MANUTENIBILIDADE DE SOFTWARE

VALÉRIO BRUSAMOLIN

Instituto Científico de Ensino Superior e Pesquisa – ICESP

http://brusamolin.sites.uol.com.br e-mail: brusamolin@yahoo.com

RESUMO

Manutenibilidade é uma das características de qualidade de software, determinando o grau de facilidade com que o mesmo pode ser corrigido ou aperfeiçoado. Um software com alto índice de manutenibilidade necessita de menos tempo e pessoas para ser modificado. Este artigo busca identificar os fatores que afetam a manutenibilidade de um artefato de software e suas métricas.

ABSTRACT

Maintainability is one of the software characteristics of quality, determining how easy it can be corrected or improved. A software with a high maintainability grade needs less time and people to be modified. This article aims to identify factors that affect the maintainability of a software artifact and it metrics.

Keywords: software maintainability, software quality, maintainability metrics, software maintenance.

1 INTRODUÇÃO

A atividade de manutenção de software é dispendiosa e consome tanto ou mais recursos do que o desenvolvimento: entre 40 e 60 por cento do custo total de um projeto (COLEMAN, ASH, 1994). Entretanto, as equipes de desenvolvimento são premidas por cronogramas e orçamentos, descartando trabalhos que poderiam reduzir custos futuros de manutenção. No próprio levantamento de requisitos, não é prática usual se especificar ou verificar itens de qualidade que facilitem futuras manutenções, sejam elas corretivas, adaptativas ou perfectivas.

Este trabalho visa relacionar os conceitos, práticas e idéias que visem a produção de software com maior índice de manutenibilidade, estabelecendo um ponto de partida para a investigação mais aprofundada de como atuar efetivamente no ciclo de desenvolvimento de software de forma a facilitar as necessárias correções e evoluções futuras.

2 CICLOS DE VIDA DO SOFTWARE

A Norma NBR ISO/IEC 12207 - Processos do Ciclo de Vida do *Software* - tem como principal objetivo fornecer uma estrutura comum para que os envolvidos com o desenvolvimento de *software* utilizem uma mesma linguagem. Para esse fim, estabelece um ciclo de vida padrão composto pelos seguintes processos fundamentais:

a) Aquisição;
b) Fornecimento;
c) Desenvolvimento;
d) Operação;
e) Manutenção:

Segundo a mesma norma, o processo de manutenção é ativado quando o produto de software é submetido a modificações no código e na documentação associada devido a um problema, ou à necessidade de melhoria ou adaptação. O objetivo é modificar um produto de software existente, preservando a sua integridade.

Thomas Pigoski (1996, p.37-50) é de parecer que este e outros ciclos de vida levam a uma interpretação equivocada do processo de manutenção, pois tem início apenas após o produto já estar pronto e em produção. Não se concebe a participação do pessoal de manutenção nos processos iniciais do ciclo.

Pigoski sugere que o processo de manutenção inicie juntamente com o desenvolvimento. Dessa forma, especialistas em manutenção poderiam atuar de forma a se obter um produto com

arquitetura propícia a reparos mais rápidos e baratos.

3 CONCEITO DE MANUTENÇÃO

Existem várias definições para manutenção de software. Pigoski sintetiza em uma definição a sua opinião de como deveria ser o processo de manutenção: “Manutenção de software é a totalidade de atividades necessárias para prover, minimizando o custo, suporte a um sistema de software. As atividades são executadas tanto nos estágios pré-entrega quanto nos pós-entrega. As atividades de pré-entrega incluem o planejamento para entrada em operação, suportabilidade e definição de logística. As atividades de pós-entrega incluem modificação do software, treinamento e operação de um help-desk” (PIGOSKI, 1996, p.46)

4 TIPOS DE MANUTENÇÃO

Segundo E. B. Swanson (1976), existem três tipos de manutenção:

a)Corretiva: modificações necessárias para corrigir erros;
b)Adaptativa: qualquer esforço desencadeado como resultado de modificações do meio ambiente em que o software deve operar;
c)Perfectiva: todas as mudanças feitas para atender a evolução das necessidades do usuário.

