Mestrado em Engenharia Informática e Tecnologia Web

Arquitetura e Padrões de Software

(22304)

Ano letivo 2025/2026

Antipadrões e Refatorização

ProPlan

Síntese

ProPlan é um módulo *Activity Provider* que apresenta ao aluno uma série de desafios de gestão de projeto simulados, baseados em cenários realistas.

André Sousa

1300012@estudante.uab.pt

Repositório Projecto: AndreMacielSousa/proplan-activity-provider: proplan-activity-provider

ANTIPADRÕES E REFATORIZAÇÃO

1. Introdução

O presente trabalho insere-se no âmbito da unidade curricular Arquitetura e Padrões de Software, do Mestrado em Engenharia Informática e Tecnologia Web, e tem como objeto de estudo o desenvolvimento e evolução do *Activity Provider ProPlan*, integrado na arquitetura *Inven!RA*. Ao longo do semestre, o projeto foi sendo construído de forma incremental, acompanhando a introdução progressiva de conceitos arquiteturais e padrões de projeto, culminando, nesta fase final, numa análise crítica da qualidade estrutural da solução desenvolvida.

Ao contrário de uma perspetiva estritamente funcional, em que o foco recai sobre a correção dos resultados produzidos pelo sistema, este trabalho assume como premissa central a distinção entre funcionalidade e qualidade arquitetural. Um sistema pode cumprir integralmente os requisitos funcionais e, ainda assim, apresentar fragilidades significativas ao nível da sua estrutura interna, comprometendo a manutenibilidade, extensibilidade e evolução futura. É precisamente nesta dimensão que se insere o contributo da análise de antipadrões e da aplicação consciente de refatorização.

Assim, o objetivo desta etapa do projeto não consiste em adicionar novas funcionalidades ao ProPlan, mas sim em avaliar criticamente as decisões de desenho adotadas, identificar um antipadrão emergente no contexto real do projeto e aplicar uma refatorização arquitetural fundamentada, preservando o comportamento observável do sistema e a compatibilidade com a especificação da *Inven!RA*. Esta abordagem permite consolidar aprendizagens ao nível da engenharia de software, reforçando a importância da reflexão arquitetural como parte integrante do processo de desenvolvimento.

2. Enquadramento Teórico

Antipadrões em Engenharia de Software

Os antipadrões constituem um corpo de conhecimento complementar ao dos padrões de projeto, resultante da observação sistemática de soluções recorrentes que, apesar de frequentemente adotadas, conduzem a consequências negativas a médio e longo prazo. Enquanto os padrões de projeto descrevem soluções comprovadas para problemas recorrentes, os antipadrões descrevem respostas igualmente recorrentes, mas estruturalmente inadequadas, acompanhadas das respetivas estratégias de correção.

Um antipadrão não corresponde a um erro isolado ou a uma simples má prática pontual. Pelo contrário, caracteriza-se por ser reconhecível, repetitivo e frequentemente motivado por decisões aparentemente racionais em contextos de pressão, incerteza ou restrição de recursos. A sua perigosidade reside no facto de muitas vezes, funcionar inicialmente, mascarando os seus efeitos colaterais até que estes se tornem difíceis e dispendiosos de corrigir.

Entre os efeitos típicos associados aos antipadrões encontram-se o aumento da complexidade acidental, o acoplamento excessivo entre componentes, a redução da coesão interna e a dificuldade crescente em introduzir mudanças sem provocar regressões.

Antipadrões, decisões sob restrição e dívida técnica

Importa sublinhar que muitos antipadrões não emergem por desconhecimento técnico, mas sim como consequência de decisões tomadas sob restrições reais, como prazos apertados, requisitos voláteis ou necessidade de validação rápida de conceitos. Neste sentido, existe uma relação estreita entre antipadrões e dívida técnica, em particular quando esta é introduzida de forma consciente.

Contudo, a fronteira entre dívida técnica estratégica e antipadrão consolidado reside menos na decisão inicial e mais na ausência de mecanismos que promovam a sua reversão. Quando uma solução estruturalmente frágil é mantida indefinidamente, sem plano ou oportunidade de refatorização, a dívida técnica deixa de ser transitória e cristaliza sob a forma de um antipadrão.

Esta perspetiva reforça a importância de reconhecer os antipadrões não como falhas morais do código, mas como sinais de alerta que devem informar decisões arquiteturais futuras.

