Teste de Software, Melhoria de Processos e Body Shop

POR MAURICIO AGUIAR

Durante alguns anos, a atividade formal denominada "teste de software" perdeu um pouco de espaço como instrumento de garantia da qualidade, ocorrendo a valorização das atividades preventivas, como é o caso das inspeções e peer-reviews. Atualmente, a evolução das ferramentas para suporte ao teste vêm mudando este quadro, fazendo com que o teste de software volte a despertar o interesse das grandes organizações. Mas, é claro, teste de software não é uma coisa nova.

No início da década de 80 deparei-me, pela primeira vez, com um problema que exigia uma abordagem estruturada para o teste de software. Nossa equipe vinha desenvolvendo, há mais de um ano, um sistema que seria distribuído a 32 clientes, que utilizavam diferentes tipos de mainframe – IBM com MVS, IBM com DOS/VS, Burroughs, etc. Isso foi antes da revolução do IBM-PC, da arquitetura cliente/servidor e, é claro, da Web.

O desenvolvimento do sistema, que iria substituir aquele utilizado por nossos clientes, tinha sofrido os usuais atrasos, decorrentes de estimativas incorretas, mudanças de gerentes, de analistas, dificuldades técnicas, mau gerenciamento de projetos e tudo o mais que, infelizmente, contribui até hoje para que os projetos de TI continuem a desafiar os prazos.

Agora estava chegando o limite de atraso suportado pelos clientes e, finalmente, o sistema teria que ser entregue e implantado. Sabíamos, contudo, que as condições sobre as quais o desenvolvimento havia sido efetuado eram totalmente incompatíveis com o nível de qualidade requerido pelos usuários e pela natureza do sistema. Surgiu a idéia de realizar um teste abrangente do sistema, com a finalidade de identificar e sanar todos os erros que suspeitávamos ocultos sob os milhares de linhas COBOL. Como nossos recursos eram limitados – principalmente o tempo – era preciso algum critério para uma alocação ótima desses mesmos recursos.

Estávamos na época da Revolução Estruturada, quando nossos gurus eram Yourdon, Constantine, DeMarco e Page-Jones, autores dos primeiros livros sobre análise e projetos estruturados. Um outro pesquisador contemporâneo dedicara-se ao teste de software: Glenford Meyers, autor de "The Art of Software Testing" [MEYE79], hoje um clássico que pode ser adquirido nas livrarias da Internet pelo salgado preço de US$ 97. Busquei auxílio na obra de Meyers e produzi uma pequena apostila. Com base nesse texto, nossa equipe operacionalizou um sistema de trabalho que permitiu construir casos de testes, executá-los e identificar algumas centenas de ocorrências. Uma ocorrência é uma incompatibilidade ou diferença entre o resultado de um teste e a correspondente expectativa. Para que o leitor entenda o que significa isso, é bom dar uma idéia do método utilizado nesse trabalho. Os conceitos básicos continuam válidos até os dias de hoje [BEIZ95].

As técnicas de teste podem ser divididas, basicamente, em técnicas denominadas "caixa-preta" e aquelas chamadas "caixa-branca". O teste "caixa-preta", também chamado teste funcional, testa o sistema do ponto de vista do usuário, isto é, não considera a estrutura interna ou a forma de implementação do sistema. Este é o único tipo de teste possível quando não se dispõe do código-fonte, por exemplo. O teste "caixa-branca", por outro lado, procura exercitar todas as partes do código de um sistema. Dessa forma, é necessário que tenhamos a fonte de todos os programas disponíveis, para podermos controlar o que foi e o que não foi testado.

Adotando uma abordagem "top-down", concentramo-nos inicialmente no teste "caixa-preta" (funcional), que parte de uma visão externa do sistema. O conceito-chave até hoje na condução do teste funcional é o de classes de equivalência, introduzido por Meyers em 1979. A idéia das classes de equivalência no teste de software é que, já que não podemos testar todos os casos existentes, vamos dividí-los em classes, de modo que os casos dentro de cada classe sejam "equivalentes", o que nos permitirá testar apenas um subconjunto deles, mantendo a representatividade. O problema passa a ser: como determinar as classes de equivalência?

