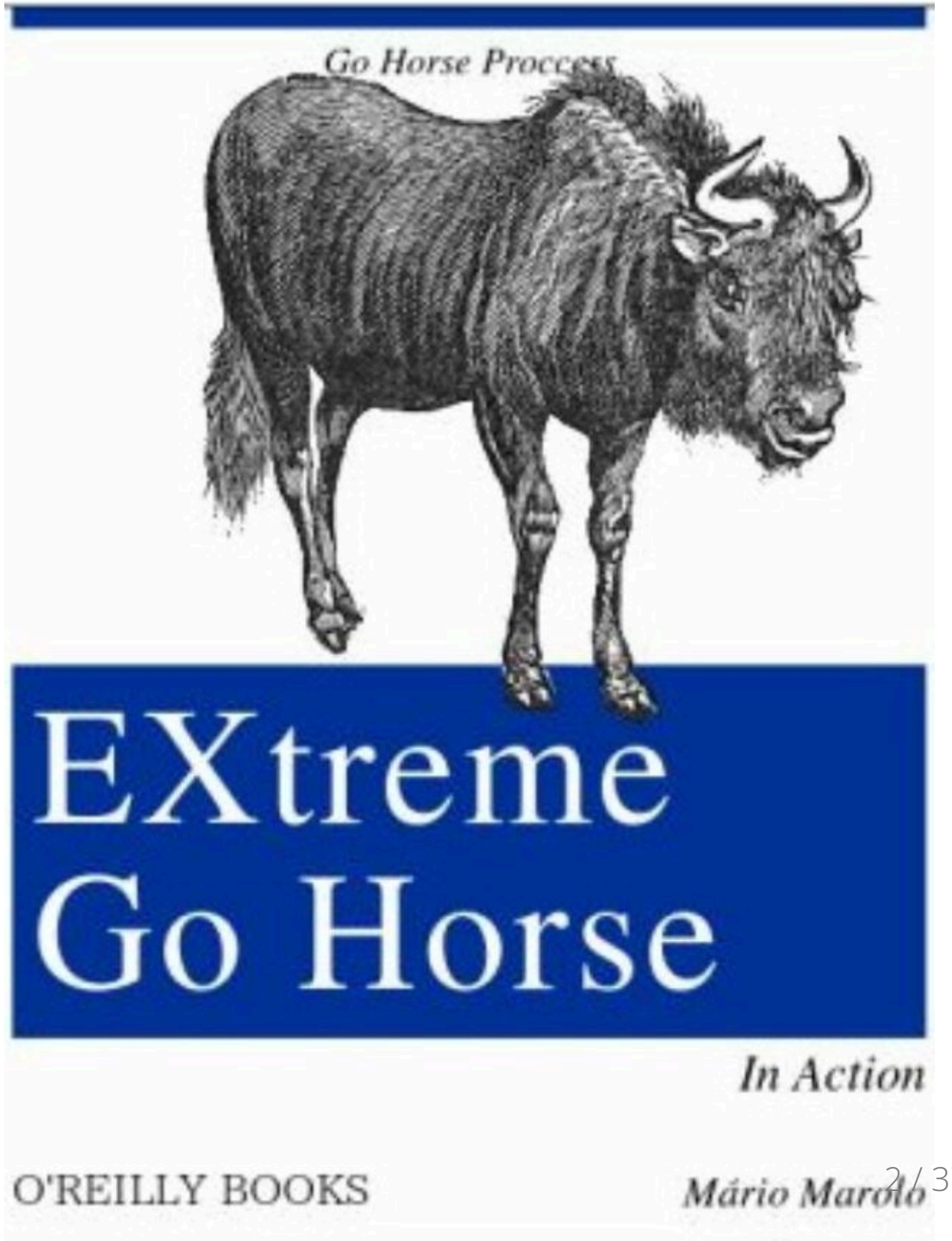
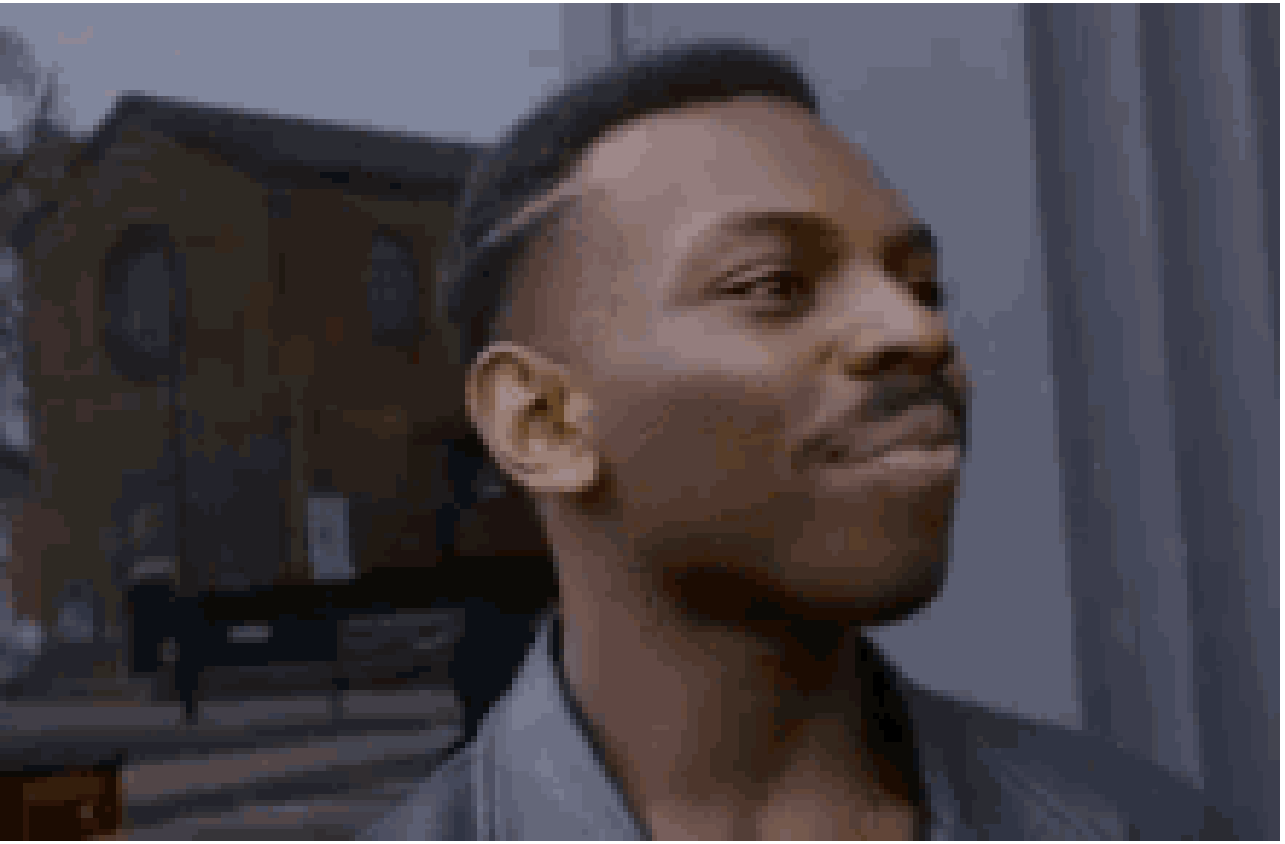
TÓPICO 16 - TÓPICOS ESPECIAIS

