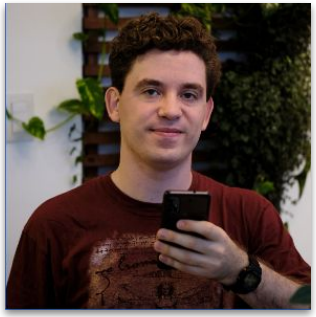
Clean code já era!

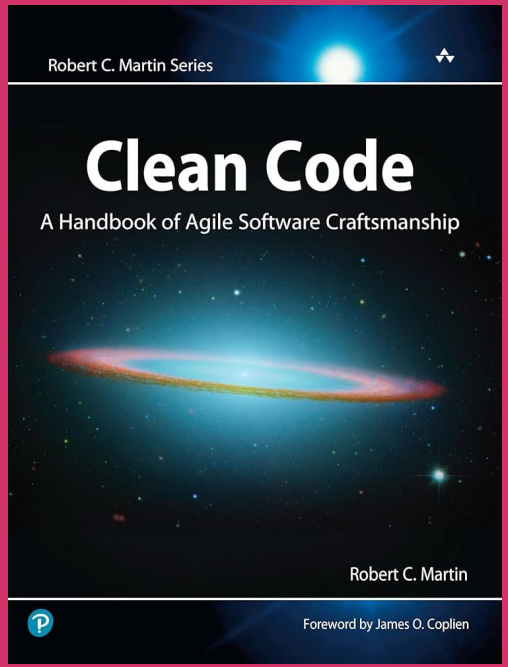
Lucas Rogério Masotti



Lucas Rogério Masotti

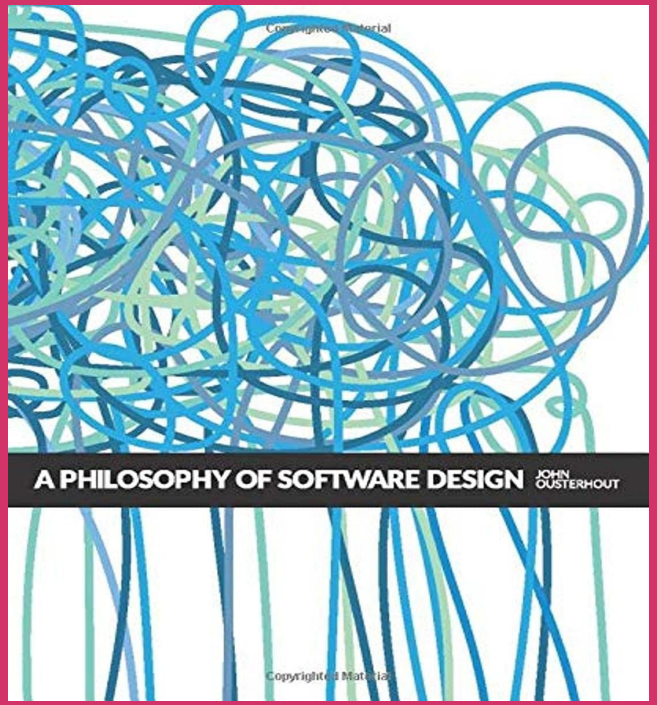
- 12+ anos de experiência, principalmente backend e setor financeiro
- Staff Engineer Meios de Pagamentos - Sicredi
- Vice coordenador do ARGU
- Pós UX, MBA Gestão Marketing e pós Gestão de Negócios
- **Arquitetura, Design Organizacional e Diminuição de Complexidade Acidental**

Quem aqui já ouviu falar do Clean Code?



Quem aqui já ouviu falar de Clean Code como sinônimo de código de qualidade?

E do Philosophy of Software Design?



Software Engineering/Programming
Clean Code
PRENTICE HALL
PEARSON EDUCATION
$42.99 US • $46.99 CANADA
Martin
Clean Code
PRENTICE HALL
Robert C. Martin Series
PRENTICE HALL
Clean Code
A Handbook of Agile Software Craftsmanship
Robert C. Martin
nick HD

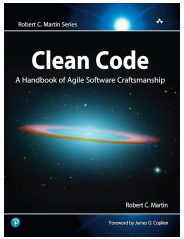
Cargo culting

- Seguir Clean Code à risca, como se fosse mágico
- Não considerar outras fontes e autores

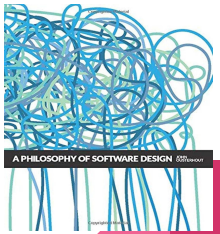
Algumas recomendações são realmente nocivas

Comentários

“comments are, at best, a **necessary evil**. (...) We must have them because we cannot always figure out how to express ourselves without them, but **their use is not a cause for celebration**”

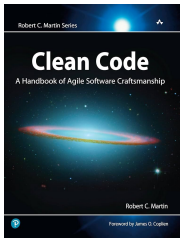


“In-code documentation plays a **crucial role** in software design. Comments are **essential to help developers understand a system and work efficiently** (..)”

