Prime

CCS-A

Desenvolvimento e Técnica de Codificação Segura

Desenvolvimento de Software

- É um processo complexo, com várias fases:
  - Design;
  - Codificação;
  - Teste;
  - Implantação;
  - QA.
- Segurança = Automação e scripts.

Ambientes

- Vários ambientes para isolar:
  -  Desenvolvimento;
  -  Teste;
  -  Preparação (*sandbox*);
  -  Produção;
- Para evitar acessos a mais de um ambiente:
  -  Segregação dos ambientes;
  -  Lista dos controles de acesso.

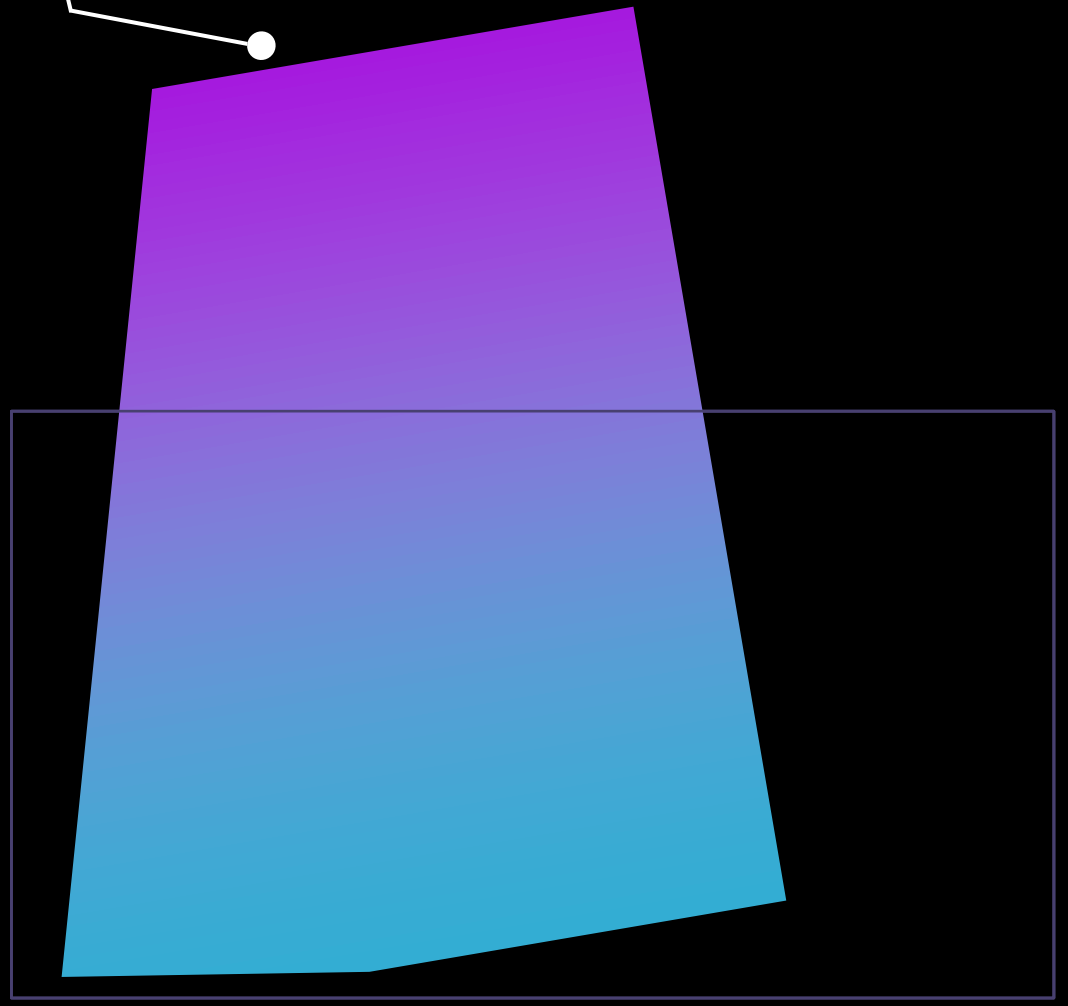
Ambiente de Desenvolvimento

Onde o sistema é criado e depurado para corrigir quaisquer erros.

Ambiente é dimensionado, configurado e ajustado para o desenvolvimento.

- O hardware não precisa ser escalável.

Depois que desenvolvido, o código vai para teste.



Ambiente de Teste

- Imita o ambiente de produção
  - Mesmas versões de software;
  - Mesmos níveis de patch;
  - Mesmos conjuntos de permissões;
  - Mesmas estruturas de arquivos.
- Objetivo: Testar antes de implementar.
- É preciso ser igual à produção.

Ambiente Staging

Serve como um *sandbox* após o teste. É o mais próximo possível do ambiente de produção.

Pense como um “ambiente falso” de produção.
Se nenhum problema ocorrer, a implantação continua.

Implementar por etapas pode evitar a perda total de produção por conta de falhas.

