

Engenharia de Software

Prof. Vinicius Cardoso Garcia

vcg@cin.ufpe.br :: [@vinicius3w](https://twitter.com/vinicius3w) :: viniciusgarcia.me

[IF977] Engenharia de Software

<http://bit.ly/vcg-es>

Licença do material

Este Trabalho foi licenciado com uma Licença

Creative Commons - Atribuição-NãoComercial-
Compartilhual 3.0 Não Adaptada



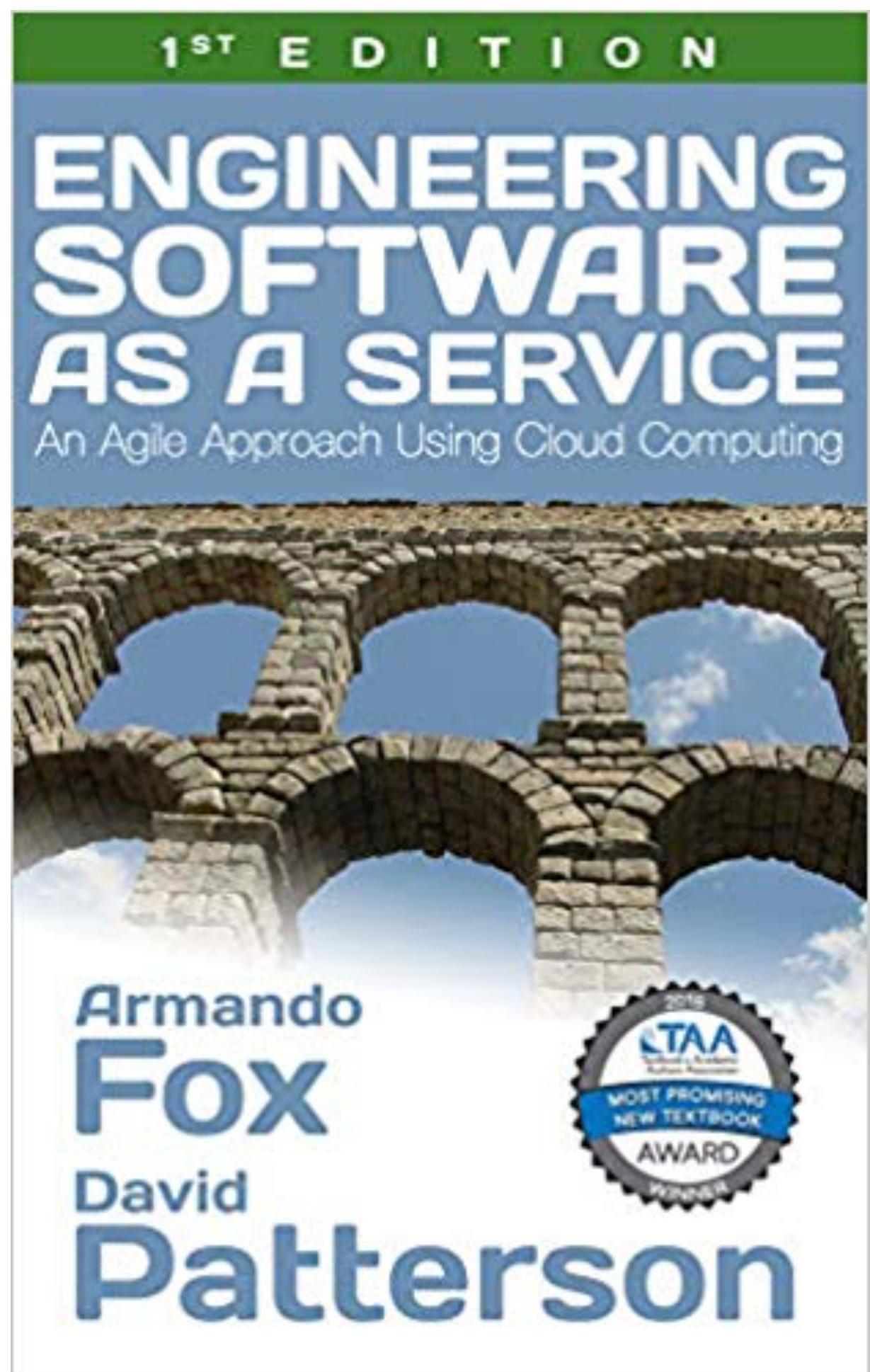
Mais informações visite

[http://creativecommons.org/licenses/by-nc-sa/
3.0/deed.pt](http://creativecommons.org/licenses/by-nc-sa/3.0/deed.pt)



Referências

- A biblioteca do Desenvolvedor de Software dos dias de hoje
 - <http://bit.ly/31WYK5f>
- SWEBOK: Guide to the Software Engineering Body of Knowledge (SWEBOK)
 - <http://www.computer.org/web/swebok>
- Engineering Software as a Service: An Agile Approach Using Cloud Computing
 - <http://www.saasbook.info/>
- Marco Tulio Valente. Engenharia de Software Moderna
 - <https://engsoftmoderna.info/>





Garantia de Qualidade

Imagen: <https://www.softwaretestinghelp.com/software-quality-assurance/>

Definição global

- Atributo, condição natural, propriedade pela qual algo ou alguém se individualiza, distinguindo-se dos demais; maneira de ser, essência, natureza.
- Excelência, virtude, talento.
- Grau de perfeição, de precisão, de conformidade a um certo padrão



Uma pergunta polêmica

- O Windows é um produto de software de alta qualidade?



Claro! Adoro aquele
clipe que corrige
meus e-mails!



Windows

Não, ele é a causa
do monopólio da
MS e seus prejuízos
à sociedade!

Mac



Posso ver o código-
fonte antes de
responder?



