Em Direção a um Modelo de Custos de Desenvolvimento de Software Orientado a Objetos

Cláudia Dib Cruz[1,2]
cdib@br.ibm.com

Cláudia Maria Lima Werner[1]
werner@cos.ufrj.br

Jeferson Ferreira Soares[2]
Jeferson@br.ibm.com

[1]COPPE - Programa de Engenharia de Sistemas e Computação
Universidade Federal do Rio de Janeiro
Caixa Postal 68511 - CEP. 21945-970
Rio de Janeiro, RJ

[2]IBM Brasil
Av. Pasteur, 138 Botafogo
Rio de Janeiro, RJ – CEP. 22290-900

Resumo

Este artigo apresenta um estudo preparatório à elaboração de um modelo de custos de desenvolvimento de software orientado a objetos. O objetivo deste estudo é discutir aspectos de custos relacionados ao paradigma de Orientação a Objetos, e a partir de métricas propostas na literatura, selecionar variáveis que possam ser utilizadas num modelo de custos.

Neste artigo, são descritos os elementos que devem compor o modelo de custos, considerando a literatura técnica sobre o assunto e projetos reais utilizados como base de estudo deste trabalho.

1. Introdução

Quando os computadores começaram a ser utilizados, em escala comercial, os custos de software representavam menos de 20% dos custos totais dos sistemas de computação. Entretanto, esta relação foi se alterando ao longo do tempo, com os custos de software tendo uma maior participação nos custos totais. Isso explica o crescente interesse, por parte das organizações, na medição e controle dos custos dos projetos de desenvolvimento de software.

As estimativas de custos de projetos de desenvolvimento de software, regularmente, são subestimadas e, por isso, em muitas organizações, não possuem credibilidade.

Para os executivos de negócio, este é um dos aspectos atuais mais frustrantes do desenvolvimento de software. Isso resulta em oportunidades perdidas e insatisfação de clientes. Assim sendo, é importante que um produto de software seja desenvolvido com os recursos e cronograma previstos. O sucesso financeiro de um projeto depende da habilidade do gerente em estimar seus custos, no início de seu desenvolvimento, e controlá-los, ao longo do processo de desenvolvimento.

Atribui-se ao aumento do tamanho e da complexidade dos novos produtos, ao caráter dinâmico e iterativo do processo de desenvolvimento, à falta de procedimentos padronizados de avaliação de qualidade, e à falta de métodos e ferramentas de desenvolvimento adequadas,

a baixa produtividade nos projetos de desenvolvimento de software [1]. Neste sentido, os gerentes de projeto estão, cada vez mais, sentindo a necessidade de um maior controle no processo de desenvolvimento de software [11], aumentando a demanda por medidas de software, ou métricas para a gerência do processo.

A importância destas métricas deve-se à limitação de recursos financeiros para suprir a demanda de novos produtos de software. Isto torna o cálculo de custo-benefício uma prioridade para a maioria das organizações que, através de uma análise dos custos de desenvolvimento dos produtos propostos e dos benefícios que podem gerar, selecionam e estabelecem prioridades para construção.

Entretanto, por falta de teorias a respeito do comportamento das variáveis que determinam os custos de um projeto, os modelos de custos mais conhecidos se baseiam em experiências passadas para projetar o tamanho do produto e, a partir daí, projetar os custos de seu desenvolvimento. Estes modelos estabelecem, através de métodos estatísticos, relações funcionais entre as variáveis de tamanho e o esforço de desenvolvimento.

Contudo, visto que a produção de software é realizada sob encomenda (i.e. solução particular), as condições e as circunstâncias de desenvolvimento dos diferentes produtos variam. Para se produzir estimativas precisas, engenheiros de software desenvolveram técnicas para capturar a relação entre esforço e as características da equipe e do projeto e outros fatores que afetam o prazo, esforço e custo. Daí ser importante a inclusão de variáveis, nestes modelos, que representem características de projetos que influenciam na produtividade das equipes de desenvolvimento. Como exemplo, o modelo "Constructive Cost Model" (COCOMO) apresentado em [2], considera, além da variável de tamanho quantidade de linhas de código produzidas, outras variáveis de projeto, entre as quais experiência e capacidade da equipe de desenvolvimento.

A maior dificuldade surge, entretanto, quando são utilizadas novas tecnologias no desenvolvimento de um produto de software. Isto porque a falta de experiência impossibilita o uso de métodos estatísticos, para encontrar as novas variáveis de custos dos modelos. Com isso, são utilizados modelos tradicionais que, muitas vezes, são compostos de variáveis que não representam as características mais importantes para a determinação dos custos dos novos projetos desenvolvidos. Este é o caso da Orientação a Objetos (OO), que vem alterando, sensivelmente, a maneira de se desenvolver software e se tornando cada vez mais popular.

Apesar do ainda reduzido número de publicações que relatem experiências nesta área, o processo de desenvolvimento de software orientado a objetos vem apresentando singularidades que indicam que as métricas/métodos tradicionais de tamanho não devem ser adotados [1]. Há ainda uma grande demanda no mercado por parâmetros de custos específicos para a OO, principalmente para comprovar se as expectativas geradas quando do surgimento desta tecnologia, já podem ser comprovadas, empiricamente.

Este artigo apresenta um estudo preparatório à elaboração de um modelo de custos de desenvolvimento de software orientado a objetos. O objetivo deste estudo é discutir alguns aspectos de custos relacionados ao paradigma de OO e, a partir de métricas propostas na literatura, selecionar variáveis que possam ser utilizadas num modelo. O estudo mostrou-se necessário quando tentou-se aplicar alguns métodos encontrados na literatura para estimar custos de três projetos de desenvolvimento de software orientado a objetos, e os resultados encontrados não foram satisfatórios (ver seção 2). Não é objetivo deste artigo apresentar um modelo de custos com seus coeficientes calculados, visto que a insuficiência de dados históricos de projetos ainda não permite o cálculo dos coeficientes com a precisão desejada.

Na seção 3, apresentamos uma descrição sobre os elementos necessários a um modelo de custos para o desenvolvimento de projetos de software orientado a objetos.

2. Base de Estudo

Nesta seção apresentamos os resultados obtidos em três experiências de medição de custos em projetos de desenvolvimento de software, utilizando-se métricas e métodos propostos na literatura, especificamente para a Tecnologia de OO.

