# Métodos de Desenvolvimento de Software

Carla Rocha

carlarocha.org

# Quem sou eu

Professora Doutora na área de Desenvolvimento de Software, com experiência em projetos de grande escala e métodos ágeis. Minha pesquisa foca na otimização de processos de desenvolvimento e na formação de equipes técnicas de alto desempenho.

Trabalho com comunidades de software livre e acredito que a colaboração é a chave para resolver problemas complexos na engenharia de software.

boss

# Experiência Prática e Acadêmica

Combinando teoria acadêmica com experiência real em projetos de software

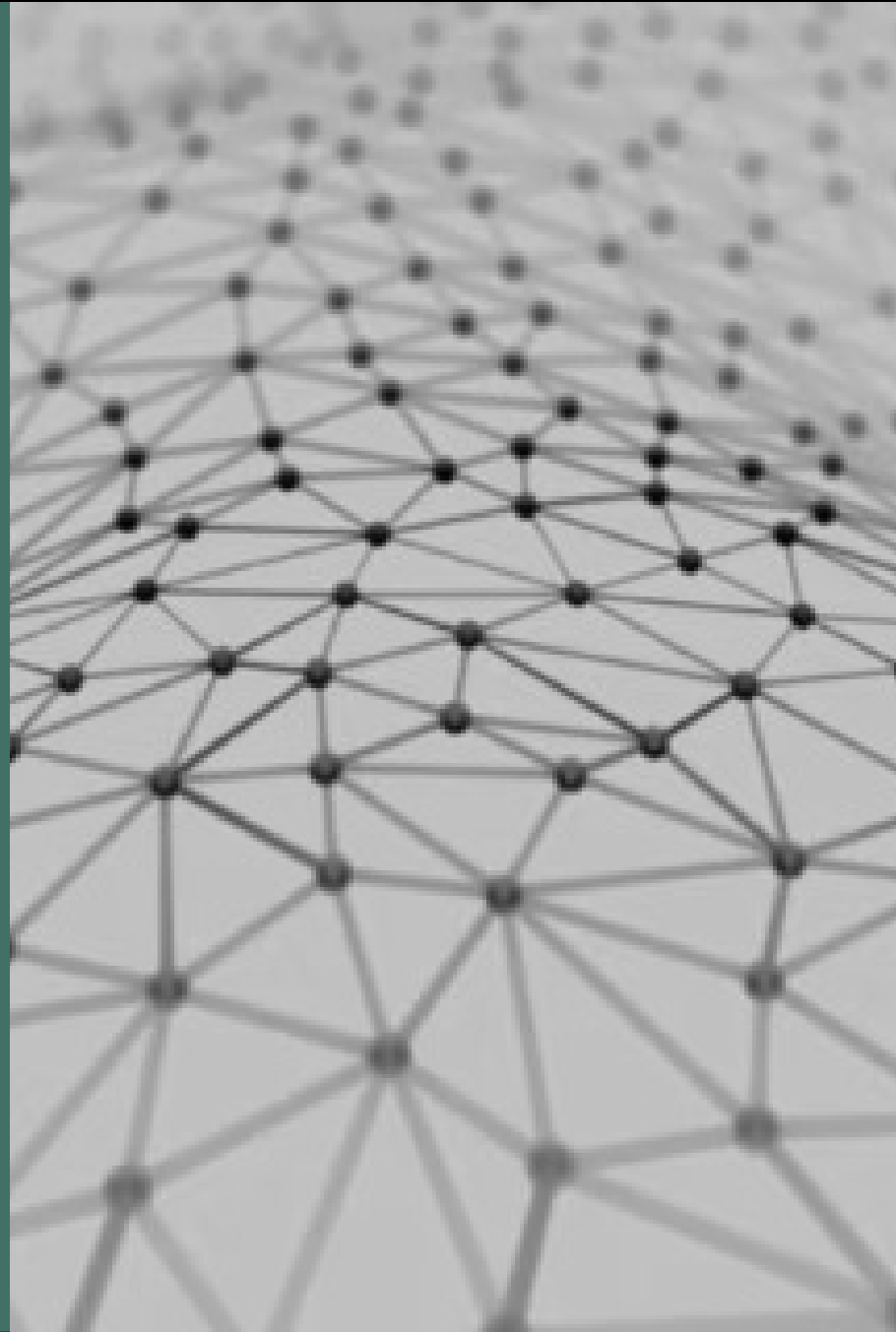
Especialização em metodologias ágeis e desenvolvimento colaborativo

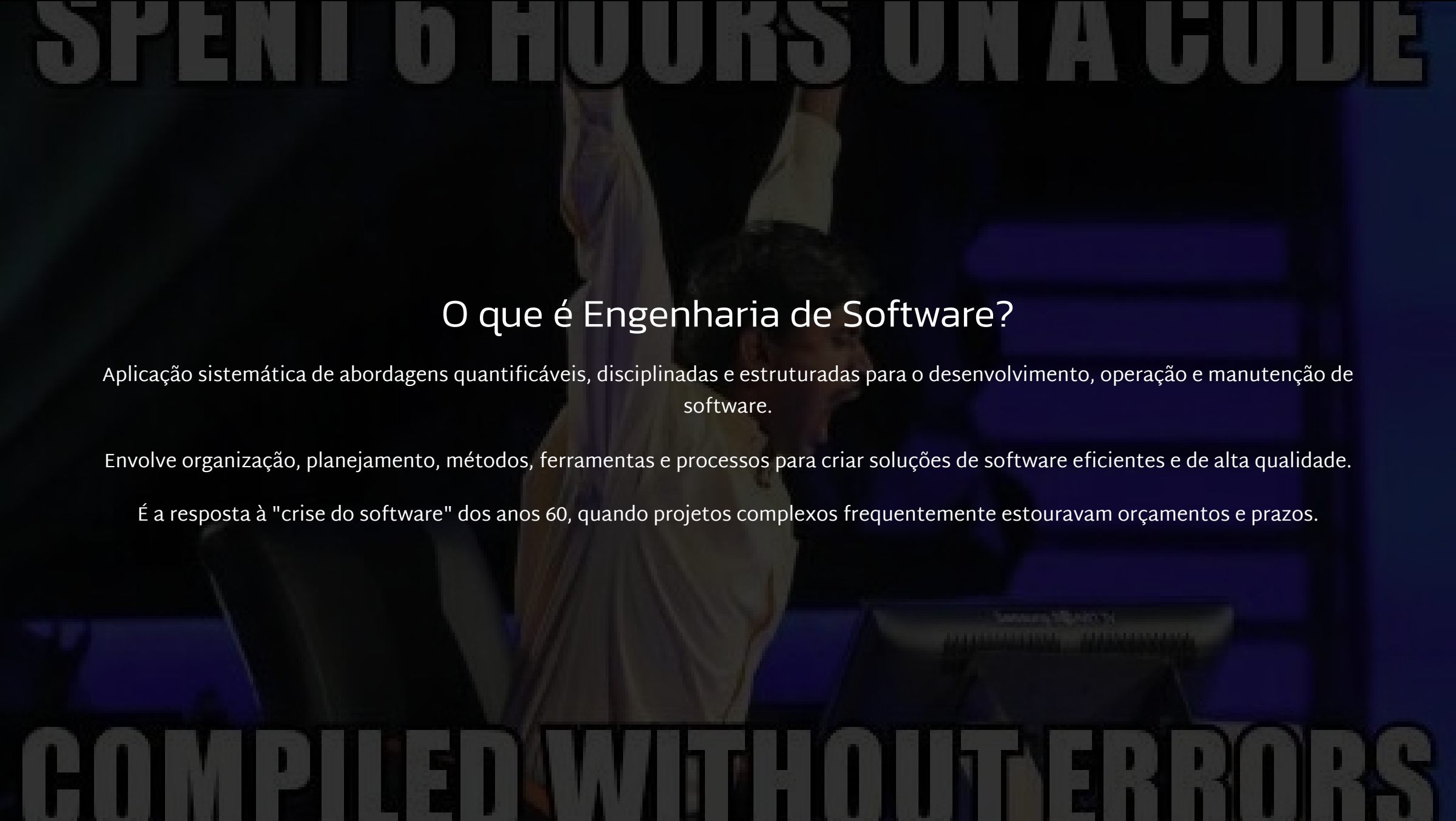
Contribuições para comunidades de software livre

Capítulo 1

# Engenharia de Software

Fundamentos e conceitos essenciais para o desenvolvimento de sistemas complexos






# O que é Engenharia de Software?

Aplicação sistemática de abordagens quantificáveis, disciplinadas e estruturadas para o desenvolvimento, operação e manutenção de software.

Envolve organização, planejamento, métodos, ferramentas e processos para criar soluções de software eficientes e de alta qualidade.

É a resposta à "crise do software" dos anos 60, quando projetos complexos frequentemente estouravam orçamentos e prazos.

# Ciclo de Vida do Software

O desenvolvimento de software segue um ciclo estruturado de fases interconectadas, independentemente da metodologia utilizada. Compreender este ciclo é fundamental para gerenciar projetos de forma eficaz.


Cada fase possui objetivos específicos e gera artefatos que servem de entrada para as fases subsequentes, criando um fluxo contínuo de desenvolvimento.