Linux

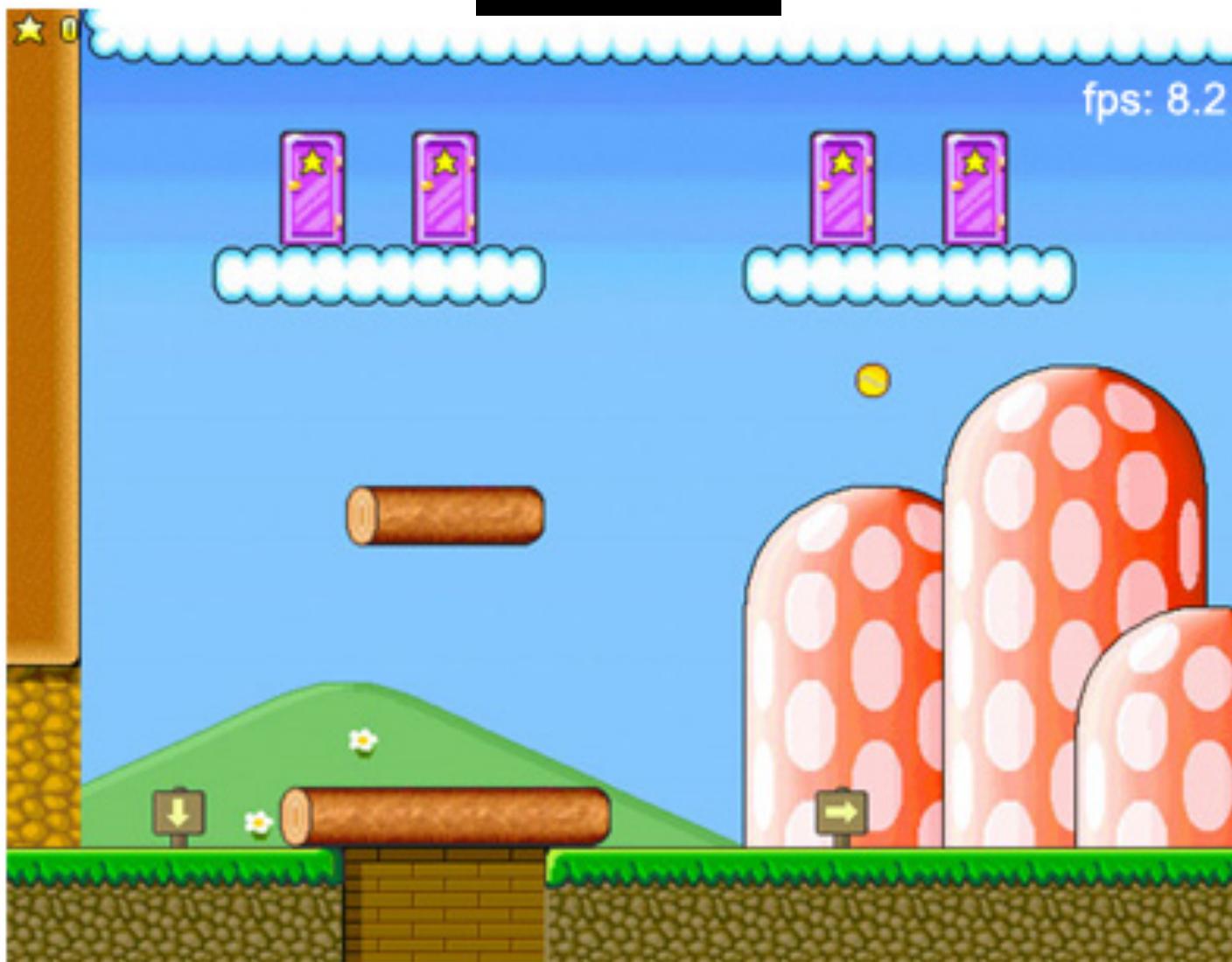
Não gosto
desse negócio
de usar
mouse...

Solaris





3999\$

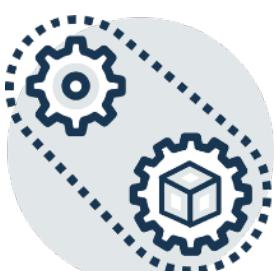


1299\$



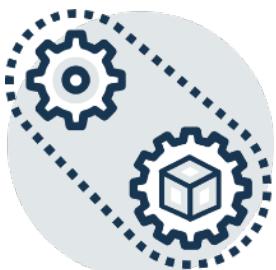
Qualidade

- Altamente subjetivo
- Muda de indivíduo para indivíduo
 - Percepções/visões/experiências distintas
- Difícil responder de maneira absoluta
- Como melhorar a qualidade?
 - Se não sabemos nem responder a esta pergunta?

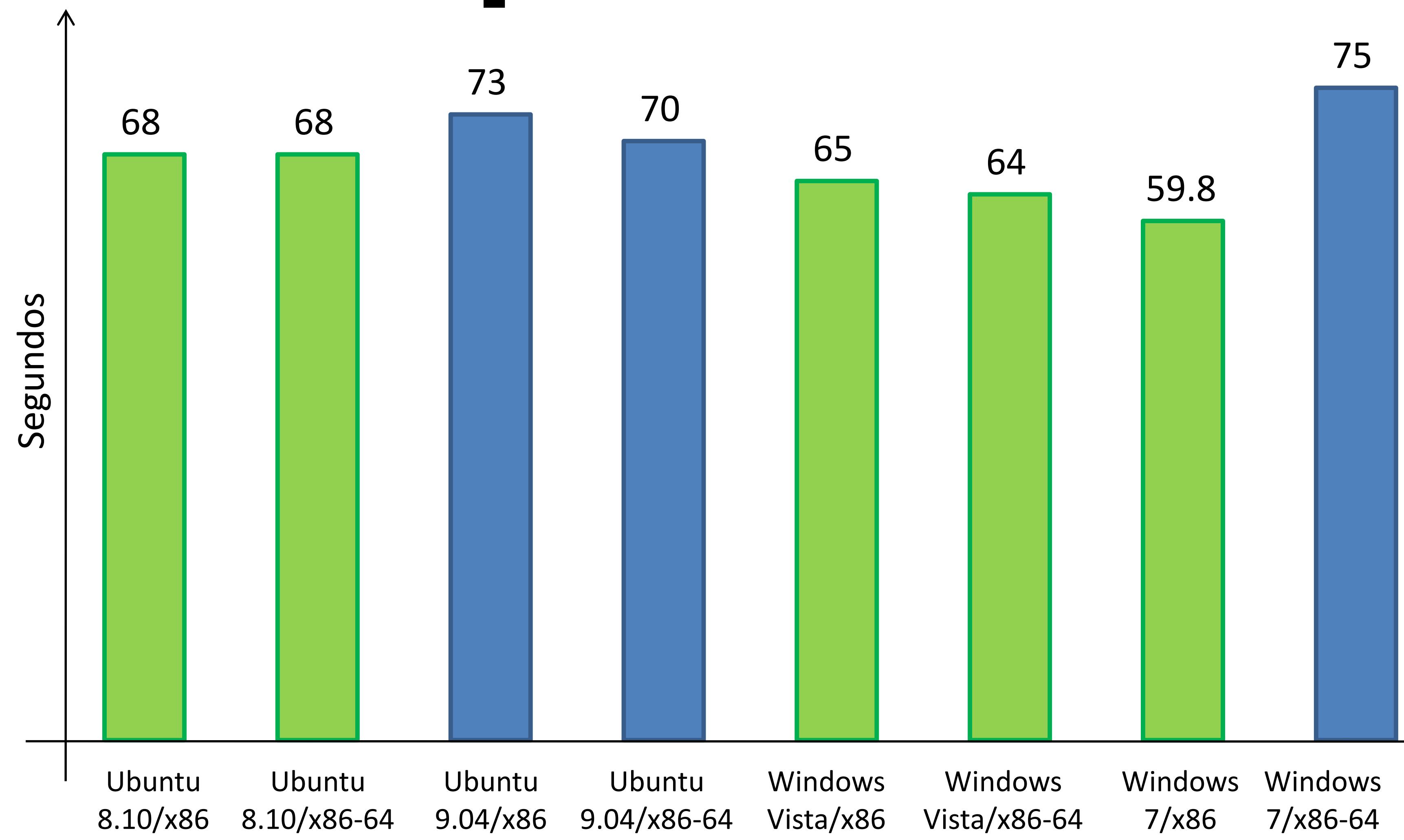


Refinando a pergunta

- O Windows 7 será adequado para meu ambiente?
 - Preciso que o tempo de inicialização seja no máximo 70 segundos
 - Estarei utilizando uma arquitetura 32 bits
 - Tenho possibilidade para instalar outros sistemas operacionais

