



# **De Invasivo a Convidado: A Nova Era da Extensibilidade no S/4HANA Cloud**

## **Dominando Cloud BAdIs para um "Clean Core" Robusto e à Prova de Upgrades**

Esta apresentação é um guia prático para desenvolvedores ABAP que buscam dominar o novo paradigma de extensibilidade no S/4HANA Cloud, abandonando as práticas legadas por uma abordagem moderna, segura e estável.

# Nossa Jornada de Aprendizagem



## Compreender o Paradigma

Entender o papel arquitetural dos Cloud BAdIs como a evolução segura dos User Exits.



## Dominar a Descoberta

Navegar e localizar BAdIs liberados (Released) exclusivamente via ADT (Eclipse).



## Implementar com Confiança

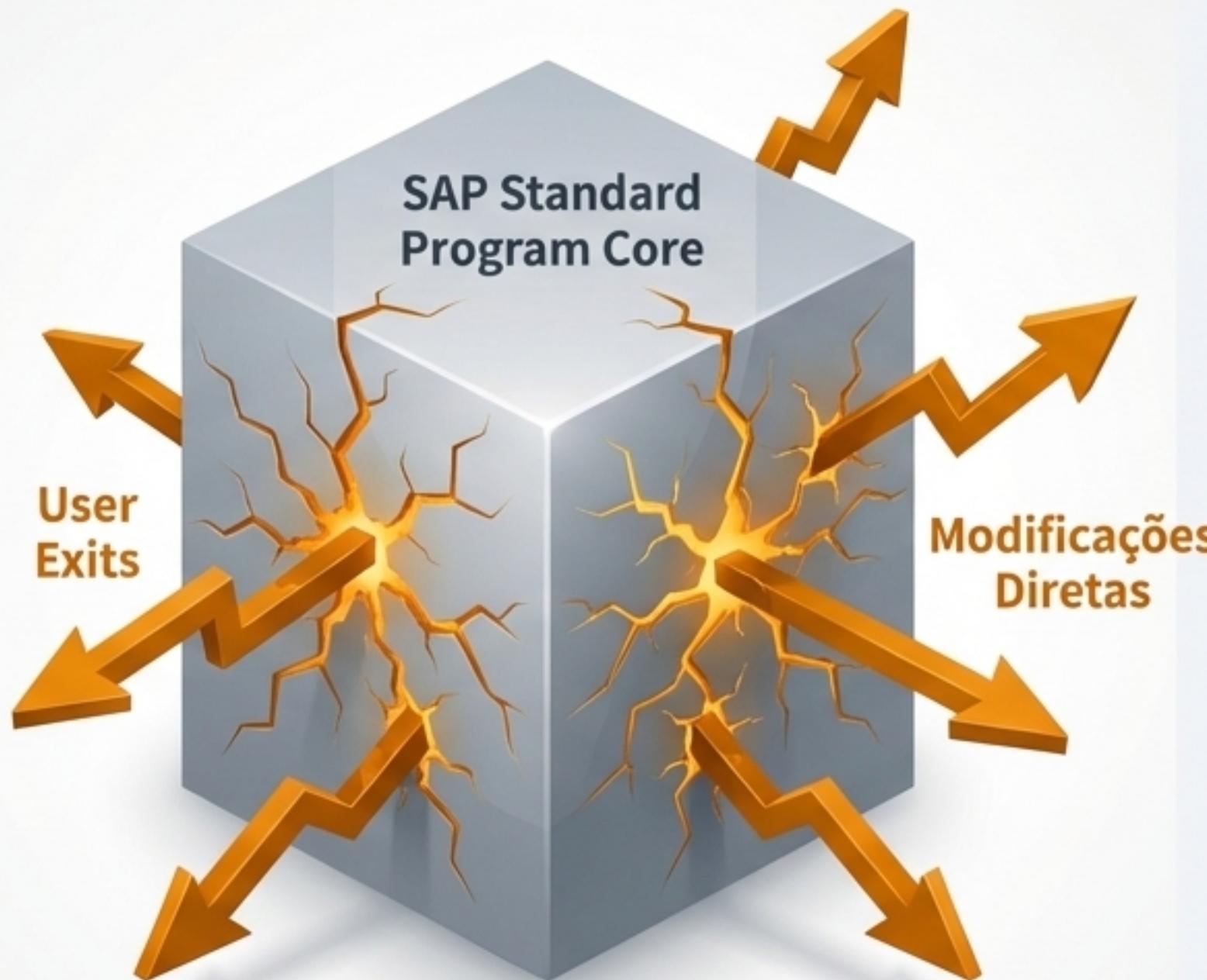
Criar uma Enhancement Implementation completa para cenários de validação e modificação.