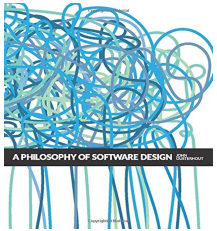


- Comentários compensam nossa falha de nos expressar em código
- Comentários mentem, ficam desatualizados
- Comentários imprecisos são piores que nenhum comentário

Chapter 17: Smells and Heuristics 285
Comments 286
C1: *Inappropriate Information* 286
C2: *Obsolete Comment* 286
C3: *Redundant Comment* 286
C4: *Poorly Written Comment* 287
C5: *Commented-Out Code* 287

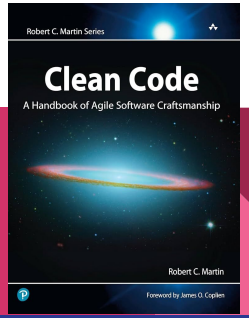


- Quatro desculpas para não escrever comentários
  - Bom código é auto documentado
  - Eu não tenho tempo para escrever comentários
  - Comentários ficam desatualizados e se tornam enganosos.
  - Os comentários que vi são todos inúteis; por que me dar ao trabalho?

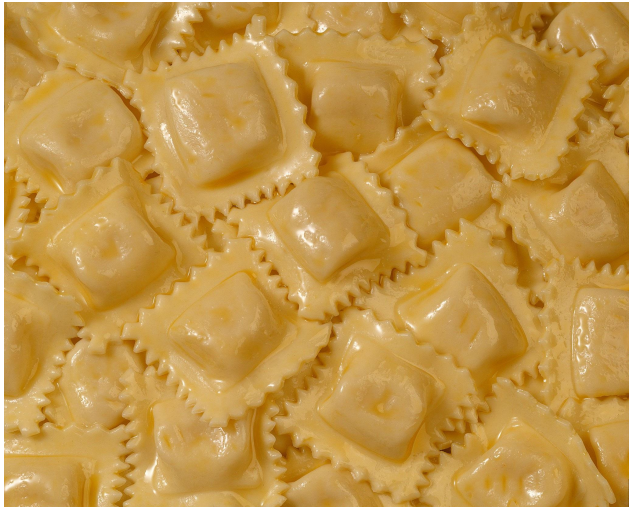


Síndrome do Quanto Menor, Melhor

- Arquivos pequenos:
  - “It appears to be possible to build significant systems (...) out of files that are typically 200 lines long, with an upper limit of 500”
- Classes pequenas:
  - “The first rule of classes is that they **should be small.** The second rule of classes is that they should be **smaller than that**”
- Funções pequenas:
  - “The first rule of functions is that they **should be small.** The second rule of functions is that they should be **smaller than that.**”



