

# SugarLoafPloP'2008

7<sup>th</sup> Latin American Conference on Pattern Languages of Programming  
Fortaleza, Brazil, August 24-27, 2008

## Preliminary Proceedings

**Rossana M. C. Andrade  
Neil B. Harrison (eds.)**

---

## Supporting Agencies



---

## Sponsor



---

## Organization



# **SugarLoafPLoP'2008 Organization Committee**

## ***Conference Chair***

Jerffeson Teixeira de Souza (UECE, Brazil)

## ***Program Committee Co-chairs***

Neil B. Harrison (Utah Valley University, USA)  
Rossana M. C. Andrade (DC/UFC, Brazil)

## ***Technical Program Committee***

Fabio Kon (IME/USP, Brazil)  
Jerffeson Teixeira de Souza (UECE, Brazil)  
Joseph Yoder (U. Illinois / The Refactory Inc., USA)  
Linda Rising (Independent Consultant, USA)  
Lisa Hvatum (The Refactory Inc., USA)  
Neil Harrison (Utah Valley University, USA)  
Paulo Cesar Masiero (ICMC/USP, Brazil)  
Rosana Braga (ICMC/USP, Brazil)  
Rossana M. C. Andrade (DC/UFC, Brazil)  
Sergio Soares (DSC/UPE, Brazil)  
Tiago Massoni (DSC/UPE, Brazil)

## ***Local Organization***

Fabrício Gomes de Freitas  
Rafael Augusto Ferreira do Carmo  
Shirliane Matos de Souza  
Tarciane de Castro Andrade

### *Reviewers*

- Carlo Giovano (Instituto Atlântico, Brazil)  
Cidcley de Souza (CEFET-CE, Brazil)  
Fabiana Gomes Marinho (MDCC/UFC, Brazil)  
Francisco (Vieira UFPI, Brazil)  
Francisco Heron de Carvalho Júnior (DC/UFC, Brazil)  
Fátima (PPGETI/UFC, Brazil)  
James Coplien (Gertrud & And Cope, Denmark)  
Lincoln Rocha (MDCC/UFC. Brazil)  
Misael Santos (SERPRO, Brazil)  
Paula Cibele (COPPE/UFRJ, Brazil)  
Paulo Henrique Mendes (Imperial College, England)  
Pedro Neto (UFPI, Brazil)  
Pedro Porfirio Muniz Farias (MIA/UNIFOR. Brazil)  
Rute Castro (GREAT/DC/UFC, Brazil)  
Tarciane de Castro Andrade (UECE, Brazil)  
Valéria Lelli (MDCC/UFC, Brazil)  
Windson Viana de Carvalho (IMAG, France)

# Table of Contents

## *I – Writers’ Workshop*

---

<b>Padrões de Projeto para Frameworks e Componentes Baseados em Metadados</b> Eduardo Guerra, Fernando Pavão, Clovis Fernandes (Instituto Tecnológico de Aeronáutica)	<b>3</b>
 <b>Linguagem de Padrões para Avaliação de Conhecimento em Objetos de Aprendizagem Parte II</b> Ingrid Monteiro, Clayson Sandro Celes, Cidcley de Souza (Centro Federal de Educação Tecnológica do Ceará)	<b>25</b>
 <b>Evento Interno Anônimo</b> Renato Costalima, Gustavo Campos, Jerffeson Souza (Universidade Estadual do Ceará)	<b>59</b>
 <b>Adaptação Dinâmica Guiada por Contrato</b> Jonivan Lisboa, Orlando Loques (Universidade Federal Fluminense)	<b>65</b>
 <b>Uma Linguagem de Padrões para Estimativa de Projeto de Software nas Micro e Pequenas Empresas</b> Tarciane C. Andrade, Jerffeson Souza (Universidade Estadual do Ceará)	<b>72</b>

---

## ***II – Pattern Applications***

---

**Método para Desenvolvimento Utilizando Padrões de Software** 105

Alessandra Chan, Rosana Braga  
(Universidade de São Paulo)

---

**Uso de Padrões em Linhas de Produtos de Software:**

**Uma Revisão Sistemática** 123

Luiz Alberto Gomes (Pontifícia Universidade Católica de Minas Gerais)  
Rosana Braga (Universidade de São Paulo)

---

**Experiência de Aplicação de Padrões de Software para o**

**Cálculo de ICMS** 138

Francisco Aquino, Jerffeson Souza  
(Universidade Estadual do Ceará)

---

**Adapting the Strategy Pattern for Micro Applications** 151

Davi Sales Pinheiro, Lincoln Rocha, Rossana M. C. Andrade  
(Universidade Federal do Ceará)

---

**A proposal for Discovering Relationships among Software**

**Patterns using Latent Semantic Analysis** 160

Rute Castro (Universidade Federal do Ceará)  
Jerffeson Souza (Universidade Estadual do Ceará)  
Rossana M. C. Andrade (Universidade Federal do Ceará)

---

## ***III – Writing Patterns***

---

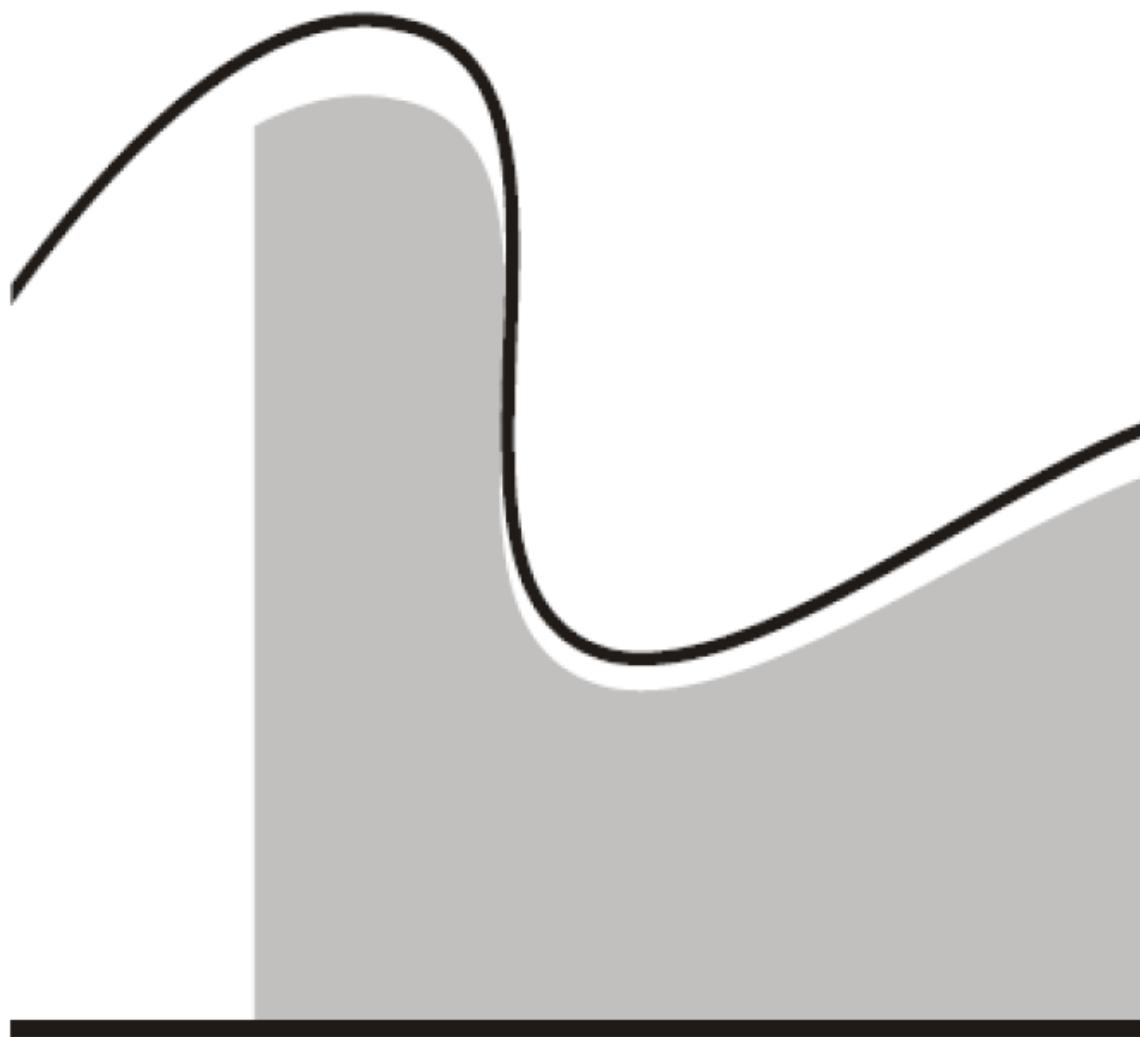
**Conflict Manager** 177

Valéria Leitão, Rute Castro, Rossana M. C. Andrade  
(Universidade Federal do Ceará)

---



# SugarLoafPLoP'2008



Writers' Workshop



# Padrões de Projeto para Frameworks e Componentes Baseados em Metadados

Eduardo M. Guerra<sup>1</sup>, Fernando Pavão<sup>1</sup>, Clóvis T. Fernandes<sup>1</sup>

<sup>1</sup> Instituto Tecnológico de Aeronáutica (ITA)  
São José dos Campos – SP – Brasil

guerra@ita.br, fernaaando@gmail.com, clovistf@uol.com.br

**Abstract.** *Metadata-based components are those that process their logic based on the metadata of the class whose instance they are working with. Many recent frameworks use this approach to get a higher reuse degree and to adapt it to the application needs. But there is not a study that addresses the design of components that works with metadata. This paper presents the results of an ongoing study that is analyzing existing frameworks that use this approach and identifying design patterns for common problems of this kind of component.*

**Resumo.** *Componentes baseados em metadados são aqueles que adaptam sua lógica de acordo com os metadados das classes cujas instâncias ele está trabalhando. Alguns frameworks mais recentes utilizam essa abordagem para conseguir um maior reuso e se adaptar às necessidades da aplicação. Porém, ainda não existe estudo relativo a modelagem de componentes que trabalham com metadados. Este artigo apresenta o resultado de um estudo em andamento que visa analisar os frameworks existentes que utilizam essa abordagem e identificar padrões de projeto para problemas comuns a esse tipo de componente.*

## 1. Introdução

Existem várias formas de tornar um componente e um framework mais flexível. Uma das formas mais comuns de se fazer isso em um framework orientado a objetos, é a especialização de suas classes e implementação de suas interfaces (Pree e tal, 1995) (Fayad et al, 1999). Uma outra abordagem para a flexibilidade é o uso de inversão de controle, onde os componentes internos a uma classe são definidos externamente, permitindo que, com o uso do polimorfismo, diversos comportamentos possam ser configurados (Fowler, 2004).

Muitos frameworks mais recentes utilizam uma abordagem diferente, para que possam se adaptar às necessidades da aplicação. Eles utilizam metadados configurados para a classe da aplicação como base para a execução de sua lógica. Este artigo classifica como componente baseado em metadados todo componente ou framework que modifica o seu comportamento de acordo com os metadados da classe cuja instância está trabalhando. O uso desse tipo de abordagem se popularizou na plataforma Java depois da incorporação das anotações a linguagem (JSR 175, 2003), porém existem outras formas de armazenamento dos metadados.

Vários frameworks já maduros, como o Hibernate (Bauer and King, 2004) e o JUnit (2007) são baseados nesse tipo de abordagem. Apesar de tudo, esses conceitos nunca foram abstraídos de forma a mostrarem características comuns a componentes que precisem trabalhar com metadados. Até então, não existe documentação referente a técnica de desenvolvimento, padrões de projeto ou mesmo boas práticas para o uso de metadados dentro de uma modelagem orientada a objetos.

O objetivo deste artigo é retratar um trabalho em andamento que visa identificar padrões de projeto para a modelagem de componentes e frameworks baseados em metadados. O artigo irá abordar tanto padrões para a estruturação interna desses componentes, quanto relativos a aplicabilidade da abordagem baseada em metadados. Os padrões catalogados neste artigo surgiram da análise de frameworks que utilizam essa abordagem.

A seção 2 apresenta as principais formas de armazenamento de metadados. A seção 3 fala sobre a importância da documentação desses padrões de projeto e apresenta os principais frameworks utilizados como base para a pesquisa. A seção 4 apresenta o critério adotado para classificação dos padrões e o formato utilizado para apresentação dos mesmos, que em seguida serão detalhados nas seções 5 e 6. A seção 7 conclui o trabalho ressaltando as principais contribuições e o que ainda precisa ser feito.

## 2. Formas de Armazenamento de Metadados

Os metadados podem ser armazenados de diversas formas, sendo que o mais importante é que precisam estar acessíveis ao componente no momento da execução. Podem existir situações em que podem ser suportadas mais de uma forma de armazenamento, ou mesmo, combinar diferentes formas para obter todos os metadados desejados. Um exemplo dessa abordagem é a especificação EJB 3.0 (JSR 220, 2003), que utiliza padrões de nomenclatura, anotações e ainda permite sobrepor essas informações em tempo de implantação a partir de um descriptor XML.

As principais formas de armazenamento de metadados utilizadas hoje são as seguintes:

- Metadados Implícitos: O uso de padrões de nomes para classes, métodos e atributos que significam algo para o componente, é uma forma simples de adicionar metadados. Um exemplo bastante utilizado é o uso de métodos iniciados em “get” para a recuperação de propriedades e métodos iniciados em “set” para a atribuição de dados em propriedades. Também conhecido na comunidade de desenvolvimento como uso de convenções de nomenclatura (Dov, 2006).
- Definição Programática de Metadados: Os metadados de uma classe são passados diretamente ao componente de forma programática. Este método é indicado somente quando os metadados forem simples, caso contrário o código de criação dos metadados pode-se tornar tedioso e sujeito a erros.
- Metadados em Arquivos Externos: O componente lê os metadados presentes em arquivos externos para poder configurar suas regras de negócio, permitindo que os metadados sejam alterados sem que o código da aplicação seja recompilado. Nesse tipo de abordagem é muito comum o uso de arquivos XML (XML 2008).

- Anotações: As anotações são metadados que se pode agregar diretamente nas classes e em seus membros. A linguagem Java passou a dar suporte a anotações de forma nativa a partir de sua versão 5 (JSR 175, 2003). Anteriormente, elas podiam ser utilizadas com o uso da ferramenta XDoclet (2005), que utilizava essas anotações para gerar descritores XML de diversos frameworks.
- Banco de Dados: Os bancos de dados também são uma alternativa para o armazenamento dos metadados. Normalmente, o armazenamento dos metadados em uma base de dados relacional vem da necessidade deles serem alterados com maior freqüência e isso poder ser feito de forma transacional.

A programação orientada a atributos (Rouvoy et al, 2006) (Schwarz, 2004) foca na inserção de anotações nas classes como estilo de programação. Esse estilo não se preocupa em como e para que essas informações serão consumidas: para a geração de código, para realizar alterações nas classes no momento de seu carregamento ou para acesso em tempo de execução. O estudo de componentes baseados em metadados aborda apenas o consumo dessas informações em tempo de execução, porém também não se limita ao uso de anotações, podendo os metadados estarem armazenados de outras formas.

### **3. Padrões de Projeto para Componentes Baseados em Metadados**

Em trabalhos sobre padrões de projeto (Gamma et al, 1994) e modelagem de aplicações e frameworks, não existem referências sobre a modelagem de componentes que utilizam metadados como base de seu processamento. Esse tipo de componente possui peculiaridades que dizem respeito a forma com que os metadados são empregados e a estrutura interna desse tipo de componente.

Diversos componentes baseados em metadados disponíveis atualmente utilizam abordagens que podem ser observadas e abstraídas. A estratégia de modelagem utilizada nesses componentes refletem a experiência dos profissionais que os desenvolveram e já enfrentaram problemas relacionados com a utilização de metadados. Ao abstrair essas soluções em padrões de projetos, esse estudo visa facilitar a construção de novos componentes utilizando soluções de modelagem já comprovadas na prática.

A seguir estão listados os frameworks que foram analizados para a identificação dos padrões:

- Hibernate (Bauer and King 2004) e JPA (JSR 220, 2003): O Hibernate (2008) é um framework que faz um mapeamento das classes persistentes da aplicação para as tabelas do banco de dados. Ele utiliza metadados em XML para definir o mapeamento das classes. A JPA (*Java Persistence API*) é a API que padronizou o mapeamento objeto-relacional na plataforma Java Enterprise Edition, foi inspirada nos conceitos do Hibernate (2008), porem é independente do

fornecedor do framework. Seu grande diferencial foi permitir que os metadados sejam definidos como anotações nas classes e sobrepostos por arquivos XML.

- Hibernate Validator (2007): API utilizada para validação multi-camadas da integridade de dados, de forma que as restrições são expressas apenas nas classes que modelam o domínio de dados. Através de anotações o componente é capaz de identificar as regras de validação e buscar por inconsistências nas instâncias passadas a ele.
- SwingBean (2007): Framework que fornece componentes pouco granulares para a criação de formulários e tabelas. A configuração desses componentes gráficos ocorre a partir de metadados configurados em arquivos XML. A partir dos metadados, as propriedades são mapeadas para campos de um formulário ou colunas de uma tabela, o que permite que o desenho de tela, a validação dos dados, a inserção e a recuperação do objeto do formulário sejam feitos pelo componente.
- JUnit (Masson and Husted, 2004): É um framework que auxilia a criação e execução de testes de unidade sobre classes Java. Nas versões anteriores a 4, o JUnit (2007) utilizava padrões de nomenclatura, ou seja, metadados implícitos, para dar um significado aos métodos de uma classe de teste. As novas versões utilizam anotações para marcar os métodos e dar um significado a eles dentro do ciclo de vida de uma classe de teste.
- EJB (JSR 220, 2003): Os EJBs (Enterprise Java Beans) são objetos de negócio que executam dentro de um container e seu principal objetivo é deixar que o desenvolvedor se preocupe com as regras de negócio, deixando a implementação de questões não-funcionais como transações e segurança para o fabricante do container. Nesse contexto, a aplicação indica, a partir de metadados dos EJBs, qual deve ser o comportamento do container em relação a essas características. As primeiras versões os EJBs utilizavam extensos descritores em XML para a definição desses metadados. Pelo fato desses arquivos terem se mostrado improdutivos e difíceis de manter (Dov, 2006), a versão 3.0 (Sriganesh et al, 2006) da especificação define os metadados utilizando anotações e com diversas configurações feitas por padrões de nomenclatura, mas que ainda podem ser sobrepostas durante a implantação nos descritores XML.
- JAXB (JSR 222, 2006): O JAXB (Java API for XML Binding) utiliza anotações para fazer o mapeamento entre uma estrutura de classes e elementos de um documento XML.
- Gênesis (2008): O Gênesis é um framework que utiliza metadados para ligar componentes de uma interface gráfica em uma classe de controle. Ele possui metadados para o mapeamento de eventos de componentes de interface para métodos e para a ligação de atributos com fontes de dados em componentes como tabelas e listas. Além disso, ainda existem metadados para a realização da sincronização entre campos da classe de controle e valores de componentes de formulário.

## 4. Classificação e Contexto dos Padrões

De acordo com a sua natureza, os padrões de projeto apresentados neste artigo são classificados em padrões estruturais ou padrões de aplicabilidade, conforme descrição abaixo:

- Padrões Estruturais: referem-se a forma como as classes, objetos e metadados são compostos para formar a estrutura de um componente baseado em metadados. Apresentam soluções para manipulação, leitura e processamento da lógica com os metadados. Esses padrões focam na estrutura interna do componente.
- Padrões de Aplicabilidade: tentam generalizar famílias de problemas onde o uso de um componente baseado em metadados é adequado. Seu objetivo é identificar e abstrair situações onde o uso de metadados é aplicável e apresentar a estrutura de soluções para cada situação. Esses padrões focam na visão externa do uso desse tipo de componente.

Os padrões documentados neste artigo estão inseridos no contexto de uma aplicação que deseja aumentar o reúso de seus componentes e diminuir a quantidade de código repetitivo criado manualmente, utilizando como estratégia para isso a construção de componentes baseados em metadados.

## 5. Padrões Estruturais

A seção 5.1 apresenta o padrão Fábrica de Metadados, utilizado para o desacoplamento entre a leitura dos metadados e o processamento da lógica. A seção 5.2 apresenta o padrão Metadados Extensíveis, utilizado para permitir a extensão dos metadados de um componente.

## 5.1. Fábrica de Metadados

### Intenção:

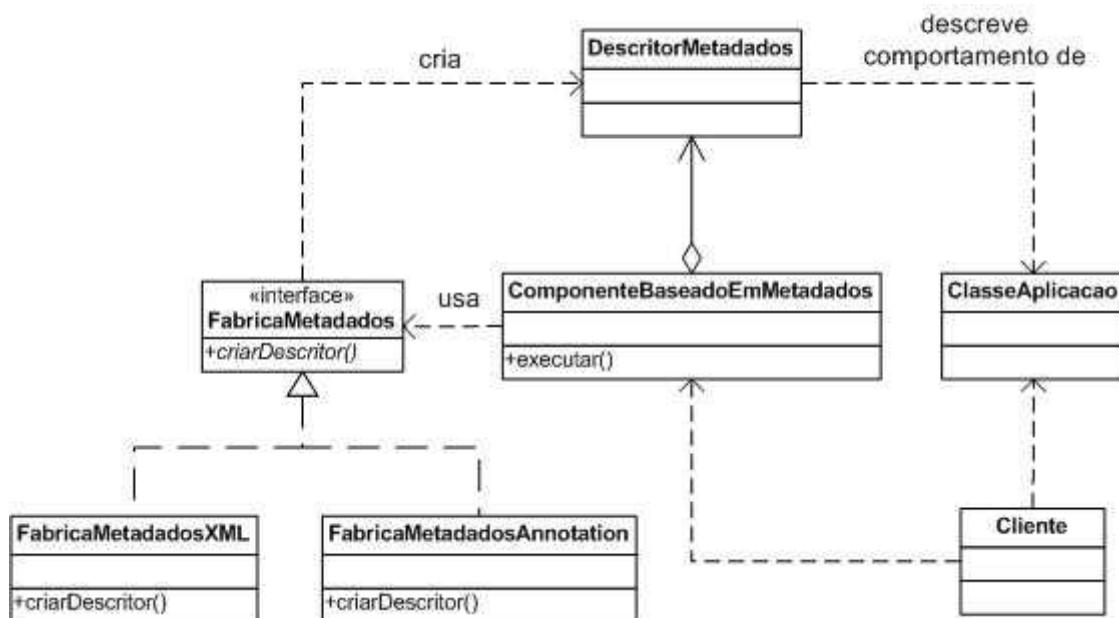
Desacoplar o mecanismo de leitura dos metadados da execução da lógica do componente, permitindo a leitura dos metadados de várias fontes diferentes.

### Forças:

- Se a leitura dos metadados se misturar com a lógica do componente, o mesmo ficará acoplado a uma determinada forma de leitura.
- Diferentes formas de armazenamento de metadados devem poder coexistir
- Uma forma de armazenamento pode ser utilizada de forma combinada com a outra.

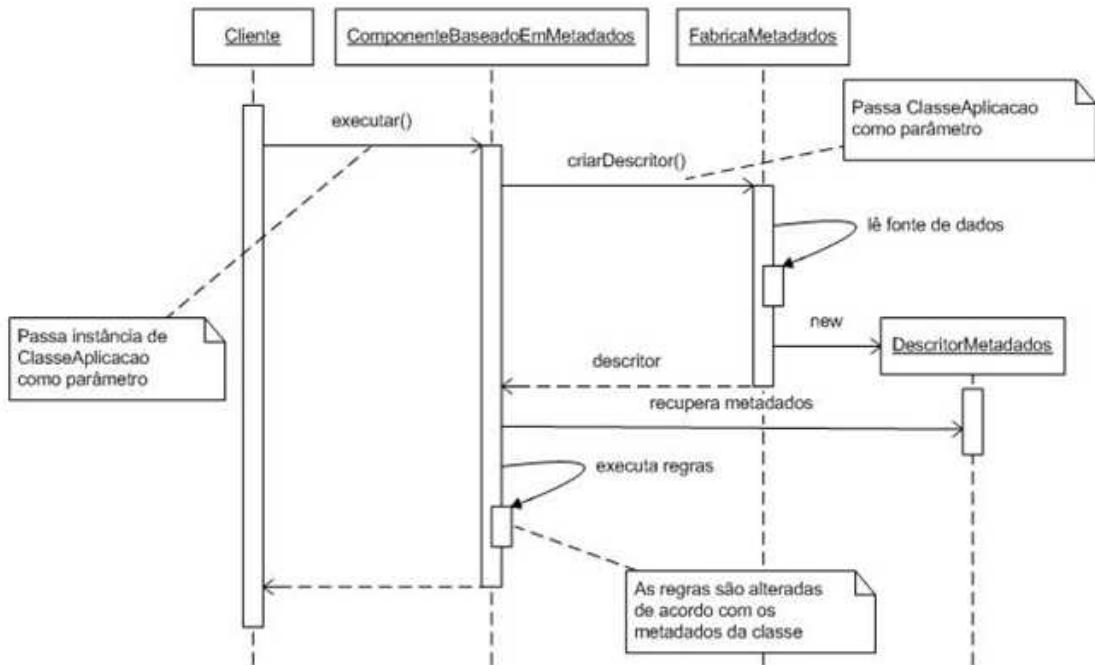
### Solução:

O padrão Fábrica de Metadados encapsula a recuperação dos metadados e fornece uma estrutura reutilizável para seu uso nos componentes. A Figura 1 apresenta um diagrama de classes com estrutura do padrão Fábrica de Metadados.



**Figura 1. Estrutura do padrão Fábrica de Metadados**

A Figura 2 apresenta um diagrama de sequência que mostra a interação entre as classes no padrão Fábrica de Metadados.



**Figura 2. Diagrama de sequência de execução do padrão Fábrica de Metadados**

Como pode ser observado na Figura 2, o Cliente faz uma chamada ao componente passando como parâmetro a instância de uma classe. Essa classe é utilizada pelo componente como parâmetro para recuperar o DescritorMetadados a partir de uma implementação de FabricaMetadados. A partir dos metadados, o componente sabe como deve ser a sua lógica de processamento para aquela instância. Essa estrutura pode ser utilizada como base para qualquer componente baseado em metadados.

Apesar de no diagrama estar sendo representada uma fábrica genérica, uma fábrica concreta deve ser utilizada. Através de alguma configuração, por exemplo, o componente deve saber que fábrica deve ser utilizada. Uma outra opção seria deixar para o cliente a utilização da fábrica desejada para a criação do descritor, o qual seria passado como parâmetro para o componente.

## Participantes

- **FabricaMetadados:** É uma interface que abstrai a leitura dos metadados armazenados. Possui um método que recebe uma representação de uma classe como parâmetro e retorna uma instância de `DescriptorMetadados` com os metadados da classe recebida.
- **FabricaMetadadosXML** e **FabricaMetadadosAnnotation**: Representam classes concretas que implementam a interface `FabricaMetadados` e possuem a lógica de leitura de alguma forma de armazenamento dos metadados. Encapsulam a produção de um `DescriptorMetadados` a partir de uma forma de armazenamento específica.

- DescritorMetadados: É uma classe que representa os metadados de determinada classe. É a partir dos atributos de DescritorMetadados que o componente irá configurar seu algoritmo para cada classe. De acordo com a complexidade dos metadados, pode ser uma estrutura complexa formada por várias classes.
- ClasseAplicacao: Representa uma classe da aplicação que será descrita com metadados, cujas instâncias serão passadas como parâmetro para o ComponenteBaseadoEmMetadados.
- Cliente: Representa a aplicação que utiliza o componente e gerencia as instâncias de ClasseAplicacao.
- ComponenteBaseadoEmMetadados: Representa o componente cuja lógica é adaptada nos metadados, representados por uma instância de DescritorMetadados, da classe da instância recebida como parâmetro. Utiliza alguma implementação de FabricaMetadados para obter os metadados.

### **Consequências:**

- Facilita o uso de diferentes formas de armazenamento de metadados. A interface FabricaMetadados pode ser implementada para diferentes origens de metadados facilitando a substituição da forma de armazenamento dos mesmos.
- Permite a reutilização de algoritmos. O desacoplamento do cliente em relação a implementação através do uso do ComponenteBaseadoEmMetadados permite o reuso de algoritmos que serão adaptados em tempo de execução de acordo com os metadados carregados para o objeto.
- Melhoria no desempenho da aplicação. O padrão Fábrica de Metadados faz com que os metadados sejam lidos antes do processamento da lógica de negócios. Isso permite que a leitura dos metadados possa ser feita anteriormente e mantida em memória, diminuindo o tempo de processamento da lógica.

### **Usos Conhecidos:**

O padrão Fábrica de Metadados pode ser observado no Hibernate (2008) e demais implementações da JPA (JSR 220, 2003), para possibilitar que os metadados referentes ao mapeamento objeto-relacional possam ser armazenados utilizando-se anotações (JSR 175, 2003) ou descritores XML.

O framework SwingBean (2007), apesar de não haver mais de uma forma de armazenar os metadados, utiliza o padrão. Isto faz com que a leitura do XML não prejudique no desempenho do componente. Planeja-se para a próxima versão do framework que os metadados também possam ser configurados utilizando anotações.

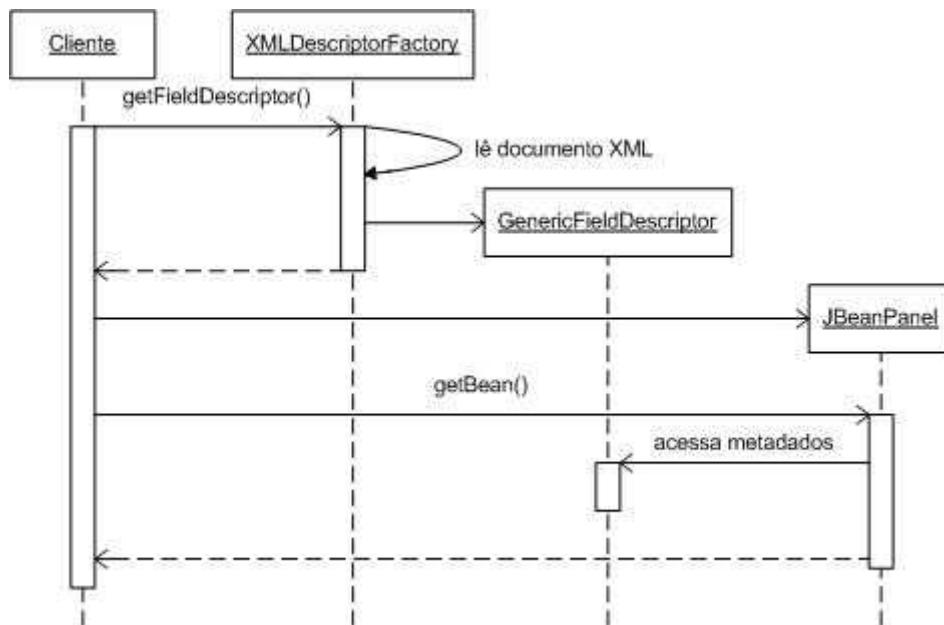
### **Exemplo:**

O SwingBean (2007) faz uso da variação do padrão na qual o próprio cliente chama a fábrica para a criação do descritor, o qual é passado como parâmetro no construtor do

componente, no caso as classes JBeanPanel, que representa um formulário, ou BeanTableModel, que representa o modelo de dados de uma tabela.

Essa variação é justificada pelo fato de ser possível criar formulários e tabelas diferentes para a mesma classe. Com isso, o acesso do cliente direto a fábrica torna possível que o mesmo passe o caminho do documento XML que contém os metadados desejados. A Figura 3, apresenta um diagrama de sequência com a interação dos componentes do SwingBean. A Figura 4 apresenta um trecho de código, retirado do tutorial do site do projeto, com a criação de um JBeanPanel.

Para que o SwingBean possa no futuro utilizar anotações para a definição dos metadados, bastaria a criação de uma nova fábrica. Dessa forma, o cliente utilizaria essa nova fábrica para a criação da instância de GenericFieldDescriptor.



**Figura 3. Diagrama de sequência com a aplicação do padrão Fábrica de Metadados no SwingBean**

```

GenericFieldDescriptor descriptor = XMLDescriptorFactory
    .getDescriptor(
        Cachorro.class,
        "org\\swingBean\\example\\simpleform\\cachorroForm.xml",
        "CachorroForm");
final JBeanPanel<Cachorro> panel = new JBeanPanel<Cachorro>(
    Cachorro.class, descriptor);

```

**Figura 4. Criação do descritor e do componente no SwingBean**

## Padrões Relacionados

O padrão Fábrica de Metadados utiliza o padrão Factory (Gamma et al, 1994) para permitir o uso de diferentes formas de armazenamento de metadados.

## 5.2. Metadados Extensíveis

### Intenção:

Permitir que o modelo de metadados utilizado pelo componente possa ser estendido.

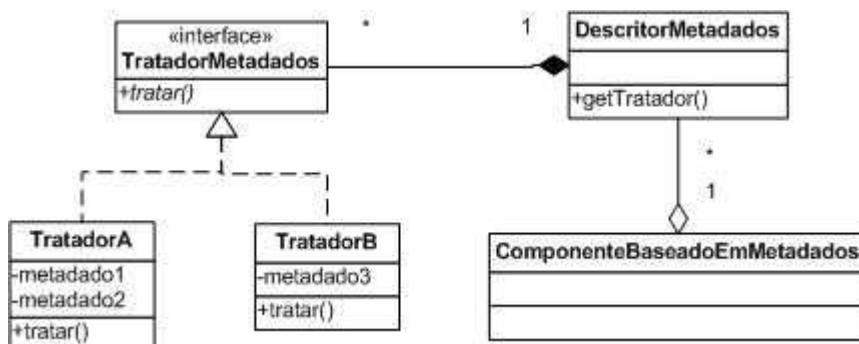
### Forças:

- Os metadados podem estar armazenados de diversas formas.
- As aplicações podem necessitar de metadados específicos.
- Novos metadados podem acrescentar novos comportamentos na execução do componente.

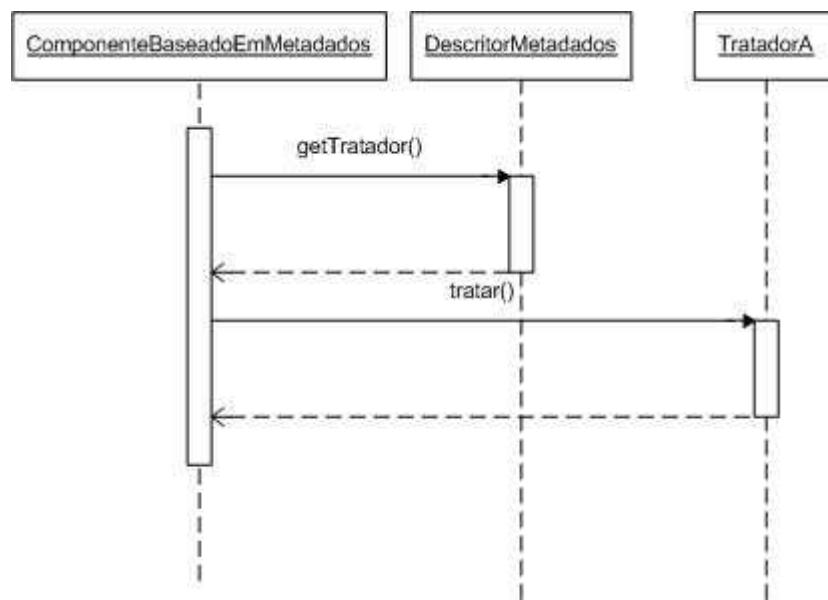
### Solução:

O padrão Metadados Extensíveis, cuja estrutura pode ser observada no diagrama de classes da Figura 5, abstrai o tratamento de cada metadado através da interface TratadorMetadados. Dessa forma, a aplicação pode implementar essa interface e associar novos metadados à classe criada. No momento da criação do DescritorMetadados, a fábrica de metadados deve detectar essa associação e criar a classe de tratamento correta, inserindo os metadados corretamente em sua instância.

No momento de execução da lógica, o componente recupera no DescritorMetadados o tratador adequado e delega para ele parte da execução. Na Figura 6 está representado um diagrama de sequência que mostra a lógica executada para cada tratador.



**Figura 5. Estrutura do padrão Metadados Extensíveis**



**Figura 6. Seqüência de execução da lógica do componente para cada tratador do padrão Metadados Extensíveis**

#### Participantes:

- **DescriptorMetadados:** É composto por diversas instâncias de implementações de **TratadorMetadados**, devidamente populadas com os metadados da classe.
- **TratadorMetadados:** Interface que contém os métodos de tratamento previstos para um determinado tipo de metadado que serão utilizados pelo **ComponenteBaseadoEmMetadados**. Esses métodos representam a parte da lógica que será delegada aos tratadores.
- **TratadorA, TratadorB:** Classes que contém o algoritmo para o tratamento de determinado tipo de metadado.
- **ComponenteBaseadoEmMetadados:** Componente que terá seu comportamento configurado através dos metadados e irá delegar parte dessa lógica aos tratadores.

#### Consequências:

- Desacopla parte da lógica de execução do componente, encasulando-a dentro de tratadores.
- A fábrica de metadados deverá possuir algum mecanismo para associar os tratadores aos tipos de metadados, seja por configuração ou programaticamente.
- Facilita a adição de novos tratadores, sem influenciar na lógica central do componente.

- Permite que a aplicação crie novos tipos de metadados, associando-os a novos tratadores. Isso permite que metadados mais específicos de uma aplicação possam ser incluídos de forma fácil.

### Usos Conhecidos:

O padrão Metadados Extensíveis pode ser observado no framework Hibernate Validator (2008) que associa metadados às classes com o objetivo de fazer a validação de instâncias. Novas anotações podem ser criadas e através de uma anotação nela mesmo, definido qual será a classe que fará a validação que utilizará aqueles novos metadados.

O framework SwingBean (2007) também permite que novos atributos sejam adicionados ao seu documento XML. Novos componentes gráficos podem ser criados e configurados programaticamente, sendo que para receber os novos atributos basta que esse componente possua métodos “set” com o mesmo nome.

### Exemplo:

O framework Hibernate Validator utiliza esse padrão para permitir que a aplicação possa adicionar novos validadores aos que já existem. Esse framework utiliza anotações para adicionar as regras de validação nas classes. Novas anotações podem ser adicionadas, sendo que através da anotação @ValidatorClass configura-se a classe que irá tratar aquela anotação. Na Figura 7 está representado o código fonte da anotação @NotNull do próprio framework.

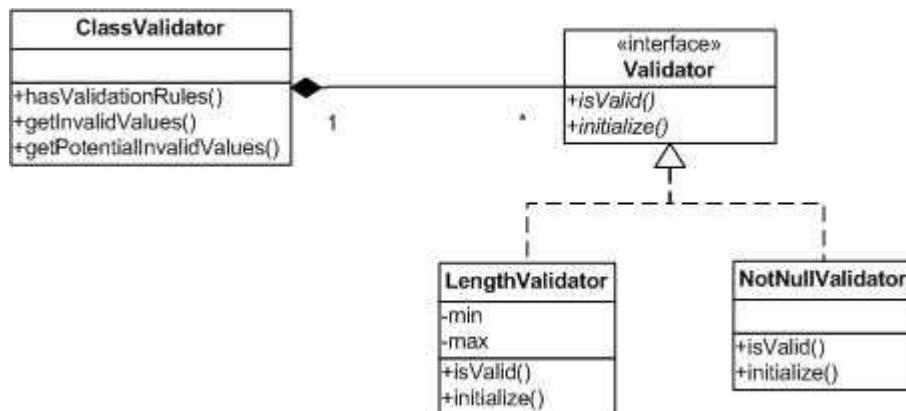
No Hibernate Validator, a classe que representa o componente principal é a classe ClassValidator. Ela recebe a classe que deve ser validada como parâmetro e ela mesma lê as anotações da classe e cria as classes que irão fazer cada validação. O descriptor de metadados nesse caso é representado por uma lista de validadores, conforme representado na Figura 8. Como a classe delega toda a validação aos validadores, que são criados a partir das anotações, é possível a criação de novos tratadores e novas anotações com regras de validação.

```

@Documented
@ValidatorClass(NotNullValidator.class)
@Target({METHOD, FIELD})
@Retention(RUNTIME)
public @interface NotNull {
    String message() default "{validator.notNull}";
}

```

**Figura 7. Definição da anotação @NotNull com a configuração de um tratador**



**Figura 8. Estrutura de Classes do Hibernate Validator**

### Padrões Relacionados:

Para implementação do padrão Metadados Extensíveis é imprescindível que a leitura dos metadados esteja desacoplada da lógica de execução. Para isso é fortemente recomendado sua utilização junto com o padrão Fábrica de Metadados, descrito na seção 5.1 deste artigo.

Esse padrão utiliza uma estrutura baseada no padrão Command (Gamma et al, 1994), onde cada tratador possui uma mesma interface e possui um método para o tratamento dos metadados invocado pelo componente principal. Também existe um relacionamento com o padrão Strategy (Gamma et al, 1994), pois cada tratador pode ser considerado como uma estratégia de execução da lógica do componente.

## 6. Padrões de Aplicabilidade

A seção 6.1 apresenta o padrão Gerenciador de Eventos, utilizado para a marcação de métodos que devem ser chamados de acordo com eventos ou fases de um ciclo de vida. A seção 6.2 irá apresentar o padrão Mapeamento entre Representações, utilizado para o mapeamento entre diferentes paradigmas, como, por exemplo, entre uma estrutura de classes e elementos de um documento XML.

## 6.1. Gerenciador de Eventos

### Intenção:

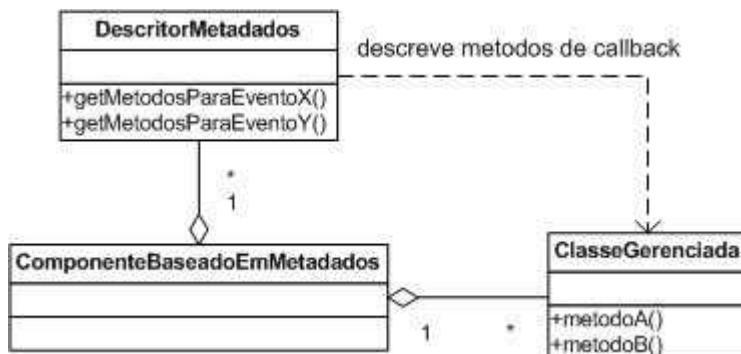
Invocar métodos em classes ou componentes gerenciados de acordo com eventos ocorridos na aplicação.

### Forças:

- As classes cujos métodos serão invocados não devem ser acopladas com as classes do framework.
- Pode haver mais de um método para o tratamento de um mesmo evento.

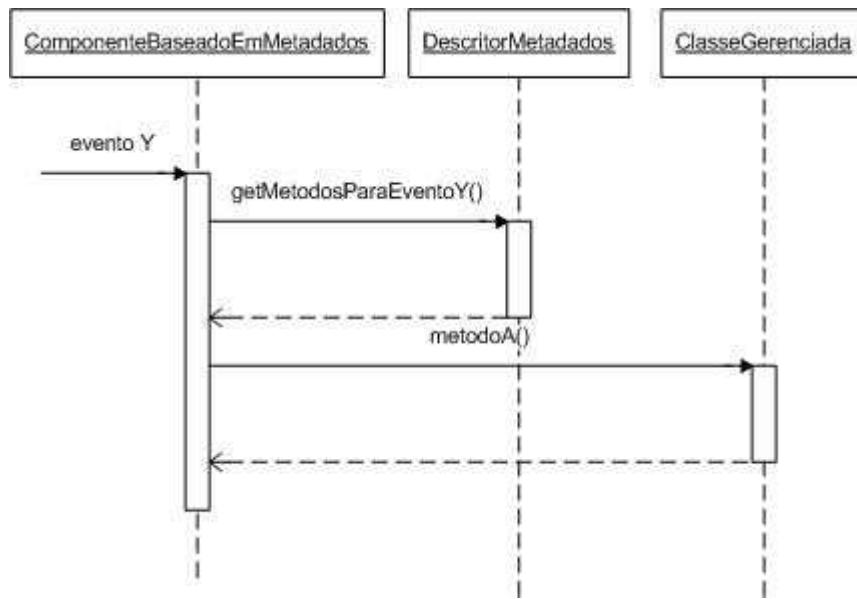
### Solução:

O padrão Gerenciador de Eventos propõe uma solução onde metadados associados aos métodos irão descrever a situação em que eles devem ser invocados pelo componente. Isso pode estar associado a eventos da aplicação ou a transições das fases do ciclo de vida das instâncias. No caso de uma classe persistente, exemplos dessas transições seriam antes e depois do seu armazenamento e antes e depois do seu carregamento. A Figura 9 apresenta a estrutura básica desse padrão.



**Figura 9. Estrutura do padrão Gerenciador de Eventos**

O **ComponenteBaseadoEmMetadados** verifica no **DescritorMetadados** quais serão os métodos a serem invocados em resposta a um determinado evento ocorrido. O componente irá invocar na **ClasseGerenciada** o método adequado. A Figura 10 apresenta um diagrama de sequência com essa lógica.



**Figura 10. Seqüência utilizada no padrão Gerenciador de Eventos**

#### Participantes:

- ClasseGerenciada: É a classe da aplicação que possuirá métodos marcados com metadados para serem executados em resposta a determinado evento. É importante notar que ela não herda nenhuma classe nem implementa interfaces do componente.
- DescritorMetadados: Descreve os métodos da ClasseGerenciada que deverão ser invocados em resposta a um evento.
- ComponentBaseadoEmMetadados: É o componente responsável por, a partir dos dados do DescritorMetadados, invocar os métodos que serão executados em resposta a determinado evento da ClasseGerenciada.

#### Consequências:

- Desacoplamento das classes da aplicação para as classes do componente, devido a estratégia de utilizar metadados e não interfaces, para que o componente saiba qual método deve ser invocado.
- Os metadados poderiam incluir informações mais específicas a respeito de quando aquele método seria invocado (exemplo: uma faixa de horário). Isso daria uma maior granularidade na invocação dos eventos.
- É possível possuir mais de um método em uma mesma classe para um mesmo tipo de evento.

#### Usos Conhecidos:

O padrão Gerenciador de Eventos é adotado no framework de testes JUnit (2008) para marcar os métodos de teste e os relativos ao ciclo de vida de uma suite de testes. Isso possibilita que uma mesma classe possa possuir diversos métodos de teste.

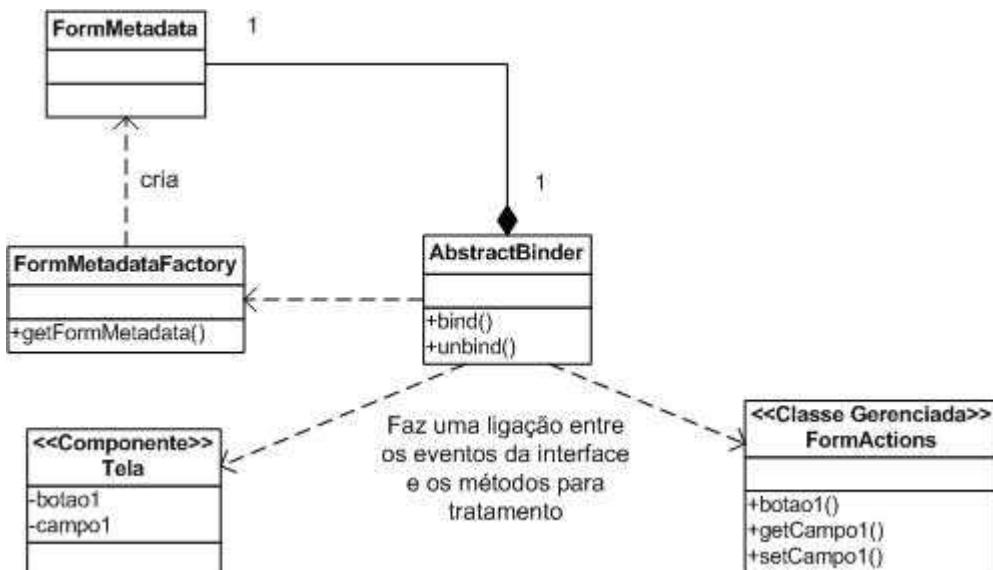
A especificação EJB 3 (JSR 220, 2003) utiliza essa abordagem para que determinados métodos sejam chamados pelo container durante as fases do ciclo de vida de um EJB. A API JPA também define metadados para as entidades que definem métodos para serem chamados durante momentos específicos de atividades de persistência daquela instância.

O framework Genesis (2008) utiliza o mesmo padrão, mas para eventos de interface com o usuário. Os metadados mapeam eventos de botões e outros componentes de interface para métodos.

### **Exemplo:**

O framework Genesis (2008) utiliza esse padrão para o tratamento de eventos de interface. Com o uso do framework é possível a definição de métodos que irão tratar eventos em uma classe e mapeá-los para os componentes correspondentes usando anotações e convenções de código. A Figura 11 apresenta de forma simplificada a estrutura do Genesis.

A classe AbstractBinder é especializada por classes como o SwingBinder e SWTBinder permitindo uma certa extensibilidade para a suite de componentes gráficos utilizada. Essas especializações recebem como parâmetro no construtor a classe anotada para o tratamento dos eventos e um container de componentes da suite gráfica utilizada. O método bind() cria os componentes mais específicos para fazer a ligação dos eventos.



**Figura 11. Resumo da estrutura do Genesis**

A Figura 12 apresenta um exemplo, retirado do site do projeto, no qual a classe ExemploCallWhenForm faz a sincronização de um campo chamado numeroDependentes e chama o método calculaTaxasAdicionais() quando o campo assume um valor maior que zero. Esse exemplo ilustra bem a maior granularidade que pode se conseguir com a utilização de metadados para o gerenciamento de eventos.

```

@Form
public class ExemploCallWhenForm {
    private int numeroDependentes;
    public int getNumeroDependentes() {
        return numeroDependentes;
    }
    public void setNumeroDependentes(int numeroDependentes) {
        this.numeroDependentes = numeroDependentes;
    }
    @Action
    @CallWhen("form.numeroDependentes > 0")
    public void calculaTaxasAdicionais() {
        // ...
    }
    // ...
}

```

**Figura 12. Configuração de um evento condicional com o Genesis**

#### **Padrões Relacionados:**

O padrão Gerenciador de Eventos pode ser utilizado em conjunto com o padrão Fábrica de Metadados, descrito na seção 5.1 deste artigo.

## **6.2. Mapeamento entre Representações**

#### **Intenção:**

Mapear entidades de negócio para a tradução dentre as diversas representações existentes em uma aplicação.

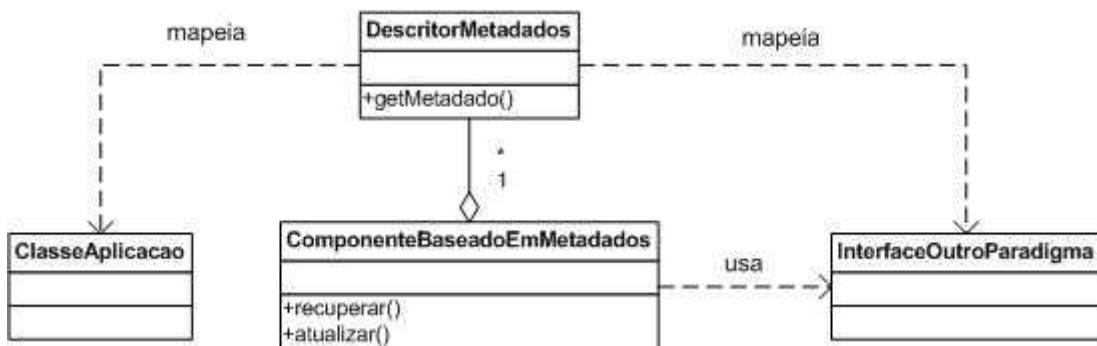
#### **Forças:**

- Uma representação não deve ser dependente da outra.
- Não se deseja criar classes de tradução para cada entidade.

#### **Solução:**

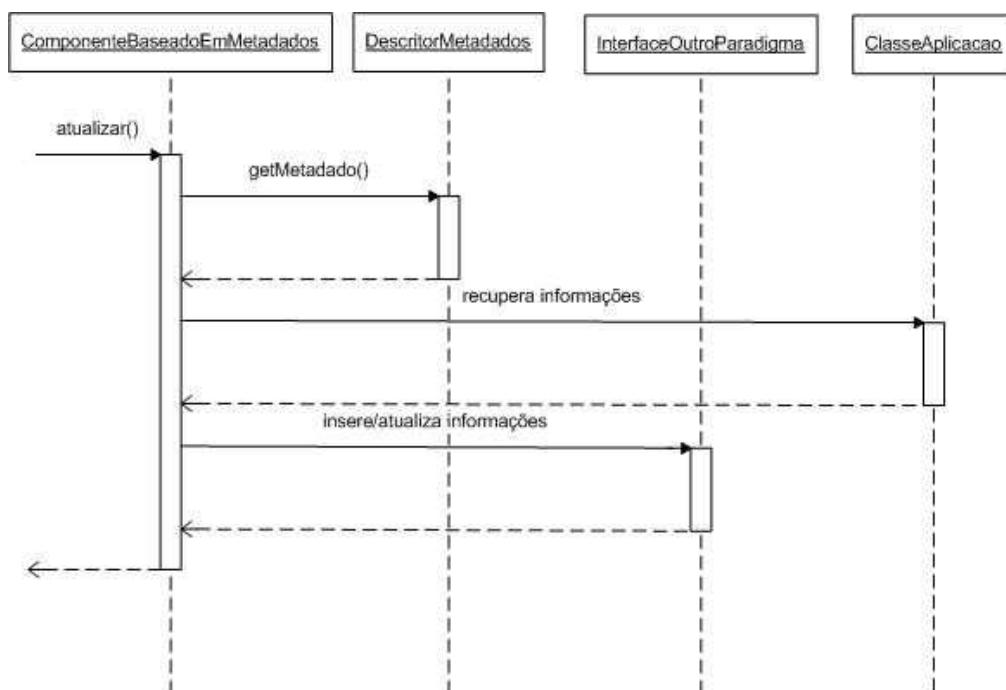
Esse padrão propõe a utilização de metadados para realizar o mapeamento entre diferentes representações de uma mesma entidade. Exemplos de representação de uma entidade são através de classes, bancos de dados, XML e interfaces gráficas.

Muitas vezes, a representação da entidade não pode ser acessada de forma direta, como quando essa representação é uma classe. Dessa forma, a representação será acessada através de uma API ou um componente. Por exemplo, para acesso a entidade no formato XML é preciso utilizar uma API como DOM e SAX. O diagrama de classes para o padrão está representado na Figura 13.

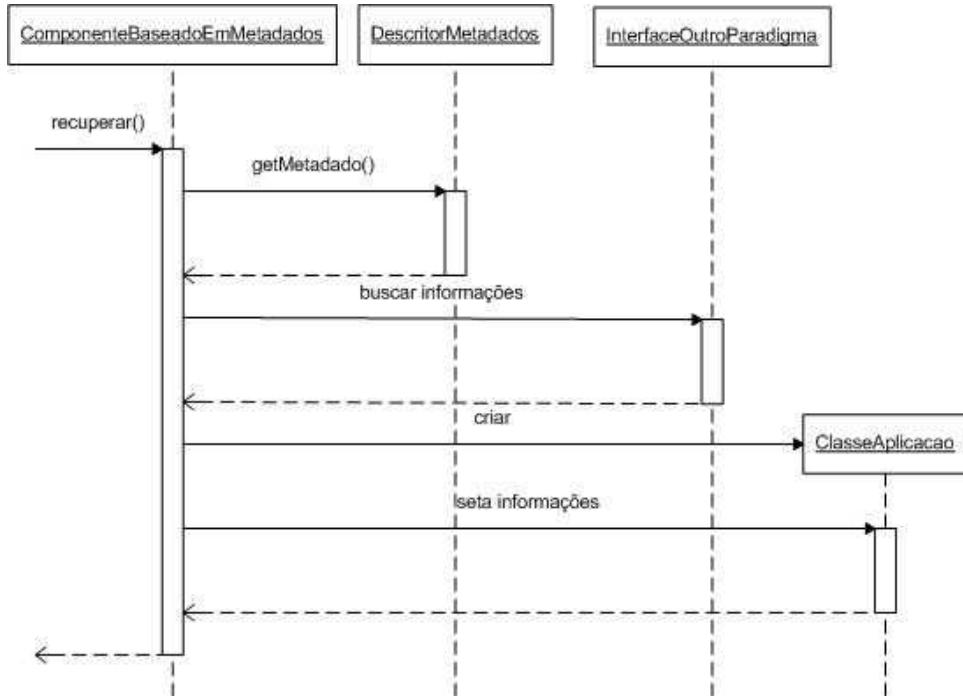


**Figura 13. Estrutura do Padrão Mapeamento entre Representações**

Nesse padrão, o componente se baseia nos metadados de mapeamento para acessar os dados da classe da aplicação e a interface de uma API que dará acesso a representação feita em outro paradigma. A Figura 14 apresenta um diagrama de sequência que mostra a atualização da entidade representada no outro paradigma a partir da classe da aplicação. Na Figura 15 o diagrama de sequência mostra o procedimento contrário, onde a classe da aplicação que recebe os dados da representação feita em outro paradigma.



**Figura 14. Seqüência de atualização do ComponenteBaseadoEmMetadados**



**Figura 15. Seqüência do método de recuperação de dados do ComponenteBaseadoEmMetadados**

#### Participantes:

- DescritorMetadados: Possui informações que mapeiam a ClasseAplicacao em relação a representação em outro paradigma.
- InterfaceOutroParadigma: Representa as interfaces ou classes para acesso as representações em outro paradigma. Como exemplo, no caso da JPA (JSR 220, 2003) essa interface seria o JDBC (2007) e no caso do SwingBean (2007) a API Swing (JSR 296, 2008).
- ComponenteBaseadoEmMetadados: Responsável pela recuperação e atualização de dados entre os diferentes paradigmas.
- ClasseAplicação: Classe da aplicação que representa uma entidade que também será representada em outro paradigma.

#### Consequências:

- Permite que a aplicação interface com o componente utilizando as próprias classes da aplicação.
- Desacopla a aplicação da API que realiza a interface com o outro paradigma onde a classe da aplicação também será representada.
- Elimina o código imperativo gerado para transformar as classes da aplicação na outra representação e vice-versa.

## Usos Conhecidos:

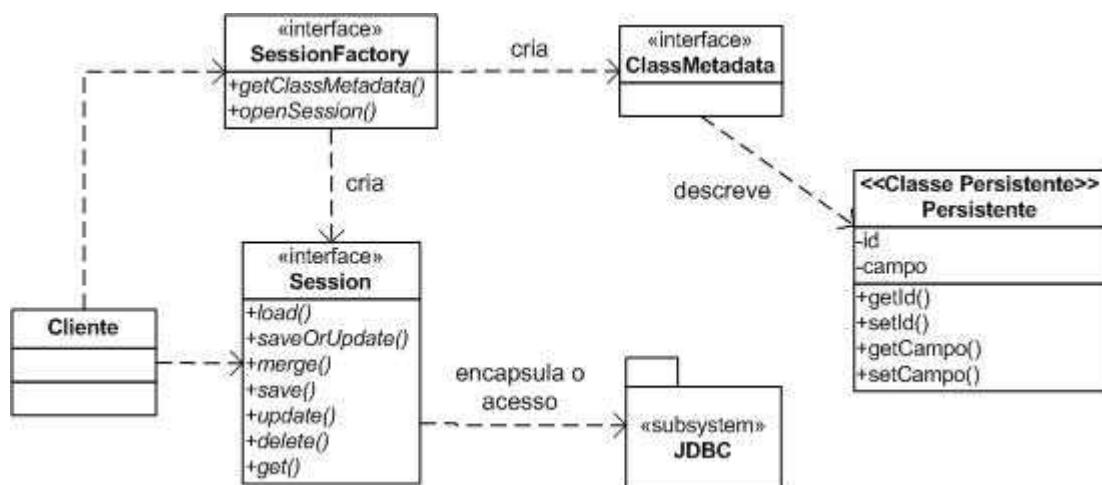
O padrão Mapeamento entre Representações pode ser observado no Hibernate (Bauer and King 2004) e na JPA (JSR 220, 2003), no mapeamento das classes persistentes da aplicação para as tabelas do banco de dados, utilizando anotações (JSR 175, 2003) ou arquivos XML. A API JAXB (2008) utiliza a mesma estratégia para fazer o mapeamento entre uma estrutura de classes e elementos de um documento XML.

