APRESENTAÇÃO
DE APOIO

TÉCNICAS ÁGEIS DE PROGRAMAÇÃO

Guilherme Lacerda – Aula 02

PUCRS online uol edtech

Professores

DANIEL WILDT
Professor Convidado

Profissional de tecnologia preocupado com desenvolvimento de produtos e serviços com equipes focadas em aprendizado, melhoria contínua e autonomia. Mentora e produz conteúdo em vídeo, áudio e texto sobre: consciência de tempo, experiência de usuário, empreendedorismo e metodologias ágeis. Sócio e mentor na Wildtech, Blogger/Youtuber no danielwildt.com, sócio e diretor na uMov.me.

GUILHERME LACERDA
Professor Convidado

Graduado em Informática pela Universidade da Região da Campanha (2000). Mestre em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (2005). Atualmente, cursa Doutorado em Ciência da Computação na UFRGS, na área de Engenharia de Software. Consultor/Instrutor associado da Wildtech, trabalhando com coaching e mentoring nas áreas de Engenharia de Software, Gerência de Projetos e Produtos e Metodologias Ágeis (eXtreme Programming, SCRUM, Lean). Possui mais de 20 anos de experiência em desenvolvimento de software. Atuou por vários anos como analista/projetista/desenvolvedor de software. Possui as certificações de SCRUM Master (SCM) e SCRUM Professional (CSP) pela SCRUM Alliance. Membro do IASA (International Association of Software Architects). Fundador do Grupo de Usuários de Métodos Ágeis (GUMA), vinculado a SUCESU-RS. É docente de graduação (Ciência da Computação, Análise e Desenvolvimento de Sistemas e Sistemas de Informação, Gestão de TI - Unisinos) e pós-graduação (Engenharia de Software, Desenvolvimento de Aplicações Móveis - Unisinos e Desenvolvimento Full Stack - PUCRS).

Professores

MICHAEL DA COSTA MÓRA
Professor PUCRS

Graduado em Ciência da Computação pela Universidade Federal do Rio Grande do Sul (UFRGS), mestre em Computação e doutor em Ciência da Computação pela mesma universidade. Professor-adjunto do Instituto de Informática. Tem experiência na área de ciência da computação com ênfase em inteligência artificial, atuando principalmente nos seguintes temas: inteligência artificial, aprendizagem de máquina, agentes inteligentes e sistemas multiagentes, engenharia de software e desenvolvimento de sistemas, ensino de programação e de ciência da computação.

Ementa da disciplina

Fundamentos da agilidade: primórdios, manifesto ágil, princípios da agilidade. Panorama das metodologias ágeis. Extreme programming: características, valores, práticas, as práticas na prática. Test driven development (TDD): origens, codificar – testar – projetar, benefícios e armadilhas, variações, TDD na prática. Behaviour driven design (BDD): origens e princípios, BDD x TDD, benefícios e armadilhas, BDD na prática.

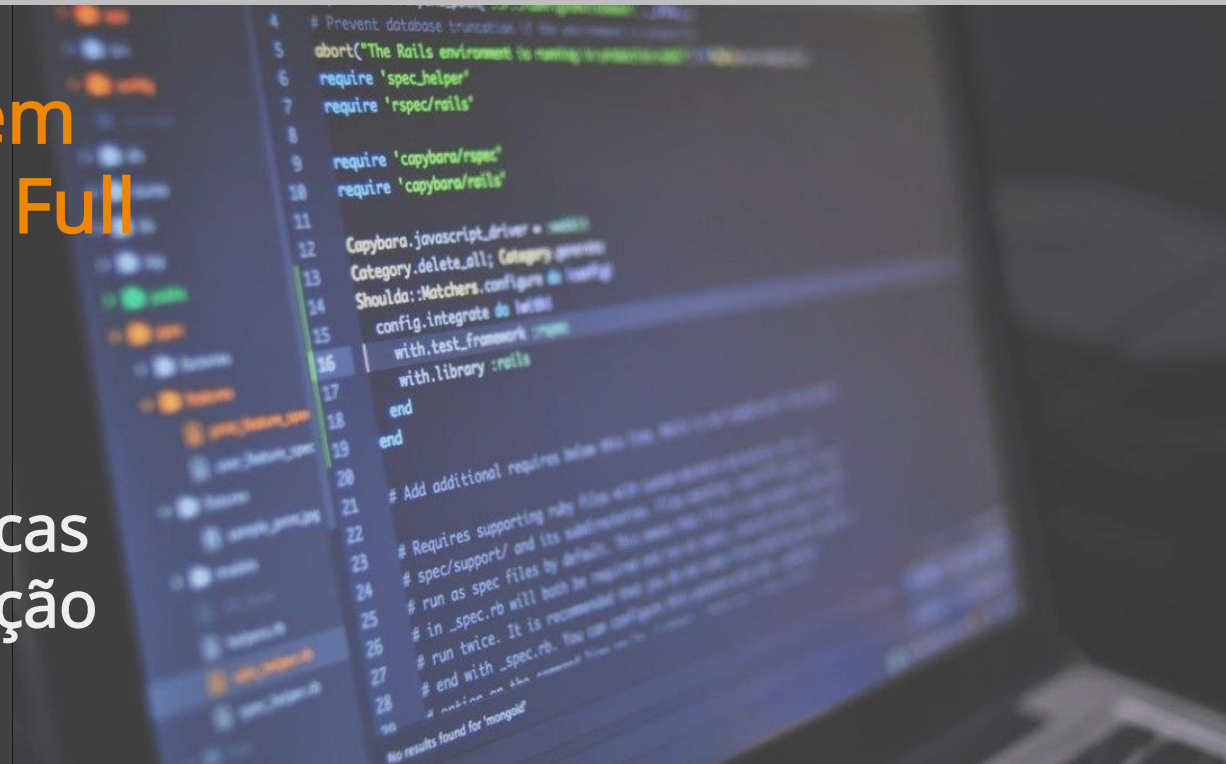
PUCRS

uol edtech

WILDTECH
start wild, keep wild

Especialização em Desenvolvimento Full Stack

Disciplina de Técnicas Ágeis de Programação



Parte 3

Agenda - Práticas
Parte 3

1. Dívida Técnica
2. Refactoring e Heurísticas de Limpeza

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

Kent Beck

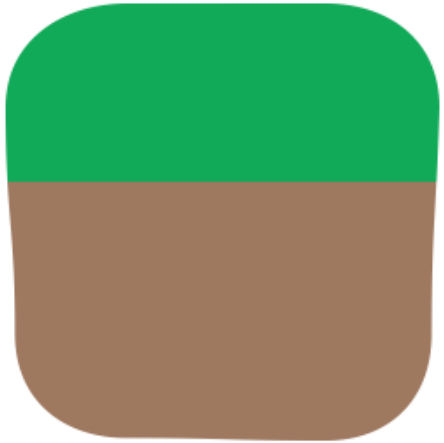
Dívida Técnica

- Criado por **Ward Cunningham**
- Originária do **setor financeiro**

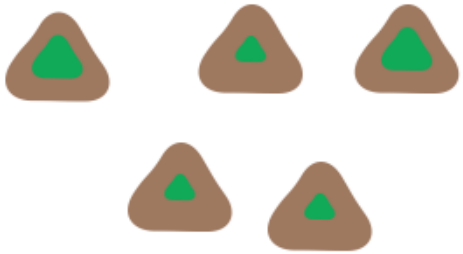
Dívida Técnica

Any software system has a certain amount of ***essential*** *complexity required to do its job...*

... but most systems contain ***cruft*** *that makes it harder to understand.*



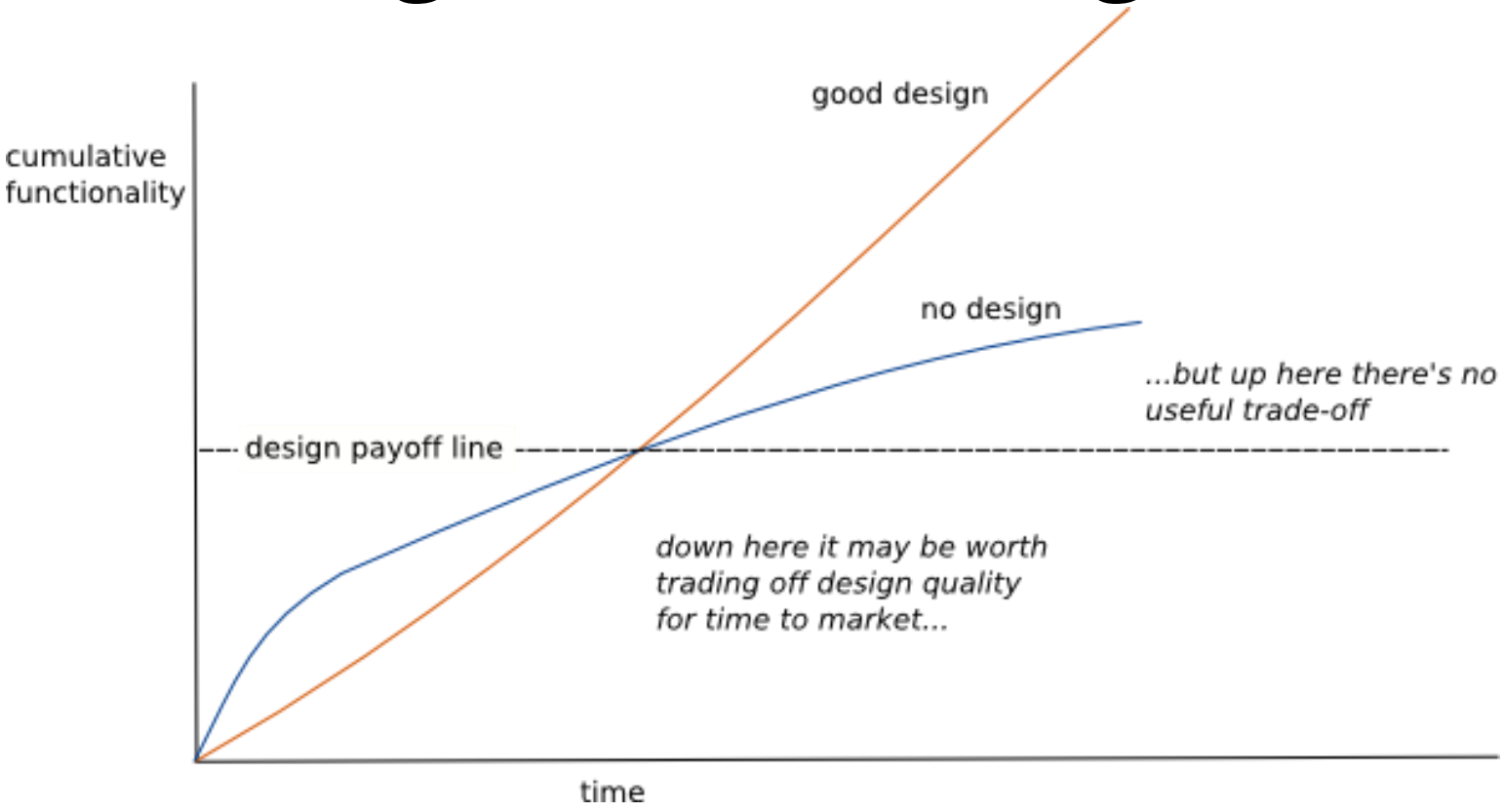
Cruft causes changes to take ***more effort***