# Pilares da Engenharia de Software


A engenharia de software vai muito além da simples codificação. Ela integra conceitos de gerenciamento de projetos, análise de requisitos, arquitetura, testes e manutenção em um processo abrangente e estruturado.

O objetivo é desenvolver sistemas que não apenas funcionem, mas que sejam robustos, escaláveis, seguros e fáceis de manter ao longo do tempo.

# Lição #1

## Engenharia de Software

## *Não é somente* programação!

É um conjunto de disciplinas, práticas e processos que garantem a qualidade, manutenibilidade e escalabilidade do software.

# A Escala dos Desafios Modernos

## 2B

### Linhas de Código

Volume aproximado de código em sistemas modernos de grande escala

## 40K

### Commits Diários

Número de submissões de código por dia no Google

## 250K

### Arquivos/Semana

Arquivos modificados semanalmente em projetos de grande porte

Estes números demonstram a complexidade e escala dos projetos de software atuais, evidenciando a necessidade de processos robustos de engenharia que vão muito além da simples codificação.

# Complexidade no Desenvolvimento de Software

A complexidade no desenvolvimento de software moderno aumenta exponencialmente com o tamanho do sistema. Fatores como requisitos dinâmicos, tecnologias em evolução, e a necessidade de integração com sistemas existentes tornam o processo desafiador.

Métodos estruturados de engenharia de software são essenciais para gerenciar esta complexidade e entregar produtos de qualidade dentro de prazos e orçamentos razoáveis.

# A disciplina...

Esta disciplina busca fornecer uma base sólida nos princípios e práticas da engenharia de software, com ênfase em métodos de desenvolvimento que possam ser aplicados em projetos reais.

Você aprenderá sobre diferentes metodologias, suas vantagens e limitações, bem como técnicas para selecionar e adaptar o método mais adequado a cada contexto específico.

O foco será tanto na teoria quanto na prática, com projetos em grupo que simulam ambientes reais de desenvolvimento.

# Recursos para o Curso

Utilizaremos diversos recursos de aprendizagem, incluindo materiais teóricos, estudos de caso, projetos práticos e ferramentas colaborativas.

Para recursos visuais, utilizaremos bancos de imagens gratuitos como Pixabay:

https://pixabay.com/

# Perfil Típico de Projetos na Disciplina

Utilizaremos o Pexels como um dos repositórios de recursos visuais para nossos projetos:

https://www.pexels.com/

## Características comuns dos projetos:

- Conhecimento profundo do problema a ser resolvido
- Recursos financeiros limitados
- Ideias de software complexas e inovadoras

Estes projetos simulam cenários reais onde você precisará balancear escopo, recursos e qualidade, aplicando métodos de desenvolvimento apropriados para cada contexto.

Lição #1

# Inovação origina da colaboração voluntária entre pessoas resolvendo problema similar




As soluções mais inovadoras frequentemente surgem quando pessoas com diferentes perspectivas e habilidades colaboram para resolver problemas comuns.

Lição #2:

# Talvez você não precise de um software

Um dos erros mais comuns em projetos é assumir que todos os problemas necessitam de uma solução baseada em software.

Antes de mergulhar no desenvolvimento, pergunte-se:

- Este problema realmente exige uma solução tecnológica?
- Existem processos que poderiam ser otimizados sem software?
- O custo e complexidade do desenvolvimento justificam os benefícios?

Às vezes, a melhor solução é mais simples do que imaginamos.

Lição #3:

# Use soluções/plataformas prontas

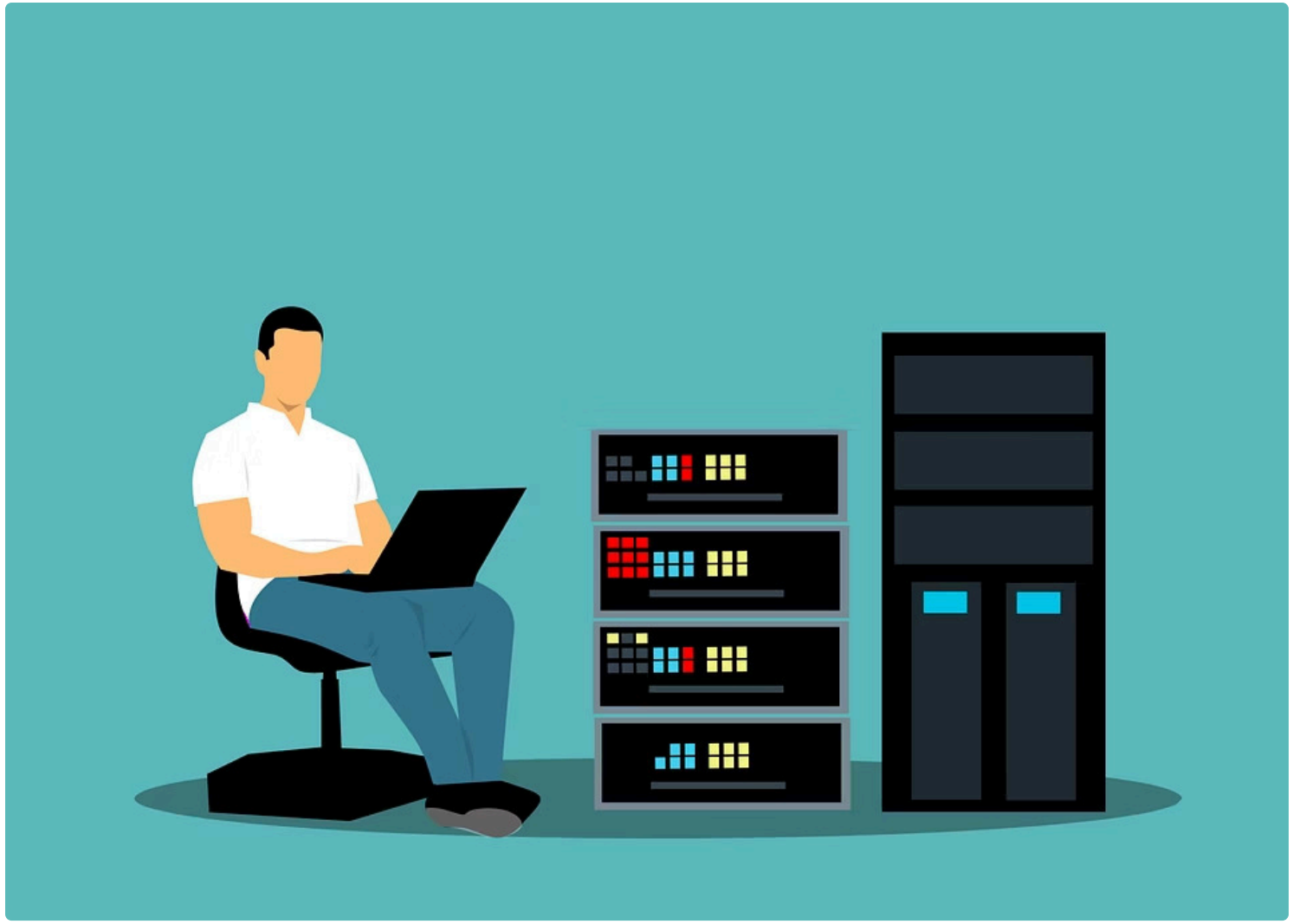
Quando o desenvolvimento de software é necessário, considere aproveitar soluções existentes:

- Bibliotecas e frameworks que resolvem problemas comuns
- Plataformas de desenvolvimento com baixo código (low-code)
- APIs e serviços que podem ser integrados à sua solução
- Software de código aberto que pode ser adaptado às suas necessidades

Não reinvente a roda. Construa sobre os ombros de gigantes para acelerar seu desenvolvimento e reduzir riscos.

