

# Programação e Desenvolvimento de Software 2

## Versionamento e Revisão de Código

---

Prof. Julio Cesar S. Reis  
[julio.reis@dcc.ufmg.br](mailto:julio.reis@dcc.ufmg.br)

# Versionamento de Código

---

# Gerência de Configuração de Software

- Durante o processo de desenvolvimento de software, nós queremos saber:

- O que mudou?

- Quando mudou?

- Por que mudou?

- Quem fez essa mudança?

- Podemos reproduzir essa mudança?

- Podemos recuperar o estado anterior à mudança?

# Gerência de Configuração de Software

- Identificação

- Documentação

- Controle

- Manutenção

- Auditoria

- Artefatos:

- Código Fonte

- Documentação do Sistema

- Manual do Usuário

# Gerência de Configuração

## ■ Problema Exemplo:

- Você precisa editar um código que está no seu Dropbox
- Você faz o download do arquivo
- Faz as alterações necessárias
- Salva novamente o arquivo no dropbox

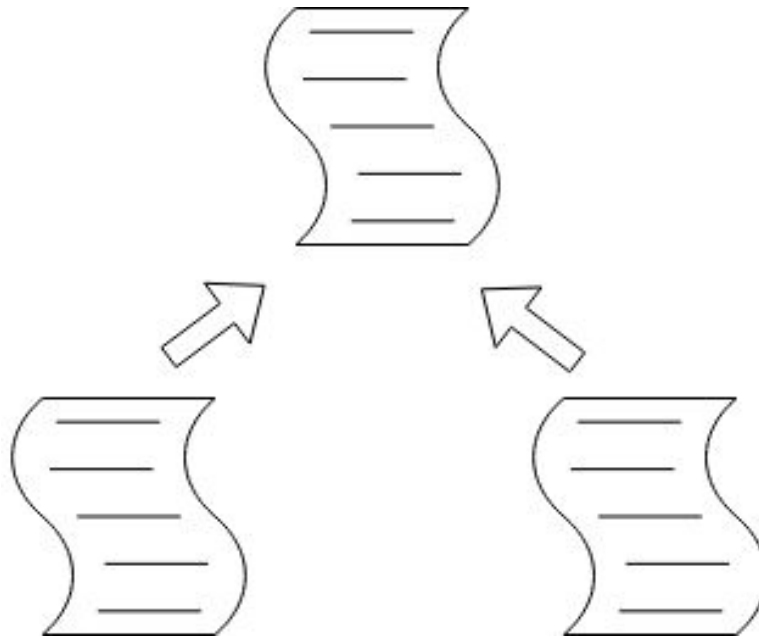
# Gerência de Configuração

## ■ Problema Exemplo:

- Agora seu colega de turma também quer editar o mesmo código
- Vocês baixam o arquivo
- Você edita e salva
- Seu colega edita e salva, sobrescrevendo seu código

# O Versionamento do Código Resolve

- O controle de versão do código realiza o ‘merge’ das alterações



# Versionamento de Código

- Controle de versão é uma sistema que mantém um registro das modificações
- Permite desenvolvimento colaborativo
- Permite saber quem fez as mudanças e quando
- **Permite reverter qualquer mudança e voltar para um estado anterior**



# Ferramentas para Versionamento de Código

- Subversion (SVN)

- Mercurial

- CVS - Concurrent Versioning System

- Bazaar

- Git

- Rápido, eficiente

# Git

- Criado em 2005 por Linus Torvalds para auxiliar no desenvolvimento do kernel do Linux
- Como vimos, ele não é o único sistema de controle de versão, mas é o mais utilizado
- [github.com](https://github.com)
  - Serviço para armazenar repositório

# Conceitos Básicos: Snapshot

- A forma que o git mantém o registro do histórico do seu código
- Registra como todos os seus arquivos são em um dado ponto no tempo
- Você decide quando fazer um snapshot, e de quais arquivos
- Poder voltar para visitar qualquer snapshot

# Conceitos Básicos: Commit

- O ato de criar um snapshot
- Um projeto é essencialmente feito de vários commits
- Um commit contém três informações:
  - Informação de como o arquivo mudou comparado com anteriormente
  - Uma referência ao commit que veio antes
  - Um código hash