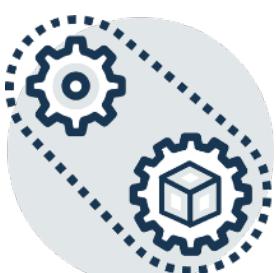


Tempo de boot



Qualidade

- A pergunta agora define um contexto
 - Permite filtrar as possibilidades de avaliação
- Define termos específicos para o que é bom e o que não é bom
 - Está falando sobre “tempo de boot”
- Define valores que eliminam subjetividade
 - Qualquer pessoa chega à mesma conclusão
- Ou seja, precisamos fazer a pergunta **correta!**



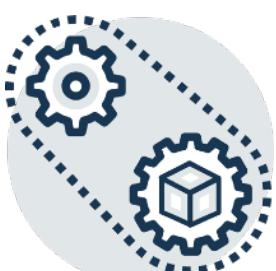
Qualidade de software

- Como **saber** se um software tem qualidade alta ou baixa?
- O que está envolvido na qualidade de um software?
 - De que maneiras um software pode apresentar qualidade?
- Como garantir que um software tem a qualidade desejada?
 - Saber x construir



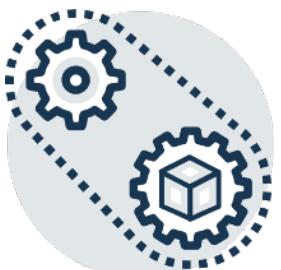
Qualidade de software

- Objetivo recorrente em qualquer organização que produz software
- Impossível uma organização dizer que seu software não tem qualidade
- Qualidade de Software: Interna e Externa
- Qualidade Externa: pode ser avaliada sem conhecimento do código fonte
 - Inclui fatores como correção, robustez, portabilidade, reusabilidade, compatibilidade, facilidade de uso, etc
- Qualidade Interna: depende de conhecimento do código para ser avaliada
 - Modularidade, legibilidade, testabilidade, changeability, comprehensibility, manutenibilidade



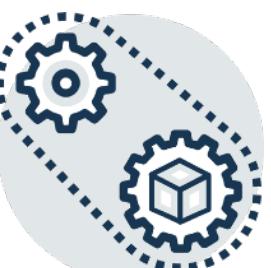
Visão geral e histórica de qualidade

- Código de Hamurabi (1700 a.C.)
 - **Responsabilidade profissional:** um arquiteto que construir uma casa que se desmorone, causando a morte de seus ocupantes, é condenado à morte!



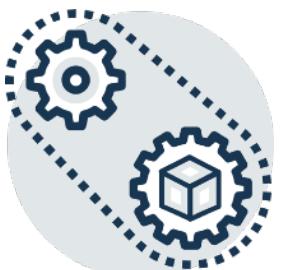
O que é Qualidade de Software?

- Qualidade, de maneira simplista, significa que um **produto deve atender às sua especificação**
- Isso é **problemático** para os sistemas de software
 - **Tensão** entre os requisitos de qualidade do cliente (eficiência, confiabilidade, etc.) e requisitos de qualidade do desenvolvedor (facilidade de manutenção, reusabilidade, etc.)
 - Alguns requisitos de qualidade **são difíceis de especificar de uma maneira não-ambígua**
 - Ex. Facilidade de uso
 - As especificações de software são, geralmente, incompletas e freqüentemente inconsistentes

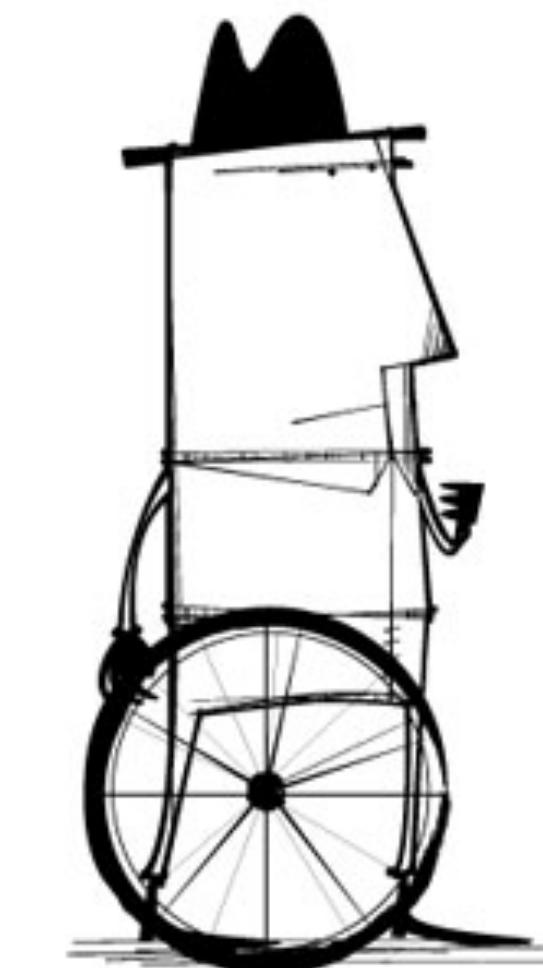


Qualidade de Software x Tipos ABC de Sistemas

- Classificação proposta por Bertrand Meyer:
 - Sistemas C (Casuais): qualidade não é tão importante
 - Sistemas B (Business)
 - Sistemas A (Acute): requerem procedimentos específicos para garantir qualidade, incluindo certificação por órgãos externos
- Nosso foco: qualidade interna de sistemas do tipo B



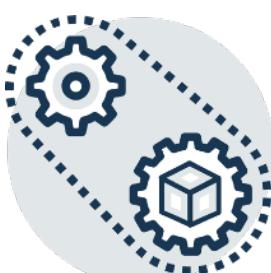
ERRR...