O teste funcional avalia o comportamento do sistema. Dessa forma, as classes

de equivalência devem ser construídas de modo que agrupem casos de teste para os quais o comportamento do sistema seja o mesmo. Na prática, precisaremos selecionar atributos das transações do sistema, que sejam determinantes para o comportamento esperado. Em um sistema bancário de contas-correntes, um atributo determinante para uma transação de depósito poderá ser o tipo de depósito (dinheiro, cheque da praça, cheque de outra praça, etc.). A combinação desse atributo com outros irá definir as classes de equivalência.

As classes de equivalência podem ser determinadas para as entradas das transações do sistema, caso em que são denominadas classes de entrada. De forma análoga, as saídas do sistema podem ser organizadas em classes de equivalência, denominadas classes de saída. Uma vez identificadas e refinadas as classes de entrada e as classes de saída, será o momento de construir a matriz de entradas/saídas do sistema. Essa matriz conterá, para cada classe de entrada, a(s) classe(s) de saída correspondente(s) que serão geradas. Para dar um exemplo simples, consideremos um sistema de cobrança. Para a classe de entrada "fatura vencida e não paga", deverá ser gerado um resultado na classe de saída "aviso de cobrança".

Determinadas as classes de entrada, as classes de saída e a matriz de correspondência entre as duas, será o momento de construir os casos de teste. Uma forma de operacionalizar esta tarefa é gerar pelo menos três casos de teste para cada classe de entrada: o caso mínimo, o caso médio e o caso máximo. Mínimo, médio e máximo deverão ser determinados de acordo com a natureza de cada classe. No caso anterior de uma fatura vencida, mínimo seria a data imediatamente posterior ao vencimento, médio seria um atraso típico e máximo seria um atraso inusitado (100 anos).

De posse dos casos de teste, esses devem ser combinados em suites ou roteiros de teste. Um roteiro de teste é constituído por um ou mais casos de teste, de modo a reproduzir um conjunto de transações (ou casos de uso) do mundo real. Um roteiro de teste poderia ser, por exemplo, a abertura de uma conta, seguida de um depósito e, finalmente, uma retirada. Para cada roteiro de teste definido, devem ser previamente determinadas as classes de resultados esperados. Em alguns casos será necessário calcular os valores exatos dos resultados. Em outros casos, bastará a classe de saída correspondente.

Com os roteiros de teste e os resultados esperados, passaremos à execução dos testes. Após a execução, os resultados obtidos deverão ser comparados com os esperados. As divergências serão as ocorrências a resolver. Uma ocorrência pode ser um erro do sistema ou da especificação. Erros de especificação devem ser remetidos aos usuários para correção e gerarão uma alteração no sistema. Erros do sistema deverão dar ensejo à identificação dos componentes afetados, que por sua vez serão objeto de correção. Resolvida a ocorrência, o teste que deu origem à mesma deverá ser novamente executado. Adicionalmente, todos os demais testes deverão ser novamente executados em algum momento, antes da entrega do sistema. A reexecução dos testes anteriores é denominada teste regressivo.

Através da execução dos procedimentos acima delineados, conseguimos identificar cerca de 400 ocorrências, as quais foram resolvidas ao longo de alguns meses. O sistema foi finalmente implantado, apresentando um índice de erros muito baixo para nossas expectativas. Na época, nossas medições limitavam-se a registrar as ocorrências e seus tipos. Nossa organização não estava conscientizada para a implementação de métricas.

É importante registrar que, além dos testes funcionais, também utilizamos testes estruturais (ou "caixa-branca") como complementação de nosso trabalho. Conseguimos, com uma empresa parceira, um sistema que dava suporte a teste de cobertura. Este tipo de teste consiste em exercitar todas as partes do código de cada programa. O sistema tomava o programa a ser testado como entrada e gerava um novo programa. Esse novo programa possuía um statement adicional para cada statement do programa original, com o objetivo de registrar quantas vezes cada statement do programa original havia sido executado. Ao final de cada execução, recebíamos um relatório. Se houvesse algum statement do programa original que não tivesse sido executado, isso significava que deveríamos aumentar nossos casos de teste, de modo a causar a execução da parte do código ainda não exercitada.

