E-Revista Facitec
Editorial, Artigos, Resumos e Resenhas
www.facitec.br/erevista

MANUTENÇÃO DE SOFTWARE & SOFTWARE LIVRE: REFLEXÕES

SOFTWARE MAINTENANCE & FREE SOFTWARE: REFLECTIONS

Por:

Eduardo Quesado Filgueiras
Otaviano Cavalcanti Wanderley Neto

E-Revista Facitec, v.1 n.1, Art.5, março. 2007.

http://www.facitec.br/erevista/index.php?option=com_content&task=view&id=9&Itemid=2

MANUTENÇÃO DE SOFTWARE & SOFTWARE LIVRE: REFLEXÕES

SOFTWARE MAINTENANCE & FREE SOFTWARE: REFLECTIONS

Resumo

Este artigo discute os elementos necessários para se garantir a operação continuada de software, utilizando ferramentas de código aberto como alternativa para a sua manutenção, especialmente para o desenvolvido, visando à operação em ambiente governamental, adaptando a fase de manutenção no ciclo de vida de software à nova realidade que emprega software livre, considerando principalmente a transição do contexto de desenvolvimento para o de produção.

Palavras-chave: Manutenção de Software, Software livre, ferramentas de código aberto

Abstract

This paper discuss the necessary elements to guarantee continuous operation of software, using open source tools as an alternative for its maintenance, specially on government environments, adapting the maintenance phase of the software life cycle to the new reality of free software deployment, chiefly considering the transition from development to production.

Key words: software maintenance, free software, open source tools

INTRODUÇÃO: DESENVOLVIMENTO UTILIZANDO SOFTWARE LIVRE

A migração dos ambientes de desenvolvimento de software em âmbito governamental e mesmo corporativo para software livre é uma realidade (APPLEWHITE, 2003). Esta ação visa reduzir custo total de propriedade, otimização de emprego de recursos públicos e fomento à obtenção de conhecimento. Outro fator importante, que é freqüentemente desconsiderado quanto à adoção de software livre na administração pública, é que seu uso elimina os demorados processos administrativos burocráticos necessários à aquisição de ferramentas proprietárias. Estas são limitadas por restrições de orçamento, pela demora na obtenção de autorizações de aquisição e pela grande formalização processual, em flagrante contraste com o que ocorre em relação ao software livre, que pode ser distribuído instantaneamente de forma legal via Internet (KSHETRI, 2004).

Este entendimento é viável ao se considerar a consolidação de ferramentas de desenvolvimento e ambientes de produção em software livre que utilizem tecnologias multiplataforma, como J2EE ou CORBA, construídos com JAVA e XML, operando *web services* ou mesmo *desktop*. Esta abordagem permite interoperabilidade e migração gradativa do ambiente dos clientes, sem interrupção de serviços. Ademais, não é mais apenas uma tendência, mas um fato a adoção de gerenciadores de projeto, de banco de dados (SGDB) e de versão de software, de ferramentas que se utilizam de engenharia de software auxiliada por computador (CASE), além de sistemas operacionais, compiladores e ambientes integrados de desenvolvimento totalmente baseados em software livre, sem perda de funcionalidades disponíveis em ambientes proprietários, nem comprometimento da qualidade (FILGUEIRAS, 2003).

Outrossim, é sabido que, independentemente do modelo de desenvolvimento, a atividade de manutenção de software é a mais longa e complexa das atividades do ciclo de vida de software (PRESSMAN, 2000),

porém extremamente necessária para garantir sua longevidade, evitando obsolescência prematura, já que, embora software não seja objeto de desgaste por sua natureza abstrata, esta mesma natureza permite que diversos fatores ligados à imaterialidade tornem o software extremamente sensível a mudanças no ambiente, nos requisitos dos usuários e nas características essenciais de qualidade.

A manutenção necessita, portanto, ser endereçada, a fim de garantir que não exista solução de continuidade da transição entre a fase de desenvolvimento e a produção efetiva, permitindo o retorno adequado do investimento financeiro e tecnológico sob a forma de produtos que atendam aos anseios da sociedade. Tal fato ocorrerá ao ser evitada a paralisação da operação do software por erros, falhas ou defeitos não evidenciados no desenvolvimento (o que é esperado pela natureza do software, que por definição sempre os possui), ou pela evolução de requisitos dos usuários. Portanto, pode-se considerar que o escopo da manutenção de software é mais abrangente, com mais atividades para controlar e rastrear que o processo de desenvolvimento.

