Aula 4º

Construção de sistemas e gerenciamento de releases

Olá, alunos(as),

Bem-vindos(as) a nossa quarta aula da disciplina "Gerência de Configuração". Nesta aula, vamos continuar os nossos estudos a respeito das atividades da Gerência de Configuração. Vamos estudar agora como funcionam as atividades de Construção de Sistemas e de Gerenciamento de Releases, também igualmente importantes em um processo de desenvolvimento de software.

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

Bons estudos!

Objetivos de aprendizagem

Ao término desta aula, vocês serão capazes de:

- entender como funciona a Construção de Sistemas;
- saber a importância de se adotar um processo de Construção de Sistemas;
- conhecer como funciona a atividade de Gerenciamento de Releases.

Seções de estudo

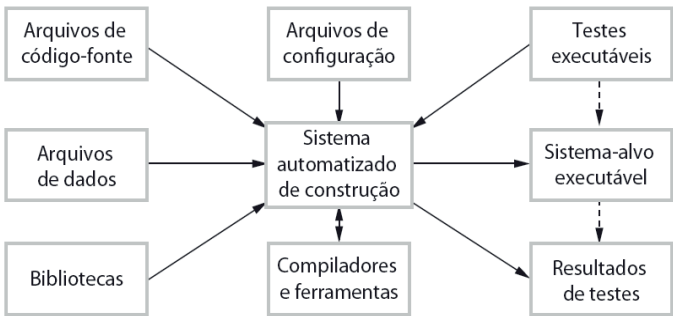
1 – Construção de sistemas
2 – Gerenciamento de releases

1 - Construção de sistemas

Começamos a nossa aula trazendo até a você a definição da atividade de Construção de Sistemas: "o processo da criação de um sistema completo, executável por meio da construção e ligação dos componentes de sistema, bibliotecas externas, arquivos de configuração etc." (SOMMERVILLE, 2011, p. 484). Essa atividade é apoiada por meio de aplicações que realizam a construção do sistema, que agilizam todas as tarefas relacionadas.

> A construção de sistemas envolve a montagem de uma grande quantidade de informações sobre o software e seu ambiente operacional. Portanto, para qualquer coisa além dos sistemas muito pequenos, sempre faz sentido usar uma ferramenta de construção automatizada para criar uma construção de sistema (...). Note que você não precisa apenas de arquivos de código-fonte envolvidos na construção. Você pode ter de ligá-los com bibliotecas externamente fornecidas, arquivos de dados (como um arquivo de mensagens de erro) e arquivos de configuração que definem a instalação-alvo. Talvez você precise especificar as versões do compilador e outras ferramentas de software que serão usadas na construção. A ideia é você ser capaz de construir um sistema completo com um único comando ou clique de mouse (SOMMERVILLE, 2011, p. 485).

Figura 1 – A construção de sistemas



Fonte: SOMMERVILLE, 2011, p. 485.

Essas ferramentas podem interagir com o sistema de controle de versões, para recuperar a versão mais recente do programa. Além disso, essas ferramentas podem usar uma descrição de configuração usada para identificar uma baseline.

Normalmente, quando encontramos um sistema para apoio a construção de sistemas, percebemos algumas características. Vejamos quais são a seguir:

1. **Geração de Script de Construção:** o programa analisa o projeto em que estão abrigados os arquivos do projeto, em busca dos componentes dependentes, gerando um script de construção automaticamente, para a instalação do programa e suas dependências;
2. **Integração com o Sistema de Gerenciamento de Versões:** Como já dissemos nessa aula, um sistema de construção pode procurar a versão mais recente dos arquivos-fonte no repositório central;
3. **Recompilação mínima:** O sistema de construção pode analisar o arquivo-fonte e decidir se deve ser compilado novamente, caso exista um arquivo compilado anteriormente. A verificação pode ser feita através da associação de *timestamps* de modificação – data e hora da última modificação do arquivo-fonte, caso tenha a última modificação seja depois da última compilação, o arquivo deve ser compilado anteriormente – ou a geração de *checksums* de código-fonte – onde a cada compilação é gerado uma assinatura (*checksum*) a partir do conteúdo do arquivo. Se a assinatura do arquivo atual não bater com a assinatura da última versão, uma nova compilação é necessária. A assinatura é gerada através de uma função matemática, como MD5 ou SHA-1;
4. **Criação de sistemas executáveis:** O sistema deve realizar a ligação dos arquivos de código compilados e as suas bibliotecas, para a criação de um programa executável;
5. **Automação de testes**: Alguns sistemas de construção podem executar arquivos de testes automatizados, executando ferramentas de testes como JUnit, PHPUnit, entre outras – que vimos na matéria de Verificação e Validação de Software. O intuito disso é verificar se o sistema não está quebrado, antecipando possíveis descobertas de bugs;
6. **Emissão de Relatórios**: O sistema pode fornecer relatórios a respeito do processo de construção e testes do sistema, podendo disparar alertas sobre problemas no processo via e-mail ou SMS;
7. **Geração de documentação:** O sistema de construção pode gerar documentação (notas de release e páginas de ajuda) do sistema construído. (SOMMERVILLE, 2011)

