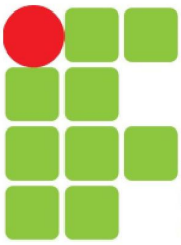


INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
PERNAMBUCO

CAMPUS RECIFE
DEPARTAMENTO ACADÊMICO DE CONTROLE DE SISTEMAS
ELETROELETRÔNICOS
TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

EDMILSON MANOEL GUILHERME DE SANTANA

# UM PROCESSO PARA IDENTIFICAÇÃO DE ANTI-PADRÕES DE DESEMPENHO EM SISTEMAS ARMAZENADOS NO PORTAL DE SOFTWARE PÚBLICO BRASILEIRO

RECIFE
2018

EDMILSON MANOEL GUILHERME DE SANTANA

# UM PROCESSO PARA IDENTIFICAÇÃO DE ANTI-PADRÕES DE DESEMPENHO EM SISTEMAS ARMAZENADOS NO PORTAL DE SOFTWARE PÚBLICO BRASILEIRO

*Trabalho de Conclusão de Curso desenvolvido no Curso de Tecnologia em Análise e Desenvolvimento de Sistemas, como parte dos requisitos para obtenção do Título de Tecnólogo em Análise e Desenvolvimento de Sistemas, no Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco, sob orientação da Professora Renata Freire de Paiva Neves.*

Orientador: *MSc. Renata Freire de Paiva Neves*
Co-Orientador: *MSc. Roberto Luiz Sena de Alencar*

RECIFE
2018

Ficha elaborada pela bibliotecária Emmely Cristiny Lopes Silva CRB4/1876

S232p
2018 Santana, Edmilson Manoel Guilherme de.
Um processo para identificação de anti-padrões de desempenho em sistemas armazenados no portal de software público brasileiro / Edmilson Manoel Guilherme de Santana. --- Recife: O autor, 2018.
53f. il. Color.

TCC (Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Pernambuco, Departamento Acadêmico de Controle de Sistemas Eletroeletrônicos - DASE, 2018.

Inclui Referências.

Orientadora: Professor M.e. Renata Freire de Paiva Neves

1.Anti-pradrões de desempenho. 2. Portal de software público brasileiro. 3. Gerenciamento de processos de negócio. 4. Engenharia de software. I. Título. II. Neves, Renata Freire de Paiva (orientadora). III. Instituto Federal de Pernambuco.

CDD 005 (21ed.)

Trabalho de Conclusão de Curso apresentado pelo aluno **Edmilson Manoel Guilherme de Santana** à coordenação de Análise e Desenvolvimento de Sistemas, do Instituto Federal de Pernambuco, sob o título de "***UM PROCESSO PARA IDENTIFICAÇÃO DE ANTI-PADRÕES DE DESEMPENHO EM SISTEMAS ARMAZENADOS NO PORTAL DE SOFTWARE PÚBLICO BRASILEIRO***", orientado pela Profa. **Msc. Renata Freire de Paiva Neves** e pelo **Prof. Msc. Roberto Luiz Sena de Alencar** e aprovado pela banca examinadora formada pelos professores:

Recife, 05 de Junho de 2018.

Profa. Msc. Renata Freire de Paiva Neves
CTADS/DASE/IFPE

Prof. Msc. Roberto Luiz Sena de Alencar
IFPE

Profa. Msc. Lizianne Priscila Marques Souto
CTADS/DASE/IFPE

Prof. Msc. José Paulo da Silva Lima
Faculdade Nova Roma

Edmilson Manoel Guilherme de Santana

*Dedico este trabalho a todos que colaboraram direta ou indiretamente em minha formação acadêmica.*

# AGRADECIMENTOS

Aos meus pais pelo apoio durante toda a minha formação profissional e pessoal. A esta instituição de ensino pela oportunidade de fazer o curso.

A todos os professores que acompanharam minha jornada acadêmica e foram essenciais à minha formação como profissional.

Aos meu orientadores, Renata Freire, Roberto Alencar e Arthur Carvalho, pela orientação, empenho e dedicação durante a elaboração deste trabalho.

À Clarice Queiroz, pelo apoio e paciência nas horas difíceis e durante todo esse tempo em que estive me formando enquanto profissional e pessoa.

A todos os amigos e amigas que estiveram comigo nessa jornada.

# RESUMO

O Portal de Software Público Brasileiro (SPB) fornece um serviço social e científico através de um repositório de *software*. Dada a importância deste serviço, observam-se oportunidades de melhoria através de propostas para avaliar a qualidade dos *softwares* residentes no portal, uma vez que este não possui um controle de qualidade e auditoria para os *softwares* que são submetidos, e nem são exploradas as carências destes no que se refere a qualidade. Este trabalho apresenta um processo que realiza a identificação e correção de problemas de desempenho, utilizando como referencial as características de anti-padrões de desempenho apresentadas na literatura. O processo proposto é modelado utilizando o *Business Process Model and Notation* (BPMN) e baseado em atividades de inspeção, *profiling* e medição. Com o objetivo de contribuir e enriquecer o conhecimento relacionado ao desempenho dos *softwares* armazenados no SPB, foi realizado um estudo de caso, possibilitando a identificação e correção de anti-padrões de desempenho, obtendo uma redução no tempo de resposta entre 86.0% e 96.0%. Foi possível integrar o processo de identificação e correção de anti-padrões no processo de desenvolvimento do *software* participante do estudo de caso, validando de forma empírica, a eficácia do processo proposto neste trabalho.



**Palavras-chave:** Anti-padrões de Desempenho. Portal de Software Público Brasileiro. Gerenciamento de Processos de Negócio. Engenharia de Software. Desempenho de Software.

ABSTRACT

The Brazilian Public Software Portal (SPB) provides a social and scientific service through a software repository. Given the importance of this service, opportunities for improvement are observed through proposals to evaluate the quality of the software residing in the portal, since it does not have a quality control and audit for the softwares that are submitted and are not exploited the needs of these in terms of quality. This work presents a process that performs the identification and correction of performance problems, using as reference the characteristics of performance anti-patterns standards presented in the literature. The proposed process is modeled using Business Process Model and Notation (BPMN) and based on inspection, profiling and measurement activities. In order to contribute and enrich the knowledge related to the performance of the software stored in the SPB, a case study was carried out, allowing the identification and correction of performance anti-patterns standards, obtaining a reduction in the response time between 86.0% and 96.0%. It was possible to integrate the process of identification and correction of anti-patterns in the process of development of the participant software of the case study, validating empirically, the effectiveness of the process proposed in this work.

**Keywords:** Performance Anti-patterns. Public Brazilian Software Portal. Business Process Management. Software Engineering. Software Performance.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

| | |
|---|---|
| **IDE** | Ambiente de Desenvolvimento Integrado |
| **API** | *Application Programming Interface* |
| **SOA** | Arquiteturas Orientadas a Serviços |
| **BPM** | *Business Process Management* |
| **BPMN** | *Business Process Model and Notation* |
| **MDE** | Engenharia Orientada a Modelos |
| **XML** | *eXtensible Markup Language* |
| **HQL** | *Hibernate Query Language* |
| **JPA** | Java Persistence API |
| **JPQL** | *Java Persistence Query Language* |
| **KPI** | *Key Performance Indicator* |
| **ORM** | Mapeamento objeto-relacional |
| **MP** | Ministério do Planejamento, Orçamento e Gestão |
| **MCD** | *Model Clone Detection* |
| **SPB** | Portal de Software Público Brasileiro |
| **GGAS** | Sistema de Gestão Comercial de Gás Natural |
| **SQL** | *Structured Query Language* |
| **WEB** | *World Wide Web* |

SUMÁRIO

# 1 INTRODUÇÃO

*Software* Público refere-se ao tipo de *software* que adota um modelo de licença livre e disponibiliza o código-fonte abertamente ao público. Este tipo de *software* pode ser disponibilizado para uso através de um repositório de *software* público, tais como o Portal de Software Público Brasileiro (SPB). O SPB disponibiliza um serviço através do fornecimento de um espaço de compartilhamento de soluções em *software*. O uso de *softwares* do portal no setor público, permite uma gestão de recursos de *software* mais racionalizada e consequentemente o reforço da política de *software* livre no setor público (MEIRELES, 2015).

As soluções em *software* fornecidas pelo portal já permitiram informatizar o sistema educacional de escolas municipais, através por exemplo, do uso do *software* público i-Educar. Disponibilizado pelo Ministério do Planejamento, Orçamento e Gestão (MP) através do Portal de Software Público, o i-Educar tem como objetivo o gerenciamento das ações da rede municipal de ensino e gerou uma economia anual de R$ 2,4 milhões ao município de Monte Alegre (BRASIL, 2015).

A contribuição social obtida através dos serviços do SPB é de fato relevante. Entretanto, estes benefícios imediatos podem ser suplantados pelas consequências negativas resultantes de problemas referentes a qualidade do *software*. A qualidade de um *software* pode ser definida como o grau em que o mesmo satisfaz os seus requisitos especificados (RADATZ; GERACI; KATKI, 1990). A qualidade não se restringe apenas a conformidade com as funcionalidades especificadas, mas também das restrições aplicadas as mesmas, tais como o grau de desempenho em que o *software* deve oferecer tais funcionalidades.

Um *software* que possui um requisito não-funcional de desempenho de baixa qualidade é impactado por uma variedade de consequências negativas, pois os *softwares* com problemas de desempenho geralmente não entregam seu benefício destinado a seus usuários, criando um custo de tempo e dinheiro, e uma perda de elogios de seus usuários (MOLYNEAUX, 2009). A imagem da organização sofre por conta dos problemas de desempenho em um produto de *software*, visto que os usuários associarão estes problemas aos produtos produzidos pela organização mesmo, que o problema seja solucionado (SMITH; WILLIAMS, 2001a).

Uma das causas que acarretam um *software* de baixa qualidade é o processo de desenvolvimento, visto que a qualidade do mesmo é diretamente relacionada com a qualidade de seu processo de desenvolvimento e bons processos são mais suscetíveis de conduzir um *software* de boa qualidade (SOMMERVILLE, 2011). Entretanto, o SPB não possui métodos, ferramentas e procedimentos que realizem um controle de qualidade e auditoria dos *softwares* que são submetidos para o seu repositório, e consequentemente, não é possível aferir a qualidade das soluções em *software* que residem no portal.

Para que o SPB mantenha a boa qualidade dos *softwares* que estão e que serão submetidos, se faz necessária a existência de um controle da submissão destes. Este controle pode ser obtido na forma de um processo de auditoria da qualidade do *software*. E inserido neste processo, etapas que auditem a qualidade de desempenho.

Entretanto, para que esse processo seja implantado é necessário uma compreensão dos problemas dos *softwares* do portal, tais como os problemas em desempenho. Estes problemas devem ser identificados para que as etapas que auditem o desempenho sejam construídas com base nas carências apresentadas pelos *softwares* já existentes no portal.

Problemas de desempenho podem ser resultantes de anti-padrões de desempenho. O conceito de anti-padrão é semelhante ao de um padrão de projeto: uma solução geral para um problema recorrente. Entretanto, o anti-padrão fornece um solução que é efetiva superficialmente e que resulta em problemas na qualidade do *software*. (RISING, 1998).

Os anti-padrões de desempenho são relevantes para as etapas de auditoria de *software*. Nestas etapas é importante que o problema de desempenho seja estruturado, possua sua definição bem estabelecida na literatura e consequentemente seja fácil de se rastrear, de forma que possíveis soluções sejam sugeridas.

As atividades necessárias para identificar um anti-padrão de desempenho, dependem do contexto do *software* avaliado. Contexto esse, que pode ser influenciado tanto pela arquitetura do *software*, quanto pelo paradigma de desenvolvimento do mesmo (DIN; AL-BADAREEN; JUSOH, 2012). Portanto, um processo para a identificação de anti-padrões aplicado ao SPB deve ser voltado para o contexto de negócio do mesmo, visto que este possui *softwares* com diferentes características.

Um processo de identificação de anti-padrões de desempenho alinhado com os objetivos do SPB costuma agregar valor aos clientes. O trabalho de entrega de valor compõe um processo de negócio em que as atividades deste processo podem ser modeladas utilizando a disciplina de Gerenciamento de Processos de Negócio (BPM) (BENEDICT et al., 2013).

A modelagem de um processo para identificação de anti-padrões de desempenho documenta uma representação do mesmo e facilita sua integração a um processo que realize o controle de qualidade dos softwares do SPB. A integração destes processos permite que a identificação de um anti-padrão de desempenho seja utilizada como critério de rejeição da submissão de um *software* ao repositório do SPB.

