

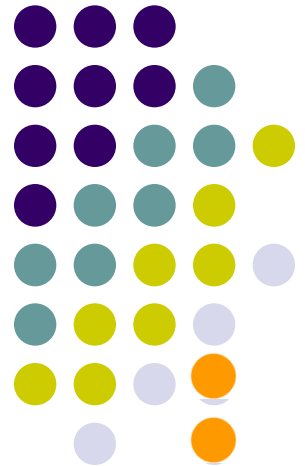
Arquitetura de Software

P&D Software

Processo de Mudança Contínua

José Motta Lopes

josemotta@bamplicom



Agenda



- **Metodologia Waterfall**
- **Ciclo do Processo Waterfall**
- **Metodologia Agile**
- **Story Mapping**
- **Ciclo do Processo Agile**
- **Github**
- **Github Flow**
- **Git**



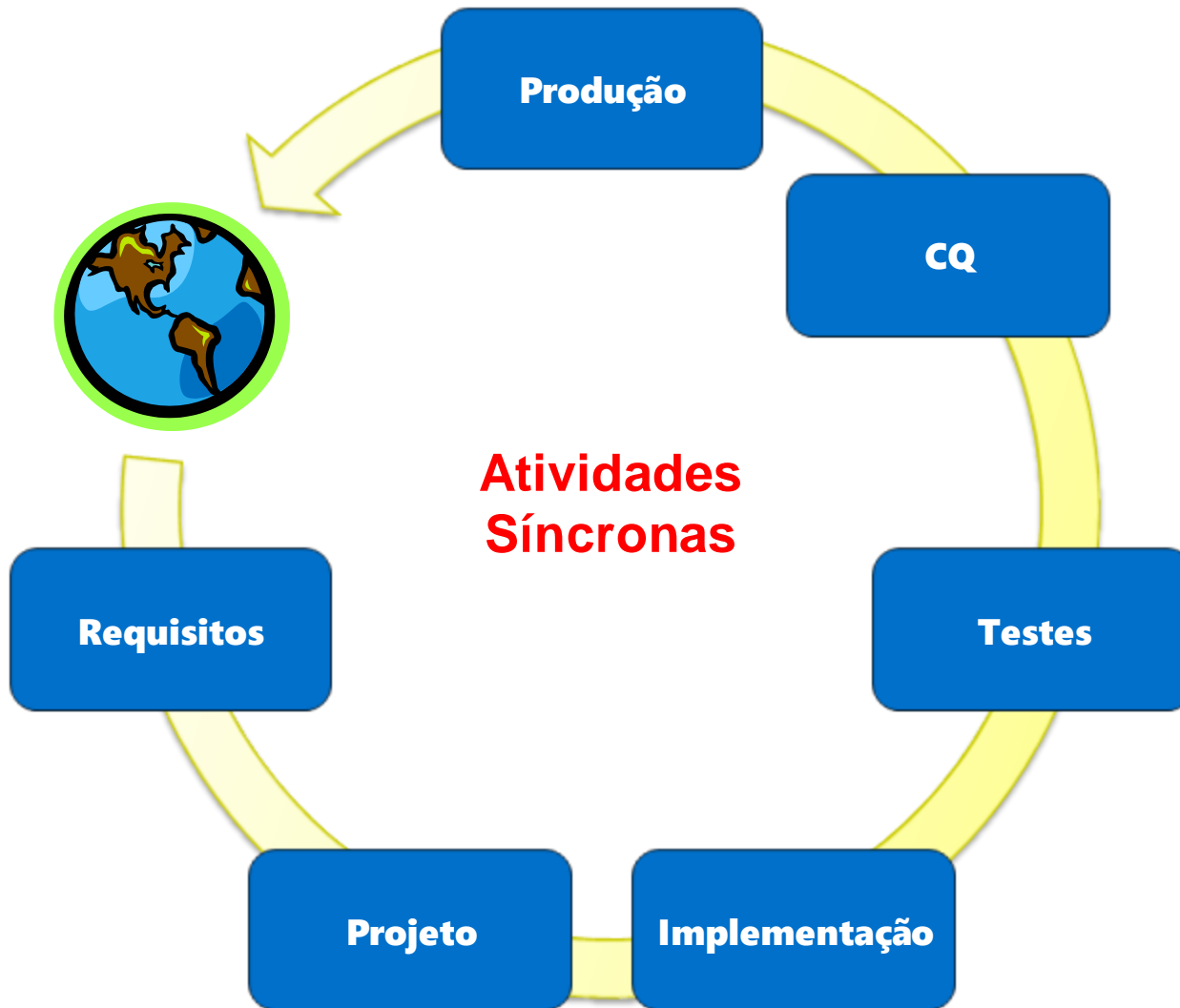


Metodologia Waterfall

- Surgiu em 1970, origem acadêmica
- Documentação longa e detalhada
 - Requisitos dos negócios
 - Especificação funcional
- Especificação Técnica
 - Arquitetura da Aplicação
 - Estruturas de Dados
 - Projeto Orientado a Objetos
 - Interfaces Usuário
- Codificação
- Integração
- Testes (Unit Tests e Integration Tests)
- Produção (Deploy, Operação e Manutenção)



Ciclo do Processo Waterfall



ciclo do processo com atividades síncronas e serializadas:

- atividades enormes
- longa duração
- baixa velocidade



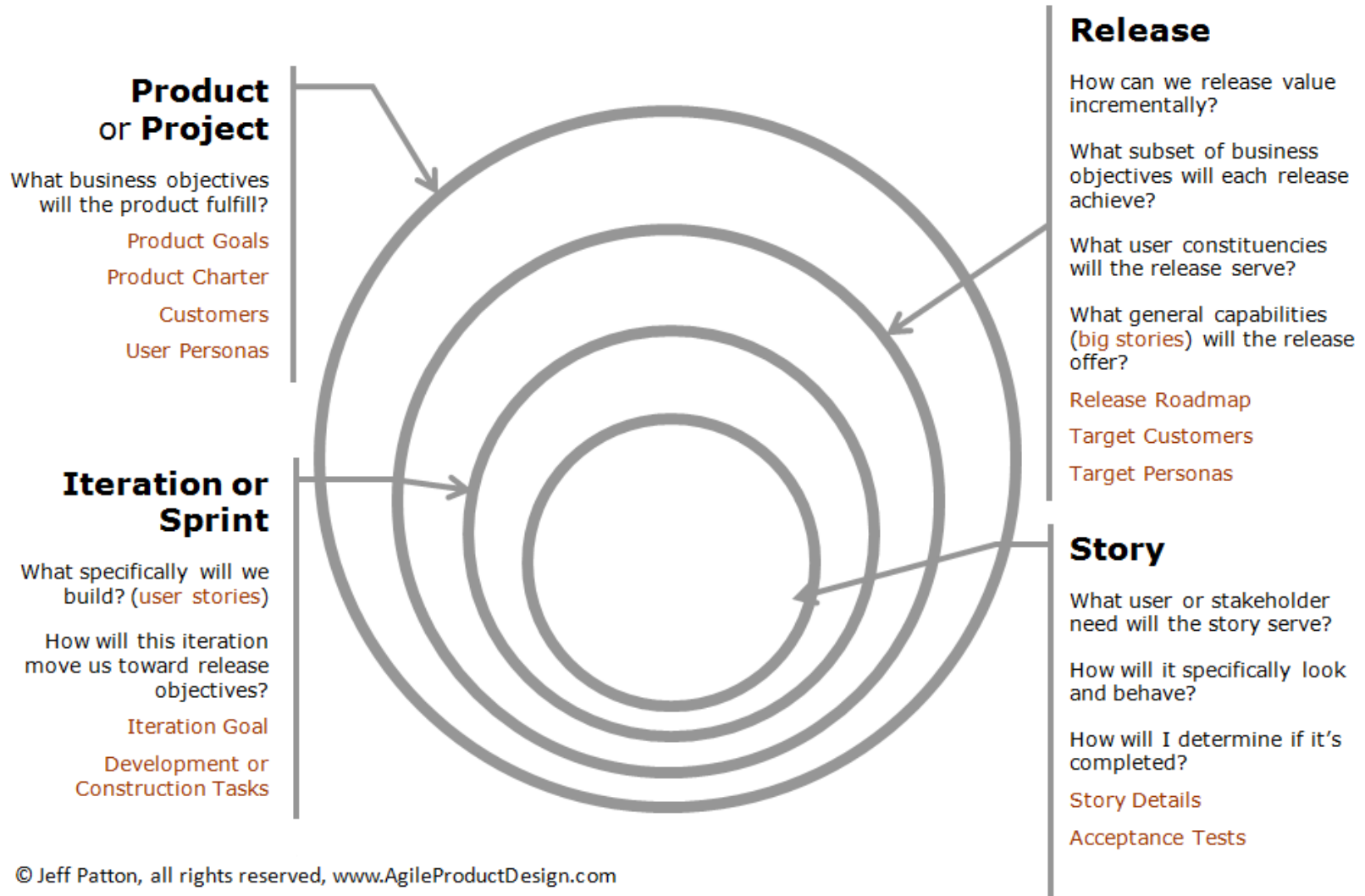
Ciclo ~ 1-2 anos

Metodologia Agile

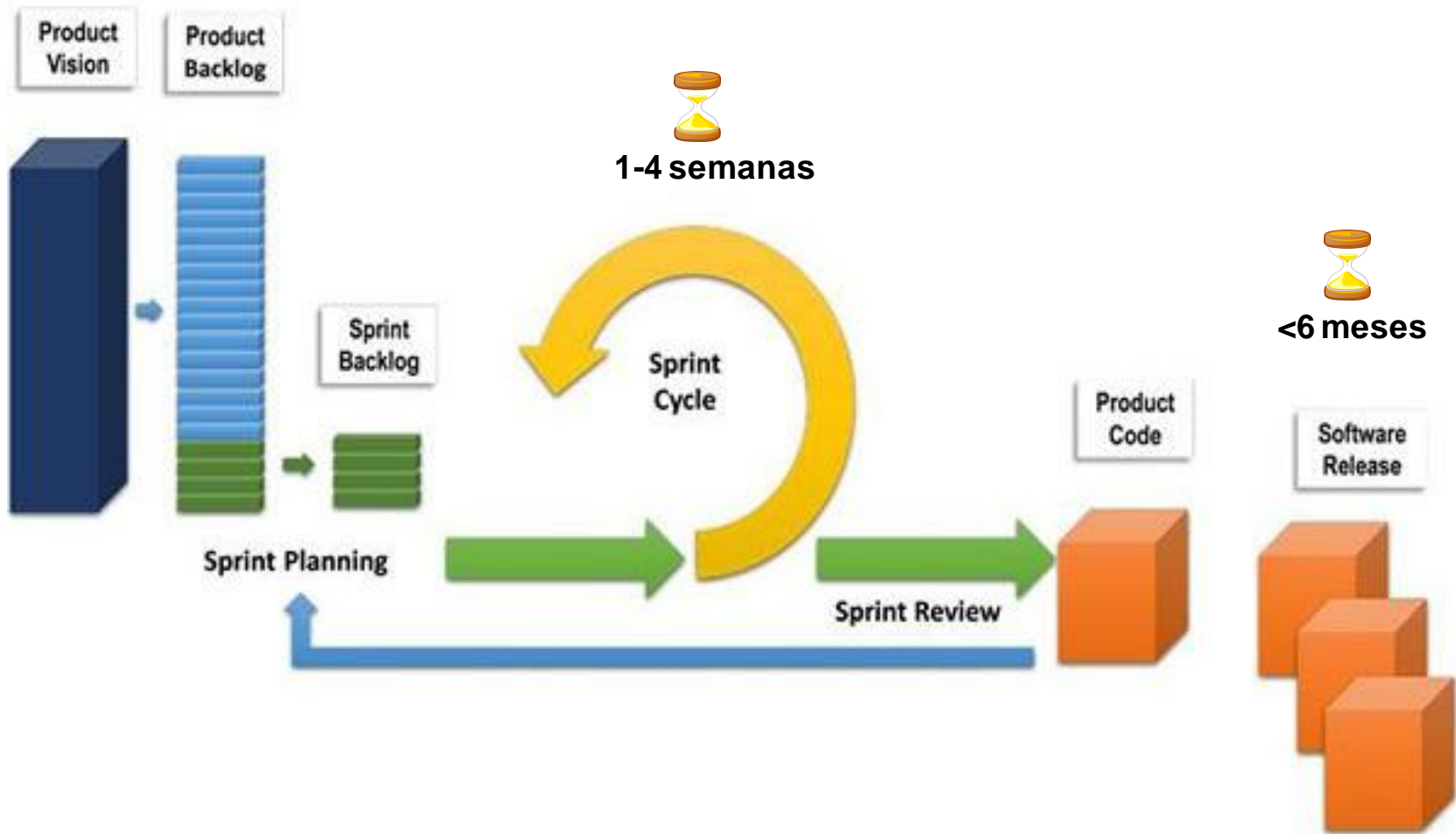


- Surgiu em 2001, origem no mercado
- Desenvolvedores trabalhavam com aplicações Internet
- Startups sofriam pressão competitiva para produzir rápido
- Empresas pequenas, sem formação tradicional de sistemas
- Prioridade total para ouvir a opinião de usuários
- Desenvolvedores não aceitavam regras “waterfall”
- Não dava tempo para fazer primeiro toda a documentação
- Cronogramas de longo prazo dos gerentes não funcionavam
