CENTRO UNIVERSITÁRIO DA FEI

ANDREY ARAUJO MASIERO

fAGNER CRISTIANO DONADON

GUILHERME WACHS LOPES

MATHEUS HENRIQUE PEREIRA GONÇALVES

TOM MIX MARTINI PETRECA

SIGEpaPP: patterns e anti-patterns associados à personas

ANDREY ARAUJO MASIERO

fAGNER CRISTIANO DONADON

GUILHERME WACHS LOPES

MATHEUS HENRIQUE PEREIRA GONÇALVES

TOM MIX MARTINI PETRECA

SIGEpaPP: patterns e anti-patterns associados à personas

Trabalho de Conclusão de curso, apresentado no Centro Universitário da FEI, como parte dos requisitos necessários para obtenção do título de Bacharel em Ciências da Computação, orientado pelo professor Dr. Plínio T. Aquino Junior.

**Resumo**

Este projeto tem por objetivo trabalhar uma técnica de apoio ao desenvolvedor de software. Ela utiliza três conceitos já existentes no mercado: *Patterns*, *Anti-Patterns* e *Personas* – estruturas denominadas neste trabalho como *APPP*. Embora os três conceitos citados já sejam conhecidos e aplicados por profissionais da área, a técnica proposta por este trabalho destaca-se especialmente pelo relacionamento entre elas, ou seja, uma agregação dos benefícios de cada conceito em prol de um outro. A técnica visa atender os desenvolvedores através da documentação de boas e más práticas no desenvolvimento de software. Por meio da documentação de problemas e soluções recorrentes, minimiza-se o tempo de esforço empregado em um problema já resolvido. É de conhecimento comum que soluções diversas, bem como boas e más práticas podem ser apropriadamente documentadas – tal como é feito atualmente por meio de *Patterns* e *Anti-Patterns* – porém, por meio desta técnica a documentação poderá ainda ser orientada a um ou mais perfis de usuários: as *Personas*. Visando sustentar a viabilidade e eficácia desta proposta, o trabalho apresenta também uma ferramenta que possibilita aos projetistas e desenvolvedores gerirem todo o processo de criação, pesquisa, aplicação e validação das estruturas *APPP* e seus respectivos relacionamentos. A ferramenta poderá ainda, no futuro, ser adaptada para receber as estruturas *APPP* automaticamente, por exemplo, através de RBC – Raciocínio Baseado em Casos – visando uma melhor autonomia na disponibilização de recursos para um bom desenvolvimento de software.

**Palavras Chaves:** Patterns, Anti-Patterns, Personas, Picaps.

**Abstract**

This project aims to work a technique in order to support the software developer. It uses three concepts already on the market: *Patterns*, *Anti-Patterns* and *Personas* - structures called in this work as APPP. While the three concepts mentioned are already known and implemented by professionals in the area, the technique proposed by this work stands out especially by the relationship between them, an aggregation of the benefits of each concept in favor of another one. The technique aims to support the developers through the documentation of good and bad practices in software development. Documenting recurring problems and solutions, minimizes the time it is used in an effort to issue already resolved. It is common knowledge that many solutions as well as good and bad practices can be properly documented - as it’s currently done through *Patterns* and *Anti-Patterns* - however, this technique through the documentation may be directed to one or more users profile: *Personas*. Seeking to sustain the viability and effectiveness of this proposal, the work also presents a tool that enables designers and developers to manage the whole process of creation, research, application and validation of APPP structures and their relationships. The tool can also in future be adapted to receive the structures APPP automatically, for example, through RBC - Reasoning Based on Cases - for a better autonomy in the provision of resources for a good software development.

**Key-words:** Patterns, Anti-Patterns, Personas, Picaps.

**Sumário**

1 Introdução 9

1.1 Objetivo 10

2 Trabalhos Relacionados 11

3 CONCEITOS FUNDAMENTAIS 14

3.1 *Patterns ou* Padrões 15

3.1.1 Origem 15

3.1.2 Definição 16

3.1.3 Linguagem de *Patterns* 18

*3.2* *Anti-Patterns* 20

3.2.1 Definição 20

3.2.2 Formato 20

3.2.3 Criação de *Anti-Patterns* 21

3.3 Personagens Fictícios ou *Personas* 22

3.3.1 Definição 22

3.3.2 O processo de criação 22

3.4 Raciocínio Baseado em Casos (RBC) 26

3.4.1 Definição 26

4 METODOLOGIA 29

4.1 Entrada 31

4.2 *Defin*iç*ão da Estrut*ura */ Template* 31

4.3 Inserção no *SiGePAPP* 32

4.4 Associação / Relacionamento 33

4.4.1 PICAPs – Relacionamento entre *Patterns* e *Personas* 33

4.4.2 Anti-PICAPs – Relacionamento entre *Anti-Patterns* e *Personas* 33

4.4.3 Relacionamento entre *Anti-Patterns* e *Patterns* 34

4.4.4 *APPP* – Relacionamento entre *Anti*-*Pattern*, *Pattern* e *Personas*. 34

4.5 Utilização 35

*4.6* *Feedback /*Avaliação 35

4.7 Análise De Feedback / Ações de Bloqueio 35

4.7.1 Cálculo da média: 36

4.7.2 Análise do resultado: 36

5 MODELAGEM 37

5.1 Requisitos 37

5.1.1 Requisitos Funcionais 37

5.1.2 Não Funcionais 40

5.2 Diagramas de Caso de Uso 42

5.2.1 Caso de Uso: Login 42

5.2.2 Caso de Uso: Cadastro 42

5.2.3 Caso de Uso: Consulta 43

5.2.4 Caso de Uso: Relacionamento 44

5.2.5 Caso de Uso: Avaliação 45

5.3 Diagrama de Classe 47

5.3.1 Classe Login 49

5.3.2 Classe Mensagem 49

5.3.3 Classe FeedBack 49

5.3.4 Classe FeedBackConteudo 50

5.3.5 Classe FeedBackEstrutura 50

5.3.6 Classe Estrutura 51

5.3.7 Classe Padrao 51

5.3.8 Classe Pattern 52

5.3.9 Classe AntiPattern 52

5.3.10 Classe Usuario 52

5.3.11 Classe Personas 53

5.3.12 Classe Solucao 53

5.4 Diagramas de Seqüência 54

5.4.1 Login 54

5.4.2 Cadastro de Usuário 55

5.4.3 Cadastro de Estrutura 55

5.4.4 Cadastro do APPP (Conteúdo) 56

5.4.5 Relacionamento 57

5.4.6 Busca 58

5.4.7 Feedback 58

5.4.8 Ações de Bloqueio 59

6 banco de dados 61

6.1 Modelo Entidade Relacionamento 61

7 Avaliação do Sistema 62

7.1 Avaliação da Interface 62

7.1.1 Avaliação Heurística 62

7.1.2 Testes com o Usuário 62

8 Planejamento 62

8.1 Cronograma 62

8.2 Tecnologias utilizadas 63

9 RESULTADOS ESPERADOS 64

10 CONCLUSÃO PARCIAL 65

Referências 66

**Lista de figuras**

Figura 1. Grafo da relação entre os padrões Fonte: (GERBER, 1999) 19

Figura 2. Diagrama de processo para criação de um Anti-Patterns Fonte: Os Autores 21

Figura 3. Metodologia do *SiGePAPP* Fonte: os autores. 30

Figura 4. Exemplo de PICAP. Fonte: (AQUINO JR., 2008) 34

Figura 5. Diagrama de Caso de Uso: Login Fonte: os autores. 42

Figura 6. Diagrama de Caso de Uso: Cadastro Fonte: os autores. 43

Figura 7. Diagrama de Caso de Uso: Consulta Fonte: os autores. 44

Figura 8. Diagrama de Caso de Uso: Relacionamento Fonte: os autores. 45

Figura 9. Diagrama de Caso de Uso: Avaliação Fonte: os autores 46

Figura 10. Digrama de Classes SiGePAPP Fonte: os autores 48

Figura 11. Diagrama de Seqüencia: Login Fonte: os autores. 54

Figura 12. Diagrama de Seqüência: Cadastro de Usuário Fonte: os autores. 55

Figura 13. Diagrama de Seqüência: Cadastro de Estrutura Fonte: os autores. 56

Figura 14. Diagrama de Seqüência: Cadastro do APPP Fonte: os autores. 57

Figura 15. Diagrama de Seqüência: Relacionamento Fonte: os autores. 57

Figura 16. Diagrama de Seqüência: Busca Fonte: os autores. 58

Figura 17. Diagrama de Seqüência: Feedback Fonte: os autores. 59

Figura 18. Diagrama de Seqüência: Ações de Bloqueio Fonte: os autores. 60

Figura 19. Modelo Entidade Relacionamento 61

**Lista de TABELAS**

[Tabela 1. Atributos fundamentais de um Pattern (GAMMA, et al., 1995) 16](#_Toc223881565)

[Tabela 2. Modelo de atributos opcionais para *Pattern*(GAMMA, et al., 1995) 17](#_Toc223881566)

[Tabela 3. Exemplo de *Pattern* 17](#_Toc223881567)

[Tabela 4. Atributos fundamentais de um *Anti-Pattern* 20](#_Toc223881568)

[Tabela 5. Tabela de exemplos de *Personas* 24](#_Toc223881569)

[Tabela 6. Cronograma de Desenvolvimento do Projeto 62](#_Toc223881570)

# Introdução

Nos dias atuais existem softwares controlando boa parte de tudo que nos cerca, desde um aparelho eletrônico até as grandes corporações. A qualidade do desenvolvimento de um software tornou-se algo fundamental para o sucesso do produto e, de forma significativa, vê-se intensificar esse quadro nos últimos anos. A demanda por sistemas desenvolvidos de forma mais rápida, o inevitável aumento da dependência das empresas de softwares que atendam aos seus negócios e a enorme diversidade de perfis de usuários acabam impactando diretamente a maneira como os sistemas computacionais são planejados, desenvolvidos e finalmente implementados. Por outro lado, o dinamismo, a necessidade de processamento, transformação de informações e a flexibilidade do mercado pressionam os desenvolvedores de tal forma que o tempo entre a solicitação do software e sua entrega final tem sido significativamente minimizado. Como conseqüência, a relação custo, qualidade e prazo tem sido seriamente desafiada e/ou comprometida. Ademais, soma-se ainda a este quadro o crescimento exponencial de usuários com seus mais variados perfis, características e conhecimentos, trazendo com eles a natural necessidade de que os sistemas envolvidos atendam de alguma forma as particularidades de cada perfil. Assim, os desenvolvedores carecem cada vez mais de técnicas e recursos que efetivamente os auxiliem a maximizar o tempo, minimizar o retrabalho e atender às crescentes necessidades de seus clientes. Tudo isso sem comprometer a qualidade do produto final.

Esse equilíbrio exigido pelo mercado – custo, qualidade e prazo – faz com que a criação, compartilhamento e a gerencia adequada de documentos se tornem ainda mais necessários.

Muitos são os itens a serem documentados: boas e más práticas, perfis de usuários, problemas e também possíveis soluções. Todas essas informações, quando disponíveis de forma padronizada e completa, auxiliam os desenvolvedores a, por exemplo, entenderem a maneira como um determinando componente foi desenvolvido, qual a melhor prática para resolução de um determinado problema e como essas soluções podem ser aplicadas a perfis específicos de usuários. Atualmente, os conceitos de *Patterns* e *Anti-Patterns* atendem parcialmente estas necessidades relacionadas ao desenvolvimento de software.

Em Aquino Jr. (2008) o autor descreve os *Patterns*, ou Padrões, como *s*oluções de problemas que se repetem em diferentes contextos. Ele esclarece que a idéia de padrões é capturar conhecimento do bom projeto e documentá-lo de forma que possa ser compartilhado pela comunidade.

Por outro lado os *Anti-Patterns* descrevem soluções ineficientes ou que resultam em conseqüências negativas (KOTZÉ, et al., 2006).

Essas duas estruturas – *Patterns* e *Anti-Patterns* – são ótimas técnicas de documentação durante o desenvolvimento de software, contudo poucas levam em consideração o perfil do usuário final e suas mais diversas características. Tal deficiência pode ser resolvida através da técnica de *Personas*.

De acordo com (COOPER, 1999) as *Personas* são um conjunto de informações realísticas e representativas que incluem detalhes fictícios para caracterização mais completa do usuário. Isso possibilita o planejamento do sistema focando um pequeno conjunto de perfis, porém atendendo uma grande quantidade de usuários e suas respectivas características (AQUINO JR., 2008).

## Objetivo

Este trabalho explora e relaciona os conceitos de *Anti-Patterns, Patterns e Personas* (APPP) promovendo uma técnica para apoio ao desenvolvimento de software. Também apresenta uma ferramenta, intitulada SiGePAPP (Sistema de Gerenciamento de *Patterns*, *Anti-Patterns* e *Personas*), que possibilita o compartilhamento, busca e gerenciamento das documentações.

# Trabalhos Relacionados

Os conceitos relacionados a este projeto de formatura são utilizados em diversos documentos, artigos e ferramentas existentes. Foram pesquisados trabalhos relacionados ao tema proposto, bem como publicações que possam embasar o uso das técnicas citadas.

Dentre esta busca, destaca-se o trabalho chamado SAMOA (SILVA, et al., 2004), que descreve um assistente automatizado para detecção de padrões em Diagramas UML, na WEB. O grupo defende a idéia de que a tarefa de encontrar padrões em um projeto de software qualquer se caracteriza por ser extremamente tediosa para o engenheiro de software, pois, segundo eles, requer encontrar todas as realizações possíveis desses padrões. Explica também que a busca por estes padrões força os profissionais a analisar todos os diagramas existentes no projeto, e em seguida, aplicar regras para aprimorá-las.

O SAMOA apresenta justamente a facilidade da automação desta tarefa, viabilizando o trabalho dos desenvolvedores. Este sistema pode receber dois tipos de entradas: a primeira é um diagrama de classes UML exportado no formato XMI e a segunda, o código fonte para realizar o processo de engenharia reversa (SILVA, et al., 2004). O sistema trabalha numa interação com o profissional de software e propõe criticas a padrões específicos, sugerindo melhorias baseadas em regras pré-definidas, por exemplo :

* Nome para as classes: o nome de um f*actory method* no padrão *Factory* *Method* deverá ter o sufixo *Factory*.
* Escopo para operações: Métodos *hook* no padrão *Template* *Method* devem ser declarados como *protected*.

A ferramenta proposta por este grupo poderá ser utilizada em conjunto com este projeto de formatura. Uma vez que ele objetiva justamente a análise de um padrão, o conteúdo do SiGePAPP (Sistema de Gerenciamento de *Patterns*, *Anti-Patterns* e *Personas*) pode servir como insumo para o SAMOA. Embora este projeto apresente uma avaliação prévia das estruturas APPP, não se pretende gerar críticas ou sugerir melhorias como o faz o SAMOA.