## Respeitar as Regras

Justificar as limitações técnicas (ex: proibição de COMMIT) e sua relação com a integridade da LUW.

# O Paradigma Legado: A Era da Extensibilidade Invasiva

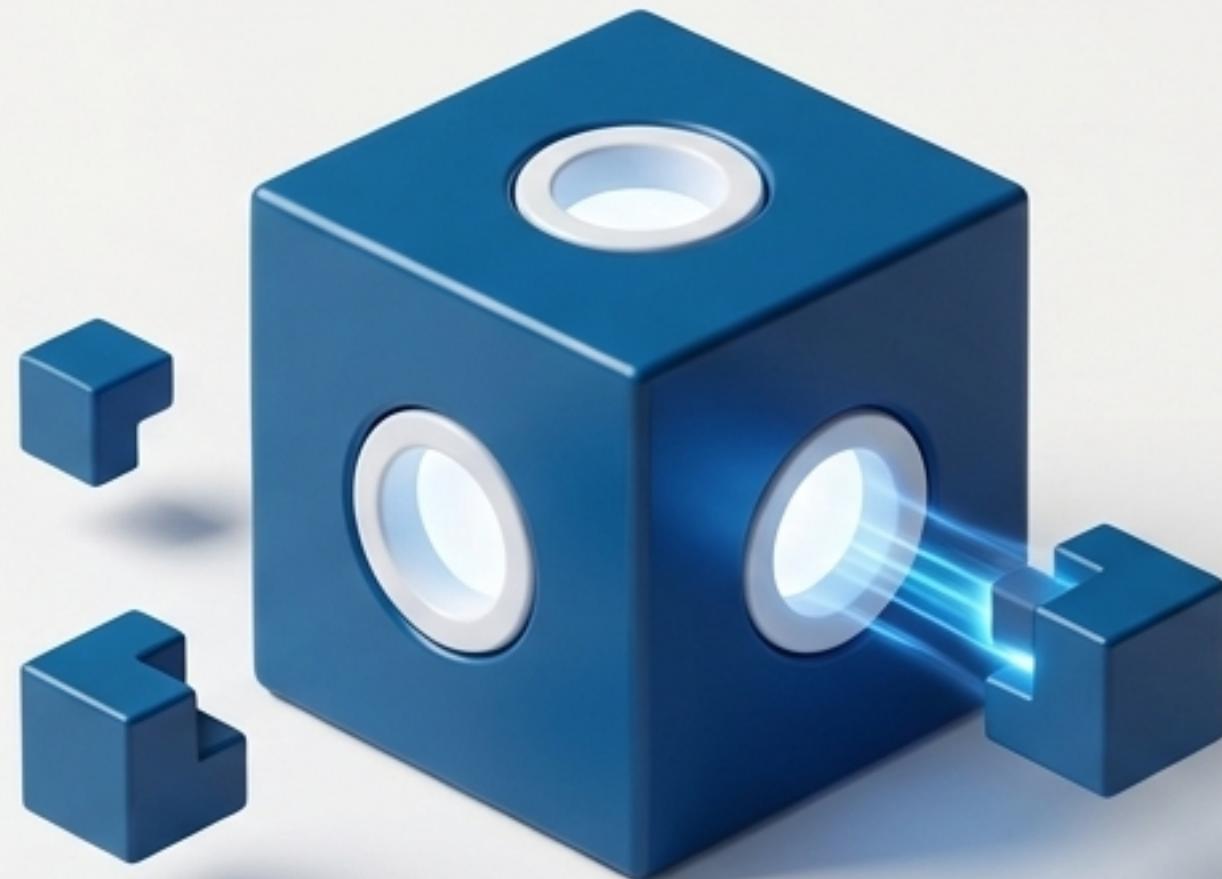


No SAP ECC, a extensibilidade era frequentemente uma “cirurgia de alto risco”. Modificávamos diretamente Includes de sistema (como o famoso `MV45AFZZ`) ou usávamos Customer Exits (CMOD/SMOD) que dependiam de código procedural e variáveis globais.