Clean Code - Professor Ramon Venson - SATC 2025.2

eXtreme Go Horse (XGH)

Metodologia de desenvolvimento de software que enfatiza a entrega rápida e contínua de funcionalidades



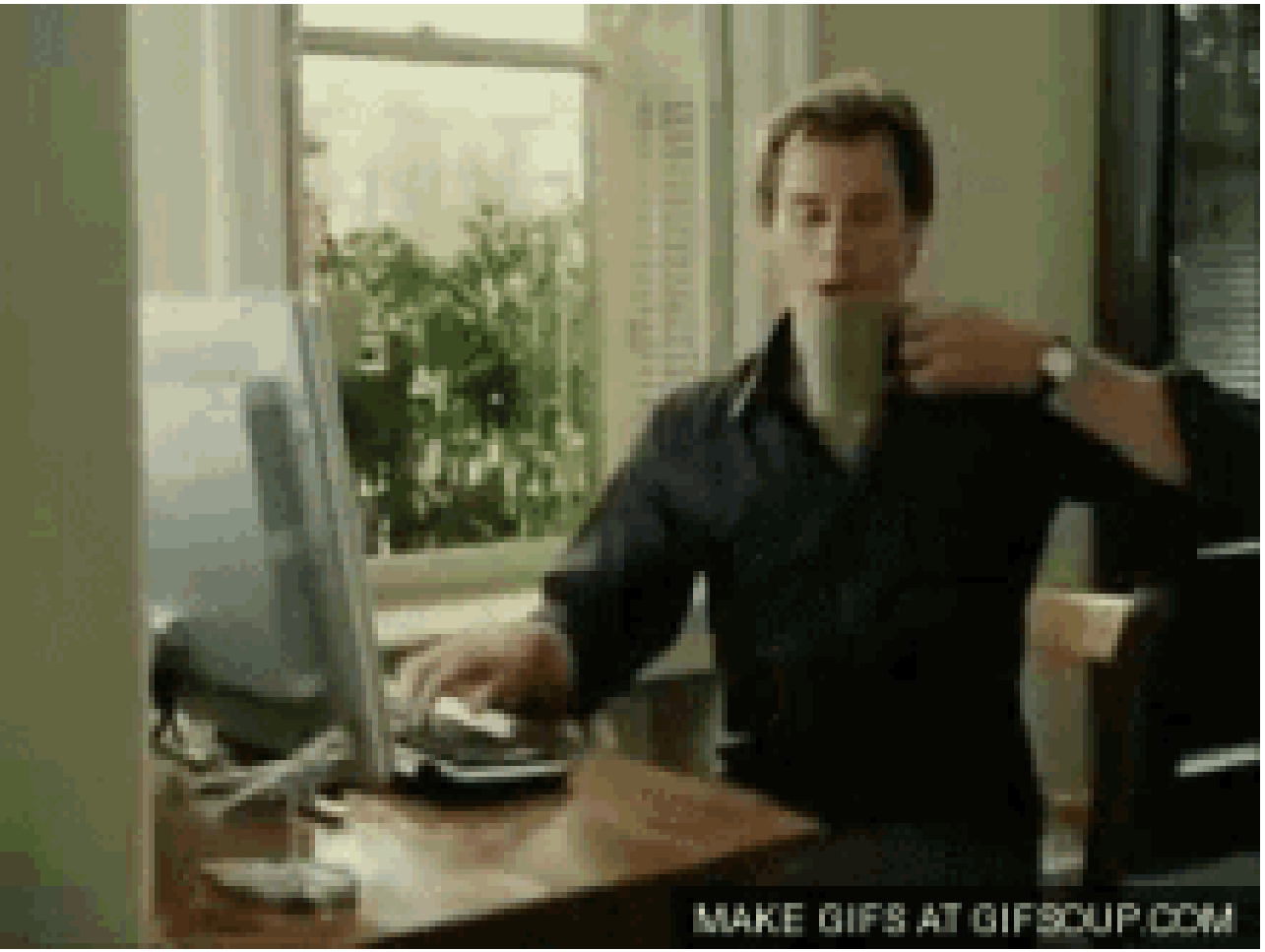


1 - Pensou, não é XGH.

Planejar ou discutir ideias tomam tempo de desenvolvimento. Se tiver uma ideia, comece a codificar imediatamente.

2 - Existem três formas de se resolver um problema: a correta, a errada e a do XGH.

A solução correta pode levar tempo para ser implementada, a errada pode causar problemas futuros, use a solução rápida e eficaz, cobrindo ambos os cenários.