## 1.1 OBJETIVOS

### 1.1.1 Objetivo Geral

Dado o exposto, este trabalho tem como objetivo a proposição e implementação de um processo para a identificação e correção de anti-padrões de desempenho em uma aplicação do SPB. Este processo deriva de experimentos empíricos que consistem da aplicação das práticas difundidas na literatura alinhadas com as necessidades apresentadas pelo SPB. A aplicabilidade do processo será verificada através de um estudo de caso realizado em um *software* do SPB. Com o objetivo de validar empiricamente o processo proposto, será realizada a integração do mesmo com o processo de desenvolvimento do *software* participante do estudo de caso.

### 1.1.2 Objetivos Específicos

- Identificar e corrigir problemas de desempenho de um *software* do SPB, apresentando o impacto destes problemas para o usuário final.
- Melhorar a qualidade do processo de desenvolvimento do *software* participante do estudo de caso.
- Fornecer ao SPB um processo que permite avaliar a qualidade do desempenho de seus *softwares*.

- Contribuir com a comunidade científica através da publicação deste trabalho em uma revista científica.

É importante definir que neste trabalho, não são realizados comparativos entre problemas de desempenho, nem definidos os principais problemas que os *softwares* do SPB apresentam, mas são expostas as carências destes no que se refere a qualidade em desempenho e é apresentado o impacto dos problemas mensurados para o usuário final.

## 1.2 ESTRUTURA DO TRABALHO

Este trabalho possui 4 capítulos além desta introdução. O capítulo 2 apresenta conceitos apresentados pela literatura necessários para um maior entendimento das seções subsequentes do trabalho. Já no capítulo 3 é apresentada a solução proposta, definindo a metodologia do trabalho. No capítulo 4 são apresentados os experimentos realizados em um *software* do SPB e os resultados obtidos a partir destes experimentos. Por fim, o capítulo 5 apresenta as conclusões e os trabalhos futuros.

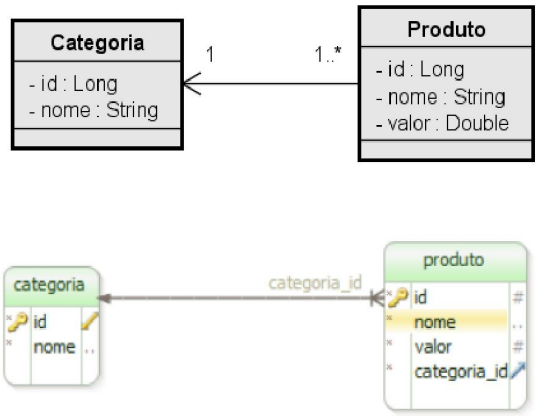
# 2 REFERENCIAL TEÓRICO

## 2.1 MAPEAMENTO OBJETO-RELACIONAL

Esta seção visa introduzir o conceito de Mapeamento objeto-relacional (ORM), visto que a literatura apresenta o uso do mesmo como uma potencial causa de problemas de desempenho (CHEN et al., 2014) e que tal paradigma é abordado no *software* do estudo de caso. O ORM provê aos desenvolvedores uma abstração conceitual para mapear registros do banco de dados em objetos. Este mapeamento envolve mapear uma classe a uma tabela do banco de dados, especificando quais atributos correspondem à quais colunas da tabela, além dos relacionamentos entre as classes e suas respectivas tabelas.

O ORM permite que os desenvolvedores mantenham o foco na lógica do negócio sem se preocupar com os detalhes de acesso a banco de dados. A Figura 1 apresenta os paradigmas envolvidos no ORM, sendo estes um esquema do banco de dados, representando o modelo relacional, e um diagrama de classes, representado o paradigma orientado a objetos.

**Figura 1: Paradigma Objeto/Relacional**




Fonte: O autor (2018)

A persistência de objetos no banco de dados consiste em salvar os objetos em um repositório de dados e recriar estes objetos posteriormente. No contexto da linguagem Java, a persistência normalmente refere-se ao mapeamento e armazenamento

de objetos em um banco de dados, utilizando a *Structured Query Language* (SQL). Entretanto, o paradigma objeto/relacional possui incompatibilidades, dentre estas estão: os diferentes níveis de granularidade dos dados entre as classes do modelo de domínio e do banco de dados; a ausência de um mecanismo de herança e polimorfismo no paradigma relacional, etc (BAUER; KING, 2016). O mapeamento objeto-relacional busca solucionar estas incompatibilidades do paradigma objeto/relacional através de *frameworks* como o Hibernate.

### 2.1.1 Hibernate

O Hibernate é um *framework* de mapeamento objeto-relacional que realiza o mapeamento de classes e tipos de dados Java para tabelas do banco de dados e tipos de dados SQL, respectivamente. O Hibernate é um provedor da Java Persistence API (JPA), o que significa que o mesmo implementa a especificação JPA. O JPA descreve as interfaces com as quais o provedor opera e quais os metadados para o mapeamento objeto-relacional (anotações Java ou descritores XML). O código-fonte abaixo apresenta o diagrama de classes apresentado na Figura 1 utilizando o mapeamento JPA (MIHALCEA et al., 2017).

```
@Entity
public class Produto {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(name="nome",  nullable=false)
        private String nome;

        @Column(name="valor",  nullable=false)
        private Double valor;

        @ManyToOne(fetch = FetchType.LAZY, optional = false)
        private Categoria categoria;

}

@Entity
public class Categoria {

        @Id
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private Long id;

        @Column(name = "nome", nullable = false)
        private String nome;

}
```

No código-fonte apresentado, o mapeamento objeto-relacional é realizado utilizando metadados definidos por meio de anotações Java. Através da anotação *@Entity*, uma classe, ou entidade JPA, é mapeada em uma tabela do banco de dados. Por padrão, a especificação utiliza o nome da classe como o nome da tabela. O atributo que representa a chave primária da tabela, e consequentemente o identificador da entidade, recebe a anotação *@Id*. A anotação *@GeneratedValue* especifica que o valor do identificador da entidade é gerado automaticamente usando uma coluna de identidade, que, dependendo do banco de dados, pode ser um auto-incremento (MIHALCEA et al., 2017).

O mapeamento objeto-relacional consiste além do mapeamento dos atributos da classe em colunas da tabela do banco de dados, de relacionamento entre entidades em uma direção ou de forma bidirecional. No modelo apresentado na Figura 1, existe um relacionamento muitos-para-um, entre Produto e Categoria. Este relacionamento é implementado no código-fonte através da anotação *@ManyToOne* na entidade Produto. No contexto de desempenho, ainda pode ser observado nesta associação, a configuração do carregamento por demanda, através do atributo *fetch* da anotação *@ManyToOne*. Está estratégia de busca define que a categoria do produto não é obtida ao consultá-lo do banco de dados, mas sim quando for utilizada (MIHALCEA, 2016).

Utilizando o Hibernate, pode-se expressar consultas SQL em termos orientados a objetos, utilizando classes e propriedades das classes. Para isso, utiliza-se a *Java Persistence Query Language* (JPQL), de acordo com a especificação JPA, ou a *Hibernate Query Language* (HQL), fornecida pelo provedor Hibernate. Antes da especificação JPA existir, a linguagem de consulta do Hibernate era chamada de HQL. Entretanto, ambas, HQL e JPQL, possuem seus fundamentos de sintaxe e semântica idênticos (BAUER; KING, 2016).

O Hibernate também oferece, além do HQL, consultas utilizando a API Criteria. Nas consultas com o *Criteria* são utilizadas interfaces e classes para representar partes estruturais de uma consulta, o que incluí cláusulas de seleção, ordem, condicionais, etc. Também é possível utilizar o SQL nativo do banco de dados, o que é útil para quando se deseja utilizar recursos específicos do banco de dados (MIHALCEA et al., 2017). A Tabela 1 apresenta uma consulta no banco de dados para o modelo apresentado na Figura 1. O código-fonte desta consulta utilizando a API Criteria pode ser visualizado

abaixo.

```
CriteriaBuilder builder = entityManager.getCriteriaBuilder();

CriteriaQuery<Produto> criteria = builder.createQuery(Produto.class);

Root<Produto> i = criteria.from(Produto.class);
Join<Produto, Categoria> join = i.join("categoria");

ParameterExpression<String> parametroNomeCategoria =
                            builder.parameter(String.class);

TypedQuery<Produto> query = entityManager.createQuery(criteria.select(i).where(builder.equal(join.get("nome"),
      parametroNomeCategoria)));

query.setParameter(parametroNomeCategoria, "Alimentos");
```

**Tabela 1: Consulta utilizando SQL e JPQL**

| SQL | JPQL |
|---|---|
| SELECT * FROM PRODUTO as produto JOIN CATEGORIA AS categoria ON produto.categoria_id = categoria.id WHERE categoria.nome = 'Alimentos' | select produto from Produto as produto join fetch produto.categoria as categoria where categoria.nome = 'Alimentos' |

Fonte: O autor (2018).

## 2.2 ANTI-PADRÕES DE DESEMPENHO

Segundo Alexander et al. (1977): "cada padrão descreve um problema no nosso ambiente e o cerne da solução, de tal forma que você pode usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira". Dentro desta afirmação, encontra-se os elementos que representam um padrão de projeto: uma solução para um problema comum em um determinado contexto (GAMMA et al., 2000).

Padrões de projeto e anti-padrões estão relacionados. Um anti-padrão é originado da aplicação de um padrão em um contexto equivocado, resultado da falta de conhecimento ou experiência na tentativa de resolver um tipo de problema. Porém, diferente de um padrão de projeto, um anti-padrão possui duas soluções. A primeira é uma solução para um problema, que gera consequências negativas. A segunda é conhecida como: *Refactored Solution*, e descreve como resolver a solução problemática (BROWN et al., 1998).

Anti-padrões e padrões de projeto não destacam explicitamente o impacto no desempenho do *software*. Entretanto, os anti-padrões definidos por Smith e Williams

(SMITH; WILLIAMS, 2000; 2001b; 2002; 2003) destacam o impacto desses problemas no desempenho, identificando potenciais problemas em arquiteturas e projetos de *software* e concentrando-se no nível certo de abstração. Estes anti-padrões identificam o problema fundamental e sua solução ao invés de uma ”correção” específica que pode estar desatualizada ao longo do tempo.

Observando alguns anti-padrões apresentados na literatura, pode-se observar que alguns destes são consequência de práticas realizadas no desenvolvimento de acordo com o paradigma orientado a objetos. Tais como observado nos anti-padrões *The ”God” Class* e *Excessive Dynamic Allocation*. O anti-padrão *Excessive Dynamic Allocation* aborda os impactos no desempenho de um *software*, ao realizar criação e destruição frequentes e desnecessárias de objetos da mesma classe (SMITH; WILLIAMS, 2000). O anti-padrão *The ”God” Class* apresenta o problema de tráfico excessivo de mensagens, através da execução de métodos de classes que desempenham a maior parte do trabalho em um *software*, relegando funções secundárias a outras classes (SMITH; WILLIAMS, 2000).

Na revisão da literatura, foi possível identificar anti-padrões que descrevem problemas de desempenho relacionados ao processamento executado em funções de um *software*. Enquanto o anti-padrão *Unnecessary Processing* informa dos problemas de desempenhos resultantes de um processamento que é executado, mas que não é necessário. O anti-padrão *The Ramp* apresenta as consequências para o desempenho, quando a quantidade de processamento necessária para realizar determinada tarefa, aumenta ao longo do tempo (SMITH; WILLIAMS, 2002).

Na arquitetura de um *software*, também podem ser identificados anti-padrões de desempenho, como é demonstrado através dos anti-padrões *Unbalanced Processing* e *Falling Dominoes*. O anti-padrão *Falling Dominoes*, ocorre quando uma falha em um componente causa falhas de desempenho em outros, devido ao componente falho não estar isolado ou muito acoplado aos demais (SMITH; WILLIAMS, 2003). O anti-padrão *Unbalanced Processing* está intimamente relacionado à escalabilidade do *software* e ocorre quando uma tarefa a ser realizada deve aguardar até a conclusão de determinado processamento ou execução de outra tarefa. Este anti-padrão pode se manifestar em arquiteturas ”Pipe and Filter” e/ou em *softwares* que realizam processamento concorrente ou extensivo (SMITH; WILLIAMS, 2002).

