Proposta de Padrões de Software para a Reutilização Sistemática em Sistemas de Software Orientados a Objetos[1]

Gabriela Elisa da Cunha, Juliana Silva Herbert

ESICenter UNISINOS
Universidade do Vale do Rio dos Sinos – UNISINOS
Av. Unisinos, 950. São Leopoldo, RS

{Gabriela, Juliana}@exatas.unisinos.br

Abstract *The technology of the reuse grew substantially in the latest years. With the increase of levels of reuse in the process of developing systems, the organizations have reduced significantly its costs and have reached incomparable profits in the quality of the final product. Besides, all these factors are associated with the reduction of development time. As a result of this, the software patterns appeared and they not only describes standard solutions, but also give subsidies to support the solution. From the analysis of relations between aspects considered important for the reuse, it was possible to suggest specific patterns to describe solutions in this area, contributing, thus, in the research area related to quality improvement of the object-oriented software.*

Resumo: *A tecnologia da reutilização tem se destacado substancialmente nos últimos anos. Através do aumento dos níveis de reaproveitamento no processo de desenvolver sistemas, as organizações buscam reduzir significativamente seus custos e alcançar ganhos incomparáveis em qualidade de produto, associados à redução do tempo de desenvolvimento. Como decorrência disso, emergem os padrões de software, que além de descrever soluções padrão, também fornecem subsídios para apoiar a solução. A partir da análise de correlações entre aspectos considerados importantes para a reutilização, são propostos padrões de software específicos para descrever soluções nesta área, de forma a contribuir aperfeiçoamento da qualidade do software orientado a objetos.*

1. Introdução

O processo de desenvolvimento de sistemas tem sido marcado pela constante busca de qualidade e de produtividade. Dessa forma, percebem-se consideráveis investimentos em metodologia, treinamento e processos de aferições de qualidade [1]. O surgimento do paradigma de desenvolvimento de sistemas orientado a objetos (OO) contribuiu para o alcance destes objetivos, motivando e subsidiando a reutilização de elementos previamente construídos e testados.

Várias linhas de estudo em orientação a objetos têm sido promissoras para a melhoria da qualidade de desenvolvimento. Dentre elas, os esforços voltados para a obtenção de reutilização merecem destaque, uma vez que contribuem diretamente para o alcance dos objetivos de diminuição de custos e de aumento de produtividade no desenvolvimento de sistemas.

Já foi constatado em vários trabalhos [4] [6] um aumento significativo em reutilização e o conseqüente incremento de qualidade, devido à utilização de *frameworks*. A utilização destas estruturas de aplicação desencadeou a necessidade de documentar formas e regras para a obtenção das melhores soluções. São os chamados padrões de software, mais comumente associados ao âmbito do projeto e do teste do sistema.

1.1 Contribuição

Ao ser identificada uma lacuna no âmbito da documentação de soluções padrão, foi possível propor uma inovação na área de descrição de soluções: os *reuse patterns*, ou padrões de software específicos para a reutilização. Para a confecção de tais padrões de software considerou-se a modelagem de elementos OO através da *Unified Modeling Language* (UML), relacionada à aplicação de métricas de qualidade e aos decorrentes ganhos em reutilização. Esta nova forma de documentar soluções permite o incremento da qualidade e da produtividade no desenvolvimento OO.

Até agora, algumas categorias de descrições de solução padrão já obtiveram aceitação entre projetistas e desenvolvedores de sistemas, tais como os padrões de software para projeto [4]. Outras são mais recentes, tais como os padrões de software para teste [5]. Ambas as categorias abordam fases específicas do ciclo de desenvolvimento. Os padrões de software para reutilização, cuja proposição é apresentada neste artigo, caracterizam-se por identificar aspectos importantes para a reutilização em qualquer etapa do ciclo de desenvolvimento de sistemas. Na pesquisa realizada para a elaboração deste trabalho não foi encontrada nenhuma categoria de padrões de software similar à proposta neste artigo.

A presente seção apresentou a motivação para a realização deste trabalho, assim como destacou sua principal contribuição à área, citando, em seguida, alguns trabalhos relacionados.