Inicialmente, são apresentadas as características dos projetos utilizados como base de estudo e discutidos os resultados obtidos de tamanho e produtividade. Em seguida, é justificada a necessidade de utilização de variáveis para o ajuste de resultados, levando-se em consideração os aspectos ambientais e características das equipes dos projetos, conforme sugerido no modelo COCOMO II [3], através dos fatores de ajuste.

2.1 Características dos Projetos

No projeto 1, foi desenvolvido um sistema de aplicação para o planejamento corporativo da rede básica de uma empresa de telecomunicações. Foram adotados a linguagem de programação C++, a ferramenta para construção de interface gráfica UIM da IBM Corp, a plataforma de estações de trabalho IBM-RISC 6000, com sistema operacional AIX da IBM Corp, e o método Coad-Yourdon [5] de análise para desenvolvimento orientado a objetos. A equipe apresentou, em média, 8 integrantes.

No projeto 2, foi desenvolvido um sistema de aplicação para o controle operacional de transmissões de dados via satélite, vital para a empresa, em substituição a alguns sistemas que operavam de maneira não integrada. Foi adotada a linguagem C++, com um banco de dados orientado a objetos, a plataforma utilizada foi a de estações de trabalho IBM-RISC 6000, com sistema operacional IBM-AIX. Foi adotado o método Coad-Yourdon e a ferramenta para construção da interface gráfica foi a IlogView da Ilog. A equipe apresentou 7 integrantes.

No projeto 3, foi desenvolvido um sistema de aplicação para planejamento e controle da produção, contemplando atividades desde a transformação de um pedido em uma especificação de processo para fabricação até o controle do processo como um todo. Foi utilizada a linguagem VisualAge/Smalltalk para desenvolvimento, construção da interface gráfica e prototipagem, ferramentas Smartsuite da Lotus para edição, o método OMT [10] para modelagem e projeto de classes e a ferramenta Select OMT, da Select Software Tools, para especificação do modelo. Foi utilizado um banco de dados orientado a objetos, plataforma de microcomputador 486, com sistema operacional Windows/NT. A equipe foi composta, em média, por 35 integrantes.

Foram desenvolvidos nos três projetos, sistemas corporativos, de grande importância para as organizações. Considerando os critérios de avaliação de complexidade do modelo COCOMO II, os sistemas podem ser vistos com complexidade média, embora no projeto 3 tenham sido desenvolvidas algumas rotinas matemáticas otimizadas.

2.2 Resultados de Tamanho e Produtividade

As classes foram categorizadas em *classes de domínio*, *classes de interface* com o usuário, *classes de dados* e *classes auxiliares*, conforme definidas por Haynes *et al.* [6], para serem contabilizadas nos projetos. Embora alguns autores [6] [8] estabeleçam taxas fixas de classes de dados, de interface e auxiliares por classes de domínio, na medição dos projetos, as taxas foram calculadas, para cada tipo de classe, para que durante a avaliação dos resultados obtidos, as diferenças de gerenciamento de dados, de interface com o usuário e outras

características sejam consideradas, resultando numa melhor avaliação das medidas encontradas.

A tabela 1 apresenta os resultados das medições de tamanho obtidos.

Os projetos 2 e 3 utilizaram um banco de dados orientado a objetos, não necessitando portanto de classes adicionais para realizar a gerência de dados. No projeto 1, foi implementado um gerenciador de objetos para garantir a persistência dos objetos.

	Número de Classes de Domínio	Número de classes de interface	Número de classes de dados	Número de classes auxiliares	Número total de classes de apoio	Relação de classes de apoio por domínio
Projeto 1	71	104	73	83	260	3,67
Projeto 2	88	226	0	13	239	2,72
Projeto 3	189	372	0	67	439	2,32

Tabela 1. Quantidade de classes por projeto

Os projetos 2 e 3 foram construídos com ferramentas de construção gráfica de mais alto nível, quando comparadas com a ferramenta utilizada pelo projeto 1, resultando em uma proporção de classes de interface com o usuário construídas para cada classe de domínio de, aproximadamente, 2 para 1, nesses projetos.

Em [6], sugere-se uma relação estável de 1 classe de interface com o usuário para cada classe de domínio, quase metade da relação encontrada nos projetos 2 e 3, e bem inferior a relação encontrada no projeto 1.

Do ponto de vista estatístico, Haynes *et al.* [6], não oferecem maiores informações a respeito da dispersão das observações utilizadas em seu trabalho em torno da média encontrada. Isso impede que se construa um intervalo de confiança que permita avaliar se as médias dos três projetos estudados são, estatisticamente, diferentes da média proposta. Porém, admitindo-se, hipoteticamente, que a relação de classes de interface por classe de domínio se distribua *normalmente*, se o desvio-padrão for suficientemente grande para incluir no intervalo de confiança resultados tão dispersos, cabe questionar a validade deste método para projetar o número de classes de um projeto.

Por outro lado, como é de se esperar, se o desvio-padrão da amostra utilizada em [6] for pequeno, isto é, grande parte das observações se distribua próxima a média, os resultados encontrados revelam discrepâncias significativas que podem indicar que devam ser consideradas outras variáveis para o cálculo da relação em questão.

Em suma, em ambos os casos, tendo em vista os resultados apresentados nestes projetos, há fortes indícios de que há singularidades nos projetos que não estão sendo consideradas na relação proposta em [6].

Em Lorenz e Kidd [8], há um tratamento mais criterioso em relação à complexidade de construção da interface. Os autores afirmam que o tipo de interface com o usuário é determinante da quantidade total de classes de apoio, e que projetos *GUI*‘s tendem a apresentar duas vezes mais classes de apoio do que os projetos sem interface com o usuário. Os autores optaram por obter relações funcionais entre as *classes-chave* (ou de domínio) e as classes de apoio, discriminando os projetos por complexidade da interface com o usuário.

Contudo, nas experiências aqui relatadas, a complexidade dos três projetos foi avaliada na categoria *interface gráfica simples*, de acordo com os critérios Lorenz e Kidd. Contudo, é fato que, quanto mais recursos gráficos são utilizados para a construção da interface com o usuário, mais classes são definidas para a sua implementação.

Por outro lado, o tipo de base de dados se mostrou um aspecto importante para determinar a quantidade de classes de apoio nos projetos relatados. Pode ser que, com uma

quantidade maior de dados, os autores não tenham encontrado justificativas estatísticas para a utilização deste fator. Contudo, a proporção encontrada no projeto 1 foi similar, de 1 classe de dados para cada classe de domínio.