# Serviços de Nuvem

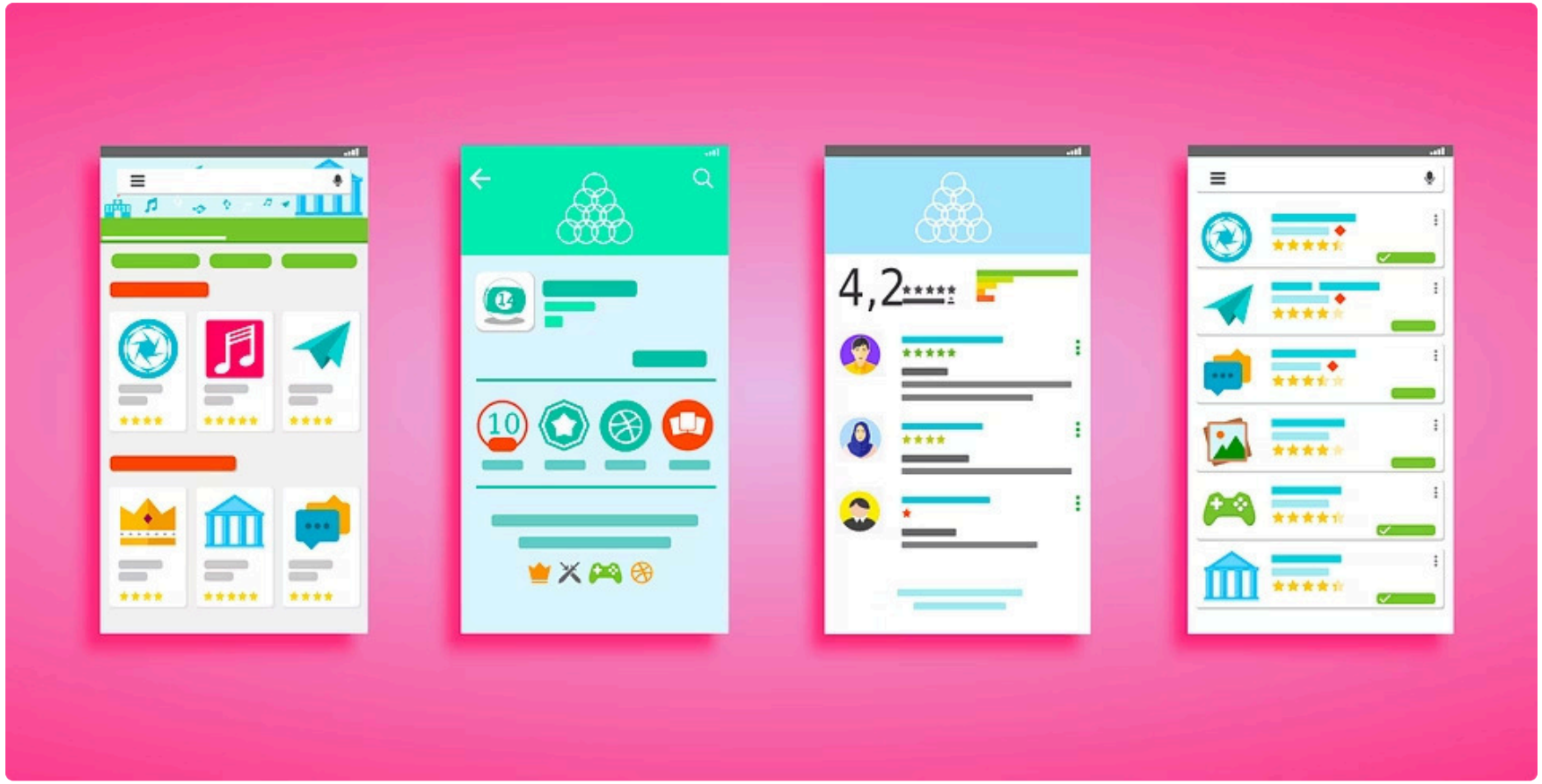
Plataformas de nuvem oferecem infraestrutura, ferramentas e serviços que facilitam o desenvolvimento e implantação de aplicações:



Utilizando serviços em nuvem, você pode focar no valor único que sua solução oferece, enquanto deixa a infraestrutura e serviços comuns para provedores especializados.

# App Store e Soluções Existentes

Muitas vezes, a solução que você precisa já existe como um aplicativo ou serviço:




Antes de desenvolver uma nova solução, pesquise lojas de aplicativos e marketplaces de software para verificar se há algo que atenda às suas necessidades, mesmo que parcialmente.

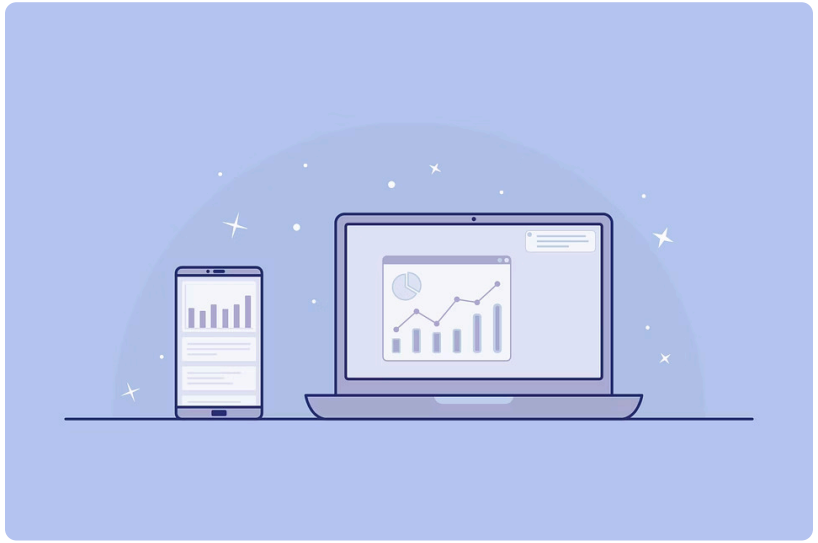
A customização ou integração de soluções existentes pode ser muito mais rápida e econômica do que o desenvolvimento do zero.

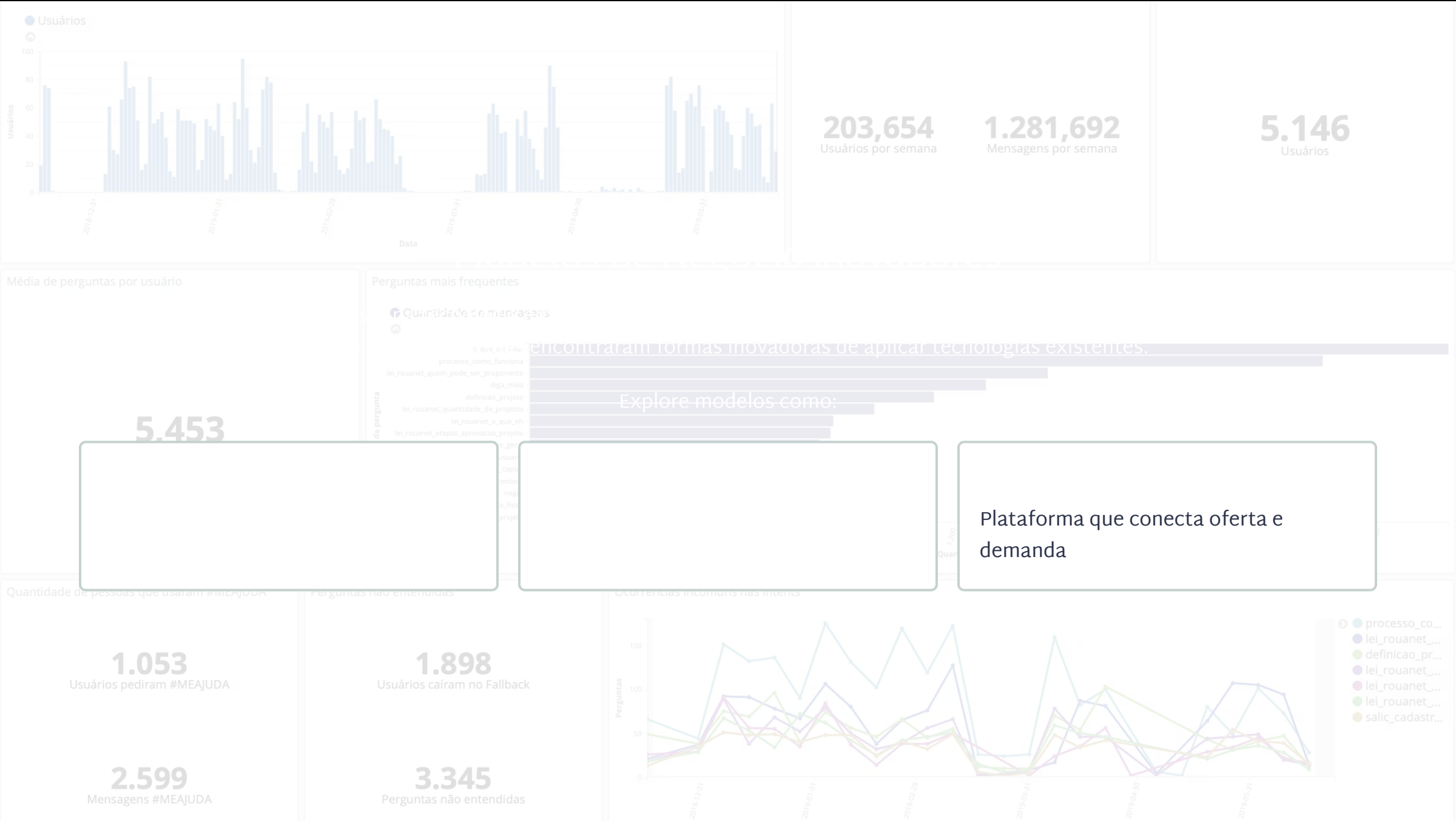
# Business Intelligence (BI) e Pivotamento

Ferramentas de BI podem transformar dados brutos em insights acionáveis sem necessidade de desenvolvimento complexo:

- Dashboards interativos para visualização de dados
- Relatórios automatizados para tomada de decisão
- Análises preditivas para antecipar tendências

Quando a solução inicial não atende às expectativas, esteja preparado para **pivotar** - mudar a direção do projeto com base em aprendizados e feedbacks.




Usuários
Data
203,654
Usuários por semana
1.281,692
Mensagens por semana
5.146
Usuários
Média de perguntas por usuário
5.453
Perguntas mais frequentes
Quantidade de mensagens
encontraram formas inovadoras de aplicar tecnologias existentes.
Explore modelos como:
Plataforma que conecta oferta e demanda
Quantidade de pessoas que usaram #MEAJUDA
1.053
Usuários pediram #MEAJUDA
2.599
Mensagens #MEAJUDA
Perguntas não entendidas
1.898
Usuários caíram no Fallback
3.345
Perguntas não entendidas
Ocorrências incomuns nas intents
Perguntas

# Validação de Mercado

Antes de investir recursos significativos no desenvolvimento, valide se existe demanda real para sua solução.

