UNIVERSIDADE
FEDERAL DO CEARÁ

Code smells em frameworks front-end

Disciplina: Qualidade de Software
Equipe: Camile Isidorio Araujo e Francisco Werley da Silva
Projeto: https://github.com/taiga-family/taiga-ui.git
Fork: https://github.com/Camilemarqs/taiga-ui.git

Eu estou atualmente trabalhando na refatoração do seguinte code smell: Large Class (LC)
Minhas principais dificuldades na remoção do code smell são: 1. Manter compatibilidade: preservar todos os inputs públicos individuais para não quebrar templates HTML e testes existentes que fazem binding direto. 2. Angular Signals: trabalhar com input(), computed() e signal() sem alterar a API pública, organizando a lógica interna. 3. Organização sem quebrar funcionalidade: separar responsabilidades sem criar subcomponentes ou alterar a interface pública. 4. Tipos e imports: garantir que as interfaces usem os tipos corretos (ex: TuiContext vem de @taiga-ui/cdk/types, não de @taiga-ui/polymorpheus).
Eu estou usando os seguintes métodos de refatoração para remover o code smell: 1. Extract Interface / Introduce Parameter Object: criação de interfaces para agrupar conceitos relacionados: 2. Organize Data (Grouping Related Data): inputs organizados por responsabilidade em seções comentadas. 3. Extract Configuration Objects: uso de computed() signals privados que agregam inputs relacionados, permitindo que a lógica interna trabalhe com objetos agrupados. 4. Preserve API / Wrapper Pattern: manter inputs públicos originais para compatibilidade; lógica interna usando configurações agrupadas. 5. Add Comments / Section Organization: comentários de seção para identificar responsabilidades (Serviços, Inputs, Estado, Computed Values, Métodos, etc.). 6. Extract Helper Methods: organização de métodos privados em grupos lógicos (validação, cálculos, inicialização, scroll, etc.).

Eu estou atualmente trabalhando na refatoração do seguinte code smell: Long Function (LF)
Minhas principais dificuldades na remoção do code smell são: 1. Identificar pontos de extração que reduzam duplicação sem afetar a funcionalidade 2. Balancear granularidade: funções/métodos pequenos demais vs. grandes demais 3. Organizar blocos de teste extensos em grupos lógicos sem perder contexto

4. Evitar que helpers genéricos percam clareza ou aumentem a complexidade
Eu estou usando os seguintes métodos de refatoração para remover o code smell: 1. Extract Method/Function — extraí métodos privados helper (ex: calculateTotalMilliseconds, extractTimeUnits em time.ts) 2. Extract Constant — movi strings literais longas e repetitivas para constantes (ex: EXPECTED_PACKAGE_JSON_MAIN) 3. Extract Function Helper — criei funções helper reutilizáveis para reduzir duplicação (ex: createAttrReplacement, createMainTsFile) 4. Decomposição de Test Blocks — quebrei describe/it grandes em blocos menores agrupados logicamente (ex: day.spec.ts — 7 blocos; month.spec.ts — grupos por tipo) 5. Extract Setup Logic — movi inicializações comuns para beforeEach hooks (ex: day-range.spec.ts) 6. Grouping/Organization — agrupei dados e testes por categoria/tipo para melhor organização

Eu estou atualmente trabalhando na refatoração do seguinte code smell: ANY (Overusing Any Type)
Minhas principais dificuldades na remoção do code smell são: 1. A principal dificuldade é que muitos usos de any aparecem em partes genéricas do framework, como InjectionToken, utilitários de tipagem avançada e funções reutilizáveis. Nesses casos, o tipo real dos dados não é explícito, o que exige uma análise para identificar qual contrato semântico aquele any realmente representa. 2. Substituir any por tipos muito restritivos pode quebrar a inferência de tipos ou tornar o código menos flexível, exigindo um equilíbrio entre segurança de tipo e reutilização.
Eu estou usando os seguintes métodos de refatoração para remover o code smell: 1. Introdução de contratos explícitos (interfaces e tipos) para substituir any por abstrações semânticas. 2. Uso de tipos genéricos para preservar flexibilidade sem perder segurança de tipo. 3. Substituição de any por unknown em casos onde o tipo exato não é conhecido, forçando validações explícitas. 4. Criação de tipos de contexto e contratos de componentes em InjectionToken e serviços, evitando objetos ou componentes dinamicamente tipados.

Eu estou atualmente trabalhando na refatoração do seguinte code smell: IIC (Inheritance Instead of Composition)
Minhas principais dificuldades na remoção do code smell são:

1. A principal dificuldade é que as classes-base no framework concentram muitos comportamentos importantes. Isso faz com que componentes que herdam essas classes passem a depender implicitamente de funcionalidades que não utilizam, e para conseguir refatorar é precisa uma análise de quais comportamentos são realmente utilizados.

Eu estou usando os seguintes métodos de refatoração para remover o code smell:

1. Substituição de herança por composição, extraindo comportamentos de superclasses em serviços ou interfaces independentes.
2. Introdução de contratos (interfaces) para que os componentes dependam de abstrações em vez de implementações concretas.
3. Uso de injeção de dependência para montar os comportamentos necessários em tempo de execução, em vez de fixá-los na hierarquia de classes.