Assegurando a Consistência da Documentação de Software por Meio do Uso de Assinatura Digital.

Marcos Gonçalves Rios e Roberto da Silva Bigonha
Universidade Federal de Minas Gerais

SUMÁRIO

Durante o ciclo de vida de um software, parte significativa do custo total é gasta com manutenção. A prática da manutenção se baseia no uso de uma documentação consistente. Considerando que a parte principal de uma documentação é o seu respectivo código-fonte, a técnica proposta neste trabalho objetiva assegurar a consistência entre o código-fonte e os demais artefatos que compõem a documentação técnica ou do desenvolvedor. Durante o desenvolvimento da primeira versão de um software, isto é, antes dele ter sido colocado em operação, é útil, também, a utilização de mecanismos controladores de versão, especialmente, no desenvolvimento de grandes sistemas. A técnica apresentada fudamenta-se na utilização de assinatura digital. Um sistema protótipo desenvolvido para testá-la é, também, apresentado por meio do uso de programas desenvolvidos em Java e documentados por meio de hipertextos.

Palavras Chaves: Documentação de Software, Controle de Versão, Assinatura Digital, Manutenção de Software, Desenvolvimento de Software e Hipertexto.

INTRODUÇÃO

É parte integrante do processo de desenvolvimento de software a geração simultânea de sua documentação. Esta documentação deve ser completa e atualizada. O seu objetivo é armazenar de forma organizada e acessível toda a informação necessária para que um profissional da área, diferente daquele que desenvolveu o programa, possa compreendê-lo e alterá-lo de maneira eficiente e com sucesso. Os seguintes artefatos fazem parte da documentação: especificação de requisitos, código-fonte, especificação de testes, especificação de projeto, planos de desenvolvimento, registro de acompanhamento de problemas e especificação de arquitetura, etc. Estes e outros componentes da *documentação técnica*[1] são representados, na maioria das vezes, por diagramas e textos. Esta documentação técnica, por sua vez, raramente é lida seqüencialmente [Staa, 1995]. Para resolver esta questão, a composição de diagramas, textos e códigos deve poder ser explorada por meio de um navegador *(browser)* de hipertexto [Bigelow, 1988]. Estes hipertextos são chamados *hiperdocumentos* quando são compostos por documentos elementares e até por outros *hiperdocumentos* [Martin, 1992].

[1] Sob um outro ponto de vista, o da compilação, a *documentação técnica* engloba: textos de documentação e códigos-fonte [Staa, 2000], portanto é dirigida aos programadores.

A documentação deve ser apresentada em pelo menos três formas: documentação interna, que é dirigida aos programadores que implementarão e manterão o módulo; documentação externa, que é dirigida aos programadores de módulos cliente do módulo em questão e documentação de uso, que é dirigida aos usuários dos programas que contêm o módulo. Esta nomenclatura não é um consenso entre os autores pesquisados. B. Meyer [Meyer, 1997] define documentação do usuário como documentação externa, documentação de interface do módulo como a destinada ao utilizador do módulo; e documentação interna como a destinada ao desenvolvedor e mantenedor do módulo.

Após a colocação em produção ou a entrega do produto, os requisitos podem mudar ou erros serem detectados. A manutenção é uma fase do ciclo de vida de um software que, por meio da utilização de documentação técnica escrita pelo desenvolvedor [Jacobson, 1992], foca na solução destes problemas. A manutenção pode ser feita de cinco formas distintas: corretiva, em que se pretende realizar a correção de uma falha remanescente; evolutiva, por meio da alteração da funcionalidade do programa; adaptativa, por meio do atendimento a uma alteração do ambiente, na qual um programa correto é alterado sem modificar a sua funcionalidade; perfeccionista, para melhorar o desempenho ou a interface com o usuário e preventiva, também denominada reengenharia de software, onde alterações são efetuadas para permitir que o programa seja mais facilmente alterado. Quando a documentação é inexistente ou pouco confiável, como costuma ocorrer em *sistemas legados*[2], faz-se necessário extrair do código-fonte a sua especificação e arquitetura. A esta atividade de recuperação da documentação de sistemas já desenvolvidos dá-se o nome de engenharia reversa [Pressman, 1977] [Staa, 2000].

