, entre outros).

O fundamento de alinhamento da TI ao negócio, presente na Gestão de Serviços de TI, pode ser percebido, por exemplo, quando a área de Vendas de uma empresa de varejo requisita (e é atendida) à área de Tecnologia da Informação um serviço de correio eletrônico que esteja disponível vinte e quatro horas por dia, sete dias por semana e sem limite de tamanho para as caixas de email de seus vendedores. Nesse caso, o sistema de correio eletrônico corporativo é um serviço de TI e os requisitos de tamanho e disponibilidade do serviço de email são as necessidades de negócio da área de Vendas que devem ser consideradas para a entrega do serviço.

No contexto da Gestão de Serviços de TI, a área da Tecnologia da Informação assume o papel de prestadora de serviços e passa a gerenciar sua relação com as demais áreas de negócio por meio de acordos de nível de serviço (SLA – Service Level Agreement), que são pequenos contratos estabelecidos para medir a qualidade dos serviços de TI entregues. Estabelecer que a disponibilidade do sistema de correio eletrônico corporativo deverá ser igual ou superior a 95% ao mês é um exemplo de SLA.

Análogo a um "menu" de restaurante, na Gestão de Serviços de TI os serviços de TI devem ser organizados em um Catálogo de Serviços de TI, que contém a descrição, os atributos, as restrições e, em alguns casos, o preço do serviço. O serviço de correio corporativo, por exemplo, pode possuir preços diferentes que variam em função do tamanho da caixa de email. Assim, uma área de negócios pode escolher um serviço avaliando o melhor custo benefício para o seu orçamento e, inclusive, decidir se o melhor fornecedor para esse serviço é interno ou externo a organização.

A Gestão de Serviços de TI traz um novo modelo para apoiar a gestão da Tecnologia da Informação. Na estrutura de TI tradicional, os componentes de hardware, software, armazenamento de dados e redes são considerados o objetivo principal da gestão. Consequentemente, o foco recai principalmente na tecnologia e os requisitos de negócio podem ser deixados em segundo plano.

Por outro lado, a Gestão de Serviços de TI privilegia as necessidades e requisitos do negócio e considera a tecnologia, juntamente com pessoas e processos, como um conjunto integrado para a entrega de serviços de valor para organização. Nesse novo modelo, a área da Tecnologia da Informação mantém sua estrutura organizada em diferentes áreas como Desenvolvimento, Suporte, Redes e Banco de Dados. No entanto, os processos são organizados de tal forma que as pessoas, ao executá-los, utilizam os recursos tecnológicos para realizar o alinhamento entre Tecnologia da Informação e necessidades do negócio.

Atualmente, o mercado dispõe de alguns modelos e padrões destinados à Gestão de Serviços de TI como, por exemplo, o MOF (Microsoft Operation Framework) e a ISO/IEC 20000. Apesar de existirem diversas propostas, ao analisar o mercado, é perceptível que as organizações têm, na maioria das vezes, optado por utilizar a ITIL como principal apoio para implementar esse novo paradigma de gestão.

ITIL - Information Technology Infrastructure Library

A ITIL descreve um conjunto de melhores práticas para gestão dos serviços de TI. Foi criada no final dos anos 80 pela CCTA (Central Computing and Telecommunications Agency) para atender a uma necessidade do governo Britânico e recebeu esse nome devido à quantidade de livros gerados para descrever as melhores práticas.

A ITIL passou por duas atualizações e atualmente está em sua 3ª versão, chamada ITIL v3. Trata-se de um conjunto de cinco livros com orientações para a gestão dos serviços de TI, considerando os estágios do ciclo de vida dos serviços de TI ilustrado na **Figura 2**.

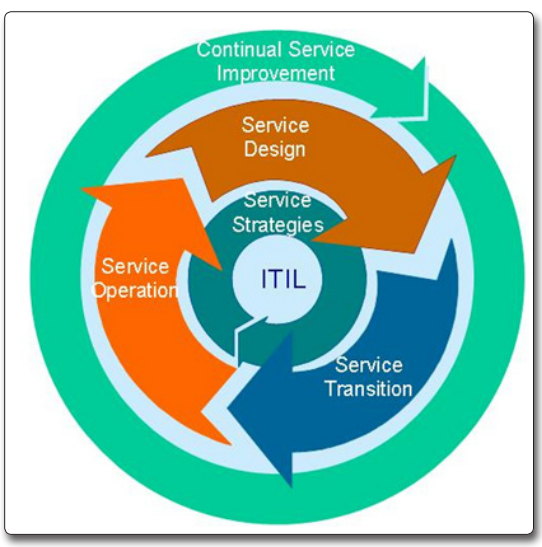


Figura 2. Ciclo de vida dos serviços de TI

Analisando-se a **Figura 2**, percebe-se que o ciclo de vida de um serviço de TI é contínuo, ou seja, seus estágios se repetem enquanto o serviço de TI estiver disponível para a organização. A seguir é realizada uma breve descrição de cada estágio do ciclo de vida de serviço.

No estágio Estratégias de Serviço (Service Strategies) são tratados os aspectos relacionados ao alinhamento entre negócio e Tecnologia da Informação. Uma empresa de varejo, por exemplo, pode utilizar um site na Internet como parte de sua estratégia para aumentar a receita de vendas. Nesse estágio, a Gestão dos Serviços de TI deve buscar entender as necessidades e os requisitos de negócio para entregar um serviço de TI alinhado aos propósitos estratégicos da organização.

No estágio Desenho do Serviço (Service Design) são tratadas arquiteturas, processos, políticas e documentação necessárias para atender aos requisitos de negócio atuais e futuros. No exemplo do site, a Gestão de Serviços de TI deve identificar os requisitos do negócio, definir os requisitos do serviço e desenhar a solução (serviço). A definição de métricas, métodos de medição e dos processos necessários para as fases de transição, operação e melhoria dos serviços são atividades desse estágio do ciclo de vida.

No estágio Transição do Serviço (Service Transition) é realizada a inserção, em ambiente de produção em plena operação, de um serviço que acabou de sair do estágio de desenho de serviço. A inclusão de um novo serviço no ambiente de TI requer um planejamento para transição do serviço. Por exemplo, no caso do site, os softwares envolvidos na solução devem ser testados, validados e depois liberados para instalação, e as equipes responsáveis pelo suporte devem ser treinadas antes do serviço entrar em operação.

No estágio Operação do Serviço (Service Operation) é realizada a coordenação e execução das atividades necessárias para entregar e suportar os serviços de TI dentro dos níveis de serviço estabelecidos junto aos clientes. Um nível de serviço descreve o nível de atendimento aos requisitos estabelecidos para o serviço. A estruturação de um ponto único para suporte aos usuários do site é um exemplo da operação de serviços.

No estágio Melhoria Continuada do Serviço (Continual Service Improvement) é realizado o gerenciamento de melhorias nos processos do Gerenciamento de Serviço de TI e nos serviços de TI propriamente ditos. No exemplo do site, avaliar os resultados das métricas para suporte e propor ajustes e correções para melhorar o desempenho do serviço ou de seus processos são atividades da melhoria de serviço continuada.

Conforme apresentado anteriormente, a Gestão de Serviços de TI busca integrar de forma eficaz e eficiente pessoas, processos e tecnologias. As melhores práticas e orientações da ITIL buscam guiar a Gestão de Serviços de TI nesse sentido. As recomendações da ITIL são apresentadas na estrutura de atividades, funções e processos para cada estágio do ciclo de vida de serviço.

Para conhecer um pouco mais sobre a ITIL, a seguir são apresentados os principais aspectos relacionados a cada um dos livros que compõem a ITIL v3. Vale ressaltar que os livros da ITIL levam o mesmo nome dos estágios do ciclo de vida de serviços: Service Strategy, Service Design, Service Transition, Service Operation e Continual Service Improvement.

Service Strategy

O livro Service Strategy apresenta orientações sobre como as políticas e processos do gerenciamento de serviços de TI podem ser desenhadas, desenvolvidas e implementadas como ativos estratégicos ao longo do ciclo de vida do serviço. Possui quatro processos considerados essenciais para uma boa gestão dos serviços de TI: Estratégia, Gerenciamento Financeiro de TI, Gerenciamento do Portfólio de Serviços e Gerenciamento da Demanda. Além disso, apresenta conceitos chave para o gerenciamento de serviços de TI: Portfólio de Serviços, Business Case e Business Impact Analisys.

Service Design

O livro Service Design fornece as orientações para trabalhar os aspectos sobre tecnologia e arquitetura e os processos necessários para transição, operação e melhoria do serviço. Contém sete processos: Gerenciamento de Nível de Serviço, Gerenciamento do Catálogo de Serviço; Gerenciamento da Disponibilidade; Gerenciamento da Segurança da Informação; Gerenciamento de Fornecedor; Gerenciamento da Capacidade e Gerenciamento da Continuidade dos Serviços de TI. Em síntese, o desenho de serviço analisa, documenta e avalia os requisitos e restrições que satisfaçam as estratégias do negócio. Para cada serviço são desenhadas soluções considerando níveis de serviço, arquiteturas e métodos de medição, capacidade, disponibilidade, continuidade e segurança. A estratégia para fornecimento do serviço e a escolha de fornecedores também são atividades do desenho do serviço. Apresenta conceitos chave para o gerenciamento de serviços de TI: Catálogo de Serviços de TI, Pacote de Desenho do Serviço, Acordos de Nível de Serviço, Acordos de Nível Operacional e sistemas de informação para o gerenciamento de fornecedores, contratos, disponibilidade e capacidade dos serviços.

Service Transition

O livro Service Transition trata dos conceitos e processos necessários para colocar um serviço em operação levando em consideração o cumprimento dos requisitos de custos, prazo e qualidade. A transição do serviço deve gerar o mínimo de impacto nas operações atuais da organização. A ITIL propõe a utilização de um sistema de gerenciamento do conhecimento de serviços para gerenciar todo o conhecimento e informação gerados durante o ciclo de vida do serviço. Os processos apresentados neste livro são: Gerenciamento de Mudança, Gerenciamento da Configuração e de Ativos de Serviço, Gerenciamento de Liberação e Implantação, Validação e Teste de Serviço, Avaliação e Gerenciamento do Conhecimento.

Service Operation

O livro Service Operation apresenta orientações para gerenciar o cotidiano das operações de TI. O livro está organizado em funções: Central de Serviço, Gerenciamento Técnico, Gerenciamento das Operações de TI e Gerenciamento de Aplicativo

e processos: Gerenciamento de Evento, Gerenciamento de Incidente, Gerenciamento de Problema, Cumprimento de Requisição e Gerenciamento de Acesso.

Continual Service Improvement

O livro Continual Service Improvement trata do alinhamento contínuo dos serviços de TI às necessidades do negócio. Seu escopo é formado pelos processos: Melhoria em 7 passos, Relatório de Serviço e Medição de Serviço.

Implantando as Práticas da ITIL em uma Organização

Algumas pessoas dizem que a ITIL não se implanta, se adota. Em outras palavras, a ITIL pode ser adotada como base para que organizações criem suas próprias metodologias de Serviços de TI. Não existe nenhuma "receita de bolo" para implementar as práticas da ITIL. Seu escopo é abrangente e a escolha do que adotar depende das necessidades da organização.

Se por um lado, a abrangência do escopo da ITIL dificulta a definição do que colocar em prática, por outro, a abordagem estruturada em atividades, funções e processos sugere como caminho mais natural o redesenho dos processos da área de Tecnologia da Informação. Definir um processo para o Gerenciamento de Mudanças de TI fundamentado nas orientações da ITIL é um exemplo de adoção de suas práticas.

Para implementar as práticas da ITIL, uma questão inicial é a definição da responsabilidade pela condução do projeto de aderência à ITIL (recomenda-se que a adoção da ITIL seja tratada como um projeto). Nesse caso, há dois caminhos: conduzir o projeto internamente ou contratar uma consultoria para isso. Independente da opção escolhida para conduzir a implementação das práticas da ITIL, quem vai efetivamente 'movimentar a roda' serão as equipes da organização, uma vez que são elas que executarão os processos definidos.

A identificação das mudanças que serão necessárias nos processos da organização para adequar-se às práticas da ITIL é realizada através de uma análise de maturidade dos processos da organização, onde são percebidas as diferenças entre processos e estrutura atuais da TI e as melhores práticas preconizadas pela ITIL. O resultado da análise orienta a organização sobre quais práticas adotar e quais processos implementar. Algumas empresas têm utilizado abordagens semelhantes à do CMMI para representar o nível de maturidade dos processos.

Paralelamente à análise dos processos e estrutura da TI, tipicamente, é realizada a capacitação das equipes em ITIL.

Identificadas as diferenças entre o cenário atual da TI e as práticas da ITIL, a partir dos resultados da análise de maturidade, deve ser definido o escopo do projeto de aderência à ITIL. Uma prática interessante de apoio ao planejamento do projeto é considerar a entrega de quick wins. Em síntese, quick wins são objetivos intermediários que podem ser alcançados rapidamente pelo projeto. Os resultados chegam mais cedo e, assim, as equipes se mantêm mais motivadas. Estruturar o projeto para entregar gradativamente produtos e processos como, por exemplo, o Catálogo de Serviços de TI, o Gerenciamento de Incidentes e o Gerenciamento de Níveis de Serviço, são exemplos de entregas de quick wins.

Durante a execução do projeto, as atividades estarão voltadas principalmente para a estruturação dos Serviços de TI e o redesenho dos processos. As palavras-chave são necessidades e requisitos de serviços e negócio. A questão básica é garantir o alinhamento da TI aos negócios. Os Serviços de TI serão pensados segundo seu ciclo de vida e os processos serão estruturados para apoiar a gestão da TI. A estruturação de equipes para suporte, a definição de SLA para os Serviços de TI e a implementação de sistemas de informação para apoiar a Gestão de Serviços de TI são exemplos de atividades que podem ocorrer nessa fase do projeto.

Após o encerramento do projeto é importante estabelecer critérios para medir, manter e melhorar os serviços e processos de TI. O ciclo PDCA (Plan, Do, Check, Act) é a abordagem indicada para prover a melhoria contínua dos serviços.

Casos de Sucesso

A ITIL é reconhecida internacionalmente como o conjunto de melhores práticas para a Gestão dos Serviços de TI. Após seu lançamento no Brasil, em 2004, várias organizações nacionais ou internacionais com sede no Brasil iniciaram projetos para aplicar as orientações da ITIL em suas atividades. A seguir são apresentados alguns casos de adoção da ITIL como base para a Gestão de Serviços de TI.

- O Grupo JBS-Friboi, maior frigorífico do mundo, é uma das companhias que mais cresceram no Brasil na última década. Hoje, a empresa fatura 11,5 bilhões de dólares e tem 40 mil funcionários nos quatro países onde atua. O crescimento rápido impediu, porém, que o departamento de TI tivesse tempo de organizar a prestação dos serviços para as áreas usuárias. Havia pouca documentação das práticas e parte importante do conhecimento sobre os processos do grupo estava apenas na cabeça dos profissionais de TI. O projeto para implementação da ITIL, que teve início em junho de 2008, teve o investimento total de 250 mil reais e inclui consultoria, treinamento e investimento em uma ferramenta web para o registro dos chamados online ao service desk.
- A rede de supermercados de atacado Makro implantou a ITIL para gerenciar seus processos de TI e resolveu de vez a questão de prioridade de atendimento aos seus usuários. O Makro adota a ITIL desde 2007 e o primeiro processo de TI implementado foi o de gestão de incidentes.
- Prestadores de serviço também têm utilizado a ITIL como parte de suas estratégias de negócio. Nos últimos três anos, por exemplo, empresas como Nexa, CPM Braxis e Net Service conseguiram obter a certificação de Gestão de Serviços de TI (ISO/IEC 20000) utilizando a ITIL como principal alicerce.

Considerações Finais

A Gestão de Serviços de TI é abordagem que considera a Tecnologia da Informação sob o ponto de vista do negócio. Essa abordagem foi iniciada pelo governo Britânico e, atualmente, é utilizada por várias organizações públicas e privadas ao redor do mundo. A ITIL é considerada o conjunto de melhores práticas para a Gestão dos Serviços de TI e, em 2007, passou por sua 3ª atualização.

Para as organizações, a idéia de criar Serviços de TI alinhados aos propósitos de negócio representa a possibilidade de uma melhor gestão dos recursos envolvidos na estrutura da TI e, em alguns casos, a geração de receitas.

Algumas organizações têm considerado a adoção da ITIL para apoiar a Engenharia de Software na manutenção e distribuição de sistemas. Conforme dito no artigo publicado na 15ª edição da Engenharia de Software Magazine, os gestores de TI, gerentes de projetos e engenheiros de software não são concorrentes na área de TI e, na verdade, estão todos no mesmo barco, devendo remar em uma mesma direção.

A Gestão de Serviços de TI fundamenta-se na harmonia entre pessoas, processos e tecnologias para entrega de Serviços de TI que gerem valor para organização. Sua abordagem tem se mostrado uma excelente oportunidade para integração das tradicionais áreas da TI. Pense nisso! ●

Referências

BOM, J. V., JONG, A., KOLTHOF, A., PIEPER, M., TJASSING, R., VEEN, A. V. D., VERHEIJEN, T., 2007, "Foundations of IT Service Management Based on ITIL V3", ITSMF Library, Van Haren Publishing.

FERNANDES, A. A., ABREU, V. F., 2008, "Implantando a Governança de TI: da Estratégia à Gestão dos Processos e Serviços", 2ª. Edição, Brasport, Rio de Janeiro.

MAGALHÃES, I. L.; PINHEIRO, W. B., 2007, "Gerenciamento de Serviços de TI na Prática: Uma Abordagem com Base na ITIL", Novatec, São Paulo.

Dê seu feedback sobre esta edição!

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

www.devmedia.com.br/esmag/feedback

UML – Explorando outros diagramas

Conhecendo diagramas de utilização menos frequente, mas de grande relevância na modelagem de sistemas

De que se trata o artigo?
Este artigo tem por objetivo apresentar a sintaxe de três diagramas importantes da UML: diagrama de objetos, diagrama de atividades e diagrama de máquina de estados.

Para que serve?
Fornecer aos desenvolvedores ou estudantes da área de sistemas uma linha de entendimento com o intuito de orientá-los a modelar diagramas de menor utilização na UML, mas de grande relevância.

Em que situação o tema é útil?
Para quem ainda não modelou diagramas de objetos, atividades e/ou máquina de estados, ou para quem tem experiência e quer revisar a sintaxe permitida nesses tipos de diagramas.

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, sendo os últimos 11 anos no serviço público. 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.

Dos treze diagramas da versão atual da UML, são essenciais os Casos de Uso, Diagramas de Classes e Diagramas de Seqüências. Mas há outros de relevância comprovada, que podem ser explorados pelos desenvolvedores para refinar sua modelagem. Entre esses, podemos citar o Diagrama de Objetos, o Diagrama de Máquina de Estados e o Diagrama de Atividades.

Veremos a seguir a relevância desses diagramas na modelagem dos sistemas, bem como sua integração com outros modelos da UML.

Diagrama de Objetos

Sabemos que um objeto nada mais é do que a instância de uma classe, num determinado ponto (instante) do tempo. Pois podemos utilizar a mesma definição para designar o diagrama de objetos, tornando-o uma instância do diagrama de classes, representando essa modelagem num determinado instante do tempo. O Diagrama de Objetos utiliza grande parte da notação do Diagrama de Classes.

E qual seria o objetivo de um diagrama desses? Pense que uma modelagem parte de uma abstração, e quanto mais abstraímos mais longe do mundo real estamos, e isso pode nos levar a dúvidas ou interpretações errôneas sobre o que o usuário realmente deseja.

Assim, cada vez que criamos um diagrama de objetos, passamos a trabalhar com os valores dos atributos, ou seja, com dados reais, que nos dizem muito mais do que apenas a identificação dos atributos dentro da modelagem de uma classe.

O diagrama de objetos pode também auxiliar o desenvolvedor no momento de identificar problemas na execução de uma aplicação. Durante a depuração de um programa, podemos ir mapeando o conteúdo dos objetos envolvidos na execução, escrevendo os valores assumidos num diagrama de objetos. Assim, com o diagrama, temos a visão não só do objeto envolvido, mas por expansão dos seus relacionamentos, os objetos vizinhos.

Se considerarmos todos os objetos e relacionamentos possíveis, além de suas multiplicidades, teremos milhares de objetos. É evidente a impossibilidade de exibirmos todas essas instâncias. Por isso, mostramos num diagrama de objetos, criado com intuito de entendimento do modelo, somente um subconjunto que tenha relevância dentro da modelagem. Esse conjunto procura se basear em colaborações. Assim, a partir de um determinado objeto, percorrem-se os relacionamentos existentes, congelando-se um determinado estado do mesmo.

Sabemos que o estado de um objeto significa o conjunto de valores que ele assume num determinado instante. E é a partir do estado de um objeto que se determina o estado dos objetos com ele relacionados.

O Diagrama de Objetos irá demonstrar, portanto, um conjunto de objetos e suas respectivas ligações (instâncias dos relacionamentos), em um determinado instante do tempo (estados dos objetos congelados).

Então, de forma resumida, temos os seguintes usos para um diagrama de objetos:

• **Entendimento e validação do modelo**: permite que o analista valide, no acompanhamento das colaborações entre os objetos, se a abstração do modelo de classes está coerente com o problema em estudo;

• **Depuração de código**: permite que o desenvolvedor mapeie o conteúdo dos objetos, em tempo de execução, a fim de que de forma visual se identifique qualquer erro no código, que venha ou não a gerar exceção no momento da execução;

• **Auxílio no processo de modelagem**: dependendo da complexidade do sistema, o analista pode ter dificuldades de abstrair o problema, gerando sua solução para o modelo de classes. A visão desse diagrama numa versão mais próxima da realidade (diagrama de objetos) pode trazer a clareza necessária para se validar as classes identificas e/ou definir os relacionamentos;

• **Preparar base para casos de teste**: um diagrama de objetos pode auxiliar o projetista a montar os casos de teste do sistema, enxergando os valores de entrada ou determinando os valores de saída.

Notação dos diagramas de objetos

Diferente da classe, não é relevante num diagrama de objetos identificarmos seu comportamento. O foco do diagrama de objetos é o seu estado, portanto, o que há de relevante são os valores dos atributos.

Para identificarmos um objeto, utilizamos a mesma notação de uma classe, sem o compartimento das operações.

No compartimento do nome temos o nome do objeto, acompanhado da identificação de qual instância a partir da qual ele foi criado. Os objetos se diferenciam das classes por se apresentarem sublinhados. E podem ser desenhados com três notações diferentes:

```
nome do objeto : nome da classe
                ou
        : nome da classe
                ou
        nome do objeto
```

A primeira notação traz primeiro um nome aleatório de objeto que será útil se precisarmos usar esse objeto como parâmetro de um método.

A segunda notação indica um objeto anônimo, ou seja, desconhecemos o nome do objeto por ele ser irrelevante, mas conhecemos a classe da qual ele é instanciado.

A terceira notação é a menos indicada, pois desconhecemos o nome da classe. Nesse caso o nome do objeto deve ser claro o suficiente para que o leitor identifique de qual classe ele foi instanciado.

No compartimento dos atributos, não há a necessidade de se citar a visibilidade dos mesmos (privado, público, protegido ou pacote). Basta o nome, acompanhado do sinal de igual (=) e do valor congelado. Veja exemplo na **Figura 1.**

Figura 1. Exemplo de objeto criado a partir de uma classe

Repare que a classe *Cliente* possui quatro atributos e uma operação. Porém ao criarmos o objeto *cliente1* instanciado da classe *Cliente*, apenas preenchemos valores para os atributos. Da mesma forma, note que para os atributos do tipo string, colocamos valores entre aspas simples. Para a data, já não há necessidade. Agora repare no atributo *sexo*. Ele não é de um tipo básico, mas de um tipo enumerado, da classe *ESexo*. Sendo

assim, os valores esperados para esse objeto serão os valores enumerados da classe: *masculino* ou *feminino*.