O framework SwingBean (2007) utiliza os metadados relativos as propriedades de uma classe para mapea-las para campos de um formulário ou colunas de uma tabela. Esse mapeamento é utilizado não só para o desenho da interface gráfica, mas também para a inserção e recuperação das instâncias das classes da aplicação nos componentes.

## Exemplo:

O framework Hibernate (Bauer and King 2004) encapsula o acesso ao banco de dados pela API JDBC e realiza o mapeamento objeto-relacional utilizando anotações e/ou documentos XML. A Figura 16 apresenta um diagrama com algumas interfaces do Hibernate, sem entrar em muitos detalhes de como é o seu funcionamento interno.

A uma implementação de SessionFactory pode ser acessada pelo cliente para a criação de uma instância da classe Session. Através dessa implementação um classe cliente pode executar operações relativas a persistência de uma classe mapeada para o banco de dados. A implementação das classes do Hibernate interage com a API JDBC para o acesso a base de dados.



**Figura 16. Representação da estrutura do Hibernate**

## Padrões Relacionados:

O padrão Mapeamento entre Representações cria um componente que funciona como um Adapter (Gamma et al, 1994) para a interação com o outro paradigma de

representação, e para sua implementação recomenda-se a utilização conjunta do padrão Fábrica de Metadados, descrito na seção 5.1 deste artigo.

## 7. Conclusões

Esse artigo apresentou o conceito de frameworks e componentes baseados em metadados e alguns padrões de projeto para sua construção. Apesar de já existirem diversos componentes desse tipo disponíveis no mercado, esse artigo foi a primeira iniciativa para a definição de um conjunto de padrões de projetos para facilitar a criação de novos componentes baseados nesta abordagem. Dessa forma, de maneira mais explícita, segue uma lista com as principais contribuições trazidas por esse artigo:

- Generalização do conceito e definição do termo “componente baseado em metadados”.
- Análise de diversos frameworks que utilizam metadados, identificando e abstraindo soluções comuns utilizadas por eles.
- Classificação em padrões estruturais e padrões de uso dos padrões de projetos encontrados, visando facilitar a consulta e utilização dos mesmos.
- Documentação de dois padrões de projeto que abstraem a estrutura interna de um componente baseado em metadados.
- Documentação de dois padrões de projeto que abstraem situações onde existe e aplicabilidade de um componente baseado em metadados.

Diversas situações em que componentes baseados em metadados poderiam ser aplicados, passam desapercebidas por grande parte dos desenvolvedores, que deixam de explorar a flexibilidade e produtividade que os mesmos podem proporcionar. Os padrões de projeto descritos neste artigo auxiliam na identificação destas situações e na estruturação interna desses componentes. Esses primeiros padrões de projeto para uso de metadados são considerados a principal contribuição desse artigo.

Este artigo é parte de um trabalho de pesquisa, que ainda está em andamento, onde novos padrões ainda estão sendo identificados. O objetivo futuro desse estudo é a criação de um modelo que abstraia a estrutura de um componente baseado em metadados e defina uma linguagem de padrões que documente soluções e boas práticas para o seu desenvolvimento. Espera-se que esse estudo possa servir como ponto inicial para pesquisas relativas a modelagem de componentes baseados em metadados.

## Referências

BAUER, CHRISTIAN e KING, GAVIN. *Hibernate in Action*. Manning Publications, 2004.

DOV, A. B. Convention vs. Configuration. [S.n.t]. Disponível em <http://www.javalobby.org/java/forums/t65305.html>, 2006.

Esfinge Framework (2007), <http://sourceforge.net/projects/esfinge>, Abril.

- FAYAD, M. E., SCHIMIDT, D. C. AND JOHNSON, R. Building application frameworks: Object-oriented foundations of framework design. John Wiley & Sons, 1999.
- FOWLER, M. Inversion of Control Containers and the Dependency Injection pattern. [S.n.t.]. Disponível em <http://www.martinfowler.com/articles/injection.html>, 2004.
- GAMMA, E. HELM, R. JOHNSON, R. VLISSIDES, J. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1994.
- Genesis Framework (2008), <https://genesis.dev.java.net/>, Março.
- Hibernate (2008), <http://www.hibernate.org/>, Janeiro.
- Hibernate Validator (2008), <http://www.hibernate.org/412.html>, Fevereiro.
- JDBC (2007), <http://java.sun.com/javase/6/docs/technotes/guides/jdbc/>, Outubro.
- JSR 175. JSR 175: A Metadata Facility for the Java Programming Language. Disponível em <http://www.jcp.org/en/jsr/detail?id=175>, 2003.
- JSR 220. JSR 220: Enterprise JavaBeans 3.0. Disponível em <http://www.jcp.org/en/jsr/detail?id=220>, 2006.
- JSR 222: Java Architecture for XML Binding (JAXB) 2.0. Disponível em <http://jcp.org/en/jsr/detail?id=222>, 2006.
- JSR 296: Swing Application Framework. Disponível em <http://jcp.org/en/jsr/detail?id=296>, 2008.
- JUnit, Testing Resources for Extreme Programming (2008) <http://www.junit.org/>, Janeiro.
- KEITH, MIKE e SCHINCARIOL, MERRICK. Pro EJB 3 Java Persistence API. Apress, 2006.
- MASSOL, VICENT e HUSTED, TED. JUnit in Action. Manning Publications, 2004.
- PREE, W., POMBERGER, G. SCHAPPERT, A. SOMMERLAD, P. Active Guidance of Framework Development. Software - Concepts and Tools. v.16, i. 3, p. 136 – 145, 1995.
- ROUVOY, R. PESSEMIER, N. PAWLAK, R. e MERLE, P. Using attribute-oriented programming to leverage fractal-based developments. In Proceedings of the 5th International ECOOP Workshop on Fractal Component Model (Fractal'06), Nantes, France, Julho 2006.
- SCHWARZ, D. Peeking Inside the Box: Attribute-Oriented Programming with Java 1.5, In ON Java.com, O'Reilly Media, Inc., Junho 2004.
- SRIGANESH, RIMA PATEL. BROSE, GERALD AND SILVERMAN, MICAH. Mastering Enterprise JavaBeans 3.0. Wiley Publishing, Inc, 2006.
- SwingBean (2007), <http://swingbean.sourceforge.net/>, Abril.
- UML (2008), <http://www.uml.org/>, Janeiro.
- XDOCLET. XDoclet: Attribute Oriented Programming. Disponível em <http://xdoclet.sourceforge.net/xdoclet/index.html>, 2005.
- XML. Extensible Markup Language (XML). Disponível em <http://www.w3.org/XML/>, 2008.

# Linguagem de Padrões para Avaliação de Conhecimento em Objetos de Aprendizagem – Parte II

Ingrid T. Monteiro, Clayton Sandro, Cidcley T. de Souza

Centro Federal de Educação Tecnologia do Ceará  
 NASH (Núcleo Avançado em Engenharia de Software Distribuído e Sistemas Hipermídia) /ITTI (Instituto de Telemática)

{ingridtm, claysonsandro}@gmail.com, cidcley@cefetce.br

**Abstract.** Learning Objects (LO) are pedagogic resources which have been frequently applied in educational practices, mainly those ones computer-aided. Even with all the diversity of the objects from Web, it's possible identify some development patterns, considering some key aspects related to knowledge evaluation. This paper aims to give continuity to Part One of this Pattern Language previously developed, expanding the other patterns not detailed in its first part.

**Resumo.** Objetos de Aprendizagem (OA) são recursos pedagógicos cada vez mais recorrentes nas práticas educacionais, principalmente aquelas apoiadas por computador. Mesmo em meio a tanta diversidade de OA presente na Web, é possível identificar padrões de desenvolvimento destes objetos, considerando alguns aspectos chave, no caso desta pesquisa, em relação à avaliação do conhecimento. O presente artigo visa dar continuidade à Parte I desta Linguagem de Padrões anteriormente desenvolvida, detalhando os demais padrões não tratados em sua primeira parte.

## 1. Introdução

A partir da exploração de alguns repositórios de Objeto(s) de Aprendizagem – OA, disponíveis na Web, foi desenvolvida uma Linguagem de Padrões [1] que trata da avaliação da aprendizagem em OA, levando em consideração vários aspectos inerentes à forma de construir-se OA e de realizar a avaliação dentro dele. À época da criação da linguagem, foi divulgada apenas uma parte dos padrões criados, permanecendo o restante como trabalho futuro. O presente artigo propõe-se então a retomar os estudos desta linguagem e detalhar os demais padrões pertencentes a ela.

Antes de nos aprofundarmos na descrição dos padrões, será apresentada uma exposição da linguagem, seus fundamentos e motivações. Em seguida, será detalhado o relacionamento entre os aspectos identificados e os padrões desenvolvidos. Após essa fase, entram em cena os padrões da linguagem e a descrição dos seus elementos.

## 2. A Linguagem de Padrões

Os padrões descritos a seguir pertencem a uma linguagem de padrões [1] que abrange uma quantidade considerável de fatores relacionados à avaliação do conhecimento através de OA. Para o desenvolvimento da linguagem, foi definido um conjunto de aspectos importantes, relacionados à noção de avaliação. É a partir destes aspectos que os padrões da linguagem são organizados, de forma sistemática e didática.

Tomando o OA, apenas do ponto de vista da avaliação, seis Aspectos foram então considerados: *Tipo de Avaliação*, *Propósito do Objeto de Aprendizagem*,

*Seqüência das Questões, Relação entre Conteúdo e Avaliação, Recursos Utilizados e Comportamento Diante das Respostas.* Conforme descrito na Parte I da Linguagem [1], cada aspecto define um grupo/conjunto de padrões. A Tabela 1 relaciona os grupos que ordenam esta linguagem de padrões, descrevendo a questão básica que cada conjunto procura tratar.

**Tabela 1 - Definição dos aspectos/conjuntos da linguagem**

Aspecto	Descrição
<i>Tipo de Avaliação</i>	Determina quem avalia o aluno.
<i>Propósito do Objeto de Aprendizagem</i>	Estabelece o principal intuito do OA: conteúdo ou avaliação.
<i>Seqüência das Questões</i>	Trata da forma que se dá a seqüência das questões.
<i>Relação entre Conteúdo e Avaliação</i>	Determina quem vem primeiro: o conteúdo ou a avaliação.
<i>Recursos Utilizados</i>	Corresponde à maneira de apresentar as questões e problemas.
<i>Comportamento Diante das Respostas</i>	Estabelece o que acontece depois que o aluno responde a uma questão.

O número de padrões para cada um destes aspectos varia entre dois e quatro, totalizando dezesseis padrões para a linguagem criada. A Tabela 2 discrimina cada padrão de acordo com o aspecto considerado. Na Parte I da linguagem, foram trazidos os padrões referentes aos dois primeiros aspectos. Dessa forma, serão abordados aqui todos os demais padrões, destacados na tabela.

**Tabela 2 - Conjuntos da linguagem e seus padrões**

Aspecto	Padrões
1 Tipo de Avaliação	AUTO-AVALIAÇÃO <sup>1</sup> AVALIAÇÃO SUPERVISIONADA
2 Propósito do Objeto de Aprendizagem	OBJETO DE APRENDIZAGEM TEÓRICO OBJETO DE APRENDIZAGEM PRÁTICO
3 Seqüência das Questões	SEQÜÊNCIA LINEAR (3.1) SEQÜÊNCIA NÃO LINEAR (3.2)
4 Relação Entre Conteúdo e Avaliação	CONTEÚDO ANTES DA AVALIAÇÃO (4.1) CONTEÚDO E AVALIAÇÃO INTERCALADOS (4.2) AVALIAÇÃO ANTES DO CONTEÚDO (4.3)
5 Recursos Utilizados	QUESTÕES DE MÚLTIPLA ESCOLHA (5.1) QUESTÕES ABERTAS (5.2) SIMULAÇÕES (5.3) IMAGENS E GRÁFICOS (5.4)
6 Comportamento Diante das Respostas	TENTATIVA E ERRO (6.1) APRESENTAÇÃO DAS CONSEQUÊNCIAS (6.2) AUSÊNCIA DE GABARITO (6.3)

Apesar de os primeiros padrões da lista não fazerem parte do escopo do artigo, é importante descrevê-los minimamente, juntamente com os demais padrões a fim de que possamos compreender a linguagem como um todo.

## 2.1 Tipo de Avaliação

---

<sup>1</sup> Deste ponto em diante, o nome dos padrões serão exibidos em caixa-alta e, para aqueles detalhados neste artigo, haverá a indicação da seção correspondente, a fim de facilitar sua localização.

O primeiro aspecto, *Tipo de Avaliação*, como o nome indica, diz respeito ao tipo de avaliação presente no OA. A quem se direciona o resultado da avaliação? Quem deve tomar conhecimento da quantidade de acertos, do desempenho do aluno na resolução dos problemas? Dessa forma, foram definidos os seguintes padrões: AUTO-AVALIAÇÃO, obviamente, em que os alunos se auto-avaliam e AVALIAÇÃO SUPERVISIONADA, por meio dos quais os professores têm acesso aos resultados, acompanhando o processo de avaliação.

O que de fato determina a adoção de um ou outro padrão é a forma como o OA lida com os resultados das avaliações. É possível observar que aqueles do primeiro tipo estimulam os alunos a tomarem conhecimento de seu desempenho, sem intervenção do professor. Já os do segundo tipo direcionam os resultados ao avaliador, disponibilizando formas de comunicação, como e-mail, por exemplo.

## **2.2 Propósito do Objeto de Aprendizagem**

O aspecto *Propósito do Objeto de Aprendizagem* relaciona-se com a natureza do OA: para efeito desta linguagem de padrões, existem aqueles que apesar de oferecerem recursos para fixação e avaliação do conteúdo, possuem uma ênfase teórica, concentrando-se em abordar os conteúdos (OBJETO DE APRENDIZAGEM TEÓRICO) e aqueles que não possuem uma matéria explicativa explícita: eles são os próprios exercícios, são a própria avaliação (OBJETO DE APRENDIZAGEM PRÁTICO)<sup>2</sup>. Como não há matéria formal dentro do OA, para utilizá-lo, ou seja, resolver os problemas existentes, o aluno deve conhecer o assunto previamente, é preciso que ele tenha passado por uma aula a respeito ou tenha estudado o conteúdo.

## **2.3 Seqüência das Questões**

Seguimos então com a explicação dos aspectos analisados, para que se compreenda a relação dos conjuntos entre si e se perceba a linguagem de padrões em sua essência, agora tratando daqueles grupos cujos padrões serão descritos formalmente neste artigo.

A respeito da seqüência das questões (aspecto *Seqüência das Questões*), observou-se que há duas possibilidades: a) As questões são dependentes umas da outras e dadas de forma seqüencial (SEQÜÊNCIA LINEAR (3.1)). Para passar à questão seguinte (ou até para continuar no OA), é preciso resolver o problema, solucionar a pergunta anterior. b) Os problemas podem ser resolvidos em qualquer ordem (SEQÜÊNCIA NÃO LINEAR (3.2)). Caso não saiba a resposta de uma questão, ou até mesmo caso não tenha interesse por ela, o aluno pode passar para a próxima (se existir) ou prosseguir com a “leitura” do OA.

## **2.4 Relação entre Conteúdo e Avaliação**

---

<sup>2</sup> Existem ainda OA que não apresentam forma alguma de avaliação dentro dele, como exercícios e questões, trazendo apenas a matéria teórica. Por esse motivo (ausência de avaliação diretamente do OA), este tipo não é considerado dentro da definição da linguagem.

O aspecto seguinte, *Relação entre Conteúdo e Avaliação*, considera a relação entre o conteúdo apresentado e os exercícios, no que diz respeito à ordem de apresentação das matérias e das avaliações, dentro do OA. Alguns OA expõem todo o conteúdo primeiro e depois disponibilizam uma ou mais questões para o aluno (CONTEÚDO ANTES DA AVALIAÇÃO (4.1)). Outros intercalam conteúdo com exercícios: na medida em que vão sendo adicionados conceitos, são apresentados exemplos para o aluno aplicá-los (CONTEÚDO E AVALIAÇÃO INTERCALADOS (4.2)). Um terceiro tipo (AVALIAÇÃO ANTES DO CONTEÚDO (4.3)) é aquele que não apresenta conteúdo inicialmente. Há, antes dele, uma situação-problema que o aluno deve solucionar. Depois de resolvida, vem a explicação do assunto envolvido no contexto apresentado.

## 2.5 Recursos Utilizados

O próximo aspecto estabelecido, *Recursos Utilizados*, está relacionado aos recursos gráficos e de interface utilizados nas questões e exercícios. Há uma infinidade deles: de questões de múltipla escolha a simulações, as mais diversas, passando ainda por preenchimento de lacunas e outras respostas em aberto. Definimos para nossa linguagem os seguintes padrões: QUESTÕES DE MÚLTIPLA ESCOLHA (5.1), QUESTÕES ABERTAS (5.2), SIMULAÇÕES (5.3) e IMAGENS E GRÁFICOS (5.4).

## 2.6 Comportamento Diante das Respostas

*Comportamento Diante das Respostas*, o último aspecto identificado, refere-se ao tratamento dado pelo objeto às respostas dos alunos. Diz respeito ao que acontece dentro do OA após um erro ou um acerto. Para elaboração dos padrões deste grupo foram desconsiderados os objetos de aprendizagem que fornecem explicitamente a resposta dos problemas para os alunos, pois em maior ou menor grau não correspondem ao objetivo principal da linguagem em abordar os OA no contexto da avaliação do conhecimento.

Desta forma, nos padrões descritos, o aluno por si só deve descobrir a solução das questões apresentadas, não há como ele solicitar a resposta ao objeto. Algumas vezes, é possível retornar e tentar outra resposta (TENTATIVA E ERRO (6.1)), outras vezes, é apresentada a consequência para a resposta escolhida (APRESENTAÇÃO DAS CONSEQÜÊNCIAS (6.2)) e em outros casos, não é possível conhecer, pelo OA, a resposta certa (AUSENCIA DE GABARITO (6.3)).

## 2.7 Relacionamento Entre os Padrões

É importante observar que os padrões da linguagem relacionam-se entre si de duas maneiras, conforme apresentado na Figura 1: a) os padrões de um grupo podem ou não se relacionar com os padrões de outro grupo; b) os padrões de um mesmo grupo podem ou não se relacionar entre si. Em outras palavras, é possível que um OA adote vários padrões da linguagem ao mesmo tempo. De fato, um OA deve utilizar pelo menos um padrão (mas não necessariamente apenas um) de cada aspecto considerado, dependendo da intenção do desenvolvedor. Por exemplo, o primeiro objeto de aprendizagem mostrado neste artigo (item *a* do tópico **Usos Conhecidos** na seção 3.1), usa os

seguintes padrões: AUTO-AVALIAÇÃO, OBJETO DE APRENDIZAGEM TEÓRICO, SEQÜÊNCIA LINEAR (3.1), CONTEÚDO E AVALIAÇÃO INTERCALADOS (4.2), QUESTÕES ABERTAS (5.2), TENTATIVA E ERRO (6.1). Desta forma, é possível desenvolver uma variedade imensa de OA, combinando-se os padrões entre si de cada conjunto.

Na descrição dos padrões, isto ficará claro com alguns exemplos de OA que aparecem em mais de um padrão, demonstrando assim o intercâmbio existente entre os padrões de cada aspecto.

A Figura 1 a seguir apresenta a relação entre os grupos e padrões da linguagem. Destacamos em cinza, os grupos de padrões descritos neste artigo.

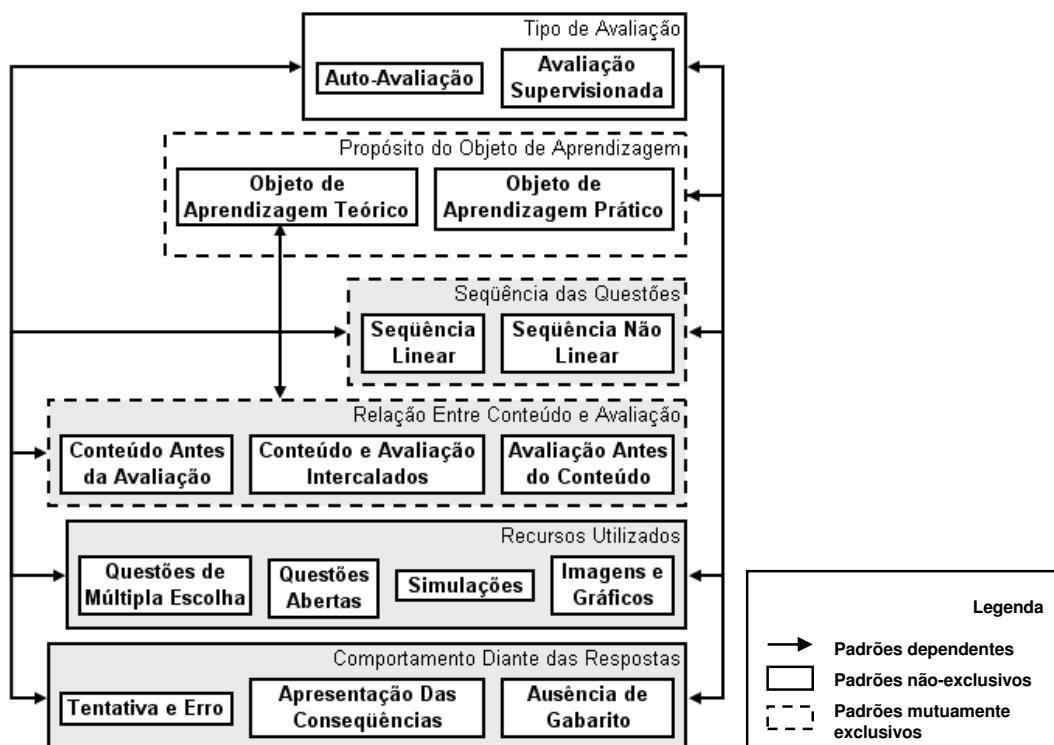


Figura 1 - Relação entre os padrões da linguagem

Conforme pode ser visto, existe relacionamento entre todos os grupos (indicado pela seta). Pela imagem, entende-se que todos os padrões de um grupo devem usar pelo menos um padrão de todos os outros grupos, com apenas uma exceção: vê se que no grupo *Propósito do Objeto de Aprendizagem*, apenas o OBJETO DE APRENDIZAGEM TEÓRICO relaciona-se com os padrões do grupo *Relação entre Conteúdo e Avaliação*, pois para utilizar os padrões deste último, é necessário que exista conteúdo formal no OA, o que não é o caso do OBJETO DE APRENDIZAGEM PRÁTICO. Todos os demais grupos relacionam-se livremente entre si, sem mais ressalvas.

A respeito do relacionamento entre padrões de um mesmo grupo, é possível perceber que três grupos apresentam padrões exclusivos entre si (caixa tracejada). Isso significa que um OA pode adotar apenas um dos padrões do grupo. Por exemplo, se um OA adota o SEQÜÊNCIA LINEAR (3.1), ele não pode utilizar o SEQÜÊNCIA NÃO LINEAR

(3.2), por razões lógicas<sup>3</sup>. Os outros três grupos possuem padrões que podem usar um outro padrão do mesmo grupo. Por exemplo, nada impede que um OA que use o SIMULAÇÕES (5.3), adote também o QUESTÕES ABERTAS (5.2) e o QUESTÕES DE MÚLTIPLA ESCOLHA (5.1).

A relação entre os grupos e padrões será reforçada na seção **Padrões Relacionados** de cada padrão descrito a seguir.

### **3. Aspecto Seqüência das Questões**

#### **3.1. Padrão SEQÜÊNCIA LINEAR**

##### **Contexto**

Ao desenvolver um objeto de aprendizagem, deseja-se constituí-lo de elementos que serão apresentados em seqüência e de forma hierárquica, para que o aprendizado de uma parte do OA seja adquirido após o aprendizado de uma parte anterior.

##### **Problema**

Como apresentar questões em objetos de aprendizagem de forma que o aluno siga sempre a mesma ordem previamente estabelecida?

##### **Forças**

Para utilizar esse padrão, é necessário haver alguma continuidade após a submissão da avaliação ao aluno. O padrão não se aplica a objetos de aprendizagem que, por exemplo, exponham um conteúdo e depois um (apenas um) problema a respeito. Depois de solucioná-lo, não há como dar continuidade ao objeto<sup>4</sup>. É necessário haver elementos (questões, problemas e também conteúdo) que possam ser postos em seqüência.

##### **Solução**

Disponha as questões e exercícios dentro do OA de forma seqüencial e evolutiva, de maneira que, apenas solucionando o problema apresentado, o aluno consiga prosseguir dentro do OA (outra questão ou conteúdo). O aluno, neste caso, deve ser informado de que não acertou a resposta e que deve tentar novamente.

##### **Racional**

---

<sup>3</sup> Tal afirmação pode levar à crítica de que pode ocorrer de um OA, em um primeiro momento, apresentar as questões sequencialmente, obrigando o aluno a solucioná-las linearmente e, em um segundo momento, passe a deixar livre a resolução das questões, conforme a vontade do usuário. Entretanto, a adoção dos padrões é considerada no OA como um todo, assim, se ocorrer de o objeto apresentar uma seqüência linear entre suas questões, mesmo que antes disso tenha dado uma liberdade de navegação entre um outro grupo de questões, ele está de fato utilizando o SEQÜÊNCIA LINEAR (3.1).

<sup>4</sup> Um objeto apresentado mais à frente que corresponde ao que está sendo descrito é o “Reação do amadurecimento da banana”. Depois de descobrir o local para armazenar as bananas, o OA é finalizado.

Como o conteúdo é dado em uma seqüência obrigatória, o OA deve ser desenvolvido considerando-se bem a melhor ordem de se expor os conteúdos, escolhendo aquela mais eficaz para o aprendizado do aluno.

### Contexto Resultante

A utilização deste padrão gera de imediato duas consequências: a) as questões podem ser dispostas evolutivamente, começando das mais simples até chegar a um nível maior de complexidade; b) o aluno é constantemente desafiado a prosseguir, desejando sempre solucionar os problemas para dar continuidade ao OA.

### Usos Conhecidos

- A química dentro de um bolo [2]: Logo no início do OA, o usuário precisa fornecer as quantidades certas de ingredientes para fazer quatro bolos, a partir de uma receita. Caso ele não preencha as lacunas corretamente, não há mais nada a fazer, não há como continuar utilizando o OA. Mais adiante, depois de apresentar o conteúdo sobre balanceamento, há exercícios sobre o assunto. Novamente, para passar para a próxima questão, é necessário acertar a anterior. Pela Figura 2, após preencher as lacunas o OA informa que a resposta não está correta e estimula o aluno a tentar novamente, sem disponibilizar meios de mudar de tela.

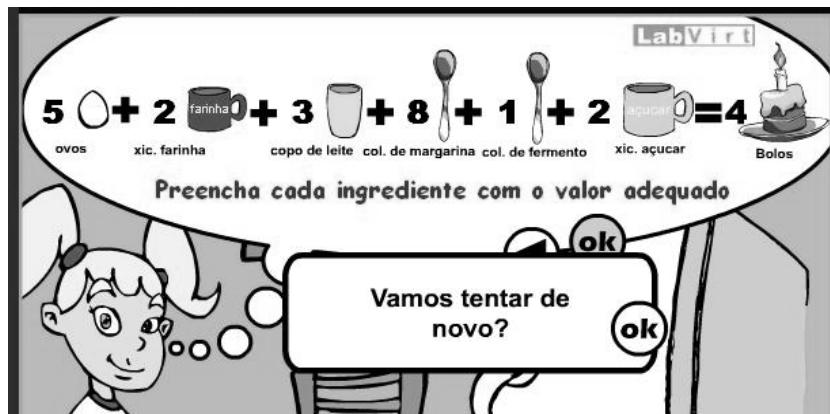


Figura 2 - SEQÜÊNCIA LINEAR no OA “A química dentro de um bolo”. Não é possível prosseguir sem acertar as medidas.

- Show atômico [3]: O objeto inicia expondo o conteúdo sobre os modelos atômicos e em seguida apresenta os problemas a serem resolvidos. Este objeto em particular é repleto de questões e todas seguem o padrão linear: uma coisa depois da outra, obrigatoriamente. A seta à esquerda abaixo do balão (Figura 3) indica o único caminho possível: voltar e tentar novamente.

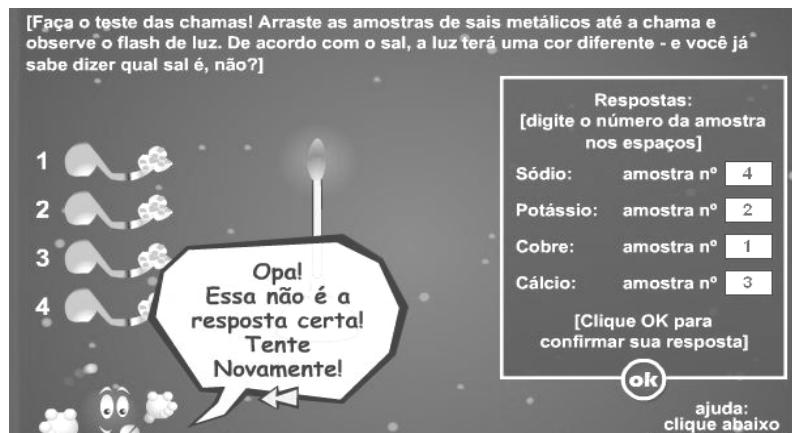


Figura 3 - SEQÜÊNCIA LINEAR no OA “Show atômico”. Não é possível prosseguir sem acertar os números das amostras.

### Padrões Relacionados

O autor do OA pode definir quem é o maior interessado no resultado da avaliação adotando os padrões AUTO-AVALIAÇÃO ou o AVALIAÇÃO SUPERVISIONADA.

Os padrões do grupo *Propósito do Objeto de Aprendizagem* são utilizados para determinar o foco do OA, que pode ter uma ênfase teórica ou prática.

Os padrões do grupo *Seqüência das Questões* são mutuamente exclusivos, logo os OA com SEQÜÊNCIA LINEAR não podem adotar o segundo padrão do grupo.

A relação entre o conteúdo abordado e os recursos da avaliação é determinada pelos padrões do quarto grupo da linguagem (*Relação entre Conteúdo e Avaliação*).

Toda a avaliação dentro do OA é feita através de recursos pedagógicos e de interface. Os padrões do grupo *Recursos utilizados* determinam as formas de apresentação dos problemas dentro do OA.

A maneira de se apresentar a resposta das questões ao aluno contribui para que ele tome conhecimento da qualidade de sua aprendizagem. Dessa forma, para decidir a maneira com que as respostas serão apresentadas, devem ser usados os padrões do grupo *Comportamento Diante das Respostas*.

### 3.2. Padrão SEQÜÊNCIA NÃO LINEAR

#### Contexto

O aprendizado através do OA deve se dar de forma não tão estática. Procura-se disponibilizar os elementos, mas deseja-se que o aluno os utilize conforme sua vontade e interesse.

#### Problema

Como distribuir as questões de um objeto de aprendizagem de forma que o aluno tenha liberdade para definir a ordem de seu uso?

#### Forças

O aluno pode estabelecer o modo de uso do OA tanto conforme as questões que consegue responder, quanto conforme seu interesse nas atividades apresentadas. Caso considere que pouco será agregado ao seu aprendizado, poderá abrir mão da questão atual em busca de outra atividade ou conteúdo de maior interesse, seja anterior ou posteriormente.

Dependendo do nível de conhecimento e de velocidade no aprendizado do aluno, ele tem condições de avançar no conteúdo do OA, conforme seu ritmo, sem seguir uma seqüência estabelecida.

A menos que o aluno já tenha conhecimento do assunto do OA, não é qualquer conteúdo que pode apresentar suas questões de forma independente. O aprendizado se dá, em geral, de maneira seqüencial e evolutiva.

O aluno pode preferir o método convencional, partindo-se de um nível mais simples para um mais complexo das questões.

### **Solução**

Disponha as questões e exercícios dentro do OA de forma livre, sem estabelecer uma seqüência única, de maneira que o aluno possa “navegar” entre as atividades do OA.

### **Racional**

Cabe ao professor/desenvolvedor do OA estabelecer o tipo de matéria e o tipo de aluno (nível, idade, período escolar) a que se aplica o SEQUÊNCIA NÃO LINEAR, considerando a complexidade que o padrão pode provocar dentro do objeto, o que em alguns casos pode ser prejudicial para o aprendizado.

### **Contexto Resultante**

A utilização de objetos de aprendizagem que seguem este padrão pode ser bastante estimulante para os alunos, pois é ele quem determina a forma de utilização do objeto, conforme seu modo de aprendizagem. Uma consequência negativa é que o aluno pode ignorar as questões que não quer ou não pode resolver, conformando-se com as partes/conteúdos que conseguiu atingir.

### **Usos Conhecidos**

- a. O salto dos recordes [4]: Este objeto de aprendizagem expõe algumas questões dispostas em seqüência. Entretanto, o aluno não é obrigado a segui-la. Caso deseje, pode “caminhar” pelas questões, respondendo ou não apenas aquelas que sabe ou se interessa. O aluno pode prosseguir com as questões a qualquer momento, clicando nas setas de esquerda e direita (em destaque na Figura 4).

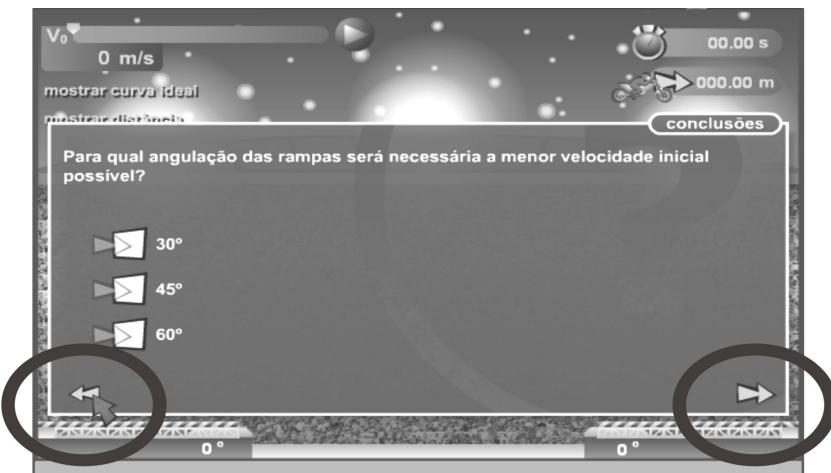


Figura 4 - SEQÜÊNCIA NÃO LINEAR no OA “Show atômico”. É possível navegar entre as questões.

- b. Calculadora quebrada [5]: Antes de iniciar o uso da “calculadora”, o usuário indica o nível escolhido (de 1 a 6), selecionando-o em um *combobox*. No primeiro nível, há as operações de adição e multiplicação, enquanto que no último, há apenas as operações de divisão e segunda potência, percebendo-se assim, a complexidade crescente dos níveis. O aluno fica livre para escolher o nível que achar mais adequado.

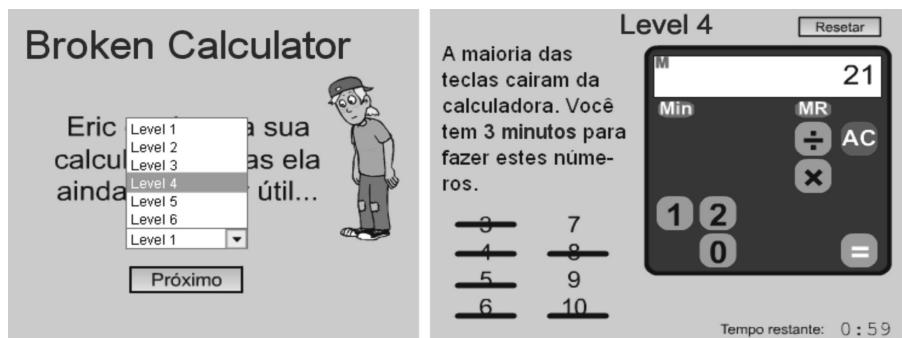


Figura 5 - SEQÜÊNCIA NÃO LINEAR no OA “Calculadora quebrada”. Antes de iniciar o uso do OA, o aluno seleciona um nível.

### Padrões Relacionados

Em um OA que utilize o SEQÜÊNCIA NÃO LINEAR é preciso definir se apenas o aluno terá acesso aos resultados de sua avaliação ou se eles serão consultados pelo professor. Conforme o objetivo, será usado então o AUTO-AVALIAÇÃO ou o AVALIAÇÃO SUPERVISIONADA.

A ênfase para a prática ou teoria é definida na escolha dos padrões do grupo *Propósito do Objeto de Aprendizagem*.

Conforme demonstrado aqui, as questões apresentadas devem seguir um dos tipos de seqüência, neste caso apenas o SEQÜÊNCIA NÃO LINEAR é possível.

As maneiras pelas quais o aluno vai praticar seu conhecimento por meio das questões são estabelecidas pelos padrões do grupo *Recursos Utilizados*.

O tratamento dado pelo OA às respostas dos alunos é determinado pelos padrões do grupo *Comportamento Diante das Respostas*.

#### **4. Aspecto Relação Entre Conteúdo e Avaliação**

##### **4.1. Padrão CONTEÚDO ANTES DA AVALIAÇÃO**

###### **Contexto**

Muitos professores são habituados ao método linear de ensino, preferindo expor o conteúdo antes de submeter seus alunos a uma avaliação. O mesmo é válido para alguns alunos que se interessam em resolver questões e exercícios apenas quando já sabem ou quando já viram a matéria associada.

###### **Problema**

Como permitir que o aluno, por meio de um OA, inicialmente aprenda o conteúdo e depois tenha seu aprendizado avaliado?

###### **Forças**

O estudo levado de forma totalmente teórica e depois prática pode ser cansativo para o aluno, que não vê aplicação imediata do que está aprendendo.

A elaboração da avaliação pós-conteúdo exige maior esforço, pois deve abranger de forma satisfatória tudo o que foi dado.

Pode ser difícil para o aluno solucionar as questões ao final do objeto, pois elas serão referentes a todo o conteúdo do OA.

A apresentação do conteúdo de forma contínua pode, entretanto, gerar um melhor aproveitamento, pois o aluno estará concentrado na compreensão do que está sendo lido e não terá interrupções em seu raciocínio.

É mais fácil a construção de OA cujos conteúdos sejam apresentados de forma contínua.

###### **Solução**

Desenvolva objetos de aprendizagem que apresentem o conteúdo informativo e apenas ao final deles exibam os questionamentos referentes à avaliação do aluno.

###### **Racional**

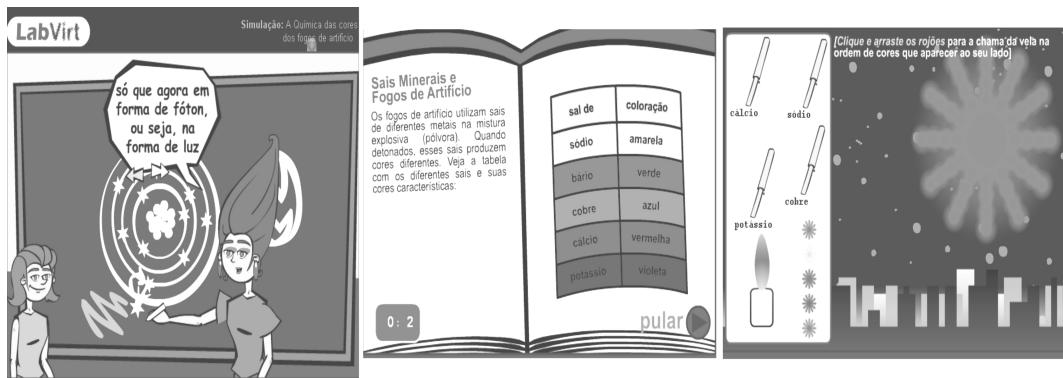
A questão da avaliação dever sempre se relacionar a todo o conteúdo do OA pode ser um ponto positivo pois os objetos de aprendizagem, em sua maior parte, referem-se a um assunto específico, geralmente de pequenas dimensões.

###### **Contexto Resultante**

A adoção do padrão CONTEÚDO ANTES DA AVALIAÇÃO corresponde à forma mais convencional de avaliar o aprendizado de um assunto qualquer: primeiro se aprende, depois se avalia...

## Usos Conhecidos

- A química das cores nos fogos de artifício [6]: Este OA tem um caráter teórico, portanto apresenta conteúdo formal. A última missão do aluno ao explorar o objeto é estabelecer (e acertar) as relações entre as cores emitidas pelos fogos de artifício e os elementos químicos. Toda essa teoria (duas primeiras telas da Figura 6) foi apresentada ao aluno antes de ele iniciar a atividade (última tela).



**Figura 6 – CONTEÚDO ANTES DA AVALIAÇÃO no OA “Química das cores nos fogos de artifício”**

- Show atômico [3]: Este é um ótimo exemplo de aplicação deste padrão. A explicação dos modelos atômicos é feita em várias telas e, apenas depois de tudo apresentado, inicia-se uma bateria de questões a respeito do conteúdo abordado.
- Propriedades das emissões radioativas – cargas [7]: O objeto tem por objetivo abordar as radiações, suas utilizações, características e elementos químicos envolvidos. Ele está estruturado da seguinte forma: inicialmente, apresenta exemplos de radiações presentes na natureza e nas tecnologias, explicando porque e como elas ocorrem; em seguida, o aluno é levado à simulação de um experimento de laboratório a partir do qual é possível conhecer as propriedades radioativas de alguns elementos químicos; após isso, inicia-se uma seqüência de questões a respeito do assunto abordado, momento em que o uso do objeto é finalizado.

Os radioisótopos são utilizados por meio de sua administração aos pacientes e passam a emitir suas radiações no órgão para onde são conduzidos.

Diagnóstico da tireoide - O estudo da tireoide é feito por meio da administração ao paciente de uma solução de iodo-131. A absorção e a distribuição do iodo na glândula é identificada por meio de um detector de radiação, colocado na frente do paciente.

**Questão 2**

Observe, agora, a equação de desintegração dos radioisótopos:

1.  $^{131}_{53}\text{I} \rightarrow ^{131}_{54}\text{Xe} + \beta + \gamma$
2.  $^{241}_{95}\text{Am} \rightarrow ^{237}_{93}\text{Np} + \alpha + \gamma$
3.  $^{232}_{90}\text{Th} \rightarrow ^{228}_{88}\text{Ra} + \alpha$
4.  $^{99m}_{43}\text{Tc} \rightarrow ^{99}_{43}\text{Tc} + \gamma$
5.  $^{60}_{27}\text{Co} \rightarrow ^{60}_{28}\text{Ni} + \beta$

O que acontece com a variação de massa e de número atômico quando as emissões são constituídas de:

- a. Apenas alfa ou alfa e gama.  O número atômico aumenta de 1, e o número de massa permanece constante.
- b. Apenas beta ou beta e gama.  O número atômico diminui de 2, e o número de massa diminui de 4.
- c. Apenas gama  Não há variação no número atômico e nem no número de massa.

**Experimento**

**Questão Anterior** **Próxima Questão**

Figura 7 - CONTEÚDO ANTES DA AVALIAÇÃO no OA “Propriedades das emissões radioativas – cargas”

## Padrões Relacionados

Um OA com conteúdo antes da avaliação é indicado para uso em conjunto tanto com o AUTO-AVALIAÇÃO quanto com o AVALIAÇÃO SUPERVISIONADA, dependendo de quem vai receber os resultados da avaliação.

O relacionamento deste padrão com o grupo *Propósito do Objeto de Aprendizagem* dá-se apenas com o OBJETO DE APRENDIZAGEM TEÓRICO, pois é necessário haver conteúdo e exercícios para que haja uma relação entre eles.

Os padrões dos grupos *Seqüência das Questões* são necessários para determinar, por exemplo, se o aluno tem acesso ao conteúdo apenas quando acerta uma questão inicial ou se pode “navegar” entre as questões apresentadas, solucionando-as na ordem que desejar.

O grupo *Recursos Utilizados* provê padrões para auxiliar no desenvolvimento de questões relacionadas ao conteúdo presente no OA.

Com os padrões do grupo *Comportamento Diante das Respostas*, é possível fornecer um retorno ao aluno sobre suas respostas e relacioná-las com o conteúdo apresentado.

## 4.2. Padrão CONTEÚDO E AVALIAÇÃO INTERCALADOS

### Contexto

Conteúdos abordados em um processo de ensino variam muito em complexidade. Dependendo do quanto denso ele é, faz-se necessário o uso de exercícios relativos a partes menores da matéria.

### Problema

Como permitir que o aluno estude um determinado conteúdo e que, ao mesmo tempo, seja submetido a avaliações “periódicas” dentro do OA?

### Forças

A abordagem de conteúdo e avaliação intercalados em objetos de aprendizagem é indicada para os casos em que se deseja separar o assunto abordado em seções

delimitadas pelas questões apresentadas. Conforme se for avançando na matéria, são apresentados problemas relacionados.

Pode ser difícil para o desenvolvedor do OA (ou professor) decidir em que ponto do objeto deve parar para apresentar um problema relacionado.

### **Solução**

Elabore objetos de aprendizagem que apresentem os questionamentos e problemas “dentro” da matéria apresentada, para que conteúdo e avaliação apareçam intercalados.

### **Racional**

Sempre que houver uma conclusão mínima sobre uma parte do conteúdo, é possível inserir uma questão a respeito, deixando os exercícios mais focados nos detalhes apresentados, ao invés de genéricos em relação à matéria como um todo.

### **Contexto Resultante**

O uso de objetos de aprendizagem que adotem esse padrão proporciona um aprendizado mais sistemático e organizado, pois vai avaliando o aluno à medida que ele avança no assunto apresentado.

O aluno pode sentir-se mais confiante em prosseguir no estudo do OA, pois já passou pela avaliação relativa à fase anterior do conteúdo.

### **Usos Conhecidos**

- a. A química dentro de um bolo [2]: É possível perceber a intercalação entre conteúdo e avaliação no momento em que é apresentada a base matemática relacionada ao balanceamento de equações na química: explica-se como se dá a dinâmica das proporções em uma receita de bolo e então o aluno aplica seu aprendizado em um problema apresentado. Em seguida, ele é introduzido ao conceito de balanceamento, propriamente dito. O objeto termina com algumas questões práticas sobre o assunto.
- b. Química dos alimentos [8]: Este é um objeto com bastante conteúdo e que apresenta avaliação em dois momentos: ainda no início do OA, quando leva o aluno a analisar em um laboratório virtual a composição (lipídios, proteínas e carboidratos) de alguns alimentos; e, depois de toda a explicação sobre as moléculas que formam os elementos dos alimentos, são lançadas perguntas a respeito da matéria apresentada.

**1 Tabela de Resultados**

Alimento	Lípidos	Proteínas	Carboidratos
Açúcar	0 % ✓	0 % ✓	99,5 % ✓
Arroz	2,90 % ✓	2,30 % ✓	32,30 % ✓
Macarrão a bolonhesa	2,94 % ✓	4,04 % ✓	20,44 % ✓

**2 Os Polímeros que formam o amido...**

...podem ser hidrolisados e depois de passar por tamanhos menores podemos chegar a monossacarídeos, como a glicose.

**3**

lipídios: 0    proteinas: 0    carboidratos: 99,5    outros: 0,50  
**Arroz**  
 lipídios: 2,90    proteinas: 2,30    carboidratos: 32,30    outros: 62,50  
**Macarrão a bolonhesa**  
 lipídios: 2,94    proteinas: 4,04    carboidratos: 20,44    outros: 72,58

1. Considerando que cada grama de proteína e de carboidrato libera em média 4 Kcal e cada grama de lipídio libera 9 Kcal, calcule a energia total dos alimentos que

**ok**

**Biblioteca**

**Figura 8 - CONTEÚDO E AVALIAÇÃO INTERCALADOS no OA “Química dos alimentos”:**  
**Avaliação (1) → Conteúdo (2) → Avaliação (3)**

## Padrões Relacionados

Os padrões do grupo *Tipo de Avaliação* determinam se apenas o aluno ou também o professor serão os destinatários do resultado da avaliação.

Tal como o padrão anterior, este também se relaciona apenas com o padrão OBJETO DE APRENDIZAGEM TEÓRICO do segundo grupo de padrões, que exige a presença de conteúdo e avaliação no mesmo OA.

Na dinâmica de avaliação e conteúdos intercalados, a questão da continuidade é relevante. Ela é definida adotando-se o padrão SEQÜÊNCIA LINEAR (3.1) ou SEQÜÊNCIA NÃO LINEAR (3.2).

Em relação aos demais padrões do mesmo grupo (*Relação Entre Conteúdo e Avaliação*) eles são excludentes entre si, não sendo possível o relacionamento entre eles.

No decorrer do OA são utilizados recursos didáticos para complementar a narrativa do objeto. Seus tipos são definidos utilizando-se os padrões do grupo *Recursos Utilizados* individualmente ou em conjunto.

Por fim, o *feedback* do OA ao aluno é determinado por meio dos padrões do último grupo *Comportamento Diante das Respostas*.

### 4.3. Padrão AVALIAÇÃO ANTES DO CONTEÚDO

#### Contexto

Uma maneira didática de ensinar é antes de iniciar o conteúdo propriamente dito, apresentar alguma situação real que se relate com este conteúdo. Uma estratégia para isso é lançar um problema, apresentar uma situação prática que deve ser solucionada antes de ser apresentada a matéria correspondente.

#### Problema

Como propiciar ao aluno à possibilidade de reflexão sobre um determinado assunto antes de ser apresentado o conteúdo, propriamente dito?

## Forças

Permitir o início do aprendizado a partir de um problema exige criatividade e conhecimento do professor/desenvolvedor para identificar situações que podem ser abordadas antes de expor o conteúdo apresentado.

Como este padrão pressupõe o início do uso do objeto por uma avaliação, é necessário que o aluno tenha os pré-requisitos necessários para solucionar o(s) problema(s) inicial(is)<sup>5</sup>.

## Solução

Desenvolva objetos que apresentem uma situação-problema para que o aluno solucione, configurando-se como uma avaliação, e apenas depois disso ingressem no assunto relacionado ao problema delineado.

## Racional

A maneira mais simples de implantar este padrão é buscando situações cotidianas que estejam ao alcance da realidade do aluno, pois assim ele vai buscar a solução nas suas próprias experiências.

## Contexto Resultante

A adoção deste padrão nos OA é bastante proveitosa, no que diz respeito ao incentivo dado ao aluno à reflexão, pois ele começa a aprender a partir de um problema cotidiano que ilustra a aplicação do conteúdo a ser abordado. Dessa forma, a matéria poderá ser compreendida de maneira mais natural, pois o aluno terá na memória a situação-problema pouco antes por ele enfrentada.

## Usos Conhecidos

- a. Reação do amadurecimento da banana [9]: O objeto insere-se em um contexto de simulação, em que o usuário deve escolher o melhor lugar para depositar um cacho de bananas verdes, para que elas amadureçam em dois dias. Não é fornecida nenhuma explicação prévia. O aluno utiliza seu “bom-senso” ou conhecimento anterior para solucionar o problema. Após a escolha feita, é apresentada a consequência: se a banana amadurece ou não. O aluno tem acesso ao conteúdo, depois que escolhe sua resposta. As opções estão indicadas pela caixa escura na Figura 9.

---

<sup>5</sup> Esta restrição é semelhante àquela presente no padrão OBJETO DE APRENDIZAGEM PRÁTICO, que exige o conhecimento prévio do conteúdo apresentado nas questões do objeto.



**Figura 9 - AVALIAÇÃO ANTES DO CONTEÚDO no OA "Reação do amadurecimento da banana"**

### Padrões Relacionados

OA que utilizam este padrão podem focar o destino dos resultados das questões e exercícios conforme o padrão escolhido do conjunto *Tipo de Avaliação*.

Como este padrão pressupõe, ainda que em um segundo momento, a existência de conteúdo, pode ser adotado apenas o OBJETO DE APRENDIZAGEM PRÁTICO, do segundo aspecto considerado.

Mesmo com conteúdo *a posteriori*, o OA pode exigir do aluno a resolução das questões antes de prosseguir com o conteúdo. Neste caso seria adotado o padrão SEQÜÊNCIA LINEAR (3.1). Caso contrário (não haver obrigatoriedade nas soluções), será adotado o segundo padrão do grupo.

A forma visual e didática da avaliação é definida com a utilização dos padrões do conjunto *Recursos Utilizados*, assim como o tratamento das soluções dos alunos é determinado com os padrões do último grupo.

## 5. Aspecto Recursos Utilizados

### 5.1. Padrão QUESTÕES DE MÚLTIPLA ESCOLHA

#### Contexto

Certos conteúdos são bastante ricos em detalhes e particularidades difíceis para o aluno memorizar. Nesses casos, é mais coerente que ele tenha a capacidade de relacionar conceitos e não exatamente saber a resposta ao pé da letra, pois é certo que quando precisar utilizar algo do conteúdo, poderá pesquisar e buscar meios para conhecer seu uso. Neste contexto, a opção por questões de múltipla escolha é mais justa do ponto de vista de não exigir a memorização absoluta do aluno.

#### Problema

Como avaliar o aluno objetivamente, permitindo que ele conheça de forma direta as possibilidades para a solução de um problema dado?

#### Forças

Questões objetivas são mais fáceis de serem avaliadas, pois a resposta é pré-definida, é estabelecido um “gabarito”. Entretanto, são mais difíceis de serem elaboradas, pois o professor deve pensar nas outras possibilidades e evitar ambigüidades, que são prejudiciais neste tipo de questão.

### **Solução**

Desenvolva problemas em que a resposta está entre as opções indicadas, num sistema de múltipla escolha.

### **Racional**

Diante da dificuldade de elaborar várias opções para a mesma questão, algumas estratégias podem ser adotadas: sentenças “óbvias”, complementares, opostas, “impossíveis” etc.

Cabe ao professor/desenvolvedor do OA estabelecer que tipo de conteúdo mais se aplique ao recurso de questões de múltipla escolha.

### **Contexto Resultante**

Em questões de múltipla escolha, o aluno pode ser avaliado em vários pontos da mesma matéria abordada, como se cada opção fosse, na realidade, uma questão diferente. Há então uma contribuição na capacidade do aluno em relacionar vários assuntos de um mesmo tema em uma mesma questão.

O aluno que passa por esse tipo de avaliação pode solucionar mais questões em menos tempo já que muitas vezes o trabalho de procurar a solução diretamente nas opções economiza esforço. Em compensação, o aprendizado pode ser comprometido, pois o aluno pode abrir mão de desenvolver a resolução até chegar à solução.

### **Usos Conhecidos**

- a. O salto dos recordes [4]: No ambiente de simulação do objeto, há um botão “conclusões”, pelo qual o aluno acessa as questões referentes à física presente no salto da motocicleta na rampa. São ao total, cinco perguntas com três opções, a respeito do ângulo da rampa, da velocidade da moto e de conclusões sobre a simulação.
- b. A química das cores nos fogos de artifício [6]: A avaliação presente neste objeto difere do OA anterior, pois não apresenta questões com opções diretas de resposta; o usuário deve relacionar os elementos químicos com as cores dos fogos de artifício. Ainda assim, o objeto utiliza-se do padrão QUESTÕES DE MÚLTIPLA ESCOLHA, pois a resposta do problema está explícita, todas as possibilidades são apresentadas.
- c. Show atômico [3]: Este é um bom exemplo de OA que adota este padrão, pois apresenta questões tanto de relacionamento (cores com elementos químicos, modelos atômicos com seus autores), quanto de múltipla escolha, propriamente dito (modelos atômicos).

### **Padrões Relacionados**

Os dois primeiros padrões da Linguagem estabelecem se o aluno será auto-avaliado (AUTO-AVALIAÇÃO) ou se o professor acompanha diretamente o processo (AVALIAÇÃO SUPERVISIONADA)

Tal padrão aplica-se tanto ao OBJETO DE APRENDIZAGEM TEÓRICO quanto ao OBJETO DE APRENDIZAGEM PRÁTICO, já que não estabelece a obrigatoriedade de conteúdo.

A forma da continuidade do OA é determinada pelos dois padrões do grupo *Sequência das Questões* e a ordem entre conteúdo e avaliação é estabelecida pelos padrões do grupo *Relação Entre e Conteúdo e Avaliação*.

Os padrões do conjunto deste tópico não são excludentes entre si, portanto OA que adotam QUESTÕES DE MÚLTIPLA ESCOLHA podem adotar os outros três padrões do grupo.

*Comportamento Diante das Respostas* é o grupo que estabelece o tratamento dado às soluções fornecidas pelo usuário do OA.

## **5.2. Padrão QUESTÕES ABERTAS**

### **Contexto**

Alguns conteúdos de cunho prático exigem que o próprio aluno solucione os problemas, sem que sejam apontadas no OA as possibilidades existentes.

### **Problema**

Como avaliar o aluno deixando para ele a tarefa de informar a solução dos problemas?

### **Forças**

Como é o aluno que fornece as repostas, questões técnicas, como diferença entre maiúsculas e minúsculas, número de caracteres permitidos, uso de acentuação e uso de abreviações, devem ser consideradas no julgamento da resposta pelo software.

### **Solução**

Elabore objetos de aprendizagem em que os problemas apresentados exijam respostas diretamente a partir do aluno, não exibindo opções ou possibilidades de solução.<sup>6</sup>

### **Racional**

Os objetos de aprendizagem que adotam este padrão devem obrigatoriamente utilizar caixas de texto para “receber” as respostas digitadas pelos alunos.

As particularidades do software que comprometam a aferição da resposta correta fornecida pelo aluno devem ser informadas no OA, no momento da resolução das questões.

---

<sup>6</sup> Ao falar em questões abertas, não nos referimos exatamente a questões subjetivas, que exigem grande elaboração do aluno... São questões que possuem uma resposta única, mas que o aluno tem que fornecer de “próprio punho”, não escolhendo entre opções determinadas e expostas no OA.

## Contexto Resultante

Ao utilizar OA com este padrão, o aluno é desafiado a obter a resposta apenas com o que aprendeu, deve lembrar por si só do que foi visto no OA ou fora dele, já que não são oferecidas opções de resposta.

## Usos Conhecidos

- A química dentro de um bolo [2]: Como o OA trata do balanceamento de equações, a maneira mais lógica de aplicar o conhecimento é fornecendo os valores corretos para que as equações fiquem balanceadas. E é exatamente o que é feito. Desde a analogia com a receita de bolo, o aluno deve fornecer as quantidades exatas de cada substância (no caso da receita, os ingredientes) para deixar a equação coerente.
- Até as últimas consequências [10]: O objeto inicia falando sobre um fotógrafo distraído que cai num precipício, mas se salva por uma pequena extensão de terra existente na parede do penhasco. O resgate chega e vai salvá-lo usando uma corda. O aluno deve informar qual a força necessária para o caminhão retirá-lo do buraco. São fornecidas algumas informações (botões “dica”) a respeito dos conceitos e cálculos necessários para descobrir o valor correto. Não há opções; ao aluno cabe apenas informar o valor na caixa de texto, solucionando o problema apresentado.



Figura 10 - QUESTÕES ABERTAS no OA "Até as últimas consequências". O aluno deve digitar o valor correto.

## Padrões Relacionados

O QUESTÕES ABERTAS relaciona-se com os padrões dos grupos *Tipo de Avaliação*, *Propósito do Objeto de Aprendizagem*, *Seqüência das Questões* e *Comportamento Diante das Respostas*, da mesma forma como já descrito nas seções **Padrões Relacionados** anteriores.

Apesar deste padrão, assim como os demais do grupo, ser direcionado ao formato das questões, ele também se relaciona com os padrões do grupo *Relação Entre Conteúdo e Avaliação* já que é possível existir conteúdo teórico dentro do OA.

Em relação aos outros padrões do grupo é possível que o mesmo OA possua vários padrões ao mesmo tempo, inclusive na mesma questão, como por exemplo, uma questão com SIMULAÇÕES (5.3) e QUESTÕES ABERTAS simultaneamente. A única

ressalva é que, apesar desse diálogo entre os padrões do grupo, é mais raro encontrar QUESTÕES ABERTAS e QUESTÕES DE MÚLTIPLA ESCOLHA (5.1) na mesma questão.

### **5.3. Padrão SIMULAÇÕES**

#### **Contexto**

Muitos conteúdos apresentados para os alunos são “chatos” e cansativos exatamente por não estabelecerem relações diretas com a realidade. O aluno não percebe, não visualiza facilmente a aplicação do assunto na prática.

