Avaliação de um Método para Estimativa de Esforço para Testes baseado em Casos de Uso

Érika R. C. de Almeida[1], Bruno T. de Abreu[1], Regina Moraes[2], Eliane Martins[1]

[1]Instituto de Computação – Universidade Estadual de Campinas (Unicamp)
Caixa Postal 6.176 – 13.083-970 – Campinas – SP – Brasil

[2]Centro Superior de Educação Tecnológica – CESET – Universidade Estadual de Campinas (Unicamp) Caixa Postal 456 – 13.484-332 – Limeira – SP – Brasil

erikarca@gmail.com, brunotx@gmail.com, regina@ceset.unicamp.br, eliane@ic.unicamp.br

Abstract*. This experience report evaluates an effort estimation method for software testing, as well as the results from its application on three Brazilian Federal Revenue Service's software specifications. The methodology is based both on the software's use cases, usually developed before the software itself, and technical and environment factors. The results showed that the method was simple to be applied, but it had weaknesses that make its use not efficient on a real environment due to its high percent of error on the estimations.*

Resumo. *Este relato de experiência avalia um método para estimar o esforço (em horas) para a atividade de testes de software, bem como os resultados obtidos após aplicá-lo em especificações de sistemas de software da Receita Federal do Brasil. O método utilizado é baseado tanto na especificação dos casos de uso do software, normalmente elaborada antes mesmo do software, quanto em fatores técnicos e de ambiente de testes. Os resultados mostraram que o método foi simples de ser utilizado, porém este apresentou deficiências que fazem com que seu uso seja inviável no ambiente real pelo fato de gerar estimativas com um grau elevado de erro.*

1. Introdução

Estimar esforço para testes é um processo pouco explorado, ainda mais quando comparado com a quantidade de opções existentes para estimar o tempo para o desenvolvimento de sistemas de software [Aranha e Borba 2007b]. Apesar de ser pouco explorada, ela não é menos importante, uma vez que os testes são essenciais para verificar se o software atende aos requisitos especificados. Cada vez mais é dada a devida importância para o procedimento de testar um sistema, pois as empresas, independente de seu tamanho, não querem colocar em risco seu prestígio, entregando para seus clientes produtos que têm grandes chances de não funcionar conforme especificado, ou apresentarem falhas de qualquer natureza [Aranha e Borba 2007a].

Para que a atividade de testes faça parte do ciclo de vida de um sistema, é necessário planejar e, conseqüentemente, obter uma estimativa de quanto tempo será necessário para realizá-la [Pressman 2004], para que os recursos sejam devidamente alocados e os prazos possam ser minimamente delineados. O planejamento também se faz necessário para que haja tempo suficiente para que os testes sejam realizados e o

produto entre no mercado suficientemente testado, evitando reações negativas nos consumidores e na imagem da empresa [Nageswaran 2001].

No entanto, estimar tempo e recursos a serem alocados para testes não é uma tarefa fácil, tanto que muitos especialistas em gerenciamento de projetos têm dificuldade com essa área [Black 2006]. Para ajudar na escolha do método adequado, existem modelos sobre o que deve ser analisado para estimar o esforço para testes como, por exemplo, os passos de testes [Aranha e Borba 2007a], regras e histórico da empresa [Cardoso 2002] e casos de uso do sistema [Nageswaran 2001].

Dessa forma, o objetivo deste trabalho é descrever os resultados obtidos por uma Equipe Independente de Verificação e Validação (EIVeV) ao aplicar um método para estimar esforço de testes a partir da especificação de casos de uso de sistemas de software da Receita Federal do Brasil (RFB). O método escolhido foi aplicado em três sistemas da RFB, e os resultados das estimativas foram comparados com os tempos efetivamente consumidos pela equipe para preparar e realizar os testes nos sistemas, oferecendo informações relevantes para analisar o método utilizado.

O presente trabalho é organizado da seguinte forma: a Seção 2 descreve brevemente alguns trabalhos relacionados a estimativas de esforço para testes, enquanto a Seção 3 apresenta de forma detalhada o método escolhido para avaliação. A seção seguinte apresenta os resultados obtidos com a aplicação de tal método e, por último, são apresentadas as conclusões do trabalho e sugestões para trabalhos futuros.

