Aspectos organizacionais: programas de reuso, serviços de suporte

QXD0068 - Reuso de Software

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br

Agenda

- Tipos de reuso na prática
- Por que reuso precisa ser organizacional?

Tipos de Reuso

- **Reuso Vertical**
  - Dentro de um mesmo domínio.
  - Exemplo: fábrica de software bancária criando múltiplos sistemas de internet banking a partir de um modelo genérico.

- **Reuso Horizontal**
  - Em diferentes domínios.
  - Exemplo: bibliotecas matemáticas, bibliotecas gráficas (Qt, GTK), bibliotecas de UI (Material UI).

Tipos de Reuso

- **Reuso Planejado**
  - Estratégia formal com apoio gerencial.
  - Exemplo: fábricas de software buscando certificação CMMI.

- **Reuso Composicional**
  - Construção de novos sistemas a partir de blocos/componentes.
  - Exemplo: frameworks (React, Angular, JSF, Spring Boot).

Tipos de Reuso

- **Reuso Baseado em Geradores de Código**
  - No nível de especificação.
  - Exemplo: CASE tools, UML, MDD (Model Driven Development), OpenAPI → gera código de APIs automaticamente.

- **Reuso Caixa Branca**
  - Necessidade de conhecer e modificar a implementação.
  - Exemplo: herança em Java/C++, refatoração com acesso ao código.

Tipos de Reuso

- **Reuso Caixa Preta**
  - Uso apenas via interfaces/contratos, sem precisar ver implementação.
  - Exemplo: APIs REST, bibliotecas fechadas, componentes de terceiros.

- **Reuso de Código-Fonte**
  - Mais comum na prática.
  - Exemplo: copiar/migrar classes utilitárias entre projetos.

Tipos de Reuso

- **Reuso de Projetos**
  - Reuso em nível maior → padrões de projeto, frameworks de arquitetura.
  - Exemplo: padrões GoF, arquitetura MVC, Clean Architecture.

- **Reuso de Especificação**
  - Especificação → projeto → código.
  - Exemplo: reusar diagramas UML ou requisitos de sistemas similares.

Por que reuso precisa ser organizacional?

- Reuso técnico isolado (copiar código, usar biblioteca open source) não garante consistência nem economia real.

- Só gera impacto quando a organização decide institucionalizar.

- Analogia:
  - Reuso ad hoc → cozinheiro improvisando receita.
  - Reuso organizacional → restaurante com ficha técnica, fornecedores padronizados, estoque controlado.

O que significa institucionalizar?

- Política oficial da empresa/universidade/órgão público.
- Processos definidos: ciclo de vida do artefato (criação → aprovação → distribuição → descontinuação).
- Infraestrutura: repositórios, métricas, suporte.
- Cultura: os times confiam nos artefatos e preferem reusar ao invés de reinventar.

Benefícios esperados

- Redução de custo e esforço.
- Maior consistência e qualidade.
- Time-to-market mais rápido.
- Melhor manutenção e evolução.

Programas de Reuso

- Conjunto estruturado de políticas, práticas, incentivos e métricas para institucionalizar o reuso.

- Objetivo: fazer o reuso deixar de ser escolha individual e se tornar parte da estratégia organizacional.

Políticas Organizacionais

- **Reuso como prioridade estratégica**
  - Empresas de software maduras tratam o reuso como ativo central, não como “atividade opcional”.
  - Exemplo: Microsoft e Google têm políticas internas de reuso para evitar retrabalho entre times.

- **Padrões mínimos de documentação e qualidade**
  - Artefatos só podem entrar no repositório de reuso se atenderem requisitos mínimos (testes unitários, cobertura mínima de código, guia de uso).
  - Exemplo: um componente de autenticação só é liberado se tiver README com exemplos de integração e testes automatizados.

- **Reuso em auditoria de software**
  - Projetos passam por checklists de conformidade, avaliando o uso de componentes já disponíveis antes de reinventar.
  - Exemplo: auditoria ISO/IEC 12207 ou CMMI pode exigir evidências de reuso.