A figura 1 mostra os percentuais de esforço aplicados em cada tipo de manutenção. É equivocada a idéia de que um sistema ideal, com nenhum erro, não teria manutenção. Os requisitos do usuário evoluem, tentando adquirir vantagens em relação a concorrentes, reduzir custos operacionais ou simplesmente se adequar a mudanças na legislação e processos.

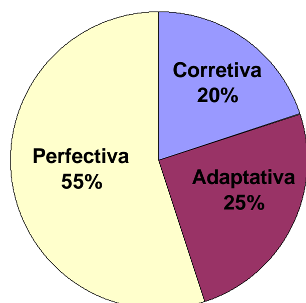


Figura 1 - Percentuais de esforço de manutenção (PIGOSKI, 1996)

5 MANUTENIBILIDADE

Manutenibilidade de software diz respeito à facilidade com que o mesmo pode ser modificado para satisfazer requisitos do usuário ou ser corrigido quando deficiências são detectadas (PIGOSKI, 1996).

O IEEE (1993), estabelece que manutenibilidade é a facilidade com que um sistema de software ou componente pode ser modificado para corrigir falhas, melhorar performance ou outros atributos, ou adaptado para uma mudança de ambiente.

Se os softwares forem manuteníveis, diminui a demanda por desenvolvimento, na medida que os softwares atuais podem evoluir para atender novas necessidades. Por outro lado, aumentam as solicitações de manutenção (GLASS, 1993).

6 MANUTENIBILIDADE E QUALIDADE

A norma ISO/IEC 9126-1 (ISO/IEC 9126-1) - modelo de qualidade, classifica os atributos de qualidade de *software* em seis características. Uma delas é a manutenibilidade, que é definida como o esforço necessário para fazer modificações específicas no software. A referida norma desdobra a manutenibilidade ainda, em cinco subcaracterísticas (ROCHA et Al, 2001, p. 117), (MARTINS, VOLPI, 2002):

a)Analisabilidade: atributos do software que evidenciam o esforço necessário para deiagnosticar deficiências ou causas de falhas, ou para identificar partes a serem modificadas;

b)Modificabilidade: atributos do software que evidenciam o esforço necessário para modificá-lo,remover seus defeitos ou adaptá-lo a mudanças ambientais.

c)Estabilidade: atributos do software que evidenciam o risco de efeitos inesperados ocasionados por modificações.

d)Testabilidade: atributos do software que evidenciam o esforço necessário para validar o software modificado.

e)Conformidade: atributos do software que o tornam consonantes com padrões ou convenções relacionadas àportabilidade.

7 FATORES QUE IMPACTAM NA MANUTENIBILIDADE

a) Arquitetura

O *design* da arquitetura influencia mais na manutenibilidade do que o *design* do algoritmo (ROMBACH, 1990, p. 22).

A Companhia de informática do Paraná, no intuito de definir recomendações para o seu processo de desenvolvimento de sistemas, elaborou várias versões de um mesmo software, com arquiteturas diferentes e verificando os resultados de qualidade obtidos (MARTINS, VOLPI, 2002).

A experiência da empresa paranaense baseou-se no processo unificado, na UML e na tecnologia de componentes da Microsoft (COM). Foram utilizadas as mesmas tecnologias, variando-se apenas a arquitetura em seis experimentos, que iniciaram com uma solução em duas camadas até várias arquiteturas de *n* camadas *multithreadeds*.

Como resultado, verificou-se que o investimento em arquitetura de software aprimora, entre outras características de qualidade, a manutenibilidade. A divisão da aplicação em componentes e dos componentes em classes tornou cada unidade de código bem menor do que em uma abordagem monolítica, de modo que o esforço para modificar o código foi pequeno e o potencial de inclusão de erros nessa modificação foi reduzido.

b) Tecnologia

Uma das preocupações que o gerente deve ter é a de verificar se a tecnologia que vai empregar na implementação do software permite um bom grau de manutenibilidade, pois a fácil proliferação de programas pode se tornar mais tarde, um pesadelo de manutenção.

