**Aula 1 - introdução ao versionamento de código, Git e GitHub**

Fundamentos do versionamento de código

**O QUE É VERSIONAMENTO DE CÓDIGO?** O versionamento de código refere-se ao processo de manter um registro de alterações feitas em um arquivo ou um conjunto de arquivos ao longo do tempo.

Snapshots e deltas: os sistemas de versionamento podem registrar alterações por meio de duas maneiras: “snapshots” ou “deltas”.

Snapshots: Captam uma imagem completa de todos os arquivos em um determinado momento.

Deltas: Apenas registram o que foi alterado desde a última versão.

**Aula 2 - O que é considerado como uma nova aplicação?** Uma nova aplicação é, geralmente, um programa ou sistema que é introduzido pela primeira vez, sem antecessores diretos. É um produto que foi desenvolvido para atender a uma necessidade específica ou preencher uma lacuna no mercado.

Exemplo: Essa aplicação pode ser um novo aplicativo de rede social, um software de gerenciamento de projetos, uma ferramenta de edição de imagens ou qualquer outro tipo de programa que foi criado do zero.

Como determinar a versão inicial e o número que as melhorias ou correções devem receber? A numeração de versões é uma prática padronizada na indústria de software, que ajuda desenvolvedores e usuários a entender o estágio e a evolução de um produto. A numeração de versão mais comumente utilizada é a "Sem Ver" (Versionamento Semântico).

A estrutura geralmente segue o padrão "MAIOR.MENOR. CORREÇÃO “Exemplo: Android Nougat (7.0.0 em seu lançamento), depois complementado com versões de correção e ajustes.

Histórico de versões: Versão beta/alfa: Antes de um software ser oficialmente lançado, ele pode passar por fases de testes chamadas "alfa” e “beta”

A fase alfa é geralmente interna e limitada a desenvolvedores.

A fase beta é uma versão de pré-lançamento disponibilizada para um grupo mais amplo de usuários, a fim de testarem e reportarem bugs.

Versão inicial: Quando um software é lançado pela primeira vez, é comum denominá-lo como versão "1.0.0". O número "1" indica a primeira versão principal do software.

Melhorias e novas funcionalidades Quando novas funcionalidades são adicionadas, o número "MENOR" é incrementado. Por exemplo, se após o lançamento inicial uma nova funcionalidade significativa for adicionada, a versão poderia se tornar "1.1.0”.

Correções de bugs e segurança: Para correções menores, como correções de bugs ou atualizações de segurança, o número "CORREÇÃO" é incrementado. Por exemplo, "1.1.1".

**Aula 3 - Versionar: existem ferramentas que podem me auxiliar?** Existem boas práticas que podem ser aplicadas para manter o padrão dentro de diferentes organizações e ferramentas que facilitam muito o cotidiano de trabalho.

Baseadas no conceito que aprendemos sobre versionamento de código, estas são algumas das boas práticas para a manutenção do padrão:

Conhecer a importância da consistência, mantendo um padrão no versionamento

Revisar códigos: como o versionamento facilita a revisão de código e por que a revisão é importante? Introduzir o conceito de Integração Contínua e mostrar como ele se relaciona com o versionamento.

**A importância de manter o padrão no versionamento**

Versionar código é uma prática essencial para o desenvolvimento de software. Ela permite que os desenvolvedores rastreiem e revertam mudanças, colaborem de forma eficiente e gerenciem diferentes versões do software.

**Versionamento de código e revisões: uma parceria essencial**

A revisão de código é uma etapa vital no ciclo de vida do desenvolvimento de software. Envolve a análise sistemática do código-fonte para: Versionamento de código e revisões:

uma parceria essencial identificar e corrigir erros;

melhorar a qualidade do código;

garantir conformidade com padrões e práticas recomendadas.