Estrutura de Governança

- **Gestor de Reuso**
  - Profissional responsável por manter o catálogo de componentes e monitorar sua utilização.
  - Atua como “curador” garantindo que só entrem artefatos de qualidade.

- **Comitês técnicos**
  - Reúnem representantes de áreas diferentes (backend, frontend, QA) para homologar componentes.
  - Exemplo: decidir se um framework de logging será padrão corporativo.

- **Políticas de versionamento e ciclo de vida**
  - Definir quando um componente entra em fase de “manutenção”, “obsolescência” e “descontinuação”.
  - Exemplo: componente de login v1.0 continua disponível, mas novos projetos devem migrar para v2.0 até uma data limite.

Estratégias de Implantação

- **Bottom-up**
  - Inicia em times pequenos que criam boas práticas de reuso organicamente.
  - Vantagem: adesão natural.
  - Risco: falta de padronização se não for centralizado depois.

- **Top-down**
  - Alta gestão impõe políticas obrigatórias.
  - Vantagem: padronização rápida.
  - Risco: resistência dos times, se não houver incentivo.

- **Híbrida**
  - Mistura de diretrizes corporativas com autonomia dos times.
  - Exemplo: empresa define que todos devem usar repositório interno, mas não obriga tecnologias específicas.

Incentivos e Métricas

- **Metas de uso**
  - Exemplo: até 40% do código novo deve vir de componentes reutilizáveis.
  - Recompensas para equipes
  - Reconhecimento, bônus ou premiações internas para equipes cujos componentes são amplamente usados.

- **Indicadores de ROI (Retorno sobre o Investimento)**
  - Comparar custo de desenvolver “do zero” vs custo de reutilizar.
  - Exemplo: relatório trimestral mostrando economia de horas de desenvolvimento graças ao reuso.

Exemplos Reais de Programas de Reuso

- NASA: repositório de software aberto (NASAs Open Source Software Catalog) para garantir confiabilidade em missões.

- Empresas de telecom: bibliotecas comuns de protocolos (ex.: 5G stack) para reduzir falhas e duplicação de esforço.

- Startups de fintechs: criam SDKs internos de pagamentos para padronizar integração em múltiplos apps.

Serviços de Suporte

- Repositórios Corporativos
- Sistemas de Catalogação e Busca
- Ferramentas de Integração
- Treinamento e Capacitação
- Suporte Organizacional e Curadoria
- Medição e Acompanhamento

Repositórios Corporativos

- **Definição:**
  - Catálogo centralizado de componentes, com versionamento, documentação e metadados (autor, status, dependências, licença).
  - Evita que cada time “reinvente a roda” ou use versões diferentes do mesmo componente.

- **Tipos:**
  - Bibliotecas de código: repositórios internos do tipo npm, Maven, PyPI ou NuGet.
  - APIs internas: serviços documentados e versionados (exemplo: catálogo da AWS).
  - Serviços SOA/microserviços: funções de negócio expostas para múltiplos times (ex.: serviço de login, faturamento, notificações).

- **Exemplo:**
  - Netflix criou internamente componentes que depois se tornaram open source (Hystrix, Eureka, Zuul), reforçando a importância do reuso bem estruturado.

Sistemas de Catalogação e Busca

- **Importância:**
  - Repositório sem busca eficiente é praticamente inútil (“reuso morto”).
  - Desenvolvedor só reutiliza se encontrar rapidamente o que precisa.

- **Funcionalidades essenciais:**
  - Busca por nome, domínio de negócio e tags.
  - Selo de qualidade (ex.: “estável”, “deprecated”, “experimental”).
  - Visualização de dependências (quem usa, onde é usado).

- **Exemplo:**
  - IBM desenvolveu o Asset Locator, um sistema de indexação e busca de artefatos de software que facilitava localizar componentes reutilizáveis entre equipes globais.

Ferramentas de Integração

- **Papel no reuso:**
  - Garante que componentes sejam confiáveis e estáveis antes de serem publicados no catálogo.
  - Evita que bugs de um time se espalhem para toda a empresa.

- **Práticas comuns:**
  - CI/CD pipelines: testes automatizados, lint, verificação de vulnerabilidades.
  - Gate de segurança: só entra no catálogo quem passa nas regras (ex.: OWASP dependency check).