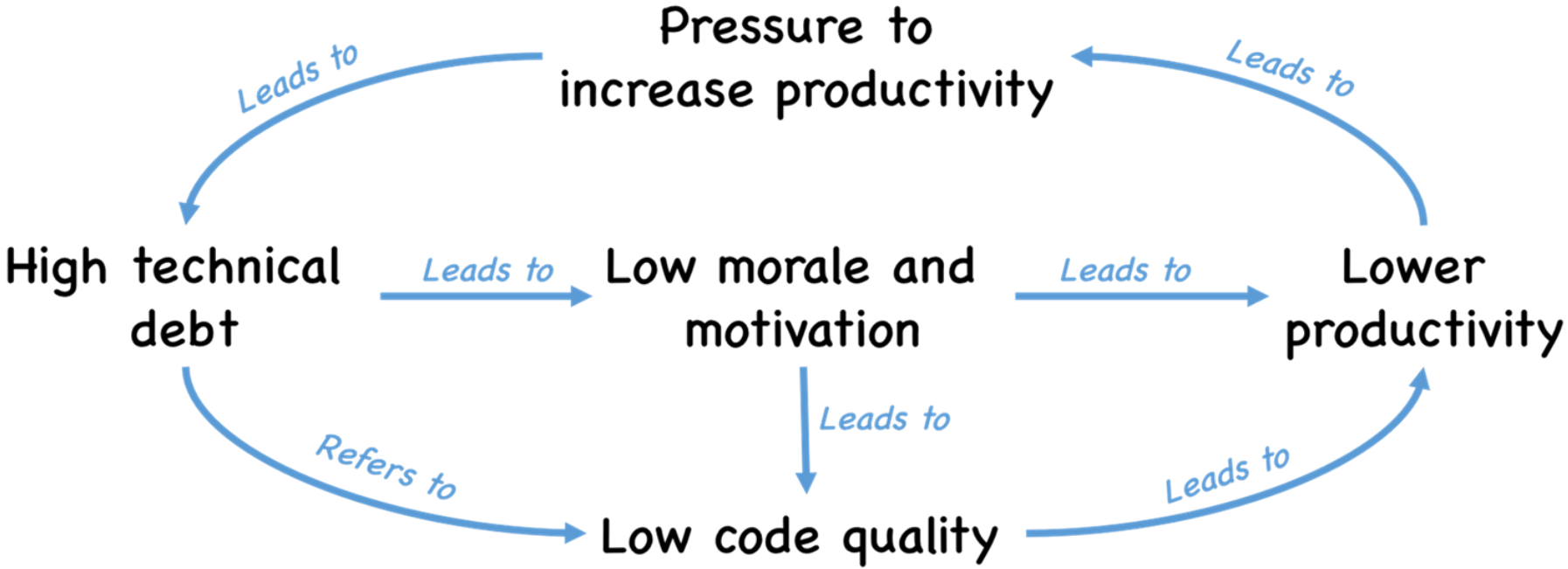
The technical debt metaphor treats the cruft as a debt, whose interest payments are the extra effort these changes require.

https://martinfowler.com/bliki/TechnicalDebt.html

Design Bom X Design Ruim

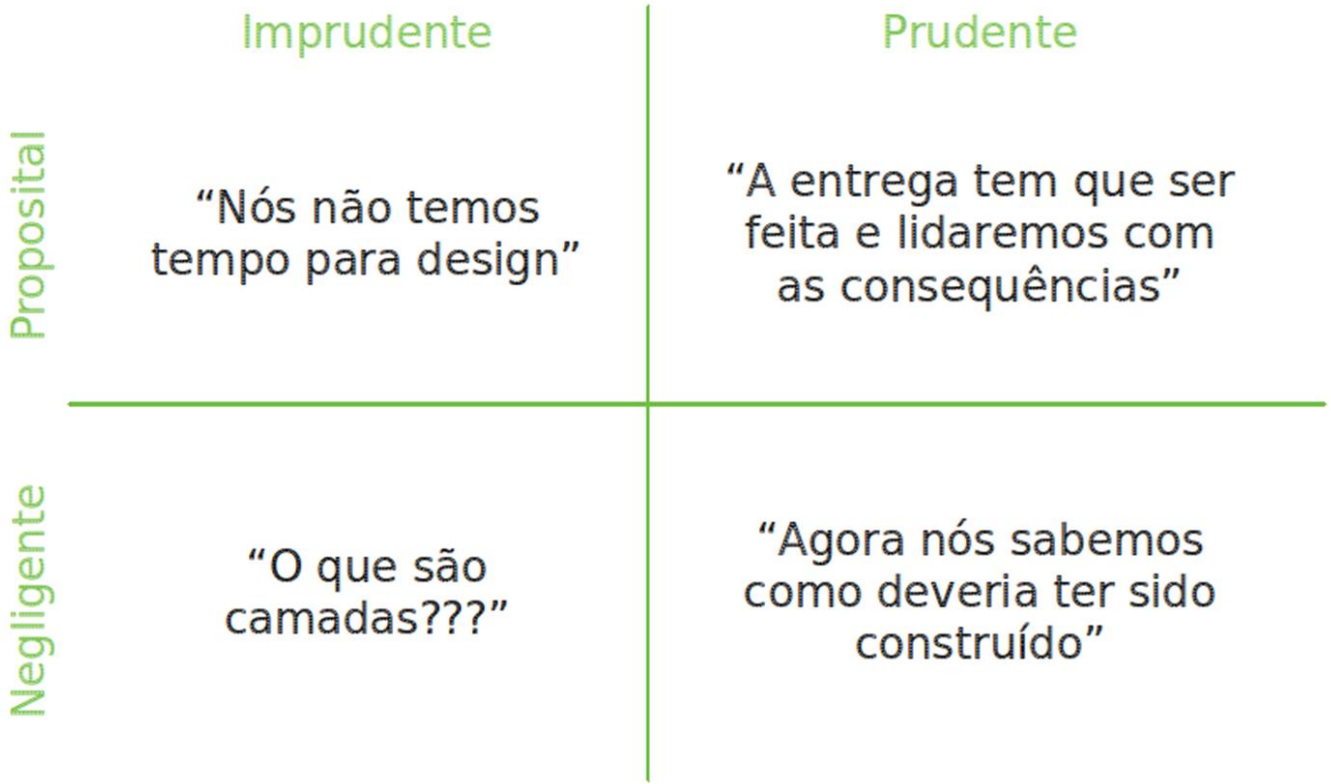


https://martinfowler.com/bliki/DesignStaminaHypothesis.html



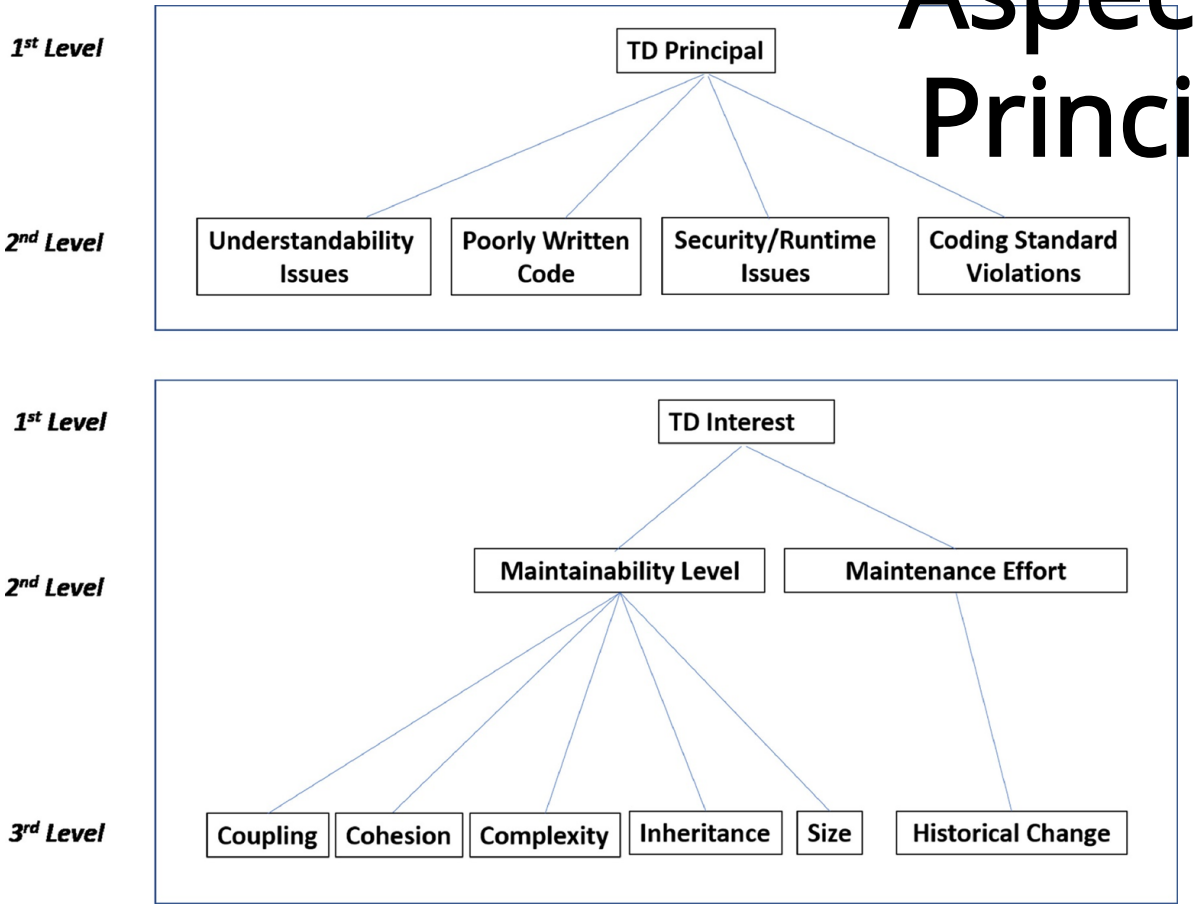
https://tusharma.medium.com/four-strategies-for-managing-technical-debt-9d3d37a0c862

Quadrantes



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

Aspectos entre o Principal e Juros



Amptzoglou A. et al (2020). Exploring the relation between technical debt principal and interest: An empirical approach, Information and Software Technology, Volume 128.

Tipos de Dívida Técnica

- Design Debt
- Architecture Debt
- Documentation Debt
- Test Debt
- Code Debt
- Defect Debt
- Requirements Debt

- Infrastructure Debt
- Test Automation Debt
- Process Debt
- Build Debt
- Service Debt
- Usability Debt
- Versioning Debt

Alves N. et al (2016). **Identification and management of technical debt: A systematic mapping study**, Information and Software Technology, Volume 70.

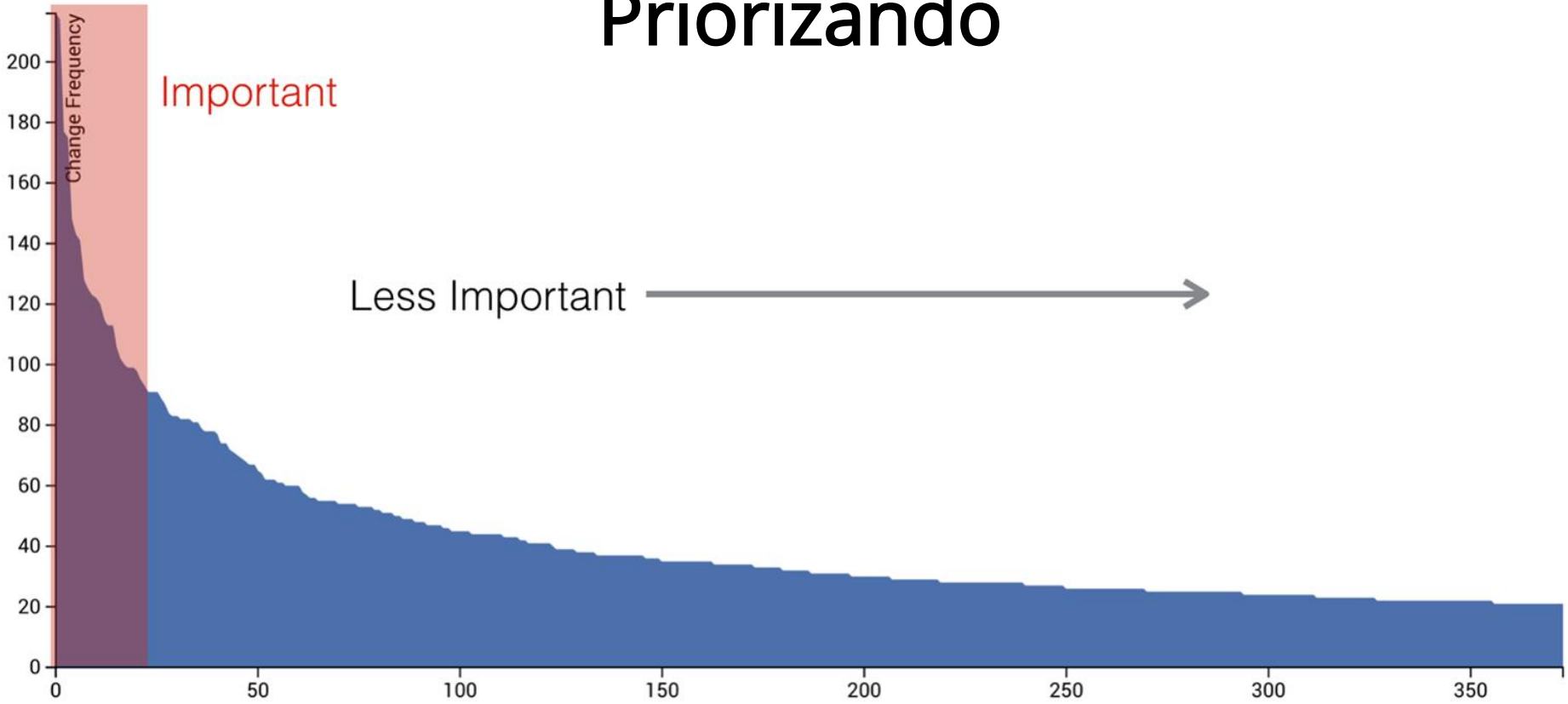
Atividades Relacionadas

- Identification
- Measurement
- Prioritization
- Prevention
- Monitoring
- Repayment
- Representation/documentation
- Communication

