

Guia para Redução do Custo Total de defeitos pela aplicação de Modelos de Processo de Teste de Software

Autores

Ariane Louise Corso

Orientador

Plinio Roberto Souza Vilela

1. Introdução

Devido a uma alta demanda de insatisfações na qualidade de software, as organizações têm, hoje, a difícil tarefa de melhorar o processo de desenvolvimento de software, entregando-os com qualidade, dentro de prazos e custos controlados e previamente estabelecidos. De acordo com [8], na medida em que o emprego de sistemas de informação pela sociedade cresce ao ponto em que boa parte dos negócios depende cada vez mais de software e computadores, passa a ser de vital importância contar com software de qualidade, que fornece resultado correto quando alimentado com dados válidos e que identifica corretamente dados de entrada inválidos. A área de teste de software é fundamental para avaliação da qualidade do software desenvolvido.

Segundo [10], a confiabilidade do software é a probabilidade de um programa operar corretamente por um tempo específico em um ambiente específico. O custo total dos defeitos que permanecem no software liberado e que eventualmente se manifestam como falhas, têm sido observados com maior criticidade, por muitos autores, a fim de se fazer uma estimativa para alocar melhor o esforço ligado aos testes e também reduzir o custo esperado das falhas.

Existem alguns modelos de maturidade de teste atualmente disponíveis como: Testability Maturity Model, Software Testing Maturity Model (SW-TMM), Test Process Improvement(TPI), Teste Organization Maturity, Test Assessment Program. Estes modelos buscam deixar o processo de teste mais robusto para que os softwares se tornem mais confiáveis quando liberados para produção. O trabalho para melhoria dos processos de teste de software nas organizações é um instrumento interessante que pode auxiliar em mudanças significativas na busca pela excelência no desenvolvimento de software.

2. Objetivos

Atualmente, existe a necessidade das organizações em agilizar o processo de teste e garantir a qualidade do software entregue, aumentando a quantidade de defeitos encontrada durante o processo de desenvolvimento e procurando identificar primeiro os defeitos que poderiam ter um custo mais elevado dentro do processo de desenvolvimento, ou impactar de forma mais significativa o negócio do cliente.

A intenção deste trabalho é propor um guia de Implementação de Modelos de teste parametrizado de acordo com o custo total do defeito do software, para que haja a redução deste custo. Embora existam vários modelos de processo de teste, os modelos de maturidade de teste mais “usáveis” pelas organizações são o SW-TMM e TPI [4]. As atividades descritas nos modelos SW-TMM e TPI serão avaliadas para identificar quais se aderem melhor para encontrar, o quanto antes, os defeitos no desenvolvimento de software. O

cálculo do custo total de defeitos também será estudado e analisado para que as organizações consigam estimá-lo com maior precisão.

3. Desenvolvimento

O desenvolvimento deste trabalho envolve a definição do termo custo total de defeitos, como calcular e estimar o custo do defeito de um software baseando em fatores como: qual a influência do tipo do software desenvolvido e a fase de desenvolvimento onde o defeito foi descoberto, além do estudo dos modelos de processo de teste.

De acordo com o cenário capturado através dos estudos destes pontos de influência, conseguiremos obter o custo total do defeito, e assim, poderemos identificar quais atividades dos processos de teste SW-TMM / TPI são mais adequadas para serem aplicadas ao projeto afim de que haja a redução do custo total do defeito e a produção de um software com maior qualidade.

Custo total de defeitos

O custo de um defeito é o custo estimado para se corrigir um determinado erro encontrado no software. Este defeito, quando propagado pode gerar custos inestimáveis. Em [10] é citado que todos temos diferentes atitudes com relação à correção dos defeitos, especialmente em qual corrigir e quando fazê-lo. A escolha para fazer ou não a correção depende do tipo de produto que está sendo desenvolvido, do risco associado com a entrega do software com defeitos conhecidos ou não; o processo de desenvolvimento utilizado; e qual o custo de se corrigir o defeito, quando você realmente o faz. A junção destes fatores combinados com o grau de influência de cada um, no custo do defeito do software, pode ajudar a estimar, com maior precisão, o custo total do defeito.

Influência do tipo do software no custo do defeito

Os riscos associados a ocorrência de falhas em produção também deve ser levado em consideração quando estamos avaliando qual as atividades de teste precisam ser realizadas para que o defeito seja encontrado precocemente.

Quanto maior este risco, mais robusta deve ser a fase de teste e a metodologia utilizada.

