10 Gerenciamento de Configuração e Mudança

Este capítulo vai apresentar a disciplina de gerenciamento de configuração e mudança, iniciando pelos *conceitos básicos* (Seção 10.1), como itens de configuração, versões, *baseline*s e *release*s. Na seção seguinte é apresentado o funcionamento de um sistema de controle de versão (Seção 10.2) para desenvolvimento de produtos de software. Em seguida são apresentados os conceitos de *controle de mudança* (Seção 10.3) e *auditoria de configuração* (Seção 10.4). O capítulo termina com uma apresentação de ferramentas para controle de configuração e mudança (Seção 10.5).

O Gerenciamento de Configuração de Software (GCS[128]) ou Gerenciamento de Configuração e Mudança (GCM) é considerado uma disciplina à parte dentro do gerenciamento de projetos. Essa área é especialmente importante na indústria de software porque componentes e produtos de software são desenvolvidos e modificados muitas vezes ao longo do tempo, durante a após o projeto, e assim, diferentes versões de um mesmo componente ou produto podem estar disponíveis em diferentes momentos.

O gerenciamento de configuração, portanto, é a área que vai indicar como as diferentes versões dos artefatos envolvidos no desenvolvimento de software devem ser modificadas e identificadas.

Pressman (2005) indica que o gerenciamento de configuração e mudança deve ter como objetivo responder às seguintes perguntas:

a) O que mudou e quando?
b) Por que mudou?
c) Quem fez a mudança?
d) Pode-se reproduzir essa mudança?

Essas questões serão respondidas pelas três grandes atividades do gerenciamento de configuração e mudança:

a) O *controle de versão* vai dizer *o que* mudou e quando.
b) O *controle de mudança* vai dizer *por que* as coisas mudaram.
c) A *auditoria de configuração* vai dizer *quem* fez a mudança e como ela pode ser reproduzida.

Essas atividades serão detalhadas nas Seções 10.2 a 10.4.

10.1 Conceitos Básicos

Antes de discorrer mais detalhadamente sobre as atividades de gerenciamento de configuração, alguns conceitos básicos são apresentados nas próximas subseções.

[128] *Software Configuration Management* (SCM) em inglês.

10.1.1 Item de Configuração de Software

Um *Item de Configuração de Software* (*ICS*[129]) é um elemento unitário para efeito de controle de versão, ou ainda, um agregado de elementos que são tratados como uma entidade única no sistema de gerenciamento de configuração.

Não apenas código de programação deve ser controlado pelo gerenciamento de configuração, mas também documentação, diagramas, planos, ferramentas, casos de teste, dados, e quaisquer outros artefatos relacionados ao desenvolvimento do software.

Cada item de configuração recebe usualmente um número, que poderá inclusive ser identificado por várias partes usualmente separadas por ponto. Por exemplo:

a) Plano de projeto, versão 2.
b) Biblioteca de funções matemáticas, versão 4.1.
c) Disco de instalação do sistema, versão 1.2.4.

Um dos problemas com o gerenciamento de configuração consiste em determinar a melhor granularidade para os itens de configuração. Ter itens demais poderá dificultar o controle dos mesmos e as configurações de software, que são formadas por um conjunto de itens de configuração, serão listas muito extensas. Por outro lado, ter itens de menos também poderá dificultar o controle, porque cada item terá um número muito grande de versões.

Em sistemas orientados a objetos usualmente um item de configuração terá a granularidade de um pacote ou componente de classes afins que não ultrapasse poucas dezenas. Mas dependendo do projeto esse tamanho poderá variar. Projetos muito grandes tenderão a ter itens maiores, com mais classes, e projetos muito pequenos poderão ter até classes individuais sendo definidas como itens de configuração.

Os itens de configuração podem ser básicos ou compostos. Os *básicos* são os artefatos arbitrados como individuais para efeito de controle de versão. Os *compostos* podem ser formados por outros itens básicos e compostos. Cada item de configuração será definido por quatro elementos:

a) Um *nome*, que é uma cadeia de caracteres que identifica claramente o item básico ou composto.
b) Uma *descrição*, que consiste da definição do tipo de item (documento, diagrama, dados, código fonte, etc.) e detalhes sobre ele.
c) Uma lista de *recursos* que são outros itens de configuração exigidos pelo item básico. No caso de itens compostos a lista de recursos poderá ser definida como a união das listas de seus itens básicos.
d) Uma *realização*, que no caso do item básico é um ponteiro ou endereço para o artefato que efetivamente realiza o item. No caso de itens compostos a realização é definida como o conjunto das realizações de seus itens básicos.