TDM activity	No. of studies	%
TD repayment	59	63
TD identification	51	54
TD measurement	49	52
TD monitoring	19	20
TD prioritization	17	18
TD communication	17	18
TD prevention	9	10
TD representation/documentation	4	4

Zengyang Li et al (2016). **A systematic mapping study on technical debt and its management**, Journal of Systems and Software, Volume 101.

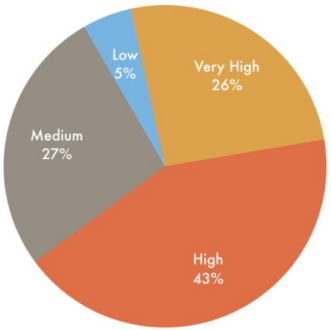
Priorizando



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

Alguns dados da indústria

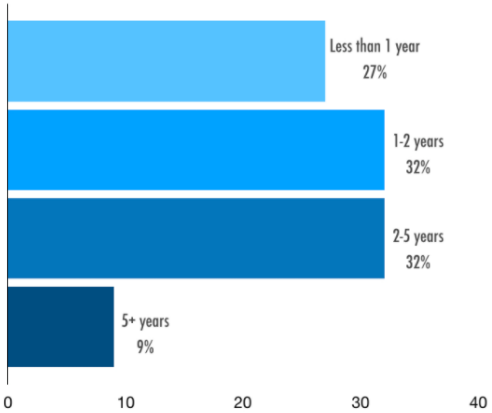
How much Tech Debt is present in the project you work on?



60% of software developers work on products with 'high' or 'very high' tech debt.

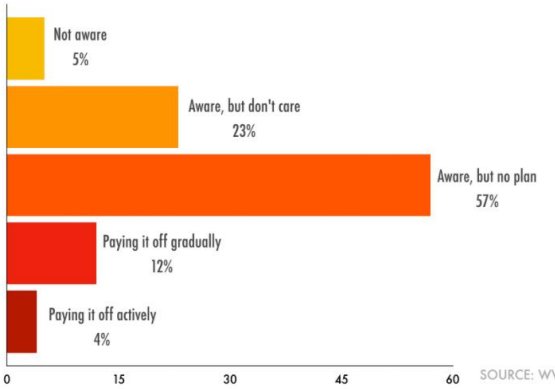
SOURCE: WWW.CODEAHOY.COM

How long have you been working on the product with Tech Debt?



SOURCE: WWW.CODEAHOY.COM

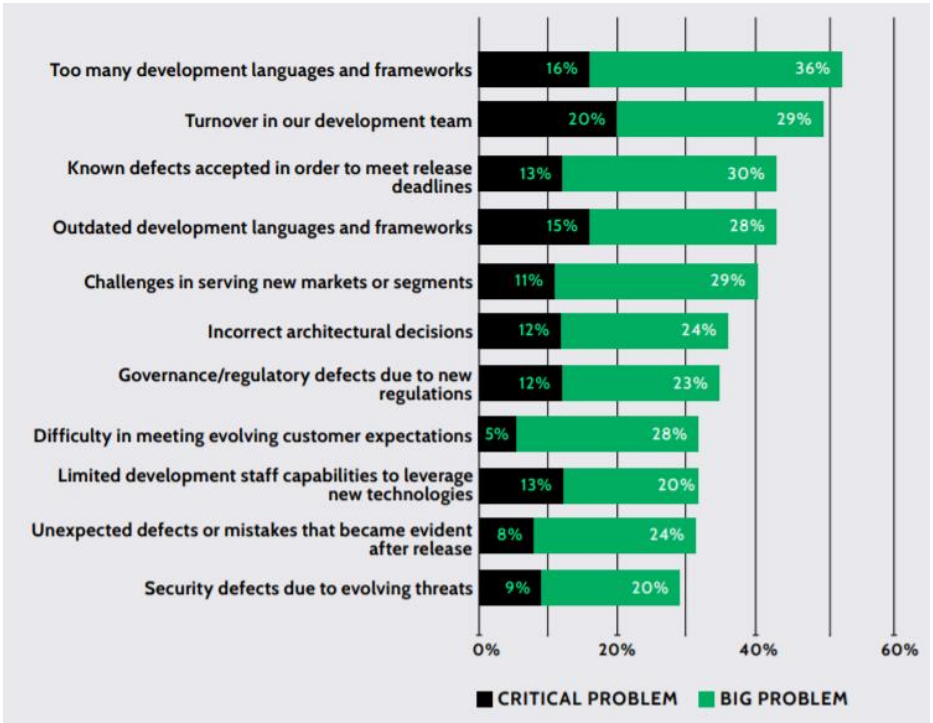
Is your management aware of Tech Debt and are they taking action to pay it off?

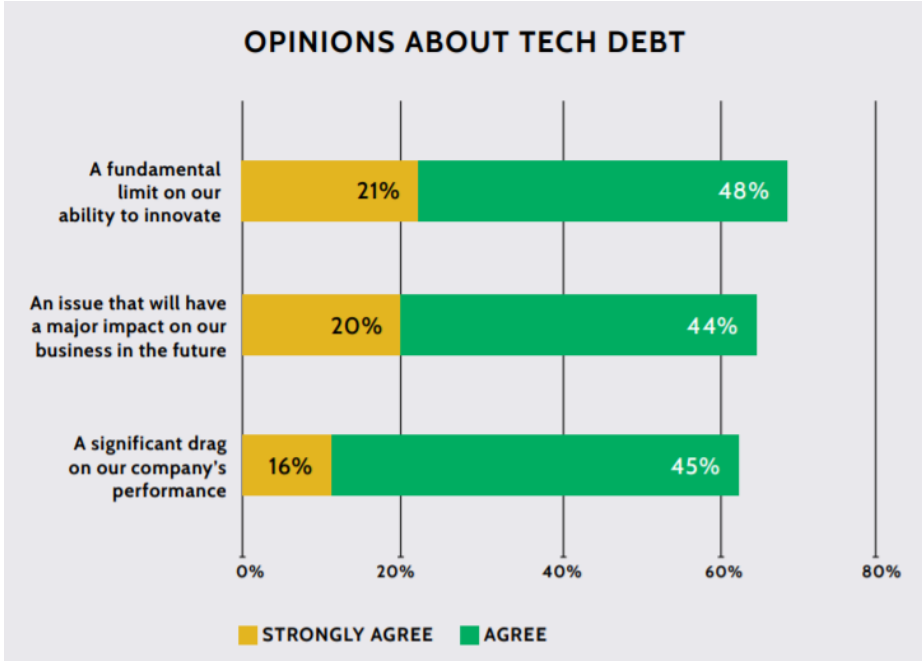


SOURCE: WWW.CODEAHOY.COM

https://codeahoy.com/2020/02/17/technical-debt-survey/

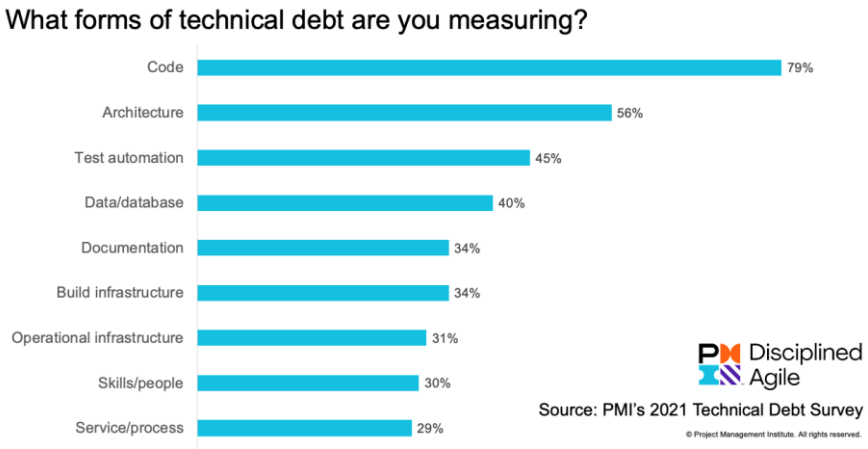
Alguns dados da indústria

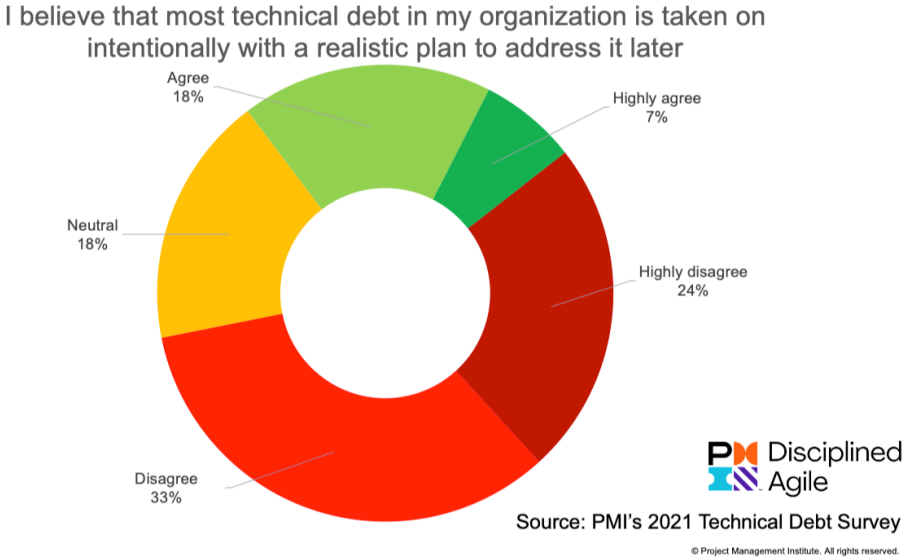




https://www.outsystems.com/news/study-reveals-technical-debt-is-threat-to-innovation/

Pesquisa do PMI sobre Dívida Técnica





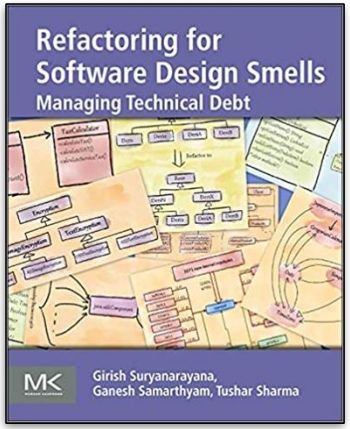
https://www.projectmanagement.com/blog/blogPostingView.cfm?blogPostingID=71979&thisPageURL=%2Fblog-post%2F71979%2Fis-technical-debt-a-management-problem---survey-says---&s=03#_=_

