Capítulo
10

Gerenciamento de Configuração e Mudança

Este capítulo apresenta 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 uma *ferramenta* para controle de configuração e mudança (Seção 10.5).

O *gerenciamento de configuração de software* (GCS) 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 e após o projeto. Assim, diferentes versões de um mesmo componente ou produto precisam 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:

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

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

- O *controle de versão* vai dizer *o que* mudou e *quando*.
- O *controle de solicitações de mudança* vai dizer *por que* as coisas mudaram.
- 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 e mudança, alguns conceitos básicos são apresentados nas próximas subseções.

10.1.1 Item de Configuração de Software

Um *item de configuração de software* (ICS) 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 o código de programação deve ser controlado pelo gerenciamento de configuração, mas também a documentação, os diagramas, os planos, as ferramentas, os casos de teste, os dados e quaisquer outros artefatos relacionados ao desenvolvimento do software.

Em geral, cada item de configuração recebe um número, que poderá inclusive ser identificado

por várias partes que costumam ser separadas por ponto, como por exemplo:

- Plano de projeto, versão 2.
- Biblioteca de funções matemáticas, versão 4.1.
- 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 seu controle, e as configurações de software, formadas por um conjunto de ICS, serão listas muito extensas. De 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, em geral, 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 definidas como itens de configuração.

Cada item de configuração será definido por quatro elementos:

- Um *nome*, que é uma cadeia de caracteres curta que identifica claramente o item básico ou composto.
- Uma *descrição*, que consiste da definição do tipo de item (documento, diagrama, dados, código fonte etc.) e detalhes sobre ele.
- 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.
- 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.

Os itens de configuração podem ser básicos ou compostos. Os *básicos* são os artefatos arbitrados como individuais e indivisíveis para efeito de controle de versão. Os *compostos* podem ser formados por outros itens básicos e compostos.

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. Podem existir vários tipos de dependência ou relacionamento entre os itens de software. A maioria desses relacionamentos é prevista 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 a seguir, embora não seja exaustiva, apresenta alguns exemplos de relacionamentos entre ICS que também devem ser mantidos por um sistema de controle de configuração de software:

- *Dependência*: costuma indicar 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, nesse sentido, já que uma classe vai estar associada à outra normalmente para poder utilizar funções implementadas na outra.
- *Agregação* e *composição*: indicam que um componente é formado por outros.
- *Realização*: indica que um componente concreto é a implementação de um componente mais abstrato.
- *Especialização*: indica que um componente é uma variante mais específica de outro. Em geral,

o componente mais especializado depende do componente mais genérico. O inverso só será verdade se o componente genérico tiver métodos abstratos que precisem ser necessariamente implementados no componente mais especializado. Nesse caso, pode-se dizer que existe dependência mútua entre os componentes.

Dentre estas, especialmente a relação de dependência é importante para determinar quais testes de integração devem ser realizados quando ICS são atualizados.

10.1.3 RASTREABILIDADE

Rastreabilidade ou *controle de rastros* refere-se a um dos princípios da Engenharia de Software cuja implementação ainda implica significativa dificuldade. Manter um controle de rastros entre *módulos de código* não é difícil, porque as dependências entre os módulos costumam ser indicadas de forma sintática pelos próprios comandos da linguagem de programação (*import* ou *uses*, por exemplo). Contudo, 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 respectivos diagramas de sequência 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 feito manualmente (Cleland-Huang, Zemont & Lukasik, 2004).

Também são reportadas pesquisas que procuram automatizar a recuperação de relações de rastreabilidade entre artefatos a partir de evidências sintáticas. Porém, em razão das escolhas arbitrárias dos nomes de artefatos (falta de padrão), esses sistemas de recuperação costumam retornar muitos falso-positivos, o que inviabiliza seu uso (Settimi, Cleland-Huang, Khadra, Mody, Lukasik & de Palma, 2004).

Outra abordagem experimental apresentada por Santos e Wazlawick (2009). Consiste em modificar a maneira como os 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 podem ser criados como uma derivação de algum outro elemento que já exista.

Dessa 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 vêm do cliente, ou seja, são externos ao projeto) podem ser adicionados ao repositório sem que isso ocorra por derivação de outros itens.

No geral, porém, na maioria dos casos, caberá à equipe de desenvolvimento definir e executar uma política de registro de artefatos no sistema de gerenciamento de versões de forma que as dependências entre diferentes documentos (por exemplo, requisitos e código) esteja registrada e seja mantida atualizada.

10.1.4 VERSÕES DE ITENS DE CONFIGURAÇÃO DE SOFTWARE

A *versão* de um ICS é um estado particular desse item durante o desenvolvimento de um sistema. Normalmente, como já foi visto, uma versão é identificada por um número composto por uma, duas ou três partes.

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

- Uma *revisão*, que é uma nova versão de um item que foi criada para substituir uma versão anterior.
- Uma *variante*, que é uma nova versão do item que será adicionada à configuração sem a intenção de substituir uma versão antiga.