3 - Quanto mais XGH, mais você poderá fazer.

Quanto mais você usar XGH, mais funcionalidades poderá entregar rapidamente.

4 - XGH é um modelo reativo

Antecipar problemas ou planejar soluções a longo prazo não é adequado para 99% dos projetos. Foque em resolver o que é mais urgente.

5 - Foco na entrega

O objetivo principal do XGH é entregar funcionalidades rapidamente, mesmo que isso signifique comprometer a qualidade do código.

6 - Commit antes do update

Atualizar o código de outras pessoas pode causar conflitos. Faça um commit das suas mudanças antes de atualizar o repositório.

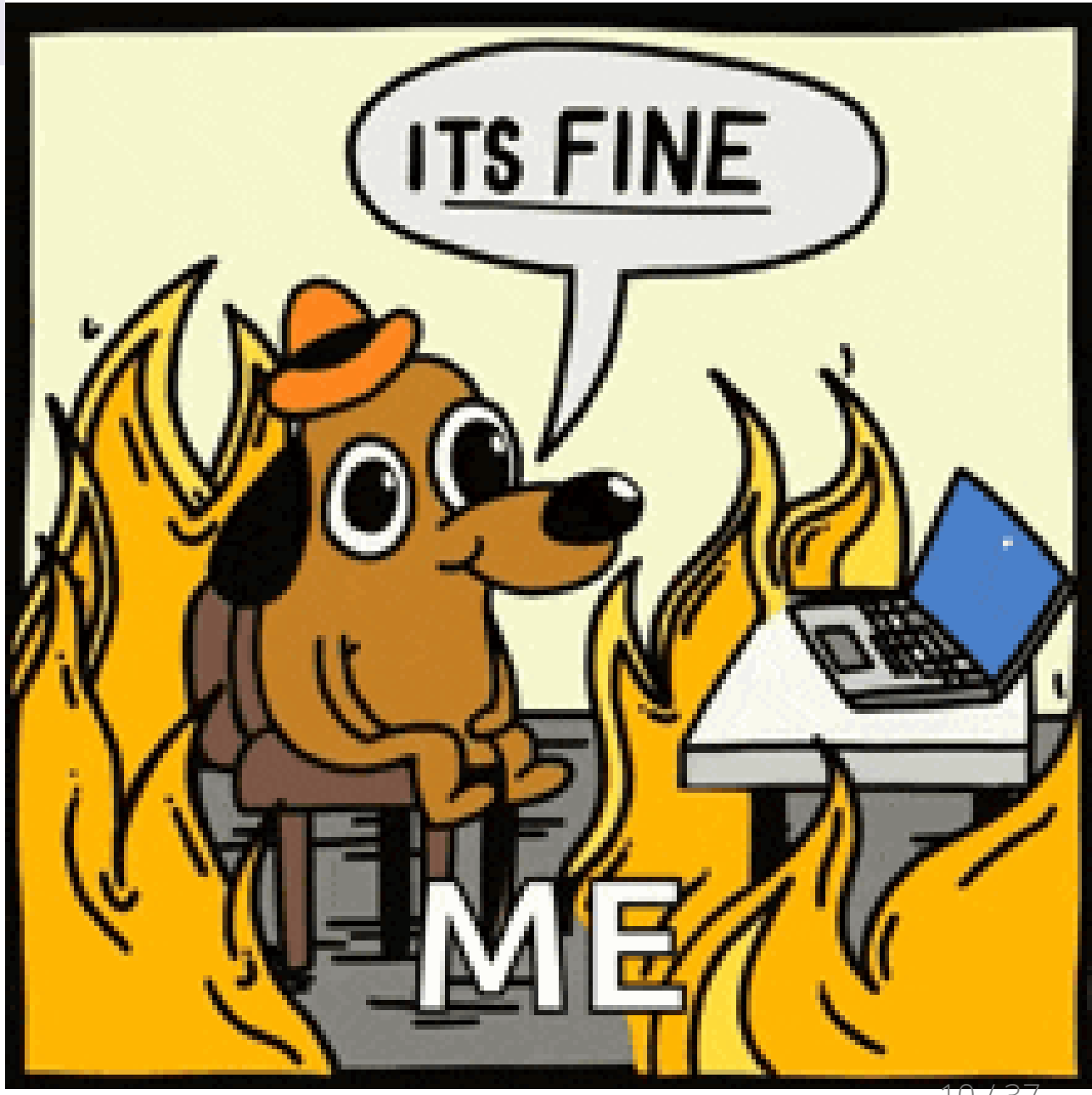
7 - XGH não tem prazo

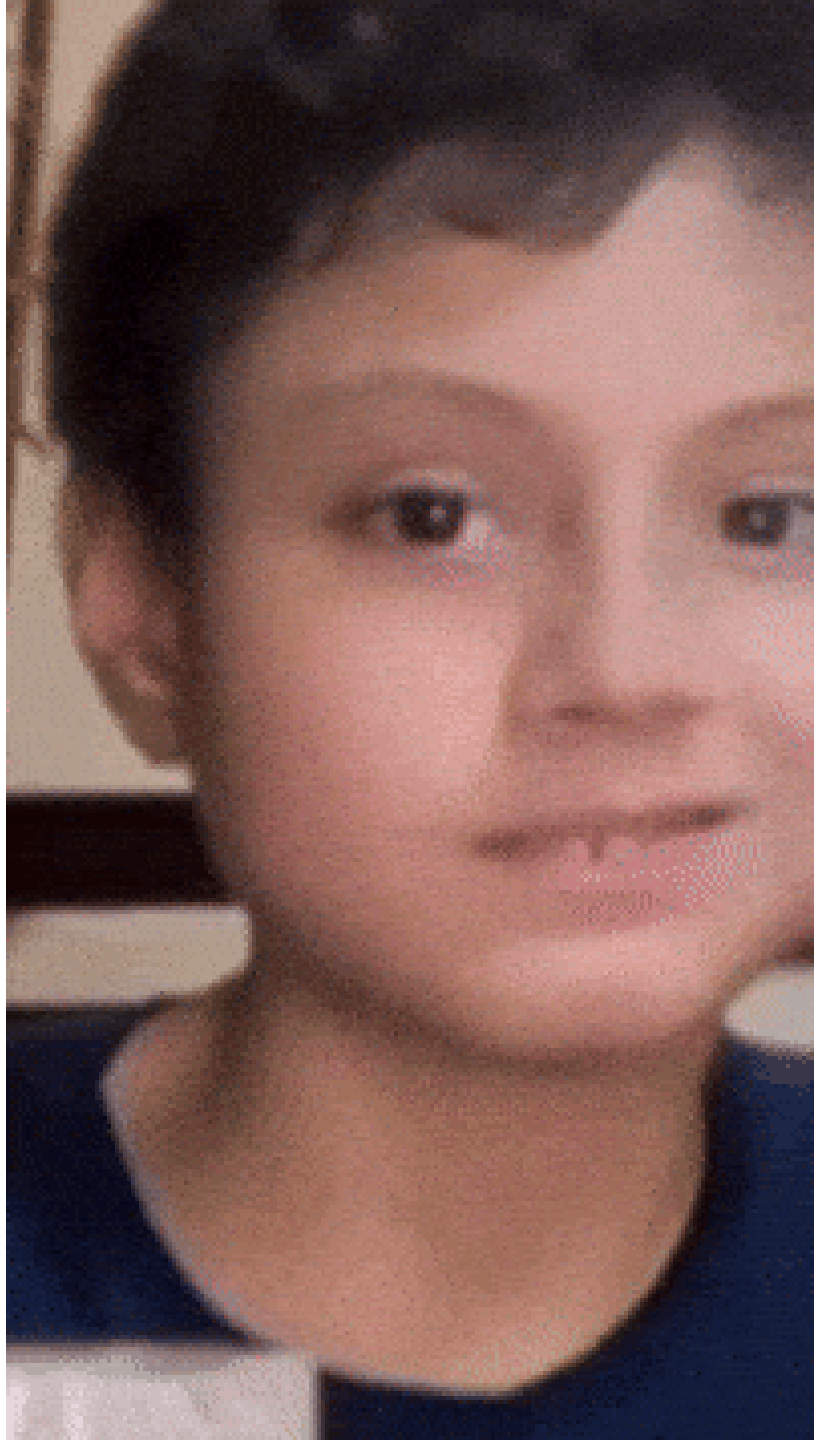
Não se preocupe com prazos ou cronogramas. O foco é entregar funcionalidades rapidamente, independentemente do tempo que isso levará.

8 - Esteja preparado para o pior

Esteja sempre pronto para lidar com problemas inesperados que possam surgir no meio do caminho.

Se algo der errado, é porque alguem do time não seguiu corretamente a filosofia do XGH.