Refatorização como resposta aos antipadrões

A refatorização consiste na modificação da estrutura interna de um sistema com o objetivo de melhorar a sua qualidade, sem alterar o seu comportamento observável. No contexto dos antipadrões, a refatorização assume um papel central, pois fornece um conjunto de estratégias concretas para mitigar ou eliminar os problemas identificados.

Ao contrário de intervenções corretivas pontuais, a refatorização associada a um antipadrão deve ser entendida como uma decisão arquitetural, frequentemente com impacto transversal no sistema. Por esse motivo, a sua aplicação requer compreensão do contexto, avaliação de *trade-offs* e validação cuidadosa, nomeadamente através de testes de regressão.

Neste trabalho, a refatorização é utilizada como instrumento para transformar uma estrutura funcionalmente correta, mas arquiteturalmente frágil, numa solução mais coesa, legível e preparada para evolução, mantendo a compatibilidade com os contratos externos definidos pela *Inven!RA*.

3. Descrição do Projeto ProPlan

O ProPlan é um Activity Provider desenvolvido para integração na arquitetura *Inven!RA*, tendo como objetivo suportar atividades pedagógicas relacionadas com a simulação e análise de processos de gestão de projetos. O serviço disponibiliza um conjunto de web

services RESTful que permitem à plataforma anfitriã configurar atividades, instanciar sessões específicas e recolher dados analíticos sobre a execução das mesmas.

Do ponto de vista funcional, o ProPlan implementa os serviços previstos pela especificação da *Inven!RA,* nomeadamente os endpoints associados à configuração da atividade, ao *deployment* de instâncias e à disponibilização de analytics quantitativos e qualitativos. Estes dados são atualmente gerados de forma fictícia (mock), uma vez que o foco do projeto reside na arquitetura e não na persistência real de informação.

Arquiteturalmente, o projeto foi inicialmente desenvolvido com uma estrutura simples, concentrando grande parte da lógica de orquestração no módulo responsável pelos endpoints HTTP. Embora esta abordagem fosse adequada numa fase inicial, revelou limitações à medida que o sistema evoluiu, motivando a análise crítica apresentada nas secções seguintes.

4. Identificação do Antipadrão

Antipadrão principal: *God Object / The Blob* (emergente)

A análise do projeto permitiu identificar a emergência do antipadrão *God Object*, também designado *The Blob,* caracterizado pela concentração excessiva de responsabilidades num único componente. No contexto do ProPlan, este fenómeno manifestava-se de forma incipiente no módulo app.py, que acumulava funções relacionadas com:

- definição de endpoints *HTTP*;
- validação de parâmetros;
- leitura de ficheiros *JSON*;
- lógica de *deploy*;
- geração de respostas de *analytics*;
- interação com observadores.

Embora funcionalmente correto, este acoplamento excessivo comprometia a coesão do módulo e aumentava o risco de fragilidade estrutural, dificultando a manutenção e a evolução futura do sistema.

Antipadrões secundários e potenciais

Para além do antipadrão principal identificado, foram também considerados antipadrões secundários ou potenciais:

- ***Cut-and-Paste Programming (potencial):*** risco de duplicação de lógica associada à leitura e interpretação de esquemas JSON.
- ***Poltergeists (potencial):*** surgimento de classes excessivamente pequenas e transitórias caso a refatorização não fosse cuidadosamente planeada.

A refatorização aplicada procurou mitigar estes riscos, evitando tanto a concentração excessiva como a fragmentação artificial da lógica do sistema.

5. Decisão Arquitetural e Refatorização Aplicada

A refatorização realizada teve como objetivo central reduzir a concentração de responsabilidades, sem alterar o comportamento observável do sistema nem os contratos definidos pela *Inven!RA*. A decisão arquitetural adotada baseou-se na introdução de duas abstrações principais: uma camada de Application Service e um Repositório agregador.

A camada de *Application Service* passou a assumir a orquestração dos casos de uso, funcionando como intermediária entre os endpoints *HTTP* e a lógica de domínio. Desta forma, os controladores Flask foram deliberadamente mantidos "magros", limitando-se ao tratamento de pedidos e respostas *HTTP*.

Complementarmente, foi introduzido um repositório responsável por encapsular o acesso a dados de configuração, contratos de analytics e estado de *deploy*, reduzindo o acoplamento entre a fachada e os detalhes de infraestrutura.

Esta refatorização permitiu mitigar o antipadrão identificado, reforçando a coesão dos componentes e preparando o sistema para evolução futura.

6. Aplicação dos Padrões de Projeto

Factory Method

O padrão Factory Method foi aplicado na criação de repositórios de analytics, permitindo isolar a lógica de instanciamento e preparar o sistema para a introdução futura de diferentes fontes de dados, sem impacto nos componentes consumidores.