A Seção 2 apresenta a importância da UML para a obtenção de ganhos em reutilização através da modelagem de elementos OO. Na Seção 3 justifica-se a adoção de métodos de reutilização em todas as fases de desenvolvimento de sistemas. A Seção 4 salienta a utilização de padrões de software para o compartilhamento das melhores soluções em software. Na Seção 5, é apresentada a proposta dos padrões de software

para reutilização como uma nova categoria de padrões de software, baseada em relacionamentos existentes entre alguns aspectos considerados importantes para a reutilização em um ambiente de desenvolvimento de sistemas OO. A Seção 6 apresenta considerações sobre a realização deste trabalho, assim como extensões imediatas possíveis a partir destas idéias. E, finalmente, na Seção 7, são apresentadas as principais referências bibliográficas utilizadas para a realização deste trabalho.

2. *Unified Modeling Language* – UML

A UML é uma linguagem de modelagem que agrega conceitos pertencentes a um consenso que vários autores possuíam da OO [7]. A UML destina-se a delimitar fronteiras de um sistema, modelar sua estrutura estática e seu comportamento, além de permitir visualizar a funcionalidade e a implementação. Como significativa contribuição para o desenvolvimento, a UML comporta conceitos considerados de alto nível, como *frameworks* e padrões de projeto (*design patterns*) [6] [4].

A UML foi considerada a linguagem de modelagem de sistemas OO mais adequada para a utilização nos padrões de software para reutilização, uma vez que pode ser considerada simples, poderosa e independente do domínio no qual é aplicada, além de estar notoriamente se transformando em um padrão para modelagem.

3. Reutilização

A principal motivação para a reutilização está relacionada ao aumento dos níveis de qualidade e produtividade no desenvolvimento de software [1]. O aumento de qualidade é uma conseqüência da reutilização de componentes que foram previamente documentados, testados e aprovados. O aumento da produtividade é resultado de uma redução no tempo de desenvolvimento, evitando a reconstrução de partes do sistema que já existem.

A reutilização sugere níveis mais altos de abstração, dos quais resultam as estruturas de aplicação (*frameworks*). Considera-se como *framework* uma arquitetura semi-pronta de aplicação, que consiste em um conjunto de classes que são especificamente desenhadas para serem refinadas e usadas como um grupo. Este conjunto de classes incorpora um desenho abstrato do domínio do problema e encoraja a reutilização. Os *frameworks* representam o nível mais alto de reutilização. Sua aplicabilidade sugere a necessidade da adoção de modelos de solução padrão, que são denominados padrões de software. Além destes, existem os *metapatterns* (Subseção 4.2), que são tipos de padrões de software para projetos (Subseção 4.1), propostos por Pree [6], que suportam o desenvolvimento e a reutilização de *frameworks* na fase de projeto OO.

A reutilização caracteriza-se como uma das principais motivações para a concentração de esforços no desenvolvimento de padrões de software. Para a obtenção destas soluções padrão, é necessário identificar os elementos estáveis do sistema, que poderão fazer parte de várias soluções para problemas similares. O entendimento dos aspectos fundamentais que contribuem para a reutilização é de fundamental importância para que seja possível propor padrões de software específicos de reutilização.

4. Padrões de Software

Apesar de todas as melhorias já propostas no processo de desenvolvimento de sistemas, as organizações ainda precisam estruturar o seu conhecimento para minimizar problemas na comunicação das soluções e práticas adotadas [8]. Neste sentido, a utilização de padrões de software tem-se fixado como uma das abordagens mais promissores voltadas à melhoria da qualidade de desenvolvimento. Estes padrões têm como característica fundamental a identificação e o registro apropriado da solução, para que seja possível disseminá-la posteriormente.

O padrão de software será sempre oriundo de alguma experiência prática, embora não possa ser validado através de testes. A sua identificação é muito importante, devendo, portanto, direcionar o pensamento do usuário para o problema em questão. Um padrão de software é necessário para documentar, de forma genérica, a solução de um problema importante e de ocorrência repetida.

A Subseção 4.1 apresenta os padrões de software para projetos e a sua importância. A utilização de meta-padrões de software é abordada na Subseção 4.2. Por fim, a Sub-seção 4.3 compreende as linguagens específicas para a descrição de *padrões de software*.