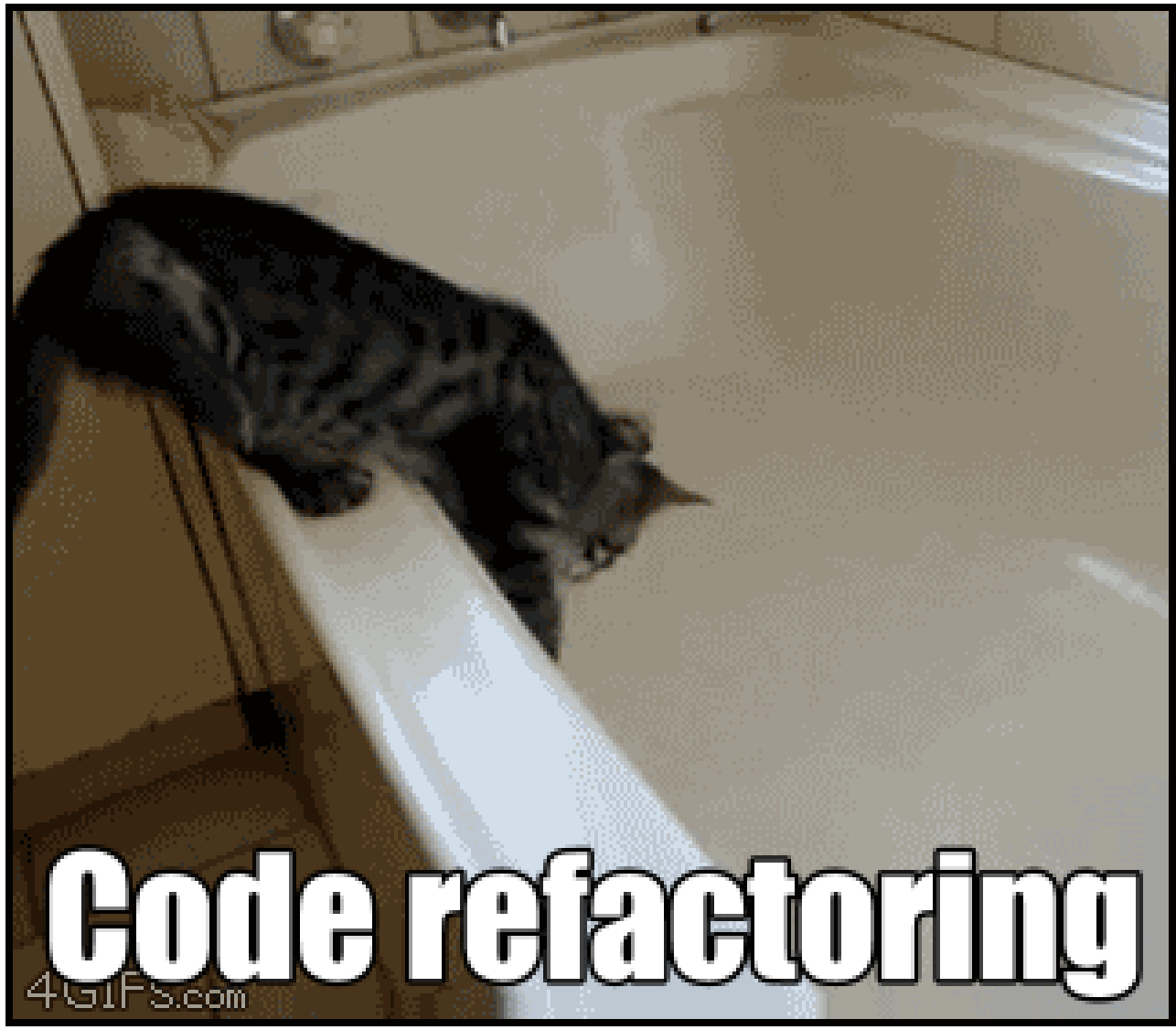
9 - Seja autêntico

Práticas tradicionais de desenvolvimento ou modismos são ineficientes. Abrace a filosofia e escreva seu próprio código.

10 - Retrabalho antes de refatoração

Se algo não funcionar como o esperado, crie uma solução rápida para corrigir o problema, em vez de tentar refatorar o código existente.

Se o problema persistir, o projeto vai afundar rapidamente pois não está preparado para o mercado.



11 - Anarquia no código

Gerente de projeto que tenta impor regras ou padrões ao código está indo contra a filosofia do XGH.

As vezes é necessário quebrar algumas regras para entregar os resultados.