Para darmos um exemplo de sistema de apoio de Construção de Sistema, vamos falar um pouco do Apache Maven, usado em projetos Java:

Apache Maven

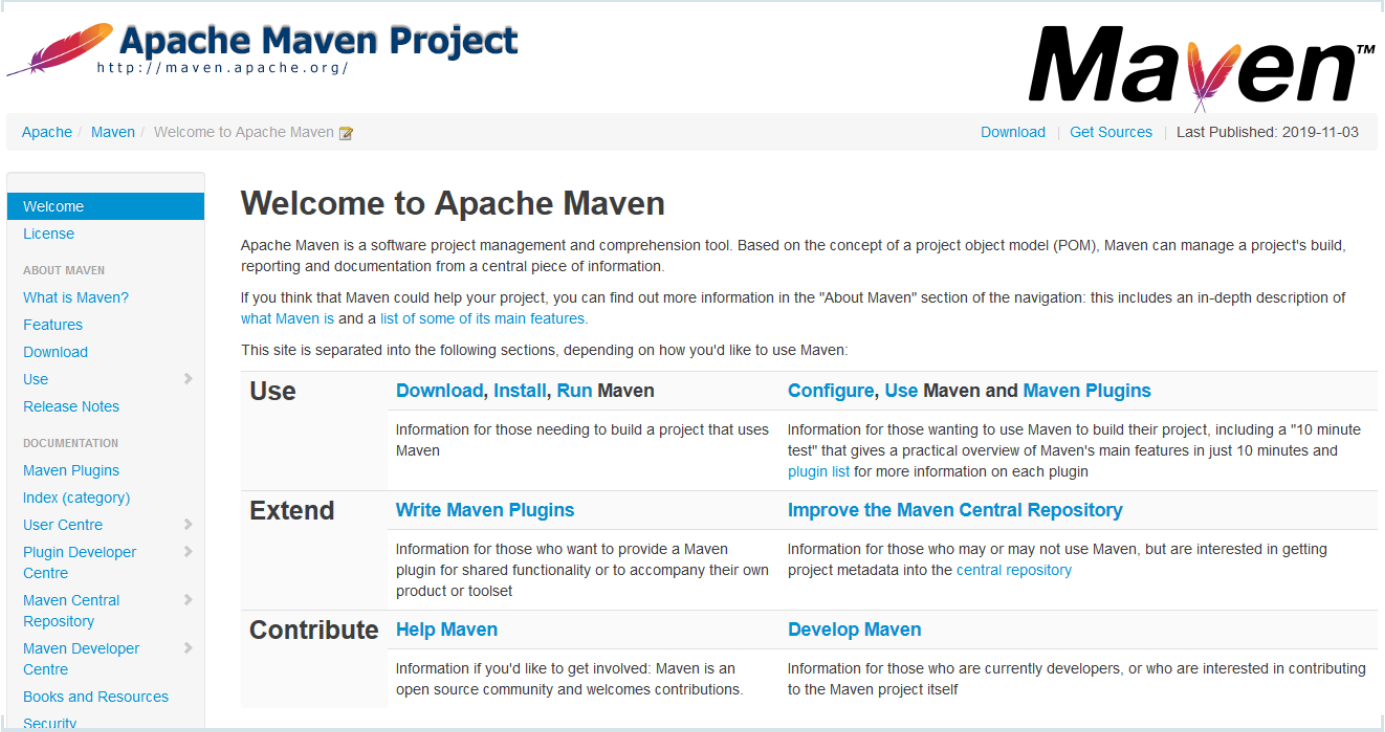
Em seu cerne, o Maven é uma ferramenta de gerenciamento e automação de construção (build) de projetos. Entretanto, por fornecer diversas funcionalidades adicionais através do uso de plugins e estimular o emprego de melhores práticas de organização, desenvolvimento e manutenção de projetos, é muito mais do que apenas uma ferramenta auxiliar.