Devido a necessidade de facilitar a busca e aplicação de padrões, desenvolveu-se um repositório de padrões de software (MARINHO, et al., 2003). Este repositório possui a característica de integrar o modelo de processo de desenvolvimento de software - RUP (Rational Unified Process), tendo como objetivo oferecer um conjunto amplo e abrangente de padrões, de forma que possa ser consultado e utilizado durante as diversas fases de desenvolvimento. O repositório, portanto, tem a tarefa de catalogar os padrões que servirão de entrada para o RUP e contém significativas semelhanças com este projeto de formatura, contudo, não apresenta nenhuma relação com *Anti*-*Patterns* e/ou soluções voltadas para perfis de usuários.

Este repositório proposto é referenciado em outro trabalho chamado IIMPaR, “Uma Interface de Integração de Modelos de Padrões de Software para o Rose” (ANDRADE, et al., 2005).

O IIMPaR é uma ferramenta que tem por finalidade reutilizar modelos de padrões a partir de um formato aberto, construindo junto a ferramenta Rational Rose uma biblioteca de padrões que podem ser aplicados automaticamente. O grupo esclarece que o Rose é uma ferramenta de modelagem de software, baseada na linguagem UML, que tem sido usada com sucesso, tanto no meio acadêmico quanto no comercial. O IIMPaR apresenta uma interface de integração para o Rose, de forma a viabilizar a utilização de modelos de padrões no ambiente de modelagem do próprio Rose. A referência do IIMPaR com o repositório apresentado anteriormente vem justamente do fato de que os modelos de padrões utilizados por esta interface, podem estar armazenados em qualquer repositório, evidenciando a eficácia de uma base apropriadamente catalogada de padrões de software.

A ferramenta proposta neste projeto de formatura embora mais detalhada em termos de conteúdo e relacionamento com outras estruturas, como as *Personas*, poderá também servir de fonte e proporcionar um ambiente completo para o armazenamento, busca e aplicação desses padrões, trabalhando em conjunto com o IIMPaR.

A utilização dos *Patterns* pode ir além dos domínios da programação em computadores ou até mesmo da Engenharia de Software propriamente dita. A implantação de suas recomendações contribui beneficamente para o sucesso de qualquer trabalho, mostrando que a utilização dessa documentação como conceito – e não meramente algo inerente à computação – pode sim ser considerada em todos os contextos onde as boas práticas e soluções são compartilhadas e a experiência valorizada. Um trabalho que evidencia o parágrafo anterior é o Cognitor (ANACLETO, et al., 2007). No trabalho propõem-se a utilização de padrões, contudo, aplicando-os à melhoria no desenvolvimento de materiais de aprendizagem para Educação à Longa Distância (EAD). O Cognitor é um Framework baseado na Linguagem de Padrões Cog-Learn. O trabalho contextualiza a idéia de que projetar material de aprendizagem estruturado na forma de objetos de aprendizagem para Educação à Distância (EAD), em ambiente Web, é uma tarefa difícil para os professores – especialmente aqueles com pouca experiência em projeto de atividades de aprendizagem num ambiente computacional – além de exigir tempo e recursos muitas vezes não disponíveis. Os padrões criados pela equipe têm como finalidade gerar um vocabulário comum que possa ser utilizado por diversos participantes (ANACLETO, et al., 2007). O artigo divulgado pela equipe explica que esses padrões são então documentados em uma Linguagem de Padrões (LP) chamada Cog-Learn, que são, por sua vez, instanciados na ferramenta Cognitor que funciona como um framework para facilitar a utilização pelos professores – nesta ferramenta, os professores são auxiliados em suas tarefas de projetar e editar o material de aprendizagem.

Em contra partida, a diversificação de usuários quando não considerada, altera a eficácia na implantação de uma determinada solução. Em (COSTA, et al., 2008) foi desenvolvido um trabalho nomeado DEAN que, através de pesquisas de campo e utilização do algoritmo K-Means para mineração de dados, compõe personagens fictícios - *Personas*. Embora esta ferramenta crie as *Personas*, ela não faz nenhuma relação com *Patterns* e/ou *Anti-Patterns*.

Em (AQUINO JR., 2008), foi desenvolvido um trabalho chamado “PICaP: Padrões e *Personas* para expressão da diversidade de usuários no Projeto de Interação”. A técnica criada por ele – as PICaPs – é a junção de um *Pattern* a uma ou mais *Personas*. O autor explica que a falta de conhecimento de quem são os usuários, por parte daqueles que desenvolvem o software, é tão grave quanto à ausência de boas práticas no decorrer do desenvolvimento. No entanto, as PICaPs não abrangem o conceito de *Anti-Patterns*.

# CONCEITOS FUNDAMENTAIS

Nesta sessão, são apresentados os conceitos de *Patterns, Anti-Patterns, Personas* e Racionio Baseado em Caso, que são utilizados na composição da metodologia apresentada na sessão deste projeto de formatura.

## *Patterns ou* Padrões

O processo de desenvolvimento de software apresenta problemas, dúvidas e desafios, muitas vezes reincidentes, cuja resolução já foi encontrada. Através dessa recorrência de problemas, soluções e melhores práticas surgem os *Patterns*, uma documentação que emerge através da experiência de diversos profissionais e que pode ser compartilhada e recomendada a fim de se produzir software com resultado previsível, menor custo, minimização de retrabalho, ganhos de produtividade, desempenho e escalabilidade .

Em(AQUINO JR., 2008)o autor esclarece que um padrão deve conter uma solução comprovada para um problema recorrente de projeto cujo formato seja genérico, fácil de entender e legível. Os padrões devem prover soluções genéricas, documentadas em um formato tal que não seja necessariamente amarrado a um problema particular. É importante considerar que, da mesma forma, um padrão não é uma modelagem ou um componente pronto de forma a viabilizar a codificação final.

Os padrões são de extrema importância no caso de profissionais com pouca experiência. A documentação pode ser utilizada como uma leitura geral, ajudando o profissional na composição de idéias, resolução de problemas específicos, contribuindo em projetos de sistemas de forma a viabilizar e enriquecer a comunicação e discussão entre os desenvolvedores (AQUINO JR., 2008).

Neste projeto de formatura, os Padrões são comumente referenciados conforme a língua inglesa: *Patterns.* É de conhecimento geral que a palavra Padrão pode ser traduzida, compreendida e/ou aplicada de diversas maneiras. Dentre elas, destaca-se o uso da palavra *Standard,* representando normas ou regras que devem ser seguidas com precisão (WEHMEIER, 2000). Portanto, idéias ou conceitos provenientes das diversas traduções existentes para Padrões – como o abordado por *Standards* - não serão utilizadas.

### Origem

Os *Patterns* foram inicialmente definidos como uma forma de aplicar conhecimento ao trabalho(TAYLOR, 1911). Em 1977, um arquiteto chamado Christopher Alexander (ALEXANDER, 1977) percebeu que, com o passar do tempo, seu trabalho resumia-se a resolver problemas que de alguma forma já haviam sido tratados em algum momento. Alexander percebeu que os problemas eram relativamente parecidos e que, em muitos casos, a solução era genérica e que poderia, sem muitas dificuldades, ser aplicada a diversos problemas específicos em seus mais variados contextos. A partir da disponibilização dessas soluções, um arquiteto sem experiência poderia alcançar um excelente resultado por meio da implementação dessas regras de projeto, compartilhando um conjunto de padrões de forma que possa ser utilizado pela comunidade em geral.

O conceito de *Pattern* definido por Alexander tem, como referência, portanto, problemas e soluções relacionadas com a construção civil. Contudo, este conceito pode ser aplicado em quaisquer áreas de trabalho, inclusive em computação.

### Definição

Christopher Alexander definiu um padrão como*: “*Cada *Pattern* descreve um problema que ocorre várias e várias vezes em nosso meio, descrevendo, portanto, a solução para este problema,de tal forma que possa ser utilizada milhares de vezes, sem que haja o retrabalho desta*.”*(ALEXANDER, 1977).

Embora a aplicação prática de um *Pattern* seja um processo relativamente claro e simples, a definição exata de sua estrutura não é algo único. Existem diversas estruturas que contém atributos diferentes que podem ser atribuídos.

Em (GAMMA, et al., 1995) os autores explicam que, embora notações gráficas sejam importantes e úteis, elas por si só não são suficientes para se documentar um padrão. Essas notações simplesmente capturam o produto final do processo de modelagem, seus relacionamentos entre as classes e objetos. Para que uma solução possa ser reutilizada torna-se também necessário documentar decisões, alternativas e até mesmo exemplificar com casos concretos.

Um *Pattern* possui basicamente quatro atributos fundamentais. A apresenta estes atributos:

Tabela 1. Atributos fundamentais de um Pattern

|  |  |
| --- | --- |
| Nome | * O nome de um *Pattern* deve representá-lo de maneira abrangente. |
| Contexto | * Descrição da situação onde ocorre o problema. |
| Problema | * Descreve a situação na qual é encontrado este problema. |
| Solução | * Descreve os elementos que compõem a situação onde o *Pattern* deve ser aplicado. * Nunca deve ser descrito uma solução específica, como por exemplo, um código que implementa um algoritmo. A solução é como um modelo que deve ser aplicado a qualquer projeto. |

Além dos atributos fundamentais ainda podem ser inseridos diversos outros atributos que ajudem a compreender melhor o padrão descrito. (GAMMA, et al., 1995) cita ainda outros atributos opcionais que podem compor um padrão. A apresenta alguns destes atributos.

Tabela 2. Modelo de atributos opcionais para *Pattern*(GAMMA, et al., 1995)

|  |  |
| --- | --- |
| Conseqüências | * Descrevem quais são as vantagens, os resultados esperados na utilização deste *Pattern*. |
| Intento | * Uma declaração curta que responda as seguintes perguntas: O que exatamente esse *Pattern* faz? Qual é seu racional e objetivo/intento? Qual problema ou prática especificamente ele trata? |
| Também conhecido como | * Outros nomes pelos quais o *Pattern* é conhecido. |
| Exemplo de Código | * Uma exemplificação literal de como o *Pattern* pode ser codificado. Essa codificação pode ser feita em qualquer linguagem. |
| *Patterns* Relacionados | * Quais *Patterns* estão associados com este? Quais são as principais diferenças e semelhanças? Com quais outros *Patterns* este deveria ser utilizado? |

A apresenta um exemplo prático de um *Pattern*. Descreve uma boa maneira de visualização das informações como estrutura de árvores. Os atributos utilizados neste padrão envolvem uma imagem, a descrição do problema, o contexto e sua solução.

Tabela . Exemplo de *Pattern*

|  |  |
| --- | --- |
| Tree-Table    From Outlook Express's news reader | |
| Quando usar: | A interface do usuário apresenta diversas informações, de tal forma que a tabela torna-se completamente funcional (ou permite que os dados sejam ordenados). No entanto, os itens são inicialmente organizados de forma hierárquica, sendo apresentados em formato de árvore ou *outline*. |
| Por que: | A combinação das duas abordagens de apresentação de dados fornece o melhor resultado, a um custo que se resume em programação e design. É possível mostrar a hierarquia dos itens, além de uma matriz com dados adicionais ou atributos de cada item, tudo em uma estrutura unificada. |
| Como: | Os exemplos mostram o que deve ser feito: coloca-se a árvore na primeira coluna e os atributos dos itens nas colunas subseqüentes. As linhas – um item por linha – são normalmente selecionáveis. Naturalmente, pode ser combinada numa tabela ordenada com o objetivo de produzir uma estrutura mais usável e interativa.  Esta técnica tornou-se comumente utilizada em clientes de e-mail e noticias. |

Fonte: (TIDWELL, 2006)

### Linguagem de *Patterns*

Um repositório de *Patterns*, ou catalogo de padrões é um conjunto de padrões que podem ser utilizados isoladamente. Um conjunto de padrões, que pertencem a um mesmo contexto, quando combinados, gera uma linguagem de padrões. Essa linguagem torna a utilização dos padrões mais eficiente.

Uma linguagem de padrões é composta, por exemplo (AQUINO JR., 2008), por:

* Padrões individuais;
* Titulo;
* Descrição do argumento de aplicação;
* Grafo ilustrando o relacionamento;
* Etc.

Para que seja construído este grafo, é necessário saber alguns tipos possíveis de relacionamento. Exemplo de possíveis relacionamentos entre o padrão A (PA) e o padrão B (PB) (MARINHO, et al., 2003):

* Se PA usa PB, então a solução de PA é gerada utilizando PB;
* Se PA refina PB, então o problema de PA é uma especialização de PB;
* Se PA requer PB, então a aplicação de PB é exigida na aplicação de PA;
* Se PA é alternativo a PB, então ambos os padrões oferecem, apesar de diferentes, soluções a um problema.

A ilustra um grafo gerado para representar uma linguagem de *Patterns.* Cada *Pattern* é representado por um retângulo e estes se relacionam entre si através das setas, sendo que estas descrevem ainda o tipo de cada relacionamento.

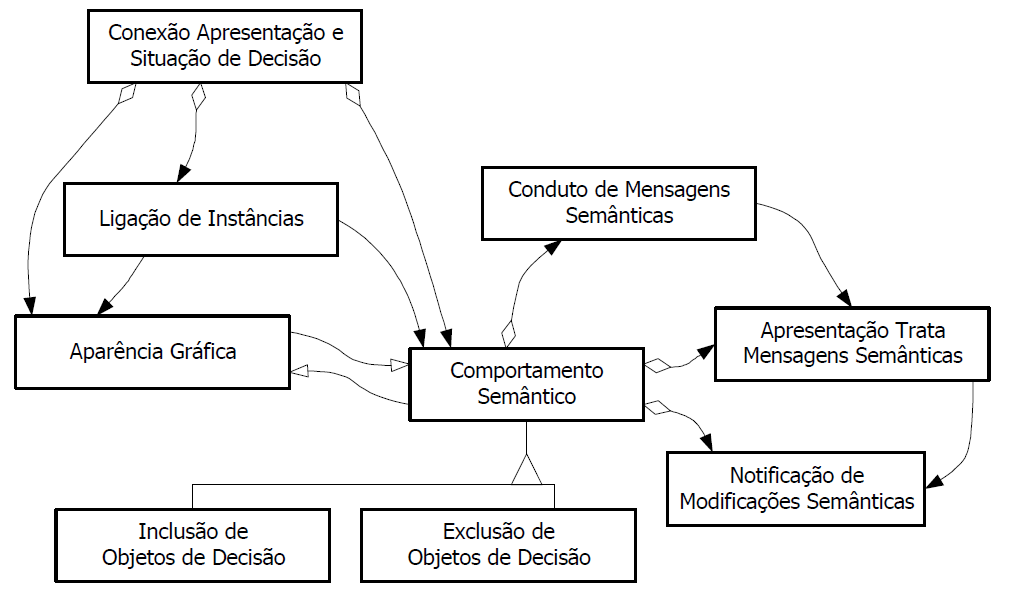


Figura 1. Grafo da relação entre os padrões  
Fonte: (GERBER, 1999)

Portanto, uma linguagem de padrões é um conjunto de padrões estruturado hierarquicamente conduzindo o projetista a uma abstração e ligação entre eles. Caso não exista tal estrutura hierárquica entre eles, isso é considerado apenas como um conjunto ou catalogo de padrões.