Na integração do *software* com um banco de dados, podem surgir problemas de desempenho resultantes de anti-padrões, tais como o *Circuitous Treasure Hunt* e o *Empty Semi Trucks*. Os anti-padrões *Circuitous Treasure Hunt* e *Empty Semi Trucks* estão relacionados ao acesso ao banco de dados, através de consultas SQL. O *Empty Semi Trucks* ocorre quando um número excessivo de requisições é necessária para executar uma tarefa, que poderia ser realizada em uma única requisição. Esta requisição pode ser interpretada como uma consulta realizada no banco de dados (SMITH; WILLIAMS, 2003). Já o *Circuitous Treasure Hunt*, ocorre quando o *software* e o banco de dados realizam processamento adicional ao recuperar os dados de uma tabela do banco de dados, em seguida, utilizam esses resultados para pesquisar uma segunda tabela, recuperam os dados dessa tabela e assim, até que os ”resultados finais”sejam obtidos (SMITH; WILLIAMS, 2000).

### 2.2.1 Anti-padrões e o Mapeamento Objeto-Relacional

Os anti-padrões de desempenho abordados por Smith e Williams (SMITH; WILLIAMS, 2000; 2001b; 2002; 2003), apresentam problemas de desempenho em diferentes contextos, desde problemas arquiteturais do *software* a práticas de desenvolvimento do mesmo. Observando esta característica, autores propuseram trabalhos abordando anti-padrões em contexto específicos, tais como a relação dos mesmos com o paradigma objeto/relacional.

Wegrzynowicz (2013) afirma que na literatura, os anti-padrões não são retratados em trabalhos focados na camada de dados. Ressaltando que os *frameworks* de mapeamento objeto relacional se tornam cada vez mais comuns, o autor aponta que o uso indiscriminado destes, podem gerar problemas na comunicação com o banco de dados, produzindo consultas ao banco em excesso ou ineficientes. O objeto de estudo do autor foram as associações um-para-muitos mapeadas utilizando o *framework* Hibernate, nas quais ele identifica problemas de desempenho comuns e também define cinco novos anti-padrões de desempenho, assim como provê recomendações de como solucioná-los. A Tabela 2 apresenta os anti-padrões que foram identificados.

Algumas conclusões podem ser apresentadas a respeito dos anti-padrões demonstrados. Alguns dos anti-padrões são especializações dos problemas expostos por Smith e Williams, como por exemplo, o anti-padrão *One-By-One*, que apresenta-

**Tabela 2: Anti-padrões Identificados em Associações Um-Para-Muitos**

| Anti-padrão | Descrição / Problema |
|---|---|
| *Inadequate Collection Type On Owning Side* | Diferentes características de desempenho são apresentadas a depender da estrutura de dados utilizada. O anti-padrão descreve como o Hibernate se comporta ao trabalhar, com os seguintes tipos de coleção utilizados na Linguagem Java: java.util.Collection, java.util.List, java.util.Set. Para solucionar problemas de desempenho relativos a este anti-padrão, deve-se analisar o perfil de uso das coleções no *software* e ajustar o tipo de coleções Java de acordo. |
| *OneToMany As Owning Side* | Realizar o mapeamento na direção de um-para-muitos, implica na obtenção de uma coleção de objetos. Como consequência, consultas adicionais são realizadas e processamento adicional é executado pelo Hibernate, para a conversão de registros consultados em objetos. Ao realizar o mapeamento de um-para-muitos, não deve-se mapear na direção de um-para-muitos e sim na direção de muitos-para-um. |
| *Inadequate Collection Type On Inverse Side* | Este anti-padrão ocorre em associações bidirecionais, onde recomenda-se sincronizar o estado dos objetos na memória. A convenção é que a coleção mapeada seja atualizada através do método *setter*, ao ocorrer uma associação na direção muitos-para-um. O problema está ao utilizar o java.util.Set como tipo da coleção mapeada. Este tipo de coleção, trabalha com elementos únicos, o que significa que todas os itens precisam ser carregadas na memória principal em resposta a cada adição de novos elementos, mesmo que não ocorram acessos a coleção e seus elementos. Para solucionar este problema, é recomendado utilizar no lugar do java.util.Set, os tipos java.util.Collection ou java.util.List. |
| *Lost Collection Proxy On Owning Side* | Em uma associação um-para-muitos, o campo que representa a coleção de objetos persistentes, é um *proxy* do Hibernate. Caso esta coleção de objetos seja sobrescrita com uma nova coleção, o *proxy* é perdido, e consequentemente o Hibernate não é capaz de rastrear o que foi alterado. Neste caso, a política do *framework* é recriar toda a coleção, mesmo que os elementos não tenham sido alterados, removendo todas as associações e realizando a inserção dos itens presentes na coleção. A solução recomendada é utilizar o *proxy* retornado pelo Hibernate, visto que na maioria dos casos, é uma abordagem eficiente. |
| *One-By-One* | O anti-padrão *One-By-One*, refere-se a iteração sobre uma coleção, e para cada elemento, a ocorrência de uma operação no banco de dados. Uma das consequências deste anti-padrão é a execução de um excessivo número de operações no banco de dados. Para solucionar este problema, é recomendado que se utilize os recursos oferecidos pelo banco de dados relacional, tais como as operações em lote e as funções de agregação. |

Fonte: Elaborada pelo autor baseando-se em Wegrzynowicz (2013).

se como uma releitura do anti-padrão *Empty Semi-Trucks* aplicado ao contexto do mapeamento-objeto-relacional. Observando as características de anti-padrões como: *Lost Collection Proxy*, pode-se observar que as convenções de mapeamento objeto-relacional, assim como as políticas ou limitações do *framework* ORM, podem gerar problemas de desempenho.

## 2.3 MÉTRICAS

Para que seja realizada a avaliação de desempenho nos *softwares* do SPB, se faz necessário a definição de métricas para mensurar o impacto do problema de desempenho. Considerando que: “Performance é o grau em que um componente ou sistema realiza suas funções designadas dentro de determinadas restrições, tais como velocidade, precisão ou uso de memória, impostas” (RADATZ; GERACI; KATKI, 1990, p. 55); para medir este grau, indicadores-chave de desempenho (KPI) podem ser utilizados.

Os indicadores-chave de desempenho (KPI) são divididos em dois tipos: orientados a serviço e orientados a eficiência. Os indicadores orientados a serviço medem o quão bem um *software* provê os seus serviços aos usuários finais e os orientados a eficiência medem o quão bem o *software* faz uso de seus recursos (MOLYNEAUX, 2009). Na Tabela 3, é apresentada uma breve descrição sobre os indicadores de desempenho agrupados por tipo.

## 2.4 IDENTIFICAÇÃO DE ANTI-PADRÕES

A implementação de um processo para a identificação de anti-padrões de desempenho, requer a observação do estado da arte no que se refere as abordagens utilizadas para a identificação de anti-padrões, afim de implementar um processo utilizando as melhores práticas alinhadas com o objetivo do trabalho proposto. Na literatura podem ser encontrados trabalhos que abordam a identificação de anti-padrões utilizando diferentes processos, em abordagens manuais ou automatizadas.

As abordagens manuais demandam tempo e esforço, especialmente em *softwares* de grande porte. A detecção automatizada foi desenvolvida para resolver os problemas relacionados a tempo e esforço que ocorrem na detecção manual. Entretanto, abordagens automatizadas sofrem com os problemas de incerteza, como por

**Tabela 3: Indicadores-chave de desempenho**

| Tipo | KPI | Descrição |
|---|---|---|
| **Serviço** | Disponibilidade | Refere-se a quantidade de tempo em que um *software* está disponível para o usuário final. |
| | Tempo de Resposta | Refere-se ao tempo necessário para que o *software* responda a uma solicitação do usuário. |
| **Eficiência** | Vazão | A taxa na qual os eventos em um *software* ocorrem, tal como o número de acessos em uma página *web*, em um determinado período de tempo. |
| | Utilização | Refere-se a porcentagem da capacidade de um recurso que está sendo utilizado. |

Fonte: O autor (2018).

exemplo, a ordem em que os artefatos devem ser inspecionados (DIN; AL-BADAREEN; JUSOH, 2012).

Para contornar o problema de incerteza das abordagens automatizadas, técnicas baseadas em modelos probabilistícos, como as redes bayesianas podem ser utilizadas. Estas técnicas de identificação semi-automatizadas fornecem a probabilidade da ocorrência de um anti-padrão. Entretanto, os modelos criados utilizando as redes bayesianas são dependentes do contexto em que são aplicados, não podendo ser generalizados (KHOMH et al., 2011).

É possível observar na literatura a existência de estudos que utilizam um conjunto de regras para a detecção de anti-padrões. Uma combinação de métricas é requerida para definir as regras para detecção. Tais regras são definidas manualmente e são caracterizadas pela combinação de indicadores quantitativos, estruturais e/ou léxicos. Entretanto, é exigido um esforço para atribuir a cada métrica, um valor limiar correto, de forma que tais regras identifiquem um anti-padrão (KESSENTINI et al., 2011).

Métodos baseados em regras, acompanhados de técnicas de Mineração de Dados, são utilizados para a identificação de anti-padrões em Arquiteturas Orientadas a Serviços (SOA). Nesta abordagem, um conjunto de métricas específicas de domínio é definido, baseando-se em hipóteses sobre a relação de acoplamento e coesão entre

serviços. Utilizando as métricas, e combinando-as, é possível especificar anti-padrões na forma de um conjunto de regras. Utilizando as regras definidas, algoritmos de detecção de anti-padrões são gerados. Por fim, os traços de execução dos serviços da arquitetura são minerados, para a descoberta de associações entre estes, e são aplicados os algoritmos de detecção para realizar filtragem do conhecimento resultante da mineração (NAYROLLES; MOHA; VALTCHEV, 2013).

Os anti-padrões de desempenho em *softwares* que utilizam mapeamento objeto-relacional, são objetos de estudo para a identificação de anti-padrões. Chen et al. (2014), implementam um *framework* automatizado para a detecção de anti-padrões ORM, utilizando a análise estática de código-fonte. Através da análise estática, são extraídos os dados acerca dos trechos do código que realizam acesso ao banco de dados. Após a extração dos dados, os mesmos são utilizados para a criação de regras para cada anti-padrão que será identificado. Para o anti-padrão *One-by-one*, por exemplo, o *framework* identifica as funções que são invocadas direta ou indiretamente dentro de estruturas de repetição. Devido a grande quantidade de instâncias de um anti-padrão identificadas em um *software*, o *framework* ao identificar uma instância, atribui uma prioridade de correção a mesma, com base nos ganhos de desempenho esperados. Quanto maior o ganho em desempenho, maior a prioridade de correção do anti-padrão.

Na revisão da literatura são apresentados processos na Engenharia Orientada a Modelos (MDE), para a identificação de padrões e anti-padrões. A identificação pode ser obtida realizando a comparação de modelos, que consiste em contrastar dois ou mais modelos para identificar semelhanças e diferenças. Uma das abordagens de comparação é a detecção de clones de modelo (MCD). O MCD envolve a identificação de pares de clones dentro de um projeto MDE, os quais são semelhantes a um limite específico, e pode ser utilizado para a detecção de padrões e anti-padrões expressos como um modelo a serem comparados com outros (STEPHAN; CORDY, 2015).

## 2.5 PROCESSOS

Para a compreensão do processo implementado neste trabalho, é necessário introduzir o conceito de processo, sua distinção quando aplicado no contexto de *software* e de negócio, assim como da disciplina de gerência de processos de negócio

e suas representações através de notações gráficas.

Um processo pode ser definido como um conjunto de ações e atividades inter-relacionadas que são executadas para criar um produto, serviço ou resultado pré-especificado; e as características de um processo envolvem as suas entradas, ferramentas, técnicas e saídas resultantes (PROJECT MANAGEMENT INSTITUTE, 2013). O conceito de processos, pode ser aplicado também no contexto de negócios e de desenvolvimento de *software*.

O conjunto de atividades de um processo, no contexto de desenvolvimento de *software*, é voltado para produção de um produto de *software*. Neste contexto, um processo deve ser composto por atividades fundamentais. Dentre estas atividades, encontram-se as que especificam as funcionalidades e as restrições ao funcionamento do *software*; as que produzem o mesmo segundo as suas especificações; as atividades que o validam com o objetivo de garantir a conformidade com a sua especificação e atividades que visam atender as necessidades de mudanças no *software* (SOMMERVILLE, 2011).

### 2.5.1 Modelagem de Processos de Negócio

Pode-se definir negócio como a interação de um grupo de indivíduos para realizar um conjunto de atividades e entregar valor aos clientes (CAPOTE, 2012). O trabalho para a entrega de valor para os clientes é definido como Processo de Negócio, e o gerenciamento destes processos engloba a disciplina gerencial de Gerenciamento de Processos de Negócio (BPM), que pretende alcançar os objetivos organizacionais através da definição, desenho, controle e transformação contínua de processos de negócio (BENEDICT et al., 2013).