Do spaghetti ao ravioli



```
InscricaoController
└── InscricaoService
    ├── InscricaoValidatorService
    │   ├── EventoRepository
    │   └── InscricaoRepository
    ├── PagamentoService
    │   ├── PagamentoRequestBuilder
    │   ├── GatewayPagamentoClient
    │   └── PagamentoResponseParser
    ├── InscricaoRepository
    ├── InscricaoProducerService
    │   ├── InscricaoEventFactory
    │   └── DomainEventPublisher
    └── InscricaoResponseMapper
```

```java
@RestController
@RequestMapping("/inscricoes")
class InscricaoController {
    private final InscricaoService service;
    InscricaoController(InscricaoService service) { this.service = service; }

    @PostMapping
    ResponseEntity<InscricaoResponse> inscrever(@RequestBody InscricaoRequest req) {
        return ResponseEntity.ok(service.inscrever(req));
    }
}
```

```java
// ── Service que só orquestra outros services ──────────────
@Service
class InscricaoService {
    private final InscricaoValidatorService validator;
    private final PagamentoService pagamento;
    private final InscricaoRepository repository;
    private final InscricaoProducerService producer;
    private final InscricaoResponseMapper responseMapper;

    InscricaoResponse inscrever(InscricaoRequest req) {
        validator.validar(req);
        var resultado = pagamento.cobrar(req);
        var inscricao = repository.save(
            new Inscricao(req.eventoId(), req.email(), resultado.transactionId()));
        producer.publicar(inscricao);
        return responseMapper.toResponse(inscricao);
    }
}
```

```java
@Service
class InscricaoValidatorService {
    private static final Pattern EMAIL = Pattern.compile("^[^@\\s]+@[^@\\s]+\\.[^@\\s]+$");
    private final EventoRepository eventos;
    private final InscricaoRepository inscricoes;

    void validar(InscricaoRequest req) {
        if (req.eventoId() == null) throw new CampoObrigatorioException("eventoId");
        if (req.email() == null)    throw new CampoObrigatorioException("email");
        if (!EMAIL.matcher(req.email()).matches())
            throw new EmailInvalidoException(req.email());
        if (!eventos.existsById(req.eventoId()))
            throw new EventoNaoEncontradoException(req.eventoId());
        int vagas = eventos.capacidade(req.eventoId()) - inscricoes.contarPorEvento(req.eventoId());
        if (vagas <= 0)
            throw new SemVagasException(req.eventoId());
    }
}
```

```
@Service
class PagamentoService {
    private final PagamentoRequestBuilder builder;
    private final GatewayPagamentoClient client;
    private final PagamentoResponseParser parser;
    // ctor omitido...
    ResultadoPagamento cobrar(InscricaoRequest req) {
        var payload  = builder.build(req);
        var resposta = client.enviar(payload);
        return parser.parse(resposta);
    }
}
```

```
@Component
class PagamentoRequestBuilder {
    GatewayRequest build(InscricaoRequest req) {
        return new GatewayRequest(req.email(), req.valo
    }
}

@Component
class PagamentoResponseParser {
    ResultadoPagamento parse(GatewayResponse resp) {
        return new ResultadoPagamento(resp.transactionI
    }
}
```

```java
// ── Publicação ──────────────────────────────────────────
@Service
class InscricaoProducerService {
    private final InscricaoEventFactory eventFactory;
    private final DomainEventPublisher publisher;
    // ctor omitido...
    void publicar(Inscricao inscricao) {
        publisher.publish(eventFactory.criar(inscricao));
    }
}

@Component
class InscricaoEventFactory {
    InscricaoRealizadaEvent criar(Inscricao i) {
        return new InscricaoRealizadaEvent(i.getId(), i.getEventoId(), i.getEmail());
    }
}
```

```
InscricaoController
└── InscricaoService
    ├── EventoRepository
    ├── InscricaoRepository
    ├── GatewayPagamento
    └── ApplicationEventPublisher
```

```java
@Service
class InscricaoService {
    private static final Pattern EMAIL =
        Pattern.compile("^[^@\\s]+@[^@\\s]+\\.[^@\\s]+$");

    private final EventoRepository eventos;
    private final InscricaoRepository inscricoes;
    private final GatewayPagamento pagamento;
    private final ApplicationEventPublisher publisher;

    /**
     * Inscreve um participante em um evento.
     *
     * Valida a request e a disponibilidade de vagas, cobra o pagamento no
     * gateway externo, persiste a inscrição e publica o evento de domínio.
     * Em caso de qualquer regra violada, lança {@link InscricaoInvalidaException}
     * e nada é cobrado nem persistido (transação revertida).
     */
    @Transactional
    Inscricao inscrever(InscricaoRequest req) {
        var evento = validarEObterEvento(req);

        var resultado = pagamento.cobrar(req.email(), evento.valorInscricao());
        var inscricao = inscricoes.save(
            Inscricao.nova(evento.id(), req.email(), req.nome(), resultado.transactionId()));

        publisher.publishEvent(
            new InscricaoRealizada(inscricao.id(), evento.id(), req.email()));
        return inscricao;
    }
```

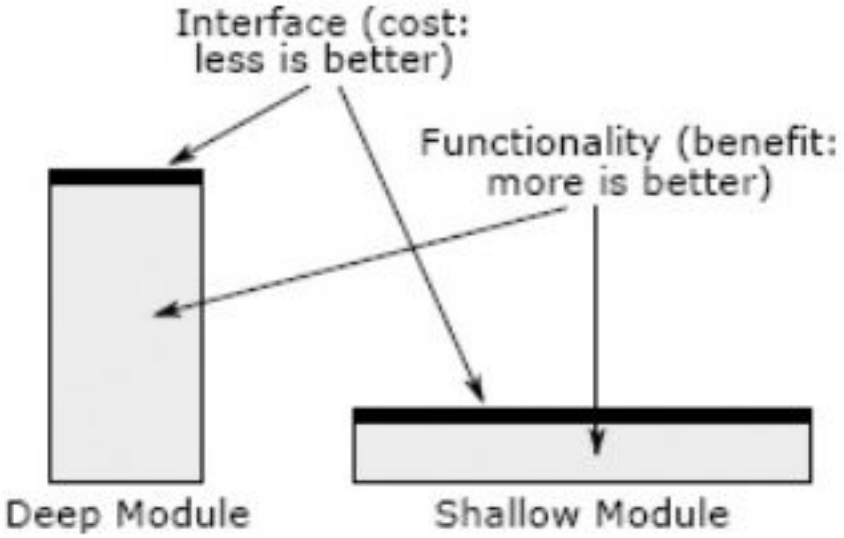
```
    private Evento validarEObterEvento(InscricaoRequest req) {
        if (req.eventoId() == null || isBlank(req.email()))
            throw new InscricaoInvalidaException("eventoId e email são obrigatórios");
        if (!EMAIL.matcher(req.email()).matches())
            throw new InscricaoInvalidaException("email inválido: " + req.email());

        var evento = eventos.findById(req.eventoId())
            .orElseThrow(() -> new InscricaoInvalidaException("evento não encontrado"));
        if (inscricoes.contarPorEvento(evento.id()) >= evento.capacidade())
            throw new InscricaoInvalidaException("evento sem vagas");
        return evento;
    }
}
```

