UNIVERSIDADE CESUMAR - UNICESUMAR
CENTRO DE CIÊNCIAS EXATAS TECNOLÓGICAS E AGRÁRIAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

ANÁLISE DE VULNERABILIDADES E REFATORAÇÃO DE CODE SMELLS EM CÓDIGOS C#: UM ESTUDO DE CASO

LUCAS MATHEUS NAKASIMA SOARES

Maringá-PR

2021

LUCAS MATHEUS NAKASIMA SOARES

ANÁLISE DE VULNERABILIDADES E REFATORAÇÃO DE CODE SMELLS EM CÓDIGOS C#: UM ESTUDO DE CASO

Artigo apresentado ao Curso de Graduação em Engenharia de Software da Universidade Cesumar - UNICESUMAR como requisito parcial para a obtenção do título de Bacharel(a) em Engenharia de Software.

Orientador: Prof. Me. Maurilio Martins Campano Junior

Maringá-PR

2021

FOLHA DE APROVAÇÃO

LUCAS MATHEUS NAKASIMA SOARES

ANÁLISE DE VULNERABILIDADES E REFATORAÇÃO DE CODE SMELLS EM CÓDIGOS C#: UM ESTUDO DE CASO

Trabalho de conclusão de curso de graduação apresentado ao Centro de Ciências Exatas Tecnológicas e Agrárias da Universidade Cesumar – UNICESUMAR como requisito parcial para a obtenção do título de Bacharel(a) em Engenharia de Software, sob a orientação do Prof. MAURILIO CAMPANO JUNIOR.

BANCA EXAMINADORA

BANCA

RESUMO

Um *code smell* é uma indicação superficial que geralmente corresponde a um problema maior no código. Sendo assim, a presença dos *code smells* são indícios de que um código-fonte pode ser melhorado, podendo chegar a condenar uma aplicação se não tratados a tempo, levando então a necessidade da refatoração desses códigos. Visando esse contexto, este trabalho tem como objetivo realizar um estudo de caso sobre *code smells* em códigos de linguagem de programação C#. Realizando a detecção e análise das vulnerabilidades dos *code smells* em um projeto já existente, sequentemente com o auxílio de ferramentas foi realizada a remoção e refatoração dos casos detectados. Obtendo como resultado dos estudos aplicados no projeto, diversas melhorias na qualidade do código-fonte e remoção das vulnerabilidades detectadas, seguindo as métricas do *code smell*.

Palavras-chave: *Code smells*. Linguagem de programação C#. Análise de vulnerabilidades. Refatoração.

ABSTRACT

A *code smell* is a superficial indication that generally corresponds to a major problem in the code. Thus, the presence of *code smells* is an indication that a source code can be improved, and may even condemn an application if not treated in time, thus leading to the need for refactoring these codes. In this context, this work aims to carry out a case study about *code smells* in C# programming language codes. Performing the detection and analysis of *code smells* vulnerabilities in an existing project, subsequently with the help of tools, the detected cases were removed and refactored. Obtaining as a result of the studies applied in the project, several improvements in the source code quality and removal of detected vulnerabilities, following the *code smell* metrics.

Keywords: *Code smells*. C# programming language. Vulnerability analysis. Refactoring.

1 Introdução

A computação de forma geral, está sempre em constante evolução mudando e facilitando a vida das pessoas no decorrer da história da humanidade. Seja por exemplo, com a criação de telecomunicações, ou até mesmo passando pela automação de processos complexos (BARRA et al., 2006).

Grande parte desses benefícios gerados pela tecnologia, são alcançados através do desenvolvimento de *softwares*. Ato ao qual, é realizado um conjunto de procedimentos estratégicos e organizados com o propósito de arquitetar, desenvolver, manter e testar uma aplicação. Sempre buscando desenvolver um produto ao qual a união da tecnologia e estratégia, se torne uma solução para necessidades de um negócio (PÁDUA PAULA FILHO, 2003).

Um *software* em sua estrutura básica, é composto por uma série de conjunto de códigos interligados e relacionados, utilizando determinadas linguagens da computação que acabam formando um produto virtual. Sendo assim, os usuários ao utilizar esses produtos não visualizam os códigos, pois essas instruções são traduzidas ou materializadas para ser apresentadas de forma diferente nas telas dos aparelhos eletrônicos (PÁDUA PAULA FILHO, 2003).