## O Problema da Abordagem Clássica:

- ⌚ **Fragilidade:** Uma simples mudança no programa standard da SAP durante um upgrade poderia quebrar a lógica customizada.
- ⌚ **Conflito de Nomes:** A falta de isolamento de variáveis levava a colisões e comportamentos imprevisíveis ("Variable Shadowing").
- ⌚ **Sequenciamento Caótico:** Múltiplos projetos no mesmo exit transformavam o código em uma 'colcha de retalhos' de `IFs`, impossível de gerenciar.

# A Solução 'Clean Core': Bem-vindo aos Cloud BAdIs



A SAP substituiu os mecanismos legados por uma abordagem moderna. Cloud BAdIs são pontos de extensão explícitos, seguros e liberados para a nuvem. Não "invadimos" mais o código; somos "convidados" a estendê-lo em pontos pré-definidos.

## Principais Vantagens:

- ⚙️ **Orientação a Objetos:** Cada BAdI é uma Interface ABAP com um contrato claro: você sabe exatamente o que entra (`Importing`) e o que pode ser modificado (`Changing`).
- ⚙️ **Estabilidade Contratual:** A SAP garante que a assinatura do método do BAdI não mudará em upgrades, protegendo seu desenvolvimento.
- ⚙️ **Múltiplas Implementações:** O framework suporta múltiplas implementações ativas, permitindo que diferentes soluções coexistam sem conflitos.

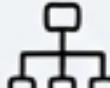
# Confronto de Paradigmas: Extensão Clássica vs. 'Clean Core'

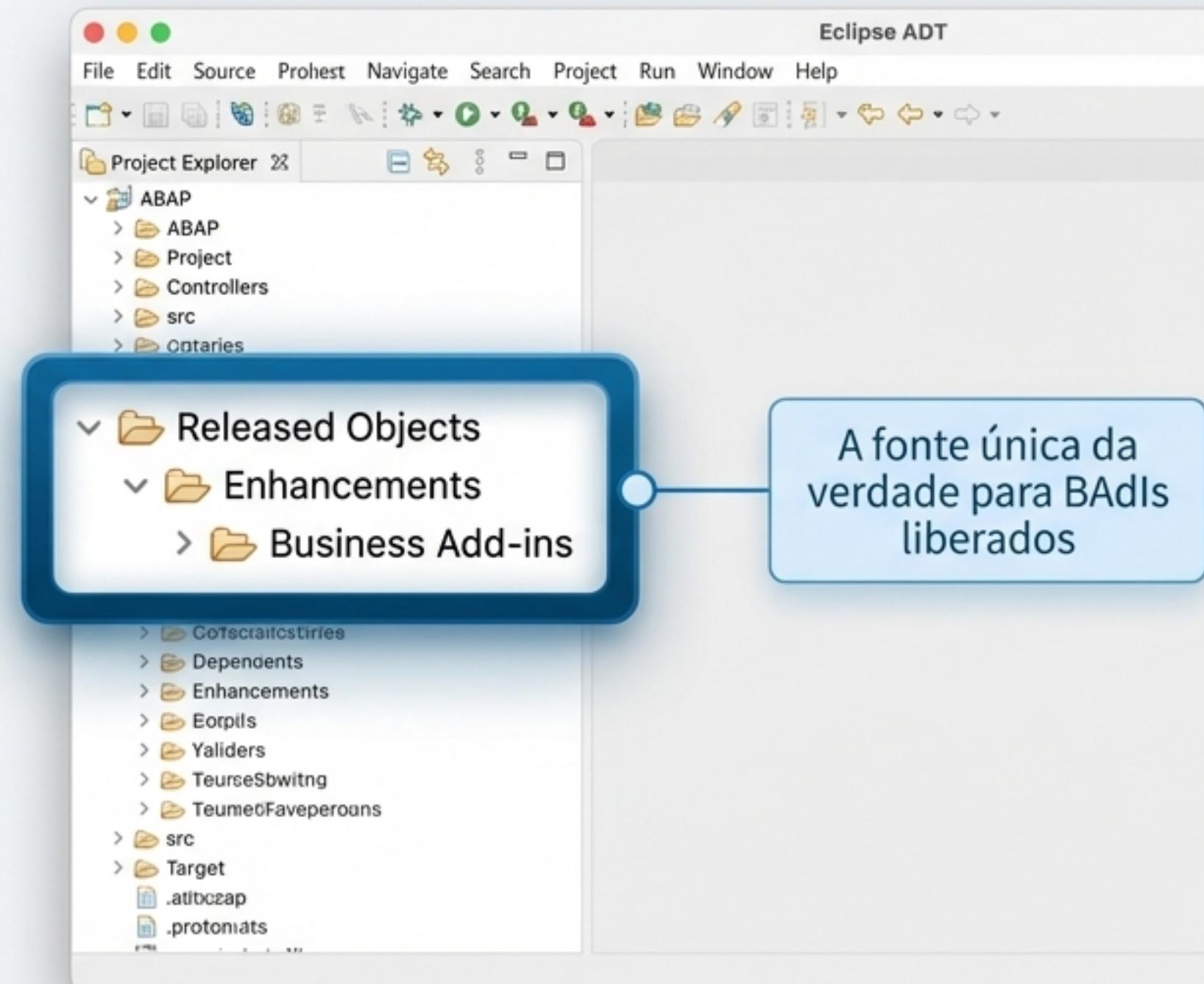
Característica	User Exit / Customer Exit (Legado)	Cloud BAdI (Moderno)
Tecnologia	Subrotinas (FORM), Function Modules.	 Classes e Interfaces ABAP OO.
Estabilidade de Upgrade	Baixa. Dependente de variáveis globais.	 Alta. Contrato de Interface garantido pela SAP.
Múltiplas Implementações	Geralmente Não. Conflitos de código.	 Sim. Framework suporta coexistência.
Filtros de Execução	Não. `IFs` manuais e ineficientes no código.	 Sim. Filtro otimizado pelo Kernel ABAP.