	Ravióli 🍝 (Clean Code ao extremo)	Módulo profundo 🥩 (Ousterhout)
Nº de classes	~8	2 (+ colaboradores de I/O)
Ler uma inscrição	pular por 8 classes	ler 1 método
Complexidade total	**maior** (espalhada)	**menor** (concentrada)
Termo do sintoma	*classitis* / classe rasa	módulo profundo

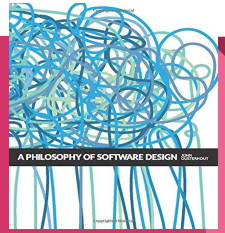
```
InscricaoController
└── InscricaoService
    ├── InscricaoValidatorService
    │   ├── EventoRepository
    │   └── InscricaoRepository
    ├── PagamentoService
    │   ├── PagamentoRequestBuilder
    │   ├── GatewayPagamentoClient
    │   └── PagamentoResponseParser
    ├── InscricaoRepository
    ├── InscricaoProducerService
    │   ├── InscricaoEventFactory
    │   └── DomainEventPublisher
    └── InscricaoResponseMapper
```

```
InscricaoController
└── InscricaoService
    ├── EventoRepository
    ├── InscricaoRepository
    ├── GatewayPagamento
    └── ApplicationEventPublisher
```

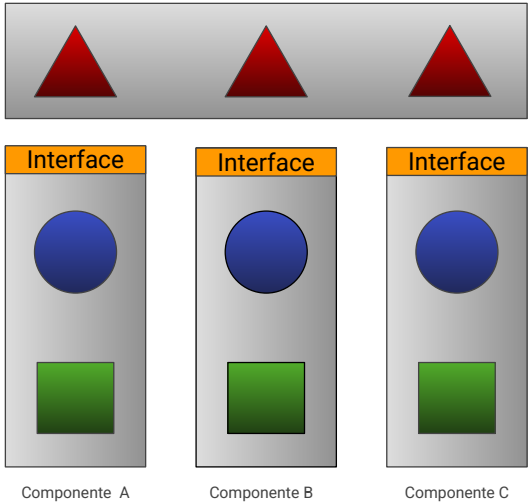
Deep modules para combater *classitis*



“In systems suffering from classitis, developers are encouraged to minimize the amount of functionality in each new class: if you want more functionality, introduce more classes. **Classitis may result in classes that are individually simple, but it increases the complexity of the overall system**”



Package by Component

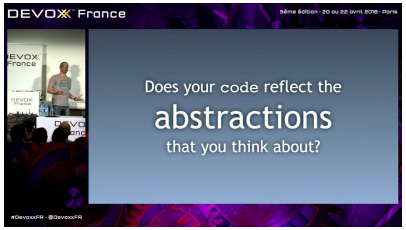


Controller (web)

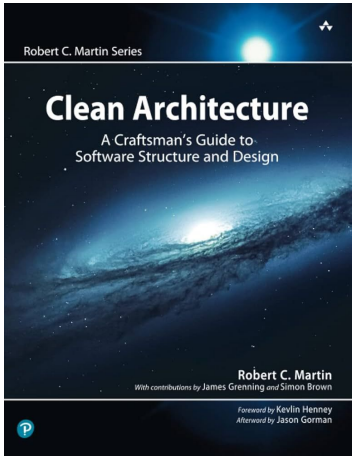
Service (business)

Repository (data)

Modular Monoliths • Simon Brown • GOTO 2018



Modular monoliths (Simon Brown)