- **Exemplo:**
  - A Red Hat usa GitHub Actions + pipelines internos: toda biblioteca corporativa passa por testes funcionais e de segurança antes de ser liberada para outros times.

Treinamento e Capacitação

- **Problema comum:**
  - Sem treinamento, desenvolvedores continuam “refazendo do zero” porque não sabem que os componentes já existem.

- **Boas práticas:**
  - Onboarding: novos devs já aprendem a usar os repositórios internos.
  - Workshops: hackathons internos de reuso (ex.: “quem acha o maior número de componentes reutilizáveis ganha”).
  - Documentação viva: wiki, Confluence, GitBook, com exemplos de uso.

- **Exemplo:**
  - O Spotify treina novos devs no uso do Backstage, sua plataforma interna de catálogo de serviços, que virou referência open source.

Suporte Organizacional e Curadoria

- **Curadoria:**
  - Sem curadoria, o repositório vira “cemitério de código” → cheio de coisas desatualizadas e não confiáveis.
  - Reuso precisa de manutenção contínua, não é só “publicar e esquecer”.

- **Responsabilidades da equipe de curadoria:**
  - Retirar ou marcar como obsoletos componentes desatualizados.
  - Garantir que segurança e conformidade (ex.: LGPD, GDPR) sejam atendidas.
  - Exigir documentação mínima e boas práticas de versionamento.

- **Exemplo:**
  - Amazon exige que cada microserviço interno tenha um “service owner” oficial, responsável por sua manutenção.

Medição e Acompanhamento

- **Por que medir?**
  - “Não se gerencia o que não se mede”.
  - Reuso só mostra valor se houver métricas claras.

- **Indicadores típicos:**
  - Adoção: quantos times estão utilizando o repositório.
  - Qualidade: quantos bugs foram evitados pelo uso de componentes testados.
  - Produtividade: tempo de onboarding de novos devs antes vs depois.

- **Exemplo prático:**
  - Onboarding de devs: de 2 meses (sem reuso) → para 2 semanas (com catálogo organizado).
  - ROI: comparar custo de desenvolver do zero vs reutilizar (ex.: economia de 500h de dev em um trimestre).

Estudos de Caso

- Caso 1 — Startup
- Caso 2 — Empresa Tradicional
- Caso 3 — Governo (Brasil)

Caso 1 — Startup

- **Contexto:**
  - Startup em crescimento acelerado, várias equipes pequenas desenvolvendo produtos independentes.
  - Cada time criava seu próprio sistema de login e autenticação (usuário/senha, recuperação de senha, tokens).

- **Problema:**
  - Alto risco de inconsistência e vulnerabilidade de segurança.
  - Custos duplicados: cada time gastava tempo resolvendo o mesmo problema.
  - Experiência ruim para o usuário: cadastros e logins diferentes em cada produto.

Caso 1 — Startup

- **Solução:**
  - Desenvolvimento de uma API corporativa de autenticação (centralizada, segura, com tokens JWT e integração OAuth2).
  - Implantação em todos os novos projetos da startup.

- **Resultado:**
  - Redução de 60% dos bugs de segurança (falhas de login e vulnerabilidades de autenticação).
  - Mais agilidade no lançamento de novos produtos (tempo de desenvolvimento de login caiu de semanas para dias).
  - Experiência do usuário unificada (mesmo login para todos os sistemas).

Caso 2 — Empresa Tradicional

- **Contexto:**
  - Empresa consolidada, com décadas de operação e muitos sistemas legados desenvolvidos de forma independente.
  - Problema recorrente: cada sistema tinha suas próprias regras de negócio implementadas do zero.

- **Problema:**
  - Duplicação massiva de lógica (exemplo: cálculo de imposto, cadastro de cliente, relatórios).
  - Diferenças de implementação → inconsistência entre sistemas.
  - Dificuldade de integração entre departamentos.

Caso 2 — Empresa Tradicional

- **Solução:**
  - Criação de um catálogo corporativo de web services (baseados em SOA, depois evoluindo para microserviços).
  - Governança centralizada para homologar e versionar serviços.
  - Capacitação das equipes para buscar serviços existentes antes de programar.