# Onde Encontrar os Pontos de Extensão?

## Adeus SE18, Olá ADT (Eclipse)

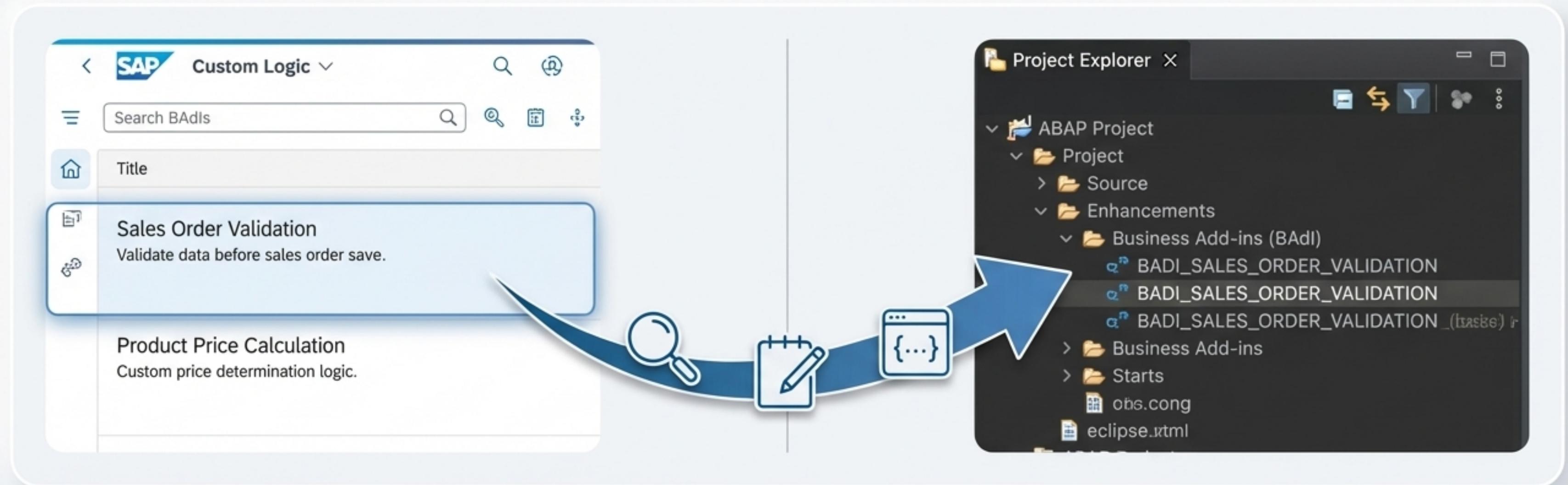
No ABAP Cloud, a descoberta de pontos de extensão acontece exclusivamente no **ABAP Development Tools (Eclipse)**. A transação SE18 mostra BAdIs não liberados e inseguros para a nuvem.