Então a etapa de desenvolvimento de códigos, se torna um processo complexo ao qual pode envolver diversos profissionais, sendo que o papel do desenvolvedor é criar diversas sequencias de códigos podendo ser um conjunto de ordens, ações consecutivas, dados e algoritmos. Esses códigos quando executados, fazem com que a aplicação opere e execute suas funções para qual foi programada, permitindo o programa criado controlar comportamentos físico e lógico de uma máquina (MEDINA; FERTING, 2006).

Para o desenvolvimento dos códigos, os programadores utilizam-se das linguagens de programação, que devem ser dominadas pelos profissionais da área. As linguagens funcionam como um sistema de comunicação estruturado, composto por conjuntos de símbolos, palavras-chave, regras semânticas e sintáticas, que permitem o entendimento entre um programador e uma máquina (MEDINA; FERTING, 2006).

Porém, quando o desenvolvedor não possui o domínio necessário sobre a linguagem, ele está muito propenso a acabar gerando *bugs* no *software* ao qual está desenvolvendo, gerando falhas no momento de executar a aplicação, resultando em um comportamento incorreto, inesperado ou fora do que tenha sido pretendido pelo desenvolvedor. Essas falhas podem desencadear em diversos problemas incômodos, como perda de desempenho, travamentos e roubo de informações sigilosas (MEDINA; FERTING, 2006).

Um dos erros mais comuns, que são cometidos por profissionais que não possuem

o domínio necessário da tecnologia utilizada no processo de desenvolvimento de códigos, são os *code smells*. Que segundo (FOWLER, 1999), um *code smell* é uma indicação superficial que geralmente corresponde a um problema maior no código. E apesar de não serem *bugs*, certamente podem condenar um *software* se não forem tratados a tempo, sendo um indicador de que há um problema maior na aplicação, mas podendo não ser o problema em si, levando então a necessidade da refatoração desses códigos.

Sendo também um dos fatores que mais contribuem para geração dos *code smells*, as diversas intervenções realizadas por diferentes desenvolvedores durante o ciclo de vida de um *software*, causando inúmeros problemas estruturais no mesmo.

Desde que o termo *code smells* foi utilizado pela primeira vez, ele vem sendo catalogado, possibilitando atualmente dividir os tipos em grupos. Como por exemplo, o *code smell Duplicated Code*, que pode acabar diminuindo a performance do *software*, causando problemas em atividades na fase de manutenção e aumenta as chances da introdução de erros na aplicação (LANZA; MARINESCU, 2010).

Visando esse contexto, o presente trabalho tem como objetivo a realização de um estudo de caso onde foi desenvolvido uma análise de vulnerabilidades do tipo *code smells* e na sequência sua refatoração. Tendo como objeto de estudo uma base de códigos na linguagem de programação C#, pertencentes ao projeto Agrinvest da empresa Bitzen Tecnologia de Maringá, PR.

Para atingir o objetivo, foi efetuado por meio da utilização e estudo das ferramentas Resharper e SonarQube, a realização da detecção dos *code smells* presentes na base, analisando os tipos detectados, suas vulnerabilidades, impactos negativos e por fim realizando a refatoração dos mesmos. Obtendo os dados gerais do projeto e seus *code smells* mapeados por meio da análise efetuada, sendo sequentemente tratados com o intuito de melhorar a qualidade do código final e remover suas vulnerabilidades.

2 Trabalhos Relacionados

O trabalho de Freitas (FREITAS, 2017), apresenta um estudo sobre a relação entre *code smells* e problemas de design, se utilizando de estudos qualitativos em micro-empresas de *software*, para a realização do desenvolvimento do trabalho.

Tendo em vista que o *code smell* é um potencial problema de *desing* localizado em determinado parte do código de fonte, e pode indicar deficiências que podem influenciar na facilidade de manutenção, de testes e confiabilidade do *software*. O autor utilizou três tipos de estudos, sendo um estudo preliminar e dois estudos experimentais, com o foco em investigar a qualidade do design de sistemas de *softwares* através da utilização de

ferramentas de detecção. Obtendo como resultado a indicação que existem relação entre os *Code Smells God Class*, *Data Class*, *Shotgun Surgery*, *Refused Bequest* e *Feature Envy* com problemas de design associados a dificuldades no processo de manutenção de sistemas de *softwares* (FREITAS, 2017).