4.1 Padrões de Software para Projeto

De maneira geral, os padrões de software para projetos podem ser entendidos como uma descrição da comunicação entre classes e objetos destinada a resolver um problema genérico de projeto em um contexto particular [4]. São padrões de software destinados a descrever soluções simples e elegantes para problemas específicos do projeto OO, sendo elaborados a partir da captura de soluções genéricas que foram desenvolvidas e evoluíram ao longo do tempo, resultando em uma forma sucinta e fácil de ser aplicada.

O catálogo de padrões de software para projetos, proposto por Gamma [4], já obteve aceitação no âmbito da Engenharia de Software por representar o processo de captura, abstração e classificação dos aspectos importantes para a criação de um projeto OO reutilizável.

4.2 Meta-padrões de Software

Entende-se por meta-padrões de software o conjunto de padrões de software para projetos que descreve como construir *frameworks*, independente de um domínio específico [6]. Os meta-padrões de software constituem uma abordagem poderosa que pode ser aplicada para categorizar e descrever qualquer padrão de software para projeto que exemplifique um *framework*. Desta forma, os meta-padrões de software não substituem a especificação dos padrões de software para projetos, mas servem como complemento, auxiliando na sua documentação.

4.3 *Linguagem de Padrões de Software*

A linguagem de padrões de software é a coleção de padrões de software, obtidos em todos os níveis de abstração, que oferece um trabalho conjunto na resolução de um problema complexo, cuja solução sugere uma linha de conduta [8]. Desta forma, cada padrões de software, então, possui uma instrução explícita de como se realiza algo.

A linguagem de padrões de software proposta por Coplien [2] tem como objetivo apoiar as técnicas emergentes na comunidade de projetos de software, sendo voltada, principalmente, para a evolução organizacional. Coplien sugere a estrutura dos padrões de software em seis seções:

- problema: descrição do problema a ser resolvido;
- contexto: descrição do ambiente que abrange a solução do problema e das circunstâncias envolvidas;
- forças: relação das características relevantes que têm influência no problema;
- contexto resultante: descrição do ambiente resultante após a aplicação da solução;
- racionalidade: descrição das razões e justificativas para a adoção da solução proposta.

A estrutura desta linguagem foi utilizada para descrever os padrões de software para reutilização. Desta forma, a comunicação das idéias de reutilização, propostas neste artigo, utiliza um formato de linguagem de padrões de software já conhecida e divulgada na literatura relacionada.

5. Padrões de Software para Reutilização

O conhecimento a respeito dos domínios de aplicação deve ser reutilizado com o objetivo de gerar soluções padrão. Para tanto, é necessário identificar os elementos estáveis que estão presentes em várias soluções de problemas similares. Os resultados em reutilização dependerão do entendimento destes aspectos de estabilidade ou variabilidade do domínio do problema.

Este trabalho propõe uma nova categoria de descrições de solução padrão: padrões de software específicos para a reutilização. Estes padrões de software não foram obtidos a partir de resultados práticos porque, até o presente momento, não foram adequadamente aplicados (esta é uma das extensões futuras mais imediatas no seguimento deste trabalho). Eles foram deduzidos a partir de estudos das características de OO mais relevantes para o estabelecimento de um bom nível de reutilização. A identificação destas características OO, potencializadoras de reutilização, visa ocorrer em qualquer fase ou atividade do processo de desenvolvimento de sistemas, não caracterizando como objetivo único a reutilização de código. Para a descrição dos padrões de software para reutilização, utilizou-se a linguagem de padrões de software proposta por Coplien [2], por ser considerada a mais adequada para a documentação das soluções padrão em reaproveitamento. Além dos itens propostos por esta linguagem, para cada solução padrão são apresentados, também, seus usos conhecidos.

Acredita-se que os padrões de software para reutilização possam servir de apoio às ferramentas CASE *(Computer Aided Software Engineering)* OO, na avaliação da modelagem em relação aos indicativos fundamentais de reutilização. A ferramenta de modelagem Rational Rose [7], é atualmente a principal ferramenta de modelagem de sistemas OO, através da UML. Repositórios de elementos reutilizáveis constituem outra maneira de aplicar este tipo de padrão de software, sob forma de prover uma melhor documentação e a conseqüente organização dos seus elementos.