O relacionamento entre os objetos é feito por meio de *links* (ligações). Um *link* é a instância de uma associação. Sendo assim, repare na **Figura 2**. Veja que o primeiro diagrama é um diagrama de classes que possui as classes *Motorista* e *Veiculo*. O relacionamento entre eles determina que *um motorista é proprietário de nenhum ou vários veículos*. No segundo diagrama que é o de objetos, a classe *Motorista* deu lugar a sua instância *Joao*. Já a classe *Veiculo* deu lugar a sua instância *gol*. Assim, na ligação entre as instâncias, a leitura correta é feita como: *João é proprietário do gol*.

Figura 2. Comparação entre a associação no diagrama de classes e a ligação no diagrama de objetos

Contudo, considerando que a multiplicidade do diagrama de classes anterior é de 0..*, podemos e devemos representar no diagrama de objetos essa diversidade de instâncias. Veja a **Figura 3**.

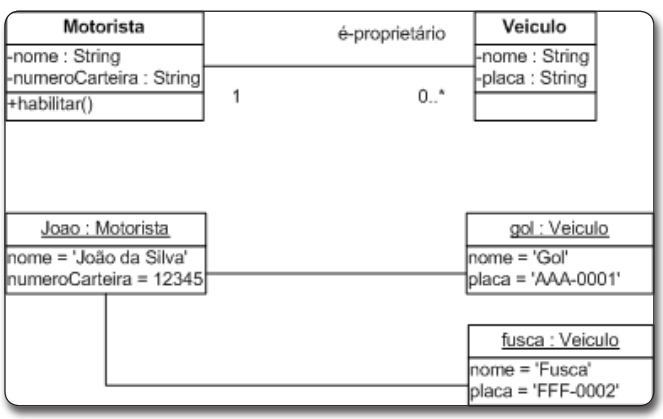


Figura 3. Representação das multiplicidades de uma associação no diagrama de objetos

O nome do fim de associação (*é-proprietário*) é opcional. Torna-se necessário quando uma classe possui mais de um relacionamento com a mesma classe. Sendo assim, no momento de se criar as ligações é preciso ter certeza a qual relacionamento se refere a mesma. Veja exemplo na **Figura 4**. Repare que do lado esquerdo, as instâncias de cidade *rj* e *sp* se relacionam com a instância de vôo *100*. Mas só a ligação não identificaria qual cidade era a origem do vôo e qual era o destino. Por isso, nesse tipo de caso, é imprescindível manter o nome do papel da associação. Assim, podemos saber que o vôo 100 tem o Rio de Janeiro como cidade de origem e São Paulo como cidade de destino. Da mesma forma, fazemos do lado direito. Repare que no primeiro grupo o vôo não tem escalas, pois não há nenhuma ligação com a identificação de escala. Já no segundo grupo, o vôo 101 possui como origem a cidade do Rio de Janeiro, como destino, a cidade de Belo Horizonte, mas há uma escala na cidade de São Paulo.

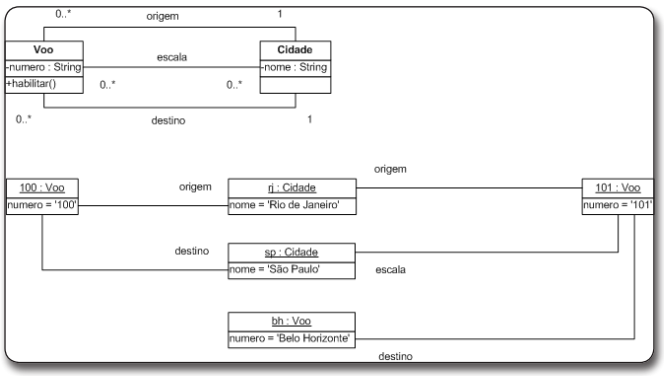


Figura 4. Representação das multiplicidades com nomes de associação diferentes

No caso de um relacionamento de generalização/especialização, se a classe-pai for uma classe abstrata, basta que as classes-filhas apresentem os seus atributos e os atributos herdados da superclasse. Veja exemplo na **Figura 5**. Nesse exemplo, repare que o diagrama de objetos não apresenta a superclasse *Pessoa*. Ela está implícita no diagrama fazendo a subclasse *PrestadorServico* herdar seus atributos.

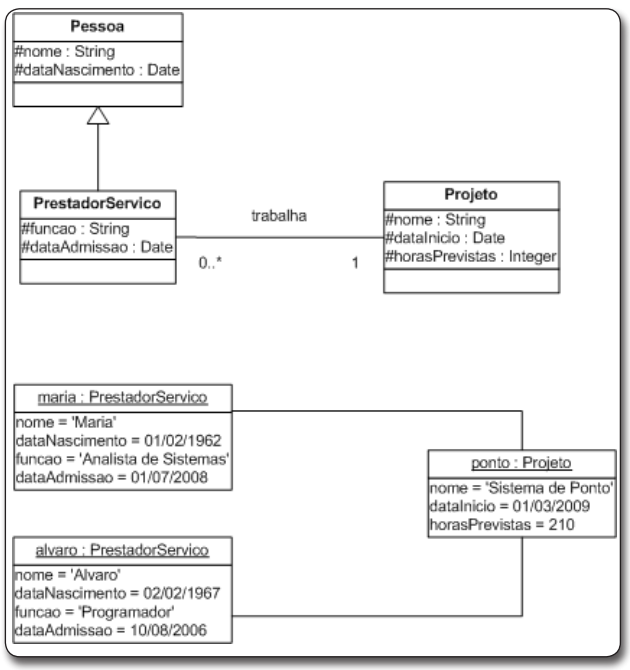


Figura 5. Exemplo de um diagrama de objetos envolvendo herança

Considerando o que foi visto já podemos entender o diagrama de objetos da **Figura 7**, criado a partir do diagrama de classes da **Figura 6**. Nesse diagrama também podemos perceber

uma outra regra inerente ao modelo. Podemos omitir alguns valores de atributos se os mesmos não forem relevantes para o entendimento. Veja que isso acontece com o atributo *descricao* para os objetos *i002* e *i003*.

Repare que uma instância de *Corretor* pode se relacionar com *nenhum ou vários imóveis*. Pois bem, no diagrama de objetos representamos a multiplicidade zero desse relacionamento ao deixarmos o objeto *joao*, instância de *Corretor*, sem qualquer ligação com outra classe. Isso significa que o objeto existe, mas não se relaciona com nenhum outro, ou seja, está com a multiplicidade zero. Já para representar a multiplicidade 1, nesse mesmo relacionamento, colocamos a instância *carla* de corretor se relacionando com a instância *i001* de apartamento. Para representar a multiplicidade muitos, usamos a instância *pedro* de corretor. Ele foi desenhado com ligações a duas instâncias: *i002* e *i003*.

Lido ao contrário, uma instância de *imovel* poderia se relacionar com no mínimo uma e no máximo duas instâncias de *corretor*. Sendo assim, tem-se como exemplo da multiplicidade unitária, as ligações de *i002* com *pedro* e *i003* com *pedro*. Já para representar a multiplicidade de 2, criou-se duas ligações a partir da instância *i001*, ligadas às instâncias *carla* e *vera*. Isso significa que esse imóvel tem dois corretores responsáveis.

Seguindo o mesmo raciocínio, representamos a multiplicidade de 0..* da classe Bairro com a classe Endereço, deixando a instância *bons* de *Bairro* solta, sem nenhuma ligação, indicando que esse bairro não tem nenhum endereço associado, ou seja, não há imóveis sendo vendidos no bairro de Bonsucesso.

Diagrama de Máquina de Estados

Um modelo de estados contém vários diagramas de máquina de estados, um para cada classe, que relacionam eventos e estados. Os eventos representam os estímulos externos e os estados representam os valores dos objetos. Assim, a máquina de estados consiste num comportamento que especifica a seqüência de estados que um objeto atravessa durante sua vida, em resposta a eventos, junto com suas responsabilidades e ações.

As máquinas de estados são empregadas para a modelagem dos aspectos dinâmicos de um sistema. Na maior parte, isso envolve a especificação do tempo de vida das instâncias de uma classe, um caso de uso ou um sistema inteiro.

Booch, *Rumbaugh* e *Iacobson*, no livro *UML: Guia do Usuário – 2ª edição* apresentam de forma bem clara o contexto para utilização de uma máquina de estados:

"Todo objeto tem um tempo de vida. Na criação, o objeto nasce; na destruição, o objeto deixa de existir. Entre esses dois momentos, o objeto poderá agir sobre outros objetos (enviando-lhes mensagens), assim como também poderão agir sobre ele (quando é o destinatário de uma mensagem). Em muitos casos, essas mensagens serão simples chamadas síncronas a operações. Por exemplo, uma instância da classe *Cliente* poderá invocar a operação *pegarSaldo* ou uma instância da classe *ContaCorrente*. Objetos como esses não necessitam de uma máquina de estados para especificar seu comportamento, pois o comportamento atual não depende de seu passado.

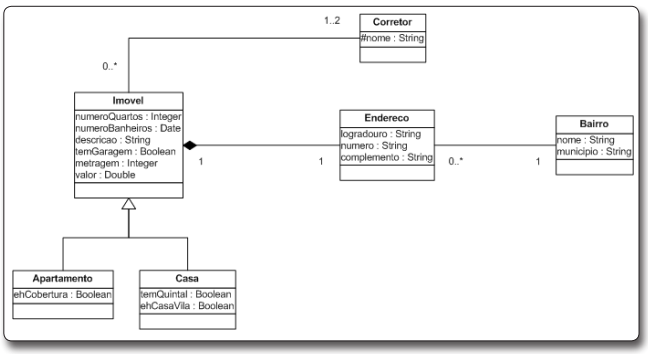


Figura 6. Modelo de classes usado como exemplo para o diagrama de objetos da figura 7

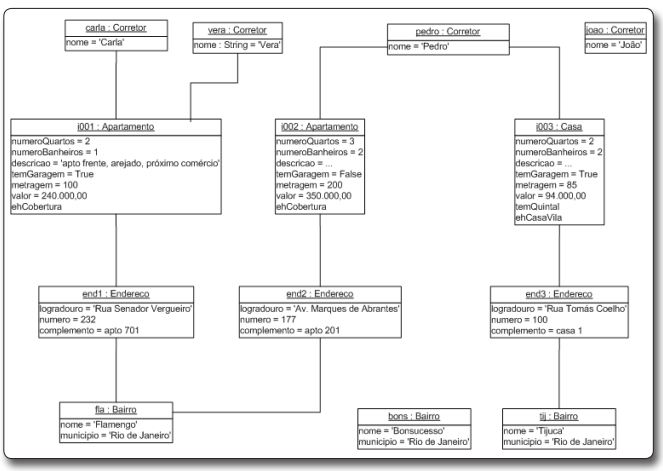


Figura 7. Exemplo de diagrama de objetos criado a partir do modelo de classes de imóveis associados aos corretores de uma imobiliária

Em outros tipos de sistemas, você encontrará objetos que precisam responder a sinais, que são mensagens assíncronas comunicadas entre instâncias. Por exemplo, um telefone celular precisa responder a chamadas telefônicas aleatórias (de outros telefones), a eventos no teclado numérico (do cliente iniciando uma ligação) e os eventos da rede (quando o telefone passa de uma chamada para outra). De modo semelhante, você encontrará objetos cujo comportamento atual depende de seu comportamento passado. Por exemplo, o comportamento de um sistema de orientação de mísseis aéreos dependerá de seu estado atual, como *NaoVoando* (não é uma boa idéia lançar um míssil, enquanto ele se encontra em uma aeronave que ainda está na terra) ou *Procurando* (o míssil não deverá ser armado até você ter uma boa idéia do que servirá como alvo).

O comportamento de objetos que devem responder a mensagens assíncronas ou cujo comportamento atual depende de seu passado é mais bem especificado pela utilização de uma máquina de estados. A máquina de estados é capaz de abranger instâncias de classes que podem receber sinais, incluindo muitos objetos ativos. De fato, um objeto que recebe um sinal, mas não dispõe de uma máquina de estados, simplesmente ignorará esse sinal. A máquina de estados também pode ser empregada para a modelagem do comportamento do sistema inteiro, especialmente de sistemas reativos, que devem responder aos sinais de atores externos ao sistema."

Um **evento** é uma ação, uma ocorrência, disparada pelo usuário ou por um outro sistema, em um determinado instante do tempo. Imagine num sistema o usuário clicar sobre o botão calcular. Isso é um evento, da mesma forma que é um evento, a gaveta de papel da impressora ficar vazia. Não é o usuário que dispara essa ocorrência, mas a própria gaveta que tem o "poder" de informar que ficou vazia.

Existem vários tipos de eventos, que não necessariamente são mutuamente exclusivos. Os mais comuns são:

• **Evento de sinal**

É um evento que acontece quando há recepção de um sinal explícito de um objeto para outro. Por exemplo: a informação de que um *voo* partiu da sua cidade de origem.

• **Evento de mudança**

É um evento causado pela satisfação de uma expressão booleana. Sempre que a expressão tem seu valor modificado de falso para verdadeiro, o evento acontece.

A notação da UML para um evento de mudança é a palavra-chave *when* seguida por uma expressão booleana entre parênteses. Por exemplo: *when (qtdEstoque < limiteMinimoEstoque).*

• **Evento temporal**

É um evento causado pela ocorrência de um determinado momento ou pela passagem de um intervalo de tempo. A notação da UML para um evento temporal é a palavra *when* seguida por uma expressão entre parênteses envolvendo o tempo. Ou ainda, para um intervalo de tempo, é a palavra-chave *after* seguida por uma expressão entre parênteses que equivale a uma duração de tempo. Por exemplo: *when (data = 31 de dezembro); after (1 minuto)*

Um **estado** é uma abstração dos valores e ligações de um objeto, que ficam agrupados no estado, de acordo com o comportamento do objeto. Os estados geralmente correspondem a verbos no gerúndio, pois esse verbo indica continuação, ou à continuação de alguma condição. Por exemplo: *aguardando, calculando, ligado.*

Na UML, um estado é representado por um retângulo de cantos arredondados. O nome do estado é colocado no centro, caso não esteja subdividido em compartimentos. Veja exemplo na **Figura 8**.

Figura 8. Exemplo de notação de estados

Os objetos de uma classe possuem um número finito de estados possíveis – um ou possivelmente algum número maior. Cada objeto só pode estar em um estado de cada vez. Os objetos podem passar por um ou mais estados durante seu tempo de vida. Só incluímos num modelo de estados aqueles estados que são relevantes.

Ao modelarmos um estado devemos ter em mente que esse estado especifica a resposta de um objeto aos eventos de entrada. Contudo, por *default*, todos os eventos são ignorados em um estado, exceto aqueles para os quais a resposta é explicitamente definida. Essa resposta pode incluir a chamada de uma operação ou uma mudança de estado.

Um estado pode ser opcionalmente subdividido em compartimentos separados cada qual por uma linha horizontal. São os compartimentos de *nome* e de *atividades internas*.

O *compartimento do nome* armazenará o nome do estado, como uma string.

O *compartimento das atividades internas* armazenará uma lista de ações ou atividades internas que são executadas enquanto o objeto se apresenta no estado em foco.

Essas transições internas são representadas por expressões que identificam as circunstâncias sobre as quais a ação associada será invocada. Uma transição interna não modifica o estado de um objeto e são representadas com algumas palavras reservadas:

Entry: identifica uma ação que é executada na entrada do estado.

Exit: identifica uma ação que é executada na saída do estado.

Do: identifica uma atividade em andamento, ou seja, que é executada continuamente durante o tempo em que o objeto permanece nesse estado.

Veja exemplo na **Figura 9**. Ao entrar no estado digitando senha, o mesmo aciona a transição interna *entry* que determina que a máscara de formatação da caixa deve ser desabilitada para que o valor seja digitado sem máscara -- *habilitaMascara(false).* Ao sair, a máscara deve ser novamente habilitada, para exibir o valor formatado – *habilitaMascara(true).* Durante a digitação, cada caractere deve ser validado para não permitir a entrada de um intervalo indevido. Por exemplo, se for um valor monetário, não deve ser permitida a entrada de caracteres do tipo string. Como essa atividade é contínua para cada caractere digitado, a mesma é representada com a palavra-chave *do.*

Figura 9. Exemplo de compartimento de transições internas

Uma **transição** é uma mudança instantânea de um estado para outro, ou seja, representa o relacionamento entre dois estados, indicando que houve uma mudança de estado e determinadas ações serão executadas.

É representada graficamente como uma linha sólida na qual o estado de partida é identificado como estado de origem e o estado-alvo é o estado de destino. A linha de transição finaliza com uma seta. Tem a seguinte string de identificação:

```
Assinatura-do-evento [ condição-de-guarda ] /
expressão-ação
```

A *assinatura-do-evento* descreve um evento, com seus argumentos.

A *condição-de-guarda* é uma expressão booleana que determina, quando verdadeira, o início da transição.

É possível termos várias transições a partir do mesmo estado de origem, identificadas com o mesmo evento, diferenciando apenas na sua condição de guarda. Veja exemplo na **Figura 10**. Repare que o sistema está no estado *Aguardando pagamento* até que acontece o evento *Pagamento lançado*. Porém, a ocorrência desse evento não leva sempre a um único estado. Dependendo do que ocorra junto com esse evento, o estado pode ser diferente. Então, se ao lançar o pagamento, o mesmo tiver sido *em dinheiro (condição de guarda = [pagto dinheiro])*, a transição ocorrerá para o estado *Liberando mercadoria*. Se o pagamento for *em cheque (condição de guarda = [pagto cheque]*), o pagamento não pode ainda ser considerado, então a transição ocorrerá para o estado *Aguardando compensação*. Uma vez nesse estado, quando ele receber o evento *comunicação do banco*, com a condição de guarda *cheque compensado*, a transição ocorrerá enfim para o estado *Liberando mercadoria*.

Figura 10. Exemplo de transições entre estados

Um diagrama de estados precisa começar, assim como precisa terminar. Por isso, temos dois estados especiais chamados: *estado inicial* e *estado final*.

Um **estado inicial** é um tipo de estado que indica o local de início na máquina de estados. É representado por um círculo preenchido seguido de uma transição que o levará ao primeiro estado da máquina.

Um **estado final** é um tipo de estado que indica que a máquina de estados concluiu sua execução. É representado por um círculo envolvendo um pequeno círculo preenchido. Veja a notação na **Figura 11**.

Veja na **Figura 12** o exemplo de um diagrama simplificado de máquina de estados referente a um pedido de compra.

Diagrama de Atividades

Um diagrama de atividades tem o objetivo de focalizar um fluxo de atividades que ocorrem internamente em um processamento, dentro de um período de tempo. Sua notação foi

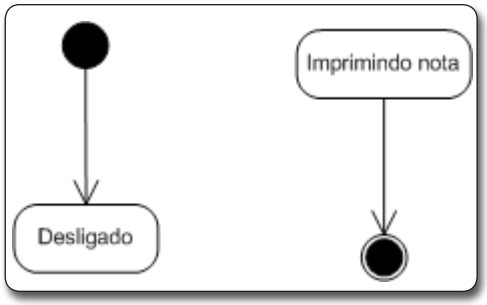


Figura 11. Representação dos estados inicial e final

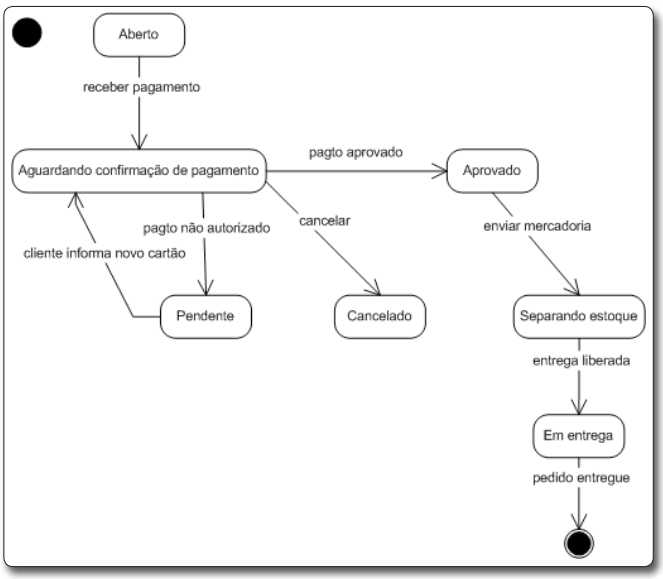


Figura 12. Exemplo de um diagrama de máquina de estado

ajustada na versão 2.0 da UML para torná-lo mais similar a um diagrama de workflow, ou ainda, ao fluxograma tradicional. A vantagem sobre o velho fluxograma é que o diagrama de atividades pode mostrar fluxos de controle seqüenciais e concorrentes.

Um diagrama de atividades pode estar ligado a um classificador, como um caso de uso, um pacote ou a implementação de uma operação. Em outras palavras, isso significa que podemos usar um diagrama de atividades para representar um algoritmo ou um fluxo de trabalho.

Quando começamos a levantar os requisitos de um sistema é comum nos depararmos com um fluxo de trabalho de extrema complexidade, cujas etapas parecem se misturar e nos deixam uma sensação de estarmos perdidos. Para que possamos entender como o processo de trabalho ocorre, e, a partir daí termos segurança para começarmos a modelagem, podemos fazer uso do diagrama de atividades. Modelando o fluxo de trabalho, temos a noção exata do processo do cliente e do que precisa ser informatizado, ou até melhorado dentro desse processo.

O diagrama de atividades é um dos cinco diagramas disponíveis na UML para a modelagem de aspectos dinâmicos de sistema (os outros são: diagrama de máquina de estados, diagrama de seqüências, diagrama de comunicação e diagrama de casos de uso).

Uma **ação** é uma unidade básica existente para a especificação de um comportamento, que venha a representar alguma transformação ou processamento na modelagem do sistema. A ação é atômica e seu comportamento interno não é visível.

Um **nó de atividade** é composto de um grupo de ações aninhadas ou outros nós de atividades aninhados. Não existe uma distinção notacional entre ações e nós de atividades. Ambos são representados graficamente por um retângulo de cantos arredondados, com o nome da ação ou do nó dentro da figura. Se há algo que os distingue é o fato de um nó de atividade admitir partes adicionais.

Veja na **Figura 13** exemplos de ações.

Figura 13. Exemplos de ações

Algumas atividades são executadas indefinidamente até que um evento externo as interrompa, mas a maioria das atividades eventualmente completa seu trabalho e termina por si mesma. A conclusão de uma atividade/ação geralmente indica que a próxima atividade/ação pode ser iniciada. Essa transição é representa pelo **fluxo de controle**, uma linha simples que parte da ação predecessora para a sucessora, sem rótulo de evento, com uma seta de direção na ponta. Veja exemplo na **Figura 14**.

Figura 14. Exemplo de fluxo de controle entre duas ações

É comum termos fluxos de controle que levem a transições seqüenciais num diagrama, mas tão comum quanto é a necessidade de termos desvios de processamento, em função de algum caminho alternativo. Esses desvios, chamados de **nós de decisão**, se dão pelo uso de expressões booleanas e possuem um caminho de entrada e dois ou mais de saída. Quando dois caminhos de controle fundem-se novamente, você também pode voltar a usar o mesmo símbolo, que assumirá a função de **nó de intercalação**. Ambos são representados pela figura de um diamante (losango). No nó de decisão, a figura recebe uma seta, representando um fluxo de chegada, e parte da mesma duas ou mais setas, representando os fluxos de saída, cada uma identificada por uma condição de guarda distinta sem nenhum evento de disparo. Veja exemplo na **Figura 15**.