Pode-se falar, ainda, em *versões experimentais* para determinados itens: uma revisão é feita no item, mas ainda não está decidido se a nova versão será efetivamente mantida. Nesse caso, a revisão pode até ser considerada uma variante, já que ainda não existe 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 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 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 suas respectivas versões. Cada uma dessas versões deve ser armazenada de maneira que possa ser recuperada sempre que determinada configuração do software precise ser reproduzida.

10.1.6 *BASELINE* E *RELEASE*

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

A *baseline* só pode ser modificada através de um processo formal de mudança. 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 e formalizado por um contrato de desenvolvimento. Uma vez aprovado por todos os interessados, ele só pode ser modificado mediante processos formais (por exemplo, termos aditivos). Outro exemplo de *baseline* seria um conjunto de requisitos e interfaces aprovados pelo cliente para desenvolvimento. Outro exemplo, ainda, seria uma versão do sistema entregue ao cliente e aprovada nos testes de aceitação.

Em todos os casos, a ideia da *baseline* é que ela é considerada uma versão acordada e aprovada pelas partes; quaisquer novas versões geradas a partir dela não têm automaticamente o mesmo *status* de versão acordada. Porém, ao longo de um processo de desenvolvimento, é normal que a evolução de um produto continue com sucessivas versões sendo geradas no tempo até que um processo formal de homologação transforme uma dessas novas versões em uma nova *baseline*.

Release ou *entrega* é a 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 versão pode levar a problemas bastante característicos. Responder “sim” a qualquer uma das questões a seguir indica que um projeto carece desse tipo de controle:

- Já perdeu uma versão anterior do arquivo do projeto e precisou dela?
- Já teve dificuldade em manter duas versões diferentes do sistema rodando ao mesmo tempo?

- Alguém já modificou indevidamente um código-fonte e o original não poderia ter sido perdido?
- 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 manter o controle sobre o trabalho paralelo de vários desenvolvedores através das seguintes funcionalidades:

- Manter e disponibilizar cada versão já produzida para cada item de configuração de software.
- Ter mecanismos para disponibilizar diferentes ramos de desenvolvimento de um mesmo item, ou seja, diferentes variantes de um item poderão ser desenvolvidas em paralelo.
- Estabelecer uma política de sincronização de mudanças que evite a perda de trabalho e o retrabalho.
- Fornecer um histórico de mudanças para cada item de configuração.

O controle de versão vai, assim, manter um histórico dos ICS. Se, por exemplo, foi feita uma alteração nos métodos de uma classe para modificar uma funcionalidade ou otimizar um procedimento qualquer e depois se descobriu 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 fazer a versão atual do artefato retornar a uma versão anterior. Dessa forma, diferentes modificações podem ser tentadas em um artefato, seja código, diagrama, seja texto, sem maiores riscos, já que qualquer modificação pode ser desfeita posteriormente. Tal característica é fundamental em qualquer projeto de desenvolvimento de software não trivial.

Para que fique bem claro, cada versão que é gravada (*comitada*) no sistema de gerenciamento de versões passa a ser *read-only* (apenas leitura). Isso significa que nenhuma versão de um artefato pode ser alterada, mas pode-se obter uma cópia dela e produzir uma nova versão a partir de alterações na cópia.

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 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).

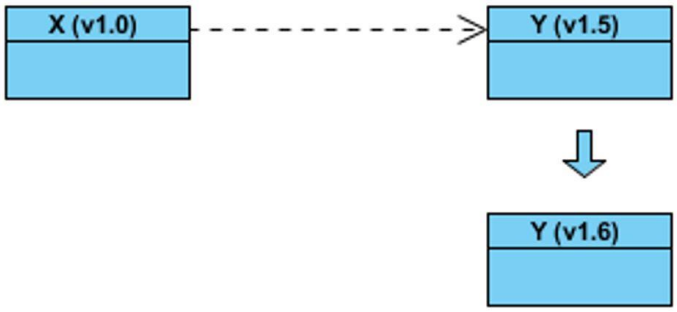

Figura 10.1 Uma classe que ainda não foi testada com uma nova versão de outra classe

Apenas depois que a combinação entre Y 1.6 e X 1.0 passar nos testes de integração é que essa nova dependência poderá ser aprovada. Até esse ponto, a classe X 1.0 continuará a depender de Y

1.5, com uma anotação no sistema de controle de versões de que Y 1.5 está desatualizada e que novos testes de integração são necessários.

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 (banco de dados) em que as diferentes versões de cada item são automaticamente mantidas e identificadas.

Um bom sistema de controle de versões deverá ser capaz de otimizar o espaço de armazenamento do repositório. Em geral, tal sistema deverá ser capaz de armazenar apenas o artefato original e, dali para a 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 a 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 dessa última versão, sem que haja necessidade de gerá-la a partir da original e das modificações, como as anteriores. Quando essa versão deixar de ser a atual, a cópia pode ser apagada, ficando no repositório apenas as modificações que permitem gerá-la a partir das versões anteriores e a cópia 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 desses itens. Assim, um desenvolvedor deve solicitar acesso a determinado item no repositório, obtendo uma cópia dele. Esse desenvolvedor vai fazer as alterações nessa cópia do item e, posteriormente, salvar no repositório uma nova versão do item.