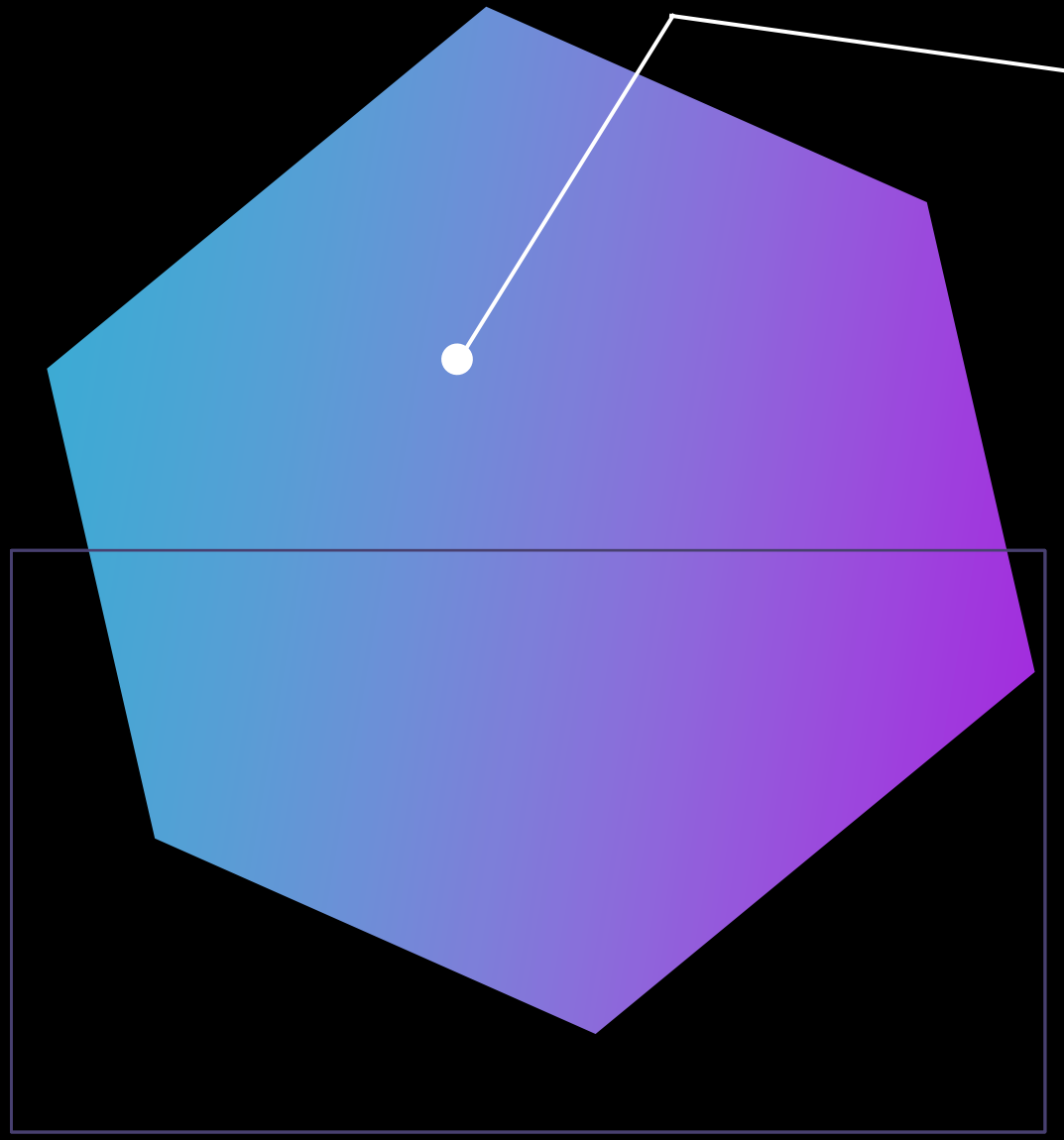
Ambiente de Produção

Software ativo, pronto para o uso e com dados reais.

É esperado poucas mudanças;

Se houver necessidade de mudança, será necessário a aprovação e teste novamente.

É importante considerar o gerenciamento de mudanças, liberação etc.

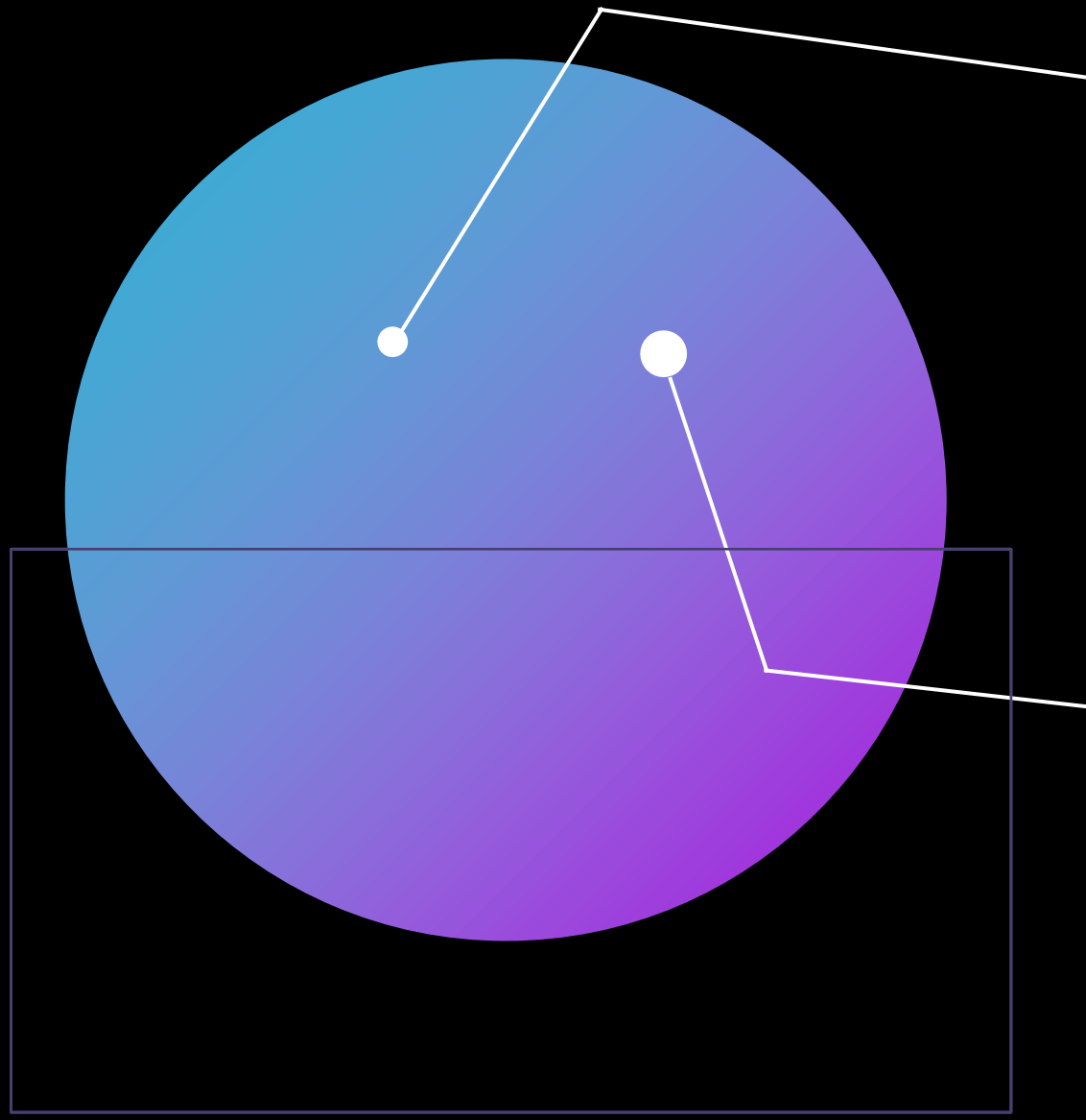


Quality Assurance (QA)

Garante a qualidade do produto, ou seja, se o atende às expectativas do cliente.

Testes de controle de qualidade e segurança e manutenção do registro de bugs.

Auxílio de obtenção de informações corretas em relação a construção de software seguro.



Provisionamento e Desprovisionamento

- **Provisionamento:** Atribuição de recursos a uma nova função ou tarefa.
  - Atribuição de permissões ou autoridades a objetos.
  - Ajuda a evitar a implantação de um novo elemento sem recursos suficientes.
- **Desprovisionamento:** Remoção de recursos, permissões ou autoridades.
  - Liberação de recursos para iniciar novas instâncias em um servidor.
  - Retorno de permissões para um nível de acesso inferior.
- A combinação entre os dois diminui riscos de manter um sistema com nível maior de autoridade.

Medição de Integridade

Determina se os dados foram alterados sem autorização.

Controle de versão não é suficiente. Requer um conjunto de ferramentas diferentes.

Exemplos de controles de integridade dos códigos:

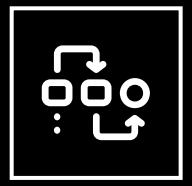
- Garantir o trabalho com cópias corretas;
- Comparar *hashes* dos sistemas e códigos.

Técnicas de Codificação Segura

- O segredo é garantir verificações e testes de qualidade de código.
- A segurança começa com um código seguro e mitigação das vulnerabilidades.
- Existem vários elementos no desenvolvimento que auxiliam em um código seguro:
  - Teste de segurança;
  - Fuzzing (inserir propositalmente informações inválidas);
  - Gerenciamento de patches.
- Dependem de boas rotinas de tratamento de exceções e validar entradas de dados.

Normalização

Técnica de programação e gerenciamento usada para reduzir a redundância e remoção de dados duplicados em BD.