- Desenvolvedores decidiam sobre a engenharia das aplicações
- Cronograma iterativo passou a descrever os compromissos
- Intervalos normalmente de 1-4 semanas
- A partir de princípios práticos, foi elaborado o [Agile Manifesto](#).

Metodologia Agile



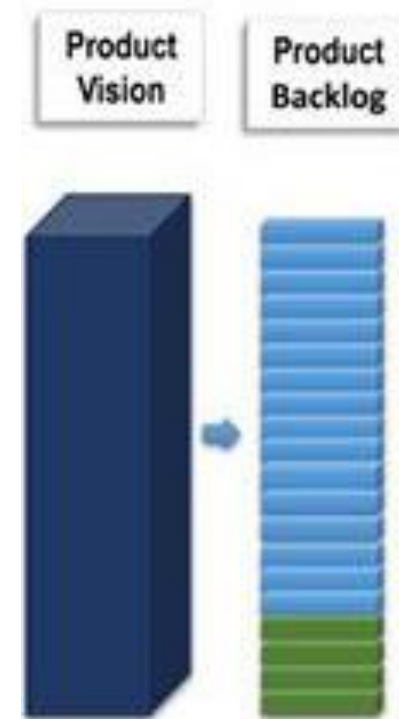
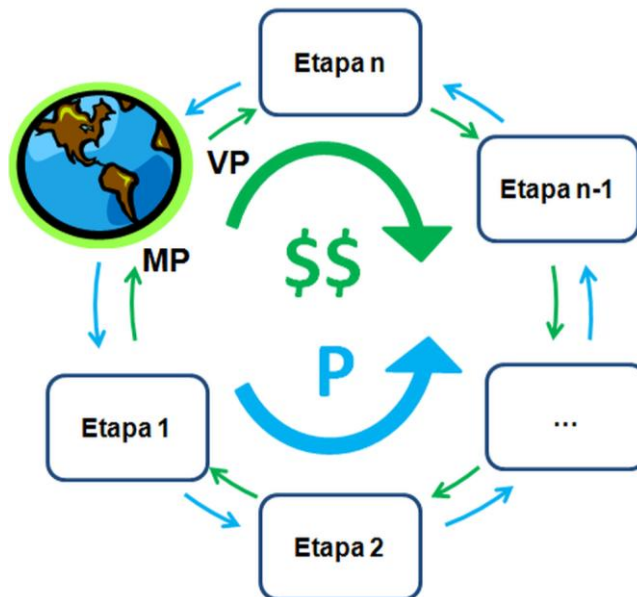
Metodologia Agile





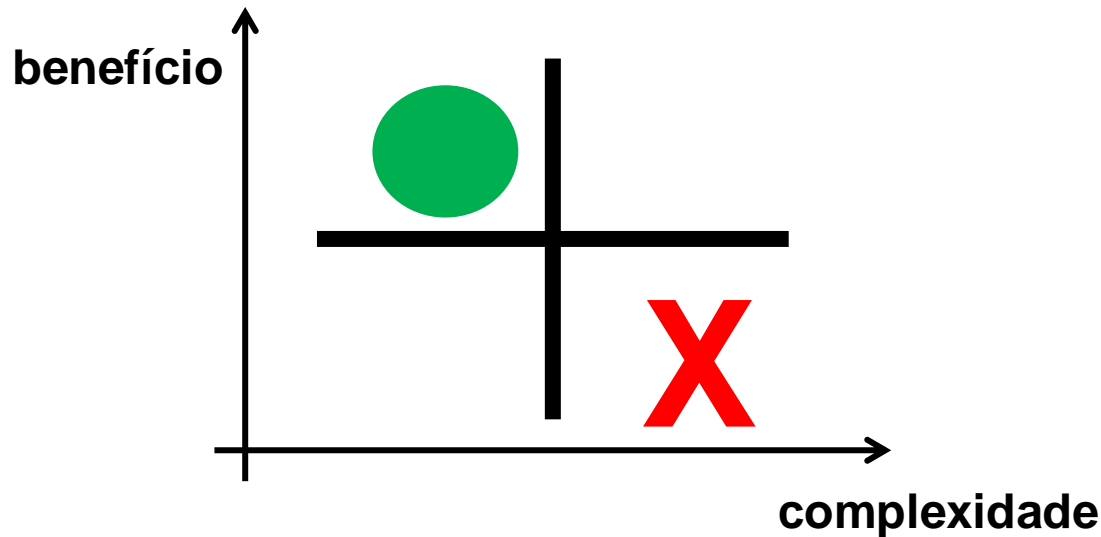
Produto ou Projeto

- Qual o ciclo do processo do negócio?
- Quais os objetivos do produto/projeto?
- O que os consumidores querem obter?
- Quais os personagens envolvidos?