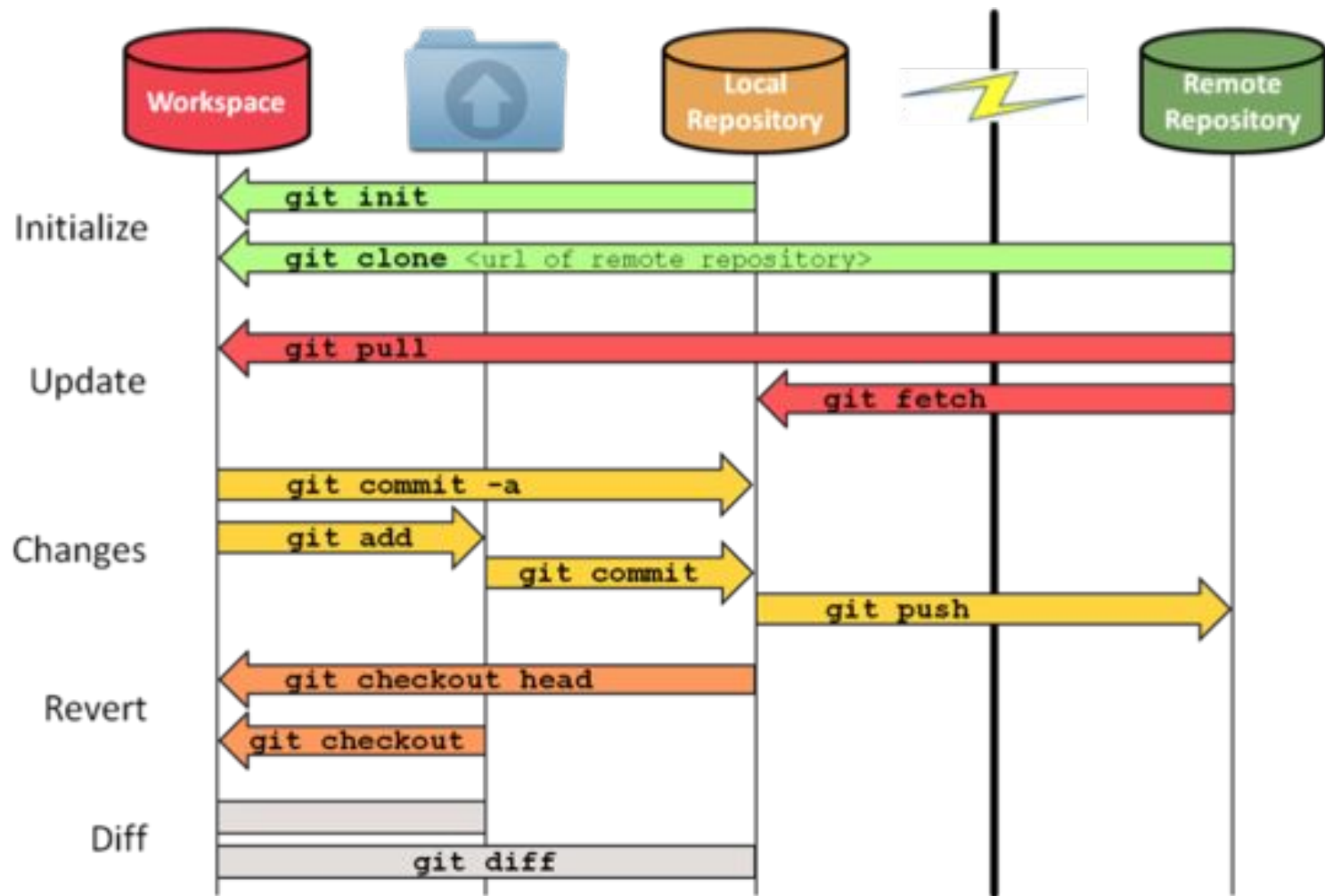
# Conceitos Básicos: Repositório

- Frequentemente resumido para repo
- Uma coleção de arquivos e o histórico dos mesmos
  - Consiste de todos os seus commits
- Pode existir na máquina local ou em um servidor remoto (github)
- O ato de copiar um repositório de um servidor remoto é chamado cloning (clone)