-  1. Abra a view **Project Explorer** no ADT.
-  2. Navegue pela árvore de objetos:  
**Released Objects > Enhancements > Business Add-ins.**
-  3. **Resultado:** Esta árvore exibe *apenas* os BAdIs que a SAP marcou como seguros para uso (Released API), garantindo a estabilidade da sua solução.



# Dica de Produtividade: Descobrindo BAdIs com o App Fiori

Para uma visão mais funcional e de negócio, consultores e Key Users podem utilizar o aplicativo Fiori “Custom Logic”.



1. **\*\*Busca Visual\*\***: Use o app para buscar BAdIs por processo de negócio e ler a documentação funcional de forma amigável.
2. **\*\*Identificação Técnica\*\***: Anote o nome técnico do BAdI encontrado no app.
3. **\*\*Implementação no ADT\*\***: Volte para o Eclipse e use o nome técnico para implementar a lógica com controle total do código.

# O Fluxo de Implementação de um BAdI no ADT



## Criar o Container (Enhancement Implementation)

**Ação:** Clique com o botão direito no seu Pacote > **New** > **BAdI Enhancement Implementation**.

**Descrição:** Este objeto (`.badi`) agrupa suas implementações e define filtros.



## Escolher a Definição (BAdI Definition)

**Ação:** No assistente, selecione o BAdI standard que deseja estender (ex: `SD\_SLS\_CHECK\_HEAD`).

**Descrição:** Isso define o "contrato" que sua classe deve seguir.



## Implementar a Classe (Implementation Class)

**Ação:** O ADT gera automaticamente a classe Z que implementa a interface do BAdI.

**Descrição:** Seu trabalho é preencher a lógica de negócios nos métodos vazios.

# Exemplo Prático: Validando um Pedido de Venda

- \*\*Cenário\*\*: Regra de negócio: impedir a criação de um Pedido de Venda se o campo 'Referência do Cliente' (`PurchaseOrderByCustomer`) estiver vazio.
- \*\*BAdI Utilizado\*\*: SD\_SLS\_CHECK\_HEAD

```
CLASS zcl_sd_check_po IMPLEMENTATION.  
METHOD if_sd_sls_check_head~check_header.  
    IF sales_order_header-purchase_order_by_customer IS INITIAL.  
        APPEND VALUE #(  
            msgid  = 'ZMSG_SALES'  
            msgno  = '001'  
            msgty  = 'E'  
            msgv1  = 'Campo Referência é obrigatório'  
        ) TO messages.  
  
        check_result = if_sd_sls_check_head->rejection.  
    ENDIF.  
ENDMETHOD.  
ENDCLASS.
```

O parâmetro 'sales\_order\_header' contém os dados do pedido em processamento.

Para rejeitar, adicionamos uma mensagem à tabela 'messages'. Não usamos 'MESSAGE ... RAISE'.