Portanto, o principal objetivo do desenvolvimento da documentação técnica é a obtenção de redução do custo de manutenção. O custo de manutenção durante o ciclo de vida de um software é parte relevante do custo total. Segundo vários autores, este custo pode ser de 40% [Brooks, 1975], 70% [Meyer, 1997], 60% [Putnam-Fitzsimmmons, 1979] e [Hanna, 1993], e 82% [Stone, 1979] . O tipo de software e a metodologia de estimativa utilizada levam a resultados bem diferentes. Segundo vários autores citados em [Boehm, 1981], sistemas de sofwtare básico são os de custo de manutenção mais elevados, seguidos de softwares aplicativos empresariais, e, por último, os de tempo real são os de custo de manutenção mais baixos. Boehm, em seu modelo de estimativa de custo COCOMO (*Constructive Cost Model*) [Boehm, 1981], apresenta uma fórmula de cálculo que contém um parâmetro de esforço que depende do tipo da aplicação e da linguagem utilizada. Para o desenvolvimento de software de grande índice de interação humana esse parâmetro de esforço se eleva de centenas a milhares de vezes.

[2] Diz-se *sistema legado*, ou sistema herdado, aquele sistema cujo profissional ou equipe de software responsável pela sua manutenção não participou de sua construção.

Uma pesquisa desenvolvida por Lientz e Swanson [Lientz, 1979], publicada em 1979 e citada em [Meyer, 1997] e [Boehm, 1981], em 487 instalações de desenvolvimento de software de diversos tipos, mostra que 41,8% dos custos envolvidos em manutenção se devem a alterações dos requisitos, que por sua vez são divididas em extensões e modificações. Embora os dados de entrada devam ser considerados especializados, pois a pesquisa foi realizada em aplicações de lote de tamanho médio de 23.000 instruções, hoje obsoletos, os resultados desta pesquisa, de uma forma geral, são ainda aplicáveis, conforme B. Meyer [Meyer, 1997].

Uma outra pesquisa interessante foi desenvolvida pelo *Software Engineering Group of the National Research Council of Canada* com o objetivo de tabular as práticas de manutenção de software com doze grupos privados e uma instituição governamental [Singer, 1998]. Este trabalho foi desenvolvido em ambientes baseados em plataformas, linguagens, tipos de aplicações e tempos de operação diferentes A primeira conclusão interessante dessa pesquisa é que a fonte mais importante para a manutenção é o conhecimento de um profissional experiente. Em segundo lugar, a referência mais segura é o próprio código-fonte e, por último, a documentação, caso não haja outro profissional para opinar. A pesquisa também aponta a necessidade de ferramentas para a navegação no código-fonte. O maior problema, ainda segundo a pesquisa, é a manutenção da consistência da documentação. Durante as entrevistas, os funcionários foram questionados se consultavam a documentação para efetuar a manutenção. Foi apurado que: algumas vezes o faziam por curiosidade; que às vezes encontravam algum material relevante, mas sempre inconsistente; que havia documentação demais; que quando o software de documentação é interativo, uma página diz algo e outra muda o que foi dito; e que independentemente da quantidade de documentação existente, ela não é confiável, pois sempre está ultrapassada.

Uma outra questão importante em relação à manutenção é a introdução de novas falhas nessa fase. Durante o período em que gerenciou o projeto de desenvolvimento do sistema operacional OS/360, F. Brooks [Brooks, 1975] constatou que o problema fundamental durante a manutenção é a chance de se introduzir um outro erro, estimada na faixa de 20% a 50%. Isto acontece por dois motivos. O primeiro é que qualquer defeito, por mais simples que seja, pode possuir ramificações sutis, e para repará-lo com o menor custo possível é necessária uma documentação excelente ou uma estrutura pura. Do contrário, apenas o defeito local será corrigido. O segundo é que o reparador, geralmente, não é a mesma pessoa que escreveu o código e, também, é menos experiente.

Das pesquisas apresentadas pode-se concluir que:

1. O gasto com o software, após ter sido colocado em produção, pode ser superior ao de seu desenvolvimento;
2. A alteração de requisitos é responsável pela maior parte dos custos de manutenção, cerca de 41,8%;

3. A fonte para consulta ao se efetuar a manutenção de um software é o programador experiente em primeiro lugar, o programa fonte em segundo lugar e a sua documentação por último;
4. Para consultar o programa- fonte, é necessária a existência de um mecanismo de navegação que facilite a identificação do problema e o seu reparo;
5. Para que a utilização da documentação seja útil, ela deve ser objetiva, estar localizada ou ligada junto ao código-fonte e ser consistente;
6. A chance de serem introduzidas novas falhas durante as intervenções de manutenção é alta, cerca de 20% a 50%;
7. Um mecanismo de controle de versões se faz necessário em grandes sistemas.