# Conceitos Básicos: Repositório

- O ato de fazer o download de commits que não existem na sua máquina é chamado de pulling (pull)
- O processo de adicionar as suas mudanças locais no repositório remoto é chamado de pushing (push)

# Exemplo Simples



# Começando a usar

- Instalando o Git (<https://git-scm.com>)
  - `sudo apt-get install git`
- Windows:
  - <https://git-scm.com/download/win>
- Escolhendo sua interface gráfica
  - <https://git-scm.com/downloads/guis>



# Criação de conta no GitHub

- Acessar: <https://github.com/>
- Crie sua conta
- Se você é estudante não precisa pagar
  - <https://education.github.com/pack>
- Lembre-se que com esta conta você poderá contribuir com milhões de projetos open source

**GitHub**



# Configurações Iniciais

- Conferindo sua versão
  - `$ git --version`
- Usuário
  - `$ git config --global user.name`  
“Julio Reis”
  - `$ git config --global user.email`  
“julio.reis@dcc.ufmg.br”
  - `$ git config --list`

# Criando um Repositório

- Iniciando um repositório

- `$ cd project`

- `$ git init`

- Status

- `$ git status`

- Adicionamento arquivos para versionamento

- `$ git add <file> ou git add . (para todos)`

- Reset

- `$ git reset <file> ou git reset .`

# Operações

- Commit (consolidando/validando alterações feitas)

- `$ git commit -m "detailed message"`

- Log

- `$ git log`

- Show

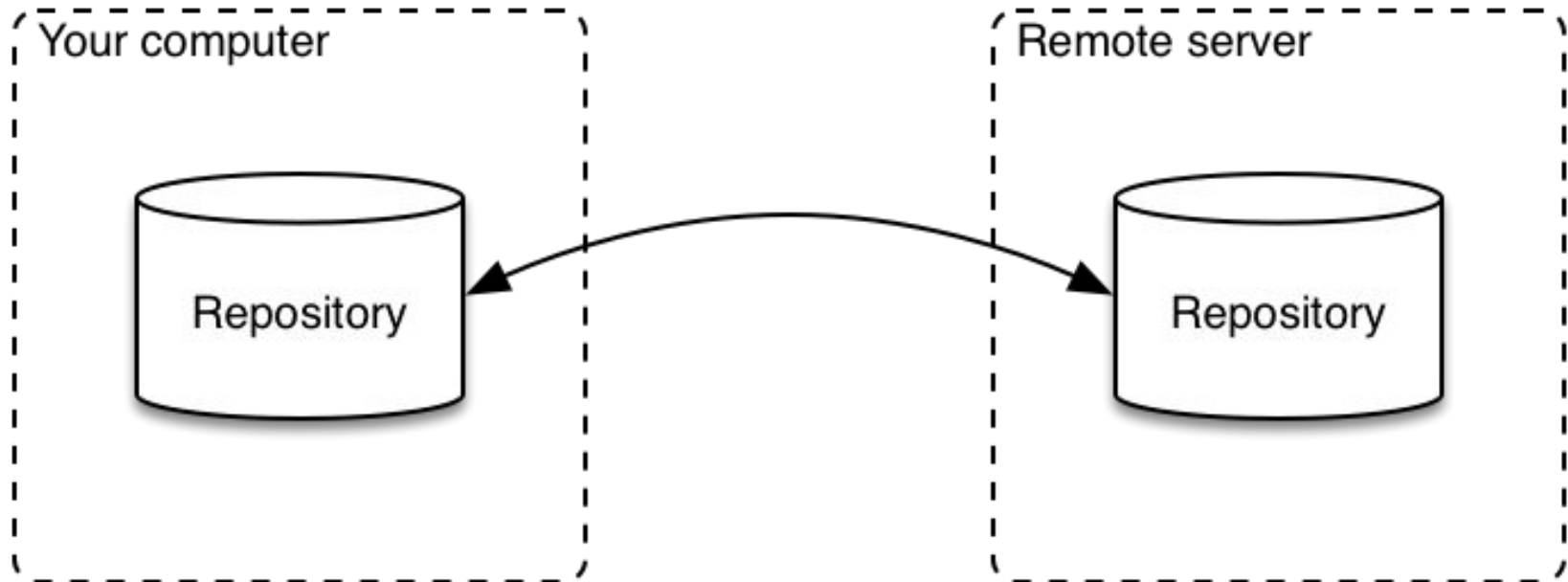
- `$ git show` ou `$ git show <hash_id>`