**CAN'T STOP.
TOO BUSY!!**



Débito Técnico



Débito Técnico

- Gerentes e executivos às vezes não valorizam aspectos de qualidade interna de software, pois eles são "invisíveis" para os clientes finais
 - Não envolvem a correção de bugs ou a implementação de features
- Débito Técnico (Technical Debt):
 - Analogia para explicar para "leigos" a importância de qualidade interna
 - Baseada em termos da área financeira (débito, principal, juros, etc)
 - Soluções não-ótimas de design e implementação que permitem lançar uma feature de modo mais rápido, mas que no futuro tornam a implementação de novas features mais custosas



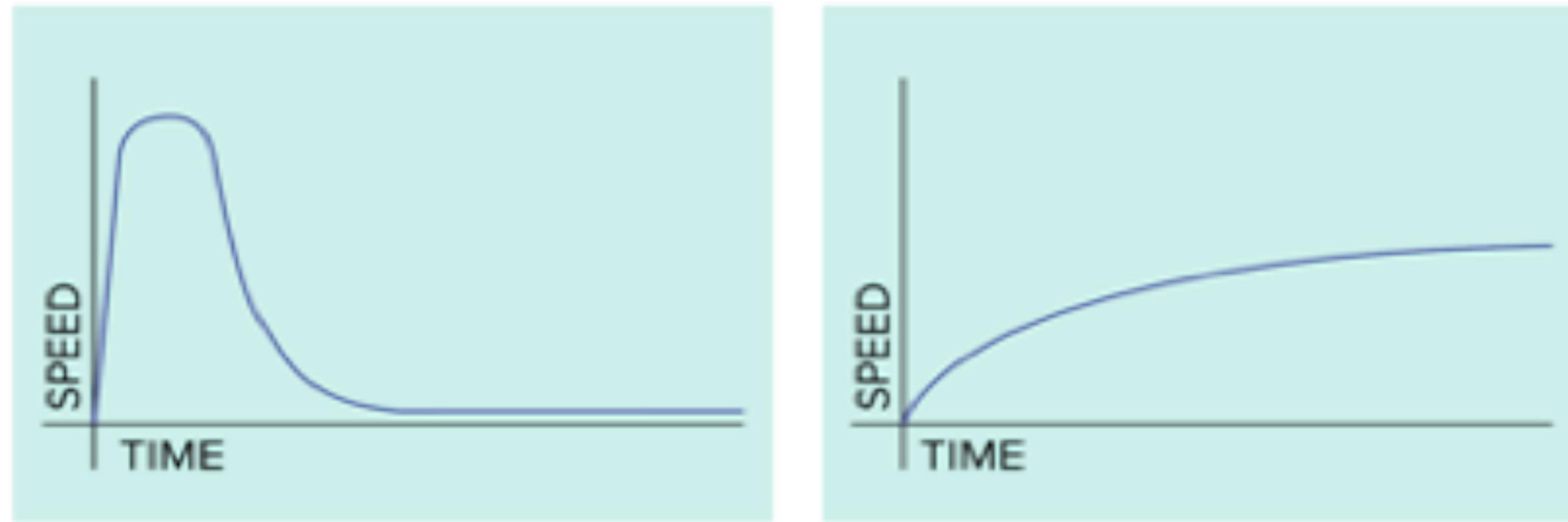
Primeiro Artigo sobre Débito Técnico

- Ward Cunningham. [The WyCash Portfolio Management System](#) (Experience Report). OOPSLA 1992.
 - Although immature code **may work fine** and be **completely acceptable**... excess quantities will make a program **unmasterable**...
 - Leading to... an **inflexible** product.
 - Shipping first time code is like going into debt.
 - A little debt speeds development so long as it is paid back promptly
 - The danger occurs when the debt is not repaid
 - Every minute on not-quite-right code counts as interest on that debt.

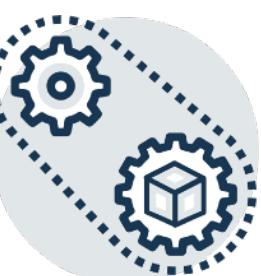


Consequências do Acúmulo de Débito Técnico

- 1o gráfico: projeto onde débito técnico se acumulou ao longo dos anos; implementação de novas features é "quase impossível". Situação mais frequente do que a gente imagina
- 2o gráfico: projeto onde o débito técnico foi sendo pago ao longo dos anos; velocidade do time foi preservada mesmo com o envelhecimento do software

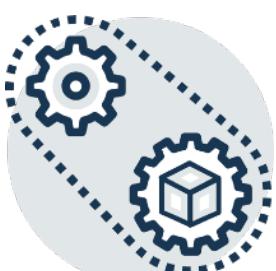


<https://builttoadapt.io/why-tdd-489fdcd05e>



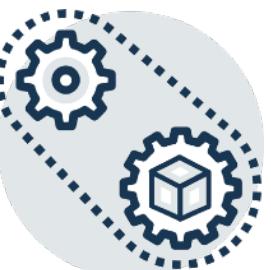
Exemplos de Débito Técnico

- Sistemas sem testes
- Sistemas onde o build não é automatizado
- Sistemas com problemas arquiteturais. Por exemplo, não existe uma separação clara entre camadas (interface, regras de negócio, persistência)
- Sistemas lotados de code smells (large class, large method, feature envy etc)
- Sistemas com métodos complexos, pouco coesos e com alto acoplamento
- Sistemas sem nenhuma documentação
- Sistemas cujo código não segue uma convenção de estilo e layout
- Resumindo: **tudo aquilo que atrasa a implementação de novas features**



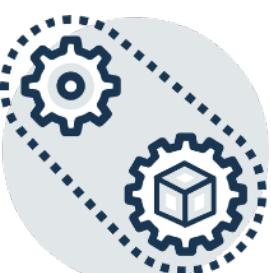
Definição Ampliada

- Como todo termo que faz sucesso, sua definição tende a ser ampliada, para cenários não relacionados com a definição original
- Definição original: TD agiliza a implementação de uma feature, mas atrasa a implementação de novas features; logo, relaciona-se com manutenibilidade
- Definição ampliada inclui:
 - Problemas de performance
 - Vulnerabilidades de segurança
 - Problemas na interface com o usuário
 - etc



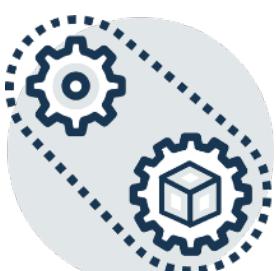
Exemplos Reais

- Leia a seguinte thread do **Hacker News**:
 - What's the **largest amount of bad code** you have ever seen work?
 - <https://news.ycombinator.com/item?id=18442637>
- Vejam esse comentário:
 - You can't change a line of code in the product without breaking 1000s tests.
 - Generations of programmers have worked on that code under difficult deadlines
 - They filled the code with all kinds of crap.
 - Very complex pieces of logic ... all held together with thousands of flags.
 - Sometimes one needs to understand the values and the effects of 20 different flag to predict how the code would behave in different situations. Sometimes 100s too!