As Tabelas 1, 2 e 3 apresentam, de forma completa alguns padrões de software para reutilização do catálogo proposto em [3].

O padrão de software para reutilização"Quantidade de vezes que a Classe é Referenciada", explicado através da Tabela 1, foi identificado com base na análise de estruturas de aplicação *frameworks*, e indica a confecção de classes abstratas como sendo o tipo de classe de maior índice de reutilização. Estas classes, porém, devem ser planejadas no projeto do sistema, pois necessitam de maior atenção na fase de análise e geram maior custo. As estruturas de aplicação semi-prontas são constituídas, em grande parte, de classes abstratas. Decorre daí a justificativa para a análise dos pontos de flexibilidade do sistema que poderão ser reaproveitados no domínio da aplicação.

O padrão de software para reutilização"Implementação da Arquitetura em Camadas", pertinente à Tabela 2, sugere a organização do sistema sob a forma de camadas de abstração. A utilização deste tipo de estrutura implica em uma análise profunda e detalhada dos requisitos do sistema, bem como a identificação dos objetos que serão necessários. Cada objeto, por sua vez, atua em uma das quatro camadas estruturais. A comunicação entre objetos de camadas distintas deve respeitar o fluxo "Interface-Controle-Dados-Negócio", para que sejam asseguradas ao sistema as características de flexibilidade e manutenibilidade e atomicidade, visto que os objetos contemplam apenas a sua própria responsabilidade.

Por sua vez, o padrão de software para reutilização "Utilizando Coleções no Modelo de Classes", apresentado na Tabela 3, tem como objetivo avaliar o tratamento dos relacionamentos 1:N no Modelo de Classes. A partir dos requisitos advindos do mundo real, reconhece-se a existência deste tipo de relacionamento e, assim, a importância de modelar de maneira adequada a responsabilidade existente entre as funcionalidades. Desta forma, considerando o âmbito da orientação a objetos, propõem-se a utilização de coleções no modelo de classes, com o intuito de representar as classes instanciadas de um relacionamento.

Tabela 1 – Padrão de Software para Reutilização 1 – Quantidade de vezes que a Classe é Referenciada.

Problema	Como identificar se uma classe abstrata deve ser criada?
Contexto	A classe foi identificada e deve-se verificar se ela oferece condições de ser uma classe abstrata.
Forças	As classes abstratas são constituídas através da união de conceitos genéricos, que serão adequados ao domínio da aplicação através apenas de suas subclasses. As classes abstratas requerem um esforço maior na fase de análise. Classes abstratas são as candidatas em potencial para a reutilização, já que são modeladas para este fim. Elas representam a melhor solução para reutilizar comportamentos não associados diretamente ao domínio da aplicação. Uma classe com muita funcionalidade torna-se mais difícil de ser reutilizada, uma vez que tende a ser mais complexa.
Solução	Identificar, no domínio da aplicação, as classes cuja responsabilidade poderia ser generalizada para facilitar a reutilização. Como as classes abstratas são mais complexas, elas devem justificar a sua existência. Devem ser definidas para servirem ao domínio público da organização, deixando flexível a implementação específica. A definição do comportamento das classes abstratas deve ser sob a forma de métodos virtuais. Testar muito bem a classe, identificá-la de forma clara e associar a ela uma documentação. Construir classes pequenas, de funcionalidade específica.
Contexto	Excelente nível de reutilização. O modelo do sistema torna-se mais robusto e

Resultante	flexível a mudanças. Permite manutenções mais rápidas. Força uma utilização correta dos conceitos de herança. Biblioteca com classes reutilizáveis bem documentadas e de fácil entendimento.
Racionalidade	A experiência comprova que em um modelo de sistemas, de 10 a 15% das classes devem ser abstratas. A intenção em reutilizar a classe pode acarretar em manutenções na classe, sob forma de seu aprimoramento.
Usos Conhecidos	Fortemente utilizado para derivar uma métrica de projeto, no que se refere à potencialidade da classe abstrata. Esta solução padrão apresenta validade no contexto de projeto de classes do modelo do sistema, podendo ser aplicado tanto a uma estrutura de sistema do tipo *framework*, quanto a um sistema de informação (neste caso com mais ênfase nas classes de interface, de controle e de negócio do que na classe de dados, de acordo com a divisão de pacotes proposta pela UML).

