Aula 1º

Introdução à gerência de configuração

Olá, alunos(as),

Bem-vindos(as) a nossa primeira aula da disciplina "Gerência de Configuração". Veremos uma visão geral a respeito da Gerência de Configuração. Vocês vão entender o que é a Gerência de Configuração e quais são as atividades que são incluídas nessa disciplina.

Além disso, vocês saberão o motivo da existência desta disciplina e alguns termos que são muito utilizados.

Assim, não se esqueçam de ler esta aula, assistir aos nossos vídeos e fazer as atividades. Se tiverem alguma dúvida, vocês poderão usar o nosso quadro de avisos, que eu responderei.

Bons estudos!

Objetivos de aprendizagem

Ao término desta aula, vocês serão capazes de:

- saber o motivo da existência da Gerência de Configuração;
- entender o que é a Gerência de Configuração;
- conhecer alguns termos da Gerência de Configuração.

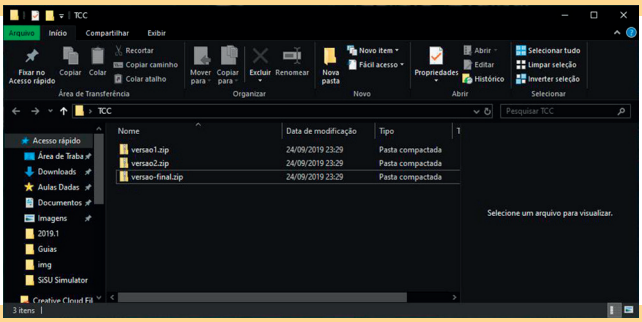
Seções de estudo

1 - O que é a gerência de configuração
2 - Terminologia de gerência de configuração

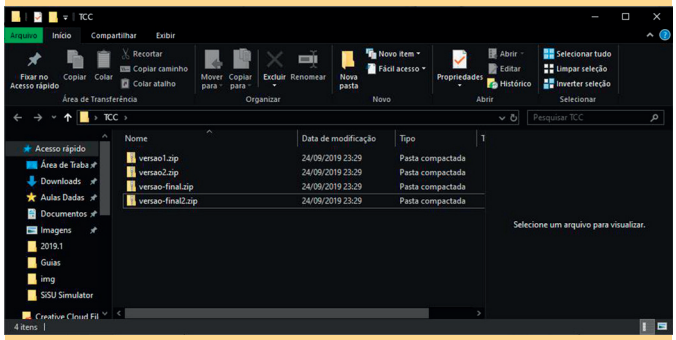
1 - O que é gerência de configuração

Antes de começarmos a ver o que é a Gerência de Configuração, conhecer ver algumas histórias. Veja se você se identifica ou já presenciou algumas dessas histórias que apresentaremos a seguir:

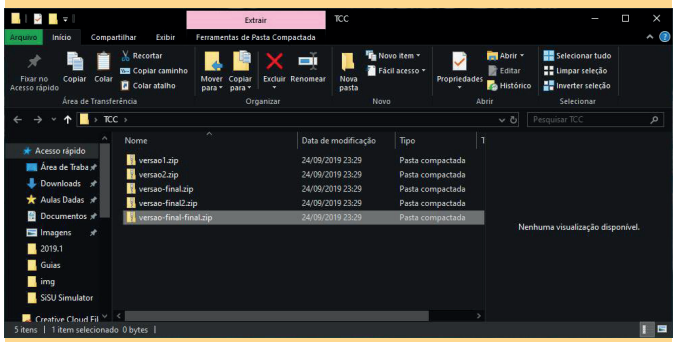
História #1: Você está a 1 dia de apresentar o seu novo projeto ao cliente. Você fez três versões e acredita que a terceira é a final. Assim, você dá o nome de versão-final.zip, como consta na imagem abaixo:



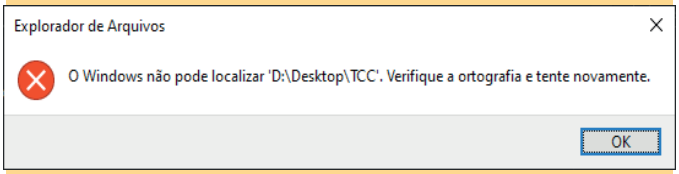
Eis que ao executar novamente a versão final, você descobre que precisa de mais algumas correções. Assim, você decide fazer mais algumas correções e decide nomear de versão-final2.zip.