Não se aplica apenas à código

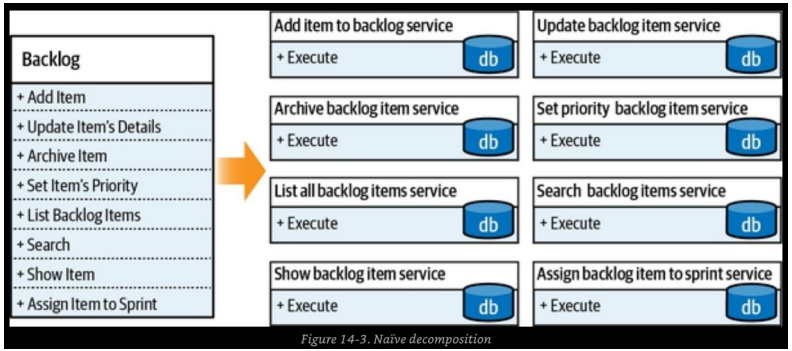


Figure 14-3. Naïve decomposition

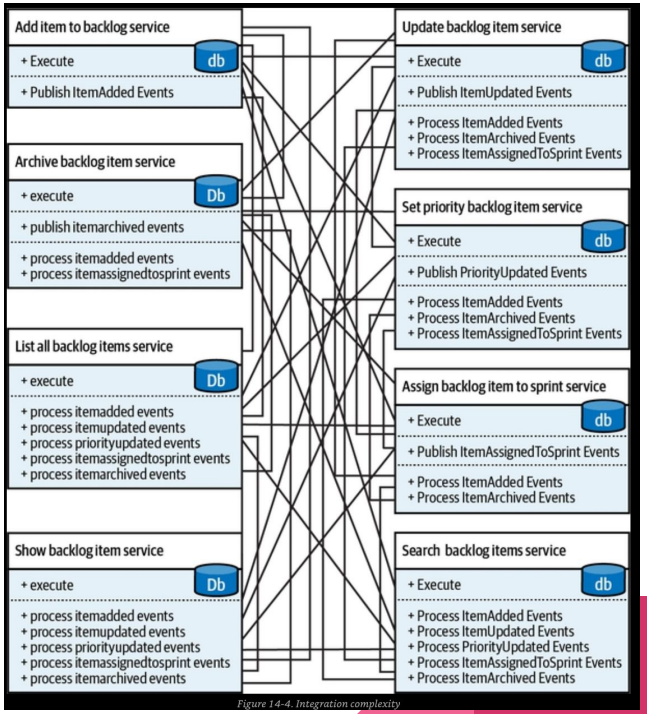
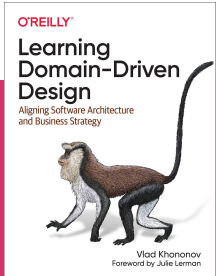


Figure 14-4. Integration complexity



Poucas classes

Complexidade local alta
Complexidade global baixa

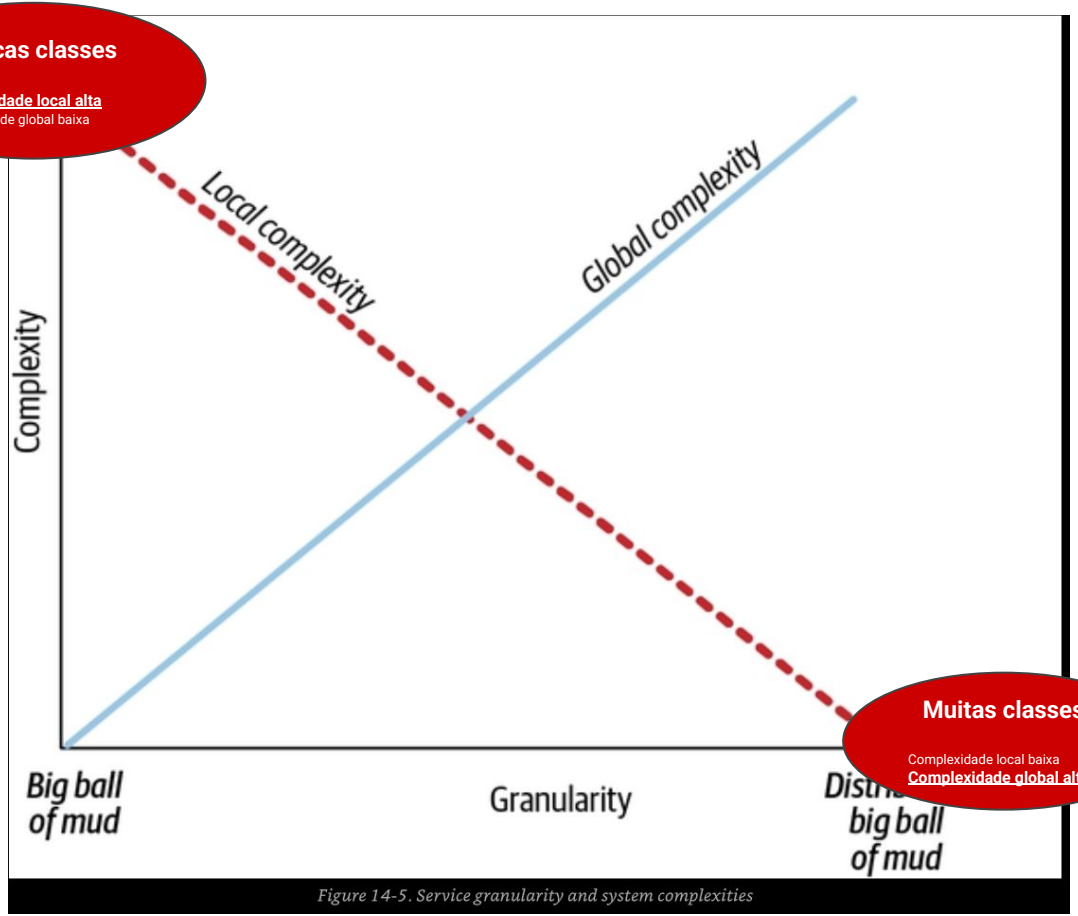
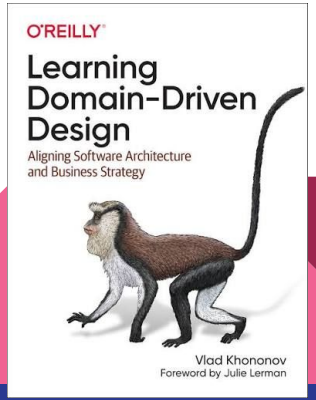