Hoje em dia, é muito mais fácil implementar as atividades de teste de software do que na década de 80. Dispomos de ferramentas automatizadas, que fazem-nos considerar nosso antigo sistema de cobertura de código como rudimentar, embora o mesmo fosse um avanço à época. Uma dessas ferramentas é o produto Code Coverage. Hoje existem ferramentas capazes de gravar e reproduzir todos os comandos emitidos por um usuário de ambiente CICS, TSO ou Windows, como o é o caso do produto Hiperstation. As atuais ferramentas de cobertura de código utilizam a complexidade ciclomática de McCabe como base para determinar quando um módulo foi suficientemente testado, como fazem esses produtos. O controle da execução dos testes, o armazenamento dos casos e dos roteiros de teste, a comparação entre os resultados gerados e os esperados, tudo isso foi automatizado. Hoje em dia, é indiscutivelmente algumas ordens de grandeza mais fácil realizar as atividades de teste do que era na década de 80. Afinal, passaram-se quase 20 anos! Será que, nesse contexto, as empresas estão efetivamente utilizando técnicas de teste de software? Pesquisa realizada pelo QAI (Quality Assurance Institute, organização norte-americana dedicada à qualidade de software), indica que 71% das organizações pesquisadas mantém padrões e procedimentos para teste de software. O percentual de organizações que pretende melhorar seus procedimentos de teste de software é de 88%. Cerca de 63% das organizações treinam seus funcionários em técnicas de teste. Um número bastante alto (91%) coleta dados sobre defeitos.

Nos últimos anos, as organizações mundiais vêm desenvolvendo esforços crescentes no sentido de promover a melhoria de seus processos de software. O padrão CMM do Software Engineering Institute tem sido um importante estímulo e balisador deste esforço. No nível 2 do CMM, temos a KPA (Key Process Area) referente à garantia de qualidade de software, que tem como objetivo a aderência dos produtos e atividades de software aos padrões, procedimentos e requisitos aplicáveis [KCAP98]. Um dos instrumentos para a implementação da garantia de qualidade de software é o teste de software, disciplina exigida pela ASQ – American Society for Quality, para a obtenção da certificação SQE – Software Quality Engineer [GSCH97].

Em abril deste ano, tive a oportunidade de visitar as instalações da Ford, em Detroit, e da Texas Utility, em Dallas. Ambas as companhias utilizam processos e ferramentas para implementar as atividades de teste de software. Nos Estados Unidos, existem as profissões de test engineer (engenheiro de testes), test architect (arquiteto de testes) e outras relacionadas à atividades de teste. Tudo indica que essas atividades virão a se expandir em nosso país.

Espero que o leitor tenha adquirido uma idéia dos conceitos básicos envolvidos na atividade de teste de software e, principalmente, tenha percebido que é necessário voltar sua atenção para esta área. Para uma visão prática e atual do problema, recomendo o livro de Black [BLAC99].

CONTRATANDO O DESENVOLVIMENTO COM BASE EM MÉTRICAS

Uma forma muito popular de contratação de serviços de desenvolvimento de sistemas é aquela denominada, nos Estados Unidos, body shop (loja de corpos). Em nosso país utiliza-se a expressão locação de mão-de-obra. Na operação tipo body shop, uma empresa fornecedora aloca um profissional a uma empresa cliente, cobrando por isso um valor mensal. Essa é uma operação com um risco muito baixo para o locador, pois no caso do profissional alocado não atender ao pretendido pelo cliente, o que se faz é simplesmente substituí-lo. O risco dos projetos não é compartilhado pelo locador de mão-de-obra, que não precisa participar do pesadelo que costuma ser o gerenciamento dos projetos de TI. Claro que há soluções intermediárias, nas quais o fornecedor divide o gerenciamento dos resultados com o cliente, mas essas não podem ser consideradas como body shop. No body shop propriamente dito, não existe o comprometimento do fornecedor de mão-de-obra com os resultados. Por essa razão, é natural que as empresas busquem soluções alternativas para a contratação do desenvolvimento de sistemas.