No projeto 1, a duração das etapas de levantamento de requisitos, especificação e modelagem do domínio foi de 5,5 meses. Estiveram dedicados ao projeto: 2 desenvolvedores, em regime parcial, e 2 desenvolvedores, em regime integral. Considerando que em regime integral, um desenvolvedor dedique 160 horas de trabalho por mês, o esforço total de desenvolvedores foi de 2640 pessoas/hora, nesta fase.

A implementação do projeto, incluindo as fases de projeto, código, testes e aceitação, teve a duração de 30 meses. Estiveram dedicados: 2 desenvolvedores, em regime parcial, e 4 desenvolvedores, em regime integral, sendo que, no último mês, apenas um dos dois analistas em tempo integral permaneceu no projeto. Portanto, o esforço total de desenvolvedores para implementação do projeto foi de 23840 horas. No total, o esforço de desenvolvimento foi de 26480 horas, ou 165,5 pessoas/mês, no projeto 1.

No projeto 2, as etapas de levantamento de requisitos, especificação e modelagem do domínio duraram 4 meses, com 4 desenvolvedores dedicados, parcialmente, e 2 analistas dedicados, integralmente, com um gasto de 1920 horas. Na implementação, estiveram dedicados 7 desenvolvedores, em tempo parcial, durante 32 meses e consumiu 17920 horas. No total, foram gastas 124 pessoas/mês, para o desenvolvimento do projeto 2.

No projeto 3, foram gastas 9.280 horas de trabalho, ou 58 pessoas/mês, nas fases de levantamento de requisitos e especificação do domínio. Nas fases de projeto e codificação, foram gastas 21.943 horas de desenvolvimento, ou 137 pessoas/mês[1]. No total, foram consumidas 31.223 horas, ou 195 pessoas/mês.

A tabela 2 apresentada as quantidades de esforço de cada projeto.

	Fase de especificação do domínio	Fase de implementação do produto	Total	Produtividade
Projeto 1	16,5	149	165,5	2
Projeto 2	12	110	124	2,63
Projeto 3	58	137	195	3,22

Tabela 2. Esforço gasto nos projetos pessoas/mês

De acordo com Lorenz e Kidd [8], o esforço gasto para o desenvolvimento de uma classe seria de 15 a 20 homens/dia, considerando critérios de percentual de classes reutilizadas e a capacidade e a experiência da equipe de desenvolvimento nas ferramentas e plataforma do projeto. As avaliações foram realizadas com base em comparações, visto que os autores não fornecem critérios quantitativos para a determinação da taxa de esforço, como visto a seguir.

Os três projetos aqui estudados representam projetos pilotos em tecnologia de OO, nas organizações onde foram desenvolvidos, e não houve reutilização. Sendo assim, as taxas de produtividade dos projetos não podem ser diferenciadas por este fator.

Utilizando os critérios de avaliação do fator de experiência da equipe do modelo COCOMO (i.e., experiência na linguagem de programação, nas ferramentas de software e nos métodos utilizados), os três projetos podem ser avaliados da seguinte forma.

No projeto 1, boa parte da equipe de desenvolvimento conhecia a linguagem C++ e o método de Coad e Yourdon [5]. Entretanto, metade da equipe não conhecia OO e nenhum membro da equipe conhecia a ferramenta UIM.

[1] Estes resultados foram obtidos diretamente de um sistema de relatório de horas de trabalho.

Também no projeto 2, a maior parte da equipe de desenvolvimento conhecia C++ e o método de Coad-Yourdon, mas nenhum membro da equipe conhecia IlogView. Além disso, foi utilizado um banco de dados orientado a objetos.

No projeto 3, alguns desenvolvedores tinham experiência com a Tecnologia de OO, mas a maioria adquiriu durante o projeto. Os projetistas do sistema conheciam o método OMT [10] e os desenvolvedores foram treinados em VisualAge/Smalltalk. Foi utilizado, pela primeira vez na empresa, um banco de dados orientado a objetos.

Sendo assim, considerando-se, isoladamente, o aspecto de experiência nas ferramentas e linguagem, pode-se afirmar que as equipes tinham experiências similares, uma vez que, os projetistas dos produtos já tinham adquirido alguma experiência em desenvolvimento orientado a objetos e os desenvolvedores adquiriram conhecimento na ferramenta/linguagem de construção do aplicativo, durante os projetos.

Em suma, a combinação de fatores, experiência de equipes e reutilização, não explicam as diferentes taxas de produtividade encontradas nos projetos, embora estes fatores devam ser considerados na projeção de custos de projetos, visto que é esperado que equipes mais experientes tenham maior nível de produtividade e que, a reutilização de componentes de software reduza o esforço gasto.

A seguir, são analisados os resultados de produtividade, com base na proposta apresentada por Haynes *et al.* [6]. A linguagem de programação e o tamanho da equipe foram combinadas num único fator para a seleção da taxa de produtividade.

Em relação à linguagem, nas experiências de Haynes *et al.* e Lorenz e Kidd [8], os autores afirmam que a linguagem Smalltalk é mais produtiva que a linguagem C++. De acordo com Lorenz e Kidd, existem três hipóteses para explicar este fato: desenvolvedores quando utilizam C++ tendem a desenvolver classes maiores; classes em C++ são mais difíceis de serem reutilizadas, resultando em desenvolvimento desnecessário de novas classes; e C++, por ser uma linguagem híbrida, permite o desenvolvimento de código não orientado a objetos.

Em relação ao tamanho do projeto, os modelos mais tradicionais, como o COCOMO [2], consideram este fator no cálculo do esforço de desenvolvimento de um projeto. Segundo Boehm, a produtividade não se mantém constante a medida que a equipe cresce, uma vez que a coesão e a comunicação entre os membros se torna mais difícil e o gerenciamento do projeto mais complexo. Neste modelo, os projetos são fatorados em três categorias de desenvolvimento, e o que distingue estas categorias é o aspecto ambiental.

Em relação a ferramenta de construção gráfica, os projetos 2 e 3 podem ter sido beneficiados, uma vez que os desenvolvedores destes projetos consideraram esta ferramenta simples, o que não ocorreu no projeto 1, onde a ferramenta adotada foi criticada pela equipe.

Em [6], as taxas de produtividade de pequenos projetos desenvolvidos em Smalltalk, 11 classes por pessoa/mês, é bem superior à taxa de produtividade para grandes projetos em Smalltalk, que é de 4 classes por pessoa/mês.

