



# Embarcadero Conference

Um único esforço, uma única base de código, múltiplas  
plataformas, múltiplos dispositivos

Guinther Pauli

# Padrões de Projeto, Refatoração e Migração



# Introdução

- Uma premissa básica no desenvolvimento de software é a necessidade de evoluir
- Evoluir para
  - uma nova plataforma (Web, Mobile)
  - uma nova arquitetura (Multicamadas)
  - um novo banco de dados
  - um novo engine de acesso a dados
  - um novo framework gráfico (FireMonkey)
- Exemplos:
  - Client/Server para DataSnap, dbExpress para FireDac, VCL para FMX etc.





# Introdução

- Sistemas de software reais evoluem e tornam-se mais complexos ao longo do tempo
- Novos requisitos, novas tecnologias, difícil manutenção
- O software se afasta do seu projeto original
- Diversas técnicas podem ser usadas para auxiliar no gerenciamento da complexidade
  - Refatoração
  - Padrões de Projeto
  - Refatoração para Padrões



# Introdução

- Estima-se que de 50% a 70% do custo total de um sistema de software é gasto com sua manutenção
- O que torna um software difícil de manter e evoluir:
  - Código rígido, difícil de ler, com bad smells, complexidade, acoplamento entre units / classes, separação incorreta de responsabilidades, duplicação de código, falta de padrões, condicionais etc.
- Lembre-se, você passará a maior parte da sua vida clicando em “File > Open Project” e não “File > New X Application”



# O paradigma RAD

- A programação seguindo a abordagem RAD permite que muitos desenvolvedores abram mão de boas práticas de OO e padrões para obter um mesmo resultado mais rapidamente, seja por falta de conhecimento, prazos curtos ou para reduzir custos abrindo mão da qualidade
- O sistema parece funcionar bem por fora, mas por dentro está mal projetado, ou não tem projeto, ou está um verdadeiro caos que nem mesmo o programador original o entende
- Aqui é importante que o líder tenha alto conhecimento técnico e também de arquitetura



# O paradigma RAD

- Todo o código está em um único formulário / data module. Não existem classes de negócio, e as poucas que existem estão fortemente acopladas. Conforme a aplicação cresce, mais componentes, código e funcionalidades (e bugs, muitos deles) são adicionados ao projeto original
- O que era rápido no começo se tornou muito lento. Aqui a equipe já perde a credibilidade no próprio sistema, no negócio em si, nos demais profissionais, a gerência tenta achar a solução em fatores equivocados, há uma alta rotatividade na equipe, o caos já está generalizado
- Então cogita-se escrever tudo do zero



# O que é um bom software?

- Para muitas empresas, senão a maioria delas (exceto startups), o foco principal não é criar software rapidamente, mas software que seja fácil de ser mantido ao longo do tempo, pois isso reduz custos a longo prazo
- A refatoração e padrões, neste caso, entram como a ponte, a chave, o elo, a solução para se criar software com código sempre limpo e funcional, preparado para mudanças, que sempre existirão





# Bad Smells

- Large Class, ou Classe “Deus”
- Código difícil de ler / entender
- Métodos longos
- Código duplicado
- Sentenças condicionais
- Acoplamento / Dependência
- Lista longa de parâmetros
- Variáveis locais / temporárias
- Comentários (quando usados como “desodorante”)
- Obsessão primitiva



# Lembre-se

*"Qualquer tolo pode escrever código que um computador possa entender. Bons programadores escrevem código que humanos podem entender."*

Martin Fowler



# Boas Práticas e Code Clean

- Classes com no máximo 500 linhas
- Métodos com no máximo 20 linhas
- Código deve ser uma “redação”
- Princípio da Responsabilidade Única
- Você deve ser capaz de ler / pronunciar os nomes de métodos, classes, atributos, variáveis etc
- Cuidado com abreviações, “magic numbers”, use constantes bem nomeadas
- Encapsule o que muda
- Programe para Interfaces



# Padrões de Projeto

- Padrões de projeto são soluções reutilizáveis usadas para resolver problemas comumente encontrados em sistemas orientados a objetos
- Usar padrões de projeto torna mais fácil reutilizar soluções de arquiteturas bem-sucedidas
- Maximizam a reutilização de código, facilitam a manutenção e evolução de sistemas de software
- Padrões ajudam a projetar sistemas de software mais bem preparados para mudanças