Tabela 2 – Padrão de Software para Reutilização 2 – Implementação da Arquitetura em Camadas.

Problema	Como estabelecer uma arquitetura em camadas?
Contexto	O projeto de sistemas OO deve possuir uma arquitetura adequada para a implementação de especificações do modelo de classes.
Forças	Consideram-se camadas da arquitetura os vários níveis de abstração entre os diversos serviços que o sistema deve possuir. O conhecimento dos objetos do sistema e de seus relacionamentos é imprescindível para o estabelecimento deste tipo de arquitetura. Devem ser conhecidas as dependências entre os objetos para poder estabelecer a camada na qual estes devem estar situados. É importante, ainda na fase de modelagem, identificar os objetos que terão função de controle. Estes objetos serão geralmente responsáveis pela coordenação das regras do negócio.
Solução	Utilizar o conceito de pacotes, propostos pela UML [7]. Isto significa que o sistema deve possuir seus objetos dispostos sob a forma de pacotes, cuja organização é apresentada a seguir: • Pacote Interface: objetos de comunicação com o usuário final. • Pacote Controle: objetos de lógica transacional. Funcionam como gerentes que coordenam o trabalho de uma equipe de objetos. Podem atuar, inclusive, como seqüencializadores das atividades entre objetos dependentes. • Pacote do Negócio: objetos que contêm o conhecimento específico do negócio. • Pacote de Dados: objetos que compõem o modelo estável do sistema. Os pacotes devem comunicar-se apenas na seqüência através da qual foram apresentados, uma vez que há dependência entre eles. Por exemplo, um objeto de negócio não pode solicitar nada para um objeto de interface, sob pena de violar a dependência entre as camadas.
Contexto Resultante	A reutilização é a grande conseqüência. A partir da arquitetura em camadas, os elementos reutilizáveis, bem como sua atuação, são de fácil localização.

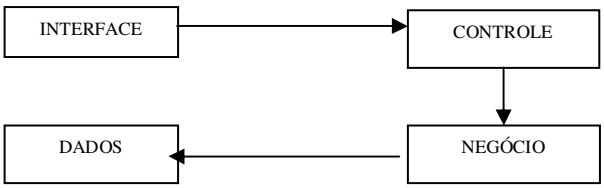
Racionalidade	A disposição dos sistemas que a organização pretende desenvolver deveria basear-se em:  Observa-se a importância da existência dos objetos de controle, mesmo em casos onde a classe é considerada bastante simples. Quando do acréscimo de funcionalidade, os objetos de controle são necessários para coordenar as atividades.
Usos Conhecidos	Oferece aplicabilidade para as fases de projeto e análise do sistema, mais especificamente nas atividades de modelagem. Entretanto, é importante observar que este conceito de pacotes necessita ser considerado de forma coordenada com a arquitetura da instalação para o desenvolvimento de software. A estruturação dos objetos do sistema em pacotes e o desenvolvimento de uma modelagem que respeite a forma de comunicação proposta entre os mesmos, propiciam uma melhor visibilidade sobre a distribuição da responsabilidade do sistema entre os objetos, favorecendo, assim, a localização de pontos motivadores para a reutilização.

Tabela 3 – Padrão de Software para Reutilização 3 – Utilizando Coleções no Modelo de Classes.

Problema	Como tratar um relacionamento de cardinalidade 1:N no Modelo de Classes
Contexto	Necessidade de relacionamentos do Modelo de Classes apresentarem cardinalidade 1:N.
Forças	Uma coleção conhece seu estado. Este estado pode também se aplicar a todas as suas partes, ou instâncias, seja por uma questão de proximidade física ou temporal. Se os relacionamentos 1:N forem resolvidos com a criação de classes de agregação, a responsabilidade existente nas classes pode variar e pertencer a mais de uma classe. A coleção deve ser utilizada em um modelo de classes quando é necessário representar as classes instanciadas de um relacionamento. Apenas algumas linguagens OO mais evoluídas suportam o conceito de coleção.
Solução	O relacionamento 1:N deve ser modelado através de um relacionamento com uma classe Coleção sempre que existe uma relação todo-parte (composição) em um domínio de negócio. A classe todo mantém um relacionamento 1:1 para a classe coleção, ou seja, a classe todo possui uma classe coleção de instâncias da classe parte. A classe parte mantém um relacionamento de 1:1 com a classe coleção. Utiliza-se o conceito de coleção quando é necessário obter informações tanto a respeito de uma determinada instância da classe parte, quanto a respeito de várias instâncias da classe parte.