GERENCIAMENTO DE CONFIGURAÇÃO

A necessidade de suportar a evolução de um software e de seus componentes levou ao aparecimento de várias propostas de mecanismos de controle de versões e configurações [Magalhães, 1991]. O processo de gerenciamento de mudanças de sistemas de software, que é crítico para a manutenção de grandes sistemas, denomina-se gerência de configuração [Lucena, 1991]. A gerência de configuração controla e coordena a evolução dos diversos artefatos e assegura a sua integridade ao longo do ciclo de vida. Estão sob o controle da gerência de configuração: os códigos, projetos, documentos de requisitos, ferramentas, compiladores e todos os demais artefatos necessários a um ambiente de software [Buckle, 1982].

Mudanças são inerentes ao desenvolvimento e à manutenção de um software. Com o aumento do tamanho e da complexidade dos sistemas se faz necessária a coordenação do desenvolvimento e da manutenção destes processos. As atividades que compõem o gerenciamento de configuração são: identificação da mudança, controle da mudança, asseguramento da correta implementação da mudança e divulgação aos interessados da mudança efetudada. À medida que um sistema de software avança no seu ciclo de vida, o número de componentes que são produzidos aumenta significativamente. Estes componentes podem pertencer a três categorias: programas (códigos e objetos), documentos que descrevem os programas (para usuários ou para técnicos) e dados. Uma configuração de software é o conjunto de itens obtidos durante o processo de software. Na realidade, uma configuração de software é um conjunto complexo composto de programas, ferramentas de software, documentos e testes relacionados a marcos específicos do ciclo de vida. Para tanto, é necessário que cada item seja separadamente nomeado para a sua identificação. Uma configuração também deve contemplar as relações existentes entre os diversos itens. Durante o desenvolvimento e evolução de um software, os seus itens podem possuir diferentes versões. Mudanças podem ocorrer em apenas uma versão, parte das versões ou em todas versões de um determinado item. O controle de versão combina procedimentos e ferramentas com o objetivo de gerenciar as diversas versões criadas. O gerenciamento de configuração permite a um usuário

especificar diferentes configurações de um sistema de software a partir da seleção das versões apropriadas. Isto e possível pois a cada configuração correspondem uma série de atributos que caracterizam a versão do sistema de software [Pressman, 1997]. Existe uma grande variedade de ferramentas de suporte ao gerenciamento de configuração em virtude da real impossibilidade de controle manual de alterações em grandes sistemas de software.

Criptografia

Criptografia significa a arte de escrever secretamente por meio de abreviaturas ou sinais convencionados entre duas ou mais pessoas. O vocábulo Criptografia é formado pelas palavras *Kriptos*, do grego, oculto e *graphein*, escrever. Na informática, significa a ação de criptografar ou codificar dados para que possam ser lidos somente por pessoa autorizada. Este procedimento é empregado principalmente para garantir a segurança de transmissão de dados, por exemplo no caso de compras na rede Internet [Larousse, 1995]. Um algoritmo de criptografia (*cipher*) é uma função matemática usada para os processos de codificação e decodificação criptográfica. Este algoritmo faz uso de uma chave e a segurança dos dados criptografados é baseada em dois itens: o segredo da chave e a qualidade do algoritmo. A criptografia convencional ou simétrica se baseia no uso de uma única chave para os processos de codificação e decodificação.

O sistema convencional de criptografia possui suas vantagens, é bem rápido e especialmente útil quando o dado codificado possui um destino único bem definido. A grande desvantagem está em seu alto custo e no risco do envio da chave secreta. O problema da distribuição das chaves foi resolvido em 1975 por W, Diffie e M. Hellman [Diffie, 1976]. Existem evidências de que o Serviço Secreto Britânico teria obtido esta solução alguns anos antes, mas a manteve como segredo militar conforme J. H. Ellis (em *The Possibility of Secure Non-Secret Digital Encryption, CESG Report, January 1970*) citado em [PGP, 2000]. A criptografia convencional é utilizada por grandes corporações e por governos, por necessitar de canais seguros para a transmissão de chaves secretas.

