2026

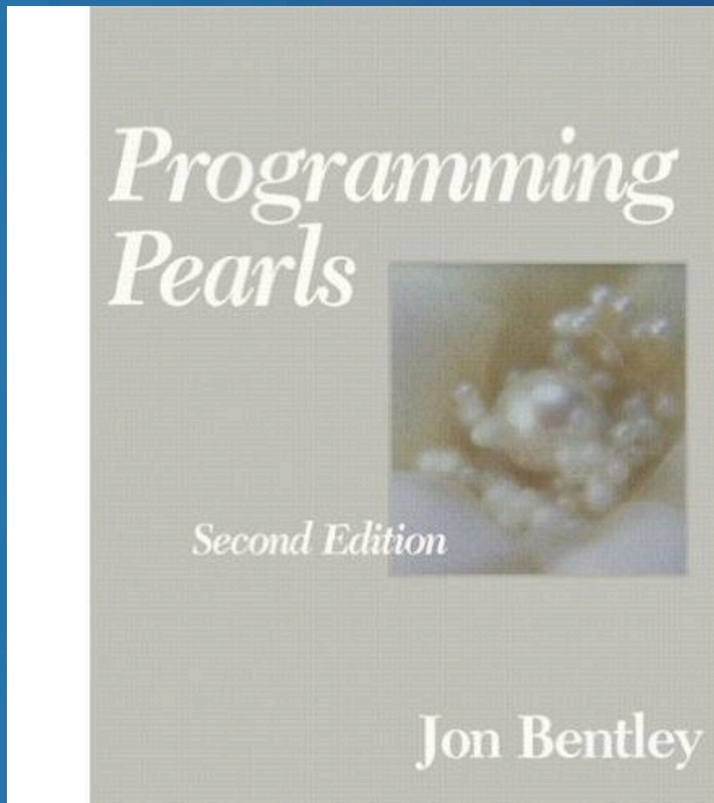
UNIVERSIDADE VILA VELHA /
TEORIA DA COMPUTAÇÃO

ANDRÉ LUIZ GOMES
THOMÁS KRIGER

PROGRAMMING PEARLS

COLUMN 5: A SMALL MATTER OF PROGRAMMING

Como transformar um projeto pensado e esboçado em código real e funcional



SOBRE O QUE FALA ESSA COLUNA NO LIVRO?

Trata da ponte entre a ideia e o programa funcionando corretamente. As medidas tomadas a partir da ideação de um código utilizando de medições, testes, verificações e depurações.

- **Engineering gap**

DO ALGORITMO AO CÓDIGO REAL

O processo de tranformar uma solução é o momento onde aparecem os erros

Erros não surgem só da lógica,e sim de:

Bentley reforça

Escrever código não é linear

Detalhes ignorados

Suposições erradas

Casos extremos

DETALHES IGNORADOS

- Erros pequenos
- Inicialização
- Índices e limites
- Estados inválidos

```c
int sum(int v[], int n) {
    int i;
    int total = 0;

    for (i = 0; i <= n; i++) {   // ERRO AQUI
        total += v[i];
    }
    return total;
}
```

SUPOSIÇÕES ERRADAS

- Suposições implícitas
- Entrada “bem comportada”
- Estados assumidos
- Falta de validação

```
double average(int sum, int count) {
    return sum / count;    // suposição:
}
```

CASOS EXTREMOS

- entrada vazia
- entrada máxima
- apenas um elemento
- valores repetidos
- valores fora do esperado

```c
int max(int v[], int n) {
    int i;
    int m = v[0];   // ERRO se n == 0

    for (i = 1; i < n; i++) {
        if (v[i] > m) {
            m = v[i];
        }
    }
    return m;
}
```

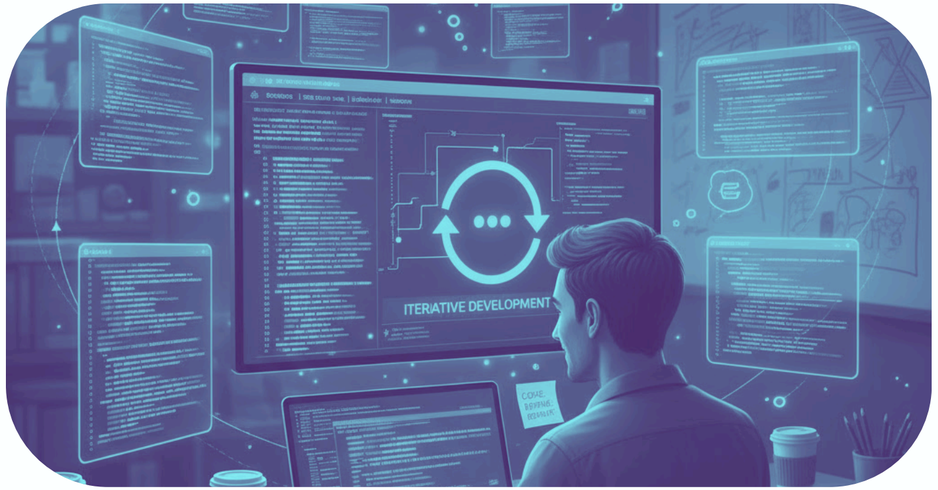
EXEMPLO CLÁSSICO DE ERRO SUTIL

BUSCA BINÁRIA

```c
int binary_search(int arr[], int n, int target) {
    int low = 0;
    int high = n - 1;

    while (low <= high) {
        int mid = (low + high) / 2;

        if (arr[mid] == target)
            return mid;
        else if (arr[mid] < target)
            low = mid + 1;
        else
            high = mid - 1;
    }

    return -1;
}
```