Com o propósito de criar uma representação completa e precisa do processo, a modelagem envolve um conjunto de atividades na criação de representações de processos de negócio. Um processo de negócio pode ser representado por um modelo, representando diferentes perspectivas, servindo a diferentes propósitos, através da notação de modelagem do processo (BENEDICT et al., 2013).

A Notação de Modelagem de Processos de Negócio (BPMN), é um exemplo de notação para modelagem de um processo. BPMN utiliza um conjunto de símbolos, através de uma notação gráfica, que permitem diagramar modelos de processos,

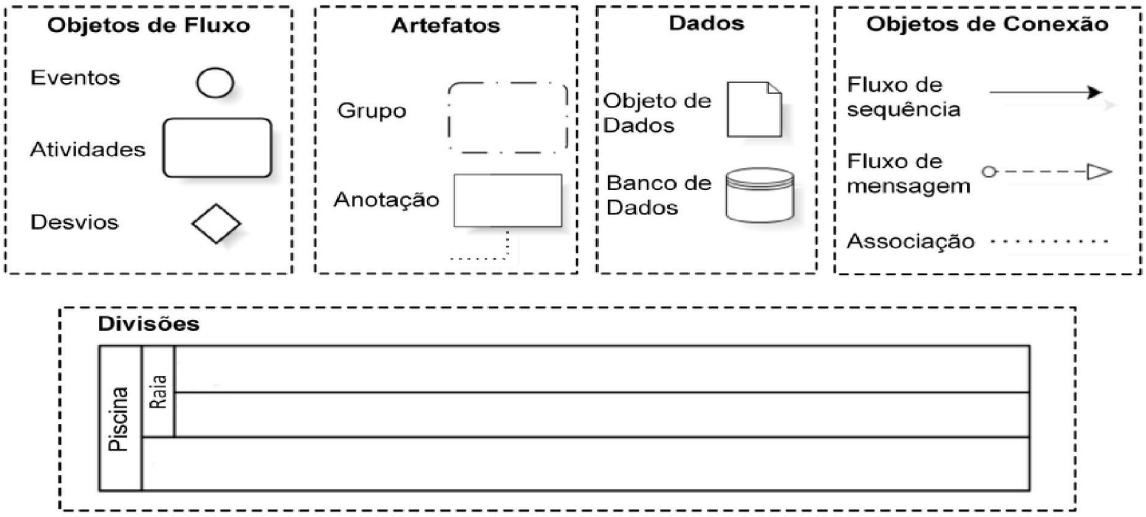
transmitindo a lógica das atividades e toda a informação necessária para que um processo seja analisado, simulado e executado (MINISTÉRIO PÚBLICO FEDERAL, 2013).

### 2.5.2 Conceitos Básicos de BPMN

O BPMN inclui em sua notação gráfica três elementos centrais, nomeados de Objetos de Fluxo. Estes objetos são conectados por Objetos de Conexão com o objetivo de representar a estrutura básica de um processo de negócio. A modelagem utilizando o BPMN não é limitada apenas ao uso dos objetos de conexão. É possível realizar a extensão da notação básica através do conceito de artefatos, que são objetos com a capacidade de adicionar contexto apropriado a uma situação de modelagem específica. Além da adição de objetos e conectores entre estes, as atividades do processo podem ser organizadas em categorias visuais separadas,através do conceito de piscinas (*swimlanes*), afim de ilustrar diferentes capacidades ou responsabilidades funcionais (WHITE, 2015).

A Figura 2 e a Tabela 4 abaixo apresentam a descrição e a representação, respectivamente, de alguns dos elementos citados que compõem o BPMN.

**Figura 2: Representação do conjunto básico de elementos do BPMN**




Fonte: Santos (2016, p. 22).

**Tabela 4: Descrição do conjunto básico de elementos do BPMN**

| **Elemento** | **Descrição** |
|---|---|
| Objetos de Fluxo | |
| Evento | Um evento é algo que ”acontece”durante o fluxo de um processo. Esses eventos afetam o fluxo do processo e geralmente têm uma causa ou resultado. |
| Atividade | Uma atividade é um termo genérico para um trabalho que é realizado. |
| Desvio (*Gateway*) | Um desvio é usado para controlar a divergência e convergência do Fluxo de Sequência. |
| Artefatos | |
| Grupo | Pode ser utilizado para fins de documentação e análise, mas não afeta o Fluxo de Sequência. |
| Anotação | Anotações são um mecanismo para fornecer informações de texto adicionais sobre o processo modelado. |
| Dados | |
| Objetos de dados | Os objetos de dados são um mecanismo para mostrar como os dados são necessários ou produzidos pelas atividades. |
| Banco de dados | Representam um conjunto de dados estruturado. |
| Objetos de Conexão | |
| Fluxo de Sequência | É utilizado para mostrar a ordem (a sequência) das atividades que serão realizadas em um processo. |
| Fluxo de Mensagem | É usado para mostrar o fluxo de mensagens entre dois participantes do processo que recebem ou enviam mensagens. |
| Associação | Uma associação é utilizada para associar dados, texto e outros Artefatos com Objetos de Fluxo. As associações são utilizadas para mostrar as entradas e saídas das atividades. |
| Piscinas (*Swimlanes*) | |
| Piscina | Contém um processo de negócio. Processos de negócio distintos devem estar contidos, cada um, em uma piscina. |
| Raia | Caracteriza os participantes envolvidos na realização das atividades do processo. |

Fonte: Adaptado de White (2015)

# 3 METODOLOGIA

## 3.1 CONTEXTO DA IDENTIFICAÇÃO DE PROBLEMAS

Na literatura, os problemas de desempenho são originados por diferentes fatores, identificados em diferentes componentes do *software* e mensurados utilizando diferentes métricas. Observada esta condição, se fez necessário a definição de um contexto para o levantamento dos problemas de desempenho. Este contexto, inclui a definição de uma métrica para mensurar os problemas de desempenho, assim como um eixo temático que caracterize o tipo de problema de desempenho que será abordado.

Para mensurar o impacto dos anti-padrões de desempenho nas funcionalidades do *software* será utilizado o tempo de resposta. O tempo de resposta ou tempo de transação, foi definido como métrica devido a sua importância ao caracterizar o desempenho de um *software* da perspectiva de um usuário (GRINSHPAN, 2012). A importância de utilizar uma métrica focada no usuário é devido ao objetivo do SPB, de fornecer um serviço de alta qualidade aos usuários de suas soluções em *software*.

Após a definição da métrica, se faz necessário definir um eixo temático, que aborde problemas mensuráveis pela métrica do tempo de resposta. Como temática para os problemas a serem abordados, foram definidas as atividades de entrada/saídas realizadas na persistência dos dados em um banco de dados. No que se refere ao tempo de resposta, é de fato que estas atividades são custosas para o desempenho de um *software*, além da importância que os bancos de dados exercem na sociedade moderna, sendo estes um componente essencial (ELMASRI; NAVATHE, 2011).

Como observado na literatura, e apresentado no referencial teórico, existem anti-padrões identificados em *softwares* com diferentes paradigmas. Alguns destes anti-padrões são uma especialização de outros, como pode ser observado no referencial teórico a relação entre os anti-padrões apresentados por Smith e Williams (2003) e Wegrzynowicz (2013). Os anti-padrões abordados no processo para identificação de anti-padrões não irão restringir a sua aplicação a determinados *softwares*, visto que o SPB é um repositório que possui *softwares* com diferentes características.

Os anti-padrões catalogados por Smith e Williams abordam uma variedade de indicadores de qualidade não-funcional e serão utilizados como objeto de estudo neste

trabalho. Os anti-padrões que serão abordados no processo proposto localizam-se dentro do contexto definido para a identificação de problemas de desempenho e são relevantes para a métrica proposta. Porém, apesar do processo de identificação de anti-padrões abordar esta metodologia, não é desconsiderada a aplicação prática e a relevância dos anti-padrões não apresentados.

A definição de um contexto para o levantamento dos problemas de desempenho, influencia na escolha de quais anti-padrões serão abordados no processo para identificação destes. Deve ser possível mensurar os anti-padrões utilizando a métrica escolhida e estes devem apresentar problemas na temática definida. Assim, o processo definido neste trabalho pode ser replicado e estendido utilizando diferentes anti-padrões, que dependem diretamente da métrica e temática definida.

## 3.2 CENÁRIO DE ESTUDO

O cenário de aplicação será um estudo de caso em um *software* específico do SPB. O estudo de caso será realizado com o objetivo de adquirir conhecimento do fenômeno estudado a partir da exploração detalhada de um único caso (VENTURA, 2007). Os critérios utilizados na seleção do *software* do estudo de caso, consistem na identificação do tipo do *software*, sendo este um *software* corporativo; e na quantidade de usuários. O *software* selecionado para o estudo de caso foi o Sistema de Gestão Comercial de Gás Natural (GGAS) onde é possível acessar o seu repositório por meio do Portal de Software Público Brasileiro.

O GGAS é um *software* escrito na linguagem Java, executado na plataforma WEB, sendo os seus usuários finais as distribuidoras de gás do Brasil e seus respectivos clientes. De acordo com informações fornecidas pela Mitsui Gás e Energia do Brasil, que possui participação acionária nas distribuidoras, o GGAS ainda não está em produção em todas as distribuidoras, mas quando for concluída a implantação estima-se que poderá ser acessado por aproximadamente 150.000 usuários (número aproximado de clientes de todas as distribuidoras que estão realizando a implantação). Com um custo de aproximadamente R$ 6.500.000,00 para a implantação nas companhias distribuidoras de gás natural dos estados de Alagoas, Sergipe, Bahia, Paraná, Paraíba e Rio Grande do Norte, .

## 3.3 PROCESSO DO LEVANTAMENTO DE PROBLEMAS

Os esforços iniciais para implementação do processo se concentram em definir as principais atividades que serão realizadas. Como observado no referencial teórico, existem diferentes estratégias para identificação de anti-padrões, adequadas a diferentes paradigmas de desenvolvimento de *software*. Diversas estratégias utilizam o conceito de extração de determinadas características ou métricas do *software* e algumas realizam este procedimento de forma manual e outras automatizam utilizando ferramentas. Assim, no processo proposto será realizado uma inspeção manual e uma análise dinâmica, utilizando ferramentas para extrair características que correspondam as de um anti-padrão de desempenho.

A análise dinâmica ou *profiling* é a análise do *software* em tempo de execução que mede, por exemplo: a memória, o uso de instruções em particular (instruções em SQL) e a duração e frequência de chamadas de uma função (COPPA; DEMETRESCU; FINOCCHI, 2014). A inspeção manual consiste da exploração do *software*, de seu código-fonte e consequentemente da sua arquitetura. A inspeção fornece uma compreensão das funcionalidades do *software*.

Para o processo proposto, a análise dinâmica será realizada com o objetivo de obter as consultas SQL executadas nas funcionalidades do *software*, assim como o tempo de resposta dos métodos que executam as mesmas. Busca-se nestas consultas SQL a existência de características de um anti-padrão, e estas consultas são alvo de análise devido aos problemas de desempenho estarem contextualizados na temática de atividades de entrada/saída no banco de dados.

Assim como na análise dinâmica, a inspeção é realizada no código-fonte das funcionalidades do *software* com o objetivo de identificar as características de um anti-padrão. A identificação do anti-padrão é obtida através da verificação de uma correspondência entre as características descritas no problema e o conjunto de características identificadas na inspeção e na análise dinâmica.

Além das atividades de inspeção e análise dinâmica, deve-se realizar atividades para mensurar o impacto do anti-padrão no desempenho da funcionalidade, assim como da redução obtida do tempo de resposta, caso o problema de desempenho seja solucionado. De acordo com o objetivo do trabalho, estas atividades estão presentes no processo para apresentar a relevância destes problemas de desempenho para os

usuários finais dos *softwares* do SPB.

Por fim, o processo deve possuir atividades dedicadas a documentação das atividades de inspeção, análise dinâmica e medição, para auxiliar na troca de dados entre estas atividades. Após a definição das principais atividades que serão realizadas, é implementada a modelagem do processo para que fiquem explicitas as entradas, saídas, artefatos e fluxos principais e alternativos. Para realizar a modelagem, será utilizada a notação BPMN, visto que este trabalho busca um processo para a identificação de anti-padrões voltado para o contexto de negócio do SPB. A Figura 3 apresenta a modelagem do processo.