2. Trabalhos Relacionados

A criação de métodos de estimativa de tempo necessário para testar um sistema muitas vezes se baseia em métodos para estimativa de esforço de desenvolvimento do mesmo. Em meados dos anos 70, Allan Albrecht encontrava dificuldades para fazer medidas, em relação a desenvolvimento de software, utilizando linhas de código. Por isso, desenvolveu e publicou o método *Function Point Analysis* (FPA), que baseava suas métricas nas funções do software [Ferens e Gurner 1992].

Algum tempo depois, o FPA serviu de base para a criação de um método de estimativa de testes chamado *Test Point Analysis* (TPA) [Veenendaal e Dekkers 1999]. O TPA utiliza o FPA, pois precisa do resultado final dele, os *Function Points* (FPs), para obter seu próprio resultado. Além de utilizar os FPs, ele também leva em conta outros três elementos para a obtenção de uma variável importante, chamada de *Test Points* (TPs): **(i)** o tamanho do software, **(ii)** a estratégia de teste e **(iii)** o nível de produtividade da equipe de testes.

Aranha e Borba desenvolveram uma outra abordagem que considera os passos de um caso de teste, descritos em linguagem natural, para a estimativa de esforço [Aranha e Borba 2007a]. Cada passo deve ser analisado conjuntamente a uma lista de características e, se o passo exercitar determinada característica, o impacto dela é verificado, resultando na atribuição de um valor chamado de *Execution Point* (EP). O valor de EP para um passo de teste é dado pela soma dos EPs após a verificação de todas as características. Já a estimativa de esforço total é obtida através da multiplicação de um fator de conversão, na forma *tempo em segundos / EP*, pela soma dos EPs dos passos de teste.

Uma abordagem mais simplista envolve, primeiramente, cálculos simples

baseados no tempo que uma equipe de desenvolvimento necessitou para desenvolver o sistema [Cardoso 2002]. Num segundo momento, são impostas regras para esta estimativa, sendo que são considerados dados como o histórico da equipe e os requisitos do software. Este método não foi comparado neste artigo, pois ele não provê uma estimativa no início do projeto, já que depende do tempo gasto no desenvolvimento do sistema e faz considerações pouco refinadas e detalhadas, podendo interferir em sua aplicação em um estudo de caso.

Finalmente, vale citar o trabalho de Nageswaran [Nageswaran 2001], cujo método foi o escolhido para este relato de experiência. Este método estima o esforço de testes utilizando informações contidas nos casos de uso do software. A idéia básica é separar as informações de cada caso de uso em informações sobre os atores envolvidos, o número de transações ou classes de análise, fatores técnicos e de ambiente. Tais informações são convertidas em pontos que, quando multiplicados por um fator de conversão, resultam em uma estimativa do tempo que será necessário para os testes do software. É importante comentar que este trabalho é uma adaptação da metodologia *Use Case Points Method* (UCPM) [Schneider 1998], proposta por Gustav Karner como um algoritmo de estimativa de esforço de desenvolvimento que emprega casos de uso como uma representação da complexidade do sistema [Merrick 2005].

3. O Método de Nageswaran

O método de Nageswaran se baseia nos casos de uso do sistema, permitindo que as estimativas sejam obtidas no início do ciclo de vida de um projeto, pois é o momento em que os casos de uso ficam disponíveis. O primeiro passo do método prevê a identificação e análise dos atores descritos em cada caso de uso do software. Os atores são classificados e pontuados de acordo com a interação que provê ao usuário, conforme ilustra a Tabela 1.

Tabela 1. Classificação dos atores de casos de uso

Tipo de Ator	Descrição	Peso
Simples	GUI (mais conhecidos como *end-users*, ou usuários finais)	1
Médio	Interfaces interativas ou de protocolo; gerenciamento de banco de dados	2
Complexo	API e interações de baixo nível	3

Nesta classificação, um ator simples é aquele que utiliza o sistema através de suas interfaces gráficas; por outro lado, um ator médio utiliza o sistema através de protocolos ou cuida do armazenamento dos dados. Já atores complexos realizam interações utilizando APIs[1]. Ao final, cada caso de uso terá uma pontuação final (igual ao somatório dos pesos identificados para cada ator) que, somada, resultará em uma pontuação total chamada de *Unadjusted Actor Weights* (UAW).