Dessa forma, ainda que sejam implantados ferramentas e modelos baseados em software livre para desenvolvimento de software, é necessário compreender e adotar princípios gerais de manutenção de software, adaptando-os para a nova realidade, a fim de garantir as vantagens que a liberdade de expressão e liberação de conhecimento proporcionam, sempre observando a maturidade, a longevidade e a flexibilidade das ferramentas adotadas como características determinantes (NORRIS, 2004).

MANUTENÇÃO DE SOFTWARE – CARACTERÍSTICAS E CATEGORIAS

Considerando que manutenção de software pode ser definida como "a totalidade de atividades que são necessárias para se prover suporte com equilíbrio de custos" (BOURQUE, 1988), o escopo do trabalho a ser realizado normalmente se divide em duas áreas: *manutenção de software*

propriamente dita, de acordo com os procedimentos contidos no Software Engineering Book of Knowledge (SWEBOK) (BOURQUE; DUPUIS, 2004); e a necessária *auditoria de configuração*, o controle sistêmico sobre as ações de manutenção que segue os padrões de qualidade da norma IEEE 1028-1997, "IEEE Standard for Software Reviews" (1997), e que será discutido neste artigo após a reflexão inicial mais abrangente sobre relacionamento entre software livre e a manutenção de software apresentada pelo SWEBOK.

A manutenção de software, de acordo com o SWEBOK, se divide, classicamente, em duas categorias. A primeira é a chamada **manutenção corretiva**, que se preocupa em *atividades corretivas* propriamente ditas, ou seja, modificações sob demanda no software para corrigir problemas descobertos na sua operação; e *atividades preventivas* realizadas para detectar e corrigir falhas latentes antes que as mesmas se tornem problemas de fato. A segunda é a **manutenção evolutiva**, que executa atividades *adaptativas*, segundo as quais as modificações no software são realizadas para manter o produto utilizável no ambiente dinâmico dos usuários, sujeito à constante evolução, e *perfectivas,* para melhorar desempenho e manutenibilidade.

MANUTENÇÃO CORRETIVA

Conforme apresentado, e embora possam ser diferenciadas, as atividades de manutenção corretiva e preventiva são normalmente agrupadas e conhecidas como manutenção corretiva. A atividade de manutenção corretiva, portanto, deve ser entendida como aquela realizada sob demanda, em decorrência de falhas observadas, causadas por erros e defeitos no software. Conceitualmente, é importante compreender que, por sua natureza abstrata, a realização de testes formais durante o processo de desenvolvimento apenas comprova o aforismo que a realização de teste pode ser utilizada para mostrar a

presença de fatores causadores de falhas (*bugs*), mas nunca comprovar sua ausência. A razão óbvia é que a assunção da testabilidade absoluta não é factível em software desenvolvido para operar no mundo real, pois não existe uma forma de eliminar as dependências incontroláveis que são criadas no momento em que se desenvolve código. Dessa forma, considerando que não existe critério teórico para determinar definitivamente quais são as entradas causadoras de falhas, já que a identificação inequívoca é impossível, é importante a existência de um grupo dedicado a resolver os problemas de software à medida que os mesmos se apresentem.

Em conseqüência, entende-se que o escopo do trabalho de manutenção corretiva deve conter correção dos defeitos, erros e falhas encontrados no software, eliminação de problemas de configuração encontrados, reinstalação de sistemas com problemas de configuração, emissão sazonal de novas versões do software, combinação (*merge*) de versões diferentes de configuração e software existentes, bem como a criação de versões paralelas (*branches*) de software e/ou configuração para teste e correção de problemas ou necessidades particulares de usuários.