A fase de desenvolvimento onde o defeito foi descoberto

Segundo Pressman, de acordo com estudos feitos pela indústria, as atividades do projeto introduzem entre 50% e 60% de todos os defeitos durante a fase de desenvolvimento do processo de engenharia de software. A utilização de revisões técnicas formais é vista como um “filtro” para o processo de engenharia de software,

sendo até 75% efetivas, conseguindo detectar os erros precocemente e reduzir o custo dos próximos passos nas fases de desenvolvimento e manutenção. Supondo que um erro descoberto durante a fase de projeto custe 1 unidade monetária para ser corrigido, o mesmo erro descoberto logo antes que as atividades de teste se iniciem, custará 6,5 unidades; durante os testes, 15 unidades; e após o lançamento, entre 60 e 100 unidades.

Não é simples entender o custo de se corrigir um defeito. O custo para se fazer a correção também é influenciado pela escolha do ciclo de vida e do processo de desenvolvimento e auxilia na decisão do risco de se colocar algo em produção com defeito. Entretanto, muitos não sabem o custo para se corrigir um defeito dentro de sua organização.

Rothman em[10] elaborou uma estimativa para medir este custo. Vale lembrar que esta estimativa foi elaborada quando os esforços para correções estão voltados, totalmente, para a fase de teste de sistema, onde você tem 100% das pessoas dedicadas a encontrar e corrigir eventuais defeitos, contando o número de correções efetuadas. Nesta fase você sabe quantas pessoas (desenvolvedores, testadores, e todos os envolvidos) trabalharam no projeto, e você também sabe a duração do teste de sistema. Isto permite que você calcule o custo para corrigir um defeito durante esta fase do projeto. Segue a forma onde é possível encontrar o custo médio para encontrar e corrigir um defeito:

Custo médio para se corrigir um defeito =	$\frac{(\text{Numero de pessoas} * \text{numero de dias}) * \text{custo por pessoa-dia}}{(\text{Numero de defeitos corrigidos})}$
---	---

Note que o número de defeitos encontrados não é informação suficiente para estimar o custo, e sim o número de defeitos corrigidos. A fase de detecção de defeitos é somente um primeiro passo. Localizar a falha, decidir como corrigi-la, desenvolver testes (unitários, de sistema, etc) para as correções, e procurar outros defeitos que esta correção causou é o porque o valor do acerto ser o que importa.

Considerações gerais sobre os Modelos de Maturidade de Teste

TPI – Test Process Improvement

Este modelo foi desenvolvido baseado no conhecimento prático e em experiências no desenvolvimento do processo de teste. Para organizar a eficiência dos testes, diferentes níveis de teste são usados, e cada nível endereça um certo grupo de requisitos ou especificações técnicas ou funcionais.

Os passos para melhoria do processo de teste no modelo são:

1. Determinar metas e áreas de consideração: qualidade do teste para determinar se a meta é fazer o teste mais rápido, mais barato ou com uma cobertura maior? Quais os processos de teste são melhores na necessidade de melhoria, quanto tempo os processos de melhoria podem durar e com qual esforço?

2. Determinar a situação corrente: Pontos fortes e fracos da situação corrente são determinados.
3. Determinar a situação requerida: as ações necessárias para se chegar a situação requerida são determinadas.
4. Implementar mudanças: a Implementação é feita de acordo com um plano e situações são checadas para verificar que as metas foram alcançadas.

No modelo TPI é observado um processo de teste através dos diferentes pontos de vista, como: o uso de ferramentas de teste, técnicas de especificação de teste e relatórios. Estas são chamadas de áreas chaves no modelo TPI, onde cada área é classificada dentro de um nível de maturidade. Existem algumas dependências entre as áreas chave e os níveis. Por isso, uma matriz de maturidade de teste é usada. As classificações dos níveis são feitas através de pontos de checagem.

A metodologia TMap, para teste estrutural, é usada como base para o modelo TPI. Ela contém quatro alicerces: o ciclo de vida (L) das atividades relacionadas ao ciclo de desenvolvimento, boa Organização (O), a correta infra-estrutura e ferramentas(I), e técnicas usáveis para realização das atividades (T).

