THE
DEVELOPER'S
CONFERENCE

Análise de Código:

precisamos falar disso!

Quem somos?

Daniel Wildt
Wildtech – Umov.me
@dwildt

Guilherme Lacerda
Wildtech – Unisinos
@guilhermeslac

“Grande parte do **dinheiro gasto** com desenvolvimento de **software** é **usado** para **entender códigos existentes”**

Kent Beck

***Implementation Patterns**, Addison-Wesley Professional, 2008*

Para refletir

quanto tempo você leva para “aprender” sobre
o repositório de código que você trabalha?

E se você trocar de empresa, em quanto tempo você consegue
efetivamente “colocar a mão na massa”?

O que é análise de código?

Essência da Análise: Legibilidade e Compreensão

- Estruturas **pequenas**
- Nomes **significativos**
- **Formatação** e uso de padrões *(code conventions)*
- **Organização** das estruturas e algoritmos
- Aplicação dos **princípios do paradigma**
- **Testes** automatizados

Para começar, o que preciso olhar?

- Coesão
- Acoplamento
- Tamanho
- Complexidade

Por que analisar código é importante?

- **Ampliar** nossa **capacidade cognitiva** de programação
- **Conhecer** outros **paradigmas, padrões e linguagens** (e **problemas** também!)
- **Dilemas do Programador**

  manter/evoluir X construir do zero, refactorings X testes
- **Ampliar** nossas **habilidades**

Quais habilidades eu preciso desenvolver?

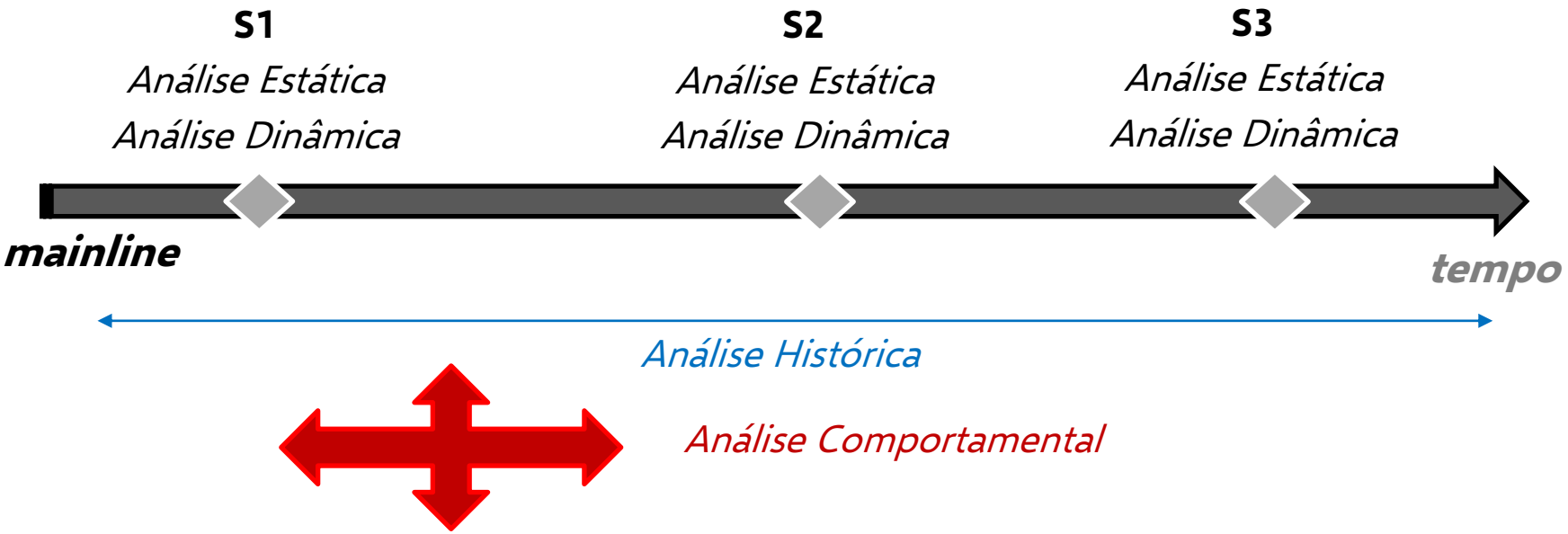
- Conhecer **heurísticas de análise**
  Módulos/Pacotes, Classes, Métodos

- Compreender aspectos de **qualidade de software**
  Atributos externos e internos

- Aplicar **métricas** de análise, estratégias de **visualização e ferramentas de apoio**
  Compreensão de software, mineração de repositórios

- **Estratégias**
  Análise Estática, Análise Dinâmica, Análise Temporal, Análise Comportamental

Estratégias Combinadas



“Na maioria das vezes, os **problemas já existem** desde sua **criação** e **não são introduzidos na evolução** do software”

Palomba et al

On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Empirical Software Engineering 23, 1188–1221, 2018*

Qual o melhor momento para fazer análises?