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

- 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 seja salva. Essa 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 dessa política reside no fato de que, muitas vezes, os desenvolvedores até poderiam trabalhar simultaneamente sobre um mesmo item, pois vão alterar partes diferentes dele, e mesmo assim terão de esperar, perdendo tempo ou realizando outras tarefas menos importantes. Esse problema fica mais crítico à medida que os ICS ficam maiores.
- Política *copia-modifica-resolve* (*optimistic merges*), na qual dois desenvolvedores ou mais podem trabalhar simultaneamente sobre um mesmo ICS e, no momento de salvar a nova versão, resolver 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:

- O desenvolvedor *A* pega uma cópia do item *X* para modificar (por exemplo, versão 1.1).
- O desenvolvedor *B* pega outra cópia do mesmo item *X* para modificar (versão 1.1).
- O desenvolvedor *A* salva (*commit*) uma nova versão de *X* com suas alterações (gerando a versão 1.2).
- 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 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 também se o *merge* das duas versões implica 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

Normalmente, o *envio de versões* (*commit*) é feito a critério do desenvolvedor. Porém, é importante que uma nova versão de qualquer artefato só seja enviada ao repositório se estiver minimamente estável, isto é, razoavelmente livre de defeitos e revisada em relação a padrões de escrita.

No caso de artefatos de código, isso pode ser garantido com a realização de testes de unidade, os quais devem inclusive acompanhar o código como parte do item de configuração. Estes mesmos testes de unidade poderão ser usados para compor os testes de integração quando aplicados a um *build* do sistema em vez de a um item isolado.

10.3 Controle de Mudança

O *controle de mudança* ou *gerência de solicitações de mudança* é uma parte importante da gerência de configuração que permite saber por que determinada versão de um ICS foi sucedida por outra. Um bom sistema de controle de versões consegue identificar quais linhas foram alteradas em um artefato. Mas ele não saberá indicar *por que* essas linhas foram alteradas. O motivo da alteração de um artefato da versão 1.2 para a 1.3 poderia, ser, por exemplo, a correção de um defeito no código. Então, cabe ao controle de mudança manter juntamente com o controle de versão o histórico dos motivos que levaram a cada alteração nos artefatos.

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 produto de software 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 a adição de funcionalidades referentes aos requisitos ou casos de uso próprios das iterações, 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 uma iteração por falta de tempo e foram retiradas 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).

Além desse 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. Dessa 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 descrição será unicamente sintática, costumando ser necessário adicionar as informações que indiquem a motivação da mudança. Por exemplo, não basta informar que a linha X teve seu código alterado para tal sequência de caracteres; é necessário informar *por que* isso foi feito. Se a ferramenta de gerenciamento de mudança for integrada à ferramenta de distribuição de tarefas entre os desenvolvedores, então mesmo essas descrições de motivação poderão ser obtidas de forma praticamente automática, pois já estarão na descrição da tarefa a ser feita.

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 à *baseline* ou versão, a auditoria deve ainda verificar se todos os itens necessários estão realmente presentes no repositório.

A auditoria também 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 passos abaixo:

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

Por vezes, poderá ser necessário realizar auditorias de versões antigas de um produto. Por exemplo, um determinado usuário de uma versão antiga do produto pode alegar ter enfrentado problemas e demanda indenização. Assim, se a organização desenvolvedora for capaz de gerar essa versão anterior para que seja analisada pela auditoria, a real causa do problema poderá ser identificada ou afastada.

10.5 Ferramenta para Controle de Versão

O controle de versões em projetos de software já não é mais feito manualmente, com a utilização de diretórios e padrões de nomeação de arquivos, porque esse tipo de controle não é efetivo quando se deseja obter relatórios de versões ou quando se realiza o desenvolvimento de variantes. Além disso, essa forma de controle consome espaço de armazenamento, pois cada versão é 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. Atualmente (2018) a ferramenta mais largamente usada para este fim é Git um projeto de código aberto originalmente desenvolvido por Linus Torvalds a partir de 2005 (ver *QR code*). [qrc 10.1]

Git é considerado uma ferramenta de controle de versões distribuída, já que, ao contrário de outras, que tem apenas um repositório para os itens, ela faz com que cada desenvolvedor que esteja trabalhando em um repositório tenha uma cópia completa deste.

A ferramenta permite que se trabalhe simultaneamente em várias ramificações (*branch*) de um projeto. Por exemplo, um analista poderá estar trabalhando na implementação da versão 2.0 de um item, a qual será futuramente incluída em um *build* e poderá momentaneamente retomar a versão 1.3 do mesmo item, já incluída em uma *release*, para corrigir um defeito, gerando assim a versão 1.3.1 deste item, e depois retornar a trabalhar na versão 2.0.

REMISSIVO DO CAPÍTULO