[129] *Software Configuration Item* (*SCI*) em inglês.

10.1.2 Relacionamentos de Itens de Configuração de Software

Conforme visto na seção anterior, um ICS pode estar ligado a outros a partir de uma lista de recursos exigidos. Existem vários tipos de dependência ou relacionamento que podem existir entre itens de software. A maioria destes relacionamentos é previsto nas ferramentas CASE que permitem desenhar diagramas de classe UML. Tais tipos de relacionamento podem tanto ser representados por associações de um tipo específico quanto por associações estereotipadas. A lista abaixo, embora não seja exaustiva, apresenta alguns exemplos de relacionamentos entre ICS que devem também ser mantidos por um sistema de controle de configuração de software:

a) *Dependência*, que usualmente indica que um componente utiliza funções que são implementadas em outro componente. A associação pura e simples da UML não deixa de ser uma forma de dependência, neste sentido, já que uma classe vai estar associada à outra normalmente para poder utilizar funções implementadas na outra.
b) *Agregação* e *composição*, que indicam que um componente é formado por outros.
c) *Realização*, que indica que um componente concreto é a implementação de um componente mais abstrato.
d) *Especialização*, que indica que um componente é uma variante mais específica de outro. Usualmente o componente mais especializado depende do componente mais genérico. O inverso só será verdade se o componente genérico tiver métodos abstratos que precisam ser necessariamente implementados no componente mais especializado. Neste caso, pode-se dizer que existe dependência mútua entre os componentes.

10.1.3 Rastreabilidade

Rastreabilidade ou controle de *rastros*, refere-se a um dos princípios da engenharia de software para o qual ainda existe significativa dificuldade em sua implementação.

Manter um controle de rastros entre *módulos de código* não é difícil porque as dependências entre módulos usualmente é indicada de forma sintática pelos próprios comandos da linguagem (*import* ou *uses*, por exemplo).

Mas manter controle de rastros entre artefatos de análise e *design* não é tão simples. A rastreabilidade é importante porque quando são feitas alterações em artefatos de análise, *design* ou código, é necessário manter a consistência com outros artefatos, caso contrário, a documentação fica rapidamente desatualizada e inútil.

A técnica de rastreabilidade mais utilizada nas ferramentas CASE é a *matriz de rastreabilidade*, que associa elementos entre si, por exemplo, casos de uso e seus respctivos diagramas de sequencia, ou classes e módulos de programa. Porém, esse tipo de visualização matricial torna-se impraticável à medida que a quantidade de elementos cresce, especialmente se o controle das relações entre os elementos precisar ser feita manualmente (Cleland-Huang, Zemont, & Lukasik, 2004).

São reportadas também pesquisas que procuram automatizar a recuperação de relações de rastreabilidade entre artefatos a partir de evidências sintáticas. Porém, devido às escolhas arbitrárias dos nomes de artefatos (falta de padrão), usualmente estes sistemas de

recuperação retornam muitos falsos positivos, o que inviabiliza seu uso (Settimi, Cleland-Huang, Khadra, Mody, Lukasik, & de Palma, 2004).

Outra abordagem, ainda experimental, é apresentada por Santos e Wazlawick (2009), a qual consiste em modificar a maneira como elementos são criados nos editores de diagramas das ferramentas CASE, de forma que, com exceção dos requisitos, que são produzidos externamente, outros elementos quaisquer (como casos de uso, classes, código, diagramas etc.) só possam ser criados como uma derivação de algum outro elemento que já exista.

Desta forma sempre que um item é criado no repositório do projeto, um rastro é automaticamente criado entre ele e o elemento que lhe deu origem. Apenas itens que representam requisitos (que vem do cliente, ou seja, são externos ao projeto) podem ser adicionados ao repositório sem ser por derivação de outros itens.

10.1.4 Versões de Itens de Configuração de Software

A *versão* de um ICS é um estado particular deste item durante o desenvolvimento de um sistema. Uma versão normalmente é identificada por um número.