# Refatoração

- Refatoração: reestruturar um sistema de software aplicando-se uma série de transformações sem modificar seu comportamento externo observável
- É uma mudança feita na estrutura interna de um sistema de software para torná-lo mais fácil de entender e modificar
- Melhora um projeto de código existente, torna o código mais simples, claro e mais fácil de manter e evoluir



# Refatoração e Padrões de Projeto

- Uma boa prática é utilizar a refatoração para modificar um código existente, tendo como resultado final dessa transformação um código em conformidade com a estrutura de um padrão de projeto
- A refatoração serve como uma alternativa ao chamado Grande Projeto Antecipado (GPA), ou *excesso de engenharia*
- Assim, não se cria inicialmente um grande projeto, que tente prever todas as possibilidades de mudanças que possam ocorrer
- “Programadores não são videntes!”



# Refatoração e Padrões de Projeto

- A aplicação de padrões de projeto, dentro de um contexto evolutivo baseado em técnicas de refatoração, se torna uma atividade atraente no ciclo de vida de uma aplicação
- Evita complexidade desnecessária na concepção de um sistema de software, evita também a *escassez de engenharia*, ou evoluir um sistema de software sem nunca aplicar um padrão (*débito de projeto*)
- Refatoração para padrões é o aspecto chave neste processo evolutivo, e é na evolução que se encontra a verdadeira sabedoria

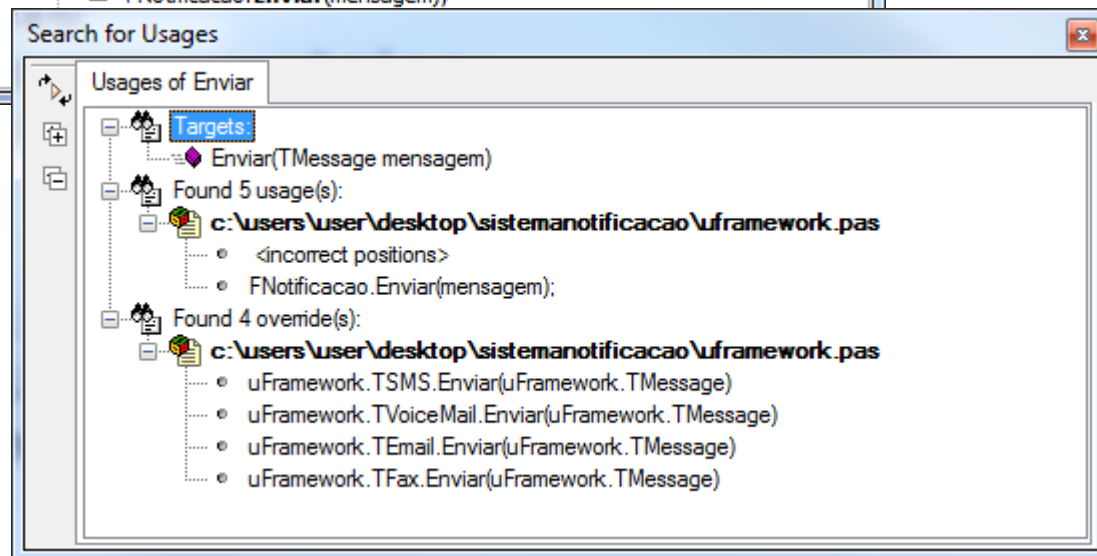
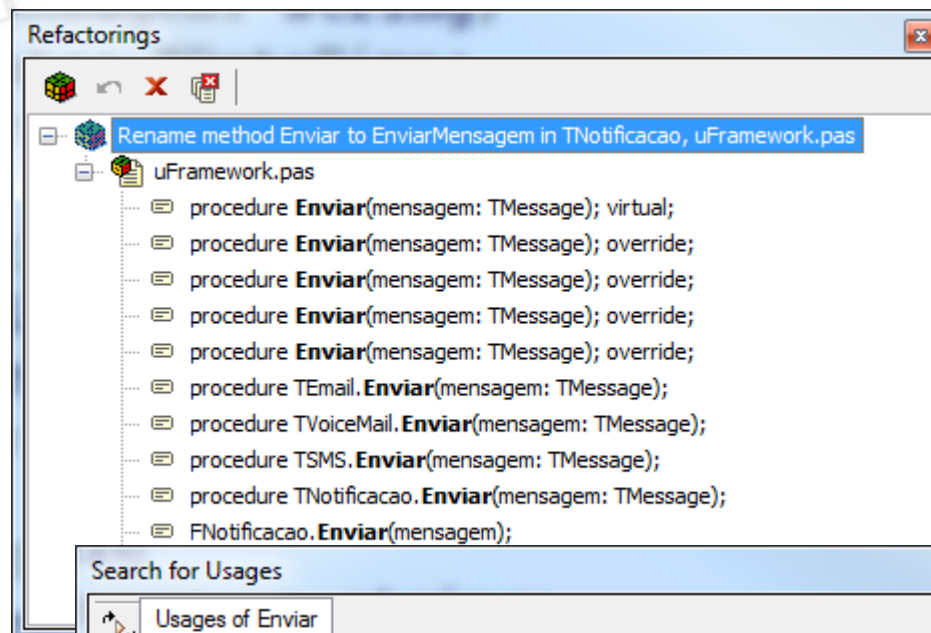
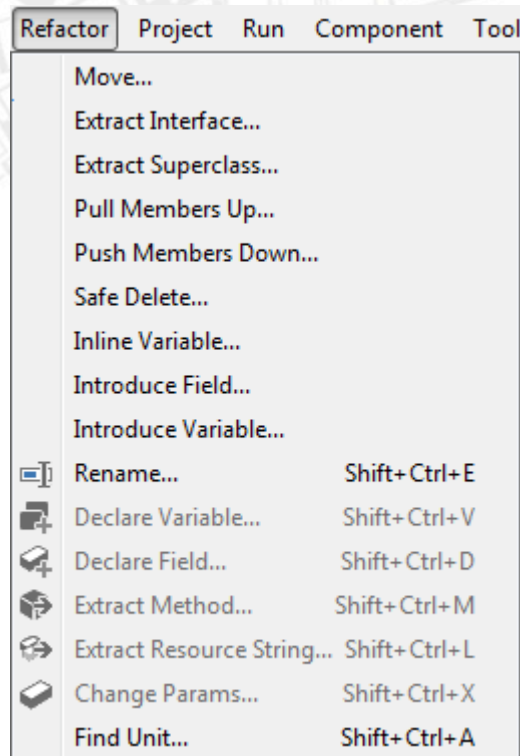