Além disso, mesmo o projeto 3 tendo sido maior que os projetos 1 e 2, apresentou produtividade mais alta. O fator linguagem/construtor gráfico de desenvolvimento possivelmente contribuiu para justificar esta diferença.

Em relação às diferenças de taxas de produtividade apresentadas pelos autores e os resultados dos projetos, a hipótese mais provável é que, como as equipes dos projetos não eram experientes, havendo aquisição de conhecimento ao longo do projeto, e os autores não fornecem qualquer referência a respeito da experiência das equipes da amostra, é provável que esta contenha projetos com equipes experientes e inexperientes e que, por isso, as taxas obtidas pelos autores tenham sido mais elevadas.

Concluindo, existem fatores, além dos considerados por Lorenz e Kidd e Haynes *et al.*, que possivelmente ajudariam a explicar as diferenças de produtividade apresentadas pelos projetos de software, e que devem ser utilizados para aumentar a precisão dos resultados.

2.3. Uso de Fatores de Ajuste

A etapa de projeção inicial de custos é fundamental para a decisão de se construir ou não um produto de software. Entretanto, devido à escassez de informações no início de um projeto, os seus resultados não são precisos neste momento e precisam ser reavaliados à medida que novas informações são obtidas. Porém, algumas características de produto e projeto, obtidas nesta etapa, ajudam a aumentar a precisão dos resultados obtidos.

A falta de teorias a respeito do comportamento das variáveis, para a determinação de custos de software, dificulta a afirmação de qualquer hipótese que não seja por vias empíricas. Sendo assim, é necessário a seleção de um conjunto de variáveis que, supostamente, têm maior influência nos custos de um projeto.

Neste trabalho, propomos que as variáveis de custos sejam selecionadas do modelo COCOMO II, que é amplamente difundido e aceito. Este modelo indica 17 fatores de ajuste, combinados em 7 fatores, quando utilizados no início da fase de projeto.

Os critérios do modelo COCOMO II, para tratamento da complexidade são dados em termos de operações de controle (i.e., complexidade de código), operações computacionais, operações dependentes de dispositivos, operações de gerência de dados e operações de gerência de interface com o usuário.

No contexto OO, outros aspectos necessitam ser avaliados, como profundidade da hierarquia de classes, número de filhos, número de métodos e número de mensagens.

Em relação a fatores como capacidade e experiência da equipe, com o aumento da utilização de pacotes na construção de novos produtos, há uma crescente importância na habilidade dos desenvolvedores em lidar com estes pacotes, refletida nas taxas de produtividade dos projetos. Se por um lado a construção da interface entre módulos, na OO, é menos complexa e os novos geradores de aplicativos tendem a exigir menos conhecimento de programação, por outro lado, cada vez mais, produtos são construídos a partir da composição de módulos em diferentes linguagens, ferramentas e plataformas, exigindo maior habilidade e experiência para lidar com esta diversidade tecnológica.

Em [6], os autores afirmam que encontraram grandes diferenças de produtividade entre os membros de uma mesma equipe, de projetos de software orientado a objetos, chegando a uma relação de 7/1. Os autores aconselham ajustar a produtividade de uma organização, individualmente, de acordo com a capacidade de cada desenvolvedor, para calibrar os resultados dos cálculos de estimativas. Além disso, boas práticas de programação orientada a objetos garantem que os objetivos de tamanho sejam alcançados.

Outro fator influente, na produtividade da equipe, é a continuidade de pessoal. Uma pessoa que integra uma equipe de um projeto, em andamento, precisa conhecer o ambiente de desenvolvimento, modelos de objetos, especificações e outros componentes já produzidos. A tendência é que, quanto maior a rotatividade de uma equipe, menor a sua produtividade.

A utilização de ferramentas de desenvolvimento também é um fator importante a ser avaliado. Plataforma e ferramentas mais simples de serem utilizadas, aumentam a produtividade.

Embora cada fator tenha uma justificativa particular para ser utilizado, é certo que é difícil avaliar os seus efeitos, isoladamente. Sendo assim, é necessária a seleção de um método estatístico que relacione todos estes fatores para a obtenção dos seus coeficientes.

3. Composição de um Modelo de Custos

Os custos de um projeto normalmente são calculados com base nas medidas de tamanho e complexidade do produto e nos fatores e características ambientais do projeto, denominadas *variáveis* do modelo de custos.

Quando o comportamento destas variáveis é conhecido, é possível determinar os custos de um projeto com precisão, com base em experiências similares passadas. Métodos estatísticos relacionam o comportamento destas variáveis nos diversos projetos da amostra.

Para se determinar o tamanho de um produto, é necessário conhecer bem o domínio do seu problema e delimitar com precisão o seu escopo, o que acontece à medida em que a equipe de desenvolvimento e os analistas de negócios interagem. Sendo assim, é necessária a reavaliação da projeção inicial de custos ao longo do seu desenvolvimento.

Para se compor um modelo de custos, quatro aspectos são considerados essenciais:

- determinar as etapas em que as projeções e as medições de custos devem ser realizadas;
- selecionar variáveis de tamanho e complexidade que lidem com as características específicas da OO;
- selecionar variáveis para ajuste dos resultados obtidos às características e condições da organização desenvolvedora do projeto estimado;
- selecionar um método estatístico para cálculo dos coeficientes das variáveis selecionadas que considere as relações estabelecidas entre elas.

