\documentclass{beamer}

\usepackage{graphicx}

\usepackage{booktabs}

\usepackage[utf8]{inputenc}

\usepackage[portuguese]{babel}

\usetheme{Madrid}

\institute[UTFPR]

{

Universidade Tecnológica Federal do Paraná

\medskip

}

\title{Tópicos em Computação}

\author{Alunos: Rafael da S. Cruz / Maycon Camargo}

\date{\today}

\begin{document}

\begin{frame}

\titlepage{Integração Contínua}

\end{frame}

\begin{frame}

\frametitle{Overview}

\tableofcontents

\begin{itemize}

\item Conceitos de Build

\item Versionamento

\item Trevis - CL

\item referências

\end{itemize}

\end{frame}

\begin{frame}

\frametitle{Conceito de Build}

A criação de um software geralmente e criado por um conjunto de procedimentos, tanto no processo gerencial e no ponto de vista técnico do projeto. Quando é iniciado um processo as boas praticas de desenvolvimento, surge um favorecimento para uma maior qualidade, consistência e organização de um software.

A integração continua tem origem do seu termo na metodologia ágil XP, onde é integrado o código alterado e/ou desenvolvido ao projeto principal,nisso cada integração que é feita com o projeto principal é verificada por um build automatizado, com o intuito de encontrar erros de integração com o minimo de tempo.

\end{frame}

\begin{figure}

\includegraphics[width=0.6\linewidth]{latex\_1.jpg}

\end{figure}

\begin{frame}

\frametitle{Características}

\framesubtitle{Independência}

As tarefas de integração são realizadas otimizadamente sem terem que concorrer com outros aplicativos que normalmente estão rodando num computador de desenvolvimento. Além disso, a utilização de um ambiente independente fomenta o descobrimento de vários problemas, como por exemplo, problemas de dependência (arquivos, DLLs, chaves de registro) que fazem o software funcionar no computador de desenvolvimento, mas provavelmente não funcionaria no computador do cliente;

\end{frame}

\begin{frame}

\frametitle{Características}

\framesubtitle{Frequência}

Quanto mais cedo uma modificação puder ser integrada ao projeto principal e testada, mais cedo os erros serão detectados e corrigidos. A frequência em que as tarefas de integração serão realizadas, sem dúvida, vão depender do software que estiver sendo desenvolvido. Em certos casos, onde os projetos são pequenos, o processo poderá ser iniciado assim que o desenvolvedor enviar as modificações para o repositório do controle de versões. Porém, quando o software é grande e demora horas só para compilar, as tarefas de integração deverão ser rodadas à noite ou duas vezes por dia;

\end{frame}

\begin{frame}

\frametitle{Características}

\framesubtitle{Sincronização}

A sincronização das modificações frequente e contínua, serve como um termômetro para identificar a qualidade dos esforços da equipe de desenvolvimento. Além disso, integrações bem sucedidas aumentam a moral de todos os membros da equipe;

\end{frame}

\begin{frame}

\frametitle{Características}

\framesubtitle{Automação}

Entre outros benefícios, a Integração Contínua fornece um meio de automatizar processos manuais e repetitivos, evitando que os processos sejam esquecidos ou que algum passo importante não seja executado. Assim, você poderá realmente ter confiança de que essas tarefas serão realizadas com precisão e consistência todas as vezes que for preciso;

\end{frame}

\begin{frame}

\frametitle{Características}

\framesubtitle{Simplicidade}

Uma vez que todo o processo seja automatizado, qualquer operação poderá ser realizada por meio de um clique do mouse. Ninguém mais precisará encontrar o “CheckList” para realizar a compilação das bibliotecas compradas no ano passado, lembrar como se faz para gerar o manual nos formatos desejados ou até mesmo lembrar como se faz para gerar a instalação do software.

\end{frame}

\begin{frame}

\frametitle{Versionamento}

No gerenciamento de software existe um processo que se refere ao inferno das dependências (“dependency hell”), isso pode ocorrer devido ao crescimento do sistema adicionando pacotes e quanto maior o sistema, maior a probabilidade de encontrar essas dependências.

E com isso sistemas com grandes dependências inserir novos pacotes de versões pode se tornar um processo complicado, se essas dependências forem muito amarradas pode ocorrer uma perca de versão.

E com essas regras de versionamento simples podemos solucionar este problema na otimização de um sistema

\end{frame}

\begin{frame}

\frametitle{Especificação de Versionamento semântico(SemVer)}

Software que usa o SemVer DEVE ter uma API pública, há duas maneiras de ser declarada no código ou na documentação.

O formato de versão DEVE ser no formato X.Y.Z, valores sempre positivos, NÃO DEVE ter o valor zero na maior versão, sendo que X é maior versão, Y a menor versão e Z a de correção. Todo o elemento DEVE ser aumentado nunca diminuído. Ex:1.9.0-> 1.10.0.

Todo pacote que é lançado (released), as informações desta versão NÃO DEVE ser alterado. Se for ter alguma alteração DEVE ser feito uma nova versão.

No inicio do processo de desenvolvimento a versão Maior DEVE ser zero (0.y.z). A qualquer momento algo pode mudar e por isso a API não deve ser estável.

Após ser gerado a versão 1.0.0 a API se define como pública. E o incremento é dependente da API.

\end{frame}

\begin{frame}

\frametitle{Especificação de Versionamento semântico(SemVer)}

Correção Z (x.y.Z | x>0) DEVE ser incrementado somente se manter a mesma compatibilidade e inserir a correção de algum bug.