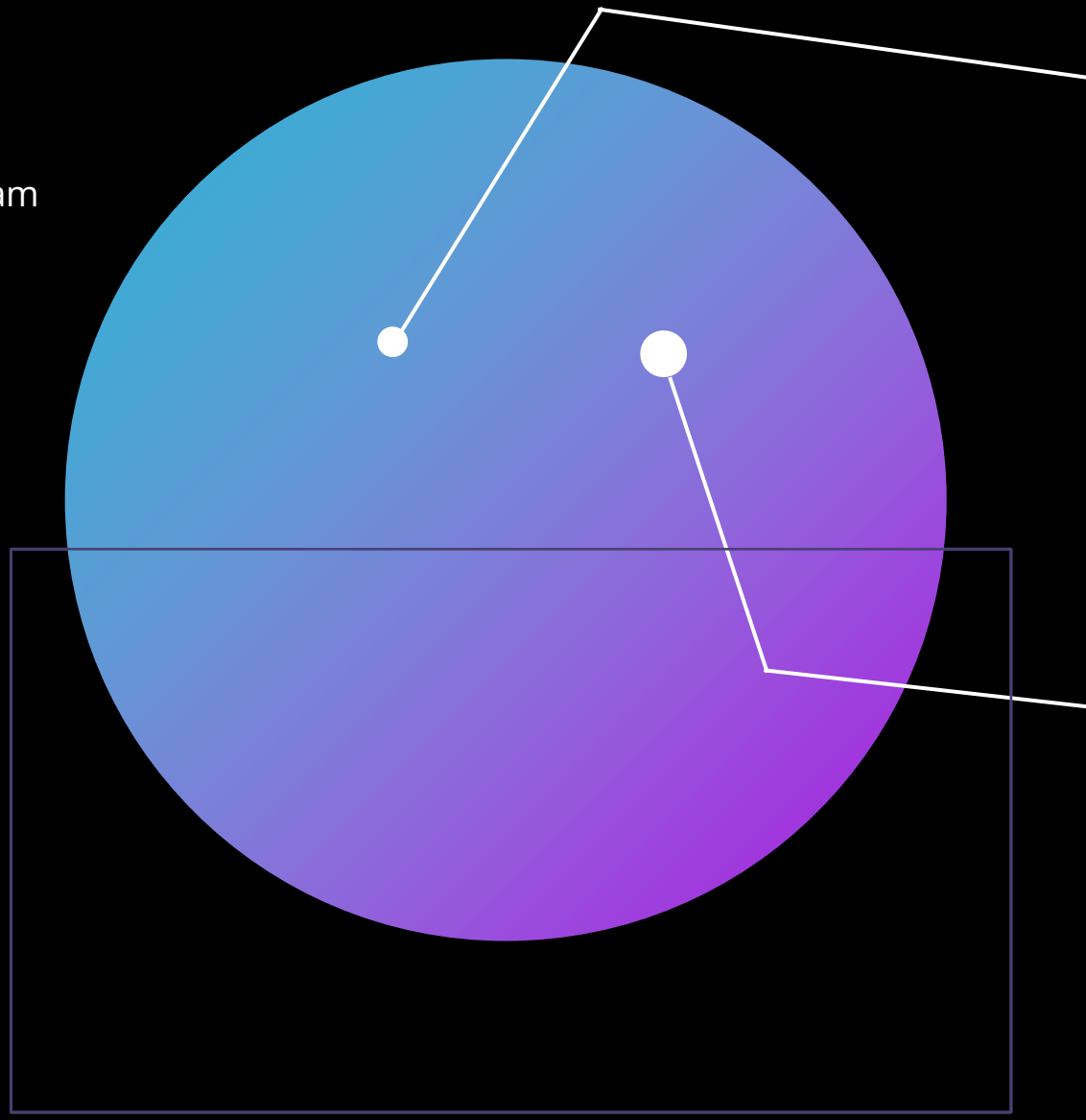
Exemplo, apenas uma tabela de clientes que armazena informações como o CustomerID.

O objetivo é evitar desperdício de espaço e aumento da carga de processamento.

A normalização define uma chave única primária que relaciona os campos. Evita perda de confiança e integridade dos dados.

Stored Procedures

- Métodos de interface com mecanismos do banco de dados.
- Sub-rotinas que podem ser usadas em aplicativos que acessam bancos de dados, gerando ganhos:
  - Velocidade;
  - Validação de entradas e de controle de acesso;
  - Manutenção centralizada da lógica do BD, pois os aplicativos chamam esses procedimentos;
  - Encapsula instruções SQL.
- Usados em:
  - SQL;
  - Java;
  - C++;
  - C.

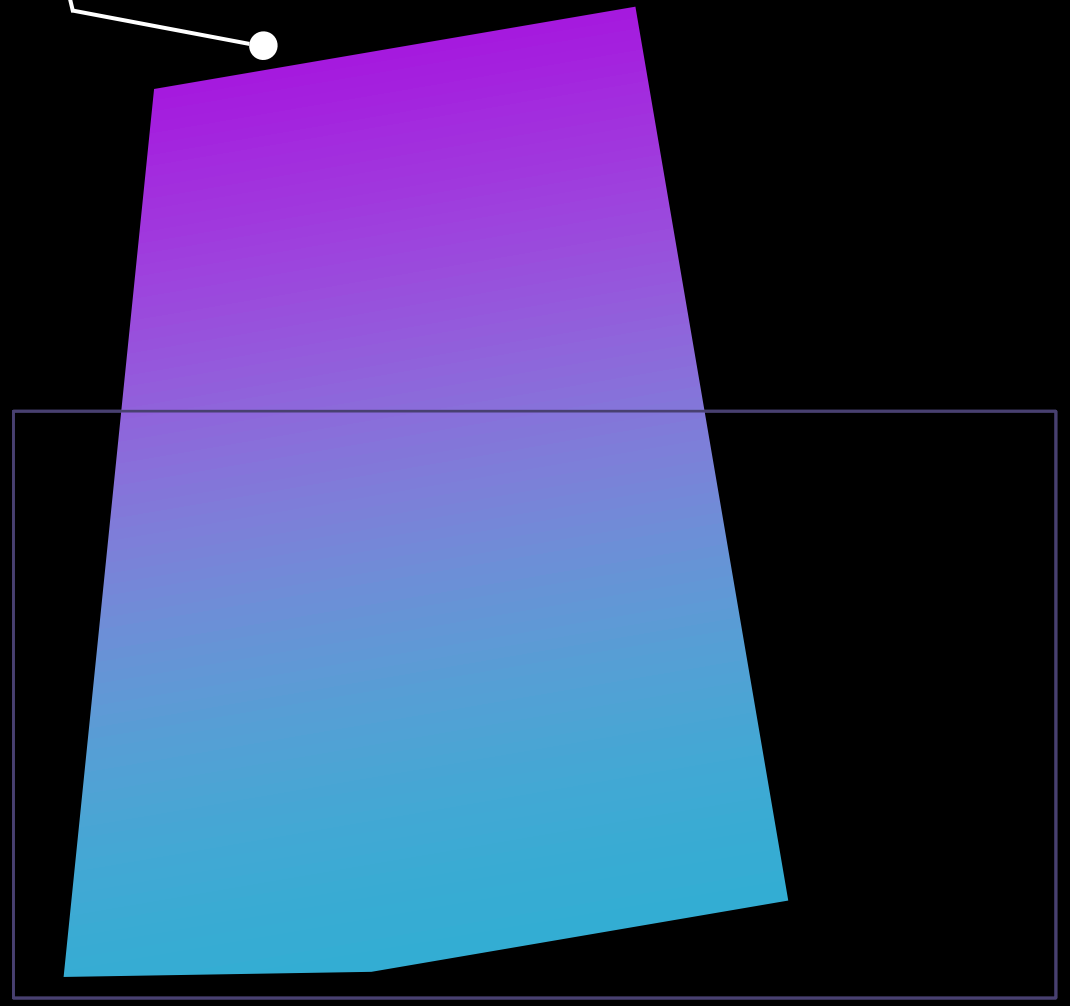


Ofuscação/Camuflagem

Princípios de segurança que aumentam a segurança, tornando um recurso ou componente obscuro ou pouco claro.