O segundo passo leva em conta a quantidade de cenários ou transações existentes em cada caso de uso do software para obter a pontuação associada ao peso dos casos de uso. Um cenário (ou transação) pode ser definido como um conjunto atômico de atividades que levam a um fluxo normal de uso do sistema ou a um fluxo excepcional (ou de exceção), no qual um erro se manifesta no software. A Tabela 2

[1] Uma API é um conjunto de padrões para utilizar o software, normalmente composto por funções acessíveis somente por programação que permitem acesso a características menos evidentes ao usuário final.

mostra a correspondência entre a quantidade de cenários ou transações e o tipo do caso de uso. De acordo com o tipo identificado, um peso é considerado na classificação.

Tabela 2. Classificação de caso de uso conforme a quantidade de cenários ou transações

Tipo do Caso de Uso	Descrição	Peso
Simples	<= 3 transações / cenários	1
Médio	4-7 transações / cenários	2
Complexo	>7 transações / cenários	3

Tal como para os atores dos casos de uso, a soma total dos pesos atribuídos para cada caso de uso também é avaliada e recebe o nome de *Unadjusted Use Case Weights* (UUCW). Dado que o UAW e UUCW foram obtidos, é possível calcular o *Unadjusted Use Case Points* (UUCP), que é igual à soma de UAW e UUCW.

No terceiro passo do método, Nageswaran considera fatores técnicos e de ambiente no qual serão realizados os testes. Os fatores que são levados em conta são: ferramentas de teste, entradas documentadas, ambiente de desenvolvimento, ambiente de testes, reutilização de ferramenta de teste, sistema distribuído, objetivos de desempenho, atributos de segurança e interfaces complexas. No entanto, o autor não deixa claro quais são as diretrizes detalhadas para analisar estes fatores e escolher os valores adequados de uma maneira que não seja subjetiva. Sua proposta diz que um valor no intervalo [0-5] deve ser atribuído a cada fator, onde 0 (zero) significa que o fator é irrelevante e 5 (cinco) significa que tal fator é essencial. Além disso, cada fator recebe um peso no intervalo [0-5]. A multiplicação do valor atribuído ao fator pelo seu peso é chamada de valor estendido, e a soma total dos valores estendidos de todos os fatores é chamada de *Technical Environment Factor* (TEF).

Já o quinto passo do método obtém o valor final de pontos de caso de uso, chamado de *Adjusted Use Case Point* (AUCP), a partir da fórmula descrita na Figura 1. Para chegar a um valor final de esforço em homens-hora, é necessário multiplicar o valor de AUCP por um fator do tipo *homens-hora / ponto de caso de uso*, conforme ilustra a Figura 2. Para que esse valor possa ser convertido em total de horas ou dias, deve ser feito um levantamento da disponibilidade das pessoas da equipe de testes.

AUCP = UUCP * [0,65 + (0,01 * TEF)]

Figura 1. Fórmula de cálculo de AUCP

Esforço = AUCP * [homens-hora / ponto de caso de uso]

Figura 2. Fórmula de cálculo do esforço (em horas) para testes

O último passo do método prevê a adição de um percentual (aplicado no valor do esforço final) relativo à complexidade do projeto e ao esforço para coordenação e gerenciamento dos testes no software. Tal como na análise dos fatores técnicos, Nageswaran também não fornece diretrizes para obter o valor destes percentuais de maneira sistemática.

4. Aplicação do método e lições aprendidas

Uma vez entendido o método, a EIVeV aplicou-o para estimar o esforço para os testes de três sistemas *web* da RFB, cujas características gerais são mostradas na Tabela 3.