## *Anti-Patterns*

É senso comum que dificilmente um projeto seja entregue sem que erros aconteçam durante sua concepção. Naturalmente, os projetistas tencionam buscar melhores soluções gerando experiência e aprendizado, porém algumas dessas resoluções podem conter falhas.

Um meio de documentação para os problemas existentes que descreve erros ou procedimentos ineficientes é conhecido como *Anti-Patterns*(LONG, 2001). Em (AKROYD, 1996) foi apresentado este conceito*,* que tem como objetivo completar e agregar valor a idéia já existente de *Patterns*.

*Anti-Pattern* mostra inclusive o caminho seguido para tentar chegar ao resultado, evitando assim que outros profissionais cometam os mesmos equívocos.

### Definição

Pesquisas comprovam que cinco entre seis projetos de software falham (BROWN, et al., 2000). Essas falhas podem ser aproveitadas para documentar o domínio de soluções insatisfatórias por meio de *Anti-Patterns*, que permitem uma comunicação entre os projetistas.

Podem ser destacados dois tipos de Anti-Patterns (KOTZÉ, et al., 2006):

* *Simple Anti-Pattern*: Apresenta soluções negativas.
* *Amelioration Anti-Pattern*: Descreve os passos de uma solução ineficiente na tentativa de encontrar uma solução positiva.

### Formato

A apresenta os atributos fundamentais de um *Anti-Pattern* (MICROSOFT CORPORATION, 2005) (LONG, 2001):

Tabela 4. Atributos fundamentais de um *Anti-Pattern*

|  |  |
| --- | --- |
| **Nome:** | O nome de um *Anti-Pattern* deve representá-lo de maneira abrangente. |
| **Contexto:** | Descrição da situação onde ocorre o problema |
| **Problema:** | Descrição do problema |
| **Barreiras:** | Considerações adicionais que auxiliam o reconhecimento da situação |
| **Sintomas e Conseqüências:** | Os sintomas descrevem quais foram os erros encontrados que motivaram os especialistas a criá-los e as conseqüências descrevem o que pode acontecer caso o *Anti-Pattern* seja seguido. |
| **Recomendações:** | Como a solução pode ser alcançada, podendo ser direcionada há um *Pattern*. Este atributo está presente apenas em Anti-Patterns de Melhoria. |

Fonte: os autores

Outros atributos como: *Anti-Patterns* relacionados e palavras-chave podem ser considerados para orientar o leitor sobre problemas relacionados.

### Criação de *Anti-Patterns*

A partir de pesquisas realizadas junto ao Laboratório de Engenharia de Usabilidade da FEI (LEU-FEI), foi observado que o processo de criação de um *Anti-Pattern* depende, significativamente, do componente humano . Esse processo é constituído, basicamente, por três etapas fundamentais, conforme descrito na seqüência:

1. Análise e Estudo de Caso: Levantamento dos problemas
2. Agrupamento dos Erros: Abstrair problemas semelhantes e encontrar erros em comum.
3. Criação de Anti-Patterns: Documentar os problemas encontrados nos atributos dos Anti-Patterns.

A apresenta as etapas do processo para a documentação:

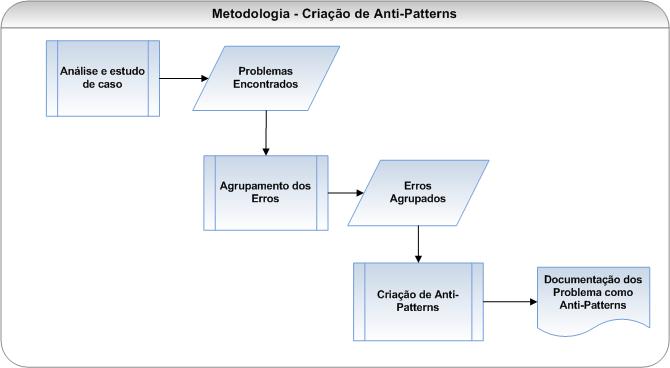


Figura . Diagrama de processo para criação de um Anti-Patterns  
Fonte: Os Autores

## Personagens Fictícios ou *Personas*

Devido ao crescimento da diversidade de usuários, torna-se evidente as suas diferenças de comportamento, necessidade, habilidade e experiência computacional. Essa variedade de características e perfis impõe aos projetistas um grande desafio, que é o de desenvolver um produto acessível a todos os perfis de usuários que o utilizarão . Uma resposta para este problema é unir as características dos usuários envolvidos e representá-los na forma de personagens fictícios, chamados *Personas.*

É sabido por meio de pesquisas e pela prática comum entre os projetistas, que estes possuem a tendência de desenvolver sistemas tendo como base o seu próprio perfil ao invés do perfil do usuário. As *Personas,* portanto, auxiliam a romper esta tendência, contribuindo para que estes profissionais desenvolvam o sistema focando nos perfis dos usuários(AQUINO JR., 2008).

### Definição

De acordo com (COOPER, et al., 2003), as *Personas* são personagens fictícios que caracterizam de forma mais completa o usuário final. Elas devem possuir um nome, características e imagem para agregar realismo e facilitar sua aplicação, comunicação e reconhecimento entre os profissionais que irão utilizá-las (ver exemplo em Tabela 5). Através deste conceito, é possível desenvolver um sistema baseado em poucos perfis, mas que, na verdade, abrange uma quantidade maior de usuários .

### O processo de criação

Dentre as muitas razões pelas quais um projeto não obtém sucesso – e partindo do pressuposto que este sucesso nada mais é do que aquilo que o cliente deseja – destaca-se, conforme já mencionado, a falta de conhecimento do projetista quanto às características dos usuários do sistema. Conhecer os usuários, saber exatamente quem eles são, como trabalham, quais são suas habilidades e experiências, quais seus reais perfis e atributos - todos esses itens devem ser considerados como fatores determinantes e de peso considerável em uma avaliação final por parte daqueles que efetivamente pagarão pelo produto.

Segundo o próprio criador de Personas, Cooper (COOPER, 1999) explica que os passos necessários para a criação de uma Persona são:

#### **Coleta de dados:** Para conhecer o usuário, as equipes de projeto devem realizar pesquisas, entrevistas e atividades correlatas. Esse estudo de campo, corpo a corpo com o usuário, é parte fundamental para a elaboração das *Personas*. Os projetistas são até mesmo aconselhados a recolher fotos dos usuários, descrever seus hábitos, experiências, como e quais tarefas desempenham e quem as executa. Qual a freqüência e nível de destreza necessária para cada tarefa, quais os caminhos que as pessoas normalmente utilizam para realizar essa ou aquela função em específico, dentre outros. Ao término da coleta de dados, os desenvolvedores estarão munidos de um volume enorme de informações que, embora ainda passível de refinamento, já os dará uma visão inicial dos tipos de usuários com quem estarão efetivamente trabalhando.

#### **Definir variáveis de Comportamento:** Esta etapa inicia o processo de mineração e tratamento dos dados obtidos por meio da coleta. O objetivo deste passo consiste na distribuição das pessoas em variáveis que definem o comportamento dos usuários, variáveis estas que são resultados das entrevistas e atividades explicadas no passo anterior. Em resumo, o produto final é agrupar as pessoas que possuem comportamentos próximos.

#### **Identificar padrões de comportamento significantes:** Este passo consiste numa análise mais profunda dos grupos formados pelo passo anterior identificando padrões de comportamento que se destacam. Segundo Cooper, uma provável *Persona* é identificada com o agrupamento de seis a oito variáveis diferentes.

#### **Sintetizar características e metas relevantes:** Toda *Persona* necessita de um nome, sobrenome e até mesmo uma ilustração. Embora fictício, a *Persona* deve ser personificada em todos os sentidos possíveis. Isso facilitará a comunicação entre os projetistas. Esta parte do processo responsabiliza-se justamente por essa personificação, descreve o ambiente de uso potencial, dia de trabalho típico, soluções e relacionamentos, etc.

#### **Verificar redundâncias:** Devido ao grande volume de informações coletadas, as redundâncias podem ser removidas. Por meio de pesquisas adicionais direcionadas a comportamentos em particular, torna-se possível descartar redundâncias, agrupar *Personas*, descartar atributos irrelevantes, etc.

#### **Descrição expandida:** A descrição expandida consiste na melhor maneira de literalmente estreitar e personificar o relacionamento entre os projetistas e as *Personas*. Essa descrição consiste na narrativa em terceira pessoa do usuário ou grupo de usuários representados pelas *Personas*. Nesta parte do processo descreve-se de maneira mais contextualizada as características do usuário representado.

Com base nesse processo, é apresentado na três *Personas* como exemplos. Aqui, assume-se que a coleta de dados já foi feita e o processo encontra-se completo, os dados já foram filtrados, os atributos relevantes escolhidos, os redundantes descartados e o grupo de usuário foi representado nessas três *Personas*. Seguindo ainda a metodologia de Cooper, cada *Persona* possui uma descrição expandida visando aumentar a personificação e contribuir para a viabilização de seu uso entre os projetistas.

Tabela 5. Tabela de exemplos de *Personas*

|  |  |
| --- | --- |
|  | **Perfil dos Usuários** |
| Persona 1.jpg | **Nome da Persona**: Antônio da Silva  **Experiência Computacional**: Alta  **Profissão**: Cientista da Computação  **Escolaridade**: Superior Incompleto  **Freqüência que utiliza computadores**: Diariamente  **Temperamento**: Calmo  **Descrição:** O Antônio é uma pessoa bastante ocupada. Normalmente, mesmo que não esteja sob forte pressão, enfatiza a necessidade de existirem processos rápidos e claros. Não tem paciência para ler instruções nem manuais. Diz sempre “não tenho tempo para isso”. Gosta de utilizar o computador apenas para as coisas que realmente não podem ser feitas de outro modo. |
| Persona 2.jpg | **Nome da Persona**: Fernanda Rodrigues  **Experiência Computacional**: Baixa  **Profissão**: Modelo  **Escolaridade**: Ensino Fundamental Completo  **Freqüência que utiliza computadores**: Semanalmente  **Temperamento**: Impulsiva  **Descrição:** A Fernanda é modelo e na verdade não entende porque precisa da informática. Para ela, é muito mais simples pagar para alguém fazer por ela, contudo, confessa que o mínimo de conhecimento possa ser eventualmente necessário. |
| Persona 3.jpg | **Nome da Persona**: Carlos Ribeiro  **Experiência Computacional**: Baixa  **Profissão**: Analista de Marketing  **Escolaridade**: Superior Completo  **Freqüência que utiliza computadores**: Diariamente  **Temperamento**: Calmo  **Descrição:** O Carlos utiliza o computador diariamente, pois seu trabalho exige que ele o faça, porém, não gosta de explorar funções que vão além daquilo que ele já faz no seu dia a dia. Na verdade, o Carlos é bastante inseguro e normalmente quando precisa aprender uma nova função necessita que alguém esteja ao seu lado explicando passo a passo, embora o computador dê instruções claras. |

Fonte: os autores

Vê-se, portanto, que a consolidação dos passos propostos por Cooper para a fabricação de uma *Persona* permite ao desenvolvedor personificar em sua mente o usuário para quem o sistema está sendo desenvolvido. A descrição visual permite aos desenvolvedores discutirem possíveis soluções para determinada *Persona* sem que, para isso, seja necessário listar todos os seus atributos novamente. Por meio da descrição expandida e seus mais variados atributos, é possível expandir a compreensão de quem realmente é o usuário, quais são suas dificuldades na realização de suas funções e até mesmo prever qual será seu comportamento diante de situações impostas naturalmente pelo uso da aplicação desenvolvida.

## Raciocínio Baseado em Casos (RBC)

É comum do ser humano adquirir conhecimento através de suas experiências durante a vida. Quanto maior seu conhecimento, maiores as chances de sucesso em suas tarefas.

Um estudante de Ciência da Computação, por exemplo, aprende diversos conceitos relacionados ao curso durante todos os anos de estudo, esse processo consiste, portanto, na obtenção daquilo conhecido como *conhecimento genérico*(GRESSE, 2003). Ao ingressar como estagiário no mercado de trabalho, ele naturalmente irá se deparar com inumeráveis problemas e desafios onde o conhecimento adquirido poderá então ser aplicado. Esse ciclo de aprendizagem e aplicação do conhecimento resulta em uma base de experiência extremamente rica, pois o indivíduo ganha experiência baseado em suas próprias soluções. A grande vantagem de acumular experiência está ligada ao natural reaparecimento de problemas com sintomas semelhantes, assim, ao reconhecer o problema, o aluno ou um indivíduo qualquer poderá aplicar então uma solução baseada no conhecimento passado, em geral, com poucas alterações.

Essa é a idéia do raciocínio baseado em casos: solucionar problemas atuais relembrando problemas similares do passado.

### Definição

O Raciocínio Baseado em Casos tem como objetivo descobrir soluções para um determinado problema. Essa idéia parte do princípio de que, para um domínio comum, existe uma grande possibilidade de ocorrência de um problema – ou seja, um caso (ABEL, 1996) – muito similar a algum já resolvido anteriormente. Portanto, sua solução é provavelmente muito próxima àquela desenvolvida no passado.

No processo de RBC, para encontrar uma solução, é necessário seguir algumas etapas, abaixo são listadas as principais etapas(GRESSE, 2003):

* + - Resgate
    - Adaptação
    - Retenção

#### Resgate

A etapa de resgate é responsável por buscar casos semelhantes a partir de uma situação qualquer. O algoritmo efetua uma comparação de características para gerar um grau de *similaridade,* entre 0.0 e 1.0, com os casos armazenados.