A natureza da manutenção preventiva é comumente associada às áreas em que atividades de missão crítica estão envolvidas, onde segurança é fator determinante. Dessa forma, entende-se que estas atividades devem ser realizadas no contexto de avaliação contínua das interfaces internas e externas do sistema, a fim de evitar interrupções indesejadas no mesmo, considerando, portanto:

a) o processamento e a apresentação de dados oriundos de subsistemas que operem em tempo real;
b) os mecanismos de comunicação entre os subsistemas;
c) a definição e execução de políticas de limpeza (*purging*) do banco de dados;
d) a definição e execução de políticas de organização dos sistemas de arquivos; e

e) os procedimentos de arquivamento em memória terciária.

MANUTENÇÃO EVOLUTIVA

As atividades de manutenção adaptativa e perfectiva são normalmente consideradas no contexto da melhoria do software, ou seja, sua evolução em termos de atualizações e aperfeiçoamento, porém sem adição de novos requisitos ao sistema, incluindo, inclusive, a integração de novas ferramentas de software.

Para a manutenção evolutiva, o escopo de atualização a ser considerado pode conter melhoria da arquitetura de segurança do sistema, incluindo as substituições que afetem a proteção do sistema e a adoção de novas formas de proteção aos dados, incluindo criptografia no trânsito e no armazenamento dos dados, a atualização das versões de software e a integração de novos subsistemas.

Considerando que esta atividade é exaustiva e que sistemas são dinâmicos, avaliações feitas junto aos usuários devem ser feitas periodicamente para a garantia de maior longevidade sistêmica.

MANUTENÇÃO DE SOFTWARE E SOFTWARE LIVRE

A fim de compreender a diferença principal entre o modelo de software livre e os modelos tradicionais de engenharia de software, é necessário primeiro entender a alegoria de Raymond, "A Catedral e o Bazar" (RAYMOND, 1988). Nesta, o modelo de desenvolvimento de software livre é apresentado como uma nova abordagem ao processo de criação de software, o que levaria a discussões futuras de temas como desenvolvimento colaborativo distribuído utilizando a Internet, teste com envolvimento continuado dos usuários finais e a proatividade dos desenvolvedores, em oposição ao formalismo e hierarquização dos modelos tradicionais. Assim sendo, é importante a sensibilidade de se agregar conceitos e aprendizados obtidos do modelo de software livre

mesmo ao se desenvolver software governamental ou corporativo que não terá seu código fonte distribuído abertamente e de forma colaborativa através de mecanismos como a Licença Pública GNU (GPL) (WILLIAMS, 2002), por sua unicidade, aplicação ou desdobramentos, tais quais: Segurança Nacional e proteção ao interesse público. Em suma, mesmo que se realize o desenvolvimento em um ambiente fechado, cujos requisitos sejam tais que o desenvolvimento tradicional seja a escolha de referência, é possível adaptar ao modelo idéias comuns ao mundo de software livre, ao invés de apenas se utilizar ferramentas livres como se fossem tão-somente programas obtidos gratuitamente, já que o "livre" de software livre significa "liberdade" e não "grátis". O ganho maior é obtido ao se entender o conceito de software livre – a liberdade – para utilizar as ferramentas em sua plenitude, colaborando com o desenvolvimento delas pela comunidade. Ademais, é importante reconhecer que a propriedade do código fonte em todos os níveis da aplicação, dos sistemas operacionais dos servidores e clientes às aplicações e serviços mais especializados é o fator maior em termos de Segurança Nacional, uma vez que protocolos e acessos escondidos (*back doors*) em ferramentas proprietárias constituem ameaças que não podem ser ignoradas (KSHETRI, 2004).

Outro elemento a considerar na manutenção do software é que, embora seja reconhecida como a mais custosa e de maior duração em engenharia de software tradicional, não costuma receber muito enfoque. Em software livre, porém, esta é altamente centrada nos usuários, considerando-se a continuidade das versões e sua periodicidade, fatores que são observados para a própria adoção da ferramenta ou produto nas fases embrionárias. De fato, tendo em vista que o conceito de ciclo de vida da forma como a engenharia de software enxerga é muito mais flexível em software livre, a manutenção na verdade é também a busca pela auto-satisfação do desenvolvedor, que, estando em consonância com os objetivos dos usuários, estará produzindo continuamente versões melhores do software.