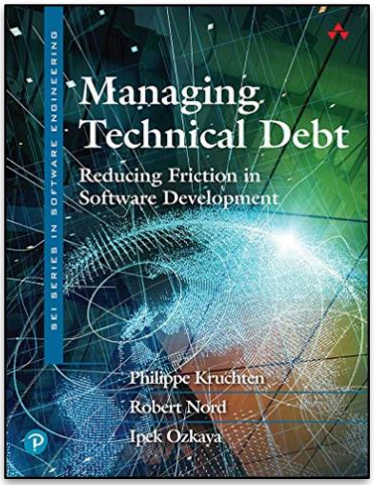
Alguns dados da indústria

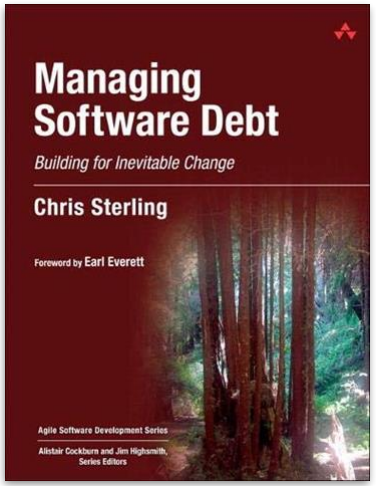
- Software development is rarely sustainable. The average organization wastes 23- 42% of their development time due to technical debt.
- Hiring more developers increases coordination costs, which in turn makes the development less efficient, particularly in codebases rife with technical debt.
- If your organization spends more than 15% on Unplanned Work, then that's a warning sign that delivery potential is wasted. Technical debt is likely to be a significant chunk of that waste.
- Technical debt is often mistaken for "bad code in general". This is a dangerous fallacy that leads organizations to waste months on improvements that don't have a clear business outcome or aren't urgent.
- Instead, the costs of technical debt can be quantified by calculating excess unplanned work via the formula we provid.
- Based on data, many organizations pay for 100 developers, but are only getting the output equivalent of 75 developers.
- Technical debt is only one factor that often comes together with team or process issues that need to be understood and addressed. Modern tooling helps detect the bottlenecks.
- By addressing the root causes, an organization is likely to increase their effective development capacity by least 25%.
- 25% extra capacity means you could deliver more features and also get a clear win in customer satisfaction due to improved quality.

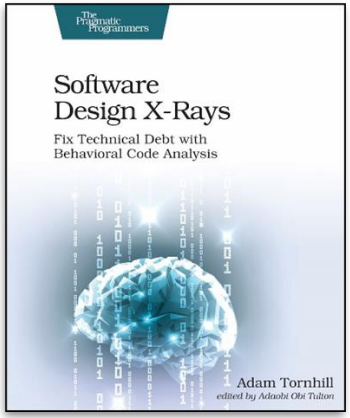
https://codescene.com/whitepapers

Para aprofundar os estudos...









“Profissionais passam 40% da sua jornada semanal lidando com problemas de manutenção, como depuração e refatoração, além de correção de ***‘código mal escrito’****. Segundo a pesquisa, o impacto disso equivale a quase US$ 85 bilhões em custo de oportunidade perdido anualmente em todo o mundo, de acordo com os cálculos sobre o salário médio do desenvolvedor por país”*

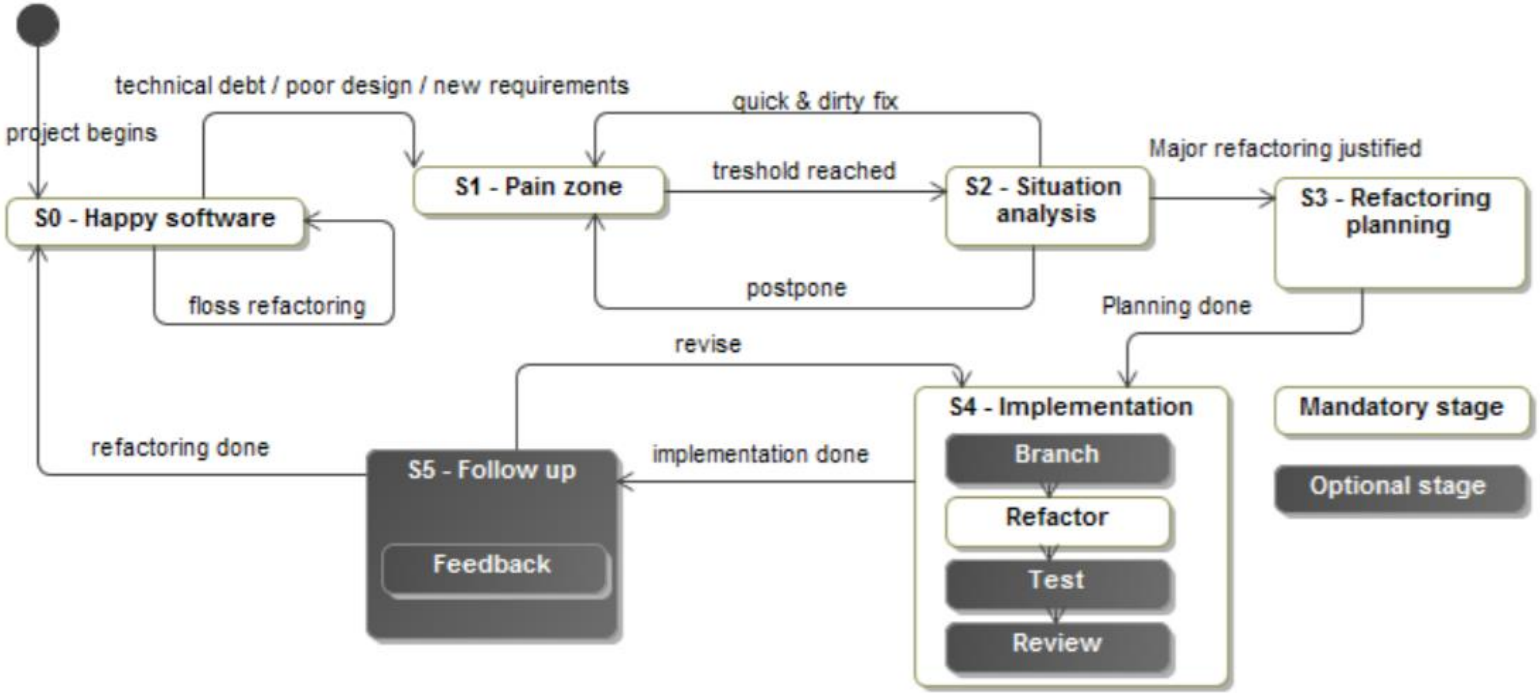
Stripe, The Developer Coefficient (2018)

Refactoring: um processo complexo

1. **Identify where** the software **should be refactored**;
2. **Determine which refactoring(s)** should **be applied** to the identified places;
3. **Guarantee** that the applied refactoring **preserves behaviour**;
4. **Apply** the refactoring;
5. **Assess the effect** of the **refactoring on quality characteristics** of the software (e.g., complexity, understandability, maintainability) or the process (e.g., productivity, cost, effort);
6. **Maintain the consistency** between the **refactored program code** and other **software artifacts** (such as documentation, design documents, requirements specifications, tests and so on)

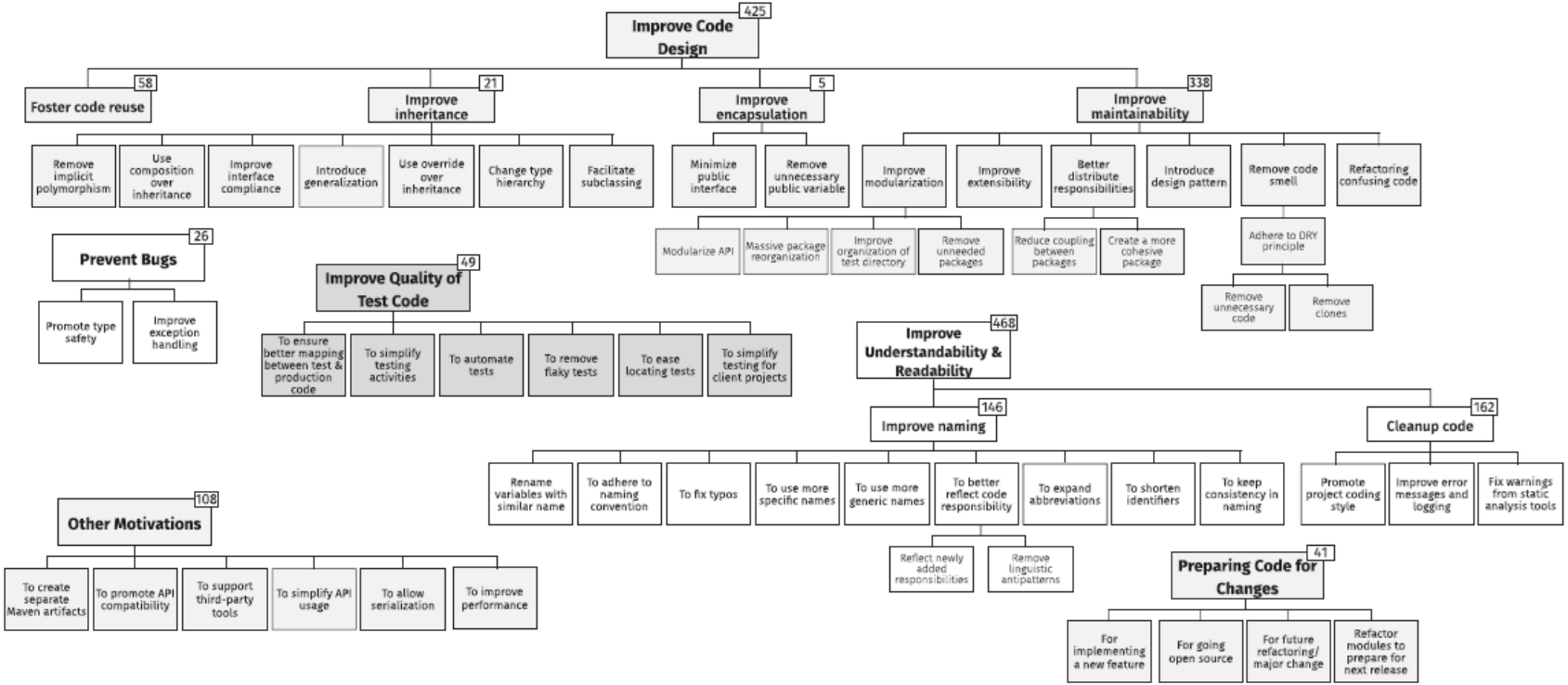
Mens T. & Tourwe T. (2004). **A survey of software refactoring**, In: IEEE Transactions on Software Engineering, vol. 30, no. 2.

Processo de Decisão



Leppanen M. et al. (2015) **Decision-making framework for refactoring**. In: 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD). [S.l.: s.n.]

