**Por que é importante usar gerência de configuração?**

R: Garantir a qualidade e a rastreabilidade de todas as mudanças e versões envolvidas no desenvolvimento do projeto.

**O que é integração contínua em um projeto de software? Quais ferramentas podem ser usadas para realizar a integração contínua?**

R: Integração Contínua (CI) é uma prática no desenvolvimento de software em que os membros da equipe colaboram regularmente para incorporar suas alterações de código a um repositório compartilhado. Isso é automatizado por meio de ferramentas de construção e testes automatizados, como Jenkins, Travis CI, CircleCI, GitLab CI/CD, entre outras. A integração contínua visa identificar problemas de integração e qualidade rapidamente, permitindo correções imediatas. Os passos envolvem a submissão de código, compilação automática, execução de testes automatizados, análise estática e implantação em diferentes ambientes. Isso leva a uma entrega mais ágil e confiável de software, facilitando a detecção precoce de problemas e a colaboração entre os desenvolvedores.

**O que é débito técnico e quais ferramentas podem ser usadas para verificar débito técnico?**

R: Conceito - Também chamado de dívida técnica ou technical debt em inglês, o débito técnico diz respeito ao resultado de ações tomadas pelas equipes de desenvolvimento para agilizar a entrega de uma funcionalidade ou um projeto que precisa ser refatorado posteriormente. Em outras palavras, é o resultado de priorizar a entrega rápida sobre o código perfeito.

Apesar de entregas excepcionalmente rápidas trazerem uma boa impressão ao cliente, o débito técnico cria dificuldades na manutenção de códigos, gera custos adicionais, atrasos, retrabalho, projetos engessados ou necessidade de realizar mudanças no que já foi implantado.

**Os 4 tipos de débito técnico**

**Imprudente intencional:** o time de desenvolvimento sabe da existência de [erros](https://cloudster.com.br/blog/maiores-erros-de-nuvem-que-voce-nao-pode-cometer/?utm_source=blog-72049&utm_medium=referral) no código, porém lança a solução sem resolver os problemas existentes;

• **Imprudente não intencional:** programadores usam linguagens de programação que não conhecem bem, gerando necessidade de correções futuras;

• **Consciente intencional:** uma implementação é feita ou solução enviada com problemas, mas dentro do prazo. Nesse caso os desenvolvedores se comprometem a corrigir erros no futuro;

• **Consciente não intencional:** o time conclui um projeto, mas apenas percebe as falhas no código posteriormente.

Algumas ferramentas são: SonarQube, Checkstyle, ESLint para anáise de código estático e VisualVM para identificar gargalos de desempenho no código.

**O que é DevOps? Qual o relacionamento da integração contínua com o conceito de DevOps?**

R: DevOps é um composto de Dev (desenvolvimento) e Ops (operações) é a união de processos, pessoas e tecnologia para fornecer valor ao cliente. Tem como objetivo agregar mais valor aos negócios por meio de entregas de serviços rápidos e de alta qualidade por meio da disponibilização de serviços de TI iterativa e rápida, incluindo segurança, análise de dados, entre outras. Ao adotar uma cultura de DevOps o usuário ganha a capacidade de responder melhor as necessidades dos clientes, cumprir metas mais rapidamente, aumentar a confiança nos aplicativos, entre outros.

A integração continua esta ligada ao DevOps com uma união de equipes de desenvolvimento e alterações para acelerar uma entrega de softwares confiáveis. O DevOps busca eliminar barreiras entre essas equipes enquanto a integração continua é o que facilita a colaboração contínua e a integração suave entre o desenvolvimento e operações.

**Quando usar o controle de versão centralizado e quando usar o distribuído? Pontos positivos e negativos de cada um.**

R: O controle de versão centralizado e distribuído são duas abordagens diferentes para gerenciar as mudanças em projetos de software. Cada abordagem tem seus pontos positivos e negativos, e a escolha entre elas depende das necessidades específicas do projeto e da equipe. Vamos explorar os pontos-chave de cada uma:

**Controle de Versão Centralizado:**

Nesse modelo, há um único repositório central que contém o histórico completo do código-fonte. Os desenvolvedores clonam o código do repositório central e trabalham com versões locais. Mudanças são enviadas (commit) para o repositório central.

**Pontos Positivos:**

**Controle Central:** O repositório central atua como ponto único de verdade, o que pode facilitar a gestão de versões e controle de acesso.

**Simplicidade Inicial:** Para equipes menos experientes, um sistema centralizado pode ser mais fácil de entender e gerenciar.

**Regras de Acesso:** O repositório central permite aplicar regras de acesso e segurança de forma mais direta.

**Pontos Negativos:**

**Dependência do Servidor Central:** O servidor central é um ponto único de falha e gargalo potencial. Se ele estiver inacessível, a colaboração e o desenvolvimento podem ser interrompidos.

**Fluxo de Trabalho Limitado:** O desenvolvimento paralelo de recursos separados pode ser mais complicado, especialmente quando há conflitos frequentes.

**Colaboração Assíncrona Limitada:** A colaboração entre desenvolvedores que não estão conectados ao servidor central pode ser mais difícil.

Controle de Versão Distribuído:

Nesse modelo, cada desenvolvedor possui um repositório local completo com histórico de versões. As mudanças podem ser compartilhadas diretamente entre repositórios locais antes de serem enviadas para um repositório remoto.

**Pontos Positivos:**

**Descentralização:** Cada desenvolvedor tem uma cópia completa do histórico, tornando o desenvolvimento mais resiliente em caso de falhas no servidor.

**Trabalho Desconectado:** Os desenvolvedores podem trabalhar offline, tornando a colaboração assíncrona mais eficaz.

**Fluxo de Trabalho Flexível:** O desenvolvimento paralelo e o gerenciamento de ramos (branches) são mais flexíveis e naturais.

**Pontos Negativos:**

**Complexidade:** O modelo distribuído pode ser mais complexo para equipes menos experientes, especialmente em cenários com muitos ramos.

**Possível Conflito de Paradigmas:** Para equipes acostumadas com o modelo centralizado, a transição pode ser desafiadora.

**Escolha da Abordagem:**

A escolha entre o controle de versão centralizado e distribuído deve considerar fatores como a natureza da equipe, o tamanho do projeto, o nível de experiência da equipe, a necessidade de colaboração assíncrona e a tolerância a falhas.

Muitas equipes modernas optam pelo controle de versão distribuído, como Git, devido à flexibilidade, colaboração assíncrona e recursos avançados de gerenciamento de ramos. No entanto, equipes com requisitos mais simples ou equipes menos experientes podem preferir um modelo centralizado como o SVN. A decisão depende das prioridades do projeto e das preferências da equipe.