Versões de ICS sucedem-se no tempo. Pode-se dizer que há basicamente dois tipos de sucessores para uma versão:

a) Uma *revisão* é uma nova versão de um item que foi criada para substituir uma versão anterior.
b) Uma *variante* é uma nova versão de o item que será adicionada a configuração sem a intenção de substituir uma versão antiga.

Pode-se falar ainda em versões experimentais para determinados itens, ou seja, uma revisão é feita no item, mas ainda não está decidido se a nova versão será efetivamente mantida. Neste caso, a revisão pode até ser considerada como uma variante, já que não existe ainda a determinação de que a versão anterior é obsoleta.

10.1.5 Configuração de Software

Uma *configuração de software* é o estado em que o software se encontra em um determinado momento. A configuração de um sistema pode incluir todos os elementos físicos ou abstratos relacionados com o produto. Porém, uma configuração de software normalmente incluirá apenas o estado dos artefatos que são mantidos em formato eletrônico, como por exemplo, os programas, documentos eletrônicos, ferramentas CASE utilizadas no desenvolvimento do sistema, sistema operacional, bibliotecas de suporte, e assim por diante.

A configuração de software é, portanto, definida como uma lista dos ICS que compõem o software e de suas respectivas versões. Cada uma destas versões deve ser armazenada de maneira que possa ser recuperada sempre que uma determinada configuração do software precise ser reproduzida. A sessão 10.5 vai tratar de sistemas de controle de versão, que auxiliam o gerente de configuração a manter o controle das diferentes versões destes itens.

10.1.6 *Baseline* e *Release*

Uma *baseline* é uma configuração de software especialmente criada para uma situação específica. Uma *baseline* é formalmente aprovada em um determinado momento do ciclo de vida do software e servirá de referência para desenvolvimento posterior.

Uma *baseline* só pode ser modificada através de um processo formal de mudança. A *baseline*, juntamente com todas as mudanças aprovadas para ela, representa a configuração correntemente aprovada.

Um exemplo de *baseline* é um projeto que foi aprovado para execução. Uma vez aprovado por todos os interessados, ele só pode ser modificado mediante processos formais (ex. termos aditivos). Outro exemplo de *baseline* poderia ser um conjunto de requisitos e interfaces aprovados pelo cliente para desenvolvimento. Outro exemplo ainda poderia ser uma versão do sistema entregue ao cliente e que tenha sido aprovada nos testes de aceitação.

Em todos os casos a ideia da *baseline* é que ela deverá servir de base para desenvolvimentos posteriores e que o que ali está, a princípio, não deve ser mudado. Já no caso de uma configuração de software que não seja *baseline*, esta pode ser alterada sem maiores formalidades.

Também é possível considerar que mudanças feitas após o estabelecimento de uma *baseline* serão consideradas como mudanças "delta" até que uma nova *baseline* seja definida.

Uma *release* é uma distribuição de uma versão do software (ou mesmo de um ICS) para fora do ambiente e desenvolvimento. A vantagem do uso de sistemas de controle de versão reside no fato de que *releases* atuais ou anteriores podem ser geradas a qualquer momento a partir das *baselines* e das mudanças armazenadas para elas.

10.2 Controle de Versão

A falta de controle de versões pode levar a problemas bastante característicos. Responder "sim" a qualquer uma das questões abaixo indica que um projeto carece deste tipo de controle[130]:

a) Já perdeu uma versão anterior do arquivo do projeto e precisou dela?
b) Já teve dificuldade em manter duas versões diferentes do sistema rodando ao mesmo tempo?
c) Alguém já modificou indevidamente um código fonte e o original não poderia ter sido perdido?
d) Tem dificuldade em saber quem modificou o que em um projeto?

O controle de versão vai rastrear todos os artefatos do projeto (itens de configuração) e assim manter o controle sobre o trabalho paralelo de vários desenvolvedores através das seguintes funcionalidades:

a) Manter e disponibilizar cada versão já produzida para cada item de configuração de software.

[130] pt.wikipedia.org/wiki/Gerência_de_configuração_de_software

b) Ter mecanismos para disponibilizar diferentes ramos de desenvolvimento de um mesmo item, ou seja, diferentes variantes ou variações de um item poderão ser desenvolvidas em paralelo.
c) Estabelecer uma política de sincronização de mudanças que evite a perda de trabalho e o retrabalho.
d) Fornecer um histórico de mudanças para cada item de configuração.