O sistema de chaves públicas é assimétrico, o que significa que faz uso de duas chaves: uma pública, para a codificação, e uma secreta, para a decodificação. A necessidade do remetente e do destinatário compartilharem uma chave secreta que necessita ser enviada por um meio seguro é eliminada. Exemplos de sistemas de chaves públicas são: Elgamal, que leva o nome de seu inventor, Tagher Elgamal, RSA, desenvovido por: Ron Rivest, Adi Shamir e Leonard Adleman [Rivest, 1978], Diffie-Hellman [Diffie, 1976] e DAS *(Digital Signature Algorithm)*, inventado por David Kravitz [PGP, 2000]. A criptografia de chaves públicas pode ser considerada uma revolução que disponibilizou a criptografia para os demais setores [PGP, 2000].

Assinatura Digital

Um grande benefício da criptografia de chaves públicas é que ela fornece a técnica da assinatura digital. A assinatura digital permite ao destinatário verificar a autenticidade da origem da informação e, também, verificar se a informação está intacta. Em outras palavras, a técnica de assinatura digital de chaves públicas assegura a autenticação e a integridade dos dados. Assinar um texto digitalmente é nele apor uma marca que permita verificar se o texto foi alterado e se a sua origem é a desejada. Esta marca é formada por um texto criptografado pela chave secreta do remetente que pode ser decodificada pela sua chave pública, disponível a todos, o que assegura a origem da mensagem.

A manutenção da inviolabilidade é feita por meio de uma função *hash* que produz um texto de tamanho fixo (160 bits, por exemplo). Se a mensagem for mudada em um bit que seja, a função *hash* produzirá um texto diferente. Como entrada para esta função é usada a mensagem a ser transmitida. O destinatário com a chave pública decodifica a assinatura e confere com a mensagem submetida à função *hash* e verifica se a mensagem foi violada. A assinatura decodificada tem que ser idêntica ao resultado da função *hash* que tem como entrada a mensagem recebida. A mensagem enviada pode ser criptografada ou não, dependendo do objetivo.

Criptografia em Sistemas Reais

Nos sistemas reais, a primeira etapa de um processo de criptografia é a compactação. A compactação economiza tempo de transmissão e dificulta o uso de técnicas de decodificação baseadas na busca de padrões. A segunda fase é a elaboração da chave ou das chaves. Chaves são números grandes da ordem de 100 a 3.000 bits. É importante observar que uma chave de 8 bits na criptografia convencional equivale, em termos de resistência à tentativa de decodificação não autorizada, a uma chave de 1.024 bits na de chaves públicas, e uma de 128 bits equivale a uma de 3.000. O tamanho da chave deve levar em conta quanto tempo esta mensagem deverá ser mantida, pois com novos computadores e com tempo de computação suficiente, a mensagem pode ser decodificada [PGP, 2000].

Para a obtenção de assinaturas digitais é necessário o uso de funções *hash* (*one-way hash*) *h*, fácil de ser computada, que tem como entrada uma mensagem *M*, longa a ser enviada, e como saída, uma mensagem curta, por volta de 160 bits no sistema PGP [PGP, 2000]. Esta função é construída de tal forma que é praticamente impossível encontrar duas mensagens *M* e *M'* tal que $h(M) = h(M')$ [Cormen, 1995].

Por motivos de eficiência alguns sistemas de criptografia como, RSA, adotam uma solução híbrida, mostrada na Figura 6.5, no gerenciamento da chave. Considerando os tradicionais personagens dos livros sobre criptografia: Alice e Bob, Alice deseja mandar uma longa mensagem criptografada *M* para Bob. Ela seleciona uma chave gerada aleatoriamente *K*, criptografa *M* usando *K*, e obtém a

mensagem criptografada *C,* que é tão longa quanto *M.* Então ela criptografa a chave *K* fazendo uso da chave pública de Bob *K.* Uma vez que *C* é bem maior que *Pb(K),* a transmissão de *Pb(K)* é bem mais rápida e segura. Bob recebe *Pb(K)* e a decodifica usando a sua chave secreta, obtendo, assim, a chave *K.* Com esta chave ele decodifica a mensagem *C* [Cormen, 1995]. Esta solução híbrida é bastante útil, uma vez que o sistema convencional, dependendo do tamanho das chaves, chega a ser mil vezes mais rápido que o sistema de chaves públicas [PGP, 2000].