A hipótese de que softwares que utilizam tecnologias orientadas a objetos possuam alta manutenibilidade foi objeto de estudo por Sallye M. Henry e Mathew Humphey (HENRY, HUMPHREY, 1990). O software produzido com linguagem orientada a objetos necessita menos modificações no código fonte, em locais específicos. Em grandes sistemas, que são manutenidos por longo tempo, o impacto positivo do uso de linguagens orientadas a objetos é grande, pois modificações mais localizadas levam a menor degradação do código. A degradação ocorre em função do número de linhas acrescidas ou modificadas (LAND, 2003); logo, manutenção com feita com menos código significa menos entropia. Entropia pode ser definida como a diferença entre o todo e suas partes.

c) Documentação

Quando documentação ou especificações de design do software não estão disponíveis, o mantenedor tem de investir muito tempo para compreender o produto antes de modificá-lo (PIGOSKI, 1996, p. 276).

d) Compreensibilidade do Programa

Existem estimativas de que os programadores ficam entre 47% e 62% do tempo de trabalho tentando entender a documentação e lógica dos programas (PIGOSKI, 1996, p.276). Portanto, a compreensibilidade dos programas deve ser aumentada se desejarmos diminuir os custos de manutenção.

O uso de abreviaturas na declaração de variáveis dificulta a compreensibilidade de programas fonte (LAITINEN et al, 1997).

8 MÉTRICAS DE MANUTENIBILIDADE

Para controlar a manutenibilidade de um software, deve-se saber como medi-la. Para isso existem as métricas, que auxiliam na verificação do software produzido. As métricas que aferem a manutenibilidade verificam a complexidade do software (BANDI et Al, 2003).

9 MÉTRICAS DE MANUTENIBILIDADE A NÍVEL DE CÓDIGO

Existem muitas métricas de manutenibilidade a nível de código. A dificuldade não reside em se encontrá-las, mas sim em selecionar as mais adequadas a determinado projeto. As métricas podem ser utilizadas isoladamente ou fazer parte de métricas híbridas.

10 MÉTRICAS DE COMPLEXIDADE DE HALSTED

Halsted (HALSTEAD, 1977) desenvolveu métricas de complexidade que ainda hoje são utilizadas para se derivar a manutenibilidade de software.

As métricas de Halsted são baseadas em quatro números escalares derivados diretamente de um programa fonte(SEI, 2003):

n1=número de operadores distintos
n2=número de operandos distintos
N1=número total de operadores
N2=número total de operandos

A partir destes números, cinco métricas são derivadas, como consta da tabela1.

Tabela 1 – Métricas de Halsted

Métrica	Símbolo	Fórmula
Tamanho do Programa	N	N=N1+N2
Vocabulário do Programa	n	n=n1+n2
Volume	V	V=N*(LOG2 n)
Dificuldade	D	D=(n1/2)* (N2/n2)
Esforço	E	E=D * V

11 MÉTRICAS DE MANUTENIBILIDADE A NÍVEL DE DESIGN

As métricas de código ignoram as interdependências entre módulos, assumindo que cada componente é uma entidade separada. As métricas de design de arquitetura tentam quantificar o nível de interação entre módulos, partindo do pressuposto que as interdependências contribuem para a complexidade geral do software (HENRY, SELIG, 1990).

H. Dieter Rombach, faz uma distinção entre design da arquitetura, que é de alto nível, e design algorítmico, de baixo nível. O design da arquitetura impacta mais na manutenibilidade do que *design* do algoritmo (ROMBACH, 1990, p. 22), e sua experiência demonstra que não vale a pena medir o código: maior ganho é obtido ao investir na arquitetura, e o fator mais importante é a experiência e conhecimento do *designer*.

As métricas de complexidade a nível de design que afetam a manutenibilidade mais citadas na literatura são:

a) Complexidade Ciclomática

Desenvolvida por McCabe para indicar a testabilidade e manutenibilidade de software, medindo o número de caminhos linearmente independentes do programa (ou método). Para determinar os caminhos, representa-se o método como um grafo fortemente conectado com uma única entrada e uma única saída. Os nodos são blocos seqüenciais de código, e as arestas são decisões que causam uma ramificação. A complexidade é dada por:

$$CC = E - N + 2$$

Onde E = número de arestas e N = número de nodos

b) Fluxo de Informação

Definido por Sallie Henry e Denis Kafura (HENRY et al., 1981), o valor da métrica é dado pelo quadrado da multiplicação do fan-in de um módulo pelo seu fan-out.

$$Cc = (\text{fan-in} * \text{fan-out})^2$$

Fan-in é a quantidade de fluxos de informações que entram no programa; fan-out é a quantidade de fluxos que saem do programa. No cálculo também devem ser consideradas as variáveis globais utilizadas e modificadas pelo programa ou método.