O controle de versão vai assim manter um histórico dos ICS. Se uma alteração, por exemplo, nos métodos de uma classe é feita para modificar uma funcionalidade ou otimizar um procedimento qualquer e depois se descobre que ela não deveria ter sido feita, ou que foi feita de maneira inadequada e deve ser desfeita, então o controle de versão vai permitir desfazer a mudança (*undo*) e retornar o artefato ao seu estado anterior. Dessa forma, diferentes modificações podem ser tentadas em um artefato, seja código, diagrama ou texto, sem maiores riscos, já que qualquer modificação pode ser posteriormente desfeita. Tal característica é fundamental em qualquer projeto de desenvolvimento de software não trivial.

Não apenas os itens de configuração devem ser controlados pelo sistema de gerenciamento de configuração, mas também os relacionamentos entre eles (Seção 10.1.2). Alguns itens poderão ser compatíveis apenas com determinadas versões de outros itens, ou ainda, necessitar de conexões com outros itens a partir de uma determinada versão.

Apenas o teste de integração vai poder dizer se dois ICS são compatíveis entre si. Por exemplo, se uma classe *X* versão 1.0 dependia de métodos da classe *Y* versão 1.5, então se a classe *Y* for atualizada para a versão 1.6, talvez ela não seja mais compatível com a classe *X* versão 1.0.

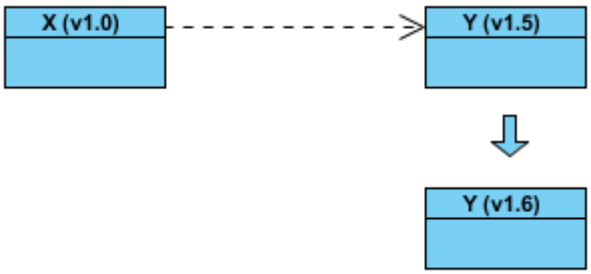

Figura 10-1: Uma classe que depende de uma versão desatualizada de outra classe.

Apenas depois que a integração entre *Y* 1.6 e *X* 1.0 passar nos testes é que essa nova dependência poderá ser aprovada. Até este ponto, a classe *X* 1.0 continuará a depender de *Y* 1.5 (com uma anotação no sistema de versões de que *Y* 1.5 está desatualizada e que uma nova integração é necessária).

10.2.1 Repositório

Todos os arquivos referentes às realizações dos itens de configuração ficam em um local chamado *repositório*, sob a guarda do sistema de controle de versão. O repositório pode ser entendido como um local (diretório) onde as diferentes versões de cada item são mantidas e identificadas. Esse controle poderá ser manual, mas o ideal é que seja automático.

Um bom sistema de controle de versões deverá ser capaz de otimizar o espaço de armazenamento do repositório. Usualmente, tal sistema deverá ser capaz de armazenar

apenas o artefato original e dali para frente armazenar apenas as modificações que forem feitas, e não novas cópias completas do mesmo artefato. Dessa forma, uma versão qualquer será produzida pelo sistema a partir da versão original, na qual o sistema aplicará sequencialmente todas as modificações até chegar à versão desejada.

Para otimizar velocidade de acesso, uma vez que normalmente a última versão é a que será efetivamente utilizada na maioria das vezes, o sistema pode armazenar em *cache* uma cópia completa desta última versão, sem que haja necessidade de gerá-la a partir da original e das modificações, como as anteriores. Quando esta versão deixar de ser a atual, essa cópia pode ser apagada, ficando no repositório apenas as modificações que permitem gerá-la a partir das versões anteriores e a copia da nova versão atual.

10.2.2 Políticas de Compartilhamento de Itens

Os desenvolvedores não trabalham diretamente sobre os itens do repositório, mas sobre cópias destes itens. Assim, um desenvolvedor deve solicitar acesso a um determinado item no repositório, obtendo uma cópia deste. Este desenvolvedor vai fazer as alterações nesta cópia do item e posteriormente salvar no repositório uma nova versão do item.