12 - Motive-se com o futuro

Acredite que, em algum momento no futuro, o código será melhorado ou refatorado.

Isso ajuda a evitar ansiedade desnecessária.

13 - XGH é absoluto

Pratique XGH em todas as situações possíveis. Não há exceções para a filosofia do XGH.

14 - XGH é atemporal

Scrum, XP, Kanban e outras metodologias ágeis são apenas modismos passageiros.

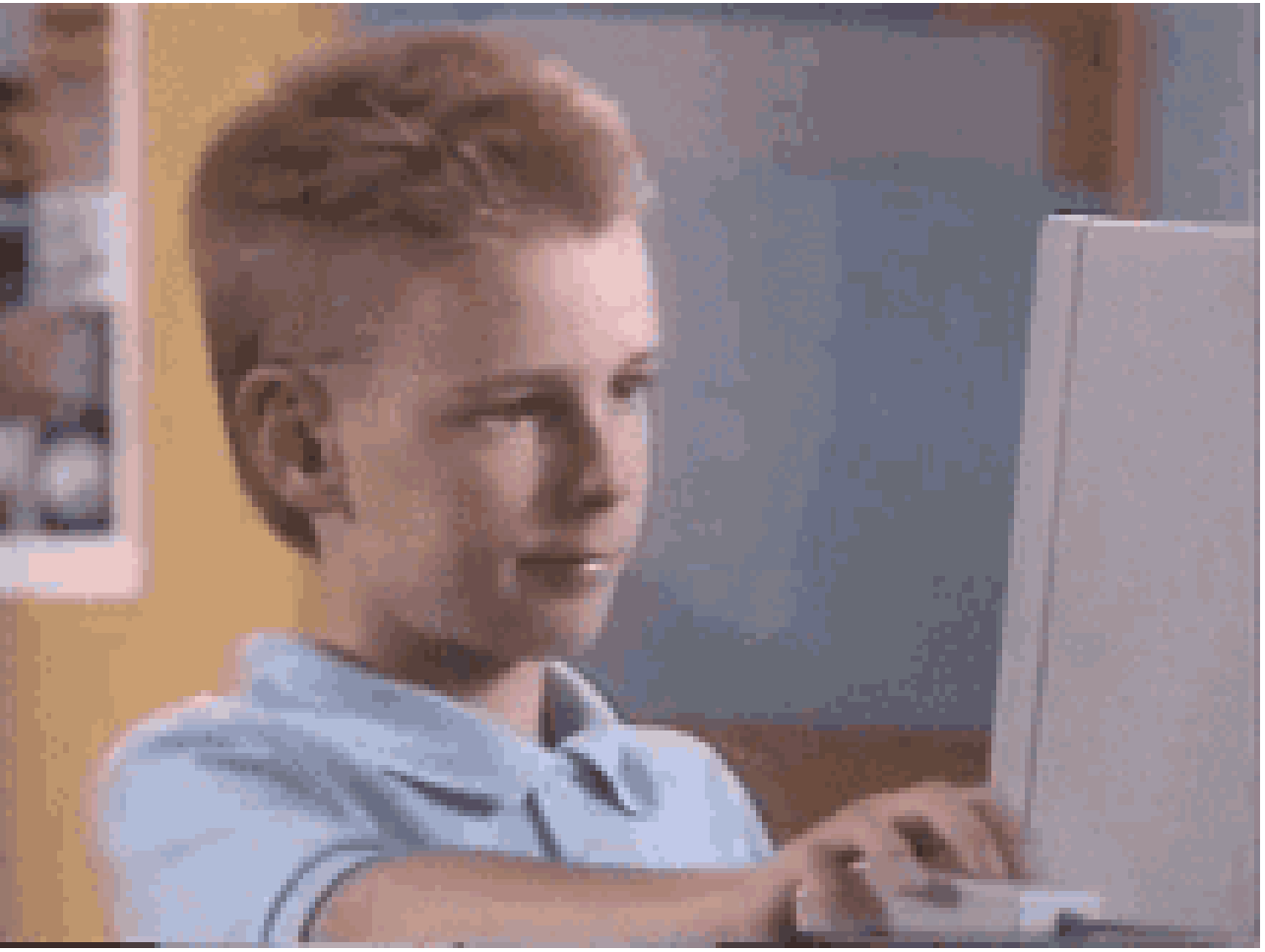
XGH é a única abordagem verdadeira para o desenvolvimento de software que já se provou eficaz ao longo do tempo.

15 - XGH nem sempre é POG

POG exige raciocínio lógico e planejamento, enquanto XGH é sobre adaptação e rapidez.