Se for difícil de entender ou encontrar, será mais difícil explorá-lo.

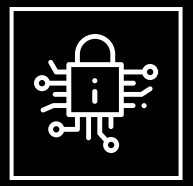
Dados e outros elementos ofuscados = bom!
Código ofuscado = perigoso!



Reutilização de Código e Código Morto

Reutilização de código (ou funções) proporciona:

- Confiabilidade de um código já testado;
- Redução de custos.

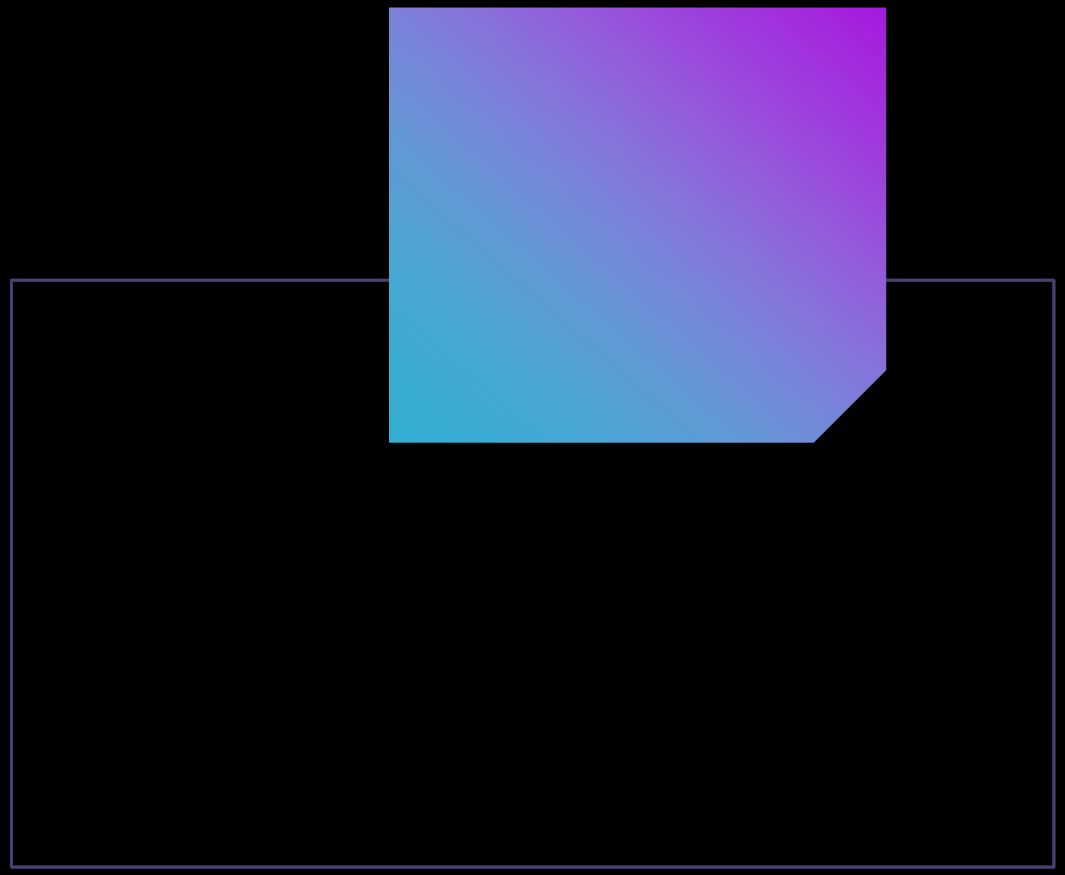


Para funções complexas como a criptografia, a reutilização é ideal.

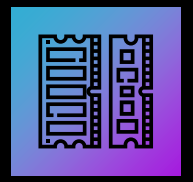
Código morto = Nunca é usado em outras partes do programa e pode conter vulnerabilidades.

Execução e Validação no Lado do Servidor Vs. No Lado do Cliente

- Verificações no cliente:
  - Elimina atrasos e poupa tempo;
  - Melhora a usabilidade das interfaces de software.
- Não é um local adequado para verificações críticas.
  - Cliente pode alterar qualquer coisa;
  - Dados podem ser alterados em trânsito.
- Servidor = Livre de influências externas.
- Implemente validações em ambos os lados.



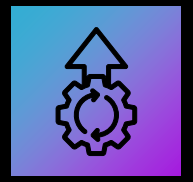
Gerenciamento de Memória



Fundamental que os desenvolvedores aprendam a gerenciar adequadamente a memória.

Limpeza de memória = coleta de lixo.

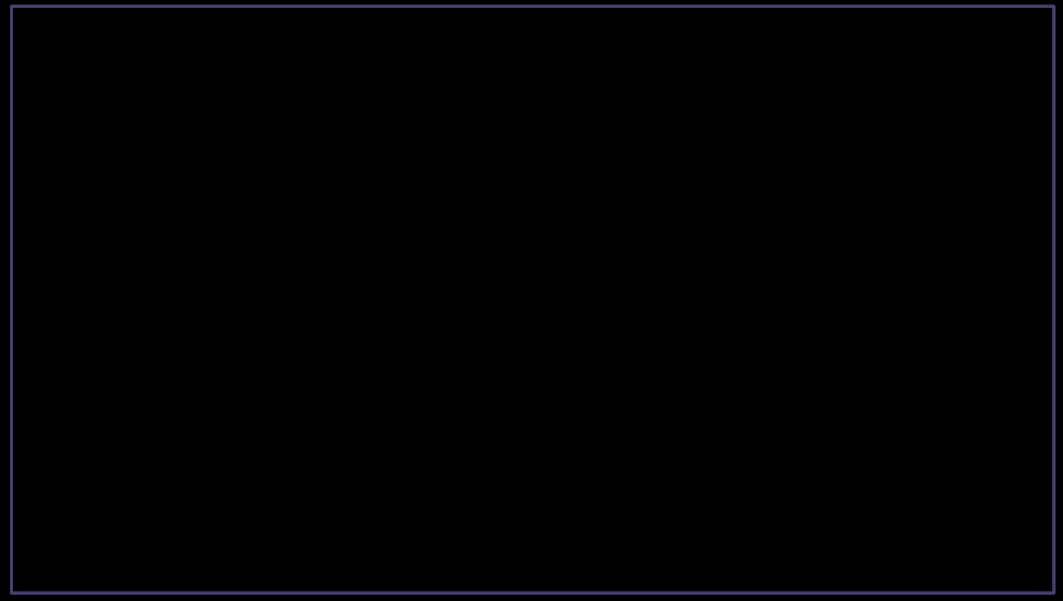
- C;
- C++.



Se não limpar a memória, poderá causar vazamentos de memória.