Uma vez que as informações a respeito de cada um dos sistemas não são públicas, estes serão chamados no decorrer desta seção de “Software I”, “Software II” e “Software III”. É relevante comentar que o passo-a-passo da aplicação do método será descrito somente para o Software I, uma vez que os mesmos procedimentos aplicados neste foram aplicados ao Software II e III. Todas as atividades realizadas pela equipe foram monitoradas em relação ao tempo consumido para executá-las e, ao final, o resultado real obtido foi comparado com a estimativa do método de Nageswaran.

Tabela 3. Características dos sistemas onde o método foi aplicado

Nome do software	Número de casos de uso	Número de LOCs (*lines of code*)	Tipo de sistema
Software I	50	44.275	*Web*
Software II	10	25.750	*Web*
Software III	13	18.340	*Web*

No primeiro passo, que é a identificação e classificação dos atores dos casos de uso, a equipe analisou cada um dos casos de uso do Software I. A partir deles, os seus respectivos atores foram obtidos e verificou-se em qual classificação, conforme a Tabela 1, eles se encaixavam. O resultado final deste passo encontra-se na Tabela 4, que cita o nome do ator, a quantidade de casos de uso em que ele participa, seu peso e o UAW parcial. Como dito anteriormente, a soma de todos UAW parciais resulta em UAW que, para o Software I, foi igual a 73 (setenta e três).

Tabela 4. Identificação dos atores do Software I

Nome do ator	Número de casos de uso em que participa	Peso	UAW parcial
Ator I	24	1	24
Ator II	19	1	19
Ator III	18	1	18
Ator IV	12	1	12
UAW (Unadjusted Actor Weights)			73

Em seguida, foram analisadas as quantidades de cenários em cada caso de uso e aplicada a classificação prevista na Tabela 2. De acordo com a análise, foram identificados no Software I 50 (cinqüenta) casos de uso, sendo 44 (quarenta e quatro) deles do tipo “Simples” (de 1 a 3 cenários / transações) e 6 (seis) do tipo “Médio” (de 4 a 7 cenários / transações). Como se atribui peso 1 (um) para um caso de uso do tipo simples e 2 (dois) para o médio, o valor obtido para UUCW foi de 56. Conseqüentemente, foi possível obter o valor de 129 para UUCP, igual à soma de UAW com UUCW.

Dando seqüência à aplicação do método, a equipe teve grandes dificuldades no terceiro passo do método, relativo ao cálculo do fator de complexidade técnica ou TEF. Isto ocorreu devido à (i) falta de detalhes a respeito de *como* atribuir um valor e um peso para cada fator técnico especificado, além do (ii) próprio significado de cada fator. Em relação à primeira questão, a EIVeV adotou as seguintes definições: o valor atribuído ao fator se refere à disponibilidade dele no ambiente de testes, enquanto que o seu peso representa o grau de importância dele para os testes. A segunda questão, que trata do significado dos fatores, foi resolvida utilizando o conhecimento dos membros da EIVeV para se chegar um consenso em relação ao grau de importância de cada fator.

Os valores atribuídos para cada fator no cálculo de TEF seguiram o que Nageswaran indicou em seu trabalho, uma vez que o software utilizado como exemplo

por ele possuía características semelhantes aos da RFB, avaliados neste trabalho. No entanto, os pesos atribuídos a cada fator foram diferentes, e seguiram o que é indicado na Tabela 5. Logo, o valor de TEF obtido foi 85 (oitenta e cinco).

Tabela 5. Pesos dos fatores técnicos e de ambiente

Fator	Peso	Fator	Peso
Ferramenta de Teste	3	Sistema Distribuído	3
Entradas Documentadas	5	Objetivos de Desempenho	1
Ambiente de Desenvolvimento	1	Atributos de Segurança	3
Ambiente de Testes	3	Interfaces Complexas	1
Reutilização de Ferramenta de Teste	1		

O quarto passo do método tratou dos cálculos finais do esforço para testes. O cálculo do valor de AUCP, ilustrado na Figura 1, foi simples pelo fato dele ser obtido a partir dos valores de UUCP e TEF. No entanto, o cálculo final de esforço, apresentado na Figura 2, gerou dúvidas quanto ao valor do fator de conversão que deveria ser utilizado. O método diz que este valor deve ser 20 (vinte) no caso de software que faz uso da tecnologia *Enterprise Java Beans* (EJBs), pois é o tempo gasto para planejar, escrever e executar um ponto de caso de uso neste caso específico.