Dentro destes alicerces um total de 20 áreas chaves podem ser reconhecidas para o modelo TPI. Estas áreas chaves cobrem o processo total de teste. Para permitir uma percepção no estado das áreas chaves, o modelo as fornece com níveis ascendentes, geralmente de A a D. Em média existem de 3 a 4 níveis por área chave. Cada nível mais alto é melhor do que o anterior em termos de tempo, custo e qualidade. A tabela 1 representa os diferentes níveis das áreas chave, incluindo os alicerces.

TABELA 1

Pontos de checagem são utilizados para determinar os requisitos dos certos níveis. Baseados nos pontos de checagem um processo de teste pode ser avaliado e para cada área chave o nível apropriado pode ser estabelecido. Os pontos de checagem são também cumulativos: para classificar para o nível B, o processo de teste precisa responder positivamente os pontos de checagem tanto do nível B como do nível A.

Devido ao grau de importância diferenciada entre cada área chave e nível associado, foi criada a Matriz de Maturidade de Teste que relaciona cada uma delas. A estrutura da Matriz de Maturidade de Teste é descrita na tabela 2.

TABELA 2

As escalas da maturidade do teste pode ser divididas dentro de 3 categorias: Controlada (de 1 a 5), eficiente (de 6 a 10) e otimizada (de 11 a 13).

SW-TMM

O SW-TMM foi projetado pela Dra. Ilene Burnstein do Instituto de Tecnologia de Illinois e seus associados, para ser um companheiro do SW-CMM, endereçando importantes questões de gerentes de testes, especialistas de teste e certificação da qualidade de software. O SW-TMM é uma ferramenta de melhoria do processo de teste e seu uso não somente ajuda a documentar o nível corrente, mas fornece um guia para fazer as melhorias necessárias para o processo.

Os componentes para suportar os objetivos do desenvolvimento TMM são:

- Um conjunto de níveis que definem uma hierarquia na maturidade de teste.
- Um conjunto de objetivos de maturidade para cada nível
- Um modelo de avaliação que consiste em três componentes: um conjunto de questões de objetivos relacionados de maturidade projetados para avaliar a maturidade do processo de teste, um programa de treinamento para selecionar e instruir o time de avaliação, um método de avaliação que permite a organização fazer uma auto avaliação.

O SW-TMM contém 5 níveis de maturidade, com objetivos e sub-objetivos para cada nível.

Nível 1: Inicial - onde o processo de teste é caótico.

Nível 2: Definição de Fase - o teste é uma atividade planejada e é separado do “debugging” e é definido como uma fase que segue a codificação.

Nível 3: Integração – o teste não é mais uma fase que segue a codificação, ele está integrado no ciclo de vida do software. Existe uma organização do teste e testar é reconhecido como uma atividade profissional.

Nível 4: Gerenciamento e Medições – teste é um processo medido e quantificado, revisões de todas as fases do processo de desenvolvimento são reconhecidas como teste e atividades de controle de qualidade.

Nível 5: Otimização e Prevenção de Falhas e Controle de Qualidade – devido a intra-estrutura fornecida pela realização dos níveis de maturidade dos 1 ao 4, o processo de teste pode ser definido, com custo gerenciado e a eficácia monitorada.

A seguinte figura descreve os objetivos de maturidade para todos os nível, exceto par o nível 1, o qual não possui um objetivo de maturidade.

FIGURA 1

Cada um dos objetivos de maturidade descritos, possui sub-objetivos que auxiliam a suportá-los.

4. Resultados

Este estudo que está sendo realizado busca intensificar a preocupação com relação à melhoria da qualidade de softwares desenvolvidos, agregando questões sobre o custo do defeito do software e fornecendo algumas diretrizes que podem auxiliar na redução do custo total de defeitos encontrados.

Os fatores que influenciam direta ou indiretamente no custo do defeito, serão cuidadosamente analisados, ou até adicionados outros fatores, para que sejam medidos com o grau de importância correto com relação ao custo do defeito. Isso fará com que o resultado que se espera com a utilização do guia seja mais significativo.

Acredita-se que, com a utilização disciplinada do guia, aumente a detecção precoce de defeitos no processo de desenvolvimento de software, melhorando sua qualidade e reduzindo seu custo de manutenção.

5. Considerações Finais

Acredita-se que a utilização somente deste guia no desenvolvimento de software não é suficiente para garantir sua qualidade total. Outras fases do ciclo de vida do desenvolvimento também devem ser estudadas, como por exemplo levantamento e abstração de requisitos, para que haja uma melhora ainda mais significativa na qualidade do software. Porém, este guia é um ponto de apoio para que haja maior credibilidade na entrega do software.