**Ferramentas auxiliares: complementando o versionamento de código**

O versionamento de código é fundamental para um desenvolvimento de software eficaz, mas, por si só, não abrange todas as necessidades de uma equipe. Vários desafios surgem ao longo do ciclo de vida de um projeto de software, e é aí que as ferramentas auxiliares entram em cena.

1. Sistemas de Bug Tracking (Rastreamento de Erros): Esses sistemas, também conhecidos como sistemas de gestão de defeitos, permitem que as equipes registrem, rastreiem e gerenciem bugs e defeitos em seus softwares. Exemplos populares incluem: JIRA, Bugzilla e Redmine

2. Ferramentas de Integração Contínua (CI): Refere-se à prática de integrar frequentemente alterações de código em um projeto. Ferramentas como Jenkins, Travis CI e CircleCI auxiliam nesse processo, automatizando a compilação e os testes do código sempre que novas mudanças são integradas, garantindo que os problemas sejam identificados e corrigidos rapidamente.

3. Ferramentas de Entrega Contínua (CD): Continuando o processo de CI, a Entrega Contínua se concentra na automação das etapas de lançamento. Com ferramentas como Spinnaker e GitLab CI/CD, as equipes podem garantir que o software seja consistentemente entregue em ambientes de produção de maneira eficiente e confiável.

4. Dashboards e ferramentas de monitoramento: Plataformas como Grafana, Kibana e Datadog fornecem visualizações em tempo real do desempenho do software, ajudando as equipes a identificarem e solucionar problemas rapidamente.

**Semana 2: Fluxo de trabalho indivídual com GIT**

**Aula 1- Introduzir o conceito de Git e Configuração do Ambiente**

O controle de versão é essencial para a programação diária, pois o gerenciamento manual é propenso a erros. Por isso, o Git tornou-se uma referência para essa tarefa. Você conhecerá mais sobre ele nesta aula, em que vamos:

Diferenciar Git e outros sistemas de controle de versão;

Configurar o Git nos principais sistemas operacionais;

Conhecer o commit e funcionamento desse recurso.

**Recapitulando: O que é controle de versão de código?**

É um sistema que grava alterações em arquivos, possibilitando ao usuário acessar versões anteriores quando for necessário.

Ele é essencial para gerenciar versões do código-fonte no desenvolvimento de software.

Permite voltar arquivos ou o projeto todo a estados anteriores, comparar alterações e melhorar a colaboração entre desenvolvedores.

**Aula 2- Estruturar commits e Fluxos Básicos de Trabalho no Git**

O Fluxo de Trabalho Básico no Git envolve gerenciar mudanças no código, incluindo clonar o código, registrar “commits” e fazer “push” do código finalizado para o repositório remoto. É o que você vai ver nesta aula. Confira os objetivos:

Conhecer as características do Fluxo Básico de Trabalho no Git;

Estruturar commits;

Introduzir o conceito de branches;

**O que é um branch?** Um branch no Git é uma ramificação do projeto, que permite trabalhar em características ou correções de forma independente, sem impactar a linha principal ou outros branches.

**Introdução à integração contínua de software**

A integração contínua, frequentemente abreviada como CI (do inglês continuous integration), refere-se a uma prática de desenvolvimento de software na qual as alterações no código-fonte são automaticamente testadas e integradas a um projeto existente em uma frequência regular – em alguns casos, várias vezes ao dia.

Essa prática é crucial para identificar e corrigir problemas rapidamente, garantindo que o software permaneça em um estado funcional ao longo do desenvolvimento.

**O que é CD (entrega contínua)?**

frequentemente abreviada como CD (do inglês continuous delivery), é uma abordagem de desenvolvimento de software em que o código é construído, testado e preparado para ser lançado em produção de forma contínua. O que é CD (entrega contínua)?

A principal ideia por trás da entrega contínua é ter um sistema de lançamento de software automatizado e confiável, permitindo que as organizações liberem novas versões de software em resposta direta às necessidades de negócio.