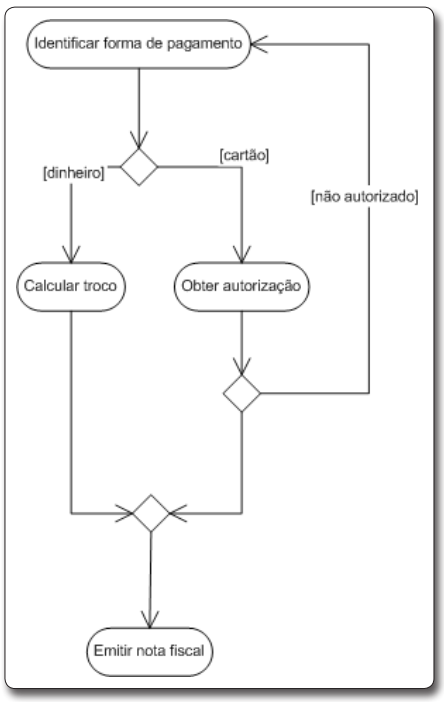


Figura 15. Exemplo de diagrama de atividade com o uso de **nó de decisão e nó de intercalação.**

Comum também é encontrarmos fluxos concorrentes quando estivermos fazendo uma modelagem. Quando isso ocorre, representamos a concorrência no diagrama de atividades com uma barra inicial que é identificada como **nó de bifurcação** e ao final da concorrência a mesma barra, identifica como **nó de união**.

A bifurcação separa um fluxo de entrada em vários fluxos concorrentes, sendo que todos eles são disparados ao mesmo tempo. A bifurcação poderá ter uma única transição de entrada e duas ou mais transições de saída, cada uma representando um fluxo de controle independente e de execução concorrente.

A união é um nó que sincroniza múltiplos fluxos concorrentes, ou seja, concatena fluxos de regiões concorrentes em um único fluxo simples. A união poderá ter duas ou mais transições de entrada e uma única transição de saída. Na união, os fluxos concorrentes são sincronizados, ou seja, significa que cada um aguarda até que todos os fluxos de entrada tenham alcançado a união, para prosseguir com o processamento no fluxo de controle abaixo da barra. A representação gráfica dos nós de bifurcação e união é uma linha grossa (mais grossa que a linha do fluxo de controle).

Veja exemplo na **Figura 16**. Nesse exemplo, após terminar a atividade *abrir arquivo*, o sistema precisa realizar de forma paralela o carregamento do texto e a sua exibição. Porém, só quando ambas as ações estiverem concluídas, é que se pode liberar a edição do texto.

Um outro recurso muito utilizado no diagrama de atividades são as **raias** (*swinlanes*). As raias são usadas para organizar e agrupar ações de acordo com as responsabilidades de uma unidade organizacional ou grupo de trabalho.

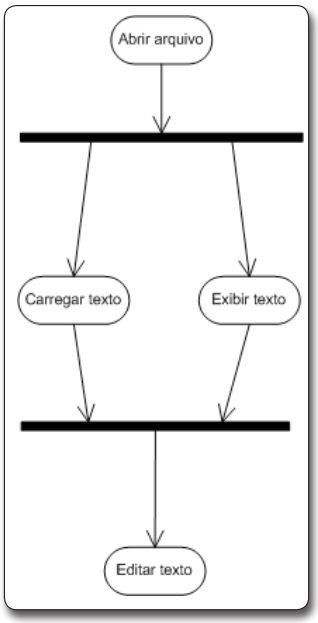


Figura 16. Exemplo de diagrama de atividades com **nós de bifurcação e união**

Um diagrama de atividades pode ser dividido visualmente em raias, criando grupos separados por linhas sólidas verticais de ambos os lados. A ordenação relativa das raias não tem significado semântico. As ações são executadas dentro das raias, de acordo com quem é responsável pela mesma. Fluxos podem atravessar as zonas das raias.

Cada raia tem um nome único em seu diagrama, representando uma entidade do mundo real. Veja exemplo na **Figura 17**.

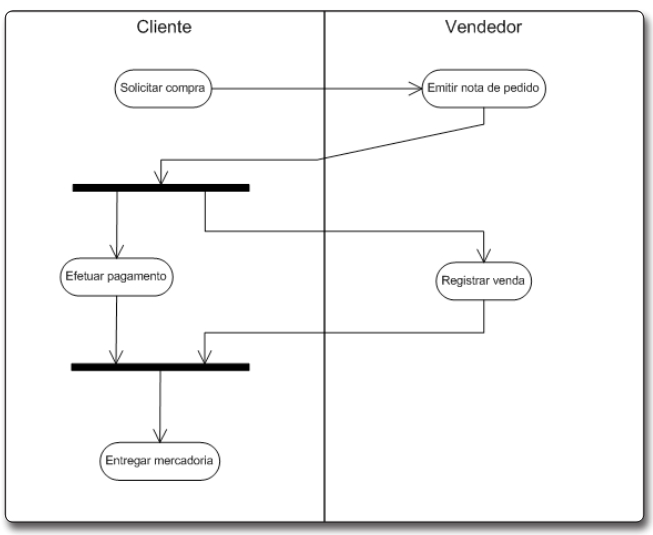


Figura 17. Exemplo de utilização de raias em diagramas de atividades

A UML 2.0 acrescenta uma segunda maneira de mostrar responsabilidades – são os nomes de divisão (*partition names*). Este caso é usado quando não é possível fazer uso das raias. Assim, a solução é colocar o responsável entre parênteses, dentro do retângulo da ação. Veja exemplo na **Figura 18**.

Figura 18. Exemplo de ação com *partition name*

Conclusão

Percebemos que a UML é muito rica quanto aos diagramas que nos oferece. Como uma grande caixa de ferramentas, com instrumentos que sirvam a qualquer trabalho, nem sempre faremos uso de todas ao mesmo tempo, ou seja, de todos os diagramas para qualquer projeto. Mas compreender a capacidade que cada ferramenta (diagrama) é essencial para saber o momento de usá-las e tirar todo o proveito que elas podem nos oferecer. ●

Referências

BOOCH, Grady, RUMBAUGH, James, JACOBSON, Ivar. UML: Guia do Usuário. Rio de Janeiro: Campus, 2005.

MELO, Ana Cristina. Desenvolvendo aplicações com UML 2.0: do conceitual à implementação. Rio de Janeiro: Brasport, 2004.

RUMBAUGH, James et al. Modelagem e projetos baseados em objetos com UML 2: Campus, 2006.

Aplicando MDD na Prática

Utilizando uma Ferramenta de Geração Automática de Código

De que se trata o artigo?

Este artigo tem como objetivo apresentar uma introdução sobre a aplicação de Desenvolvimento Dirigido a Modelos (Model-Driven Development - MDD), entretanto, numa perspectiva prática e eficaz, voltada para a realidade diária dos profissionais da área. O artigo aborda o desenvolvimento de um mini estudo de caso envolvendo um sistema embarcado de tempo-real.

Para que serve?

Trabalhar com modelos, MDD, torna-se mais eficiente e menos propenso a erros do que trabalhar somente em nível de linhas de código-fonte, especificações textuais e documentações como projetos separados e desintegrados. Quando alguém modela uma solução para um problema já o resolveu 50%.

Em que situação o tema é útil?

Aplicar MDD em sistemas embarcados não é somente viável, mas também recomendado. A aplicação correta, conforme padrões preconizados, pode facilitar o desenvolvimento, uma vez que modelos são melhor entendidos e fornecem uma visão muito mais ampla do que informações simplesmente apresentadas na forma de texto plano.

Denis Silva Loubach

dloubach@ieee.org

É aluno de Doutorado do Instituto Tecnológico de Aeronáutica - ITA. Possui os títulos de Mestre em Ciências (2007) na área de Engenharia Eletrônica e Computação pelo ITA e Engenheiro de Computação (2004) pela Universidade de Mogi das Cruzes - UMC. Atua principalmente nas áreas de Sistemas Embarcados e de Tempo-Real, Sistemas Operacionais, Escalonamento, Tolerância a Falhas e Engenharia de Sistemas. Denis Loubach já publicou mais de 15 artigos, entre nacionais e internacionais, e um capítulo de livro. É membro do IEEE e atualmente trabalha como pesquisador e engenheiro de computação em projetos de pesquisa e desenvolvimento no ITA.

Este artigo tem como objetivo apresentar uma introdução sobre a aplicação de Desenvolvimento Dirigido a Modelos (*Model-Driven Development - MDD*). Entretanto, forneceremos uma visão prática, voltada para a realidade diária dos profissionais da área.

Fornecendo uma visão geral sobre o tema, iniciaremos abordando o OMG. *Object Management Group* (OMG) é um consórcio aberto que produz padrões para interoperabilidade entre aplicações. Ele também é responsável pela UML, que é a ideia central por trás do MDD. O OMG também conduz o

desenvolvimento da iniciativa MDA (*Model-Driven Architecture*). Pode-se afirmar que MDA é uma formalização da abordagem MDD.

Conforme definido pela OMG, MDA é uma maneira de se organizar e gerenciar arquiteturas corporativas suportadas por ferramentas automatizadas e serviços, visando definir modelos e facilitar as transformações entre seus diferentes tipos. Em outras palavras, isto significa que MDA visa trabalhar com modelos que são compatíveis entre si e ainda que as transformações entre esses modelos seguem padrões previamente definidos.

Neste artigo, apresentamos uma introdução sobre desenvolvimento de sistemas de maneira tradicional, conceitos e teoria sobre MDD e uma possível ferramenta para se aplicar MDD na prática, levando-se em conta a teoria de MDD. Na seqüência, apresentaremos também um mini estudo de caso envolvendo o desenvolvimento de um sistema embarcado de tempo-real e algumas conclusões que se podem inferir com todo esse processo.

Desenvolvendo da forma tradicional

Mesmo nos dias de hoje, diante de tantas tecnologias diferentes, aptas a "resolver" quase todos os problemas existentes, ainda encontramos muitos projetos sendo desenvolvidos de maneira tradicional. Ou seja, sem muita (ou quase nenhuma) documentação associada.

Muitas são as justificativas apresentadas para tal fato. Algumas delas fazem grande sentido, como por exemplo, a falta de recursos humanos, prazos apertados e orçamento reduzido para investir em ferramentas de software.

Além disso, quando o projeto é multidisciplinar, isto é, pertence ao domínio de conhecimento de engenharia de sistemas, muitos elementos e especificações textuais ainda são complicados para se modelar.

Quando nenhuma destas justificativas se aplica na prática, a falta de treinamento e capacitação dos profissionais também pode influenciar.

Por se tratar de algo relativamente "novo", o MDD ainda não é prática comum.

Em suma, modelos tradicionais de desenvolvimento de sistemas, seja software ou hardware, ainda são largamente utilizados. Talvez isso possa ser explicado pelo fato de tais modelos já serem bem conhecidos e dominados pelos profissionais.

Entretanto, "inovar é preciso". Aos poucos MDD vem ganhando espaço e confiança de que se aplicado apropriadamente, pode fornecer resultados positivos em médio prazo. Não havia quem declarasse que a programação orientada-a-objetos (POO) não iria se estabelecer?

Conceitos e teoria sobre MDD

Pode-se dizer que o desenvolvimento dirigido a modelo (*Model-Driven Development - MDD*) representa uma especificação de diferentes estilos de utilização de modelos para o desenvolvimento de sistemas. Em outras palavras, pode-se dizer que MDD representa uma classe (conceito de orientação-a-objetos), com algumas definições principais que podem ser instanciadas, herdadas ou mesmo sobrecarregadas pelos objetos que instanciam essa classe.

Enquanto o MDD representa um número de estilos para utilizar modelos para desenvolver sistemas (conjunto maior), a arquitetura dirigida a modelo (*Model-Driven Architecture - MDA*) consiste num estilo particular adotado pelo OMG, ou seja, uma instância de MDD.

A MDA representa a especificação de diferentes estilos específicos estabelecidos pela OMG. Dentre eles, destaca-se o padrão da UML 2.0, juntamente com uma filosofia emergente sobre como melhor aplicar modelagem ao processo de desenvolvimento de software.

A MDA define modelos em diferentes níveis de abstração, bem como as transformações que propiciam o mapeamento e o gerenciamento dos relacionamentos entre esses modelos e suas tecnologias de implementação.

Na seqüência, são apresentados os principais termos e definições utilizadas no MDD/MDA, conforme ilustrado na **Figura 1**:

- **Modelo de Computação Independente** (*Computation-Independent Model - CIM*) - Endereça o ambiente e os requisitos do sistema de software, sem considerar a sua estrutura ou processamento;
- **Modelo de Plataforma Independente** (*Platform-Independent Model - PIM*) - Endereça a operação do sistema de software sem detalhes relacionados a uma plataforma particular;
- **Modelo de Plataforma Específica** (*Platform-Specific Model - PSM*) - Combina o PIM com os detalhes relacionados a uma plataforma de hardware específica;
- **Modelo de Plataforma** (*Platform Model - PM*) - Define, para o uso de uma PIM, os conceitos técnicos, elementos e serviços que compõem a plataforma de hardware específica;
- **Modelo de Transformação** (*Transformation Model - TM*) - Especifica e define a transformação requerida para se migrar de uma PIM para uma PSM.

Embora a MDA não seja considerada como um padrão, ela propicia a realização, explicitamente, de chamadas para o uso de vários padrões do OMG.

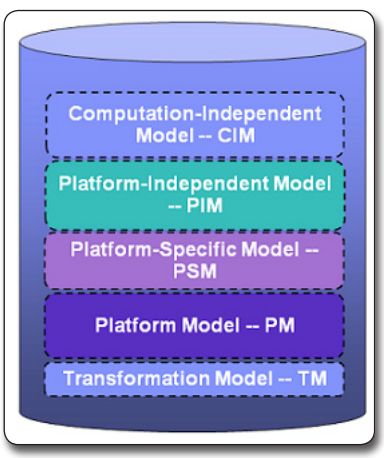


Figura 1. Camadas MDD/MDA

É possível aplicar MDD em sistemas embarcados?

Geralmente, modelar e implementar software embarcado e de tempo-real torna-se mais complexo do que em outros contextos. Isto em virtude de algumas restrições inerentes de aplicações embarcadas, tais como limite de memória e poder de processamento da CPU, entre outras.

Estudos nesse domínio de conhecimento, de software embarcado, mostram que muitos projetos falham por não satisfazerem todos os requisitos propostos ou parte deles.

Em função deste cenário, torna-se necessário o emprego de novas técnicas para construção de software embarcado.

Uma técnica que pode parecer inapropriada, a princípio, é o MDD. Normalmente, emprega-se a modelagem em outras áreas (sistemas *web*, sistemas de gestão ou sistemas puramente de software) como uma via para decisão e verificação de projeto antes de se tomar outras decisões que acarretem num comprometimento maior. Ou seja, modela-se o sistema e as verificações são realizadas, a priori, baseadas no modelo antes de se seguir com o desenvolvimento.

Modelos permitem realizar abstrações de projeto durante suas fases, apresentando-se como um enfoque alternativo para se resolver um problema.

A diferença reside em não pensar somente no problema, mas também no projeto e construção de software. De acordo com o MDD, ferramentas podem levar a modelos em um alto nível de abstração e gerar códigos fonte eficientes e robustos, com qualidade, confiabilidade e segurança (*safety*). Modelos trazem informações visuais ricas referentes à estrutura e ao comportamento de um sistema.

Trabalhar com modelos torna-se mais eficiente e menos propenso a erros do que trabalhar somente em nível de linhas de código-fonte, especificações textuais e documentações como projetos separados e desintegrados.

Portanto, aplicar MDD em sistemas embarcados não é somente viável, mas também recomendado. A aplicação correta, conforme padrões preconizados, pode facilitar o desenvolvimento, uma vez que modelos são melhor entendidos e fornecem uma visão muito mais ampla do que informações apresentadas na forma de texto plano.

MDD, UML & SysML

Conforme mencionado no início do artigo, UML é a uma das bases do MDD. Adotada pela primeira vez em novembro de 1997, a UML, como linguagem visual, vem sendo utilizada como padrão notacional para representar as abstrações do software. Fundamentalmente, ela define dois conjuntos de diagramas para exibir os relacionamentos entre objetos e elementos de sistemas computadorizados: Diagramas Estruturais e Comportamentais.

Entretanto, quando o domínio do sistema que se está desenvolvendo vai além dos limites do software, como, por exemplo, um sistema embarcado, a UML precisa de um complemento para a modelagem utilizando engenharia de sistemas. Este complemento pode ser obtido através da SysML.

A Linguagem de Modelagem de Sistemas (*Systems Modeling Language - SysML*) consiste numa linguagem de modelagem de domínio específico para o desenvolvimento de aplicações de Engenharia de Sistemas. Ela suporta a especificação, análise, projeto, verificação e validação para uma larga amplitude de sistemas e sistemas-de-sistemas, que podem incluir hardware, software, informações, processos, pessoas e construções.

A SysML, originalmente desenvolvida em 2003, constitui-se como um projeto de especificação de código aberto. Trata-se de uma especificação de domínio público, disponível para ser baixada via Internet e que inclui licença de código aberto para sua utilização e distribuição.

Esta linguagem define-se como uma extensão da OMG UML 2.1 *Superstructure Specification* e deve ser suportada por padrões de interpolaridade, envolvendo a XMI 2.1, pertencente à OMG. A XMI constitui o padrão para intercâmbio de modelos entre ferramentas de modelagem aderentes à UML 2.1. A **Figura 2** ilustra a extensão da UML propiciada pela SysML.

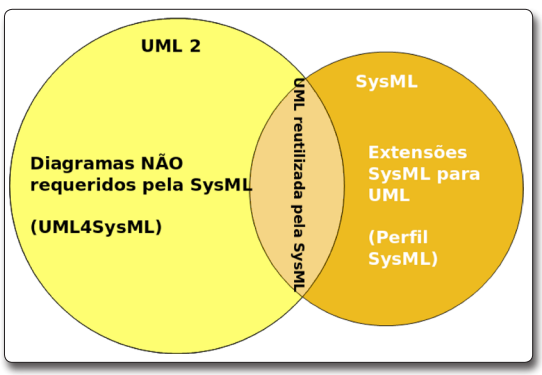


Figura 2. UML & SysML

Tais conceitos serão explorados no mini estudo de caso apresentado neste artigo, justificando, portanto a necessidade de se empregar UML e SysML juntas. Talvez, para sistemas unicamente de software não se aplique SysML. Entretanto, a SysML trouxe um benefício relativamente considerável para a modelagem de sistemas que é o diagrama de requisitos, não existente na UML.

Ferramentas CASE disponíveis

Procurando na literatura, podemos encontrar inúmeras ferramentas CASE que propiciam desenvolver sistemas compatíveis com MDD.

Para desenvolver o sistema embarcado de tempo-real do mini estudo de caso deste artigo, de acordo com MDD, escolheu-se uma ferramenta denominada *Rhapsody*. A justificativa de escolha baseia-se no fato de que tal ferramenta encontra-se entre as principais utilizadas no mercado e na academia. Outras análises também foram realizadas, entretanto não serão apresentadas aqui.

Tais informações farão parte de outro artigo ainda a ser publicado nesta revista, contendo as análises estáticas e dinâmicas baseadas nos códigos-fonte gerados pela ferramenta, análise de métricas como de *Halstead* e *McCabe*, comparações com códigos desenvolvidos manualmente, entre outros.

Rhapsody

Rhapsody é uma ferramenta (*Integrated - Computer-Aided Software Engineering - Environment*) I-CASE-E, pertencente a IBM Rational. Para isso a IBM comprou a Telelogic em 2008, desenvolvedora original da ferramenta Rhapsody.

Esta ferramenta é compatível com as especificações do MDD, da UML 2.0 e da SysML. Ela pode gerar código fonte em *C*, *C++* e *Ada*, a partir de um modelo no padrão da UML. Pode ainda propiciar testes em nível de modelo, via mensagens animadas de diagramas de seqüência, máquinas de estado e diagramas de atividades.

Uma aplicação desenvolvida por meio da ferramenta Rhapsody pode ou não utilizar um Sistema Operacional - SO. Ao usar um SO, o esqueleto (*framework*) aberto do Rhapsody é utilizado para abstrair a camada do SO, permitindo que o mesmo modelo da UML possa ser portado para outros SOs. O código gerado utiliza também o *framework* para prover serviços críticos de tempo-real tais como *threadings*, passagens de mensagens, proteções de recursos e estouros de tempo.

Essa ferramenta baseia-se num conjunto de tecnologias chave para a correta aplicação do MDD. Utilizar efetivamente o MDD significa adotar uma metodologia onde defeitos são encontrados, logo cedo no processo de desenvolvimento, corrigidos e verificados, contra os requisitos, em todas as fases do ciclo de vida do processo de desenvolvimento de um sistema de software. Isto ocasiona a redução dos custos de desenvolvimento e o cumprimento de prazos com a entrega das funcionalidades prometidas.

A ferramenta Rhapsody possui, como principais características, a possibilidade de propiciar:

• **Suporte à Modelagem de Domínio Específico** (*Domain-Specific Modeling – DSM*) - Além de conformar com a UML, a Rhapsody também permite a utilização de uma variedade de linguagens de domínio específico como a linguagem do Esqueleto de Arquitetura do Departamento de Defesa norte-americano (*Department of Defense Architecture Framework - DoDAF*), a SysML e a *Functional C*. Essas linguagens, criadas a partir de perfis da UML 2.0, são totalmente compatíveis com outros sistemas que suportam o padrão XMI (*eXtensible Markup Language - **X**ML **M**etadata **I**nterchange*);

• **Rastreabilidade e Gerenciamento Integrados de Requisitos** (*Requirements Definition and Management*) - Esta ferramenta permite rastreabilidades visuais dos requisitos físicos do projeto, desconsiderando ferramentas como *Microsoft Word*, *Excel* ou até mesmo uma ferramenta especializada como IBM Rational *DOORS*, das quais os requisitos podem ser gerenciados. Atividades tradicionais de Engenharia de Sistemas elaboradas em documentos são facilmente ligadas ao projeto, para mostrar como o modelo satisfaz os requisitos estabelecidos. Isso assegura rastreabilidade de requisitos em todos os caminhos até o código e pode propiciar o gerenciamento de impactos de mudanças, de maneira visual e integrada;

• **Geração Completa de Software de Aplicação** - A partir da Rhapsody, pode-se preparar aplicativos independentes de plataformas para Sistemas Operacionais de Tempo-Real. As sínteses desses aplicativos não se restringem somente a geração de código, mas também incluem a construção de todos os artefatos necessários, como por exemplo, a inclusão de bibliotecas de terceiros (*third-party libraries*) ou quaisquer outros aplicativos externos, por intermédio de engenharia reversa completa do código, ou construindo-se um processo diretamente gerenciável pela Rhapsody;

• **Projeto para Testabilidade** (*Design for Testability – DFT*) - Esta característica da ferramenta permite o teste do sistema durante o tempo de construção, reduzindo o número de defeitos, logo cedo, sempre e em paralelo com seu desenvolvimento, provendo uma infra-estrutura de testes automatizados para permitir testes de regressão; e

• **Fluxo de Trabalho Centralizado** - A habilidade de propiciar o trabalho em nível de código ou de modelo, com o mínimo de esforço, é crítica para o desenvolvimento de sistemas embarcados de tempo-real. Esta característica da ferramenta Rhapsody permite um nível, sem paralelização, de depuração de aplicativos para o sistema alvo real, identificando e corrigindo defeitos e propiciando mudanças no código, refletidas, automaticamente, na ferramenta de modelagem.

Apesar de ser uma ferramenta paga, existe uma versão *demo/trial* para se fazer *download* gratuito e avaliar todas as suas funcionalidades durante 30 dias. O *download* pode ser feito através do site da IBM.

MDD na Prática

Quando o termo "MDD na Prática" é apresentado neste artigo, ele quer dizer que o sistema deve ser desenvolvido dirigido a modelos ou modelagem. Isto é, todas as especificações do sistema devem se encontrar inseridas no modelo do sistema.

Este modelo pode ser desenvolvido preferencialmente utilizando uma ferramenta CASE, pois através desta pode-se aproveitar algumas facilidades como geração de código-fonte, relatórios e documentações, como ilustrado na **Figura 3**.