Existem dois tipos de *similaridade*:

* + *Similaridade Local*: Similaridade referente à comparação de uma única característica dos casos. Os métodos usados aqui podem ser(GRESSE, 2003):
    - *Correspondência exata*: Caso as *strings* são escritas da mesma forma. Por exemplo, "e-commerce" e "e-commerce" retorna 1.0, enquanto "e-business" e "e-commerce" retorna 0.0
    - *Correção ortográfica*: Ponderação entre o número de caracteres que são iguais pelo número total de caracteres. Por exemplo, "menino" e "menina", a quantidade de caracteres idênticos: 5. Quantidade total de caracteres: 6.Portanto, a similaridade é dada por: 5/6 = 0,83.
    - *Contagem de palavras:* Utilizado em textos maiores. Faz um contagem por palavras idênticas nos casos comparados.
    - *Taxa do maior substring comum:* Taxa entre a maior sequência de caracteres entre os dois casos pelo número total da consulta.
  + *Similaridade Global*: Similaridade referente à comparação de todas as características dos casos. Essas características, ou atributos, podem ter pesos, classificando-o como maior importância no cálculo.

Para ter maior eficiência, a busca pode ser feita em uma base de casos indexada. A fórmula abaixo apresenta o cálculo do algoritmo Nearest Neighbour (vizinho mais próximo)(GRESSE, 2003):

Sendo:

Wi: Peso do Atributo

Sim(X,Y): Similaridade global da consulta X com o caso Y

sim(qi,ci): Similaridade local entre atributo qi de X e ci de Y.

#### Adaptação:

De posse dos casos similares, o objetivo seguinte é adaptar as soluções encontradas para o problema proposto.

Abaixo são listadas algumas técnicas de adaptação de casos:

* + - Adaptação Nula: quando o caso já soluciona o problema proposto.
    - Adaptação Transformacional Substitutiva: Quando o caso possui a mesma estrutura, esta adaptação substitui informações baseadas em regras.
    - Adaptação Derivacional: Obtém informações sobre como um caso foi construído para que outro seja elaborado. Ou seja, nesta adaptação não se utiliza sua solução.

#### Retenção:

Depois de finalizado o processo de adaptação, o caso é então armazenado na base de casos. Para tanto, é necessário coletar informações relevantes para a indexação e colocá-lo na estrutura correta. Usando técnicas de manutenção de base de casos é possível mantê-la redundante e com um tamanho aceitável(GRESSE, 2003).

Com base nesses conceitos a metodologia do trabalho é definida com objetivo de guiar e embasar a criação da ferramenta proposta neste projeto de formatura, conforme segue na sessão .

# METODOLOGIA

Nesta seção, é apresentada a metodologia utilizada para a associação das estruturas *APPP*, bem como a apresentação do produto final como resultado deste vínculo. O fluxograma abaixo detalha todo o processo proposto, desde suas entradas, definições intermediárias, inserção no sistema e avaliação de bloqueio e disponibilização das estruturas entre os usuários.

A apresenta a metodologia do *SiGePAPP* e na seqüência o detalhamento de cada um dos artefatos que a compõe:

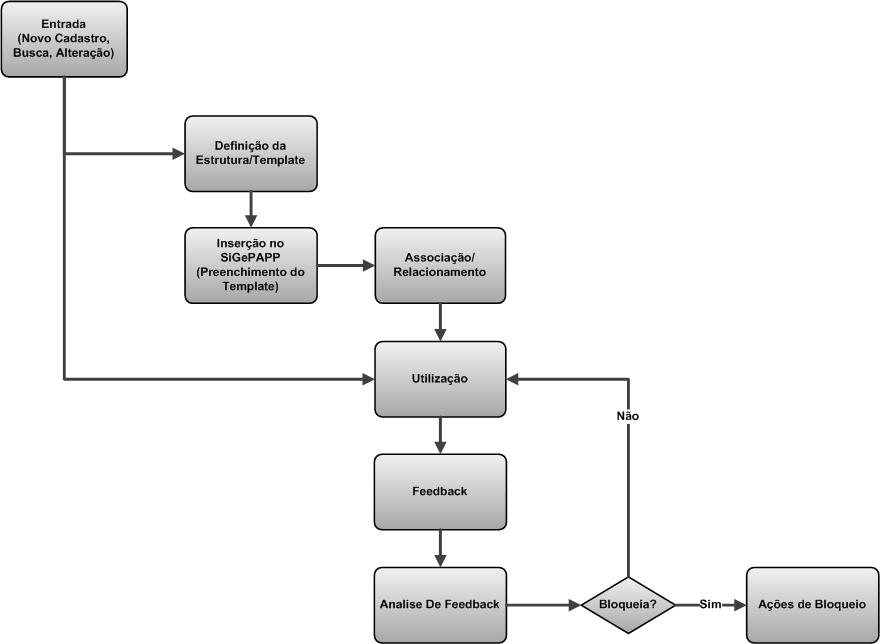


Figura 3. Metodologia do *SiGePAPP*  
Fonte: os autores.

## Entrada

Como parte inicial da metodologia do SiGePAPP, o processo de entrada resume-se nos meios pelos quais o usuário poderá iniciar o fluxo de criação, pesquisa ou alteração de uma estrutura APPP:

#### **Busca:** Por conter uma base de documentações inseridas pelos mais diversos profissionais da área, os usuários do SiGePAPP poderão pesquisar por um Pattern, Anti-Pattern ou Persona, sendo possível também encontrar outros documentos relacionados (conforme descrito no item ). O processo inicia-se com o preenchimento dos seguintes campos: Nome, Contexto, Problema e/ou Solução. A partir disso, o sistema utilizar-se-á das técnicas provenientes da busca por similaridade em RBC (Raciocínio Baseado em Casos) (conforme descrito no item ) para comparar as informações digitadas com os atributos comuns entre Patterns e Anti-Pattern. Ao final, o sistema apresenta uma lista dos Patterns, Anti-Patterns e Personas, sendo que a última é obtida através do relacionamento desta com as primeiras.

#### **Novo Cadastro:** Conforme descrito no item .

#### **Alteração:** O usuário poderá alterar uma estrutura por dois motivos: primeiro, acrescentar informações de relevância ou qualquer alteração necessária para a melhoria do conteúdo, desde que esta documentação não tenha sido avaliada por outros usuários e segundo, caso a documentação tenha recebido uma avaliação negativa (conforme item ).

## *Defin*iç*ão da Estrut*ura */ Template*

Uma estrutura bem definida – com seus atributos e características essenciai*s – é tão import*ante quanto o próprio conteúdo que será inserido no sistema. Em os autores explicam que estruturas como os *Patterns*, por exemplo, devem ser descritos num formato claro e consistente. Cada estrutura segue um tipo de *template* (modelo)variando de acordo com o c*ontexto e necess*idades específicas. Os autores enfatizam que uma estrutura bem formatada é essencial para a compreensão e utilização por parte dos usuários. Seus campos ou atributos devem ser escolhidos de maneira que efetivamente contribuam para clareza e consistência mencionada pelos autore*s.*

*Devid*o aos mais diversos contextos nos quais um**a** estrutura *APPP* poderá vir a ser enquadrada, o *SiGePAPP* permitirá ao usuário desfrutar de uma considerável flexibilidade para determinar, ele próprio, cada atributo que sua estrutura apresentará. Visando manter a organização e facilitar a associação entre os contextos inseridos no sistema, o *SiGePAPP* possui uma propriedade denominada “Estrutura Mínima”, ou seja, a despeito dos atributos que poderão ser criados pelo usuário, eles sempre partirão de alguns campos que serão obrigatória e automaticamente inseridos na documentação no momento que o usuário optar pela criação de uma nova estrutura. Os atributos que compõem as estruturas mínimas para cada componente do *APPP* são:

* *Pattern*
  + Nome;
  + Contexto;
  + Descrição do Problema;
  + Solução.
* *Anti-Pattern*
  + Nome;
  + Contexto;
  + Solução;
  + Descrição do Problema;
  + Barreiras;
  + Sintomas e Conseqüências;
  + Recomendações;
* *Personas*
  + Nome;
  + Descrição;

Caso o usuário julgue que os atributos acima não são suficientes para uma boa descrição de sua documentação, ele terá, portanto, a flexibilidade para criar outros atributos que irão compor sua estrutura final. Ademais, o usuário poderá ainda usufruir de estruturas criadas por outros usuários.

## Inserção no *SiGePAPP*

O usuário documenta suas próprias experiências no SiGePAPP por meio dos APPPs. Essa documentação consiste no registro de soluções, boas e más práticas e perfis de usuários envolvidos no contexto abordado pela documentação. Primeiramente o usuário deve escolher entre utilizar uma estrutura já existente ou criar uma nova. Caso o usuário escolha criar uma nova estrutura APPP ele procederá com a definição de seus atributos e esta então será disponibilizada para uso. Uma vez tendo a estrutura definida é feito o preenchimento.

## Associação / Relacionamento

O sistema permite ao usuário efetuar o relacionamento entre os documentos APPP gerados*.* Este relacionamento pode ocorrer entre as três, ou apenas duas das estruturas. Em ou autor expõe uma estrutura chamada de PICaP, que é justamente a relação entre *Pattern* e *Personas* (ver ) *.* Ele explica que a relação entre as duas estruturas enriquece a documentação, pois embora o problema seja o mesmo, a solução abordada pode ser feita tendo como base as características de cada uma das *Personas* envolvidas.

A seguir, é detalhado como é possível realizar estes relacionamentos:

### PICAPs – Relacionamento entre *Patterns* e *Personas*

Conforme apresentado na seção , um *Pattern* possui diversos atributos. Ao descrever a solução para um determinado problema, o *Pattern* apresenta detalhes de como este problema poderá ser solucionada para esse ou aquele perfil de usuário (conforme ), sendo que estes perfis estarão representados pelas *Personas* anexadas ao *Pattern*. A grande vantagem dessa estrutura é que uma solução pode ser extremamente eficaz para um determinado perfil de usuário, contudo, completamente inútil para outro. O relacionamento, portanto, do *Pattern* com as *Personas* assegura que o usuário final seja realmente considerado como sendo parte fundamental para uma solução adequada.

### Anti-PICAPs – Relacionamento entre *Anti-Patterns* e *Personas*

O relacionamento entre um *Anti-Pattern* e *Personas* possui objetivo semelhante ao explicado no item anterior. Contudo, o atributo utilizado como meio de relacionamento entre as duas estruturas é a “Conseqüência” ou “Sintoma” do *Anti-Pattern* (atributos explicados nos Conceitos Fundamentais). Por meio dessas duas documentações relacionadas, é possível documentar exatamente como cada perfil de usuário é afetado, tornando possível ao desenvolvedor precaver-se de situações indesejáveis e tomar as ações preventivas de maneira personalizada para cada perfil de usuário do sistema em desenvolvimento.

### Relacionamento entre *Anti-Patterns* e *Patterns*

O relacionamento entre essas duas documentações resume-se praticamente no fato de que uma é conseqüência ou resposta natural da outra. Em um *Pattern*, o seu atributo “Problema” pode ser referenciado diretamente a um *Anti-Pattern*. Da mesma forma, em um *Anti-Pattern*, o atributo “Solução” pode ser referenciado a um *Pattern*. Isso torna a documentação muito mais objetiva e direta, características tão claramente defendidas por diversos autores, como descritas em (GAMMA, et al., 1995).

### *APPP* – Relacionamento entre *Anti*-*Pattern*, *Pattern* e *Personas*.

Este relacionamento consiste na junção de todas as três junções descritas acima. Devido à flexibilidade contida na criação de cada estrutura, o usuário poderá facilmente propor o relacionamento entre todas elas, usufruindo assim dos benefícios conjuntos de cada uma, resultando em uma estrutura completa APPP.

Torna-se importante enfatizar a realidade de que existem casos onde um determinado problema tem características demais primitivas ou abstratas a ponto de não permitir nenhum vínculo entre as estruturas.

A exemplifica uma estrutura *APPP* contendo a relação entre um *Pattern* e duas *Personas*.

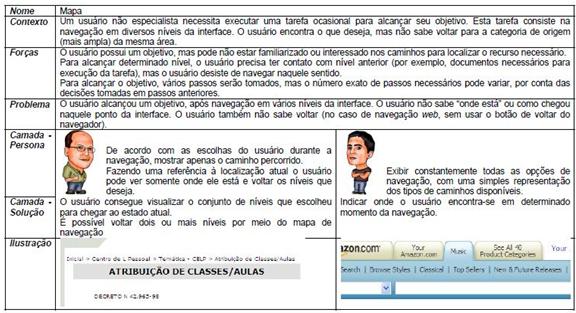


Figura . Exemplo de PICAP.  
Fonte: (AQUINO JR., 2008)

## Utilização

Uma vez finalizado o processo de inserção e disponibilização, o usuário tem a possibilidade de navegar pelo SiGePAPP, verificar estruturas e conteúdos de APPP, com o intuito de aplicá-los em seus projetos usufruindo de todos os recursos do sistema.

## *Feedback /*Avaliação

Conforme as documentações são disponibilizadas para uso, os usuários podem avaliá-las à medida que são aplicados em seus projetos. Essa avaliação é feita, através de um questionário cujas respostas poderão ser entre 0 (péssimo) e 5 (excelente), contendo ainda um campo de comentário.

Essa avaliação é essencial para a filtragem dos dados inseridos no sistema, pois somente por meio deste *feedback* por parte dos usuários é possível classificar a confiabilidade de cada documento. A avaliação divide-se em dois tipos:

* Estrutural: Consiste na avaliação da estrutura do APPP. Ela deve apontar se os atributos conseguem realmente atender aos conceitos delineados pelo contexto da estrutura.
* Conceitual: Consiste na avaliação do conteúdo propriamente dito de cada estrutura, ou seja, avalia se a idéia sugerida pela documentação realmente surtiu efeito conforme descrita. Tal avaliação obviamente só será possível após a utilização prática dos conceitos documentados.

Com o intuito de garantir a integridade da avaliação, cada usuário avalia apenas uma vez os documentos criados por outros usuários e não poderá avaliar estruturas criadas por ele próprio. A experiência de uso aliada às avaliações freqüentes fará com que somente as melhores estruturas sejam destacadas, contribuindo para o sucesso no compartilhamento de conhecimento e experiências. O resultado desta avaliação alimenta o processo de análise descrito no item .

## Análise De Feedback / Ações de Bloqueio

Após todo evento de *feedback,* será executada uma análise para verificar se o item avaliado deve continuar disponível para uso ou não. Esse procedimento é importante para garantir que as melhores descrições de problemas e soluções sejam apresentadas de forma prioritária ao usuário. Com o intuito de manter os resultados confiáveis, a análise só será efetuada caso o documento avaliado tenha uma amostra de, no mínimo, dez notas.

De acordo com etapas específicas, listadas a seguir, é verificado se os itens são considerados adequados para permanecerem no sistema:

### Cálculo da média:

Através da , é possível obter a média dos usuários que avaliaram um documento ou estrutura:

Equação 1. Equação para o cálculo de média da avaliação

Onde:

N(x) = Nota da análise

n(i) = Nota de um usuário i

Sendo:

Para assegurar a acuracidade deste cálculo é necessário obter a tendência do desvio padrão por meio de uma amostra de dados concretos, os quais serão obtidos através de teste preliminares realizados durante o desenvolvimento do sistema.