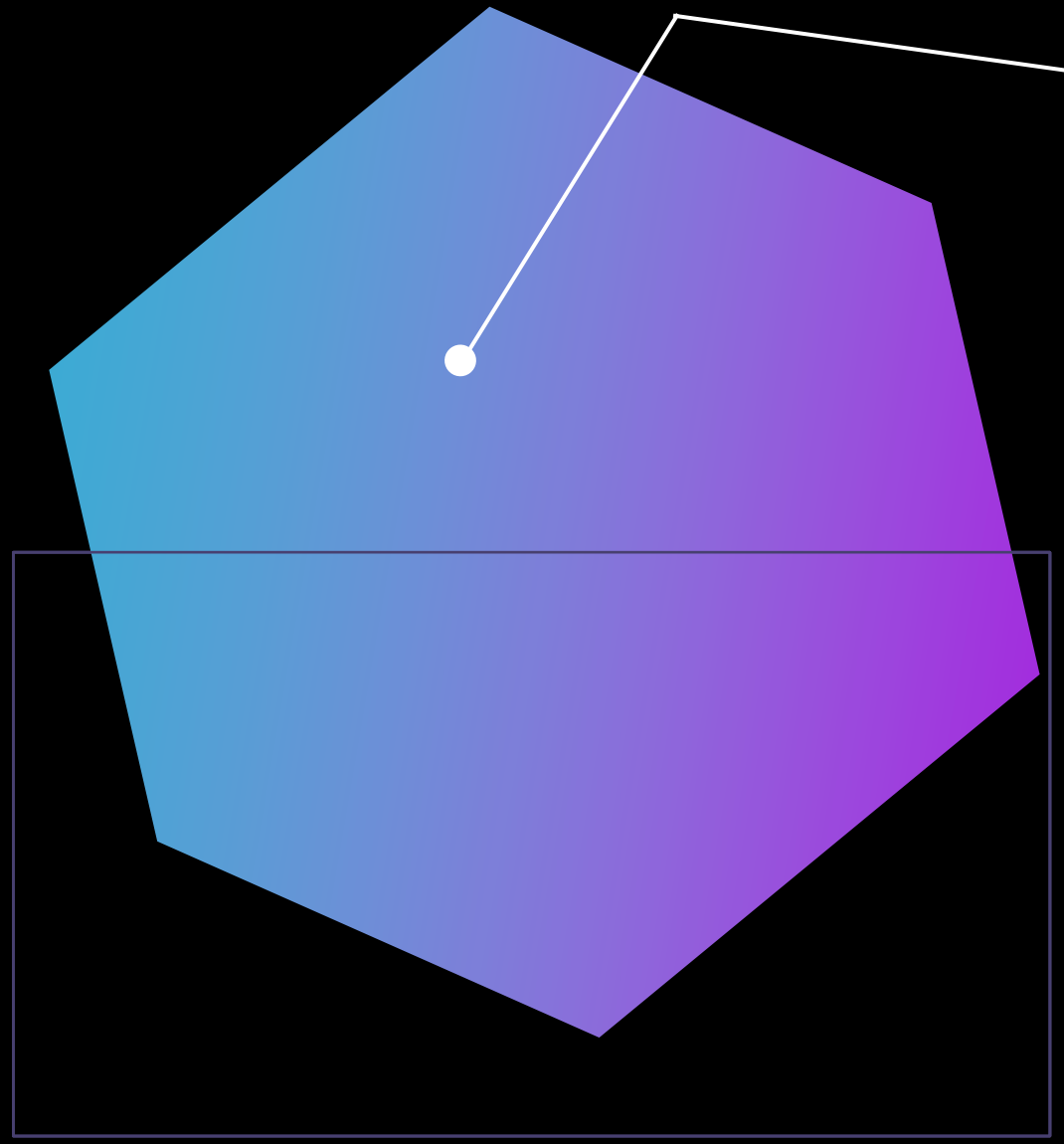
Uso de Bibliotecas de Terceiros e Kits de Desenvolvimento de Software (SDKs)

- Programadores usam bibliotecas de terceiros e SDKs. Por que?
  - Reescrever código dá trabalho;
  - Certas rotinas são complexas.
- Desenvolvedores de software criam aplicativos para plataformas específicas através dessas ferramentas.
- Evite SDK pois não há como saber o quão seguro é o código porque você não o escreveu.



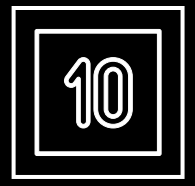
Exposição de Dados

- Perda de controle e exposição dos dados de um sistema durante as operações.
- Dados devem ser protegidos durante:
  - Armazenamento;
  - Comunicação;
  - Uso.
- Crie métodos públicos que interagem com os dados sem disponibilizá-los diretamente.



Projeto Aberto de Segurança em Aplicações Web (OWASP)

OWASP - Fundação sem fins lucrativos.



Contém uma lista de 10 vulnerabilidades de software.

www.owasp.org

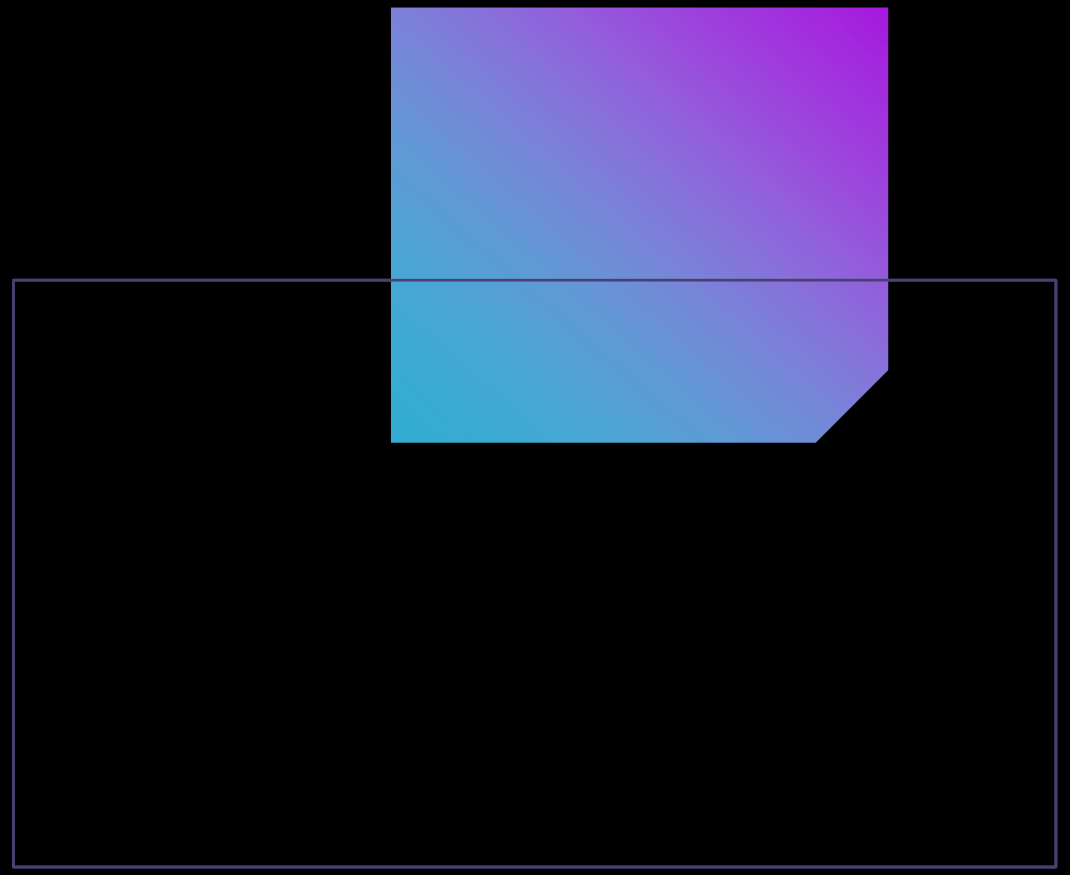
Útil para conhecer vulnerabilidades e ajudar os desenvolvedores.