16 - Não tente remar contra a maré

Não se cobre tanto. Para cada *design pattern* que você usa corretamente, inevitavelmente, haverá adaptações de design.



17 - XGH não é perigoso até surgir ordem

Não tente estabelecer ordem desnecessária em um código desenvolvido com XGH. Isso só trará mais problemas e complicações do que soluções.

18 - XGH é vingativo

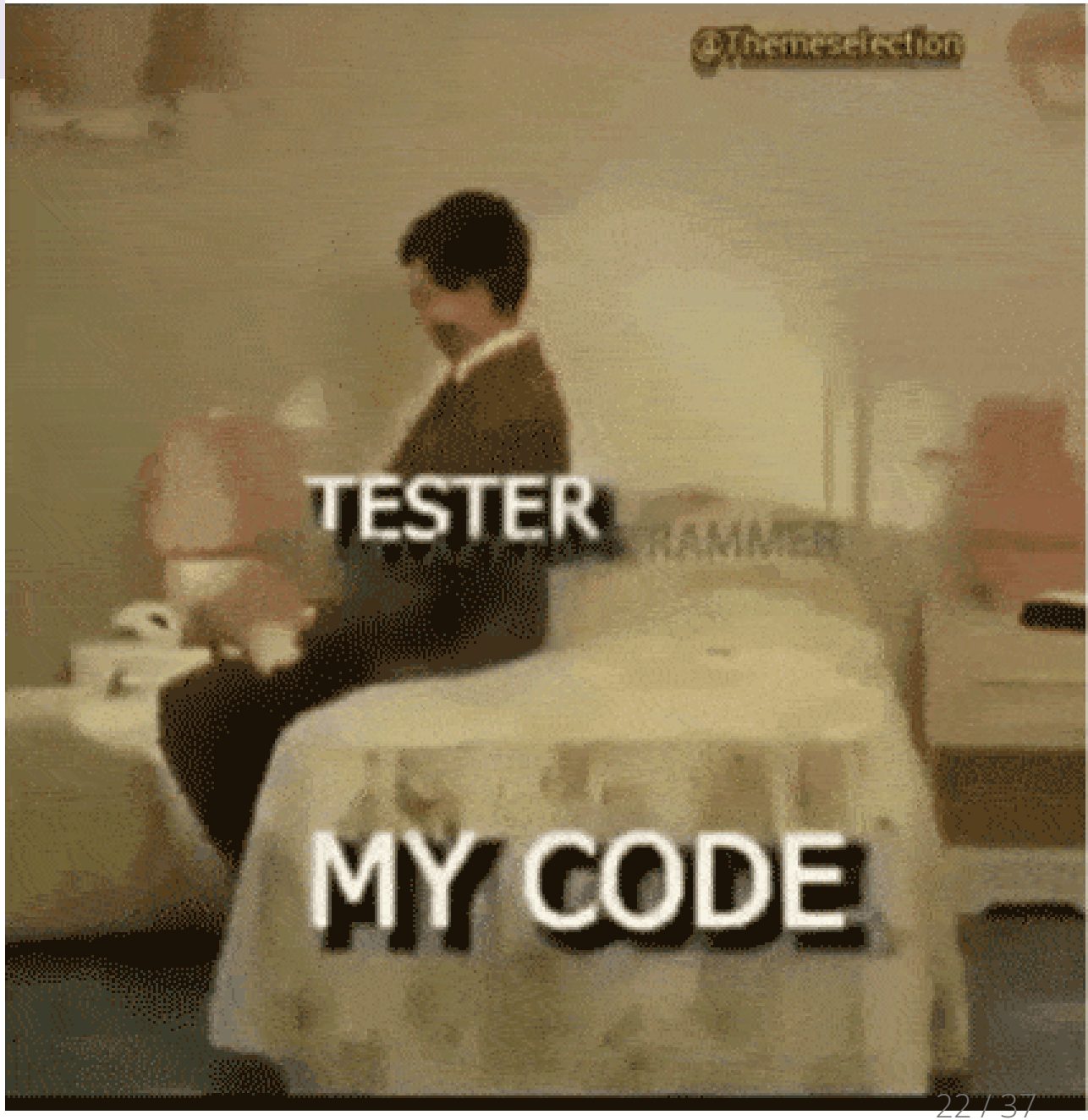
XGH não recomenda refatoração. Um sistema que foi desenvolvido com XGH nunca poderá ser refatorado sem causar colapso completo.