- **Pesquisa de mercado**
  Identifique seu público-alvo e entenda suas necessidades específicas
- **MVP (Produto Mínimo Viável)**
  Desenvolva apenas as funcionalidades essenciais para testar a aceitação
- **Feedback contínuo**
  Estabeleça canais para receber e analisar o feedback dos usuários

# Estratégias de Crescimento

Após validar seu conceito e conquistar os primeiros usuários, planeje estratégias para escalar seu produto.


### Marketing Digital

SEO, mídias sociais, marketing de conteúdo

### Parcerias Estratégicas

Alianças com empresas complementares

### Internacionalização

Adaptação para mercados globais

O crescimento sustentável requer não apenas um bom produto, mas também estratégias eficazes de aquisição e retenção de usuários.

# Equipe de Desenvolvimento

O sucesso de um projeto de software depende significativamente da qualidade e organização da equipe de desenvolvimento. Uma equipe eficaz combina:

- Diversidade de habilidades técnicas
- Comunicação clara e constante
- Autonomia com responsabilidade
- Processos bem definidos mas adaptáveis
- Cultura de aprendizado contínuo

Ao longo desta disciplina, vocês trabalharão em equipes para simular um ambiente real de desenvolvimento, aplicando os conceitos estudados em projetos práticos.

Lição #4:

# Escolha um time com motivação intrínseca

Equipes verdadeiramente motivadas pela missão e desafios do projeto tendem a ser mais criativas, resilientes e produtivas.

https://dulce-work-schedule.github.io/index.html

## Lição #5:

# Colabore com:

### Comunidades

Participe de grupos, fóruns e eventos onde desenvolvedores compartilham conhecimento, experiências e soluções para problemas comuns.

### Software Livre

Contribua para projetos de código aberto e aproveite soluções existentes que podem acelerar seu desenvolvimento e reduzir custos.

A colaboração amplia perspectivas, traz novas ideias e permite aproveitar o conhecimento coletivo para resolver problemas complexos.

# Comunidades Open Source

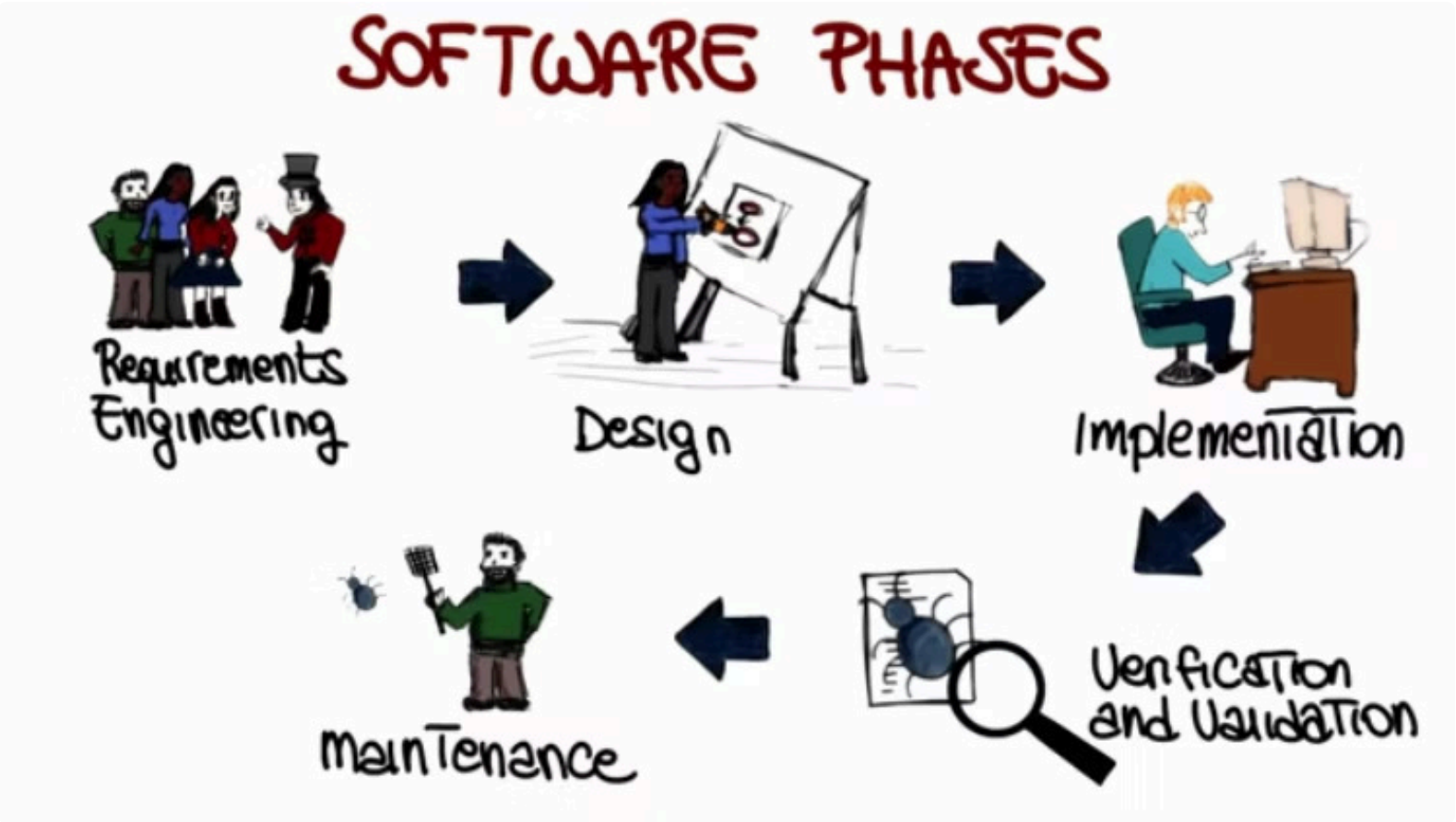

Comunidades de software livre são ecossistemas vibrantes onde desenvolvedores colaboram para criar e melhorar soluções tecnológicas.

Benefícios de participar:

- Aprendizado prático com problemas reais
- Feedback de desenvolvedores experientes
- Networking profissional
- Visibilidade para potenciais empregadores
- Satisfação de contribuir para algo maior

# Sistematizar Projetos de Software (Fases do Software)

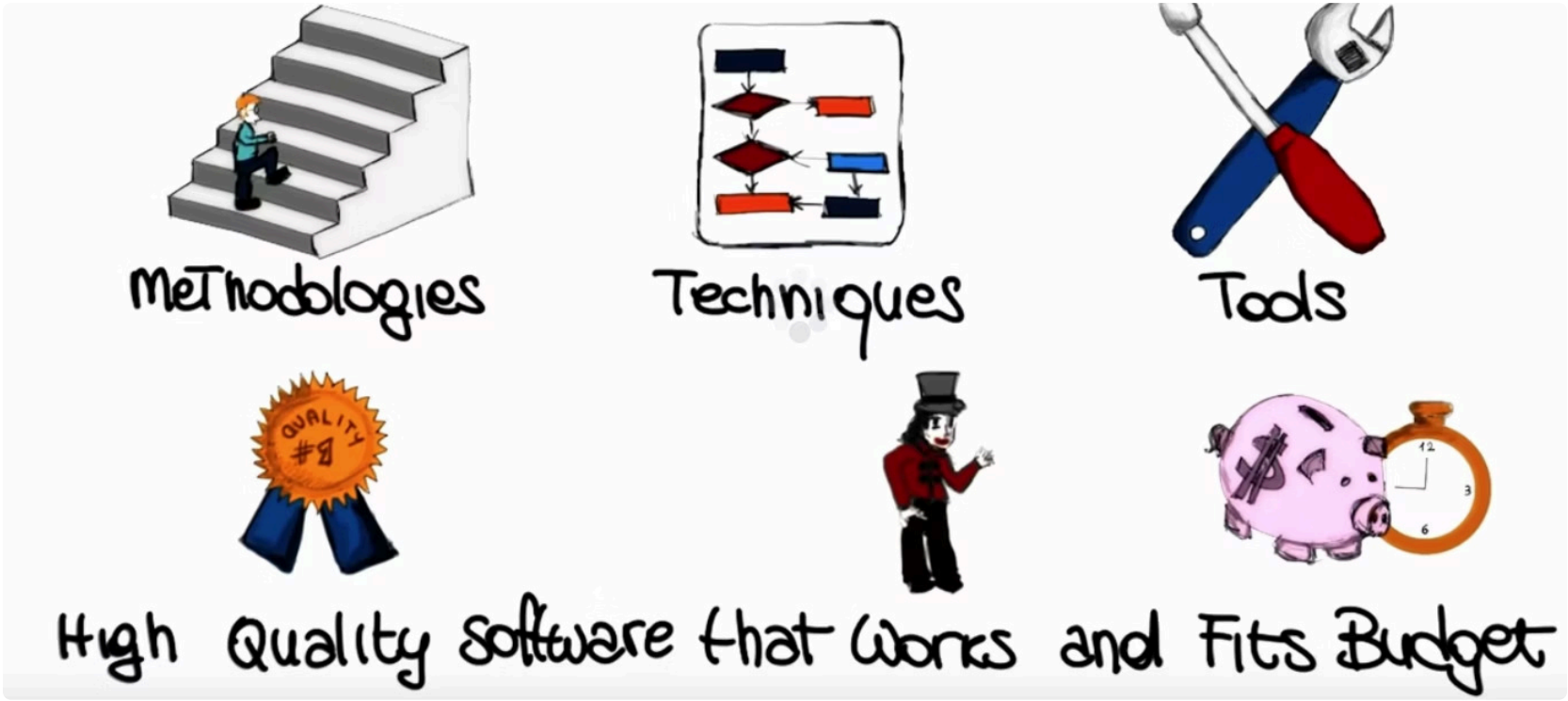
O desenvolvimento de software pode ser sistematizado em fases bem definidas, independentemente da metodologia adotada. Estas fases garantem uma abordagem estruturada para transformar necessidades em soluções:




Cada fase produz artefatos específicos que servem como entrada para as fases subsequentes, criando um fluxo contínuo e estruturado de desenvolvimento.