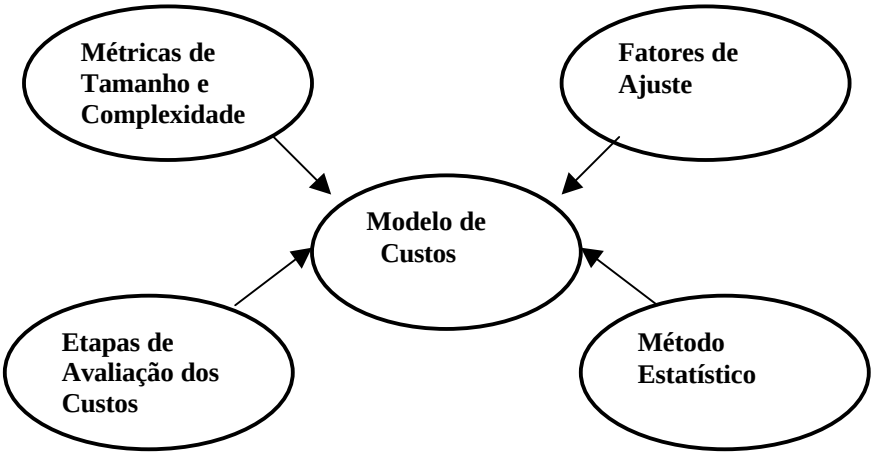

Figura 1 - Componentes de um modelo de custos

Inicialmente, é apresentada uma síntese do método estatístico de mínimos quadrados, baseada na descrição de Wonnacott e Wonnacott [12]. Este método é sugerido para o cálculo de coeficientes das variáveis de tamanho e de esforço propostas para o modelo de custos específico para a tecnologia de OO. Estes parâmetros são obtidos através do estabelecimento de relações supostamente estáveis entre a quantidade de classes-chave do domínio e o tamanho da aplicação, e o tamanho da aplicação e o esforço gasto para sua construção. Com o propósito de desenvolvimento futuro de um método para se estimar custos de desenvolvimento de software, a idéia é que se possa obter a tendência de gasto de mão-de-obra a partir do comportamento e relacionamento das variáveis propostas.

Nas subseções que se seguem, apresentamos as etapas em que as projeções e as medições de custos devem ser realizadas, as variáveis de tamanho, as variáveis e as equações

para projeção de tamanho, e as variáveis e as equações para a projeção de esforço propostas para a composição do modelo de custos.

3.1. Método Estatístico

A técnica de regressão múltipla[2] é usada para descrever como uma variável dependente se relaciona a duas ou mais variáveis independentes, também chamadas de *regressores*. Por exemplo, no caso de produtos de software, o tamanho de um produto - variável dependente - pode estar relacionado com a complexidade da interface com o usuário e com o tipo de gerenciador de dados, sendo estas últimas variáveis independentes.

Neste caso, a técnica de regressão deve ser utilizada com o objetivo de descobrir se algumas características do produto e do projeto afetam o tamanho do produto e a produtividade da equipe de desenvolvimento, respectivamente, e de que maneira afetam, através de uma equação relacionando, no primeiro caso, o tamanho do produto com as características do produto e, no segundo caso, a produtividade da equipe com as características e condições de projeto.

Resolver esta equação é equivalente, geometricamente, a ajustar uma curva através de um diagrama de dispersão, ou seja, como um simples modelo matemático, a equação é útil como um modo de prever a variável dependente a partir de valores pré-estabelecidos das variáveis independentes.

Uma das grandes vantagens deste método é que ele trata as variáveis em conjunto, considerando os seus relacionamentos. No caso de custos de desenvolvimento de software, isto é fundamental, uma vez que as variáveis representam fatores de ajuste, ou características do projeto, que se relacionam. Por exemplo, quando se avalia o fator de complexidade de um produto, não se pode deixar de levar em consideração a experiência da equipe, que representa outro fator e, assim, outra variável.

3.2. Etapas propostas para um modelo de custos

A quantidade de informação que se tem a respeito de um projeto determina a precisão dos resultados obtidos no cálculo dos custos de seu desenvolvimento. Daí a importância de se reavaliar as projeções de esforço realizadas, sempre que são obtidos novos dados que aumentem, consideravelmente, a sua precisão.

A abstração é um princípio importante na tecnologia de OO, para lidar com o aspecto da complexidade cognitiva. As funcionalidades do domínio do problema são entendidas e adicionadas em ciclos ao modelo de objetos. Após a modelagem das classes de domínio, são inseridas no modelo as classes para implementação de cada ciclo. Neste sentido, um ciclo de um projeto de software orientado a objetos é composto pelas fases de levantamento e especificação do domínio, definição e especificação da solução e codificação e testes do produto implementado.

No início da etapa de levantamento de requisitos, são abordados os principais objetos que compõem o domínio do problema, com a utilização de técnicas que ajudam a vislumbrar o universo de funcionalidades do usuário, entre elas, cenários de utilização[3]. Todavia, nesta etapa, ainda não são definidas as características destes objetos.

Ao longo da fase de especificação do domínio do problema, outros objetos do domínio são descobertos. As características dos objetos são definidas em termos de métodos,

---

[2] Uma descrição detalhada sobre o método de regressão múltipla pode ser encontrado em [12].

[3] Cenários de utilização é uma tradução do termo *Scenarios Scripts* [8].

funcionalidades e atributos. As interações entre as classes são representadas, através das conexões de mensagens e das associações. No início da fase de projeto, o domínio do problema está satisfatoriamente modelado.

Durante a segunda fase, são levantados os objetos relacionados com o aspecto de implementação do produto. As classes que compõem as partes visuais, o mapeamento de objetos quando utilizado um banco de dados relacional ou gerente de objetos, e outros aspectos de implementação do produto são definidos, além das suas interações com as classes de domínio. No final da fase de projeto, o modelo de objetos está concluído, e já são conhecidos todos os componentes implementáveis do produto.

Com isso, um aspecto importante da OO é que, através do modelo de objetos, é possível segregar as partes referentes ao domínio do problema e as partes referentes à implementação, que podem ter seus tamanhos medidos.

No desenvolvimento de software orientado a objetos, o domínio do problema tende a ser particionado e cada partição pode ser construída, separadamente. Contudo, para cada ciclo de construção, a sequência de fases é a apresentada acima.

Neste sentido, as estimativas de custos podem ser calculadas em três momentos diferentes de cada ciclo de desenvolvimento: após o levantamento inicial dos requisitos do produto, no início da fase de projeto e no início da fase de codificação. Após a conclusão do projeto, é realizada, então, a medição de custos do projeto.

O modelo COCOMO II, tendo em vista o volume de informação disponível a cada etapa do desenvolvimento de um produto, propõe três etapas para se estimar os custos de software: no início do desenvolvimento do produto, no início da fase de projeto, e, após a conclusão da arquitetura.

3.3. Métricas propostas de Tamanho/Complexidade

Um aspecto importante na Tecnologia de OO, para diminuir a complexidade cognitiva, está na forma de representação da informação. O domínio e a solução do problema são modelados com base em classes, ou seja, na primeira fase do desenvolvimento de um projeto, são incluídas classes e detalhes de classes no modelo de domínio desenvolvido na etapa de levantamento de requisitos. Na segunda fase, são incluídas as classes referentes aos aspectos de implementação do produto e, na fase de codificação, as classes são desenvolvidas com a mesma estrutura que são apresentadas no modelo.