Deve-se decidir o que fazer caso de mais de um desenvolvedor esteja trabalhando sobre o mesmo item (concorrência). Neste caso, existem duas políticas usuais:

a) Política *trava-modifica-destrava* (*exclusive lock*), na qual um desenvolvedor que copia um item para modificá-lo deve travar o item no repositório, de forma que nenhum outro desenvolvedor poderá alterá-lo até que a modificação seja concluída e a nova versão salva. Esta política tem como vantagem o fato de evitar colisões, isto é, situações em que dois desenvolvedores alteram um item e as alterações do primeiro a salvar acabam sendo perdidas porque o segundo vai salvar suas próprias modificações sobre uma versão anterior às modificações do primeiro. Mas a desvantagem desta política reside no fato de que muitas vezes os desenvolvedores até poderiam trabalhar simultaneamente sobre um mesmo item, pois vão alterar partes diferentes dele, mas mesmo assim terão de esperar, perdendo tempo, ou realizando outras tarefas menos importantes.
b) Política *copia-modifica-resolve* (*optimistic merges*), na qual dois desenvolvedores ou mais podem trabalhar simultaneamente sobre um mesmo item e no momento de salvar a nova versão resolvem entre si possíveis interferências. Na prática os conflitos são raros e causados por falta de comunicação entre os desenvolvedores, e a mesclagem das versões pode ser feita automaticamente pelo sistema de controle de versões.

No caso da política copia-modifica-resolve, na maioria das vezes as alterações de um item vão ocorrer em locais diferentes, e o próprio sistema poderá resolvê-las. Nas poucas vezes em que as alterações ocorrerem na mesma parte do artefato, os desenvolvedores deverão decidir como tratá-las. Esse tipo de conflito, porém, normalmente ocorre somente quando há uma divisão de trabalho inadequada.

Quando um sistema de controle de versões é usado, normalmente o tratamento de conflitos na política copia-modifica-resolve ocorre da seguinte forma:

a) O desenvolvedor *A* pega uma cópia do item *X* para modificar (por exemplo, versão 1.1).
b) O desenvolvedor *B* pega outra cópia do mesmo item *X* para modificar (versão 1.1).
c) O desenvolvedor *A* salva (*commit*) uma nova versão de *X* com suas alterações (gerando a versão 1.2).
d) Quando o desenvolvedor *B* vai salvar sua cópia (baseada na versão 1.1), o sistema avisa que a versão do item está desatualizada (porque já existe a versão 1.2), e avisa que o desenvolvedor *B* deve fazer uma mesclagem (*merge*) de sua cópia com a versão 1.2. Um bom sistema vai mostrar exatamente quais linhas mudaram da versão 1.1 para 1.2 e vai avisar também caso o merge das duas versões implique em alterar as mesmas linhas.

Dessa forma, o desenvolvedor *B* deverá avaliar as alterações do desenvolvedor *A* e verificar se existe conflito entre estas e as que ele próprio produziu. Se necessário, os dois desenvolvedores devem conversar para resolver possíveis conflitos. Ao final do processo o desenvolvedor *B* salvará a versão 1.3 do item.

10.2.3 Envio de Versões

O *envio de versões* (*commit*) é feito normalmente a critério do desenvolvedor. Porém, é importante que uma nova versão de qualquer artefato só seja enviada ao repositório se ela estiver minimamente estável, isto é, razoavelmente livre de defeitos.

Isso pode ser garantido, no caso de artefatos de código, com a realização de testes de unidade, os quais devem, inclusive acompanhar o código como parte do item de configuração.

10.3 Controle de Mudança

O *controle de mudança* ou *gerência de mudança* é uma parte importante da gerência de configuração, a qual permite saber por que uma determinada versão de um ICS foi sucedida por outra.