- **Sempre!!**
  E de forma antecipada e quando possível

- **Individualmente**, antes de fazer commits
  Inspeção na fonte (Poka-yoke, do Lean)
  Apoiado por ferramentas (Jidôka, do Lean)

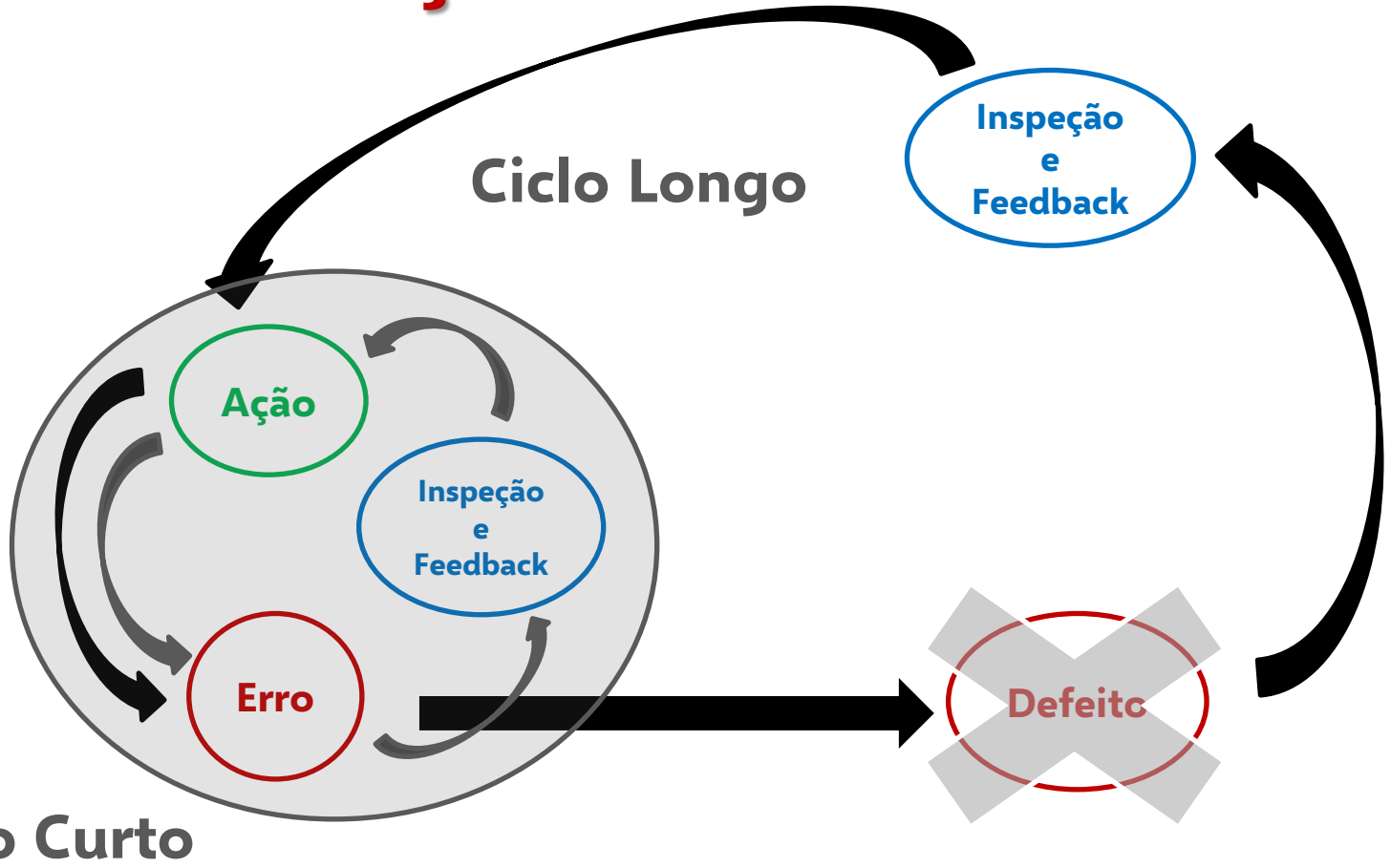
- **Em par**, para discutir situações específicas