Porém, mais erros são descobertos. E você os corrige. A partir daí, você dá o nome de versão-final-final.zip.



Seguro que essa versão é a correta, você despluga tudo do seu notebook e coloca o seu computador na bolsa, achando que tudo vai dar certo. Na hora de apresentar ao cliente, você se pergunta: cadê o código?



Você descobre que as versões estavam em um pendrive – que você desplugou e está em casa, bem longe do seu cliente...

Situação #2: Você está fazendo uma das atividades da sua matéria. Você está implementando um cadastro importante, cujo código tem mais de 100 linhas. Mas, você descobre que o cadastro está "errado" e decide apagar tudo. Com isso, você reescreve o seu cadastro do zero, se preocupando em fazer a melhor implementação possível.
Mas, enquanto você estava fazendo a reescrita, você pensa: "Poxa... Essa parte da inserção poderia ter mantido... Estava tudo certo. 200 linhas que poderia ter deixado no arquivo. Seria menos tempo para eu fazer as mudanças..."

Situação #3: Você e seu colega estão trabalhando com um projeto na sua empresa. E eis que quando você está testando as mudanças... Você verifica que o sistema parou de funcionar. Assim, você decide averiguar o que aconteceu, vendo o código do módulo que parou de funcionar.
Ao auditar o código, você descobre que o seu colega acabou sobrescrevendo o arquivo, colocando uma versão que quebrou o sistema. Como não havia uma versão anterior salva no seu projeto, você vai dedicar o resto do dia para consertar o erro do seu colega, e, talvez, você gaste o seu fim de semana...

Situação #4: Você está trabalhando em uma equipe de desenvolvimento de um software customizado para um cliente. Eis que o seu cliente pede para que vocês resgatem a versão 1.0 do sistema, desenvolvido há dois anos e usado durante todo o ano passado, para verificar se o sistema causou um erro de cálculo no último ano fiscal. Porém, como não existe um histórico consolidado de versões na empresa, sua equipe começa a sofrer para achar essa versão. Após vasculhar em vários computadores, não se acha nada e o seu cliente fica aborrecido, achando que o software causou o erro, que na verdade, foi causado por um funcionário vingativo da empresa cliente. Porém, esse erro nunca será descoberto, por falta de provas...

As quatro situações citadas aqui são inúmeras que podem ocorrem em um projeto de desenvolvimento de software, quando não se aplicam processos de controle. Sommerville cita mais algumas:

> Se você não tem procedimentos de gerenciamento de configuração efetivos, você pode desperdiçar esforço modificando a versão errada de um sistema, entregá-la para os clientes ou esquecer onde está armazenado o código-fonte do software para uma versão específica do sistema ou componente. (SOMMERVILE, 2011, p. 475)

Isso são apenas algumas situações, pois como você deve ter estudado desde os primórdios da sua jornada neste curso,

um software é um produto mutável:

> Os sistemas de software sempre mudam durante seu desenvolvimento e uso. Bugs são descobertos e precisam ser corrigidos. Os requisitos do sistema mudam e é preciso implementar essas mudanças em uma nova versão do sistema. Novas versões do hardware e novas plataformas de sistema tornam-se disponíveis e você precisa adaptar seus sistemas para trabalhar com elas. Os concorrentes introduzem novos recursos em seu sistema, aos quais você precisa corresponder. Mudanças são feitas para o software e cria-se uma nova versão de um sistema. **Portanto, a maioria dos sistemas pode ser pensada como um conjunto de versões, sendo que cada uma delas necessita ser mantida e gerenciada**. (SOMMERVILLE, 2011, p. 475, grifo nosso)

Perceba que grifamos a última parte neste texto. Reflita que essa colocação de Ian Sommerville faz sentido, pois a maioria dos softwares tem seus marcos registrados em versões. Em cada versão existe um conjunto de recursos que são adicionados, um conjunto de recursos que são modificados e (talvez) um conjunto de recursos que são removidos.

Para instigarmos a nossa reflexão, observe a evolução de um software famoso, como o Microsoft Windows™. Durante seus mais de 30 anos de história, ele passou por diversas versões, com vários marcos de evolução:

Figura 1 – A evolução das versões do Microsoft Windows™

Fonte: Disponível em: https://www.supinfo.com/articles/resources/228031/4881/0.png. Acesso em 11 out. 2019.