Facade

O padrão Facade é materializado no componente *ProPlanServiceFacade*, que atua como ponto único de acesso à lógica interna do sistema. Esta fachada valida pré-condições, coordena o acesso ao repositório, aplica serialização e publica eventos de domínio, garantindo uma separação clara entre interface e implementação.

Observer

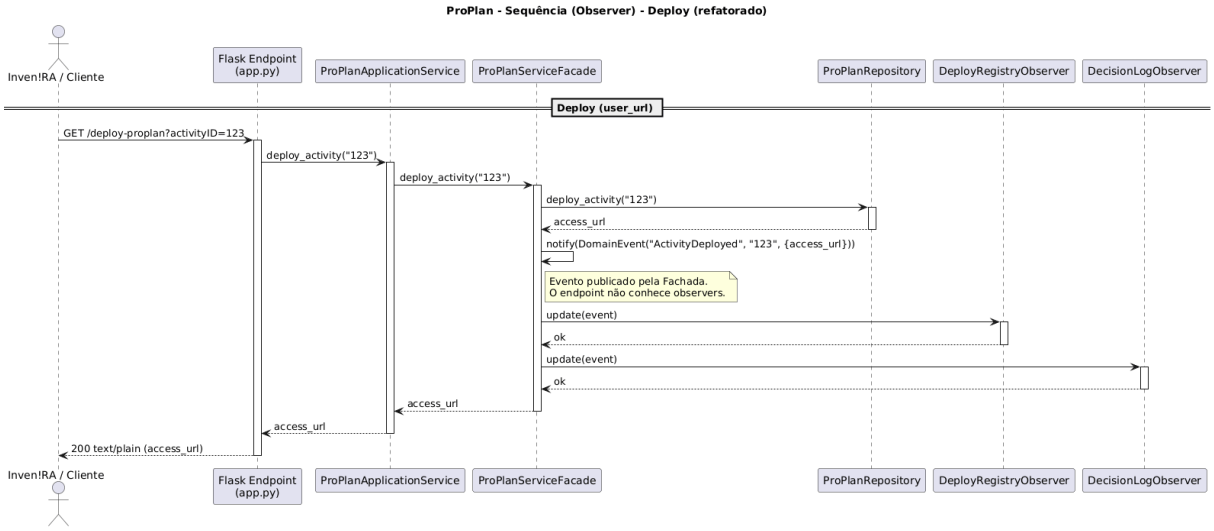
O padrão *Observer* foi aplicado para reagir a eventos relevantes do domínio, como o *deployment* de uma atividade ou a solicitação de analytics. A fachada atua como *Subject*, emitindo eventos que são consumidos por observadores independentes, responsáveis por registo de *deploy*, contagem de pedidos e manutenção de registos qualitativos.

A aplicação deste padrão promove baixo acoplamento e facilita a extensão futura do sistema com novos comportamentos reativos.

7. Diagramas Arquiteturais

Diagrama de Sequência (Observer)

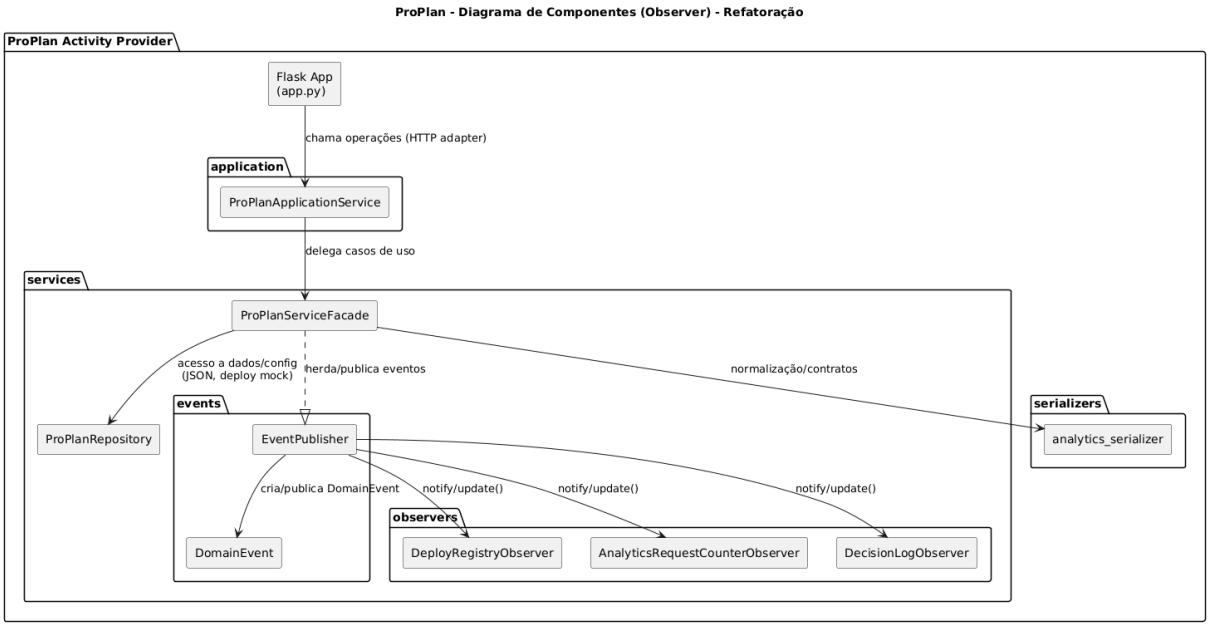
O diagrama de sequência representa o cenário de *deploy* de uma atividade, evidenciando a cadeia de delegação entre o endpoint HTTP, o Application Service, a fachada e o repositório, bem como a publicação do evento *ActivityDeployed*.