Um desenvolvedor que seja alocado em um projeto Java EE que utilize o Maven corretamente não terá que saber de imediato quais dependências (bibliotecas) o projeto necessita para compilar e executar, não precisará descobrir onde obtê-las e nem irá se preocupar em como realizar a construção do pacote do aplicativo. Com um comando simples, como mvn install, na raiz do código-fonte do projeto, instruirá o Maven a gerar o código extra necessário (cliente de um web service, por exemplo), validar e compilar o projeto, testá-lo através de seus testes unitários e gerar o pacote com o código compilado. Outras etapas poderiam incluir auditoria de qualidade de código, documentação, geração de estatísticas, entre diversas possibilidades.

Outra característica do Maven é estimular a adoção de boas práticas, porque uma das formas utilizadas por ele para reduzir o esforço de configuração do projeto é a utilização do conceito de programação por convenção (do inglês convention over configuration), em que a ferramenta assume que o seu usuário fará as coisas da forma como ela preconiza como ideais (estrutura de diretórios padrão, por exemplo), e o livra de ter que declarar algo que se repetirá em todo projeto. o incorporar as práticas aceitas pela comunidade Java como as mais indicadas para projetos Java EE, o Maven acaba não só disseminando-as para novos desenvolvedores, como também as padroniza entre os projetos em que ele é empregado, permitindo que novatos se localizem muito mais rapidamente dentro de projetos novos. Obviamente, pode-se definir manualmente o que é assumido como padrão, ao preço do aumento na carga de trabalho para a configuração inicial do projeto.

Seu site oficial é o https://maven.apache.org/, que contêm informações a respeito da instalação e uso.

Figura 2 – Imagem do site oficial do Apache Maven



Fonte: Extraído a partir do site https://maven.apache.org/. Acesso em 03 nov. 2019

Fonte: OTTERO, 2012

Agora que você sabe quais são as características de um sistema de apoio à construção, vamos agora ver dois paradigmas de construção de sistema: A Integração Contínua e a Construção Diária.

1.1 – Paradigmas de construção do sistema

Vamos conhecer dois paradigmas de construção do sistema. Eles definem processos e momentos em que a construção do sistema é realizada.

Em resumo, podemos definir esses paradigmas como:

- **Integração Contínua:** Envolve a frequente reconstrução da *mainline*, após a realização de pequenas alterações no código-fonte (SOMMERVILLE, 2011, p. 487);
- **Construção Diária:** Onde o processo de construção do sistema é realizado em um período

predeterminado.

A integração contínua é usada normalmente em projetos em que métodos ágeis são utilizados, para agilizar o processo de construção do sistema, no qual possíveis erros podem ser detectados rapidamente. Já a Construção Diária é usada quando o sistema é muito grande, demandando muito tempo para ser reconstruído ou quando a plataforma de desenvolvimento é diferente da plataforma-alvo, exigindo mais tempo para testar o sistema. (SOMMERVILLE, 2011)

Feito essa diferenciação, vamos estudar agora a Integração Contínua. Vamos a uma definição mais abrangente:

> A integração contínua é uma prática de desenvolvimento de software de DevOps em que os desenvolvedores, com frequência, juntam suas alterações de código em um repositório central. Depois disso, criações e testes são executados. Geralmente, a integração contínua se refere ao estágio de criação ou integração do processo de lançamento de software, além de originar um componente de automação (ex.: uma CI ou serviço de criação) e um componente cultural (ex.: aprender a integrar com frequência). Os principais objetivos da integração contínua são encontrar e investigar bugs mais rapidamente, melhorar a qualidade do software e reduzir o tempo que leva para validar e lançar novas atualizações de software. (AWS, s.d.)