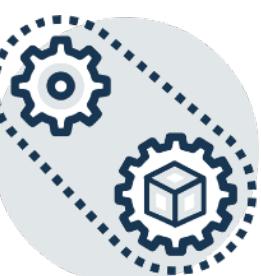
Exemplo Hipotético

- Suponha que uma feature F1 foi implementada de forma "apressada"
- A implementação "ideal" demandaria mais 2 dias de trabalho
- Logo, débito técnico = 2 dias
- Suponha que uma nova feature F2 deve ser implementada, meses depois
- F2 é relacionada com F1; ambas são implementadas no mesmo módulo
- Implementação de F2 levou 5 dias
- Mas se não houvesse débito técnico relativo a F1, poderia ter levado 4 dias
- Logo, o principal do débito técnico de F1 = 2 dias
- E os juros relativos ao débito técnico de F1 = 1 dia (tempo a mais gasto em novas features)

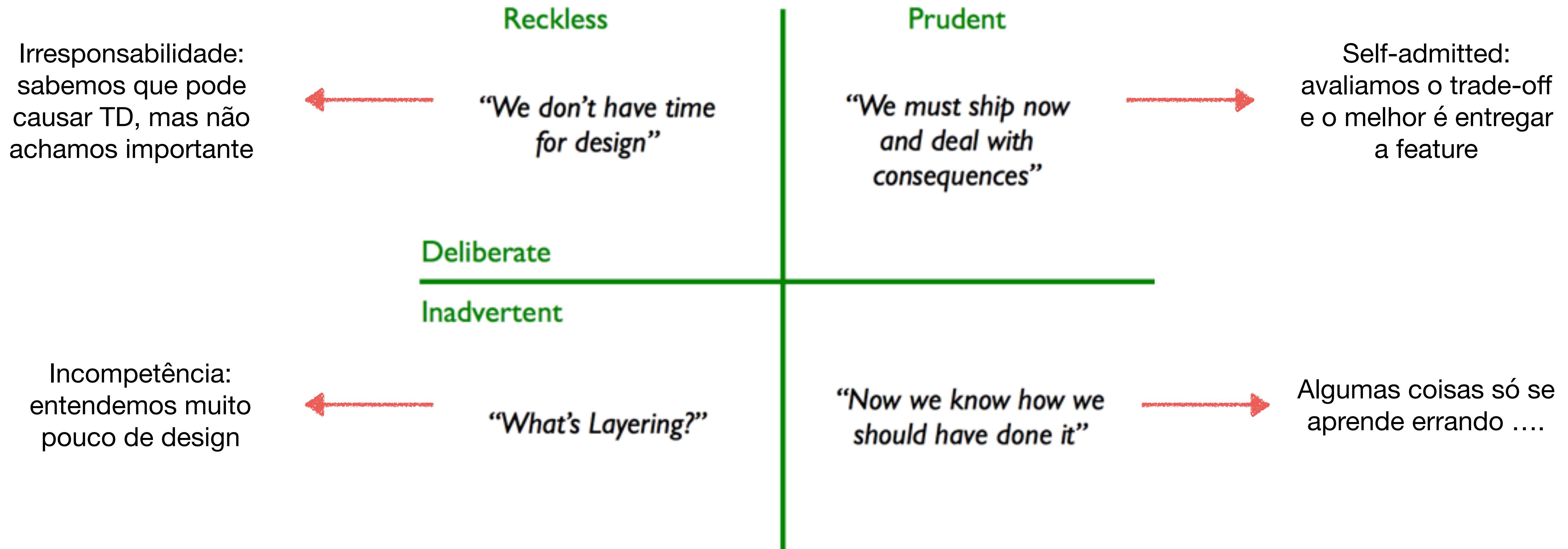


Principal Desafio: Como quantificar TD?

- Débito técnico é um **problema real**, que afeta todo sistema, em algum grau
- Porém, é difícil quantificar seu principal:
 - Quanto tempo **a mais** eu precisaria para implementar essa feature de **modo ideal**? Isto é, sem introduzir um débito técnico ...
- E mais difícil ainda quantificar os juros:
 - Quanto tempo **a mais** eu vou levar para implementar essa nova feature, devido ao débito técnico **acumulado** no sistema?
- Porém, existe uma recomendação: **pagar primeiro o débito técnico de partes do sistema que são sempre modificadas**
- Débito técnico em partes muito **estáveis** do código **não é uma preocupação**



Quadrante de Débito Técnico



<https://martinfowler.com/bliki/TechnicalDebtQuadrant.html>

Vídeo Complementar

- Neste vídeo, de 5 minutos, Ward Cunningham descreve o conceito de TD e também sua motivação para propor essa metáfora

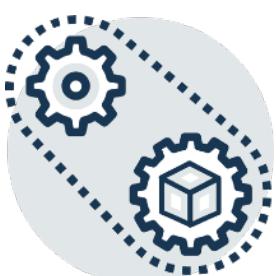


We should stop pretending that

- Distributed systems are easy
- People can be available 24/7
- Unicorn companies don't have technical debt
- That new product will solve all of your problems

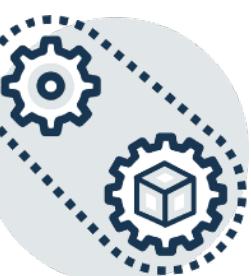


Sasha Rosenbaum - [Admitting that hard problems are hard](#) @ [Devopsdays Ghent](#)



Descarte de Sistemas

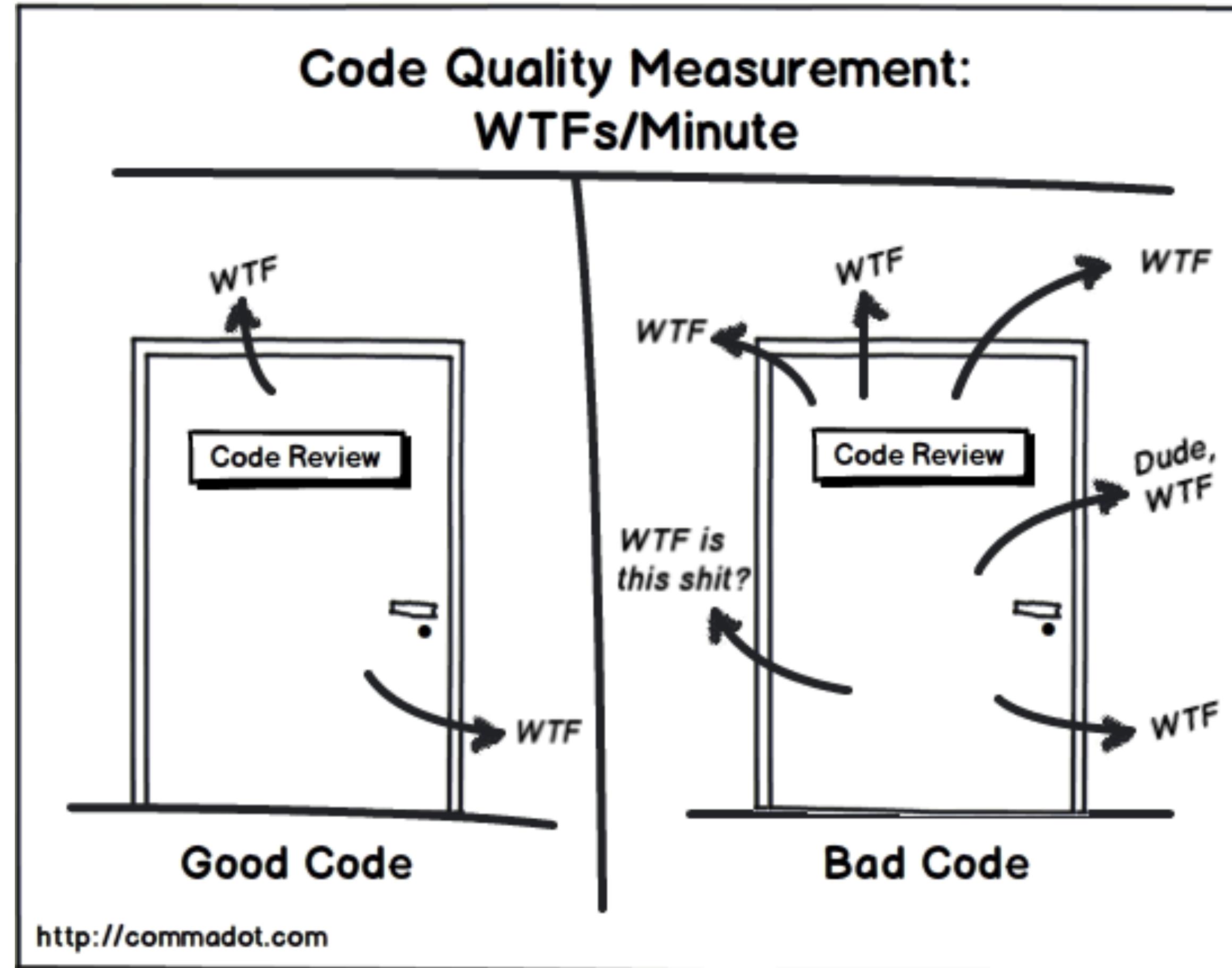
- Sistemas **envelhecem** e ficam mais **difíceis de manter** (Leis de Lehman)
- Chega-se a um ponto que a manutenção fica **extremamente custosa**; por exemplo, devido ao **montante de TD**
- Neste momento, eles são **descartados** e **substituídos** por um novo sistema
- Descarte de sistemas ocorre com mais frequência que podemos imaginar
- Exemplo: Google (<https://arxiv.org/abs/1702.01715>)
 - Most software at Google gets rewritten every few years. This may seem incredibly costly. Indeed, it does consume a large fraction of Google's resources.
 - However, it also has some crucial benefits that are key to Google's agility and long-term success.