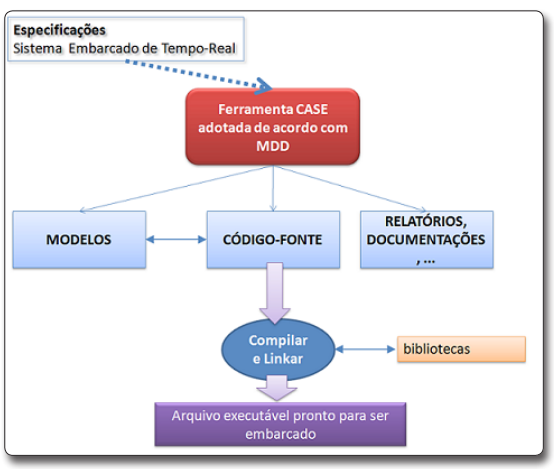


Figura 3. Processo para aplicar MDD na prática

Uma das funcionalidades mais interessantes que uma ferramenta CASE pode propiciar é a associação bi-direcional entre modelo e código-fonte. Se o modelo é alterado, isto é refletido automaticamente no código-fonte e vice-versa. Felizmente, a ferramenta Rhapsody possui tal característica.

Outras características como de gerenciamento e rastreabilidade de requisitos também são desejáveis. Isso facilita identificar no desenvolvimento do sistema se tal requisito está sendo contemplado e por quais casos de uso, blocos do sistema e funções ele está sendo contemplado. Ter isso através de uma única ferramenta e poder visualizar estas informações de maneira gráfica e rápida agiliza muito o processo de desenvolvimento.

Outro fato comum é que os engenheiros de desenvolvimento não se identificam muito com a atividade de elaboração de relatórios de documentação dos sistemas que eles desenvolvem. De fato, essa não é uma tarefa muito estimulante. Entretanto, documentar é essencial. Produtos que não possuem documentação estão fadados a fracassar. Escolhendo uma ferramenta CASE adequada, os relatórios de documentação do sistema podem ser gerados automaticamente, bastando apenas uma configuração inicial e preparação do formato desejado. Baseado nos modelos introduzidos na ferramenta ela é capaz de gerar um grande número de tipos de relatórios de maneira muito eficaz.

Em suma, muitas são as vantagens de se utilizar uma ferramenta CASE o mais completa possível para se desenvolver sistemas. Alguns pontos que podem ser encarados como uma "desvantagem" são talvez o custo dessas ferramentas e o custo de treinamento e capacitação inicial para se desenvolver segundo os novos paradigmas existentes. Além disso, tudo o que é "novo" implica numa curva de aprendizado.

O processo ilustrado na **Figura 3** será exemplificado na prática a seguir.

Desenvolvendo um mini projeto de sistema embarcado de tempo-real

Visando mostrar na prática a utilização de MDD, ilustrando um processo de desenvolvimento apresenta-se um mini estudo de caso.

Este mini estudo de caso foi desenvolvido com o auxílio da ferramenta Rhapsody, que se encontra de acordo com os padrões introduzidos pelo MDD, pela UML e pela SysML.

Este estudo de caso compreende o desenvolvimento de um sistema embarcado de tempo-real, que por questões de simplicidade está utilizando como sistema operacional o Linux. O tempo-real considerado para este sistema é da ordem de segundos, sendo portanto considerado *soft* tempo-real.

Iniciou-se o desenvolvimento desse sistema introduzindo e modelando:

1. os requisitos do sistema embarcado e de tempo-real;
2. os diagramas de casos de uso;
3. os diagramas de seqüência;
4. os diagrama de blocos do sistema; e finalmente
5. os diagrama de estados.

Após esses passos, algumas configurações foram realizadas na ferramenta Rhapsody para se escolher o sistema operacional e o compilador utilizado. Relatórios de exemplo também foram gerados.

Este desenvolvimento considerou como linguagem de programação a linguagem C, por ser a mais indicada quando se trata do desenvolvimento de sistemas embarcados de tempo-real.

Somente por questões de esclarecimentos, o sistema apresentado não possui completeza de funções, uma vez que este não constitui o objetivo principal deste artigo que é mostrar na prática a aplicação de MDD para sistemas embarcados de tempo-real.

Descrição do sistema embarcado de tempo-real

O sistema embarcado de tempo-real apresentado aqui se trata de um protótipo didático para determinação e controle de atitude (*attitude and orbit control system – AOCS*) de satélites artificiais, sem completeza. Isto é, ele não contempla todas as funções necessárias para desempenhar tal papel. Note (**Figura 4**), que os sensores e atuadores utilizados nesse tipo de sistema estão sendo simulados pelos atores: *Sensor* e *Atuador*.

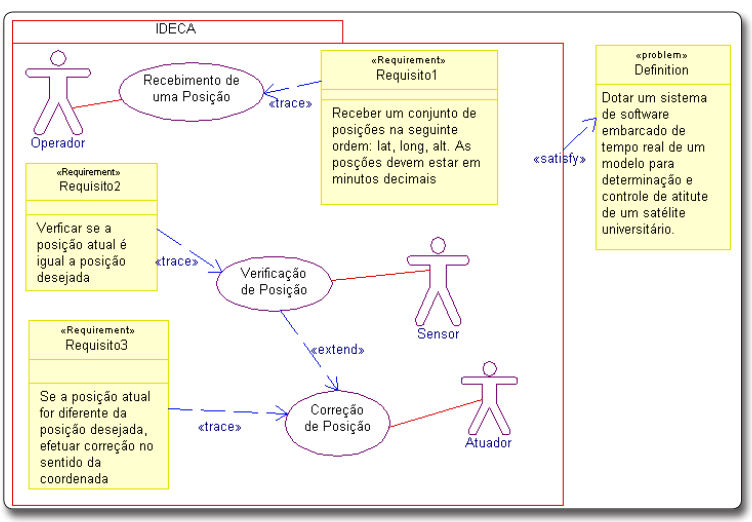


Figura 4. Diagrama de Requisitos e Casos de Uso integrados

Um sistema como esse tem o papel de orientar a atitude de um satélite, ou seja, guiá-lo de maneira apropriada no espaço. Para fins ilustrativos apenas três requisitos, bem simples, foram considerados, conforme ilustrado na **Figura 4**.

No requisito 1 desta figura, as variáveis *lat*, *long* e *alt* significam latitude, longitude e altitude.

Basicamente, a "grosso modo", o que um satélite artificial faz para se guiar é consultar seus sensores de posição espacial (atitude), confrontar com a posição alvo e finalmente efetuar as correções de posição necessárias através dos atuadores de correção de atitude.

Modelando os Requisitos e os Casos de Uso

A **Figura 4** ilustra tanto a modelagem dos casos de uso do sistema como também os requisitos associados. Esse relacionamento entre requisitos e casos de uso é possível através do emprego da SysML na construção do modelo.

Atente para os estereótipos do tipo <<*trace*>> no diagrama. Eles servem para rastrear quem está atendendo a um requisito. Desta maneira a ferramenta consegue gerar relatórios de rastreabilidade de requisitos.

Modelando o Diagrama de Sequência

Uma vez que já se tem os diagramas de requisitos e casos de uso modelados, pode-se passar para a modelagem do diagrama de seqüência. Este diagrama serve para dizer qual a seqüência de mensagens será trocada entre os elementos do sistema, considerando sua ordem.

Através deste diagrama (**Figura 5**) é possível ter uma visão geral do sistema, sabendo qual elemento vai se comunicar com qual, através de que mensagem e a ordem das mensagens trocadas. Tal diagrama pode ser usado para auxiliar no processo de testes e verificação do modelo.

Modelando o Diagrama de Blocos do Sistema

Neste passo, quando já se tem a ideia de como o sistema se comporta no tempo e todos os seus elementos já foram identificados, torna-se possível modelar o diagrama de blocos do sistema (**Figura 6**).

Este diagrama mostra os relacionamentos entre os elementos, seus parâmetros e funções. Assim como o diagrama de seqüência (**Figura 5**), o diagrama de blocos também fornece uma visão geral do sistema, entretanto sob um aspecto ou ponto de vista diferente. Desta vez mostrando os relacionamentos entre os elementos sem a informação de mensagens trocadas no tempo.

Modelando os Diagramas de Estado

Este passo concentra-se em modelar o diagrama de estados de cada elemento identificado no sistema. Uma máquina de estados modelada pode ser vista na **Figura 7**.

O diagrama de estados já é bem conhecido dos desenvolvedores de sistemas embarcados de tempo-real.

Uma facilidade que a ferramenta de modelagem provê é que o desenvolvedor precisa escrever o código-fonte somente para

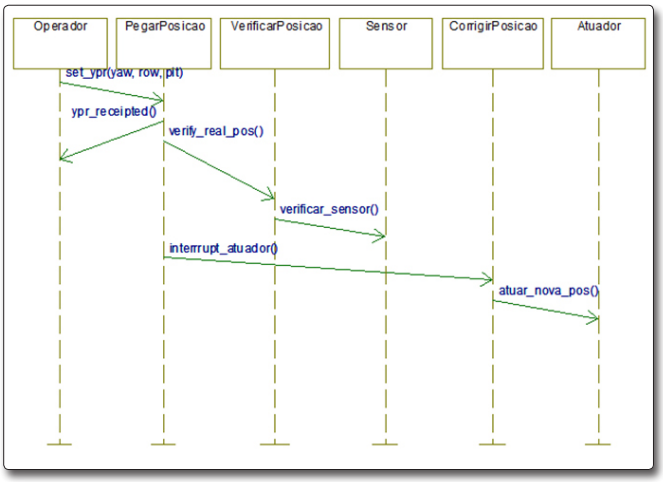


Figura 5. Diagrama de Sequência de execução do sistema

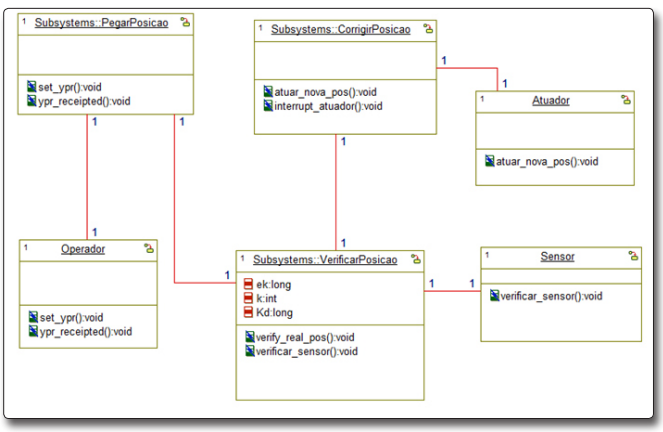


Figura 6. Diagrama de Blocos do Sistema

as trocas de estados, condições de entradas ou saídas. Quem gera o código "braçal" do mecanismo de troca de estados é a ferramenta CASE.

Tal característica poupa o tempo do desenvolvedor também o tornando mais produtivo, uma vez que este estará focado especificamente na codificação das regras de negócio do sistema.

Neste caso, é uma opção da ferramenta visualizar ou não no diagrama os códigos entrados nas transições da máquina de estados. A **Figura 7** ilustra um diagrama com os códigos associados às transições.

Gerando os Códigos-Fonte Finais

Finalmente, após o desenvolvedor colocar nos modelos todos os códigos-fontes (**Figura 8**) da regra de negócio do sistema, este está a poucos passos de seu final. A maioria do código-fonte, cerca de 80%, corresponde ao que a ferramenta gera automaticamente. Este código é passível de modificações, testes e validação.

Um ponto interessante a comentar é que a ferramenta possui associação bi-direcional entre modelos e códigos-fonte. As alterações realizadas no código-fonte refletem no modelo e vice-versa. Isso evita modelos desatualizadas ou que não correspondem fielmente aos códigos-fontes. Ou seja, a documentação do sistema estará sempre atualizada.

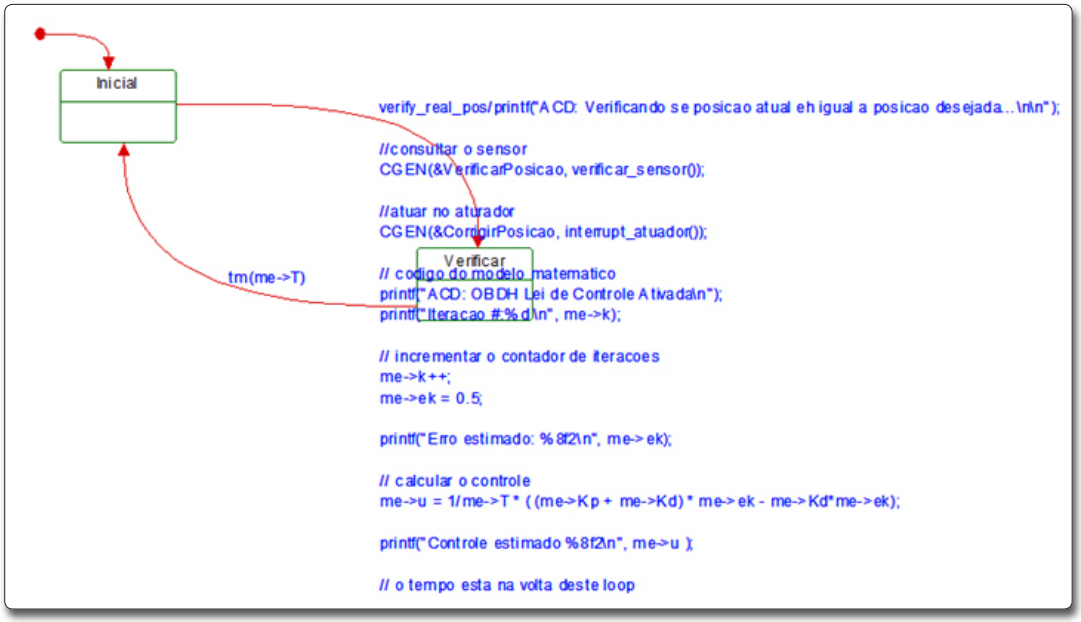


Figura 7. Diagramas de Estado do Sistema

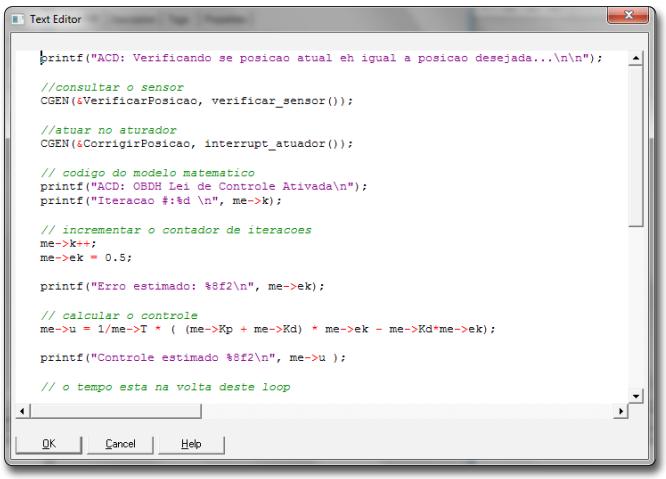


Figura 8. Porção de código-fonte introduzido pelo desenvolvedor

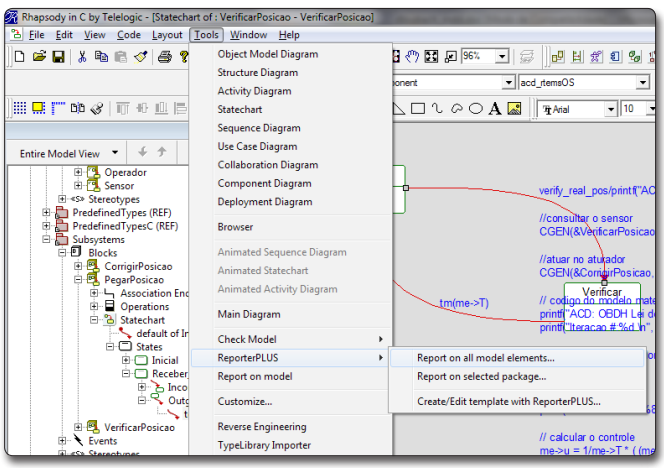


Figura 9. Menu para geração de relatório de documentação do sistema

Gerando Relatórios do Sistema

Outra característica mencionada sobre a ferramenta Rhapsody é sua facilidade para gerar relatórios de desenvolvimento. A **Figura 9** mostra o menu para geração dos relatórios desejados. Note que, obviamente, a ferramenta só vai conseguir gerar relatórios baseados nas informações fornecidas para ela. Assim recomenda-se que os modelos sejam desenvolvidos agregando-se o máximo de descrições (comentários) possíveis. Isso viabilizará um relatório mais completo e eficaz.

Esta ferramenta fornece uma série de padrões de relatório como relatório de classes, diagramas, relatório baseado na SysML, entre outros. Além disso, se o desenvolvedor precisar criar seu próprio meta-modelo de relatório ele poderá fazer isso através dos recursos que a Rhapsody possui.

Compilando e Linkando o Sistema

Neste passo deve-se fazer a configuração final do sistema para gerar o arquivo executável, capaz de ser testado e embarcado no hardware. A **Figura 10** ilustra algumas configurações que a ferramenta requer, entre elas o sistema operacional alvo. Neste caso utilizou-se o Linux.

Parâmetros do compilador e do *linker* também são fornecidos para a ferramenta neste ponto (**Figura 10**).

Realizada esta configuração, deve-se gerar o arquivo executável final, conforme mostrado na **Figura 11**. A ferramenta se encarrega de realizar as chamadas do compilador e do *linker*. Relembrando, os parâmetros de compilação e linkagem devem ser fornecidos para a ferramenta ter condições de se encarregar de realizar as chamadas.

Resultados Finais

Neste último passo o sistema já se encontra em execução (**Figura 12**). Testes de modelo são possíveis neste ponto. Todo o trabalho do lado do software já foi feito. Agora basta embarcar o arquivo executável gerado no seu hardware alvo, rodando Linux.

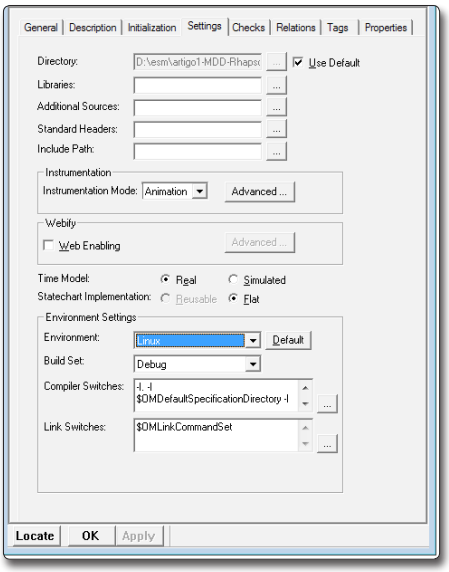


Figura 10 - Ilustração da janela de configurações do sistema

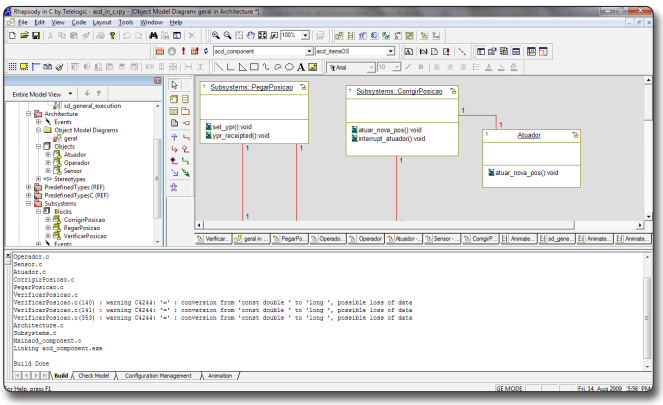


Figura 11. Janela mostrando a geração dos códigos-fontes finais, compilação e linkagem

```
**************************************************
OPERADOR INICIALIZADO
**************************************************

**************************************************
ACD INICIALIZADO
**************************************************

--------- NOVO CICLO INICIALIZADO ------------

OPERADOR: Nova posicao enviada ...

ACD: Nova posicao adquirida (y:-21 p:500 r:-47) !

ACD: Verificando se posicao atual eh igual a posicao desejada...

ACD: OBDH Lei de Controle Ativada
Iteracao #:0
Erro estimado: 0.0000002
Controle estimado 0.0000002
OPERADOR: ACD confirmou recebimento

ACD: Correcao de posicao solicitada

ATUADOR: atuando na roda de reacao

--------- NOVO CICLO INICIALIZADO ------------
```

Figura 12. Janela de terminal ilustrando as mensagens de acompanhamento da execução do programa

No começo do artigo, foram apresentadas as definições de Modelo de Computação Independente, Modelo de Plataforma Independente, Modelo de Plataforma Específica, Modelo de Plataforma e Modelo de Transformação (**Figura 1**). Com a utilização da ferramenta Rhapsody, o usuário não precisa se preocupar com a separação e transformação desses modelos. A ferramenta se encarrega de realizar todas as transformações necessárias através de seu *framework* aberto (que pode ser adaptado) mantendo compatibilidade entre eles.

Principais Vantagens e Desvantagens

Fazendo uma análise geral da aplicação de MDD pode-se dizer que as vantagens são muitas. Entre elas é possível verificar a agilidade no desenvolvimento, a melhor compreensão do problema e da solução através de modelos, o ganho de qualidade, menor retrabalho, entre outros.

Infelizmente, como não existem somente vantagens, alguns pontos contribuem negativamente. Podemos destacar que o MDD ainda não é aceito por todas as comunidades de desenvolvedores em função de parecer "burocratizar" o processo ou parecer "documentar demais". Além disso, quase sempre se torna necessário adotar uma ferramenta CASE. O processo de escolha da ferramenta e depois a sua curva de aprendizado podem soar negativamente.

Entretanto, vencidas essas dificuldades iniciais, os ganhos podem recompensar em médio prazo.

Conclusão

Este artigo teve por objetivo apresentar uma introdução rápida quanto à aplicação de Desenvolvimento Dirigido a Modelos (*Model-Driven Development - MDD*). Entretanto, buscou-se fornecer uma visão bem prática voltada para a realidade diária dos profissionais da área.

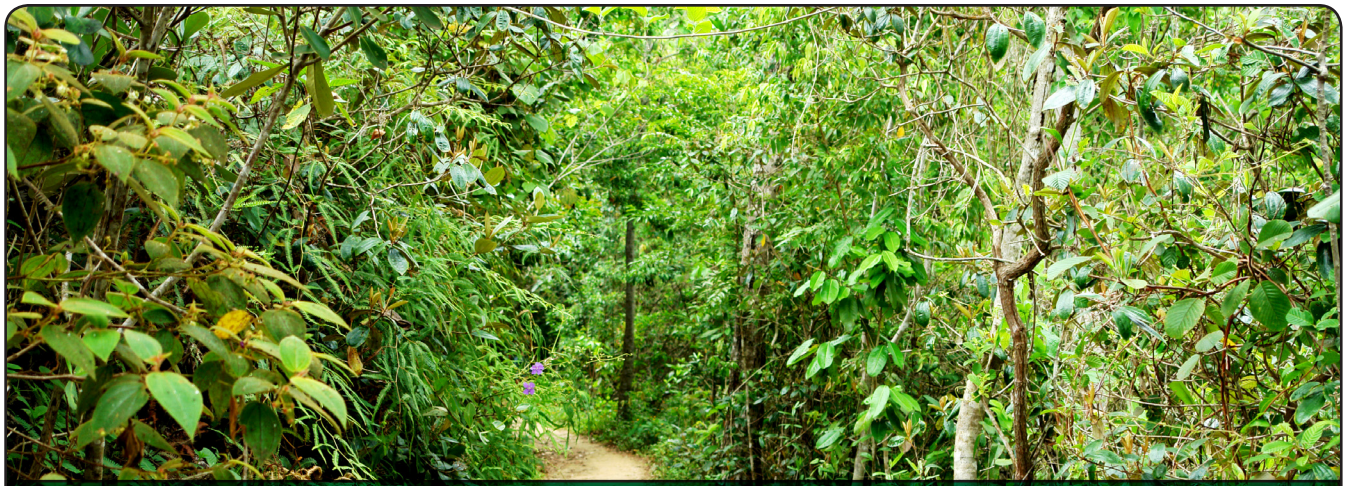
O artigo também abordou uma síntese sobre desenvolvimento de sistemas de maneira tradicional, conceitos e teoria sobre MDD e uma possível ferramenta para se aplicar MDD na prática.