# Como o RAD Studio pode ajudar?

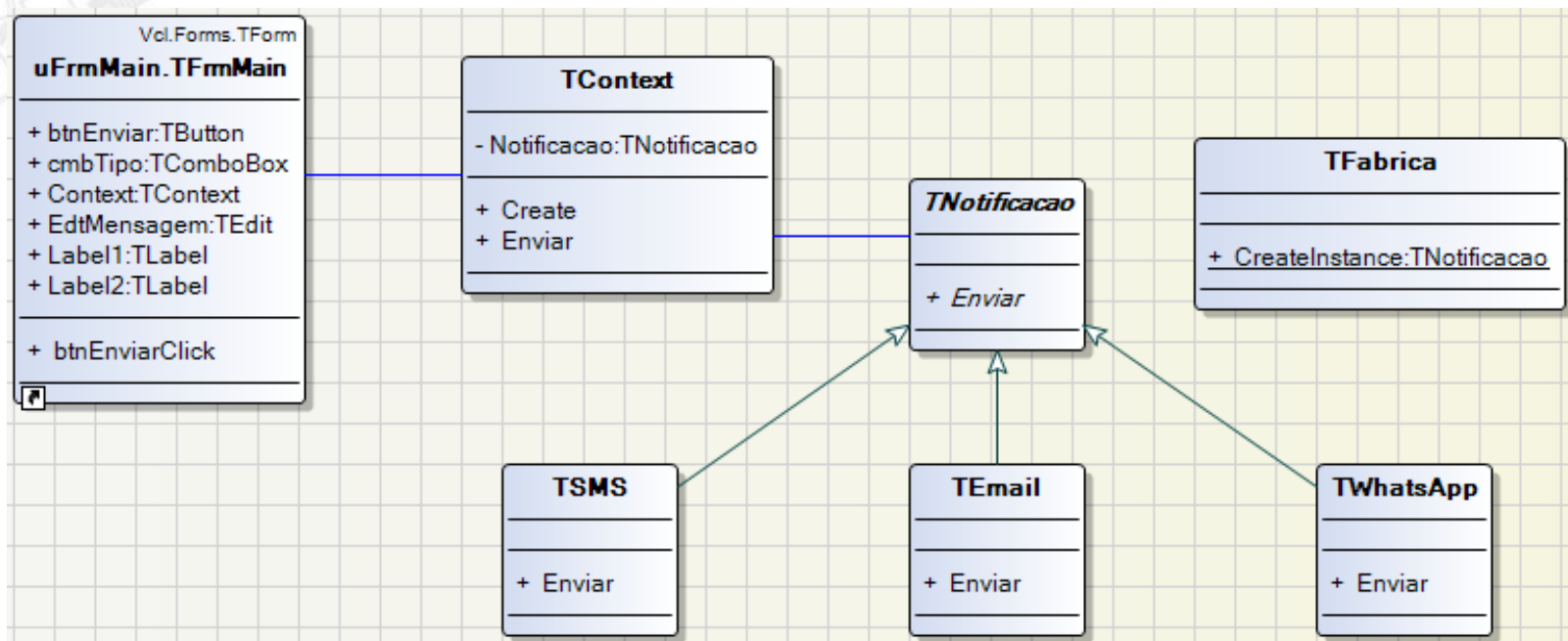
- Opções para refatoração integradas ao IDE (menu Refactor)
- Suporte integrado à UML
- QA Metrics
- QA Audits
- Gráficos
- Métricas
- Testes Unitários integrados ao IDE