SOFTWARE LIVRE E MANUTENÇÃO CORRETIVA

Os objetivos da manutenção corretiva são corrigir falhas ou evitar que estas aconteçam. Para entender como esta premissa pode ser aplicada em ambiente de desenvolvimento e produção baseado em ferramentas livres, é necessário relembrar que qualquer software aplicativo desenvolvido utiliza um ou mais tipos de software básico e, conforme a arquitetura, uma camada intermediária de serviços (*middleware*). Isto quer dizer que, embora os efeitos benéficos do uso de software livre não sejam tão evidentes na manutenção corretiva do código fonte específico do aplicativo desenvolvido, a agilidade com que os desenvolvedores se apresentam na correção de *bugs* em ferramentas livres de software básico ou *middleware* é maior do que a apresentada pelas proprietárias, que dependem de fatores como viabilidade comercial, disponibilidade de programadores no suporte de manutenção e processos formais da empresa, em oposição ao fator elementar na manutenção de software livre: motivação (GACEK, 2004). O contraste é ainda maior no caso de produtos que estão fora do plano de manutenção da empresa e são considerados descontinuados para fins de suporte, ou em casos que a própria desenvolvedora original foi adquirida por terceiros ou simplesmente saiu do mercado, situações em que a ausência do código fonte impossibilita em definitivo a eliminação da falha, a não ser por uma atualização de versão "sugerida" como "vantajosa", mas que pela multiplicação de dependências incontroláveis pode causar elevação exponencial de custos, considerando a reengenharia e a verificação recursiva de compatibilidade que pode demandar.

SOFTWARE LIVRE E MANUTENÇÃO EVOLUTIVA

A discussão da manutenção evolutiva se caracteriza melhor em software livre ao serem observados os limitantes normalmente associados

à transferência de tecnologia e às limitações estratégicas de exportação pelos pólos desenvolvidos, como no caso da criptografia.

Dessa forma, o aspecto mais importante a discutir no contexto de manutenção evolutiva de software é a questão do domínio tecnológico. É fato que, para uma aplicação comum de processamento de dados, a utilização de software livre ou proprietário não produzirá efeitos práticos, além do nível filosófico, que evidenciem diferença. Entretanto, quando os requisitos do software são tais que demandam customização ou adaptação que vão além das capacidades pré-definidas de mecanismos de acesso das interfaces de aplicação para programas (API) de bibliotecas proprietárias ou do sistema operacional. O uso de software livre permite que as necessidades sejam atendidas pela adaptabilidade, profundidade e transferência tecnológicas que o controle sobre o código-fonte permite. Exemplificando, é muito menor o esforço total para se inserir algoritmos desenvolvidos localmente em um país em bibliotecas básicas de criptografia de conhecimento comum, por exemplo, mantendo-se as características dos programas que os utilizarão como opções que se agregam aos algoritmos existentes, do que se construir totalmente uma nova biblioteca criptográfica para de acomodar algoritmos locais. Por conseguinte, pode-se considerar como completo o domínio tecnológico sobre o software desenvolvido por ferramentas livres operando em ambiente livre, já que o controle, traduzido pela disponibilidade do código fonte, é total, e a transferência de propriedade e tecnologia é absoluta, permitindo incorporação de novos requisitos motivados pela alteração do contexto atual.

AUDITORIA DE CONFIGURAÇÃO

Considerando que, ainda segundo o SWEBOK, as ações de manutenção se dividem em quatro espécies, quais sejam: manter controle sobre as funções realizadas no software, manter controle sobre as modificações, aperfeiçoar funções existentes e prevenir que o

desempenho do software caia a níveis de degradação inaceitáveis, em âmbito governamental e corporativo é necessário que seja definido e executado um processo de auditoria de configuração do sistema, a fim de evitar que versões diferentes sejam executas em cada um dos locais onde este opera, de forma descontrolada, ou que usuários possam descaracterizar a configuração, inviabilizando a manutenção ou causando falhas que não sejam diretamente associadas ao código fonte. Cabe ao processo de auditoria de configuração identificar e sanar estas discrepâncias através de inspeções realizadas periodicamente e também pela implantação de procedimentos que coíbam tais práticas, bem como recuperando configurações ou solicitando incorporação de alterações realizadas à configuração padrão. Portanto, discutir manutenção de software sem refletir sobre a importância da auditoria de configuração seria um exercício incompleto, assim como também o seria ao se deixar de evidenciar como o modelo de software livre endereça o tema.