O que é DevOps?

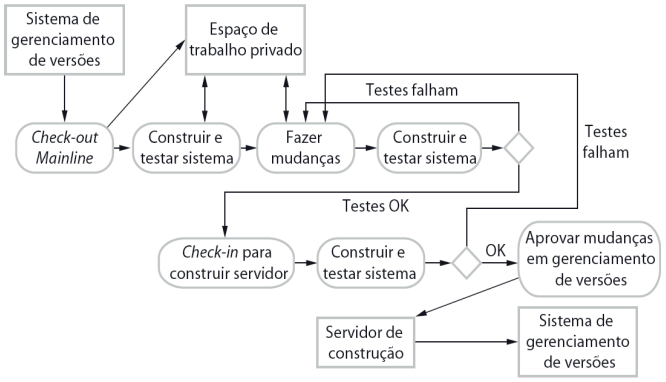
Devops é um termo criado para descrever um conjunto de práticas para integração entre as equipes de desenvolvimento de softwares, operações (infraestrutura ou sysadmin) e de apoio envolvidas (como controle de qualidade) e a adoção de processos automatizados para produção rápida e segura de aplicações e serviços. O conceito propõe novos pensamentos sobre o trabalho para a valorização da diversidade de atividades e profissionais envolvidos e atitudes colaborativas. É um processo que torna possível o desenvolvimento ágil de aplicações em um modelo de gestão de infraestrutura definido sob regras rígidas e burocráticas.
Fonte: 4LINUX, s.d.

As vantagens de se adotar a Integração Contínua são:

- melhora na produtividade dos desenvolvedores;
- detecção de bugs mais rápida;
- atualizações serão disponibilizadas mais rapidamente.

Agora que você sabe o que é a integração contínua e quais são as vantagens, vamos ver como funcionam os passos para um processo de integração contínua. Primeiramente, observe a figura a seguir, que descreve o processo de maneira visual, através de um fluxograma. Elencaremos as etapas a seguir.

Figura 3 – Integração Contínua



Fonte: SOMMERVILLE, 2011, p. 487.

As etapas da integração contínua são, segundo SOMMERVILLE (2011, p. 487):

1. O sistema realiza o *check-out* (*download*) do sistema, em um espaço de trabalho privado do desenvolvedor, com o intuito de pegar as atualizações mais recentes;
2. O sistema faz a construção do sistema e a execução dos testes automatizados na máquina do próprio desenvolvedor. Se o processo terminar em problemas, a última pessoa que fez as alterações (check-in) deve ser notificada para corrigir os problemas. Se der tudo certo, o processo avança para a próxima etapa;
3. O processo é novamente repetido (*check-in*, construção e testes) em um servidor de construção, caso outras pessoas tenham modificados os componentes após a sua modificação. Se o processo falhar, modificações devem ser feitas para consertar o problema;
4. Caso dê tudo certo, as mudanças são aprovadas e homologadas como uma nova *baseline* na *mainline* do sistema.

Agora que você viu o processo, a essa altura da aula, você deve estar se perguntando: Quais sistemas apoiam a Integração Contínua? Existem vários. Vamos descrever alguns deles a seguir:

Alguns sistemas de apoio à Integração Contínua

TravisCI: Funciona de maneira on-line, através do acesso aos repositórios do Github. O uso é gratuito para projetos open-source, e pago para projetos privados. Seu site oficial é https://travis-ci.org/ e informações a respeito da configuração inicial podem ser encontradas no site: https://docs.travis-ci.com/user/tutorial/.
Jenkins: É um sistema open-source, onde é instalado em um servidor. Permite a extensibilidade através de plugins e uma configuração fácil. Seu site oficial é o: https://jenkins.io/.

Bem, já vimos como funciona a Integração Contínua. Mas, como funciona a Construção Diária? Primeiramente, é feito um acordo com a equipe de desenvolvimento para acertar um prazo de entrega para as mudanças. Por exemplo, digamos que a equipe acertou o prazo de entrega para às 16 horas. Todas as mudanças que forem submetidas até às 16 horas serão aceitas para a construção.