12 MÉTRICAS HÍBRIDAS

O Software Engineering Institute (SEI, 2003) adotou uma técnica específica:o Maintainability Index (MI), que foi desenvolvido por Paul Oman (OMAN, HAGEMEISTER, 1994). Artigo de Don Coleman (COLEMAN et Al, 1995), cita e explica a como os elementos do MI foram calibrados, validando a sua utilização para a indústria de software.

Neste índice, a manutenibilidade é calculada utilizando uma combinação de métricas. O MI de um conjunto de programas é dado pelo seguinte polinômio:

MI=171 - 5.2 * ln(mediaV) - 0.23 * mediaV(g')
- 16.2 * ln (mediaLDC) + 50 * sen (sqrt(2.4 * perCM))

Onde:

mediaV = Média do Volume de Halstead por módulo
mediaV(g') = Média estendida da complexidade ciclomática por módulo

mediaLDC = Média de linhas de código (LDC) por módulo
perCM = Média percentual de linhas de comentários por módulo (opcional)

A manutenibilidade é diretamente proporcional ao MI, ou seja, quanto maior o índice, mais manutenível é o programa. Um índice de 6, por exemplo, é de manutenção quase impossível. Já um índice de 70 é muito bom.

13 PRÁTICAS DE MANUTENIBILIDADE

Thomas Pigoski nos dá uma pista sobre como produzir sistemas manuteníveis: pensar em manutenção já no levantamento de requisitos (PIGOSKI, 1996, p.46). O mesmo autor explica que os processos de desenvolvimento e manutenção devem evoluir e se integrar de alguma forma, sugerindo algumas práticas que levam à manutenibilidade:

- Revisões de averiguação;
- Caminhamentos estruturados;
- Uso de design orientado a objetos;
- Assegurar que cada linha de código tenha no máximo uma declaração;
- Assegurar que comentários tenham informação útil;
- Assegurar que programação "esotérica" seja evitada;
- Empregar convenções de programação;
- Usar definições de dados comuns;
- Estabelecer padrões para desenvolvimento de procedimentos e documentos do sistema;
- Registrar o processo de desenvolvimento explicando a filosofia de desenvolvimento e o processo decisório;
- Estimular a simplicidade;
- Estudar possíveis mudanças futuras e aperfeiçoamentos;
- Medir a complexidade dos componentes do sistema;
- Registrar os pontos fracos do sistema e pontos problemáticos;
- Estabelecer critérios de aceitação para avaliar a qualidade do software, com particular atenção na qualidade da manutenibilidade;
- Previsão de falhas.

Os objetivos de qualidade são atingidos atuando-se nos processos e nas métricas. Somente o uso das métricas não garante o aumento da manutenibilidade, pois deve ser definido quando serão utilizadas, quem fará a avaliação, quais serão as medidas corretivas, quais serão os índices aceitáveis ou não.

14 CONCLUSÃO

A manuteniblidade de um software é desejável que pode ser verificada.

Como os processos de desenvolvimento atuais não estimulam a preocupação com a manutenibilidade, e devem ser aperfeiçoados para melhorar a qualidade dos produtos obtidos.

As práticas de manutenibilidade podem ser inseridas na especificação de um processo de desenvolvimento, e a qualidade do produto pode ser aferida através das métricas apresentadas.

Futuros trabalhos podem investigar o aumento da manuteniblidade obtido com a adaptação de um processo de desenvolvimento de software.

15 REFERÊNCIAS BIBLIOGRÁFICAS

BANDI, Rajendra K, VAISHNAV, Vijay K, TURK, Daniel E. Predicting Maintenance Perfformance Using Object Oriented Design Complexity Metrics. IEEE, 2003.