Inicialmente, é importante deixar explicito alguns termos utilizados no processo. Na modelagem apresentada, existe uma distinção entre o conceito de funcionalidade e método. O termo funcionalidade, empregado neste processo, refere-se a um requisito do *software*, a uma função que o mesmo executa. Os métodos representam um nível de organização do código-fonte do *software*, isolando determinado comportamento executado em um funcionalidade.

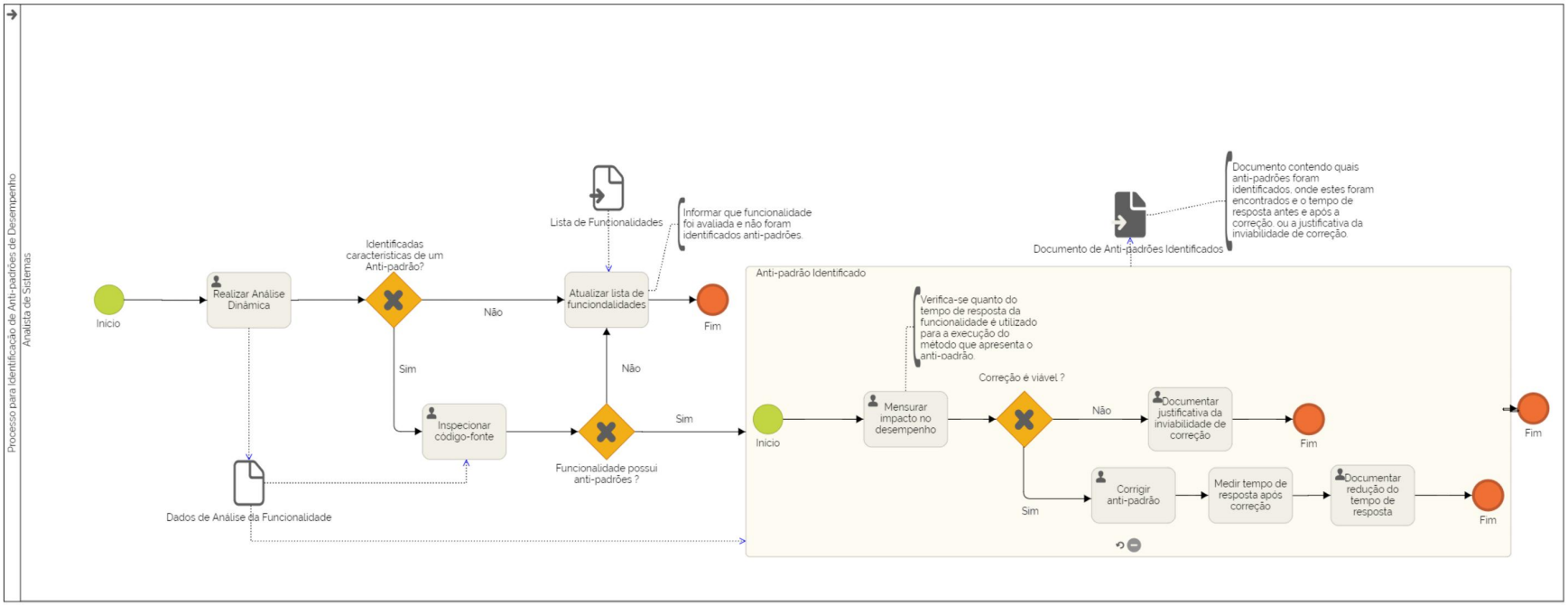
Como pode-se observar na modelagem, o processo se inicia com a análise dinâmica. Uma funcionalidade do *software* é escolhida para execução, utilizando uma ferramenta de análise dinâmica. Caso seja utilizado o XRebel (apresentado na seção 3.4) para realizar a análise dinâmica, ao executar a funcionalidade é possível visualizar a pilha de métodos executados, assim como as consultas ao banco de dados realizadas em cada método.

No caso de um anti-padrão como o *Empty Semi Trucks*, na analise dinâmica, busca-se um método com consultas SQL que se repetem e possuem estrutura lógica semelhante. Para o anti-padrão *Unnecessary processing* deve-se observar a assinatura do método e a sua função, buscando métodos que realizam consultas SQL que não condizem com o proposito do mesmo.

Para alguns anti-padrões, a análise dinâmica é insuficiente para determinar a existência do mesmo. Para o anti-padrão *The Ramp*, apontar que o processamento cresce com o tempo depende da execução da funcionalidade em diferentes cenários e da necessidade de observar o algoritmo envolvido na ocorrência do anti-padrão.

Já na identificação do *Circuitous Treasure Hunt*, pode-se observar quais tabelas são acessadas nas consultas SQL. Entretanto, não é possível afirmar quais tabelas

**Figura 3: Processo para Identificação de Anti-padrões de Desempenho**




Fonte: O autor (2018).

são acessadas desnecessariamente sem observar um modelo do domínio do *software*. Para estes anti-padrões, é necessário realizar uma inspeção manual no *software*.

A inspeção manual é realizada em métodos da funcionalidade que, de acordo com a análise dinâmica apresentam as características de um anti-padrão. A inspeção pode ser utilizada para complementar a análise dinâmica, identificando características de um anti-padrão que não puderam ser identificadas na primeira etapa do processo. Também pode-se utilizar a inspeção para comprovar as evidências apontadas durante analise dinâmica, de forma a observar no código-fonte as características identificadas dinamicamente.

No modelo do processo pode-se observar a presença de um documento com os dados resultantes da análise dinâmica. Este documento apresenta para cada método, o seu tempo de resposta e as características de um anti-padrão que puderam ser identificadas. No documento também deve constar o tempo de resposta da funcionalidade executada.

Caso seja identificado ao menos um anti-padrão, inicia-se um subprocesso, que é executado para cada anti-padrão identificado. Neste subprocesso calcula-se a razão entre o tempo de resposta da funcionalidade e o tempo de resposta do método que apresenta a ocorrência do anti-padrão, afim de verificar percentualmente, quanto do tempo de resposta da funcionalidade é utilizado para a execução do método.

Considerando que o eixo temático foi definido como as atividades entrada/saída realizadas no banco de dados, o tempo de resposta do método é resultante da soma do tempo de resposta das consultas SQL realizadas. E consequentemente, o tempo de resposta da funcionalidade é resultante do tempo de resposta das consultas SQL que ocorrem durante a execução da funcionalidade.

Para reduzir a influência de variáveis externas que não podem ser controladas, será considerado um tempo de resposta médio, resultante de múltiplas execuções da análise dinâmica sobre a funcionalidade. Assim, pode-se identificar se o anti-padrão tem um impacto relevante no desempenho da funcionalidade.

A medição do impacto no desempenho é importante para a verificação de viabilidade de correção do anti-padrão. Um anti-padrão de maior impacto no desempenho e de complexidade de correção menor, tem maior prioridade para a correção do que um anti-padrão de menor impacto e de complexidade maior para correção.

Caso a correção do anti-padrão seja realizada, realiza-se a medição do tempo de resposta do método após a correção. Neste trabalho iremos obter o tempo de resposta fornecido pela ferramenta de análise dinâmica. Entretanto, o processo será flexível e não ficará explícito no mesmo como esta medição será realizada, podendo ser manual ou com auxilio de ferramentas. Para obter a redução do tempo de resposta, calcula-se a diferença entre o tempo de resposta antes e após a correção. Por fim, documenta-se a redução do tempo de resposta obtida na correção do anti-padrão ou uma justificativa para a correção não ter sido realizada. Não corrigir um anti-padrão pode ser justificado pelo mesmo não impactar significativamente o desempenho da funcionalidade ou possuir um grau de complexidade alto para a correção.

A partir do modelo definido podemos verificar a necessidade de fornecer como entrada para o processo, uma lista das funcionalidades que serão analisadas, atualizando-a ao informar em quais funcionalidades foram ou não identificados anti-padrões. Através de artefatos/documentos produzidos, a modelagem permite observar a troca de dados entre as atividades do subprocesso e as atividades de análise dinâmica e inspeção.

Como saída do subprocesso e consequentemente do processo primário, deve ser produzido um documento contendo quais anti-padrões foram identificados, os métodos em que estes foram encontrados e o tempo de resposta antes e após as correções realizadas. Caso não seja viável a correção do anti-padrão de desempenho, deve constar no documento produzido a justificativa da inviabilidade de correção.

## 3.4 FERRAMENTAS E AMBIENTE

Para realizar o estudo de caso, o ambiente a ser configurado é específico do *software* a ser estudado. Portanto, as atividades para configuração do ambiente não estão documentadas no processo, visto que busca-se um processo independente do *software*. O ambiente foi configurado de acordo com a documentação de configuração do ambiente do *software*.[1] De acordo com a documentação, utiliza-se para a construção do *software* a ferramenta Gradle e para implantação, um servidor de aplicação Tomcat.

Para a inspeção manual, pode ser utilizado qualquer editor de código-fonte ou IDE. Porém, foi dada preferência ao Eclipse IDE, dada a sua popularidade no

[1] A documentação encontra-se disponível em: http://docs.ggas.com.br/docs/ggas_documentacao/

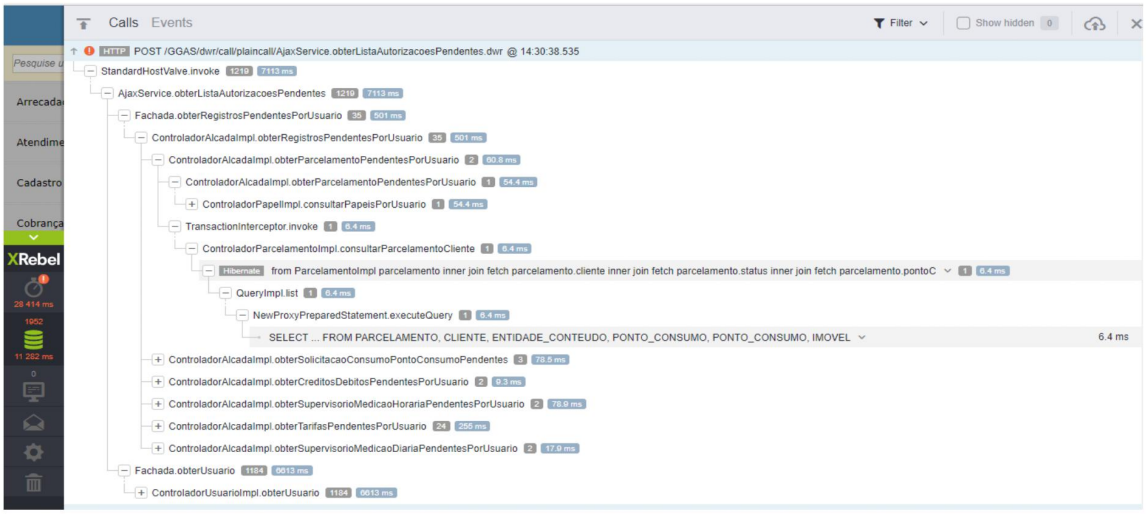
desenvolvimento na linguagem Java. A IDE auxilia nas atividades a serem realizadas na inspeção manual, visto que existe a possibilidade de rastreabilidade dos métodos. Esta rastreabilidade facilita na etapa do processo onde é necessário identificar no código-fonte, os métodos que foram identificados na análise dinâmica com as características de um anti-padrão.

Para realizar a análise dinâmica pode-se utilizar qualquer ferramenta, desde que a mesma permita realizar as atividades descritas no processo. Uma solução *open source* como Apache JMeter pode ser utilizada para medir o tempo de resposta de uma requisição realizada a um *software* Java Web. O Apache JMeter realiza diferentes testes de desempenho, incluindo testes de stress e de carga (MATAM; JAIN, 2017). Outra alternativa de ferramenta para realizar a analise dinâmica é a ferramenta XRebel. O XRebel é um agente Java que permite analisar as atividades de banco de dados do *software*.

O XRebel foi a ferramenta utilizada neste trabalho, pois permite executar as funcionalidades do *software* em uma plataforma WEB, através de um navegador WEB, e obter os resultados da análise dinâmica na interface do mesmo (PRAKASH, 2015). Esta característica é útil para o estudo de caso proposto, visto que através dos resultados da análise dinâmica apresentados em tempo real, pode-se explorar um *software* do SPB realizando as atividades comuns de seu usuário final. A exploração do *software* permite compreender a arquitetura e comportamento deste.