### Análise do resultado:

De posse da média das avaliações, o sistema irá analisar o resultado seguindo duas regras:

* Se : Documento ou estrutura continua habilitado para uso.
* Se 50%: Neste caso, o sistema bloqueia e oculta dos usuários o documento e/ou estrutura.

A cada APPP inserido no sistema, o usuário opta por receber a avaliação por e-mail. Em caso de bloqueio o autor recebe uma mensagem que informa essa ocorrência. Tratando-se de um *Pattern* o sistema sugere documentá-lo como *Anti-Pattern* ou efetuar as devidas correções. Caso esta estrutura venha a ser desativada e corrigida, sua avaliação é zerada e novas avaliações são realizadas.

# MODELAGEM

Nesta sessão serão abordados os seguintes itens:

* Requisitos
* Diagramas de Casos de Uso
* Diagramas de Classes
* Diagramas de Sequência

## Requisitos

### Requisitos Funcionais

|  |  |
| --- | --- |
| Identificação: | RF-1 |
| Requisito: | Cadastro / Alteração de *Pattern* |
| Descrição: | Possibilidade de cadastrar um *Pattern* baseado em uma estrutura definida A estrutura poderá ser a mínima ou personalizada pelo usuário. Caso a estrutura ou o conteúdo tenha recebido algum *Feedback* (RF 11 e 12) a alteração não será mais permitida. |

|  |  |
| --- | --- |
| Identificação: | RF-2 |
| Requisito: | Cadastro / Alteração de *Anti-Pattern* |
| Descrição: | Possibilidade de cadastrar um *Anti-Pattern* baseado em uma estrutura definida. A estrutura poderá ser a mínima ou personalizada pelo usuário Caso a estrutura ou o conteúdo tenha recebido algum *Feedback* (RF 11 e 12) a alteração não será mais permitida. |

|  |  |
| --- | --- |
| Identificação: | RF-3 |
| Requisito: | Cadastro / Alteração de *Persona* |
| Descrição: | Possibilidade de cadastrar uma *Persona* baseada em uma estrutura definida A estrutura poderá ser a mínima ou personalizada pelo usuário. Caso a estrutura ou o conteúdo tenha recebido algum *Feedback* (RF 11 e 12) a alteração não será mais permitida. |

|  |  |
| --- | --- |
| Identificação: | RF-4 |
| Requisito: | Cadastro / Alteração da Estrutura de *Pattern* |
| Descrição: | Caso não exista uma estrutura que descreva eficientemente um *Pattern*, o usuário poderá criar uma nova estrutura a patir da mínima com atributos personalizados. |

|  |  |
| --- | --- |
| Identificação: | RF-5 |
| Requisito: | Cadastro / Alteração de Estrutura de *Anti-Pattern* |
| Descrição: | Caso não exista uma estrutura que descreva eficientemente um *Anti-Pattern*, o usuário poderá criar uma nova estrutura a partir da mínima com atributos personalizados. |

|  |  |
| --- | --- |
| Identificação: | RF-6 |
| Requisito: | Cadastro / Alteração de Estrutura de *Persona*. |
| Descrição: | Caso não exista uma estrutura que descreva eficientemente um *Persona*, pode , o usuário poderá criar uma nova estrutura a partir da mínima com atributos personalizados |

|  |  |
| --- | --- |
| Identificação: | RF-7 |
| Requisito: | Encontrar *Patterns*, *Personas* ou *Anti-Patterns* similares |
| Descrição: | A partir da visualização de um determinado *APPP,* será possível encontrar *APPPs* similares. |

|  |  |
| --- | --- |
| Identificação: | RF-8 |
| Requisito: | Busca por Contexto. |
| Descrição: | Encontrar outros *APPP* a partir de atributos específicos digitados pelo usuário, tais como nome, contexto etc. |

|  |  |
| --- | --- |
| Identificação: | RF-9 |
| Requisito: | Associações entre os *APPP* |
| Descrição: | Há quatro tipos de relacionamentos entre os *APPPs*:   * Relacionar um *Pattern* à *Personas(PiCAPS)* * Relacionar um *Anti-Pattern* à *Personas(Anti-PiCAPS)* * Relacionar *Pattern* com *Anti-Pattern*. * Relacionar *Pattern* com *Anti-Pattern* e *Personas*. |

|  |  |
| --- | --- |
| Identificação: | RF-10 |
| Requisito: | Cadastro de Usuários |
| Descrição: | Os usuários devem estar cadastrados previamente de acordo com seu perfil para:   * Criação de *APPP* * Edição de *APPP* * Visualização de um *APPP* |

|  |  |
| --- | --- |
| Identificação: | RF-11 |
| Requisito: | *FeedBack* de Estruturas |
| Descrição: | Usuários podem atribuir notas às estruturas. As notas variam de 0 (zero) a 5 (cinco) sendo 0 (zero) a pior nota e 5 (cinco) a melhor. Cada usuário poderá avaliar uma estrutura uma única vez. Usuários não poderão avaliar suas próprias estruturas. |

|  |  |
| --- | --- |
| Identificação: | RF-12 |
| Requisito: | *FeedBack* de Conteúdo |
| Descrição: | Usuários podem atribuir notas ao conteúdo. As notas variam de 0 (zero) a 5 (cinco) sendo 0 (zero) a pior nota e 5 (cinco) a melhor. Cada usuário poderá avaliar o conteúdo de uma documentação uma única vez. Usuários não poderão avaliar conteúdos inseridos por eles próprios. |

|  |  |
| --- | --- |
| Identificação: | RF-13 |
| Requisito: | Ação de bloqueio |
| Descrição: | De acordo com dados obtidos com a análise de *FeedBack*(RF-14), o sistema bloqueará temporariamente o *APPP*, avisando o autor desse bloqueio. Caso o *APPP* publicado for um *Pattern*, o sistema irá sugerir ao usuário documentá-lo como um *Anti-pattern*. |

|  |  |
| --- | --- |
| Identificação: | RF-14 |
| Requisito: | Análise de *Feedback* |
| Descrição: | Avaliações automáticas baseadas em regras definidas que classificarão o *APPP* como bloqueado ou liberado para uso. |

|  |  |
| --- | --- |
| Identificação: | RF-15 |
| Requisito: | Exibir *APPP* relacionado |
| Descrição: | Mostrar ao usuário as associações efetuadas em RF-9 |

### Não Funcionais

|  |  |
| --- | --- |
| Identificação: | RNF-1 |
| Requisito: | Estrutura *Client-Server* |
| Descrição: | Estrutura que contém computadores clientes e computadores servidores onde existe uma comunicação em rede. |

|  |  |
| --- | --- |
| Identificação: | RNF-2 |
| Requisito: | Usabilidade de Resultados Impressos |
| Descrição: | Capacidade de visualização simplificada da página. |

|  |  |
| --- | --- |
| Identificação: | RNF-3 |
| Requisito: | Uso de SGBD |
| Descrição: | Sistema de gerenciamento de banco de dados para criação de banco de dados relacional. |

|  |  |
| --- | --- |
| Identificação: | RNF-4 |
| Requisito: | Segurança |
| Descrição: | Controle de usuários por meio de *Login*. |

|  |  |
| --- | --- |
| Identificação: | RNF-5 |
| Requisito: | Ambiente WEB |
| Descrição: | Sistema deverá ser projetado em plataforma WEB. |

|  |  |
| --- | --- |
| Identificação: | RNF-6 |
| Requisito: | Portabilidade (IE, FIREFOX) |
| Descrição: | Portabilidade aos navegadores:   * Internet Explorer (A partir da V. 6.0) * Firefox (A partir da V. 2.0) |

|  |  |
| --- | --- |
| Identificação: | RNF-7 |
| Requisito: | Manutebilidade |
| Descrição: | Código estruturado para prover manutenção facilitada. |

|  |  |
| --- | --- |
| Identificação: | RNF-8 |
| Requisito: | Usabilidade de interface |
| Descrição: | Facilidade e eficiência na interface humano-computador. |

## Diagramas de Caso de Uso

Nesta seção serão apresentados os diagramas de casos de uso para os cenários levantados.

### Caso de Uso: Login

Na é apresentado o diagrama de caso de uso para o cenário login, onde o usuário realizará sua autenticação para acessa e utilizar o sistema.

Ao acessar o caso de uso “Efetuar Login”, o usuário informará ao sistema seu nome de usuário e senha, com estes dados o sistema iniciará o processo de validação de login e este sendo aceito abrirá a tela inicial do sistema.

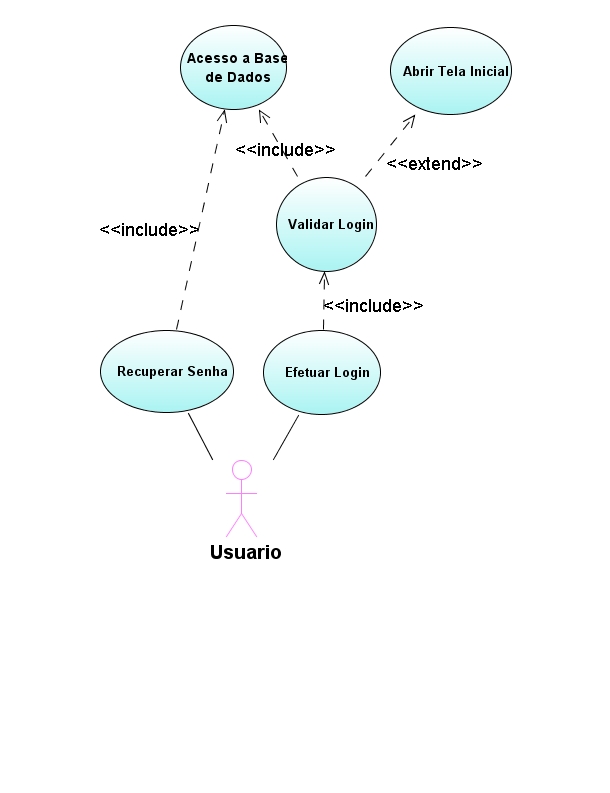


Figura . Diagrama de Caso de Uso: Login  
Fonte: os autores.

### Caso de Uso: Cadastro

Na é apresentado o diagrama de caso de uso para o cenário de cadastro, onde o usuário poderá efetuar o seu próprio cadastro de acesso ao sistema, cadastro das estruturas envolvidas no *APPP*, bem como o conteúdo de cada *APPP*.

Ao realizar o acesso ao caso de uso “Cadastro de Usuário”, o usuário poderá efetuar o cadastro de seu acesso e alterações de seu perfil no sistema. Para realizar o cadastro de uma nova estrutura *APPP*, o acesso é feito pelo caso de uso “Cadastro de Estrutura”, se não existir os atributos desejados pelo usuário este deverá inseri-lo através do caso de uso “Criação e/ou Alteração de atributos”. Por fim, o cadastro dos *APPP* será realizado através do caso de uso “Cadastro de Conteúdo”.

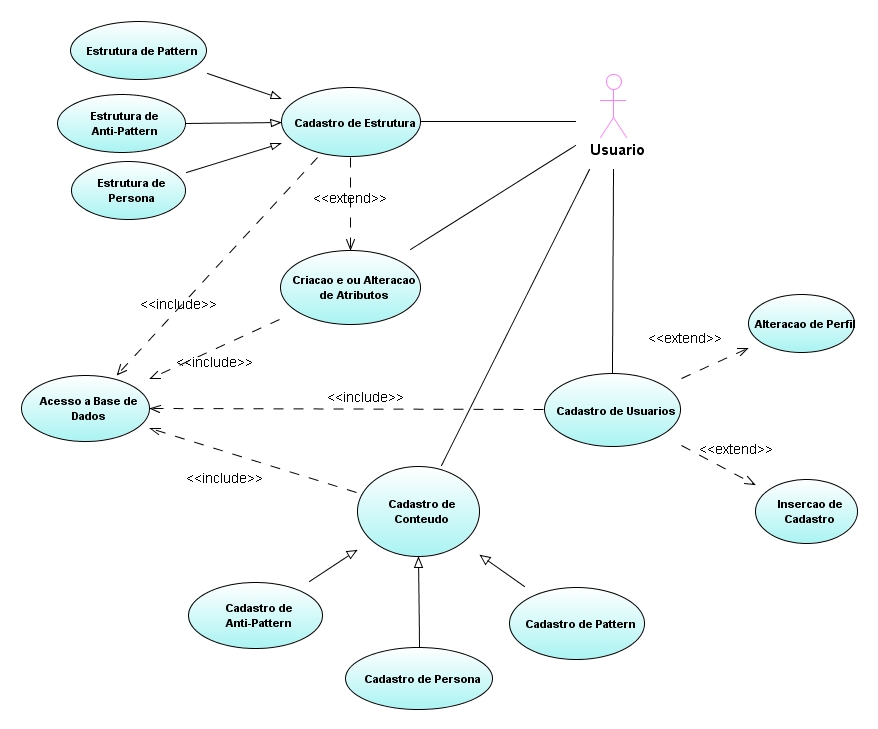


Figura . Diagrama de Caso de Uso: Cadastro  
Fonte: os autores.

### Caso de Uso: Consulta

Na é apresentado o diagrama de caso de uso para o cenário de pesquisa onde o usuário conseguirá realizar suas pesquisas por problemas e soluções apresentadas através das estruturas *APPP*.

Pelo caso de uso “Busca”, o usuário poderá realizar dois tipos de pesquisas no sistema, a primeira é através do contexto do problema vivenciado por ele e uma segunda pelos *APPP* similares ao que o usuário estará cadastrando. No caso de uso “Selecionar os Campos para a Busca”, o usuário realizará o filtro para exibição dos seus resultados.