Release



- Como entregar incrementalmente o maior valor?
- Quais os objetivos do negócio alcançados?
- Como os consumidores serão beneficiados?
- Quais personagens serão afetados?
- Quais as “big stories” do Release?
- Detalhamento do roteiro (roadmap)

Sprint (Iteração)



- Quais histórias serão construídas?
- Como essa iteração nos aproxima dos objetivos do Release?
- Quais os objetivos do Sprint?
- Quais tarefas a completar?

Story



- Qual necessidade do consumidor / investidor estará atendida?
- Como irá se comportar?
- Qual a aparência esperada?
- Como determinar se está completada?
- Quais os testes de aceitação?

Story Mapping



Uma pilha achatada de cartões não ajuda muito a verificar se todas as estórias foram identificadas!

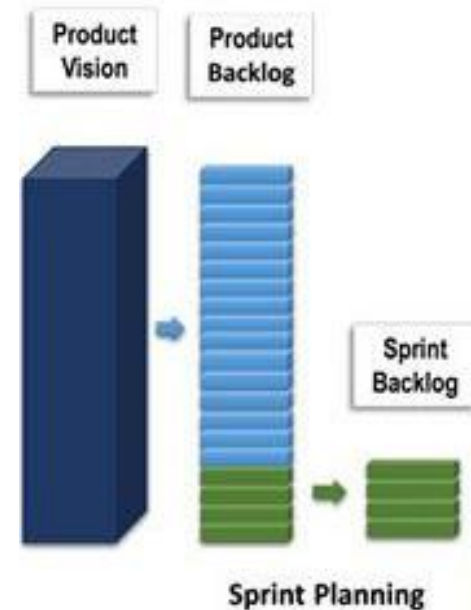
Referências:

[Jeff Patton Associates](#)

[The New User Story Backlog is a Map](#)

[How you slice it](#)

[Story Map Quick Reference](#)





Processo do Story Map

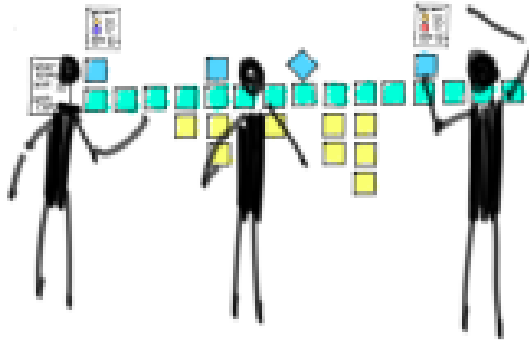
1. Roteirizar
2. Mapear o esqueleto (Big Picture)
3. Completar e explorar o mapa
4. Especular sobre “*releases*” viáveis
5. Estabelecer estratégia de desenvolvimento

Story Map - Roteirizar



- **Antes de mapear, resumir as características do produto**
- **Nomear os diferentes tipos de usuário que utilizarão o produto**
- **Qual a importância do produto para a empresa?**
- **Quais os problemas a serem resolvidos?**
- **Quais os benefícios que a empresa terá com produto?**

Story Map – Big Picture



- **Focar na história completa**
- **Construir o esqueleto do mapa**
- **Qual a jornada de trabalho do consumidor?**
- **Qual o tipo de usuário mais crítico para o sucesso do produto?**
- **Identificar as atividades dos consumidores / personagens**
- **Adicionar personagens, à medida que usos típicos são criados**

Story Map – Explorar



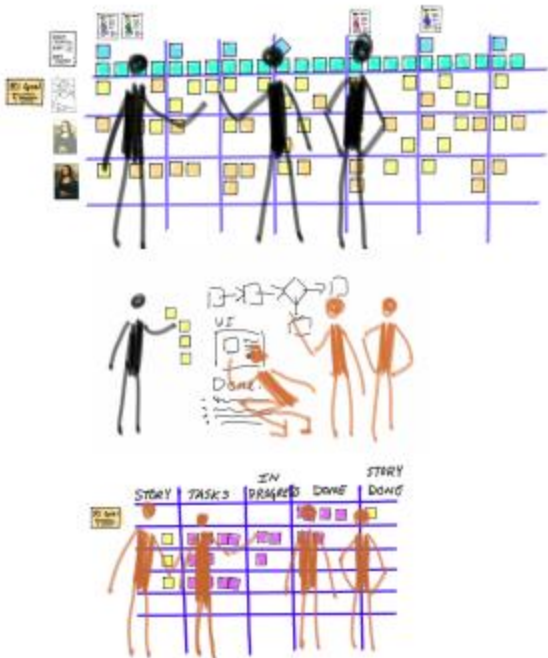
- **Preencher o corpo do mapa com subtarefas, dividir estórias**
- **Pensar no que pode dar errado**
- **Pensar no que seria ótima idéia para o produto**
- **Quais as possíveis variações?**
- **Qual seriam as alternativas para UI?**
- **Discutir as regras de negócio envolvidas**
- **Envolver outros interessados para validar o que já foi feito**

Story Map – Releases viáveis



- **Fatiar o mapa para expor Releases mínimos viáveis do produto**
- **Como cada Release irá contribuir para o objetivo global?**
- **Preveja como usuários poderão se comportar para ajudar**
- **Para cada Release identificar as métricas para o sucesso**

Story Map – Estratégia



- **Fatiar primeiro Release**
- **O time deve aprender e engrenar rápido**
- **Evitar riscos**
- **Qual a menor versão funcional do produto para iniciar o desenvolvimento?**
- **Depois vai completando funcionalidades**
- **Testar performance**
- **Testar escalabilidade**
- **Realizar os refinamentos do produto**

Story Mapping



Personagens:

- Usuários
- Investidores
- Dono do Produto
- Gerentes de Negócio
- Time de Desenvolvedores

Story:

Como um <Personagem>
Eu quero <Algo>
Para obter um <Benefício>

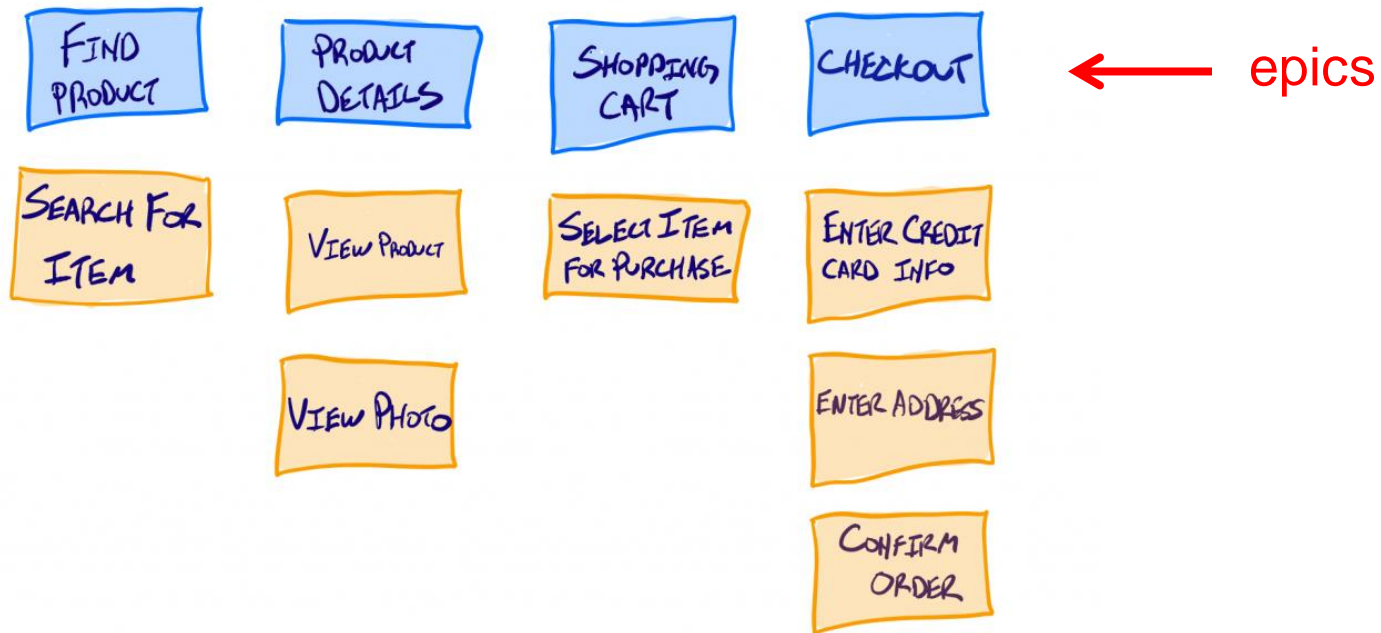
Agile Framework:

Scrum

Story Mapping



GROUP + DEFINE ACTIVITIES



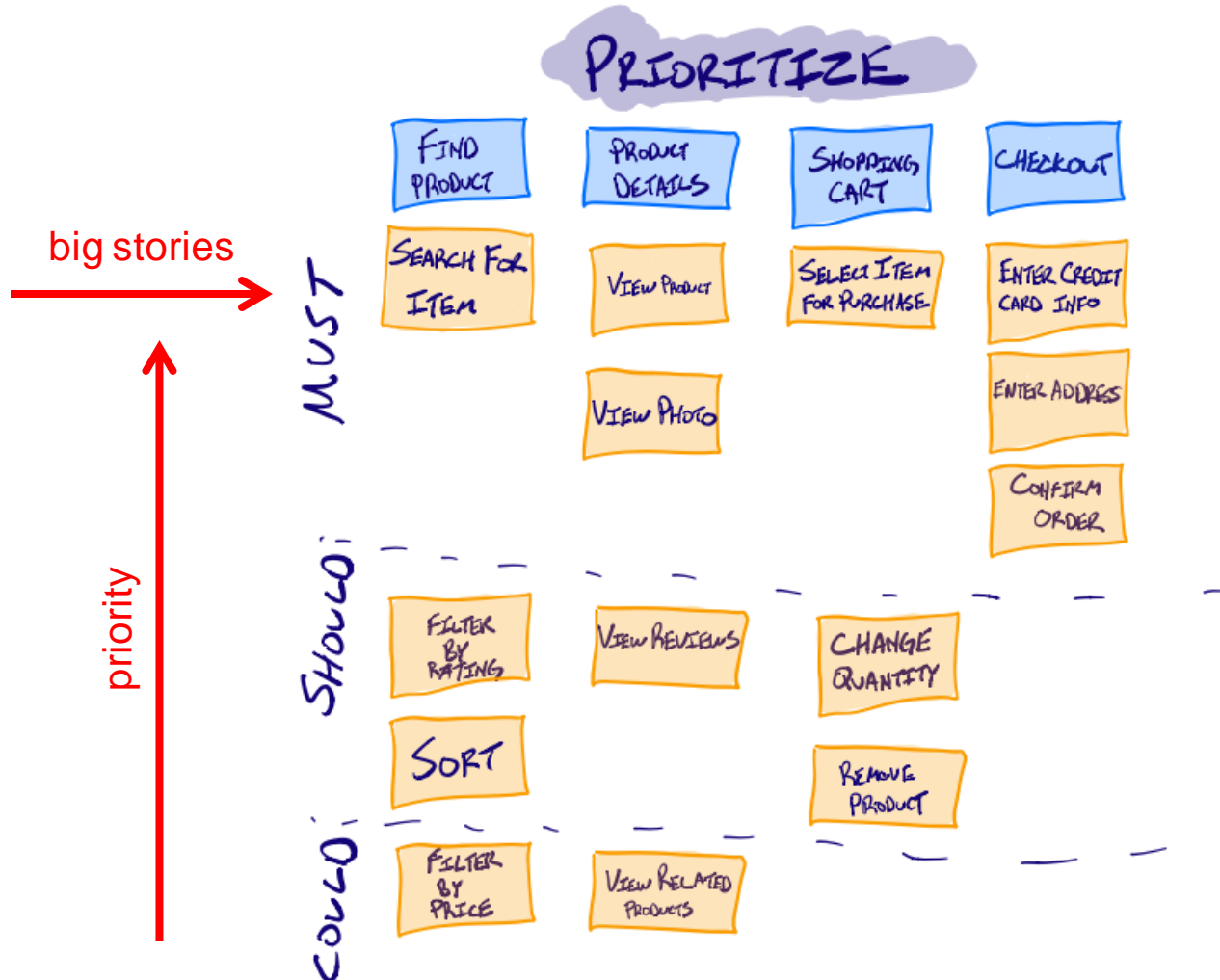
Story Mapping



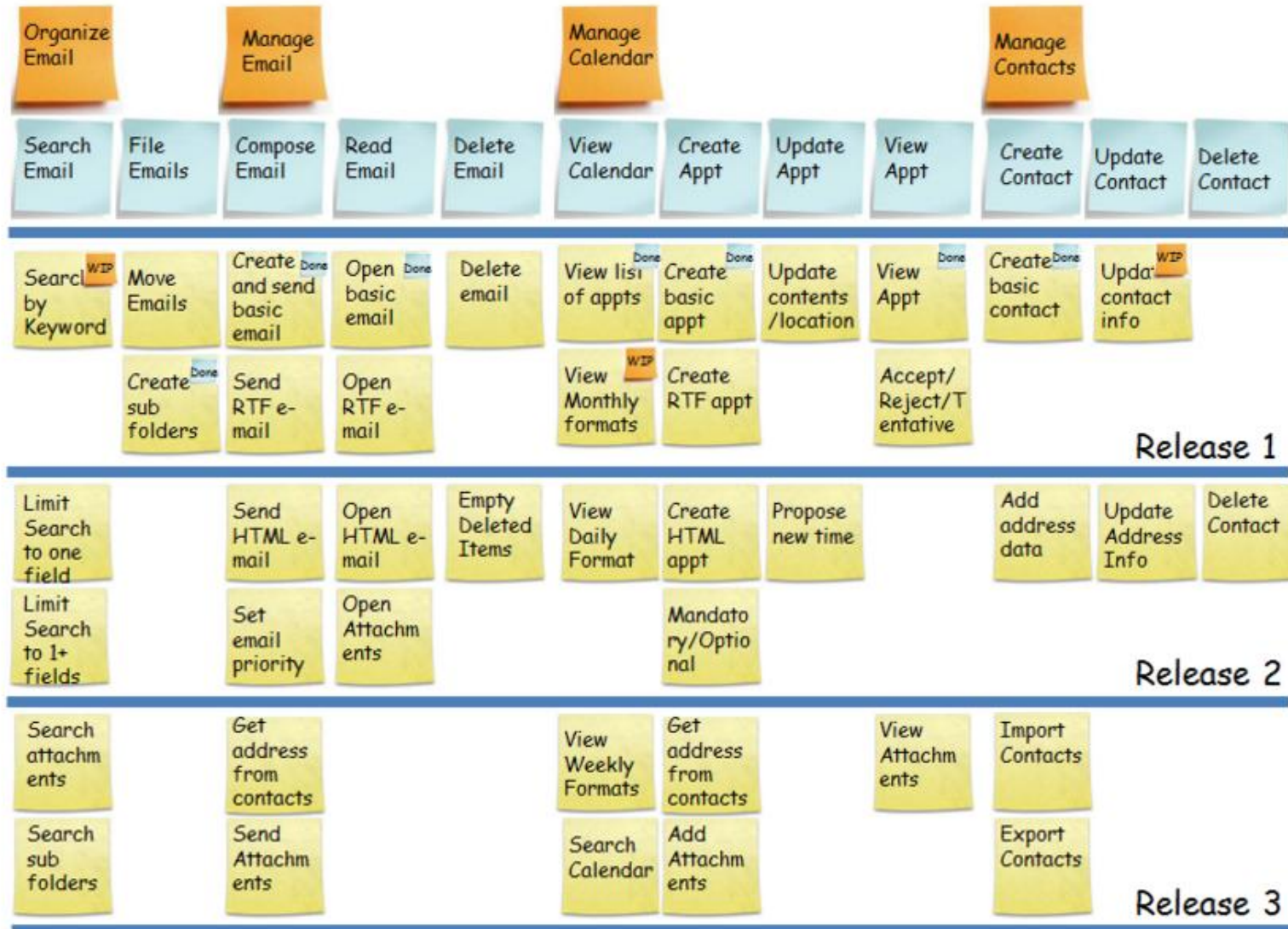
TEST FOR GAPS



Story Mapping



Story Mapping

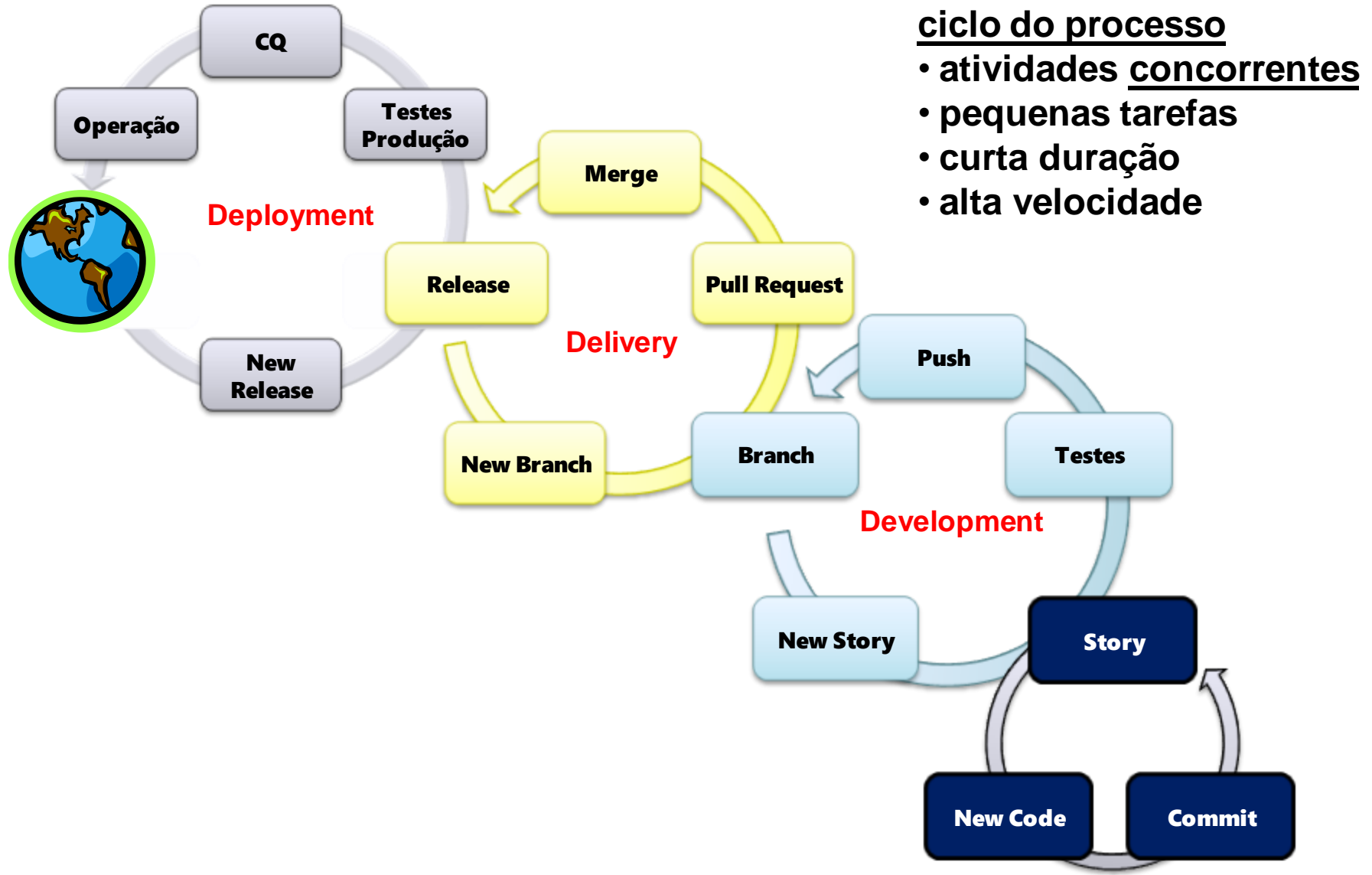


Story Mapping





Ciclo do Processo Agile



Github

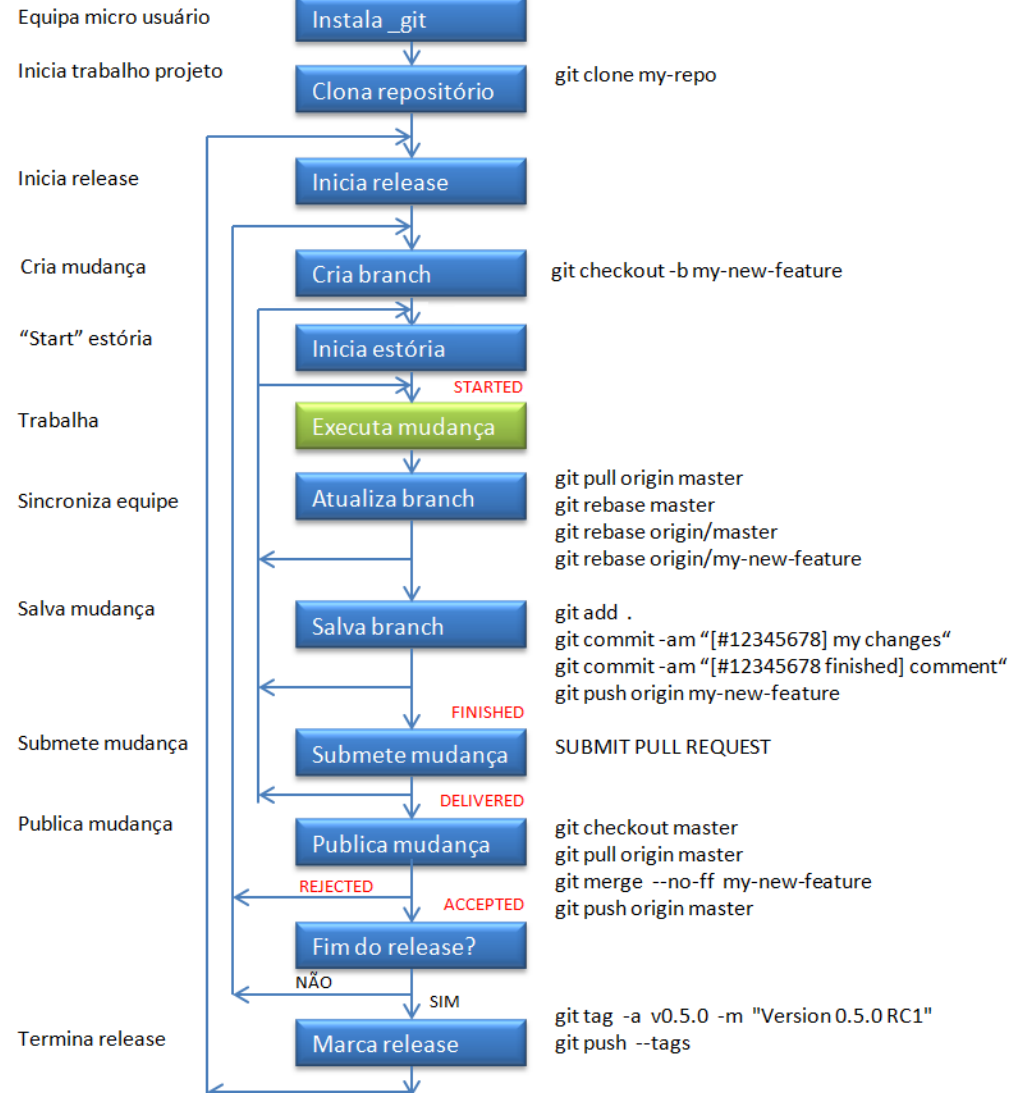


- **Serviço de hosting para repositórios Git**
- **Controle de versão distribuído e gerenciamento de código fonte**
- **Maior host de código fonte do planeta**
 - 28 Milhões de usuários
 - 85 milhões de repositórios
- **Contas gratuitas para projetos open source**
- **Completo 10 anos, foi fundada em 2008**
- **Empresa adquirida pela MS por US\$8.5Bi (Junho 2018)**