De dois ou três anos para cá, algumas grandes empresas brasileiras resolveram partir para a contratação do desenvolvimento de sistemas baseado em métricas de software. Nessas contratações, o objeto produzido através do contrato (programa, sistema, documento, etc.) é quantificado e pago através de alguma medida objetiva, realizada sobre o próprio objeto gerado. Teoricamente, essa é uma solução ideal, pois o que está sendo pago é o resultado, ao invés dos recursos ou insumos utilizados na sua geração. O problema dessa abordagem passa a ser a escolha, a medição e a interpretação da métrica a ser utilizada na quantificação dos serviços contratados.

No caso do desenvolvimento de sistemas, o principal item que se busca medir é o software produzido através do respectivo contrato. Há contratos específicos para o desenvolvimento de um único sistema, bem como outros destinados ao desenvolvimento e manutenção de diversos sistemas, programas e outros artefatos. Este último tipo de contrato é por vezes denominado "guarda-chuva", por abrigar uma grande variedade de serviços.

As medidas mais utilizadas para a medição de software, tanto no mundo acadêmico quanto na indústria, são as linhas de código e os pontos de função. As linhas de código possuem a grande vantagem da medição automática, através de programas de computador. Tal não é possível com os pontos de função, embora algumas ferramentas busquem realizar essa tarefa, com graus variáveis de acerto. Por outro lado, é difícil estimar linhas de código no início de um projeto, quando ainda não se tem definida a arquitetura; além disso, a quantidade de linhas de código varia com a linguagem utilizada e, o que é pior, com o próprio estilo de programação da equipe. Essas dificuldades têm contribuído para a crescente disseminação dos pontos de função, regulamentados e periodicamente atualizados pelo International Function Point Users Group (IFPUG), organização sem fins lucrativos sediada nos EUA.

Os pontos de função permitem medir a funcionalidade de um sistema independentemente da técnica utilizada em sua implementação. Dessa maneira, os pontos de função são independentes da linguagem de programação e da plataforma utilizada. Funcionam como a medida de um imóvel em metros quadrados: um apartamento de 100 metros quadrados terá os mesmos 100 metros quadrados, esteja em Manaus ou em Porto Alegre, em um bairro da periferia ou na região mais cara, construído com materiais de primeira linha ou com os mais baratos, finamente decorado ou vazio. Igualmente, um sistema com 500 pontos de função terá esse mesmo tamanho, seja ele implementado através de COBOL/CICS e DB2 no mainframe, em Delphi com Oracle em uma arquitetura de 3 camadas, ou como uma aplicação Web utilizando Java, Solaris e Sybase.

Ao medir um aplicativo utilizando pontos de função, os elementos considerados são componentes visíveis e reconhecidos pelo usuário, tais como arquivos lógicos, transações de entrada e de saída e consultas efetuadas. A visão lógica e centrada no ponto de vista do usuário garante a independência com relação à implementação. Na contagem de pontos de função, é necessário exercitar o processo de abstração utilizado para identificar os componentes contáveis no modelo do sistema, conforme descrito pelo usuário ou registrado na documentação existente. A fim de garantir que todos os contadores de pontos de função utilizem o mesmo procedimento padrão, o IFPUG oferece a certificação profissional. Através dela, tanto o próprio profissional quanto as empresas contratantes podem certificar-se de que as contagens estarão sendo efetuadas corretamente.