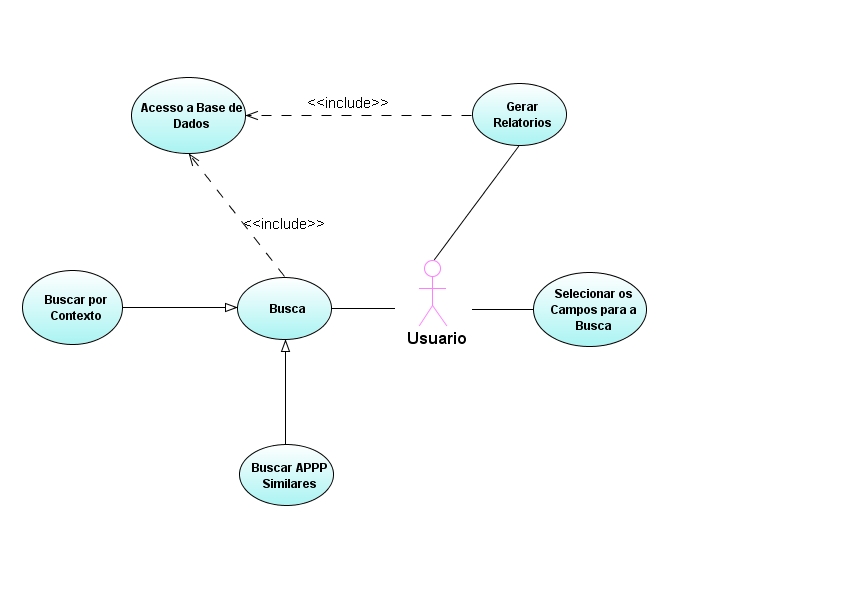


Figura . Diagrama de Caso de Uso: Consulta  
Fonte: os autores.

### Caso de Uso: Relacionamento

Na é apresentado o diagrama de caso de uso para o cenário de relacionamento entre as *APPP*, conforme o usuário for realizando os relacionamentos, o sistema também poderá efetuar o relacionamento automaticamente entre os *patterns* e *anti-patterns.*

Através do caso de uso ”Relacionar / Associar”, o usuário poderá realizar os relacionamentos descritos na seção, além dos relacionamentos realizados manualmente pelo usuário, o sistema também poderá realizar o relacionamento automático entre *patterns* e *anti-patterns.*

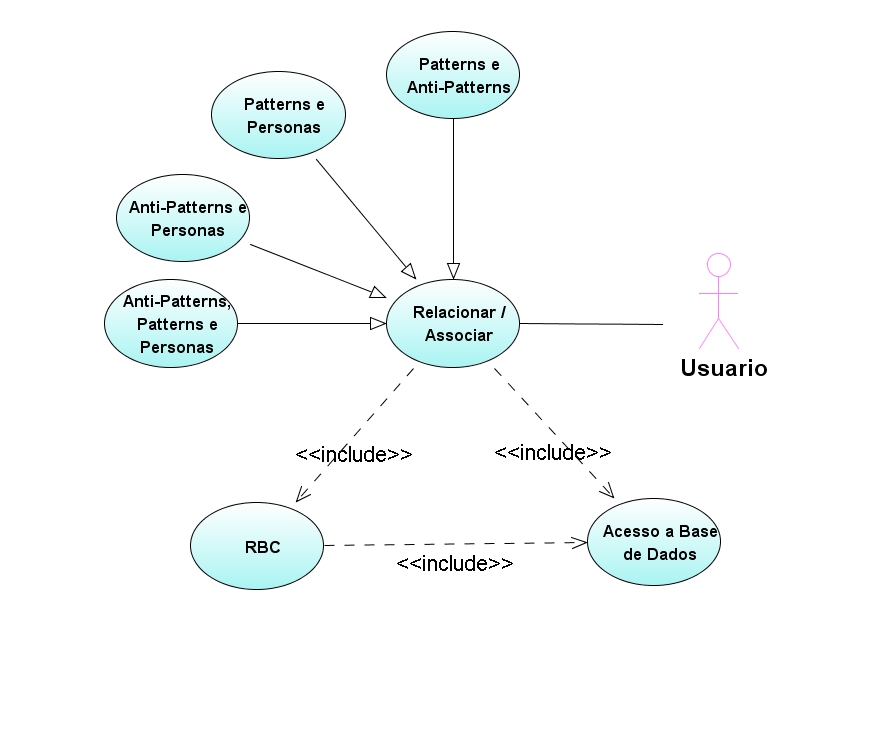


Figura . Diagrama de Caso de Uso: Relacionamento  
Fonte: os autores.

### Caso de Uso: Avaliação

Na é apresentado o diagrama de caso de uso para o cenário de avaliação das estruturas *APPP* e os próprios *APPP* pelos usuários do sistema.

Através dessas avaliações o sistema poderá bloquear temporariamente a exibição, de uma determinada *APPP* ou sua estrutura até que a mesma seja alterada pelo seu criador. Além disso, caso trate-se de um *Pattern* e este receba constantes más avaliações, o sistema poderá sugerir ao autor que o documente como um *Anti-Pattern.*

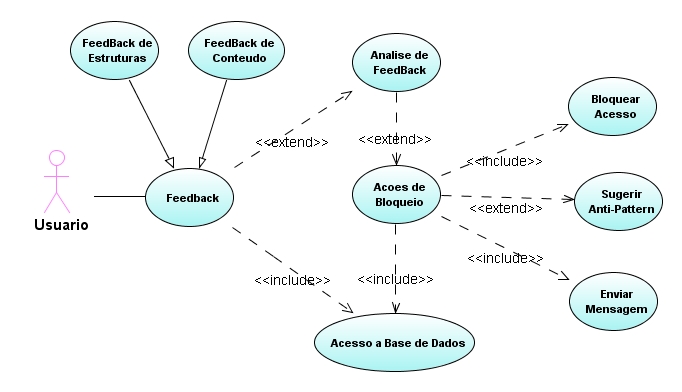


Figura . Diagrama de Caso de Uso: Avaliação  
Fonte: os autores

## Diagrama de Classe

O Diagrama de Classe tem por finalidade representar as classes utilizadas, de forma gráfica, de acordo com as regras de UML. A apresenta o diagrama de classe gerado para este TCC.

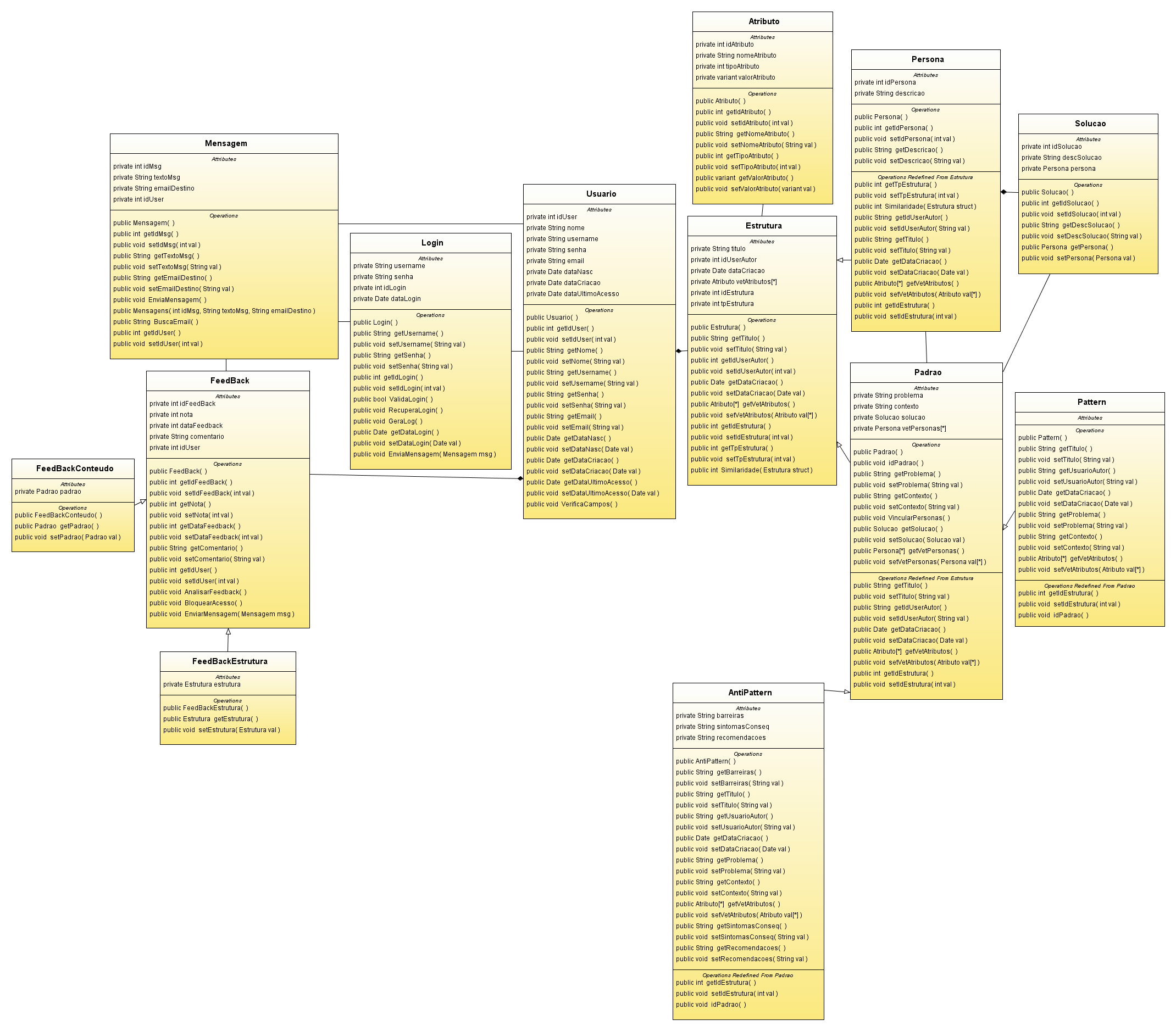


Figura . Digrama de Classes SiGePAPP  
Fonte: os autores

### Classe Login

Esta classe contém informações sobre o login do sistema.

Atributos:

* + - * + username: Nome de usuário usado para o login.
        + senha: senha usada para login.
        + idLogin: Código do usuário que possui esse login.
        + dataLogin: data de ocorrência de login. Usado para controle e log.

Métodos:

* + - * + ValidaLogin(): Executa a validação de nome de usuário e senha.
        + RecuperaLogin(): Caso o usuário esqueça sua senha, esse método permite recuperar o login do usuário, enviando para seu e-mail cadastrado, sua senha.
        + GeraLog(): Á cada login efetuado, é gerado um registro de log, para segurança.
        + EnviaMensagem(Mensagem msg): É o método responsável por enviar a mensagem para o usuário com seus dados de acesso.

### Classe Mensagem

Esta classe contém as instruções para envios de mensagens para os usuários do sistema.

Atributos:

* + - * + idMsg: Guarda o código de controle da mensagem enviada.
        + textoMsg: É o texto da mensagem enviada.
        + emailDestino: Endereço eletrônico do usuário que receberá a mensagem.
        + idUser: Código do usuário que receberá a mensagem.

Métodos:

* + - * + EnviaMensagem(): Método responsável pelo envio efetivo da mensagem.
        + BuscaEmail(): A partir do código do usuário, busca seu e-mail.

### Classe FeedBack

Esta classe contém as informações das possíveis avaliações feitas pelos usuários. É a classe pai para as classes FeedBackConteudo e FeedBackEstrutura.

Atributos:

* + - * + idFeedBack: Guarda o código de controle da avaliação enviada.
        + nota: É o o valor da nota da avaliação.
        + dataFeedback: Data da avaliação, para controle.
        + comentário: Guarda o possível comentário feito na avaliação.
        + idUser: Código do usuário que está fazendo a avaliação.

Métodos:

* + - * + AnalisarFeedback(): Método responsável pela análise da nota da avaliação. Nele é implementado a verificação que garante se o item deve continuar habilitado ou não.
        + BloquearAcesso(): Caso o item avaliado seja classificado como inapto, esse método é quem efetiva o bloqueio e envia uma mensagem para o usuário sugerindo a criação de um Anti-Pattern, usando o método EnviarMensagem().
        + EnviarMensagem(Mensagem msg): Método que efetivamente envia a mensagem para o usuário.

Dependências:

* + - * + Relação de composição com a classe Usuario.

### Classe FeedBackConteudo

Atributos:

* + - * + Herda todos os atributos da classe FeedBack.
        + padrão: atributo que pode referenciar o conteúdo de um Pattern ou AntiPattern.

Métodos:

* + - * + Herda todos os métodos da classe FeedBack.

### Classe FeedBackEstrutura

Atributos:

* + - * + Herda todos os atributos da classe FeedBack.
        + estrutura: atributo que pode referenciar a estrutura de um Pattern ou AntiPattern.

Métodos:

* + - * + Herda todos os métodos da classe FeedBack.

### Classe Estrutura

Essa classe é base para as classes Padrao e Persona.

Atributos:

* + - * + titulo: é o atributo que guarda o nome da estrutura.
        + idUserAutor: atributo que guarda o código do usuário criador.
        + dataCriacao: Data da criação da estrutura.
        + vetAtributos[\*]: vetor de atributos que não os mínimos.
        + idEstrutura: É o atributo que armazena o código da estrutura.
        + tpEstrutura: Tipo da estrutura(Pattern ou AntiPattern).

Métodos:

* + - * + Similaridade(Estrutura struct): método que calcula a similaridade entre o objeto instanciado e o parâmetro.

Dependências:

* + - * + Relação de composição com a classe Usuario.

### Classe Padrao

Essa classe é base para as classes Pattern e AntiPattern.

Atributos:

* + - * + Herda todos os atributos da classe Estrutura.
        + problema: atributo que armazena a descrição textual do problema enfrentado.
        + contexto: atributo que armazena a descrição textual do contexto onde esse problema é encontrado.
        + solução: Passos adotados para a solução do problema e suas respectivas personas vinculadas.
        + vetPersonas[\*]: vetor de personas vinculadas a esse padrão.

Métodos:

* + - * + Herda todos os métodos da classe estrutura.

### Classe Pattern

Essa classe representa o Pattern no sistema. Todos os atributos e métodos herdados passam agora a representar itens de um Pattern.

Atributos:

* + - * + Herda todos os atributos da classe Padrao.

Métodos:

* + - * + Herda todos os métodos da classe Padrao.

### Classe AntiPattern

Essa classe representa o AntiPattern no sistema. Todos os atributos e métodos herdados passam agora a representar itens de um AntiPattern.

Atributos:

* + - * + Herda todos os atributos da classe Padrao.
        + barreiras: atributo que armazena a declaração textual das barreiras e limitações que usuário está enfrentando.
        + sintomasConseq: armazena textualmente os sintomas que descrevem os erros encontrados e as conseqüências para os casos onde o anti-pattern seja seguido.
        + recomendacoes: armazena a forma como a solução pode ser encontrada.

Métodos:

* + - * + Herda todos os métodos da classe Padrao.

### Classe Usuario

Essa classe armazena todas as informações necessárias para o sistema funcionar, no que tange o usuário.

Atributos:

* + - * + idUser: código do usuário no sistema.
        + nome: nome do usuário no sistema.
        + username: nome de usuário usado para efetuar o login no sistema.
        + senha: em conjunto com o nome de usuário, é usada para acessar o sistema.
        + dataNasc: data de nascimento do usuário.
        + dataCriacao: data da criação do cadastro do usuário no sistema.
        + dataUltimoAcesso: data do ultimo login do usuário no sistema. Existe para poder controlar a fidelização e para estudos de perfis.