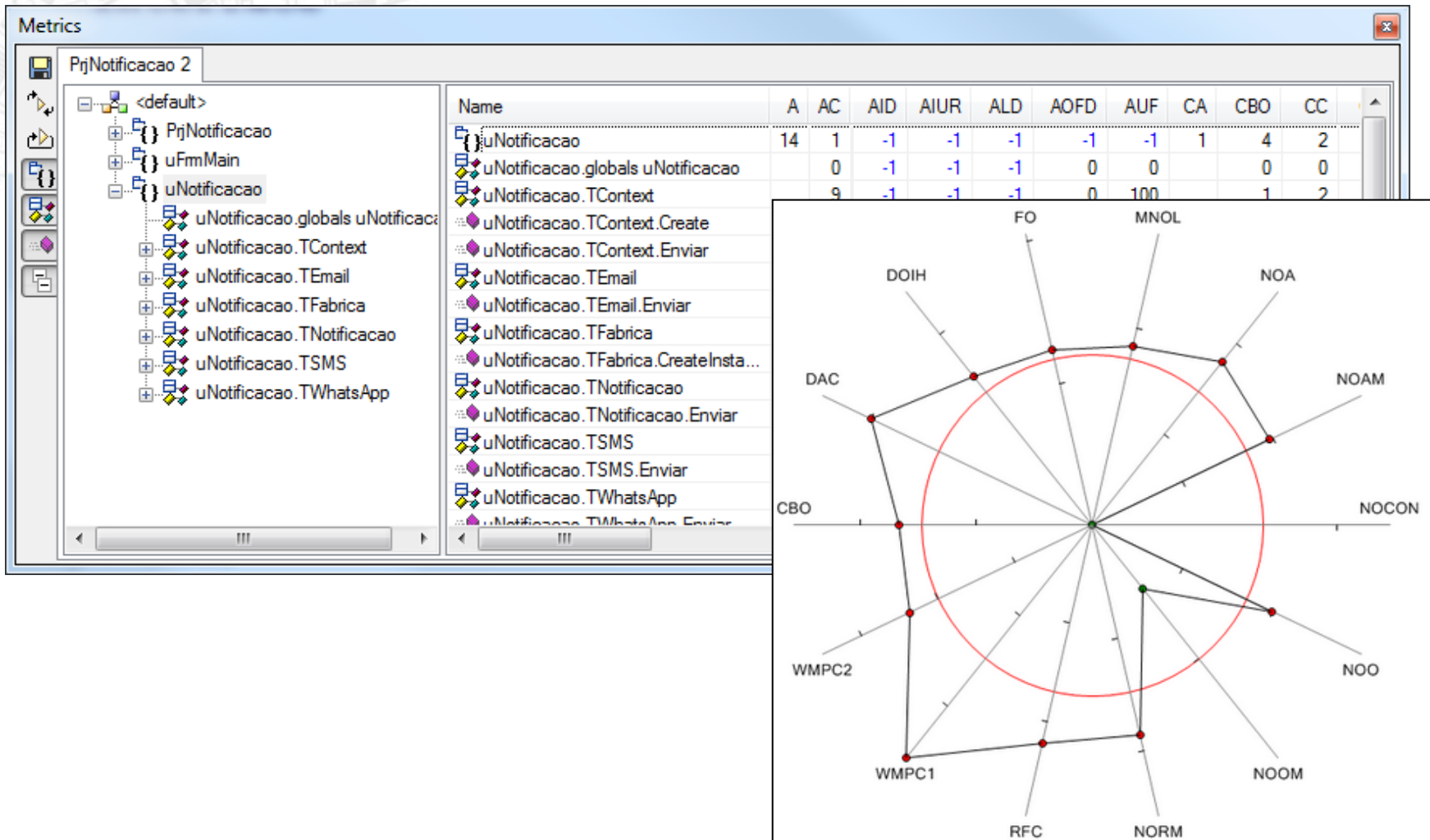
# RAD Studio - Refactor



# RAD Studio - UML



# RAD Studio – QA Metrics



# RAD Studio – Testes Unitários

The image shows two windows from the RAD Studio IDE. The 'New Items' dialog is on the left, and the 'DUnit: An Xtreme testing framework' test runner is on the right.

**New Items Dialog:**

- Left pane: A tree view showing project categories. 'Unit Test' is selected under 'Other Files'.
- Right pane: Two icons are shown: 'Test Case' and 'Test Project'. 'Test Project' is highlighted.
- Buttons: 'OK' and 'Cancel' are at the bottom right.

**DUnit: An Xtreme testing framework:**

This window displays the test hierarchy and execution results.

**Test Hierarchy:**

- Project1.exe
  - Form1 (TForm1) tests [OpenCTF 1.1]
    - TCustomActionListTestHandler
      - ActionList1 actions
        - Action1 (TAction) Events: OnExecute
        - Action2 (TAction) Events: OnExecute
      - TTabSheetHandler
        - TabSheet1 (TTabSheet)
        - TabSheet2 (TTabSheet)
      - TImageListHandler
        - ImageList1 (TImageList)
      - TMenuItemHandler
        - Quit1 (TMenuItem) Properties: OnClick
        - Copy1 (TMenuItem) Properties: OnClick
        - Paste1 (TMenuItem) Properties: OnClick
        - Cut1 (TMenuItem) Properties: OnClick
        - Unassigned1 (TMenuItem) Properties: OnClick
        - Action11 (TMenuItem) Properties: OnClick
        - Action21 (TMenuItem) Properties: OnClick

**Progress:**

Score:

Tests	Run	Failures	Errors	Elapsed
12				0:00:00.906

Test Name	Failure Type	Message	Location
Action2 (TActi...	ETestFailure	Action2.OnExecute is not assigned.	\$004728CA
TabSheet2 (T...	ETestFailure	TabSheet is invisible, should be hidden at run time	\$0049C394
ImageList1 (TI...	ETestFailure	ImageList is empty	\$0049C444
Unassigned1 (...)	ETestFailure	Unassigned1.OnClick is not assigned.	\$00472B62
Action21 (TM...	ETestFailure	Action21.OnClick is not assigned.	\$00472B62





# Estudo de Caso

- Desenvolver código mais limpo e funcional, com menos bugs
- Tornar arquiteturas de software mais fáceis de evoluir e manter
- Facilitar a implementação de novos requisitos, em menor tempo, sem causar bugs
- Melhorar a qualidade do código fonte existente
- Programar usando padrões já consolidados e empregados em softwares de grande escala
- Reduzir custos com manutenção