COLEMAN, Don, ASH, Dan. Using Metrics to Evaluate Software System Maintainability. IEEE, 1994.

COLEMAN, Don, LOWTHER, Bruce, OMAN, Paul. The application of Software Maintainability Models in Industrial Software Systems. J. Systems Software, 1995; 29:3-16;

GLASS, Robert L. Editor's Corner – Which do You Think? Modern Methods Will Lead to Less Maintenance, or More?. J. Systems Software. 1993; 23:209-210.

HENRY, Sallie, SELIG, Calvin. Predicting Source-Code Complexity at Design Stage. IEEE, 1990.

HENRY, Sallie M, HUMPHEY, Matthew. A controlled Experiment to Evaluate Maintainability of Object-Oriented Software.

IEEE, 1990.

LAND, Rikard. Software Deterioration And Maintainability – A Model Proposal. Mälardalen University. 2003.

LI, Wei, HENRY, Sallie. Maintenance Metrics for the Object Oriented Paradigm. IEEE, 1993.

MARTINS, Vidal, VOLPI, Lisiane M. Influência da Arquitetura na Qualidade do Software. Bate Byte, 2002.

PALM, D. Jeffrey, ANDERSON, Keneth M., LIEBERHERR, Karl M. Investigating the Relationshio Between VioLations of the Law of Demeter and Software Maintaninability.

PIGOSKI, Thomas M*. Pratical Software Maintenance : Best Practices for Managing Your Software Investment.* Wiley Computer Publishing, 1996.

DUNKE, Reiner R., FOLTIN, Erik. Metrics-based Evaluation of Object-Oriented Software Developmente Metrics. University of Magdeburg, 1996.

ROCHA, Ana R. C, MALDONADO, José Carlos, WEBER, Kival Chaves. Qualidade de Software – Teoria e Prática. Prentice Hall, 2001.

ROMBACH, H. Dieter. Design Measurement: Some Lessons Learned. IEEE, 1990.

SEI Software Technology Review, Maintainability Index Technique for Measuring Program Maintainability, http://www.sei.cmu.edu/str/descriptions/mitmpm.html, acessado em 10/11/2003.

STAVRINOUDIS, Dimitris, XENOS, Michalis, CHRISTODOULAKIS, Dimitris. Relation Between Software Metrics and Maintainability. 1999.

BRIAND, Lionel C., BUNSE, Christian, DALY, John. A Controlled Experiment for Evaluating Quality Guidelines on the Maintainability of Object-Oriented Designs. IEEE, 2001.

CHIDAMBER, Shyam R., KEMERER, Chris F. A metrics Suite for Object Oriented Design. MIT Sloam School of Management, 1993.

DEKLAVA, S. M. The Influence of the Information System Development Approach on Maintenance. MIS, 1992.

FENTON, Norman, KRAUSE, Paul, NEIL, Martin. Software Measurement: Uncertainty and Causal Modelling. 2001.

HALSTEAD, Maurice H. Elements of Software Science, Operating and Programming Systems Series. Vol. 7. Elsevier, 1977.

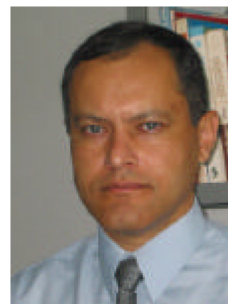
HARRISON, R., COUNSELL, S., NITHI, R. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. J. Systems and Software, 2000.

HENRY, Sallie, KAFURA, Denis. Software Structure Metrics Based on Information Flow. IEE Transactions on Software Engineering. 1981.

SAKKINEN, M. Comments on “the Law of Demeter” and C++. ACM SIGPLAN Notices, New York, v.23, n.12, p.38-44, Dec. 1988.

ZHUO, Fang, LOWTER, Bruce, OMAN, Paul, HAGEMEISTER, Jack. Constructing and Testing Software Maintainability Assessment Models. IEEE, 1993.

16 BIOGRAFIA



Valério Brusamolin, especialista em Desenvolvimento de Sistemas, é professor do Instituto Científico de Ensino Superior e Pesquisa, nas disciplinas de Análise Orientada a Objetos e Programação Orientada a Objetos.