Passado o horário acertado, uma nova versão do sistema é construída a partir da versão mais recente dos arquivos, no repositório, naquele horário. Esse sistema é entregue para a equipe de testes, que realiza um conjunto de testes do sistema predefinidos, enquanto a equipe de desenvolvimento conserta bugs e problemas verificados em construções anteriores.

Por fim, os defeitos que forem detectados são passados para a equipe de desenvolvimento, para que possam ser consertadas em novas versões do sistema.

Com isso, terminamos os nossos estudos a respeito da atividade de Gerenciamento da Construção do Sistema. A seguir, vamos estudar como funciona o Gerenciamento de Releases.

2 - Gerenciamento de releases

Vamos começar o nosso estudo a respeito do Gerenciamento de Releases, definindo o que é um release de um sistema: É uma versão de um sistema de software distribuída aos clientes (SOMMERVILLE, 2011, p. 488). Assim, qualquer versão que seja liberada aos clientes é considerada um release (ou um lançamento – dependendo do autor).

Em um sistema distribuído em massa, podemos perceber dois tipos de releases que são liberadas ao público:

- **Releases Maiores (Major):** Indica uma nova versão, fornecendo uma nova e significativa funcionalidade ao sistema. Por exemplo, uma versão pode ser enquadrada nessa parte quando são liberados novos recursos a ele.
- **Releases Menores (Minor):** Indica uma versão em que houve correção de bugs e problemas do sistema. Podem também ser denominadas de Patches. (SOMMERVILLE, 2011)

Vamos dar um exemplo. O Windows, no seu contexto atual, sendo chamado de Windows 10, libera várias versões por ano. Sendo que duas delas, que são liberadas uma vez a cada semestre, são versões que fornecem novas funcionalidades ao sistema. Essas atualizações exigem um tempo maior para a instalação e configuração. As demais atualizações são correções de problemas no sistema.

A seguir, exibimos um detalhe de uma tela com os detalhes do sistema. Observe dois números. O número "versão" se refere à versão maior do Windows que está instalado na máquina. Já o número "Compilação do SO" é o número da versão do Windows que está instalado, considerando os patches instalados nele.

Figura 4 – Versão do Windows

Fonte: Acervo Pessoal

Na maioria dos sistemas, a ordem de exibição do número de versão segue outra lógica. Normalmente, as numerações seguem um padrão X.Y, onde X é o número da versão maior e Y é o número da versão menor. Sommerville nos dá um exemplo:

> Por exemplo, este livro está sendo escrito em um computador Mac da Apple, no qual o sistema operacional é OS 10.5.8. Isso significa release menor 8 do release maior 5 do OS 10. Os releases principais são economicamente muito importantes para o fornecedor de software, pois os clientes precisam pagar por eles. Em geral, os releases menores são distribuídos gratuitamente. (SOMMERVILLE, 2011, p. 488)

Já nas bibliotecas de código-fonte aberto é usado um padrão de numeração denominado de **versionamento semântico**. A seguir, apresentamos esse padrão.

Versionamento Semântico 2.0.0

Introdução

No mundo de gerenciamento de software existe algo terrível conhecido como inferno das dependências ("dependency hell"). Quanto mais o sistema cresce, e mais pacotes são adicionados a ele, maior será a possibilidade de, um dia, você encontrar-se neste poço de desespero.

Em sistemas com muitas dependências, lançar novos pacotes de versões pode se tornar rapidamente um pesadelo. Se as especificações das dependências são muito amarradas você corre o risco de um bloqueio de versão (A falta de capacidade de atualizar um pacote sem ter de liberar novas versões de cada pacote dependente). Se as dependências são vagamente especificadas, você irá inevitavelmente ser mordido pela 'promiscuidade da versão' (assumindo compatibilidade com futuras versões mais do que é razoável). O inferno das dependências é onde você está quando um bloqueio de versão e/ou promiscuidade de versão te impede de seguir em frente com seu projeto de maneira fácil e segura.

Como uma solução para este problema proponho um conjunto simples de regras e requisitos que ditam como os números das versões são atribuídos e incrementados.