# Estudo de Caso

- Promover a reutilização de código
- Desenvolver de forma ágil, com entregas constantes para o cliente, em pequenos releases, sem bugs
- Capacidade de detectar locais de código fonte que podem ter sua estrutura melhorada
- Promover a modularização de aplicações, tornando mais fáceis de serem distribuídas (deploy)
- Reduzir riscos no projeto de software, promovendo a adoção de padrões já testados e de funcionalidade comprovada
- Aumentar a velocidade de desenvolvimento

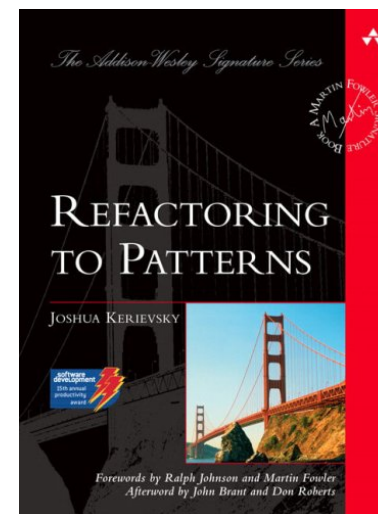
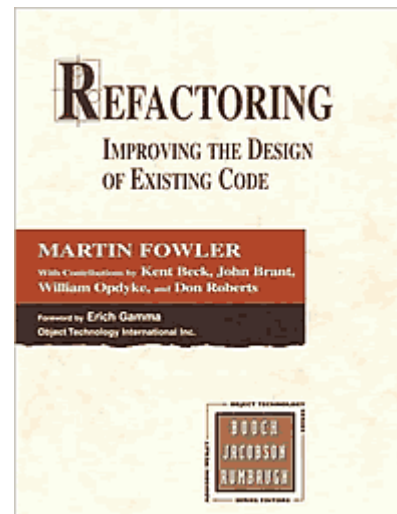
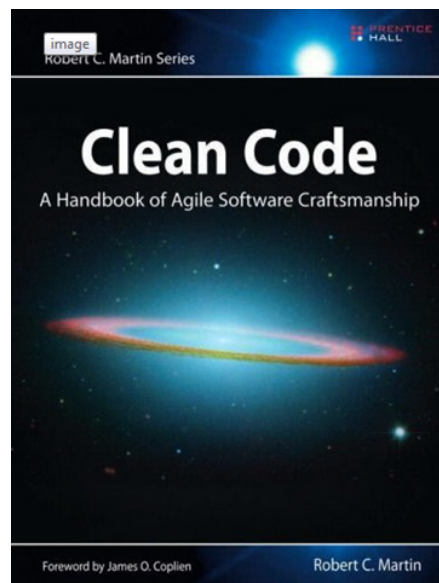
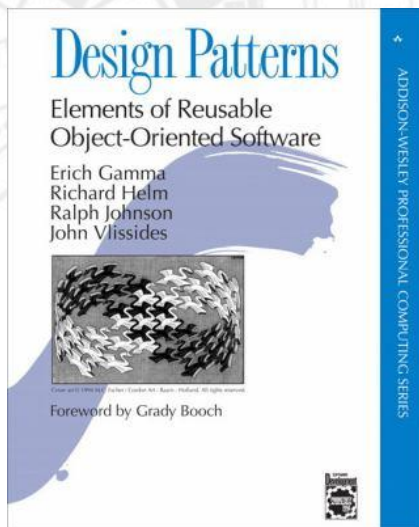


# Download

Disponível no Code Central:

- Exemplo do estudo de caso
- Apresentação Power Point
- Aplicação com fontes em Delphi dos 23 Padrões de Projeto GoF


# Bibliografia sugerida





# Muito obrigado!

Guinther Pauli – MVP


 [guintherpauli@gmail.com](mailto:guintherpauli@gmail.com)

 <http://www.facebook.com/guintherpauli>

 <http://www.twitter.com/guintherpauli>

 <http://br.linkedin.com/in/guintherpauli>

 <http://www.gpauli.com>

 <http://guintherpauli.blogspot.com>

 [guinther.pauli](https://github.com/guintherpauli)