Contexto Resultante	Modelo de Classes normalizado até a 5ª Forma Normal[2].
Racionalidade	Deve-se decidir, em tempo de projeto, qual será a maneira de implementar as coleções existentes no Modelo de Classes. No caso de poucos elementos, pode-se implementar a Coleção através de Listas. Muitos elementos requerem uma implementação mais elaborada, do tipo Algoritmo Hash ou árvore Btree. O ideal é que todas as classes coleção do projeto sejam implementadas da mesma maneira. Isto ocasiona a utilização de uma metaclasse para a classe coleção, o que configura um padrão de software aplicado no projeto. Como exemplo de aplicação de coleção pode-se considerar, no domínio acadêmico, o relacionamento Aluno possui um rol de Disciplinas. Estas disciplinas podem ser já aprovadas, não aprovadas, não cursadas ou sendo cursadas no semestre letivo atual. 
Usos Conhecidos	Este padrão de solução pode ser aplicado em contextos de modelagem de classes, em casos nos quais a organização possui uma linguagem e um sistema gerenciador de banco de dados que suportam o conceito de coleções. A forte aplicabilidade das coleções no modelo de classes ocorre quando se precisa resolver um relacionamento 1:N e, ao mesmo tempo, é importante que ambas as classes consigam responder sobre suas instâncias e, principalmente, sobre o relacionamento existente entre elas. A implementação de coleções no modelo facilita as manutenções futuras do modelo, além de garantir uma alocação pertinente de responsabilidade para as classes.

De forma resumida, as Tabelas 4, 5, 6 e 7 apresentam, os demais padrões de software específicos para reutilização propostos em [3].

Tabela 4 – Padrão de Software para Reutilização 4 – Quantidade de Colaboração em uma Classe.

Problema	Qual o volume de colaboração que deve existir em uma classe?
Solução	Avaliar novas requisições considerando todo o modelo, as relações de colaboração e as novas demandas. Inovações no sistema geram, muitas vezes, novas classes com novos métodos. A hierarquia deve ser avaliada, buscando colocar em um nível mais alto as classes com necessidade de adicionar mais métodos. Verificar a quantidade e a necessidade dos relacionamentos que a classe mantém com outros objetos do sistema, fator que está diretamente relacionado com a

[2] A 5ª Formal Normal caracteriza-se pela eliminação de dependências funcionais, utilizando a decomposição de um relacionamento sem ocasionar perda de informações [9].

	quantidade necessária de variáveis de instância. Avaliar, entre os métodos da classe, aqueles que sejam de responsabilidade de outra classe. Neste caso, pode ser necessária a fragmentação da classe. Incentivar, na fase de modelagem, a definição de métodos da superclasse com generalidade suficiente para atender as subclasses. Verificar a funcionalidade da subclasse, avaliando no modelo se era intenção sobrescrever os métodos que assim foram implementados.
Usos Conhecidos	Esta solução padrão apresenta potencial para ser utilizado como uma métrica nas atividades de projeto e de análise de sistema, considerando que, para cada tipo de classe (negócio, controle, interface, dados – segundo a UML), a funcionalidade precisa ser avaliada por um perfil de profissional distinto, no intuito de ser possível julgar a distribuição da colaboração.

Tabela 5 – Padrão de Software para Reutilização 5 – Quantidade de Classes Principais/Índice de Especialização da Subclasse.