- Diff

- Working: `$ git diff` ou `git diff <file>`

- Staging: `$ git diff --cached` ou `$ git diff --cached <file>`

# Remote



# Clone/Obtenção de Repositório

- `$ git clone [url]`

- `$ git clone git@github.com:<user>/<project>.git`

ou

- `$ git clone https://github.com/<user>/<project>.git  
<folder>`

- Verificando o status do seus arquivos:

- `$ git status`

- Monitorando novos arquivos:

- `$ git touch new_file`

- `$ git add new_file`

# Outras Operações

- Ignorando arquivos

- `$ touch .gitignore`

- Log

- `$ git log`

- Show

- `$ git show` ou `$ git show <hash_id>`

- Diff

- Working: `$ git diff` ou `git diff <file>`

- Staging: `$ git diff --cached` ou `$ git diff --cached <file>`

# Outras Operações

- Removendo arquivos

- `$ git rm <my_file>`

- Movendo arquivos

- `$ git mv <my_file>`

- Desfazendo operações:

- Modificando o último commit

- `$ git commit -amend`

- Desfazendo arquivo modificado

- `$ git checkout <my_file>`

- Removendo arquivo da área de validação

- `$ git reset HEAD <my_file>`

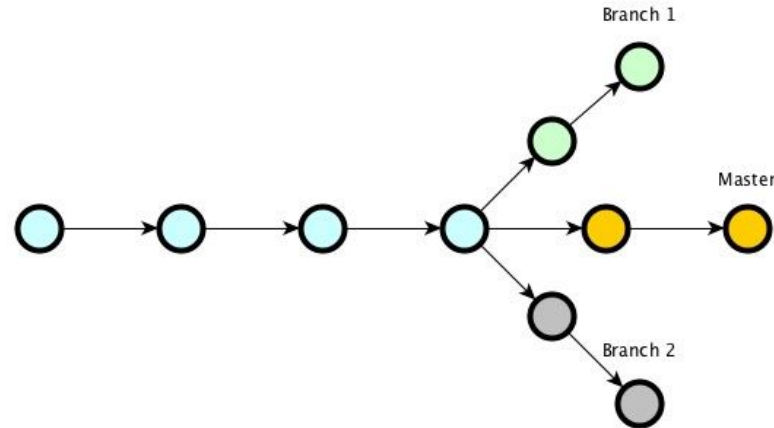


# Trabalhando com Remoto

- Criando repo a partir de um remoto
  - `$ git clone <url>`
- Enviando alterações para o seu remoto
  - `$ git push origin master`
- Recebendo alterações do seu remoto
  - `$ git pull origin master`

# Branching

- # Uma feature = Uma branch

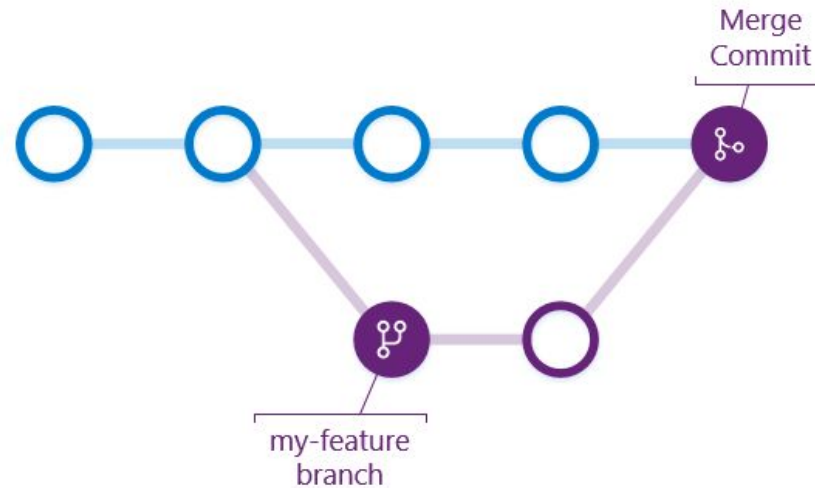


- ```
$ git branch my-feature
```

- ```
$ git checkout my-feature
```