Considerando o *software* como algo em evolução, onde sua evolução está estruturada em versões, precisamos ter alguma forma de gerenciar as versões e as suas mudanças, para que possamos ter controle no projeto:

> Você precisa gerenciar os sistemas em evolução, pois é fácil perder o controle de quais mudanças e versões de componentes foram incorporadas em cada versão de sistema. As versões implementam propostas de mudanças, correções de defeitos e adaptações de hardware e sistemas operacionais diferentes. Pode haver várias versões em desenvolvimento e em uso ao mesmo tempo. (SOMMERVILLE, 2011, p. 475)

Assim, com essas reflexões em mente, fica constatada a necessidade de termos processos e ferramentas para gerenciar as versões e as mudanças do sistema. Assim, existe a **Gerência de Configuração**, que é considerada uma das disciplinas da Engenharia de Software. Vamos a sua definição, segundo Ian Sommerville:

> O gerenciamento de configuração (CM, do inglês *configuration management*) está relacionado com as políticas, processos e ferramentas para gerenciamento de mudanças dos sistemas de software. (...)
> Políticas e processos de gerenciamento de configuração definem como gravar e processar propostas de mudanças de sistema, como decidir quais componentes de sistema alterar, como gerenciar diferentes versões de sistema e seus componentes e como distribuir as mudanças para os clientes.
>
> ***(SOMMERVILLE, 2011, p. 475-6, grifo do autor)***

Hélio Engholm Jr. nos oferece a seguinte definição, e justifica a necessidade de adotarmos processos de Gerência de Configuração:

> O gerenciamento de configuração é necessário para que possamos controlar os itens de configuração do projeto e o versionamento dos mesmos identificando as características funcionais e físicas de um produto, serviço ou componente e controlando todas as mudanças realizadas nessas características. O gerenciamento de configuração relaciona-se às funcionalidades e características físicas do aplicativo, documentação de hardware e software utilizados no projeto, além de versões de aplicativos, banco de dados e frameworks, entre outros. (ENGHOLM JR., 2010, p. 235)

Com a definição exposta, vamos falar sobre as vantagens de adotar processos de Gerência de Configuração no seu projeto. Se você tiver trabalhando sozinho, a Gerência de Configuração pode ser útil, para que você tenha controle de quais mudanças foram feitas no sistema, de forma ágil. Se você trabalha em uma equipe, principalmente de forma remota, adotar processos e ferramentas da Gerência de Configuração pode ser útil para que as "as equipes tenham acesso a informações sobre um sistema que está em desenvolvimento e não interfiram no trabalho umas das outras". (SOMMERVILLE, 2011, p. 476)

Além disso, adotar processos e ferramentas de gerência de configuração significa também cumprir um passo para obter certificações de qualidade. Isso se deve pelo fato de que a Gerência de Configuração é exigida para as certificações ISO 9000, CMM e CMMI. Sommerville (2011) ressalta que algumas normas, como a IEEE 828-1998 (atualizada em 2005 e 2012), definem normas para a Gerência de Configuração. As empresas normalmente criam os seus planos e processos de acordo com as normas padrão.

Agora que você sabe sobre o que é a Gerência de Configuração e as vantagens em adotar seus processos e ferramentas, vamos ver em quais áreas esta disciplina atua. Vamos lá?

1.1 – Atividades da Gerência de Configuração

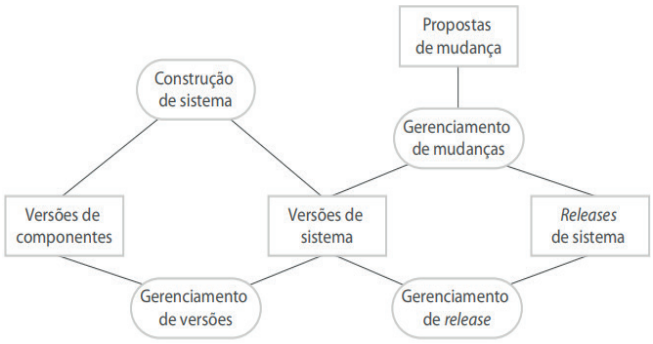
Existem quatro áreas (também denominadas de atividades afins) que as ferramentas e processos de Gerência

de Configuração podem atuar. Vamos ver quais são?