Menor Y (x.Y.z | x>0) DEVE ser somente incrementada se a nova versão for compatível e não afetara a funcionabilidade.

Maior X (X.y.z | X>0) só DEVE ser incrementada se a nova versão for incompatível com a API pública.

Release (x.y.z-RELEASE) PODE ser identificado por um hífen (dash) após o ponto (dot). Essa versão pode ser instável e não satisfazer os requisitos de compatibilidade. Ex: 1.0.0-alpha.

Build (x.y.z-release+BUILD) PODE ser identificada pelo sinal (+), metadados de construção PODEM ser ignorados quando se determina a versão de precedência.Ex: 1.0.0-alpha+001.

A precedência das versões DEVE ser calculada separando os identificadores, sempre nas versões Maior, Menor e Correção são sempre comparadas numericamente. Ex: 1.0.0 < 2.0.0 < 2.1.0 < 2.1.1

\end{frame}

\begin{frame}

\frametitle{Por que se deve usar o versionamento}

O uso Versionamento Semântico pode fazer do inferno de dependência uma coisa do passado, as modificações e correções serão feitas conscientes sabendo onde e como faze-las.

\end{frame}

\begin{frame}

\frametitle{Trevis - CL}

\framesubtitle{Conceito}

O Travis CI é um serviço de integração contínua distribuído e disponível na nuvem, utilizado para criar e testar projetos de software hospedados no GitHub

Quando você executa uma compilação, o Travis CI clona seu repositório GitHub em um ambiente virtual totalmente novo e executa uma série de tarefas para criar e testar o código compilado, se caso os testes falharem, a compilação entra em um estado interrompido, e cabe ao programador resolver as pendencias.

\end{frame}

\begin{frame}

\frametitle{Trevis - CL}

\framesubtitle{Etapas da execução}

O trevis divide a execução do job em duas etapa, primeiramente é atualizado as dependências, seja do projeto, ou seja do ambiente, depois na segunda etapa é diretamente voltada na execução da build e em outras tarefas relacionadas, como por exemplo: tests, reports e deploy, podemos detectar eventos do ciclo de vida e adicionar ações de acordo com a necessidade do projeto.

\end{frame}

\begin{frame}

\frametitle{Trevis - CL}

\framesubtitle{Etapas da execução}

1-Primeiro passo a ser feito é logar no Travis CI com a sua conta do GitHub.

2-Após o login, devemos sincronizar os repositórios da sua conta. Feito isso, precisamos habilitar o Travis para o nosso repositório indicando qual o arquivo que queremos habilitar

3-Em seguida, precisamos criar um arquivo no diretório raiz do projeto chamado de .travis.yml.

.Esse arquivo .travis.yml consiste em obter as informações que irá fazer nossa integração com o travis, criado no diretório raiz do repositório, ele especifica a linguagem de programação usada, o ambiente de construção e teste desejado (incluindo dependências que devem ser instaladas antes que o software possa ser construído e testado) e muitos outros parâmetros, suportando as linguagens C, C++, C\#, Clojure, D, Erlang, F\#, Go, Apache Groovy, Haskell, Java, JavaScript, Julia, Perl, PHP, Python, R, Ruby, Rust, Scala e Visual Basic.

\end{frame}

\begin{frame}

\frametitle{Trevis - CL}

\framesubtitle{Etapas da execução}

Na primeira linha do arquivo iremos especificar a linguagem do projeto(language: ).

.Na segunda linha a imagem que será usada para fazer a build do projeto(oxc-image: ).

.Depois precisaremos instalar as dependências que utilizaremos no projeto(install : ).

.Após obter as dependências do projeto, precisamos configurar alguns parâmetros que são o Path do ‘xcworkspace’ ou ‘xcproject’, depois algum scheme com a opção shared ativada e a SDK que iremos usar.

.Após essa configurações iremos realizar a build do projeto, vamo adicionar na etapa ‘scrip’ a chamada do ‘xcodebuild’.

.Depois no mesmo arquivo .travis.yml iremos implementar as notificações, para que em toda a execução o trevis possa notificar a cada integrante do processo, das possíveis problemas encontrados no decorrer do processo. Existem varias maneiras de se manter informado dos acontecimentos ocorridos nas execuções, sejam elas, por email, por sms e entre outros. Essa informação de como vc queira ser informado é validada no comando(notifications: )

\end{frame}

\begin{frame}

\frametitle{Referencias}

https://medium.com/@mari\\_azevedo/utilizando-travis-ci-como-servidor-de-integraC3%A7%C3%A3o-cont%C3%ADnua-em-projetos-open-source-um-exemplo-com-a-e30082f3ef46

https://translate.googleusercontent.com/translate\\_c?depth=1\&hl=pt-PT\&prev=search\&rurl=translate.google.com\&sl=en\&sp=nmt4\&u=https://docs.travis-ci.com/user/for-beginners\&xid=17259,15700019,15700186,15700190,15700256,15700259,15700262,15700265\&usg=ALkJrhjav8tl8AapEtsJjl9uZ3BrOFOJdw

https://en.wikipedia.org/wiki/Travis\\_CI

http://equinocios.com/ios/2017/03/26/integracao-continua-com-travis-ci/

https://www.devmedia.com.br/integracao-continua-criacao-de-um-processo-de-build/11552

https://www.devmedia.com.br/integracao-continua-uma-introducao-ao-assunto/28002

\end{frame}

\end{document}