Uma questão a que o usuário de sistemas de criptografia de chaves públicas deve ficar atento é se está usando a chave pública do destinatário correto. O certificado digital simplifica a tarefa de se verificar se a chave pública usada corresponde ao destinatário ao qual se quer mandar a mensagem. Um certificado digital é composto de uma chave pública, um grupo de informações, como por exemplo, nome e telefone do usuário, e uma ou mais assinaturas digitais.

A Técnica Proposta

Uma vez caracterizada a necessidade de mudança, a configuração anterior a ela deve ser mantida. Esta configuração anterior é composta de atributos que fazem referência a artefatos tais como: código-fontes, especificação de requisitos, etc. A traceabilidade de uma determinada configuração é uma característica importante pois leva a facilitação da atividades de desenvolvimento e manutenção. Cada versão de um sistema obtida após a implementação de uma mudança é composta de novas versões de parte ou de todos artefatos que compõem a documentação. Existem ferramentas sofisticadas que mantêm registro das mudanças de um sistema assim como também administram outras atividades como, por exemplo, desenvolvimento paralelo por grupos de desenvolvedores.

Na grande maioria das mudanças, modificações deverão ser efetuadas nos códigos-fontes que por sua vez implicarão na geração de novos códigos-binários. Parte destas mudanças ocorre na detecção de defeitos durante os testes ou mesmo após o sistema ter sido colocado em operação. Este tipo de defeito possui como referência inicial a execução de determinado programa-objeto, ou código intermediário como bytecode em Java. Nesta situação, comum no dia a dia de profissionais responsáveis pela manutenção, a situação de ocorrência do defeito deve ser caracterizada e, se possível, capaz de ser reproduzida e o código-fonte respectivo capaz de ser identificado. Em grandes instalações, configurações diferentes de um mesmo sistema convivem, o que leva à existência de diferentes versões de um mesmo artefato. A existência de um mecanismo simples, rápido e eficiente de identificação segura dos artefatos relacionados a este código-objeto específico permite uma intervenção rápida que minimiza a inclusão de novos defeitos.

O propósito desta técnica é obter uma solução para o problema da consistência física entre o código-fonte e o hipertexto, e assegurar que um código-fonte em

estudo seja identificado por meio de um código-objeto específico, sem se refazer a compilação deste código. Não se pretende prescindir da utilização dos sistemas de gerenciamento de configuração existentes, mas oferecer uma técnica complementar que possa melhora-las. Esta técnica faz parte de um trabalho de pesquisa do Grupo de Linguagens da Universidade Federal de Minas Gerais, que em conjunto com outras técnicas, objetiva gerar de forma automática uma documentação de sistemas de software que atenda a requisitos de qualidade, tais como: consistência, precisão e navegabilidade.

Tratando cada unidade ou conjunto de unidades compiláveis que darão origem a uma unidade executável é possível obter deste texto inicial uma assinatura digital. Esta assinatura digital, por sua vez, é inoculada como um vírus inócuo no código executável de modo a não alterar o seu funcionamento. Partindo do pressuposto que os demais artefatos que compõem a documentação estão organizados na forma de um hipertexto, esta assinatura também pode ser aposta nas partes do hipertexto que possui artefatos ligados a este código-fonte. A geração e conseqüente colocação de uma nova assinatura significa a criação de uma nova configuração.

A solução adotada para o problema da consistência entre o fonte e o objeto é apor neste último, o objeto, a assinatura do primeiro, o fonte. Caso o código-fonte tenha sido alterado, a sua assinatura não será a mesma existente no código-objeto. A assinatura do código-fonte colocada no código-objeto funciona, também, como uma chave de acesso para se localizar qual a versão de determinada unidade de compilação está em exame.

A assinatura do código-objeto pode ser colocada no código-fonte. A alteração de um código- objeto por meio de sua edição direta é uma ação rara, mas pode ser feita por hackers ou por vírus. Para se prevenir desta situação, o código-objeto, também, pode dar origem a uma assinatura digital a ser colocada junto ao código-fonte até mesmo junto aos artefatos a ele relacionados.