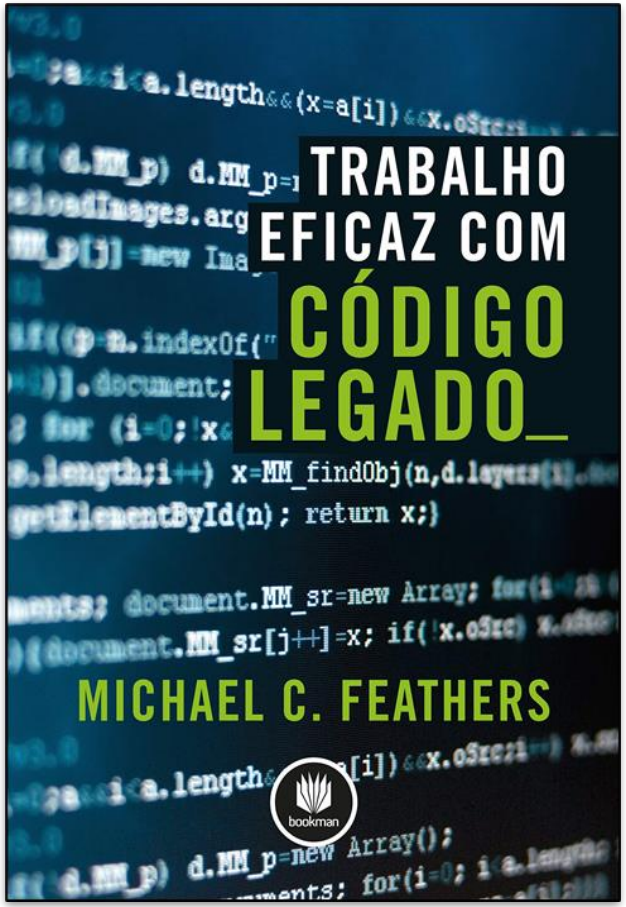
Motivadores



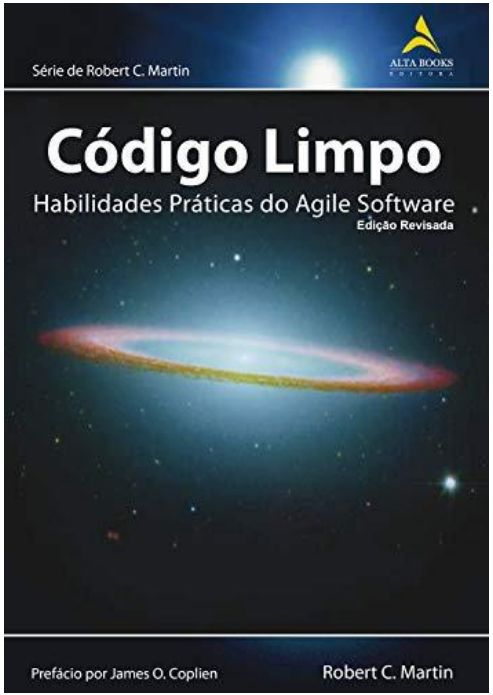
Pantiuchina J. et al. (2020) **Why developers refactor source code: A mining-based study**. ACM Trans. Software Engineering Methodology 29, 4.

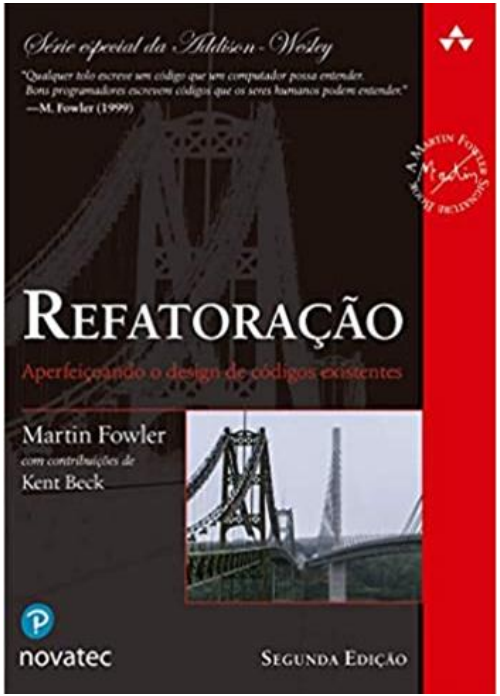
E o código legado?

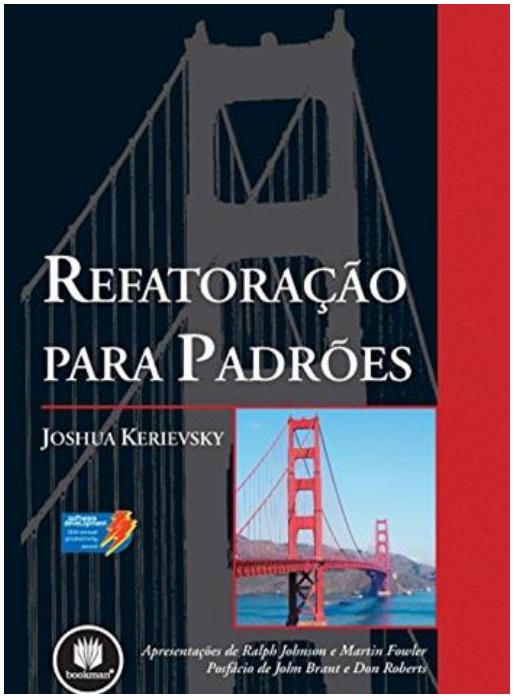
- Dilema Refactoring X Testes
- Considerar o “tempo de latência”
- Gerenciando dependências
  - pontos de intercepção
  - pontos de fixação
- Use notas/diagramas
- Identifique responsabilidades
- Refatoração Transitória



Para aprofundar os estudos...







Fim Parte 3

Parte 4

Agenda - Práticas
Parte 4

1. Análise de Código
2. DR-Tools
3. Práticas de Engenharia e o Agilista/Agile Coach/Agile-Scrum Master?

Para refletir

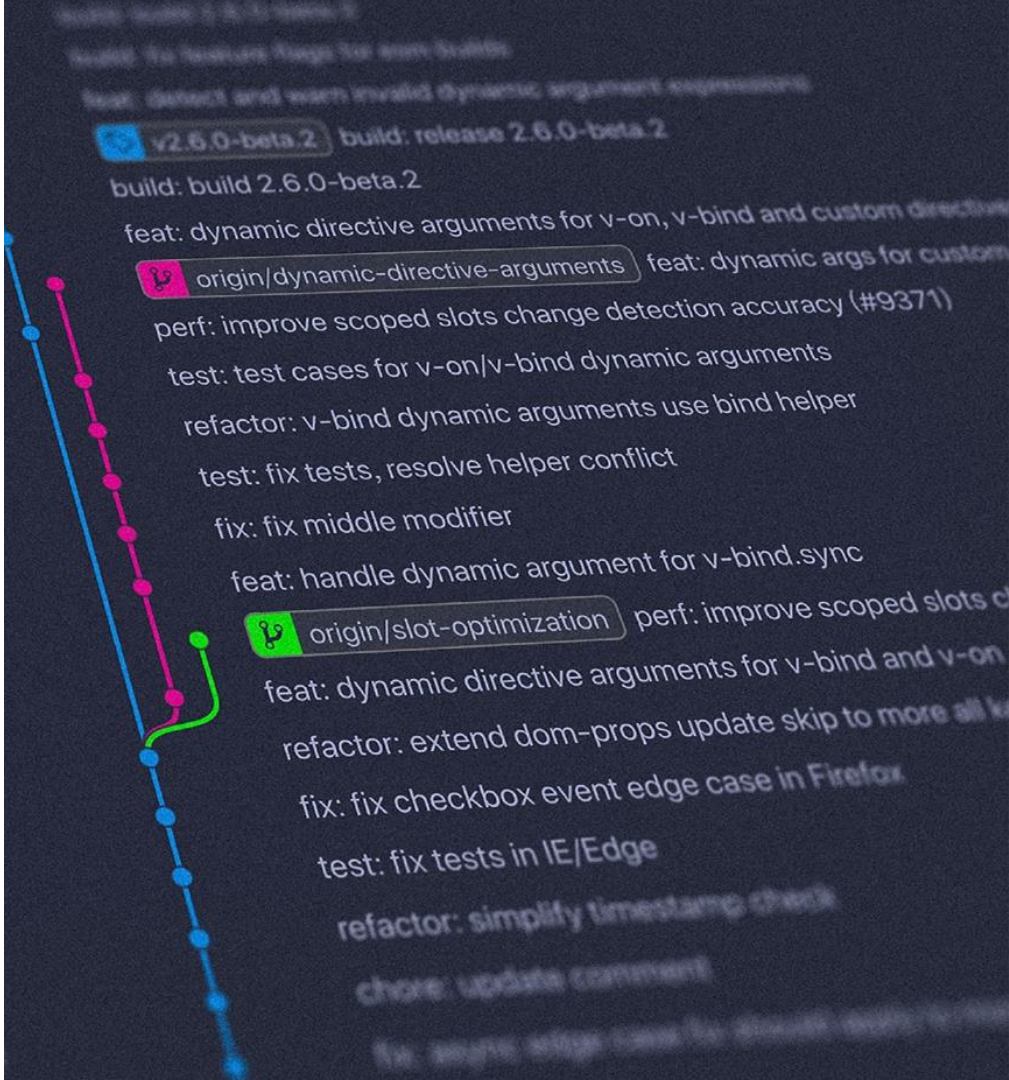
quanto tempo você leva para “aprender” sobre
o repositório de código que você trabalha?

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

O que é análise de código?

Por que analisar código é importante?

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



Quais habilidades precisamos desenvolver?

- Conhecer **heurísticas de análise para as estruturas**
  - Módulos/Pacotes, Classes, Métodos
- Compreender aspectos de **qualidade de software**
  - Atributos externos e internos

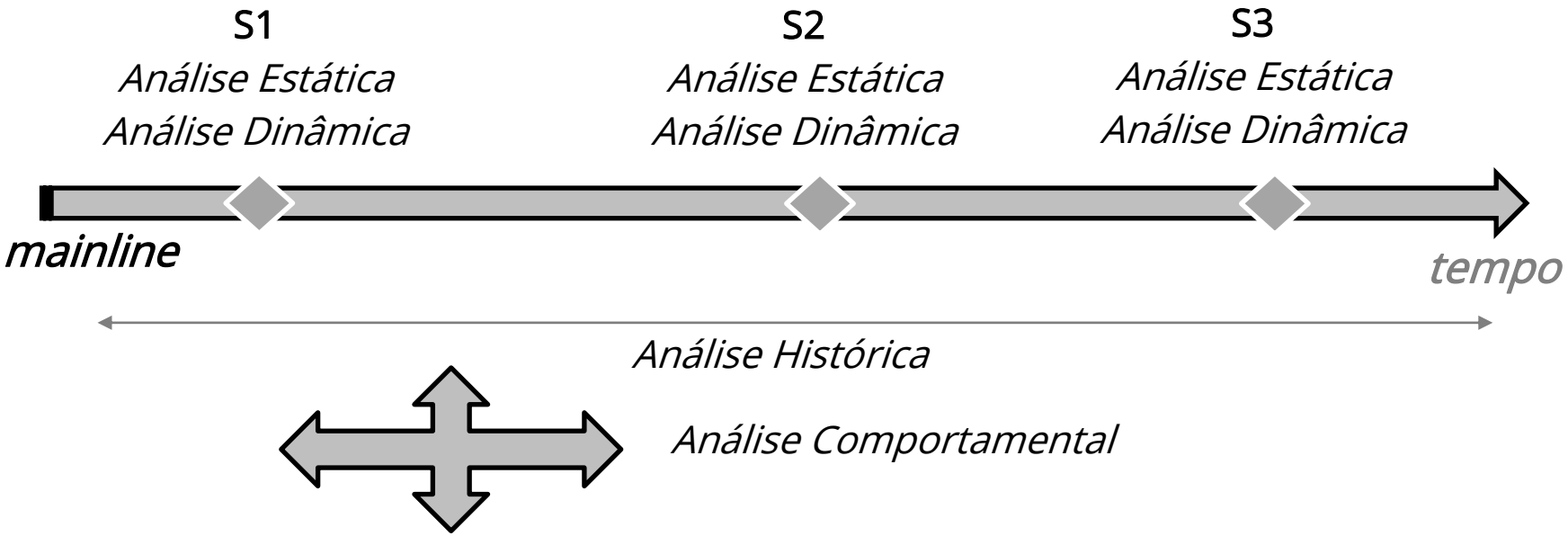
Quais habilidades precisamos desenvolver?