- **Resultado:**
  - Hoje, 70% dos novos sistemas consomem serviços já existentes.
  - Menos inconsistência em relatórios e cálculos (mesma lógica usada por todos).
  - Redução do custo de manutenção de sistemas legados.

Caso 3 — Governo (Brasil)

- **Contexto:**
  - Governo Federal, com centenas de órgãos desenvolvendo sistemas próprios (educação, saúde, pesquisa, previdência).
  - Histórico de duplicação de esforços (cada órgão mantendo cadastros próprios de cidadãos e servidores).

- **Problema:**
  - Redundância de dados (mesmo cidadão cadastrado em dezenas de sistemas).
  - Custos elevados de manutenção e falta de integração.

Caso 3 — Governo (Brasil)

- **Solução:**
  - Criação de plataformas centrais:
    - Plataforma Lattes (CNPq) → base curricular de pesquisadores, usada por universidades e agências de fomento.
    - Gov.br → sistema unificado de autenticação para cidadãos, acessando múltiplos serviços públicos.
  - Abertura de APIs oficiais para que órgãos consumam dados sem duplicar cadastros.

- **Resultado:**
  - Redução significativa da duplicação de informações.
  - Maior integração entre serviços (ex.: autenticar no Gov.br e acessar INSS, Receita Federal, Siorg, etc.).
  - Base confiável e auditável, aumentando transparência.

Discussão

- O que esses três casos têm em comum?
  - Reuso estruturado melhora qualidade, reduz custo e aumenta padronização.

- O que muda entre eles?
  - Escala: startup (time pequeno), empresa tradicional (sistemas legados), governo (integração nacional).

- Como aplicar esses aprendizados em contextos acadêmicos ou locais?
  - Ex.: universidade poderia ter repositório interno de módulos de ensino, em vez de cada professor criar seu próprio sistema isolado.

Bibliografia Básica

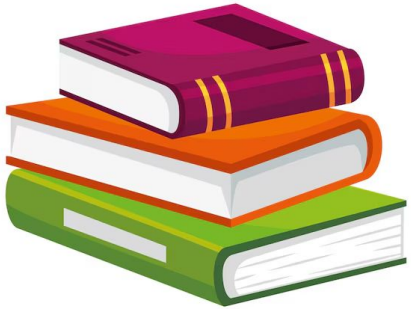
- MILI, Hafedh. Reuse-based software engineering: techniques, organization and measurement. New York: Wiley, 2002. 636 p. ISBN 0471398195.

- EZRAN, M.; MORISIO, M.; TULLY, C. Practical software reuse. Berlim: Springer, 2002

- POHL, K.; BÖCKLE, G.; LINDEN, F. J. Software product line engineering: foundations, principles and techniques. Berlim: Springer, c2010. xxvi, 467 p. ISBN 9783642063640 (broch.).

Bibliografia Complementar

- SOMMERVILLE, I. Engenharia de software. 7. ed. São Paulo: Pearson Addison-Wesley, 2007.
- PRESSMAN, R. Engenharia de software. 6. ed. São Paulo: Mc Graw-Hill,2006. ISBN 8586804576
- GAMMA, E.; HELM, JOHNSON, R.; R.; VLISSIDES , J. Padrões de projeto: soluções reutilizáveis de software orientado a objetos. Porto Alegre: Bookman, 2000. 364 p.
- BRAUDE, Eric J.; BERNSTEIN, Michael E. Software engineering: modern approaches. 2nd ed. Hoboken, New Jersey: J. Wiley & Sons, 2011. xvi, 782 p.
- BASS, Len; CLEMENTS, Paul; KAZMAN, Rick. Software architecture in practice. 3rd ed. Upper Saddle River, NJ: Addison-Wesley, c2013. xix, 589 p. (SEI series in software engineering). ISBN 9780321815736 (enc.).



Obrigado!

Dúvidas?

Universidade Federal do Ceará - *Campus* Quixadá

Prof. Francisco Victor da Silva Pinheiro
victorpinheiro@ufc.br