- Em sessões de **Code Review**

“**Classes** afetadas por **mais de um smells** são mais **propensas a mudanças e a falhas**”
Palomba et al

A large-scale empirical study on the lifecycle of code smell co-occurrences, *Information and Software Technology, Volume 99, 2018*

Relação Erro e Defeito

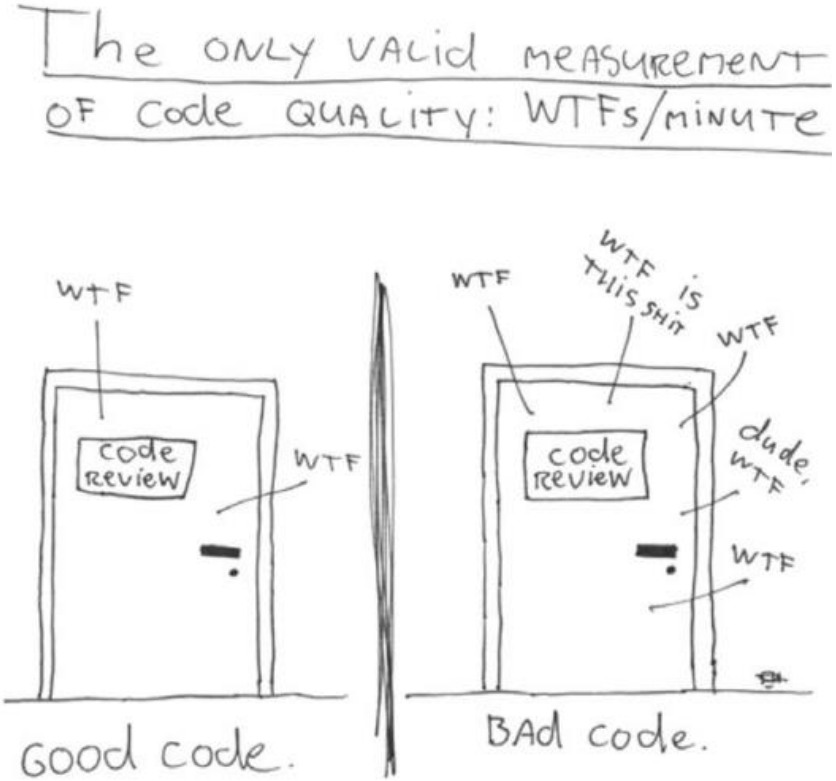


"As **ferramentas de qualidade** podem **ajudar a evitar** com que **problemas aconteçam**, simplesmente pela adoção dessas **ferramentas antes das confirmações** (commits), **evitando** ou **limitando a introdução** de **novos problemas no código**..."

Tufano et al

When and Why Your Code Starts to Smell Bad, *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, 2015*

Como você avalia a qualidade do seu código?



“Os **resultados** revelam **apenas 11% dos builds** estão sujeitos a **verificações de qualidade** de código”

Vassallo et al

Continuous code quality: are we (really) doing that? In Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, 2018