Mostrou-se que é possível utilizar MDD com uma ferramenta de geração automática de código-fonte para desenvolver um sistema embarcado de tempo-real para determinação e controle de atitude de um satélite artificial. Entretanto, sem completeza. Em seguida, algumas vantagens e também desvantagens foram apresentadas após uma análise da aplicabilidade de MDD.

Acredita-se que, aos poucos, esta prática ainda considerada um pouco tímida, evolua e se aprimore viabilizando sua aceitação por uma comunidade mais ampla de desenvolvedores e engenheiros de sistemas. Afinal, "mudar (evoluir) é preciso".

Agradecimentos

Ao Instituto Tecnológico de Aeronáutica pelos seus laboratórios e ao Prof. Dr. Adilson Marques da Cunha pelas motivações realizadas. ●



Prototipação no Desenvolvimento de Software

Aprenda a criar protótipos de interfaces com a ferramenta Axure

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

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

Marco Antônio Pereira Araújo
maraujo@granbery.edu.br
Doutor e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Matemática com Habilitação em Informática pela UFJF, Professor do Curso de Bacharelado em Sistemas de Informação da Faculdade Metodista Granbery, Analista de Sistemas da Prefeitura de Juiz de Fora, Editor da Engenharia de Software Magazine.

De que se trata o artigo?

Esse artigo apresenta algumas definições sobre a prototipação no desenvolvimento de sistemas de informação, seus diferentes tipos como prototipação de baixa e alta fidelidade, prototipação throw-away e prototipação evolutiva, wireframes, suas aplicações, vantagens e desvantagens. Além disso, mostra a utilização de uma ferramenta capaz de gerar protótipos de vários tipos dependendo de qual linha o desenvolvedor deseja seguir.

Para que serve?

No passado, protótipo tinha como principal finalidade avaliar os requisitos de um sistema através do desenvolvimento tradicional. Atualmente, os limites entre a prototipação e o desenvolvimento do sistema se confundem, e muitas vezes estes sistemas são construídos numa abordagem evolucionária. Serão esclarecidos alguns conceitos que permitirão ao desenvolvedor avaliar qual é o melhor caminho a seguir e utilizar uma ferramenta para tal, o Axure RP Pro.

Em que situação o tema é útil?

De maneira geral a prototipação é positiva, trazendo melhoria na facilidade de uso do sistema, maior aproximação do sistema com as necessidades dos usuários, melhoria da qualidade do projeto, facilidade de manutenção e redução no esforço de desenvolvimento. Entretanto, não pode ser um processo que tome muito tempo nem esforço dos programadores. Para isso, será demonstrado uma forma rápida de se construir protótipos inteligentes e wireframes através do Axure RP Pro.

De acordo com o dicionário, protótipo significa: primeiro tipo ou exemplar; original, modelo. Versão parcial e preliminar de um novo sistema de computador ou de um novo programa, destinada a teste e aperfeiçoamento. Essa definição nos desperta para a sua funcionalidade, que nos faz pensar e decidir qual é a melhor maneira de utilizá-lo.

O protótipo pode ser interpretado como uma maquete, que contém os conceitos do que se pretende obter do produto final. Pode ser considerado também como a primeira versão de um produto que, mesmo com as funcionalidades incompletas, permite ser criticado e aperfeiçoado a fim de se obter qualidade no que está sendo produzido.

O protótipo no Desenvolvimento de Sistemas

A prototipagem de sistemas tem se tornado uma prática cada vez mais comum entre os desenvolvedores, e tem se mostrado uma alternativa interessante para a solução de vários problemas. Tem como objetivo principal validar os requisitos, abordar questões de interface, e avaliar tanto a viabilidade quanto a complexidade do sistema.

Durante a criação do protótipo, usuário e desenvolvedores ficam em constante interação, facilitando assim, alterações nas funcionalidades do sistema, bem como o levantamento e modificação de requisitos. Neste período, o esforço para qualquer intervenção, tanto na documentação quanto na implementação da aplicação, é muito menor, reduzindo mão-de-obra e, consequentemente, custos de produção. Esta atividade permite que, desde muito cedo, se obtenha uma melhor percepção dos requisitos do sistema e também simplificando o seu esboço.

Até agora foram discutidos alguns benefícios da prototipação, porém existem tipos diferentes e cabe ao desenvolvedor escolher qual é a melhor utilização para o seu caso.

Serão vistos a partir daqui três vertentes importantes de prototipação: a Prototipação de baixa fidelidade, média fidelidade e alta fidelidade. Na prototipação de alta fidelidade existem basicamente dois tipos que serão vistos com detalhe, a prototipação Throw-Away, ou Descartável, e também a Prototipação Evolutiva.

Protótipos de Baixa Fidelidade

Este tipo de prototipagem é aquela onde o protótipo não se assemelha com o produto final, serve somente de base para críticas. Pode ser realizado de várias formas, com ferramentas próprias ou até um esboço feito com papel e caneta. Esse tipo se encaixa mais facilmente na fase inicial de desenvolvimento, onde ainda cabe a compreensão dos requisitos. São fáceis de construir, de custo muito reduzido, de produção extremamente rápida, antecipando o momento de validação do usuário.

Porém, esse tipo de prototipação possui alguns aspectos que podem inviabilizar sua escolha pelo desenvolvedor. Geralmente os protótipos gerados dessa maneira não são utilizados após a fase de documentação, são descartados, pois não podem ser aprimorados devido a sua simplicidade. Sua funcionalidade pára no processo de documentação, não podendo servir de base para codificação, testes de navegabilidade e usabilidade.

Protótipos de Média Fidelidade

Os protótipos de média fidelidade possuem um nível de detalhamento maior do que os de baixa fidelidade, porém já possuem algumas definições importantes tanto para o usuário quanto para o desenvolvedor. São feitos por ferramentas mais específicas, pois não são somente esboços e, por isso, devem seguir alguns critérios.

Esse tipo de protótipo é bastante positivo, pois o usuário já consegue visualizar um futuro para o seu produto e o desenvolvedor já tem no que se apoiar para a construção do sistema final, além de ser geralmente de fácil e rápida construção.

Deve-se considerar que, dependendo da situação, pode não ser a melhor escolha, pois se pode perder tempo com este processo sem necessidade, ou ser preciso um detalhamento maior do que esta técnica oferece.

Protótipos de Alta Fidelidade

Protótipos de alta fidelidade são aqueles que se assemelham bastante ao produto final. Muitas das vezes utiliza a mesma tecnologia que o sistema final e servem especialmente para o teste de componentes e a solução de problemas técnicos, onde questões de compreensão dos requisitos já ficaram para trás. Um protótipo desse tipo permite pensar nos detalhes do produto bem mais a fundo do que com especificações no papel, e reduz significativamente o tempo de desenvolvimento do produto final, pois ele já está bem mais definido, e vários problemas já foram sanados antecipadamente na construção do protótipo.

Além disso, esse protótipo permite mensurar com mais precisão a complexidade e os custos do processo de desenvolvimento, resolvendo os problemas mais significativos, economizando mão-de-obra, tempo e dinheiro.

No entanto, esse método não é completo, pois possui alguns aspectos que devem ser levados em consideração. Mesmo reduzindo o tempo de produção da aplicação definitiva, estes protótipos de alta fidelidade demandam um custo elevado e um tempo grande para o desenvolvimento do mesmo.

A prototipação de alta fidelidade poderá ser implementada seguindo basicamente dois métodos: prototipação Throw-Away, na qual seu objetivo é identificar e validar requisitos, e prototipação Evolutiva, que tem o objetivo de minimizar o tempo de desenvolvimento do sistema.

Prototipação Throw-Away (Descartável)

Este tipo de prototipação é utilizado especialmente durante a fase de levantamento de requisitos. O desenvolvedor gera uma documentação provisória, e cria um protótipo baseado nesta documentação para obter novos requisitos, até então não visualizados, e validar os já implementados. Este protótipo não segue necessariamente a mesma tecnologia de desenvolvimento do software final, normalmente busca-se métodos e linguagens mais ágeis, já que ele será descartado depois de cumprida a sua finalidade.

O processo de utilização deste método tem como intuito refinar a documentação para gerar qualidade para o produto que se deseja criar. Após esse processo, o protótipo criado é abandonado e então parte-se para o desenvolvimento da aplicação definitiva.

Prototipação Evolutiva

A prototipação evolutiva, ao contrário da descartável, é utilizada em protótipos que, através de incrementos, evoluirão até o sistema final. São utilizados requisitos já compreendidos, e não há uma documentação definida, visto que a prototipação evolutiva é um método que visa à redução de custo e de tempo. A interação com o usuário é constante e o objetivo é fornecer

um sistema, ou módulos de um sistema, funcionando. A seguir estão os passos no processo de construção de um software adotando o processo de prototipação evolutiva (ver **Figura 1**).

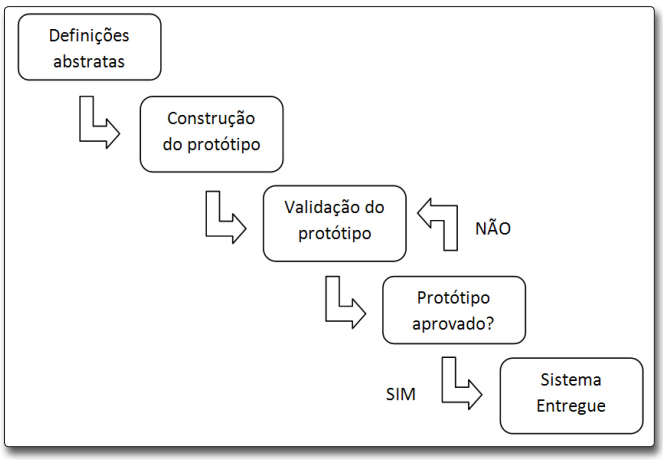


Figura 1. Fluxo dos processos de criação do protótipo.

Esse método propicia uma entrega rápida do produto, reduz custos e proporciona ao usuário uma visualização precoce do sistema. Além disso, com esse método é possível realizar vários tipos de testes nos casos de uso já implementados, deixando o sistema mais confiável.

No entanto, este método carrega algumas características que podem trazer problemas futuros. Geralmente os sistemas gerados dessa maneira não possuem uma especificação devidamente documentada, causando uma dificuldade para mensurar custos, elaborar contratos e estimar tempo de entrega. Se o processo de desenvolvimento não for muito bem planejado, os sistemas se tornam muito difíceis de manter, especialmente se for feita por outros senão aqueles que desenvolveram o produto. Com isso, a vida útil do sistema diminui e, o que era para ser barato, sai muito mais caro no longo prazo.

Wireframes

Wireframes são documentos elaborados durante a fase de especificação de um projeto Web com o objetivo de registrar e esclarecer questões de interface, navegabilidade e usabilidade. O projeto de um Wireframe pode ser considerado como um protótipo e, assim como o protótipo, pode ser desenvolvido de acordo com as necessidades do desenvolvedor e do cliente, bastando escolher a priori o nível de fidelidade com que se deseja construí-lo.

Como já foi explicado anteriormente, o Wireframe pode ser de baixa, média e alta fidelidade, podendo ser de um simples esboço até um protótipo contendo localização e tamanho de janelas e tipos de fontes.

A escolha destes atributos deve ser cuidadosa pois, dependendo dela, o projeto pode ficar muito "pobre" ou com detalhes demais, deixando o designer limitado, não permitindo o uso de sua criatividade. Além disso, na apresentação de um Wireframe para o usuário, deve-se esclarecer que é somente um "esqueleto" do que será desenvolvido, não gerando expectativas ou alguma má impressão.

O Wireframe, por se tratar de um projeto voltado para Web, pode ser desenvolvido de várias formas, podendo ser criados com programas específicos ou com qualquer ferramenta que gere arquivos HTML.

A seguir está um exemplo de um protótipo de média fidelidade. Não possui definição de cores nem de tamanhos das janelas, mas já é possível ter uma idéia dos elementos que estarão presentes (ver **Figura 2**).

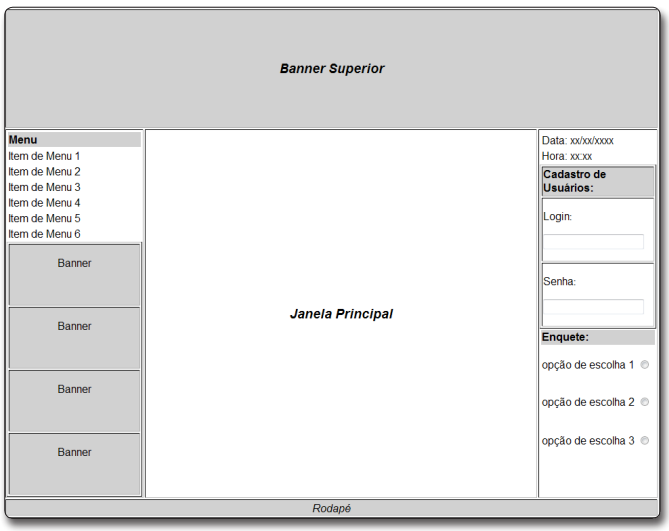


Figura 2. Protótipo de média fidelidade

Ferramentas de Prototipação

Existem softwares específicos para o desenvolvimento de protótipos e wireframes, e também existem aqueles que podem ser utilizados para tal dependendo do grau de detalhamento que se deseja obter. Ferramentas de desenho e até de modelagem 3D podem ser aproveitadas para gerar esboços do layout dos sistemas que serão desenvolvidos. A própria IDE em que será desenvolvido o sistema pode ser utilizada para se criar um protótipo, especialmente em protótipos de alta fidelidade.

Alguns exemplos de ferramentas são o Microsoft Visio, Adobe InDesign, Pencil (como extensão do Firefox) e o Axure RP pro, aplicativo que será utilizado no estudo de caso deste artigo.

A Ferramenta de Prototipação Axure

Neste artigo faremos um estudo de caso a fim de demonstrar o uso de protótipos no desenvolvimento de sistemas. Usaremos a ferramenta Axure RP Pro para a criação dos protótipos que, durante a elaboração deste artigo, estava na versão 5.5. O download pode ser feito no site http://www.axure.com/downloads.aspx e vale lembrar que o Axure não é uma ferramenta gratuita e será utilizada a versão Trial do aplicativo.

O Axure permite que se crie um diagrama de fluxo dos seus protótipos, protótipos do tipo wireframe e um documento de especificação contendo os detalhes dos campos existentes nas

páginas dos protótipos, além de projetos que podem ser compartilhados. Os protótipos gerados pelo Axure são páginas do tipo HTML, que podem, dependendo do projeto, ser usadas como base para a criação das páginas reais do sistema.

Compartilhando o protótipo

O Axure permite criar protótipos que podem ser acessados e modificados por outras pessoas que estão desenvolvendo os protótipos. Permite criar um repositório com o uso da ferramenta de controle de versão Subversion, que já vem integrado ao Axure. Neste artigo não será feito o uso de projetos compartilhados, mas eles podem ser facilmente criados e gerenciados. Primeiramente, cria-se o repositório selecionando a opção em "New Shared Project". A primeira janela serve para informar o nome do projeto. Na tela seguinte, informa-se a localização do repositório que conterá o projeto a ser compartilhado e, em seguida, na última janela informa-se a pasta em será construído o protótipo.

Se outras pessoas tiverem que usar o mesmo projeto, basta acessar o repositório e, em seguida, indicar a pasta que contém o protótipo. Qualquer alteração no protótipo pode ser enviada para o repositório e, com isso, outros que utilizarem o protótipo podem receber as alterações feitas.

Conhecendo a Ferramenta

Após aberta a ferramenta, o Axure já inicia com quatro páginas criadas e abas que serão usadas para elaboração dos protótipos (ver **Figura 3**):

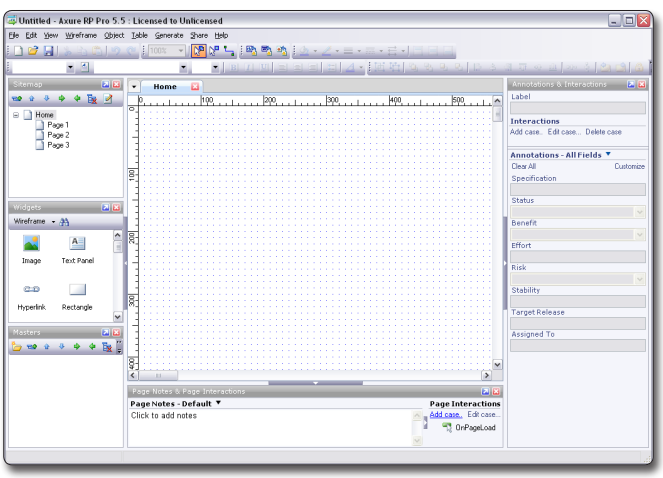


Figura 3. Visualização da tela inicial da ferramenta.

• A aba "Sitemap" é responsável por conter todas as páginas e diagramas de fluxo referentes ao protótipo.
• A "Widgets" contém as bibliotecas de componentes que são usados para criar os protótipos e diagramas de fluxos de páginas. Como padrão, possui as bibliotecas de Wireframe (usadas na elaboração das páginas) e de Flow (usadas na elaboração de diagramas de fluxo), com a possibilidade de adicionar bibliotecas extras.
• A aba "Masters" é responsável por conter fragmentos de páginas que podem ser adicionadas nas telas dos protótipos inúmeras vezes, como o caso de menus e cabeçalhos.
• Na aba "Page Notes & Page Interactions" é possível inserir anotações sobre as páginas ou diagramas de fluxo e habilitar interações para quando a página for acionada.
• A aba "Annotations & Interactions" é onde se fazem as principais configurações. Pode-se adicionar anotações gerais ou detalhadas dos componentes, como especificar se o uso desses está aprovado ou não. É ainda possível habilitar interações para esses componentes interagirem, como realizar navegações entre as páginas, abrir ou fechar janelas dos navegadores, ou ativar e desativar componentes específicos.

Criando os Protótipos

Para exemplificar a utilização da ferramenta, será criado um protótipo que simula um sistema que gerencia as notas de alunos cadastrados. O protótipo terá uma página principal que apresenta uma lista com todos os alunos cadastrados no sistema, a opção de Inserir as notas do aluno desejado por uma janela pop-up, selecionando a opção na tabela, e a opção de incluir um novo aluno.

Iremos criar as páginas para depois fazermos as interações. Primeiramente, foram removidas as páginas já criadas pelo Axure e criadas três novas páginas para melhor exemplificarem o caso de uso na aba "Sitemap" (ver **Figura 4**).

Figura 4. Páginas criadas na aba "Sitemap"

Em "listarAlunos", adicionamos em "Page Notes" uma anotação informando o propósito desta página. Usamos o componente tabela, arrastando-o para dentro da página com 4 linhas e 4 colunas. A primeira linha corresponderá ao cabeçalho com as colunas código, nome, status e inserir notas. As demais linhas correspondem aos alunos cadastrados. A formatação da fonte e das tabelas, como em qualquer outro componente, podem ser alterados pelo menu, que é semelhante aos existentes em ferramentas de edição de texto.

Na tabela criada foram adicionados 3 alunos com seus respectivos dados. Informamos, também, quais são as especificações das colunas "Status" e "Inserir Notas" pois, dependendo das ações realizadas, seus comportamentos deveram ser alterados. Na coluna "Status" foi informado que, dependendo das notas inseridas, informará se o aluno foi Aprovado ou Reprovado e, no caso da coluna "Inserir Notas", se as notas já tiverem

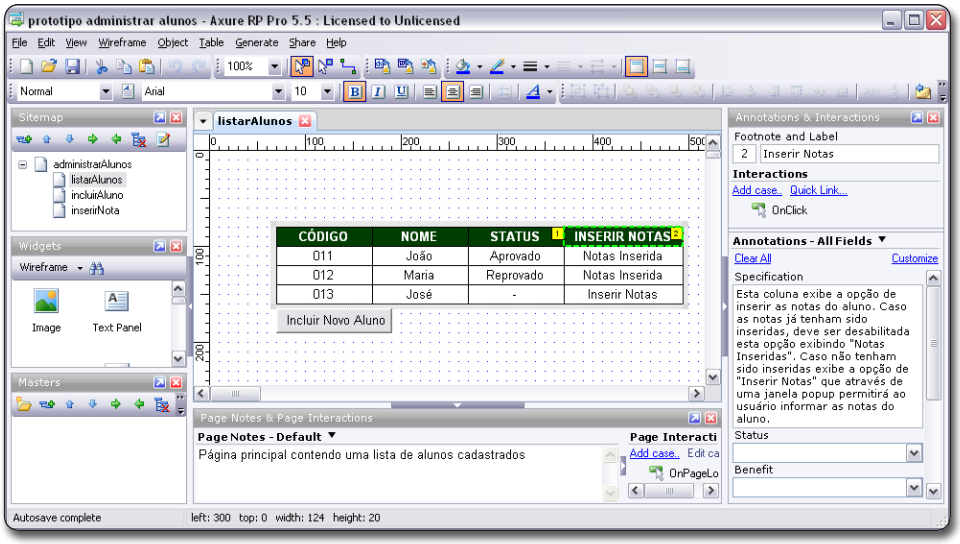


Figura 5. Tabela criada na página "listarAlunos" e configurações das colunas

sido informadas deve ser exibido o texto"Notas Inseridas", desabilitando a opção de inserir notas. Caso não tenham sido inseridas, deve permitir esta opção exibindo o texto "Inserir Notas". Note que o Axure vai adicionando uma marcação numerada à medida que são informadas essas especificações, ou qualquer evento inserido, sendo que a numeração pode ser alterada (ver **Figura 5**).

Para permitir a navegação entre as páginas e permitir a exibição da janela pop-up, devemos adicionar as interações dos componentes para realizarem essas ações. Para habilitar a criação de novos alunos, selecionamos o botão "Incluir Novo Aluno" e clicamos em "Quick Link..." em "Interactions" que é a configuração mais simples, pois queremos apenas ir para outra página que, no nosso exemplo, é "incluirAluno". Caso queria detalhar mais, pode-se informar a descrição dessa interação. Quando o protótipo for criado, ao clicar no botão, o usuário será direcionado para a página de inclusão de aluno.

Para habilitar a janela pop-up, o processo é semelhante, porém devemos informar como a página deve ser aberta. Para isso selecionamos o campo da tabela "Inserir Nota" e clicamos em "Add case..." e informamos a descrição para essa interação. Selecionamos "Open Link in Popup Window" e, em seguida, selecionamos Link em "Step 3" e, novamente, selecionamos a página desejada, que é "inserirNota". Depois selecionamos, também em "Step 3", a opção Popup que permite configurar as propriedades da janela pop-up. Pode-se definir o tamanho da janela e desmarcar algumas opções, como para que não apareça o menu e scrollbars na nova página.

Note que, após essas configurações, o Axure adiciona novamente as marcações numeradas. Essas marcações estarão tanto no documento de especificações quanto no protótipo, possibilitando ao desenvolvedor uma melhor compreensão de qual ação o componente deve fazer. Todas essas configurações podem ser personalizadas, permitindo a quem está desenvolvendo o protótipo adicionar as anotações desejadas. Pode-se, ainda nas navegações, criar condições que através de variáveis previamente criadas permitem criar protótipos ainda mais reais em relação às regras de negócio do sistema a ser projetado. Pode-se, também com as interações, permitir que um componente fique habilitado ou desabilitado, em relação a uma ação realizada pelo usuário. Não demonstraremos nesse artigo essas ações, que podem ser facilmente configuradas.