PROGRAMAR É UM PROCESSO ITERATIVO

ESCREVER → TESTAR → FALHAR → CORRIGIR → MELHORAR



Bentley critica a visão ingênua de que o programador “pensa tudo antes” e depois apenas implementa.

Bons programadores não erram menos – eles detectam erros mais cedo.

TESTES HUMANOS

O raciocínio do autor é:

- humanos são inconsistentes
- esquecem casos
- não repetem testes da mesma forma
- param quando “parece que funcionou”

PORTANTO...

TEST HARNESS

Criação de programas testadoras de programas que:

- executam o código com muitas entradas
- repetem testes automaticamente
- permitem testar após cada modificação

SEM TESTES AUTOMATIZADOS, O PROGRAMADOR NÃO TEM COMO CONFIAR QUE UMA MUDANÇA NÃO QUEBROU ALGO ANTIGO.

MÉTODOS DE TESTAGEM

QUAL A IDEIA?

- gerar entradas automaticamente
- usar testes aleatórios
- comparar a saída com uma versão simples e confiável (mesmo que lenta)

QUAIS AS VANTAGENS?

- elimina parte dos vieses humanos
- cobre casos inesperados
- encontra falhas sutis

ESSES MÉTODOS SÃO PODEROSOS JUSTAMENTE PORQUE EXPLORAM SITUAÇÕES QUE O PROGRAMADOR NÃO PENSARIA MANUALMENTE

OS LIMITES INEVITÁVEIS DOS TESTES

- Testes dependem de modelos e suposições
- Mesmo testes aleatórios seguem regras definidas por humanos
- Passar em todos os testes ≠ estar correto

CASOS FORA DA NOSSA IMAGINAÇÃO CONTINUAM INVISÍVEIS.

PROGRAMAR COMO ATIVIDADE INTELECTUAL, NÃO MECÂNICA

JÁ QUE NEM TESTES AUTOMATIZADOS GARANTEM CORREÇÃO, PRECISAMOS ESCREVER CÓDIGO QUE AJUDE O PROGRAMADOR A RACIOCINAR MELHOR.

- Programar é construir um argumento lógico
- O código deve ser compreensível antes de ser executável
- Clareza reduz bugs mais do que complexidade

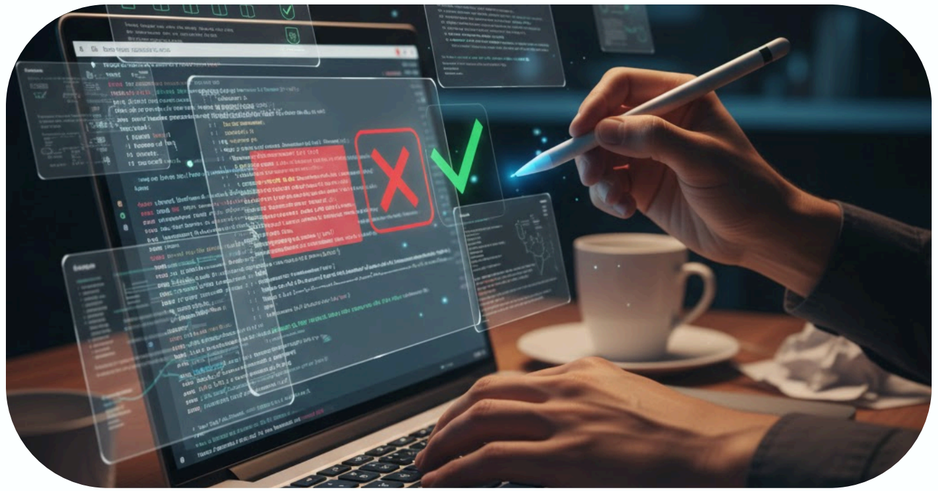
CÓDIGO CLARO É MAIS FÁCIL DE REVISAR, TESTAR E CORRIGIR.