Métodos:

* + - * + VerificaCampos(): Método criado para validar campos obrigatórios.

### Classe Personas

Essa classe guarda as informações sobre os perfis usados no sistema, seja na descrição de uma solução ou no vínculo com outros dos conceitos englobados pelo APPP. Todos os atributos e métodos herdados da classe Estrutura passam agora a representar itens de um Pattern.

Atributos:

* + - * + Herda todos os atributos da classe Estrutura.
        + descricao: declaração textual da descrição dessa persona.

Métodos:

* + - * + Herda todos os métodos da classe Estrutura.

### Classe Solucao

Essa classe armazena as informações da solução adotada para Patterns ou Anti-Patterns, direcionadas, como o tema deste trabalha propõe, a determinados perfis de usuário.

Atributos:

* + - * + idSolucao: guarda o código da solução no sistema.
        + descSolucao: descrição detalhada da solução adotada.
        + persona: indica a Persona á qual essa solução se relaciona.

Métodos:

* + - * + Construtores, Getters e Setters.

Dependências:

* + - * + Relação de composição com a classe Persona.

## Diagramas de Seqüência

Nesta seção, será apresentado os diagramas de seqüência dos principais cenários identificados durante a construção desta monografia.

### Login

Na representa o processo encontrado no cenário para efetuar o *login* no sistema ou realizar o processo de recuperação do sistema.

* Através da classe “Usuario” o usuário informa seus dados e transmite para classe “Login” os dados necessários para a sua realização.
* A classe “Login” acessa o banco de dados por intermédio da classe “DAO” e verifica se os dados estão cadastrados no banco, nos casos positivos são gravados logs de acesso ao sistema.
* Para a realização do processo de recuperação de senha o usuário, através da classe “Usuario” acionará a classe “Login” com o método “RecuperarLogin()” que irá buscar no Banco de Dados os dados necessário e enviará para e-mail cadastrado do usuário.

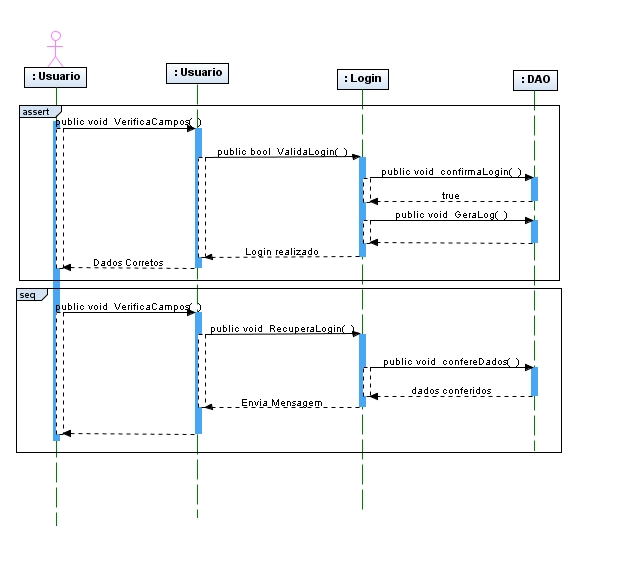


Figura 11. Diagrama de Seqüencia: Login  
Fonte: os autores.

### Cadastro de Usuário

Na é apresentado o processo encontrado no cenário para o cadastro de um novo usuário no sistema.

* Através do método “PreencheCadastro()” o usuário enviará ao sistema os dados necessários para que seja realizado seu cadastro.
* A classe “Usuario” enviará os dados para inserção destes no banco de dados por meio da classe “DAO”.
* No fim do cadastro, a classe “Usuario” informará a classe “Mensagem” o email do novo usuário, e esta enviará a mensagem de confirmação de cadastro, para que seja efetuado a ativação do mesmo.

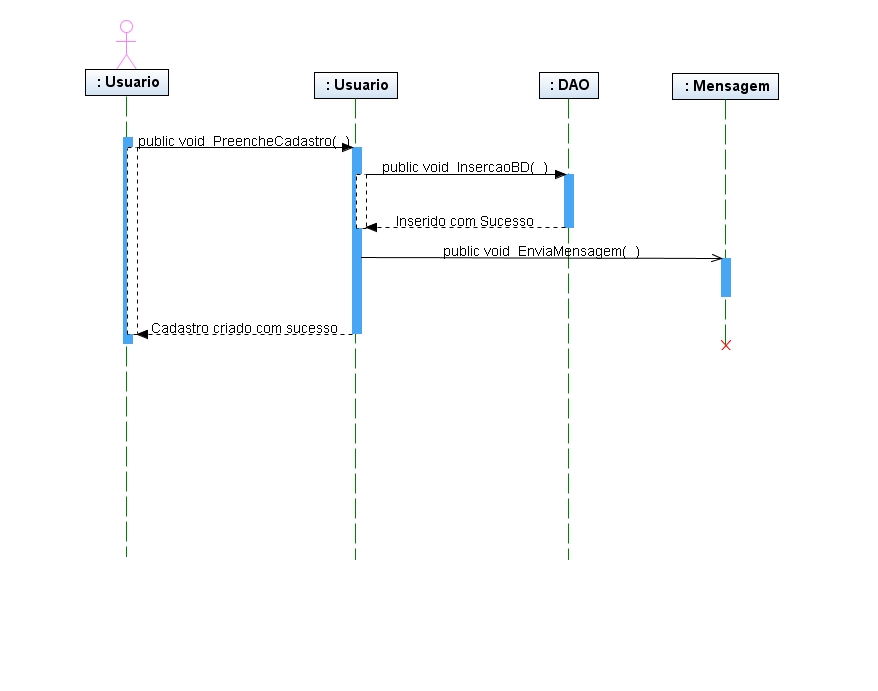


Figura 12. Diagrama de Seqüência: Cadastro de Usuário  
Fonte: os autores.

### Cadastro de Estrutura

Na apresenta o processo para a criação de uma nova estrutura de *APPP*.

* Antes da criação da estrutura o usuário deverá criar os atributos para esta, caso estes atributos ainda não estejam cadastrados no banco de dados. A criação do atributo ocorre pelo método “CriaAtributo()” da classe “Atributo” que faz a comunicação com o banco de dados através da classe “DAO”.
* Através da classe “Estrutura” o usuário buscará os atributos desejados e os inserirá na mesma.
* Ao salvar está estrutura o sistema percorrerá a base em busca de uma estrutura similar, e notificar o usuário em caso positivo, para que seja avaliado antes de salvar o novo.

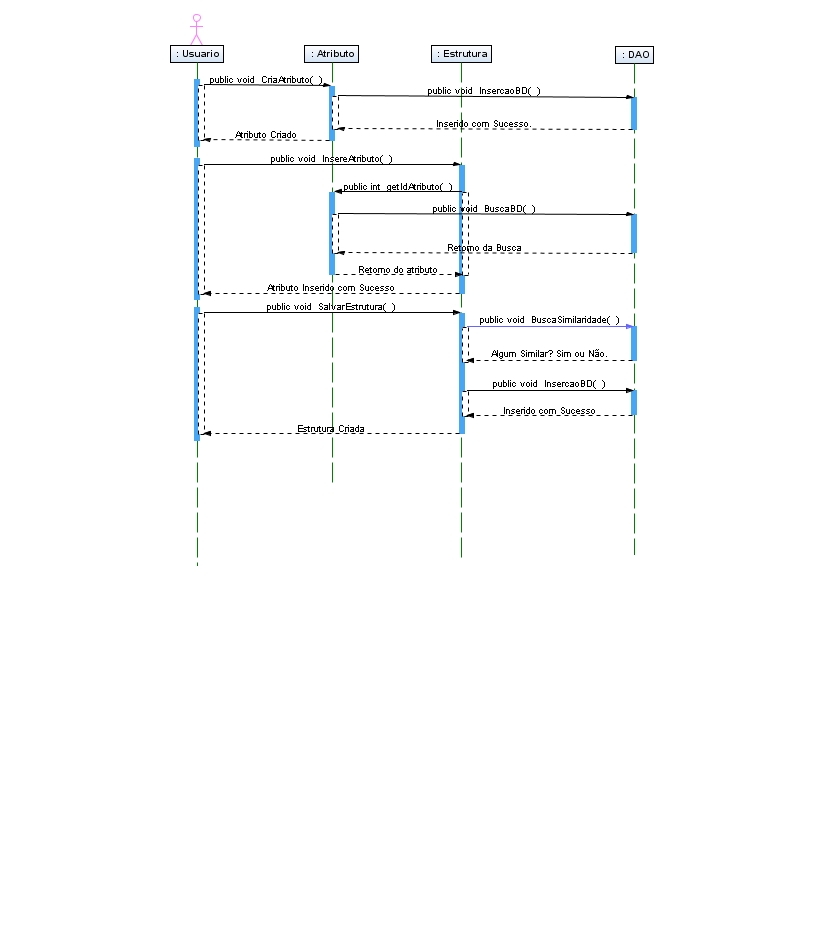


Figura 13. Diagrama de Seqüência: Cadastro de Estrutura  
Fonte: os autores.

### Cadastro do APPP (Conteúdo)

Na é demonstrado o processo de criação de um *APPP* através do diagrama de seqüência.

* O usuário selecionará o tipo de estrutura.
* Após o preenchimento dos valores para os atributos, através da classe “Padrão” o sistema buscará no banco de dados por algum *APPP* similar.
* Em caso positivo, o usuário deverá ser notificado para que possa verificar se tal *APPP* não contempla seu problema.

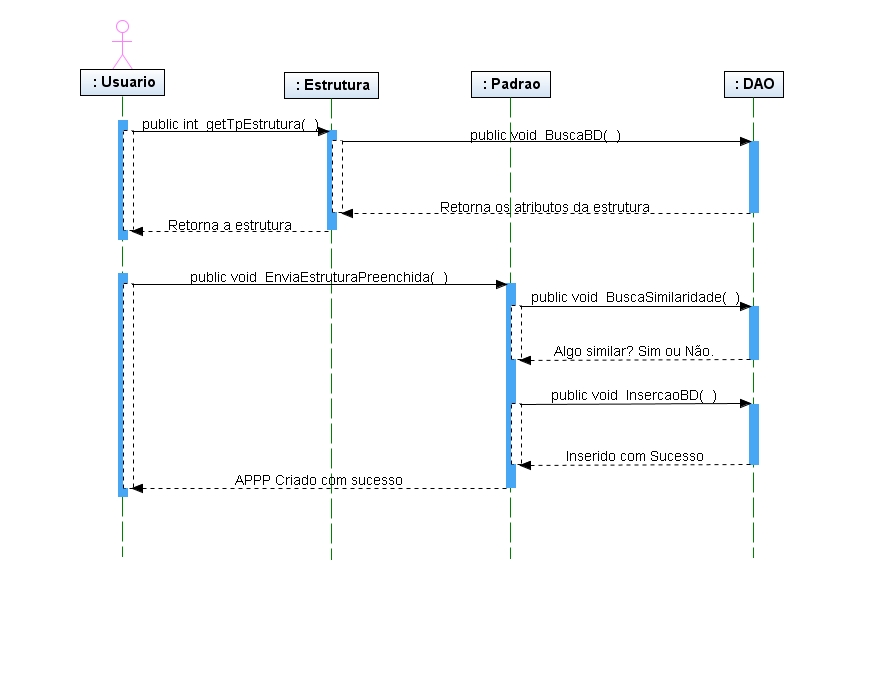


Figura 14. Diagrama de Seqüência: Cadastro do APPP  
Fonte: os autores.

### Relacionamento

Na é apresentado o processo de relacionamento automático entre *Patterns* e *Anti-Patterns*.

* Através da classe “Pattern” o sistema buscará os *Anti-Patterns* cadastrados no sistema.
* Após isso será realizado a busca pela similaridade entre os dois conceitos, se positivo será gravado no banco de dados o relacionamento entre eles.

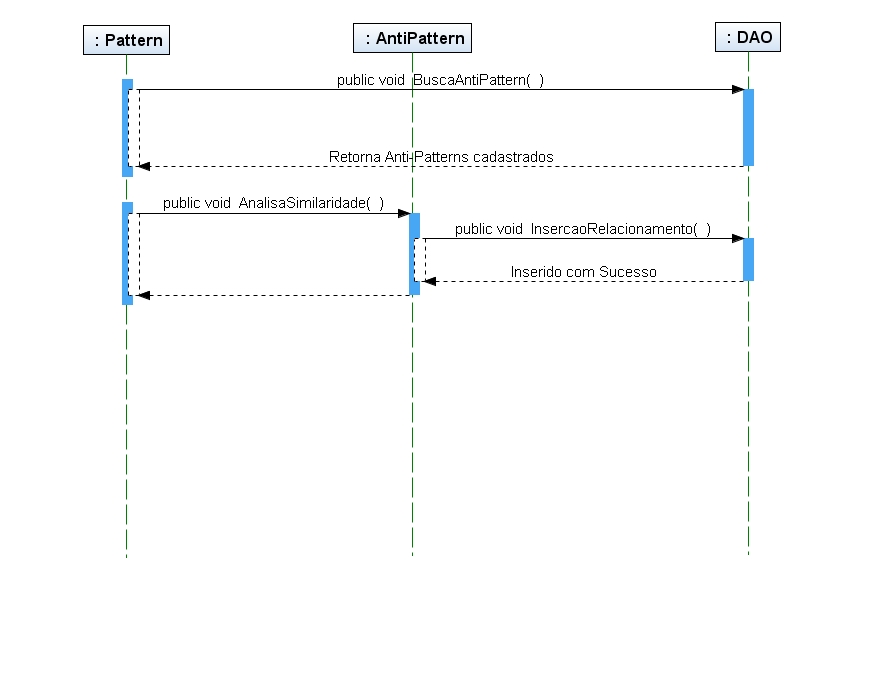


Figura 15. Diagrama de Seqüência: Relacionamento  
Fonte: os autores.

### Busca

Na é apresentado o cenário de busca que o usuário poderá realizar no sistema.

* O usuário entrará com o contexto do problema o qual ele busca solução.
* O sistema irá efetuar a busca no banco de dados e encontrando um ou mais resultados, estes serão exibidos ao usuário.