Iremos para a criação da página de inclusão. Com a página previamente criada, inserimos os campos código, nome e curso, que é um campo do tipo dropdown. Para esse campo, selecionamos a opção de edit list com o botão direito no componente para adicionar uma lista de dados desejados que devam aparecer no campo. Para os campos, com seus respectivos nomes, utilizamos o componente textpanel. Inserimos anotações tanto na página quanto nos componentes e adicionamos os botões de Salvar e Voltar com as anotações de navegações (ver **Figura 6**).

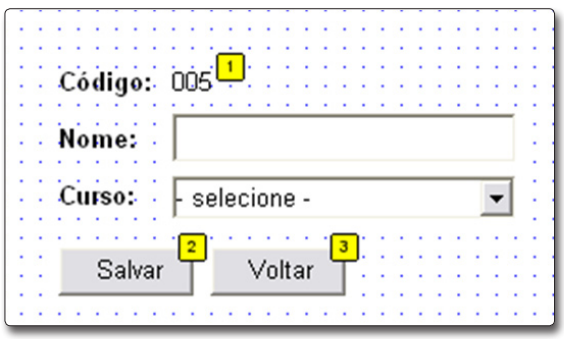


Figura 6. Formulário da página "incluirAluno"

Criaremos, também, a página de inserção de notas, que irá abrir em uma janela pop-up com os campos de código, nome, curso e nota, além dos botões de Salvar e Fechar. Para o botão Fechar, configuramos de maneira semelhante ao campo de inserir nota, mas apenas selecionamos a opção "Close Current Window", para fechar a janela pop-up (ver **Figura 7**).

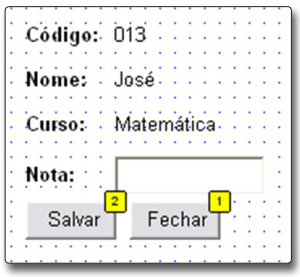


Figura 7. Formulário da que será exibido na janela *popup* "inserirNota"

O Axure permite que sejam criadas partes de telas que podem ser utilizadas por mais de uma página, chamadas masters, como o caso de menus, cabeçalhos e rodapés. Para isso, criaremos três partes na aba "Masters", que corresponderão ao cabeçalho, rodapé e conteúdo (ver **Figura 8**).

Figura 8. Fragmentos de páginas criados na aba "Masters"

Para o cabeçalho, usaremos o componente de imagem que exibirá o logo do sistema e componentes de texto para o nome do sistema. Semelhantes ao cabeçalho, serão usados os mesmos componentes para a exibição do rodapé. Após essas configurações, será possível adicionar tanto o cabeçalho, quanto o rodapé nas páginas já criadas anteriormente. Iremos adicioná-las às páginas listarAlunos e incluirAlunos (ver **Figura 9**).

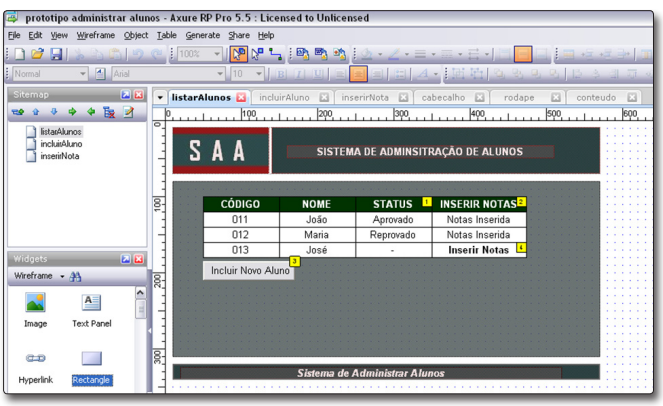


Figura 9. Página "listarAlunos" com a tabela de alunos e *masters* adicionados.

O uso de masters é essencial se for necessário reutilizar várias vezes o mesmo pedaço de página em várias outras páginas do sistema como, por exemplo, o uso de menus, que o Axure possui tanto os verticais quanto os horizontais, que podem também ser reaproveitados no sistema real. O uso de master deve ser usado com cuidado, pois qualquer modificação feita em um master será alterada também em todas as páginas que o utilizam.

Criando o Diagrama de Fluxo

O Axure permite ainda criar os diagramas de fluxo das páginas criadas. Para isso, criamos uma página "Sitemap" chamada de "fluxoPaginas" e modificamos o tipo de página para flow, com o botão direito em Diagram Flow (ver **Figura 10**).

Figura 10. Diagrama de fluxo "fluxoPaginas" na aba "Sitemap"

Arrastamos as páginas criadas anteriormente para dentro de fluxoPaginas, para que não seja necessário criar suas representações no diagrama. Depois criamos as condições e direção do fluxo. Da mesma forma, pode-se alterar a formatação pelo menu, como foi feito nas páginas anteriores (ver **Figura 11**).

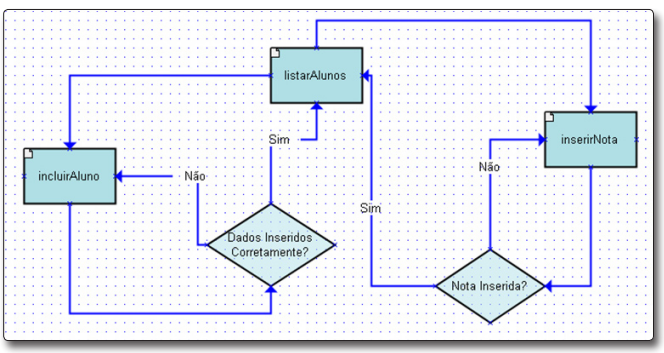


Figura 11. Diagrama de fluxo com as navegações já criadas

Gerando o Protótipo e a Especificação

Para gerar o protótipo, basta selecionar no menu a opção de Prototype ou pressionar F5. Será exibida uma janela para configurações em que se pode configurar com deve ser feita a geração do protótipo. Neste artigo, não serão usadas essas configurações, apenas a seção general em que se indica o local onde será criado o protótipo. Quando o protótipo é criado, o Axure cria as páginas e pastas correspondentes às páginas contendo imagens, arquivos CSS e arquivos JavaScript que realizam todas as ações configuradas anteriormente.

Após o protótipo ter sido criado, é aberta uma página contendo uma lista de todas as páginas e diagramas de fluxos, a página selecionada e a descrição dessa página que foi inserida previamente em "Page Notes" (ver **Figura 12**).

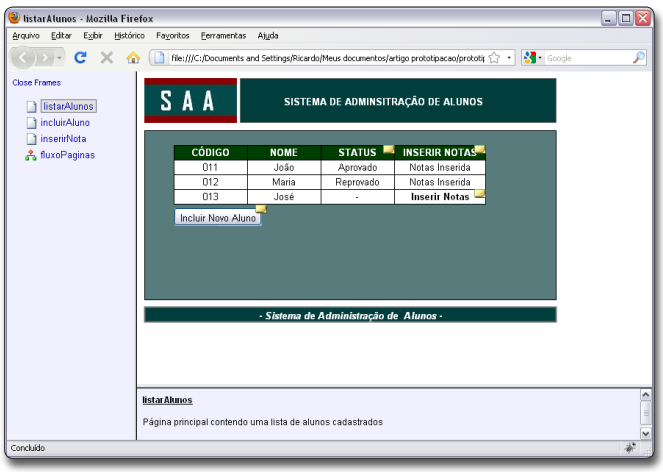


Figura 12. Exibição dos protótipos no navegador

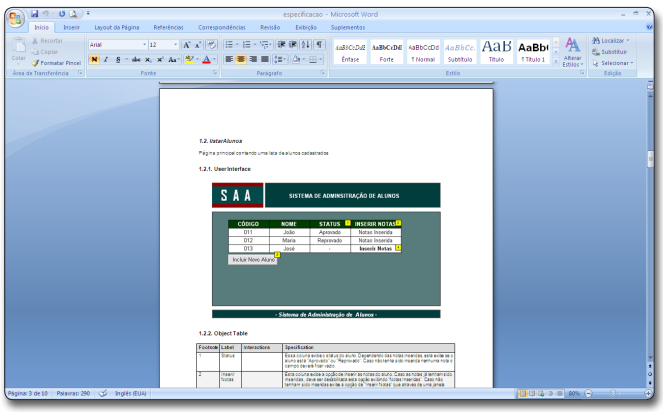


Figura 13. Exibição do documento de especificação no Word 2007

Para gerar a especificação, aperte F6 ou selecione a opção Specification no menu. O Axure irá gerar um documento no formato padrão Word, com todas as informações existentes, podendo ser utilizados outros editores de texto (ver **Figura 13**).

Pode-se ainda gerar imagens de suas páginas, selecionando a opção no menu "File". Serão geradas imagens somente das páginas e diagramas e fluxo selecionados, com o formato desejado que pode ser Bitmap, PNG, JPG e GIF.

Conclusão

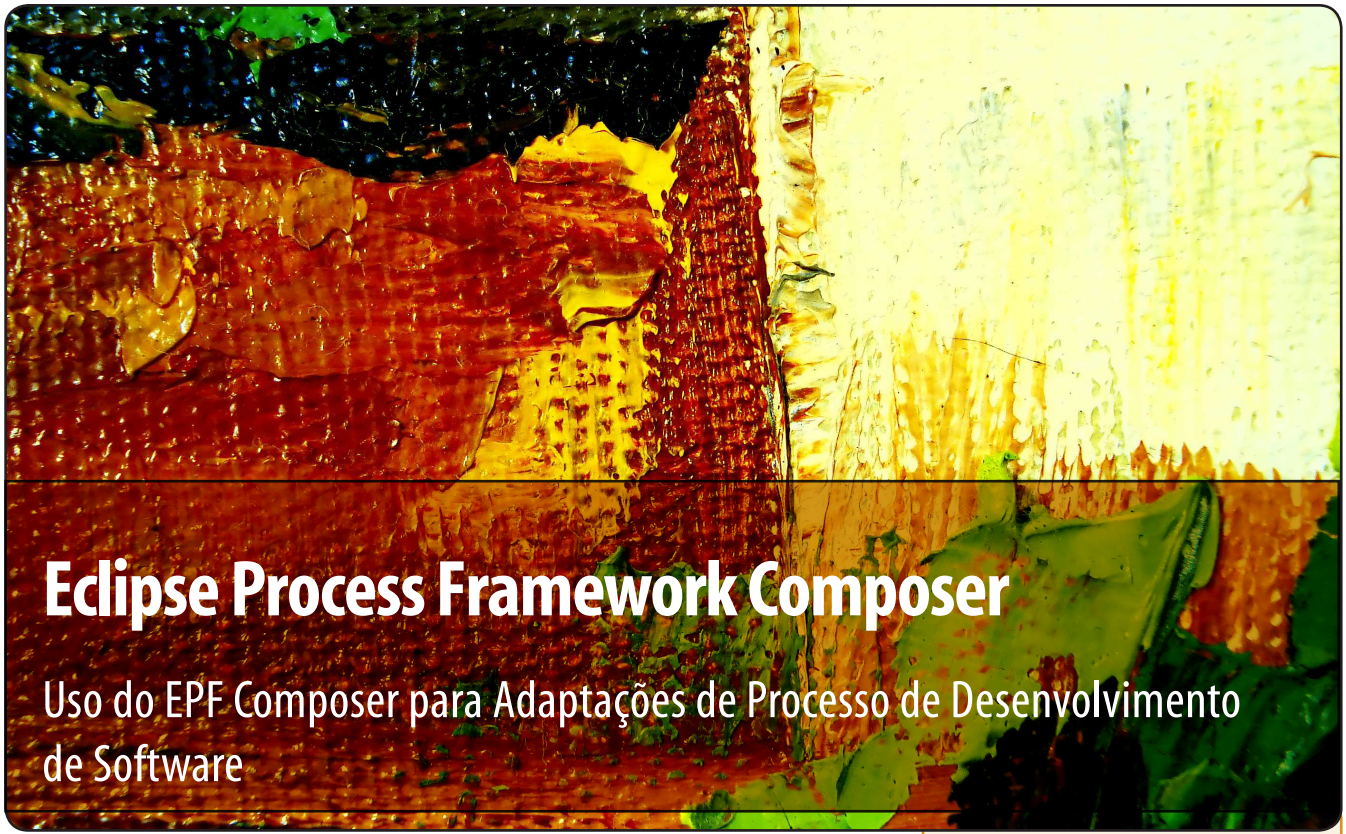
O artigo apresentou alguns conceitos sobre prototipação, definindo e exemplificando os tipos existentes. Foi visto que a prototipagem pode ser positiva em vários casos, somente dependendo de uma análise do desenvolvedor para saber qual o melhor método a se adotar.

Foram citadas ferramentas que auxiliam neste processo, em especial o Axure RP Pro. O aplicativo que foi utilizado na elaboração do caso de uso se mostrou simples, eficiente e, com certeza, irá ajudar a qualquer desenvolvedor que deseja construir um protótipo. ●

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



Eclipse Process Framework Composer

Uso do EPF Composer para Adaptações de Processo de Desenvolvimento de Software

Felipe Furtado
felipe.furtado@cesar.org.br
Graduado em Engenharia Mecânica pela UFPE, especialista em Tecnologias da Informação pelo Centro de Informática da UFPE e mestre em Ciência da Computação na área de produtividade de software também pela UFPE. 13 anos na área de desenvolvimento de software, é coordenador da garantia da qualidade de software e do SEPG, liderando também a melhoria dos processos das áreas de gestão de projetos no C.E.S.A.R. Possui experiência na definição e implantação de processos de gerenciamento de projetos de TI aderentes ao CMMI e em avaliações SCAMPI, com participação em avaliações Classe A CMMI níveis 2 e 3.

Andrea Pinto
andrea.pinto@cesar.org.br
É graduada em Ciência da Computação pela UFPE e mestra em Ciências da Computação pela UFPE na área de Engenharia de Requisitos. Com 9 anos de experiência na área de desenvolvimento de software, atualmente é Engenheira de Qualidade de software do C.E.S.A.R. Possui experiência na definição e implantação de processos de garantia da qualidade aderentes ao CMMI e em avaliação SCAMPI, tendo participado da avaliação Classe A CMMI 3 do C.E.S.A.R. Ministra aulas de Garantia da Qualidade de Software na faculdade Unibratec e orienta alunos em monografias para essa área. Tem interesse pricipalmente nas áreas de garantia de qualidade, modelos de qualidade, requisitos e metodologias ágeis.

Teresa Maria Medeiros Maciel
teresa@cesar.org.br
É bacharel em ciência da computação pela UNICAP, MBA em gestão de negócio pela UFPE, mestre e doutoranda em engenharia de software pelo Centro de Informática da UFPE. É auditora líder ISO9001 pelo BVQI, com formação BlackBelt em SixSigma. Atua há cerca de 20 anos em tecnologia da informação, tendo dedicado os últimos 10 anos a área de qualidade de software. Durante esse período exerceu a função de gerente de qualidade no C.E.S.A.R liderando projetos de melhoria, incluindo a obtenção do nível de maturidade 2 do SW-CMM, nível 3 do CMMI e 2 certificações ISO9001:2000.

Elisabeth Morais
beth.morais@cesar.org.br
É graduada em Ciência da Computação pela UFPE e mestra em Ciências da Computação pela UFPE. Com mais de 20 anos na área de desenvolvimento de software, atualmente é engenheira de qualidade de software do C.E.S.A.R. Possui experiência na área de Ciência da Computação, com ênfase em Engenharia de Software, atuando principalmente nos seguintes temas: qualidade de software, melhoria de processo de software e modelos de processo de software. Foi gerente do projeto CMMI 3 do C.E.S.A.R e participou dos projetos avaliados no CMM-SW 2. Publicou artigos na área de processo de software e gestão de projetos em eventos nacionais e internacionais.

De que se trata o artigo?
Este artigo relata a experiência de uma Organização CMMI, nível 3, no uso da ferramenta Eclipse Process Framework Composer para definição, adaptação e publicação de processos de desenvolvimento de software.

Para que serve?
A utilização de um framework desta natureza é útil para as Organizações que possuem um ambiente de melhoria dos processos de software e tem a necessidade de adotar uma ferramenta de definição de processos robusta que possibilite reuso e uma maior agilidade na definição e publicação dos processos organizacionais e dos processos adaptados para os projetos.

Em que situação o tema é útil?
Em ambientes que possuem uma grande diversidade de tipos de projetos e necessidade de padronização de um processo organizacional de forma produtiva e consistente.

Este artigo relata a experiência de uma Organização CMMI (*Capability Maturity Model Integration*), nível 3, no uso da ferramenta EPF (*Eclipse Process Framework Composer*) para estruturação, documentação, adaptação e publicação de processos

de desenvolvimento de software. A principal necessidade da organização em questão era a definição e institucionalização de um processo organizacional que atendesse a uma grande diversidade de projetos de desenvolvimento de software de forma produtiva e consistente.

Com a adoção de práticas de definição de processos aderentes às áreas OPD (*Organizational Process Definition*) e OPF (*Organizational Process Focus*) do nível 3 do CMMI, um dos principais objetivos da Organização é a melhoria dos processos de desenvolvimento de software. Para o atendimento desse objetivo, a Organização deveria adotar o uso de uma ferramenta robusta de definição de processos que possibilitasse o reuso de componentes do processo e uma maior agilidade na definição e publicação dos processos organizacionais e dos processos adaptados para os projetos [SEI, 2006].

Além desse objetivo principal, a Organização possui uma grande diversidade de projetos em função de várias características que requerem definições específicas a cada contexto, por exemplo, natureza do projeto, porte e complexidade do projeto, domínio de atuação, linha do produto, tipo do cliente, nível de criticidade dos testes, grau de definição dos requisitos, uso de uma arquitetura padrão, necessidade de uma gestão ágil, variações nas técnicas de *peer review*, dentre outras.

Devido a estas necessidades, foi realizado um processo de tomada de decisão formal para a escolha da ferramenta mais adequada, considerando os seguintes fatores:

- Possibilidade de adquirir uma ferramenta única de baixo custo para edição, adaptação e publicação de processos;
- Custo da reestruturação do processo já documentado na Organização;
- Fomentar a reutilização dos ativos do processo: *templates*, atividades, papéis etc.;
- Facilidade da descrição dos critérios de adaptação dos processos para os projetos;
- Apoio na análise de impacto entre os elementos do processo;
- Grau de usabilidade da ferramenta;
- Facilidade de manutenção do processo;
- Aproveitamento de frameworks pré-definidos da indústria.

Várias ferramentas foram analisadas, dentre elas, destacam-se: *Eclipse Process Framework (EPF)*, *Microsoft Visio* e *Rational Method Composer*.

Com base nisso, este artigo relata a experiência da Organização na escolha e uso da ferramenta EPF [Haumer, 2007] para documentação, adaptação e publicação do processo de software, e está organizado da seguinte forma: fundamentação do conceito de adaptação de processos, principais conceitos e funcionalidades da ferramenta escolhida, relato da experiência de uso do EPF, e, finalmente, as principais conclusões e lições aprendidas.

Adaptação de Processo

Organizações que possuem um processo padrão definido, onde são descritas todas as fases e atividades do desenvolvimento de software, geralmente precisam adaptá-lo no escopo dos seus projetos, isto é, definir uma instância do processo padrão adequada às necessidades do projeto e aderente ao processo organizacional [SEI, 1994].

As adaptações do processo organizacional devem ser documentadas na fase de planejamento do projeto e estão freqüentemente relacionadas às necessidades específicas do projeto, por exemplo, procedimentos padrões do cliente que devem ser seguidos, não realização de determinada atividade do processo padrão organizacional, como documentação de requisitos, num projeto onde os requisitos serão documentados pelo cliente; não utilização de determinados *templates* pré-definidos, etc. [Buldlong *et AL*, 1996].

Para orientar como as adaptações devem ser realizadas no contexto do projeto, o processo organizacional deve conter os critérios de adaptação (vide **Figura 1**). Os critérios de adaptação descrevem as atividades do processo organizacional que são obrigatórias independentemente do escopo do projeto, atividades de acompanhamento do projeto, por exemplo; e deve também descrever como as atividades não obrigatórias devem ser adaptadas levando em consideração, por exemplo, variáveis como duração e tamanho do projeto, tamanho da equipe, requisitos de qualidade a serem atendidos no projeto, entre outros.

A definição de critérios de adaptação para o processo organizacional é uma atividade particularmente complexa, pois exige um conhecimento profundo dos tipos de aplicação desenvolvidas na organização a fim de se prever as possibilidades e diminuir o esforço da atividade de definição do processo adaptado [Xu, 2005].

Uma grande dificuldade relacionada às adaptações do processo é a avaliação de quais atualizações no processo organizacional devem ser refletidas no processo adaptado, isto é, como manter o processo adaptado atualizado levando em consideração as melhorias e correções realizadas no processo organizacional.

O uso de uma ferramenta auxilia a documentação do processo organizacional e dos critérios de adaptação, bem como a documentação do processo adaptado, como veremos nas próximas seções.

Nota do DevMan

CMMI (Capability Maturity Model Integration): é um framework que descreve princípios e práticas relacionadas ao processo de desenvolvimento de produtos e serviços tecnológicos. O modelo visa ajudar organizações envolvidas com o desenvolvimento de software a melhorar a capacidade de seus processos, por meio de um caminho evolucionário que considera desde processos com resultados imprevisíveis e até mesmo caóticos para processos disciplinados e definidos, com resultados previsíveis e com possibilidade de melhoria contínua. Uma organização que possui nível 3 significa que existe um processo composto por atividades de gerenciamento e engenharia, é documentado, padronizado e integrado em um processo padrão da organização. Todos os projetos utilizam uma versão aprovada e adaptada do processo organizacional para desenvolvimento e manutenção de produtos e serviços tecnológicos [SEI, 2006].

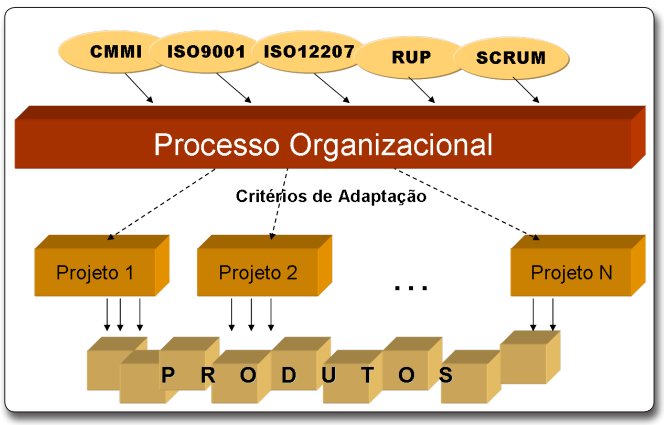


Figura 1. Modelo de adaptação de processo

EPF Composer

O *Eclipse Process Framework Composer* é definido como um ecossistema colaborativo e aberto para processos de desenvolvimento de software evolutivos [Haumer, 2006]. Ele é fundamentado em um meta-modelo que é a base para uma grande variedade de processos que endereçam às necessidades de uma Organização na definição, adaptação e publicação de processos.

O EPF é formado por vários elementos básicos dividido em duas perspectivas, conforme mostra a **Figura 2**:

• **Perspectiva de disciplinas ou sub-processos**: descreve os elementos base de um processo, por exemplo, tarefas, papéis, produtos de trabalho, dentre outros, também chamado de *Method Content*. A perspectiva *Method Content* é dividida em três pacotes que facilitam a organização dos elementos de um processo:

- *Content Package*: Biblioteca dos ativos de um processo sem a descrição dos relacionamentos entre os ativos: atividades, guias, papéis, templates, etc.;
- *Standard Categories*: Categorias padronizadas disponibilizadas pelo EPF, como *Disciplines*, *Domains* e *Work Product Kind*;
- *Custom Categories*: Estruturação das visões do processo; a partir de uma *Custom Category* pode-se elaborar de que maneira os elementos do processo serão apresentados na publicação do processo em formato web.