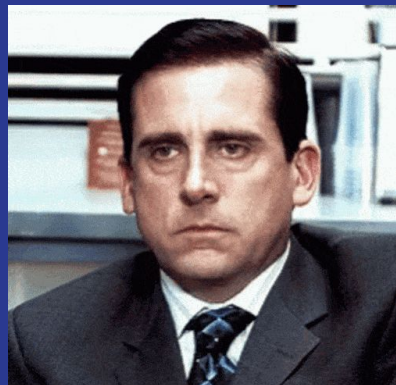
Figure 14-5. Service granularity and system complexities

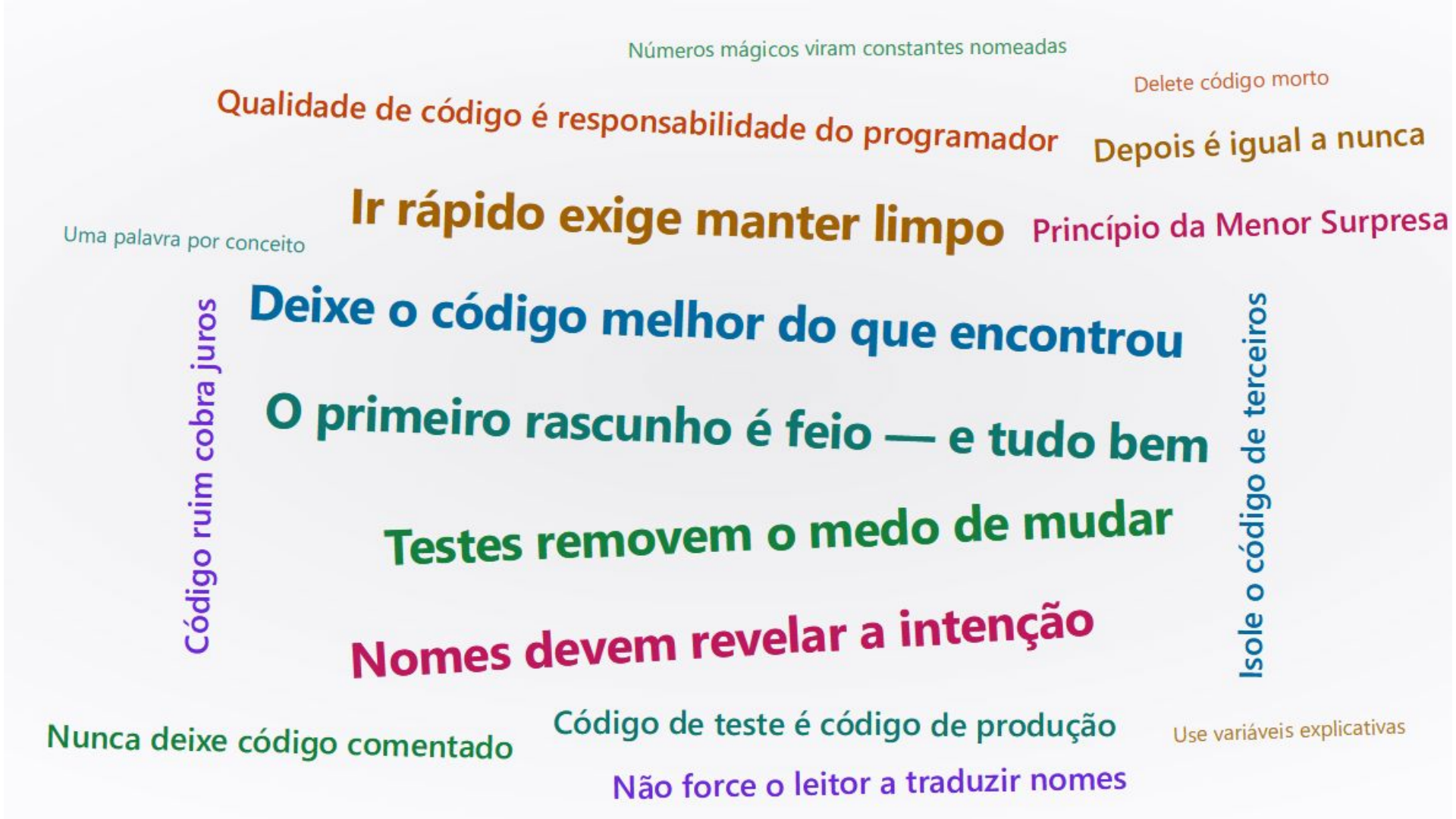
Muitas classes

Complexidade local baixa
Complexidade global alta



Clean code já era?
É tudo ruim?