Diversidade de Software

- Usar diversos compiladores para diferentes versões de códigos binários.
- Um software pode ser categorizado por:
  - Plataforma;
  - Linguagem de programação;
  - Interfaces;
  - Finalidade.
- Reduzir e gerenciar problemas de segurança quando descobertos.
- A diversidade tornar o código mais difícil de ser explorado, igual na diversidade de hardware.

Automação/Scripting

- Grande utilidade no desenvolvimento de software.
- Útil para a SI pois os scripts são testados e consistentes.
- DevOps:
  - Combinação de desenvolvimento e operações;
  - Comunicação e colaboração;
  - Modelo anti-cascata;
  - Alterações incrementais.
- Secure DevOps - Adição de etapas de segurança ao DevOps.



Cursos de Ação Automatizados

O DevOps depende da automação. O objetivo da automação é automatizar tarefas ou cursos de ação.

Automatizar a segurança também é necessário.

A automação:

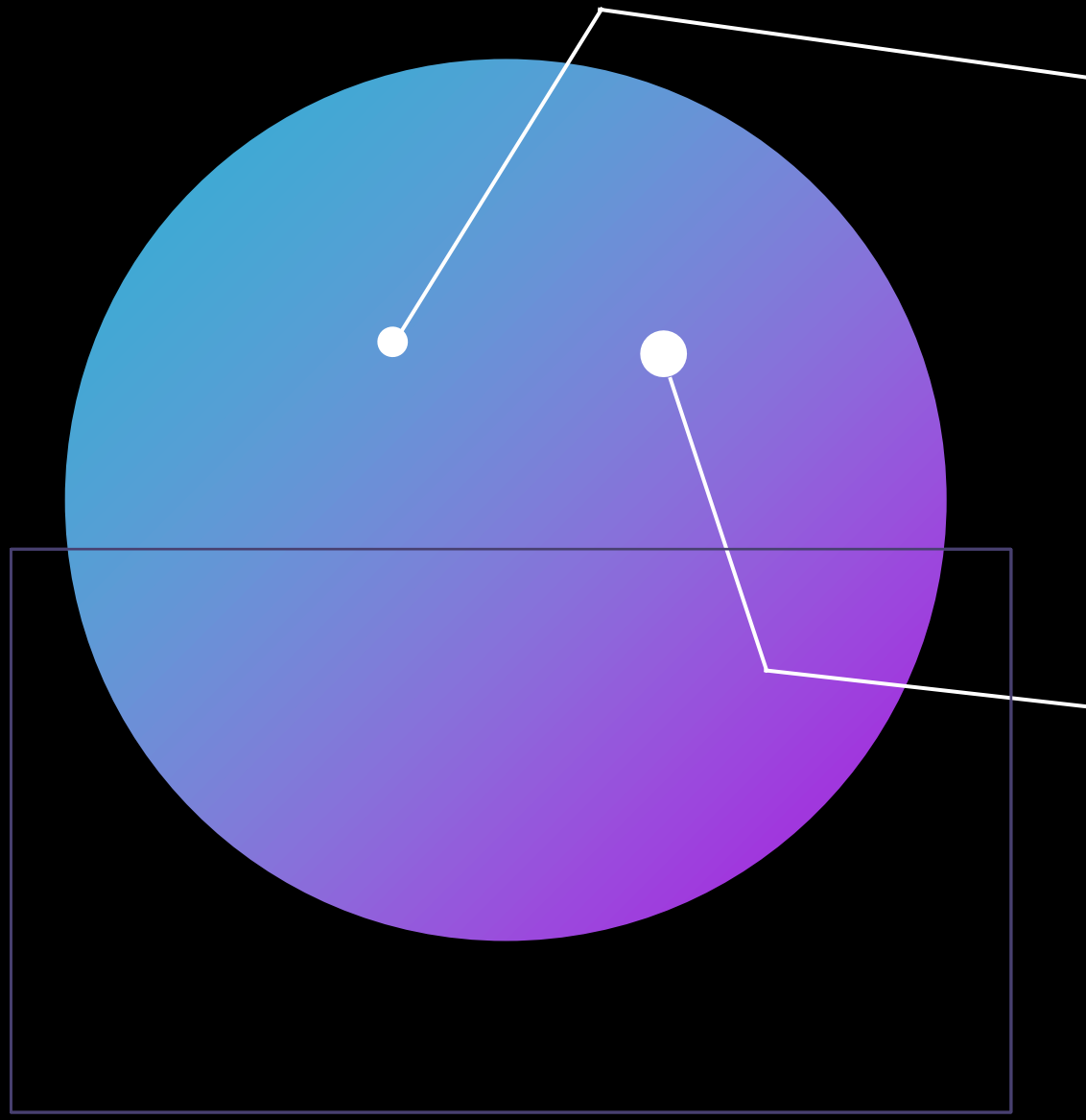
- Elimina custos;
- Concentra o trabalho em tarefas de valor.

Monitoramento Contínuo

Processos que detecta rapidamente problemas de conformidade e segurança.

Ferramenta para a gestão de riscos.

- Automação e scripts são usados para o monitoramento contínuo.



Validação Contínua

Uso de scripts que realizam verificações de código antes que o aplicativo seja **compilado** e eventualmente **implantado**.

Essencial para o desenvolvimento contínuo. É uma extensão dos testes.

Novos códigos devem ser testados para garantir:

- Funcionalidade;
- Estabilidade.