Neste sentido, a unidade básica de representação da informação ao longo de todo o processo de desenvolvimento, a classe, também pode ser a unidade básica do processo de projeção de tamanho do produto e, consequentemente, de esforço para o seu desenvolvimento.

Para cada etapa de projeção de custos, métricas de tamanho foram selecionadas, considerando-se as informações quantitativas disponíveis naquele momento.

Isto não quer dizer, entretanto, que os dados levantados a cada fase do projeto não são alterados em fases posteriores. Contudo, os refinamentos sucessivos nos resultados das estimativas se devem, principalmente, às conclusões das fases do desenvolvimento.

3.3.1. Métricas para a primeira etapa. Devido ao alto grau de abstração em que o domínio é tratado, no início de um projeto, momento em que a primeira projeção de custos é realizada, a métrica de tamanho utilizada é a quantidade de classes-chave do domínio do produto, de acordo com a proposta de Lorenz e Kidd [8]. Embora com a utilização desta métrica, não possam ser avaliados os aspectos relacionados à interação entre módulos e o aspecto de numerosidade seja tratado num nível muito abstrato, o número de classes-chave é uma medida

mais estável nesta fase do que outras medidas como, por exemplo, número de métodos, atributos ou mensagens que, sob o ponto de vista da complexidade, seriam medidas mais adequadas para preencher os requisitos de numerosidade. Em suma, a hipótese é que é possível obter relações estáveis entre o número de classes-chave e o número das diferentes classes de apoio, desde que algumas características - fatores de ajuste - do produto sejam consideradas para segregação adequada dos projetos, quando do cálculo dos coeficientes de tamanho.

3.3.2. Métricas para a segunda etapa. Na segunda etapa de projeção de custos, após a modelagem do problema do domínio, já se conhece os atributos, os métodos - que lidam com o aspecto de numerosidade da complexidade - e as mensagens – que lidam com o aspecto da conectividade da complexidade - das classes de domínio. Nesta fase, também, é possível obter a profundidade da hierarquia do modelo do domínio e a quantidade de descendentes imediatos, que são indicadores da complexidade adaptativa do produto. Estas métricas foram selecionadas, principalmente, dos trabalhos de Chimdamber e Kemerer [4] e Li et al. [7].

As características de uma classe e as suas interações com outras classes determinam a quantidade de métodos implementados, que poderá ser então utilizada como métrica de tamanho na segunda etapa de um modelo.

Com base nestas novas informações, projeções de tamanho mais aprimoradas podem ser realizadas, estabelecendo-se relações funcionais entre o número total de métodos da aplicação e as quantidades de métodos privados, atributos e mensagens do modelo do domínio do problema, profundidade e número de descendentes imediatos da hierarquia do modelo de domínio, partindo-se do pressuposto de que, quanto maior o número de detalhes do modelo de objetos, maior a possibilidade de se construir ítens de implementação de interface com o usuário. No caso de utilização de banco de dados relacional ou de construção de um gerente de objetos, a tendência, também, é que sejam construídas classes de dados mais detalhadas.

3.3.3. Métricas para a terceira etapa. Na última etapa de projeção de custos, após a conclusão da arquitetura do projeto, o modelo de objetos já está concluído com os seus métodos, atributos e mensagens de todas as classes definidos, tanto as que se referem ao domínio do problema, quanto as que se referem aos aspectos de implementação. Neste momento, é possível o cálculo de estimativas mais definitivas, uma vez que o tamanho do produto a ser desenvolvido já pode ser calculado com bastante precisão. O esforço de desenvolvimento é calculado com base no número total de métodos, atributos e mensagens, profundidade de herança e número médio de descendentes, como na etapa anterior, incluindo as informações relativas às classes de implementação.

Ao final do projeto, é possível realizar uma medição de seu tamanho.

3.4. Projeção de tamanho

Parâmetros de tamanho e esforço são obtidos a partir de medições realizadas em projetos concluídos, como visto anteriormente. Entretanto, algumas características de projeto não podem ser quantificadas, daí ser necessário tratar estas características como variáveis enumerativas. Embora haja a possibilidade de distorções na avaliação destas variáveis, visto que os seus valores são dependentes do ponto de vista de quem avalia, a outra opção seria não utilizar estas características para ajuste das estimativas, o que é menos recomendável.

3.4.1. Váriáveis de tamanho. Para medição do tamanho do produto, na primeira etapa de realização das estimativas, devem ser consideradas três variáveis independentes enumerativas: complexidade da interface com o usuário, gerência de dados e categoria da ferramenta de construção gráfica. A quarta variável, considerada quantitativa, seria o número inicial de classes de domínio.

A primeira variável, complexidade de interface com o usuário, pode ser valorada de acordo com cinco categorias diferentes, sendo as quatro primeiras, equivalentes às utilizadas no método proposto em [8]. As categorias são: sem interface com o usuário, valor 4; com interface simples, valor 3; com interface gráfica simples, valor 2; com interface gráfica complexa, valor 1; com multimídia, valor 0.

A segunda variável, gerência de dados, pode ser valorada em quatro categorias diferentes: sem banco de dados e sem persistência, valor 3; com banco de dados orientado a objetos, valor 2; com banco de dados relacional, valor 1; sem banco de dados e com persistência (gerência de objetos), valor 0.

A terceira variável, nível da ferramenta de construção, pode ser valorada em quatro categorias: gerador de alto nível, com vasta biblioteca de classes auxiliares de implementação, valor 3; gerador de médio nível e biblioteca com muitas classes de implementação, valor 2; e, gerador de baixo nível e biblioteca com poucas classes de implementação, valor 1; sem biblioteca de classes, valor 0.

Para atribuir valores à variável de tamanho, devem ser consideradas três categorias diferentes: classes de interface com o usuário, classes de base de dados e classe auxiliares.

As classes devem ser contabilizadas, separadamente, por dois motivos: primeiramente, para que se possa obter a influência das diferentes características de produto, em cada categoria de classe e, em segundo lugar, para que o gasto com esforço, na próxima etapa, também seja calculado por categoria de classe.

3.4.2. Equações de tamanho. Na primeira etapa de projeção, o esforço deve ser calculado com base na quantidade total de classes a serem desenvolvidas, como visto anteriormente. As equações apresentadas abaixo fornecem esta quantidade.

São consideradas três equações para futuramente calcular os coeficientes que serão usados para se estimar o tamanho do produto, na primeira etapa. O objetivo destas equações é obter uma projeção do número total de classes a serem desenvolvidas.