- # \$ git commit (x2)

# Branching - Merge



- `$ git checkout master`
- `$ git diff master..my-feature`
- `$ git merge my-feature`
- Clean-Up:
  - `$ git branch -d my-feature`

# Remote branch

## Local

- \$ git branch my-feature

- \$ git branch -d my-feature

## Remote

- \$ git branch -a

- \$ git push origin my-feature

- \$ git push origin --delete my-feature

# Tagging

- O Git nos permite criar tags em pontos específicos do meu código (i.e. pontos importantes)

- Listar tags de um projeto

- `$ git tag`

- Cria tags anotadas

- `$ git tag -a v2.5 -m 'my_version2.5'`

- Exibe detalhes de uma tag

- `$ git tag v2.5`

- Envia tags para o repositório remoto

- `$ git push origin <tag_name>`

# Git - Hooks

- Maneira de disparar scripts personalizados quando certas ações importantes acontecem

- .git/hooks

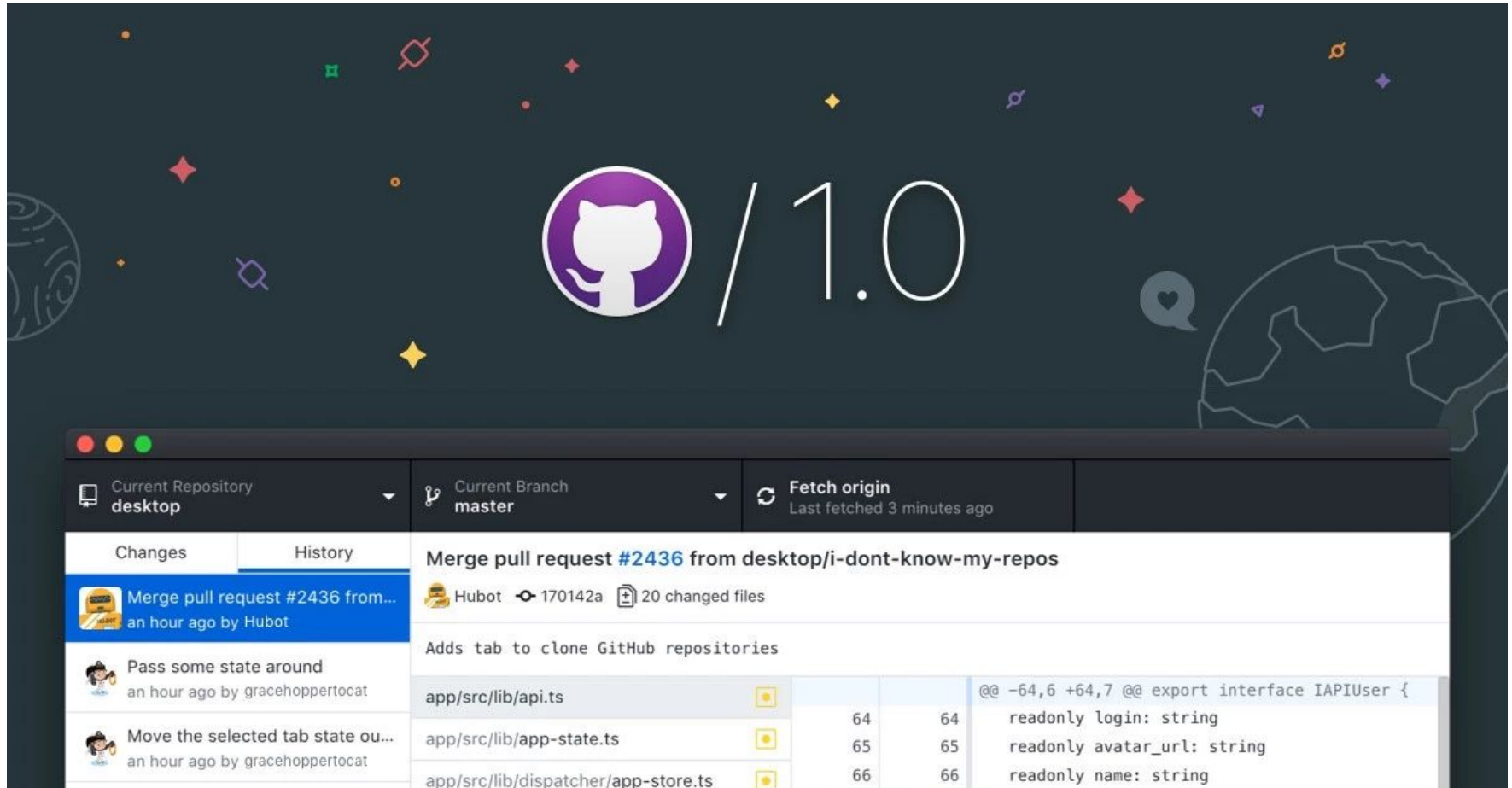
- pre-commit

- git commit no-verify

- prepare-commit-msg

# GitHub Desktop

desktop.github.com



# Revisão de Código

---



# Revisão de Código

- Sistemas grandes, complexos, é possível garantir...
  - Código legível? Sem duplicidade?
  - Facilidade de manutenção?
  - Ausência de erros?
- Taxa média de detecção de defeitos
  - Testes unitários: 25%
  - Testes de integração: 45%
- Conseguimos melhorar esses valores?

# Revisão de Código

- Técnicas para “garantir” (e/ou melhorar) a qualidade do software desenvolvido
- Verificação
  - Software de acordo com a especificação
  - “Construímos o produto corretamente?”
- Validação
  - Software faz o que o usuário realmente deseja
  - “Construímos o produto certo (esperado)?”

# Revisão de Código

- Tarefa construtiva de rever o código e a documentação para identificar erros de interpretação, incoerências e outras falhas
  - Confirmação externa (antes de alterações e inserções de novos códigos)
- Propósito:
  - Melhorar o código
  - Melhorar o programador