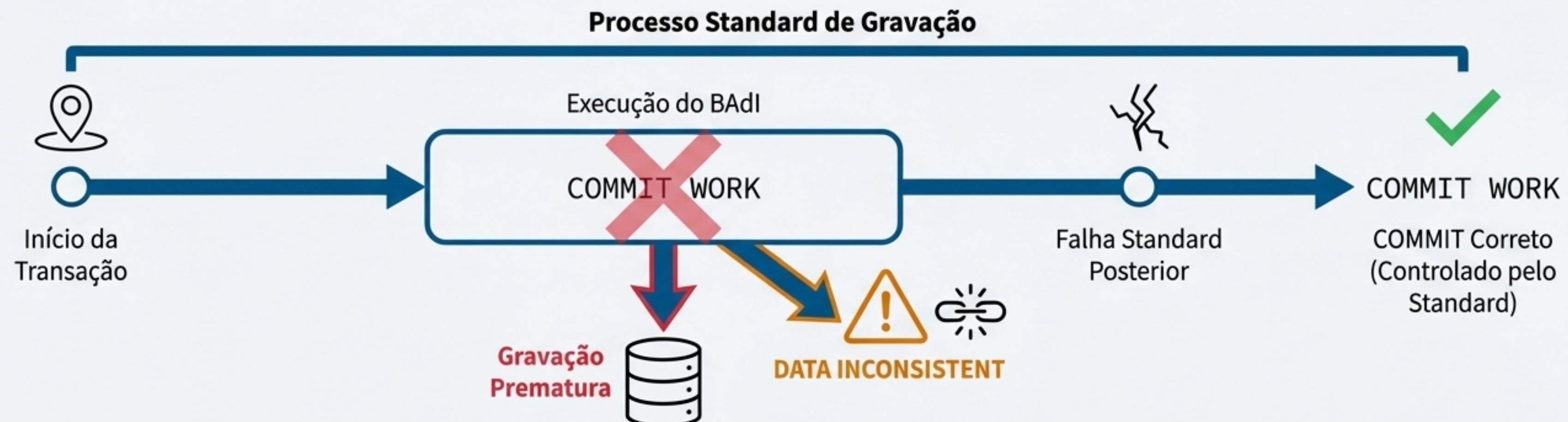
Sinalizamos explicitamente a falha ao framework. Isso garante que a transação seja abortada de forma limpa.

# As Regras do Jogo: Protegendo a Integridade do Sistema

Por que não posso dar `COMMIT WORK`?

Seu código BAdI opera como um convidado dentro da transação do processo standard. A integridade da **LUW (Logical Unit of Work)** é sagrada.

**O Risco:** Se você emitir um `COMMIT WORK`, você força a gravação prematura dos dados. Se o processo standard falhar logo depois, o resultado é uma **inconsistência grave de dados** (metade gravada, metade não). O processo pai controla o `COMMIT` final.



## Sem Acesso Direto

Nunca use `UPDATE` direto em tabelas standard (VBAK, MARA). Modifique dados apenas através dos parâmetros `CHANGING` da interface.



## Performance é Essencial

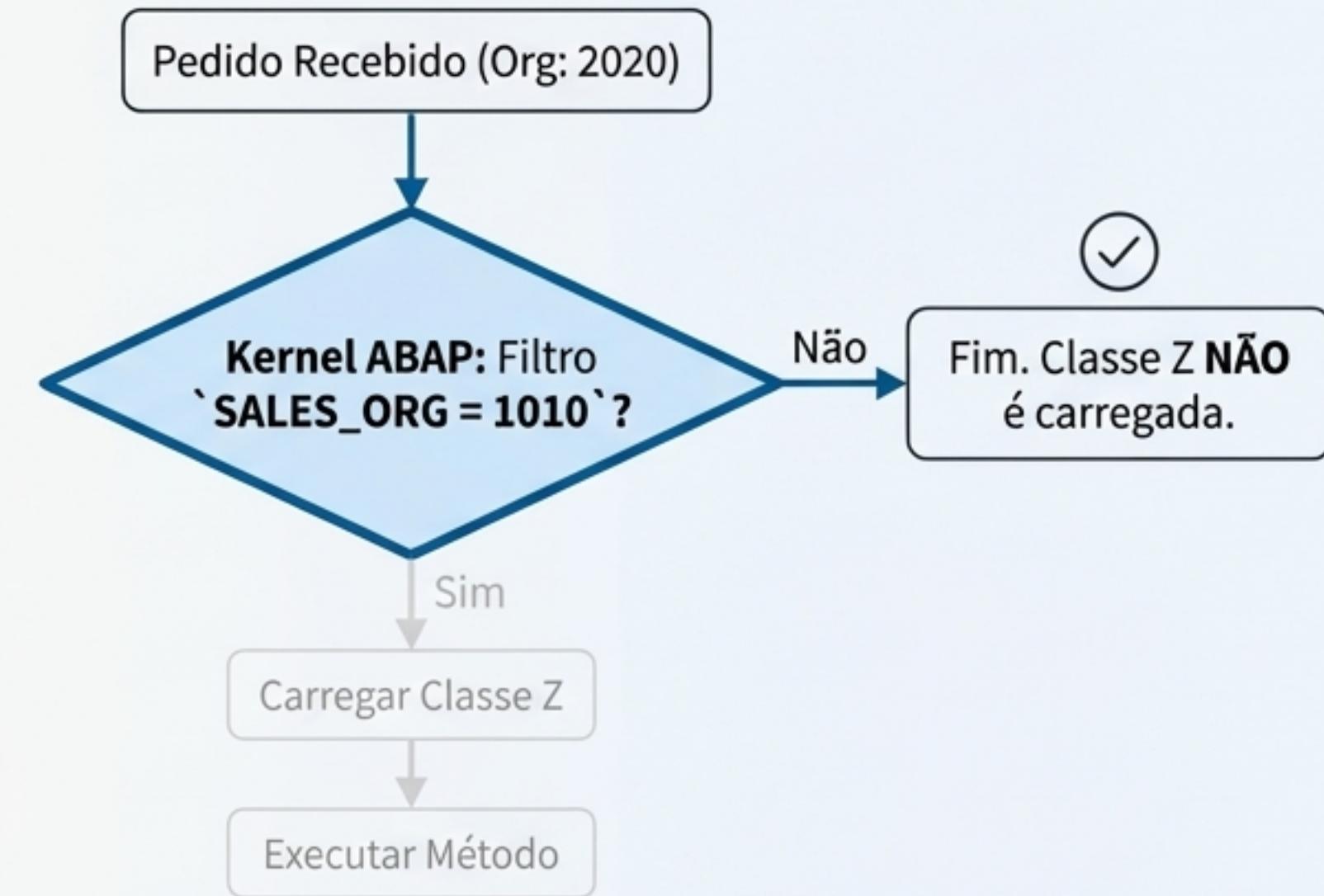
O código roda de forma síncrona. Um BAdI lento significa um processo de negócio lento. Evite `SELECTs` pesados ou chamadas de API síncronas.