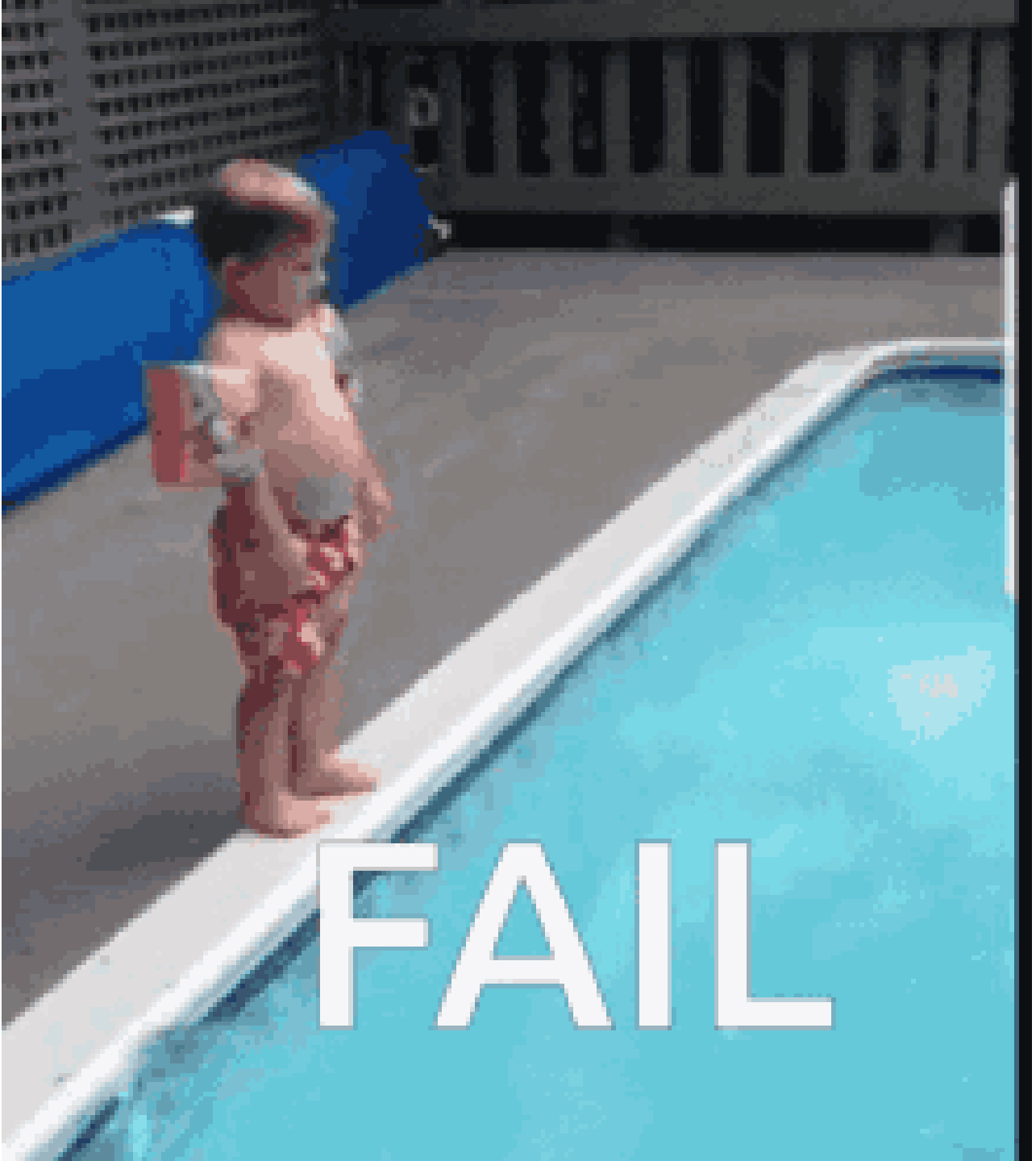
19 - Se estiver funcionando, não mexa

Nunca altere, muito menos questione o código que está funcionando. Ele funciona e não deve ser alterado.

20 - Teste é para os fracos

Testes automatizados não resolvem o problema de confiança. Seu programa deve funcionar como esperado sem eles.





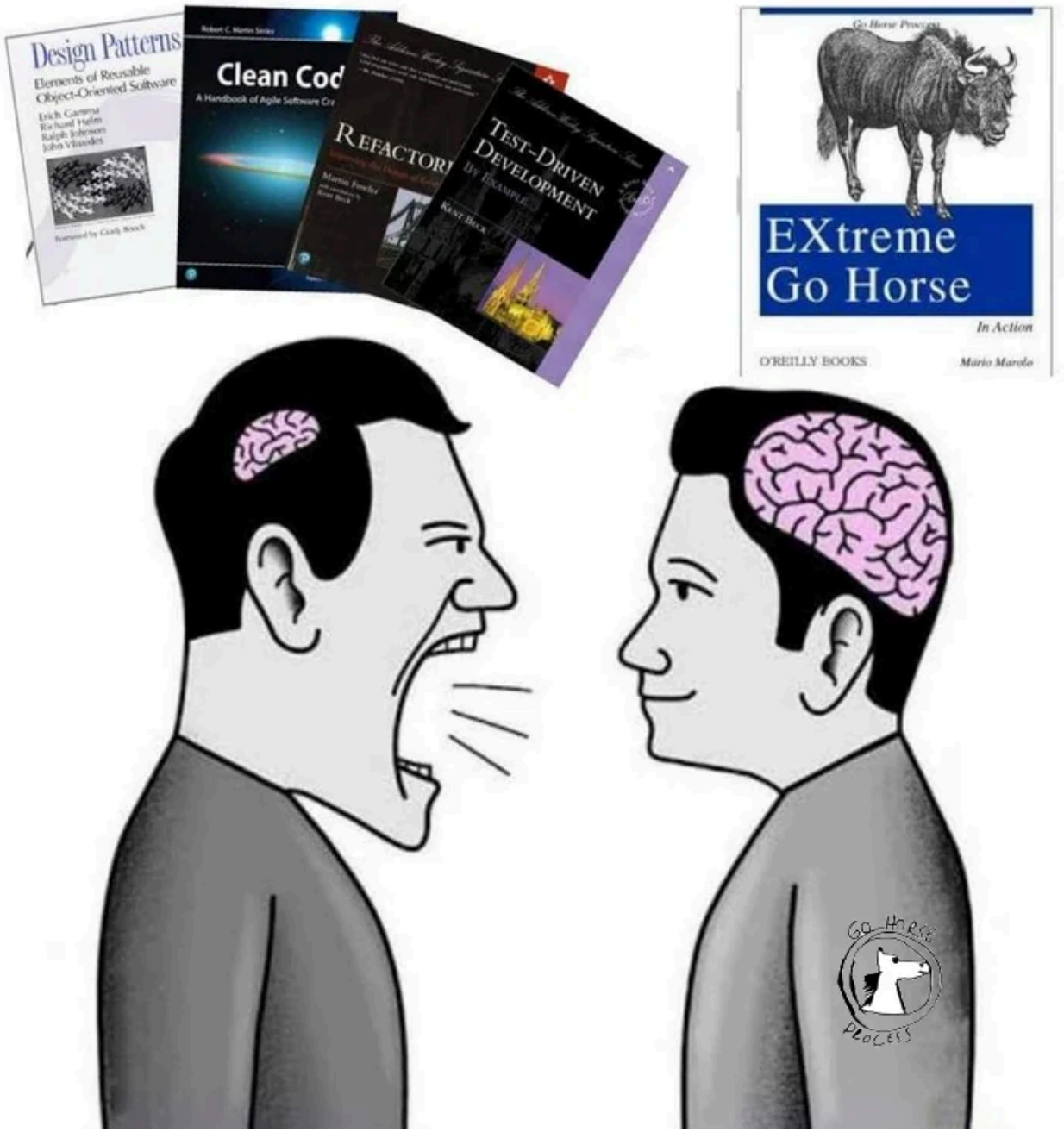
21 - Acostume-se com o fracasso

Projetos naufragam frequentemente. Aceite isso como parte do processo de desenvolvimento.