Um hipertexto possui relação com várias unidades de compilação. Uma vez que o hipertexto pode ser alterado, assim como, também, o código-fonte, deve-se estabelecer qual entidade é mandatória. Nesta solução adota-se como mandatório o código-fonte. O fonte, quando alterado, deve invalidar a documentação e não o contrário. Portanto, deve-se colocar a assinatura do código-fonte na parte do hipertexto que o documenta. Caso o código-fonte seja alterado, a sua assinatura não será a mesma da colocada no hipertexto, o que mostrará a sua inconsistência. Pode-se também, após o término da elaboração do hipertexto, assina-lo, para assegurar a sua inviolabilidade.

Tabelião

Foi desenvolvida uma ferramenta com o objetivo de validar a técnica proposta, *Tabelião*, que gera a assinatura digital, a coloca e confere a integridade (autentica) do artefato e da relação entre eles: código-fonte, código-objeto e hipertexto.

Esta ferramenta, partindo do código-fonte, assina o código-objeto e a parte do hipertexto referente à classe em análise, e faz a conferência da inviolabilidade do artefato em questão e de seus relacionamentos. Esta ferramenta pode ser considerada em estado de *betateste*, portanto totalmente funcional, e pode evoluir para operar com pacotes, indexadores automáticos de classes e com o envolvimento de um número maior de artefatos em uma documentação mais complexa.

Esta ferramenta faz uso do sistema de criptografia *freeware* PGP [PGP, 2000]. A ferramenta é de fácil utilização. O seu objetivo é assegurar a integridade (inviolabilidade) de um artefato e a integridade da relação entre dois ou mais artefatos. O uso do *Tabelião,* tendo como entrada para o cálculo da assinatura o código-fonte e a sua colocação na parte relativa no hipertexto e no código-objeto, assegura a inviolabilidade do código-fonte e da sua relação com o hipertexto e com o código-objeto (ou *bytecode*). Partiu-se do princípio que o hipertexto não pode ser editado, pois é gerado automaticamente a partir do código-fonte, entretanto nada impede que uma nova versão do *Tabelião* permita que uma parte do hipertexto, ou até mesmo um código-objeto (ou *bytecode*), possa fazer parte da entrada para o cálculo de outras assinaturas que permitirão que partes do hipertexto possam ser editada. A assinatura de um código-objeto deve ser feita de tal forma que o seu funcionamento não seja alterado. O uso desta ferramenta, no estudo de caso, provou que esta técnica assegura a consistência de versão.

Conclusão

Foram propostos três tipos de intervenção.

1. Apor no código-objeto a assinatura do código-fonte. Esta ação objetiva auxiliar a gerência de configuração no controle de versões. Em sistemas reais são centenas ou milhares de classes implementadas em centenas ou milhares de unidades de compilação. Os problemas detectados em tempo de execução se referem a um código-objeto, que por sua vez foi originado de um código-fonte. A localização deste código-fonte por meio de uma chave, que é a sua assinatura, permite verificar se ele corresponde ao objeto, sem a necessidade da recompilação. Troca-se a recompilação e a comparação dos objetos por uma aplicação da função hash, da criptografia de seu resultado e da comparação das assinaturas. Por necessidade de eficiência, pode-se apor também no código-fonte a sua própria assinatura e só quando a sua violabilidade for questionada é que a assinatura será conferida. Ganha-se em eficiência e perde-se em segurança.

2. Colocar no hipertexto, na parte referente à unidade de compilação em questão, a sua assinatura com o objetivo de garantir a integridade do código-fonte que foi documentado naquele hipertexto. Quando a documentação for consultada, o usuário (desenvolvedor) poderá conferir a assinatura do código-fonte e verificar a consistência da documentação. Conforme a intervenção de número um, pode-se colocar no código-fonte da unidade de compilação a sua própria chave com ganho de eficiência e perda de segurança.

3. No hipertexto deve-se colocar a sua própria assinatura, para assegurar a sua inviolabilidade. O hipertexto pode e deve ser editado para se construir a documentação, pois as informações contidas no código-fonte, na grande maioria das vezes, não são suficientes para a sua construção. A estrutura do contrato, determinada pelo conjunto de assinaturas de classes e pelas suas asserções, não pode ser violada.