#### **Problema**

Como fornecer aos alunos uma avaliação que considere a utilização do conteúdo na prática, próximo da realidade?

#### **Forças**

O OA que apresenta uma simulação exige um esforço extra de seu criador, pois muitas vezes exige a construção de uma narrativa, para contextualizar o problema apresentado e exige o uso de imagens, para representar personagens, ambientes e fenômenos.

#### **Solução**

Elabore objetos de aprendizagem baseados em simulações feitas por computador de situações reais, possíveis de acontecer. O caráter de simulação pode estar presente em toda a duração do OA, mas, para este padrão, é relevante que a avaliação, os problemas apresentados tragam estas simulações, de forma que os usuários solucionem as questões, baseando-se em situações “reais”, apresentadas no objeto.

#### **Contexto Resultante**

A adoção deste padrão é bastante interessante, pois propicia ao aluno uma percepção, mais próxima da realidade, do conteúdo e de suas aplicações. É uma forma didática, confortável e bastante válida de assimilar o conhecimento.

#### **Usos Conhecidos**

A grande maioria dos objetos de aprendizagem analisados para esta pesquisa, principalmente os do Labvirt e do RIVED, tiram proveito deste recurso, tanto nos momentos de explicação das matérias quanto ao aplicá-las em questões e exercícios.

- a. O salto dos recordes [4]: A avaliação inicial do OA consiste em ajustar o ângulo da rampa e a velocidade ideais para que o motociclista salte com sucesso. Conforme os valores estabelecidos pelo usuário, é apresentado o que acontece com a moto, na “realidade”: se é realizada a travessia ou se o motoqueiro derrapa. Há inclusive uma representação para a explosão da moto.

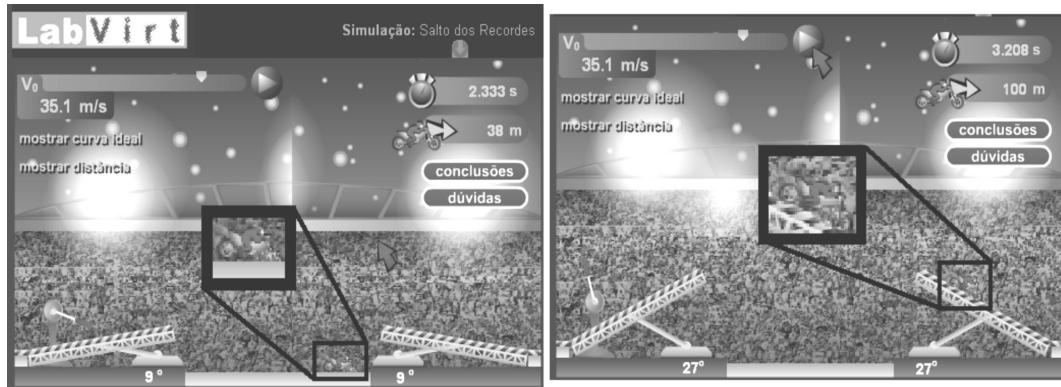


Figura 11 - SIMULAÇÕES no OA “Salto dos Recordes”

- b. Teodolito [11]: Neste OA, o aluno é desafiado a calcular a largura de um lago, para, sobre ele, ser construída uma ponte. O OA inicia, solicitando que o usuário imagine-se um engenheiro que foi convocado a executar tal tarefa. Já a partir da linguagem adotada, percebe-se o esforço em fazer do OA uma simulação da realidade, no contexto apresentado. O aluno deve, em seus cálculos, utilizar o teodolito, ou melhor, sua representação (caixa “controle” no centro da tela), como instrumento para auxiliar na aquisição das medidas e dos ângulos, tal qual aconteceria em uma situação real. As direções indicadas pelo teodolito no lago estão indicadas pelas setas.



Figura 12 - SIMULAÇÃO no OA "Teodolito". O aluno deve "usar" um teodolito da mesma forma como um engenheiro.

### Padrões Relacionados

O SIMULAÇÕES relaciona-se diretamente com todos os padrões dos grupos *Tipo de Avaliação*, *Propósito do Objeto de Aprendizagem*, *Seqüência das Questões*, *Relação Entre Conteúdo e Avaliação* e *Comportamento Diante das Respostas*, mantendo-se as restrições quando aos grupos cujos padrões são mutuamente exclusivos.

Considerando os outros padrões do grupo ao qual este padrão pertence, é livre a relação entre eles. O SIMULAÇÕES pode adotar indistintamente um ou mais dos outros três padrões do conjunto.

## 5.4. Padrão IMAGENS E GRÁFICOS

### Contexto

Diante da riqueza de possibilidades gráficas dos meios computacionais, é muito conveniente e indicado tirar proveito dela para a elaboração de objetos de aprendizagem interessantes, convidativos, que chamem o aluno para seu “consumo”. Além disso, os próprios alunos conhecem essa realidade e podem exigir formas mais elaboradas de se passar um conhecimento.

### Problema

Como permitir que o aluno avalie seu conhecimento através de recursos visuais mais aprimorados?

### Forças

Assim como ocorre no padrão SIMULAÇÕES (5.3), o uso de imagens e afins nas avaliações dos OA exige maior esforço criativo dos autores, que devem conhecer bem as técnicas de *layout* e desenho utilizadas no contexto computacional.

O uso abundante de ilustrações pode dar ao OA um caráter “infantil”, podendo desagradar a usuários mais velhos.

### Solução

Elabore objetos de aprendizagem nos quais as questões e problemas utilizem imagens, gráficos etc., como forma de deixar mais agradável e interessante a sua resolução.

### Racional

A adoção de imagens e gráficos nas resoluções das questões deve ser feita baseada no teor dos conteúdos e em seu público-alvo. A utilização de recursos visuais é mais adequada e indicada para determinadas matérias que necessitam de observação e para determinadas idades que assimilam melhor o conhecimento com o auxílio de técnicas mais didáticas, como o uso deste padrão.

### Contexto Resultante

A utilização de ilustrações no contexto da avaliação deixa o OA mais interessante, logo, o processo de aprendizagem pode ocorrer de forma mais natural, mais simples e agradável para o aluno.

### Usos Conhecidos

- a. Ácido no dia a dia [12]: Após a explicação sobre a acidez dos alimentos, o usuário deve escolher aqueles que são mais indicados para a personagem do OA, que está com uma gastrite. As opções são todas apresentadas em forma de ilustrações, representando cada alimento a ser escolhido.

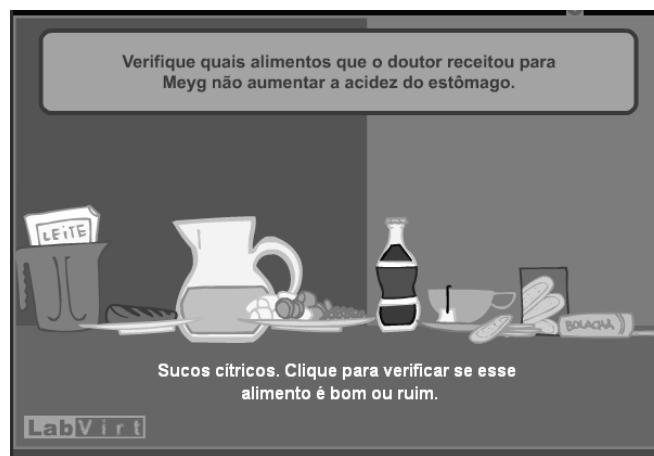


Figura 13 - IMAGENS E GRÁFICOS no OA "Ácido no dia a dia"

- b. Reação do amadurecimento da banana [9]: a personagem do OA deve armazenar as bananas verdes no melhor lugar para que elas amadureçam. O saco plástico, a geladeira, a jarra de água e a fruteira estão todos presentes em forma de desenho como opções.
- c. Mendel não sabia disso [13]: Este é um objeto de aprendizagem extremamente visual e sua avaliação é totalmente constituída por imagens. O aluno deve construir um “curta-metragem” contando a história de um casal de cachorros, desde que se conhecem até o nascimento de seus filhotes, utilizando as imagens disponíveis. Entre elas, há esquemas ilustrando a dinâmica nas células no processo de reprodução e a genética envolvida. O resultado final (filme) é exibido na tela de uma “televisão”, com direito a controle remoto e comandos de vídeo (play, pause...).

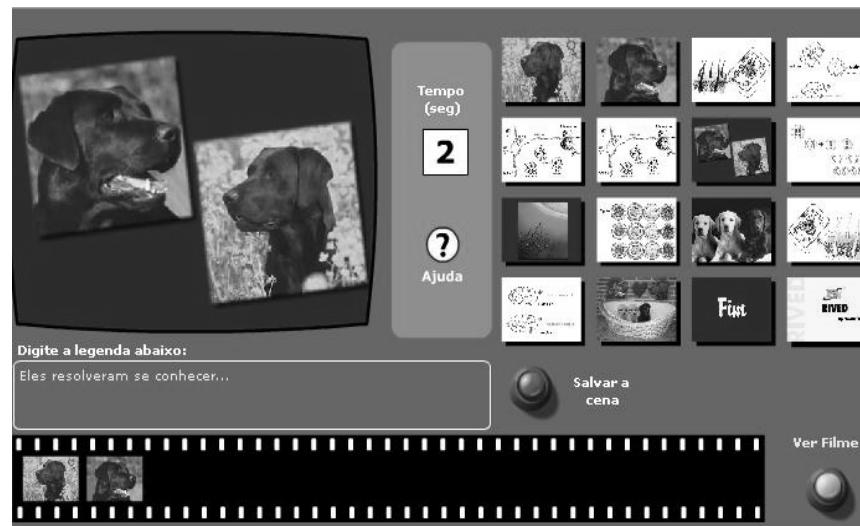


Figura 14 - Uso de imagens no OA "Mendel não sabia disso". A avaliação é a montagem de um filme.

## Padrões Relacionados

Um OA que explora imagens e gráficos em suas atividades é indicado para uso em conjunto tanto com o AUTO-AVALIAÇÃO quanto com o AVALIAÇÃO SUPERVISIONADA, dependendo de quem vai receber os resultados da avaliação.

O relacionamento deste padrão com o grupo *Propósito do Objeto de Aprendizagem* dá-se tanto com o OBJETO DE APRENDIZAGEM TEÓRICO, quanto com o OBJETO DE APRENDIZAGEM TEÓRICO.

Os padrões do grupo *Seqüência das Questões* são necessários para determinar, por exemplo, se o aluno tem acesso ao conteúdo apenas quando acerta uma questão inicial ou se pode “navegar” entre as questões apresentadas, solucionando-as na ordem que desejar.

O autor do OA fica livre para adotar o IMAGENS E GRÁFICOS isoladamente ou em conjunto com os demais padrões do grupo atual (*Recursos Utilizados*).

Com os padrões do grupo *Comportamento Diante das Respostas*, é possível fornecer um retorno ao aluno sobre suas respostas e relacioná-las com o conteúdo apresentado.

## **6. Aspecto Comportamento Diante das Respostas**

### **6.1. Padrão TENTATIVA E ERRO**

#### **Contexto**

Professores e educadores desejam que a avaliação do desempenho de seus alunos seja feita de forma justa, de maneira que o aluno desenvolva o raciocínio para chegar à solução dos problemas. Nesta perspectiva, são preferíveis questões que não apresentam as respostas de seus problemas e deixem para os alunos a missão de solucioná-los.

#### **Problema**

Como permitir que um aluno, caso não consiga solucionar um problema, não tenha acesso imediato à resposta correta?

#### **Forças**

O método da tentativa e do erro pode ser inadequado para a auto-avaliação, pois o aluno dificilmente saberá a resposta, a menos que lance seu “palpite” até ter sucesso.

Em contrapartida, este método pode aguçar a curiosidade do aluno, que tentará a qualquer custo solucionar o problema, desenvolvendo seu raciocínio, relembrando as matérias relacionadas, fazendo e refazendo cálculos até acertar a questão.

#### **Solução**

Elabore avaliações em objetos de aprendizagem que deixe apenas para o aluno a missão de encontrar a resposta, não havendo como conferi-la. Caso o aluno não conheça de imediato a solução, conseguirá resolver o problema apenas no método da tentativa e do erro.

#### **Racional**

Não seguem este padrão os OA que permitem ao aluno tentar algumas vezes, mas depois informam a solução. Como exemplo há o OA “Teodolito”, já comentado aqui, que fornece a resposta aos alunos. Em seu primeiro exercício, depois que o usuário erra a resposta pela segunda vez, aparece a mensagem: “Você errou mais de uma vez. As respostas corretas são (...”).

Dependendo de como seja lançado o OA para o aluno, caso haja o acompanhamento do professor, este poderá fornecer a resposta diretamente para o aluno explicando a causa do seu erro.

### Contexto Resultante

Uma consequência negativa é que o aluno pode permanecer tanto no erro que desista de continuar tentando. Por vezes é mais proveitoso ele responder a uma questão e depois conferir a resposta, compreendendo de imediato a causa do seu erro.

### Usos Conhecidos

- A química dentro de um bolo [2]: Depois da analogia com a receita de bolo, são apresentadas algumas equações químicas para serem balanceadas; o aluno deve preencher as lacunas com o coeficiente que considera adequado. Caso os valores não estejam corretos, o aluno não consegue prosseguir no OA. Entretanto, ele pode tentar quantas vezes quiser, até conseguir平衡ear a equação.
- O resgate [14]: O objetivo deste OA é fazer com que um suicida seja salvo ao tentar pular de um viaduto (em destaque na Figura 15). O desafio é informar o seu tempo de queda e a posição que o caminhão dos bombeiros (seta clara) deve estar no momento do pulo para que ele seja salvo. São fornecidos os dados iniciais necessários para o cálculo e se, mesmo assim, o aluno não conseguir chegar aos valores corretos pode dar seu palpite à vontade, até conseguir salvar o homem...



Figura 15 - TENTATIVA E ERRO no OA "O Resgate"

### Padrões Relacionados

O professor/desenvolvedor do objeto de aprendizagem pode definir quem é o maior interessado no resultado da avaliação, adotando os padrões AUTO-AVALIAÇÃO ou o AVALIAÇÃO SUPERVISIONADA.

Os padrões do grupo *Propósito do Objeto de Aprendizagem* são utilizados para determinar o foco do OA, que pode ter uma ênfase teórica ou prática.

Os padrões do grupo *Seqüência das Questões* são mutuamente exclusivos, logo os OA com SEQÜÊNCIA LINEAR (3.1) não podem adotar o segundo padrão do grupo.

A relação entre o conteúdo abordado e os recursos da avaliação é determinada pelos padrões do quarto grupo da linguagem (*Relação entre Conteúdo e Avaliação*).

Toda a avaliação dentro do OA é feita através de recursos pedagógicos e de interface. Os padrões do grupo *Recursos utilizados* determinam as formas de apresentação dos problemas dentro do OA.

Considerando apenas o grupo atual, o TENTATIVA E ERRO relaciona-se livremente com os outros dois, ou seja, um OA pode adotar mais de um padrão do grupo em suas avaliações.

## **6.2. Padrão APRESENTAÇÃO DAS CONSEQUÊNCIAS**

### **Contexto**

Muitas vezes, os alunos solucionam questões e problemas acreditando que estão respondendo corretamente e quando descobrem que a resposta está errada, não compreendem o porquê.

### **Problema**

Como apresentar problemas aos alunos de forma que eles compreendam o resultado de suas respostas?

### **Forças**

A apresentação das consequências para as respostas dadas pelos alunos é útil para que ele entenda de forma mais abrangente o conteúdo apresentado, pois fica mais claro compreender, em sua essência, o que é certo e o que é errado no contexto do problema.

### **Solução**

Desenvolva objetos de aprendizagem que, a cada resposta dada pelo aluno, apresente a sua consequência, explicando-se os motivos de seu erro.

### **Racional**

A apresentação das consequências é aplicada essencialmente em questões de múltipla escolha, pois assim, é possível desenvolver o “desfecho” para as respostas conhecidas pelo OA<sup>7</sup>.

### **Contexto Resultante**

---

<sup>7</sup> Algumas vezes, o mesmo se aplica a questões de preenchimento de lacunas, pois há apenas uma resposta certa e tudo diferente dela apresenta, em geral, a mesma consequência, como ocorre no OA “O Resgate”, em que, para todas as posições informadas diferentes da correta, aparece a imagem do suicida chocando-se com a pista.

Quando a consequência da solução apresentada é exibida ao aluno este comprehende melhor as particularidades do tema abordado no OA, favorecendo o aprendizado.

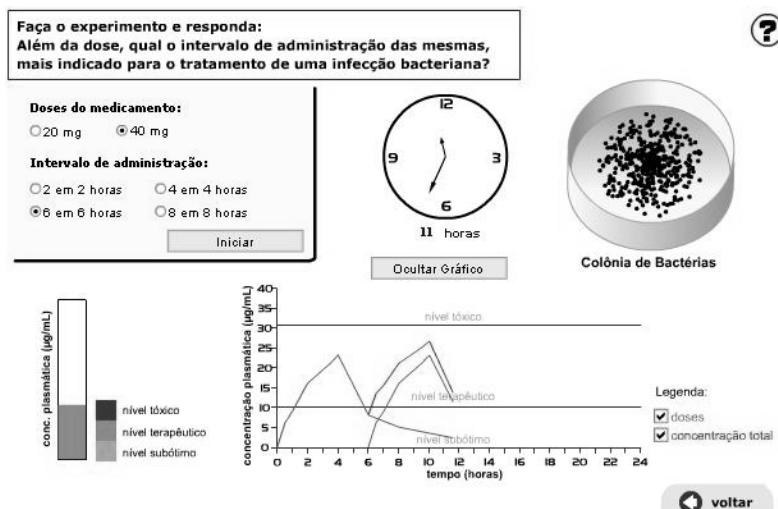
### Usos Conhecidos

- a. Ácido no dia a dia [12]: Na narrativa do OA, depois do médico explicar a Meig porque ela está com azia, o usuário deve informar os alimentos mais adequados para diminuir a acidez no estômago da paciente. A cada alimento escolhido, a personagem informa se melhorou ou piorou e são mostradas as substâncias favoráveis ou não em cada alimento.



Figura 16 - APRESENTAÇÃO DAS CONSEQÜÊNCIAS no OA “Ácido no dia a dia”.

- b. No tempo certo [15]: Este OA trata da ação de medicamentos antibióticos no combate a infecções bacterianas. Nas questões apresentadas, o aluno deve lidar com quantidade das doses, concentração, horas de medicação etc. Todas as questões são de múltipla escolha e a cada opção é apresentado um gráfico informando se a concentração do medicamento no organismo está a um nível subótimo, terapêutico ou tóxico. Há ainda, um frasco mostrando a diminuição da colônia de bactérias conforme a evolução do tratamento.



**Figura 17 - APRESENTAÇÃO DAS CONSEQUÊNCIAS no OA "No tempo certo". Todas as opções apresentam o que acontece com o organismo.**

### Padrões Relacionados

Determinando qual padrão utilizar do primeiro grupo, determina-se se o aluno será auto-avaliado (AUTO-AVALIAÇÃO) ou se o professor acompanha diretamente o processo (AVALIAÇÃO SUPERVISIONADA). OA que adota o APRESENTAÇÃO DAS CONSEQUÊNCIAS pode utilizar o primeiro ou o segundo, conforme o interesse do professor/desenvolvedor.

O padrão desta seção aplica-se tanto ao OBJETO DE APRENDIZAGEM TEÓRICO quanto ao OBJETO DE APRENDIZAGEM PRÁTICO, já que não estabelece a obrigatoriedade de conteúdo.

A forma da continuidade do OA é determinada pelos dois padrões do grupo *Seqüência das Questões* e a ordem entre conteúdo e avaliação é estabelecida pelos padrões do grupo *Relação Entre e Conteúdo e Avaliação*.

Os padrões do conjunto deste tópico não são excludentes entre si, portanto OA que adotam APRESENTAÇÃO DAS CONSEQUÊNCIAS podem adotar os outros dois padrões do grupo.

Finalmente, entre os padrões do grupo em questão (*Comportamento Diante das Respostas*) o relacionamento se dá livremente, podendo um OA adotar um ou mais padrões do grupo.

### 6.3. Padrão AUSÊNCIA DE GABARITO

#### Contexto

Ocorre, muitas vezes, dos alunos compreenderem bem um problema dado e desenvolverem uma solução que julgam ser correta. Entretanto, quando vão conferir sua resposta ou quando o professor vai corrigí-la, percebem que ela não está “certa”, apesar de eles não concordarem.

#### Problema

Como desenvolver avaliações em objetos de aprendizagem em que os alunos desenvolvam suas soluções livremente, sem se prenderem a respostas pré-determinadas?

### **Forças**

Certos conteúdos específicos são apropriados para deixar o aluno solucionar por si só os problemas, sem basear-se em respostas estabelecidas. Há questões em que se pode seguir por vários caminhos e nem todos eles podem ser determinados objetivamente.

A avaliação pelo professor da solução encontrada pelo aluno pode ser difícil de ser feita, pois alguns problemas podem ter um teor muito subjetivo a ponto de dificultar o julgamento de sua coerência.

### **Solução**

Desenvolva avaliações cujas respostas não são fechadas, não se tem um “gabarito” pré-estabelecido. O aluno tem mais liberdade para elaborar sua solução, mesmo que ela não seja a inicialmente pensada pelo professor.

### **Racional**

Temas que consideram faixas de valores e combinações entre propriedades, além de temas subjetivos que solicitem a opinião do aluno são passíveis a adotar este padrão, já que não é possível estabelecer uma resposta única.

### **Contexto Resultante**

OA que usam este padrão favorecem uma maior interação do aluno com os colegas e com o professor para compararem suas respostas e discutirem as diferenças em suas soluções.

### **Usos Conhecidos**

- a. Tive uma recaída! [16]: É um OA que adota esse padrão em um pequeno grau, mas é válido para a análise. Na última tela do OA, o usuário deve fornecer suas conclusões a respeito das reações dos antibióticos em alguns tipos de infecção bacteriana. Ele não tem acesso à resposta final. O próprio OA sugere que ele consulte os colegas para compararem suas respostas e discutirem a respeito de suas considerações.

**Atividade "Tive uma recaída!"**

**A solução estará a seu critério.**

Agora, com o resultado dos antibiogramas, classifique as bactérias quanto a sua sensibilidade em relação aos antibióticos testados. Repare que existem apenas 3 categorias de classificação, mas muitas reações diferentes. **A solução estará a seu critério.**



		:: Antibióticos A C I P E N T					
		Resistente	Sensível	Muito Sensível	Resistente	Sensível	Muito Sensível
Infectado (infecção hospitalar)	E. coli	X	X				
				X	X		
				X			X

Pneumônico (pneumonia)

		:: Antibióticos A C I P E N T					
		Resistente	Sensível	Muito Sensível	Resistente	Sensível	Muito Sensível
Infectado (infecção hospitalar)	S. aureus	X	X		X	X	X
				X			

Veja se seus colegas chegaram nas mesmas alternativas. Ao final da classificação, observe qual doença é resistente a um número maior de antibióticos.

Veja se seus colegas chegaram nas mesmas alternativas.

**Figura 18 - Ausência de gabarito no OA "Tive uma recaída". Os próprios alunos discutirão entre si suas respostas.**

- b. Mendel não sabia disso [13]: O aluno deve “editar” seu filme da maneira que desejar: pode fazer em seqüência linear, em *feedback*, em blocos, longo, curto, entre outras formas, utilizando os recursos disponíveis de legenda (funcionando como uma narração) e de tempo de exibição de cada imagem. Ao professor cabe apenas avaliar a coerência das idéias e dos acontecimentos, biologicamente falando.
- c. Colocando as coisas no lugar [17]: A idéia principal é que o aluno faça conexões com elementos ligados à genética e monte essas relações em um quadro, utilizando imagens e conectores. Analogamente ao caso anterior, este também deixa para o aluno a missão de criar o diagrama, relacionando as imagens disponíveis entre si, explicitando, assim, a maneira como compreendeu o conteúdo visto. Como são muitas possibilidades, o próprio aluno não tem como ter certeza de que seu trabalho está correto. Então, é recomendado que o professor observe a montagem feita por ele e avalie se está coerente. Também neste caso, não há “gabarito”, pois há várias possibilidades de respostas.

### Atividade "É hora de colocar as coisas no lugar!"

Além das ervilhas de Mendel, sempre ouvimos, nas aulas de Biologia, falar também de genes, cromossomos e DNA. Mas como organizar todos esses conceitos? Como eles estão relacionados? Colocar as coisas no lugar agora pode ajudar.

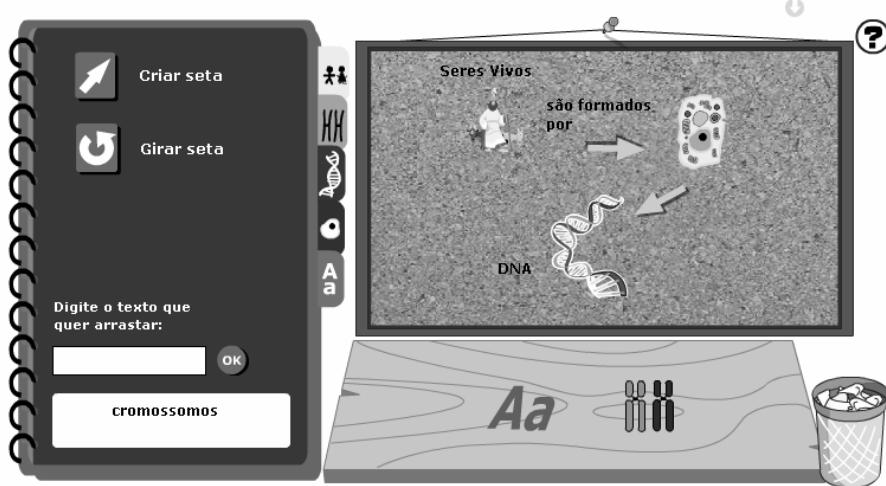


Figura 19 – AUSÊNCIA DE GABARITO no OA "Colocando as coisas no lugar". Os diagramas são montados conforme a criatividade do aluno.

### Padrões Relacionados

Padrões que utilizam o AUSÊNCIA DE GABARITO relacionam-se mais diretamente com o padrão AVALIAÇÃO SUPERVISIONADA, já que não apresenta uma única resposta possível e provavelmente o aluno precisará do auxílio do professor para tomar conhecimento de seu desempenho na atividade. Entretanto, este fato não impede que o OA seja construído baseando-se no AUTO-AVALIAÇÃO.

Quanto ao segundo grupo *Propósito do Objeto de Aprendizagem*, o padrão em questão relaciona-se livremente com ambos os padrões do grupo, enfatizando a explicação do conteúdo, por meio do padrão OBJETO DE APRENDIZAGEM TEÓRICO, ou destacando a avaliação em si, por meio do segundo padrão do grupo. Para este grupo não é possível a existência concomitante de ambos os padrões em um mesmo OA.

A respeito do grupo *Seqüência das Questões*, pelo fato de o padrão AUSÊNCIA DE GABARITO pressupor uma certa liberdade do aluno em estabelecer a solução dos problemas, há a idéia de que o padrão SEQÜÊNCIA NÃO-LINEAR é o mais indicado. Todavia, o segundo padrão do grupo pode ser adotado livremente, estabelecendo a seqüência de atividades a serem realizadas pelo aluno. Permanece a observação de que apenas um dos dois padrões deve ser adotado por vez, já que eles são mutuamente exclusivos.

A ordem entre conteúdo e avaliação é estabelecida pelos padrões do grupo *Relação Entre e Conteúdo e Avaliação*. Pode ser escolhido, entretanto, apenas um dos três padrões para ser adotado em cada OA.

A maneira como os exercícios e avaliações são lançados é determinada pela adoção de um ou mais padrões do grupo *Recursos Utilizados*.

Por fim, entre os padrões do grupo em questão (*Comportamento Diante das Respostas*), OA que adotam o AUSÊNCIA DE GABARITO, podem igualmente utilizar os demais padrões TENTATIVA E ERRO (6.1) e APRESENTAÇÃO DAS CONSEQUÊNCIAS (6.2).

## 7. Referências

- [1] Monteiro, Ingrid. et al. *Linguagem de Padrões para Avaliação de Conhecimento em Objetos de Aprendizagem – Parte I.* In: 6<sup>a</sup> Conferência Latino-Americana em Linguagens de Padrões para Programação, 2007, Porto de Galinhas – PE. Disponível em: [http://sugarloafplop.dsc.upc.br/AnaisSugar2007\\_WEB.pdf](http://sugarloafplop.dsc.upc.br/AnaisSugar2007_WEB.pdf). Acesso em: abr. 2008.
- [2] Laboratório Didático Virtual – USP. *A química dentro de um bolo.* Disponível em: [http://www.labvirt.fe.usp.br/simulacoes/quimica/sim\\_qui\\_bolo.htm](http://www.labvirt.fe.usp.br/simulacoes/quimica/sim_qui_bolo.htm). Acesso em abr. 2008.
- [3] Laboratório Didático Virtual – USP. *Show Atômico.* Disponível em: [http://www.labvirt.fe.usp.br/simulacoes/quimica/sim\\_qui\\_showatomico.htm](http://www.labvirt.fe.usp.br/simulacoes/quimica/sim_qui_showatomico.htm). Acesso em abr. 2008.
- [4] Laboratório Didático Virtual – USP. *O salto dos recordes.* Disponível em: [http://www.labvirt.fe.usp.br/simulacoes/fisica/sim\\_fis\\_saltorecorde.htm](http://www.labvirt.fe.usp.br/simulacoes/fisica/sim_fis_saltorecorde.htm). Acesso em abr. 2008.
- [5] Racha a Cuca. *Calculadora quebrada.* Acesso em abr. 2008. Disponível em: <http://rachacuca.com.br/calculadora-quebrada/>.
- [6] Laboratório Didático Virtual – USP. *A química das cores nos fogos de artifício.* Acesso em abr. 2008. Disponível em: [http://www.labvirtq.fe.usp.br/simulacoes/quimica/sim\\_qui\\_fogos.htm](http://www.labvirtq.fe.usp.br/simulacoes/quimica/sim_qui_fogos.htm).
- [7] Rede Interativa Virtual de Educação - Rived. *Propriedades das emissões radioativas – cargas.* Acesso em abr. 2008. Disponível em: <http://rived.proinfo.mec.gov.br/atividades/quimica/radioatividade/atividade1/atividade1.htm>.
- [8] Rede Interativa Virtual de Educação - Rived. *Química dos Alimentos.* Acesso em abr. 2008. Disponível em: [http://rived.proinfo.mec.gov.br/modulos/quimica/alimentos/qui4\\_ativ5a.htm](http://rived.proinfo.mec.gov.br/modulos/quimica/alimentos/qui4_ativ5a.htm)
- [9] Laboratório Didático Virtual – USP. *Reação do amadurecimento da banana.* Acesso em abr. 2008. Disponível em: [http://www.labvirtq.fe.usp.br/simulacoes/quimica/sim\\_qui\\_banana.htm](http://www.labvirtq.fe.usp.br/simulacoes/quimica/sim_qui_banana.htm).
- [10] Laboratório Didático Virtual – USP. *Até as últimas consequências.* Disponível em: [http://www.labvirt.fe.usp.br/simulacoes/fisica/sim\\_mec\\_consequencias.htm](http://www.labvirt.fe.usp.br/simulacoes/fisica/sim_mec_consequencias.htm) Acesso em abr. 2008.
- [11] Rede Interativa Virtual de Educação - Rived. *Teodolito.* Acesso em abr. 2008. Disponível em: <http://www.rived.mec.gov.br/sistema/upload/zip/23.zip>
- [12] Laboratório Didático Virtual – USP. *Ácido no dia a dia.* Disponível em: [http://www.labvirtq.fe.usp.br/simulacoes/quimica/sim\\_qui\\_acidonodiaadia.htm](http://www.labvirtq.fe.usp.br/simulacoes/quimica/sim_qui_acidonodiaadia.htm). Acesso em abr. 2008.

- [13] Rede Interativa Virtual de Educação - Rived. *Mendel não sabia disso*. Acesso em abr. 2008. Disponível em: <http://rived.proinfo.mec.gov.br/atividades/biologia/genetica/atividade5/atividade5.htm>
- [14] Laboratório Didático Virtual – USP. *O Resgate*. Acesso em abr. 2008. Disponível em: [http://www.labvirt.fe.usp.br/simulacoes/fisica/sim\\_cinematica\\_resgate.htm](http://www.labvirt.fe.usp.br/simulacoes/fisica/sim_cinematica_resgate.htm).
- [15] Rede Interativa Virtual de Educação - Rived. *No tempo certo*. Disponível em: <http://rived.proinfo.mec.gov.br/atividades/quimica/farmaciaemcasa/atividade2/atividade2.htm> Acesso em abr. 2008.
- [16] Rede Interativa Virtual de Educação - Rived. *Tive uma recaída!* Disponível em: <http://rived.proinfo.mec.gov.br/atividades/biologia/microorganismos/atividade3/atividade3.htm>. Acesso em abr. 2008.
- [17] Rede Interativa Virtual de Educação - Rived. *Colocando as coisas no lugar*. Acesso em abr. 2008. Disponível em: <http://rived.proinfo.mec.gov.br/atividades/biologia/genetica/atividade4/atividade4.htm>.

## Evento interno anônimo

Renato Lenz Costalima, Gustavo Augusto Lima de Campos, Jerffeson Teixeira  
de Souza

Laboratório de Computação Natural e Inteligente (LACONI)

Grupo de Padrões de Software da UECE (GPS.UECE)

Universidade Estadual do Ceará (UECE)

91.501-970, Fortaleza/CE, Brasil

renatolenz@uece.br, gustavo@larcos.uece.br, jeff@larcos.uece.br

Resumo. O tratamento de evento em Java é feito através de *Listeners*, objetos responsáveis por observar o acontecimento de eventos e tratá-los com o comportamento definido. Mais de uma abordagem pode ser utilizada para criar esses *Listeners*, entretanto, a maioria dessas abordagens ignora conceitos básicos da orientação a objetos, como a coesão. Para resolver esse problema, este trabalho tem a intenção de descrever boas práticas de programação na forma de um padrão de software, **Evento interno anônimo**.

*Abstract. In Java, event handling is made through event listeners, objects responsible to observe the happening of an event and handle it with the defined behavior. More than one approach can be used to create those event listeners, however, some of these approaches ignore basic concepts of object-oriented programming, such as cohesion. To solve this recurrent problem, this paper describes good programming practices in the form of the software pattern **Anonymous inner event**.*

### 1. Introdução

Na programação orientada a objetos, existem objetos capazes de disparar eventos, em geral, para informar algum acontecimento. Esses eventos podem ser “escutados” por objetos, chamados *Listeners*, para que uma determinada ação seja executada.

A tarefa de escolher, criar e delegar *Listeners* chama-se Tratamento de eventos. Várias abordagens conseguiriam realizá-la, porém, existe uma abordagem que se sobressai às outras, sendo considerada por muitos a melhor solução: **tratar eventos com classe interna anônima**.

Após análise de soluções encontradas nas aplicações existentes e desenvolvimento de aplicações próprias, percebeu-se que classe interna anônima é realmente a solução mais utilizada para tratar eventos, terminando de caracterizá-la como um **padrão de software**.

Voltado para programadores, este trabalho descreve o idioma **Evento interno anônimo** com o intuito de difundir conhecimento e experiência de forma clara e objetiva, para que melhorem suas técnicas, incorporando novas ferramentas e boas práticas de programação. O restante da introdução

apresentará conceitos básicos de orientação a objetos e o formato de padrões de software utilizado. Na seção 2, será apresentado o padrão.

### 1.1. Orientação a objetos: Objetos, classes e classes internas

A programação orientada a objetos (SUN) se baseia no mundo real e tem como peça-chave Objetos. Objetos de software, conceitualmente similares aos objetos do mundo real, compartilham duas características: estado, representado por atributos; e comportamentos, representados por métodos. Em um software, assim como no mundo real, existem objetos do mesmo tipo. Para descrever tipos de objetos são utilizadas Classes. Através delas, é possível construir objetos, chamados instâncias da classe.

O conceito de encapsulamento de dados (*data encapsulation*) é um dos princípios fundamentais da programação orientada a objetos e consiste em esconder o estado interno do objeto e prover formas de interação com o mesmo através de seus métodos. Manter a coesão em um sistema significa colocar na classe somente os atributos e métodos necessários para descrever o estado e os comportamentos do objeto, respectivamente (DEITEL, 2003).

Entre as vantagens da orientação a objetos estão:

- Modularidade: o código de um objeto pode ser escrito e mantido independentemente do código de outros objetos. Uma vez criado, o objeto pode ser facilmente utilizado no sistema.
- Encapsulamento: as interações com os objetos são feitas através de seus métodos. Assim, é possível esconder os detalhes de sua implementação.
- Reuso de código: é possível utilizar em um sistema objetos criados em outros sistemas, possivelmente, por outros autores. Dessa forma, ao implementar um sistema, é possível delegar a tarefa de implementar objetos específicos para especialistas e utilizar os objetos criados na lógica geral do sistema.
- Fácil troca e teste: ao verificar problemas em um objeto, é possível simplesmente removê-lo e colocar outro em seu lugar sem precisar alterar o sistema inteiro, da forma como seria feito no mundo real. Além disso, cada objeto pode ser testado separadamente, aumentando a confiabilidade do sistema como um todo.

Dentro de uma classe, é possível definir outra classe, chamada classe interna (DEITEL, 2003) (MUGHAL, 2001) (SINTES) (SUN). Classes internas são membros da classe externa e, portanto, têm acesso a todos os seus atributos e métodos, mesmo que sejam privados. Entre as principais vantagens de usar classes internas estão:

- Aumentar a coesão, pois separa a funcionalidade secundária da classe externa, mas sem perder a relação de dependência com a classe externa.
- Agrupar logicamente as classes: se uma classe só é usada dentro de outra, possivelmente ela deve estar dentro de forma a estar sempre junta.
- Aumentar encapsulamento: a classe interna diz respeito à implementação interna da classe, portanto deve estar escondida.

- Aumento na legibilidade e manutenabilidade do código: o código passa a estar mais perto de onde ele será usado.

Existem quatro tipos de classes internas. O tipo mais utilizado é chamado classe interna anônima, que é uma classe interna sem nome, útil em casos onde a classe só é usada uma única vez.

### 1.2. Descrição do formato

Para descrever o padrão será utilizado o formato de padrão de software organizado nas seguintes seções:

- Nome: o nome do padrão
- Motivação: conta uma história onde surge a necessidade do padrão descrito
- Contexto: mostra uma situação onde poderia ocorrer o problema
- Problema: o problema a ser resolvido pelo padrão
- Forças: as forças que influenciam na escolha da solução para o padrão
- Solução: a solução para o problema abordado
- Racional: mostra porque a solução é uma boa solução, levando em consideração as forças do problema
- Padrões relacionados: descreve a relação de outros padrões conhecidos com o padrão descrito
- Usos conhecidos: mostra situações onde foi observado o uso do padrão descrito

Seguindo as recomendações de (MESZARO, 1996) as seções Contexto, Problema e Solução são suficientes para dar um entendimento do padrão.

## **2. Evento interno anônimo**

### **Motivação**

Novato, um programador iniciante, recebeu a tarefa de desenvolver uma aplicação com interface gráfica. No centro da tela, um botão deve mostrar uma mensagem de alerta toda vez que alguém clicar nele. Esse botão pode disparar eventos - por exemplo, quando for clicado. É possível alocar objetos para escutar esses eventos e decidir o que fazer.

Nesse momento, Novato, ignorando os conceitos de coesão, pensou em aproveitar a própria janela principal para escutar os eventos do botão. A janela principal deve se ocupar de exibir todos os elementos na tela. Ficar atento aos cliques nos botões e decidir o que fazer, foge do seu escopo. Sem alguém para guiar, criou um método para tratar o evento disparado, mostrando a mensagem de alerta. Outro botão é adicionado. Ora, já existe um método escutando os eventos da aplicação, só é preciso adicionar um tratamento específico para o novo botão dentro do método. Novos objetos chegam e novos tratamentos específicos para cada objeto são adicionados ao método.

```

public class SomeGUI extends JFrame implements ActionListener {
    protected JButton b1;
    protected JButton b2;
    //...
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == b1) {
            // faz algo
        }
        else if (e.getSource() == b2) {
            // ... e assim para cada objeto...
        }
    }
}

```

Uma série de “if/else if” funcionaria perfeitamente, porém, começa a ficar longa, ocasionando em perda de desempenho, manutenibilidade e legibilidade. Para saber o que acontecerá ao ser disparado um determinado evento é preciso procurar na longa série de “if/else if”.

Pedi-se a Novato que trouxesse outra solução. Assim, ele criou uma classe para cada botão e alocou uma instância sua para escutá-lo. A manutenibilidade melhorou, pois não é mais necessário recompilar e redistribuir a classe principal, apenas um pequeno módulo será alterado. O código, embora melhor organizado, ainda é difícil de entender, o comportamento do botão está muito longe de onde o botão é criado. Além disso, esses eventos só dizem respeito à aplicação e deveriam estar escondidos do resto do mundo. Portanto, seus detalhes deveriam estar encapsulados.

Novato, percebendo uma complexidade um pouco maior que a imaginada, pensa: “De quais outras formas eu poderia fazer isso? Ah, se um amigo meu já tivesse feito isso. Ah, se eu tivesse um padrão!”

## Contexto

Na programação orientada a objetos, existem objetos capazes de disparar eventos, em geral, para informar algum acontecimento. Esses eventos podem ser “escutados” por objetos, chamados *Listeners*, para que uma determinada ação seja executada.

A tarefa de escolher, criar e delegar *Listeners* chama-se Tratamento de eventos.

## Problema

Como implementar um tratamento de eventos fácil de ler/manter e coeso em Java?

## Forças

- Centralizar o tratamento de eventos em um único objeto pode causar perda de desempenho.

- Coesão é a capacidade de uma classe em cumprir seu papel. Assim, uma classe deve possuir todos os métodos e variáveis pertinentes ao problema da classe, nem mais, nem menos.
- Classes devem prover o mínimo de acoplamento possível.
- Classes devem permitir o encapsulamento de dados, ou seja, tornar um objeto em uma "caixa-preta". A implementação interna da classe deve estar escondida, bastando para o usuário conhecer sua interface.
- Desempenho, algoritmos devem ser o mais rápido possível.
- Manutenibilidade, o código deve ser fácil de manter, ou seja, de receber alterações sem grandes custos.
- Legibilidade, para trabalhar em grupo, é necessário que outros programadores entendam o código. Assim como a manutenibilidade, a legibilidade é fortemente influenciada pela estrutura de classes e solução adotadas.

## Solução

Os objetos que disparam eventos precisam ter *Listeners* associados a eles. Para cada objeto, crie uma classe interna anônima com o código a ser executado quando o evento ocorrer. Associe a classe interna anônima criada ao objeto a ser observado.

```

1 public class Classe {
2     protected void criarInterface() {
3         ObjetoDisparaEventos objeto1 = new ObjetoDisparaEventos();
4         objeto1.addListener(new Listener() {
5             public void actionPerformed(Event e) {
6                 // faz algo
7             }
8         });
9     }
10 }
```

Observando o código, percebemos um objeto que pode disparar eventos ser criado na linha 3. Na linha 4, é criada uma classe interna anônima descrevendo o que deve acontecer quando o evento for disparado. Ainda na linha 4, a classe interna anônima é alocada para escutar os eventos do objeto.

## Conseqüências

Logo depois de adicionar cada objeto da aplicação, é definida e alocada uma classe interna anônima descrevendo seu comportamento. Assim, o código permite identificar facilmente o que cada objeto faz.

A tarefa secundária de escutar os eventos foi delegada a classes específicas facilmente alteráveis. Essa coesão, além de melhorar a organização do código, melhorou seu desempenho, uma vez que a ação a ser executada não precisa mais ser procurada em uma série de "if/else if".

Apesar de ter um conceito um pouco confuso para programadores iniciantes, classes internas anônimas facilitam na legibilidade e

manutenabilidade do código, uma vez que o código está organizado de forma coesa e mais perto de onde ocorrem os eventos.

### **Padrões Relacionados**

O padrão *Observer*, assim como este padrão, descreve uma boa solução relacionada a tratamento de eventos. O *Observer* descreve como implementar um tratamento de eventos distribuído. Para isso, são criados o que o padrão chama de observadores que descrevem o que acontece quando o evento é disparado. Esses observadores poderiam ser classes internas anônimas, como descreve **Evento interno anônimo**.

### **Usos conhecidos**

Este padrão é amplamente encontrado em aplicações existentes e em livros e tutoriais sobre o assunto. Para citar alguns, o framework Barracuda MVC (BARRACUDA) descreve a aplicação “Simple Login App” que utiliza classe interna anônima para o tratamento de eventos. O framework gui4j (GUI4J) também utiliza o padrão descrito em sua implementação.

### **Bibliografia**

- ALEXANDER, C. et al. A Pattern Language. New York: Oxford University Press, 1977.
- BARRACUDA. Barracuda MVC. <<http://barracudamvc.org/>> Acessado em: 22/04/2008
- COPLIEN, James O. Software Patterns. New York: SIGS Books, 1996.
- DEITEL, H. M. e P. J. Deitel. Java, como programar - 4ed - Porto Alegre: Bookman, 2003. (cap 1, 8 e 9)
- GAMMA, B., R. Helm, R. Johnson, e J. Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley, 1995.
- GUI4J. gui4j. <<http://www.gui4j.org/>> Acessado em: 22/04/2008
- MESZAROS, Gerard. MetaPatterns: A Pattern Language for Pattern Writing em Pattern Languages of Program Design-3. Addison-Wesley, 1996.
- MUGHAL, Khalid A. e R. W. Rasmussen. Programmer's Guide to Java Certification: A Comprehensive Primer. Addison-Wesley, 2001 (cap 7)
- SINTES Tony. Inner classes. <<http://www.javaworld.com/javaworld/javaqa/2000-03/02-qa-innerclass.html>> JavaWorld.com. Acessado em: 07/07/2007
- SUN. The Java tutorial. <<http://java.sun.com/docs/books/tutorial/java/index.html>> Acessado em: 07/07/2007

# Adaptação Dinâmica Guiada por Contrato

**Jonivan Lisbôa<sup>1</sup>, Orlando Loques<sup>1</sup>**

<sup>1</sup>Instituto de Computação – Universidade Federal Fluminense (UFF)  
Rua Passo da Pátria 156, Bloco E – 24210-240 – Niterói – RJ – Brasil

{jlisboa, loques}@ic.uff.br

**Resumo.** Este artigo apresenta um padrão para realizar adaptações dinâmicas na configuração de uma aplicação construída a partir de componentes independentes, que representam recursos computacionais. Uma adaptação ocorre em resposta a variações observadas no contexto de execução representado pelo conjunto de propriedades dos recursos, e visa adequar os requisitos não-funcionais definidos para a aplicação à situação do ambiente. A descrição do padrão contém os itens básicos previstos na forma canônica de Alexander.

## Nome

Adaptação Dinâmica Guiada por Contrato

## Contexto

A disseminação de plataformas (*middlewares*) de gerenciamento de componentes (por exemplo, CCM[OMG 2008], EJB[Sun 2008], COM+/.NET[Microsoft 2008], dentre outras) tornou possível a construção de aplicações a partir de componentes independentes, que representam em alto nível recursos de hardware e software (processadores, dispositivos móveis, protocolos, softwares aplicativos, etc.). Um conjunto de componentes pode ser configurado para prestar um determinado serviço que, além de atender aos requisitos funcionais da aplicação (a funcionalidade desejada), atende também a aspectos não-funcionais representados por objetivos não-específicos da mesma – por exemplo, propriedades como alta disponibilidade, segurança e tolerância a falhas, e requisitos de qualidade como menor custo e menor tempo de resposta.

## Problema

Com a variedade e heterogeneidade dos componentes disponíveis, é possível obter diversas configurações que atendam os requisitos funcionais da aplicação. Porém, cada uma das configurações pode atender de forma diferente os requisitos não-funcionais desejados, de acordo com a variabilidade das propriedades dos recursos ativos. Como garantir que uma aplicação seja configurada com o conjunto de componentes que seja mais adequado às condições apresentadas pelo ambiente de execução, atendendo aos requisitos não-funcionais desejados?

## Cenário

Uma aplicação deseja obter a cotação da bolsa de valores, disponibilizada por diversos provedores de informação. Para otimizar o funcionamento da aplicação, deseja-se que a aplicação esteja conectada sempre ao provedor que ofereça o menor tempo de resposta, dentre todos os disponíveis.

## Forças

- Utilizar um modelo para descrever em alto nível as configurações disponíveis;
- Definir como cada configuração atende os requisitos não-funcionais da aplicação;
- Efetivar adaptações de maneira transparente aos usuários da aplicação;
- Não influenciar tanto no desempenho da aplicação;

## Solução

Utilizar uma plataforma adicional para coordenar (ou orquestrar) o funcionamento de uma plataforma de gerenciamento de componentes, de modo que seja possível configurar um determinado conjunto de componentes de acordo com a situação observada no contexto de execução. A orquestração é baseada em um contrato que define os requisitos a serem atendidos pela aplicação, através da descrição de possíveis configurações e contextos em que estas poderão ser implantadas. Um esquema de monitoração de propriedades do ambiente de execução fornece as informações necessárias para a avaliação da situação do contexto, orientando a seleção de uma configuração.

## Estrutura e Participantes

A solução apresentada possui elementos que podem ser divididos em duas categorias:

- Elementos principais:
  - Contrato – define as configurações disponíveis e como cada uma atende os requisitos não-funcionais da aplicação, através da especificação dos valores das propriedades do ambiente que devem acontecer para a sua ativação;
  - Gerenciador de Configuração – toma decisões sobre qual configuração deve ser ativada, de acordo com o definido no contrato;
  - Configurador – elemento responsável por efetivar a configuração dos componentes, agindo sobre a plataforma de configuração;
- Elementos acessórios:
  - Contexto – contém informações sobre as propriedades monitoradas no ambiente de execução, coletadas por sensores junto a recursos e componentes da aplicação;
  - Sensores – monitoram propriedades de interesse de recursos e componentes, para formar a informação de contexto;

- o Registro de Componentes – diretório que contém referências a componentes disponíveis para configuração;

A Figura 1 ilustra os elementos da solução apresentada, e como eles estão estruturados entre si.

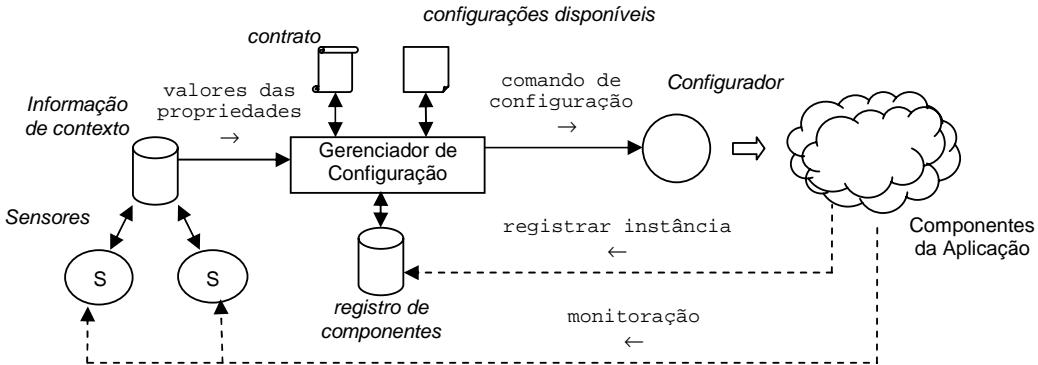


Figura 1. Estrutura da solução apresentada

### Mecanismo (*Rationale*)

A Figura 1 apresenta ainda dicas sobre o funcionamento do mecanismo de gerenciamento dinâmico de configuração. A plataforma de coordenação funciona como gerenciador de configuração, o qual admite-se ser robusto e confiável. Ela é a responsável pela implantação do contrato, que descreve: (a) as possíveis configurações para a aplicação; (b) as propriedades não-específicas do ambiente que validam cada configuração descrita; (c) a seqüência de atividades necessárias para a ativação de cada configuração. Os mecanismos de monitoração de propriedades, compostos por sensores que avaliam periodicamente as propriedades dos recursos do ambiente, alimentam bases de dados de contexto, fornecendo a entrada necessária para o gerenciador confiável analisar a situação do ambiente e confrontá-la com as opções descritas no contrato. Assim, o gerenciador seleciona a configuração mais adequada ao contexto e fornece a descrição da seqüência de atividades de ativação como entrada para o *middleware* de componentes, que atua como configurador, sendo o responsável pela efetivação da configuração selecionada – ou seja, ele procede com a ativação dos componentes e ligações adequadas entre eles.

Os componentes que são inicializados de forma independente da aplicação – por exemplo, componentes de domínio público, como os provedores de informação citados no exemplo – devem ser previamente adicionados ao sistema de registro de componentes, que funciona como um diretório para consulta a referências (nomes de componentes, URLs, etc.) que serão utilizadas na configuração da aplicação.

O processo de avaliação do contrato e seleção de configuração é periódico e pode funcionar enquanto a aplicação estiver em execução sob o contrato, fazendo com que a configuração da aplicação seja adaptada constantemente para tentar atender os requisitos não-funcionais definidos para a mesma. Como contrapartida no benefício de se obter sempre a configuração mais adequada, o processo de adaptação pode acarretar uma perda de desempenho na execução da aplicação. Assim, o funcionamento dos *middlewares* envolvidos deve ser o menos intrusivo possível.

## Contexto Resultante

O funcionamento da plataforma de gerenciamento de componentes é orientado pelo contrato submetido à plataforma de coordenação. Esta plataforma é capaz de selecionar uma configuração dentre as várias descritas, confrontando suas condições de implantação com a situação do contexto de execução. Assim, ela fornece uma entrada para a plataforma de gerenciamento de componentes, que pode ativar adequadamente a configuração selecionada.

O desempenho da aplicação pode ser afetado por alguns fatores, como a ação dos sensores de monitoração e o tempo decorrente entre o processamento da informação de contexto, a comparação com a informação do contrato e a efetivação da configuração. Tais mecanismos devem ser implementados de modo a serem minimamente intrusivos.

## Exemplo

Como exemplo, a Figura 2 ilustra a aplicação do padrão na aplicação de cotação da Bolsa de Valores citada anteriormente. Para o exemplo, serão considerados dois provedores de informação distintos (P1.com e P2.com), cujas URLs estão registradas em um serviço de diretório.

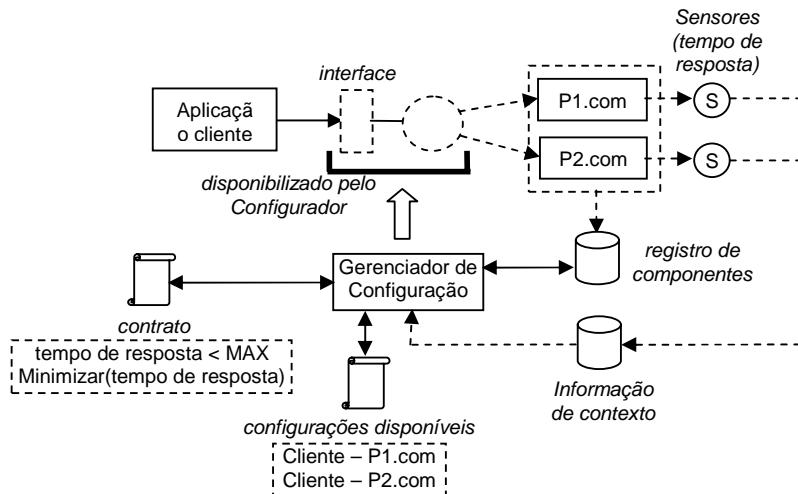


Figura 2. Aplicação do padrão no exemplo citado

O tempo de resposta dos provedores de informação é monitorado pelos sensores, compondo a informação de contexto. A informação será confrontada com o contrato determinado para a aplicação, e a configuração adequada para a situação observada será selecionada. Para tal, o gerenciador de configuração atuará para que o configurador – no caso, alguma plataforma de gerenciamento de componentes (CORBA, EJB, etc.) – disponibilize a interface do provedor selecionado para a aplicação cliente. Com o contrato apresentado, é definido um tempo de resposta máximo tolerável, e procura-se escolher o provedor que oferece o valor mínimo para essa propriedade.

## Usos Conhecidos

Aplicações ubíquas e pervasivas possuem como característica principal a possibilidade de adaptar sua configuração aos requisitos do usuário e às condições do ambiente. Essa classe de aplicações baseia-se na composição de recursos distribuídos para a realização de uma ou mais funcionalidades (serviços). A diferença básica é que as primeiras levam mais em conta a mobilidade de seus usuários para selecionar o conjunto de recursos mais adequados à realização da funcionalidade desejada.

Diversos grupos de pesquisa vêm desenvolvendo *middlewares* de nova geração para coordenar o funcionamento de plataformas de gerenciamento de componentes capazes de oferecer serviços básicos, como controle de ciclo de vida (criação, remoção, ligação de componentes), segurança, reflexão (possibilidade de avaliar propriedades), etc., de modo a fornecer um suporte para o desenvolvimento de aplicações adaptativas com base na monitoração de propriedades de recursos e definição de requisitos e atividades de adaptação através de um contrato. Podem ser citados como exemplos desses *middlewares*: QuO[Loyall et al. 1998], Rainbow[Garlan et al. 2004], Plastik[Batista et al. 2005] e CR-RIO[Cardoso et al. 2006]. O primeiro utiliza o modelo de componentes do CORBA (CCM) e suas linguagens de definição de interfaces (IDL) para descrever componentes e requisitos, enquanto que os outros utilizam modelos próprios de descrição com as ADLs ACME (no caso do Rainbow e Plastik) e CBabel (no caso de CR-RIO).

A Tabela 1 apresenta, de forma resumida, como os elementos da solução apresentada ocorrem nas abordagens citadas, de modo a identificar a recorrência da solução e, com isso, formalizá-la como um padrão. Maiores detalhes sobre as abordagens foram omitidos, por fugirem do escopo deste trabalho. Porém, podem ser encontrados nas respectivas referências citadas

Tabela 1. Ocorrência dos elementos da solução apresentada nas abordagens citadas

Abordagens Elementos	QuO	Rainbow	Plastik	CR-RIO
Contrato	<i>IDE Contract</i>	regras ( <i>Rules</i> )	predicados invariantes ( <i>predicates</i> )	<i>QoS Contract</i>
Contexto	conjunto de objetos <i>SysCond</i>	conjunto de <i>Gauges</i>	conjunto de propriedades	informação dos <i>Resource Agents</i>
Sensores	<i>SysCond</i>	<i>Probes + Gauges</i>	medidores de propriedades	<i>Resource Agents</i>
Gerenciador de Configuração	<i>Property Manager</i>	<i>Constraint Evaluator + Adaptation Engine</i>	<i>Architectural Configurator</i>	<i>Configuration Manager</i>
Configurador	<i>CORBA Delegate</i>	<i>Adaptation Executor</i>	<i>Runtime Configurator</i>	<i>Configurator</i>
Registro de Componentes	conjunto de objetos registrados no ORB	<i>Resource Discovery</i>	conjunto de componentes registrados	conjunto de objetos registrados na plataforma

Os modelos de componentes e contratos descrevem em alto nível a estrutura dos componentes, propriedades que devem ser monitoradas e ações de configuração que devem ser realizadas quando determinadas condições são invalidadas. Quanto a isso, existem algumas diferenças entre as abordagens citadas: CR-RIO preocupa-se com as condições a serem atendidas para a implantação de uma configuração, possuindo claramente uma visão de arquitetura da aplicação; as outras abordagens citadas preocupam-se somente com ações de reconfiguração – normalmente substituição de componentes e/ou ligações entre eles, para adequar a configuração a alguma alteração no contexto de execução. Na abordagem CR-RIO, são definidos mecanismos para transição entre configurações, constituindo-se assim um contrato de adaptação.