- 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

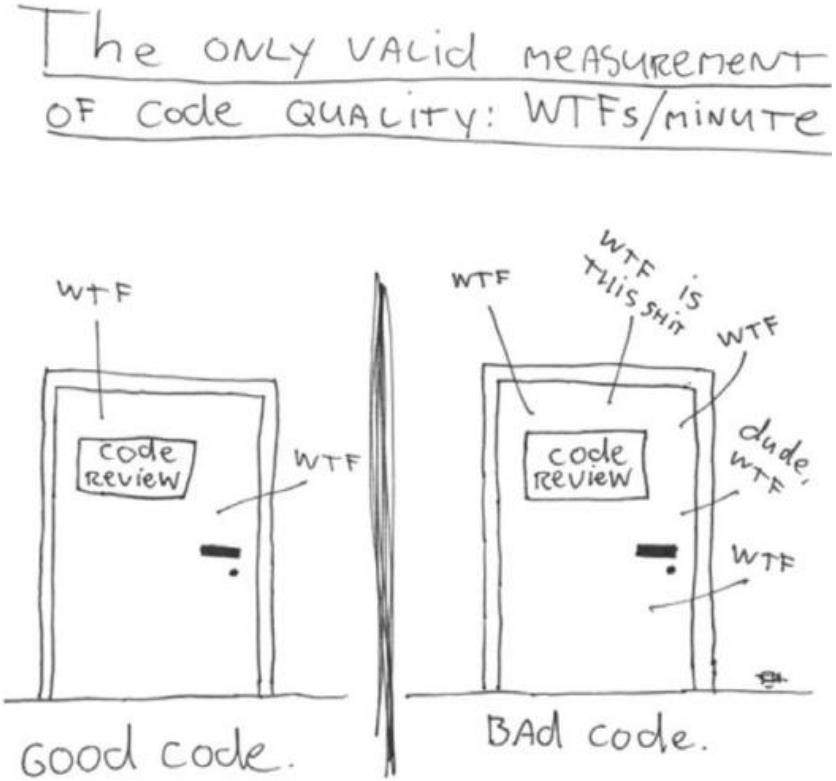
Tipos

- Estática
- Dinâmica
- Histórica
- Comportamental

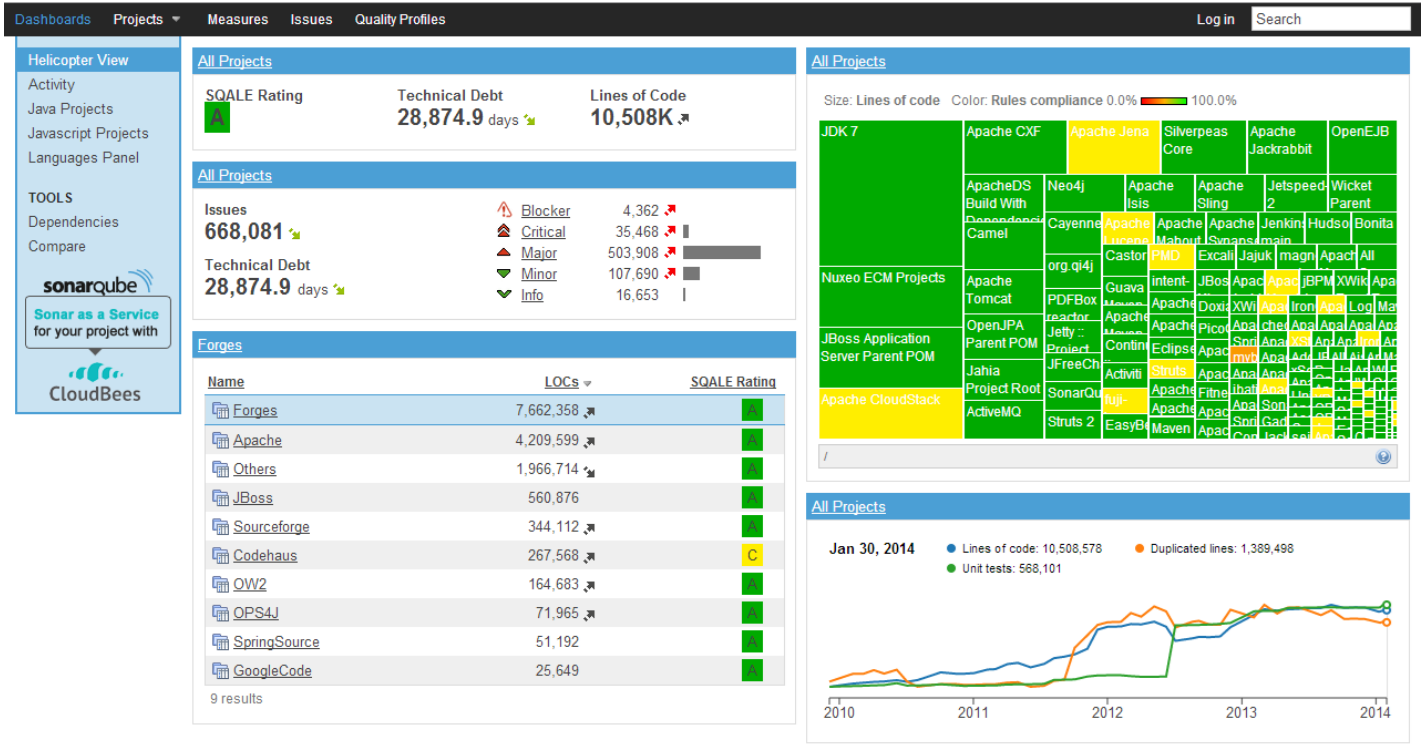
Estratégias Combinadas



Como você avalia a qualidade do seu código?



“ah... Nós temos o SonarQube”



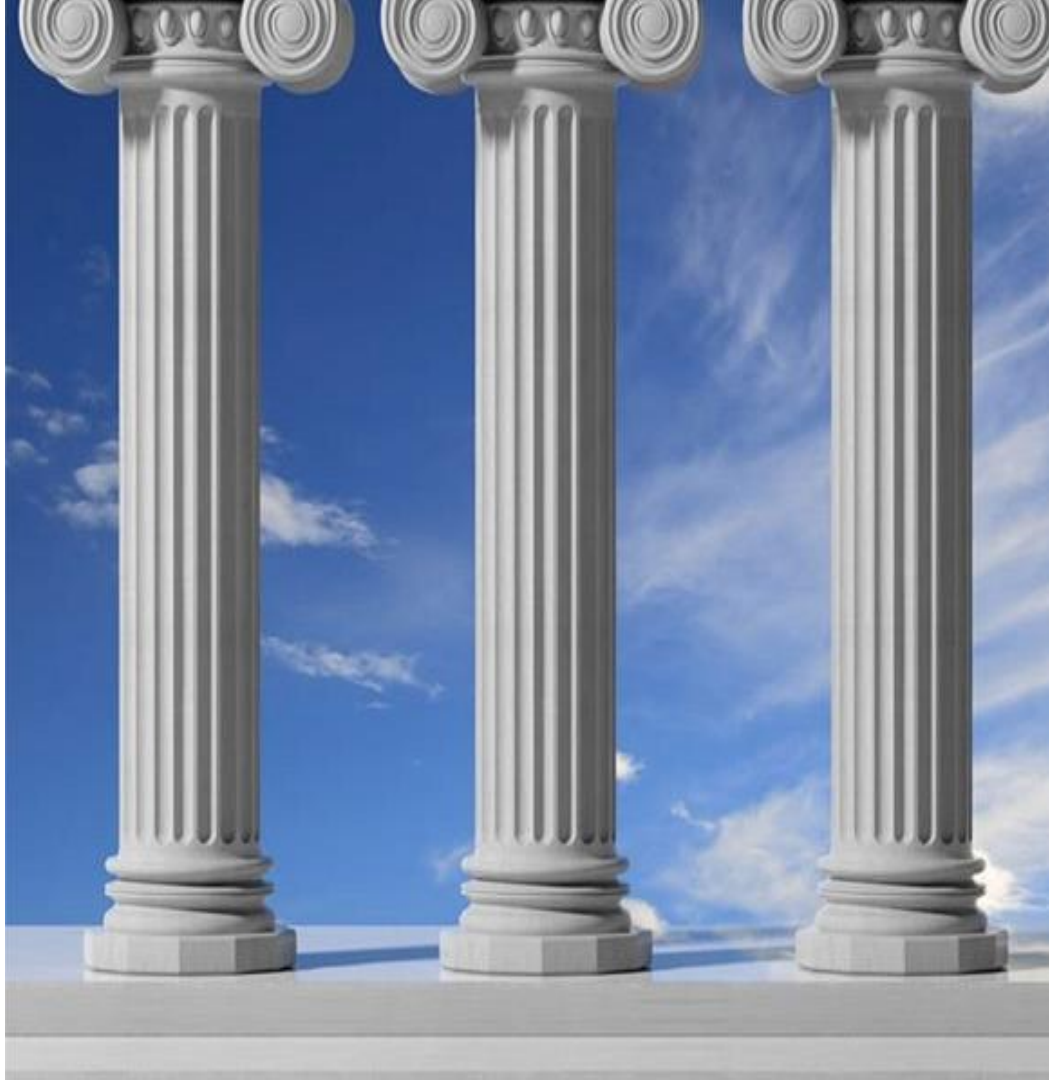
SonarQube™ technology is powered by SonarSource SA
Version 4.1.1 - Community - Documentation - Get Support - Plugins

Better Code Hub

DR-Tools

Pilares da Análise

- Coesão
- Acoplamento
- Tamanho
- Complexidade



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

Um Kata para Análise de Código

- Defina um **objetivo para análise**
- Rode alguma(s) **ferramenta(s) de análise** para encontrar o ponto que **você deseja**
  - Níveis de granularidade
  - Uso de testes automatizados
  - Padrões adotados

Um Kata para Análise de Código

- Você pode começar de **“dentro para fora”**
  - Analise as funções/métodos, suas estruturas e design
  - Considere os pilares
  - Suba a granularidade, quando necessário

Qual o melhor momento para fazer as análises?

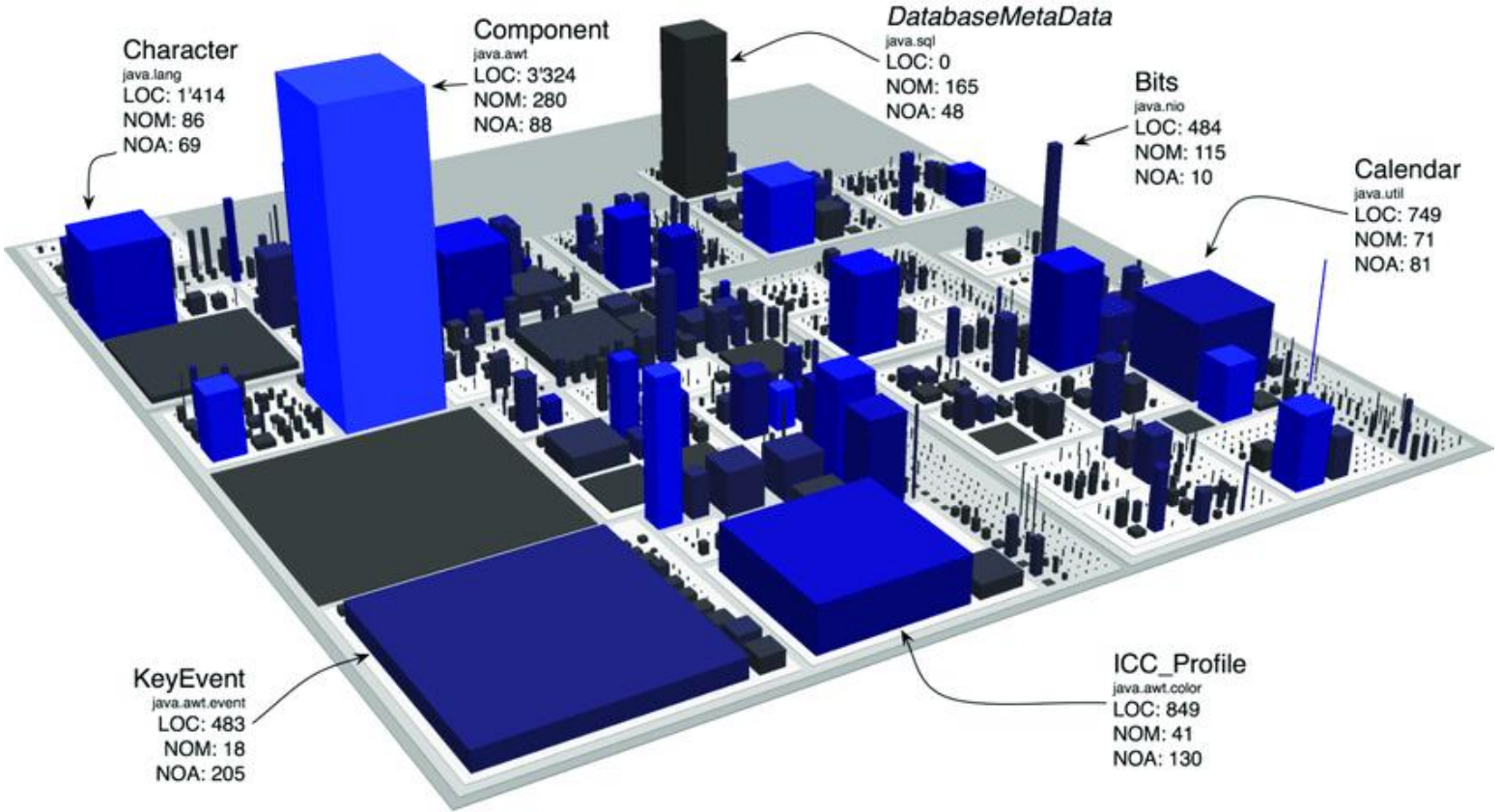
- **Sempre!!**
  - E de forma antecipada e quando possível
- **Individualmente**, antes de fazer commits
  - Inspeção na fonte (Poka-yoke, do Lean)
  - Apoiado por ferramentas (Jidôka, do Lean)
- **Em par**, para discutir situações específicas
- Em sessões de **Code Review**