Uma vez determinado o tamanho funcional do sistema, o próximo passo é dispor de um método para obter o custo do serviço de desenvolvimento a partir do tamanho em pontos de função. Uma saída simples, mas não recomendável, é estabe-

lecer um custo fixo para cada ponto de função gerado, isto é, um preço por ponto de função. Acontece que o custo de um ponto de função vai variar segundo diversos fatores, dificultando o estabelecimento de um preço único por PF. Na nossa analogia anterior, o ponto de função equivaleria ao metro quadrado, enquanto o preço por ponto de função equivaleria ao preço por metro quadrado. Fica fácil ver que não é possível estabelecer um único preço por metro quadrado para todos os tipos de imóveis, independente da localização e do acabamento utilizado. Da mesma maneira, não podemos ter um único preço por ponto de função para todos os sistemas, independente da plataforma, linguagem, etc. Esses são fatores que irão afetar a produtividade da equipe de desenvolvedores.

O esforço despendido em um serviço de desenvolvimento é o número de horas gasto para realizá-lo, o qual pode ser dado por: E = F x T, onde E representa o esforço em horas, F o tamanho em pontos de função e T a taxa de entrega em horas gastas por ponto de função. A taxa de entrega é o inverso da produtividade. Conhecido o número de horas E, o custo pode ser obtido multiplicando-se E pelo valor unitário da hora: C = E x H, onde C representa o custo do serviço, E o esforço e H o valor unitário da hora. Vemos então que, além do tamanho do sistema em pontos de função, precisamos conhecer a taxa de entrega e o valor unitário da hora, para que possamos ter o custo. O tamanho em pontos de função pode ser determinado por um contador de pontos de função experiente. O valor da hora é conhecido do mercado, mesmo porque já é utilizado nos contratos do tipo body shop. Resta conhecer a taxa de entrega, que reflete a produtividade. Quais empresas brasileiras conhecem sua própria produtividade? Parece que poucas.

Não há estatísticas conhecidas para a taxa de entrega (horas por ponto de função) ou para a produtividade (pontos de função por pessoa por mês) das empresas brasileiras que produzem aplicativos voltados para negócios. Poucas empresas mantém algum tipo de programa de métricas e, quando o fazem, nem todas utilizam uma medida padrão que permita comparações com outras empresas, o principal benefício na utilização dos pontos de função do IFPUG.

Nesse contexto, faz-se necessário o estabelecimento de programas de métricas, que permitam aos clientes e fornecedores conhecerem sua própria produtividade. "Por que uma empresa cliente desejaria conhecer sua própria produtividade?", o leitor poderia perguntar. "Não bastaria conhecer a produtividade das melhores empresas do mundo e exigir esse mesmo nível dos fornecedores?" Talvez sim, no caso da aquisição de uma mercadoria, ou de um serviço totalmente independente do ambiente do cliente. Mas tal não é o caso. No desenvolvimento de sistemas, as demoras, dificuldades e indefinições dos usuários poderão afetar fortemente a fase de concepção ou análise; os processos internos e tecnologia utilizados pela área de TI do cliente poderão, por sua vez, impactar bastante a fase de elaboração ou projeto; os procedimentos de aceitação e a forma adotada para o gerenciamento das mudanças de escopo direcionarão os riscos da fase de construção ou programação; finalmente, a localização geográfica e o cronograma dos usuários irão determinar o tempo para a transição ou implantação do sistema. Para que uma empresa possa exigir uma determinada produtividade de um fornecedor, é preciso que ela tome como ponto de partida a sua própria produtividade, buscando melhorias sucessivas ao longo da parceria resultante da contratação. O conhecimento da própria produtividade é, neste caso, o principal objetivo do programa de métricas a ser implantado.

É muito comum as pessoas buscarem números de terceiros, que possam substituir as medições acima indicadas. Este autor recebe, regularmente, uma razoável quantidade de mensagens de colegas que buscam o número mágico. As perguntas mais comuns são do tipo "Em quantas horas se faz um ponto de função, em um ambiente cliente/servidor, usando VB com SQL Server?", ou "Quantos pontos de função faz um programador COBOL por mês?". É claro que tais números existem, nas estatísticas internacionais. O problema é: eles se aplicam ao seu caso? Há uma grande probabilidade de que não.