### **Padrões Relacionados**

De maneira geral, o padrão de Adaptação Dinâmica Guiada por Contrato está fortemente relacionado ao padrão Reflection[Buschmann et al. 1996], que descreve um mecanismo para alteração dinâmica da estrutura e comportamento de sistemas de software. Em Reflection, uma parte chamada meta-nível (*meta-level*) representa a informação sobre as propriedades do sistema e o que é necessário para realizar as adaptações. Uma outra parte chamada nível-base (*base-level*) representa os mecanismos para efetivar a adaptação. No padrão proposto, o meta-nível consiste no sistema de informação de contexto e no contrato de gerenciamento de configuração, e o nível base é composto pelo gerenciador de configuração e pelo elemento configurador. Como uma extensão do padrão Reflection, o padrão apresentado detalha os mecanismos envolvidos na adaptação dinâmica. A idéia de meta-nível e nível-base é utilizada em [Kon et al. 2002], trabalho que descreve a utilização de mecanismos de reflexão estrutural em dois tipos de sistema baseados em CORBA. Neste trabalho, é feita uma previsão de disseminação do uso e desenvolvimento de *middlewares* de nova geração – ditos reflexivos – para facilitar o desenvolvimento de aplicações baseadas em componentes independentes, com características de ubiqüidade.

O padrão Service Configurator[Jain e Schmidt 1997] define os mecanismos necessários para a configuração de um serviço, envolvendo ativação de componentes e ligações entre eles para o atendimento de um determinado requisito funcional de uma aplicação. O padrão Architecture Configurator[Lisboa et al. 2002] pode ser entendido como uma extensão de Service Configurator, pois engloba mecanismos para se ativar um conjunto de componentes, que representa uma determinada arquitetura. O elemento Configurador dos padrões citados relaciona-se com o configurador de componentes deste padrão: é o responsável pela ativação dos componentes e da realização das ligações entre eles para a configuração da aplicação.

O contrato para gerenciamento de configuração relaciona-se com o padrão QoS Contract[Loyall et al. 2002], que define diferentes possibilidades de configuração, cada qual apresentando um determinado nível de satisfação de requisitos, ou qualidade do serviço prestado. No contrato, além das possíveis configurações, deve ser definido também como deve acontecer a transição entre elas. A monitoração de propriedades de recursos está relacionada ao padrão Memento[Gamma et al. 1995], utilizado para encapsular o estado de um objeto – no caso, o estado de interesse é o conjunto de propriedades de um determinado recurso do ambiente de execução. Assim, o objeto encapsulado pode ser adicionado ao sistema de informação de contexto.

## Agradecimentos

Os autores agradecem ao Prof. Cidcley Teixeira de Souza, por suas valiosas sugestões que contribuíram para aprimorar o trabalho durante o processo de *shepherding*.

## Referências

- Batista, T., Joolia, A. e Coulson, G. (2005). "Managing Dynamic Reconfiguration in Component-Based Systems". *Lecture Notes in Computer Science, Vol. 3527/2005*, Springer.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerland, P., e Stal, M. (1996). Pattern-oriented Software Architecture. A System of Patterns. John Wiley & Sons.
- Cardoso, L., Sztajnberg, A. e Loques, O. (2006). "Self-adaptive Applications Using ADL Contracts". Em *SelfMan 2006 – 2<sup>nd</sup>. IEEE International Workshop on Self-Managed Networks, Systems and Services*. Dublin (Irlanda), junho.
- Gamma, E., Helm, R., Johnson, R., e Vlissides, J. (1995). Design patterns: elements of reusable object-oriented software. Addison-Wesley.
- Garlan, D., Cheng, S., Huang, A., Schmerl, B. e Steenkiste, P. (2004) "Rainbow: Architecture Based Self-Adaptation with Reusable Infrastructure". *IEEE Computer*, 18(10): 46-54.
- Jain, P. e Schmidt, D. C. (1997) "Service Configurator: A Pattern for Dynamic Configuration of Services," Em *The 3<sup>rd</sup>. Conference on Object-Oriented Technologies and Systems – USENIX*, Anaheim (EUA), junho.
- Lisbôa, J., Carvalho, S. e Loques, O. (2002) "Um Design Pattern Para Configuração de Arquiteturas de Software". Em *The 2<sup>nd</sup>. Latin America Conference on Programming Languages of Patterns*, Itaipava, agosto.
- Kon, F., Costa, F., Blair, G. e Campbell, R. H. (2002). "The Case for Reflective Middleware". *Communications of the ACM* 45(6):33-38.
- Loyall, J. P., Schantz, R. E., Zinky, J. A. e Bakken, D. E. (1998). "Specifying and Measuring Quality of Service in Distributed Object Systems". Em *The 1st IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*. Kyoto (Japão), abril.
- Loyall, J. P., Rubel, P., Schantz, R., Atighetchi, M. e Zinky, J. (2002) "Emerging Patterns in Adaptive, Distributed, Real-Time, Embedded Middleware". Em *OOPSLA 2002 Worksop – Patterns in Distributed Real-Time and Embedded Systems*, Seattle(EUA), novembro.
- Microsoft Corporation (acessado em abril de 2008). *Microsoft's Developer News (MSDN) Library*. <http://msdn.microsoft.com/library/default.asp>
- OMG – Object Management Group (acessado em abril de 2008). *Catalog of OMG Specifications*. [http://www.omg.org/technology/documents/spec\\_catalog.htm](http://www.omg.org/technology/documents/spec_catalog.htm)
- Sun Microsystems (acessado em abril de 2008). *Developer Services: Information about Java technology*. <http://developer.java.sun.com>

# **Uma Linguagem de Padrões para Estimativa de Projeto de Software nas Micro e Pequenas Empresas**

**Tarciane de Castro Andrade, Viviane Santos, Jerffeson Teixeira de Souza**

Universidade Estadual do Ceará (UECE)

Grupo de Padrões de Software da UECE – GPS.UECE

Av. Paranjana, 1700, Campus do Itaperi

60.740-903, Fortaleza – CE

[tarcianeandrade@gmail.com](mailto:tarcianeandrade@gmail.com), {viviane.almeida, jeff}@larces.uece.br

**Resumo.** Na fase de planejamento do projeto, o gerente de projeto é encarregado por uma das atividades de maior responsabilidade – realizar estimativas quantitativas do software em relação a tamanho, esforço, custo e prazo como forma de controlar, analisar, e melhorar o processo de desenvolvimento de software, além de auxiliar nas tomadas de decisões organizacionais. Existem, atualmente, diversas técnicas para medir o software e obter tais estimativas. Entretanto, cada uma das técnicas possui peculiaridades que as diferenciam uma das outras, dificultando, para as micro e pequenas empresas a melhor escolha. Nesse sentido, este artigo descreve uma linguagem de padrões para o processo de obtenção de estimativa de software com a intenção de fornecer diretrizes fundamentais para o planejamento do projeto.

**Palavras-chave:** padrões, planejamento do projeto, métricas, estimativa de tamanho, esforço, prazo e custo do software, micro e pequenas empresas.

## **1 Introdução**

A realização das estimativas de software é considerada uma das primeiras atividades da fase de planejamento do projeto e é parte essencial da melhoria do processo de software. Estimativas eficientes permitem a verificação da viabilidade do projeto, a elaboração de propostas técnicas e comerciais, a confecção de planos e cronogramas detalhados e o acompanhamento efetivo do projeto [Monteiro, 2005].

O gerente de projeto, então, confronta-se com um dilema logo no início do projeto: produzir estimativas quantitativas, como mensurar custo, tempo e esforço de desenvolvimento do projeto. Para isto, ele deve conhecer a capacidade de sua equipe e os recursos com os quais pode contar para executar as atividades. Desta forma, adequando-se ao custo disponível e à qualidade desejada, o gerente poderá estabelecer prioridades para a realização dessas atividades.

Ao elaborar uma estimativa para o desenvolvimento de um projeto de software, é desejável que haja um conhecimento sobre técnicas de estimativas e uma visão global do escopo do projeto a ser gerenciado, o que capacita o gerente a quantificar, administrar e planejar mais efetivamente o projeto a ser produzido. Planejar o projeto baseando-se no *feeling* ou em experiências anteriores gera, na maioria das vezes, estouro de prazos e elevado custo de desenvolvimento.

Existem atualmente, inúmeras técnicas para estimativas de software [Albrecht, 1979][Abran et al, 2000][Karber, 1993][FPA, 1998][Boehm et al, 2000][Symons, 1991], todas elas na busca constante de realizar estimativas mais próximas do valor real do software.

As empresas de software de forma geral buscam estimativas exatas que retratem a eficiência do desenvolvimento e minimizem os fracassos do projeto. As grandes empresas (segundo [Mct, 2005], empresa com acima de 100 empregados), em particular, têm facilidade na implantação de técnicas que oferecem maior precisão, em virtude, principalmente, da disponibilidade de recursos financeiros e possibilidade de investimento em tais técnicas. Para essas empresas, o foco principal é minimizar a margem de erro das estimativas. As técnicas robustas e precisas necessitam de maior esforço para contabilização das medidas, onde na maioria das vezes a medição é feita por fase do projeto, para posterior contagem do todo.

O contexto das Micro e Pequenas Empresas de Software - MPEs (segundo [Mct, 2005], empresas com até 10 e 50 empregados, respectivamente), em contrapartida, possui problemas agravantes e típicos de empresas deste porte, tais como a falta de recurso, a informalidade excessiva dos processos, imaturidade nas metodologias, crescimento por demanda, resistência à inovação e emprego de novas técnicas, entre outros [Roullier, 2001]. Diante destas dificuldades, as MPEs necessitam realizar estimativas de software de forma rápida e simples e que reproduzam significadamente os valores totais de tamanho, esforço, prazo e custo do desenvolvimento do software [Vazquez, 2003].

Neste sentido, este artigo está voltado para gerentes e líderes de projetos que necessitam medir quantitativamente o software e apresenta uma linguagem de padrões para estimativa de software, que tem como objetivo guiar as MPEs através de um processo simplificado de boas práticas de como estimar o software, desde do tamanho até o custo, através da unificação das semelhanças entre as técnicas de estimativas de software largamente utilizadas em projetos acadêmicos e empresariais.

Na Seção 2 do artigo será apresentada a Linguagem de Padrões para Estimativa de Software, Running Example (exemplo que será mencionado para demonstrar a solução dos padrões), bem como os Padrões. Na Seção 3 falaremos das considerações finais do artigo, e na última Seção é apresentado uma Lista de Abreveaturas de consulta rápida para as siglas utilizadas no decorrer do artigo.

## 2 Linguagem de Padrões para Estimativa de Projeto de Software

O objetivo da linguagem de padrões descrita é auxiliar o gerente de projeto a estimar o software na fase de planejamento do projeto. Como pode-se observar na Linguagem de Padrões ilustrada na Figura 1, todos os padrões da linguagem apoiam à fase de planejamento do projeto desde da elaboração de estimativas até a sua aprovação pelo cliente, consolidação dos dados reais e posterior apoio de tais dados nas tomadas de decisões da empresa. Para diferenciá-los das demais seções deste artigo, vamos sempre referenciar os padrões em negrito, itálico e sublinhado. Na Tabela 1 pode-se encontrar um resumo de todos os padrões da linguagem, com problema e solução descritos de forma geral.

Vale lembrar que a linguagem aqui apresentada não possui o intuito de estimar o software com a menor margem de erro possível, maior precisão, trabalho este voltado para as diversas técnicas de estimativa de software existentes e que lutam para minimizar esta margem. Este trabalho está voltado para oferecer aos gerentes de projetos das MPEs um guia padrão, simples e preciso o suficiente para o processo de obtenção e calibragem das estimativas de software, como forma de atender à realidade da empresa e auxiliá-los juntamente com o cliente nas tomadas de decisões.

O formato dos padrões apresentados, a linguagem, bem como a descrição geral do problema e solução abordado por cada um deles, estão descritos abaixo:

- **Nome:** nome do padrão;
- **Contexto:** situação em que o padrão dever ser aplicado;
- **Problema:** problema que o padrão resolve;
- **Forças:** aspectos que influenciam na escolha da solução do padrão;
- **Solução:** apresenta a solução para o problema, no contexto definido;
- **Variantes<sup>1</sup>:** variações da solução para o problema, no contexto definido;
- **Exemplo<sup>2</sup>:** exemplo da aplicação do padrão;
- **Contexto Resultante:** cenário posterior à aplicação do padrão;
- **Racional:** mostra porque a solução resolve o problema, e como as forças foram priorizadas;
- **Usos Conhecidos:** descreve situações existentes onde o padrão é utilizado;
- **Padrões Relacionados:** identificam e relacionam outros padrões que são importantes para o padrão descrito;

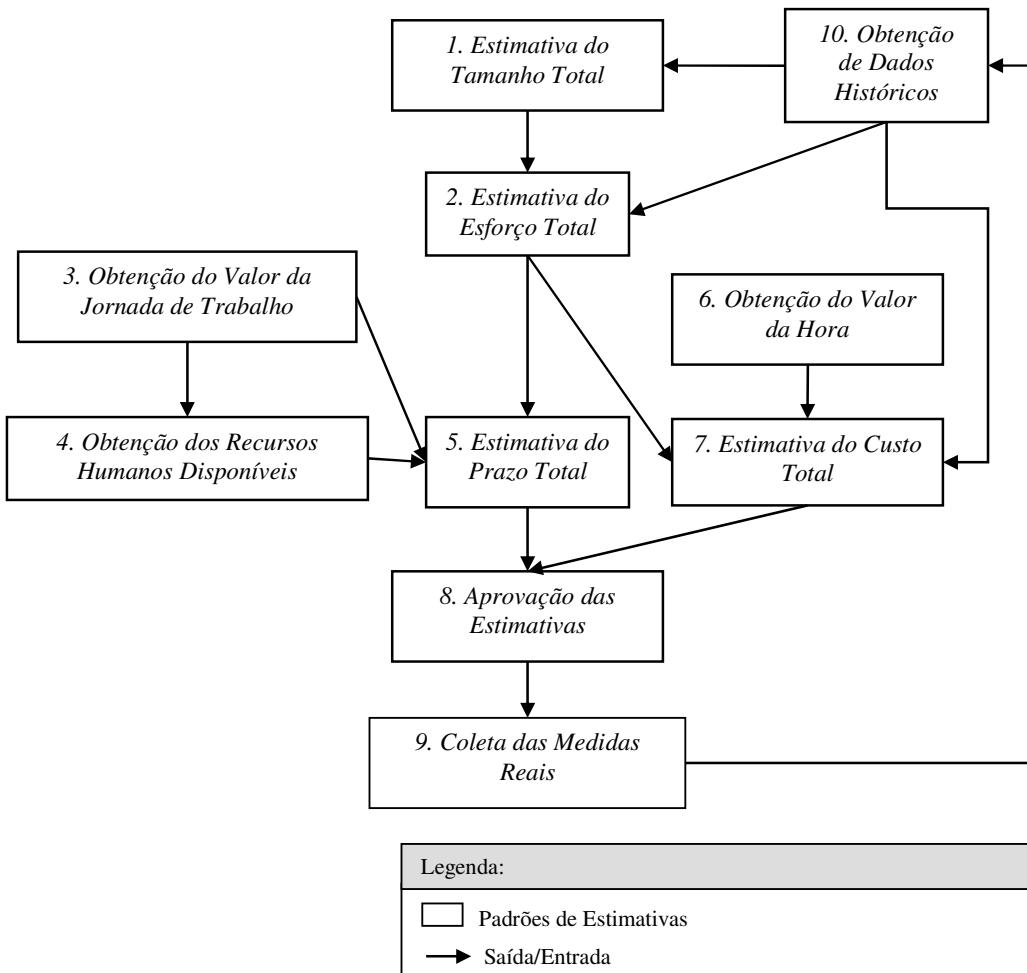


Figura 1. Linguagem de Padrões para Estimativa de Software

<sup>1</sup> Em alguns padrões, por não existir variações da solução, esta seção não foi apresentada.

<sup>2</sup> Em alguns padrões, por não se encaixar no Running Example, esta seção não foi referenciada.

Tabela 1. Resumo dos Padrões

Nome do Padrão	Problema	Solução
<u>1. Estimativa do Tamanho Total</u>	Como estimar o tamanho do projeto?	Divida o sistema de acordo com suas funcionalidades; Para cada funcionalidade classifique-a de acordo com sua complexidade: baixa, média ou alta; Dê pesos e calcule o tamanho total; Obtenha uma medida de tamanho, chamada de <i>ETT</i> .
<u>2. Estimativa do Esforço Total</u>	Como estimar o esforço necessário para execução do projeto?	Obtenha a Produtividade da Equipe ( <i>PROD</i> ); Obtenha a <u>Estimativa do Tamanho Total</u> , o tamanho do software ( <i>ETT</i> ); Estime o esforço total com a fórmula: $ESF = PROD \times ETT$ .
<u>3. Obtenção de Valor da Jornada de Trabalho</u>	Como obter valor da jornada de trabalho?	Verifique junto ao departamento de recursos humanos a jornada de trabalho, em horas, dos profissionais.
<u>4. Obtenção de Recursos Humanos Disponíveis</u>	Como obter a quantidade de recursos disponíveis?	Faça um levantamento de todos os profissionais da empresa para saber a alocação de cada um. Verifique a quantidade de recursos disponíveis e contabilize-os.
<u>5. Estimativa do Prazo Total</u>	Como estimar o prazo total de duração do projeto?	Obtenha a <u>Estimativa do Esforço Total</u> ( <i>ESF</i> ); <u>Obtenção do Valor da Jornada de Trabalho</u> ( <i>VlrJTRAB</i> ) e <u>Obtenção de Recursos Disponíveis</u> ( <i>QtdRD</i> ). Estime o prazo total com a fórmula: $EPRZ = \frac{ESF}{(QtdRD \times VlrJTRAB)}$
<u>6. Obtenção de Valor da Hora</u>	Como obter valor da hora de trabalho?	Estime com a fórmula, onde <i>Salário</i> é o salário em reais por papel e <i>VlrJTRAB</i> é o valor da jornada de trabalho obtido a partir de <u>Obtenção do Valor da Jornada de Trabalho</u> : $VlrHORA(Papel) = \frac{Salário}{VlrJTRAB \times 30}$
<u>7. Estimativa do Custo Total</u>	Como estimar o valor do custo total do projeto?	Estime com a fórmula, onde <i>ESF</i> é a <u>Estimativa do Esforço Total</u> , <i>P</i> é a porcentagem de esforço necessário para cada papel e <i>VlrHORA</i> é calculado a partir da <u>Obtenção do Valor da Hora</u> : $CUSTO = \sum ESF \times P(Papel) \times VlrHORA(Papel)$
<u>8. Aprovação das Estimativas</u>	Como aprovar as estimativas realizadas?	Marque reunião com o cliente. Apresente os valores de custo e prazo do software. Apresente ao cliente como os valores foram calculados.
<u>9. Coleta das Medidas Reais</u>	Como coletar as medidas reais no decorrer do andamento do projeto?	Definir o que será medido. Construa documentos a serem preenchidos pelos membros do projeto. Defina marcos para realização de tais coletas. Armazene-as.
<u>10. Obtenção de Dados Históricos</u>	Como obter dados de medições anteriores?	Realize consultas na base de dados para obtenção de dados históricos;

## 2.1 Running Example

Considere uma pequena empresa hipotética de software, aqui nomeada de XYZ, com cerca de 15 profissionais entre gerentes, analistas, projetistas e desenvolvedores. A empresa XYZ tem como cliente a Universidade Estadual do Ceará que necessita de um simples Sistema de Matrícula de Estudantes.

De forma geral, o Sistema possui as seguintes funcionalidades:

- Cadastro de Professores e Estudantes;
- Matrícula dos Estudantes nos cursos ofertados;

O professor pode logar no sistema, escolher os cursos que quer ministrar e submeter as notas dos estudantes no decorrer da disciplina. O estudante pode logar no sistema através de usuário e senha, realizar matrícula nos cursos disponíveis e visualizar o relatório com as suas notas. O Administrador pode logar no sistema, além de manter o cadastro dos professores e dos estudantes. O Sistema de Matrícula se integra com o sistema de graduação da universidade que mantém o catálogo atualizado dos cursos disponíveis no semestre.

A Figura 2 ilustra o diagrama de casos de usos do Sistema de Matrícula, com as responsabilidade de cada um no sistema.

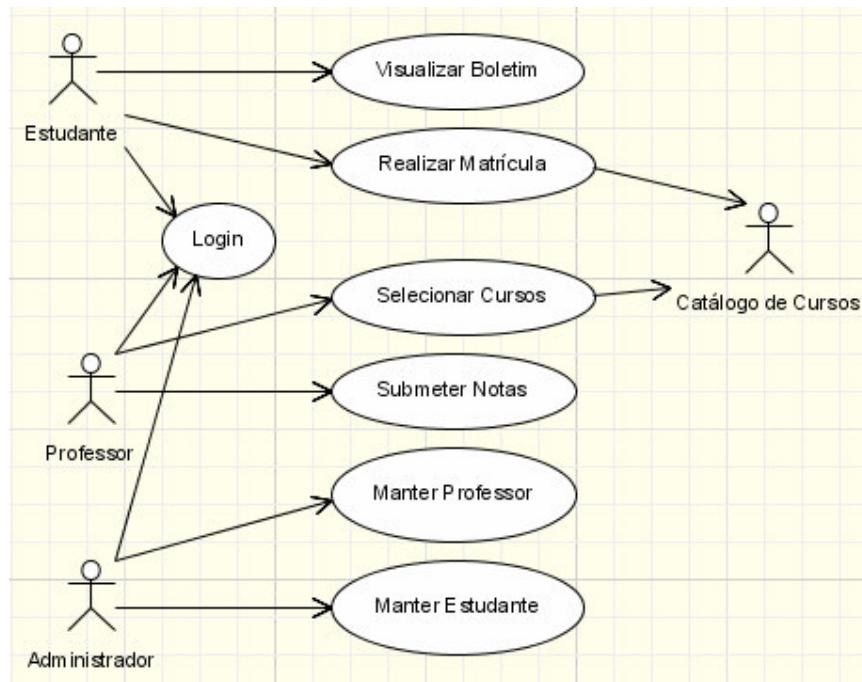


Figura 2. Casos de Uso do Sistema de Matrícula

## 2.2 Os Padrões

### 1. Estimativa do Tamanho Total

#### **Contexto:**

As MPEs de desenvolvimento de software, na figura dos gerentes de projeto, têm dificuldade em estimar o tamanho dos projetos na fase de planejamento, com escopo geral definido, devido ao grande número de técnicas existentes e da complexidade exigida por elas.

#### **Problema:**

Como elaborar estimativas do tamanho total do projeto de forma significativa, simples e a um custo satisfatório?

#### **Forças:**

- 1) O gerente de projeto, baseado em experiências anteriores próprias, pode simplesmente, estimar o tamanho do software sem realizar nenhum cálculo, porém tal estimativa pode acarretar um valor totalmente fora da realidade do projeto, podendo causar prejuízo financeiro à empresa;
- 2) O gerente de projeto pode optar por estimar o tamanho do software de acordo com a quantidade de linhas de código por funcionalidade, porém tal técnica depende da linguagem de programação utilizada, tornando-a instável;
- 3) Estimativas de alto nível em fases iniciais do projeto podem ser imprecisas, porém estimativas detalhadas demais podem despender tempo e custo às MPEs;
- 4) Estimativas baseadas na análise do projeto quanto às suas funcionalidades e quantificação delas em relação à complexidade podem ser simples e a um custo satisfatório às MPEs;
- 5) Dados históricos de estimativas podem ajudar na precisão;
- 6) Especialistas externos e o uso de componentes prontos do mercado facilitam a elaboração de estimativas de forma rápida e precisa, porém as MPEs não possuem recursos financeiros para investir em tal propósito.

#### **Solução:**

- 1) Determinação dos Fatores Não-Ajustáveis (FNA).

Os Fatores Não-Ajustáveis apresentam as funcionalidades exigidas pelo cliente para execução do projeto. Logo, para obter tais fatores:

- a. Separe o projeto por funcionalidades<sup>3</sup> e determine a quantidade de transações de acordo com os seguintes grupos:
  - i. Transações de entrada de dados por parte do usuário com persistência na base de dados (TED);
  - ii. Transações de saída de dados da base para fornecer informações ao usuário (TSD);
  - iii. Transações de consulta, onde o usuário entra com os dados, e solicita uma resposta da base de dados (TCD);

---

<sup>3</sup> No caso de projeto com Notação UML, deve-se construir o diagrama de caso de uso, e a partir dele, identificar as transações;

- iv. Transações externas, mantidas por outra aplicação (TEX);
- v. Transações de persistência de dados (TPD). Contabilize para cada TEX uma TPD;
- b. Para cada um dos grupos acima, classifique-o de acordo com sua complexidade, vide Tabela 2. Atribua complexidade baixa, média ou alta.
- c. Atribua peso menor às que possuem complexidade baixa, peso intermediário às que possuem complexidade média e maior peso às que possuem alta, de acordo com Tabela abaixo.
- d. Verifique os dados históricos em Obtención de Dados Históricos para auxiliar na classificação das transações;

Tabela 2. Classificação das Transações

Grupos de Transações	Complexidade	Quantidade	Peso
TED	Baixa	Q <sub>1</sub>	P <sub>1</sub>
	Média	Q <sub>2</sub>	P <sub>2</sub>
	Alta	Q <sub>3</sub>	P <sub>3</sub>
TSD	Baixa	Q <sub>4</sub>	P <sub>4</sub>
	Média	Q <sub>5</sub>	P <sub>5</sub>
	Alta	Q <sub>6</sub>	P <sub>6</sub>
TCD	Baixa	Q <sub>7</sub>	P <sub>7</sub>
	Média	Q <sub>8</sub>	P <sub>8</sub>
	Alta	Q <sub>9</sub>	P <sub>9</sub>
TEX	Baixa	Q <sub>10</sub>	P <sub>10</sub>
	Média	Q <sub>11</sub>	P <sub>11</sub>
	Alta	Q <sub>12</sub>	P <sub>12</sub>
TPD	Baixa	Q <sub>13</sub>	P <sub>13</sub>
	Média	Q <sub>14</sub>	P <sub>14</sub>
	Alta	Q <sub>15</sub>	P <sub>15</sub>

- e. Calcule os Fatores Não-Ajustáveis (FNA)

$$FNA = \sum_{i=1}^9 Q_i \times P_i + \sum_{j=10}^{15} Q_j \times P_j , \text{ onde } P_i \leq P_j$$

## 2) Cálculo dos Fatores Técnicos Não-Funcionais (FTNF).

Os Fatores Técnicos Não-Funcionais são as funcionalidades ou requisitos não funcionais do sistema. Logo, para obter tais fatores:

- a. Classifique o projeto quanto às características abaixo, de acordo com Tabela 3. Cada característica deve ser avaliada com notas: 0 – não atende, 3 – atende parcialmente, 5 – atende completamente.

Tabela 3. Características Não – Funcionais

Característica	Nota
Processamento Distribuído	N <sub>1</sub>
Desempenho	N <sub>2</sub>
Eficiência do Usuário	N <sub>3</sub>
Processamento Complexo	N <sub>4</sub>
Reusabilidade	N <sub>5</sub>
Facilidade de Instalação	N <sub>6</sub>
Usabilidade	N <sub>7</sub>
Portabilidade	N <sub>8</sub>
Facilidade de Manutenção	N <sub>9</sub>
Concorrência	N <sub>10</sub>

$$FTNF = 0.65 + \left( \sum_{i=1}^{10} Ni \times 0.01 \right)$$

### 3) Cálculo da Estimativa do Tamanho Total (ETT)

Logo, calcule o Tamanho Total do Software através da fórmula:

$$ETT = FNA \times FTNF , \text{ onde:}$$

FNA = Fatores Não-Ajustáveis

FTNF = Fatores Técnicos Não Funcionais

#### Variantes

Ao cálculo dos Fatores Técnicos Não-Funcionais – FTNF pode ser acrescido pesos de importância a cada uma das características. Assim sendo, ao somatório do cálculo da fórmula estaria incluso o peso de cada uma delas. Os pesos podem variar de 0.5 a 2, de acordo com o grau de complexidade que cada característica trará ao sistema. Com isso, a fórmula do cálculo do FTNF seria:

$$FTNF = 0.65 + 0.01 \times \left( \sum_{i=1}^{10} Ni \times Pi \right)$$

#### Exemplo:

Considere o Sistema de Matrícula descrito na Seção 2.1.

1) Cálculo dos Fatores Não-Ajustáveis (FNA), de acordo com Tabela 4.

a. Separação das funcionalidades por caso de uso:

i. Manter Professor

Considere três transações simples para cadastrar, alterar e excluir um professor (3 TED), e uma transação de consulta (1 TCD);

ii. Manter Estudante

Considere três transações simples para cadastrar, alterar e excluir um professor (3 TED), e uma transação de consulta (1 TCD);

- iii. Logar no Sistema  
Considere duas consultas para validar usuário e senha (2 TCD);
  - iv. Selecionar Cursos  
Considere três transações para gravar os cursos selecionados: cadastrar, atualizar e excluir (3 TED); Considere uma transação externa para consulta dos cursos disponíveis (1 TEX e consequentemente 1 TPD); Considere duas transações de saídas de dados da base para verificar se os horários dos cursos não são conflitantes com demais cursos do professor (2 TSD); Considere uma consulta aos cursos do professor (1 TCD);
  - v. Submeter Notas  
Considere uma transação de consulta para listar as notas dos estudantes do curso (1 TCD); Considere uma transação externa para listar os cursos ministrados pelo professor (1 TEX, 1 TPD); Considere três transações para: cadastrar as notas, atualizar e excluir (3 TED); Considere uma transação para que o professor selecione o curso (1 TSD);
  - vi. Visualizar Boletim  
Considere uma transação externa para listar os cursos matriculados (1 TEX, 1 TPD); Considere uma transação de consulta para as notas de cada curso (1 TCD);
  - vii. Realizar Matrícula  
Considere uma transação externa para que o estudante liste os cursos ofertados (1 TEX, 1 TPD); Considere uma transação para que o estudante escolham os cursos (1 TSD); Considere duas transações para realização e cancelamento da matrícula (2 TED); Considere uma transação de consulta aos cursos matriculados (1 TCD);
- b. Atribuição de complexidade e pesos:  
Para cada uma das transações acima, classifique-as como simples, média e complexa e contabilize, de acordo com a Tabela 4;

Tabela 4. Pesos para o Sistema de Matrícula

Grupos de Transações	Complexidade	Quantidade	Peso <sup>4</sup>	Valor
TED	Simples	6	3	18
	Média	-	4	0
	Complexa	8	6	48
TSD	Simples	4	4	16
	Média	-	5	0
	Complexa	-	7	0
TCD	Simples	3	3	9
	Média	-	4	0
	Complexa	5	6	30
TEX	Simples	-	5	0
	Média	-	7	0
	Complexa	4	10	40
TPD	Simples	-	7	0
	Média	-	10	0
	Complexa	4	15	60
<b>FNA</b>				<b>221</b>

- 2) Cálculo dos Fatores Técnicos Não – Funcionais (FTNF), de acordo com Tabela 5.

Tabela 5. Valores atribuídos aos Fatores Não-Funcionais

Característica	Nota
Processamento Distribuído	3
Desempenho	5
Eficiência do Usuário	3
Processamento Complexo	3
Reusabilidade	5
Facilidade de Instalação	5
Usabilidade	5
Portabilidade	5
Facilidade de Manutenção	5
Concorrência	3
<b>Total</b>	<b>42</b>

$$FTNF = 0.65 + (42 \times 0.01) = 1.07$$

- 3) Cálculo da Estimativa do Tamanho Total (ETT)

$$ETT = FNA \times FTNF = 221 \times 1.07 = 236.47 \text{ unid. de tamanho. Onde,}$$

FNA = Fatores Não-Ajustáveis  
FTNF = Fatores Técnicos Não Funcionais

---

<sup>4</sup> Pesos referentes e calibrados de acordo com a técnica de Pontos de Função [IFPUG, 1995].

### **Contexto Resultante:**

- Vantagens
  - A empresa poderá confiar mais em suas estimativas ao fechar contratos, ter mais segurança e visibilidade do tamanho de cada projeto. Além de possibilitar a estimativa de outras medidas como esforço, custo e prazo total do projeto;
  - O gerente de projeto de posse do tamanho do software poderá realizar estimativas mais próximas à realidade, minimizando eventual prejuízo financeiro à empresa;
  - As primeiras estimativas tendem a ser mais discrepantes em relação à realidade. À medida que novos projetos surgem, as estimativas tendem a ser mais precisas.
- Desvantagens
  - O nível de detalhamento das funcionalidades no momento do cálculo da estimativa tem que ser satisfatório. Qualquer mudança de escopo, no decorrer do projeto, afetará a estimativa realizada;
  - A empresa precisa ter maturidade ao classificar um caso de uso de acordo com sua complexidade. O que é simples pra uma empresa, pode ser complexo a outra. Nesse caso, os dados históricos e experiência da equipe auxiliam na classificação;

### **Racional:**

Os gerentes de projeto das MPEs necessitam estimar o tamanho do software de forma precisa, rápida e simples. Os principais problemas enfrentados por estas empresas são referentes à diversidade de técnicas existentes, à complexidade oferecida por elas, aliada à reduzida quantidade de recursos humanos e financeiros.

O uso de componentes ou softwares disponíveis no mercado torna o processo de estimar rápido e preciso de acordo com a técnica empregada, no entanto as MPEs não podem realizar tal investimento.

Dados históricos de estimativas podem ajudar na precisão, caso sejam consolidadas e avaliadas previamente, mas por si só não funcionam satisfatoriamente.

Então, estimar o tamanho do software baseado na divisão das funcionalidades em transações e na atribuição de complexidades a cada transação torna a estimativa precisa, rápida e simples uma vez que atende tanto a software estruturados como orientados a objetos, além de torná-los independentes de plataforma e linguagem de desenvolvimento, fazendo o processo apto às MPEs.

### **Usos Conhecidos:**

A técnica de medir o tamanho do software por Análise de Ponto por Função (*Function Point Analysis – FPA* [IFPUG, 1995]) é derivada a partir de métricas diretas, onde o projeto deve ser inteiramente decomposto em funcionalidades de acordo as interfaces com o usuário, entradas, saída e consultas, atribuindo-se pesos a cada uma dessas funcionalidades.

O *MKII Function Point Analysis* [Symons, 1991] é uma variação da *FPA* onde estende a lista dos fatores técnicos não-funcionais.

O *Cosmic Full Function Point - FPP* [Abran, 2000] é outra variação da *FPA* onde pode ser ajustada a projetos orientados a objetos. A *FPP* não leva em consideração para efeito de tamanho os fatores técnicos não-funcionais e nem os de qualidade que medidos separadamente.

O *Use Case Point – UCP* [Kerner, 1993] é uma técnica de estimativa de tamanho para software orientado a objetos, onde foi desenvolvida a técnica de diagramação para o conceito de caso de uso. O *UCP* foi criado baseado na *FPA* onde explora o modelo e a criação de casos de uso, substitui algumas características técnicas proposta pela *FPA*, cria os fatores ambientais, como experiência da equipe, e propõe uma estimativa de produtividade.

O Ponto de Caso de Uso Técnicos – *TUCP* [Monteiro, 2005] é uma extensão do *UCP* como forma de tornar a estimativa mais precisa para casos de uso mais complexos.

### **Padrões Relacionados:**

A partir dos dados estimados do tamanho do software pode-se obter a **Estimativa do Esforço Total** do desenvolvimento. Com o padrão **Obtenção de Dados Históricos** é possível obter informações sobre medições anteriores e auxiliar na classificação das funcionalidades e características do software a ser desenvolvido.

---

## **2. Estimativa do Esforço Total**

### **Contexto:**

Os gerentes de projetos das MPEs de software têm dificuldade em estimar o esforço, em horas de trabalho, necessário para desenvolver o projeto.

### **Problema:**

Como elaborar estimativas de esforço para o desenvolvimento do software?

### **Forças:**

- 1) Para maior precisão das estimativas de esforço, o gerente de projeto pode realizar a estimativa por fase do projeto de acordo com cada funcionalidade, e logo após obter a estimativa de esforço total do software, porém exige um nível de detalhamento maior, despendendo tempo e custo às MPEs;
- 2) Estimativas de esforço podem ser obtidas através de relatos de experiência dos especialistas da empresa, porém tais relatos podem remeter a dados discrepantes e fora da realidade, podendo causar prejuízos financeiros à empresa, risco que as MPEs não podem se submeter;
- 3) Toda MPE, em virtude do seu porte e condições salariais baixas comparadas às grandes empresas, possui elevada rotatividade de pessoal, o que prejudica a elaboração de estimativas de precisas;
- 4) O cliente do projeto pode exigir um valor limite de esforço para a construção do software, induzindo o gerente a elaborar estimativas de esforço próximas a esse valor e fora da realidade da empresa;
- 5) O cálculo de estimativas mais elaboradas ao considerar o tamanho do software, além de outros fatores, como por exemplo, o nível de desenvolvimento exigido, coesão da equipe e maturidade do processo tornam a estimativa mais precisa, porém necessitam de tempo para serem planejadas e calculadas;
- 6) O cálculo de estimativas de esforço baseadas no tamanho total do software e de acordo com as funcionalidades do software torna a estimativa mais precisa. Dados históricos de estimativas podem reforçar a precisão do cálculo do esforço, além de fornecer dados como a Produtividade da Equipe;

**Solução:**

- 1) Obtenha a Produtividade (*PROD*) da equipe através da **Obtenção de Dados Históricos**;
- 2) Obtenha através de **Estimativa do Tamanho Total** (*ETT*) a estimativa do tamanho total do software a ser desenvolvido;
- 3) Calcule a Estimativa de Esforço Total (*ESF*) em horas:

$$ESF(\text{horas}) = PROD \times ETT, \text{ onde,}$$

*PROD* = Produtividade,

*ETT* = Estimativa do Tamanho Total

**Variantes:**

O Esforço Total do software pode ser obtido com maior precisão ao considerar o nível de Experiência da Equipe (*EE*) nas tecnologias e processos envolvidos, conforme abaixo:

- 1) Classifique a equipe quanto sua experiência de acordo com as características abaixo, Tabela 6. Cada característica deve ser avaliada com notas: 0 – não atende, 3 – atende parcialmente, 5 – atende completamente;

Tabela 6. Características da Experiência da Equipe

Característica	Nota
Dificuldade na Linguagem de Programação	$N_1$
Conhecimento do Processo de Desenvolvimento	$N_2$
Capacidade de Análise	$N_3$
Experiência com a Orientação a Objetos	$N_4$

- 2) Estime o Esforço Total com a fórmula abaixo:

$$ESF = PROD \times ETT \times EE, \text{ onde}$$

*PROD* = Produtividade,

*ETT* = Estimativa do Tamanho Total,

$$EE(\text{Experiência da Equipe}) = 1.4 - (0.03 \times \sum_{i=1}^4 N_i)$$

**Exemplo:**

Considere, por exemplo, que a produtividade média da equipe da empresa XYZ é de *10hs/ETT*.

Logo,

$$ESF = 10 \times 212.92 \cong 2364 \text{ horas}$$

Suponha agora, que a Experiência da Equipe seja 11. Logo,

$$FA = 1.4 - (0.03 \times 11) = 1.07$$

$$ESF = 10 \times 236.47 \times 1.07 \cong 2530 \text{ horas}$$

### **Contexto Resultante:**

- Vantagens
  - O gerente do projeto terá o valor do esforço de desenvolvimento do projeto em horas. De posse desse valor, poderá contabilizar outras medidas de estimativas como custo e prazo total do projeto para auxiliar nas decisões estratégicas do projeto;
  - Ao estimar o esforço a partir do tamanho, o gerente de projeto tem um embasamento racional para a quantidade de horas necessárias, o que possibilita o aumento do poder de negociação com cliente.
- Desvantagens
  - O cálculo do esforço depende do cálculo do tamanho do software. Se as funcionalidades não estiverem sido levantadas de forma coerente, então irão impactar no cálculo das demais estimativas, incluindo o esforço.
  - O cálculo do esforço depende da produtividade da equipe. As MPEs tendem a possuir elevada rotatividade de pessoal e como consequência, queda na produtividade.

### **Racional:**

Simplesmente estimar o esforço necessário para o projeto através de relatos de experiência individuais dos membros da empresa, não é tão preciso quanto a obtenção de valores reais baseados em cálculos previamente definidos.

Outra abordagem é a obtenção de estimativas por fase do projeto, para posteriormente estimar o todo. Abordagens destes tipos tendem a possuir maior grau de precisão, porém necessitam de dedicação e tempo para planejamento das fases do projeto, itens que as MPEs não possuem.

Estimar o esforço total para desenvolvimento do software baseado no cálculo prévio do seu tamanho total e a produtividade média da equipe é uma boa solução em virtude da sua simplicidade, porém em projetos maiores convém considerar o nível de experiência da equipe nas tecnologias exigidas.

### **Usos Conhecidos:**

A técnica *Constructive Cost Model - COCOMO II* [Boehm et al, 2000] calcula esforço, prazo, tamanho de equipe e custo necessário para o desenvolvimento do software, desde que se tenha a dimensão do mesmo, ou seja, seu tamanho. Nesta técnica, além dos fatores considerados para o cálculo do esforço são considerados outros fatores de escala.

As técnicas UCP [Kerner, 1993] e TUCP [Monteiro, 2005] calculam o esforço a partir dos dados citados na solução, levando em consideração os fatores ambientais que são calculados a partir de vários itens com atribuições de pesos.

### **Padrões Relacionados:**

O padrão **Estimativa do Tamanho Total** e **Obtenção de Dados Históricos** servem como entrada desse padrão ao calcular o tamanho do software e a produtividade da equipe respectivamente. Com o valor em horas do esforço total do projeto, o gerente, pode obter as demais estimativas de prazo e custo com os padrões: **Estimativa do Prazo Total**, **Estimativa do Custo Total**.

### **3. Obtención do Valor da Jornada de Trabalho**

#### **Contexto:**

No planejamento do projeto, o gerente necessita de todas as informações referentes aos profissionais da área, entre elas, o valor da jornada de trabalho dos empregados para servir como entrada das estimativas.

#### **Problema:**

Como obter o valor da jornada de trabalho dos funcionários da empresa?

#### **Forças:**

- 1) Cada profissional tem horário e jornada de trabalho diferenciada, o que dificulta este tipo de levantamento;
- 2) O gerente pode considerar que todos os funcionários da empresa possuem a mesma jornada de trabalho, porém tal levantamento pode impactar no prazo total do projeto;
- 3) A existência de um setor de recursos humanos facilita ao gerente o levantamento do valor da jornada de trabalho, uma vez que este setor é o responsável por tal atividade;
- 4) A ausência de um setor de recursos pode dificultar por parte do gerente a obtenção da jornada de trabalho.

#### **Solução:**

- 1) Caso exista o departamento de recursos humanos verifique a jornada de trabalho diária, em horas, dos profissionais;
- 2) Se houver divergência na jornada de trabalho dos profissionais, realize uma média para obter a jornada geral.

$$\text{Valor da Jornada de Trabalho} = \text{VlrJTRAB horas/dia}$$

#### **Variantes:**

- 1) Caso não haja o departamento de recursos humanos verifique com todos os profissionais um a um o valor da jornada de trabalho;
- 2) Se houver divergência na jornada de trabalho dos profissionais, realize uma média para obter a jornada geral.

$$\text{Valor da Jornada de Trabalho} = \text{VlrJTRAB horas/dia}$$

#### **Exemplo:**

O gerente de projeto do Sistema de Matrícula verificou junto ao setor de recursos humanos da empresa XYZ que todos os profissionais possuem a mesma jornada de trabalho de 8 horas por dia. Logo,

$$\text{VlrJTRAB} = 8 \text{ horas/dia}$$

#### **Contexto Resultante:**

- Vantagens:
  - O gerente do projeto, de posse do valor da Jornada de Trabalho dos profissionais da empresa, poderá estimar e decidir quanto ao prazo total de conclusão do projeto. Podendo assim, negociar melhor com o cliente.

**Racional:**

O departamento de recursos humanos é responsável por controlar desde a contratação dos empregados, até salário e freqüência. É este departamento que possui relatórios sobre a jornada de trabalho de cada profissional. Nem todas as MPEs possuem tal setor em virtude da quantidade reduzida de profissionais, cabendo então ao gerente realizar o levantamento por si só.

**Usos Conhecidos:**

Em geral todos os gerentes de projetos que necessitam realizar estimativas obtêm o valor da jornada de trabalho dos empregados da empresa.

Uma instituição financeira atualmente em todo o Brasil obtém o valor da jornada de trabalho de todos os profissionais da área de TI como forma de estimar os projetos de software. Tal instituição utiliza da técnica de TUCP [MONTEIRO, 2005] para estimar os softwares.

**Padrões Relacionados:**

Com a informação da jornada de trabalho, o gerente de projeto poderá dar continuidade às demais medições como: Obtenção de Recursos Humanos Disponíveis, Estimativa do Prazo Total, Estimativa do Custo Total.

---

**4. Obtenção dos Recursos Humanos Disponíveis****Contexto:**

No planejamento do projeto, os gerentes de projetos das MPEs precisam ter visibilidade de todos os seus recursos humanos para devida alocação no projeto.

**Problema:**

Como obter a quantidade de recursos humanos disponíveis da empresa para serem alocados no projeto?

**Forças:**

- 1) O gerente de projeto pode se reunir com cada profissional da empresa para saber em quê está alocado e os horários livres, porém os gerentes de projeto das MPEs não possuem tempo suficiente para realizar tal atividade;
- 2) O gerente de projeto pode, via e-mail, solicitar que cada profissional o retorne com o seu horário disponível;
- 3) Nas MPEs um membro do projeto é alocado em várias atividades diferentes ao exercer diversos papéis, ou até mesmo em projetos diferentes, o que dificulta ao gerente de projeto obter a quantidade de recursos humanos disponíveis;
- 4) O gerente de projeto pode verificar junto ao setor de recursos humanos a alocação de cada um nos projetos e horários disponíveis, porém esse setor possui dados macros e muitas vezes desatualizados. Um levantamento da quantidade de recursos disponíveis junto ao profissional possibilita a obtenção de dados mais próximos da realidade, além de permitir ao gerente visualizar a alocação de cada um no projeto.

**Solução:**

- 1) Faça um levantamento, por e-mail, de todos os profissionais da empresa para saber a alocação de cada um;
- 2) Verifique através do padrão ***Obtenção do Valor da Jornada de Trabalho*** a quantidade de horas trabalhadas diariamente;
- 3) Verifique a quantidade de recursos completamente disponíveis e contabilize-os;
- 4) Verifique aqueles recursos que estão parcialmente alocados, e contabilize-os através de horas/dia disponíveis. Realize regra de três simples para calcular a porcentagem de tempo disponível em relação ao todo.
- 5) Some e obtenha a Quantidade de Recursos Disponíveis (QtdRD);
- 6) Realize mapeamento inicial dos recursos disponíveis com respectivos papéis necessários no projeto;

$$\text{Quantidade de Recursos Disponíveis} = \text{QtdRD pessoas}$$

**Exemplo:**

A empresa XYZ possui 15 funcionários, entre gerentes, analistas, projetistas e desenvolvedores. Entre eles, apenas 3 (três) estão completamente disponíveis (um analista de sistemas e dois programadores) para serem alocados ao projeto de Matrícula dos Alunos na Universidade. O gerente de projeto possui apenas 6 horas livre por dia. Suponha ainda, que nenhum dos recursos está disponível para realização de horas-extras e que o valor da jornada de trabalho é de 8horas/dia. Logo, fazendo uma regra de três simples, temos que o gerente de projeto contabiliza apenas 75% do tempo.

Então:

$$\text{QtdRD} = 3 + 0.75 = 3.75 \text{ pessoas}$$

**Contexto Resultante:**

- Vantagens
  - O gerente de projeto, de posse da quantidade de recursos disponíveis e da jornada de trabalho poderá calcular o prazo total de execução do projeto;
  - O gerente de projeto poderá distribuir os recursos disponíveis no projeto de acordo com o perfil de cada um e se possível, oferecer treinamento àqueles que não possuem capacidade devida para exercer tal papel.
- Desvantagens
  - Nas MPEs, devido ao reduzido quadro de empregados, cada um deles exerce mais de um papel, logo, tal levantamento não é trivial. Cabe ao gerente de projeto encontrar a melhor forma de quantificar a disponibilidade de cada um.

**Racional:**

Realizar o levantamento da quantidade de recursos disponíveis via e-mail é uma solução prática e com tempo de resposta satisfatório, o que facilita ao gerente o levantamento dos recursos disponíveis, uma vez que nas MPEs o quadro de funcionários é reduzido.

Ao analisar criteriosamente a distribuição e alocação dos recursos nos projetos atuais da empresa e contabilizá-los, o gerente de projeto poderá planejar melhor a utilização de tais recursos no novo projeto.

### **Usos Conhecidos:**

O *Rational Unified Process – RUP* [Jacobson et al, 1999][Kruchten, 2003], na disciplina Gerência de Projeto, possui o artefato chamado de Plano de Projeto, onde o gerente de projeto tem a missão de verificar os recursos disponíveis da empresa para alocação no projeto.

O *Capability Maturity Model Integrated - CMMI* [CMMI, 2005] na área de processo de Planejamento de Projeto possui uma prática específica chamada de *Plan for Project Resources* onde além do planejamento e obtenção dos recursos humanos disponíveis para o projeto, verifica a disponibilidade de recursos de hardware e software.

O *PMBOK* [PMI, 2004] é voltado para a gerência de projetos, entre as diversas áreas de conhecimento está a área de Gerência do Custo do Projeto que inclui o processo de Planejamento de Recursos para determinar quais recursos (pessoas, equipamentos, materiais) e em que quantidade devem ser usados para executar as atividades do projeto.

O MPS.BR [Softex, 2006], no nível G, possui o processo de Gerência de Projetos, onde entre tantas atividades encontra-se a de planejamento de recursos humanos.

### **Padrões Relacionados:**

Para obter os recursos da empresa e observar a alocação e disponibilidade de cada um, é necessário a Obtenção do Valor da Jornada de Trabalho. A quantidade de recursos disponíveis para o projeto auxilia na Estimativa do Prazo Total.

---

## **5. Estimativa do Prazo Total**

### **Contexto:**

O gerente de projeto, na fase de planejamento de projeto, de posse do esforço em horas total do projeto, da quantidade de recursos disponíveis ao projeto e do valor da jornada de trabalho dos profissionais, necessita quantificar ao cliente o prazo necessário para desenvolver o sistema solicitado, como forma de firmar um acordo e fechar o contrato.

### **Problema:**

Como elaborar estimativas de prazo do desenvolvimento do software na fase inicial do projeto?

### **Forças:**

- 1) O cliente do projeto pode exigir um valor limite de prazo para entrega do produto, induzindo o gerente a elaborar estimativas de prazo próximas a esse valor, fora da realidade da empresa e sem nenhum embasamento;
- 2) Estimativas de prazo podem ser obtidas através de relatos de experiência dos especialistas da empresa, porém tal valor pode ser subestimado ou

- sobreestimado, podendo causar, respectivamente, prejuízos financeiros à empresa ou insatisfação ao cliente;
- 3) O gerente pode obter o prazo total do projeto baseado em dados quantitativos de medição, como o esforço total e a quantidade de recursos disponíveis, o que causa maior precisão à estimativa;
  - 4) O gerente de projeto pode estimar o prazo total através de técnicas elaboradas que levam em consideração vários fatores de ponderação, porém a utilização de tais técnicas despende tempo que os gerentes de MPEs não possuem;
  - 5) As MPEs não possuem segurança ao estimar o prazo de um projeto, porém se este for decomposto e contabilizado em escopos menores para posteriormente estimar o prazo total, este será mais facilmente contabilizado, porém tal decomposição exige tempo que estas empresas não possuem;

#### **Solução:**

- 1) Obtenha através de **Estimativa do Esforço Total** o esforço total, em horas, necessário para o desenvolvimento do projeto;
- 2) Através de **Obtenção dos Recursos Humanos Disponíveis**, obtenha a quantidade de recursos disponíveis para o projeto;
- 3) Através de **Obtenção do Valor da Jornada de Trabalho**, obtenha o valor da jornada de trabalho dos funcionários da empresa;
- 4) Calcule a Estimativa de Prazo Total do Software (**EPRZ**):

$$EPRZ = \frac{ESF}{(QtdRD \times VlrJTRAB)}, \text{ onde:}$$

*ESF* = Esforço Total do Projeto

*QtdRD* = Quantidade de Recursos Humanos Disponíveis

*VlrJTRAB* = Valor da Jornada de Trabalho

#### **Exemplo:**

A empresa XYZ, para desenvolver o projeto se Sistema de Matrícula, necessitará de um prazo de aproximadamente:

$$EPRZ = \frac{2530}{3.75 \times 8} \cong 84 \text{ dias úteis}$$

#### **Contexto Resultante:**

- Vantagens:
  - A empresa obterá o valor do prazo total de desenvolvimento do projeto em dias e a partir de dados objetivos e racionais, podendo assim, negociar mais facilmente com o cliente.
- Desvantagens:
  - O cálculo do prazo depende do cálculo da jornada de trabalho e dos recursos disponíveis. Esses dois últimos são complicados nas MPEs em virtude do reduzido quadro de profissionais e da alocação deles em outros projetos.

**Racional:**

Simplesmente estimar o tempo necessário para o desenvolvimento projeto através de relatos de experiência individuais dos membros da empresa, não é tão preciso quanto a obtenção de valores reais baseados em cálculos previamente definidos.

As obtenções do prazo do projeto através de técnicas mais elaboradas e da decomposição do projeto tornam a estimativa com maior grau de precisão, porém a execução de tais estimativas necessita de tempo, preparo e custo.

Estimar o prazo baseado em dados previamente calculados, como o esforço, quantidade de recursos disponíveis e jornada de trabalho, torna a estimativa precisa o suficiente e simples, possibilitando com que as MPEs sejam objetivas no cálculo das estimativas.

**Usos Conhecidos:**

A técnica do *COCOMO II* [Boehm et al, 2000] obtém o prazo total do projeto a partir do esforço e equipe média necessária para o desenvolvimento.

Já o *UCP* [Karner, 1993] e *TUCP* [Monteiro, 2005] calculam o prazo total do projeto a partir dos valores do esforço, quantidade de recursos e horas de trabalho diário de cada profissional. No caso do *TUCP* as jornadas de trabalho dos profissionais são consideradas informações pré-definidas.

**Padrões Relacionados:**

Os seguintes padrões são necessários para o cálculo do valor do prazo total do projeto: *Estimativa do Esforço Total, Obtenção dos Recursos Humanos Disponíveis, e Obtensão do Valor da Jornada de Trabalho*. O gerente de projeto após ter mensurado o projeto quanto a prazo, deverá aprovar a estimativa e apresentá-la ao cliente através do padrão *Aprovação das Estimativas*.

---

**6. Obtenção do Valor da Hora****Contexto:**

O gerente de projeto, ao planejar o desenvolvimento do sistema, necessita obter informações do valor da hora de trabalho dos funcionários da empresa, como forma de mensurar o custo total de execução do projeto.

**Problema:**

Como obter o valor da hora de trabalho dos funcionários da empresa?

**Forças:**

- 1) O gerente de projeto pode obter o valor da hora de trabalho baseado no mercado e não em valores internos da empresa, podendo posteriormente ter um impacto significativo no custo do software;
- 2) O gerente de projeto pode obter o valor da hora de trabalho de acordo com dados de projetos anteriores, porém tais valores podem estar desatualizados;
- 3) O gerente de projeto pode calcular o valor da hora de acordo com o maior salário ou ainda com a média geral de todos os salários dos funcionários. No entanto, tal forma de calcular pode gerar um valor elevado, com impacto direto no custo do software;

- 4) Os salários dos funcionários variam de acordo com o cargo, consequentemente o valor da hora de trabalho também. O gerente pode calcular o valor da hora de trabalho de acordo com o papel de cada um;
- 5) Os funcionários das MPEs geralmente assumem mais um papel no processo de desenvolvimento, o que pode influenciar na obtenção do valor da hora de trabalho em cada projeto.

**Solução:**

- 1) Para cada papel necessário no processo de desenvolvimento do projeto calcule o valor da hora de trabalho:

$$VlrHORA(Papel) = \frac{\text{Salário}}{VlrJTRAB \times 30}, \text{ onde:}$$

$VlrJTRAB$  = Valor da Jornada de Trabalho

**Exemplo:**

Para o Sistema de Matrícula serão alocados 4 profissionais. Três deles alocados plenamente no projeto e um deles alocado parcialmente (6 horas por dia). Supondo que a alocação no projeto esteja distribuída da seguinte forma: 1 gerente de projeto (6 horas por dia), 1 analista, 2 desenvolvedores. Supondo ainda que estes funcionários possuam os respectivos salários: R\$3.500,00, R\$2.600,00, R\$ 2000,00. Logo,

$$VlrHORA(Gerente) = \frac{3.500,00}{6 \times 30} \cong R\$19.50$$

$$VlrHORA(Analista) = \frac{2.600,00}{8 \times 30} \cong R\$10.90$$

$$VlrHORA(Desenvolvedor) = \frac{2.000,00}{8 \times 30} \cong R\$8.40$$

**Contexto Resultante:**

- Vantagens
  - Nas MPEs, na maioria das vezes, cada profissional exerce mais de um papel, logo o cálculo poderá ser realizado de acordo com a quantidade de horas trabalhadas em cada papel;
  - Com os valores acima, o gerente do projeto poderá calcular o valor gasto, em reais, de cada um dos profissionais para execução do projeto. Podendo negociar os valores com o cliente;
- Desvantagens
  - Para cada cargo ou variação salarial da empresa o gerente de projeto terá que calcular o valor da hora do profissional. Para contornar isso, o gerente poderá considerar a maior variação salarial dentro de determinado cargo, por exemplo.

**Racional:**

Obter o valor da jornada de trabalho de profissionais de informática no mercado é um risco ao projeto. Cada empresa brasileira possui porte e salários diferenciados, nas MPEs esses valores são mais discrepantes, necessitando, portanto, cálculos reais de acordo com sua realidade. Os salários de quaisquer profissionais variam constantemente e o valor da hora de trabalho necessita ser recalculado periodicamente.

A opção de calcular individualmente o valor da hora de trabalho de cada papel do profissional no projeto torna a estimativa mais precisa do que, por exemplo, pegar uma média geral do valor.

### **Usos Conhecidos:**

De acordo com o Artigo 64 da Lei Trabalhista CLT [CLT, 1943] o valor da hora de trabalho de um trabalhador é obtido a partir do salário e da jornada de trabalho dividido por 30 dias.

Uma instituição financeira atualmente boa parte do território nacional utiliza a TUCP [MONTEIRO, 2005] para estimar os softwares. Para estimar o software de Central de Cadastro dos Clientes, por exemplo, tal instituição necessita como entrada da TUCP o valor da hora de trabalho de cada profissional. Os valores são calculados de acordo com a fórmula acima. Demais sistemas desenvolvidos também utilizam de tal técnica de estimativa.

### **Padrões Relacionados:**

O valor da hora de trabalho de cada profissional é dado de entrada para o cálculo da Estimativa do Custo Total.

---

## **7. Estimativa do Custo Total**

### **Contexto:**

O gerente do projeto necessita na fase de planejamento, obter o custo, ou seja, a despesa do projeto para a empresa como forma de negociar com o cliente o valor do desenvolvimento do mesmo.

### **Problema:**

Como elaborar estimativas de custo do desenvolvimento do software?

### **Forças:**

- 1) Estimar o custo de desenvolvimento ao levar em consideração apenas o valor do salário mais elevado da empresa e do esforço total de desenvolvimento do software pode acarretar no superfaturamento do software;
- 2) O gerente de projeto, baseado em experiências anteriores, pode estimar o custo do desenvolvimento do software. No entanto, tal estimativa pode superfaturar ou subfaturar o valor do software;
- 3) O gerente de projeto pode levar em consideração a necessidade financeira da MPE e assim estimar o valor do custo do software, porém tal estimativa pode prejudicar a negociação com o cliente;
- 4) As MPEs não possuem recursos suficientes para despender muito tempo com a utilização de técnicas elaboradas, como a utilização de desvio padrão, para o cálculo do custo do projeto;
- 5) O gerente de projeto pode ponderar dados históricos, Obtenção de Dados Históricos, de projetos anteriores para considerar o esforço necessário por papel do projeto, obter o valor da hora de trabalho de cada profissional e posteriormente obter o custo total de desenvolvimento do software;
- 6) O cliente pode exigir que o valor do custo de desenvolvimento do software não ultrapasse um valor teto, induzindo o gerente a obter estimativas

próximas a esse valor e totalmente diferentes do custo real do projeto para empresa, podendo causar um prejuízo financeiro significativo, risco este que as MPEs não podem arcar;

### Solução:

- 1) Obtenha a **Estimativa do Esforço Total**;
- 2) Obtenha, em porcentagem, o esforço necessário para cada papel no projeto. Para isso, verifique dados históricos anteriores em **Obtenção de Dados Históricos**;
- 3) Obtenha em **Obtenção do Valor da Hora** o valor da hora de trabalho de cada papel do projeto;
- 4) Calcule o custo total do desenvolvimento do projeto, da seguinte forma:

$$CUSTO = \sum ESF \times P(Papel) \times VlrHORA(Papel), \text{ onde:}$$

$ESF$  = Estimativa do Esforço Total

$P$  = Porcentagem de esforço necessário para cada Papel

$VlrHORA$  = Valor da Hora de Trabalho para cada Papel

### Exemplo:

Supondo que as fases do desenvolvimento do Software estejam divididas em: Requisitos, Análise, Projeto, Codificação e Teste. Supondo ainda que 15% do esforço total será gasto com o Gerente de Projeto, 35% do esforço total será gasto nas fases de Requisitos, Análise e Projeto com o Analista de Sistemas e 50% com o desenvolvedor para a codificação e testes do sistema.