- **Gerenciamento de Mudanças**: tem o objetivo de controlar as solicitações de mudanças do sistema, que podem envolver a programação de novas funcionalidades ou a resolução de bugs do sistema. Nesse processo, o gerenciamento de mudanças faz a estimativa dos custos e do impacto que essa mudança pode provocar e guiar a equipe do projeto para decidir se a mudança deve ser implementada ou não;
- **Gerenciamento de versões**: se encarrega de acompanhar a evolução das versões do sistema e assegurar que as mudanças nos componentes não causem interferência entre elas;
- **[Gerenciamento da] Construção do Sistema**: se encarrega de fazer a montagem dos componentes, dados e bibliotecas, executar a compilação e a ligação destes, para a criação de um sistema executável.
- **Gerenciamento de Releases (também denominado de Gerenciamento de Lançamentos)**: se encarrega de preparar o software para a *release* (lançamento) e manter um histórico de quais versões foram lançadas para uso externo (SOMMERVILLE, 2011).

As ferramentas e os processos de Gerência de Configuração podem se encaixar em uma ou mais áreas do sistema, de acordo com as suas funcionalidades. Nas aulas 02 e 03, vamos ver com mais detalhes cada uma dessas áreas. A Figura 2 mostra graficamente as áreas da Gerência de Configuração.

Figura 2 – Atividades da Gerência de Configuração



Fonte: SOMMERVILLE, 2011, p. 476

Vamos agora ver algumas terminologias que podem ser usadas na Gerência de Configuração. Vamos lá?

2 - Terminologia de gerência de configuração

Apesar da Gerência de Configuração ser algo discutido em padrões de mercado, existem alguns termos usados nesta disciplina que não são uniformizados, ou seja, temos empresas diferentes usando termos diferentes para mesmas coisas.

A razão para isso se deve a um fato histórico. Os primeiros projetos que usaram processos da Gerência de Configuração de Software foram de base militar, usando processos já existentes na Gerência de Configuração de Hardware. Os desenvolvedores, que não se acostumaram com alguns termos, passaram a criar os seus próprios termos. Além disso, os métodos ágeis criaram também uma nomenclatura para alguns termos da Gerência de Configuração, para criar uma distinção entre a abordagem ágil e a abordagem padrão. (SOMMERVILLE, 2011)

A seguir, vamos descrever alguns termos que são usados na Gerência de Configuração, segundo o livro do renomado professor Ian Sommerville.

Primeiramente, temos o termo **Controle de Configuração**, que denomina o processo de garantia de que versões de sistemas e componentes sejam "registradas e mantidas para que as mudanças sejam gerenciadas e todas as versões de componentes sejam identificadas e armazenadas por todo o tempo de vida do sistema" (SOMMERVILLE, 2011, p. 477).

Em seguida, vemos o **item de configuração**, também denominado de **item de configuração de software**. Sommerville (2011) diz que esse termo é dado a qualquer coisa que pertença a um projeto de software que seja colocada sobre o controle de configuração – projeto, código, dados de testes, documentos, modelos etc.

Podem ser enquadrados como exemplos de itens de configuração:

- código-fonte e compilado do software desenvolvido;
- documentação do projeto, requisitos e especificações;
- sistema operacional, frameworks, banco de dados;
- características do hardware e arquitetura utilizada no sistema;
- qualquer outro item que possa sofrer mudanças durante o ciclo de vida do projeto e que possa ser rastreado. (ENGHOLM JR., 2010, p. 235-6)

Para entendermos a necessidade de adotarmos um controle abrangente dos itens de configuração, observe este exemplo:

A importância de se manter um controle abrangente dos itens de configuração

Imagine um sistema desenvolvido para um caixa eletrônico de uma agência bancária. Com o passar dos anos, a instituição financeira passa a utilizar outros hardwares com novas versões de sistemas operacionais e frameworks. Desse modo, várias versões do aplicativo, com funcionalidades diferentes e para serem usadas em configurações de hardware diferentes podem ter sido criadas. Para que se possa reinstalar a aplicação em uma máquina mais antiga, torna-se extremamente necessário o conhecimento de toda a sua configuração, partindo do levantamento da versão de todos os itens de configuração utilizados pela máquina, desde o hardware utilizado a versões de todos os softwares que devem ser instalados. Para que essa tarefa seja possível, deve-se realizar um catálogo contendo o versionamento de todos os itens de configuração utilizados e encontrar uma maneira de recuperá-los. Imagine a situação de tentar instalar uma versão multithreading de um sistema contendo o DOS como sistema operacional.

Fonte: ENGHOLM JR., 2010, p. 236.