A Figura 4 apresenta o uso do XRebel, após a autenticação de um usuário no *software* proposto para o estudo de caso. É possível observar que o XRebel apresenta todos os métodos que foram executados pelo *software* durante o processo de autenticação. Caso determinado método realize consultas no banco de dados, estas são apresentadas. Para cada consulta, é possível visualizar o tempo de resposta, assim como a quantidade de registros consultados.

**Figura 4: Resultado do XRebel após autenticação de um usuário.**




Fonte: O autor (2018).

# 4 EXPERIMENTOS E RESULTADOS OBTIDOS

## 4.1 ESTUDO DE CASO

Antes de executar o processo de identificação de anti-padrões em um estudo de caso no *software* do SPB, é necessário definir quais módulos deste *software* serão avaliados. Para realizar a seleção dos módulos que serão avaliados, foram executadas as funcionalidades do *software*, realizando atividades de um usuário final. Após a execução das funcionalidades, foram selecionados para a avaliação os módulos de acesso e o de contrato.

O módulo de acesso foi escolhido por conter as funcionalidades de autenticação e autorização, obrigatórias para o uso do *software*. Já o de contrato, foi selecionado por ser considerado um dos maiores módulos, uma vez que o mesmo se relaciona com praticamente todos os outros, e no contexto de negócio do GGAS as informações contratuais são necessárias para o uso das principais funcionalidades do *software*.

É importante ressaltar que Santana, Alencar e Correia (2017) disponibilizaram para a comunidade científica as medições do tempo de resposta de cada anti-padrão identificado neste estudo de caso através da publicação em um periódico.

### 4.1.1 Módulo de Acesso

Ao executar a autenticação no *software*, foi possível observar na análise dinâmica, a ocorrência de um anti-padrão. Um método que, de acordo com a sua assinatura, era responsável por validar o *login* do usuário, realizava uma quantidade excessiva de consultas SQL. Dado este comportamento, suspeita-se da ocorrência do anti-padrão *Unnecessary Processing*. Este anti-padrão se caracteriza pela existência de processamento que é executado, mas que não é necessário (SMITH; WILLIAMS, 2002).

Foi possível observar durante a inspeção manual no método, que para validar a autenticação do usuário é necessário consultar no banco de dados um registro com as informações do mesmo. Este registro possui associações com outros, de outras tabelas. Estes registros associados também são obtidos, porém, não são utilizados para validação da autenticação. Um exemplo é a consulta dos endereços residenciais do usuário, que ocorre, mas é desnecessária, pois não é utilizada para validar a

autenticação.

Para solucionar o problema, a solução que é recomendada na definição do anti-padrão *Unnecessary Processing* é que seja eliminado o processamento desnecessário. Como o GGAS utiliza mapeamento objeto-relacional, pode-se alterar os metadados do mapeamento da classe que representa o usuário. Pode-se definir quais classes, associadas a classe que representa o usuário e mapeadas em outras tabelas do banco de dados também serão obtidas, caso os dados do usuário sejam consultados no banco.

Entretanto, alterar o mapeamento pode resultar em alterações em outras funcionalidades do *software*, que consultam os dados do usuário. Portanto, foi implementado um novo método para consultar apenas os dados do usuário necessários para validar a autenticação do mesmo. Ao alterar a funcionalidade, fazendo-a utilizar o método implementando, foi percebido a redução do tempo de resposta demonstrada na Tabela 5.

**Tabela 5: Identificação do anti-padrão *Unnecessary Processing* na funcionalidade de autenticação do *software***

| Tempo de resposta do método com o anti-padrão | Tempo de resposta do método após correção do anti-padrão | Redução obtida no tempo de resposta do método |
|---|---|---|
| 14.9 ms | 1.4 ms | 90.60% |

Fonte: O autor (2018).

Além do excesso de consultas SQL identificados em métodos executados durante a autenticação, na análise dinâmica verificou-se a existência de consultas de estrutura lógica idêntica, mas executadas mais de uma vez, variando apenas parâmetros na clausula condicional da consulta. Ao realizar a inspeção manual foi possível compreender que se tratavam de métodos que realizavam consultas de dados necessários para a autorização do usuário no *software*.

A funcionalidade de autorização do GGAS é baseada nos conceitos de: quais módulos do *software* o usuário tem permissão de acesso, onde estes agrupam recursos do *software*, e as operações sobre esses recursos que o usuário pode efetuar. Durante a autenticação do usuário, os módulos, os recursos e as operações são consultados no banco de dados. Inspecionando o código-fonte observa-se que, o anti-padrão é causado devido a um *HQL* mal escrito, que obtém uma operação e inicializa a

coleção de recursos da mesma, resultando em uma consulta para cada recurso de uma operação.

As características identificadas na análise dinâmica e na inspeção correspondem com as características do anti-padrão *Empty Semi Trucks*. Esse anti-padrão ocorre quando um número excessivo de requisições são necessárias para executar uma tarefa, que poderia ser executada em apenas uma requisição (SMITH; WILLIAMS, 2003). Esta requisição pode ser interpretada como uma consulta de dados do banco de dados.

Para a correção do anti-padrão *Empty Semi Trucks*, a solução recomendada pela literatura é o agrupamento das consultas em lote, agrupando várias consultas em apenas uma (SMITH; WILLIAMS, 2003). O *framework* de mapeamento objeto-relacional utilizado pelo GGAS permite definir este agrupamento nos metadados do mapeamento da classe, permitindo informar quantas consultas devem ser agrupadas em uma única consulta.

Não há risco dessa alteração no mapeamento da classe impactar negativamente em outras funcionalidades do *software* ou alterar o comportamento destas, visto que trata-se de um agrupamento nas consultas, as mesmas irão obter os mesmos dados, estes não serão removidos e nem adicionados. A tabela 6 apresenta os métodos impactados pelo anti-padrão, assim como o tempo de resposta antes e após a correção do mesmo.

**Tabela 6: Identificação do anti-padrão *Empty Semi Trucks***

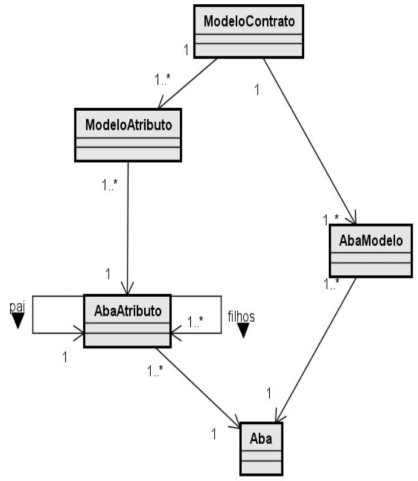
| | **Tempo de resposta do método com o anti-padrão** | **Tempo de resposta do método após correção do anti-padrão** | **Redução obtida no tempo de resposta do método** |
|---|---|---|---|
| **Listagem de Módulos** | 872.8 ms | 496.1 ms | 43.15% |
| **Listagem de Recursos** | 324.9 ms | 44.9 ms | 86.18% |

Fonte: O autor (2018).

### 4.1.2 Módulo de Contrato

Um contrato no GGAS possui um modelo que define a estrutura do contrato, as seções que compõem o mesmo (Abas) e os atributos destas. A Figura 5 apresenta uma parte do digrama de classes, resultante da inspeção realizada no código-fonte do módulo de contrato.

**Figura 5: Diagrama de Classes do Modelo de Contrato**




Fonte: O autor (2018).

Na inclusão de novos contratos, ocorre uma pesquisa dos modelos de contrato disponíveis. Ao analisar dinamicamente a inclusão, foi possível observar que o método responsável por obter os modelos de contrato realizava apenas uma consulta SQL na tabela referente ao modelo de contrato e diversas outras consultas em diferentes tabelas.

Baseando-se apenas na análise dinâmica, poderia ser afirmada a existência do anti-padrão *Unnecessary Processing*. Entretanto, inspecionando o método responsável pela pesquisa de modelos de contrato, foi possível compreender o propósito das diferentes tabelas envolvidas nas consultas SQL e consequentemente das classes de domínio do *software* referentes ao módulo de contrato.

De acordo com o que foi observado na inspeção, para consultar um objeto da classe Aba, deve-se obter uma coleção de objetos da classe denominada AbaModelo e partir destes obtemos os objetos da classe Aba. Este procedimento se caracteriza por uma das instâncias do anti-padrão *Circuitous Treasure Hunt*. Este anti-padrão ocorre

quando é necessário consultar em vários lugares para se obter um “resultado final” (SMITH; WILLIAMS, 2000).

A classe AbaModelo mapeia uma tabela de junção entre a classe ModeloContrato e a classe Aba. Este mapeamento é desnecessário, visto que poderia ser estabelecido um relacionamento muitos para muitos entre ModeloContrato e Aba sem a necessidade da classe AbaModelo. Neste cenário atual, é adicionado um custo de processamento das consultas e da criação de instância de objetos da classe AbaModelo.

Para a correção do anti-padrão, a literatura propõe a criação de uma associação direta com o “resultado final”. Entretanto, alterações nas classes do *software* podem necessitar de alterações no código-fonte das funcionalidades do mesmo. Assim, para minimizar o impacto da correção, foi implementado um novo método para a pesquisa de modelos de contrato. O método realiza uma consulta utilizando o HQL, considerando que, com a relação direta entre um modelo de contrato e suas abas, as mesmas são obtidas em uma única consulta ao banco. A Tabela 7 apresenta os resultados obtidos ao realizar a correção do anti-padrão.

**Tabela 7: Identificação do anti-padrão *Circuitous Treasure Hunt***

| Tempo de resposta do método com o anti-padrão | Tempo de resposta do método após correção do anti-padrão | Redução obtida no tempo de resposta do método |
|---|---|---|
| 55.2 ms | 2.5 ms | 95.47% |

Fonte: O autor (2018).

Para incluir um novo contrato, é necessário consultar um cliente para associá-lo ao contrato. Para as regras de negócio do GGAS, este cliente é um contratante dos serviços da distribuidora de gás. O *software* exibe uma página com uma lista, com paginação, dos clientes cadastrados e disponíveis para a criação de um contrato. A lista é apresentada com paginação, porém foi identificado na análise dinâmica, que ao selecionar qualquer página todos os clientes cadastrados são consultados.

Inspecionando a estratégia de consulta de clientes foi observado que a mesma é não escalável e cresce linearmente a quantidade de clientes cadastros na aplicação. Para cada subconjunto de clientes que o usuário solicita para apresentação, é consultado o conjunto completo dos clientes. A estratégia de consulta apresentada é

característica do anti-padrão *The Ramp*, este afirma que: “o desempenho é impactado em qualquer situação em que a quantidade de processamento necessário para atender a uma solicitação aumenta com o tempo” (SMITH; WILLIAMS, 2002).

Para a correção do anti-padrão, na definição do mesmo encontrada na literatura é recomendado a alteração do algoritmo utilizado. Assim, a consulta foi modificada, adicionando a paginação dos dados consultados do banco de dados. Como trata-se de um anti-padrão onde o seu impacto no desempenho é diretamente proporcional ao volume dos dados obtidos do banco de dados, foi solicitado do mantenedor do GGAS, uma base de dados utilizada por um distribuidora de gás, com 9834 clientes cadastrados para a avaliação do impacto da correção. A tabela 8 apresenta os resultados obtidos com a correção do anti-padrão.

**Tabela 8: Identificação do anti-padrão *The Ramp***

| Tempo de resposta do método com o anti-padrão | Tempo de resposta do método após correção do anti-padrão | Redução obtida no tempo de resposta do método |
|---|---|---|
| 31.4 s | 1.2 s | 96.17% |

Fonte: O autor (2018).

Através da análise dinâmica realizada nos módulos de contrato, foi possível observar a arquitetura do GGAS no que se refere a como os componentes do mesmo interagem entre si em tempo de execução. Na orientação a objetos, estes componentes podem ser interpretados como as classes que compõem a aplicação. Esta observação permitiu identificar um anti-padrão conhecido como: The “God” Class. Este anti-padrão define uma classe denominada como *“god” class* que realiza a maior parte do trabalho no *software*, trocando mensagens com outras classes através de chamadas de método, para que estas desempenhem papéis menores (SMITH; WILLIAMS, 2000).

Da perspectiva do desempenho, essa troca de mensagens, se excessiva, pode apresentar problemas (SMITH; WILLIAMS, 2000). Apesar do problema ter sido identificado e a sua correção ser viável, o mesmo não está inserido no contexto de atividades de entrada/saída realizadas no banco de dados, definido na metodologia. Este problema está sendo apresentado neste estudo de caso para demonstrar que foi possível utilizar o processo para identificar anti-padrões em um contexto diferente do proposto na metodologia.