22 - Não deixe rastros

Você tem culpa apenas pelo que alterou.

Não mexa no código dos outros, assim você não será responsabilizado por falhas que não causou.



Conclusão

O eXtreme Go Horse (XGH) é uma abordagem irreverente e humorística para o desenvolvimento de software que enfatiza a rapidez e a improvisação em detrimento da qualidade e da manutenção do código.

Embora não seja uma metodologia séria, o XGH serve como um lembrete divertido dos perigos de negligenciar boas práticas de desenvolvimento.

Tiger Style

Tiger Style é uma metodologia que prioriza a experiência do usuário no desenvolvimento de código baseado em dois princípios fundamentais:

- **Segurança**: Garantir que o código seja robusto e confiável, minimizando falhas e vulnerabilidades.
- **Velocidade**: Otimizar o desempenho do código para proporcionar uma experiência rápida e eficiente
- **Simplicidade**: Manter o código simples e fácil de entender, facilitando a manutenção e evolução futura.

> "Simplicidade e elegância não são populares porque requerem trabalho duro e disciplina pra alcançar"

Edsger Dijkstra

Segurança

- **Use a estrutura mais simples para controle -** Abstrações em demasia não são livres de custo e podem introduzir complexidade desnecessária.
- **Coloque um limite em todo loop -** Garanta que todos os loops tenham um limite claro para evitar execuções infinitas.
- **Evite estados globais -** Minimize o uso de variáveis globais para reduzir dependências e facilitar a manutenção do código.
- **Declare variáveis no menor escopo -** Limite o escopo das variáveis para onde elas são realmente necessárias

- **Use tipos de dados simples -** Prefira tipos de dados simples e bem definidos para facilitar a compreensão e interoperabilidade do sistema.
- **Limite as funções a 70 linhas -** Mantenha-as curtas e focadas em uma única tarefa.
- **Leia desde o início aos avisos do compilador -** Eles podem indicar problemas potenciais no código ou na forma como ele é escrito.
- **Não reaja imediatamente a eventos externos -** Seu programa deve rodar de forma previsível, independentemente de eventos externos.

- **Todos os erros devem ser tratados -** Uma análise de falhas de produção em sistemas distribuídos com uso intensivo de dados revelou que a maioria das falhas catastróficas poderia ter sido evitada com testes simples do código de tratamento de erros.
- **Sempre use valores padrão -** Eles ajudam a evitar erros inesperados e garantem que o sistema tenha um comportamento previsível.

```
# Código para conectar ao banco de dados
def connect_to_database(host=None, port=None):
    host = host or "localhost"
    port = port or 5432
```

Performance

- **Pense em performance desde o início -** O melhor momento para otimizar o desempenho é durante o design inicial do sistema, quando não é possível medir ou usar *profilers*.
- **Calcule os recursos computacionais -** Estime o uso de CPU, memória e largura de banda para garantir que o sistema atenda aos requisitos de desempenho.
- **Amortize o custo computacional com *batching* -** Agrupe operações para reduzir a sobrecarga de chamadas repetidas.

Exemplo de cálculo de logs

1. **Estime o volume** de logs:
   Considere 1.000 requisições por segundo (RPS).
   Cada entrada de log tem cerca de 1 KB.
2. **Calcule o volume diário** de logs:
   1.000 RPS * 86.400 segundos/dia * 1 KB ≈ 86.400.000 KB/dia ≈ 86,4 GB/dia.
3. **Estime o armazenamento mensal**:
   86,4 GB/dia * 30 dias ≈ 2.592 GB/mês.
4. **Estime o custo** (usando R$0,10 por GB para armazenamento em *blob*):
   2.592 GB * R$0,10/GB ≈ R$259 por mês.

Simplicidade

- **Use verbos e substativos corretos -** Escolha nomes claros e descritivos para funções e variáveis.
- **Não abrevie -** Evite abreviações que possam confundir outros desenvolvedores.
- **Adicione unidades e qualificadores na ordem -** Por exemplo, use `latency_ms_max` em vez de `max_latency_ms`. Isso permitirá um alinhamento mais organizado quando `latency_ms_min` for adicionado, além de agrupar todas as variáveis relacionadas à latência.

- **Procure por nomes simétricos -** Por exemplo, como argumentos para uma função `memcpy`, `source` e `target` são melhores do que `src` e `dest`, porque têm o efeito colateral de que quaisquer variáveis relacionadas, como `source_offset` e `target_offset`, ficarão alinhadas em cálculos e fatias.
- ***Callbacks* vão por último na lista de parâmetros -** Isso esclarece que *callbacks* são sempre invocados após a criação dos parâmetros principais.
- **Ordem importa para legibilidade -** Na primeira leitura, o arquivo é top-down. A função `main` sempre vem primeiro.

Nasa's Power of Ten

A Nasa desenvolveu uma metodologia chamada "Power of Ten" (Poder do Dez) para gerenciar a complexidade em projetos de engenharia, incluindo o desenvolvimento de software.

Conclusão

No fim do dia, o objetivo dessas metodologias é criar código que seja:

- Seguro
- Rápido
- Simples

Seguindo essas diretrizes, você pode melhorar a qualidade do seu código e facilitar a manutenção futura.

Material de Apoio

- eXtreme Go Horse (XGH) - Site Oficial
- The Power of Ten - Gerard J. Holzmann (Nasa)
- TigerBeetle - A Million Financial Transactions per Second in Zig