No sistema de matrícula o custo total do projeto é:

$$\begin{aligned} CUSTO = & (2530 \times 0.15 \times R\$19.50) + \\ & (2530 \times 0.35 \times R\$10.90) + \\ & (2530 \times 0.5 \times R\$8.40) \cong R\$27.677.00 \end{aligned}$$

### Contexto Resultante:

- Vantagens
  - A empresa terá o valor do custo total de desenvolvimento do projeto baseado em valores objetivos, calculados a partir de dados concretos do custo do software, podendo assim, acrescentar custos adicionais e negociar com o cliente.
- Desvantagens
  - O custo calculado do software pode ser maior que o esperado pelo cliente. Cabe ao gerente efetuar as devidas negociações e assumir os riscos;

### Racional:

Estimar o custo do software baseado em experiências anteriores ou de acordo com a pressão do cliente pode gerar um risco financeiro alto às MPEs que geralmente não possuem recursos financeiros extras para sustentar a empresa. Portanto o ideal é estimar o custo baseado em valores reais do tamanho e esforço do software.

A porcentagem gasta em cada fase do desenvolvimento do software varia de empresa pra empresa. Cada empresa deve guardar estas informações em dados históricos para avaliação dos projetos subsequentes. O resultado do custo total do software depende consideravelmente da porcentagem de contribuição de cada

profissional ao longo do projeto e consequentemente do valor da hora de trabalho de cada um. Estimar o software baseado em tais valores torna a estimativa simples e precisa. Superfaturar um pouco o custo do software é melhor do que a situação contrária, principalmente para as MPEs.

### **Usos Conhecidos:**

As técnicas *UCP* [Kerner, 1993] e sua variação *TUCP* [Monteiro, 2005] e *COCOMO II* [Boehm et al, 2000] utilizam esta forma de cálculo do custo do software por fase de desenvolvimento.

### **Padrões Relacionados:**

A partir da Estimativa do Esforço Total, Obtenção do Valor da Hora e Obtenção de Dados Históricos o gerente estima o tempo gasto por fase do projeto para calcular o custo. Posteriormente, o gerente necessita a Aprovação das Estimativas por parte do cliente.

---

## **8. Aprovação das Estimativas**

### **Contexto:**

O gerente de projeto calculou todas as estimativas do software quanto a prazo e custo do projeto. O gerente necessita da aprovação do cliente.

### **Problema:**

Como convencer o cliente a aprovar as estimativas?

### **Forças:**

- 1) O gerente de projeto pode optar por convencer o cliente quanto às estimativas sem despender tempo, podendo realizar a negociação através de trocas de e-mails ou contato telefônico. No entanto, tal forma de convencimento pode gerar ao cliente “má impressão” e “desleixe” por parte da empresa;
- 2) O gerente de projeto pode tentar convencer o cliente através de reuniões, o quê pode transmitir maior grau de segurança, organização e atenção ao cliente, deixando-o mais confiante;
- 3) O gerente pode tentar convencer o cliente ao explicar minuciosamente como obteve os valores de prazo e custo do software. Assim, pode aumentar a segurança do cliente em relação a essas estimativas, tornando mais fácil a sua aprovação;
- 4) O gerente pode tentar convencer o cliente ao apresentar um protótipo do projeto, com algumas telas de funcionalidades, além de fazer um bom marketing para a venda do projeto, sem se ater muito a explicar como obteve os valores estimados. No entanto, as MPEs não possuem tempo e recursos humanos e financeiros suficientes para investir na criação de protótipos e marketing pessoal nesta fase do desenvolvimento;
- 5) O gerente pode simplesmente apresentar os valores ao cliente, sem entrar nos detalhes de como foram obtidos. No entanto, tal técnica é arriscada, pois o cliente pode não se convencer dos valores de prazo e custo e desistir da negociação;

**Solução:**

- 1) Marque reunião com o cliente;
- 2) Apresente os valores de prazo e custo do desenvolvimento ao cliente;
- 3) Apresente ao cliente as informações de como foram calculados os valores apresentados;
- 4) Transmita segurança ao cliente, que o cálculo dos valores é justo e de acordo com as funcionalidades e complexidades exigidas por ele;
- 5) O cliente aceita os valores e assina o contrato;

**Variantes:**

Na reunião, após a apresentação dos valores por parte do gerente e tentativa de convencer o cliente:

- 1) O cliente não aceita os valores;
- 2) O gerente marca outra reunião com o cliente para apresentação de novos valores;
- 3) O gerente refaz os cálculos na tentativa de atender a expectativa do cliente;
- 4) O gerente apresenta os novos valores ao cliente;
- 5) O cliente aceita os valores e assina o contrato;

O cliente pode ainda não aceitar os valores e cancelar o projeto com a empresa;

**Contexto Resultante:**

- Vantagens
  - Como o cálculo das estimativas foi baseado em valores extraídos a partir das funcionalidades do software, o gerente possui maior embasamento para justificar o custo e prazo do desenvolvimento do software.
  - Caso o cliente imponha resistência quanto ao prazo/custo, o gerente também poderá renegociar tendo como base valores consistentes, sem colocar em risco o lucro da empresa.
- Desvantagens
  - O cliente pode aceitar ou não os termos de prazo e custo de desenvolvimento do software. Caso o cliente aceite, o gerente deve iniciar o desenvolvimento do software. Caso não aceite, o gerente deve analisar em quais aspectos da negociação houve falha.

**Racional:**

O cliente poderá insistir e discordar de algumas informações no intuito de reduzir o custo do software e/ou recebê-lo em menos tempo. Com isso, o Gerente terá que refazer os cálculos de prazo e custo do software, e apresentar novamente ao cliente em outra oportunidade.

Cabe ao gerente de projeto negociar o prazo com o cliente e convencê-lo que o prazo obtido foi baseado em cálculos derivados a partir de tamanho e esforço do software. O cliente deve estar ciente que em uma MPE a quantidade de recursos disponíveis para alocação nos projetos é sempre reduzida, o que pode influenciar no prazo do projeto.

**Usos Conhecidos:**

Uma instituição financeira utiliza o processo de aprovação de estimativas através de reuniões com cliente. O gerente de projeto apresenta ao cliente as estimativas e ocorrem as devidas negociações. No caso dessa instituição, o cliente é a própria

instituição, ou seja, desenvolvimento interno de software. Para todos os sistemas produzidos em tal instituição é utilizado o TUCP [MONTEIRO, 2005] para o cálculo das estimativas e posteriormente é efetuada a aprovação das mesmas.

### **Padrões Relacionados:**

Os padrões de Estimativa do Custo Total, Estimativa do Prazo Total são essenciais para a negociação com o cliente. Uma vez aprovados os valores, cabe ao gerente de projetos definir, durante todo o processo de desenvolvimento, a Coleta de Medidas Reais.

---

### **9. Coleta de Medidas Reais**

#### **Contexto:**

No decorrer do processo de desenvolvimento do software é necessário coletar informações reais sobre o andamento do projeto. O Gerente de Projeto deve definir as medições;

#### **Problema:**

Quais as medições que auxiliam no processo de estimar o software? Como coletar medidas reais do tamanho do software, do esforço, do prazo e do custo?

#### **Forças:**

- 1) Realizar medições em um ambiente sem cultura de medições pode provocar resistências;
- 2) Para definição das medições a serem realizadas é necessário planejamento por parte do gerente de projeto. O planejamento e definição de tais métricas são realizados apenas uma vez no processo de desenvolvimento e atende a todos os projetos da empresa;
- 3) A organização precisa estar disposta a encarar as medições como um investimento em longo prazo, visto que para disponibilizar a realidade da organização com base em métricas, é necessário tempo;
- 4) Para realizar as medições, o gerente de projeto pode obter pessoalmente as informações com os membros do projeto. No entanto, tal coleta por parte do gerente faz com ele deixe de executar atividades próprias do seu papel;
- 5) A compra de uma ferramenta que automatize o processo de coleta e obtenção de medidas reais e que possa ser integrada com as ferramentas de análise e desenvolvimento dos projetos torna o processo rápido e simples. No entanto, as MPEs não possuem recursos financeiros para tal investimento;
- 6) O gerente de projeto pode definir marcos no projeto para medição. Cada membro do projeto é responsável pela coleta das medidas nas datas definidas;

#### **Solução:**

- 1) Defina o quê será medido, por exemplo:
  - a. Para cada funcionalidade do projeto, classifique-a como transação de entrada de dados (TED), de saída de dados (TSD), consulta (TCD), saída de dados externa (TEX) e de persistência de dados (TPD);

- b. Para cada funcionalidade do projeto, classifique-a como baixa, média ou alta e realize a medição;
  - c. Esforço em horas gasto por fase/total do projeto;
  - d. Quantidade de horas e/ou dias total gasto por papel (contabilize também em relação ao todo); Quantidade de horas e/ou dias total gasto por fase (contabilize também em relação ao todo); Quantidade de horas e/ou dias total do projeto;
  - e. Tamanho Total Real do software no final do projeto (verifique mudanças de requisitos);
  - f. Produtividade da Equipe = Esforço Total Real/Tamanho Total Real;
  - g. Custo real do projeto total;
- 2) Defina o plano de métricas, com as informações: o que será medido, como, por quem e quando.
  - 3) Construa documentos a serem preenchidos pelos profissionais aptos do projeto, onde cada documento contém informações sobre os dados que serão medidos;
  - 4) Disponibilize base de dados para guardar as medições;
  - 5) Defina marcos no projeto para coletar as medições;
  - 6) Cada membro do projeto é responsável por preencher documento sobre as suas correspondentes medições;
  - 7) O gerente de projeto de posse das medições deve alimentar base de dados para posterior **Obtenção de Dados Históricos**;

### **Contexto Resultante:**

- Vantagens
  - A empresa terá base de dados com dados históricos reais por projeto, podendo melhorar as estimativas em projetos futuros;
  - A medição ajuda na avaliação objetiva dos dados. Não se pode melhorar o que não é medido;
  - O gerente de posse dos indicadores extraídos das medições pode, por exemplo: avaliar o status do projeto, controlar os riscos, ajustar as atividades e avaliar a capacidade da equipe de controlar a qualidade do produto de trabalho.
- Desvantagens
  - Todo trabalho de medição poderá ser colocado em risco se não for garantido a obtenção de dados confiáveis;

### **Racional:**

Apesar da resistência das pessoas para realizar as medições, a definição de medidas simples não tomam tempo dos membros do projeto. Com o decorrer do tempo, esse tipo de atividade se torna habitual, o essencial é insistir na importância da medição para o futuro da empresa.

### **Usos Conhecidos:**

O *CMMI* [CMMI] na área de processo Medição e Análise utiliza a coleta de dados no decorrer do processo de desenvolvimento.

O *MPS.BR* [Softex, 2006] no processo de Medição possui a atividade de coletar dados e armazenamento dos dados e resultados.

Em [Vazquez, 2003] é utilizado a atividade de Aprovar Estimativas no processo de estimativa de um projeto de software.

**Padrões Relacionados:**

Com a concentração de tais informações em uma base de dados compartilhada pelos gerentes, os mesmos podem consultá-las, **Obtenção de Dados Históricos**, para melhorar a precisão das estimativas em projetos futuros.

---

**10. Obtenção de Dados Históricos****Contexto:**

Na fase de planejamento de projeto, o gerente necessita estimar o tamanho do software, o esforço, o prazo e o custo para o seu desenvolvimento. Os membros dos projetos realizam a coleta de medições de dados reais dos projetos envolvidos e armazenam tais dados. Dados históricos de projetos anteriores devem ser capazes de melhorar a precisão das estimativas.

**Problema:**

Como obter dados históricos com o intuito de melhorar a precisão das estimativas?

**Forças:**

- 1) O gerente de projeto pode obter dados históricos através da sua própria experiência em projetos anteriores, dados estes puramente intuitivos, podendo comprometer as estimativas;
- 2) O gerente de projeto pode obter dados históricos de outras empresas ou do mercado de forma geral, dados estes que não estão de acordo com a realidade da empresa;
- 3) O gerente de projeto pode obter dados históricos a partir de uma base de dados consolidada na empresa que guarde essas informações;

**Solução:**

- 1) Realize consultas na base de dados;
  - a. Ao estimar o tamanho total do software, **Estimativa do Tamanho Total**, verifique em projetos anteriores como as funcionalidades foram classificadas quanto à complexidade e transações;
  - b. Ao estimar o esforço total do software, **Estimativa do Esforço Total**, verifique a média da produtividade da equipe em projetos anteriores;
  - c. Ao estimar o custo total do software, **Estimativa do Custo Total**, verifique a porcentagem de esforço gasto para cada papel do projeto em relação ao todo;

**Contexto Resultante:**

- Vantagens
  - A empresa terá dados consolidados de acordo com seu próprio perfil.
  - Com base em dados de projetos anteriores, o gerente de projeto poderá calcular as estimativas com maior precisão.
- Desvantagens
  - Possibilidade de dúvida interpretação dos dados históricos, prejudicando o cálculo das estimativas. Para tanto, a base dos dados históricos necessita estar consolidada e claramente interpretada.

### Racional:

Simplesmente consultar a base de dados não é suficiente para melhorar a precisão das estimativas. É necessário, além disso, fazer uma análise criteriosa dos dados que são realmente relevantes e podem ser considerados no cálculo das estimativas.

### Usos Conhecidos:

O *CMMI* [CMMI], na área de processo Medição e Análise, obtém dados históricos dos resultados provenientes de base de dados e possibilita a análise dos dados.

O *MPS.BR* [Softex, 2006] no processo de Medição possui a atividade de análise dos dados e resultados. As informações produzidas são usadas para apoiar decisões e para fornecer uma base objetiva para comunicação aos interessados;

Em [Vazquez, 2003] é utilizado a atividade de Obter Dados Históricos no processo de estimativa de um projeto de software.

### Padrões Relacionados:

Os dados históricos são obtidos a partir da coleta prévia de medidas reais, *Coleta de Medidas Reais*. Obtenha dados históricos de projetos anteriores para auxiliar na classificação das funcionalidades da *Estimativa do Tamanho do Software*. Na *Estimativa de Esforço Total* os dados históricos auxiliam na obtenção da produtividade média da equipe. Na *Estimativa do Custo Total* os dados históricos auxiliam na porcentagem de esforço por atividade.

## 3 Conclusão

Este trabalho tem por principal objetivo auxiliar os gerentes e líderes de projeto das MPEs a mensurar valores do software na fase de planejamento de projeto de forma rápida e precisa. Além de prover um processo simples de melhoria da qualidade ao sugerir a coleta de medidas reais no decorrer do desenvolvimento do software para apoiar em futuras estimativas e decisões estratégicas da empresa.

## 4 Agradecimentos

Agradecimentos aos colegas Mário, Renato, Robson e Viviane da disciplina de Padrões de Software, 2007.1, que contribuíram de forma significativa com sugestões de melhoria para o engrandecimento deste trabalho.

Agradecimentos especiais ao Prof. Sérgio Soares, shepherd desse artigo, que contribuiu de forma significativa com as críticas e sugestões de melhoria.

## 5 Lista de Siglas e Abreviaturas

Siglas/Abreviaturas	Significado
CUSTO	Estimativa de Custo do Projeto
EPRZ	Estimativa de Prazo do Projeto
ESF	Estimativa de Esforço Total do Projeto
ETT	Estimativa do Tamanho Total
MPE	Micro e Pequena Empresa

PROD	Produtividade
QtdRD	Quantidade de Recursos Humanos Disponíveis
VlrHORA	Valor da Hora de cada profissional
VlrJTRAB	Valor da Jornada de Trabalho

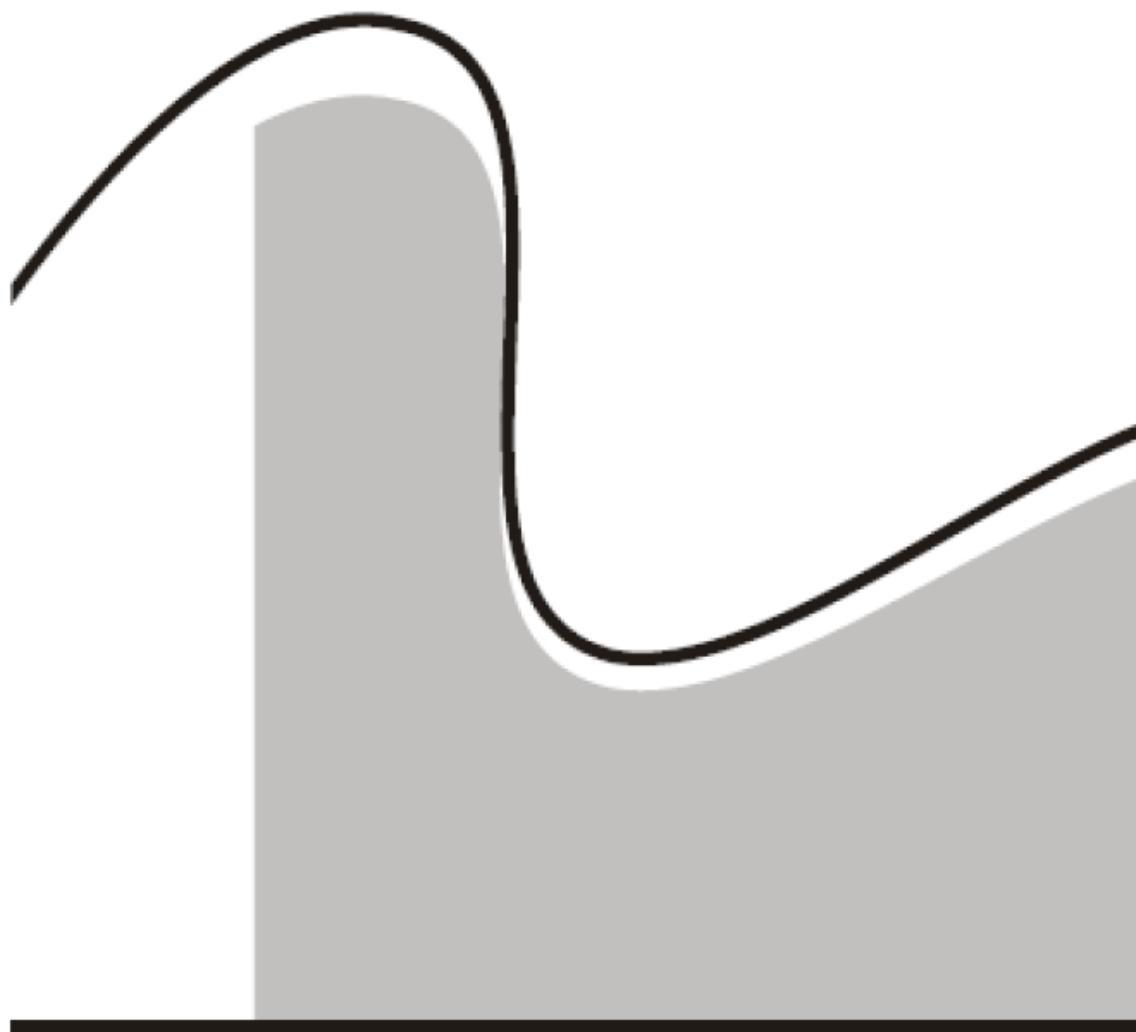
## 6 Referências

- ABRAN, A. et al. *Functional size measurement methods: COSMIC-FFP: design and fieldtrials*. In: FESMA-AEMES Software Measurements Conference, 2000.
- AGUIAR, M. *Pontos de Função ou Pontos de Caso de Uso? Como estimar Projetos Orientados a Objetos*. Developers Magazine.
- ALBRECHT, A.J. *Measuring Applications Development Productivity*. Proceedings of IBM Applic. Dev. Joint SHARE/GUIDE Symposium, Monterey, CA. 1979.
- ANDRADE, E., OLIVEIRA, K. *Pontos de Caso de Uso e Pontos de Função na gestão de estimativa de tamanho de projetos de software orientados a objetos*. Universidade Católica de Brasília. 2004.
- BANERJEE, G. *Use case Points - an Estimation Approach*. 2001.
- BOEHM, B et al. *Software Cost Estimation with COCOMOII*. Prentice Hall, 2000.
- CMMI for Development*. Disponível em: <<http://www.sei.cmu.edu/cmmi/>>. Acesso em: 05 abr. 2006.
- CLT – Consolidação das Leis do Trabalho*. Título II: Das Normas Gerais da Tutela do Trabalho. Capítulo II: Da duração do Trabalho. Seção II: Da jornada de trabalho. Artigo 64. Decreto-Lei 5.442, 1943.
- DAMODARAN, M., WASHINGTON, A. *Estimation Using Use Case Points*. 2003.
- FETCKE, T., ABRAN, A., NGUYEN, T. *Mapping the OO-Jacobson Approach into Function Point Analysis*. Université du Québec à Montréal. Software Engineering Management Research Laboratory. 2003.
- FETCKE, T., ABRAN, A., NGUYEN, T. H. *Mapping the OO Jacobon approach to function point analysis*. In Proc. IFPUG 1997. Spring Conference. (1997) 134—142.
- IFPUG: Function Point Counting Practices: Case Study 3 – Object Oriented Analysis*. Object Oriented Design. International Function Point Users Group. 1995. 229--244.
- INTERNATIONAL FUNCTION POINT USERS GROUP (IFPUG). *Function Point Counting Practices Manual*. Version 4.1, January, 1999.
- JACOBSON, I.; BOOCHE, G.; RUMBAUGH J. *The Unified Software Development Process*. Addison-Wesley, 1999.
- KARNER, G. *Metrics for Objectory*. Diploma thesis, University of Linköping, Sweden. No. LiTH-IDA-Ex-9344:21. December, 1993.
- KRUCHTEN, P. *Introdução ao RUP – Rational Unified Process*. Rio de Janeiro: Ciência Moderna, 2003.
- LÓPEZ, P. *COCOMONO II – Um modelo para estimativas de Gerência de Projetos*. UNISINOS, RS.
- MCT – Ministério de Ciência e Tecnologia. *Pesquisa Nacional de Qualidade e Produtividade no Setor de Software Brasileiro*. Brasil, 2005.

- MONTEIRO, T. *Pontos de Caso de Uso Técnicos – TUCP: Uma extensão do UCP*. Universidade de Fortaleza. 2005.
- PMI, Project Management Institute (Editor). *Um Guia do Conjunto de Conhecimentos do Gerenciamento de Projetos - PMBOK* (Project Management Body of Knowledge) Guide. PMI, Edição em português – 2004.
- PRESSMAN, R. Engenharia de Software. Makron Books, 1995.
- PROBASCO, L. *What About Function Points and Use Cases?*. Rational Software Canada.
- ROUILLER, A.C. *Gerenciamento de Projetos de Software para Empresas de Pequeno Porte*. Tese de Doutorado, UFPE, 2001.
- RIBU, K. *Estimating Object-Oriented Software Projects with Use Cases*. University of Oslo. Department of Informatics. 2001.
- SOFTEX. *MPS.BR – Melhoria de Processo do Software Brasileiro, Guia Geral*. Mai, 2006.
- SYMONS, C.R. *Software Sizing and Estimating, MKII FPA*. John Wiley and Sons, 1991.
- TAVARES, H., CARVALHO, A., CASTRO, J. *Medição por Pontos de Função a partir da Especificação dos Requisitos*. SERPRO. 2000.
- UNITED KINGDOM SOFTWARE METRICS ASSOCIATION. *Function Point Analysis Counting Practices Manual*. Version 1.3.1. 1998.
- VAZQUEZ, C., SIMÕES, G., ALBERT, R. *Análise de pontos de função: medição, estimativas e gerência de projetos*. São Paulo: Érica, 2003.



# SugarLoafPLoP'2008



Pattern Applications



# Método Para Desenvolvimento Utilizando Padrões de Software

Alessandra Chan<sup>1\*</sup>, Rosana T. V. Braga<sup>1†</sup>

<sup>1</sup>Instituto de Ciências Matemáticas e de Computação  
Universidade de São Paulo  
Caixa Postal 668 – 13560-970 – São Carlos – SP – Brasil

alechan@icmc.usp.br, rtvb@icmc.usp.br

**Resumo.** *O emprego de padrões de software, requisitos de teste, métodos e processos de desenvolvimento na criação de aplicações pode aumentar a produtividade das equipes e a qualidade do produto final. No entanto, engenheiros de software nem sempre utilizam padrões de software durante o desenvolvimento, tanto por não lembrarem, quanto por desconhecerem a existência de uma solução comprovada. Assim, este artigo apresenta a proposta de um Método Para Desenvolvimento Utilizando Padrões de Software para estimular o emprego de padrões de software durante as etapas de um processo de desenvolvimento com a associação prévia desses padrões a essas etapas.*

**Abstract.** *The use of software patterns, test requirements, methods and development processes in the creation of applications can increase teams productivity and the final product quality. However, software engineers are not using software patterns during development as they could, both for not remembering or unknowing the existence of a proven solution. Thus, this paper presents a proposal of a Method for Development Using Software Patterns to stimulate the use of software patterns during the stages of a development process with the prior association of these patterns to each stage.*

## 1. Introdução

Padrões de software constituem uma técnica eficaz de reúso, mas utilizá-los em projetos de desenvolvimento requer um certo custo, principalmente por causa da sua complexidade. Além disso, existe uma grande quantidade de padrões na literatura utilizando normas distintas de nomenclatura e definição, dificultando a consulta pelo padrão adequado a ser empregado [Pressman 2005]. Assim, é necessário que o projetista possua um conhecimento de diversos padrões para que utilize os mais adequados na solução de problemas.

Atualmente, há uma preocupação com teste dos artefatos resultantes de desenvolvimento com reúso. As atividades de VV&T (Verificação, Validação e Teste) é uma das principais para a garantia de qualidade, minimizando erros e riscos associados ao desenvolvimento [Rocha et al. 2001]. No entanto, essa atividade é uma das mais onerosas da Engenharia de Software [Maldonado et al. 2004, Myers 2004, Pressman 2005] e, no mundo competitivo atual, cresce a importância do desenvolvimento de software de alta qualidade, com preços acessíveis e em tempo reduzido [Chan 2008]. Nesse contexto, uma das dificuldades encontradas é a avaliação da qualidade do padrão que se deseja utilizar,

---

\* Apoio financeiro do CNPq

† Apoio financeiro da Fapesp

pois há poucos indícios de como ele foi validado e de como validar a solução utilizada. Uma das opções para resolver esse problema é adicionar uma seção no padrão para auxiliar os engenheiros de software a validar a solução por ele proposta [Cagnin et al. 2005].

Outro ponto a ser considerado são os problemas que ocorrem durante o desenvolvimento e manutenção de aplicações, que podem ser evitados por meio da utilização de processos de desenvolvimento disciplinados, métodos, técnicas e ferramentas, além de também colaborarem para construção de aplicações com maior qualidade.

Assim, ferramentas automatizadas têm um papel importante para que métodos possam ser empregados corretamente, além de apoiar e agilizar o reúso de padrões de software. O teste de aplicações também pode ser auxiliado pela utilização de ferramentas, permitindo maior rapidez e confiança na produção de dados sobre a execução.

Atualmente, ferramentas auxiliam engenheiros de software de diversas maneiras, como por exemplo, para apoiar a implementação, modelagem de diagramas, controle de versão, consulta a padrões de software e teste de software. Dentre as ferramentas e ambientes existentes, o ambiente Peônia [Chan 2008] foi desenvolvido com o objetivo de incentivar o emprego de padrões de software durante a execução de um processo de desenvolvimento. Esse ambiente possibilita que os usuários acrescentem processos de desenvolvimento, padrões de software e requisitos de testes e permite que sejam associados, para que durante a execução de projetos eles sejam sugeridos, estimulando a sua utilização.

Durante o desenvolvimento e teste do ambiente Peônia foi possível observar a repetição do fluxo de execução no desenvolvimento de projetos apoiado por padrões de software, motivando a elaboração do Método Para Desenvolvimento Utilizando Padrões de Software [Chan 2008], apresentado neste artigo.

Esta seção contém uma breve introdução sobre o contexto e motivação do trabalho proposto. As demais seções deste artigo estão organizadas nos seguintes tópicos: a Seção 2 resume os conceitos básicos, a Seção 3 apresenta o ambiente Peônia, a Seção 4 explica o método proposto e a Seção 5 apresenta as conclusões sobre o assunto tratado e os trabalhos futuros.

## 2. Conceitos Básicos

Esta seção apresenta um resumo dos conceitos básicos para o entendimento dos temas que envolvem o trabalho proposto. Os seguintes tópicos são abordados: definição de padrões de software e descrição de elementos obrigatórios em sua composição (Seção 2.1), diferença entre processo e método de desenvolvimento (Seção 2.2), descrição de terminologias utilizadas dentro das atividades de VV&T e o conceito de requisito de teste considerado neste trabalho (Seção 2.3).

### 2.1. Padrões de Software

No final da década de 70, Christopher Alexander introduziu na área da arquitetura as primeiras definições sobre padrões e linguagem de padrões, além de descrever o seu método de documentação [Alexander 1977, Alexander 1979]. Posteriormente, na década de 80, surgiram os primeiros padrões na área de software, com o intuito de captar a estrutura essencial e o raciocínio de uma família de soluções bem sucedidas e comprovadas para

um problema recorrente que ocorre dentro de um certo contexto [Appleton 2000]. Além de identificar a solução para um problema, padrões também devem explicar o porquê da necessidade da solução [Appleton 2000].

Padrões de Software podem trazer benefícios tanto às áreas ligadas diretamente ao projeto e implementação, quanto às áreas de outras disciplinas que fornecem suporte ao desenvolvimento de sistemas. Entretanto, se mal utilizados também podem trazer desvantagens como, por exemplo, a perda de eficiência causada pela adição de classes ou de novas camadas da aplicação e a diminuição da legibilidade e da manutenibilidade por causa do aumento da complexidade do código com a divisão de classes, mensagens, linhas de código e níveis hierárquicos de classes [Santos 2004].

Em razão da grande quantidade de padrões de software encontrados na literatura, é necessário atentar para sua qualidade. Uma das maneiras de selecionar bons padrões é verificar se possuem os elementos obrigatórios e se foram utilizados em pelo menos três aplicações. Além disso, uma outra forma do usuário selecionar e verificar a qualidade de um padrão é adicionar uma seção especial descrevendo como proceder para validar não somente o padrão como também as aplicações criadas a partir dele, conforme sugerido por Cagnin et al. (2005).

Padrões de software possuem diversos níveis de abstração. Para facilitar a sua recuperação e utilização, padrões podem ser divididos em categorias [Braga et al. 2001] de acordo com critérios de classificação. A classificação de padrões pode agilizar o aprendizado, além de direcionar esforços na descoberta de novos padrões [Gamma et al. 1995]. Alguns critérios de classificação são [Santos 2004]: Classificação Segundo o Estágio de Desenvolvimento de Software, Classificação Segundo o Domínio da Aplicação, Classificação Segundo a Camada de Aplicação do Padrão, Classificação GoF (*Gang of Four*) [Gamma et al. 1995] e Classificação POSA (*Pattern-Oriented Software Architecture*) [Buschmann et al. 1996].

Classificações não são rigorosas e alguns padrões podem pertencer a mais de uma categoria [Braga et al. 2001]. O *Almanaque de Padrões 2000* [Rising 2000] indexa os padrões por categorias, podendo ser classificados de acordo com o estágio de desenvolvimento, o domínio da aplicação ou as categorias utilizadas pela GoF.

Dos critérios de classificação mencionados, a Classificação Segundo o Estágio de Desenvolvimento de Software pode ser destacada por dividir os padrões de software considerando os estágios de desenvolvimento considerados pela Engenharia de Software [Corriveau et al. 1997, Jacobson 1992] *apud* [Andrade 2001].

Assim, segundo Andrade (2001), padrões de software podem ser classificados de acordo com o estágio de desenvolvimento em que são aplicados: *Padrões de Requisitos*, *Padrões de Análise*, *Padrões de Projeto* e *Padrões de Implementação*, ou *Idiomas*. De acordo com Santos (2004), pode ser considerada uma quinta categoria, a de *Padrões de Testes*.

## 2.2. Processos e Métodos de Desenvolvimento

Software é um produto complexo, difícil de desenvolver e testar. Freqüentemente, um software pode apresentar comportamentos inesperados e indesejados, podendo causar sérios problemas e perdas. Assim, pesquisadores têm se esforçado para aumentar a qua-

lidade do software. Uma das hipóteses é que há uma relação direta entre a qualidade do processo e a qualidade do software desenvolvido [Fuggetta 2000].

Um processo de desenvolvimento de software é o conjunto coerente de políticas, estruturas organizacionais, tecnologias, procedimentos, atividades e artefatos que são necessários para entender, desenvolver, implantar e manter um produto de software [Fuggetta 2000, Bertollo et al. 2006].

No entanto, não é trivial decidir o que deve ser incluído em um processo de desenvolvimento, pois devem ser consideradas as características da equipe de desenvolvimento e do projeto, o nível de conhecimento em Engenharia de Software, os propósitos da organização, o orçamento disponível, entre outros fatores. Assim, cada organização deve definir os seus processos e melhorá-los constantemente, de acordo com a experiência adquirida durante os projetos [Bertollo et al. 2006]. Rocha et al. (2001) consideram três etapas na definição de uma abordagem flexível de processos: definição de processos padrão para a organização, especialização dos processos padrão e instanciação para projetos específicos.

Processos de desenvolvimento, como os propostos por Pressman (2005) e por Murgesan e Ginige (2005), não definem um método para a sua utilização [Bianchini 2007]. Segundo o dicionário [Houaiss 2006], método é o “*procedimento, técnica ou meio de se fazer alguma coisa*”. Um método ensina como desenvolver um software utilizando como base um conjunto de princípios básicos da Engenharia de Software, que abrangem princípios de cada área da tecnologia, incluindo atividades de modelagem e outras técnicas descritivas [Pressman 2005].

Para o emprego correto de processos e métodos, ferramentas e ambientes podem fornecer apoio automatizado ou semi-automatizado. Para apoiar e acompanhar o emprego de processos de desenvolvimento de software têm-se, por exemplo, o ambiente WebAPSEE (*Web Process-Centered Software Engineering Environments*) [Lima et al. 2006] e a ferramenta ODE (*Ontology-based software Development Environment*) [Bertollo et al. 2006]. Com relação aos métodos de desenvolvimento, podem ser encontrados atualmente vários estudos como: HDM (*Hypermedia Design Model*) [Garzotto et al. 1991, Garzotto et al. 1993], OOHDMD (*Object-Oriented Hypermedia Design Method*) [Rossi 1996], UWE (*UML-based Web Engineering*) [Koch 2000], WAE (*Web Application Extension*) [Conallen 2002] e ECO (*Ecosystem of Agile Software Development*) [Figueiredo 2005].

Também é importante destacar que, em razão da grande quantidade de padrões de software existentes na literatura atual, é difícil para o usuário identificar o mais adequado a ser utilizado em uma etapa de um processo de desenvolvimento. Repositórios de padrões podem ser encontrados [Marinho et al. 2003, Bolchini et al. 2002], mas é necessário que o usuário tenha um conhecimento prévio da existência do padrão para saber em qual momento utilizá-lo.

### **2.3. Requisitos de Teste**

Um software deve ser previsível e consistente sem oferecer surpresa aos seus usuários. [Myers 2004]. Mesmo que o processo de desenvolvimento de software utilize uma série de técnicas, métodos e ferramentas, erros no produto ainda podem ocorrer. Assim, um conjunto de atividades, denominadas de Garantia de Qualidade de Software, são intro-

duzidas durante todo o processo de desenvolvimento de software, destacando-se as atividades de VV&T, que visam minimizar riscos e erros associados. O teste é a atividade mais utilizada nesse contexto e constitui um dos elementos para fornecer evidências da confiabilidade do software [Maldonado 1991, Maldonado et al. 2004].

Segundo Myers (2004), é impraticável e geralmente impossível encontrar todos os erros de um programa. Assim, uma estratégia deve ser estabelecida antes de iniciar os testes [Myers 2004] para que seja coberta adequadamente a lógica do programa e para garantir que as condições do projeto tenham sido cumpridas [Pressman 2005].

Quatro etapas compõem o teste de software: planejamento de teste, projeto de casos de teste, execução e avaliação dos resultados [Maldonado et al. 2004, Myers 2004, Pressman 2005]. Essas etapas devem ser desenvolvidas ao longo do processo de desenvolvimento e geralmente são concretizadas em três fases: teste de unidade, de integração e de sistema [Maldonado et al. 2004].

Casos de teste são criados seguindo os requisitos de teste, que são definidos a partir de critérios de teste, que, por sua vez, são estabelecidos de acordo com as técnicas de teste escolhidas. De acordo com o tipo de informação que se deseja testar, escolhe-se a técnica de teste. Em geral, quatro técnicas são utilizadas: *Teste Funcional, Teste Estrutural, Teste Baseado em Erros e Teste Baseado em Máquinas de Estados Finitos* [Maldonado et al. 2004].

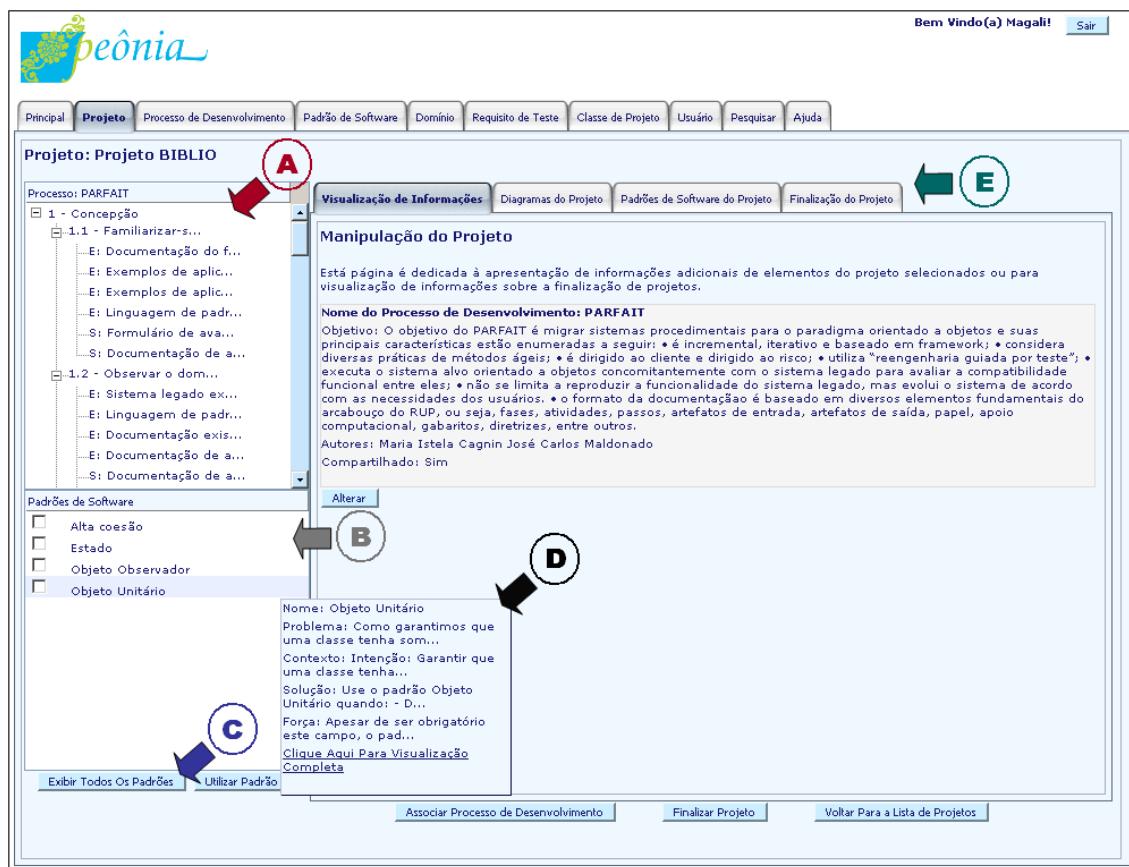
As atividades de VV&T são as mais onerosas no desenvolvimento de software [Rocha et al. 2001]. A associação de requisitos de teste a padrões de software pode auxiliar engenheiros de software na verificação das soluções propostas e reduzir o tempo despendido na atividade de VV&T [Cagnin et al. 2005]. Outra maneira de reduzir o tempo é o desenvolvimento de ferramentas de automatização, que são importantes no suporte à atividade de teste, propiciando maior qualidade e produtividade [Maldonado et al. 2004].

### **3. Ambiente Peônia**

Como há uma carência por ferramentas que apóiem engenheiros de software no emprego de padrões de software nas diversas etapas de um processo de desenvolvimento e que auxiliem a validação das soluções utilizadas [Chan et al. 2007], foi desenvolvido o ambiente Peônia [Chan 2008]. Esse ambiente apóia a organização e exibição de padrões de software previamente cadastrados, seguindo processos de desenvolvimento disciplinados, auxiliando o emprego correto dessas soluções comprovadas, minimizando esforços e melhorando a qualidade do produto de software [Chan et al. 2007].

A organização e exibição são realizadas por meio da associação dos padrões a cada uma das etapas de um processo de desenvolvimento e que serão exibidos aos usuários quando uma etapa é executada durante o projeto de uma aplicação. Também é possível associar requisitos de teste aos padrões de software cadastrados, para que usuários tenham diretrizes de como foi testado e a maneira de testar padrões utilizados em seus projetos.

Além disso, o ambiente Peônia permite que usuários realizem o acompanhamento da evolução de projetos, controlando a situação das etapas de um processo de desenvolvimento de software. Na Figura 1, é ilustrada a tela de um projeto aberto e em andamento, onde padrões de software podem ser utilizados durante a execução das etapas de um processo.

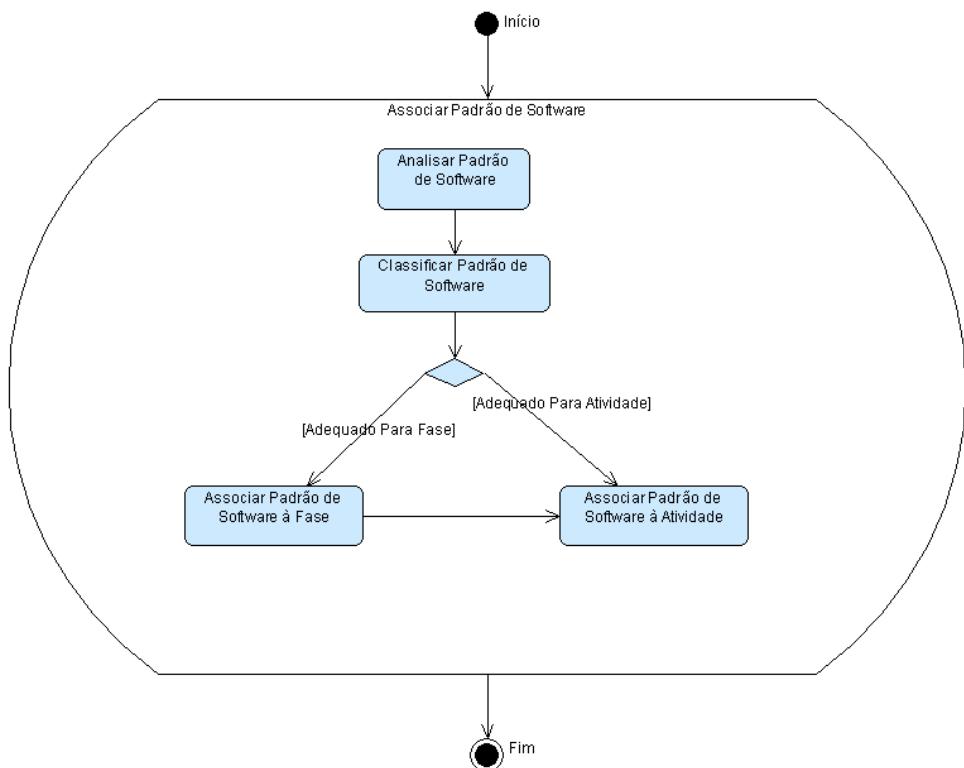


**Figura 1.** Tela exibida ao abrir um projeto

A seta “A” indica a estrutura do processo de desenvolvimento utilizado no projeto aberto. A parte inferior esquerda, sinalizada pela seta “B”, é utilizada para a exibição de padrões de software do ambiente Peônia. Ao abrir o projeto, todos os padrões do repositório são exibidos. Caso seja selecionada uma fase ou atividade, apenas os padrões de software associados à etapa escolhida são exibidos. A qualquer momento o usuário pode visualizar todos os padrões do ambiente Peônia selecionando a opção indicada pela seta “C”. Ao passar o mouse sobre um padrão, um resumo sobre os seus campos obrigatórios é apresentado. A visualização completa pode ser realizada selecionando o atalho indicado pela seta “D”. Nas abas do lado esquerdo, apontada pela seta “E”, são efetuadas as atividades referentes a execução e manipulação das informações do projeto.

#### 4. Método Para Desenvolvimento Utilizando Padrões de Software

Com o objetivo de auxiliar desenvolvedores no emprego de padrões de software e requisitos de teste, foi desenvolvido o ambiente Peônia [Chan 2008], brevemente descrito na Seção 3. Durante o desenvolvimento e teste do ambiente Peônia foi possível observar a repetição do fluxo de execução no desenvolvimento de projetos apoiado por padrões de software, motivando a elaboração do Método Para Desenvolvimento Utilizando Padrões de Software. Por meio do emprego desse método, espera-se que os usuários utilizem com maior facilidade o ambiente Peônia e apliquem mais padrões de software em projetos. O método também se preocupa com a qualidade dos projetos desenvolvidos, sugerindo o teste dos padrões de software durante a execução das etapas e criação dos artefatos.



**Figura 2. Associação de Fases e Atividades a Padrões de Software.**

Apesar de ter sido concebido para facilitar a utilização do ambiente Peônia, o método proposto concentra-se em dar diretrizes para o emprego de padrões de software durante as etapas de desenvolvimento de software, independentemente de ser apoiado ou não pelo ambiente. No entanto, é importante ressaltar que o método proposto foi elaborado a partir de projetos criados utilizando o ambiente Peônia, sendo necessário realizar medições controladas sobre sua eficiência e eficácia [Chan 2008].

Os diagramas apresentados neste artigo foram criados utilizando a notação UML (*Unified Modeling Language*) [OMG's 2006]. As caixas retangulares com os lados arredondados representam *Atividades*, enquanto as caixas elípticas achatadas representam *Ações*. Nos diagramas de atividades do método proposto, atividades são compostas por conjuntos de ações a serem executadas.

Inspirado na classificação segundo o estágio de desenvolvimento de software, mencionado na Seção 2.1, o Método Para Desenvolvimento Utilizando Padrões de Software propõe que padrões de software sejam associados às etapas de um processo de desenvolvimento para facilitar a utilização durante a execução de um projeto e estimular usuários a empregá-los. Assim, o método proposto pode ser dividido em duas fases: a *Associação de Padrões de Software às Etapas de Um Processo de Desenvolvimento* e a *Visualização e Utilização de Padrões de Software Durante a Execução de Um Processo de Desenvolvimento*.

A primeira das fases é realizada por meio da atividade de associação de padrões de software às etapas de um processo (*Atividade: Associar Padrão de Software*), ilustrada na Figura 2. Essa atividade deve ser realizada antes da execução do processo. Para que auxi-

liem efetivamente em projetos, os padrões devem ser analisados (*Ação: Analisar Padrão de Software*) para que sejam corretamente classificados (*Ação: Classificar Padrão de Software*), verificando a adequação quanto a aplicação em fases (*Ação: Associar Padrão de Software à Fase*) ou atividades (*Ações: Associar Padrão de Software à Fase ou Associar Padrão de Software à Atividade*) de um processo de desenvolvimento. É importante destacar que um padrão de software associado a uma fase também é recomendado para todas as atividades que compõem essa fase [Chan 2008].

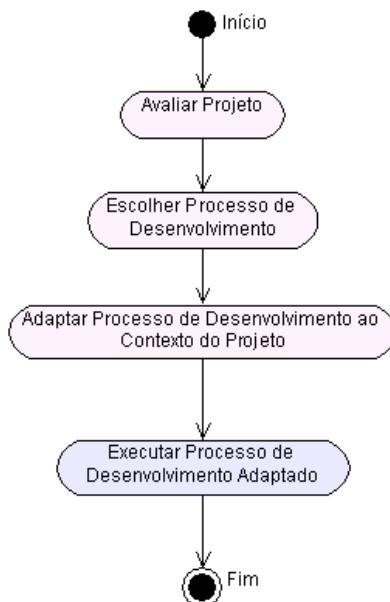
Em geral, usuários buscam padrões de software apenas quando se deparam com um problema e não encontram facilmente uma solução. Ao associá-los às fases e atividades de um processo de desenvolvimento o usuário é induzido a averiguar, antes mesmo de encontrar um problema, a necessidade de utilizar esses padrões de software nas fases e em cada uma de suas atividades e, consequentemente, nos artefatos que são produzidos. Dessa maneira, o usuário é incentivado desde o início do projeto a utilizar as soluções propostas pelos padrões, melhorando a qualidade do software a ser produzido. Além disso, ao visualizar os padrões de software associados, mesmo que não os utilize, quando encontrarem um problema, a solução provavelmente será localizada com maior facilidade [Chan 2008].

Seguindo o mesmo princípio, associando requisitos de teste a padrões de software o usuário pode testar a solução empregada com maior facilidade, principalmente porque muitas vezes o usuário não é o autor do padrão, desconhecendo a maneira mais apropriada de verificar, validar ou testar uma solução. Essa associação auxilia engenheiros de software a averiguar as soluções propostas e pode reduzir o tempo despendido nas atividades de VV&T [Cagnin et al. 2005].

Os elementos fundamentais do RUP (*Rational Unified Process*) [Kruchten 2000] são utilizados como a base da estrutura dos processos de desenvolvimento empregada na definição das atividades executadas pelo Método Para Desenvolvimento Utilizando Padrões de Software. O RUP é um processo de desenvolvimento que provê uma abordagem disciplinada para determinar tarefas e responsabilidades em um projeto. Além disso, fornece um arcabouço de processo extensível e adaptável às necessidades dos usuários [Cagnin 2005].

Exemplos de processos de desenvolvimento baseados no RUP são o PARFAIT (Processo Ágil de Reengenharia baseado em FrAmeWork no domínio de sistema de Informação com técnicas de VV&T) [Cagnin 2005] e o processo proposto por Conallen (2002) utilizando o WAE. Neste trabalho, apenas os elementos fundamentais do RUP são empregados, pois a intenção é fornecer a base essencial para a execução de um processo de desenvolvimento e, ao mesmo tempo, dar a liberdade para o usuário criar a estrutura do processo a ser empregado [Chan 2008].

De acordo com o RUP, um processo de desenvolvimento é composto por *fases* que, por sua vez, são compostas de *atividades*. A execução de uma atividade é atribuída a uma pessoa que desempenha um *papel* no desenvolvimento do produto. Uma *atividade* descreve os procedimentos de como o trabalho deve ser realizado e pode receber artefatos de entrada e produzir artefatos de saída. Um *artefato* é um produto de trabalho gerado com a execução de uma atividade, por exemplo, modelos, elementos de modelo, código-fonte e documentos [Conallen 2002].



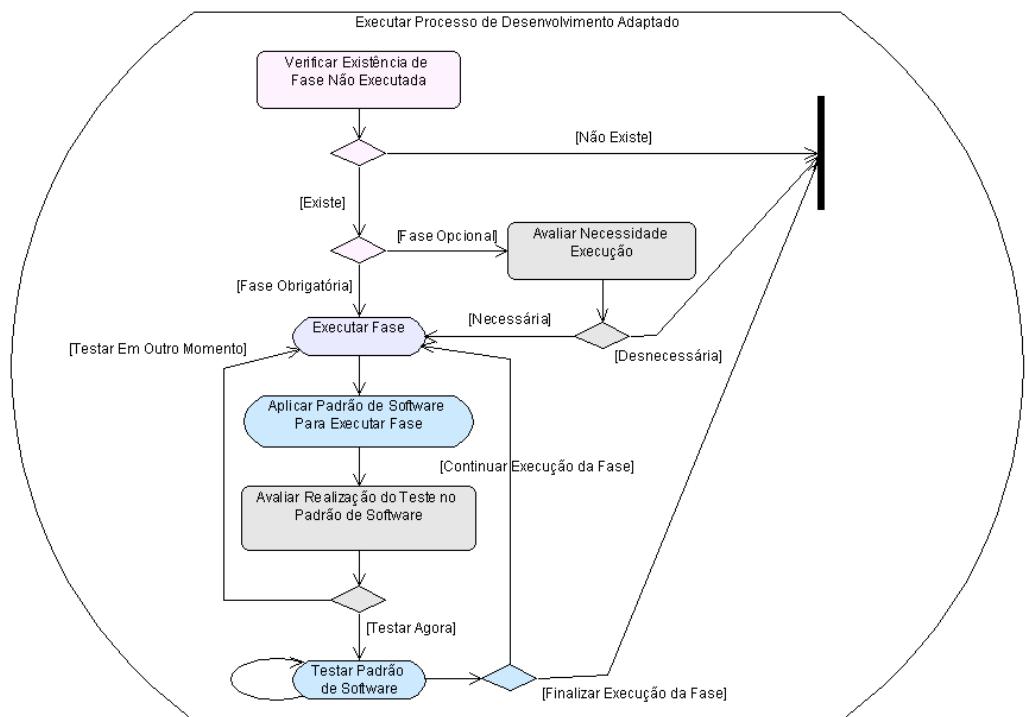
**Figura 3. Visão Geral da Visualização e Utilização de Padrões de Software Durante a Execução de Um Processo de Desenvolvimento.**

Na Figura 3, é exibida uma visão geral de como um padrão de software pode ser visualizado e utilizado durante a execução de um processo de desenvolvimento. Basicamente, são ilustradas as atividades que envolvem a preparação e a execução de um projeto. Antes de iniciar um projeto, é necessário avaliá-lo (*Atividade: Avaliar Projeto*), por exemplo, realizando a análise do domínio e cronograma. A partir das informações levantadas, um processo de desenvolvimento deve ser escolhido (*Atividade: Escolher Processo de Desenvolvimento*) para organizar, estruturar e controlar a execução do projeto.

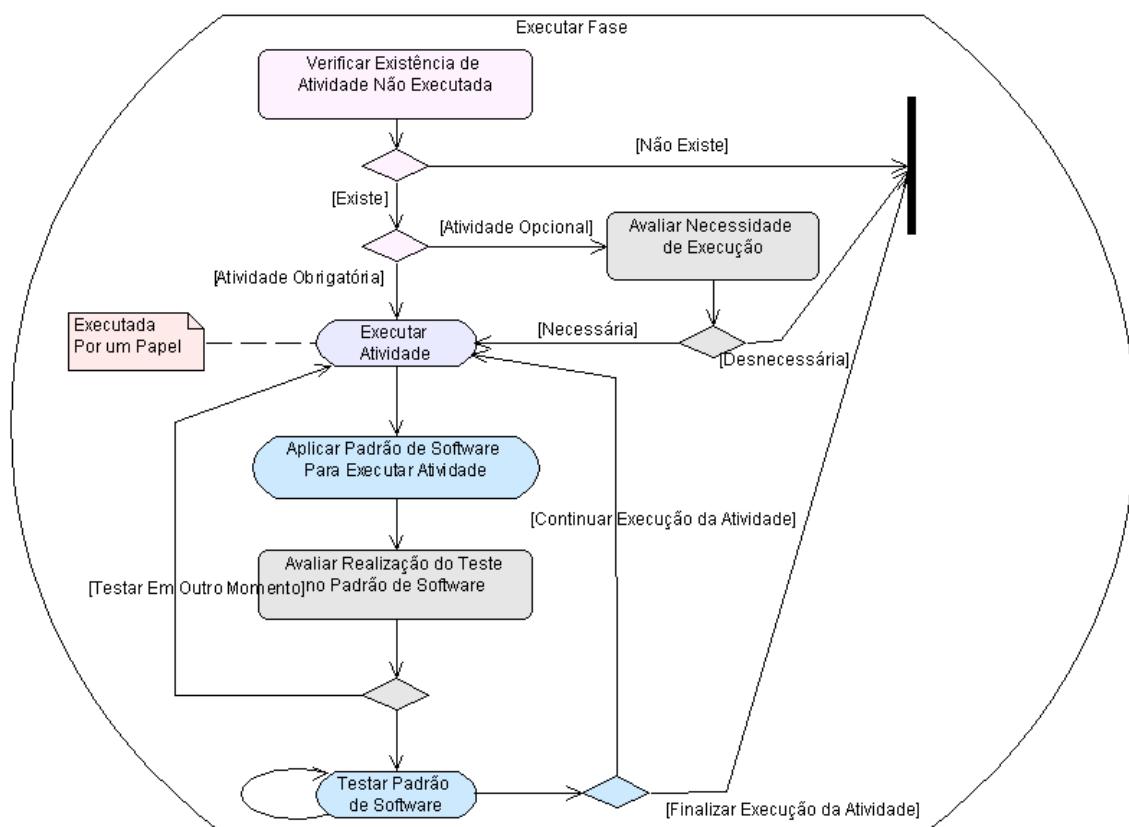
O processo de desenvolvimento escolhido pode conter elementos desnecessários ou não compatíveis com o escopo do projeto, tornando necessário selecionar o que deve ser excluído do processo e o que deve ser adicionado para que o projeto seja executado com sucesso. Além disso, a obrigatoriedade de cada elemento de um processo pode ser alterada (*Atividade: Adaptar Processo de Desenvolvimento ao Contexto do Projeto*).

Após escolher e adaptar um processo de desenvolvimento, o projeto é iniciado (*Atividade: Executar Processo de Desenvolvimento Adaptado*) por meio da realização de cada uma das suas fases. A execução de um processo de desenvolvimento é ilustrado na Figura 4. Durante a execução do processo de desenvolvimento, verifica-se a existência de fases que ainda não tenham sido cumpridas para que sejam averiguadas quanto à necessidade de executá-las (*Ação: Verificar Existência de Fase Não Executada*).