## 4.2 INTEGRAÇÃO DO PROCESSO

Com o objetivo de aplicar o processo para identificação de anti-padrões de desempenho de forma extensiva, validando a efetividade do mesmo, integrou-se o processo a um de integração contínua, executando o processo de identificação de anti-padrões a cada integração de uma nova versão do *software*. Este processo de integração contínua pertence ao processo de desenvolvimento do *software* participante do estudo de caso, e é realizado pelo mantenedor do *software*.[1]

A validação foi realizada de forma empírica, ou seja, a percepção do resultado foi fundamentada na experiência prática da utilização do processo. Em trabalhos futuros, pode-se validar o processo em um ambiente controlado e utilizando métodos estatísticos.

Para a validação, foram selecionadas novas funcionalidades, em diferentes cenários. As funcionalidades são: Consolidar Dados do Supervisório, Consistir Leitura e Faturar Grupo. Sendo a última destas apresentada em dois cenários: Residencial e Industrial. Estes cenários se diferenciam pelas suas regras de negócio, o que altera a execução da funcionalidade e consequentemente permite a observação da ocorrência de diferentes problemas de desempenho

O critério para seleção das funcionalidades foi a importância das mesmas para as regras de negócio do *software*. Elas compreendem um conjunto de processos *batch*, que tem como objetivo, a transformação e validação dos dados para a emissão de faturas dos clientes das distribuidoras de gás, os usuários finais do *software* estudado. Um detalhamento destas funcionalidades pode ser encontrado no manual do usuário do *software*.[2]

Os problemas de desempenho identificados nas funcionalidades avaliadas durante a integração do processo não se limitam a apenas uma ocorrência e são decorrentes da existência de consultas SQL desnecessárias, que obtém dados que não são utilizados, e consultas SQL em estruturas de repetição, características dos anti-padrões *Unecessary Processing* e *Empty Semi Trucks*, respectivamente. Assim, os problemas apresentados no decorrer desta seção não destacam o impacto do anti-padrão no desempenho, como apresentado no estudo de caso, mas apresentam a estratégia

[1] Detalhes sobre processo de desenvolvimento podem ser encontrados em: http://www.ggas.com.br
[2] O manual do usuário encontra-se disponível em: http://docs.ggas.com.br/docs/ggas_documentacao/

comumente utilizada para a correção dos anti-padrões *Unecessary Processing* e *Empty Semi Trucks*.

Na funcionalidade Faturar Grupo, foi possível observar o excesso de código duplicado. Métodos diferentes no código-fonte realizando a mesma consulta dos dados no banco de dados, não existindo o reaproveitando dos dados entre a execução dos métodos. Uma estratégia de *caching* foi implementada, armazenando os dados consultados em memória e disponibilizando estes aos métodos ao invés de realizar a consulta dos dados novamente.

Um dos problemas identificados na funcionalidade de Consolidar dados do Supervisório envolve a consulta no banco de dados de atributos desnecessários de uma coleção de objetos, que após serem obtidos do banco, são submetidos a um procedimento de busca, com o objetivo de obter um objeto da coleção que satisfaça determinada condição. O código-fonte que apresenta o problema de desempenho pode ser visualizado abaixo.

```
Collection < Contrato > listaContratosDataRecisao = this.consultarContratoPorPontoConsumoEPorDataRecisao(pontoConsumo);

if (listaContratosDataRecisao != null) {
 for (Contrato contrato: listaContratosDataRecisao) {
  if (contrato.getDataRecisao().compareTo(dataLeitura) > 0) {
   ContratoPontoConsumo contratoPonto = null;
   if (contrato.getDataRecisao().compareTo(dataLeitura) > 0) {
    for (ContratoPontoConsumo contratoPC: contrato.getListaContratoPontoConsumo()) {
     if (contratoPC != null && contratoPC.getPontoConsumo() != null && contratoPC.getPontoConsumo().equals(pontoConsumo)) {
      contratoPonto = contratoPC;
     }

    }
    if (contratoPonto != null) {
     mapaContratoAtivo.put(Constantes.HORA_INICIAL_CONTRATO, contratoPonto.getNumeroHoraInicial());
     possuiContrato = Boolean.TRUE;
     mapaContratoAtivo.put(Constantes.POSSUI_CONTRATO, possuiContrato);
     mapaContratoAtivo.put(Constantes.PERIODICIDADE_CONTRATO, contratoPonto.getPeriodicidade());
    }
   }
  }
 }
}
```

Para a correção deste problema, foi implementada uma consulta que seleciona apenas os atributos necessários de objetos que satisfazem a condição imposta no procedimento de busca. O código-fonte da consulta implementada pode ser observado abaixo.

```
/**
 * Verifica se existe algum contrato ativo em uma poca de leitura
 * e obt m os atributos: {@code periodicidade}, {@code numeroHoraInicial},
 * {@code contrato} e {@code chavePrimaria} do ContratoPontoConsumo
 * de um Contrato com maior data de recis o
```

```
 * @param pontoConsumo
 * @param dataLeitura
 * @return ContratoPontoConsumo - ContratoPontoConsumo do Contrato ativo na poca de leitura
 **/
@Override
public ContratoPontoConsumo verificarContratoAtivoEpocaLeitura(PontoConsumo pontoConsumo, Date dataLeitura) {

 ContratoPontoConsumo contratoPontoConsumo = null;
 Criteria criteria = createCriteria(getClasseEntidadeContratoPontoConsumo());
 criteria.createAlias("contrato", "contrato");
 criteria.setProjection(Projections.projectionList()
  .add(Projections.property(ATRIBUTO_PERIODICIDADE), ATRIBUTO_PERIODICIDADE)
  .add(Projections.property(ATRIBUTO_NUMERO_HORA_INICIAL), ATRIBUTO_NUMERO_HORA_INICIAL)
  .add(Projections.property("contrato"), "contrato")
  .add(Projections.property(CHAVE_PRIMARIA), CHAVE_PRIMARIA));


 criteria.add(Restrictions.isNotNull(ATRIBUTO_CONTRATO_DATA_RECISAO));
 criteria.add(Restrictions.eq("pontoConsumo.chavePrimaria", pontoConsumo.getChavePrimaria()));
 criteria.add(Restrictions.gt(ATRIBUTO_CONTRATO_DATA_RECISAO, dataLeitura));
 criteria.addOrder(Order.asc(ATRIBUTO_CONTRATO_DATA_RECISAO));

 criteria.setResultTransformer(Transformers.aliasToBean(getClasseEntidadeContratoPontoConsumo()));

 List<ContratoPontoConsumo> listaContratoPontoConsumo = criteria.list();
 if (listaContratoPontoConsumo != null && !listaContratoPontoConsumo.isEmpty()) {
  contratoPontoConsumo = listaContratoPontoConsumo.get(listaContratoPontoConsumo.size() - 1);
 }
 return contratoPontoConsumo;
}
```

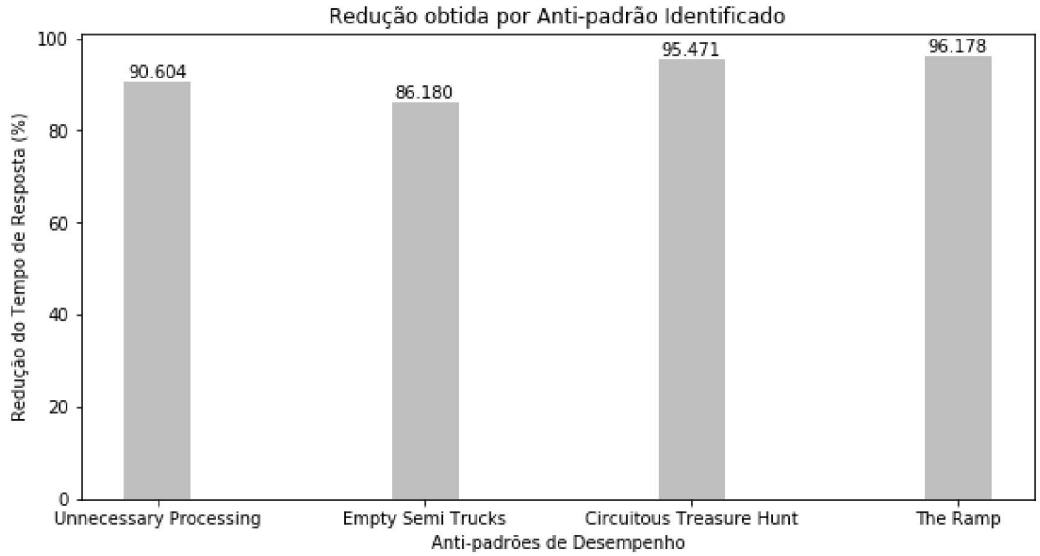
Na funcionalidade Consistir Leitura e Consumo, um percentual do esforço para a correção de anti-padrões foi dedicado a remoção das consultas SQL de estruturas de repetição. Para a correção, foi utilizada uma tabela *hash* (*Hashtable*) como estrutura de dados. Os objetos são consultados e mapeados na tabela *hash*, utilizando os objetos como valores e os parâmetros de consulta destes como chave. Em lugar de consultar os objetos em estruturas de repetição múltiplas vezes, desnecessariamente, pode-se obter os objetos da tabela *hash* em memória, apenas fornecendo o parâmetro da consulta que seria realizada. O código-fonte de uma solução implementada pode ser visualizado abaixo.

```
/**
 * Obtm as unidades padr o para convers o de temperatura e press o.
 * @return mapa de Unidade por constantes de ClasseUnidade
 */
@Override
public Map<String, Unidade> obterUnidadesPadraoTemperaturaPressao() throws NegocioException {
 Map<String, Unidade> unidadesPadraoParaConversao = new HashMap<>();
 Unidade unidadePadraoTemperatura = this.obterUnidadePadraoTemperatura();
 unidadesPadraoParaConversao.put(ClasseUnidade.TEMPERATURA, unidadePadraoTemperatura);
 Unidade unidadePadraoPressao = this.obterUnidadePadraoPressao();
 unidadesPadraoParaConversao.put(ClasseUnidade.PRESSAO, unidadePadraoPressao);
 return unidadesPadraoParaConversao;
}
```

## 4.3 CONSOLIDAÇÃO DOS RESULTADOS

O estudo de caso realizado no módulo de acesso e no módulo de contrato permitiu atestar a aplicabilidade do processo desenvolvido e resultou na identificação e correção de anti-padrões de desempenho. A Figura 6 apresenta um gráfico que descreve a melhoria de desempenho obtida, através da redução do tempo de resposta. Foi considerada a redução mais significativa para cada categoria de anti-padrão identificado. Este gráfico permite observar uma redução de tempo de resposta entre 86.0 e 96.0 por cento.

**Figura 6: Reduções obtidas agrupadas por anti-padrão de desempenho.**

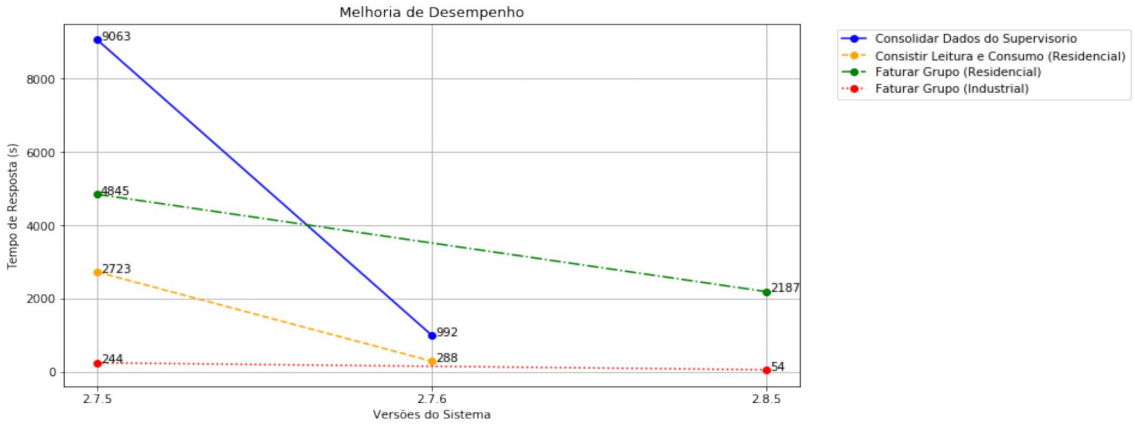



Fonte: O autor (2018).