• **Perspectiva temporal ou de execução**: descreve os ciclos de vida, fluxo das atividades, dentre outros, também chamado de *Process*, subdivido da seguinte maneira:

- *Capability Patterns*: Local onde os processos são estruturados, isto é, onde os elementos previamente criados na perspectiva *Method Content* devem ser relacionados para a construção dos processos da organização. Os processos podem ser criados independentemente do ciclo de vida a ser utilizado, isto é, os relacionamentos entre eles podem ser definidos separadamente, possibilitando a criação de ciclos de vida diferentes no contexto dos projetos;
- *Delivery Process*: Construção dos ciclos de vida de desenvolvimento, isto é, definição da ordem na qual cada um dos processos deve ser executado e dos relacionamentos entre eles. A definição de ciclos de vida pode ser realizada tanto para o processo organizacional, como para os processos adaptados, ou seja, há a possibilidade de se adaptar, também, o ciclo de vida de desenvolvimento organizacional para atendimento às necessidades específicas de um projeto.

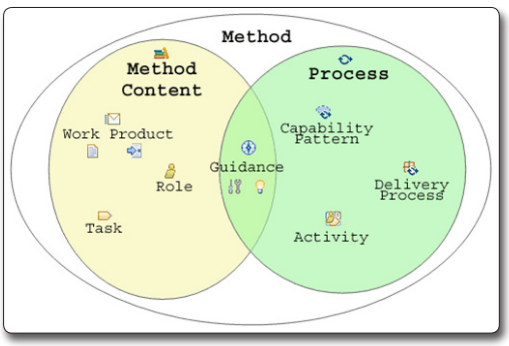


Figura 2. Elementos básicos do EPF Composer

Content Variability

O *content variability* é um mecanismo fornecido pela ferramenta que permite que os elementos do *Method Content*, descritos na **Figura 2,** possam ser customizados sem a necessidade de modificar diretamente o conteúdo original, ou seja, podemos criar uma adaptação do processo padrão organizacional para o contexto do projeto sem que seja necessário modificar o processo organizacional. Ao mesmo tempo, esse mecanismo permite que o processo padrão seja modificado sem que as adaptações no contexto dos projetos sejam perdidas.

As adaptações são realizadas através de *plug-ins* com base no processo padrão, ou seja, um *plug-in* corresponde a um PDP (Processo Definido do Projeto). Um *plug-in* pode contribuir (*contributes*), estender (*extends*) ou substituir (*replace*) elementos do processo padrão organizacional.

A opção *'contributes'* deve ser utilizada quando a adaptação de um elemento do processo para o projeto consiste na manutenção do conteúdo do elemento original e inclusão de novos conteúdos específicos para a adaptação. Essa opção deve ser utilizada quando há a necessidade, por exemplo, de incluir mais passos numa determinada atividade do processo organizacional. Com a utilização dessa opção, o elemento original do processo organizacional não consta no processo adaptado.

Na segunda opção, *'extends'*, o elemento adaptado herda o conteúdo do elemento original e permite que ele seja especializado, sendo possível, por exemplo, especializar uma técnica de revisão específica para um projeto. Com o uso do *extends*, o processo adaptado apresenta o elemento original e o elemento adaptado.

Por último, é possível utilizar *'extends-replace'*, similar ao *'extends'*, com a exceção de que apenas o elemento adaptado permanecerá no processo.

Dessa maneira, o mecanismo *Content Variability* propicia o reuso dos elementos do processo organizacional, facilita a adaptação do processo organizacional para o contexto dos projetos e a manutenção dos processos adaptados.

Nota do DevMan

OPD: É a área de processo de Definição do Processo Organizacional (OPD) que estabelece e mantém o conjunto de processos padrão da organização e outros ativos baseados nas necessidades do processo e nos objetivos da organização [SEI, 2006].

Estudo de Caso

Antes de adotar o EPF na Organização, as adaptações do processo para os projetos eram documentadas através de uma planilha Excel, a qual continha a relação de todas as atividades do processo, agrupadas por disciplina de *software*, da seguinte forma: **Gestão** (Pré-venda, Planejamento e Acompanhamento de Projetos); **Engenharia** (Requisitos, Análise e Projeto, Implementação, Testes e Implantação); e **Suporte** (Garantia da Qualidade, Gerência de Configuração, Revisão e Aprovação, Medição e Análise e Tomada de Decisão).

Um das principais dificuldades era atualizar o processo adaptado de acordo com as mudanças do processo organizacional, pois era necessário verificar cada uma das requisições de mudança do processo organizacional e atualizar a planilha Excel manualmente.

Após a adoção do EPF, para elaboração do processo organizacional, foi realizado um projeto piloto para adaptação de processos usando também essa ferramenta, cujos objetivos foram:

- Definir a configuração de ambiente no EPF para a definição dos processos adaptados dos projetos;
- Propor uma estrutura de pacotes para os processos adaptados dos projetos, visando a reutilização dos elementos do processo organizacional;
- Selecionar os tipos de relacionamentos do EPF (*variability*) mais apropriados para uso nos processos adaptados;
- Melhorar a usabilidade do processo, criando uma visão adaptada do processo para cada projeto;
- Disseminar o conhecimento para os responsáveis por definir as adaptações dos processos dos projetos (engenheiros de qualidade e demais interessados).

As seções a seguir apresentam as etapas realizadas no projeto piloto para cumprir os objetivos listados acima.

Ambiente de Trabalho no EPF

Uma vez que o repositório do processo organizacional estava sob gerência de configuração na ferramenta *Subversion* e o EPF possui integração com essa ferramenta, esse mesmo repositório foi selecionado para armazenar os processos adaptados dos projetos. Para configurar esse repositório de processos utilizamos a perspectiva *SVN Repository Exploring*.

Ao abrir uma perspectiva no EPF, são apresentadas algumas visões (*views*) pré-configuradas, entretanto, é possível acrescentar outras visões de acordo com a necessidade de cada usuário (vide opções *Views>Window>Show View*). Dessa forma, a perspectiva *Authoring* foi configurada para apresentar as seguintes visões:

- ***Library***: apresenta a biblioteca que contém os pacotes de processos e respectivos elementos do EPF;
- ***Navigator***: utilizada para visualizar o *branch* de desenvolvimento, fazer *commit*, visualizar *labels*, dentre outras opções;
- ***Console***: informa o status das operações realizadas no repositório (ex: status do *commit*);
- ***History***: apresenta o histórico das atualizações realizadas no repositório.

A **Figura 3** apresenta uma visão do ambiente de trabalho no EPF.

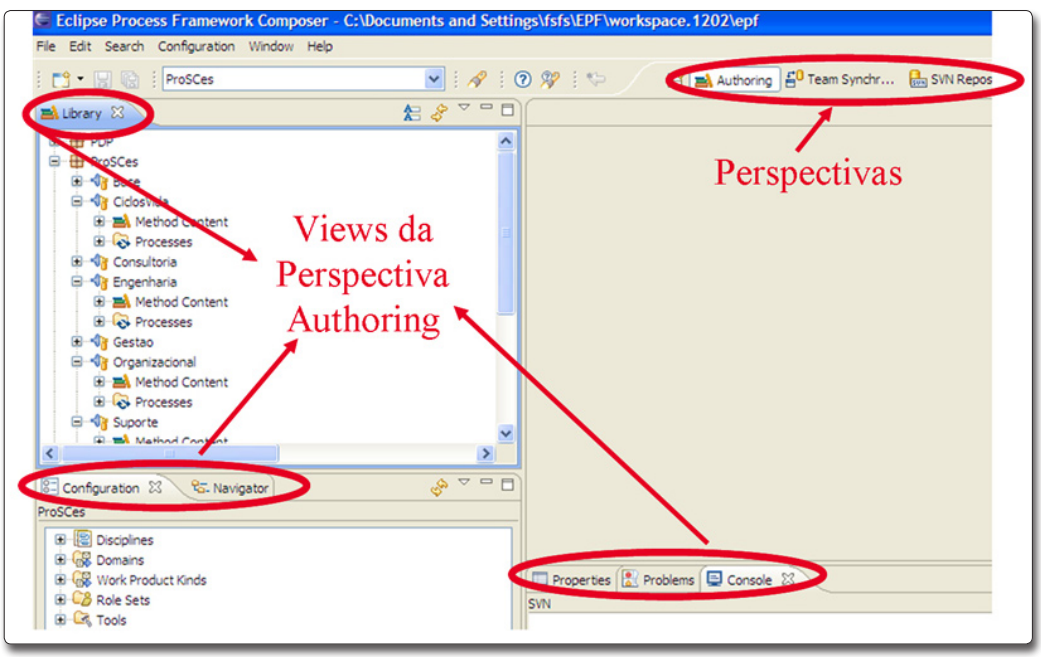


Figura 3. Ambiente de trabalho no EPF Composer

Nota do DevMan

OPF: É a área de processo de Foco no Processo Organizacional (OPF) que auxilia a organização a planejar e implementar melhorias nos processos organizacionais baseadas em um entendimento dos atuais pontos fortes e fracos dos processos e ativos de processos da organização [SEI, 2006].

Estrutura de pacotes dos processos adaptados

A visão ***Library*** é utilizada para modificar os ativos do processo (*templates*, atividades, papéis, fluxos etc.), os quais são mapeados em elementos do EPF. Foi criado um *method plug in* separado do processo organizacional para os processos adaptados, denominado PDP, abaixo do qual também foi criado um *method plug in* para cada projeto adaptado formando subpacotes.

A mesma estrutura do processo organizacional foi definida abaixo do *method plug in* de cada projeto, sendo criados pacotes de conteúdo (*content packages*) para cada uma das áreas de processo e mais um pacote base com a definição. Abaixo dos pacotes das áreas foram criados outros pacotes de conteúdo para as respectivas disciplinas, objetivando o registro das adaptações de suas atividades (tarefas). Enquanto que abaixo do pacote base, foram criados pacotes de conteúdo para os ativos comuns às diversas disciplinas do processo, tais como, artefatos, *templates*, papéis, guias e procedimentos. Tal estrutura de pacotes é apresentada na **Figura 4**.

É importante ressaltar que o *method plug in* do projeto é configurado para referenciar os *method plug ins* do processo organizacional, possibilitando o reuso de seus elementos. Dessa forma, o pacote do projeto contempla apenas elementos que possuam adaptações específicas para projeto.

Uso dos Relacionamentos do EPF (Variabilidade)

Dentre os tipos de relacionamento descritos na seção "**Content Variability**", o mais utilizado foi a variabilidade ***Contributes***, a qual permite incluir informações complementares aos elementos existentes do processo organizacional, definindo apenas as regras específicas de cada projeto.

A principal vantagem desse tipo de relacionamento é que todas as mudanças do processo organizacional são facilmente herdadas pelos projetos adaptados, bastando gerar novamente o site do projeto a partir do novo release publicado do processo organizacional. Isto reduz o retrabalho dos processos adaptados para os projetos e oferece uma maior agilidade na disponibilização das mudanças do processo organizacional para os projetos.

Alguns exemplos de informações que podem ser adicionadas são: passos, papéis, ferramentas, *templates*, guias, procedimentos, bem como considerações chave associadas à determinada atividade.

Porém, houve situações em que foi necessário criar elementos novos, tais como atividades, artefatos e *templates* específicos para os projetos. Nesses casos, nenhuma variabilidade é utilizada, pois o elemento é criado sem nenhum reuso.

Em outras situações mais raras, existiu a necessidade de sobrescrever uma atividade, mantendo apenas o mesmo nome. Nesses casos, foi utilizada a variabilidade *Replace*.

Site do Processo Adaptado

Com o uso do EPF, os processos adaptados dos projetos passaram a ser publicados para as equipes por meio de um site em formato HTML, gerado automaticamente a partir de uma configuração do processo do projeto, utilizando uma diagramação padrão da ferramenta.

A primeira página do site é composta de dois *frames*, onde o *frame* esquerdo contém as opções do *menu*, e o *frame* direito

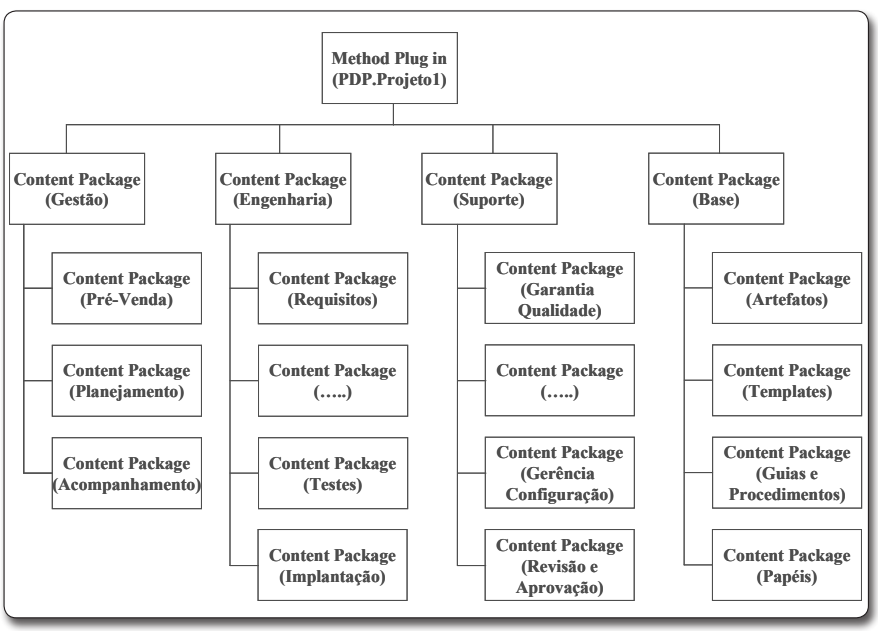


Figura 4. Estrutura de pacotes dos processos adaptados

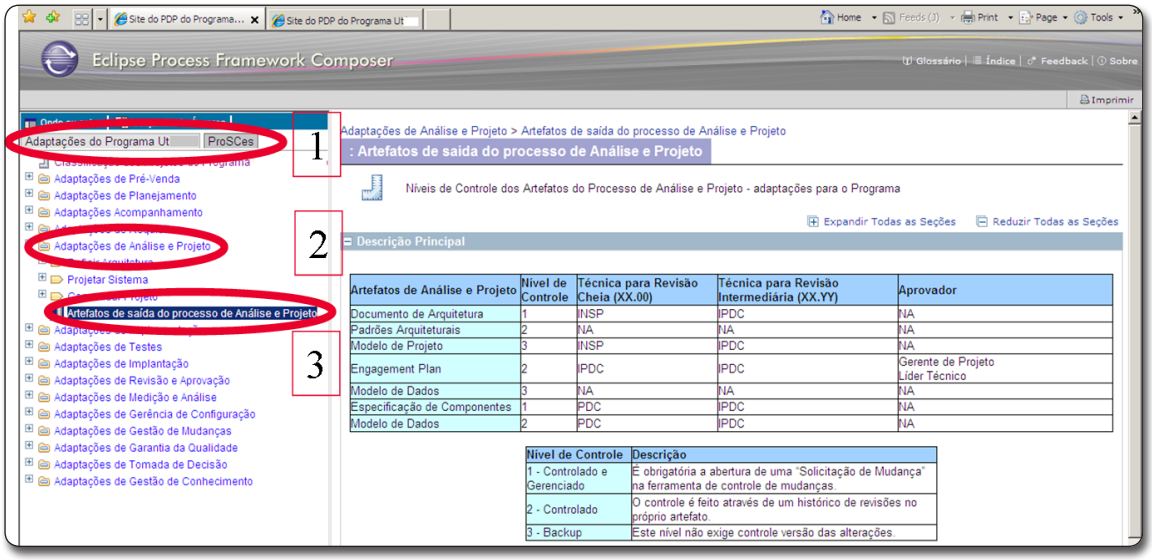


Figura 5. Site do processo adaptado

apresenta o conteúdo da página da opção selecionada. As opções do primeiro nível de *menu* são apresentadas horizontalmente, enquanto que as opções a partir do segundo nível são apresentadas na vertical, podendo ser criados tantos sub-menus quanto necessário. A **Figura 5** mostra a estrutura padrão do site gerado pelo EPF, apresentando uma página selecionada através de uma opção do terceiro nível de *menu*. O conteúdo da página foi criado em HTML, tendo sido registrado através de um formulário do tipo guia ("*guidance/guideline*").

Para cada site, foi criada uma configuração no EPF que referencia os *plug ins* e pacotes de método do próprio projeto, bem como do processo organizacional. Além disso, uma configuração do EPF referencia *views* que são "categorias customizadas", um elemento padrão do *method plugin*, que correspondem às opções do *menu* horizontal do site do projeto. Uma categoria customizada pode conter subcategorias, as quais correspondem às opções do *menu* vertical. Dessa forma, para criar o site do processo do projeto basta gerá-lo a partir da configuração padrão correspondente e publicá-lo na intranet da organização.

Conclusões e Lições Aprendidas

A escolha do EPF atendeu a necessidade de otimizar a manutenção do processo organizacional e no ganho de produtividade nas adaptações para os diferentes tipos de projetos. Os principais benefícios observados com a adoção do EPF foram:

- Utilização de uma única ferramenta para edição, adaptação e publicação de processos. Por ser uma plataforma aberta e gratuita, temos suporte da comunidade do EPF sem custo adicional;
- A ferramenta provê a simbologia homologada pela OMG para a engenharia de processos de software, o SPEM;
- Maior consistência das informações dos processos adaptados para os projetos em relação às atualizações do processo organizacional, pois as mesmas são geradas de forma automática;

Nota do DevMan

SPEM (Software Process Engineering Meta-Model): É uma linguagem para modelagem de processos de software oficializada como um padrão da OMG e encontra-se atualmente na versão 2.0.

Nota do DevMan

PDP: É o processo definido para um projeto específico. Ele é criado através de uma adaptação a partir do processo padrão da organização, em função das caracteríticas do projeto, necessidades específicas do cliente, restrições do projeto etc.

- Redução do retrabalho dos processos adaptados para os projetos e maior agilidade na disponibilização das melhorias do processo organizacional;
- Facilidade do acesso às informações do processo pelas equipes do projeto e do cliente, melhorando a visibilidade das adaptações;
- Reutilização dos ativos do processo: a biblioteca de ativos de cada sub-processo fica disponível para todos os membros da Organização responsáveis pela manutenção do processo, no nosso caso, o SEPG (*Software Engineering Process Group*);
- Aproveitamento de boas práticas de *frameworks* pré-definidos pela comunidade. Foram disponibilizadas bibliotecas prontas de ativos para Scrum, XP e OpenUP.

Entretanto, algumas lições aprendidas também foram identificadas com o objetivo de auxiliar outras Organizações que desejem adotar a ferramenta EPF:

- Curva de aprendizado elevado na ferramenta e estruturação dos elementos do processo organizacional;

• Estruturar o processo organizacional de forma a otimizar o reuso de elementos do processo sem a necessidade de herdar um pacote pesado. Também avaliar previamente a estratégia de *branches* a ser adotado para evitar conflitos durante o processo de integração;
• Definir responsáveis na Organização para prover suporte no uso da ferramenta e utilizar uma ferramenta colaborativa para compartilhar experiências do uso. ●

Dê seu feedback sobre esta edição!

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

www.devmedia.com.br/esmag/feedback

Referências

SEI (2006). "CMMI for Development, version 1.2, staged representation". http://www.sei.cmu.edu/pub/documents/06.reports/pdf/06tr008.pdf, CMU/SEI-2006-TR-008

SEI (1994). "Process Tailoring and the Software Capability Maturity Model". http://www.sei.cmu.edu/pub/documents/94.reports/pdf/tr24.94.pdf, CMU/SEI-1994-TR24

Haumer, P. (2007). "Eclipse Process Framework Composer: Key Concepts and Authoring Method Content and Processes". http://www.eclipse.org/epf/general/

Xu, Peng. (2005). "Knowledge Support in Software Process Tailoring". Proceedings of the 38th Hawaii International Conference on System Sciences – 2005.

Budlong, F., Szulewski, P., Ganska, R. (1996). "Process Tailoring for Software". The Software Technology Support Center (STSC) of the U.S. Air Force. http://www.stsc.hill.af.mil/resources/tech_docs/process_plan/

SPEM – Software Process Engineering Meta-Model: http://www.omg.org/technology/documents/formal/spem.htm

Gerenciamento de Configuração

Gerenciamento de configurações utilizando o ITIL e a ferramenta Rational ClearCase

Felipe La Rocca Teixeira
felipe.lt@pjf.mg.gov.br
Pós-Graduado em Gestão de Projetos - PMI, Bacharel em Sistemas de Informação, Analista de Sistemas da Prefeitura de Juiz de Fora e possui certificações MCITP Database Administrator, MCITP Business Intelligence Developer e OCA 10g.

Marco Antônio Pereira Araújo
maraujo@cesjf.br
Doutor e Mestre em Engenharia de Sistemas e Computação pela COPPE/UFRJ, Especialista em Métodos Estatísticos Computacionais e Bacharel em Informática pela UFJF, Professor e Coordenador do Curso de Bacharelado em Sistemas de Informação do Centro de Ensino Superior de Juiz de Fora (CES/JF), Analista de Sistemas da Prefeitura de Juiz de Fora.

Atualmente o mundo está em constante aperfeiçoamento no uso de tecnologias em todas as áreas de conhecimento, de métodos e de técnicas organizacionais que venham a possibilitar a continuidade e o crescimento dos negócios das organizações em uma sociedade moderna. Desta forma, para que as organizações de TI possam cumprir seus objetivos com qualidade, é altamente desejável estarem alinhadas com os objetivos do negócio ou da organização.

De que se trata o artigo?
Este artigo procura apresentar processos que garantam um gerenciamento completo de todos os itens de configuração de um projeto de TI, podendo assim, realizar um controle efetivo de versões e de mudanças, bem como, a auditoria destas configurações.

Para que serve?
O artigo demonstra como o gerenciamento de configuração pode ser utilizado em um projeto de TI, e de como o uso de ferramentas como o Rational ClearCase pode ser utilizado para evitar problemas inevitáveis que ocorrem durante o processo de desenvolvimento de software, como por exemplo, o uso de versões diferentes de código-fonte, sincronização de trabalho paralelo entre os desenvolvedores em um mesmo mesmo projeto, recuperação de versões anteriores e registro do histórico de modificações ocorridas durante o processo de desenvolvimento.

Em que situação o tema é útil?
Gerenciamento de Configuração em projetos de TI, como o desenvolvimento de softwares.

No mercado existem várias metodologias para implantar os processos de gerenciamento de TI, entre elas, está a ITIL (*Information Technology Infrastructure Library*). A ITIL é um dos modelos mais utilizados pelas as empresas, que buscam através do uso de um conjunto de melhores práticas, processos que garantam o alinhamento dos serviços às necessidades da empresa, gerando assim, redução de custos e um aumento na eficiência destas empresas.

Neste artigo iremos abordar um de seus processos, o Gerenciamento de Configuração. Este processo fornece um modelo lógico para administrar toda a infra-estrutura de TI, como *hardware*, *software* e documentação, e também, iremos mostrar como a utilização da ferramenta *Rational ClearCase* auxilia na tarefa de Controle de Versões, com a identificação, controle e verificação das versões dos itens de um projeto de TI.

Gerenciamento de TI baseado no ITIL

A ITIL é uma biblioteca das melhores práticas e processos para o gerenciamento operacional dos serviços de TI. A sua utilização permite o gerenciamento dos serviços de TI com qualidade a um baixo custo. A ITIL surgiu no final dos anos 80 no Reino Unido, desenvolvida pela CCTA (*Central Computer and Telecommunication Agency*), e desde 2000, é coordenada pela OGC (*Office of Government Commerce*).

Atualmente, a ITIL encontra-se na versão três e a sua principal evolução em relação à versão anterior, é a organização dos processos de gerenciamento de serviços em uma estrutura de ciclo de vida de serviços. A ITIL está relacionada com todos os processos que precisam ser executados dentro das organizações para o gerenciamento e a operação da infra-estrutura de TI. Com isso, pode-se obter um excelente fornecimento de serviços, sob um esquema de custos alinhados com as estratégias do negócio.