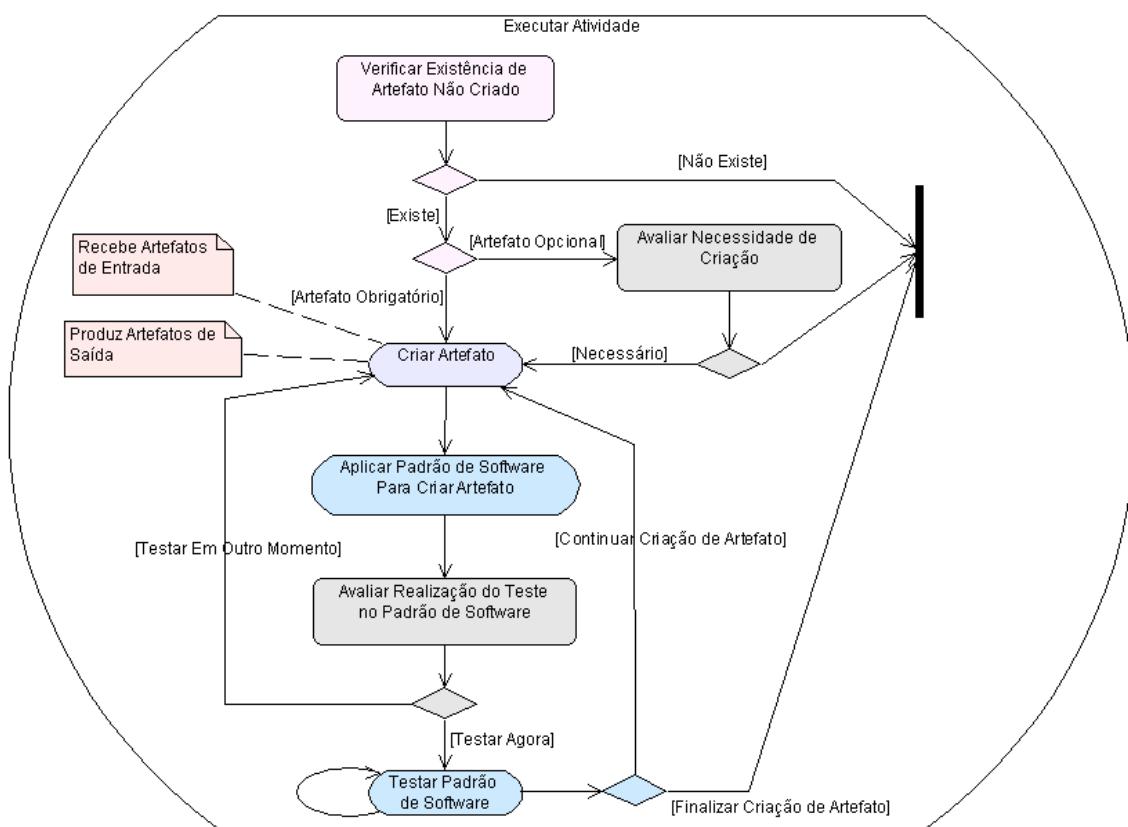
Dois tipos de fases podem existir dentro de um processo: obrigatorias e opcionais. Caso existam fases obrigatorias não cumpridas, elas devem ser executadas (*Atividade: Executar Fase*), pois impedem que um processo seja finalizado com sucesso, produzindo a aplicação com a qualidade desejada. Se fases opcionais existirem, deve-se avaliar a necessidade de cumpri-las (*Ação: Avaliar Necessidade de Execução*), pois não são essenciais para a conclusão do projeto.



**Figura 4. Diagrama de Atividades de Executar Processo de Desenvolvimento Adaptado.**



**Figura 5. Diagrama de Atividades de Executar Fase.**

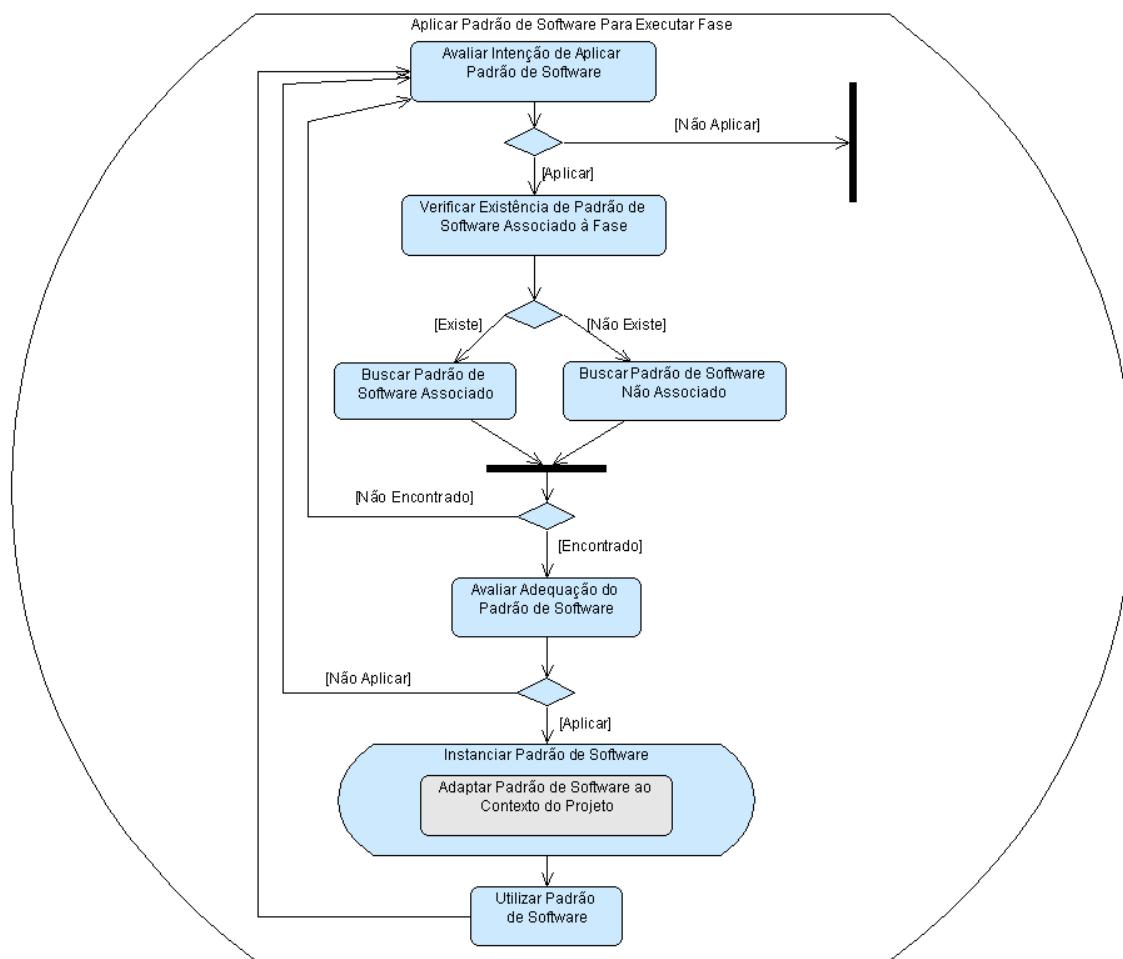


**Figura 6. Diagrama de Atividades de Executar Atividade.**

Como pode ser observado na Figura 5, uma fase é cumprida por meio da execução de todas as atividades obrigatórias que a compõem. Portanto, enquanto uma fase é executada buscam-se as atividades que ainda não foram executadas (*Ação: Verificar Existência de Atividade Não Executada*). Caso sejam obrigatórias, elas devem ser cumpridas (*Atividade: Executar Atividade*). Se forem atividades opcionais, deve-se averiguar os impactos de executá-las (*Ação: Avaliar Necessidade de Execução*).

Por fim, uma atividade é executada quando os artefatos que devem ser produzidos são criados com sucesso, como pode ser observado na Figura 6. Enquanto uma atividade é executada, é realizada a análise dos artefatos ainda não criados (*Ação: Verificar Existência de Artefatos Não Criados*). Caso sejam obrigatórios, eles devem ser criados (*Atividade: Criar Artefato*) e, se forem opcionais, deve-se averiguar a necessidade de criá-los (*Ação: Avaliar Necessidade de Criação*). Após a conclusão de todas as fases e atividades obrigatórias e a produção dos artefatos obrigatórios, o processo pode ser finalizado, pois o desenvolvimento da aplicação é considerado como concluído.

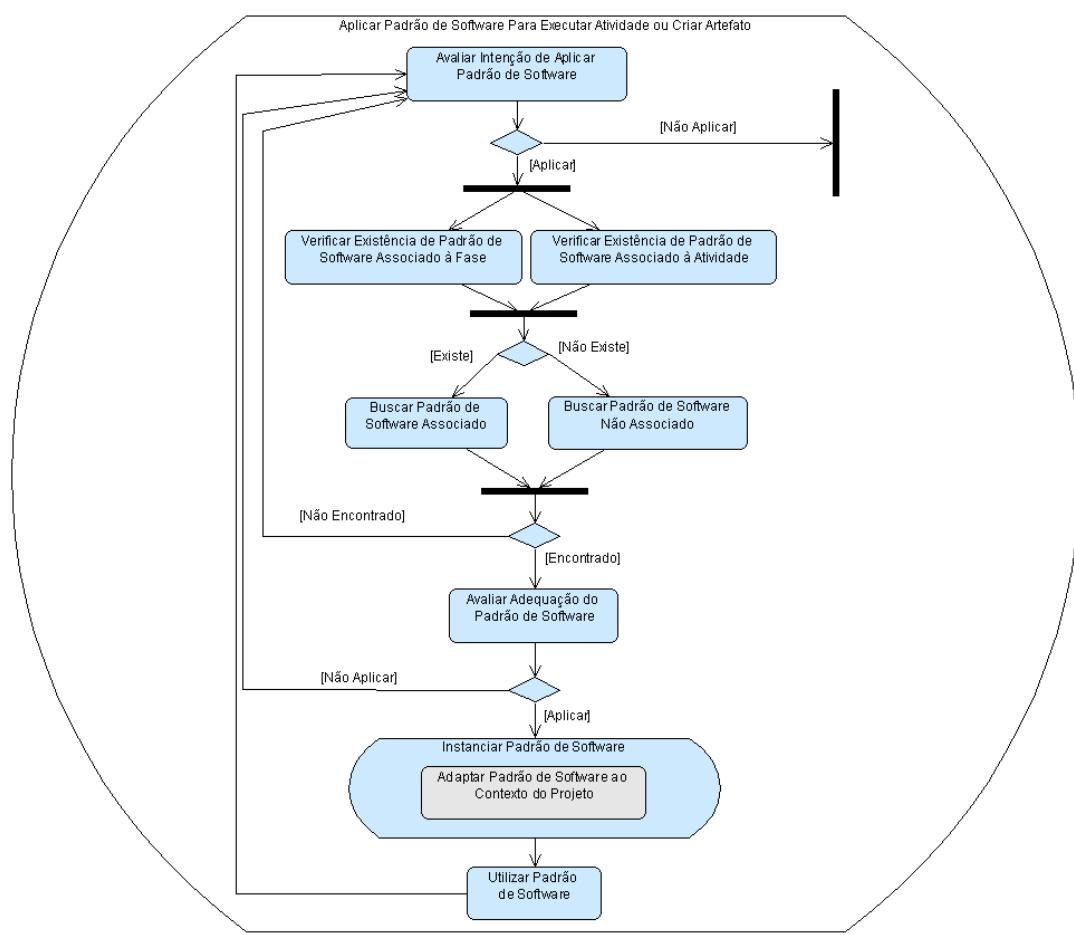
Por meio da produção dos artefatos, o projeto evolui incrementalmente até chegar à aplicação desejada. Esses artefatos podem ser construídos utilizando padrões de software instanciados, permitindo a alteração e adaptação ao contexto do projeto. Padrões de software também podem ser instanciados e utilizados para apoiar a execução de fases e atividades de um processo de desenvolvimento.



**Figura 7. Diagrama de Atividades de *Aplicar Padrão de Software Para Executar Fase*.**

Na Figura 7 é apresentada a aplicação de padrão de software durante a execução de uma fase (*Atividade: Aplicar Padrão de Software Para Executar Fase*) e na Figura 8 são ilustradas as atividades referentes ao uso de um padrão durante o desenvolvimento de uma atividade ou artefato, pois ambas possuem o mesmo fluxo de execução (*Atividades: Aplicar Padrão de Software Para Executar Atividade e Aplicar Padrão de Software Para Criar Artefato*).

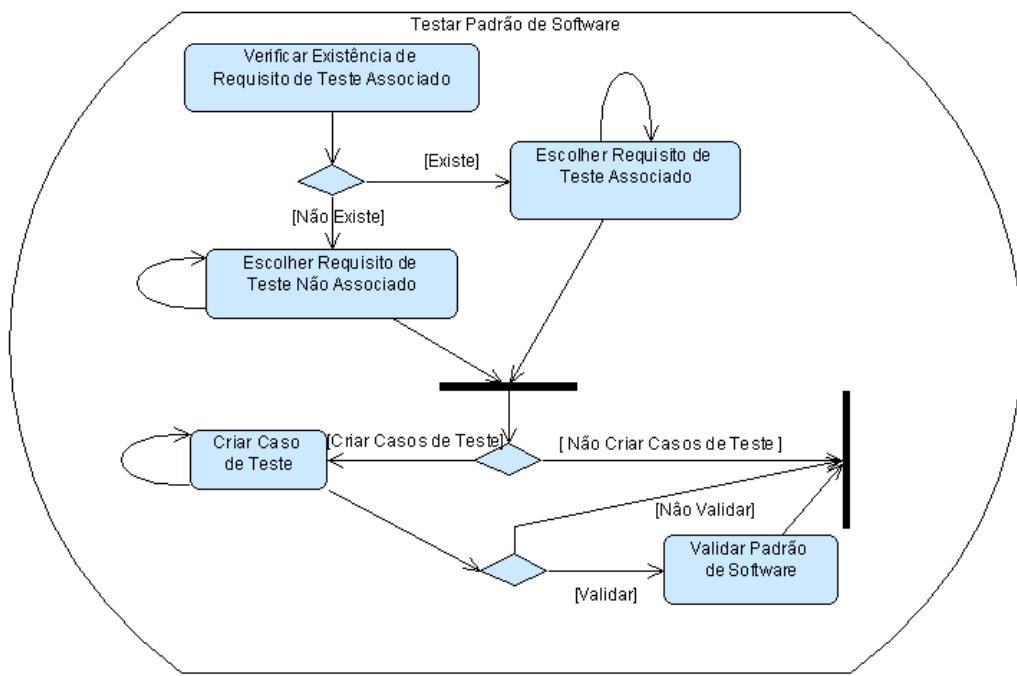
A diferença entre as duas figuras está na verificação de padrão de software associado a uma atividade (*Ação: Verificar Existência de Padrão de Software Associado à Atividade*), que não é efetuada ao empregar padrões de software na execução de uma fase (*Atividade: Aplicar Padrão de Software Para Executar Fase*). Isso ocorre porque, apesar de uma fase ser composta de atividades, padrões de software associados a uma atividade podem ser específicos demais para a fase. Assim, os padrões de software associados a uma atividade não são sempre aplicáveis à fase, sendo necessário avaliá-los separadamente. Caso não sejam aplicáveis a fase e estejam associados apenas à atividade, no método proposto, é considerado como uso de um padrão de software não associado à fase.



**Figura 8. Diagrama de Atividades de Aplicar Padrão de Software Para Executar Atividade e Aplicar Padrão de Software Para Criar Artefato.**

O emprego de padrões de software é opcional e depende da avaliação do usuário (*Ação: Avaliar Intenção de Aplicar Padrão de Software*). Caso opte por utilizar padrões de software no seu projeto, o usuário deve analisar e escolher a melhor solução para o seu problema. Como mencionado anteriormente, é difícil lembrar e escolher entre as diversas opções existentes atualmente na literatura. O método aqui proposto sugere que padrões de software sejam associados previamente às fases e atividades do processo executado, facilitando a visualização dos padrões mais adequados em cada etapa de um projeto.

Portanto, caso existam padrões associados à fase (*Ação: Verificar Existência de Padrão de Software Associado à Fase*), durante a realização de uma fase (*Atividade: Executar Fase*) o usuário pode utilizá-los para auxiliar no projeto. Padrões de software associados a uma fase também estão indiretamente associados a uma atividade. Um artefato não possui padrões associados diretamente, pois a sua criação é realizada em uma atividade, o que implica que todos os padrões associados a uma fase ou atividade também podem ser considerados como sugestões para auxiliar a criação de artefatos. Assim, o usuário tem a opção de procurar por padrões associados às fases e atividades (*Ação: Verificar Existência de Padrão de Software Associado à Fase e Verificar Existência de Padrão de Software Associado à Atividade*) para aplicar durante a realização de uma atividade (*Atividade: Executar Atividade*) e a geração de um artefato (*Atividade: Criar Artefato*).



**Figura 9. Diagrama de Atividades de *Testar Padrão de Software*.**

Destaca-se que, apesar de propor a associação, não é descartada a possibilidade de empregar padrões de software não associados. Assim, é proposta a realização de buscas pela solução mais adequada a ser empregada, tanto em padrões de software associados às etapas do processo de desenvolvimento (*Ação: Buscar Padrão de Software Associado*), quanto em outros padrões da literatura (*Ação: Buscar Padrão de Software Não Associado*). Caso seja encontrado um padrão na pesquisa realizada, deve ser realizada uma avaliação para verificar se a solução proposta é adequada para o problema do usuário (*Ação: Avaliar Adequação do Padrão de Software*).

Se o usuário concluir que o padrão de software é o mais adequado a ser empregado, o padrão de software deve ser instanciado (*Atividade: Instanciar Padrão de Software*). A instanciação de um padrão de software é realizada por meio da adaptação da solução para o contexto do projeto (*Ação: Adaptar Padrão de Software ao Contexto do Projeto*). Finalmente, o padrão instanciado é utilizado na execução de fases e atividades, ou na criação de artefatos (*Ação: Utilizar Padrão de Software*).

O momento e a maneira como o teste das soluções são realizados depende do projeto e é responsabilidade do usuário avaliar e definir quando isso deve acontecer (*Ação: Avaliar Realização do Teste do Padrão de Software*). Ao optar pela execução das atividades de VV&T, o padrão utilizado no projeto deve ser validado (*Atividade: Testar Padrão de Software*), tarefa que envolve a seleção de requisitos de teste. Como pode ser observado na Figura 9, na atividade de teste é necessário verificar a existência de requisitos de teste associados ao padrão de software (*Ação: Verificar Existência de Requisito de Teste Associado*) e, caso esses existam, são selecionados os requisitos (*Ação: Escolher Requisito de Teste Associado*) utilizados nas atividades de VV&T. Caso o usuário não queira empregar os requisitos associados, ou queira complementar o teste, também podem ser escolhidos requisitos de teste não associados (*Ação: Escolher Requisito de Teste Não As-*

*sociado).* Após a escolha dos requisitos de teste, devem ser gerados casos de testes (*Ação: Criar Caso de Teste*) utilizados no teste do padrão de software (*Ação: Validar Padrão de Software*).

Informações adicionais sobre os artefatos de entrada e saída de cada uma das atividades do Método Para Desenvolvimento Utilizando Padrões de Software podem ser encontradas no Apêndice A.

## 5. Conclusão

Engenheiros de software podem utilizar processos e métodos de desenvolvimento, padrões de software, ferramentas e ambientes para auxiliar o desenvolvimento de aplicações com maior qualidade. Explorando esse suporte comum, o ambiente Peônia foi desenvolvido com o intuito de fornecer flexibilidade para usuários cadastrarem processos de desenvolvimento e acompanharem a sua execução em projetos, sugerindo automaticamente padrões de software para serem empregados nas fases e atividades do processo de desenvolvimento utilizado pelo usuário no projeto, além de também apoiar as atividades de VV&T oferecendo requisitos de teste para validar os padrões de software empregados [Chan et al. 2007, Chan 2008].

Com o aumento da quantidade de padrões existentes, cresce também a dificuldade na visualização e escolha dos padrões de software mais adequados a serem empregados em um projeto [Chan et al. 2007]. Formalizando o método empregado no ambiente Peônia para incentivar a utilização de padrões de software durante as etapas de um processo de desenvolvimento é proposto o Método Para Desenvolvimento Utilizando Padrões de Software, descrito neste artigo.

O método propõe a associação prévia de padrões de software às fases e atividades de um processo de desenvolvimento, estimulando usuários a pelo menos considerar a possibilidade de utilizar esses padrões de software associados durante a execução de uma das etapas do desenvolvimento, além de tentar reduzir as chances de usuários deixarem de utilizá-los por não recordarem a existência das soluções.

Também é incentivado no método proposto a associação de requisitos de teste a padrões de software, como sugerido por Cagnin et al. (2005), auxiliando a validação dos padrões de software utilizados em projetos, podendo minimizar o tempo despendido nas atividades de VV&T.

O Método Para Desenvolvimento Utilizando Padrões de Software foi criado para facilitar a utilização do ambiente Peônia, pois fornece as diretrizes para o emprego de padrões de software durante as etapas de desenvolvimento de software. O método proposto não obriga usuários a utilizarem o ambiente Peônia, podendo ser empregado independentemente.

No entanto, é importante ressaltar que o método proposto foi elaborado a partir de projetos criados utilizando o ambiente Peônia, sendo necessário realizar medições controladas sobre sua eficiência e eficácia. Assim, em trabalhos futuros, planeja-se realizar essas medições e validações por meio de estudos de casos controlados experimentalmente para poder confirmar as vantagens de se utilizar o método no desenvolvimento apoiado por padrões de software, tanto em projetos criados no ambiente Peônia, quanto para validar o seu emprego de maneira independente [Chan 2008].

## Referências

- Alexander, C. (1977). *A Pattern Language*. Oxford University Press.
- Alexander, C. (1979). *The Timeless Way of Building*. Oxford University Press.
- Andrade, R. M. C. (2001). *Capture, Reuse, and Validation of Requirements and Analysis Patterns for Mobile Systems*. PhD thesis, SITE/University of Ottawa, Ottawa/Ontario - Canada.
- Appleton, B. (2000). Patterns and software: Essential concepts and terminology. Online.
- Bertollo, G., Segrini, B. M., and Falbo, R. A. (2006). Evoluindo a Definição de Processos de Software em ODE. In *SBES'06 - XIII Sessão de Ferramentas do SBES*, pages 109–114, Florianópolis/SC - Brasil.
- Bianchini, S. L. (2007). Avaliação de Metodologias de Desenvolvimento de Sistemas Web. Master's thesis, ICMC/USP, São Carlos/SP - Brasil. em andamento.
- Bolchini, D., Garzotto, F., Paolini, P., Lowe, D., Cantoni, L., Nanard, J., Rossi, G., Schwabe, D., and Ruggeri, R. (2002). HPR - Hypermedia Design Patterns Repository. Online.
- Braga, R. T. V., Germano, F. S. R., Masiero, P. C., and Maldonado, J. C. (2001). Introdução aos Padrões de Software. Nota Didática.
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1996). *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, Inc., New York/NY - USA.
- Cagnin, M. I. (2005). *PARFAIT: Uma Contribuição Para a Reengenharia de Software Baseada em Linguagem de Padrões e Frameworks*. PhD thesis, ICMC/USP, São Carlos/SP - Brasil.
- Cagnin, M. I., Braga, R. T. V., Germano, F., Chan, A., and Maldonado, J. C. (2005). Extending Patterns with Testing Implementation. In *SugarLoafPlop'2005, V Conferência Latino-Americana em Linguagens de Padrões para Programação*, Campos do Jordão/SP - Brasil.
- Chan, A. (2008). Peônia: um ambiente web para apoiar processos de desenvolvimento com utilização de padrões de software e requisitos de teste no projeto de aplicações. Master's thesis, ICMC/USP, São Carlos/SP - Brasil. em andamento.
- Chan, A., Cagnin, M. I., Maldonado, J. C., and Braga, R. T. V. (2007). Uma Proposta de Ambiente Para Apoiar a Utilização de Padrões de Software e Requisitos de Teste no Desenvolvimento de Aplicações. In *SugarLoafPlop'2007 Proceedings, VI Conferência Latino-Americana em Linguagens de Padrões para Programação*, Porto de Galinhas/PE - Brasil.
- Conallen, J. (2002). *Buildind Web Applications with UML*. Addison-Wesley, 2nd. edition.
- Corriveau, J., P., and Nel, D. N. (1997). Introduction to Object-Oriented Software Engineering Version 1.0. Course Notes.
- Figueiredo, A. (2005). ECO - um ecossistema para o desenvolvimento ágil de sistemas web. Master's thesis, ICMC/USP, São Carlos/SP - Brasil.
- Fuggetta, A. (2000). Software Process: A Roadmap. In *ICSE'00 - Future of Software Engineering Track*, pages 25–34, Limerick - Ireland.
- Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable of Object-Oriented Software*. Addison-Wesley, 2th. edition.
- Garzotto, F., Paolini, P., and Schwab, D. (1991). HDM - a model for the design of hypertext applications. In *Third Annual ACM Conference on Hypertext (Hypertext'91)*, pages 313–328, New York/NY - USA.

- Garzotto, F., Paolini, P., and Schwabe, D. (1993). HDM - a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26.
- Houaiss, A. (2006). Dicionário Houaiss da Língua Portuguesa. Online.
- Jacobson, I. (1992). *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.
- Koch, N. (2000). *Software Engineering for Adaptive Hypermedia Systems: Reference Model, Modeling Techniques and Development Process*. PhD thesis, Ludwig-Maximilians University, Munich - Germany.
- Kruchten, P. (2000). *The Rational Unified Process: An Introduction*. Addison-Wesley, 2th. edition. 298 p.
- Lima, A., Costa, A., França, B., Reis, C. A. L., and Reis, R. Q. (2006). Gerência Flexível de Processos de Software com o Ambiente WebAPSEE. In *SBES'06 - XIII Sessão de Ferramentas do SBES*, pages 97–102, Florianópolis/SC - Brasil.
- Maldonado, J. C. (1991). *Critérios Potenciais Usos: Uma Contribuição ao Teste Estrutural de Software*. PhD thesis, DCA/FEE/UNICAMP, Campinas/SP - Brasil.
- Maldonado, J. C., Barbosa, E. F., Vincenzi, A. M. R., and Márcio Eduardo Delamaro, Simone Rocio Senger Souza, M. J. (2004). Introdução ao Teste de Software. Nota Didática.
- Marinho, F., Santos, M., Pinto, R. N., and Andrade, R. (2003). Uma Proposta de um Repositório de Padrões de Software Integrado ao RUP. In *SugarLoafPlop Proceeding 2003, The Third Latin American Conference on Pattern Languages of Programming*, pages 277–290, Porto de Galinhas/PE - Brasil.
- Myers, G. J. (2004). *The art of software testing*. John Wiley & Sons, Inc., 2th. edition.
- OMG's (2006). UML Resource Page. Online.
- Pressman, R. S. (2005). *Engenharia de Software*. McGraw-Hill, 6th. edition.
- Rising, L. (2000). *The Pattern Almanac 2000*. Addison-Wesley Publishing Company, 1th. edition.
- Rocha, A. R. C., Maldonado, J. C., and K, C. W. (2001). *Qualidade de Software: Teoria e Prática*. Prentice Hall, 1th. edition.
- Rossi, G. (1996). *Um Método Orientado a Objetos para o Projeto de Aplicações Hipermídia*. PhD thesis, PUC-Rio, Rio de Janeiro/RJ - Brasil.
- Santos, M. S. (2004). Uma Proposta para a Integração de Modelos de Padrões de Software com Ferramentas de Apoio ao Desenvolvimento de Sistemas. Master's thesis, UFC, Fortaleza/CE - Brasil.

## A. Apêndice

Na Tabela 1, são listadas as atividades do Método Para Desenvolvimento Utilizando Padrões de Software. Também são apresentados os artefatos de entrada e saída produzidos em cada atividade.

**Tabela 1. Atividades e Artefatos do Método Para Desenvolvimento Utilizando Padrões de Software**

Atividade	Artefatos de Entrada	Artefatos de Saída
1. Avaliar Projeto	- Documento de Visão	
2. Escolher Processo de Desenvolvimento	- Documento de Visão - Lista de Processos de Desenvolvimento	- Documento Detalhando os Critérios Utilizados Para Selecionar o Processo de Desenvolvimento
3. Adaptar Processo de Desenvolvimento ao Contexto do Projeto	- Documento de Visão - Descrição Detalhada dos Elementos Que Compõem o Processo de Desenvolvimento Escolhido	- Documento Detalhando o Processo de Desenvolvimento Adaptado
4. Executar Processo de Desenvolvimento Adaptado	- Documento Detalhando o Processo de Desenvolvimento Adaptado	- Artefatos Criados Durante a Execução do Processo - Produto Desenvolvido - Lista de Padrões de Software Utilizados no Projeto - Documento Detalhando a Execução do Processo de Desenvolvimento Adaptado
4.1. Executar Fase	- Descrição da Fase - Lista de Atividades da Fase - Lista de Padrões de Software Associados à Fase - Lista de Padrões de Software Não Associados à Fase	- Lista de Padrões de Software Utilizados na Fase - Documento Detalhando os Testes Realizados na Fase - Artefatos Produzidos na Fase - Documento Detalhando a Execução da Fase
4.1.1. Executar Atividade	- Descrição da Atividade - Lista de Artefatos Que Devem Ser Criados Durante a Atividade - Lista de Padrões de Software Associados à Atividade - Lista de Padrões de Software Associados à Fase - Lista de Padrões de Software Não Associados à Atividade e à Fase	- Lista de Padrões de Software Utilizados na Atividade - Documento Detalhando os Testes Realizados na Atividade - Artefatos Produzidos na Atividade - Documento Detalhando a Execução da Atividade
4.1.1.1. Criar Artefato	- Descrição do Artefato - Lista de Padrões de Software Associados à Atividade - Lista de Padrões de Software Associados à Fase - Lista de Padrões de Software Não Associados à Atividade e à Fase	- Lista de Padrões de Software Utilizados Para Criar o Artefato - Documento Detalhando os Testes Realizados no Artefato - Artefato Criado - Documento Detalhando a Criação do Artefato
4.1.1.2. Aplicar Padrão de Software Para Criar Artefato, 4.1.2. Aplicar Padrão de Software Para Executar Atividade, 4.2. Aplicar Padrão de Software Para Executar Fase	- Informações Sobre o Padrão de Software Escolhido - Informações Sobre o Projeto	- Informações Sobre o Padrão de Software Adaptado ao Contexto do Projeto
4.1.1.2.1, 4.1.2.1 e 4.2.1. Instanciar Padrão de Software	- Informações Sobre o Padrão de Software Escolhido - Informações Sobre o Projeto	- Informações Sobre o Padrão de Software Adaptado ao Contexto do Projeto
4.1.1.3, 4.1.3 e 4.3. Testar Padrão de Software	- Lista de Requisitos de Teste Associados ao Padrão de Software - Lista de Requisitos de Teste Não Associados ao Padrão de Software	- Lista de Requisitos de Teste Escolhidos Para Serem Utilizados nas Atividades de VV&T - Casos de Teste Criados - Informações Sobre as Atividades de VV&T do Padrão de Software

# Uso de Padrões em Linhas de Produtos de Software: Uma Revisão Sistemática

**Luiz Alberto Ferreira Gomes<sup>1</sup>, Rosana Teresinha Vaccare Braga<sup>2</sup>**

<sup>1</sup>Curso de Ciência da Computação  
Pontifícia Universidade Católica de Minas Gerais (PUC)  
Poços de Caldas, MG – Brasil.

<sup>2</sup>Instituto de Computação e Matemática Computacional  
Universidade de São Paulo (USP) – São Carlos, SP - Brasil.

luizgomes@pucpcaldas.br, rtvb@icmc.usp.br

**Resumo.** Muitas organizações estão adotando *Linhas de Produtos de Software (LPS)* com o objetivo de aumentar sua produtividade, melhorar a qualidade de seus produtos e reduzir os custos de produção de software. Os benefícios dessa tecnologia são consideráveis e estão bem documentados na literatura. Entretanto, diversos problemas podem surgir na implantação e manutenção de uma LPS. Uma das formas para se resolver esses problemas de maneira rápida e com soluções testadas e comprovadas é a utilização de padrões. Este artigo apresenta os resultados de uma revisão sistemática que investigou o uso de padrões em LPS. A compilação desses resultados visa facilitar o uso desses padrões por desenvolvedores de LPS.

**Abstract.** Many organizations are adopting *Software Product Lines (SPL)* with the aim of increasing their productivity, improve the quality of their products and reduce production costs. The benefits of this technology are considerable and are well documented in the literature. Meanwhile, several problems may arise in the implementation and maintenance of an SPL. One way to solve these problems quickly, and with proven and tested solutions is the use of patterns. This article presents the results of a systematic review that investigated the use of patterns in SPL. The compilation of these results intends to facilitate the use of these patterns by SPL developers.

## 1. Introdução

O mercado consumidor tem exigido das empresas desenvolvedoras de software um grande esforço para entrega de produtos cada vez mais complexos, com qualidade cada vez maior e com um tempo de entrega ao cliente cada vez menor. Em razão dessas pressões, muitas organizações vêm procurando maneiras ou formas de atender às

necessidades dos usuários e clientes por meio da utilização de metodologias, métodos e ferramentas que propiciem o aumento de produtividade, a melhoria da qualidade dos seus produtos e a redução dos custos de produção de software. A tecnologia de Linhas de Produtos de Software (LPS) tem emergido como uma das soluções encontradas por muitas organizações para atingir os objetivos mencionados anteriormente (McGregor et al, 2004). Algumas definições foram encontradas para LPS, sendo que boa parte delas está baseada na definição dada por Clements e Northrop (Clements, 2001) que estabelece que:

*“Uma linha de produtos de software é um conjunto de sistemas de software que compartilham intensivamente um conjunto comum e gerenciado de funcionalidades ou características, que satisfazem a necessidades de um segmento particular de mercado ou missão, e que são desenvolvidos a partir de um conjunto de ativos comuns de maneira pré-planejada”.*

A adoção de uma LPS provoca o deslocamento do desenvolvimento de software centrado em projetos individuais para a produção de famílias de sistemas de software que, de acordo com a definição acima, tem como base ativos comuns (*core assets*). Membros dessas famílias compartilham similaridades e possuem especificidades que permitem diferenciá-los não só dentro da sua própria família, como também, no seu mercado alvo.

Um dos princípios em que se apóia uma LPS é que, na maioria das vezes, os sistemas de software não são completamente novos. Sistemas para domínios de aplicações específicos possuem mais similaridades do que especificidades. Esse fato possibilita que, através da aplicação de processos, métodos e ferramentas adequadas, seja conseguido o reúso sistemático de artefatos de software, ao invés de, como ocorre tradicionalmente, o reúso aleatório ou *ad hoc* de software.

Os benefícios trazidos pela utilização de uma LPS encontram-se bem documentados na literatura (Clements, 2006), (Cohen, 2002) e (Northrop, 2002). Entretanto, apesar desses benefícios, uma iniciativa de LPS não é algo trivial. Podem surgir problemas no seu desenvolvimento, bem como, no sua utilização. Birk et al (Birk et al., 2004) reportam, com base em um levantamento sobre as práticas utilizadas em LPS por seis organizações, que muitos obstáculos e problemas têm que ser resolvidos antes da utilização efetiva da LPS. Por exemplo, como tratar adequadamente requisitos em uma LPS, já que muitos requisitos de uma LPS são complexos, interrelacionados e divididos em requisitos comuns e requisitos específicos de um produto.

Padrões descrevem soluções para problemas recorrentes e podem ser considerados como uma forma importante de reúso. As idéias sobre padrões tiveram origem nos trabalhos desenvolvidos por Christopher

Alexander (1979) para a área da construção civil. Na área de software, um grande número padrões já foram criados e, a cada ano, novos padrões são publicados em referências ou conferências da área. Os padrões de projeto GoF (Gamma, 2005) estão entre mais populares e mais amplamente utilizados no desenvolvimento de software.

Em uma LPS, assim como em projetos de software tradicionais, muitos problemas podem se repetir, ou seja, podem ser considerados recorrentes (Northrop, 2002). A utilização de padrões pré-estabelecidos ou de novos padrões pode ajudar as organizações a encontrarem soluções comprovadas para problemas em suas LPS. Percebe-se, entretanto, que muitos padrões não foram ou não são tão divulgados como os padrões GoF. Por esse motivo, boa parte desses padrões não são bem conhecidos por muitas organizações.

O principal objetivo deste trabalho é identificar em referências, selecionadas através de um processo de revisão sistemática, os padrões convencionais e, também, novos padrões que estão sendo usados para solucionar problemas relacionados ao desenvolvimento e utilização de uma LPS. Espera-se, com isso, reunir informações em uma única publicação, para facilitar o reúso desses padrões por empresas e desenvolvedores de LPS.

O restante deste artigo está organizado da seguinte maneira. A seção 2 detalha o planejamento da revisão e como foi realizada a sua condução. A seção 3 apresenta uma visão geral dos trabalhos analisados. A seção 4 mostra uma análise e sumarização dos dados encontrados. A seção 5 apresenta a conclusão do trabalho.

## **2. Planejamento e Condução da Revisão**

A revisão sistemática apresentada neste artigo foi planejada e conduzida conforme as regras e protocolos apresentados por (Biolchini et al., 2005).

### **2.1. Objetivo**

O objetivo principal desta revisão é identificar e analisar os padrões que estão sendo utilizados pela indústria de software no desenvolvimento e utilização de uma LPS.

### **2.2. Questões de pesquisa**

Para atender aos objetivos propostos pela revisão foram elaboradas as seguintes questões que deverão ser respondidas pela revisão:

- Questão Primária: Quais padrões têm sido utilizados em LPS?  
Critério: utilização de padrões
- Questão secundária(1): Quais padrões são específicos para LPS?

Critério: proposição de padrões

- Questão secundária(2): Quais os focos dos padrões usados em uma LPS?

Critério: contexto

### **2.3. Estratégia de buscas**

- *Métodos de pesquisa*: pesquisa através da Internet utilizando de máquinas de buscas e em livros da área (restrita).
- *Fontes*: as fontes de pesquisas consideradas foram as bases eletrônicas (IEEE, ACM e Springer), as máquinas de buscas (scholar.google.com e google.com), capítulos de livros da área.
- *Língua dos trabalhos*: somente as referências escritas na língua inglesa foram consideradas.
- *Palavras-chave*: *software product line*, *software product family*, *software product families* e *pattern*.
- *String de busca*: *software AND (product line OR family OR families) AND pattern*.

### **2.4. Seleção preliminar**

O primeiro passo da seleção preliminar é a construção da *string* de pesquisa conforme estabelecido na seção 2.3. Neste ponto, qualquer customização da *string* de busca para obedecer as particularidades de cada base eletrônica ou máquina de busca deverá ser realizada. Com a *string* construída, no próximo passo, as buscas deverão ser realizadas. Para se definir a relevância ou não da referência encontrada, os resumos deverão ser lidos e, em caso de dúvida, algumas seções também deverão ser pesquisadas e lidas.

### **2.5. Seleção e Extração de resultados**

As referências selecionadas deverão ser completamente lidas e, durante essa leitura, os padrões encontrados, como também, os pontos relevantes de cada um deverão ser destacados.

### **2.6. Condução da Pesquisa**

A revisão foi realizada durante os meses de outubro e novembro de 2007. O total de trabalhos recuperados foi de 134 referências (artigos, livros e relatórios técnicos). Na seleção final, 43 referências foram identificadas como relevantes para os objetivos da revisão.

## **3. Visão Geral dos Trabalhos Analisados**

Esta seção apresenta uma síntese dos trabalhos selecionados durante o processo de revisão.

### 3.1. Padrões de Práticas

Os padrões de práticas, apresentados nesta seção, estão situados no contexto organizacional. O problema apresentado é “como uma organização pode direcionar seus esforços para implantar e institucionalizar uma LPS?”. A solução dada consiste em combinar de maneira sistemática práticas de engenharia de software, de gerenciamento técnico e de gerenciamento organizacional, bem como em coordenar os relacionamentos entre elas.

De acordo com (Clements, 2001), uma prática consiste em uma coleção de atividades que auxiliam o desenvolvimento de ativos comuns (*core assets*), o desenvolvimento do produto e o gerenciamento. Tais práticas podem ser classificadas, segundo as habilidades necessárias para executá-las, em práticas de engenharia de software, práticas de gerenciamento técnico e práticas de gerenciamento organizacional. Os dois padrões de práticas encontrados durante a revisão são discutidos a seguir:

- **Paul Clements e Linda M. Northrop** em (Clements, 2001) propõe treze padrões de práticas e mais onze variantes desses padrões para LPS. Este conjunto de padrões e variantes, segundo os autores, não devem ser aplicados de maneira isolada, já que eles possuem relacionamentos entre si e alguns englobam mais de um dos padrões de prática propostos. Um exemplo é *Adoption Factory* (Figura 1).

O padrão *Adoption Factory* mostra a ordem de aplicação e o relacionamento entre oito padrões de práticas. O objetivo desse padrão é orientar uma organização na execução de uma estratégia para a adoção de uma LPS. Uma aplicação do *Adoption Factory* em uma linha de produtos de jogos de computadores é apresentado por **Paul Clements et al.** em (Clements, 2006).

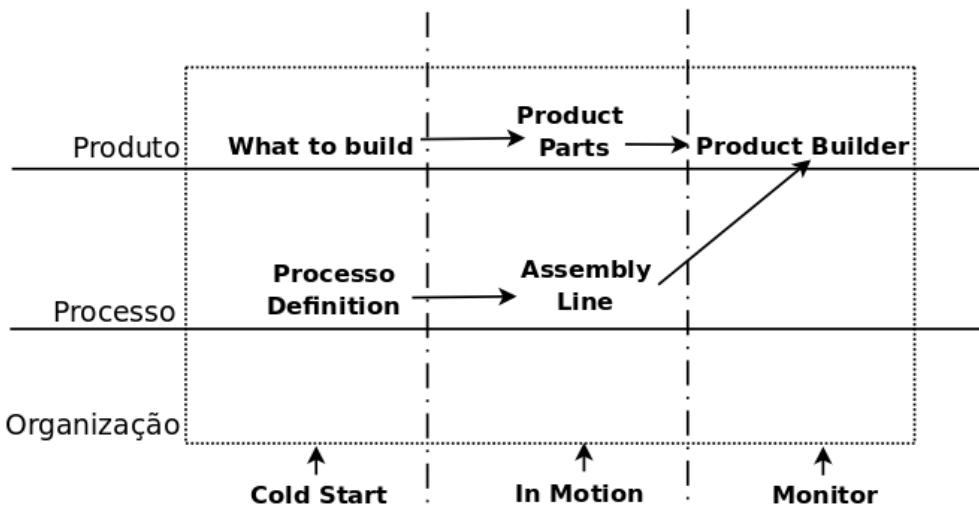
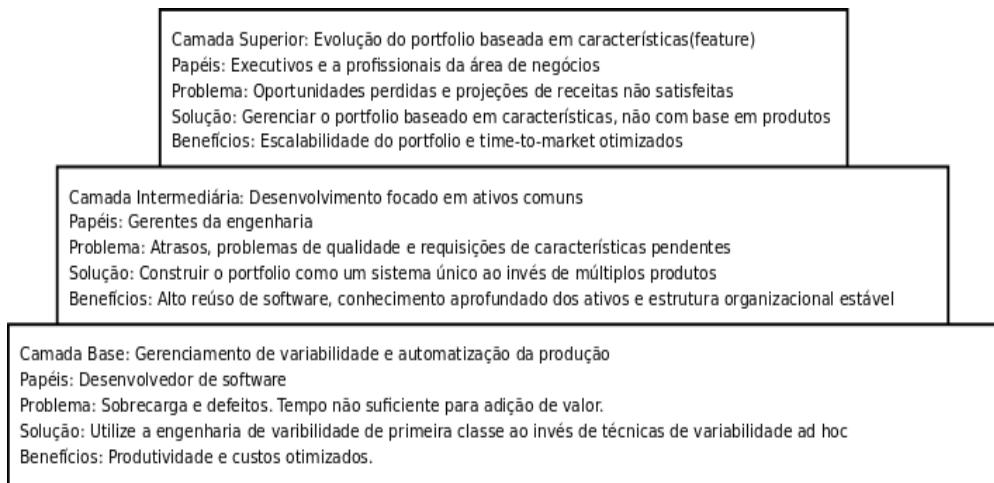


Figura 1: Padrão de Prática Adoption Factory. Adaptado de (Clements, 2006).

- Charles W. Krueger em (Kruger, 2007) apresenta um padrão para criação de uma LPS denominado de *3-Tiered*. A Figura 2 ilustra as três camadas desse padrão.



**Figura 2: Camadas do Padrão 3-Tiered. Adaptado de (Kruger, 2007).**

A camada base (gerenciamento de variabilidade e automatização da produção) têm como foco a remoção da duplicação de artefatos, a consolidação de múltiplos mecanismos *ad hoc* de gerenciamento de variabilidades e a eliminação de esforços de produção manuais e paralelos. A camada intermediária (desenvolvimento focado em ativos comuns) tem como foco a organização de ativos e de equipes de desenvolvimento em torno de componentes e subsistemas reusáveis (ativos comuns). A última camada, a camada superior, foca no gerenciamento do portfolio da linha inteira de produtos de software usando os conceitos e terminologia baseado em características do portfolio.

### 3.2. Padrões para Modelagem de Similaridades e Variabilidades

Os padrões descritos nesta seção são utilizados para modelar similaridades e variabilidades em LPS. A modelagem de similaridades e variabilidades é uma atividade importante no desenvolvimento de LPS e tem como objetivo identificar os pontos em comuns e as diferenças entre os membros de uma família de produtos.

(Keepence and Mannion, 2004) propõem um método para modelagem de similaridades e variabilidades de famílias de produtos. O método utiliza os padrões, apresentados no artigo: *Single Adapter*, *Multiple Adapter* e *Option Adapter*.

Tanto *Single Adapter* quanto o *Multiple Adapter* utilizam uma hierarquia de classes. As similaridades são modeladas nas classes base e as variabilidades nas subclasses. A diferença principal entre os dois é que no *Single Adapter* somente uma subclass pode ser instanciada e no *Multiple Adapter* muitas subclasses podem ser instâncias no sistema.

O *Option Adapter* modela características opcionais utilizando associações simples ou agregações. Como as características são opcionais em pelo menos uma das pontas a cardinalidade deverá ser 0..1.

Em (Keepence and Mannion, 2004), os autores exemplificam a utilização dos três padrões por meio da modelagem de uma LPS de sistemas de monitoração e controle de satélites.

### 3.3. Padrões Arquiteturais

Padrões arquiteturais fornecem um esqueleto ou modelo para uma arquitetura completa de software. Existem uma variedade de padrões arquiteturais amplamente utilizados em projetos de software convencionais. Esta seção relaciona os padrões arquiteturais, discutidos por **Hassan Gomaa** em (Gomaa, 2005) no contexto de LPS. O autor agrupou esses padrões em três categorias distintas, da seguinte maneira:

- **Padrões arquiteturais de estrutura:** Estes padrões definem a estrutura estática da arquitetura de uma aplicação de software. Nesta categoria estão os padrões *Layers of Abstraction*, *Kernel*, *Client/Server*, *Broker*, *Client/Agen/Server*, *Centralized Control*, *Distributed Control* e *Hierarchical Control*.
- **Padrões arquiteturais de comunicação:** Estes padrões definem a forma de comunicação dinâmica entre componentes distribuídos da arquitetura. Nesta categoria estão os padrões: *Bidirecional Asynchronous Communication*, *Synchronous Message Communication with Replay*, *Synchronous Message Communication with Callback*, *Synchronous Message Communication without Replay*, *Broker Communication*, *Discovery*, *Group Message Communication*, *Negotiated Communication*.
- **Padrões arquiteturais de transação:** Estes padrões fornecem uma arquitetura para o tratamento seguro de transações. Nesta categoria estão os padrões *Two-phase Commit Protocol*, *Compound Transaction*, *Long-Living Transaction*.

No livro, o autor destaca algumas implicações da aplicação dos padrões citados acima em LPS. Por exemplo, a aplicação do padrão *Broker* possibilita eliminação do acoplamento entre clientes e servidores, permitindo a adaptabilidade e evolução de uma LPS.

### 3.4. Padrões para Reconfiguração de Arquiteturas de Software

Os padrões arquiteturais apresentados na seção anterior são utilizados para descrever os componentes de software de uma linha de produto e a interconexão entre esses componentes. **Hassan Gomaa e Mohamed Hussein** propõem quatro padrões para reconfiguração dinâmica dessas arquiteturas. Os padrões de reconfiguração dizem como os componentes de software e interconexões podem ser alterados sob circunstâncias pré- definidas (e.g. substituição de um cliente por outro no padrão *client/server*). Os padrões de reconfiguração propostos nos artigos são baseados em uma máquina com cinco estados (*active*, *passivating*, *passive*, *quiescent*, *waiting for acknowledgement*). Somente no estado *quiescent(idle)* é que um componente pode ser removido ou substituído. Os padrões propostos em (**Gomaa and Hussein, 2004**) e (**Gomaa and Hussein, 2007**) são os seguintes:

- **Master-Slave Reconfiguration:** um componente mestre só pode ser removido ou substituído após a notificação e resposta de todos os escravos. Um componente escravo só pode ser removido ou substituído se o mestre está no estado *quiescent*.
- **Centralized Control Reconfiguration :** a substituição ou remoção de quaisquer componentes só pode ser realizada se o componente controlador estiver no estado *quiescent*.
- **Cliente/Server Reconfiguration:** tanto o cliente quanto o servidor só podem ser substituídos se já complementaram os serviços iniciados por eles e estão no estado *quiescent*.
- **Decentralized Control Reconfiguration:** os componentes enviam uma notificação aos seus vizinhos que estão indo para o estado *quiescent*. Antes do componente ser removido ou substituído, ele tem que ser desacoplado dos seus vizinhos.

### 3.5. Padrões de Reconstrução de Arquiteturas de Software

**Christoph Stoermer et al.** em (Stoermer, 2002) apresentam padrões para a reconstrução de arquiteturas. Esses padrões fornecem uma orientação de como a reconstrução de uma arquitetura pode ser realizada em um ambiente particular. Os padrões práticos descritos em (Stoermer, 2002) são os seguintes:

- **View Set:** possibilita a identificação de visões arquiteturais que descrevem de forma suficiente um sistema de software.
- **Enforce Architecture :** abrange inconsistências entre o projeto e a implementação de uma arquitetura.
- **Quality Atribute Changes :** abrange questões de como os padrões arquiteturais são usados para satisfazer aos requisitos de qualidade pré- estabelecidos.

- **Common and Variable Artifacts:** fornece modelos e ferramentas para extrair as variabilidades e similaridades de produtos existentes.
- **Binary Components:** abrange a reconstrução de arquiteturas com base nas descrições dos componentes binários.
- **Mixed-Language:** possibilita a reconstrução de arquiteturas a partir de sistemas de software escritos em linguagens diferentes.

### 3.6. Padrões de Projeto de Software

**Bryan S. Doerr e David C. Sharp** (Doerr and Sharp, 2000) utilizam os padrões de projeto Facade e Adapter (Gamma et al, 2005) para construir uma LPS. O primeiro padrão foi utilizado para a criação componentes, a partir de componentes menores, que possam ser utilizados com maior facilidade por outros componentes denominados, pelos autores, de componentes consumidores. O segundo padrão foi usado para desacoplar a interface que controlava eventos em tempo de execução dos componentes da aplicação.

**Bayer J. et. al.** (Bayer et al., 1999) apresentam um *framework* denominado RE-PLACE que tem o objetivo de auxiliar uma empresa na transição de ativos legados para uma arquitetura baseada em LPS. De acordo com os autores, padrão Observer (Gamma et al, 2005), utilizado no RE-PLACE, foi utilizado como mecanismo para permitir o tratamento de características (*features*) de objetos externos vinculados em tempo de execução.

### 4. Análise dos Dados Coletados

Os padrões encontrados durante a revisão sistemática foram agrupados em três categorias ou níveis, para facilitar sua identificação, compreensão, correlação com uma atividade ou problema de LPS, e, por fim, sua utilização. No primeiro nível, cada padrão proposto na literatura foi classificado de acordo com o seu foco principal em: organização, processo ou produto (Clements, 2001). Os padrões organizacionais têm o objetivo de orquestrar os esforços para o desenvolvimento completo de uma linha de produto. Os padrões de processos englobam práticas que estão relacionadas à capacidade da organização para a definição, melhoria e documentação de processos para uma LPS. Padrões de produtos englobam práticas necessárias para o engenheiro criar e evoluir tanto os ativos comuns quanto o próprio produto.

No segundo nível, os padrões foram classificados de acordo com as três atividades essenciais de LPS propostas em (Clements, 2001). Essas atividades, propostas pelos autores, são popularmente aceitas e se dividem em: Desenvolvimento de Ativos Comuns, também conhecida como Engenharia de Domínio; Desenvolvimento de Produtos, também

conhecida Engenharia de Produtos; e Gerenciamento (técnico ou organizacional). A atividade de gerenciamento tem como objetivo apoiar as duas outras atividades definidas por (Clements, 2001) . Por exemplo, a atividade de gerenciamento deve garantir os recursos de trabalho ou de material para a LPS. Esse agrupamento facilita a identificação de um padrão para resolver problemas específicos dentro de cada atividade essencial.

A terceira e última categoria indica se o padrão é um padrão específico , ou seja, só podem ser utilizados dentro do contexto de uma LPS ou se é um padrão clássico que foi citado como solução para problemas em LPS e, dessa forma, podem ser utilizados fora desse contexto. Em razão de não terem sido idealizados para LPS, os padrões não específicos podem exigir um esforço maior para sua utilização em linhas de produtos.

**Tabela 1: Classificação dos padrões encontrados.**

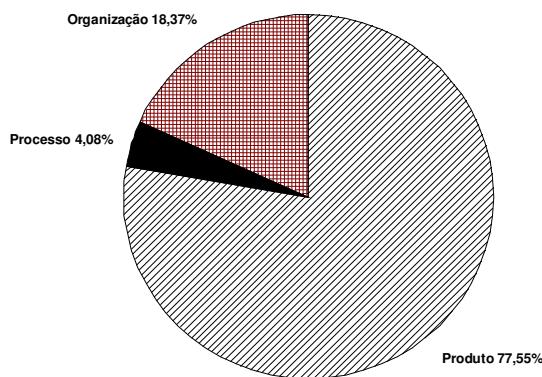
Nome do Padrão	Variante	Foco Principal	Atividade Essencial	Específico	Referência
3- Tiered	-	Organização	Gerenciamento	Sim	(Kruger, 2007)
Factory	Adoption Factory	Organização	Gerenciamento	Sim	(Clements, 2001)
In Motion	-	Organização	Gerenciamento	Sim	(Clements, 2001)
Monitor	-	Organização	Gerenciamento	Sim	(Clements, 2001)
Cold Start	Warm Start	Organização	Gerenciamento	Sim	(Clements, 2001)
Essentials Coverage	-	Organização	Gerenciamento	Sim	(Clements, 2001)
Curriculum	-	Organização	Gerenciamento	Sim	(Clements, 2001)
Assembly Line	-	Processo	Gerenciamento	Sim	(Clements, 2001)
Process	Process Improvement	Processo	Gerenciamento	Sim	(Clements, 2001)
Adapter	-	Produto	Ativos Comuns	Não	(Doerr and Sharp, 2000)
Observer	-	Produto	Ativos Comuns	Não	(Bayer et al., 1999)
Strategy	-	Produto	Ativos Comuns	Não	(Doerr and Sharp, 2000)
Option Adapter	-	Produto	Ativos Comuns	Não	(Keepence and Mannion, 2004)
Multiple Adapter	-	Produto	Ativos Comuns	Não	(Keepence and Mannion, 2004)
Façade	Façade	Produto	Ativos Comuns	Não	(Doerr and Sharp, 2000)
Composite	-	Produto	Ativos Comuns	Não	(Bayer et al., 1999)
Padrões de Reconstrução <sup>1</sup>	-	Produto	Ativos Comuns	Não	(Stoermer, 2002)
Padrões Arquitetutais <sup>2</sup>	-	Produto	Ativos Comuns	Não	(Gomaa, 2005)
Decorator	-	Produto	Ativos Comuns	Não	(Hunt and McGregor, 2007)
Single Adapter	-	Produto	Ativos Comuns	Não	(Keepence and Mannion, 2004)

<sup>1</sup> Total de 6 padrões de reconstrução.

<sup>2</sup> Total de 19 padrões de reconstrução .

What to build	Analysis Forced March	Produto	Ativos Comuns	Sim	(Clements, 2001)
Product Parts	Green Field Barren Field Plower Field	Produto	Desenv. de Produto	Sim	(Clements, 2001)
Product Builder	Product Gen	Produto	Desenv de Produto	Sim	(Clements, 2001)
Padrões de Reconfiguração <sup>3</sup>	-	Produto	Desenv.de Produtos	Não	(Gomaa and Hussein, 2004) e (Gomaa and Hussein, 2007)
Each Asset	Each Asset Apprentice Evolve Eache Asset	Produto	Gerenciamento	Sim	(Clements, 2001)

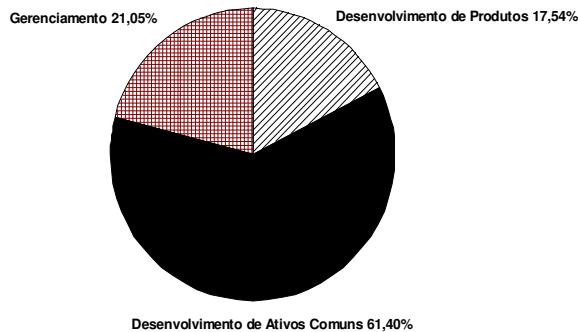
A Figura 3 mostra a porcentagem de padrões por foco principal. Percebe-se que a maioria dos padrões levantados durante a revisão tem o seu foco principal no produto de software.



**Figura 3: Distribuição de Padrões por Foco Principal.**

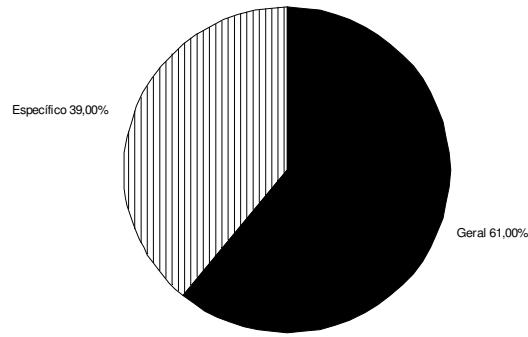
A Figura 4 apresenta a distribuição de padrões por atividades essenciais. A maior porcentagem ficou com os padrões relacionados ao desenvolvimento de ativos comuns. Percebe-se uma grande preocupação dos autores com relação aos aspectos arquiteturais de software do produto e, especificamente, de uma LPS. Tanto o gráfico da Figura 3, quanto da Figura 4 e, também, a Tabela 1 apontam essa preocupação.

<sup>3</sup> Total de 4 padrões de reconfiguração.



**Figura 4: Distribuição de Padrões por Atividades Essenciais.**

A Figura 5 apresenta a distribuição de padrões específicos e gerais. Os padrões de uso geral, não específicos para linhas de produtos, foram a grande maioria. Acredita-se, esse resultado representa o esforço natural de reutilizar padrões pré-existentes em um novo contexto.



**Figura 5: Distribuição de Padrões Específicos e Gerais.**

## 5. Conclusão

Este artigo apresentou o processo adotado para revisão sistemática para identificar padrões utilizados em linhas de produtos de software, bem como os resultados produzidos pela análise dos padrões encontrados.

A revisão foi baseada em artigos e livros selecionados a partir das ferramentas de buscas das bases eletrônicas IEEE, ACM, SPRINGER e dos sites scholar.google.com e o google.com. Com o intuito de complementar as pesquisas eletrônicas, pesquisas nas bibliotecas do ICMC-USP e da PUC Minas (campus Poços de Caldas) também foram realizadas.

Nas referências selecionadas constatou-se uma boa concentração de padrões com foco na arquitetura e no desenvolvimento de ativos comuns. Percebe-se, também, uma considerável preocupação com problemas organizacionais e gerenciais. Este fato pode ser demonstrado pelo grande número de artigos e livros que fazem algum tipo de referência aos padrões de práticas propostos por (Clements, 2001), cujos contextos são essencialmente organizacionais.

Assim, o estudo aqui apresentado permitiu identificar e classificar padrões para o desenvolvimento de linhas de produtos de software, facilitando o futuro reúso desses padrões. Permitiu também identificar algumas lacunas importante no que se refere a esses padrões, por exemplo, poucos padrões foram encontrados no contexto de modelagem de variabilidades, uma atividade, como já citado anteriormente, essencial em LPS. A ausência de padrões de testes foi outro fato interessante observado durante a revisão. Acredita-se que tais padrões possam existir na mente dos desenvolvedores, embora ainda não tenham sido propostos e escritos, o que facilitaria o uso dessas soluções por desenvolvedores menos experientes.

## 6. Referências

- (Alexander, 1979) Alexander, C. *The Timeless Way of Building*, Oxford University Press, 1979.
- (Biolchini et al., 2005) Biolchini, J., Mian, P. G., Natali, A. C. C., and Travassos, G. H. (2005). Systematic review in software engineering. Tech. Report RT-ES 679/05, Systems Engineering and Computer Science Dept., COPPE/UFRJ, Rio de Janeiro/RJ – Brazil.
- (Birk et al., 2004) Birk, A.; Heller, G.; John, I.; Schmid, K.; von der Massen, T.; Muller, K., "Product line engineering, the state of the practice" *Software, IEEE*, vol.20, no.6, pp. 52- 60, Nov.- Dec. 2003
- (Clements, 2001) Clements P, Northrop L, Northrop LM. Software Product Lines : Practices and Patterns. Addison- Wesley Professional,

2001.