Essas regras são baseadas em, mas não necessariamente limitadas às, bem difundidas práticas comumente em uso tanto em softwares fechados como open-source. Para que este sistema funcione, primeiro você precisa declarar uma API pública. Isto pode consistir de documentação ou ser determinada pelo próprio código. De qualquer maneira, é importante que esta API seja clara e precisa. Depois de identificada a API pública, você comunica as mudanças com incrementos específicos para o seu número de versão. Considere o formato de versão X.Y.Z (Maior.Menor.Correção). Correção de falhas (bug fixes) que não afetam a API, incrementa a versão de Correção, adições/alterações compatíveis com as versões anteriores da API incrementa a versão Menor, e alterações incompatíveis com as versões anteriores da API incrementa a versão Maior.

Eu chamo esse sistema de "Versionamento Semântico". Sob este esquema, os números de versão e a forma como eles mudam transmitem o significado do código subjacente e o que foi modificado

de uma versão para a próxima.

Sumário
Dado um número de versão MAJOR.MINOR.PATCH, incremente a:
versão Maior (MAJOR): quando fizer mudanças incompatíveis na API,
versão Menor (MINOR): quando adicionar funcionalidades mantendo compatibilidade, e
versão de Correção (PATCH): quando corrigir falhas mantendo compatibilidade.
Rótulos adicionais para pré-lançamento (pré-release) e metadados de construção(build) estão disponíveis como extensão ao formato MAJOR.MINOR.PATCH.

Fonte: PRESTON-WERNER, 2014

O gerenciamento de versões pode ser simples quando temos o cenário de um produto único para um mercado de massa, mas quando nos referimos a sistemas customizados, quando existem diversas versões para diversos clientes, a tarefa de gerir as versões pode ser complicada:

> Para softwares customizados ou linhas de produtos de software, o gerenciamento de releases de sistema é um processo complexo. Os releases especiais do sistema podem precisar ser produzidos para cada cliente e os clientes individuais podem estar executando vários releases diferentes do sistema ao mesmo tempo. Isso significa que uma empresa de software vendendo um produto de software especializado pode precisar gerenciar dezenas ou até centenas de diferentes releases desse produto. (SOMMERVILLE, 2011, p. 488)

Assim, é necessário que haja um controle de quais releases existem, para facilitar esse trabalho. É por esse motivo que a gerência de versões é uma atividade da Gerência de Configuração.

> (...) Seus sistemas e processos de gerenciamento de configurações precisam ser projetados para fornecer informações sobre qual release do sistema cada cliente tem e o relacionamento entre os releases e as versões de sistema. No caso de algum problema, pode ser necessário reproduzir exatamente o software entregue para um determinado cliente. (SOMMERVILLE, 2011, p. 488)

Portanto, Sommerville recomenda que a cada release produzida seja feita uma documentação, para que seja fácil a recriação dessa release quando for necessário fazer isso:

> Assim, quando um release de sistema é produzido, deve-se documentar para se garantir que ele possa ser recriado no futuro. Isso é particularmente importante para sistemas embutidos, customizados e de longa vida, tais como aqueles que controlam máquinas complexas. Os clientes podem usar um único release desses sistemas por muitos anos e podem exigir mudanças específicas para um sistema de software específico muito após a data de release original. (SOMMERVILLE, 2011, p. 488-9)

Uma coisa que devemos atentar ao documentar um release, é que ele não inclui apenas os arquivos executáveis do sistema a ser documentado. Existem outros artefatos importantes que devem ter a atenção da equipe:

> Para documentar um release, você deve gravar as versões específicas dos componentes de código-fonte que Foram usadas para criar o código executável. Você deve manter cópias dos arquivos de código-fonte executáveis correspondentes e todos os dados e arquivos de configuração. Você também deve gravar as versões do sistema operacional, bibliotecas, compiladores e outras ferramentas usadas para construir o software. Estas podem ser necessárias para construir exatamente o mesmo sistema em uma data posterior. Isso pode significar que você precisa armazenar cópias de plataforma de software e as ferramentas usadas para criar o sistema no sistema de gerenciamento de versões junto com o código-fonte do sistema-alvo. (SOMMERVILLE, 2011, p. 489)

O que tem em um release?
Um release de sistema não é apenas o código executável do sistema. Ele também pode incluir:

- Arquivos de configuração definindo como o release deve ser configurado para instalações particulares;
- Arquivos de dados, tais como arquivos de mensagens de erro, que são necessários para uma operação bem-sucedida do sistema;
- Um programa de instalação, o qual é usado para ajudar a instalar o sistema no hardware-alvo;
- Documentação eletrônica e em papel, que descrevem o sistema;
- Empacotamento e publicidade associada, projetadas para esse release.