DR-Tools

A tool quality suite to help the
developers to maintain health and code evolution

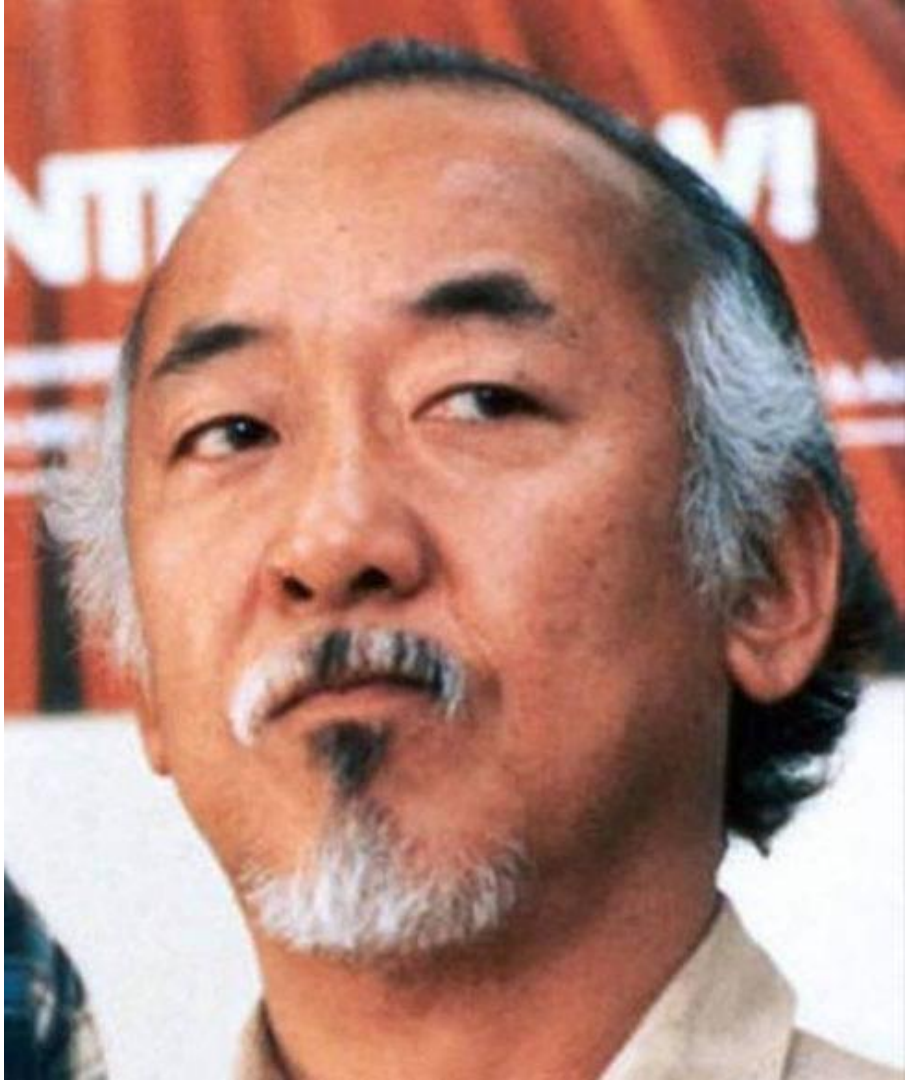
drtools.site

Existem outras metáforas

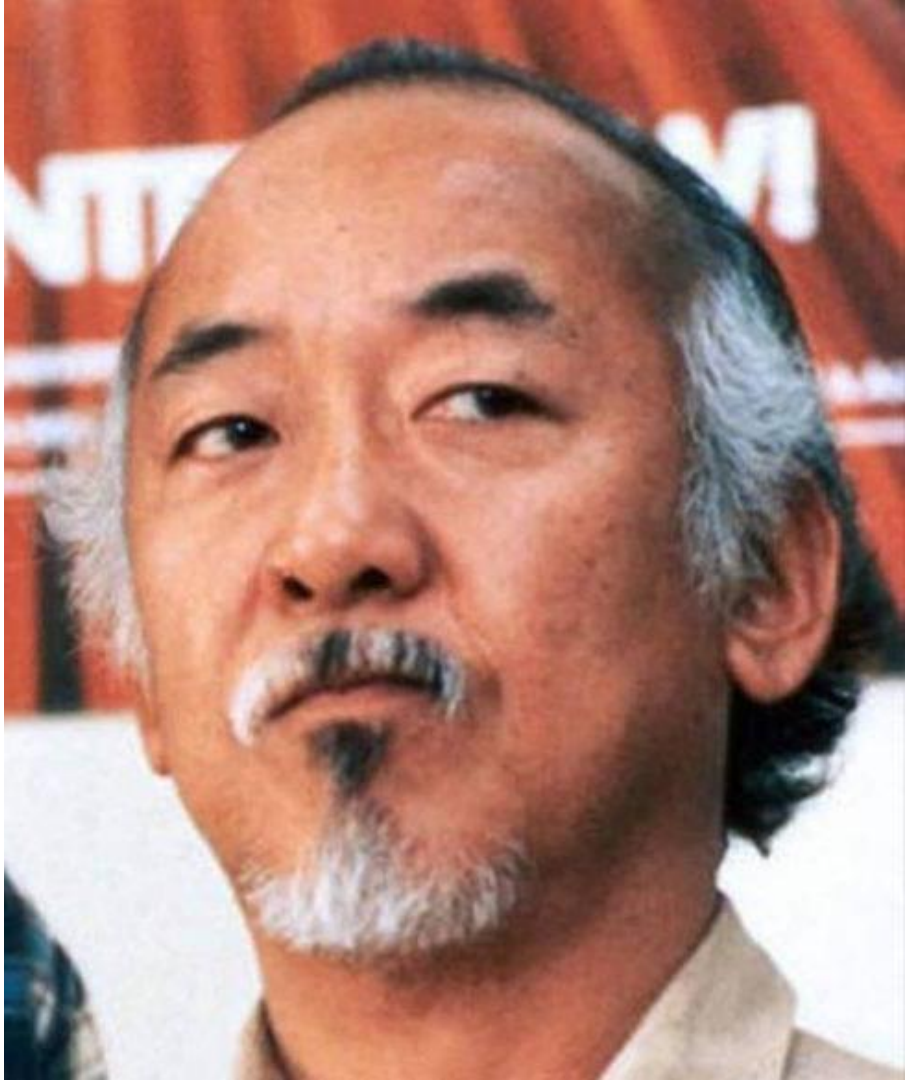


Um episódio de análise de código

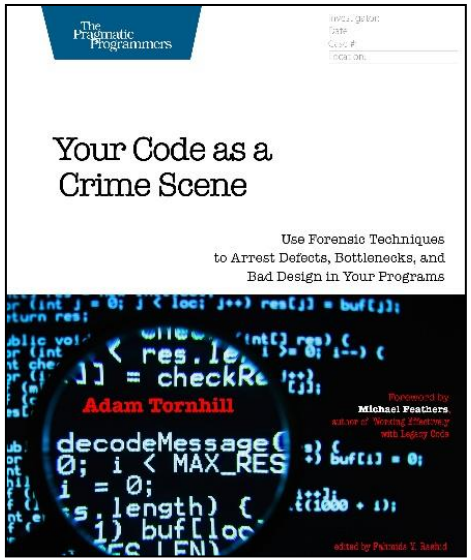
- Regra dos **30 segundos**
- Regra do **Escoteiro**/refatoração **oportunista**
- **Metáfora** do **jornal**
- **Olhe código de outros** (principalmente projetos open-source)
- **Pegou um código da Web**? Ajuste-o
  - *Antes de vincular ao seu repositório reestruture ele ao padrão do time*
- **Conheça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

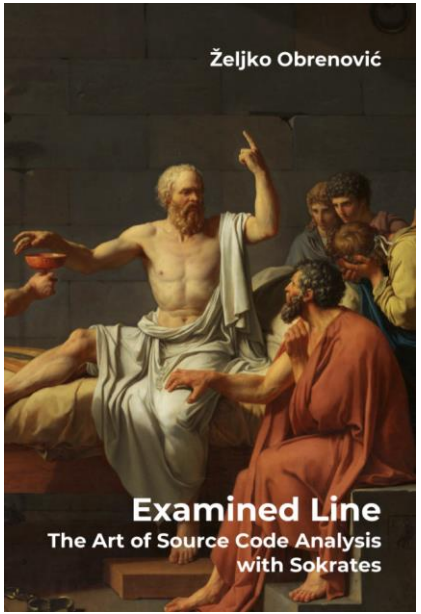


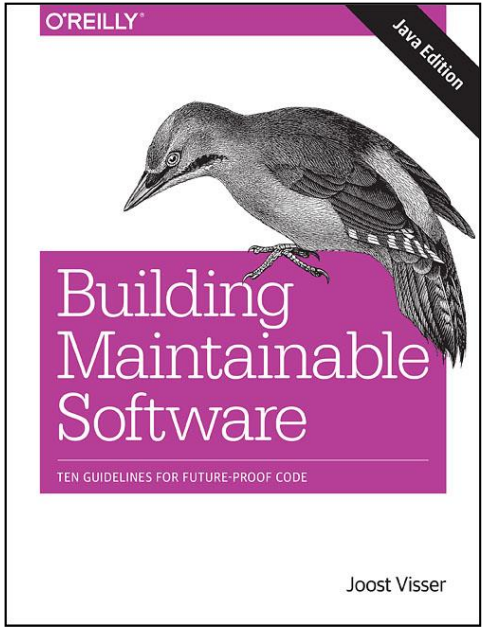
- **Ao analisar** o código, **marque** pontos **para discussão** com o time
  - *Análise de Código X Revisão de Código*
- **Estudem** e **pratiquem**!
- Monte o **plano de metas** com o time
- Criem uma **rotina** com o time para discutir **problemas, práticas e ferramentas**
- **Experimentem** (novas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

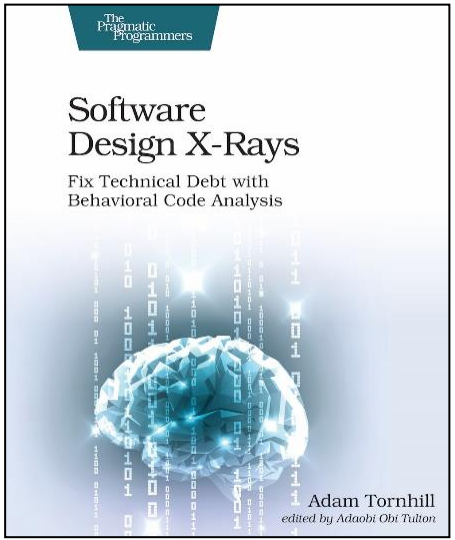


Para aprofundar os estudos...