Problema	Como dimensionar a qualidade em reutilização das classes principais?
Solução	Realizar uma análise profunda do domínio do negócio. A UML sugere, neste caso, a utilização dos Diagramas de Use Cases, que se constituem em uma ótima ferramenta para compreender a funcionalidade do negócio. Verificar se todas as áreas afins do sistema foram identificadas. Algumas classes poderão ter uma definição abstrata, para serem mais genéricas. Manter um número mínimo de métodos sobrescritos e/ou adicionados da superclasse e raras exclusões de métodos. À medida em que aumenta a subclassificação, pode aumentar a especialização da classe. Por isto, a posição da classe na hierarquia de herança deve ser avaliada.
Usos Conhecidos	A aplicação deste padrão de solução nas fases de projeto e de análise e, principalmente, como instrumento da revisão crítica do projeto pelo gerente do projeto, pelo analista de processos de negócios e pelo responsável pela garantia da qualidade, é de suma importância para avaliar a solução dada, em termos de modelo, para o entendimento do negócio. As considerações explicitadas neste padrão de solução podem, inclusive, ser utilizadas no processo como subsídio para uma espécie de *checklist* de criação de classes principais de sistema e seus inter-relacionamentos com as demais.

Tabela 6 – Padrão de Software para Reutilização 6 – Organização do no Repositório de Elementos Reutilizáveis.

Problema	Como organizar o Repositório de Elementos Reutilizáveis?
Solução	Utilizar elementos estáticos e/ou dinâmicos da UML, como diagramas de estado, diagramas de classes e diagramas de colaboração para descrever os exemplos de utilização dos componentes.

Usos Conhecidos	Apresenta extrema valia para os ambientes organizacionais voltados ao desenvolvimento OO, de forma a oferecer visibilidade para a organização dos elementos reutilizáveis. É sabido que, em alguns casos, devido a uma estrutura deficitária de alocação dos elementos no repositório e considerando as dificuldades culturais pertinentes à atividade de documentação, muitos elementos não são reutilizados em função de se ter dúvidas quanto a sua valia para o contexto da solução. Assim, desencadeiam-se alguns problemas que deturpam os objetivos de OO: a existência de mais de um componente para uma função similar, a alocação indevida das responsabilidades dos sistemas pelas classes. Frente a este contexto, verifica-se que os próprios modelos OO são adequados para a documentação do repositório, uma vez que é uma linguagem conhecida na organização e que, por motivar a documentação dos objetos com a visão do sistema, contribui para que as dificuldades associadas ao uso do repositório sejam minimizadas.

Tabela 7 – Padrão de Software para Reutilização 7 – Templates de Reutilização de Use Cases.

Problema	Como representar diferentes fluxos de execução do sistema através de Use Cases?
Solução	Utilizar os *templates* de Use Cases, que fornecem informações mais detalhadas sobre cada Use Case. A seguir é apresentada uma estrutura para *template* de Use Case [MAT 98]. • Nome do Use Case: deve ser informado um nome significativo. • Descrição: descrição sucinta do objetivo do Use Case. • Tipo do Use Case: indica se o Use Case é concreto ou abstrato. Use Cases concretos são aqueles que por si só constituem um curso completo de eventos iniciados por um ator. Já Use Cases abstratos são aqueles que constituem parte de um Use Case e que não possuem sentido quando isolados. • Atores/Papéis: lista de atores envolvidos no Use Case e os papéis que representam. • Pré-condições: indicam circunstâncias que devem ser atendidas antes do Use Case ocorrer. Se as pré-condições possuem uma ordem seqüencial, elas devem ser explicitadas. Caso esta pré-condição não seja satisfeita, o estado final do Use Case é considerado indefinido. • Curso Básico: cada Use Case deve possuir um único curso básico de execução. Este curso básico deve representar a seqüência de passos que devem ocorrer para que o objetivo do Use Case seja atingido. • Curso Alternativo: o Use Case pode ou não possuir cursos alternativos. O curso alternativo é uma seqüência diferente de passos que pode ser executada e que também permite atingir o objetivo do Use Case. • Curso de Exceção: o Use Case pode ou não possuir cursos de exceção, que se constitui em uma seqüência de passos que é realizada quando uma tarefa é interrompida. Este curso de exceção indica tanto a causa da interrupção, bem como as atividades a serem seguidas para superá-la. Cursos de exceção devem possuir, pelo menos, uma pós-condição para a etapa que valida o curso de exceção a ser seguido. • Pós-condição: as pós-condições validam o curso básico e todos os cursos