No entanto, o autor não define como identificar este fator para o caso de software que faça uso de outras tecnologias, o que torna a escolha do fator muito subjetiva. Além disso, ele não faz considerações a respeito de como este fator pode ser adaptado caso a equipe de testes possua ferramentas para automação do processo de testes. Esta dificuldade foi considerada a mais crítica e de maior impacto na aplicação do método. A criticidade alta é devida ao que foi falado anteriormente sobre não explicar maneiras para adaptá-lo, pois a EIVeV possui várias soluções para automação dos testes (implicando em redução do esforço nos testes). Já o impacto foi considerado alto, pois a escolha inadequada do fator de conversão afeta diretamente o resultado da estimativa de esforço de testes.

Os resultados obtidos para o Software I foram um valor total para AUCP de 193,5 e um valor de esforço de 3870 horas, conforme mostra a Figura 3. A este valor, foi acrescido um percentual devido à complexidade do projeto e sua coordenação e gerenciamento. Como Nageswaran não fornece diretrizes para a definição destes valores, a EIVeV chegou ao consenso que 10% do tempo total para os testes era consumido em coordenação e gerenciamento, enquanto que 15% era devido à complexidade do projeto. Dessa forma, a estimativa para testar o Software I foi de 4837,50 horas.

AUCP = 129 * [0,65 + (0,01 * 85)] = 193,50

Esforço = 193,50 * 20 = 3870

Esforço final = Esforço * (1 + 0,10 + 0,15) = 4837,50

Figura 3. Cálculos finais do método de Nageswaran.

Conforme dito no início desta seção, o método proposto também foi aplicado ao Software II e III da RFB. Além disso, o tempo real consumido na atividade de testes destes sistemas também foi registrado para fins de comparação com a estimativa obtida através do método de Nageswaran. A Tabela 6 mostra as estimativas de esforço obtidas para os outros dois sistemas testados pela EIVeV, bem como o tempo efetivamente consumido na atividade de testes para os três sistemas da RFB.

Tabela 6. Comparação entre o esforço estimado e o efetivamente consumido para os testes de três sistemas de software da RFB.

Nome do software	Tempo estimado (em horas)	Tempo efetivamente gasto (em horas)	Margem de erro
Software I	4837,5	381	1270 %
Software II	820	147	558 %
Software III	2475	190	1303 %

Os dados apresentados mostram claramente que as estimativas providas pelo método de Nageswaran foram ruins, mesmo se a EIVeV considerasse uma margem de erro associada à estimativa de 20%. A aplicação desta margem de erro é comum, visto que ela provê um número a mais de horas para que a equipe consiga diagnosticar e resolver problemas de qualquer natureza durante os testes, minimizando o impacto ao resultado final do trabalho. Note também que, mesmo se o fator de conversão utilizado pela equipe fosse igual a 5 (ao invés de 20, conforme sugerido no método), o erro na estimativa ainda seria superior a 160 % (em média) para os três sistemas avaliados.

Tal como foi comentado, o impacto pela escolha inadequada do fator de conversão é alto, podendo gerar um erro elevado na estimativa de esforço. Nageswaran não ofereceu sugestões de como adaptar este valor, o que dificultou a escolha de um valor adequado pela EIVeV. Além disso, a equipe percebeu ao final da execução dos testes que os cenários de uso do sistema que tratavam exceções consumiam menos tempo na execução dos testes do que os cenários de uso normais. Este detalhe indica que deveria existir uma maneira de distinguir os cenários durante a classificação dos casos de uso. A equipe também acredita que outra causa para o grande erro na estimativa é a ausência de detalhes a respeito do passo de atribuição de valores e pesos para os fatores técnicos e de ambiente de testes.

Um último detalhe não ilustrado no método e que e ajudaria nas estimativas seria dividir o tempo total de esforço obtido nas disciplinas previstas no processo de testes. Por exemplo, o tempo total obtido poderia ser dividido entre as disciplinas básicas de planejamento, projeto, execução e análise de resultados dos testes.