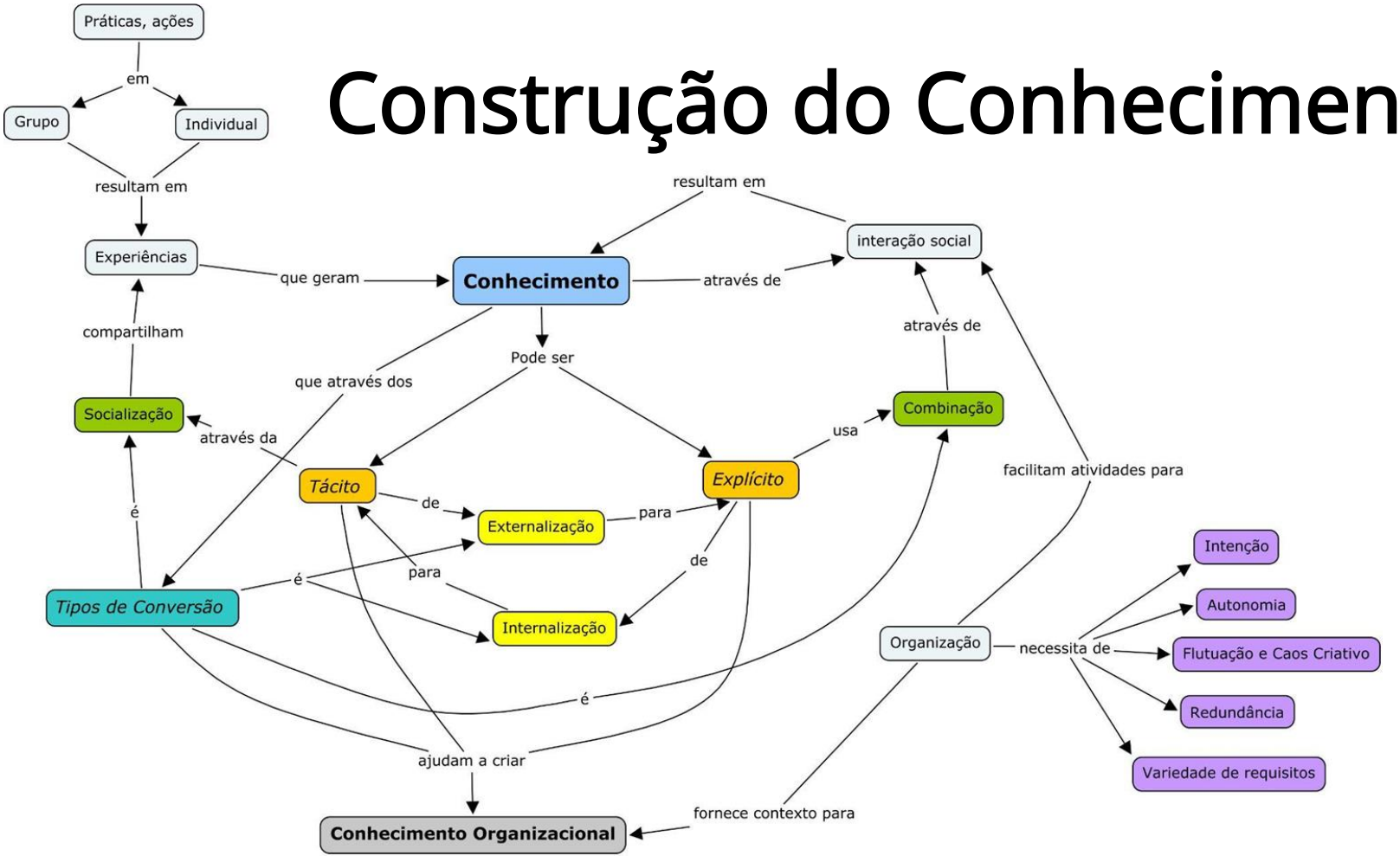
Liderança Técnica

- Um time / Vários times

- Desenvolvimento
  - *técnico do time*
  - *pessoas*
- “Manter a temperatura da água”

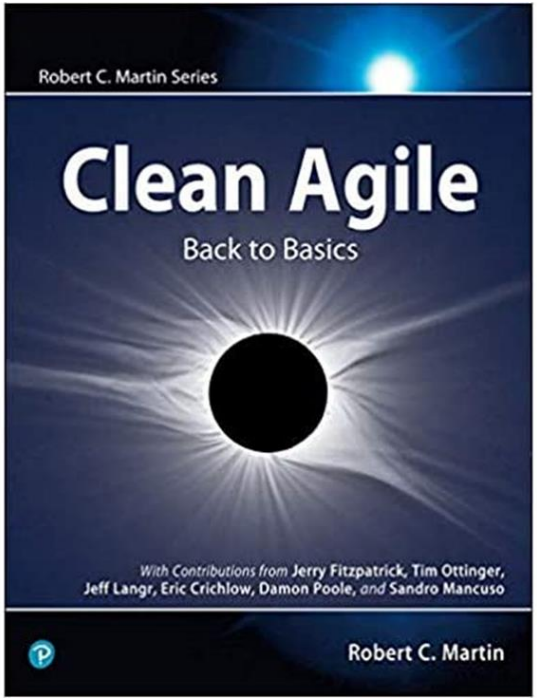
- Gestão do conhecimento

Construção do Conhecimento



Hirotaka Takeuchi e Ikujiro Nonaka, *Gestão do Conhecimento*,Bookman (2008)

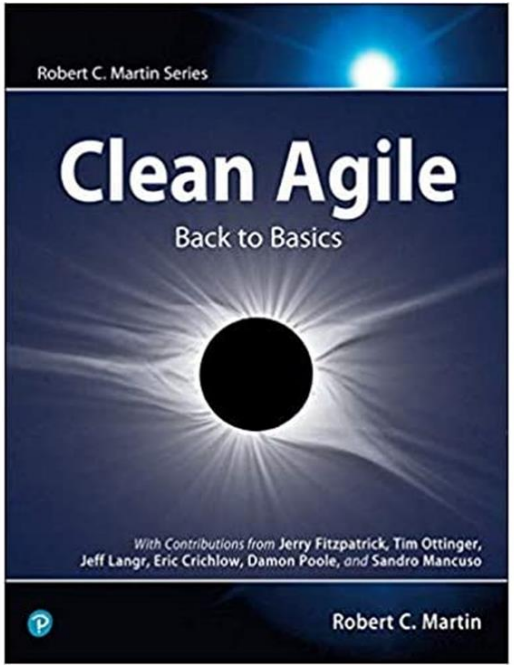
Clean Agile


Robert C. Martin Series
Clean Agile
Back to Basics
With Contributions from Jerry Fitzpatrick, Tim Ottinger, Jeff Langr, Eric Crichlow, Damon Poole, and Sandro Mancuso
Robert C. Martin

“A colaboração entre negócios e tecnologia está mais próxima do que era antes. A identificação dos problemas e dos riscos é feita com antecedência. As empresas estão respondendo de forma mais rápida à medida que obtêm novas informações, realmente colhendo frutos de uma abordagem iterativa de desenvolvimento de software.

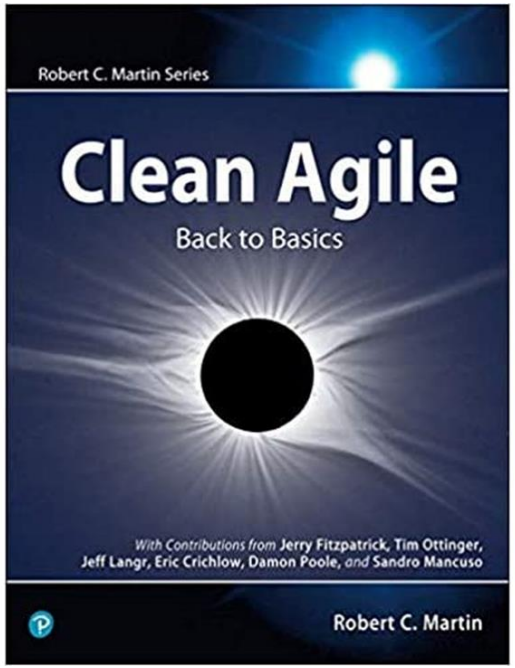
Contudo, embora sejam melhores do que antes, a divisão entre processos e tecnologia ainda as prejudica. Grande parte dos agile coaches modernos não tem habilidades técnicas suficientes (se tiveram alguma) para orientar desenvolvedores em práticas técnicas e raramente falam sobre tecnologia. No decorrer dos anos, os desenvolvedores começaram a enxergar os agile coaches como outra camada de gerenciamento: pessoas lhe dizendo o que fazer em vez de ajudá-los a melhorar o que fazem.

Os desenvolvedores estão se distanciando da agilidade ou a agilidade está se distanciando dos desenvolvedores?”



"A resposta para essa pergunta provavelmente é: as duas coisas. Ao que tudo indica, a agilidade e os desenvolvedores estão se distanciando. Em muitas organizações, a metodologia ágil é sinônimo de Scrum. XP, quando existe, é reduzida a algumas práticas técnicas como TDD e Integração Contínua.

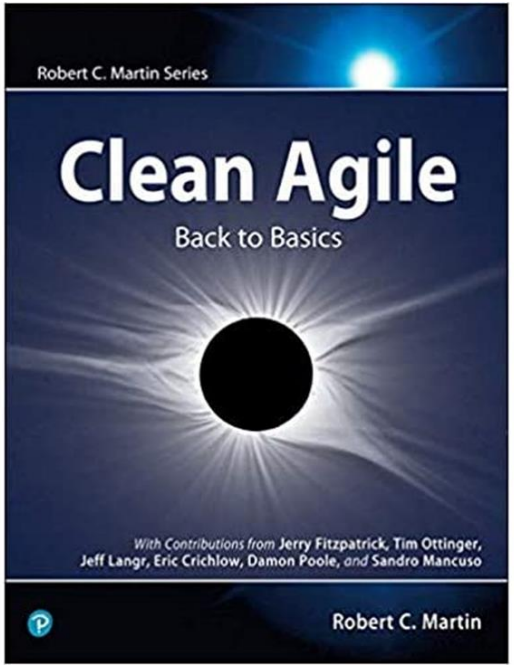
Os agile coaches esperam que os desenvolvedores usem as práticas XP, mas não ajudam nem um pouco ou se intrometem na maneira como os desenvolvedores estão trabalhando. Muitos POs ainda não se sentem parte da equipe e não se sentem responsáveis quando as coisas não saem conforme planejado. Os desenvolvedores ainda precisam negociar bastante com a empresa a fim de efetuarem as melhorias técnicas necessárias para continuar desenvolvendo e fazendo a manutenção do sistema.



As empresas ainda não estão maduras o bastante para entender que os problemas técnicos são, na verdade, problemas de negócios."

“Diferentemente dos agile coaches do início dos anos 2000, que tinham uma sólida formação técnica, a maioria dos agile coaches da atualidade não conseguem ensinar as práticas XP ou conversar com as pessoas de negócio sobre tecnologia.

Eles não conseguem se sentar com os desenvolvedores e programar em pares; não conseguem falar sobre design simples ou ajudar na configuração de pipelines de integração contínua. Eles não conseguem ajudar os desenvolvedores a refatorar seu código legado, analisar estratégias de teste e nem sustentar vários serviços em produção. Eles não conseguem explicar para as pessoas de negócios os verdadeiros benefícios de determinadas práticas técnicas, muito menos elaborar ou aconselhar sobre uma estratégia técnica.”



Fim Parte 4


PUCRS online
uol edtech