Quando uma versão é declarada como um release de um sistema, deve ser feita a identificação do código executável de programas e todos os arquivos de dados associados no sistema de gerenciamento de versões, sendo marcados com o identificador da versão. Além disso, devem ser elaboradas as instruções de instalação para cada sistema operacional e plataforma em que o sistema está sendo executado, além dos scripts, que conterão os comandos para realizar a instalação do sistema. Esses scripts e as instruções, normalmente são colocados junto com o software a ser distribuído (SOMMERVILLE, 2011).

Ao final de todo o processo, é elaborada uma imagem mestre, que servirá de base para a distribuição para os clientes do sistema. Antigamente, os *softwares* eram distribuídos em disquetes ou CDs, como podem ser vistos na figura a seguir.

Figura 5 – Antigamente, os sistemas eram distribuídos em disquetes ou em CDs.

Fonte: Disponível em: https://informatica.mercadolivre.com.br/programas-software/sistemas-operacionais/windows-95-original-em-disquetes. Acesso em: 19 nov. 2019.

Mas, com o avanço da internet, a grande maioria dos sistemas é disponibilizada para download, onde o usuário baixa o programa e executa na sua máquina.

E, com isso, finalizamos a nossa aula de hoje e a nossa incursão pelas quatro atividades da Gerência de Configuração. Na próxima aula, vamos ver sobre os Gerenciadores de Dependência. Até lá!

Retomando a aula

Chegamos ao final da nossa quarta aula. Vamos relembrar?

1 – Construção de sistemas

Você viu aqui que a atividade de Construção de Sistemas pode ser entendida como o processo da criação de um sistema completo, executável por meio da construção e ligação dos componentes de sistema, bibliotecas externas, arquivos de configuração etc. Viu, ainda, que essa atividade é apoiada por meio de aplicações que realizam a construção do sistema, que agilizam todas as tarefas relacionadas. Além disso, você viu algumas características das ferramentas de Construção de Sistemas e dois paradigmas de Construção de Sistemas: A Integração Contínua – quando uma nova construção é feita a cada mudança no código-fonte – e a Construção Diária – quando uma construção é feita a partir de uma determinada hora combinada.

2 – Gerenciamento de releases

Nesta seção, você viu um pouco a respeito do Gerenciamento de Releases, que é o gerenciamento das versões que são levadas ao público final. Você viu também que o gerenciamento de versões pode ser simples quando temos o cenário de um produto único para um mercado de massa, mas quando se referimos a sistemas customizados, quando existem diversas versões para diversos clientes, a tarefa de gerir as versões pode ser complicada.

Vale a pena

Vale a pena ler

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

4LINUX. *O que é DevOps*. s.d. Disponível em: https://www.4linux.com.br/o-que-e-devops. Acesso em 03 nov. 2019.

APACHE. Apache Maven Project. Disponível em: https://maven.apache.org/. Acesso em 03 nov. 2019.

AWS. O que significa integração contínua? s.d. Disponível em: https://aws.amazon.com/pt/devops/continuous-integration/. Acesso em 03 nov. 2019.

JENKINS. Disponível em: https://jenkins.io/. Acesso em 03 nov. 2019.

OTTERO, Rodrigo. Introdução ao Maven. DevMedia, 2012. Disponível em: https://www.devmedia.com.br/introducao-ao-maven/25128. Acesso em 03 nov. 2019.

PRESTON-WERNER, Tom. Versionamento Semântico 2.0.0. 2014. Disponível em: https://semver.org/lang/pt-BR/. Acesso em 18 nov. 2019.

ROCHA, Fábio Gomes. Integração contínua: uma introdução ao assunto. DevMedia, 2013. Disponível em: https://www.devmedia.com.br/integracao-continua-uma-introducao-ao-assunto/28002. Acesso em 03 nov. 2019.

TRAVIS CI. Disponível em: https://travis-ci.org/. Acesso em 03 nov. 2019.

Minhas anotações