# Solução para a Crise de Software

## Processo de Desenvolvimento de Software

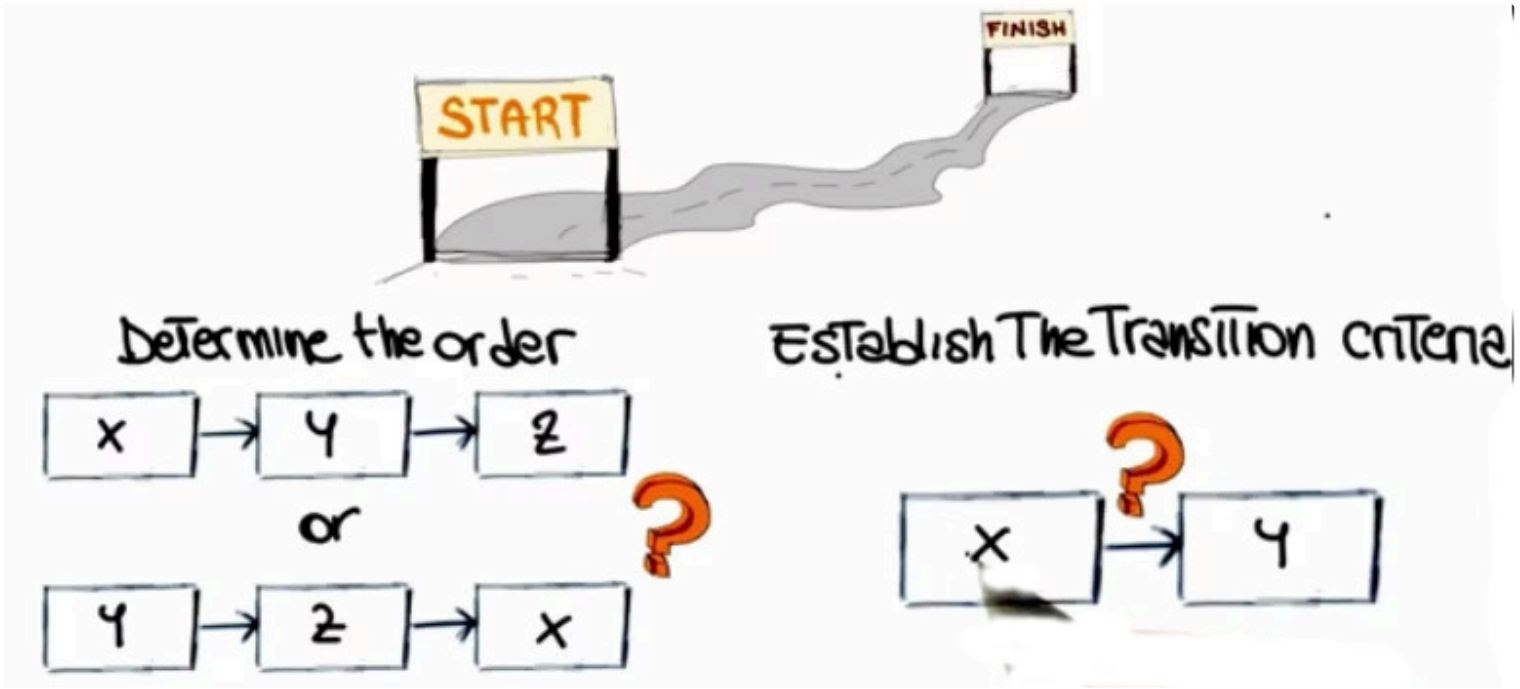



A crise de software dos anos 60 e 70 evidenciou a necessidade de processos estruturados para o desenvolvimento. Processos bem definidos ajudam a:

- Garantir previsibilidade e controle
- Facilitar a comunicação entre equipes
- Melhorar a qualidade do produto final
- Reduzir riscos e custos de desenvolvimento

# Processo de Desenvolvimento de Software

## Diversas formas de executar as fases do sistema de Software




Existem diversos modelos de processo de desenvolvimento, cada um com vantagens e desvantagens específicas:

- Modelo Cascata: fases sequenciais e bem definidas
- Modelo Iterativo: ciclos de desenvolvimento com entregas incrementais
- Métodos Ágeis: adaptabilidade, colaboração e entregas frequentes
- DevOps: integração contínua e entrega contínua

# O Grande Problema da Engenharia de Software

## Como desenvolver sistemas de software:

### Valor de Mercado

Que atendam às necessidades reais dos usuários

### Qualidade

Confiáveis, seguros e eficientes

### Entregas Rápidas

Com time-to-market competitivo

### Manutenção

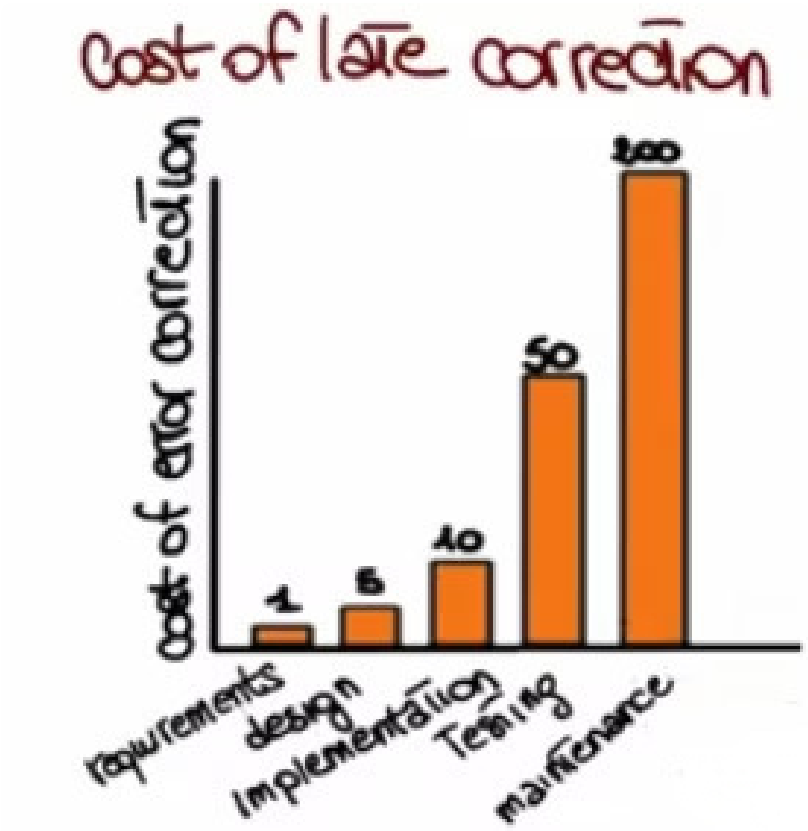
Fáceis de modificar e atualizar

### Reuso e Modularidade

Componentes reutilizáveis e bem estruturados


Balancear todos estes fatores é o desafio central da engenharia de software moderna.