A variabilidade do processo de desenvolvimento de sistemas é muito grande. Diferentes empresas vão utilizar diferentes metodologias. Muitas empresas não utilizam, consistentemente, qualquer metodologia, embora quase todas disponham de alguma. Quanto uma metodologia padrão é utilizada, os projetos por sua vez são diferentes entre si. Mesmo quando os projetos são comparáveis, muitas vezes as equipes é que não são. Resumindo, o fenômeno que se deseja estudar, o desenvolvimento de sistemas, é tão variável que desafia a definição. Se, ainda assim, aceitarmos que todos que escrevem sobre desenvolvimento de sistemas estão tratando da mesma coisa, teremos que considerar a questão das medições. Para obter a produtividade, precisamos medir o tamanho e o esforço despendido. A medida dos pontos de função pode ser razoavelmente precisa, se forem utilizados contadores experientes. Por outro lado, os critérios para registro do tempo despendido podem variar bastante. Se não houver um padrão para isso, não teremos como saber quem e o quê foi medido. Por exemplo: foi medido o tempo gasto pelos administradores de dados neste projeto? Foi considerado o tempo do pessoal de suporte? O projeto foi medido desde a primeira reunião realizada para tratar do mesmo? O anteprojeto foi considerado nas medições? O treinamento ao usuário foi incluído? Após o aceite final do usuário ainda houve alguma atividade registrada? Deveria ter havido? É mais importante ter um único critério, consistente, do que ficar discutindo se determinado tipo de atividade deve ou não entrar no cômputo das horas despendidas no projeto.

Como conseqüência da grande variabilidade do fenômeno e das respectivas estratégias de medida diferenciadas, as estatísticas internacionais sobre produtividade também variam muito. Isso pode ser verificado através da comparação entre os dados provenientes de três respeitadas fontes de dados sobre as atividades de TI: o International Software Benchmarking Standards Group (ISBSG), Capers Jones e

Howard Rubin (ver referências ao final do artigo). Comparando a produtividade do desenvolvimento, medida em pontos de função por pessoa por mês, veremos que o banco de dados do ISBSG registra cerca de 19 PF, Capers Jones indica 7,5 PF e Howard Rubin chega a 3,6 PF. Isso significa uma variação de quase 2 vezes entre Rubin e Jones, e de quase 5 vezes entre Rubin e o ISBSG. Qual desses é o caso da sua empresa? Ou será algum outro? Os dados citados são de 1998.

É inevitável a decepção daqueles que esperam do processo de mensuração de software alguma fórmula mágica ou revelação mística. Exatamente como quando tratamos de metodologias de desenvolvimento, ferramentas CASE ou técnicas de modelagem, não há uma solução brilhante que resolva todos os problemas. Como dizem os americanos, "there is no silver bullet" (não existe a bala de prata). A contratação de software com a utilização de métricas passa, primeiro, pela implantação de um programa de mensuração, também chamado programa de métricas. As estatísticas internacionais são úteis, como ajuda para a validação de nossas próprias métricas e para posicionamento de nosso processo em relação às demais organizações.

A implantação de um programa de métricas requer um esforço específico da organização. O projeto pode ser efetuado em 8 passos, resumidamente descritos a seguir: 1) Documentar o processo de desenvolvimento atualmente utilizado e padronizá-lo (sem buscar a realização de melhorias, mas tão somente a estabilização do processo); 2) Estabelecer os objetivos do programa de métricas (em nosso caso, determinar a produtividade); 3) Definir as métricas necessárias ao alcance dos objetivos pretendidos (por exemplo: tamanho funcional em pontos de função, esforço em horas, qualidade em densidade de erros, etc.); 4) Identificar os dados a serem coletados (por exemplo: horas trabalhadas de cada técnico envolvido, tamanho funcional de cada uma das solicitações de alteração do sistema, erros identificados e suas categorias, etc.); 5) Definir o processo de coleta de dados (identificar os pontos de coleta, periodicidades, formulários ou sistema de coleta, etc.); 6) Obter ferramentas (construir e/ou adquirir as ferramentas que permitam implementar o processo de coleta anteriormente definido); 7) Criar um banco de dados de métricas (implementar um repositório que possa abrigar, de forma organizada e acessível, todos os dados que virão a ser coletados); 8) Definir um mecanismo de feedback (uma maneira que permita à equipe de métricas receber feedback de todos os participantes do processo). Existem várias outras formas de se conduzir o processo de implantação de um programa deste tipo. O importante é o comprometimento da gerência e a clara definição dos objetivos do programa, de modo a garantir que as métricas geradas sejam aquelas necessárias e efetivamente utilizadas.