5. Conclusão

Este relato de experiência descreveu o método de Nageswaran [Nageswaran 2001], que utiliza dados obtidos nas especificações de casos de uso do software para obter o esforço para testes de software, e mostrou os resultados de sua aplicação prática em especificações de casos de uso de três sistemas de software da RFB.

O método avaliado é bastante simples de ser entendido teoricamente. No entanto, sua aplicação prática não teve resultados satisfatórios, uma vez que as estimativas obtidas foram muito superestimadas com relação ao tempo consumido na prática por uma EIVeV nos testes dos três sistemas da RFB. Além disso, alguns outros problemas no uso do método foram a falta de explicação de como deveriam ser classificados alguns dados de fatores técnicos e de ambiente, afora a dificuldade na adaptação de um fator de conversão muito importante para obtenção do resultado final.

Como trabalhos futuros, a equipe prevê o estudo sobre as possíveis diferenças entre cenários de uso normal e de exceção e seu impacto na estimativa dos testes, bem como a criação de definições mais precisas de como devem ser aplicados valores e pesos aos fatores técnicos e de ambiente de testes. Outras alterações que poderiam ser

avaliadas são as simplificações nas classificações de atores e de cenários, como Merrick [Merrick 2005] propõe para as estimativas de tempo de desenvolvimento.

Agradecimentos

Os autores agradecem ao CNPq e FAPESP pelo apoio parcial à realização deste trabalho, à Receita Federal do Brasil pelo apoio financeiro e estudos de caso oferecidos através do Projeto HARPIA, e à SOFIST, pelo apoio financeiro e sugestões para a elaboração do trabalho.

Referências

Aranha, E. e Borba, P. (2007a) “An Estimation Model for Test Execution Effort”. Em Proceeding of the First International Symposium on Empirical Software Engineering and Measurement, Setembro, Madri, 107-116.

Aranha E. e Borba, P. (2007b) "Empirical Studies of Test Execution Effort Estimation Based on Test Characteristics and Risk Factors." Em 2nd International Doctoral Symposium on Empirical Software Engineering, Setembro, Madri. Disponível em: http://twiki.cin.ufpe.br/twiki/pub/SPG/SoftwareEstimationModels/idoese2007-aranha_final.pdf. Último acesso realizado em 21/03/2008.

Black, R. (2006) “Factors that Influence Test Estimation”. Disponível em: http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=ART&ObjectId=5992. Último acesso realizado em 21/03/2008.

Cardoso, A. (2002) “The Test Estimation Process”, Disponível em: http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=ART&ObjectId=3510. Último acesso em realizado em 27/02/2008.

Ferens, D.V. e Gurner, R.B. (1992) “An evaluation of three function point models for estimation of software effort”. Em Proceeding of the Aerospace and Electronics Conference, 1992. NAECON 1992., Proceedings of the IEEE 1992 National, Maio, Dayton, 635-642, Volume 2.

Merrick, P. J. (2005) “Simplification of the UCPM”, Disponível em: http://www.uea.ac.uk/~a168955/effot_estimation/simplification_UCPM.html. Último acesso realizado em 19/02/2008.

Nageswaran, S. (2001) “Test Effort Estimation Using Use Case Points”. Em 14th International Internet Software Quality Week 2001, Junho, São Francisco. Disponível em: http://www.cognizant.com/html/content/cogcommunity/Test_Effort_Estimation.pdf . Último acesso realizado em 27/02/2008.

Pressman, R. S. (2004), “Software Engineering: A Practitioner’s Approach”, McGraw-Hill Professional, 6ª ed.

Schneider, G. e Winters, J. P. (1998), “Applying Use Cases”, Addison-Wesley, 1ª ed.

Veenendaal, E. e Dekkers, T. (1999) “Test Point Analysis: a Method for Test Estimation”. Em “Project Control for Software Quality”. Editado por: Rob Kusters, Arian Cowderoy, Fred Heemstra e Erik van Veenendaal, Shaker Publishing.