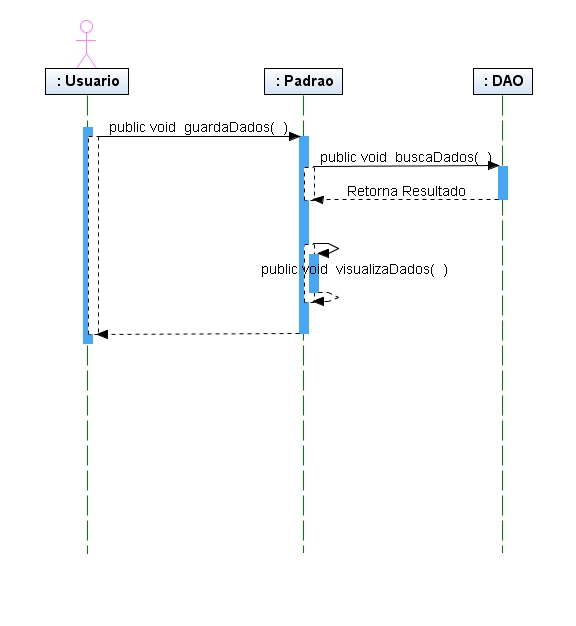


Figura 16. Diagrama de Seqüência: Busca  
Fonte: os autores.

### Feedback

Na é apresentado o processo do cenário encontrado para a realização da avaliação de uma estrutura ou um *APPP*.

* É dado pelo usuário, uma nota para a estrutura ou o *APPP*.
* O sistema realizará a analise e contabilização desta avaliação e irá inserir no banco.

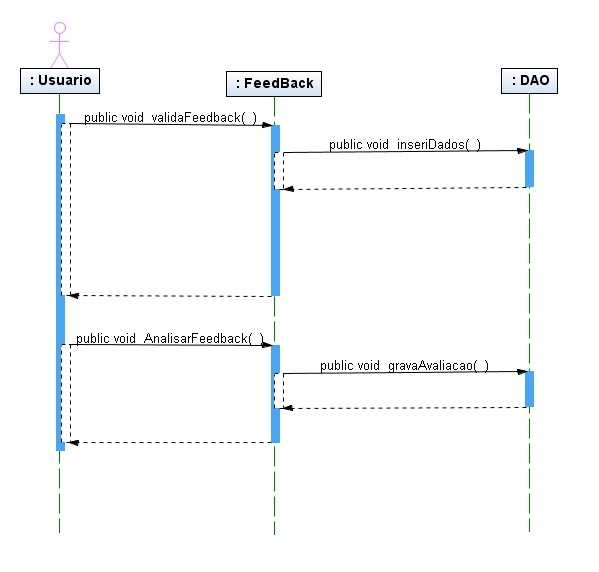


Figura 17. Diagrama de Seqüência: Feedback  
Fonte: os autores.

### Ações de Bloqueio

Na é apresentado o processo para bloqueio das estruturas e *APPP* com avaliações negativas.

* Primeiro o sistema efetua a busca dos resultados das analises de feedback
* Ele verifica se o resultado da avaliação está de acordo com o esperado, caso negativo ele bloqueia a estrutura ou o *APPP*.
* Em seguida, é enviado um e-mail informando ao criador que o *APPP* ou estrutura foi bloqueado.

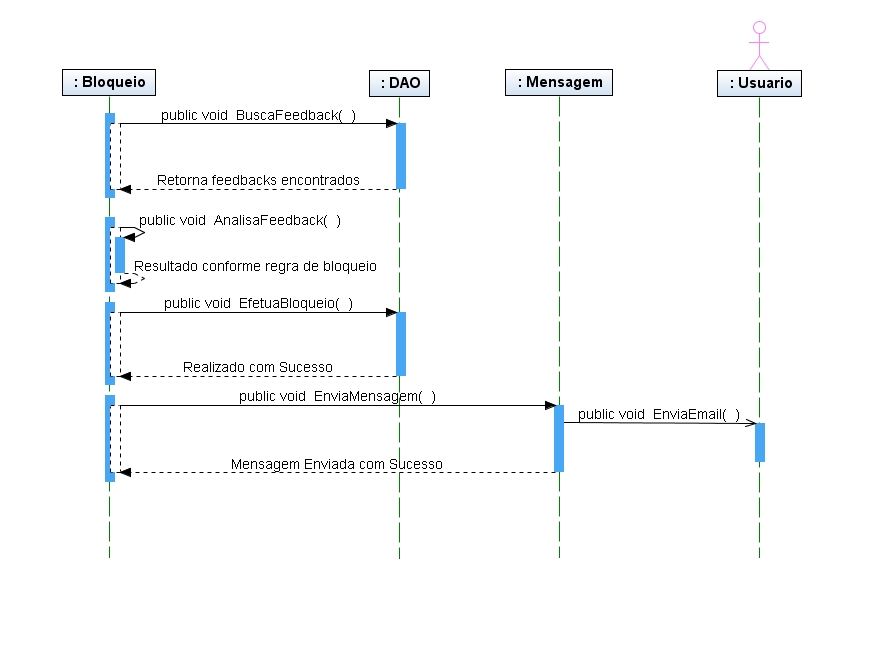


Figura 18. Diagrama de Seqüência: Ações de Bloqueio  
Fonte: os autores.

# banco de dados

## Modelo Entidade Relacionamento

O modelo abaixo demonstra o diagrama MER (Modelo Entidade Relacionamento) do banco de dados do sistema.

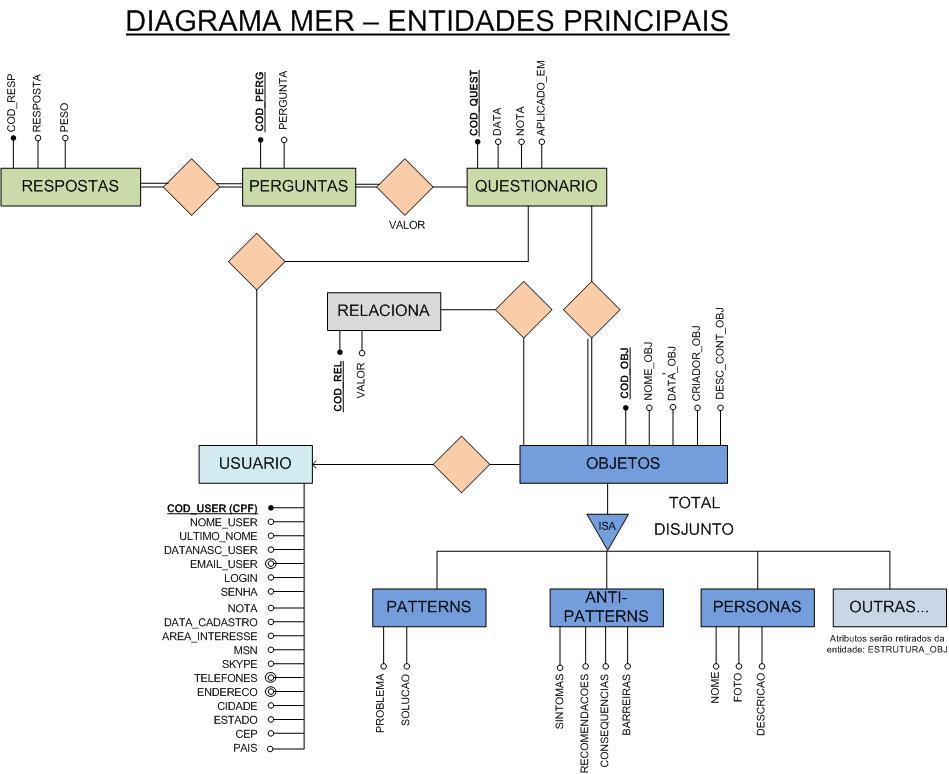


Figura . Modelo Entidade Relacionamento

# Avaliação do Sistema

## Avaliação da Interface

### Avaliação Heurística

O método de Avaliação Heurística de Nielsen consiste em avaliar uma interface ou um produto segundo regras específicas, descobrindo e avaliando o nível de severidade para cada erro encontrado. As heurísticas são recomendações em formato de afirmações e/ou premissas que indicam bons procedimentos de construções de interface, preservando a usabilidade do sistema.

### Testes com o Usuário

TBD

# Planejamento

Nesta seção, é apresentado o cronograma e tecnologia adotada para o desenvolvimento deste projeto.

## Cronograma

A demonstra o cronograma para o desenvolvimento dos principais módulos do sistema. Ressalta-se ainda que em paralelo com o desenvolvimento do software, a monografia será constantemente revisada e, eventuais ajustes serão feitos pela equipe.

Tabela 6. Cronograma de Desenvolvimento do Projeto



Fonte: os autores

## Tecnologias utilizadas

As tecnologias a serem adotadas para o desenvolvimento deste software são:

* Java

Java é uma linguagem orientada a objetos bastante difundida atualmente no mercado de desenvolvimento de softwares, trata-se de uma ferramenta de fácil acesso com documentações e componentes em sua grande maioria feitos em código aberto, podendo ser facilmente manipulados. A linguagem apresenta ainda uma considerável viabilidade no que se refere a suporte técnico por parte da SUN, além de todo material de apoio obtido por todos da equipe durante o desenrolar natural do curso de Ciência da Computação.

* SGBD

Este item será definido posteriormente após estudos de caso. Efetuaremos testes de performance e conexão além da facilidade de aplicação de funções já existentes em SGBDs como Oracle, MySql e Sql Server.

# RESULTADOS ESPERADOS

Com a ferramenta desenvolvida espera-se a viabilização da documentação de *Anti-Patterns, Patterns, Personas* o vínculo entre eles e, com isso, o compartilhamento de experiência adquirida e soluções específicas para diferentes perfis de usuários.

# CONCLUSÃO PARCIAL

Com base nas pesquisas realizadas sobre os conceitos abordados neste projeto de formatura conclui-se que o objetivo proposto é possível e já é claro como efetuar o vínculo do APPP, bem como o processo de implementação da ferramenta e suas funcionalidades.

# Referências

**ABEL, MARA. 1996.** Um Estudo Sobre Raciocínio Baseado em Casos. 1996.

**AKROYD, M. 1996.** Anti Patterns Session Notes. *Object World West, San Francisco.* 1996.

**ALEXANDER, C. et al. 1977.** A Pattern Language: Towns, Building, Constructions. *Oxford University Press, New York.* 1977.

**ANACLETO, J. C., et al. 2007.** Cognitor: Um Framework Baseado na Linguagem de Padrões Cog-Learn. *Revista Brasileira de Informática na Educação.* 2007, Vols. 15, p. 32/3-43.

**ANDRADE, R. M. C., et al. 2005.** IIMPaR – Uma Interface de Integração de Modelos de Padrões de Software para o Rose. 2005.

**AQUINO JR., P.T. 2008.** PICaP : padrões e personas para expressão da diversidade de usuários no projeto de interação. *Tese (Doutorado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.* Ed. Rev. - 224p, 2008.

**BILJON, J. V., et al. 2004.** The use of anti-patterns in human computer interaction. *In: Fulfilling the promise of ICT, Proceedings of SAICSIT 2004, edited by G Marsden, P Kotze & A Adesina-Ojo. ACM Conference Proceedings Series, SAICSIT, Pretoria.* 2004.

**BROWN, W. e al., et. 2000.** What's an Anti-Pattern? AntiPatterns. 2000.

**BUDINSKY, F.J., et al. 1996.** Automatic Code Generation from Design Patterns. *IBM Systems Journal 35.* 1996.

**COOPER, A. e REIMANN, R. M. 2003.** About Face 2.0 The Essentials of Interaction Design. *John Wiley & Sons.* 2nd edition, 2003.

**COOPER, A. 1999.** The inmates are running the asylum: Why high-tech products drive us crazy and how to restore the sanity. *Indianopolis, Ind.: Sams.* 1999.

**COSTA, D. M., et al. 2008.** DEAN Compositor de Personagens Fictícios. *Apresentação de Trabalho - Tese de Graduação.* 2008.

**GAMMA, E., et al. 1995.** *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley Professional, 1995. ISBN-13: 978-0201633610.

**GERBER, L. D. 1999.** Uma linguagem de padrões para o desenvolvimento de sistemas de apoio à decisão baseado em frameworks. *Tese de Mestrado apresentada na PUC - RS.* 1999.

**GILBRETH, F. B. 1911.** Motion study: a method for increasing the efficiency of the workman. *Imprenta New York : D. Van Nostrand Company.* 1911.

**GRESSE, CHRISTIANE von WANGENHEIM. 2003.** Raciocínio Baseado em Casos. 2003, 1.

**HANMER, R. 2003.** Introduction to Pattern Languages. *SugarLoafPLoP 2003, The Third Latin American Conference on Pattern Languages of Programming, Porto de Galinhas, PE.* 2003.

**KOTZÉ, P., et al. 2006.** Patterns, anti-patterns and guidelines – effective aids to teaching HCI principles? *in E T Hvannberg, J C Read, L Bannon, P Kotzé and W Wong (eds.), Inventivity: Teaching theory, design and innovation in HCI, University of Limerick.* 2006, 115-120.

**LONG, J. 2001.** Software Resuse Antipatterns. *IBM.* 2001.

**MARINHO, F., et al. 2003.** Uma Proposta de um Repositório de Padrões de Software Integrado ao RUP. *III Conferência Latino-Americana em Linguagens de Padrões para Programação.* 2003.

**MASIERO, A., LOPES, G. e GONÇALVES, M. H. P. 2008.** Anti-Patterns Apoiando a Documentação dos Problemas de Usabilidade. *Revista Hífen, SIMS - XIII Simpósio de Informática.* to appear, 2008.

**MICROSOFT CORPORATION. 2005.** Principles of Service Design: Service Patterns and Anti-Patterns. *MSDN.* Disponível online em msdn.microsoft.com/en-us/library/ms954638.aspx, 2005, último acesso em julho/2008.

**NIELSEN, J. 1993.** Usability Engineering. 1993.

**RIESBECK, CHRISTOPHER K. e SCHANK, ROGER C. 1989.** *Inside case-based reasoning.* Hillsdale, New Jersey : Lawrence Erlbaum Associates, 1989.

**SHNEIDERMAN, B. 2000.** Universal Usability, Pushing human-computer interaction. *COMMUNICATIONS OF THE ACM.* 5, 2000, Vols. 43, pp. 85-91.

**SILVA, E. R. e SCHIEL, U. 2004.** SAMOA Uma Ferramenta Para Detecção de Padrões de Projetos em Diagramas UML, na WEB. *Apresentação de Trabalho/Seminário - Tese de Mestrado.* 2004.

**TAYLOR, F. W. 1911.** The principles of scientific management. *Imprenta New York; London: Harper & brothers.* 1911.

**TIDWELL, J. 2006.** UI Patterns and Techniques. [Online] 2006. [Citado em: 18 de Novembro de 2008.] http://www.time-tripper.com/uipatterns/Tree-Table.

**WEHMEIER, S. 2000.** Oxford Advanced Learner's Dictionary. Oxford : Oxford University Press, 2000. 6.