Como Conseguir Qualidade de Código?

- E, assim, evitar Débito Técnico?
- Seguir as práticas e princípios que mencionamos neste curso:
 - Bom processo de desenvolvimento, com bons critérios para definir tarefas "done"
 - Adotar bons princípios e padrões de projeto e de arquitetura
 - Documentar as partes mais críticas do código, talvez usando UML
 - Escrever sempre testes e, talvez, usar test-driven development
 - Refatorar o código sempre
 - Pagar débito técnico acumulado
 - etc
- Mas falta uma prática: revisões de código





Revisões de Código

Revisões de Código

- Prática de desenvolvimento que determina que todo código **antes de entrar em produção** tem que ser **revisado e aprovado** por, pelo menos, um outro membro do time
- **Revisar:** ler e entender o código, buscando por problemas de qualidade
- Prática hoje seguida por grandes empresas produtoras de software; e também por empresas menores
- Revisão ocorre de forma **assíncrona** (ao contrário de pair programming, que exige dois programadores, trabalhando sincronamente no mesmo código)



Exemplo de Revisão (1)

feature1

```
...     ... @@ -0,0 +1,2 @@
1  + if (enabled)
2  +     area = pi * raio * raio * raio;
```

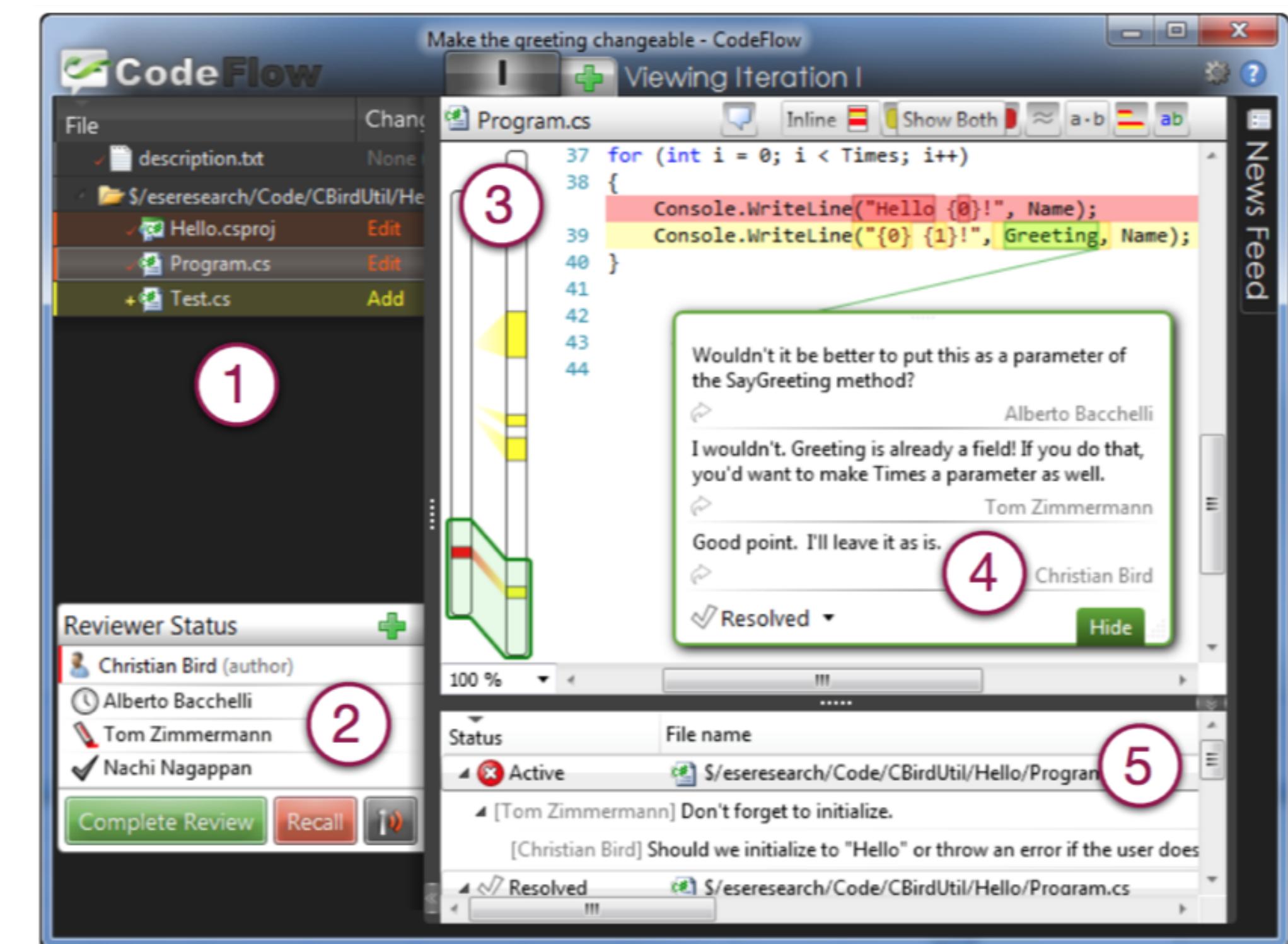
 mtov 12 minutes ago Author Owner

Não seria $\pi * \text{raio} * \text{raio}$?