Integração Contínua

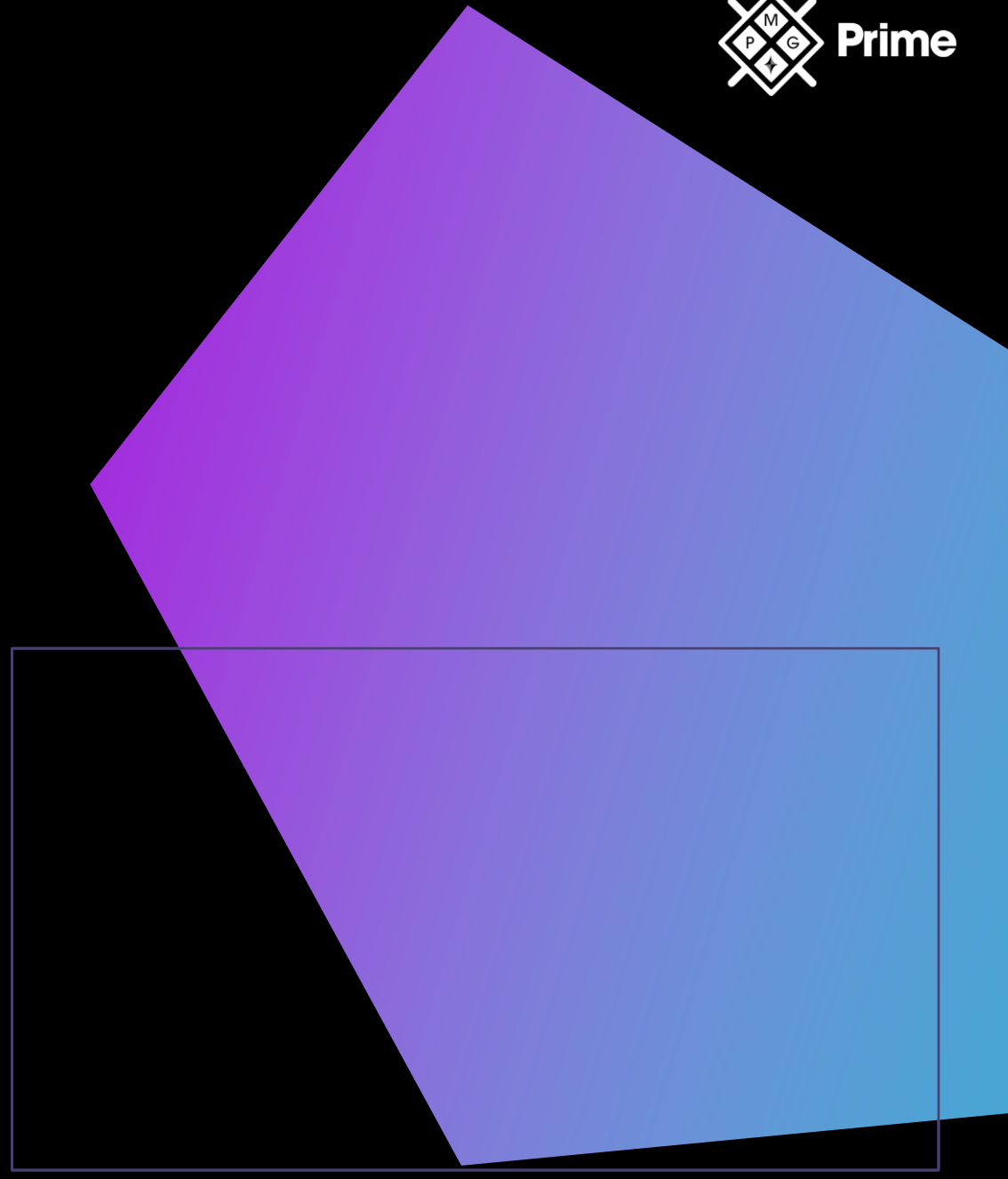
- Atualiza e melhora a base do código de produção.
- Permite:
  - Testar e atualizar pequenas alterações sem sobrecarga;
  - Executar integrações menores, com propósito único;
  - Isolar alterações em um número gerenciável;
  - Reduzir os erros de interação.

Entrega Contínua

- Extensão natural da integração contínua.
- Ciclos curtos que garante o lançamento de uma nova versão a qualquer momento.
- Permite:
  - Lançar rapidamente novas alterações;
  - Entregar atualizações quando elas são concluídas;
  - Automatizar a liberação.

Implantação Contínua

- Entrega contínua no piloto automático.
- Liberação automática:
  - Toda mudança que passa pelas etapas do pipeline é liberada;
  - Não há intervenção humana;
  - Envia automaticamente o código para produção.



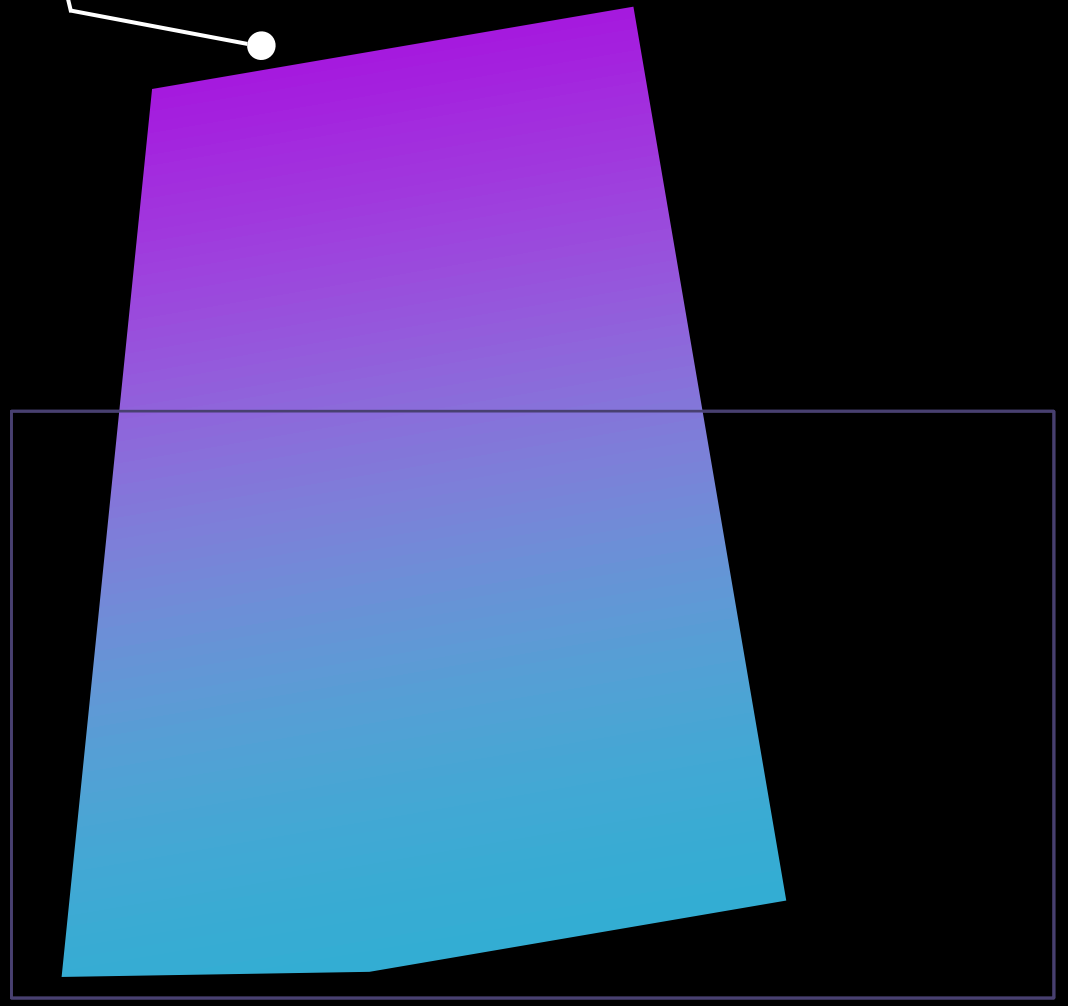
Elasticidade

Algo é capaz de mudar sem quebrar. Alocar mais (ou menos) RAM e/ou CPU quando necessários.

Computação em Nuvem = elasticidade.