Esse item deve ter um nome único para ser identificado e podem ter diferentes versões desse mesmo item. Um exemplo pode ser um Documento de Requisitos do sistema, ou um arquivo de código-fonte do projeto de desenvolvimento de software.

Já que entramos no assunto de **versão**, vamos à definição: É uma instância de um item de configuração que possui uma diferença entre outras versões dessa mesma instância. Cada versão possui um identificador único, que normalmente é composto pelo nome do item junto com um número crescente, que identifica qual é a versão.

Em um documento de requisitos ou em um componente de *software*, começamos fazendo a versões betas, com os números se aproximando de 0: 0.1, 0.2, 0.3... Em seguida, quando temos a primeira versão pronta, lança-se a versão 1.0, e a partir daí, novas versões surgem. Sendo que algumas possuem um incremento maior de número e outras possuem um incremento menor: 1.1, 1.2, 2.0, 2.1, 2.2, e assim em diante.

Ainda sobre o conceito de versões, é importante mencionar o conceito de **releases**, que são as versões de um sistema que foram liberadas para uso fora do contexto de desenvolvimento, ou seja, foram liberadas para os clientes ou outros usuários de uma organização. (SOMMERVILLE, 2011).

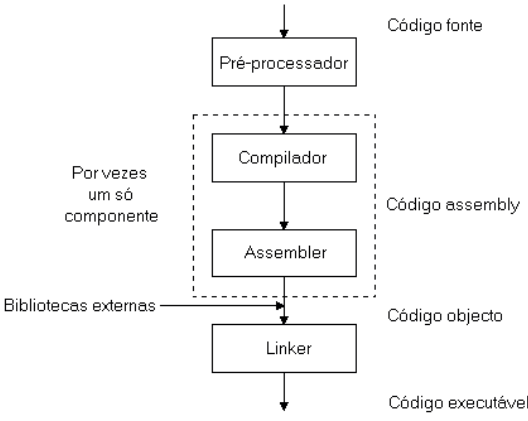
Para finalizar essa primeira parte, temos o conceito de **construção de sistema**, que se refere ao processo de "criação de uma versão de sistema executável pela compilação e ligação de versões adequadas dos componentes e bibliotecas que compõem o sistema" (SOMMERVILLE, 2011, p. 477)

Quem recordar as aulas de Linguagem de Programação, deve se lembrar dos seus primeiros programas que foram escritos na sua jornada nesse curso. Esses programas foram escritos em Linguagem C e C++, que usam os compiladores GCC e G++. Esses compiladores fazem muito bem à tarefa citada no parágrafo anterior, pois:

- Converte cada arquivo de código-fonte para código intermediário, e posteriormente para código-objeto.
- Liga os arquivos de códigos-objetos e faz a ligação, com o intuito de criar um programa executável.

A figura, a seguir, mostra o ciclo de compilação e ligação de um programa na linguagem C. Vamos ver?

Figura 3 – Ciclo de compilação e ligação de um programa C/C++



Fonte: FELGUEIRAS, Carlos Alberto. Capítulo 1 - O Programa em C. s.d. Disponível em: http://www.dpi.inpe.br/~carlos/Academicos/Cursos/LinguagemC/Cap_1.html. Acesso em 23 out. 2019.

A seguir, vamos apresentar três conceitos que são relacionados: codeline, baseline e mainline. Vamos lá?

2.1 – Codeline, baseline e mainline

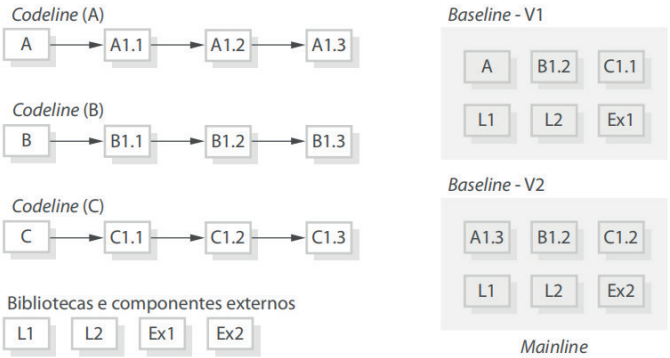
Outros conceitos que devemos mencionar são o de baseline, codeline e mainline. Esses conceitos, por trabalharem em um mesmo campo de estudo, podem parecer, à primeira vista, similares, mas não são. Primeiro, vejamos o conceito de **baseline**:

> Uma baseline é uma coleção de versões de componentes que compõem um sistema. As baselines são controladas, o que significa que as versões dos componentes que constituem o sistema não podem ser alteradas. Isso significa que deveria sempre ser possível recriar uma baseline a partir de seus componentes. (SOMMERVILLE, 2011, p.477)

Já o **codeline** não é uma coleção, mas sim "um conjunto de versões de um componente de software e outros itens de configuração a qual esse componente depende", enquanto uma **mainline** se refere a uma sequência de baselines, ou seja, um conjunto de diferentes versões de um sistema. (SOMMERVILLE, 2011, p.477)

Para que você possa entender de uma vez por todas a diferença entre os três termos, vamos apresentar a figura abaixo. Observe atentamente antes de prosseguir com a leitura:

Figura 4 – Apresentação visual dos conceitos de codeline, baseline e mainline



Fonte: SOMMERVILLE, 2011, p. 482

Cada retângulo de fundo branco nomeado corresponde a uma versão de um componente ou biblioteca do sistema. Em nosso sistema hipotético, existem três componentes desenvolvidos: A, B e C. Sendo que cada componente possui um conjunto de versões para ela: 1.1, 1.2, 1.3...

O conjunto de versões para cada componente é a representação de uma codeline. Assim, o conjunto de versões do componente A é uma codeline, e o conjunto de versões do componente B é uma codeline.

Agora, observe que no canto direito da figura existem

dois retângulos sombreados. Cada retângulo sombreado representa uma baseline, pois esses retângulos representam um conjunto de versões de componentes. Perceba que em cada baseline, existe uma determinada versão dos componentes A, B e C, além de algumas bibliotecas necessárias.

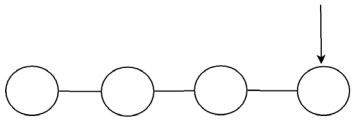
O agrupamento de duas ou mais baselines representa uma mainline. Assim, o conjunto dos retângulos sombreados compõe a mainline do nosso sistema hipotético.

Guarde esses conceitos, pois vamos utilizar bastante ainda no decorrer de nossa disciplina. Por enquanto, voltamos as nossas definições.

2.2 – Branching e Merging

Agora, vamos ver mais dois conceitos que se relacionam: Branching e Merging. Esses dois conceitos tratam diretamente sobre o controle de versões, mais especialmente quando estamos falando de codelines. Na seção anterior, você viu que uma codeline é uma sequência de versões de um componente de software. Para ilustrar, a figura a seguir mostra uma codeline de um componente de software. Cada círculo presente nele representa uma versão desse componente hipotético. A seta indica a versão atual em que o programador está trabalhando esse componente.

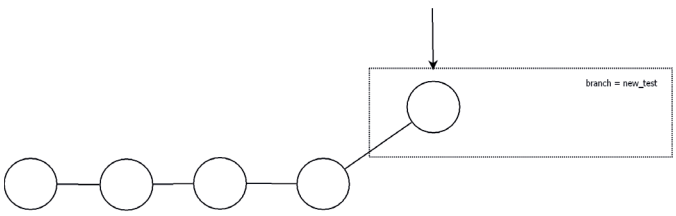
Figura 4 – Codeline hipotética de um componente de software.



Fonte: Acervo Pessoal.

Vamos supor que o programador esteja fazendo a implantação de uma nova funcionalidade, sendo que essa funcionalidade deve ser testada antes de ser incorporada a base principal do código. Assim, o programador decide criar uma ramificação na codeline, onde é criada uma nova sequência de versões, totalmente independente da sequência que originou. Isso se chama **branching** e essa nova codeline é chamada de **branch** (ramificação). Observe na figura a seguir o desvio, indicando a criação de uma branch.

Figura 5 – Criação de uma branch nova.



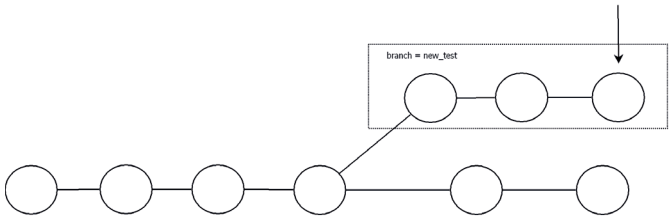
Fonte: Acervo Pessoal.