Exemplo de Revisão (2)

1. arquivos para revisar
2. revisores
3. diff
4. comentários da revisão
5. todos os comentários

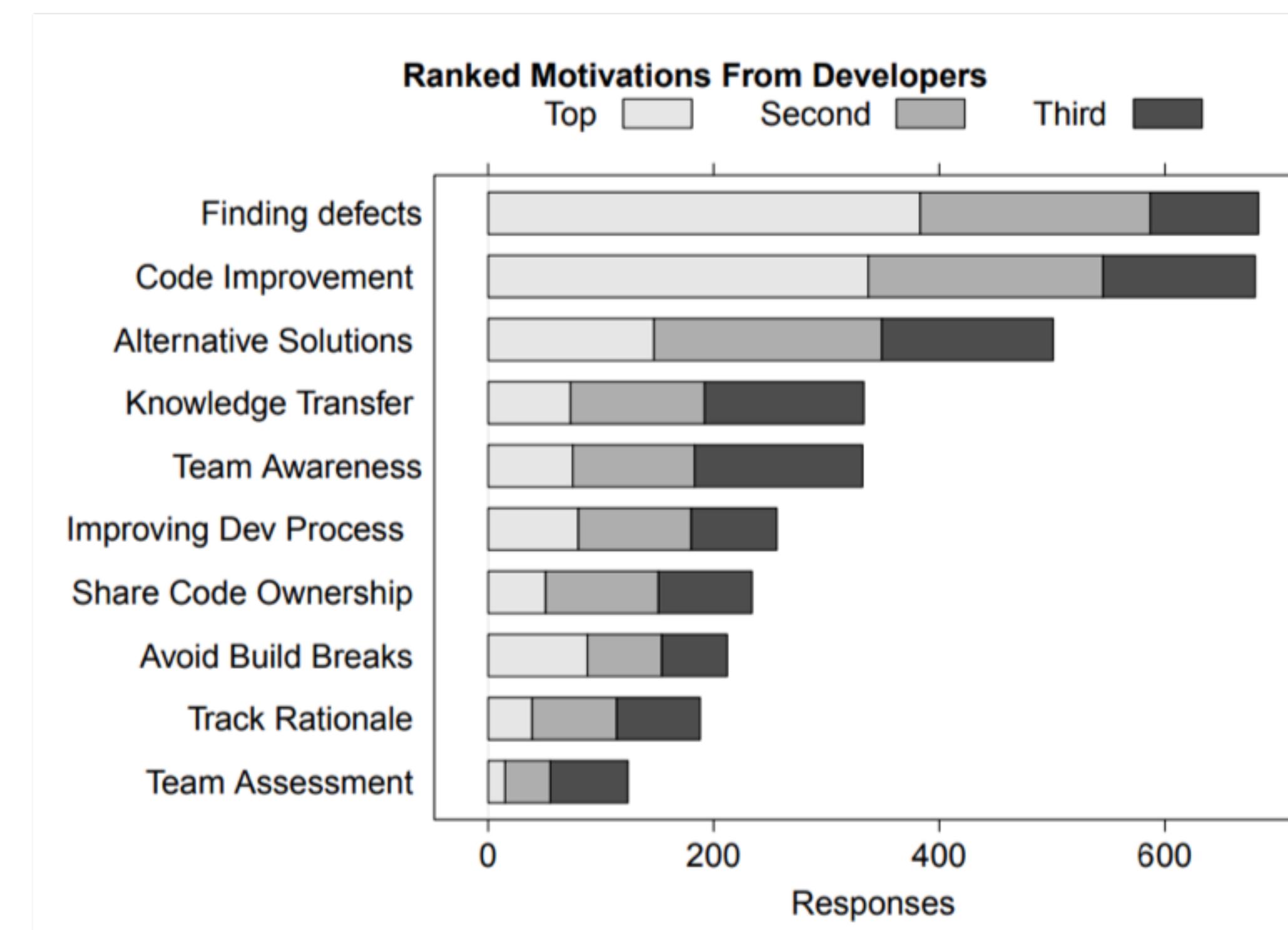


A. Bacchelli, C. Bird Expectations, outcomes, and challenges of modern code review, ICSE 2013.

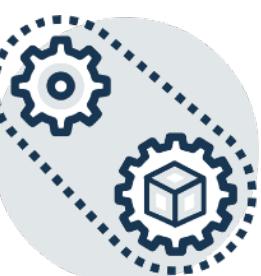


Qual o objetivo de revisões de código?

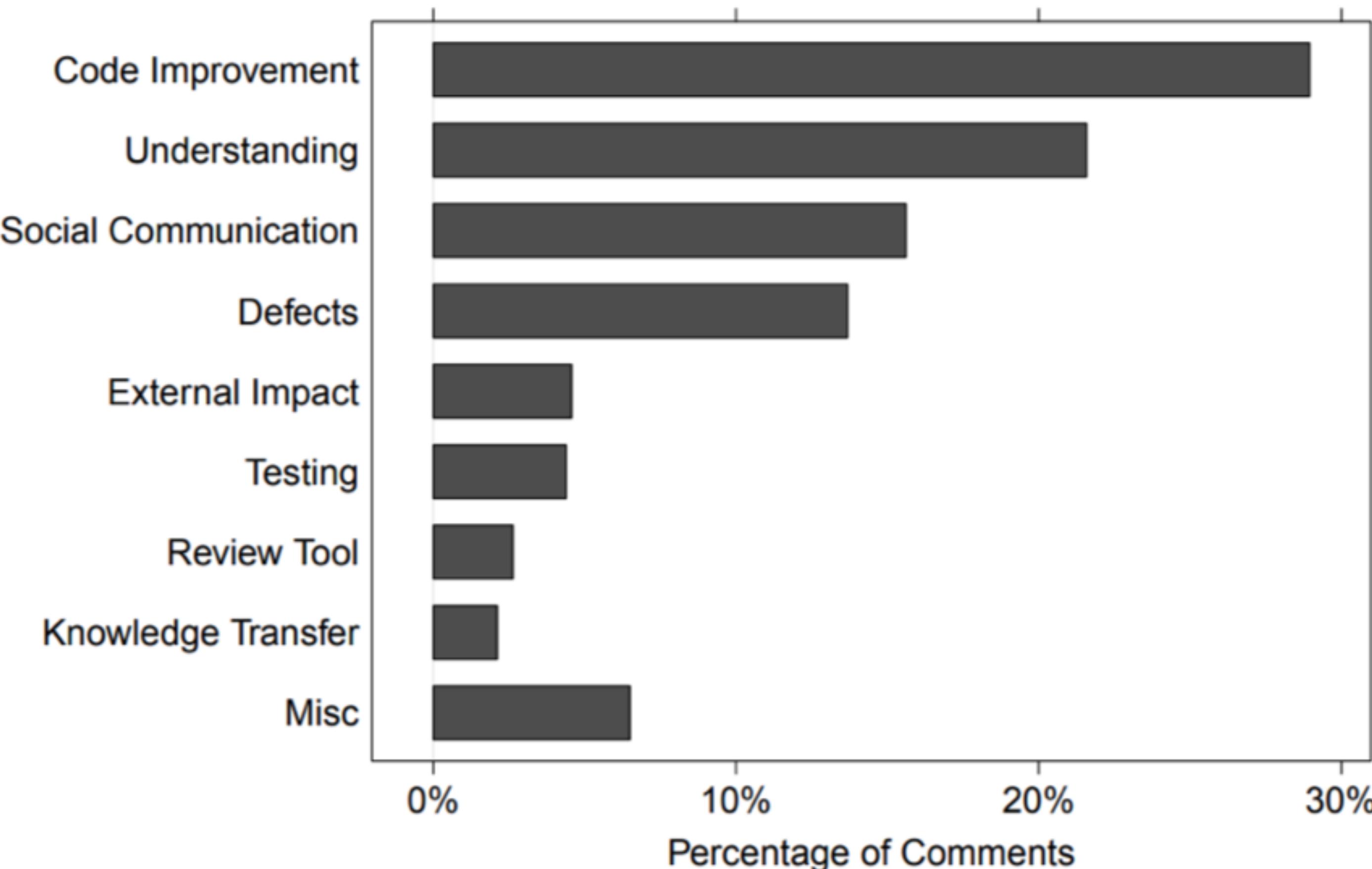
- Resultados de um survey com desenvolvedores e gerentes da Microsoft