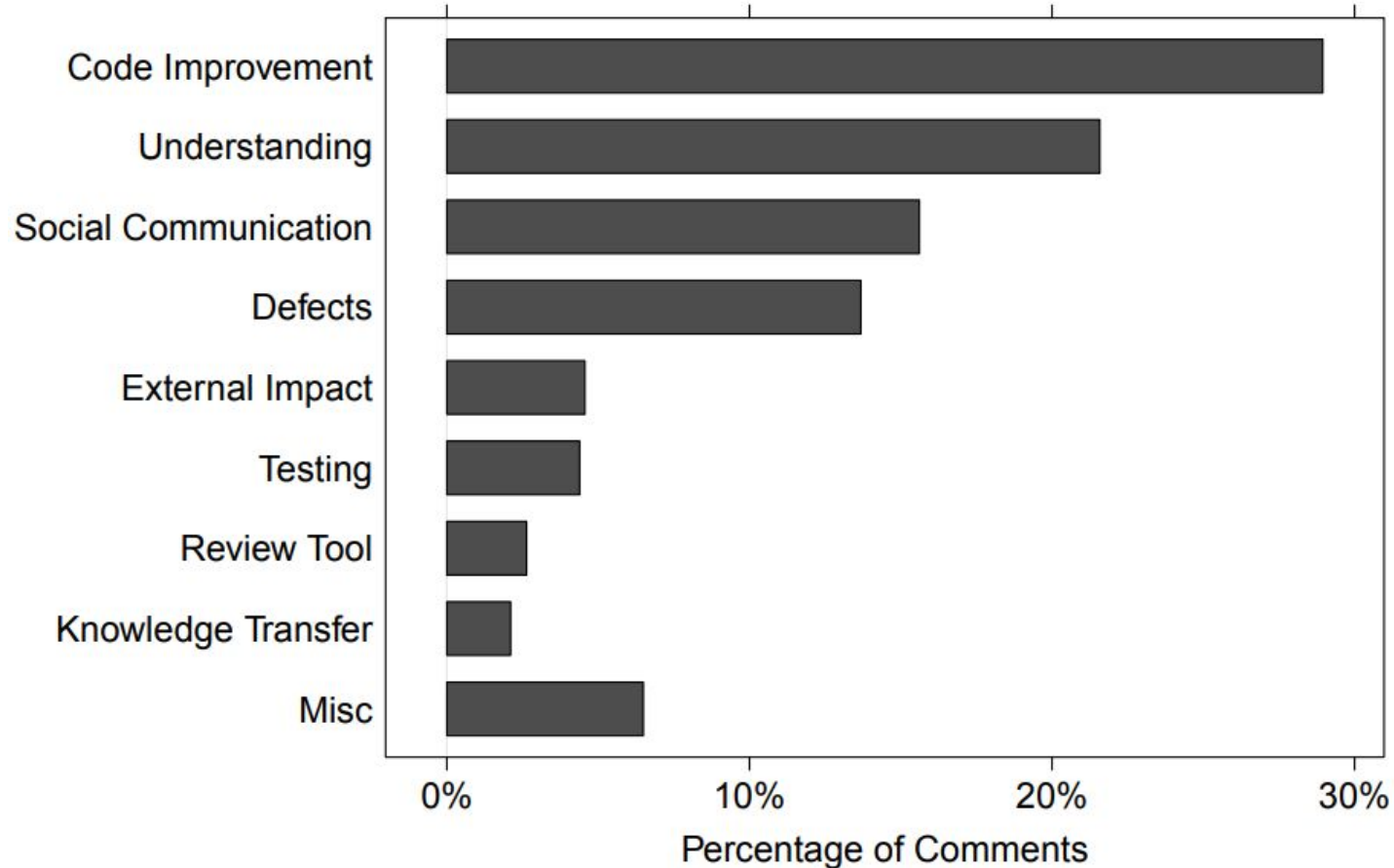
# Revisão de Código

## Benefícios

- Taxa média de detecção de defeitos
  - Inspeções de design e código: 55% - 60%
- II programas desenvolvidos (mesma equipe):
  - 5 (sem revisões): 4,50 erros a cada 100 LoC
  - 6 (com revisões): 0,82 erros a cada 100 LoC
- Conhecimento:
  - Melhor entendimento do código
  - Feedback/Programadores Iniciantes

FREEDMAN. D. P., and WEINBERG G. M. (1982) Software Inspections: An Effective Verification Process. IEEE Software.

# Revisão de Código



Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. ICSE.

# Revisão de Código

## Quem

- Desenvolvedor do código e o responsável pela revisão (desenvolvedor mais experiente), às vezes juntos pessoalmente, às vezes separados

## Como

- Revisor dá sugestões de melhoria em um nível lógico e/ou estrutural, de acordo com conjunto previamente acordado de padrões de qualidade
- Correções são feitas até uma eventual aprovação do

## Quando

- Após o autor de código finalizar uma alteração do sistema (não muito grande/pequena), que está pronta para ser incorporada ao restante

# Revisão de Código @ Google

Modern Code Review: A Case Study at Google (<https://ai.google/research/pubs/pub47025>)

“All code that gets submitted needs to be reviewed by at least one other person, and either the code writer or the reviewer needs to have readability in that language. Most people use Mondrian [Rietveld] to do code reviews, and obviously, we spend a good chunk of our time reviewing code.”

– Amanda Camp, Software Engineer, Google

# Tipos de Revisão

- **Email**

- Olá, olhe meu código.