Números mágicos viram constantes nomeadas
Delete código morto
Qualidade de código é responsabilidade do programador
Depois é igual a nunca
Ir rápido exige manter limpo
Princípio da Menor Surpresa
Uma palavra por conceito
Deixe o código melhor do que encontrou
Código ruim cobra juros
O primeiro rascunho é feio — e tudo bem
Isole o código de terceiros
Testes removem o medo de mudar
Nomes devem revelar a intenção
Nunca deixe código comentado
Código de teste é código de produção
Use variáveis explicativas
Não force o leitor a traduzir nomes

README

A Philosophy of Software Design vs Clean Code

(This document is the result of a series of discussions, some online and some in person, held between Robert "Uncle Bob" Martin and John Ousterhout between September 2024 and February 2025)

Related Links

- Follow-up Discussion with John Ousterhout and Robert Martin on the Book Overflow Podcast
- A Philosophy of Software Design, 2nd Edition, by John Ousterhout
- Clean Code: A Handbook of Agile Software Craftsmanship, by Robert C. Martin

Discussion Topics

- Method Length
  - Method Length Summary
- Comments
  - Comments Summary
- John's Rewrite of PrimeGenerator
  - A Tale of Two Programmers
- Bob's Rewrite of PrimeGenerator2

https://github.com/johnousterhout/aposd-vs-clean-code

UB:

I hope you agree that between these two examples, the former is a bit better.

```
public String makeStatement() {
    clearTotals();
    return makeHeader() + makeRentalDetails() + makeFooter();
}
```

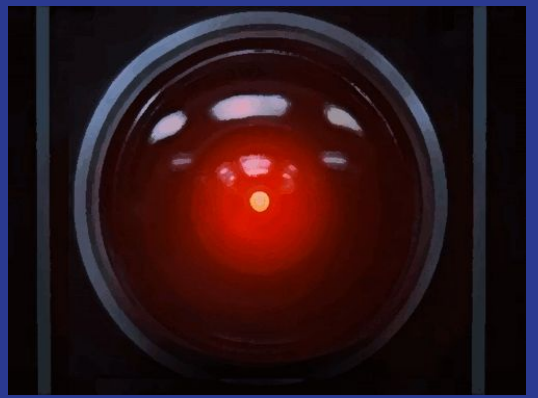
```
public String makeStatement() {
    amountOwed=0;
    totalPoints=0;
    return makeHeader() + makeRentalDetails() + makeFooter();
}
```

JOHN:

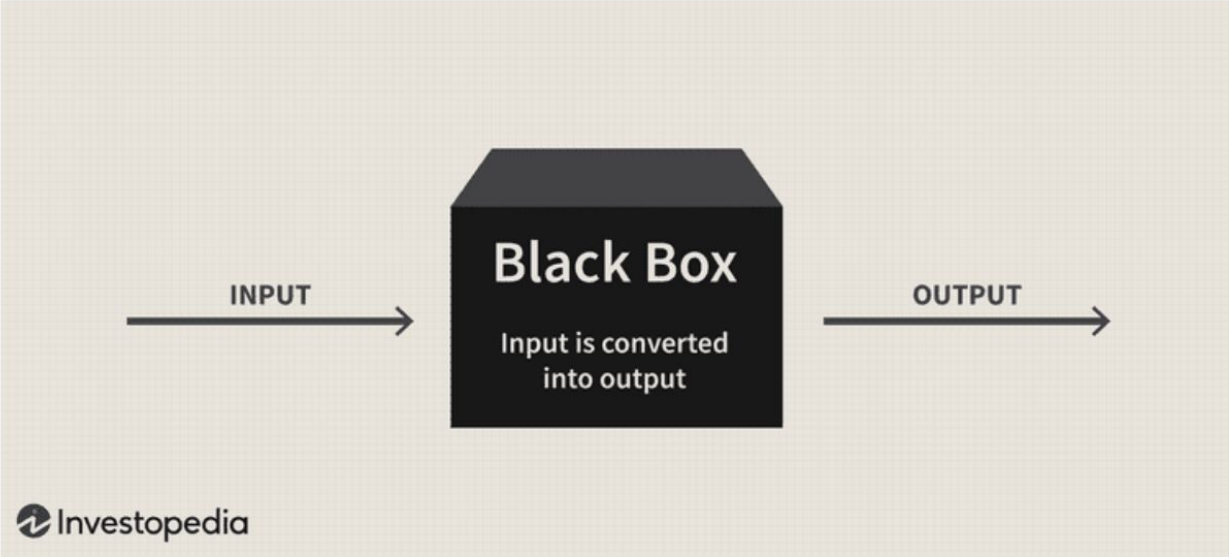
Well, actually, no. The second example is completely clear and obvious: I don't see anything to be gained by splitting it up.