A. Bacchelli, C. Bird Expectations, outcomes, and challenges of modern code review, ICSE 2013.



Tipos de comentários mais comuns em revisões

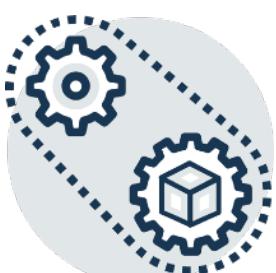


A. Bacchelli, C. Bird Expectations, outcomes, and challenges of modern code review, ICSE 2013.



"Etiqueta" para Revisão de Código (como autor)

- Primeiro, **sempre** usar um serviço de **integração contínua** (para garantir que o build não foi quebrado, que os testes estão passando etc)
- Como autor: submeta sempre mudanças **pequenas**
 - Menos de 100 linhas, idealmente
 - Revisões com um "diff" pequeno são ainda mais importante se você for novo no projeto ou na empresa. Certamente, haverá resistência para revisar um "diff" enorme vindo de um novato ... Vale aqui o princípio de XP: **baby steps!**



"Etiqueta" para Revisão de Código (como revisor)

- Como revisor: seja educado e construtivo nas revisões
- Exemplos:
 - ✖ Por que não quebrou esse método em métodos menores?
 - ✓ Talvez, possa avaliar a quebra desse método em métodos menores.
 - ✖ Você não aprendeu no colégio que a área de um círculo é $\pi * \text{raio}^2$?
 - ✓ O correto não seria $\pi * \text{raio}^2$?



Revisão de Código no Google

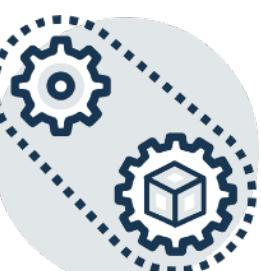
- All changes must be reviewed by at least one other engineer
- Google has tools for suggesting reviewer(s), by looking at
 - the ownership of the code being modified
 - the history of recent reviewers
 - the number of pending code reviews for each potential reviewer
- At least one of the owners must review that change
- But apart from that, the author is free to choose reviewers as they see fit.



Software Engineering at Google, <https://arxiv.org/abs/1702.01715>

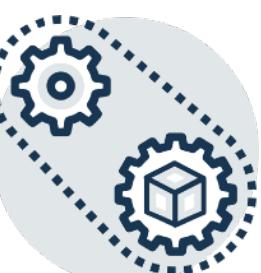
Bus/Truck Factor

- Medida da "**concentração de conhecimento**" em projetos de software
- Número mínimo de desenvolvedores que se "**atropelados por um ônibus ou caminhão**" irão colocar a manutenção e evolução do sistema em **risco**
- "atropelado por um ônibus ou caminhão" = mudar de emprego, ganhar na loteria, ficar de licença médica por meses, etc
- Por exemplo, maioria dos sistemas GitHub tem TF = 1 ou 2
 - Avelino et al. [A Novel Approach for Estimating Truck Factors](#)
- O que TF tem a ver com revisão de código?
 - Revisão de código serve para "**socializar**" o conhecimento do código, logo para reduzir dependência de desenvolvedores-chave



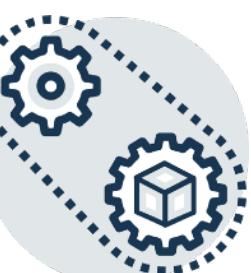
Pull Request

- Mecanismo oferecido pelo GitHub para que desenvolvedores externos possam contribuir com um projeto no qual eles não tem permissão de push
- Suponha um projeto de seu grupo do trabalho prático dessa disciplina: para integrar uma modificação, você dá um push (pois tem permissão para isso)
- Mas suponha que você queira contribuir com o facebook/react?
- Pull Request: uma solicitação para que os donos de um projeto integrem um novo código que desenvolveu (i.e., uma lista de commits)
- Logo, os donos do projeto vão ter que analisar sua contribuição e decidir se aprovam ou não



Pull Request vs Revisão de Código

- O que PR tem a ver com revisão de código?
- Revisão de código é **essencial** para **aceitação** de PRs
- Por isso, o GitHub também oferece uma ferramenta de **revisão de código**
- Para se beneficiar dessa ferramenta, algumas empresas optam por adotar o modelo de PRs, mesmo usando repositórios privados
 - Em vez de dar permissão de escrita aos seus desenvolvedores
- Outras ferramentas de revisão de código: GitLab (merge request), Gerrit, Phabricator (Facebook), CodeFlow (Microsoft)



Outras Técnicas para Garantia de Qualidade

- Inspeções de Código
- Ferramentas de Análise Estática

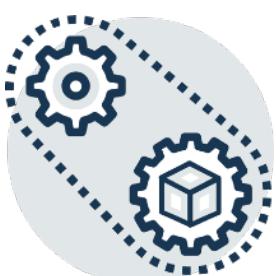


44



Inspeções de Código

- Processo **formal** de **verificação** de código
- Exemplo: processo proposto por Fagan, na IBM, ainda na década de 70:
 - Time de desenvolvedores (em geral, 4 membros) se reúne fisicamente e inspeciona cada linha do código
 - Composição dos times: um moderador, dois leitores (ou revisores, ou inspectores) e o autor do código
 - Tem como único objetivo encontrar defeitos
 - Depende de uma especificação detalhada dos requisitos do código
 - Logo, é uma técnica de verificação



Ferramentas de Análise Estática

- **Vantagem:** não demandam uma especificação como entrada; isto é, elas tentam detectar bugs automaticamente
- **Estaticamente:** via análise estática de código, sem demandar sua execução
- Às vezes, chamadas também de **linters** (não confundir com check style)
- Problema: costumam emitir um **alto número de falsos positivos**