Ferramentas

DR-Tools

CODESCENE

sonarqube

JUnit

PMD

DON'T SHOOT THE MESSENGER

ArchUnit

scitools™ Understand

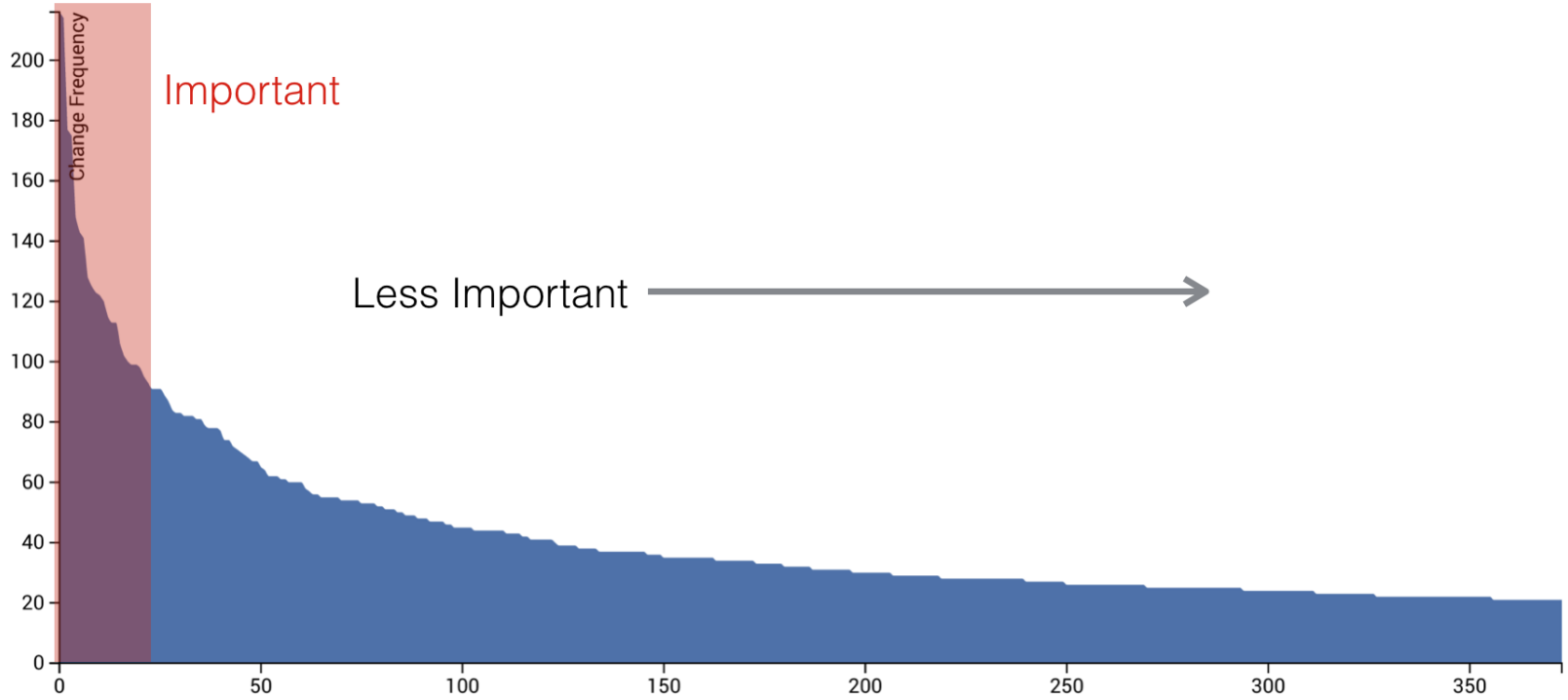
SourceLevel

“Os resultados sugerem que **projetos** que adotam uma **política explícita de melhoria da qualidade** (ex. reuniões específicas do time para qualidade) estão associados a uma **maior frequência de commits** de **códigos mais limpos”**

Digkas et al

Can Clean New Code reduce Technical Debt Density?. *arXiv preprint arXiv:2010.09161, 2020*

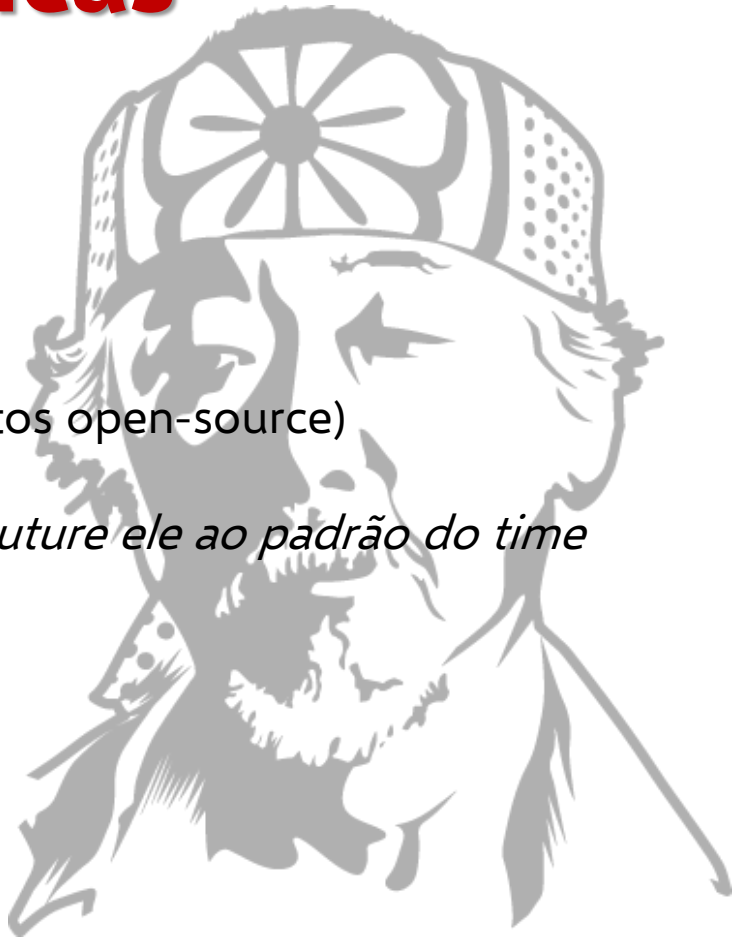
Priorizando...