(1) $Ci_i = a_1 Ti_i + b_1 Nc_i + c_1 Fc_i$

(2) $Cd_i = a_2 Td_i + b_2 Nc_i$

(3) $Ca_i = c_3 Fc_i + b_3 Nc_i$

onde a, b, e c são os coeficientes do modelo. Ci são as classes de interface, Cd são as classes de base de dados e Ca são as classes auxiliares. Ti é a categoria da interface com o usuário, Td é a categoria de gerência de dados, Fc é a categoria da ferramenta de construção e Nc são as classes de domínio, levantadas inicialmente.

A soma dos resultados das três equações, com o número de classes de domínio, fornece a quantidade total de classes do projeto. O esforço de desenvolvimento nesta primeira etapa é calculado com base nesta quantidade (ver ítem 3.5).

Na segunda etapa de projeção de tamanho, o esforço deve ser calculado com base na quantidade de métodos a serem desenvolvidos, como visto anteriormente. Nesta etapa, é proposta uma única equação:

(4) $Nm_i = a_1 Na_i + b_1 Nd_i + c_1 Ng_i + d_1 Ph_i + e_1 Di_i$,

onde a, b, c, d e e são os coeficientes a serem calculados, Nm é o número total de métodos desenvolvidos, incluindo métodos privados e mensagens, Na é o número total de atributos do modelo do domínio do problema, Nd é o número de métodos privados do modelo de domínio do problema, Ng é o número de mensagens do domínio do problema, Ph é a profundidade da hierarquia, que é fornecida pela quantidade de níveis do modelo do domínio, Di é a quantidade média de descendentes imediatos de uma classe, que é calculada pela fórmula $\Sigma\, i\, n_i$, onde i assume valores de 1 até o número máximo de descendentes imediatos de uma classe do modelo, e n_i é a quantidade de classes do modelo que apresenta o respectivo número de descendentes.

3.5. Projeção de Esforço

As variáveis qualitativas, utilizadas nas equações de esforço para as etapas 1, 2 e 3, foram selecionadas a partir dos fatores de ajuste do modelo COCOMO II.

Nas próximas subseções, são apresentadas as variáveis e equações propostas para o cálculo dos coeficientes do modelo de custos. As equações foram montadas, com base nos resultados de tamanho apresentados pelas equações do ítem 3.4, e com a utilização dos fatores de ajuste do COCOMO II, que se apresentam como variáveis qualitativas.

3.5.1. Variáveis de esforço. Com exceção das variáveis de tamanho, as demais variáveis utilizadas para estimar esforço são enumerativas. As variáveis foram selecionadas a partir dos fatores de ajuste combinados apresentados no modelo COCOMO II [3].

Como visto anteriormente, estas variáveis são utilizadas como fatores de ajuste, ou multiplicadores, no COCOMO II, e devem ser aplicados na segunda etapa do modelo, no início da fase de projeto. Entretanto, estas variáveis podem ser utilizadas na fase inicial do desenvolvimento do produto, quando as primeiras estimativas são realizadas, contudo, de forma simplificada, uma vez que, da maneira que são utilizados no COCOMO II, seriam necessárias informações ainda não disponíveis. Estes fatores e critérios devem ser utilizados nas três etapas de estimativas propostas.

Para cada fator de ajuste, o projeto é categorizado em uma de cinco categorias, no intervalo de [0,4], seguindo a linha adotada pelo COCOMO II, que apresenta as categorias muito baixo, baixo, médio, alto e muito alto.

3.5.2. Equações de esforço. O cálculo de esforço também representa uma função das variáveis de classe - interface, base de dados, auxiliares e domínio -, já que há indícios de que o esforço de desenvolvimento varia, dependendo da quantidade de cada tipo de classe envolvida na construção de um produto de software.

Moser e Nierstrasz [9] sugerem que o aspecto de reutilização de um projeto pode ser capturado, segregando os diferentes tipos de objetos. Partindo destas observações, se as classes forem segregadas por categorias de classe, e se puderem ser obtidas relações estáveis entre a quantidade de classes de domínio e a quantidade de cada uma destas categorias, os resultados das estimativas podem se tornar mais precisos.

Assim, a equação para cálculo dos coeficientes de esforço para a primeira etapa do modelo é a seguinte:

$$\mathbf{(5)}\ Et_i = a_1 Ci_i + b_1 Cd_i + c_1 Ca_i + d_1 Nc_i + e_1 Cf_i + f_1 Dc_i + g_1 Fc_i + h_1 Px_i + i_1 Ac_I + j_1 Ru_I + k_1 Dr_i$$

onde, a, b, c, d, e, f, g, h, i, j e k são os coeficientes calculados, Ci, Cd e Ca foram definidos nas equações (1), (2) e (3), Et é o esforço total de desenvolvimento do projeto, que pode ser representado por qualquer unidade de esforço – pessoas/mês, pessoas/dia, pessoas/hora – desde que os dados de todos os projetos sejam inseridos na mesma unidade, Cf (RCPX no COCOMO) é o resultado da avaliação no projeto do fator de complexidade, confiabilidade, documentação requerida e tamanho da base de dados, Dc (SCED no COCOMO) é do fator de compressão de cronograma, Fc (PCIL no COCOMO) é do fator de facilidade de uso da ferramenta/ linguagem de construção e plataforma, Px (PREX no COCOMO) é do fator de experiência da equipe, Ac (PERS no COCOMO) é do fator de capacidade da equipe, Ru (RUSE no COCOMO) indica se o produto foi desenvolvido para ser reutilizado, e Dr (PDIF no COCOMO) é o fator de disponibilidade de recursos.

Para se estimar esforço nas etapas dois e três, o modelo pode apresentar a equação a seguir:

(6) $\mathbf{Et_i = a_2Nm_i + b_2Cf_i + c_2Dc_i + d_2Fc_i + e_2Px_i + f_2Ac_i + g_2Ru_I + h_2Dr_I}$

onde a, b, c, d, e, f, g, h, i, j e k são os coeficientes das equações, Nm é o número total de métodos projetado, Cf, Dc, Fc, Px, Ac, Ru e Dr já foram definidos na equação (5).

4. Conclusões

Este artigo descreveu um estudo preparatório para a elaboração de um modelo de custos de desenvolvimento de software orientado a objetos, utilizando medidas de tamanho que tratam dos aspectos e características específicos deste novo paradigma tecnológico.