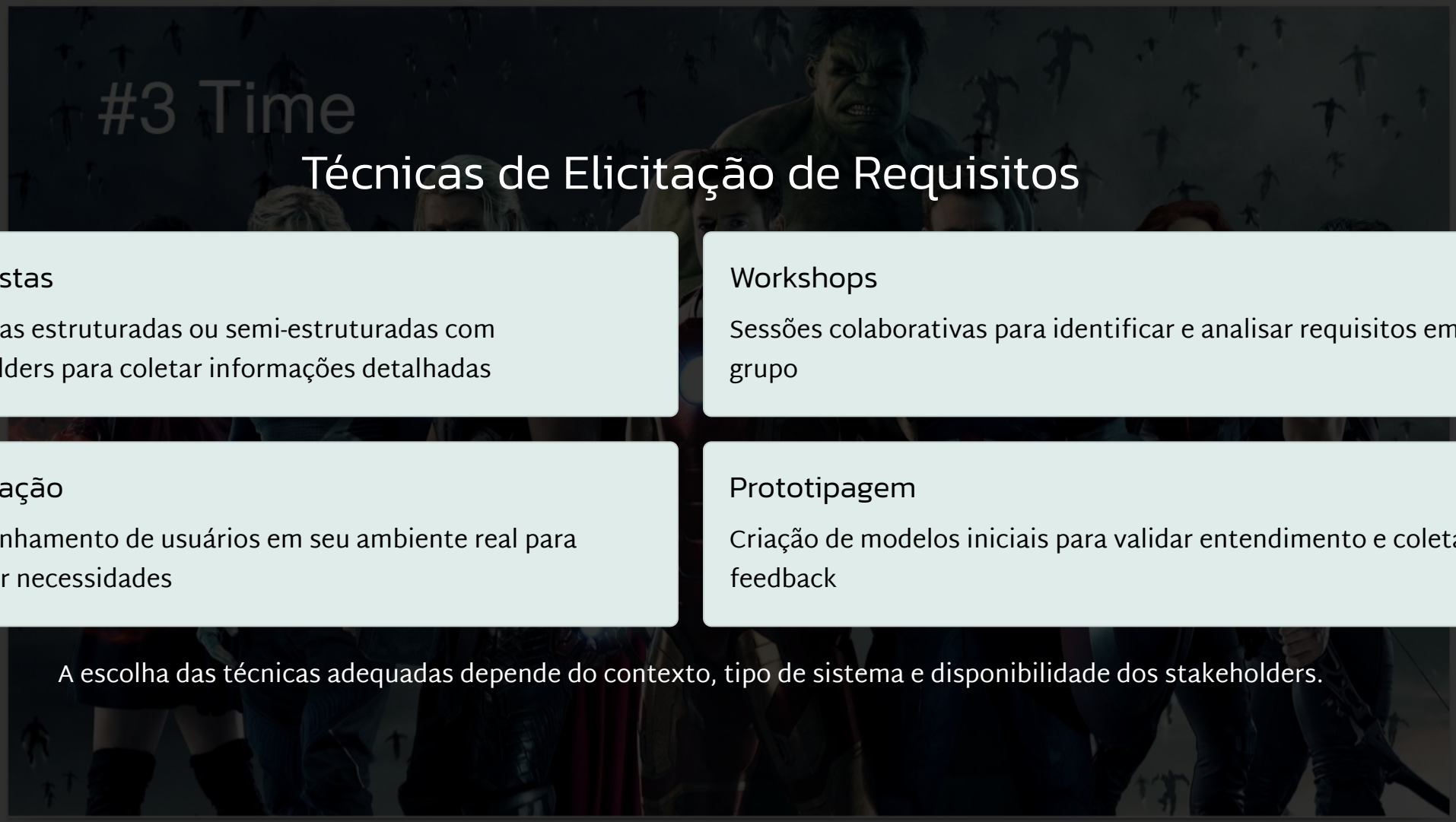
# Engenharia de Requisitos




A Engenharia de Requisitos é o processo de descobrir, analisar, documentar e validar as necessidades dos stakeholders para um sistema de software.

Fases principais:

1. **Elicitação**: Descobrir requisitos através de entrevistas, observação, workshops
2. **Análise**: Examinar requisitos para resolver conflitos e ambiguidades
3. **Especificação**: Documentar requisitos de forma clara e precisa
4. **Validação**: Confirmar que os requisitos representam o que os stakeholders realmente necessitam
5. **Gerenciamento**: Controlar mudanças nos requisitos ao longo do projeto




# Técnicas de Elicitação de Requisitos

### Entrevistas

Conversas estruturadas ou semi-estruturadas com stakeholders para coletar informações detalhadas

### Workshops

Sessões colaborativas para identificar e analisar requisitos em grupo

### Observação

Acompanhamento de usuários em seu ambiente real para entender necessidades

### Prototipagem

Criação de modelos iniciais para validar entendimento e coletar feedback

A escolha das técnicas adequadas depende do contexto, tipo de sistema e disponibilidade dos stakeholders.

# Desafios na Engenharia de Requisitos

Capturar requisitos precisos é um dos principais desafios no desenvolvimento de software. Problemas comuns incluem:

- Stakeholders não sabem exatamente o que querem
- Requisitos mudam ao longo do tempo
- Diferentes stakeholders têm expectativas conflitantes
- Dificuldade em expressar necessidades de forma clara

## Um problema

Requisitos mal definidos podem levar a retrabalho extensivo, atrasos significativos e até ao fracasso completo do projeto.

# Implementação

A fase de implementação transforma os requisitos e a arquitetura em código funcional. É quando o sistema realmente ganha vida:

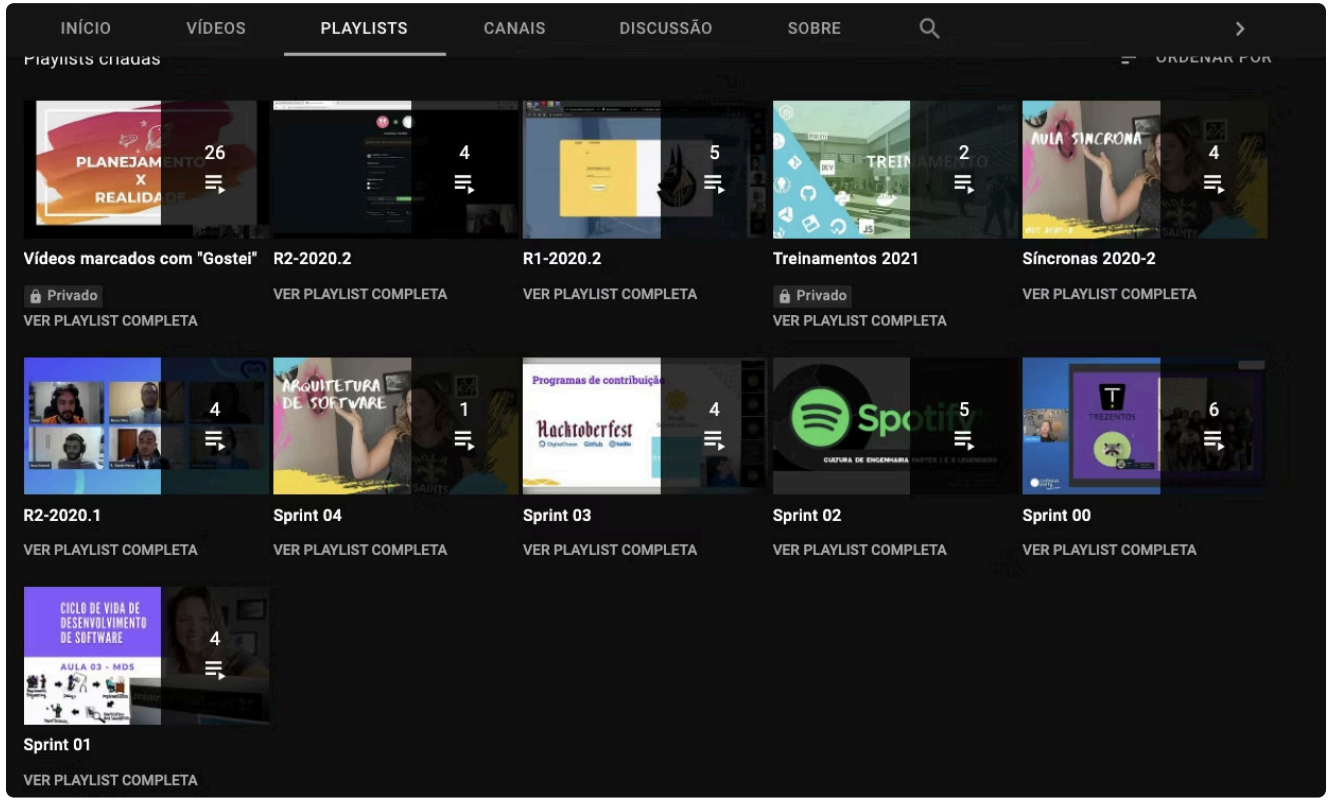

Boas práticas de programação são essenciais para criar código de qualidade:

- Legibilidade e organização do código
- Documentação adequada
- Testes unitários e de integração
- Controle de versão eficiente

open source initiative

# Material de Apoio

Para apoiar o aprendizado nesta disciplina, disponibilizamos diversos recursos complementares:




Acesse nosso canal no YouTube com playlists organizadas por tópicos:

https://www.youtube.com/channel/UC6VgsVODs17IAHuWF2HCfUQ/playlists

Lá você encontrará tutoriais, explicações detalhadas e exemplos práticos para complementar o conteúdo visto em aula.

# Mandamentos MDS

1

## Se organize

Não acumule trabalho! Planeje-se para dedicar tempo constante ao projeto.

2

## Não se apegue à linguagens

Elas são apenas ferramentas. O importante é entender conceitos e saber quando aplicá-los.

3

## As 6 horas semanais são sagradas!

Dedique este tempo mínimo para as atividades da disciplina.

4

## Tenha calma×5

Desenvolvimento de software é desafiador. Respire fundo e persista.

5

## Ouça os Tech Leaders!

Os monitores estão aqui para ajudar com sua experiência.

6

## Professora é facilitadora!

Estou aqui para guiar seu aprendizado, não para dar todas as respostas.

7

## Divirta-se

Aproveite o processo de aprendizado e criação. A jornada é tão importante quanto o destino.