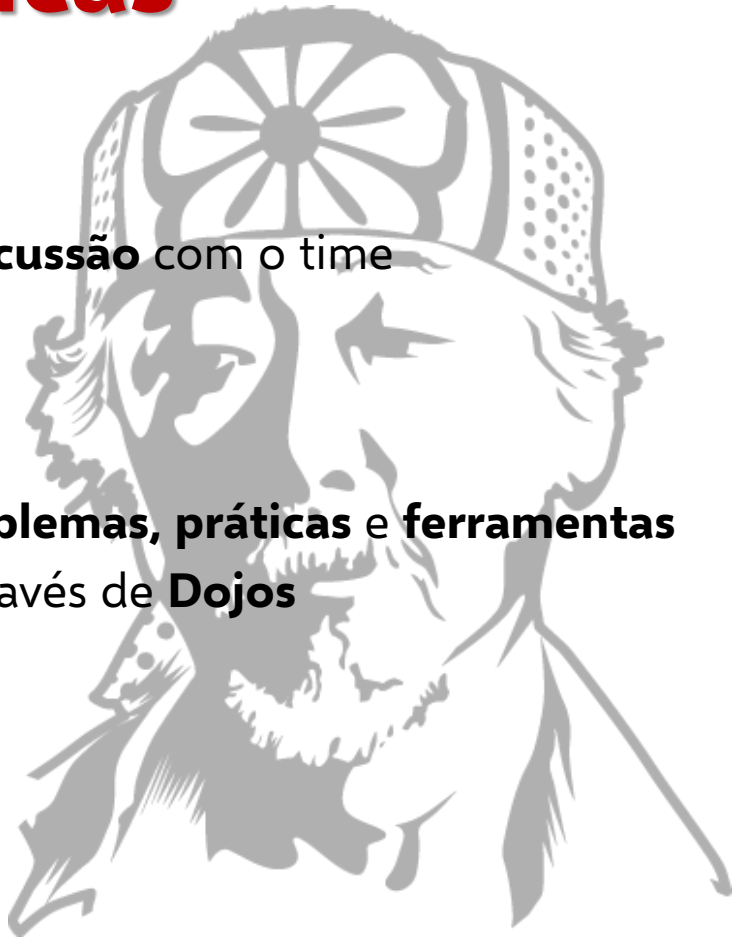
Tornhill, A. **Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis**. The Pragmatic Programmers, 2018.

Algumas dicas

- Regra dos **30 segundos**
- Regra do **Escoteiro**
- **Metáfora** do **jornal**
- **Olhe código de outros** (principalmente projetos open-source)
- **Pegou** um código **da Web**? **Ajuste-o**
  - *Antes de vincular ao seu repositório reestruture ele ao padrão do time*
- **Conheça** sua **ferramentas**
- Use **automação** em **diferentes níveis**
- **Defina políticas** de qualidade
  - *Quality Gates, Continuous Code Quality*



Algumas dicas



- **Ao analisar** o código, **marque** pontos **para discussão** com o time
  - *Análise de Código X Revisão de Código*
- **Estudem** e **pratiquem**!
- Monte o **plano de metas** com o time
- Criem uma **rotina** com o time para discutir **problemas, práticas** e **ferramentas**
- **Experimentem** (novas LPs, IDEs, ambientes) através de **Dojos**
- **Participem** das **comunidades** e **eventos**
- **Entendam** que é uma **jornada a seguir**

Questões??

THE
DEVELOPER'S
CONFERENCE

Análise de Código:
precisamos falar disso!

Daniel Wildt
Wildtech – Umov.me
@dwildt

Guilherme Lacerda
Wildtech – Unisinos
@guilhermeslac