- *AnalyticsRequestCounterObserver* não é representado neste cenário por não reagir ao evento de *deploy*;
- o método *notify*() parte explicitamente da Fachada;
- o diagrama é mantido intencionalmente limpo, focando apenas o padrão aplicado.

Diagrama de Componentes

O diagrama de componentes apresenta apenas os componentes relevantes após a refatorização, destacando a introdução da camada de Application Service e do repositório agregador, bem como as dependências principais entre módulos. Esta representação reforça a separação de responsabilidades e a clareza arquitetural do sistema.



8. Validação da Solução

A validação da solução foi realizada através de testes funcionais no ambiente de *deployment* disponibilizado pela plataforma *Render*. Todos os endpoints definidos responderam corretamente, mantendo total compatibilidade com a especificação da *Inven!RA*.

Importa salientar que a refatorização não introduziu regressões funcionais, confirmando que o comportamento observável do sistema foi preservado, conforme o objetivo inicialmente definido.

9. Discussão Crítica

A refatorização aplicada demonstra que a identificação de antipadrões não implica necessariamente erros graves, mas antes oportunidades de melhoria estrutural. A introdução de novas camadas e abstrações representa um *trade-off* entre simplicidade inicial e robustez a médio prazo.

Embora o sistema permaneça relativamente simples, as decisões tomadas preparam o ProPlan para evolução futura, reduzindo o custo de mudança e promovendo boas práticas de engenharia de software.

10. Conclusão

Neste trabalho foi realizada uma análise crítica do projeto ProPlan, identificando a emergência de um antipadrão estrutural e aplicando uma refatorização arquitetural fundamentada. A separação de responsabilidades introduzida, aliada à aplicação consciente de padrões de projeto, permitiu melhorar significativamente a qualidade interna do sistema.

Esta experiência reforça a importância da reflexão arquitetural contínua e da análise de antipadrões como instrumentos essenciais na prática da engenharia de software.

Referências Bibliográficas

BROWN, William H.; MALVEAU, Raphael C.; McCORMICK III, Hays W.; MOWBRAY, Thomas J. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. New York: John Wiley & Sons, 1998.

FOWLER, Martin. Refactoring: Improving the Design of Existing Code. 2. ed. Boston: Addison-Wesley, 2019.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. Design Patterns: Elements of Reusable Object-Oriented Software. Reading: Addison-Wesley, 1995.

PAREDES, Hugo; MORGADO, Leonel. Antipadrões. Unidade Curricular Arquitetura e Padrões de Software. Universidade Aberta, 2020.

SOUSA, André. *ProPlan – Activity Provider*. Repositório de código-fonte. GitHub, 2026. Disponível em: https://github.com/AndreMacielSousa/proplan-activity-provider. Acesso em: 27 jan. 2026.

SOUSA, André. *ProPlan – Activity Provider (ambiente de execução)*. Render, 2026. Disponível em: https://proplan-activity-provider.onrender.com/. Acesso em: 27 jan. 2026.

SOUSA, André. *Arquitetura e Padrões de Software – Materiais da Unidade Curricular*. Repositório GitHub, 2025. Disponível em: https://github.com/AndreMacielSousa/ArquitecturaPadroesSoftware2025. Acesso em: 27 jan. 2026.