Branching: Trata-se da criação de uma nova codeline de uma versão em uma codeline existente. A nova codeline e uma codeline existente podem, então, ser desenvolvidas independentemente.

Fonte: SOMMERVILLE, 2011, p. 477

No decorrer do desenvolvimento da nova funcionalidade, o programador cria versões tanto na branch principal, quanto na branch que abriga a nova funcionalidade, como pode ser visto na imagem a seguir. Quando criamos uma branch, podemos adicionar novas versões tanto a essa nova branch quanto a branch que originou ela.

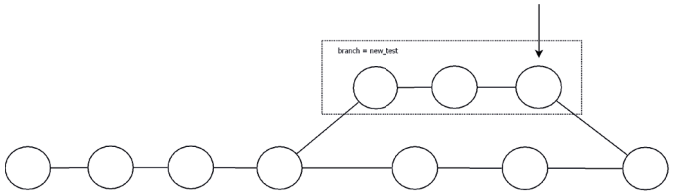
Figura 6 – Adição de novas versões a ambas as branches



Fonte: Acervo Pessoal.

Ao final, o programador conclui que a nova funcionalidade está pronta para ser colocada na branch principal do sistema. Assim, ele decide fundir a branch a qual estava trabalhando a sua funcionalidade de testes com a branch principal, criando uma versão a partir dessa fusão. Essa ação que o programador fez se chama **merging** (mesclagem).

Figura 7 – O programador decide mesclar a branch de testes com a branch principal.



Fonte: Acervo Pessoal.

Merging: Trata-se da criação de uma nova versão de um componente de software, fundindo versões separadas em diferentes codelines. Essas codelines podem ter sido criadas por um branch anterior de uma das codelines envolvidas.

Fonte: SOMMERVILLE, 2011, p. 477

Vale lembrar que esse ciclo pode se repetir várias vezes no processo de desenvolvimento de software. Veremos com mais detalhes esse processo quando estudarmos as ferramentas para a gerência de versões de software, que nos permitem armazenar um histórico de versões do nosso sistema.

Por fim, vamos ver o conceito de **espaço de trabalho**, que também é utilizado quando falamos a respeito do Gerenciamento de Versões. Esse nome é dado a "uma área de trabalho privada em que o software pode ser modificado sem afetar outros desenvolvedores que possam estar usando ou modificando o software" (SOMMERVILLE, 2011, p. 477).

E, com isso, encerramos a nossa aula. Na próxima, vamos ver as duas primeiras atividades da Gerência de Configuração: Gerenciamento de Mudanças e Gerenciamento de Versões.

Até lá!

Retomando a aula

Chegamos ao final da nossa primeira aula. Vamos relembrar?

1 - O que é a gerência de configuração

Nesta seção, você viu algumas situações que podem ocorrer quando você não adota a Gerência de Configuração em seus projetos. Além disso, você viu que a Gerência de Configuração é uma disciplina da Engenharia de Software que está relacionada com as políticas, processos e ferramentas para gerenciamento de mudanças dos sistemas de software e é exigida nas principais certificações de qualidade de software do mercado.

2 - Terminologia de gerência de configuração

Nesta seção, vimos alguns termos que são utilizados na Gerência de Configuração, tais como: item de configuração, controle de configuração, versão, baseline, codeline, mainline, release, branching, merging, entre outros.

Vale a pena

Vale a pena **ler**

IEEE COMPUTER SOCIETY. IEEE Standard for Configuration Management in Systems and Software Engineering. New Jersey, p. 71. 2012.

ENGHOLM JR., Hélio. Engenharia de software na prática. São Paulo: Novatec, 2010.

SOMMERVILLE, Ian. Engenharia de Software. 9. ed. São Paulo: Pearson, 2011.

Vale a pena **acessar**

ATLASSIAN. Git Branch. s.d. Disponível em: https://www.atlassian.com/br/git/tutorials/using-branches. Acesso em 23 out. 2019.

DANTAS, Cristine. Gerência de Configuração de Software. Devmedia, 2009. Disponível em: https://www.devmedia.com.br/gerencia-de-configuracao-de-software/9145. Acesso em: 23 out. 2019.

GIT. Básico de Branch e Merge. s.d. Disponível em: https://git-scm.com/book/pt-br/v1/Ramifica%C3%A7%C3%A3o-Branching-no-Git-B%C3%A1sico-de-Branch-e-Merge. Acesso em: 23 out. 2019.

Minhas anotações