Como pode ser observado, a técnica de assinatura digital permite verificar a integridade do artefato, e o seu uso em outros artefatos permite a sua associação a um ou mais artefatos diferentes. A consistência está diretamente ligada a inviolabilidade de um artefato de software. A utilização desta proposta associada a outros sistemas de gerenciamento de configuração permitirá um controle mais seguro dos artefatos e de suas relações assegurando a consistência da documentação.

Bibliografia

[Boehm, 1981] Barry W. Boehm. **Software Engineering Economics**. Prentice-Hall, Inc, New Jersey, USA, 1981.

[Brooks, 1975] Brooks, Derick P.,JR. **The Mythical Man-Month.** Addison-Wesley, USA, 1995.

[Buckle, 1982] Bucke, J. K., **Software Configuration Management**. MacMillan Computer Science Series. USA, 1982.

[Coleman, 1994] Coleman, Derek et al. **Object-Oriented Development** – The Fusion Method. New Jersey, Prentice-Hall, Inc. 1994.

[Cormen, 1995] Cormen, Thomas H., Leiserson, Charles E. e Rivest, Ronald L.. **Introductions to Algorithms**. The MIT Press and McGraw-Hill Book Company. 1995.

[Diffie, 1976] Whitifield, Diffie e Hellman, Martin E.. **New Directions in Criptography**. IEEE Transactions on Information Theory, Vol. IT-22, pp. 644-654, No. 6, Nov. 1976.

[Hanna, 1993] Hanna, M. **Maintenance Burden Begging for a Remedy**, Datamation, April 193, pp. 53-63.

[Jacobson, 1992] Jacobson, Ivar, Christerson, Magnus, Jonsson, Patrik and Overgaard, Gunnar. **Object-Oriented Software Engineering A Use Case Driven Approach**. ACM Press and Addison-Wesley, 1992.

[Larousse, 1995] **Grande Enciclopédia Larousse Cultural**. Editora Plural. São Paulo. 1995.

[Lientz, 1979] Lientz, Bennet P. Swanson, E. Burton. **Software Maintenance a User/Management Tug-of-War**. Datamation, April, 1979.

[Lucena, 1991] Lucena, Carlos José Pereira e Alencar, Paulo Sérgio C. **Formalização de Conceitos em Gerência de Configurações.** V Simpósio Brasileiro de Engenharia de Software da Sociedade Brasileira de Computação, pp. 75-91, em Ouro Preto de 23 a 25 de outubro de 1991.

[Magalhães, 1991] Magalhães, Geovane Cayres e Dias e Eliane Zabom Victorelli Dias. **MVC: Um Modelo para Controle de Versões e Configurações.** V Simpósio Brasileiro de Engenharia de Software da Sociedade Brasileira de Computação, pp. 93-106, em Ouro Preto de 23 a 25 de outubro de 1991.

[Martin, 1992] Martin, James. **Hiper Documentos e como criá-los.** Editora Campus, rios de Janeiro, 1992.

[Meyer, 1997] Meyer, Bertrand. **Object-Oriented Software Construction** – Second Edition. Prentice-Hall, Inc, USA, 1997.

[PGP, 2000] **Pretty Good Privacy- PGP Freeware User's Guide**. Networks Associates Technology,. Inc. Santa Clara California. 1999.

[Pressman, 1997] Pressman, Roger S.. **Software Egineering – A Practioner's Approach**. McGraw-Hill, 1997.

[Putnam-Fitzmmons, 1979] Putnam, L.H. e Fitzmmons, A. **Estimating Software Costs**. Datamation, Setembro, pp. 189-198, Outubro, pp. 171-178, Novembro, pp.1979, pp137-140,1979.

[Rivest, 1978] Rivest, R. L. Shamir, A e Adleman, L.. **A Method for Obtaining Digital Signatures and Public-Key Cryptosystems**. Communications of the ACM, 21, 2 (Feb, 1978), pp. 120-126.

[Singer,1998] Singer, Janice. **Practices of Software Maintenace.** http://searchpdf.adobe.com/proxies/1/78/56/92.html. 1998.

[Staa, 2000] Staa, Arndt von. **Programação Modular.** Editora Campus, Rio de Janeiro, 2000.

[Stone, 1979] Stone, Harold, S.. **Life-Cycle Cost Analysis of Instruction-Set Architecture Standardization for Military Computer Systems.** IEEE Computer April, pp. 35-47, 1979.