A integração com o processo de desenvolvimento do *software* resultou em melhorias no desempenho das funcionalidades, que foram disponibilizadas em versões para o usuário final do *software*. O gráfico da Figura 7 apresenta a redução do tempo de resposta obtido para cada funcionalidade.

Dada a característica *open source* do *software* analisado e a existência de um processo de integração contínua que o suporta, diferentes desenvolvedores realizam a manutenção ou implementação de novas funcionalidades e consequentemente a integração destas no *software*. As constantes entregas alteram o *software* e por consequência dificultam a etapa do processo de inspeção no código-fonte. É desejável

**Figura 7: Melhoria de Desempenho em Processos do GGAS**




Fonte: O autor (2018).

que esta condição do código-fonte constasse na documentação do processo.

Ao realizar a validação do processo desenvolvido, foi possível identificar uma repetição excessiva nas soluções aplicadas aos problemas. A estrategia utilizada para a correção de uma ocorrência de determinado anti-padrão é estruturada de maneira semelhante, independente da funcionalidade. Assim, etapas no processo que instruam a utilização de artefatos de código reutilizáveis que forneçam uma solução genérica para as correções realizadas são desejáveis.

Durante as inspeções manuais realizadas, foi possível observar que uma das possíveis causas para a ocorrência dos anti-padrões classificados como *Unnecessary Processing* é a reutilização não controlada de código-fonte, visto que muitos dos problemas de desempenho identificados são resultantes da reutilização de funcionalidades que possuem códigos não coesos e na maioria das vezes desnecessários.

Uma carência apresentada pelo processo e que se tornou aparente ao executarmos o mesmo durante as iterações de um processo de integração contínua, foi a necessidade de implementação de testes unitários para as correções realizadas. A complexidade dos problemas identificados e corrigidos e a criticidade das funcionalidades analisadas, demonstrou que se faziam necessárias atividades no processo voltadas para a implementação de testes, com o objetivo de validar as correções realizadas.

# 5 CONCLUSÃO

O processo para a identificação de anti-padrões apresentado contempla um conjunto de atividades e práticas, executadas efetivamente em trabalhos apresentados na literatura. Neste trabalho, estas atividades modeladas em um processo através do BPMN, apresentaram uma visão de um processo que permitiu realizar a identificação, correção e medição de anti-padrões de desempenho. Utilizar as características dos anti-padrões, assim como as recomendações de correção destes, que são convencionadas pela literatura, permitiu identificar e corrigir problemas de desempenho de forma rápida e objetiva.

O processo proposto neste trabalho melhora a qualidade do processo de desenvolvimento do *software* participante do estudo de caso, visto que pode-se utilizar o processo de identificação e correção de anti-padrões para realizar uma melhoria contínua da qualidade do *software* durante o seu desenvolvimento.

Como trabalhos futuros, pode-se utilizar o processo definido para a identificação de anti-padrões de desempenho em diferentes *softwares* armazenados no SPB, estendendo e ampliando o catálogo de anti-padrões abordados. Além disso, pode ser desenvolvido um *software* especialista que ajude o desenvolvedor a identificar e corrigir os anti-padrões, mapeando assim o processo através da ferramenta.

É importante apontar que este trabalho foi publicado por Santana, Alencar e Correia (2017) em uma revista cientifica. O estudo de anti-padrões abordado neste trabalho, ainda não é explorado de forma abrangente na literatura no que se refere a aplicação prática de melhorias de desempenho em *softwares*, através da correção de anti-padrões. E apresentar um abordagem prática, de identificação e correção de anti-padrões, pode auxiliar na percepção que, de fato, estes são extremamente frequentes nos *softwares* atuais e são degradantes para a qualidade destes.

Este trabalho fornece ao SPB um processo que permite avaliar a qualidade do desempenho de seus *softwares* e ajuda a preencher a lacuna de conhecimento sobre o estado atual em que se encontram alguns dos *softwares* hospedados atualmente no Portal de Software Público Brasileiro, no quesito qualidade em desempenho, e por consequência, favorecem o planejamento de um processo de controle de qualidade e auditoria de *software*, processo este que poderia contribuir para aumentar a qualidade

dos softwares armazenados no SPB.

# REFERÊNCIAS

ALEXANDER, Christopher et al. **A Pattern Language: Towns, Buildings, Construction**. New York: Oxford University Press, 1977.

BAUER, Christian; KING, Gavin. **Java Persistance with Hibernate**. Shelter Island, NY: Manning Publications, 2016.

BENEDICT, Tony et al. **BPM CBOK**: Guide to the Business Process Management Common Body Of Knowledge. Charleston, SC: CreateSpace Independent Publishing Platform, 2013.

BRASIL, Portal. **RN:** uso de *software* público gera economia de R$ 2,4 mi. 2015. Disponível em: <http://www.brasil.gov.br/economia-e-emprego/2015/09/rn-uso-de-software-publico-gera-economia-de-r-2-4-mi>. Acesso em: 23 de jan. 2016.

BROWN, William J. et al. **AntiPatterns: refactoring software, architectures, and projects in crisis**. New York: John Wiley & Sons Inc., 1998.

CAPOTE, Gart. **BPM para todos**: Uma visão geral abrangente, objetiva e esclarecedora sobre gerenciamento de processos de negócio. Rio de janeiro: Gart Capote, 2012.

CHEN, Tse-Hsun et al. Detecting Performance Anti-patterns for Applications Developed Using Object-relational Mapping. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 36., 2014, Hyderabad, India. **Anais...** Hyderabad, India: ACM, 2014. p. 1001–1012.

COPPA, Emilio; DEMETRESCU, Camil; FINOCCHI, Irene. Input-Sensitive Profiling. **IEEE Transactions on Software Engineering**, IEEE, v. 40, n. 12, p. 1185–1205, 2014.

DIN, Jamilah; AL-BADAREEN, Anas Bassam; JUSOH, Yusmadi Yah. Antipatterns detection approaches in object-oriented design: A literature review. In: INTERNATIONAL CONFERENCE ON COMPUTING AND CONVERGENCE TECHNOLOGY (ICCCT), 7., 2012, Seoul, Korea. **Anais...** Seoul, Korea: IEEE, 2012. p. 926–931.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistema de Bando de Dados**. São Paulo: Addison Wesley, 2011.

GAMMA, Erich et al. **Padrões de Projeto**: Soluções Reutilizáveis de *Software* Orientado a Objetos. Porto Alegre: Bookman, 2000.

GRINSHPAN, Leonid. **Solving Enterprise Applications Performance Puzzles**. Hoboken, New Jersey: John Wiley & Sons Inc., 2012.

KESSENTINI, Marouane et al. Search-based Design Defects Detection by eventtitle. In: INTERNATIONAL CONFERENCE ON FUNDAMENTAL APPROACHES TO SOFTWARE ENGINEERING: PART OF THE JOINT EUROPEAN CONFERENCES ON

THEORY AND PRACTICE OF SOFTWARE, 14., 2011, Saarbrücken, Germany. **Anais...** Saarbrücken, Germany: [s.n.], 2011. p. 401–415.

KHOMH, Foutse et al. BDTEX: A GQM-based Bayesian approach for the detection of antipatterns. **Journal of Systems and Software**, Elsevier, v. 84, n. 4, p. 559–572, 2011.

MATAM, Sai; JAIN, Jagdeep. **Pro Apache JMeter**: Web Application Performance Testing. United States: Apress, 2017.

MEIRELES, Paulo. **Sobre o Portal**. 2015.
Disponível em: <https://softwarepublico.gov.br/social/spb/sobre-o-portal>. Acesso em: 23 de jan. de 2016.

MIHALCEA, Vlad. **High-performance Java persistence**. Cluj-Napoca, Romania: Vlad Mihalcea, 2016.

MIHALCEA, Vlad et al. **Hibernate ORM User Guide**. 2017. Disponível em: <https://docs.jboss.org/hibernate/orm/5.2/userguide/html_single/Hibernate_User_Guide.html>. Acesso em: 30 dez. 2017.

MINISTÉRIO PÚBLICO FEDERAL. **Manual de Gestão por Processos**. Brasília, 2013. 53 p.

MOLYNEAUX, Ian. **The Art of Application Performance Testing**: Help for Programmers and Quality Assurance. Sebastopol, CA: O'Reilly Media, Inc, 2009.

NAYROLLES, Mathieu; MOHA, Naouel; VALTCHEV, Petko. Improving SOA antipatterns detection in Service Based Systems by mining execution traces. In: IEEE WORKING CONF. REVERSE ENG, 20., 2013, Koblenz, Germany. **Anais...** Koblenz, Germany: IEEE, 2013. p. 321–330.

PRAKASH, Sudeepa. **ZeroTurnaround Releases New Version of its Lightweight Java Profiler**. 2015.
Disponível em: <https://zeroturnaround.com/rebellabs/zeroturnaround-releases-new-version-of-its-lightweight-java-profiler-xrebel>. Acesso em: 23 dez. 2015.

PROJECT MANAGEMENT INSTITUTE. **Um Guia do Conhecimento em Gerenciamento de Projetos**. Newtown Square, PA: Project Management Institute, 2013.

RADATZ, Jane; GERACI, Anne; KATKI, Freny. IEEE standard glossary of software engineering terminology. **IEEE Std**, v. 610121990, n. 121990, p. 84, 1990.

RISING, Linda. **The Patterns Handbook**: Techniques, Strategies, and Applications. Cambridge, UK: Cambridge University Press, 1998. v. 13. (SIGS Reference Library).

SANTANA, Edmilson; ALENCAR, Roberto; CORREIA, Carlos. An Analysis of Performance Anti-Patterns in Systems Stored on the Brazilian Public Software Portal. **IEEE Latin America Transactions**, IEEE, v. 15, n. 4, p. 705–710, 2017.

SANTOS, C. H. **Incrementando a codificação da Notação e Modelo de Processo de Negócio**. 2016. Diss. (Mestrado) – Instituto de Informática, Universidade Federal do Rio Grande do Sul, Rio Grande do Sul.

SMITH, C. U.; WILLIAMS, L. G. More New Software Antipatterns: Even More Ways to Shoot Yourself in the Foot. In: INTERNATIONAL COMPUTER MEASUREMENT GROUP CONFERENCE, 29., 2003, Dallas, Texas. **Anais...** Dallas, Texas: Computer Measurement Group, 2003. p. 717–725.

_____. New Software Performance AntiPatterns: More Ways to Shoot Yourself in the Foot. In: INTERNATIONAL COMPUTER MEASUREMENT GROUP CONFERENCE, 28., 2002, Reno, Nevada. **Anais...** Reno, Nevada: Computer Measurement Group, 2002. p. 667–674.

_____. **Performance Solutions**: A Practical Guide to Creating Responsive Scalable Software. Boston, MA: Addison-Wesley, 2001a.

_____. Software Performance AntiPatterns: Common Performance Problems and their Solutions. In: INTERNATIONAL COMPUTER MEASUREMENT GROUP CONFERENCE, 27., 2001, Anaheim, CA. **Anais...** Anaheim, CA: Computer Measurement Group, 2001b. p. 797–806.

_____. Software Performance Antipatterns. In: WORKSHOP ON SOFTWARE AND PERFORMANCE, 2., 2000, Ottawa, Ontario, Canada. **Anais...** Ottawa, Ontario, Canada: ACM, 2000. p. 127–136.

SOMMERVILLE, Ian. **Engenharia de *Software***. São Paulo: Pearson Prentice Hall, 2011.

STEPHAN, Matthew; CORDY, James R. Identifying Instances of Model Design Patterns and Antipatterns Using Model Clone Detection. In: INTERNATIONAL WORKSHOP ON MODELING IN SOFTWARE ENGINEERING, 7., 2015, Florence, Italy. **Anais...** Florence, Italy: IEEE Press, 2015. p. 48–53.

VENTURA, Magda Maria. O estudo de caso como modalidade de pesquisa. **Revista SoCERJ**, Rio de Janeiro, v. 20, n. 5, p. 383–386, 2007.

WEGRZYNOWICZ, Patrycja. Performance antipatterns of one to many association in hibernate. In: FEDERATED CONFERENCE ON COMPUTER SCIENCE AND INFORMATION SYSTEMS (FEDCSIS), 2013, Kraków, Poland. **Anais...** Kraków, Poland: IEEE, 2013. p. 1475–1481.

WHITE, Stephen A. **Introduction to Bpmn**. 2015.
Disponível em: <https://www.omg.org/bpmn/Documents/Introduction_to_BPMN.pdf>.
Acesso em: 23 dez. 2015.