# Otimização de Performance com Filtros de BAdI

## Abordagem Ineficiente (`IF` no Código)



## A Solução Inteligente (BAdI Filters)



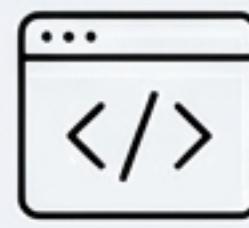
- **O Problema:** Escrever ``IF sales_org = '1010'`` dentro do código é ineficiente. O sistema precisa carregar e executar sua classe para todos os pedidos, apenas para não fazer nada na maioria das vezes.
- **O Ganho de Performance:** O Kernel do ABAP verifica o filtro *antes* de instanciar sua classe. Se a condição não for atendida, sua classe Z nem sequer é carregada na memória. Isso garante performance e escalabilidade máximas.

# Glossário Essencial do Desenvolvedor ‘Clean Core’



## Cloud BAdI

Ponto de extensão OO, liberado pela SAP (Released API), que permite injetar lógica customizada em processos standard, garantindo segurança contra upgrades.



## Enhancement Implementation

O artefato técnico no ADT (`.`badi`) que serve como container para a classe de implementação e para a configuração dos filtros de um BAdI.



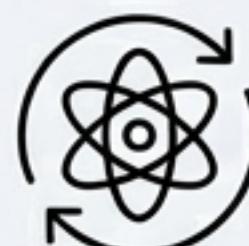
## BAdI Interface

O contrato técnico (`IF\_...`) que define quais métodos sua classe deve implementar e quais parâmetros estão disponíveis.



## Filter Value

Configuração declarativa que restringe a execução de um BAdI a critérios específicos (ex: Empresa ‘1000’), otimizando a performance ao evitar o carregamento desnecessário da classe.



## LUW (Logical Unit of Work)

Uma sequência de operações de banco de dados que devem ser tratadas como uma unidade atômica (tudo ou nada). BAdIs operam dentro da LUW do processo pai.

# Sua Nova Missão: De Invasor a Arquiteto Convidado

A transição para Cloud BAdIs é mais do que uma mudança técnica; é uma mudança de filosofia. Abandonamos a mentalidade de “modificar o standard a qualquer custo” e adotamos a disciplina de estender o sistema em pontos seguros e bem definidos.



**Core Limpo (Clean Core):** Seu sistema permanece estável, previsível e fácil de manter.



**Upgrades Sem Susto:** As atualizações do S/4HANA Cloud se tornam eventos de rotina, não projetos de crise.



**Valor Sustentável:** Suas extensões são robustas, escaláveis e alinhadas com a estratégia de longo prazo da SAP.