SPOCK (a.k.a UB):

E na era das LLMs?

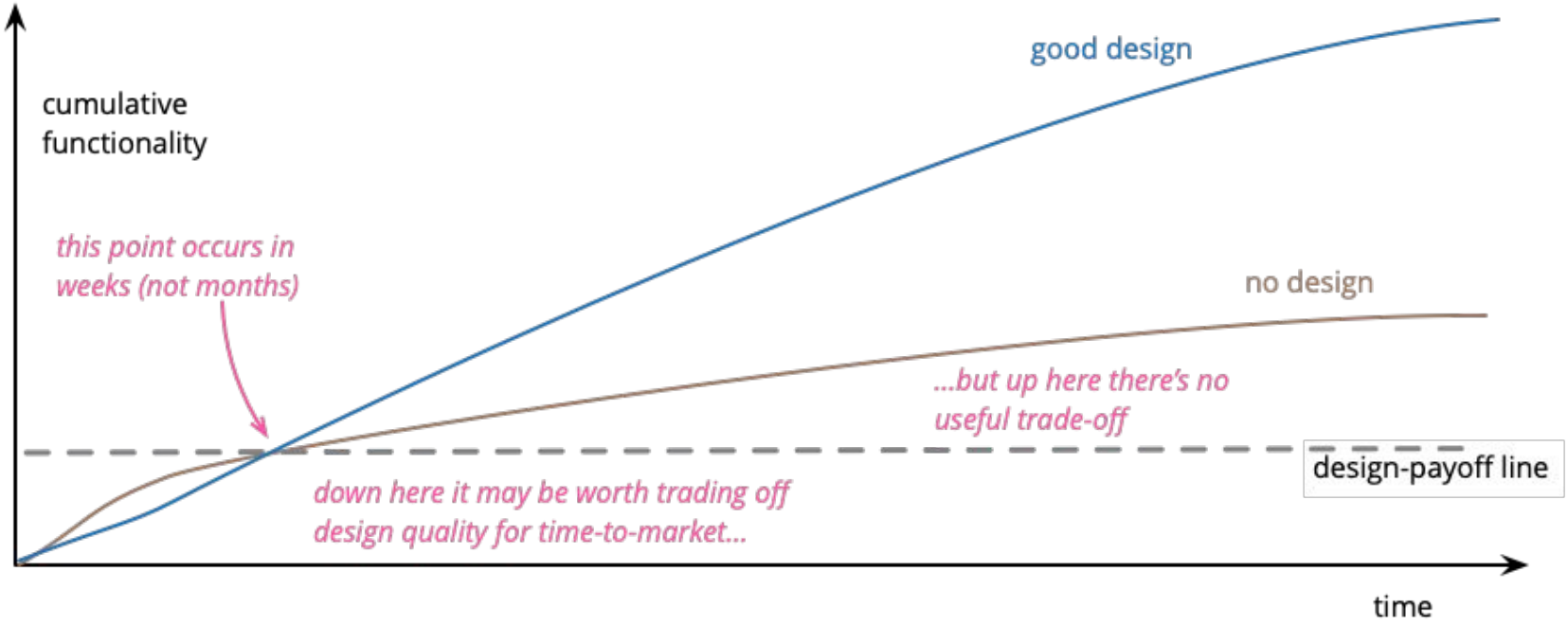


Ainda faz sentido olhar para código?



Credit: Image by Julie Bang © Investopedia 2019

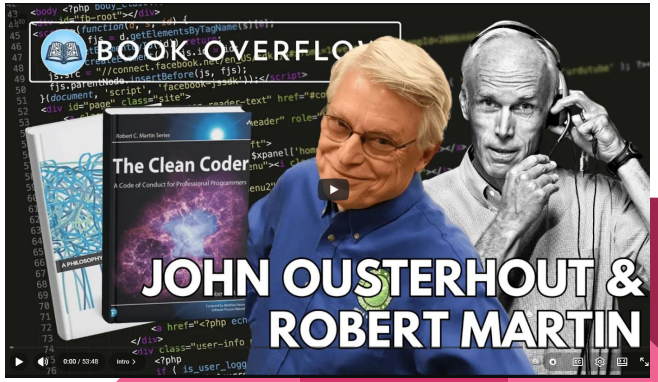
E o quê é melhor para os agentes?



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

LUROMA

Não seja um engenheiro de um livro só

Conclusões e material complementar

- **Livros/frameworks são ferramentas, não verdades absolutas**
- Considere ler *A Philosophy of Software Design*
- Pense mais sobre coesão e modularidade, não tanto sobre tamanho