- (Clements, 2006) Clements, P. C., Jones, L.G., McGregor, J.D and Northrop, L. Getting There From Here: A Roadmap for Software Product Line Adoption.
- (Cohen, 2002) Cohen, Sholom. Product Line State of the Practice Report. Disponível em <http://www.sei.cmu.edu/publications/documents/02.reports/02tn017.html>, 26/11/07.
- (Doerr and Sharp, 2000) Doerr, B. S. and Sharp, D. C. 2000. Freeing product line architectures from execution dependencies. In *Proceedings of the First Conference on Software Product Lines : Experience and Research Directions: Experience and Research Directions* (Denver, Colorado, United States). P. Donohoe, Ed. Kluwer Academic Publishers, Norwell, MA, 313- 329.
- (Gamma et al, 2005) Gamma, E., Helm, R., Johnson, R., Vlissides, J. (2005). Padrões de Projeto: soluções reutilizáveis de software orientado a objetos. Bookman: Porto Alegre, 2005.
- (Gomaa, 2005) Gomaa, H. Gomaa H. Designing Software Product Lines with UML: From Use Cases to Pattern- Based Software Architectures (The Addison- Wesley Object Technology Series). Addison- Wesley Professional, 2005.
- (Gomaa and Hussein, 2004) Gomaa, Hassan.; Hussein, Mohamed., "Software reconfiguration patterns for dynamic evolution of software architectures," *Software Architecture, 2004. WICSA 2004. Proceedings. Fourth Working IEEE/IFIP Conference on* , vol., no., pp. 79- 88, 12- 15 June 2004
- (Gomaa and Hussein, 2007) Gomaa, Hassan; Hussein, Mohamed, "Model- Based Software Design and Adaptation," *Software Engineering for Adaptive and Self- Managing Systems, 2007. ICSE Workshops SEAMS '07. International Workshop on* , vol., no., pp.7- 7, 20- 26 May 2007.
- (Hunt and McGregor, 2007) Hunt, J. M. and McGregor, J. D. 2007. When less is more: implementing optional features. In *Proceedings of the 45th Annual Southeast Regional Conference* (Winston- Salem, North Carolina, March 23 - 24, 2007). ACM- SE 45. ACM, New York, NY, 30- 35.
- (Keepence and Mannion, 2004) Keepence, B.; Mannion, M., "Using patterns to model variability in product families" *Software, IEEE* , vol.16, no.4, pp.102- 108, Jul/Aug 1999.
- (Kruger, 2007) Krueger, Charles W., "The 3- Tiered Methodology: Pragmatic Insights from New Generation Software Product Lines," *Software Product Line Conference, 2007. SPLC 2007. 11th*

- International* , vol., no., pp.97- 106, 10- 14 Sept. 2007.
- (McGregor et al, 2004) McGregor, J.D.; Northrop, L.M.; Jarrad, S.; Pohl, K., "Initiating software product lines," *Software, IEEE* , vol.19, no.4, pp.24- 27, Jul/Aug 2002.
- (Northrop, 2002) Northrop, L.M., "SEI's software product line tenets," *Software, IEEE* , vol.19, no.4, pp. 32- 40, Jul/Aug 2002.
- (Stoermer, 2002) Stoermer, C., O'Brien, L., and Verhoef, C. 2002. Practice Patterns for Architecture Reconstruction. In *Proceedings of the Ninth Working Conference on Reverse Engineering (Wcre'02)* (October 29 - November 01, 2002). WCRE. IEEE Computer Society, Washington, DC, 151.

# Experiência de Aplicação de Padrões de Software para o Cálculo de ICMS<sup>1</sup>

Francisco Wagner Costa Aquino, Jerffeson Teixeira de Souza

Grupo de Padrões de Software da UECE (GPS.UECE)

Universidade Estadual do Ceará (UECE)

Av. Paranjana, 1700 – Fortaleza – CE – Brasil

wcaquino@gmail.com, jeff@larces.uece.br

**Resumo.** O ICMS é o principal tributo do Estado, não sendo necessário mencionar a importância que ele tem para toda a população. A consolidação da arrecadação mensal do Estado gira em torno de milhões, um erro, por menor que seja, irá gerar um grande desfalque aos cofres públicos. O conjunto de regras e as possibilidades de cálculos são enormes e humanamente impossíveis de serem aplicadas sem falhas. Esse artigo apresenta de que forma os padrões de software têm sido aplicados no desenvolvimento de um sistema que visa realizar o cálculo de ICMS com corretude, aumentando a arrecadação do Estado e inibindo sonegações. Para tal, o sistema irá realizar o cálculo sem a necessidade de qualquer intervenção humana.

Palavras-chave: ICMS, padrões de projetos, *template method*, *factory method*, *transfer object*.

**Abstract.** ICMS is the main state tribute. There is no need to mention its importance to the entire population. The consolidation of monthly state revenue reaches millions, a mistake, as little as it may be, will generate a huge lost to public coffers. The set of rules and the possibilities of calculations are enormous and humanly impossible to be implemented without fails. This paper presents in which way design patterns have been applied in the development of a system to perform the ICMS calculation with precision, increasing the state revenues and inhibiting frauds. To achieve that, the system will perform the calculation without the need for any human intervention.

Keywords: ICMS, *design patterns*, *template method*, *factory method*, *transfer object*.

---

<sup>1</sup> The authors thank SEFAZ-CE for supporting this work.

Copyright © 2008, Francisco Wagner Costa Aquino, Jerffeson Teixeira de Souza. Permission is granted to copy for the SugarloafPLoP 2008 Conference. All other rights are reserved.

## 1. Introdução

O Imposto sobre Operações relativas à Circulação de Mercadorias e Sobre Prestação de Serviços de Transporte Interestadual e Intermunicipal e de Comunicação (ICMS) é um imposto de competência estadual. Ele incide sobre a circulação de mercadorias, prestações de serviços de transporte interestadual ou intermunicipal, de comunicações e de energia elétrica. Também sobre a entrada de mercadorias importadas e serviços prestados no exterior [1]. Sua regulamentação constitucional está prevista na Lei Complementar 87/1996 [2] (conhecida como “Lei Kandir”), alterada posteriormente pelas Leis Complementares 92/97, 99/99 e 102/2000 [3].

O ICMS é o maior responsável pela arrecadação dos Estados brasileiros e fica a cargo da Secretaria da Fazenda (SEFAZ) de cada Estado fazer os cálculos, realizar a cobrança e fiscalizar se a lei está sendo cumprida. Nesse artigo é abordada uma modalidade de cobrança de ICMS em especial, que é a circulação interestadual de mercadorias. Essa modalidade ocorre sempre que novos produtos cruzam a fronteira do Estado (seja de forma rodoviária, ferroviária, marítima ou aérea). Os produtos que entram no Estado devem passar por algum posto fiscal da SEFAZ. No posto é definido o valor do imposto de ICMS que deve ser pago pela mercadoria que está entrando no Estado.

Alguns produtos podem ter o cálculo de ICMS realizado de diversas maneiras, dependendo do tipo de empresa que está comprando, da finalidade da compra, do Estado de origem do produto, do preço que está sendo informado para o produto, dentre várias outras regras.

Atualmente o cálculo de ICMS está sendo realizado de maneira arcaica. O mesmo produto é cadastrado várias vezes com códigos diferentes. Cada código está associado a uma regra específica. Na escolha de qual código do produto utilizar cabe ao digitador<sup>2</sup> avaliar todas as possibilidades e utilizar a que melhor se enquadre com o caso em questão. Tal procedimento está sujeito a falhas humanas e fraudes.

Nesse artigo será apresentado um módulo para o cálculo de ICMS que está sendo implantado no novo sistema para controle de trânsito de mercadorias do Estado do Ceará (Sistema de Trânsito de Mercadoria – SITRAM). A meta principal desse módulo é retirar da competência do digitador a tarefa de escolha do cálculo que será aplicado ao produto, evitando assim dúvidas em relação a falhas humanas e idoneidade, tendo como principais consequências o aumento da arrecadação e a inibição da sonegação. Esse módulo foi implementado fazendo uso dos padrões *Factory Method* [4], *Template Method* [4] e *Transfer Object* [5].

Na seção seguinte serão exibidos alguns dados que mostram a complexidade do cálculo de ICMS, evidenciando o risco que o Estado corre mantendo o cálculo da maneira que se encontra atualmente. Na Seção 3 abordaremos todos os padrões utilizados. Na Seção 4 falaremos sobre o novo sistema que está sendo implantado e a maneira que os padrões foram utilizados no módulo de cálculo de ICMS. Por fim, na seção 5, serão apresentadas as conclusões sobre o trabalho.

---

<sup>2</sup> Funcionário responsável por realizar a digitação das notas fiscais.

## 2. O Cálculo de ICMS

O ICMS é um imposto do tipo valor adicionado<sup>3</sup>, cobrado pelo sistema de crédito fiscal [6]. Ainda em [6], o ICMS é definido como o principal tributo dos Estados e vem causando uma “guerra fiscal” entre os Estados da Federação e mais recentemente, também entre os municípios brasileiros. A importância do ICMS para o Estado é notória, com base nessa premissa vemos a necessidade que esses cálculos sejam efetuados de forma precisa, garantindo a arrecadação que é devida para o Estado.

Toda essa “guerra fiscal” seja entre os Estados, os Municípios, ou até mesmo entre os empresários que visam aumentar os lucros, geraram várias regras no cálculo de ICMS. Essas fórmulas tornaram-se complexas de tal forma que o imposto devido por um único produto pode ser calculado de diversas formas. Como exemplo, considere um simples papel higiênico em fardo de 48 pacotes que normalmente será enquadrado na rotina de cálculo padrão, porém:

- Caso o valor de venda declarado na nota fiscal seja menor que o estipulado na pauta para o produto, será utilizado na base de cálculo o valor da pauta ao invés do valor informado para o item.
- Caso o contribuinte seja Atacadista e envie suas informações fiscais referentes às operações e prestações através de meio magnético [7], será concedida uma redução na base de cálculo de valor percentual ‘x’, onde ‘x’ varia de acordo com o Estado de origem do produto.
- Caso o contribuinte possua uma Classificação Nacional das Atividades Econômicas (CNAE) de supermercado, será aplicada uma agregação de 20% na base do cálculo do produto.
- Caso o contribuinte possua uma CNAE de farmácia, será aplicada uma agregação de 28% na base do cálculo do produto.
- Caso o contribuinte possua uma CNAE de panificadora ou postos de serviços, será aplicada uma agregação de 30% na base do cálculo do produto.
- Caso seja uma empresa de segmento econômico atacadista e esteja sobre regime especial de fiscalização e controle, uma agregação de 20% será aplicada na base do cálculo do produto.
- Caso seja uma empresa de segmento econômico varejista e esteja sobre regime especial de fiscalização e controle, uma agregação de 30% será aplicada na base do cálculo do produto.
- Caso seja uma empresa de segmento econômico industrial e esteja sobre regime especial de fiscalização e controle, uma agregação de 40% será aplicada na base do cálculo do produto.

A partir desse exemplo pode-se notar a quantidade de conhecimento requerida para o cargo de digitador de notas fiscais. Vale ressaltar que existem centenas de produtos cadastrados no sistema, cada um com um leque de regras aplicáveis, dessa forma são enormes as chances de o digitador aplicar uma regra errada.

---

<sup>3</sup> Do latim *ad valorem* – A alíquota de imposto e o valor arrecadado dependem da base tributária sobre a qual incide. Essa base pode aumentar ou diminuir caso a economia esteja em expansão ou recessão respectivamente [8]

A fórmula mais simples de cálculo de ICMS é:

$$valorIcms = valorACobrar - creditoOrigem$$

Onde o *valorACobrar* é calculado através da fórmula:

$$valorACobrar = (valorItem + IPI + valorFrete + outrasDespesas) * alíquota$$

E o crédito de origem é calculado através da fórmula:

$$creditoOrigem = icmsDestacado + icmsFrete.$$

Porém, essa fórmula possui muitas variações, por exemplo:

- Todas as variáveis exibidas no cálculo (incluindo os sub-cálculos) podem ter reduções. *reducaoValorItem*, *reducaoIcmsDestacado*, *reducaoValorIcms*, etc.;
- Algumas variáveis podem ter agregações. *agregacaoBaseCalculo*;
- Ao invés de utilizar a *alíquota* pode ser utilizado um “diferencial de alíquota”, é calculado da seguinte forma:

$$alíquota = alíquotaProduto - aliquotaEstadoOrigem.$$

Outro grande problema encontrado nesse ambiente, voltando um pouco à área de TI, é a complexidade dos códigos que compõem esse algoritmo (sem mencionar a quantidade de linhas de código). Dar manutenção em um algoritmo desse porte é uma tarefa complicada e que expõe o sistema a novas chances de erros. Fato que não impede a enorme quantidade de manutenções realizadas no sistema devido ao fato de que grande parte das regras é regida por leis, decretos, convênios e protocolos. As regras possuem um alto grau de dinamicidade, podendo ter regras adicionadas, alteradas ou removidas constantemente.

### 3. Padrões Utilizados

O modulo de cálculo de ICMS que será apresentado nesse artigo utiliza os padrões *Template Method*, *Factory Method* e *Transfer Object*. Nessa seção será dada uma breve descrição de todos esses padrões.

#### 3.1. Padrão Factory Method

Segundo [9], a vantagem do *Factory Method* é que se pode retornar a mesma instância de um objeto várias vezes, ou retornar uma subclasse ao invés de um objeto do mesmo tipo. Esse padrão, também conhecido como *Virtual Constructor*, define apenas a interface do objeto que será criado, porém o objeto que será retornado dependerá da instância do *Factory* utilizado.

No diagrama da Figura 1 é apresentado um exemplo da aplicação desse padrão, exibindo com detalhes os relacionamentos entre os participantes.

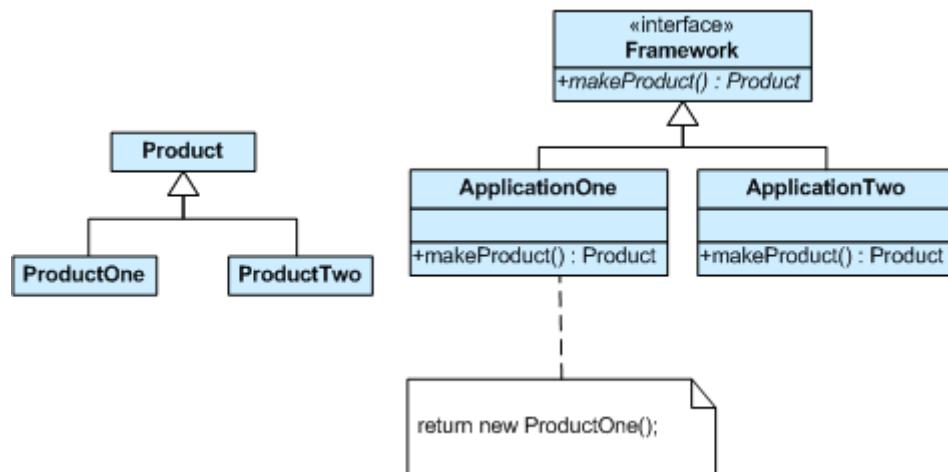


Figura 1 - Exemplo de aplicação do *Factory Method* [10]

A interface *Framework* possui o método abstrato `makeProduct()` retornando uma instância de alguma classe que estenda a interface *Product*. Ao invocar o método `framework.makeProduct()` não será possível definir em tempo de compilação qual será o produto retornado, isso dependerá de qual subclasse de *Framework* esteja instânciada no momento.

### 3.2. Padrão *Template Method*

O padrão *Template Method* é utilizado para definir a estrutura de um determinado algoritmo. Com toda a estrutura definida, alguns passos podem ser executados pelas subclasses utilizando implementações próprias.

No diagrama da Figura 2 é apresentado um exemplo da aplicação desse padrão, exibindo com detalhes os relacionamentos entre os participantes.

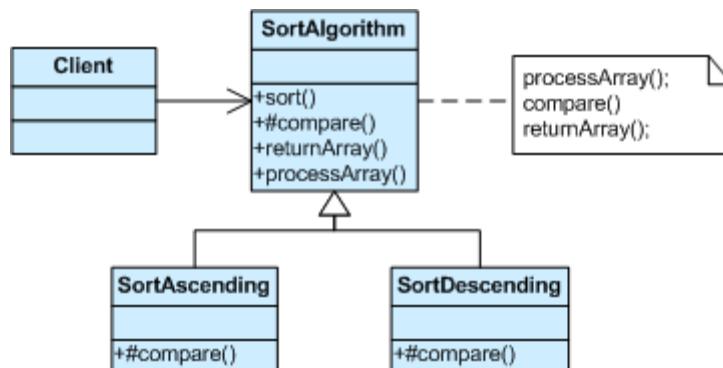


Figura 2 - Exemplo de aplicação do padrão *Template Method* [11]

A classe abstrata *SortAlgorithm* possui o método *sort()* que, em sua implementação, utiliza os métodos *processArray()*, *compare()* e *returnArray()*. Com exceção do método *compare()*, que é abstrato, todos os métodos são concretos. A implementação do método *compare()* utilizada dependerá da subclasse de *SortAlgorithm* instanciada.

### 3.3. Padrão Transfer Object (TO)

O padrão *Transfer Object*, conhecido pela sigla TO, VO (*Value Object*) ou DTO (*Data Transfer Object*) [12 e 13], é utilizado para encapsular os dados de negócio. Muito empregado quando se necessita de uma grande quantidade de dados, o uso de TO permite que todos os dados sejam obtidos com uma única requisição. Sua necessidade se mostra maior em requisições remotas.

A Figura 3 ilustra uma aplicação de TO. Nela é possível notar que um objeto *BussinessObject* é criado a partir dos dados de três outras classes (no caso, serviços) *BussinessEntity*, *BussinessSession* e *DataAccessObject*.

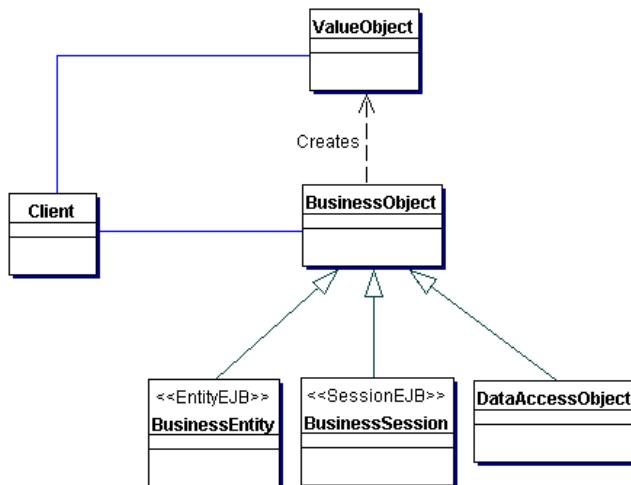


Figura 3 - Representação do padrão Transfer Object [14]

## 4. SITRAM

O SITRAM será o novo sistema da Secretaria da Fazenda do Estado do Ceará (SEFAZ-CE) para o controle da circulação interestadual de mercadorias. O sistema está sendo desenvolvido na plataforma *Java Enterprise Edition* (JEE) que segundo [15] é um ambiente de desenvolvimento em Java utilizado para aplicações corporativas.

O módulo de cálculo de ICMS foi desenvolvido após várias reuniões com os clientes da área de negócios. A cada reunião foram surgindo novas regras e fórmulas para os produtos. Para cada regra encontrada foi criada uma classe específica estendendo a classe abstrata *RegrasICMS*. Essas classes devem implementar os métodos *getTipoRegra()*, *isRegraUtilizavel()* e *ajustarValores()*, onde esses métodos têm como função:

- *getTipoRegra()*: indicar o tipo da regra instanciada no momento. Ex: *RegraPadrao*, *RegraFarmacias*, etc.

- *isRegraUtilizavel()*: indicar se o produto pode ou não ser calculado utilizando a regra instanciada no momento. Ex: Todos os produtos podem ser calculados utilizando a *RegraPadrao*; Apenas produtos enviados para contribuintes que possuam CNAE de farmácia podem ser calculados utilizando a *RegraFarmacias*.
- *ajustarValores()*: fazer os ajustes necessários nas variáveis do cálculo de acordo com a regra instanciada no momento. Ex: Na *RegraIndustrias* será aplicada uma agregação específica para indústrias na base do cálculo do produto; Na *RegraTecidos* a alíquota será substituída por uma “Carga Tributária para Tecidos e Aviamentos” e o crédito de origem será desprezado.

Mais informações acerca da estrutura das classes e seus relacionamentos podem ser observadas no diagrama de classes ilustrado na Figura 4. *ICMS* é a classe utilizada para fazer a requisição do cálculo; *RegrasIcmsFactory* e *RegrasIcmsXMLFactory* são utilizadas na aplicação do *Factory Method* (Mais detalhes na seção 4.1); *RegrasICMS*, *RegraFarmacia*, *RegraPadrao*, *RegraTecidos* e *RegraIndustrias* são utilizadas na aplicação do padrão *Template Method* (Mais detalhes na seção 4.2) e a classe *IcmsDTO* é utilizada na aplicação do padrão *Transfer Object* (seção 4.3).

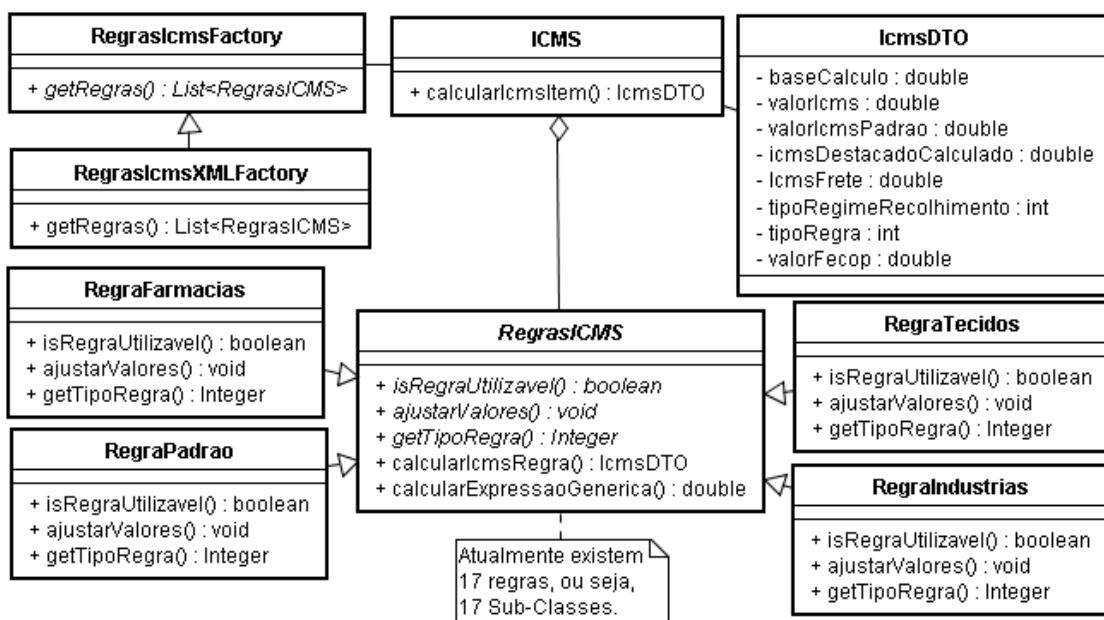


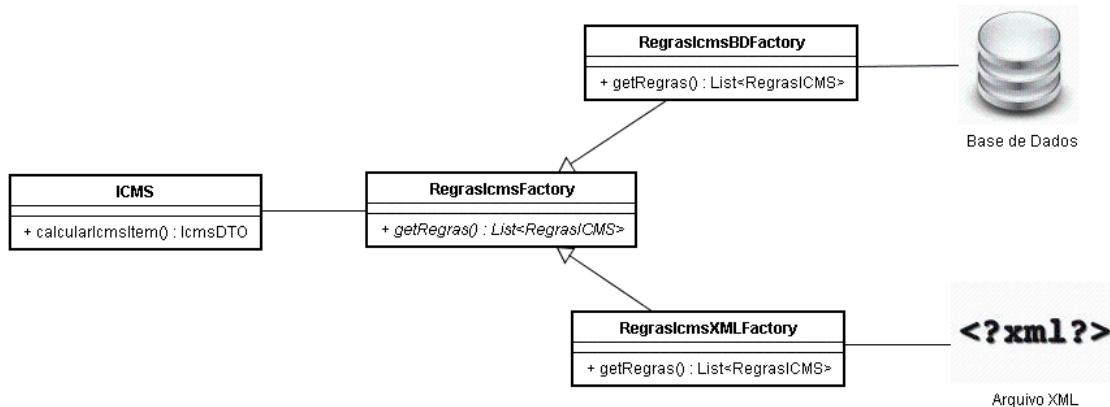
Figura 4 - Diagrama de Classes

#### 4.1. Uso do *Factory Method*

O primeiro padrão utilizado é o *Factory Method*, ele é utilizado para instanciar todas as regras que serão checadas e/ou utilizadas na realização do cálculo. A Figura 5 mostra um detalhamento do diagrama de classes exibido na Figura 4 focado no uso desse padrão. A classe *RegrasIcmsFactory* é a interface do padrão e possui o método abstrato *getRegras()* que retorna uma lista de *RegrasICMS*, ou seja, uma lista contendo as regras de cálculo de ICMS. A

implementação dessa interface é realizada pelas classes *RegrasIcmsBDFactory* e *RegrasIcmsXMLFactory* que coletam as regras a serem instanciadas em tabela na base de dados e arquivo XML respectivamente.

O uso do *Factory Method* facilita a adição/remoção de regras aumentando a escalabilidade do sistema, tendo em vista que para adicionar ou remover alguma classe (regra) no cálculo basta alterar os dados na estrutura de armazenamento que a instância do *factory* está disposta a acessar. Exemplos: editar o XML de regras quando utilizando a classe *RegrasIcmsXMLFactory* ou fazer alterações na tabela de regras quando a classe utilizada for *RegrasIcmsBDFactory*.



**Figura 5 - Uso do padrão Factory Method**

A implementação que está sendo utilizada é *RegrasIcmsXMLFactory*. Essa foi escolhida por motivos de desempenho reduzindo a quantidade de acessos à base de dados, consequentemente o fluxo na rede. O método *getRegras()* implementado tem como retorno todas as regras definidas no arquivo *regras.xml*, que é exemplificado na Figura 6.

```

<?xml version="1.0" encoding="UTF-8"?>
<regras>
    <regra id="1" nome="padrao" classe="package.RegraPadrao"/>
    <regra id="2" nome="farmacias" classe="package.RegraFarmacias"/>
    ...
    <regra id="3" nome="industrias" classe="package.RegraIndustrias"/>
    <regra id="4" nome="tecidos" classe="package.RegraTecidos"/>
</regras>
  
```

**Figura 6 - Exemplo do arquivo regras.xml**

O conjunto de regras definido na Figura 6 é utilizado para instanciar as classes correspondentes para as regras através de reflexão<sup>4</sup>, esse XML contém o nó raiz *regras* e um conjunto de nós

<sup>4</sup> Reflexão, também conhecida como RTI (*Runtime Type Information*) em algumas linguagens, é um mecanismo para descobrir dados a respeito de um programa em tempo de execução. [16]

*regra* contendo os atributos: *id* (número de identificação da regra); *nome* (nome para facilitar seu reconhecimento); e *classe* (endereço da classe Java a ser instanciada).

Outra vantagem adquirida com o uso desse padrão é a facilidade na manutenção da arquitetura do sistema, por exemplo: o trabalho necessário para trocar a forma de armazenamento das regras através de arquivo XML por uma nova tecnologia que viesse a surgir seria apenas de criar uma nova classe que implementasse a interface *RegrasIcmsFactory* (que poderia se chamar *RegrasIcmsNovaTecnologiaFactory*) e alterar a classe a ser instanciada no programa.

## 4.2. Uso do *Template Method*

O cálculo de ICMS foi generalizado o máximo possível para que fosse realizado de forma simplificada e de fácil manutenção. O primeiro passo foi criar a expressão genérica<sup>5</sup> de cálculo de ICMS. Utilizando essa fórmula é possível calcular o ICMS de qualquer produto, não importa em qual regra o item se enquadre. Na expressão criada, todas as variáveis são inicializadas de forma que não interfiram no cálculo caso as mesmas não sejam utilizadas (0 para somas e subtrações; 1 para multiplicações e divisões; 100% para variáveis que definam algum valor percentual), as variáveis que são utilizadas no cálculo devem ser ajustadas de acordo com a regra utilizada para o item em questão. É nesse ponto que é utilizado o padrão *Template Method*.

A classe *ICMS* é responsável por realizar o cálculo dos itens da nota fiscal através do método *calcularIcmsItem()*. A Figura 7 dá uma idéia de como o cálculo é realizado. Logo no início as regras são instanciadas pelo método *RegrasIcmsFactory.getRegras()*<sup>6</sup> e é feita uma iteração entre todas as regras. A primeira utilização do *Template Method* é dada nesse ponto: o algoritmo checa se a regra pode ser utilizada para o cálculo daquele item em específico (*regra.isRegraUtilizavel()*), porém a implementação dessa checagem é realizada na própria regra, ou seja, cada regra vai indicar se ela mesma pode ou não ser utilizada naquele momento.

```
public IcmsTO calcularIcmsItem(NotaFiscalItem item) {
    IcmsTO retorno = null;
    List<RegrasICMS> regras = RegrasIcmsFactory.getRegras();
    for (RegrasICMS regra: regras) {
        if (regra.isRegraUtilizavel()) {
            retorno = regra.calcularIcmsRegra(item);
        }
    }
    return retorno;
}
```

Figura 7 - Algoritmo de Cálculo de ICMS

A classe abstrata *RegrasICMS* possui o método concreto que realiza o cálculo de ICMS do item (*calcularIcmsRegra()*), a Figura 8 exibe como o cálculo é realizado.

---

<sup>5</sup> Essa expressão genérica engloba todas as variáveis utilizadas no cálculo, incluindo as variações como reduções e agregações.

<sup>6</sup> Note que a forma que as regras são coletadas é totalmente transparente para o sistema.

```

public IcmsTO calcularIcmsRegra(NotaFiscalItem item) {
    IcmsTO icmsTO = new IcmsTO();
    ajustarValores();
    icmsTO.setTipoRegra(getTipoRegra());
    icmsTO.setValorIcms(calcularExpressaoGenerica());
    return icmsTO;
}

```

Figura 8 - Algoritmo de Cálculo de ICMS por Regra

Logo no início da implementação do método *calcularIcmsRegra()* o *Template Method* é utilizado pela segunda vez, o método *ajustarValores()* não está definido na classe *RegrasICMS*, esse método é definido nas suas subclasses, ou seja, as classes com as regras definidas vão, cada uma, ajustar as variáveis do cálculo de acordo com a regra que elas representam.

A Figura 9 mostra um detalhamento do diagrama de classes geral exibido na Figura 4 focando nos pontos onde esse padrão é utilizado. Nessa figura é possível notar que as implementações dos métodos abstratos da classe abstrata *RegrasICMS* são diferentes para cada subclasse (regra) exibida na figura, exemplos: a *RegraPadrao* sempre será utilizável, mas a *RegraFarmacias* só será utilizável se o destinatário da nota fiscal possuir uma CNAE de farmácia; na *RegraTecidos* o crédito de origem será desprezado (*icmsDestacado = icmsFrete = 0*), mas na *RegraIndustrias* além do crédito de origem ser utilizado a variável *agregacaoBaseCalculo* será atribuída com o valor da agregação para base de cálculo aplicada ao contribuinte que contenham CNAE de indústria.

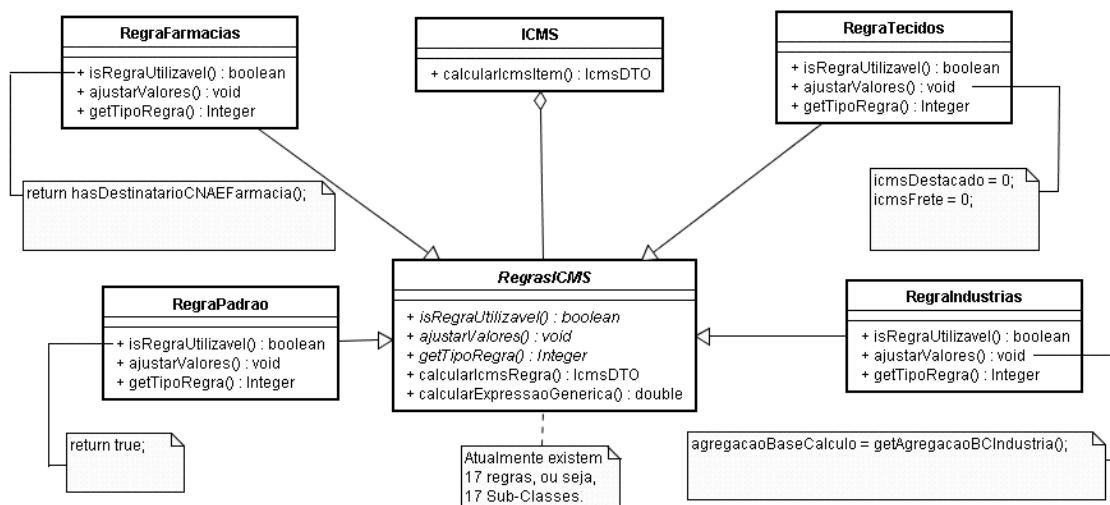


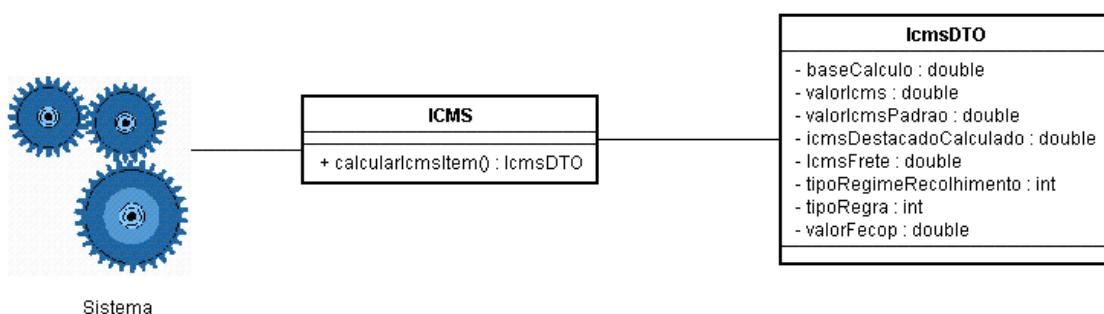
Figura 9 - Uso do padrão Template Method

O *Template Method*, além de centralizar o código evitando cópias de alguns trechos em diversos pontos do cálculo, estrutura o sistema de tal forma que cada regra pode ser tratada independentemente. A manutenção do código fica mais simples devido ao fato de cada regra possuir sua classe e nenhuma classe interferir nas demais, por exemplo: caso ocorra alguma mudança na regra padrão, não precisa pensar duas vezes para saber que o código a ser alterado estará no método *RegraPadrao.ajustarValores()*. Outro ponto que se destaca é (assim como

descrito na seção 4.1) a escalabilidade. Esse padrão permite criar novas regras para o sistema sem qualquer alteração nos códigos já escritos.

### 4.3. Uso do *Transfer Object*

Após o cálculo realizado, várias informações devem ser retornadas, dentre elas: o valor do ICMS do item, o tipo da regra que foi utilizada no cálculo, a base de cálculo que foi utilizada durante o cálculo, etc. A melhor forma encontrada para retornar todas as informações em uma única requisição é utilizando o padrão *Transfer Object*. A Figura 10 exibe como esse padrão está sendo utilizado no módulo. Note que o sistema, quando faz uma chamada ao método *ICMS.calcularIcmsItem()*, recebe como retorno uma instância da classe *IcmsDTO* contendo todos os dados que puderam ser obtidos através do algoritmo de cálculo de ICMS.



**Figura 10 - Uso do padrão Transfer Object**

O uso do *DTO* facilita no sentido de que todos os dados coletados no processo de cálculo de ICMS serão retornados em uma única requisição, caso contrário, alguns dados precisariam ser recalculados para cada requisição executada, por exemplo: A *baseCalculo* precisa ser armazenada para consultas futuras, porém esse dado também é utilizado no cálculo de ICMS. Se os dados fossem obtidos através de requisições atômicas, a *baseCalculo* seria calculada duas vezes, sendo: uma dentro da fórmula de cálculo de ICMS e outra através de um método que seria necessário, *calcularBaseCalculo()*.

## 5. Conclusões

*Factory Methods* são usualmente chamados dentro de *Template Methods* [4]. Essa mistura foi de suma importância para aumentar a escalabilidade do sistema. O *Template Method* permite que sejam criadas novas regras ou removidas regras antigas sem a necessidade de qualquer alteração no algoritmo ou nas regras já existentes; o *Factory Method* permite que essas adições ou remoções proporcionadas pelo *Template* sejam executadas sem a necessidade de alterar linha de código qualquer no sistema. O *DTO* aumenta o desempenho do sistema evitando retrabalhos ao se calcular determinados valores mais de uma vez através de várias requisições.

A principal melhoria deste novo sistema é a redução das falhas humanas. O SITRAM irá livrar o digitador da impossível tarefa de escolher como será calculado o imposto de todos os produtos que entram no estado.

O tempo médio de digitação de notas foi reduzido, pois agora cada produto possui apenas um código. Dessa forma o digitador não irá deliberar sobre qual regra será utilizada para calcular o imposto do produto, bem como não será necessário optar pelo código do produto referente à regra escolhida. Toda essa tarefa está nas mãos do SITRAM.

## 6. Referências Bibliográficas

- [1] Portal SEBRAE-SP. Acessado em 12/04/2008 na URL: [www.sebraesp.com.br/principais/melhorando%20seu%20neg%C3%B3cio/orienta%C3%A7%C3%B5es/contabilidade/legisla%C3%A7%C3%A3o/icms.aspx](http://www.sebraesp.com.br/principais/melhorando%20seu%20neg%C3%B3cio/orienta%C3%A7%C3%B5es/contabilidade/legisla%C3%A7%C3%A3o/icms.aspx).
- [2] Segundo, H., de B., M. Código Tributário Nacional, Editora Atlas, 2007.
- [3] Site Portal Tributário. Acessado em 12/04/2008 na URL: [www.portaltributario.com.br/tributos/icms.html](http://www.portaltributario.com.br/tributos/icms.html). Acessado em 12/04/2008.
- [4] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
- [5] Alur, D., Crupi, J., Malks, D. *Core J2EE Patterns – Best practices and design strategies*, Second edition, Prentice Hall, 2001.
- [6] Rossano, P., F., N. Manual de Fiscalização de Mercadorias em Trânsito na Secretaria da Fazenda no Estado do Ceará: Um Estudo de Caso – Monografia do curso de Especialização em Gestão Pública da Universidade Estadual do Ceará – UECE – 2007.
- [7] Lei 13025/2000 sobre contribuintes atacadistas. Acessado em 22/07/2008 na URL: [http://legis.sefaz.ce.gov.br/CGI-BIN/om\\_isapi.dll?clientID=104482&hitsperheading=on&infoBase=leis&record={109}&softpage=Document42](http://legis.sefaz.ce.gov.br/CGI-BIN/om_isapi.dll?clientID=104482&hitsperheading=on&infoBase=leis&record={109}&softpage=Document42)
- [8] Material de apoio da FAFIT – Faculdades Integradas do Itararé. Acessado em 13/04/2008 na url: [www.aie.edu.br/curso/apoio/apoio140920060837\\_490.doc](http://www.aie.edu.br/curso/apoio/apoio140920060837_490.doc)
- [9] Davis, Mark. Acessado em 05/04/2008 na URL: [macchiato.com/columns/Durable3.html](http://macchiato.com/columns/Durable3.html).
- [10] Descrição de *Factory Method* no *SourceMaking*. Acessado em 10/04/2008 na URL: [sourcemaking.com/design\\_patterns/factory\\_method](http://sourcemaking.com/design_patterns/factory_method).
- [11] Descrição de *Template Method* no *SourceMaking*. Acessado em 10/04/2008 na URL: [sourcemaking.com/design\\_patterns/template\\_method](http://sourcemaking.com/design_patterns/template_method).
- [12] Descrição de *DTO* na *MSDN*. Acessado em 07/04/2008 na URL: [msdn2.microsoft.com/en-us/library/ms978717.aspx](http://msdn2.microsoft.com/en-us/library/ms978717.aspx).
- [13] Fowler, Martin. *Patterns of Enterprise Application Architecture*, Addison-Wesley, 2003.

[14] Descrição de TO na Sun. Acessado em 08/04/2008 na URL: [java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html](http://java.sun.com/blueprints/corej2eepatterns/Patterns/TransferObject.html).

[15] Descrição de JEE no glossário da Sun. Acessado em 07/04/2008 na URL: [java.sun.com/javaee/reference/glossary/#88852](http://java.sun.com/javaee/reference/glossary/#88852).

[16] Definição de Reflexão. Acessado em 07/04/2008 na URL: [www.arquivodecodigos.net/arquivo/visualizar\\_dica.php?qual\\_dica=1833](http://www.arquivodecodigos.net/arquivo/visualizar_dica.php?qual_dica=1833).

# Adapting the Strategy Pattern for Micro Applications

Davi S. Pinheiro, Lincoln S. Rocha, Rossana M. C. Andrade

Federal University of Ceará - UFC  
Computer Science Department - DC

Group of Computer Networks, Software Engineering and Systems - GReat  
Fortaleza – CE – Brasil

{davi, lincoln}@great.ufc.br, rossana@ufc.br

**Abstract.** Mobile devices have several limitations, such as device memory and processing capacity, which make it difficult for software engineers to develop applications in the same way they have been doing for desktops. For example, most object-oriented systems apply design patterns to obtain good solutions for their development problems. However, for mobile device applications, called micro applications in this paper, some of these design patterns may be not appropriated. There is thus a need to develop a more optimized solution of certain design patterns for micro applications, in expense of simplicity and readability. In this paper, the Strategy Pattern is investigated in situations where it can excessively consume storage and processing resources. Then, an alternative solution with small impact in the design of micro applications is proposed.

## 1. Introduction

Mobile devices' processing and storage capacity has increased constantly, allowing the support for progressively larger applications, in size and functionalities. Since many of these applications already run in desktops, it should be possible to port them to mobile devices. However, hardware resource limitations are still present in most mobile devices, then, modifications are often necessary to allow such portability. For example, applications developed in Java Standard Edition (JSE) [9] can be adapted to mobile devices that support Java Mobile Edition (JME) [10], once JME is a JSE subset. Nevertheless, in most cases, there is a need for reducing the application requirements, increasing optimization, replacing classes that are not available, and breaking class hierarchy.

In this context, it is important to have a look at good practices in software engineering to make good decisions when developing applications for mobile devices, called micro applications in this paper. For example, most object-oriented system developers apply design patterns to obtain good solutions for their development problems. Furthermore, some of the modifications required when porting desktop applications to micro applications can also substantially change the design of applications. According to [1], "Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure". Thus, refactoring is a current practice that can be applied to port applications for mobile devices.

In this paper, refactoring is used to reach portability of a design pattern, the Strategy Pattern [3], to micro applications developed using JME. We investigate specific situations where this pattern is not feasible for micro applications and propose a new refactoring named Replace Polymorphism with Conditional to improve

performance and make possible the use of the Strategy Pattern. Furthermore, a comparison of the operational costs of the proposed solution is presented using tests in mobile devices. This comparison demonstrates that although the application code complexity increases in certain situations, it can be justified by the improvement on the response time.

The remaining of this paper is summarized as follows. Section 2 presents related works to refactoring of patterns. Section 3 gives a brief introduction to design patterns [3], presenting patterns that are necessary to understand this paper. As an example, we present a design problem of applications for reading PDF files [4], which is a file format commonly used to compact books, journal, magazines, etc. Section 3 also presents a couple of refactoring patterns, among them, the Replace Conditional With Polymorphism, which is the contrary of the refactoring proposed in this paper, called Replace Polymorphism With Conditional that is presented in Section 4. Section 5 presents a case study and the results obtained by the application of the refactoring. Finally, Section 6 concludes and presents future works.

## 2. Related Works

Although portability is a common subject in the development of micro applications, we only found one paper directly related to the migration of applications among platforms with different resources. In [7], a study of a reengineering of desktop systems for mobile devices is presented, but it focuses on the reengineering for the presentation layer, which is the main difference between mobile devices and computers, and on the identification of reusable components. This paper enters in implementation details of the reengineering. The motivation came from the crescent increase of the mobile devices' processing and storage capacity.

Another approach that we found in the literature and are also relevant to mention in this section are summarized as follows.

[6] refers to code migration of legacy applications from structured paradigm to object-oriented. It presents an approach of the use of design patterns for the reengineering of legacy code. Structured design patterns are mapped to object-oriented design patterns. There are examples of case studies taken from reengineering of legacy applications in COBOL language. In one of them, the mapping from the structured pattern Alternative Algorithms to Strategy Pattern is shown. This paper shows that the mapping related to patterns can be a solution for some applications.

There is also a converter from Java SE 1.4 to Java ME CLDC, the JDiet [12]. It has the following functionalities:

- transformation of Set and List into Vector that transforms Map into Hashtable;
- transformation of Iterator into Enumeration;
- remotion of serialization related features
  - java.io.Serializable interfaces,
  - readObject and writeObject methods.

It has also a utility class for semantic adaptation. For instance, maps Set to Vector, but the adding semantic is performed by the addSet() method of the Helper class, which avoids elements duplication in the Vector.

### 3. Design Patterns and Refactoring

In this section, we mention design patterns that are necessary for understanding this paper and give a brief introduction to refactoring.

#### 3.1 Design Patterns

The following patterns are presented: factory method, composite pattern, strategy pattern [3].

Factory Method is a useful pattern to not instantiate the objects of a class directly. It is usually used to have the control of the objects that are being instantiated at runtime in an application. Thus, this pattern avoids creating more objects than necessary, consequently, saving memory. It is also used when the type of the object being instantiated depends on a parameter. It then creates a method in a class that is responsible for object instantiation.

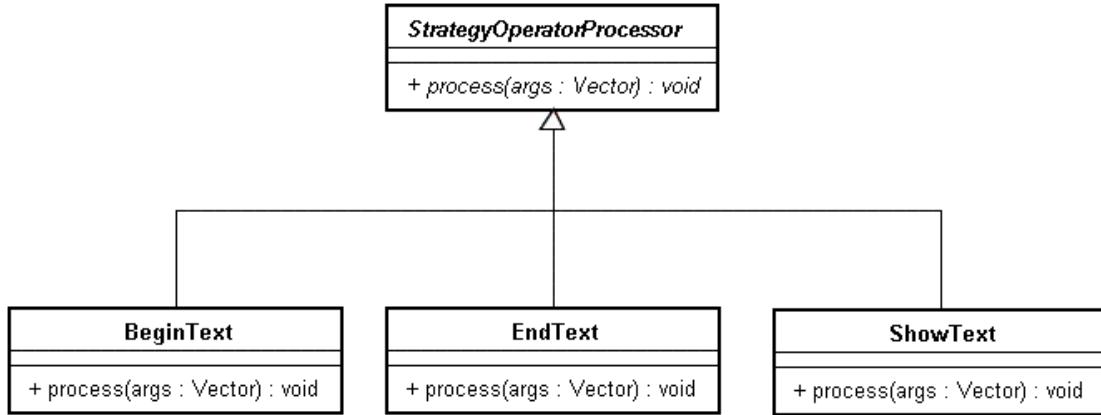
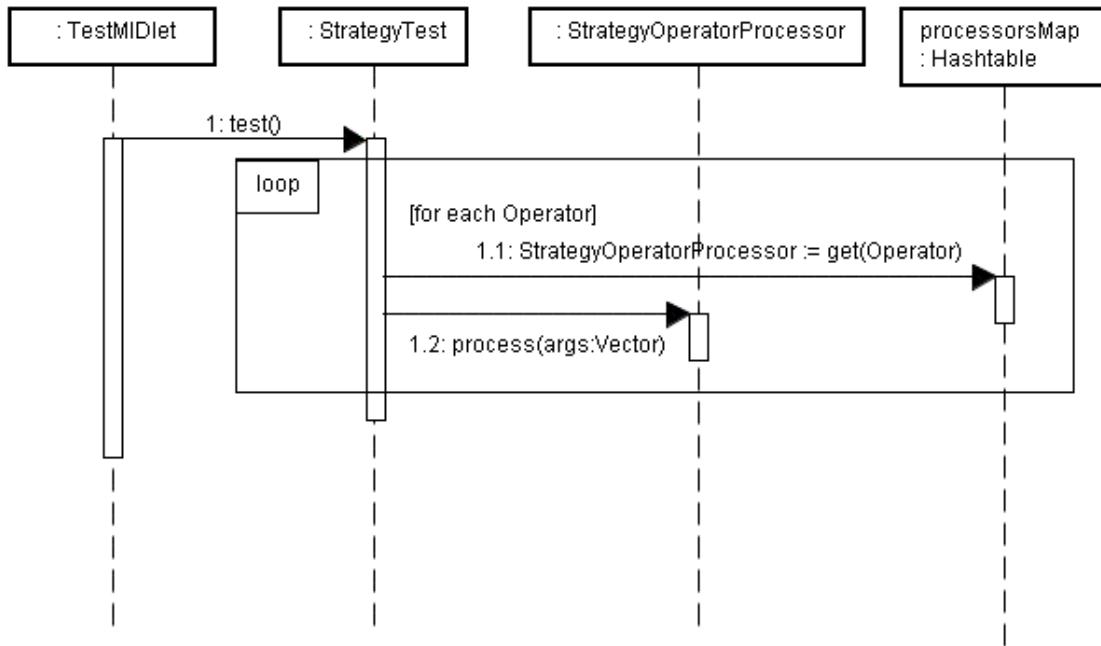
In the Composite Pattern, an object is composed of others. It allows complex objects to be composed by similar objects in a tree format.

In the Strategy Pattern, there is a class or interface that encapsulates an algorithm. It can be chosen which algorithm to execute giving its implementation. It reduces the code complexity of an application, because the use of conditional structures is avoided. In this paper, we choose the Strategy pattern to work on since it consumes many resources in micro applications as shown with an application, extracted from [4], which the use of Strategy Pattern can not be feasible when porting to mobile devices.

In accordance to [4], the pages of a PDF are organized in content stream. Each content stream contains a sequence of instructions describing graphical elements that will be displayed in the page. Each instruction includes a sequence of arguments and an operator. The notation is post-fixed, i.e., an argument followed by an operator. For instance, the texts are among the BT (begin text) and ET (end text) operators. A chain of characters that is part of the text is followed by the Tj operator. An example of content stream could be: BT 10 10 Td (Hello World) Tj ET, which 10 10 are the arguments of Td, (Hello World) is the argument of Tj. BT and ET operates without arguments.

Disregarding its operation, it can be noticed that the interface is common. A list of arguments is given to an operation. For that reason, the ideal design to process the content stream would be achieved by using the Strategy Pattern. For example, the PDFBox API [5], an open source project for PDF files reading, uses this concept. Figure 1 contains the class diagram. The StrategyProcessor class is the base of the hierarchy. Its subclasses implement each operator. BeginText processes the BT operator, EndText processes the ET, etc.

In that way, as the content stream is processed, the arguments are stored in a vector. Soon afterwards, the operator is read and the appropriate implementation of StrategyOperatorProcessor is looked for using a mapping Operator-StrategyOperatorProcessor. An Operator is a class to encapsulate the operators. In the sequence diagram of Figure 2, TestMIDlet invokes the test() method of the StrategyTest class, simulating the processing of a content stream. For each operator, the implementation responsible for it is identified and its process() method is executed.

**Figure 1. Class diagram for the solution with the Strategy Pattern****Figure 2. Sequence Diagram for the solution with the Strategy Pattern**

Unfortunately, this solution is not applicable for the great majority of the mobile devices, due to memory and processing limitations. There are a lot of necessary classes for all the PDF operators, sixty two in total, which results in the following problems:

- Each class adds an amount of bytes to the application final size. A defined class without methods or attributes contains 200 bytes approximately [8].
- Some mobile devices limit the number of classes of the applications [8].
- It may be necessary the use of reflection for class instantiation. Reflection is a mechanism for instantiating objects whose types are known only at runtime. However, it demands an extra overhead.
- There is the overhead of the polymorphism.

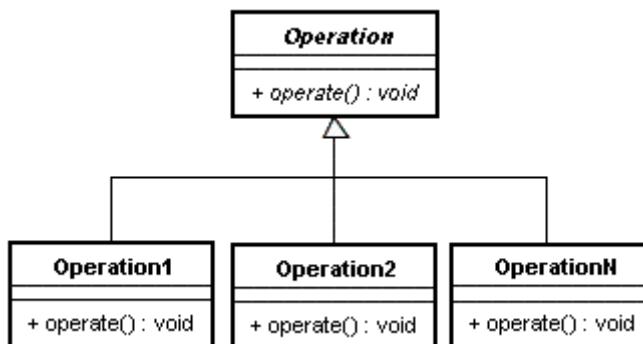
### 3.2 Refactoring

Refactoring, in its more general definition, is the "transformation preserving behavior" [2]. It is used with the objective of improving the internal structure of a system, turning

it into a more maintainable and optimized software. A couple of refactoring examples as follows:

- Extract Method - it removes code duplication by grouping the duplicated code in a method.
- Rename Method - frequently, in development cycle, a method is renamed to increase readability. All their references are then updated.
- Replace Conditional With Polymorphism - the application of this pattern happens when there is a method or code fraction that decides which operation executes based on a parameter, as shown in Table 1. Thus, for this code fraction, a class hierarchy is created and then the refactoring is applied. Those operations are called by polymorphism, and no more by conditional, which results in Strategy Pattern.

Figure 3 displays the class hierarchy resulting from the application of the pattern. Operation is the base class. For each operationI() method of Table 1 code, in which I varies from 1 to n, an OperationI class is created, subclass of Operation. The code of the operationI() method is copied to the operate() method of the OperationI class.



**Figura 3. Class hierarchy resulting from the application of the refactoring**

To overcome the increase of the application final size, the reflection usage and the polymorphism overhead, which are results from the use of the Strategy Pattern mentioned in the end of Section 3.1, the solution is to do exactly the opposite of this last refactoring, the Replace Conditional with Polymorphism as follows: starting from a class hierarchy, a code fraction is generated which decides the operation to execute.

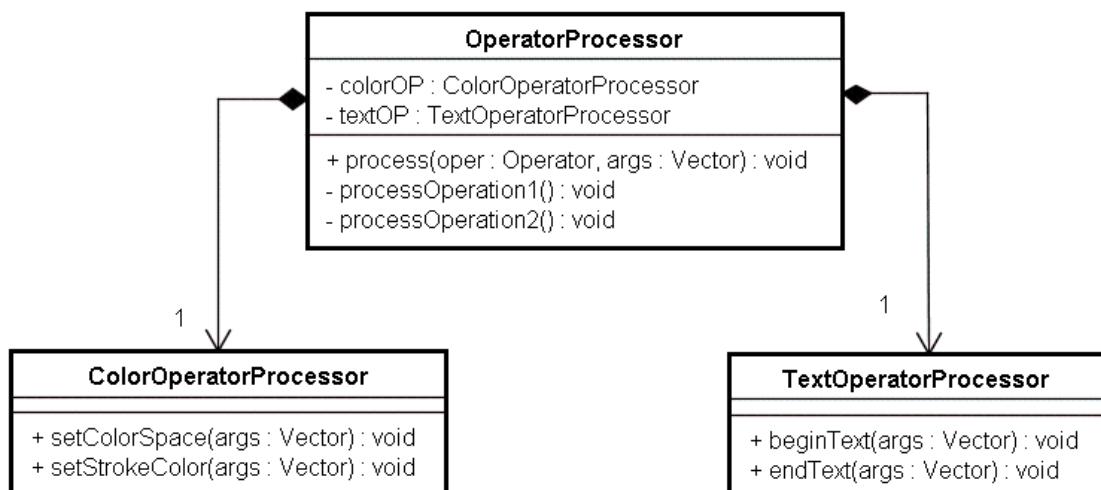
**Table 1. Structured code candidate to refactoring**

<pre> ... void execute() {     ...     int conditional;     ...     if (conditional == 1)         operation1();     else if (conditional == 2)         operation2()     ...     else if (conditional == n)         operationN(); }   </pre>
---

## 4. Replace Polymorphism With Conditional

The content stream processing example in Section 3 showed that Strategy Pattern use can reduce the performance of micro applications. Thus, we propose a refactoring named Replace Polymorphism with Conditional. It is applied in Strategy Pattern to solve the performance problems of this pattern, replacing the polymorphism for conditionals, opposite to the last pattern presented in Section 3.2. The steps for applying the proposed refactoring are presented as follows:

- 1) To turn the base class in a non abstract class. The operation, which was abstract before, will contain one more parameter that will be used to decide which operation to accomplish. This logic is the unique function of the operation. The impact for the customers of that class will be the increment of that parameter and the way of instantiation. In Figure 4 and also looking at Figure 1, we can see that the parameter operator was added and used as conditional in the operation process. The name of the class `StrategyOperatorProcessor` was renamed to `OperatorProcessor`, for convenience.
- 2) To identify which operations have relationships and to create a class that will be the compilation unit of those operations. The base class will join instances of those classes. If there are few operations, these compilation units may not be necessary. In this case, the operations would be defined in the base class itself. For example, the PDF reference [4] classifies the operators as: Text State Operators, Path Construction Operators, Path Paint Operators, Color Operators, etc. Then, it can create the `TextOperatorProcessor` classes for the text operators, `ColorOperatorProcessor` for the operators of colors, and so on.
- 3) To rename the operations that were following a common interface before, sending them through the appropriated compilation unit. Furthermore, it is necessary to eliminate the class that realized these operations before. Once again, by looking at Figure 1 and Figure 4, we can noticed that the `BeginText` class was eliminated; `TextOperatorProcessor` was created; and `TextOperatorProcessor` contains the `beginText()` method, which now will contain the code that was in the `BeginText` process(`args: Vector`) method.



**Figure 4. Class diagram of the Strategy alternative**

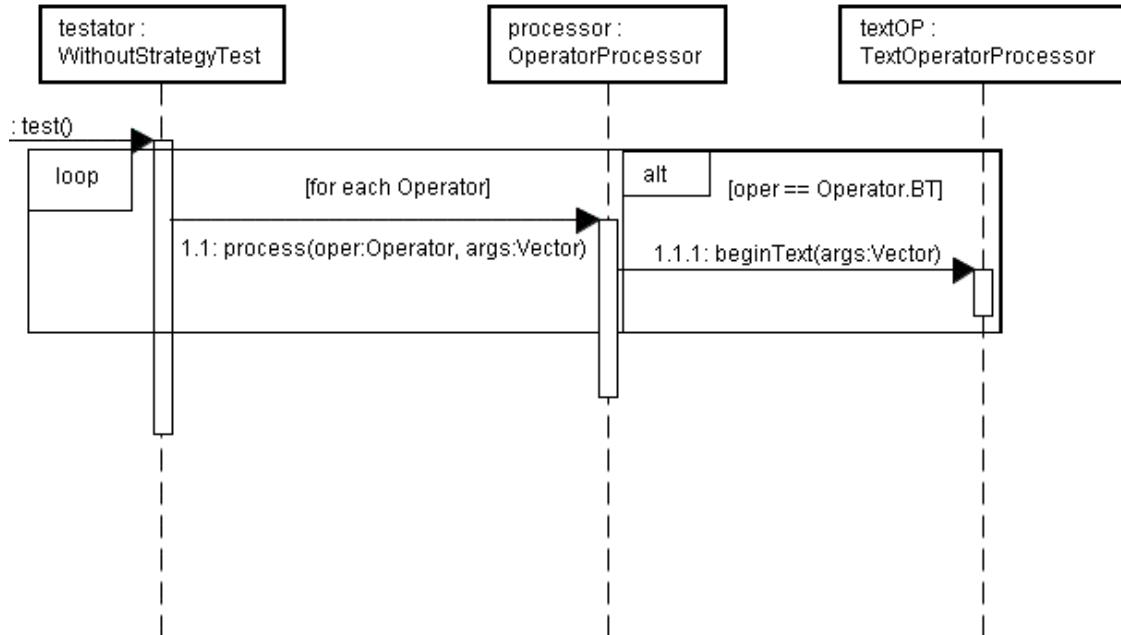
- 4) To put the conditional in the base class that executes each one of the operations. Table 2 contains the new implementation of the `OperatorProcessor`

class. In the previous related pattern, this class and the process() method were abstract, now they contain the logic to determine which operation should be executed. The sequence diagram of Figure 5 illustrates the processing of the new solution. WithoutStrategyTest contains the test() method, which simulates the processing of the content stream. For each Operator, it calls the process() method of the OperatorProcessor class, deciding which algorithm to execute through the Operator parameter.