Esses serviços de TI podem ser definidos como um conjunto de recursos da infra-estrutura de TI, que atendem a uma ou mais necessidades de seus clientes. Este conjunto de recursos está alinhado aos objetivos do negócio e é considerado pelo cliente como uma solução de apoio ao seu negócio. Os componentes que formam a infra-estrutura de TI são:

- Hardware;
- Software;
- Procedimentos;
- Documentos;
- Recursos Humanos.

A ITIL está organizada em vários módulos que permitem um alinhamento entre a tecnologia e o negócio das organizações (ver **Figura 1**). Neste artigo, será abordado somente o módulo de Administração de Serviços que se encontra no centro da biblioteca do ITIL. Este processo é dividido em duas áreas distintas, que são o Suporte de Serviços e Entrega de Serviços, e esses, por sua vez, são divididos em processos que visam garantir que os serviços de TI estejam alinhados com as necessidades do negócio das organizações.

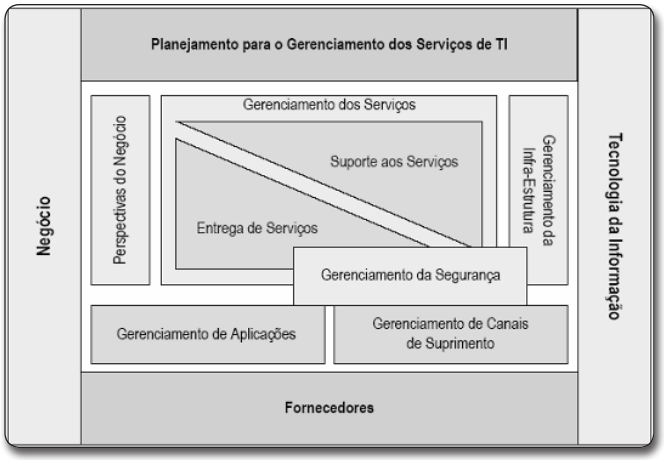


Figura 1. Representação dos módulos do ITIL

Suporte de Serviços

O suporte de serviços está relacionado com as operações diárias de suporte e de manutenção dos serviços de TI. Seu objetivo é manter a entrega dos serviços e concentrar-se em seu suporte e manutenção. A seguir, segue a descrição dos processos de serviços de suporte.

O **Gerenciamento de Configuração** é o processo que administra todas as informações sobre todos os Itens de Configuração de uma infra-estrutura de TI como *hardware*, *software* e documentação, bem como os relacionamentos entre esses itens. Um CI (*Configuration Item*) é qualquer componente ou elemento associado à infra-estrutura de TI que está sob o controle deste processo de gerenciamento.

As informações dos componentes estão armazenadas em uma base de dados chamada CMDB (*Configuration Management Database*). Uma CMDB ajuda a analisar tendências, além de proporcionar detalhes de todos os recursos da área de TI, e configurações da organização e seus serviços, sendo uma fonte importante de informação para o apoio da gestão dos serviços. O Gerenciamento de Configurações também fornece suporte para outras áreas de TI como os processos de gerenciamento de Incidentes, Problemas, Alterações e Versões. Este processo será visto em detalhes no decorrer deste artigo.

O **Help Desk** tem como objetivo principal ser um ponto central de contato entre os provedores de serviços e os usuários. É o contato onde os usuários deverão relatar os seus incidentes, e solicitar serviços e alterações, além da monitoração desses serviços.

O **Gerenciamento de Incidentes** se preocupa em restabelecer a operação normal do serviço o mais rápido possível e minimizar possíveis impactos negativos sobre o negócio, e garantir assim, que os níveis de qualidade e disponibilidade dos serviços sejam mantidos. Um incidente pode ser definido como qualquer evento que não seja parte da operação normal de um serviço e que pode causar a sua interrupção ou redução de sua qualidade.

O **Gerenciamento de Problemas** consiste em localizar erros já conhecidos na infra-estrutura de TI. É um processo pró-ativo e, dessa forma, os problemas são identificados e solucionados

antes de ocorrer o erro, visando minimizar os efeitos adversos desses incidentes e problemas nos serviços.

O **Gerenciamento de Alterações** é responsável por garantir o controle completo das modificações na infra-estrutura de TI, realizando-as com um risco mínimo. Seu objetivo não é somente o controle das modificações, mas também o seu monitoramento, sua aprovação e análise de possíveis impactos.

O **Gerenciamento de Versões** realiza o controle de versões de software, hardware e de outros componentes da infra-estrutura de TI e, com isso, pode-se assegurar a implementação de versões compatíveis, licenciadas e apropriadas para os objetivos da organização.

Entrega de Serviços

A entrega de serviços esta focada no planejamento e na administração da qualidade dos serviços de TI. Concentra-se na melhoria continua dos serviços prestados, e está dividida em cinco processos.

O **Gerenciamento de Níveis de Serviços** procura manter e melhorar a qualidade dos serviços de TI, garantindo a identificação dos acordos de níveis de serviço, dos contratos operacionais e de terceiros. Esses acordos de níveis de serviços, chamados de SLA (*Service Level Agreement*), são divididos em Baseado no Serviço, Baseado no Cliente e Multi-nível.

O **Gerenciamento de Capacidade** garante o uso correto dos recursos de TI para atender o que foi acordado com os clientes. Nesse processo é fundamental o equilíbrio entre custo e a capacidade, bem como do fornecimento e a demanda.

O **Gerenciamento de Disponibilidade** é o processo responsável por garantir que os serviços estejam disponíveis de acordo com os níveis definidos entre a organização de TI e seus usuários. Através da medição e gerenciamento da disponibilidade da infra-estrutura de TI, pequenas diminuições dos níveis de serviço são rapidamente identificadas e ações corretivas podem ser tomadas.

O **Gerenciamento da Continuidade de Serviços de TI** desenvolve e implanta opções de recuperação quando uma interrupção de um serviço atinge um ponto definido. Ele também é responsável em gerenciar os riscos de falhas nos principais serviços de TI, através da prevenção e do planejamento de um plano de contingência, para dar suporte ao funcionamento contínuo dos negócios.

O **Gerenciamento de Finanças** tem como objetivo principal gerenciar efetivamente os custos dos ativos e recursos utilizados para oferecer adequadamente os serviços de TI. É o processo responsável pela contabilidade, orçamentos e encargos dos custos dos serviços de TI.

Gerenciamento de Configuração - ITIL

Realizar o controle efetivo da infra-estrutura de TI nem sempre é algo feito pelas organizações e, por isso, o processo de Gerenciamento de Configuração ajuda a manter o acompanhamento destes elementos de infra-estrutura, tais como, *hardware*, *software* e documentação. Este processo oferece a administração de todos os itens de configuração através da identificação, controle, manutenção e verificação dos itens de configuração (CIs) e tem como seus principais objetivos:

- Disponibilizar detalhadamente todos os recursos da área de TI e configurações da empresa e seus serviços;
- Oferecer informações exatas sobre as configurações e a documentação necessária para suportar todos os processos de gerenciamento de serviços;
- Auxiliar os processos de gerenciamento de incidentes, problemas, alterações e versões;
- Verificar os registros das configurações na infra-estrutura física, permitindo corrigir qualquer problema.

A utilização do Gerenciamento de Configuração traz vários benefícios para a empresa, pois disponibiliza informações confiáveis sobre a sua infra-estrutura, mantendo um inventário de todos os itens de configuração, e assim permitirá, no caso de ser solicitado por alguma auditoria, que sejam entregues informações sobre os ativos das empresas.

Permite também o controle dos contratos de manutenção e de atualizações de licenças, melhorando a qualidade da segurança contra *software* mal intencionado ou pirata, pois tem os detalhes das versões do *software* em uso dentro da empresa.

A seguir, pode-se verificar a linguagem comum de termos utilizados pelo Gerenciamento de Configurações segundo a ITIL:

- **CI**: componentes ou elementos associados à infra-estrutura de TI que está sob a administração do Gerenciamento de Configuração;
- **CMDB**: banco de dados que contém dados referentes a cada CI e seus relacionamentos;
- **Alcance**: responsabilidades cobertas pelo Gerenciamento de Configuração.
- **Nível de elemento de configuração:** Grau de detalhamento dos dados de entrada no banco de dados CMDB;
- **Atributos**: características ou qualidades que identificam um CI;
- **Relações**: contém as relações entre todos os elementos da infra-estrutura de TI.

Alguns exemplos de elementos de configuração (CIs) que fazem parte da infra-estrutura de TI:

- *Hardware*: PCs, servidores, impressoras, placa de rede, *hubs*, etc.;
- *Software*: sistema operacional, gerenciadores de bancos de dados, aplicativos, etc.;
- Bancos de dados físicos e versões de *software*;
- Documentação de sistemas, licenças, contratos de manutenção, contratos de nível de serviço;
- Usuários e fornecedores.

Atividades do processo de Gerenciamento de Configuração

As atividades realizadas no processo de Gerenciamento de Configuração estão divididas em cinco atividades, sendo Planejamento, Identificação, Controle, Estatísticas de Status de Configuração, Verificação e Auditoria (ver **Figura 2**).

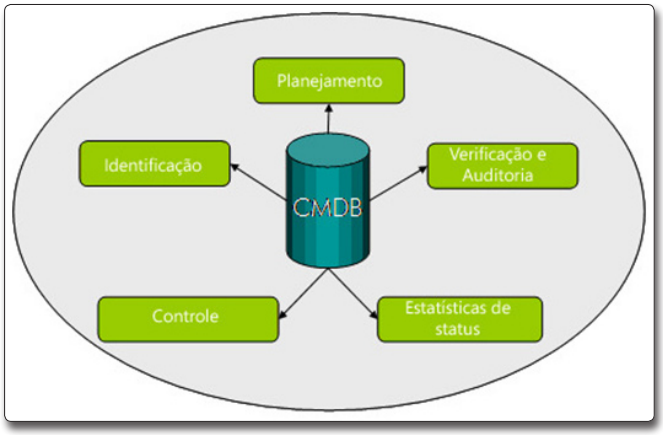


Figura 2. Atividades do processo de Gerenciamento de Configuração - ITIL

Na atividade de **Planejamento** serão definidos os objetivos e procedimentos principais para iniciar o processo de Gerenciamento de Configuração. Por isso, é altamente recomendável que essas atividades sejam sempre revisadas para a sua continuidade. As tarefas executadas na atividade de Planejamento significam exatamente o que será realizado pelo processo de Gerenciamento de Configuração. Entre essas tarefas destacam-se:

- Definir as estratégias, políticas e objetivos do Gerenciamento de Configuração;
- Analisar a situação atual do inventário e das configurações da infra-estrutura de TI;
- Administrar o relacionamento de outros processos de um projeto, tais como o Gerenciamento de Alterações e Gerenciamento de Versões;
- Localizar as áreas de armazenamento e bibliotecas utilizadas para manter o *hardware*, *software* e documentação.

A atividade de **Identificação** é responsável pela identificação dos elementos de configuração, fornecendo o detalhamento de cada um desses elementos e determinado a sua função dentro de qualquer ambiente. Esses elementos de configuração é que formarão o banco de dados de conhecimento, o CMDB. As principais atividades dessa atividade são:

- Determinar o ciclo de vida dos elementos de configuração de TI e, com isso, pode-se identificar o *status* destes elementos, como por exemplo, se um elemento, está em manutenção, alocado por outro usuário, em compra, etc.;
- Estabelecer as relações entre os elementos de configuração e, desta forma, pode-se determinar dependências entre os elementos de configuração, identificando os incidentes, problemas ou alterações relacionadas a esse elemento;
- Identificar *softwares* instalados na infra-estrutura de TI e sua documentação;
- Determinar a convenção de nomes dos elementos de configuração.

Já a atividade de **Controle** do processo de Gerenciamento de Configuração é a atividade que irá administrar todos os elementos de configuração, durante todo o ciclo de vida de um projeto de TI. Com isso, pode-se garantir que somente os elementos autorizados e identificados estarão registrados no banco de dados CMDB. Suas principais atividades realizadas são:

- Registrar novos elementos de configuração e suas versões;
- Atualizar novos registros existentes, quando houver alguma alteração de versão ou solicitação de alteração destes elementos;
- Controlar e manter licenças, verificando o *software* em uso dentro da empresa, bem como sua documentação, dados, licenças e contratos de manutenção;
- Atualizar o CMDB sempre que forem detectadas diferenças entre os registros cadastrados no CMDB e os CIs identificados.

Na atividade de **Estatísticas de Status de Configuração**, deverão ser gerados regularmente relatórios de *status* de todos os elementos de configuração, mostrando suas versões e histórico de mudanças. Esses relatórios permitem acompanhar a evolução de cada um dos elementos de configuração comparando a sua configuração base com a sua configuração atual. A seguir alguns itens que podem ser exibidos por esses relatórios:

- Configurações base, versões e *status* de todos os elementos de configuração;
- Versões finais dos *softwares*;
- Pessoa responsável pela a modificação do *status* de cada um dos elementos de configuração;
- Histórico de modificações para possíveis auditorias;
- Incidentes ou problemas em aberto para cada um dos elementos de configuração.

A atividade de **Verificação** e **Auditoria** permite, antes que seja realizada qualquer alteração no ambiente de produção, verificações que podem ser necessárias em uma auditoria para garantir a compatibilidade com os dados armazenados no banco de dados CMDB. Principais atividades relacionadas a essa etapa:

- Realizar auditoria após as alterações no ambiente de produção;
- Comparar os dados do CMDB com a infra-estrutura física para identificar incompatibilidades;
- Executar uma auditoria como resposta à identificação de um CI não autorizado;
- Verificar se o processo de Gerenciamento de Alterações está sendo realizado para a atualização do CMDB.

Rational ClearCase

A ferramenta *Rational ClearCase* oferece um gerenciamento de configuração completo, pois permite realizar um controle efetivo de versões e realizar auditorias, visando melhorar a produtividade no processo de desenvolvimento de *software*. Essa ferramenta pode ser integrada com as principais IDEs líderes de mercado, como o *Eclipse*, e suporta os sistemas operacionais AIX, HP Unix, Linux e Windows.

O *Rational ClearCase* permite à equipe de desenvolvimento identificar todos os itens de configuração de um projeto de

TI, bem como suas funcionalidades e interfaces. O controle permanente da evolução dessas funcionalidades e interfaces permite que a integração entre esses itens de configuração sejam acompanhados continuamente, e através disso, permitir que mudanças ocorridas no projeto sejam devidamente gerenciadas e documentadas. Permite também a auditoria dos elementos identificados, documentados e controlados, garantindo assim, a confiabilidade dos sistemas e melhor produtividade para a equipe de desenvolvimento.

A utilização desta ferramenta em projetos de TI facilita a criação de um ambiente onde é possível organizar e armazenar os itens de configuração através da criação de um modelo de implementação que irá definir como deverão ser organizados e armazenados os itens de configuração do projeto.

A IBM disponibiliza gratuitamente para avaliação uma versão *Trial On-Line* da ferramenta *Rational ClearCase*, que se encontra disponível no *link http://www.ibm.com/developerworks/downloads/r/rcc/*. Esta versão *Trial* da *Rational ClearCase* disponibiliza um projeto completo com vários exemplos e usuários já pré-cadastrados. Para abrir a versão *Trial On-Line* da *Rational ClearQuest*, primeiramente deve-se acessar o *link* informado anteriormente e fazer o *login* utilizando o usuário *default* do projeto de exemplo. Para isso deve-se informar o usuário "alex" para o campo *User Name* e em *Password* a senha "alex", para prosseguir (ver **Figura 3**).

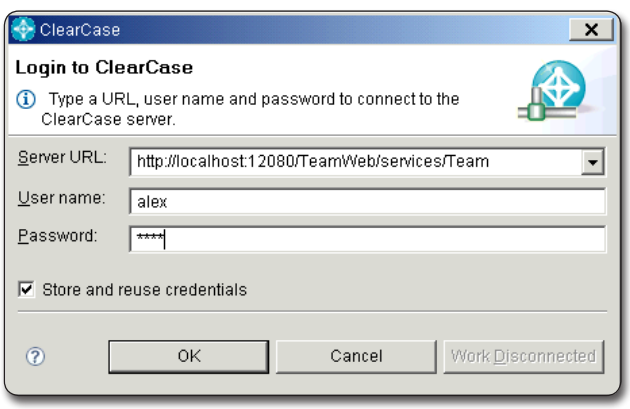


Figura 3. Tela de login da ferramenta ClearCase

Esta versão de avaliação do *Rational ClearCase* já possui um repositório de dados para que se possam armazenar as informações referentes a itens de configuração de um projeto. No *Rational ClearCase* os VOBs (*Versioned Object Bases*) irão armazenar versões de arquivos, diretórios e outros objetos. Eles atuam como um repositório de informações para o gerenciamento de configuração.

Para criar uma *View*, ou seja, visões onde os desenvolvedores poderão implementar e testar códigos, além de gerenciar os artefatos gerados pelo projeto, deve-se utilizar o *ClearCase Explorer* selecionando o *menu ClearCase* > *Create View* e, na tela *Create View*, selecionar a opção *Create a view on an UCM stream*, explorar a opção *Project VOBs* e selecionar o item "Alex_CLSICS_CD" e, por fim, clicar no botão *Finish* (ver **Figura 4**).

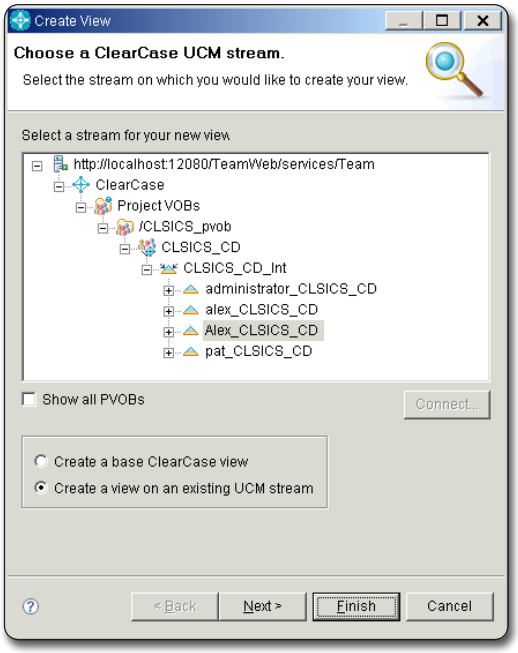


Figura 4. Visualização da tela Create View

Através desta *View* criada anteriormente, é possível visualizar seus artefatos, e a partir desse ponto, realizar os recursos de *Check-in* e *Check-out* sobre eles, realizando um controle de versões (ver **Figura 5**). Esse controle de versões busca dar suporte ao processo de gerenciamento de várias versões de artefatos que irão surgir ao longo de todo o ciclo de vida de desenvolvimento de um projeto de TI, como por exemplo, código-fonte e documentação.

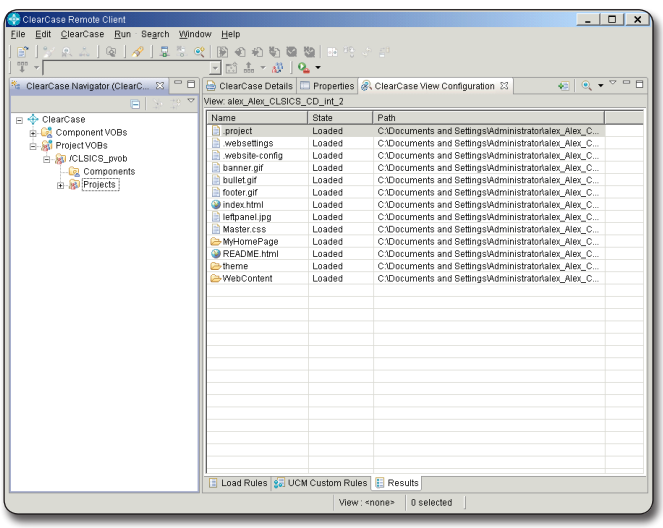


Figura 5. Visualização da aba ClearCase View Configuration

A ferramenta também disponibiliza opções interessantes visando facilitar o gerenciamento dos artefatos pertencentes ao projeto, como o *Compare with Predecessor* que tem a funcionalidade de comparar versões dos artefatos, onde essa operação de comparação é realizada comparando a versão atual do arquivo com a sua versão anterior e, com isso, é possível identificar

modificações realizadas em um artefato, como por exemplo, a adição de uma nova linha de código. Através da opção *Show Version Tree* é possível realizar o rastreamento de modificações ocorridas nos artefatos do projeto, permitindo compreender rapidamente o impacto de alterações e conseqüências geradas por modificações (ver **Figura 6**). Já no *Show History* é possível consultar toda a evolução de um artefato, obtendo dados como datas, usuários, versões, eventos e comentários.

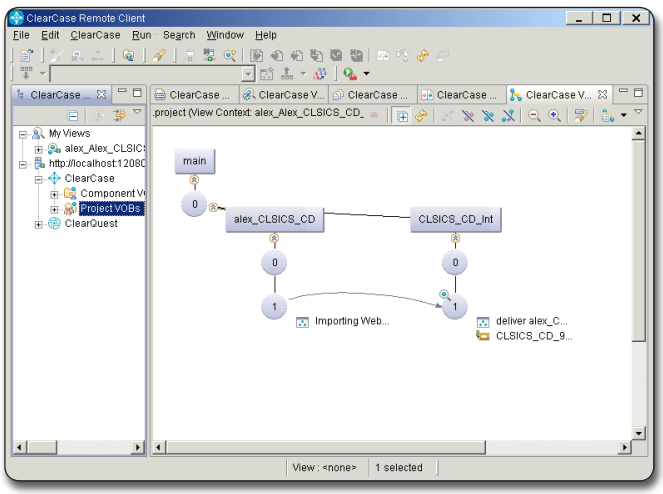


Figura 6. Visualização do Show Version Tree de um artefato

Conclusões

Vimos neste artigo como o uso de metodologias aliadas às melhores práticas existentes no mercado podem ser úteis durante o desenvolvimento de projetos de TI.

Também se pôde mostrar que a ITIL, através da utilização do processo de Gerenciamento de Configuração, proporciona um controle efetivo de diferentes versões de artefatos, gerenciando e auditando mudanças que podem vir a ocorrer durante o ciclo de vida de um projeto, e de como o uso da ferramenta *Rational ClearCase* pode ajudar neste processo. ●

Referências

MAGALHÃES, Ivan Luizio; PINHEIRO, Walfrido Brito. Gerenciamento de Serviços de TI na Prática. São Paulo: Novatec, 2007.

PRESSMAN, R. S. Engenharia de Software. São Paulo: Makron Books, 2004

VARGAS, Ricardo Viana. Manual Prático do Plano de Projetos. 3 ed. Rio de Janeiro: Brasport, 2007.

TECHNET, Microsoft. Material do curso Academia de Gerenciamento. Disponível em: <http://www.microsoft.com/brasil/technet/academia/default.aspx>. Acesso em: 20 de agosto. 2009.

GONÇALO, João Vitorino de Jesus. ITIL: Valerá a pena? Quais os processos mais afectados?. Disponível em: <http://www.student.dei.uc.pt/~gjesus/CSI/Trabalhos/ITIL.pdf>. Acesso em: 18 de agosto. 2009.

IBM. Help On-Line. Disponível em: http://publib.boulder.ibm.com/infocenter/cchelp/v7r0m1/index.jsp?topic=/com.ibm.rational.clearcase.tutorial.bcc/topics/a_bcc_welcome.htm.Acesso em: 22 de agosto. 2009.

Chegou o Sistema de Créditos DevMedia

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

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

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

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

Saiba mais sobre o Sistema de Créditos!

DevMedia GROUP

www.devmedia.com.br/creditos

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

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

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

PÓS-GRADUAÇÃO

- **Engenharia de Software: Desenvolvimento Java**

GRADUAÇÃO

- **Análise e Desenvolvimento de Sistemas**

FORMAÇÕES

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

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

TURMAS NO RIO DE JANEIRO

infnet INSTITUTO

ESCOLA SUPERIOR DA TECNOLOGIA DA INFORMAÇÃO

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

EDUCAÇÃO SUPERIOR ORIENTADA AO MERCADO