# Principais Problemas/Riscos na Disciplina

## Relatados por grupos de semestres anteriores

### Falta de compromisso

Membros que não dedicam as 6 horas semanais comprometem a entrega do projeto inteiro

### Falhas na comunicação

Tanto entre membros da equipe quanto com stakeholders, gerando retrabalho e frustração

### Não ouvir a professora

Ignorar orientações e feedback leva a problemas que poderiam ser evitados facilmente

### Reação tardia aos riscos

Não assumir riscos quando necessário e reagir tarde demais aos problemas identificados

Estar consciente destes riscos desde o início pode ajudar sua equipe a evitá-los proativamente.

# Lição #2:

"Pratique uma habilidade de engenharia de software importante: *use a ferramenta correta* para o trabalho, mesmo que isso signifique *aprender* uma *nova ferramenta* ou uma *nova linguagem*"

Adaptar-se a novas tecnologias é uma constante na carreira de um desenvolvedor. A disposição para aprender continuamente é tão importante quanto o conhecimento técnico que você já possui.

# Lição #0:

"Sejamos claros: *Sua carreira* é *sua* responsabilidade, seu empregador *não é sua Mãe*" – Robert C. Martin

Seu crescimento profissional depende das suas escolhas, iniciativas e comprometimento com o aprendizado contínuo. Ninguém fará isso por você.

# Entendendo o Problema e Levantando Necessidades

Antes de começar a codificar, é essencial compreender profundamente o problema que se pretende resolver:

01

## Entender o Problema

Identifique o problema real, não apenas seus sintomas. Quem são os afetados? Qual o impacto?

02

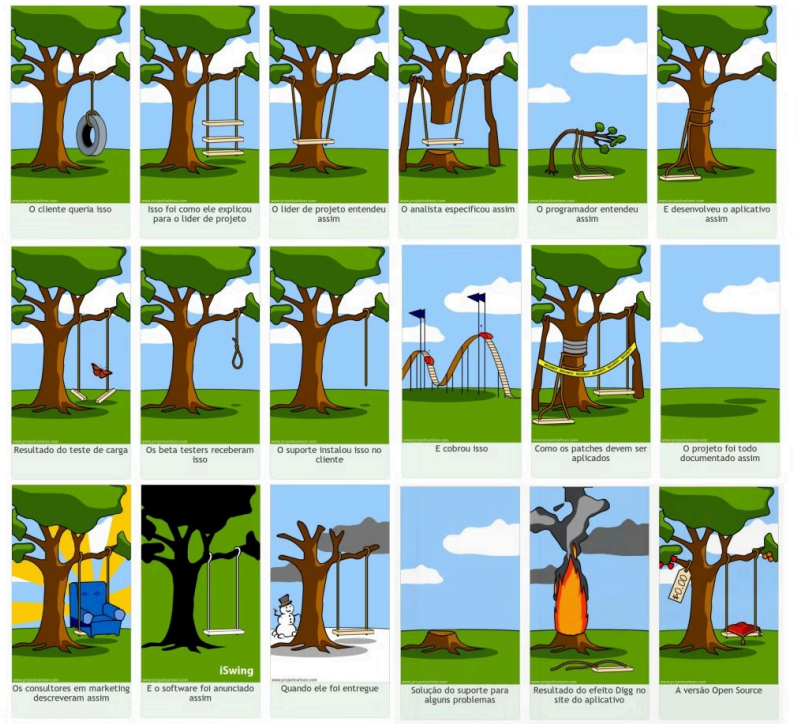
## Levantar Necessidades

Converse com stakeholders para entender suas dores, expectativas e contexto de uso.

03

## Listar Funcionalidades

Traduza necessidades em features concretas que deverão ser desenvolvidas.

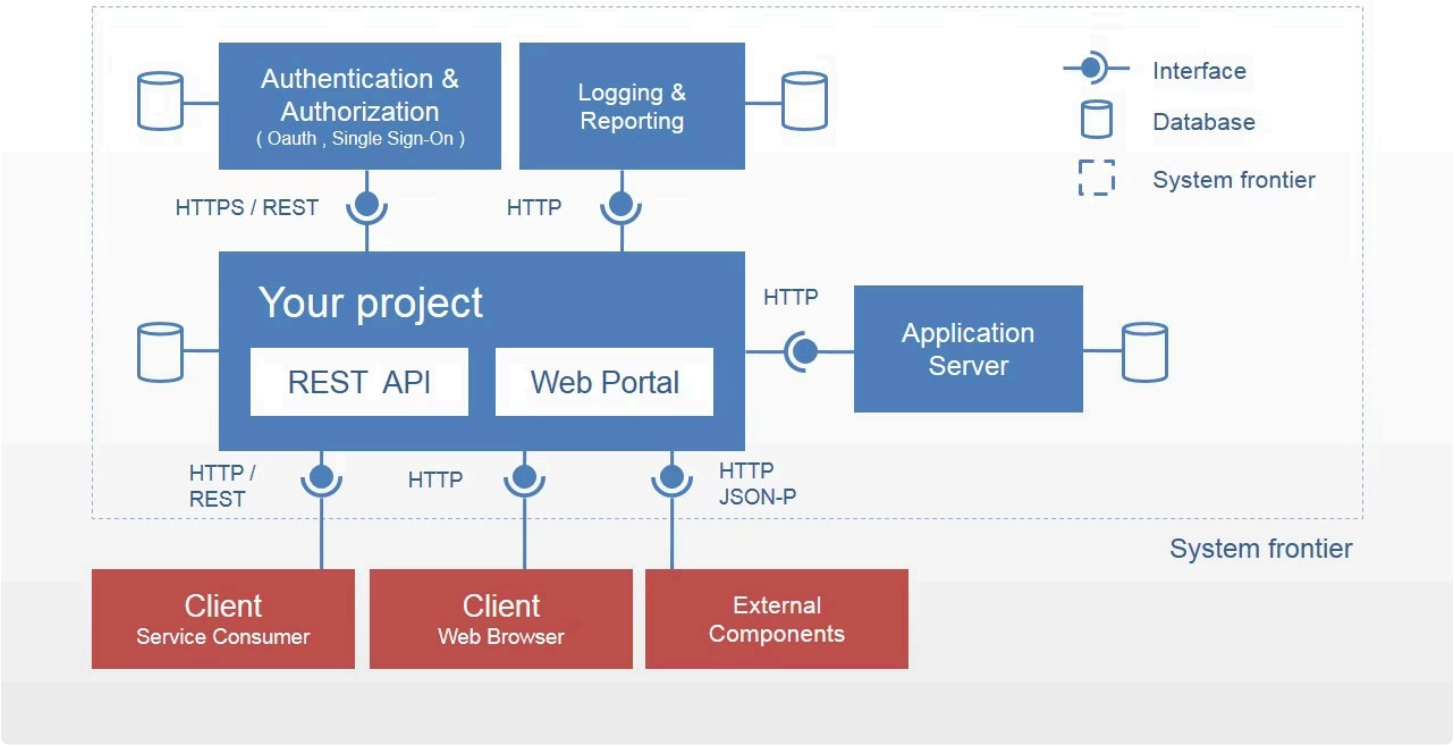



Esta etapa inicial é crítica - um problema mal compreendido levará invariavelmente a uma solução inadequada, independentemente da qualidade da implementação técnica.

# Arquitetura (Design)

## Estrutura de Alto Nível do Sistema de Software

A arquitetura define a organização fundamental do sistema, estabelecendo seus componentes, relacionamentos, e princípios que guiam seu design e evolução:

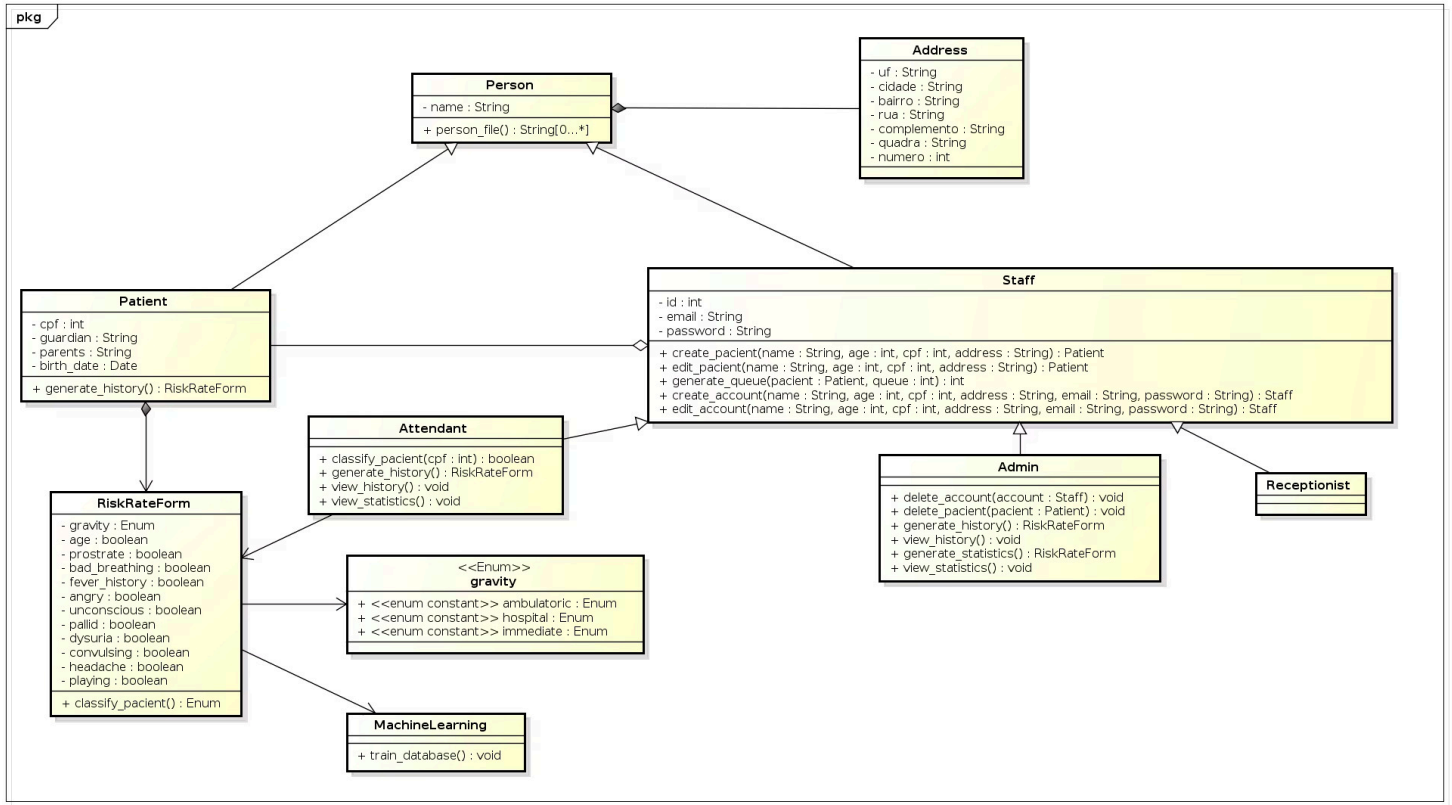



Uma boa arquitetura equilibra requisitos funcionais (o que o sistema faz) com requisitos não-funcionais (como ele faz), considerando aspectos como desempenho, segurança, escalabilidade e manutenibilidade.

# Arquitetura (Design)

## Estrutura de Alto Nível do Sistema de Software

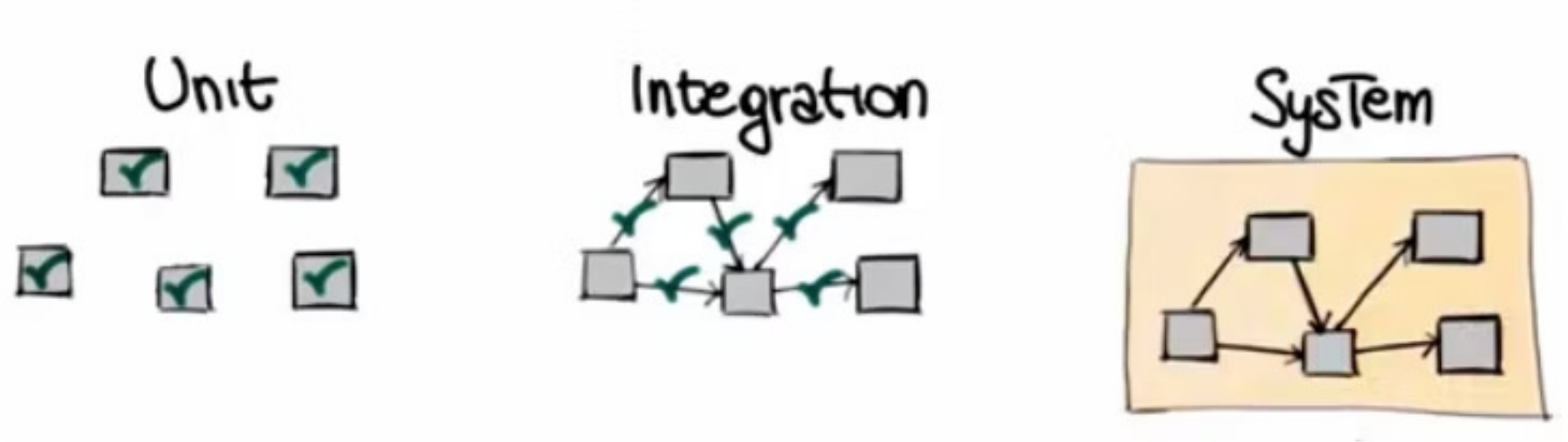
Diagramas de classes são uma das ferramentas mais comuns para representar a arquitetura orientada a objetos, mostrando as classes do sistema e seus relacionamentos:




Além de diagramas de classes, outros modelos arquiteturais incluem diagramas de componentes, diagramas de sequência, diagramas de implantação e diagramas de fluxo de dados, cada um destacando diferentes aspectos do sistema.

# Verificação e Validação

Verificação e validação são processos complementares que garantem a qualidade do software:




### Verificação

Avalia se o software está sendo construído corretamente, conforme as especificações. Foca em encontrar defeitos no produto.

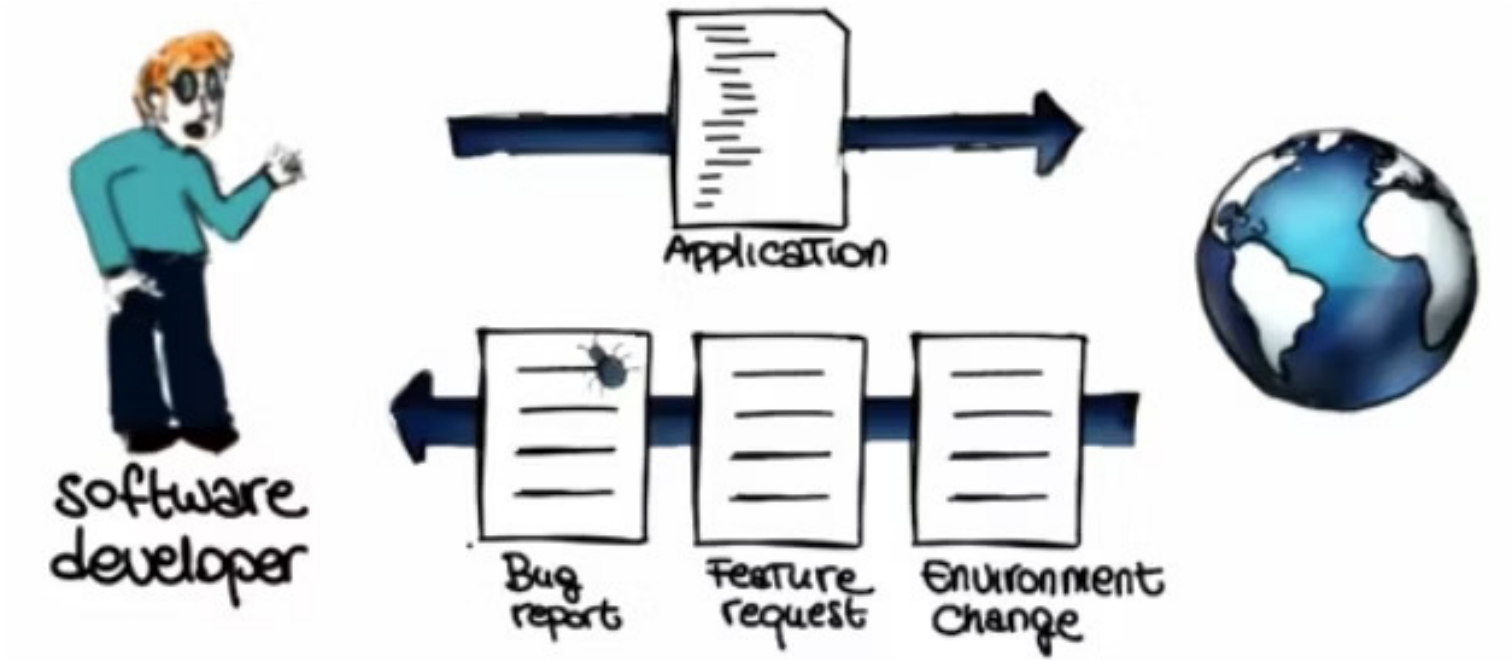
### Validação

Avalia se o software correto está sendo construído, ou seja, se atende às necessidades reais dos usuários.

Resumindo: Verificação pergunta "Estamos construindo o produto corretamente?", enquanto validação pergunta "Estamos construindo o produto correto?".

# Implementação e Integração Contínua

Práticas modernas de desenvolvimento adotam a integração contínua para detectar problemas precocemente:




A integração contínua envolve mesclar as alterações de código na base principal com frequência, executando testes automatizados para validar cada integração.

Benefícios incluem:

- Detecção precoce de problemas de integração
- Feedback rápido para desenvolvedores
- Redução do esforço para mesclar alterações
- Maior visibilidade do progresso do desenvolvimento

# Métodos de Desenvolvimento de Software


## Profa. Dra. Carla Rocha

rocha.carla@gmail.com


https://github.com/fga-gpp-mds

Obrigada pela atenção! Estou à disposição para dúvidas e discussões sobre os temas abordados.

Lembre-se: o aprendizado em engenharia de software é uma jornada contínua. A teoria que vimos aqui ganhará vida através dos projetos práticos que desenvolveremos ao longo do semestre.