PROGRAMMING PEARLS

TORNAR SUPOSIÇÕES EXPLÍCITAS NO CÓDIGO

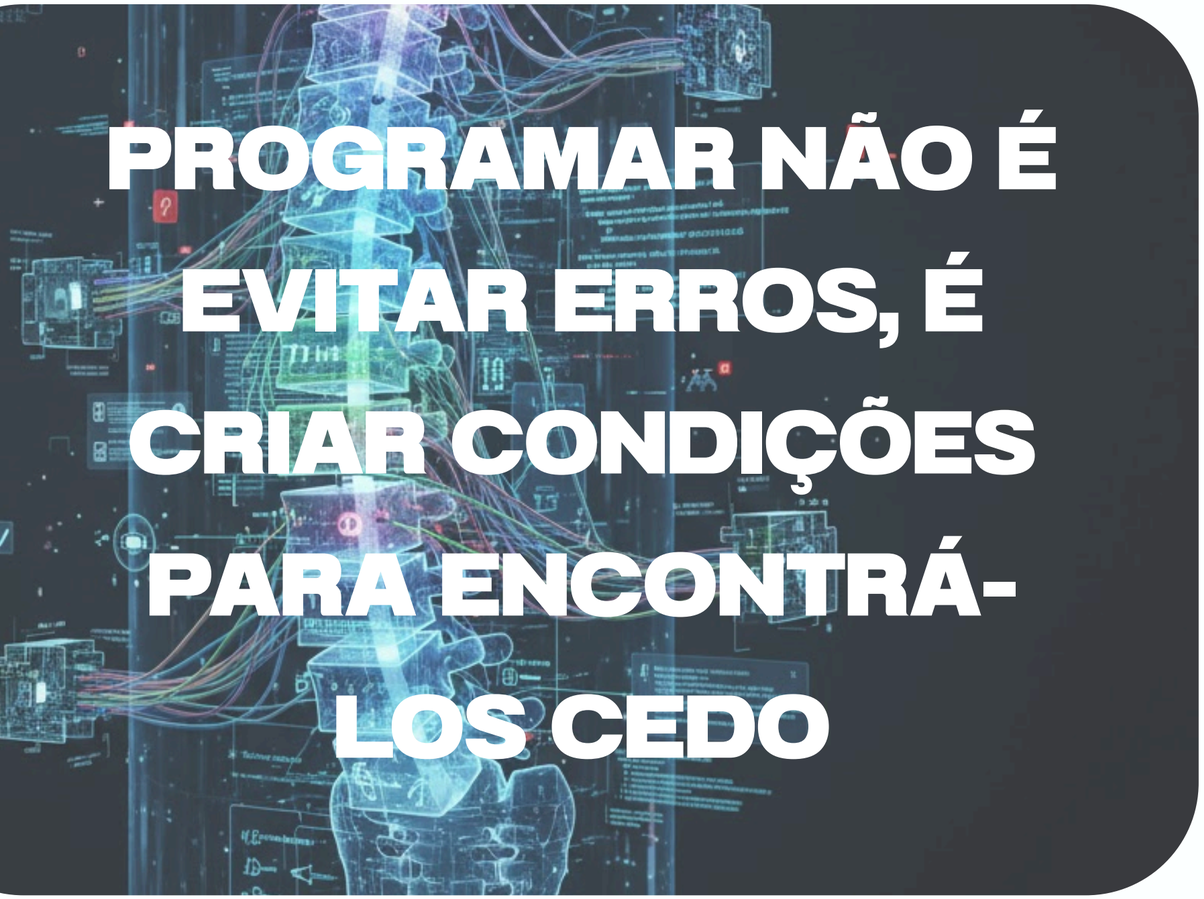
BENTLEY MOSTRA QUE MUITOS ERROS:
NÃO É A LÓGICA ERRADA.

É A EXPECTATIVA NÃO DOCUMENTADA.



- Suposições implícitas são fonte comum de bugs
- O código deve verificar o que espera
- Estados inválidos devem ser detectados cedo

COLUNA 5



O QUE ENSINA

- Bugs são inevitáveis
- Testes não garantem correção total
- Bons programadores constroem código que se explica

COMO SE ORGANIZA

- Erros surgem na transição ideia → código
- Testes ajudam a revelar falhas
- Clareza e verificação reduzem riscos

POR QUE ESSA COLUNA AINDA É ATUAL

- Os erros continuam os mesmos
- Linguagens mudaram, problemas não
- Clareza e disciplina seguem essenciais

Bentley escreveu Programming Pearls em um contexto onde:

- **computadores eram mais lentos**
- **memória era escassa**
- **ferramentas de debug eram limitadas**

O QUE MUDOU DESDE ENTÃO

- Linguagens mais seguras
- testes automatizados avançados
- debuggers poderosos
- pipelines de CI/CD

MAS...

O QUE NÃO MUDOU

- suposições erradas
- casos extremos ignorados
- confiança excessiva em testes

LINGUAGENS MUDARAM, FERRAMENTAS EVOLUÍRAM, MAS OS ERROS CONTINUAM NASCENDO DO MESMO LUGAR: DA FORMA COMO PENSAMOS AO ESCREVER CÓDIGO.

OBRIGADO PELA ATENÇÃO!