**Table 2. New implementation of OperatorProcessor**

```
public class OperatorProcessor {
    private ColorOperatorProcessor colorOP;
    private TextOperatorProcessor textOP;

    public void process(Operator oper, Vector args) {
        if (oper == Operator.BT)
            textOP.beginText(args);
        else if (oper == Operator.ET)
            textOP.endText(args);
        ...
    }
}
```



**Figure 5. New class diagram after the changes**

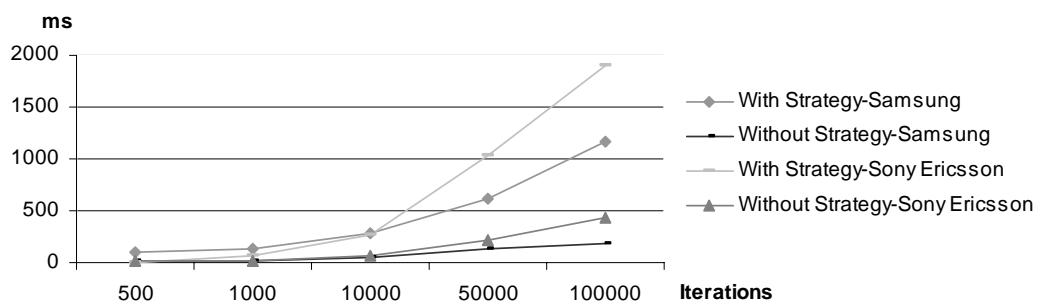
## 5. Case Study

For the case study, an application was developed to simulate both solutions presented in this paper for the processing of the content stream. First of all, we use Strategy Pattern, and, secondly, the solution proposed in this paper. The application was tested in two cellular phones, as follows: Samsung SGH-E780 and Sony Ericsson W800i. The former

has 900KB of RAM and virtual Java processor of 37,2MHz, and the latter has 1023KB and 30,8MHz, respectively. The instantiation of five classes by reflection was delayed for 94ms and 51ms, respectively. This result could be outlined with the use of a factory method [3], which could instantiate the classes through conditional. The second solution is approximately six times faster in the first device and four times in the second. The first solution with only five classes in the hierarchy, considering only the classes in subject, needed twice more memory than the second one. There is also, therefore, an evident memory economy.

Figure 6 contains the results for four different loops of iterations, processing one content stream operator for each iteration. For very small iteration values, the precision is smaller, due to the minimum resolution of clock imposed by the JTWI specification [11].

It is important to highlight that the design change is only justified when either the polymorphic method is called a lot of times or the number of classes in the hierarchy is high. In these cases, the results show that the benefits of memory and processing economy overcome the disadvantages caused by the non use of Strategy Pattern.



**Figure 6. The result in both cell phones**

## 6. Conclusions and Future Work

In this paper, we demonstrate that the use of Strategy Pattern in micro applications can reduce their performance, then, we propose a refactoring. This paper also develops an application as a case study, which is designed using Strategy Pattern at first and, then, we modify it to apply our proposed refactoring, called Replace Polymorphism with Conditional. The results showed that there is a great economy of memory and processing with the use of this refactoring. Therefore, our proposal is a good alternative in situations where the use of Strategy pattern consumes many resources and can compromise the performance of the application.

Since our focus in this paper is JME, as a future work, it is important to investigate the performance of porting desktops applications to mobile devices using C language to see the results and to compare them with the solution proposed in this paper.

Similar to the Strategy Pattern, the State Pattern may also present the same problems in micro applications. Even though the solution is not identical, it can be obtained using the same principles. Then, as a future work, the development and evaluation of an alternative for using State Pattern in micro applications can be done as

well as the identification of others possible Design Patterns that do not fit for mobile devices.

## 7. ACKNOWLEDGEMENTS

The results presented in this paper were reached thanks to a project developed within a partnership between the Samsung Institute for Informatics Development and the Computer Science Department of the Federal University of Ceará. The project was funded by Samsung Electronics under incentive of the Informatics Law number 8,248/91.

## 8. References

- [1] Fowler, Martin. *Refactoring: Improving the Design of Existing Code*. Boston, MA: Addison-Wesley, 2000.
- [2] Kerievsky, Joshua . *Refactoring to Patterns*. Addison-Wesley, 2004.
- [3] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, Mass.: Addison-Wesley, 1995.
- [4] Adobe Systems Incorporated. *PDF Reference, fifth edition: Adobe Portable Document Format version 1.6*. Adobe Press, 2004.
- [5] Java PDF Library, PDFBox. [www.pdfbox.org](http://www.pdfbox.org).
- [6] K. LANO, N. MALIK. Mapping Procedural Patterns to Object-Oriented Design Pattern. *Automated Software Engineering* 6, 3 (July 1999) 265–289.
- [7] Zhang, W., Jarzabek, S., Loughran, N., Rashid, A. Reengineering a PC-based System into the Mobile Device Product Line. In *International Workshop on Principles of Software Evolution (IWPSE'03)* (Helsinki, Finland, September 1-2, 2003). IEEE Computer Society Press, September 2003, 149-160.
- [8] Sun Developer Network. J2ME TechTips:  
<http://java.sun.com/developer/J2METechTips/2002/tt0226.html#tip1>.
- [9] Java™ Platform Standard Edition, <http://java.sun.com/javase>.
- [10] Java™ Platform Micro Edition, <http://java.sun.com/javame>.
- [11] JavaTM Technology for the Wireless Industry,  
<http://jcp.org/aboutJava/communityprocess/final/jsr185/index.html>.
- [12] Institut National de Recherche en Informatique et en Automatique. Java SE 1.4 to Java ME CLDC converter, JDiet. <http://spoon.gforge.inria.fr/JDiet/Main>.

# A proposal for Discovering Relationships among Software Patterns using Latent Semantic Analysis

## Rute Nogueira

*Group of Computer Networks, Software Engineering and Systems (GREAT,  
Department of Computer Science (DC),  
Federal University of Ceará (UFC), Brazil  
E-mail: rute@great.ufc.br*

## Jerffeson Souza

*Computer Science Department,  
State University of Ceará (UECE), Brazil  
E-mail: jeff@larcex.uece.br*

## Rossana M. C. Andrade

*Group of Computer Networks, Software Engineering and Systems (GREAT,  
Department of Computer Science (DC),  
Federal University of Ceará (UFC), Brazil  
E-mail: rossana@ufc.br*

## Abstract

Software patterns have been used in the system development process as a good practice of software engineering that promotes the reuse of good and accepted solutions for recurring problems. However, there is a lack of mechanisms to search patterns that are suitable for each system to be developed. Besides that, once these patterns are found, it is hard to detect relationships among them. In this context, this work proposes the application of a text mining technique, called latent semantics analysis (LSA), that extracts intelligent concepts from large volumes of textual information in a set of software patterns, with the goal of identifying pattern relationships.

## 1. Introduction

Software patterns come as a strategy to represent good solutions for recurrent problems, making it possible for the developer to reuse them to get better quality in the development of software and the processes [1][3][5][12][15]. Software patterns reuse has increasingly become widespread among analysts and system developers, reinforcing the relevance of this research area in software engineering.

Since the software patterns available in literature are not isolated [2][4], some related patterns appear in the *template* field of the *Related Patterns*. Besides, some pattern languages and

catalogues already have graphic representations of relationships in structures known as maps [5]. However, other relationships are not so apparent and need to be explored.

This happens because among the patterns of a same language or catalogue, the perception of existing relationships becomes easy, as they are presented in an explicit way even in the description of the pattern itself. However in the large number of patterns in literature, there is no technique yet that enables the association of one pattern to another, whether they are in a same group or not.

In this work, we propose a process to solve the problem of discovering relationships among software patterns, whether or not they are in a same group. Our proposal applies a text mining technique, called Latent Semantics Analysis (LSA), which extracts intelligent concepts from large volumes of textual information in a collection of software patterns, with the goal of identifying pattern relationships.

The documentation of a pattern is generally presented in text. So, when analyzing its content, we may identify characteristics that make possible the association of relationship between such pattern and others with related characteristics. Taking this into consideration, it is necessary to use a technique to extract valid and useful information from the software patterns' texts, in such a way that enables decisions referring to the discovery of relationships among them. Thus, a technique of Data Mining (DM) comes as a possible solution for the problem of extracting relevant information from software patterns, since its essential idea is the extraction of relationships of data from a database of texts. In DM, the Text Mining (TM) technique is used to extract concepts from volumes of textual information. In the case of software patterns, their content is treated as a textual volume with a defined structure. The defined structure of the pattern is made by the fields of its *template*.

The Text Mining technique chosen for this work is the LSA, as mentioned before, and it is, thus, applied to investigate the levels of relationship among patterns using all the fundamentation of

Data Mining, together with the rules based on the possible kinds of relationships and concepts of software patterns.

## 2. Software Patterns: Template and Relationships

The format in which patterns are documented is very important, once they must be understood in a clear and direct way by the software developers community. A good description of its fields makes the essence of the problem and its solution be immediately noticed, and it allows us to consider the consequences of its application and also provides necessary details to implement it.

The name *template* is given to the set of the composing fields of the patterns. On one hand, we can point out the heterogeneity in relation to patterns' *templates*. Each author is free to describe a solution, in the format of pattern, choosing the most suitable fields for this. On the other hand, according to [1] and [2], some of those fields are considered essential for a software pattern. Among those fields, we can highlight: pattern name, fields that describe the circumstances and situation of the problem to be solved, the problem itself, the solution itself and known uses. According to Meszaros [2], pattern documentation may be stored in distinctive fields and in different order of appearance in the pattern, depending on the style of each one, although they are the necessary and basic ideas for the documentation of a pattern.

According to the detailed study in [3], a generic *template* may be defined to group different sections that exist in literature or to document patterns, as follows: name, problem, context, forces, solution, examples, resulting context, rationale, related patterns, and known uses.

All fields in the *template* presented are used in literature to describe software patterns. We can observe that many sections of this generic *template* have relationships among themselves. For example, we can notice that exactly the same or a similar idea of a field may be represented by different names, such as *Resulting Context* and *Consequences*.

Software patterns may relate to one another, whether the patterns belong to a same group, or whether they have different origins. Different solutions for the same problem may generate different or varying patterns. Patterns can also represent an evolution of an existing pattern.

It is also common to find patterns that specialize or generate solutions for other patterns, or else, patterns can work together in languages or catalogues of patterns. In general, there are no isolated patterns: a pattern may depend on another in which it is stored, or on the parts it has, or it can be a variation of another, or a combination of others.

The previously introduced examples describe some kinds of relationships between software patterns. A lot of pattern templates bring the field Related Patterns, where the authors describe in a textual way the related patterns and the kind of relationship among them. This description may generate maps, that are graphic representations of relationships between patterns that some languages and catalogues of patterns have. Zimmer [4] organizes relationships among the project patterns of Gang of Four (GoF) [5] into categories, in order to make the understanding of the complete structure of the catalogue easier. The relationships between a pair (X,Y) of patterns were divided into three kinds described, as follows:

- X is similar to Y,
- X uses Y in its solution, and
- X can be combined with Y.

It is interesting to note that the kinds of relationship could be another useful artifice in an advanced search for patterns in a Repository or grouping of Patterns. A user could be interested in knowing, for example, which patterns “specialize” the pattern Strategy. Our proposal is an attempt to bring a solution for these searches.

### 3. Data Mining, Text Mining and Latent Semantic Analysis

Data mining is the exploration and analysis of large quantities of data in order to discover meaningful patterns (the term “pattern” used here has a similar, but somewhat different meaning to its use in “software patterns”. Here, it refers to recurring characteristics in the data which can be generalized in some kind of knowledge). Data mining is able to perform a number of tasks that represent many real-world problems, as follows. *Classification*, probably the most common data mining task, deals with the classification, categorization and grading of things into groups, called classes. While classification deals only with discrete outcomes, *estimation* deals with continuously valued outcomes. *Association rules* is the task interested in determining association among things. The classical example of association rules is discovering what products go together in a shopping cart at a supermarket. Finally, *clustering* performs the segmentation of heterogeneous populations into a number of more homogeneous groups, here named *clusters*.

The data mining process, as defined by the CRISP-DM process model [6], contains the following phases:

- **Business understanding.** This phase focuses on understanding the project objectives and converting this knowledge into a data mining problem.
- **Data understanding.** Starts with initial data collection and proceeds to identify data quality problems, to discover first insights into the data or to detect interesting subsets to form hypotheses for hidden information.
- **Data preparation.** This phase covers all activities to prepare the data for mining. Such preparation may include table, record and attribute selection as well as transformation and cleaning of data for modeling tools.
- **Modeling.** In this phase, various modeling techniques are selected and applied to the data in order to discover useful patterns.
- **Evaluation.** This deals with the thorough evaluation of the model created in the last phase.

- **Deployment.** Involves organizing and presenting the discovered knowledge in a way that the customer can use it.

Most of the previous research and development in the data mining field have focused on structured data, such as relational databases. However, the fact that most of the data electronically available today is in a non-structured format has stimulated the research on *text mining*. Text mining is a variation of data mining in the sense that it deals specifically with unstructured datasets (such as some web pages and essays) or datasets that may be somewhat structured, that is, semistructured (such as e-mail messages and XML files). A dataset containing non-structured data is often called *corpus*. Several techniques have been proposed to deal with the challenges of mining text, including the statistical approach known as Latent Semantic Analysis.

LSA is a statistical technique for extracting and representing the similarity of meanings of words, phrases or any text passages. Its underlying idea is that the aggregate of all the word contexts in which a given word appears or not in a text, provides a set of mutual constraints that largely determines the similarity of meaning of words, and sets of words, to each other [7]. LSA is a fully automated method that uses no ontology, dictionary, semantic network or any other humanly constructed knowledge base. Nonetheless, it has been shown to closely mimic human judgments of meaning, similarity and human performance based on such similarity in a variety of ways.

LSA takes as input only raw text parsed into words and separated into meaningful text passages, representing sentences, paragraphs, documents or any other text structure. The LSA analysis process is then performed as follows: first, a “word-document matrix” is constructed from the input, which stores the frequency with which each word (rows in the matrix) appears in the corresponding text passage (columns). After a preliminary transformation on the frequencies intended to highlight the importance of each word in each passage, the matrix is decomposed into the product of three smaller matrices by applying a well-known technique in matrix theory,

the Singular Value Decomposition (SVD). Such decomposition is intended to reduce the size of the original matrix. In this process, only the least significant parts of the original matrix are removed in order to minimize information loss. Next, each original document vector is replaced by a new one that excludes the terms eliminated in the previous phase. Finally, the new matrix values can be used with advanced multi-dimensional indexing techniques to determine the similarity among documents [8].

The LSA technique has been applied with significant success in several contexts. In general, LSA has been mostly used to semantically compare documents in order to find their similarity degree. Thus, [9] reports that the automatic evaluation of a student's essay by means of LSA, by comparing the student essay with essays of known quality, gives results comparable to grading an essay by a human expert. In [10], the authors report the feasibility of a LSA-based email filtering system, which is able to efficiently sort electronic mail messages into an appropriate set of folders. In addition, this technique has been applied to find relations between terms, specifically in order to deal with synonymy, where different words are used to describe the same concept, and polysemy, where the same word can have multiple meanings. As an example, in [9], the authors report that after training with a couple of thousand pages of English documents, LSA performed on the synonyms portion of TOEFL, the ETS Test of English as a Foreign Language, as well as average test-takers. Finally, by discovering terms that are semantically related, one can perform queries in a search engine and get documents that are semantically relevant to the query, even when the exact searched text does not appear directly in the searched documents. In [11], LSA is shown to significantly improve automatic information retrieval by allowing user requests to find relevant text on a desired topic even when the text contains none of the words used in the query.

## 4. Proposal for discovering relationships among software patterns using LSA

We present a process for discovering relationships among software patterns. We implemented this process using as case study 72 well-known patterns that are available in literature, known as: Gof Patterns [5], POSA1 Patterns [12], POSA2 Patterns [13], and J2EE Patterns [14].

We use LSA to compare these patterns using their template fields to find their similarity degree, which indicates the relationship among them. The method is fully automatic and does not use any previously constructed dictionaries, semantic networks, knowledge bases, conceptual hierarchies, grammatical, morphological or syntactic analysers.

The LSA process defines steps to achieve results that we use in our proposal. Additionally, we propose new steps in accordance with software patterns concepts. The first steps compose the initial phase of construction of a semantic space, and then we apply the decomposition SVD, which are essential LSA characteristics.

### 4.1 Semantic Space Construction

For this step, we use the mentioned software patterns collections, which possess the most diverse formats and classifications. Each one of these patterns first goes through preprocessing, where all the capital letters are transformed into small letters and all the characters that are not letters are deleted. After this, we select in these collections all the words that appear at least in two documents (i.e., patterns) and that do not belong to a list of words called stopwords. Stopwords are frequently occurring, insignificant words that appear in a database record, article or web page. They are not relevant for the document semantic. If we do not filter them, the computational effort to construct a semantic space will increase. After that, we construct the “word-document matrix”, which stores the frequency with which each word (rows in the matrix) appears in the corresponding text passage (columns, the patterns fields).

After preliminary transformation on the frequencies intended to highlight the importance of each word in each passage, the matrix is decomposed into the product of three smaller matrices by applying a well-known technique in matrix theory, the SVD. SVD is the general method for linear decomposition of a matrix into independent principal components of which factor analysis is the special case for square matrices with the same entities as columns and rows.

Usually, we use the SVD to find essential semantic information in a cooccurrence matrix of words. The main intent with this decomposition is to construct a semantic space which reveals the better relationships between words and the pattern template fields.

After SVD decomposition, we get a matrix Y. This matrix Y presents a collection of words in the rows, and all the fields of each template of each pattern in the columns. The returned numbers represent a degree of affinity between the words and fields of the patterns templates. So, we can perceive when a word has much similarity with one determinate field or another, therefore it will be in accordance with the degree returned in the matrix.

This will demonstrate the relationship inside the texts of those fields. From this matrix Y, we get a table with the text to text correlation indices, that is about a table with lines and columns represented for each field of each pattern of the initial collection and a degree of affinity between them. Depending on what each related field is treated there, the degrees of relationships between the patterns already can be detected from some generated rules.

## 4.2 Discovering the Relationships

We study two kinds of relationships. The first one is the relationship among pattern fields and the second is among software patterns.

The first kind of relationship defines rules and associations among the pattern template fields, so that we can implement a high level algorithm to analyse the texts semantically inside each pattern field. For example, we have a field called *Resulting Context* that sometimes presents

another problem as a result of the application of one specific pattern problem. In this case, the field *Resulting Context* may present a relationship with the field *Problem* (or Context) from another pattern. This kind of relationship deals with the essential characteristics of patterns template fields. For that, we must know which template field has a relationship with another one.

From the types of patterns relationship and also based on the knowledge of the content of each template field, we can describe an algorithm of high level where the attributes of the patterns (template fields) are similar or complementary as, for example, in a relation of cause and consequence between themselves. This type of relationship is transparent for the user of the implemented application, whose requisite is to know the related patterns and not which template fields are related with another one.

Therefore, the other level of Relationship (among patterns) is what it will be returned to the end of the textual semantic analysis

So that some rules can be implemented in order to detect consistent relationships between patterns, a deep analysis of the existing relationships between the same ones is necessary. After defining the relationship kinds and degrees, it is necessary to verify what each field is about, that is, if it is the field Problem, Solution, Aplicability and also it is necessary to know which pattern that field is from.

Considering that X and Y are software patterns, the types of existing relationships between them are:

I. X is similar to Y

When this type of relationship occurs between two patterns X and Y, it is possible that we find some relationships among the content of their template fields, for example:

- **Problem[X] and Problem[Y]**

- Forces[X] **and** Forces[Y]
- Context[X] **and** Context[Y]
- Intention[X] **and** Intention[Y]
- Abstract[X] or Description[X] **and** Abstract[Y] or Description[Y]
- Aplicability[X] **and** Forces[Y] or Problem[Y] or Context[Y] or Intention[Y]

## II. X uses Y in its solution

When this type of relationship occurs, it is possible to find relationship between some template fields, as follows:

- Solution[X] **and** Solution[Y]
- Solution[X] **and** Implementation[Y]
- Solution[X] **and** Name[Y] or As Known as[X] or Aliases[X]

## III. X can be combined with Y

When this type of relationship occurs, it is possible to find relationship between some template fields, as follows:

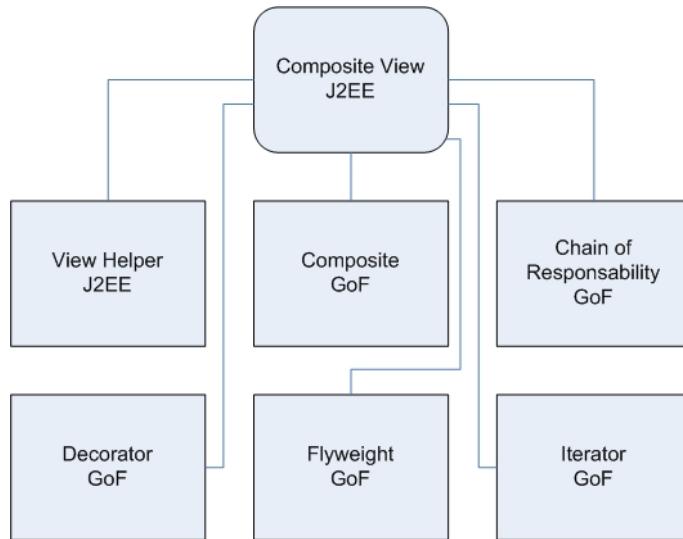
- Resulting Context[X] or Consequences[X] **and** Context[Y] or Problem[Y] or Forces[Y] or Aplicability[Y] or Intention[Y]

These relationships are in a decrescent order of relevance, the type I relationship is stronger than II that is stronger than III. This reveals that when two patterns present frequent relationships of type I, there probably is a higher affinity between they than if were only the type III relationships.

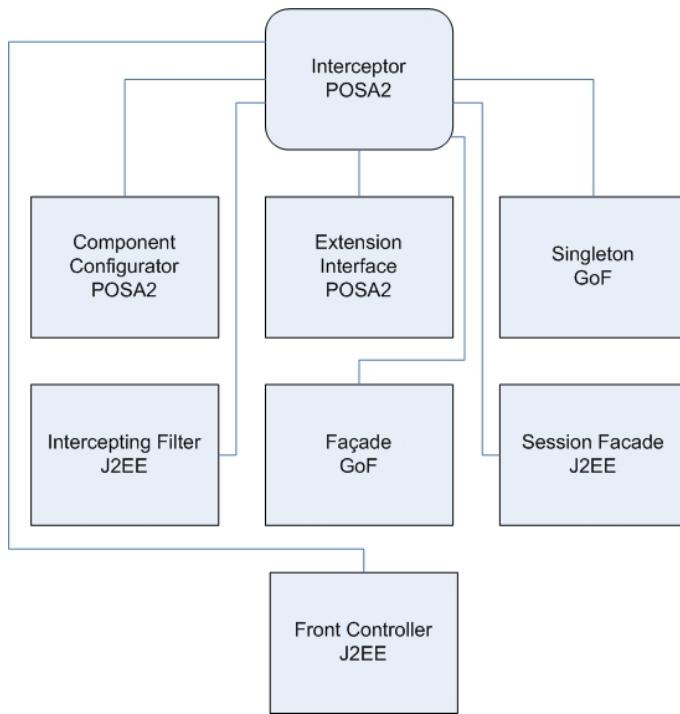
When we calculate the level of relationship among all the 72 patterns from the four famous collections, we verify in each column from a resultant table, which could be the highest numbers, that are the levels of relationships. After that, we investigate with whom these fields are related and present a high level too. If they (the template fields) are from different patterns then we have a couple of patterns X and Y that have a relationship between them. Each relationship like this detected between these two patterns increases the result of the relationship level.

If we have a large number of software patterns, it is possible to capture them in a textual format and implement the mentioned process with semantically correct results. We can confirm that by investigating the relationships among patterns from a same collection and verifying if they present the same or similar relationships as shown in their maps.

As examples, we have a pattern from each collection (GoF, J2EE, POSA1 and POSA2) related to others patterns, some of them from the same collection of the original pattern, and others from different collections as shown in Figure 1 and Figure 2. These latter relationships were not identified before, but they are presented by the implementation.



**Figure 1 – Composite View Related Patterns**



**Figure 2 - Interceptor Related Patterns**

## 5. Conclusion

We presented a process for discovering relationships among patterns using a data mining technique. We have dealt with a lot of challenges in this research related to the implementation of the whole proposal, however, the process is fully automated and we now can focus on complementary future works related to improve the rules for software patterns relationships as well as to make the proposal more friendly to be used for software developers.

Then, as future works, we still need to integrate our proposal with a Pattern Repository like the one proposed in [15], giving more flexibility to software developers to choose which pattern use.. The LSA can be also used for automatic classification, pattern summary and a more efficient search.

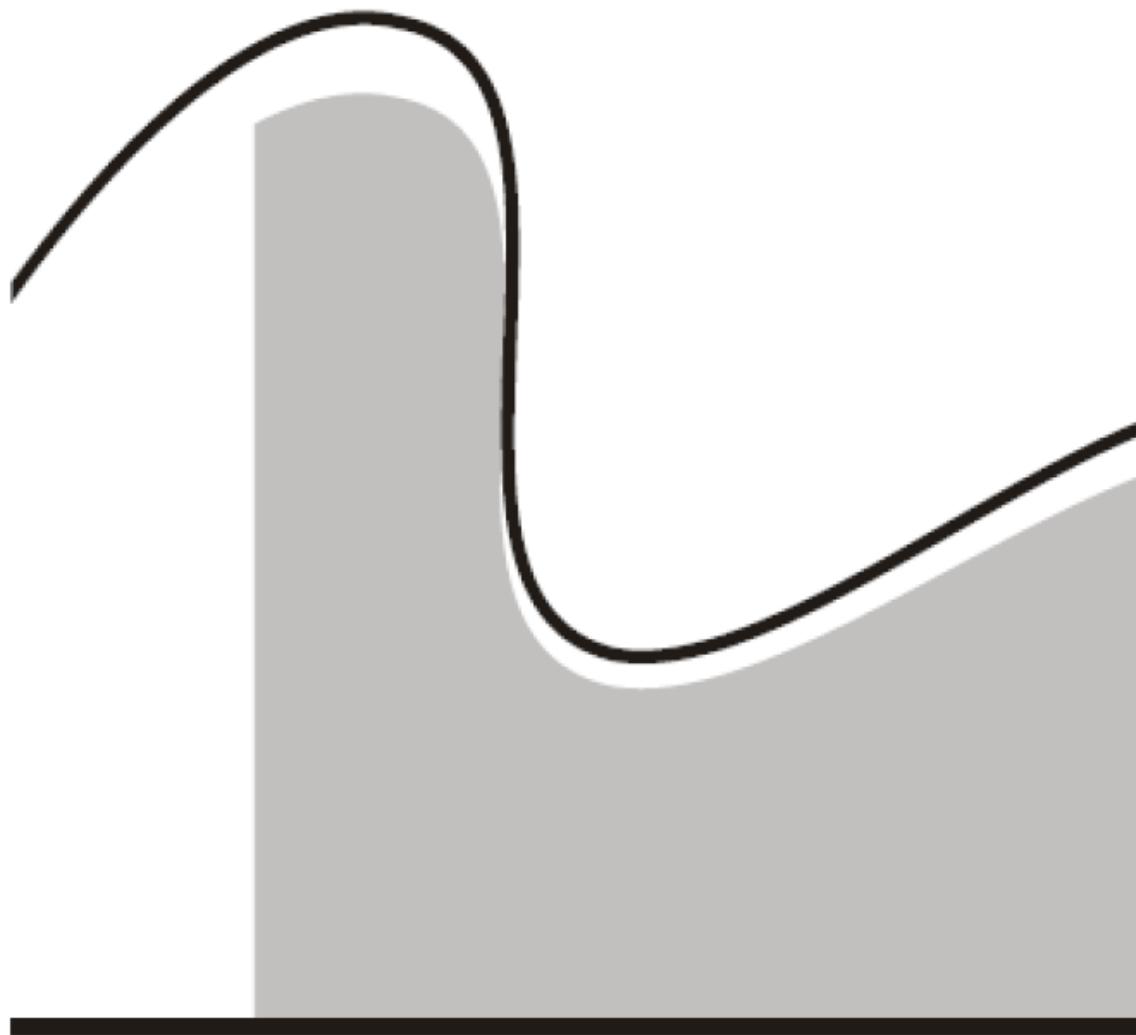
## 6. References

- [1] Andrade, R. Capture, Reuse, and Validation of Requirements and AnalysisPatterns for Mobile Systems. Ph.D. Thesis, School of Information Technology and Engineering (SITE), University of Ottawa, Ottawa, Ontario, Canada, May 2001.
- [2] Meszaros, G., Doble, J., "A Pattern Language for Pattern Writing," Pattern Language of Program Design 3, edited by Robert C. Martin, Dirk Riehle, and Frank Buschmann, Addison-Wesley (Software Patterns Series), 1997.
- [3] Santos, M. "Uma Proposta para a Integração de Modelos de Padrões de Software com Ferramentas de Apoio ao Desenvolvimento de Sistemas". Master Thesis, Master in Computer Science, Federal University of Ceará, Brazil, August 2004.
- [4] Zimmer, W. Relationships Between Design Patterns. Pattern Languages of Program Design, Addison-Wesley, 1995.
- [5] Gamma, E., Helm, R., Johnson, R., and Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Reading, MA, 1995.
- [6] Shearer, C. The CRISP-DM model: The new blueprint for data mining. Journal of Data Warehousing, Fall, 2000.
- [7] Landauer, T. K., Foltz, P. W., and Laham, D. (1998). Introduction to Latent Semantic Analysis. Discourse Processes, 25, 259-284.
- [8] Han, J. and Kamber, M. (2001). Data mining: concepts and techniques. Academic Press.
- [9] Landauer, T. K., and Dumais, S. T. (1997). A solution to Plato's problem: The Latent Semantic Analysis theory of the acquisition, induction, and representation of knowledge. Psychological Review, 104, 211-240.
- [10] Strait, M and Haynes, J. (1999). Team Readiness Assessment Program, Technical Report, Intelligent Automation, Inc.
- [11] Dumais, S. T. (1994). Latent semantic indexing (LSI) and TREC-2. In D. Harman (Ed.), National Institute of Standards and Technology Text Retrieval Conference. NIST special publication.
- [12] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Pattern-Oriented Software Architecture, John Wiley and Sons, New York, NY, 1996.
- [13] Schmidt, D., Stal, M., Rohnert, H., Buschman, F., Pattern-Oriented Software Architecture Volune 2, John Wiley and Sons, New York, NY, 2000.

- [14] Alur, D., Crupi, J., Malks, D. Core J2EE Patterns. Best Practices and Design Strategies. Prentice Hall. Second Edition, 2001.
- [15] Andrade, R.; Marinho, F.; Santos, M.; Nogueira, R. Uma Proposta de um Repositório de Padrões Integrado ao RUP. Session Pattern Application (SPA), SugarLoafPLoP 2003, The Third Latin American Conference on Pattern Languages of Programming, Porto de Galinhas, PE, August. 2003.



# SugarLoafPLoP'2008



Writing Patterns



# Conflict Manager

**Valéria Lelli Leitão<sup>1</sup>, Rute N. S. Castro e Rossana M. C. Andrade**

Universidade Federal do Ceará (UFC)  
Mestrado e Doutorado em Ciência da Computação (MDCC)  
Grupo de Redes de Computadores e Engenharia de Software e Sistemas  
(GREat)  
Fortaleza – Ceará – Brasil

`{valerialelli, rute, rossana}@great.ufc.br`

**Resumo.** *Na sociedade em que vivemos, os conflitos de idéias estão cada vez mais presentes, embora, desde a antiguidade, os homens já lutavam pelos seus interesses e as guerras comprovam tal veracidade. Ao planejar, executar e desenvolver projetos, os conflitos de interesses das partes envolvidas também aparecem. Segundo o PMBOK (Project Management Body of Knowledge) [PMI, 2000], as partes envolvidas, denominadas de stakeholders, são indivíduos e organizações diretamente relacionados no projeto, ou aqueles cujos interesses podem ser afetados, de forma positiva ou negativa, no decorrer do projeto ou mesmo após a sua conclusão. Entretanto, gerenciar idéias conflitantes entre as partes envolvidas não é uma tarefa fácil, representando um grande desafio para os Gerentes de Projetos. O objetivo deste Padrão é orientar os gerentes de projeto quando as partes envolvidas apresentarem interesses divergentes que podem afetar de alguma maneira o sucesso do projeto.*

**Abstract.** *In the society where we live the conflicts of ideas are even more present than in the ancient times where the humanity had already fought in several wars because of that. When planning, executing and developing projects there are also situations in which the conflicts of interests among the involved parties appears. According to PMBOK (Project Management Body of Knowledge) [PMI, 2000] called stakeholders, the ones involved the projects are organizations and individuals directly related to in the project, or those whose interests can be affected, in a positive or negative way, during the project or after its conclusion. It is not an easy task to manage conflicting ideas of the stakeholders, and it also represents a great challenge for Project Managers. The goal of this pattern is to guide a Project Manager when the stakeholders have some conflicts of interests that can affect in some way the success of the project.*

---

<sup>1</sup> Bolsista de mestrado (MDCC/UFC) financiada pela FUNCAP.

## 1. Introdução

Muitas pessoas acreditam que gerenciamento de projeto é uma arte e não uma ciência e segundo Scott [SCOTT, 1998], elas provavelmente estão certas. Ao empreender, desenvolver e gerenciar projetos para criar um produto ou serviço novo, geralmente, nos deparamos com interesses conflitantes.

Segundo Platão, o conflito de interesses existe desde antiguidade, ele identificou isto na Política que era praticada na Polis, antiga cidade-estado grega. Conciliar os interesses conflitantes, fazer política, foi comparado por ele a ‘arte do tecelão’, pois em sua atividade transformava a tensão entre os fios tranpostos na harmonia dos tecidos. Atualmente, na sociedade em que vivemos, os conflitos de idéias estão cada vez mais presentes e ao gerenciar um projeto onde temos partes envolvidas com diferentes interesses isso não aconteceria de forma diferente.

Entretanto, gerenciar idéias conflitantes entre as partes envolvidas não é uma tarefa fácil, representando um grande desafio para os Gerentes de Projetos. As partes envolvidas, *stakeholders*, são indivíduos e organizações diretamente envolvidos no projeto, ou aqueles cujos interesses podem ser afetados, de forma positiva ou negativa, no decorrer do projeto ou mesmo após sua conclusão; podem também exercer influências no projeto e seus resultados [PMI, 2000].

Dessa forma, com intuito de orientar os gerentes de projeto quando as partes envolvidas apresentarem interesses divergentes que podem afetar de alguma maneira o sucesso do projeto documentamos neste artigo o padrão de processo “Conflict Manager”. Segundo Scott [SCOTT, 1998], um padrão de processo de software é um padrão que descreve uma abordagem de sucesso comprovada e/ou uma série de ações para desenvolvimento de software. O padrão documentado neste artigo, apesar de utilizar alguns exemplos de projetos de software, pode ser definido para projetos em geral.

O *template* escolhido para o padrão é: Nome, Contexto, Problema, Forças, Solução, Contexto Resultante, Exemplo, Padrões Relacionados e Usos Conhecidos. Dessa forma, o *template* escolhido contém os componentes essenciais para a escrita de um padrão [MESZAROS and DOBLE, 1997]. O padrão de processo “Conflict Manager” é apresentado a seguir.

## 2. Conflict Manager

### 2.1. Contexto

Os projetos, uma vez aprovados, precisam ser bem planejados e executados. Ao final do prazo estabelecido devem gerar o produto ou serviço esperado [SOMMERVILLE, 2003]. Os *stakeholders* possuem opiniões diferentes de como desejam o produto e logo tentam influenciar no projeto para que seus interesses possam prevalecer. Dessa forma, os interesses diferentes das partes envolvidas surgirão e podem entrar em conflito durante um projeto. Essas divergências podem ocorrer em algumas situações, por exemplo:

- O cliente deseja minimizar os custos do projeto, entretanto a empresa que desenvolverá o projeto está mais interessada em maximizar os lucros, enquanto que algum membro da equipe do projeto (estagiário, programador, analista de sistemas, dentre outros) quer utilizar uma tecnologia mais avançada.
- O analista do projeto deseja desenvolver uma aplicação utilizando uma linguagem orientada a objetos Java que tenha maior portabilidade, enquanto o desenvolvedor da equipe quer continuar utilizando a linguagem que a equipe já tem experiência, ou seja, Delphi. Entretanto, essa linguagem não é tão robusta, e nem todos os usuários do produto terão computadores com capacidade para rodar essa aplicação com bom desempenho.

Entre as partes envolvidas identificamos o cliente na primeira situação e o usuário na segunda. Geralmente, cliente se refere a quem solicita o produto do projeto e usuário será quem utilizará diretamente esse produto. Dependendo da área de aplicação, os termos cliente e usuário podem ser sinônimos, mas definimos aqui que o cliente será a entidade que solicita o produto ou serviço do projeto.

Em alguns projetos, pode existir mais de um cliente, por exemplo, os patrocinadores do projeto. Nesse caso é mais difícil para o gerente do projeto gerenciar os conflitos que possam ocorrer durante o projeto.

### 2.2. Problema

Como o gerente de projeto deve gerenciar os conflitos entre as partes envolvidas, facilitando a tomada de decisões sem se desviar do curso do projeto?

## 2.3. Forças

- Chegar a um consenso entre as partes envolvidas durante o ciclo de vida do projeto aumenta a probabilidade de satisfazer os requisitos do cliente, entretanto esse consenso não é fácil de ser alcançado.
- Enfrentar e tratar os conflitos traz uma melhoria no comportamento da equipe permitindo que os integrantes da equipe do projeto se dediquem mais às atividades do próprio projeto. Porém se esses conflitos não forem logo gerenciados, atrapalhará o desenvolvimento da equipe e poderá ocorrer retrabalho.
- O gerente ao tentar gerenciar idéias muito conflitantes entre as partes envolvidas pode resultar em divergências políticas que podem conduzir a uma total improdutividade.
- Um entendimento comum do escopo do projeto entre as partes envolvidas é, algumas vezes, difícil de ser obtido e, nesse caso, pode aumentar os conflitos.
- O gerente tem que equilibrar as demandas conflitantes nas áreas de: escopo, tempo, custo, qualidade, recursos e risco para que possa produzir um produto que atenda aos requisitos do projeto conduzindo a satisfação do cliente.
- Nem sempre idéias divergentes são negativas, em algumas situações, opiniões diferentes podem ser benéficas durante o projeto, ajudando o gerente a melhorar a tomada de decisões e a desenvolver idéias mais elaboradas.

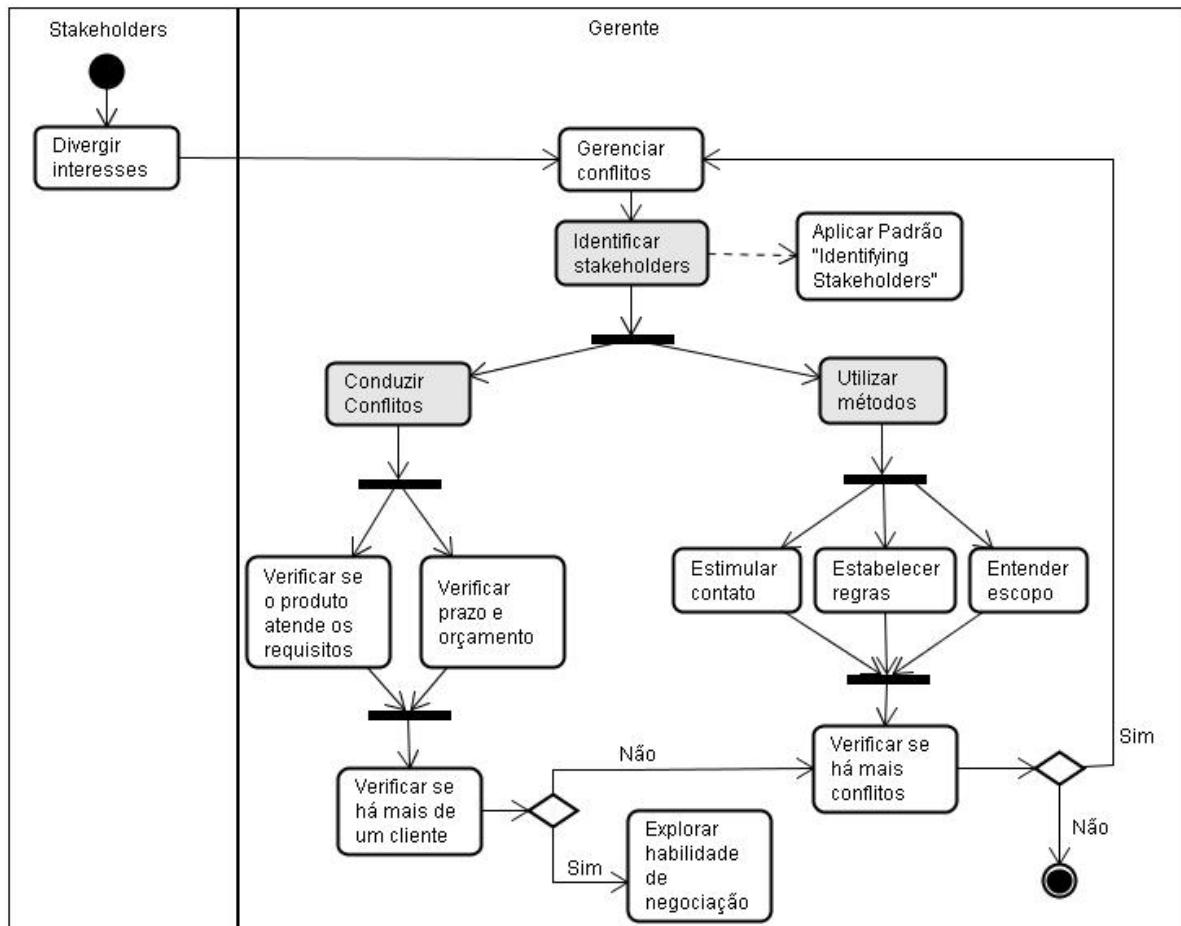
## 2.4. Solução

Um bom gerente de projeto precisa conviver com as partes conflitantes sem deixar que os interesses dos *stakeholders* possam afetar negativamente o projeto, de forma que este deixe de atender aos requisitos do cliente que solicitou o projeto. A Figura 1 apresenta um diagrama de atividades que será detalhado a seguir.

O gerente ao início do projeto deve:

1. Identificar as partes envolvidas, para que ele possa gerenciar os conflitos quando estes aparecem, identificando quais dessas partes desempenham o papel de cliente. Em alguns casos, identificar os *stakeholders* não é uma tarefa simples, pois eles podem mudar a cada etapa do projeto. Assim, se eles não forem identificados poderá haver grandes problemas para um projeto. Atualmente, existem várias técnicas para identificar *stakeholders* e futuramente será

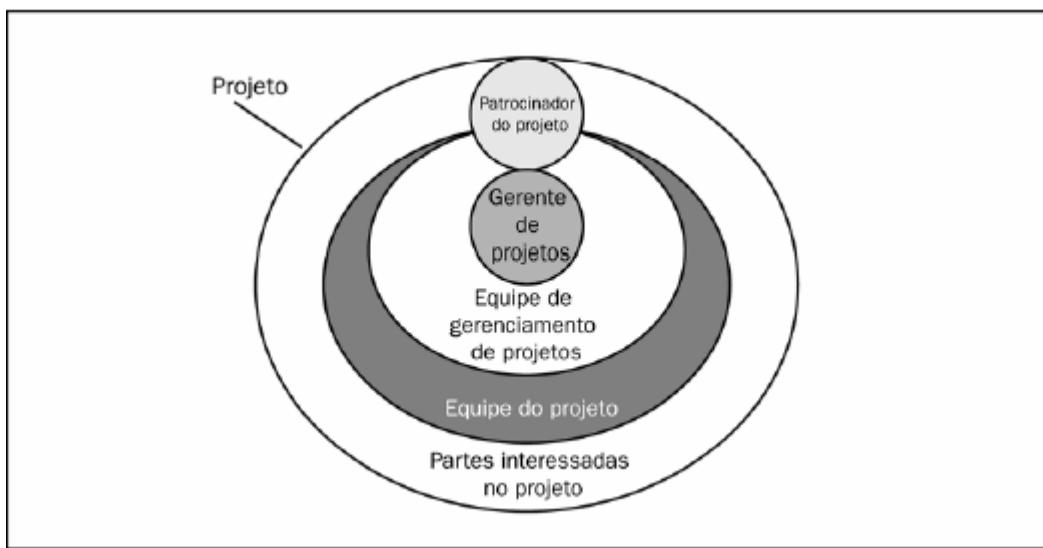
documentado em outro padrão de projeto. A Figura 2 ilustra as partes envolvidas e a equipe do projeto no PMBOK [PMI, 2000], um dos usos conhecidos, que são:



**Figura 1 – Diagrama de Atividades do padrão “Conflict Manager”**

- O patrocinador do projeto financiará o projeto;
- O gerente de projeto será responsável pela condução do projeto;
- A equipe de gerenciamento de projetos inclui o gerente de projeto e poderá também incluir outros gerentes, por exemplo, um gerente da área de negócios;
- A equipe do projeto serão os membros que trabalharão no desenvolvimento do projeto incluindo a equipe de gerenciamento de projetos, a qual o patrocinador poderá ou não pertencer;
- As partes interessadas são indivíduos e organizações diretamente relacionados no projeto, ou aqueles cujos interesses podem ser afetados, de

forma positiva ou negativa, no decorrer do projeto ou mesmo após a sua conclusão.



**Figura 2 – A relação entre as partes envolvidas e o projeto [PMI, 2004]**

2. Após o gerente identificar quem são as partes envolvidas e quais desempenham o papel de cliente (1), ele deve conduzir as situações conflitantes com foco na satisfação do cliente para garantir que o projeto esteja caminhando em direção ao sucesso. Portanto, ele deve seguir também algumas considerações:
  - Os diferentes interesses entre os *stakeholders* devem ser resolvidos em favor do cliente e de sua satisfação. O produto tem que ter as características que foram solicitadas, ou seja, o projeto tem que atender aos seus requisitos: as solicitações do cliente.
  - O gerente tem que entregar o que foi combinado dentro do prazo e orçamento previsto satisfazendo o cliente e conduzindo o projeto ao sucesso, embora as expectativas que não possam ser quantificadas, como a satisfação do cliente, tenham alto risco de não serem realizadas com sucesso [PMI, 2000].
  - Quando há mais de um cliente e suas opiniões divergem, o gerente terá que explorar sua habilidade de negociação e comunicação para que se possa chegar a um acordo comum entre essas partes envolvidas. Assim, as negociações dos conflitos entre objetivos e alternativas concorrentes têm a

finalidade de atingir ou exceder às necessidades e expectativas das partes envolvidas.

- O gerente deve ter em mente que a capacidade das partes envolvidas de influenciar as características finais do produto e o seu custo final, é alta no início e vai se reduzindo com o andamento do projeto [PMI, 2004], ou seja, há uma tendência para que as divergências entre as partes envolvidas sejam reduzidas ao longo do projeto.
3. O gerenciamento constante das partes envolvidas aumenta a probabilidade de o projeto não se desviar do seu curso por causa de interesses conflitantes. Assim, ele pode utilizar alguns métodos para melhorar o consenso entre os *stakeholders* durante o projeto, minimizando os conflitos. Os métodos que indiretamente conduzirão à satisfação do cliente são listados a seguir:
- Estimular o contato entre as partes envolvidas através de reuniões presenciais, videoconferência, e-mails, telefonemas, dentre outros. Esses são meios eficazes de comunicação, adequados para tratar os conflitos de opiniões, viabilizando um trabalho em equipe de forma ágil e auxiliando no gerenciamento de conflitos.
  - Estabelecer regras para enfrentar e lidar com conflitos. Por exemplo, o gerente pode fazer reuniões diárias com a equipe para saber o posicionamento de cada membro, ou até reuniões fora do ambiente de trabalho para melhorar as relações interpessoais. Essas atividades de desenvolvimento ajudam no crescimento e aumentam indiretamente o desempenho da equipe.
  - Envolver os *stakeholders* durante a fase de iniciação do projeto é importante para o entendimento do escopo do projeto. O gerente deve revisar os requisitos do projeto, quando o escopo não estiver bem compreendido entre as partes envolvidas. Isso ajudará a encontrar uma solução mais apropriada para gerenciar os conflitos e que atenderá aos requisitos de qualidade, aumentando a probabilidade de satisfação do cliente.

## 2.5. Contexto Resultante

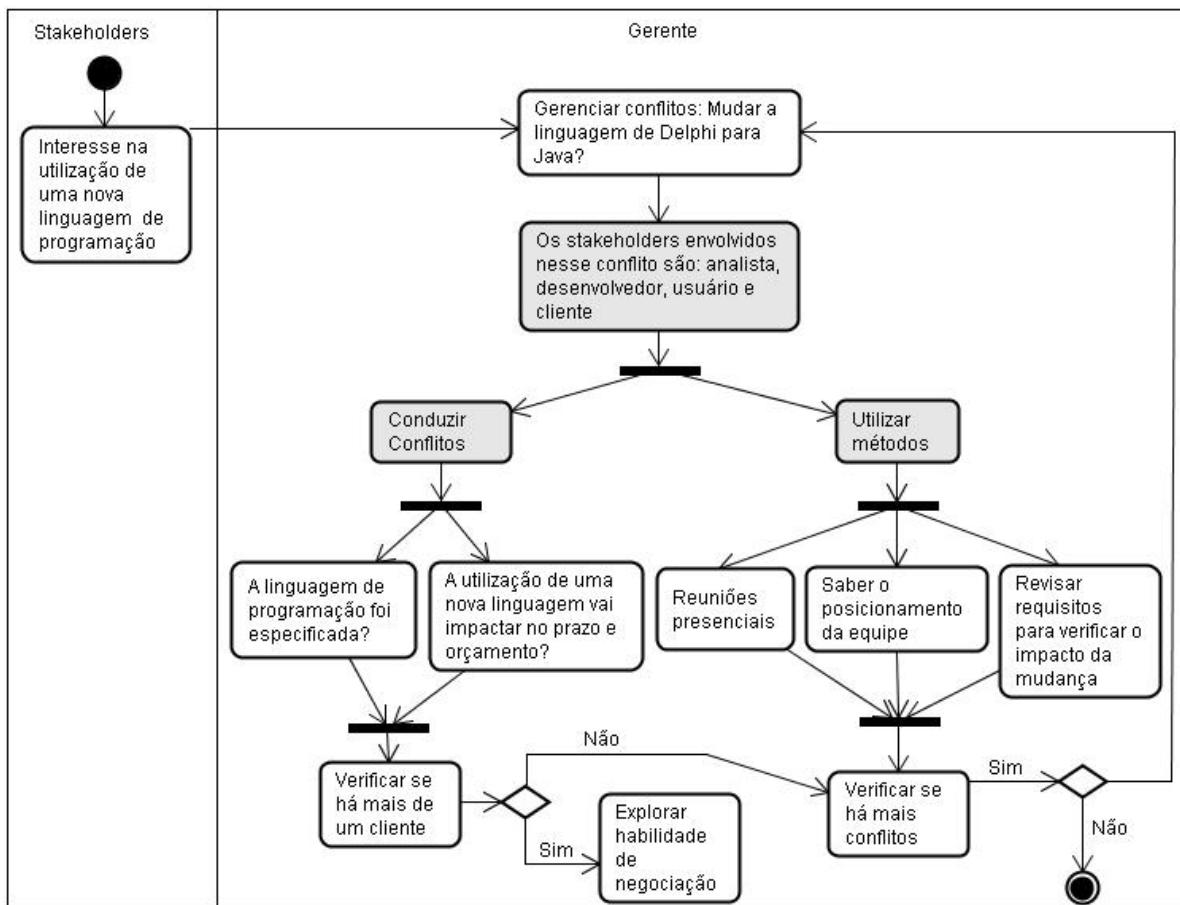
Na primeira situação apresentada no contexto (Seção 2.1), se o gerente do projeto não conduzir esses conflitos a favor do cliente, pode-se ter ao final do projeto um excelente produto, desenvolvido com tecnologia avançada, porém com custo alto para ser produzido devido às divergências apresentadas. O que teremos será um cliente insatisfeito devido ao custo alto do produto no qual usa determinada tecnologia que para ele não faz diferença.

Se o gerente levar em conta o interesse do cliente, pois é este que solicita o produto, podemos ter ao final do projeto um produto que satisfaça as especificações e os requisitos do cliente não necessariamente com uma tecnologia avançada como desejou algum membro da equipe, pois o fundamental para o cliente nesse caso é que o projeto tenha um custo baixo. Ao final do projeto teríamos um cliente satisfeito com um projeto de qualidade (produziu o que foi solicitado).

## 2.6. Exemplo

Como exemplo, podemos analisar a segunda situação apresentada na Seção 2.1 em que um analista do projeto desejava desenvolver uma aplicação utilizando a linguagem orientada a objetos Java que tem maior portabilidade, enquanto o desenvolvedor da equipe queria continuar utilizando a linguagem que a equipe já tinha experiência, ou seja, Delphi. Entretanto, essa linguagem não é tão robusta e nem todos os usuários do produto teriam computadores com capacidade para rodar essa aplicação com bom desempenho.

Nesse caso, os interesses estão divergentes entre o analista e o desenvolvedor, mas acabam impactando na necessidade do usuário e no que foi especificado pelo cliente. Logo no primeiro passo do fluxo temos a identificação desses *stakeholders*. A partir daí, temos a condução de conflitos paralela à utilização dos métodos facilitadores. Na Figura 3, apresentada a seguir, temos o problema exemplificado com a aplicação do padrão “Conflict Manager”.



**Figura 3: Exemplo de aplicação do Padrão “Conflict Manager”**

## 2.7. Padrões Relacionados

Os padrões *The Initiate Phase*, *The Define Infrastructure Stage*, *The Construct Phase* e *The Program Stage* [SCOTT, 1998]. Na solução desses padrões existem seções que tratam do gerenciamento de *stakeholders* que podem auxiliar na solução do padrão “Conflict Manager”.

## 2.8. Usos conhecidos

- **PMBOK – Project Management Body of Knowledge [PMI, 2000][PMI, 2004]**

É uma metodologia para a área de Gerência de Projetos desenvolvido pelo Project Management Institute - PMI [PMI, 2008]. Identifica o subconjunto de conhecimentos sobre a profissão que são consenso, sendo aplicáveis para a maior parte dos projetos na maior parte do tempo.

O PMBOK diz que, em geral, as divergências entre as partes envolvidas devem ser resolvidas em favor do cliente e isso não significa que as necessidades e expectativas das demais partes envolvidas devam ou possam ser desconsideradas. Dessa forma, o gerente de projeto tem que ter habilidades interpessoais como negociação e gerenciamento de conflitos.

A equipe de gerenciamento de projetos também precisa identificar as partes interessadas, determinar suas necessidades e expectativas e, na medida do possível, gerenciar sua influência em relação aos requisitos para garantir um projeto bem-sucedido. Para aprimorar o desempenho do projeto o gerente de projeto deve desenvolver a equipe do projeto, melhorar as competências e a interação de membros da equipe.

A 3º edição do PMBOK lançada em outubro de 2004 [PMI, 2004] dá mais ênfase na parte de gerenciamento de conflitos.

- **CMMi – Capability Maturity Model Integration [CMMi, 2003]**

É uma evolução do CMM [CMM, 2008] e procura estabelecer um único modelo para o processo de melhoria corporativo, integrando diferentes modelos e disciplinas. O CMMI foi criado pelo SEI [SEI, 2008] - Software Engineering Institute e é reconhecido mundialmente por atestar a maturidade dos processos de desenvolvimento da organização.

No CMMi, a resolução de conflitos envolve: encontrar as prioridades relativas para os objetivos (e.g. requisitos, qualidade do cliente, discussões com o cliente e clientes potenciais, objetivos dos negócios); identificar e envolver o cliente, os usuários finais, o gerenciamento sênior, o gerente de projeto e outros *stakeholders* relevantes (*stakeholder* que é identificado para envolvimento em atividades específicas) nas decisões de *tradeoffs* e revisar os objetivos quando necessário para refletir os resultados na resolução de conflitos.

- **ISO 10006 – Quality management; Guidelines to quality in project management [ISO 10006, 2003]**

É um padrão internacional desenvolvido pela ISO (International Organization for Standardization) [ISO, 2008], específico para gerência de projetos.

Segundo Stanleigh [STANLEIGH, 2008], seu objetivo geral é criar e manter a qualidade em projetos através de um processo sistemático que garanta:

- As necessidades explícitas e implícitas dos clientes sejam entendidas e atingidas.
- As necessidades dos *stakeholders* sejam entendidas e avaliadas.

De acordo com a ISO 10006, o foco no cliente é essencial para o sucesso do projeto, pois as organizações dependem de seus clientes e, portanto, devem entender as necessidades atuais e futuras dos clientes, devem atender os requisitos dos clientes e se esforçarem para superar as expectativas dos clientes. A qualidade e o sucesso de um projeto também dependem da participação do pessoal, o que inclui o desenvolvimento da equipe por parte do gerente de projeto.

### 3. Agradecimentos

Agradecemos a Maria Lencastre (DSC-UPE, Brasil) pela contribuição na fase inicial deste trabalho. Ao enviarmos este padrão, esperamos melhorias e refinamentos para uma posterior submissão na seção de Writers' Workshop (WW).

### 4. Referências Bibliográficas

- CMM (2008). **Capability Maturity Model**. Disponível em: <<http://www.sei.cmu.edu/cmm>> Acesso em: 30 abr. 2008.
- CMMI (2003). Chrissis, M. B.; Konrad, M.; Shrum, S. **CMMI – Guidelines for Process Integration and Product Improvement**. 2 ed. Boston: Addison-Wesley, 2003.
- ISO (2008). **International Organization for Standardization**. Disponível em: <<http://www.iso.org>> Acesso em: 30 abr. 2008.
- ISO 10006 (2003) – **Quality management – Guidelines to quality in project management**. 2003.
- MESZAROS, Gerard; DOBLE, Jim (1997). **A Pattern Language for Pattern Writing**. Pattern languages of program design 3. Boston: Addison-Wesley Longman Publishing Co., Inc., 1997. Disponível em: < <http://www.hillside.net/patterns/writing/patterns.htm#B.1>>. Acesso em: 30 abr. 2008.
- PMI (2008). **PMI, Project Management Institute – Making project management indispensable for business results**. Disponível em: <<http://www.pmi.org>>. Acesso em: 09 jul. 2008.

PMI (2000). PMI, Project Management Institute, Inc. **Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos – Guia PMBOK** (Project Management Body of Knowledge). Tradução por PMI. 2ed. EUA: 2000.

PMI (2004). PMI, Project Management Institute, Inc. **Um Guia do Conjunto de Conhecimentos em Gerenciamento de Projetos – Guia PMBOK** (Project Management Body of Knowledge). Tradução por PMI. 3ed. EUA: 2004.

SEI (2008). **Software Engineering Institute**. Disponível em: <<http://www.sei.cmu.edu>>. Acesso em: 30 abr. 2008.

SCOTT, W. Ambler (1998). **Process Patterns: Building Large-Scale Systems Using Object Technology**. Nova York, NY: SIGS Books/Cambridge University Press, 1998.

SOMMERVILLE, Ian (2003). **Engenharia de Software**. Tradução de André Maurício de Andrade Ribeiro. 6. ed. São Paulo: Addison-Wesley, 2003.

STANLEIGH, Michael. "Combining the ISO 10006 and PMBOK to Ensure Successful Projects". Traduzido e adaptado à Terceira Edição do PMBOK por Katia P. Thomaz, PMP. Disponível em: <<http://www.pmimg.org.br/artigos/Combinando10006EPMBOK.pdf>>. Acesso em: 09 jul. 2008.