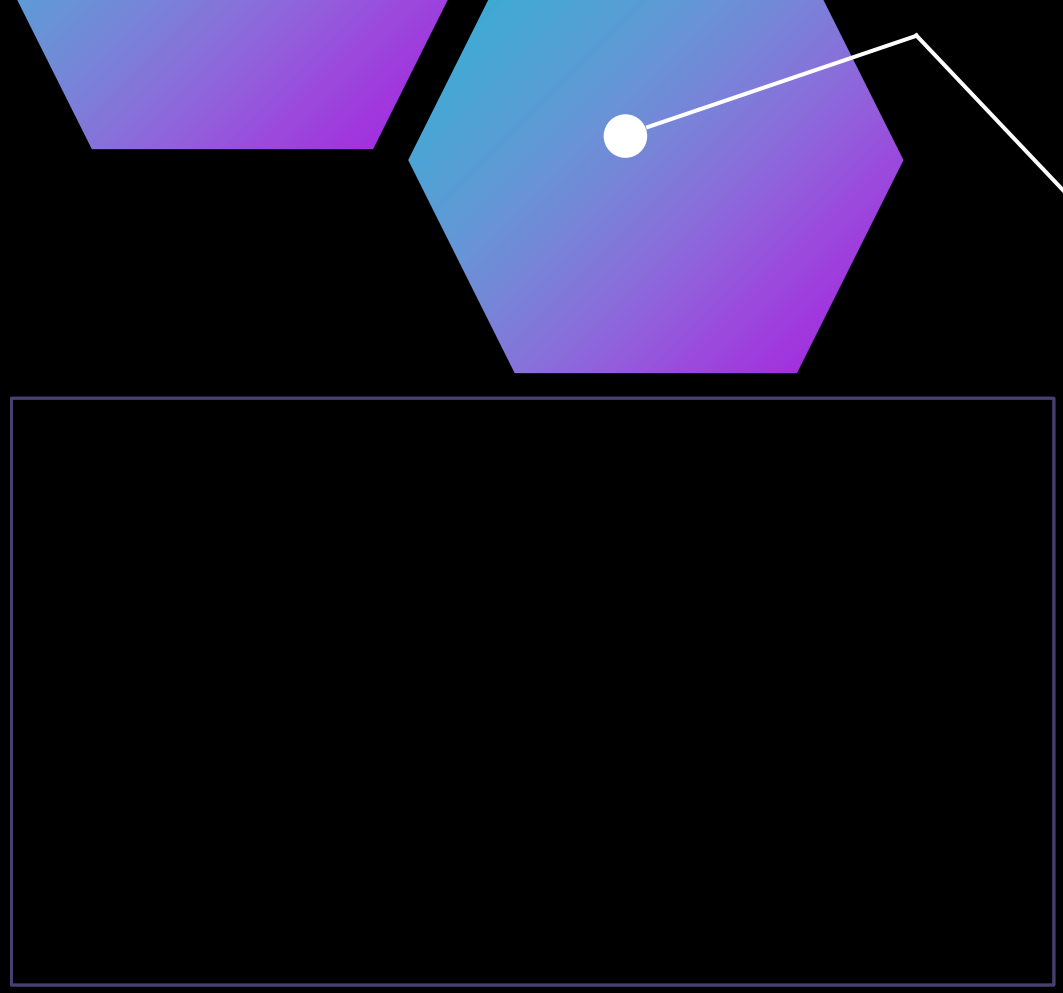
Para ser elástico:

- Deve ser capaz de ser executado sob várias condições.
- Software multithread pode se adaptar mais e melhor.



Escalabilidade

- Fornecimento de mais recursos para acompanhar um aumento de demanda.
- Importante em:
  - Sistemas web;
  - Banco de dados;
  - *Engines* de aplicativos;
  - Sistemas em nuvem.
- Escalar precisa de:
  - Sincronização (é tão rápido quanto o mais lento);
  - Coordenação.

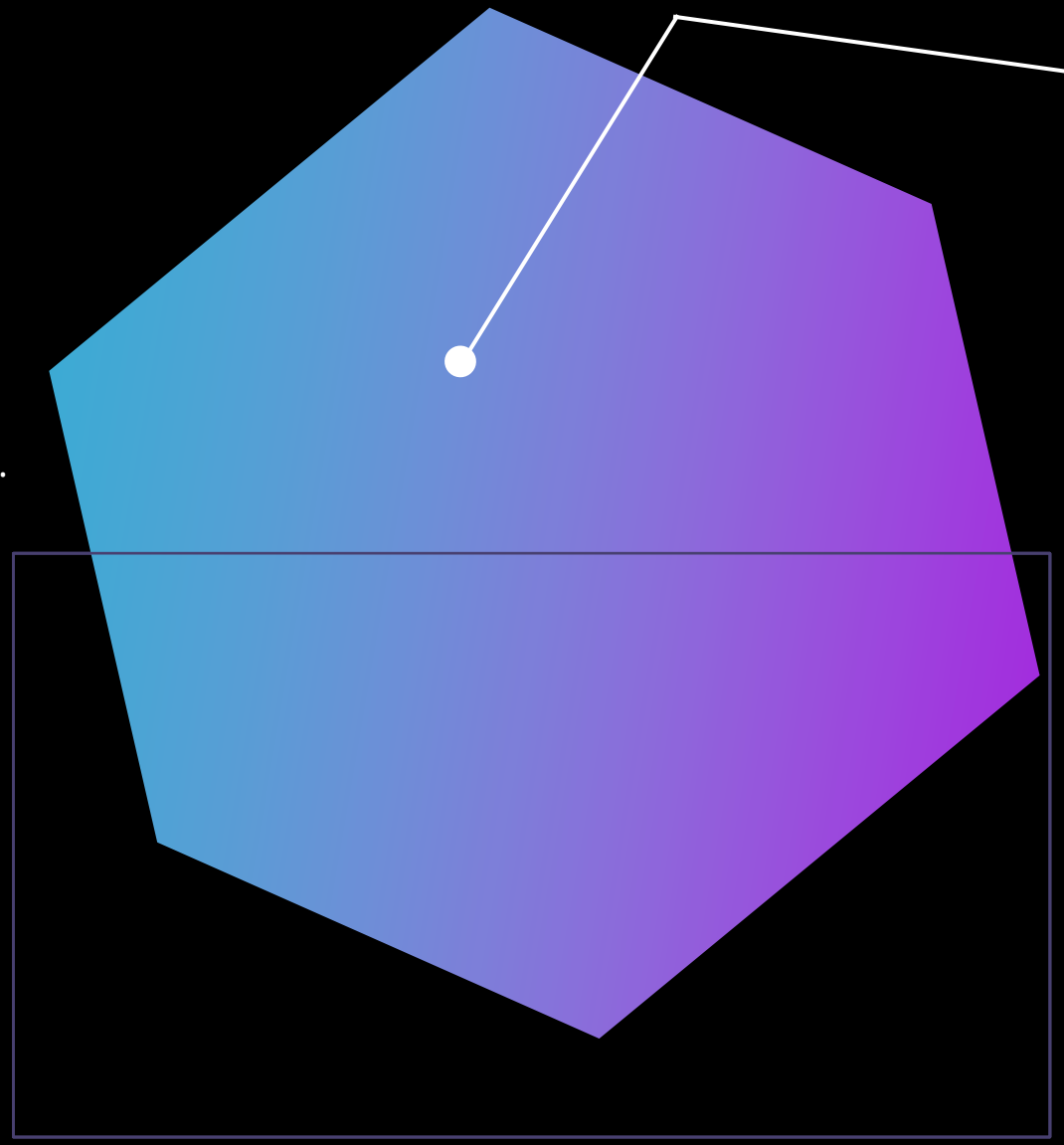


Controle de Versão

- Controle do que foi desenvolvido, lançado e testado quanto a segurança.
- Tão simples quanto rastrear a versão de um programa.
- Disponibilidade de várias versões = Gerenciamento de mudança.
- No DevOps:

É criado micro-lançamentos para os problemas não se associarem a alterações únicas;

É preciso gerenciar mudanças.



OBRIGADO!

DESENVOLVIMENTO E TÉCNICAS
DE CODIFICAÇÃO SEGURA