Um típico controle de mudança de um sistema de software vai indicar quais funcionalidades foram adicionadas, removidas ou modificadas, quais defeitos foram corrigidos e, eventualmente, quais pendências ainda restam para uma versão futura. Por exemplo, um arquivo de controle de mudança de um sistema poderia conter a seguinte informação:

Mudanças da versão 2.2 para a versão 2.3:

- Correção do defeito D345;
- Correção do defeito D346;
- Adicionada a funcionalidade do requisito R43;
- Aprimorada a usabilidade da interface I12;

Pendências para uma versão posterior:

- Defeito D347;
- Melhorar a usabilidade da interface I13;

As modificações de uma versão para outra podem ser tanto aquelas que já estavam no plano de desenvolvimento do software, como por exemplo, a adição de funcionalidades referentes aos requisitos ou casos de uso próprios dos ciclos iterativos, ou a inclusão de mudanças solicitadas pelo usuário, quando este não fica totalmente satisfeito com as funcionalidades implementadas durante os testes de aceitação, ou ainda a inclusão de mudanças referentes a características que não foram incorporadas em um ciclo iterativo por falta de tempo e foram tiradas do escopo do ciclo. Na fase de produção (evolução e manutenção do software) o gerenciamento de mudança fará o controle das atividades de manutenção corretiva, adaptativa e perfectiva (Seção 14.2).

Alem deste tipo de controle, é possível haver um controle automatizado das mudanças, pois o sistema gerenciador de versões é capaz de comparar duas versões de um artefato e indicar exatamente quais elementos foram alterados e por quem. Desta forma, quando uma nova versão de um sistema é gerada a partir de um conjunto de itens de configuração, é possível também gerar automaticamente um descritivo das mudanças feitas. Porém, essa mudança será unicamente sintática, sendo usualmente necessário adicionar um descritivo que indique a motivação da mudança. Por exemplo, não basta informar que a linha X teve seu código alterado para tal sequencia de caracteres: é necessário informar *porque* isso foi feito.

10.4 Auditoria de Configuração

A auditoria de configuração é uma atividade que tem como objetivo verificar se os itens de configuração presentes em uma versão ou *baseline* são realmente aqueles que deveriam estar ali. Em relação ainda à *baseline* ou versão, a auditoria ainda deve verificar se todos os itens necessários estão realmente presentes no repositório.

A auditoria ainda pode ter como finalidade verificar a consistência da documentação fornecida ao usuário com a configuração de sistema entregue.

Uma auditoria de configuração pode ser executada seguindo os seguintes passos:

a) Preparar um relatório que liste cada item a ser verificado na *baseline* e o procedimento de teste a ser efetuado.
b) Efetuar os testes e anotar os itens que passaram no teste.
c) Se houver algum item que não passe no teste, anotar o fato do documento de *descobertas da auditoria*, para encaminhar ao setor responsável para providências.

10.5 Ferramentas para Controle de Versão

O controle de versões pode até ser feito manualmente, com a utilização de diretórios e padrões de nomeação de arquivos, mas esse tipo de controle não é muito efetivo quando se deseja obter relatórios de versões, ou ainda quando se realiza desenvolvimento de variantes. Além disso, essa forma de controle é bastante consumidora de espaço de armazenamento, pois cada versão será armazenada integralmente no diretório.

Assim, o mais adequado para uma empresa de desenvolvimento de software seria o uso de um sistema automatizado de controle de versões. Existem algumas soluções livres como:

a) *CVS*[131], ou *Concurrent Version System*, que é uma solução de código aberto baseada em arquitetura cliente-servidor. O sistema foi iniciado por Dick Grune em 1986.
b) *Mercurial*[132], que é um sistema de controle de versões baseado em linha de comando, desenvolvido por Matt Mackall.
c) *Git*[133], que é um sistema de controle de versão que enfatiza o aspecto de performance e foi desenvolvido inicialmente por Linus Torvalds para o desenvolvimento do *kernel* do Linux.
d) *SVN*[134], ou *Subversion*, que foi projetado para substituir e suprir algumas limitações do CVS.

Além disso, existem versões comerciais de software para controle de versões:

a) *SourceSafe*[135] da Microsoft.
b) Serena *PVCS Version Manager*[136].
c) *ClearCase*[137] da IBM, que é uma das ferramentas disponibilizadas para o RUP.

O SVN vem, de fato, substituindo o CVS, especialmente na comunidade de software livre.

[131] savannah.nongnu.org/projects/cvs/
[132] mercurial.selenic.com/
[133] git-scm.com/
[134] subversion.apache.org/
[135] msdn.microsoft.com/pt-br/library/ms181038%28v=vs.80%29.aspx
[136] www.serena.com/products/pvcs-pro/
[137] www-01.ibm.com/software/awdtools/clearcase/