	alternativos no sentido de assegurar que os mesmos conduzam ao estado final esperado para o Use Case. Estas pós-condições não serão necessariamente verdadeiras em caso de utilização de um curso de exceção.
Usos Conhecidos	Muito adequado para as fases de análise do negócio e de análise de sistema. Os templates oferecem a facilidade de estabelecer uma linguagem padrão para os use cases, fato que contribui para o melhor entendimento do usuário sobre os elementos do sistema e para a melhoria da comunicação na equipe de projeto. Esta proposição de estrutura para os templates pretende ser uma descrição completa da funcionalidade, porém de maneira simples, favorecendo o entendimento. Neste sentido, é atribuição da organização, ao definir o padrão de linguagem para a descrição do use case, prezar pela não utilização de uma linguagem essencialmente técnica, mesmo porque este modelo se destina à documentação da funcionalidade do sistema.

6. Considerações Finais

Para cada um dos padrões de software para reutilização apresentado, determinadas características foram consideradas de forma a auxiliar na resolução de um problema de ocorrência repetida. A identificação e a aplicação de padrões de software no ciclo de desenvolvimento de sistemas OO indicou a conquista de um elevado nível de maturidade do processo, através do acúmulo de conhecimento e de experiência.

Os padrões de software para reutilização agregam conhecimento a respeito de problemas similares. Dessa forma, entre as várias soluções consideradas para estes problemas, foram identificados os aspectos estáveis que, por sua vez, viriam a se tornar a base da descrição da solução genérica. Os padrões de software para reutilização foram baseados em observações de modelos OO e conhecimento teórico relacionado. Na pesquisa realizada não foi encontrada nenhuma categoria de padrões de software similar à proposta neste trabalho. Como extensão futura, os padrões de software para reutilização poderiam ser reavaliados, após serem utilizados de forma prática, no âmbito de desenvolvimento de sistemas OO.

Toda a arquitetura do projeto do sistema deve voltar-se para a reutilização. Uma arquitetura é tão melhor quanto menos transformadora ela for. Isto é, quanto mais próxima ela for da realidade do negócio e dos princípios OO utilizados, menos alterações nas soluções de implementação serão necessárias. Dessa forma, o código resultante será um produto fiel do modelo projetado, refletindo características essenciais na qualidade do produto e do processo de desenvolvimento de software.

7. Referências Bibliográficas

[1] BOOCH, G. "**Quality Software and the Unified Modeling Language"**, abril 1998. *http://www.rational.com/support/techpapers/ softuml.html* .

[2] COPLIEN, J.O. "A Generative Development-Process Pattern Language". In: Coplien, J. O. e Schmidt, D. C. **"Pattern Languages of Program Design"**. 1a. ed. United States of America: Addison-Wesley, 1995.

[3] CUNHA, G. E. **Proposta de *Reuse Patterns* para a Reutilização Sistemática em Sistemas de Software Orientado a Objetos**. Trabalho de Conclusão de Curso. Centro de Ciências Exatas e Tecnológicas/Universidade do Vale do Rio dos Sinos-UNISINOS. 1999.

[4] GAMMA, E. "**Design Patterns – Elements of Reusable Object-Oriented Software.**" Massachusetts: Addison-Wesley Publishing Company, 1994.

[5] McGREGOR, John. **"A Component Testing Process",** maio 1999. *http://cyclone.cs.clemson.edu/~johnmc/new/pact/node8.html*

[6] PREE, Wolfgang. "**Design Patterns for Object-Oriented Software Development**". England: Addison-Wesley Publishing Company. 1995.

[7] RATIONAL Software Corporation. "**UML Semantics – version 1.1"**. 1997. Disponível em *http://www.rational.com/uml.*

[8] SALVIANO, C. F. **Introdução a Software Patterns.** Fundação Centro Tecnológico para Informática - CTI. Campinas. São Paulo, 1997.

[9] SETZER, Valdemar W. **Bancos de Dados – Conceitos, Modelos, Gerenciadores, Projeto Físico, Projeto Lógico**. São Paulo: Editora Edgard Blüder, 1989. 289p.