Os modelos de previsão de custos tradicionais partem de métricas de tamanho, como pontos de função e linhas de código, que não consideram elementos essenciais da Tecnologia de OO, como por exemplo, herança, encapsulamento, comunicação entre módulos através de mensagens e, em decorrência, fornecem uma medida a partir da qual não se pode derivar qualquer relação de esforço fidedigna. Visando sanar estas deficiências, alguns estudos pioneiros [6] [8] propuzeram a utilização do número de classes, como medida de tamanho de software orientado a objetos, cuja vantagem é dimensionar o tamanho de um aplicativo de software a partir do modelo de representação de suas informações. Contudo, estas medidas ainda não são utilizadas num contexto de um modelo de custos mais sofisticado. Em consequência, os resultados encontrados a partir desses estudos exigem uma excessiva amplitude das margens de segurança, para que possam ser utilizados como ponto de partida para o planejamento e avaliação do custo-benefício do projeto, tornando-os claramente insuficientes.

Na tentativa de dar mais um passo no sentido de minimizar tal deficiência, nesta trabalho apresentamos o uso das variáveis de ajuste, selecionadas do modelo COCOMO II, para o desenvolvimento de um modelo de custos que se adapte mais facilmente às peculiaridades do ambiente do desenvolvedor.

É importante ressaltar, contudo, que há diferenças do modelo proposto neste trabalho com relação às etapas de avaliação de custo do modelo COCOMO II, que parte de uma outra métrica - Pontos de Objetos - na primeira fase. Na Segunda fase, o modelo utiliza o método de Pontos de Função e a aplicação de 7 fatores de ajuste e, na fase seguinte - a última fase do projeto – é utilizada a métrica de linhas de código, juntamente com 17 fatores de ajuste, aproveitando o maior grau de detalhamento das informações. Nas equações propostas na seção 3, optou-se por aplicar as sete variáveis de ajuste por todas as fases do projeto,

reduzindo-se o grau de detalhamento da análise final, o que representa ganhos em simplificação.

Deve-se chamar a atenção para o fato de que a utilização de fatores de ajuste como variáveis do modelo implica na importação de alguns riscos presentes no modelo COCOMO II, como o fato de se operar com variáveis subjetivas, que são uma fonte de imprecisão do valor estimado de custos.

Óbviamente, o número de projetos estudados, apenas 3, não permite que se teste estatisticamente a relevância empírica das variáveis, o que seria importante, dada a insuficiente cobertura teórica fornecida pela literatura existente que contribua positivamente para o estabelecimento de relações de causa e efeito nas estimativas de custos de projetos convencionais e orientados a objetos.

Por último, neste trabalho, foi utilizado um modelo estatístico, que trata do relacionamento das variáveis de forma linear. Em [3], os autores estabeleceram uma relação linear entre a variável de esforço e os fatores de ajuste. Entretanto, ele estabelece uma relação exponencial entre a variável de esforço e o número de linhas de código. Como o tamanho do projeto no estudo apresentado foi representado pelo relacionamento de diversas variáveis, a opção foi estabelecer, inicialmente, uma relação funcional linear entre estas variáveis e a variável de esforço. Todavia, caso alguma relação se altere, o valor da respectiva variável independente terá que ser inserido na base de dados em forma de logarítmo, para que haja uma relação exponencial entre esta variável e a variável de esforço do modelo.

Com a descoberta de novas variáveis de custos, decorrente da utilização contínua da Tecnologia de OO, estas devem ser incluídas no modelo, com o intuito de aumentar a precisão de seus resultados. Por outro lado, algumas das variáveis selecionadas podem ser excluídas do modelo, desde que o seu coeficiente apresente um valor muito próximo de zero. O valor do coeficiente de cada variável é calculado, com base no relacionamento de todas as variáveis do modelo. Sendo assim, nenhuma variável pode ser eliminada do modelo, sem que apresente o seu coeficiente com valor em torno de zero. Da mesma forma que o modelo não deve ser utilizado, sem que o desvio-padrão da equação apresente um valor que mostre que está havendo uma convergência de valores dos coeficientes das variáveis dos projetos. Quanto menor o desvio-padrão, maior a precisão dos resultados do modelo [12].

O trabalho futuro mais importante que pode ser derivado a partir desse estudo é a elaboração de um modelo de custos de fato, com os coeficientes de suas variáveis calculados. A partir de dados de um número suficientemente grande de projetos, os coeficientes podem ser calculados e substituídos nas equações, formando um modelo de custos de desenvolvimento de software orientado a objetos.

Referências

[1] BASILI, V., BRIANS, L., MELO, W., 1995, *A Validation of Object-Oriented Design Metrics*, Technical Report, Department of Computer Science, University of Maryland, Abril.

[2] BOEHM, B., 1981, *Software Engineering Economics*, Prentice-Hall.

[3] BOEHM, B., CLARK, B., HOROWITZ, E., WESTLAND, C., MADACHY, R., SELBY, R., 1995, *Cost Models for Future Software Life Cycle Processes: COCOMO II*, Annals of Software Engineering.

[4] CHIDAMBER, S. e KEMERER, C., 1994, *A Metrics Suite for Object-Oriented Design*, IEEE Transaction on Software Engineering, 7(5), pp. 510-518.

[5] COAD, P. e YOURDON, E., 1992, *Análise Baseada em Objetos*, Editora Campus.

[6] HAYNES, P., MENEZES, T., PHIPPS, G., 1995, *Using the size of classes and methods as the Basis for Early Effort Prediction; Empirical Observations, initial application*; Practitioners Experience Report, Setembro.

[7] LI, W., HENRY, S., KAFURA, D., SCHULMAN, R., 1995, *Measuring Object-oriented Design*, Journal Object-oriented Programming, Julho-agosto.

[8] LORENZ, M. e KIDD, J., 1994, *Object-Oriented Software Metrics*, Prentice Hall.

[9] MOSER, S. e NIERSTRASZ, O., 1996, *The Effect of Object-Oriented Frameworks on Developer Productivity*, IEEE Transactions on Software Engineering, Setembro.

[10] RUMBAUGH, J., BLAHA, M., PREMERLANI, W., EDDY, F., LORENSEN, W., 1991, *Object-Oriented Modeling and Design*, Prentice-Hall

[11] SELLERS, B. e DUÉ, R., 1995, *The Changing Paradigm for Object Project Management*, Object Magazine, Julho-agosto.

[12] WONNACOTT, R. e WONNACOTT T., 1976, *Econometria*, Livros Técnicos e Científicos Editora.