Uma vez implantado o programa de métricas, a organização começará a obter dados de produtividade. Em uma organização imatura, ainda no nível 1 da classificação CMM, é provável que a produtividade varie de forma errática - refletindo, é claro, o fenômeno que está sendo medido. Gradativamente, será possível separar os projetos por plataforma, ramo de negócio, localização geográfica, perfil da equipe e outros fatores que estejam influenciando os resultados. O tratamento de categorias separadas tenderá a reduzir a variabilidade. Após a estabilização dos fatores mais influentes, será possível estabelecer uma produtividade média para cada categoria definida. Esse será o ponto de partida para a obtenção de melhorias junto aos fornecedores. Por exemplo, uma empresa poderia firmar um contrato com um fornecedor, tendo como alvo um aumento de produtividade de 25% no primeiro ano e de 10% no segundo (este é apenas um exemplo, valores reais só podem ser determinados mediante cuidadosa análise por parte do cliente e do fornecedor).

Este artigo procurou mostrar que a contratação do desenvolvimento de sistemas com a utilização de pontos de função (ou outra métrica) não pode ser realizada exclusivamente com base em dados de terceiros, ainda que publicados em respeitáveis fontes internacionais. Antes da contratação, é imprescindível a implantação de um programa de métricas, que permita ao cliente estabilizar seu processo de desenvolvimento e conhecer sua própria produtividade. Com base nesse conhecimento, melhorias poderão ser buscadas e obtidas junto ao mercado.

Grande ajuda pode ser conseguida através da troca de experiências com empresas assemelhadas, tanto para contratantes quanto para contratados. As melhores práticas de contratação só podem ser conseguidas com o concurso de ambas as partes, em um clima do tipo "ganha-ganha". Tal ambiente não pode ser obtido durante o calor de uma concorrência ou negociação específica. Com o intuito de dar ensejo a tal modo de interação, o Brazilian Function Point Users Group (BFPUG) instituiu, em julho deste ano, um Comitê com a missão de buscar as melhores formas de contratação para o desenvolvimento de sistemas, adequadas aos grandes contratantes e exeqüíveis para os grandes contratados. O primeiro encontro, realizado no Rio de Janeiro, foi alvo do interesse e participação de dez grandes empresas, o que demonstra a preocupação da indústria com essa importante questão.

REFERÊNCIAS

[BEIZ95] – Beizer, B.:Black-Box Testing. Wiley, USA, 1995.

[BLAC99] – Black, R.:Managing the Testing Process. Microsoft Press, USA, 1999.

[GSCH97] – Schulmeyer, G. G.; McManus, J. I. (Ed.). Handbook of Software Quality Assurance. 3rd. Edition. Prentice Hall, USA, 1998.

[KCAP98] – Caputo, Kim: CMM Implementation Guide. Addison-Wesley, USA, 1998.

[MEYE79] – Meyers, Glenford – The Art of Software Testing. Wiley, USA, 1979.

Jones, Capers T. – Estimating Software Costs – McGraw-Hill, 1998.

Rubin Systems, Inc. – IT Performance Trends '99 – Meta Group, 1999.

■ *Mauricio Aguiar (mauricioaguiar@yahoo.com.) é analista de sistemas, Presidente do Brazilian Function Point Users Group, membro do IFPUG Communications & Marketing Committee e Certified Function Point Specialist (CFPS) pelo IFPUG.*