- **Ferramentas**

- Gerrit
- Rietveld

- **Ciclo de pull requests**

- Github

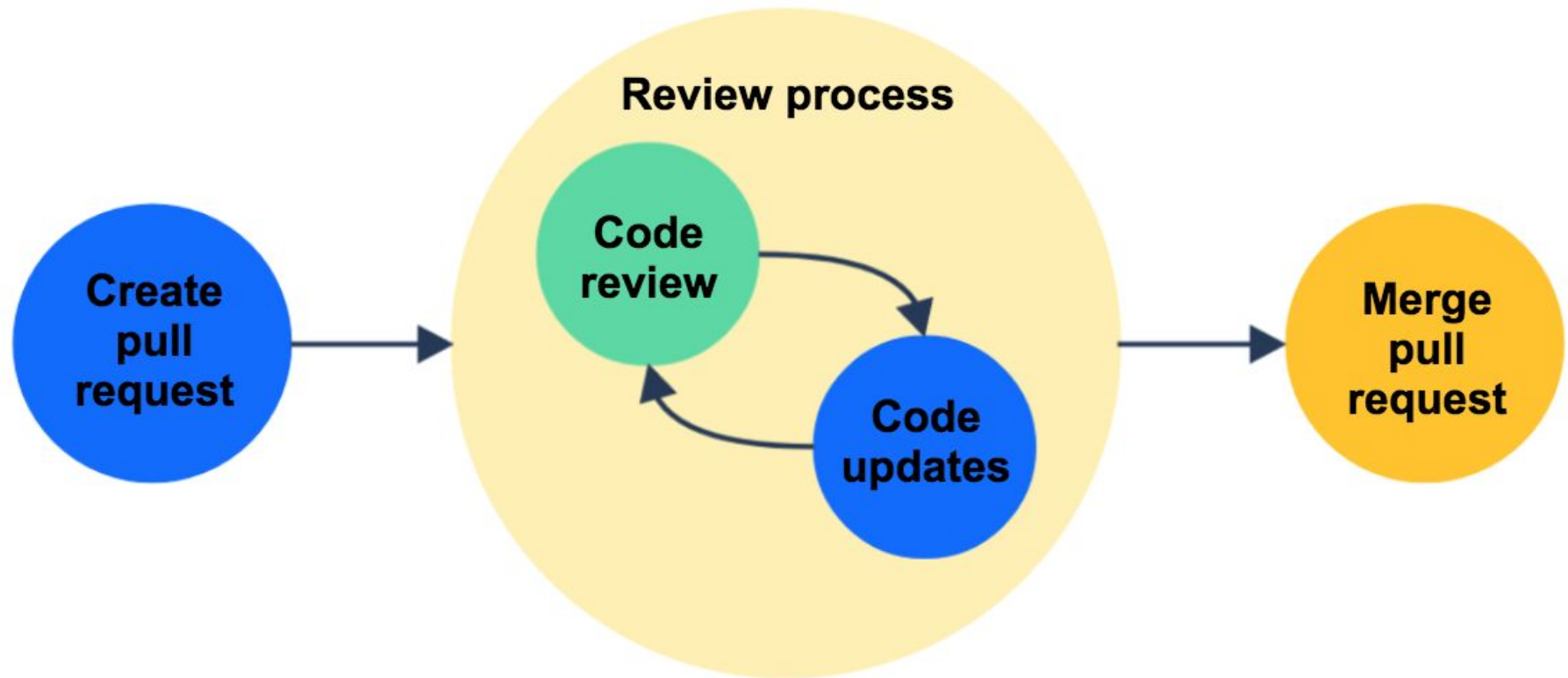


# Revisão de Código

- Geralmente todo esse processo é feito antes de código ir para o repositório
  - Push no Github
- Série de ferramentas ajudam nesta tarefa

# Revisão de Código

No Github



# Revisão de Código

## Checklist

<i>Code Review Checklist</i>	
<input checked="" type="checkbox"/>	<u>Coding standards</u>
<input type="checkbox"/>	<u>Coding Best practices</u>
<input checked="" type="checkbox"/>	<u>Non Functional Requirements</u>
<input checked="" type="checkbox"/>	<u>OOAD Principles</u>
<input checked="" type="checkbox"/>	<u>Static Code Analysis Metrics</u>
<input type="checkbox"/>	<u>.....</u>

# Revisão de código

## Boas Práticas

- Estabeleça metas quantificáveis para revisão
- Utilize listas de verificação (checklists)
- Registre e verifique se os defeitos são consertados
- Você não irá revisar tudo de uma vez!
  - Talvez será adequado priorizar partes críticas