Embora as idéias básicas de controle e auditoria de configuração sejam independentes de conceitos associados a software livre ou proprietário, o processo é parte da cultura de software livre, uma vez que seria impossível conviver com a multiplicidade de versões simultâneas que normalmente existem, frutos da dinâmica do processo, sem um controle efetivo de versões existentes, o que se traduz em ferramentas e processos de qualidade e confiabilidade altamente eficientes, superiores mesmo aos requisitos das normas de qualidade que os definem como essenciais, tais como a IEEE 1028. É importante lembrar que qualidade em software livre é um aspecto endereçado naturalmente, já que emana do interesse do programador colaborativo da comunidade de desenvolvimento, em oposição ao programador proletário da indústria tradicional de software proprietário que vê as atividades de qualidade como um peso adicional.

CONSIDERAÇÕES FINAIS

A atividade de manutenção de software é essencial para que o investimento em tecnologia aplicada produza retorno apropriado. Considerando as dificuldades e limitações enfrentadas por setores governamentais, a garantia da satisfação do interesse público e a transparência devem ser evidenciadas com a adequada longevidade e operacionalidade continuada dos sistemas de software desenvolvidos. Para tanto, necessário se faz que as atividades de manutenção corretiva e evolutiva recebam o enfoque adequado, pois a obsolescência prematura causada pela descontinuidade de ferramentas de desenvolvimento e de software básico comprometem a confiança dos usuários e, ao se observar a carência de recursos que dificulta a aquisição continuada de software proprietário, o modelo de software livre se apresenta como escolha atrativa.

Em termos de manutenção de software, o fator mais importante a considerar é que a disponibilidade do código em todos os níveis é o diferencial do processo, ao se considerar o uso de software livre em oposição a ferramentas proprietárias que limitam o acesso ao código gerado.

Neste contexto, o uso de software livre se mostra como uma alternativa eficaz para a continuidade do produto, desde que consideradas a flexibilidade, maturidade e longevidade das ferramentas em questão, tanto nas melhorias que podem trazer a modelos tradicionais de Engenharia de Software quanto a abordagens mais inovadoras. De qualquer modo, software livre deve sempre ser entendido primeiro como liberdade de conhecimento e liberação de saber tecnológico, não como fonte de programas gratuitos de computador.

REFERÊNCIAS

_____, *IEEE Standard for Software Reviews*, IEEE Std. 1028-1997, IEEE Computer Society, 1997.

APPLEWHITE A. Should Governments Go Open Source?, *IEEE Software*, vol. 20, no. 4, July/Aug. 2003, pp. 88–91.

BOURQUE, P. et al. *Guide to Software Engineering Body of Knowledge: A Straw Man Version*, IEEE Computer Society, September, 1988.

BOURQUE, P; DUPUIS, R. *Guide to Software Engineering Body of Knowledge*, IEEE Computer Society, February, 2004.

FILGUEIRAS, E.Q; NAVECA, F. G. *Estudo Comparativo Entre Ferramentas Gratuitas Versus Proprietárias em um Processo de Desenvolvimento de Software Orientado a Objetos* In: WorkShop Sobre Software Livre – WSL 2003. 4. Anais. Sociedade Brasileira de Computação. Porto Alegre, 2003.

GACEK, C; ARIEF, A. The Many Meanings of Open Source, *IEEE Software*, vol. 21, no. 1, Jan./Feb. 2004, pp. 31-40.

KSHETRI, N. Economics of Linux Adoption in Developing Countries, *IEEE Software*, vol. 21, no. 1, Jan./Feb. 2004, pp. 74–81.

NORRIS, J.S. Mission-Critical Development with Open Source Software: Lessons Learned, *IEEE Software*, vol. 21, no. 1, Jan./Feb. 2004, pp. 42-49.

PRESSMAN, R.S. *Software Engineering: A Practitioner's Approach*,. 5th Edition, Mc Graw Hill, 2000.

RAYMOND, E. *The Cathedral and the Bazar*. FirstMonday. Vol. 3, Nº 3. Março de 1998.

WILLIAMS, S. *Free As In Freedom: Richard Stallman´s Crusade for Free Software*. O´Reilly & Associates, 2002.