Referências Bibliográficas

[1] ANDERSIN, J. TPI – A model for Test Process Improvement. Seminar on Quality Models for Software Engineering Department of Computer Science, University of Helsinki, Oct. 2004.

[2] BURNSTEIN, I., SUWANNASART T., CARLSON C. Developing a Testing Maturity Model: Part I. CrossTalk, Aug. 1996

[3] BURNSTEIN, I., SUWANNASART T., CARLSON C. Developing a Testing Maturity Model: Part II. CrossTalk, Sep. 1996

[4] STAAB, T. Using SW-TMM to Improve the Testing Processes. Wing Ridge International, Nov. 2002.

[5] VEENENDAAL, E., SWINKELS R. Guidelines for Testing Maturity – Part 1: The TMM Model. Professional Tester, Vol 3, Nro 1, Mar. 2002.

[6] VEENENDAAL E. Guidelines for Testing Maturity – Part 2: Test Maturity Model level 2. Professional Tester, Vol 3, Nro 2, Jun. 2002.

[7] PRESSMAN, R. Engenharia de Software. 3º Edição, p.737 – 876, 1995.

[8] CRESPO, A., SILVA, O., BORGES C., SALVIANO C. JINO M. Uma Metodologia para Teste de Software no Contexto da Melhoria de Processo.

[9] NTAFOS, S. Testing and the Cost of Field Failures. ISSRE, 1999.

[10] ROTHMAN, J. What Does It Cost to Fix a Defect?, Feb.2002, Disponível em <<http://www.stickyminds.com/sitewide.asp?Function=edetail&ObjectType=COL&ObjectId=3223>>, Acesso em 18.May.2006.

Anexos



Level	A	B	C	D
Key area Cornerstone				
Test strategy L	Strategy for single high-level test	Combined strategy for high-level tests	Combined strategy for high-level tests and low-level tests or evaluation	Combined strategy for all test and evaluation levels
Life-cycle model L	Planning, Specification, Execution	Planning, Preparation, Specification, Execution, Completion		
Moment of involvement L	Completion of test basis	Start of test basis	Start of requirements definition	Project initiation
Estimating and planning T	Substantiated estimating and planning	Statistically substantiated estimating and planning		
Test specification techniques T	Informal techniques	Formal techniques		
Static test techniques T	Inspection of test basis	Checklists		
Metrics T	Project metrics (product)	Project metrics (process)	System metrics	Organization metrics (>1 systems)
Test tools I	Planning and control tools	Execution and analysis tools	Extensive automation of the test process	
Test environment I	Managed and controlled environment	Testing in most suitable environment	Environment on call	
Office environment I	Adequate and timely office environment			
Commitment and motivation O	Assignment of budget and time	Testing integrated in project organization	Test engineering	
Test functions and training O	Test manager and testers	(Formal) Methodical, technical and functional support, management	Formal internal Quality Assurance	
Scope of methodology O	Project specific	Organization generic	Organization optimizing (R&D)	
Communication O	Internal communication	Project communication (defect, change control)	Communication within the organization about the quality of the test process	
Reporting O	Defects	Progress (status of test and product), activities (costs and time, milestones), defects with priorities	Risks and recommendations, substantiated with metrics	Recommendations have a Software Process Improvement character
Defect management O	Internal defect management	Extended defect management with flexible reporting facilities	Project defect management	
Testware management O	Internal testware management	External management of test basis and test object	Reusable testware	Traceability system requirements to test cases
Test process management O	Planning and execution	Planning, execution, monitoring, and adjusting	Monitoring and adjusting within organization	
Evaluation (all)	Evaluation techniques	Evaluation strategy		
Low-level testing (all)	Low-level test life-cycle: planning, specification and execution	White-box techniques	Low-level test strategy	

Key area	Scale	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Test strategy			A					B				C		D	
Life-cycle model			A			B									
Moment of involvement				A				B				C		D	
Estimating and planning					A							B			
Test specification techniques			A		B										
Static test techniques						A		B							
Metrics							A			B			C		D
Test automation					A				B			C			
Test environment					A				B						C
Office environment					A										
Commitment and motivation			A				B						C		
Test functions and training					A			B			C				
Scope of methodology						A						B			C
Communication				A		B							C		
Reporting			A			B		C						D	
Defect management			A				B		C						
Testware management				A			B				C				D
Test process management			A		B								C		
Evaluation								A			B				
Low-level testing						A		B		C					