Github Flow



Workflow & Git commands



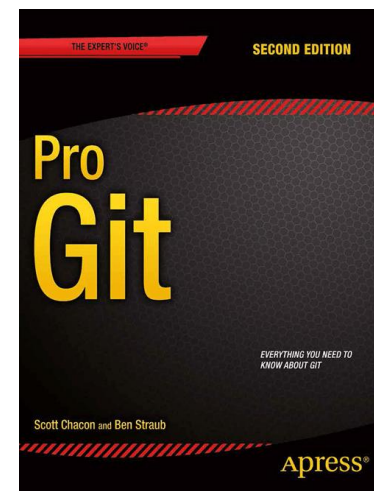
Git



- Criado por Linus Torvald em 2005 para distribuição do Linux
- Substituiu BitKeeper, software proprietário com uso livre suspenso
- Desenvolvimento Distribuído
- Compatibilidade com protocolos existentes
- Rápido e escalável, eficiente para grandes projetos
- Autenticação criptográfica do histórico

Exercícios

- Comandos Básicos: add / commit / pull / push
- Branching: checkout / branch
- Download: <https://git-scm.com/downloads>
- Pro Git (grátis): <https://git-scm.com/book/en/v2>





Exercício

Vamos utilizar o **Github** e o **Pivotal Tracker** neste curso.
Para ter acesso de escrita aos recursos, será preciso se autenticar.
Caso ainda não tenha, favor abrir uma conta gratuita em ambos os sites.

<https://github.com/>
<https://www.pivotaltracker.com>

Repositório Github:

<https://github.com/bamplifier/mba33>

Projeto Pivotal Tracker:

<https://www.pivotaltracker.com/n/projects/2181753>

GIT (instalar programa):

<https://git-scm.com/downloads>

GUI RÁPIDO:

What is Github?

<https://guides.github.com/activities/hello-world/>

Writing Stories

<https://vimeo.com/118871271>



Software Proprietário

- **Nos anos 80, quase todos os softwares eram proprietários**
- **Empresas produziam e usavam ferramentas internas de software**
- **Produtos de software eram licenciados para os clientes**
- **Softwares de código aberto eram pouco confiáveis**
- **Hoje em dia, todos usam código *open source*:**
 - **Empresas Fortune 500**
 - **Governos**
 - **Empresas de Software**
 - **Startups**
 - **Desenvolvedores individuais**

Open Source Software



- **Acelera a inovação e a produtividade do desenvolvedor**
- **Reduz o tempo de lançamento no mercado**
- **Reduz os custos de desenvolvimento**

Pesquisa anônima com 1.100 aplicativos proprietários:

- **96% contem componentes de código aberto**
- **257 componentes por aplicativo em média**

Proporção open source no código de aplicativos proprietários:

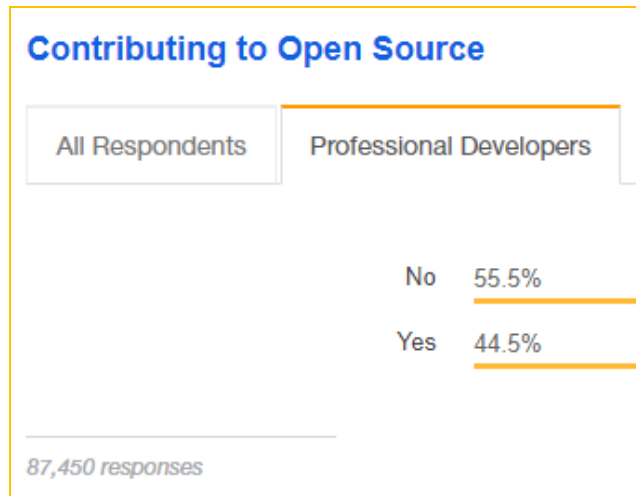
- **Cresceu de 36% em 2016 para 57% em 2017.**

Grande número de aplicativos tem agora maioria de código aberto.



Open Source Software

- 45% dos desenvolvedores profissionais contribuem
- 40% listam a contribuição em seu conhecimento informal
- 72% de usuários procuram *open source* ao avaliar ferramentas



FOSS Ecosystem



- **Free & Open Source Software.**
- **Iniciativa suportada por gigantes da indústria.**
- **Participantes de peso como Google, Facebook e Microsoft.**
- **Empresas contribuem com a comunidade *open source*.**
- **São abertos ao público grandes trechos de código fonte.**
- **Isso atrai os melhores desenvolvedores para seus produtos.**
- **O código melhora e populariza continuamente!**



Componentes *open source*



- Quais as preocupações? Como gerenciar a sua utilização?
- A maior prioridade é a segurança.
- Quanto um time de desenvolvimento conhece dos componentes *open source* que utiliza em seu código?
- Em componentes proprietários, a ameaça vem de *bugs* escondidos, à espera de serem descobertos.
- Já no *open source*, a ameaça principal são vulnerabilidades já tornadas públicas.



Vulnerabilidades *open source*



- No **software proprietário**, atualizações são enviadas aos usuários.
- No **software aberto**, o usuário deve acompanhar as vulnerabilidades, correções e atualizações.

Como funciona:

- Empresas e analistas de segurança investem tempo e habilidade analisando e procurando vulnerabilidades.
- Ao descobrir algo, procuram o dono do projeto (60-90 dias).
- DBs descentralizados registram os *bugs*, incluindo a remediação.
- Hackers acessam DBs e exploram vítimas ainda sem atualização.
- É difícil o rastreo manual no bazar *open source*.
- O tempo é crítico na análise das vulnerabilidades e suas correções.

Solução:

- Novas ferramentas pesquisam, detectam e fixam vulnerabilidades.



Melhores práticas



- Entender *frameworks & libraries*
- Atualizar-se sobre anúncios de segurança que afetem componentes e suas versões.
- Estabeleça um processo que atualize rapidamente seu produto, sempre que os componentes *open source* precisarem atualização.
- Melhor pensar em **horas ou alguns dias**, não semanas e meses.
- A maioria das brechas exploradas pelos hackers são falhas já reportadas que demoram meses ou anos para serem atualizadas.
- Todo software complexo tem falhas. Sua política de segurança deve considerar a eventual falha de seus componentes.
- Camadas de segurança em módulos de acesso público evitam que brechas na camada de apresentação capacitem acessos indevidos.
- Estabeleça a monitoração de padrões não usuais de acesso.