Já o trabalho de Flauzino (FLAUZINO et al., 2021), aborda o tema de *Pull Requests* e *Code Smells* na qual foram realizadas análises em aplicações, sendo um estudo empírico para investigar a correlação de code smells em aplicativos móveis, juntamente com suas Issues e resolução com Pull Request feitas por colaboradores no GitHub. Para a realização do trabalho, foram analisados 4 aplicativos Android e suas Issues que apresente Pull Request, obtendo como resultado final uma constatação que nem sempre as soluções de bugs sugeridas por usuários apresentam um quadro de melhora no números de *smells*.

O trabalho de Regis e Oliveira (REGIS; OLIVEIRA, 2018), apresenta os conceitos relacionados a adoção de boas práticas no desenvolvimento de *software*, por meio de pesquisas bibliográficas. Sendo definindo que anomalias em códigos são trechos de códigos fonte com problemas estruturais, que podem dificultar processos como a manutenção do código e a evolução do *software*, a pesquisa então tem como contexto a melhora da qualidade estrutural do código fonte.

Sendo assim o objetivo do autor foi levantar as práticas adotadas para detecção de anomalias de código, e para alcança-los o autor realizou mapeamento sistemático, tendo como resultados que as práticas mais adotadas para detecção de anomalias de código são as ferramentas, métricas e técnicas. E os dados levantados podem auxiliar na escolha de práticas, e na formação de futuros desenvolvedores (REGIS; OLIVEIRA, 2018).

Já o trabalho do Maciel (MACIEL et al., 2012), apresenta um experimento na indústria para mineração visual de padrões de interações de programadores, tendo em vista que as ferramentas de visualização de *software*, tem o objetivo de tornar a compreensão de *software* mais efetiva. Para o desenvolvimento do artigo o autor se utilizou de recursos de mineração de dados para analisar e visualizar estratégias adotadas na execução de atividades de compreensão de *software* apoiadas por ferramentas de visualização. Obtendo como resultados a revelação de que as estratégias com os melhores valores de precisão e revogação.

Junior e Farah (JUNIOR; FARAH, 2019), realizaram um estudo que avalia os *code smells* e padrões de projeto em aplicações que usam as tecnologias *Angular* e *React*. Os autores analisaram 15 aplicações desenvolvidas com *Angular* e 15 aplicações desenvolvidas com *React*, analisando a presença de *code smells* e os padrões utilizados nesses *softwares*. Os resultados obtidos, sugerem que ferramentas sejam desenvolvidas para auxiliar na identificação e redução de *code smells* e na aplicação de mais padrões para essas

tecnologias.

O trabalho de Gomes (GOMES et al., 2019) por sua vez, tem como objetivo estudar a relação entre *code smells* e dívida técnica auto-admtida. Sabendo que *code smells* podem representar problemas no código que podem levar a refatoração do mesmo. O autor realizou um analise de três projetos de código aberto para investigar a relação existente entre eles. Os resultados obtidos pelo autor indicam que as informações de comentários podem complementar as informações de code smells e apoiar os desenvolvedores a identificar DT.

3 Metodologia

Visando atingir o objetivo proposto neste trabalho, foi utilizado como objeto de estudo o projeto da Agrinvest, ao qual foi desenvolvido utilizando a linguagem de programação C#, pertencente a empresa de desenvolvimento de *softwares*, Bitzen Tecnologia de Maringá, PR. Que para maior conhecimento, o projeto considerado se trata de uma *Web Api*, que atualmente serve um aplicativo *mobile* e uma aplicação *Web*.

Para a construção da pesquisa e afim de iniciar o desenvolvimento da análise de vulnerabilidades, o primeiro procedimento realizado, foi a detecção de *code smells* presentes na base de códigos do projeto, por meio da utilização da ferramenta SonarQube. O SonarQube é uma ferramenta que auxilia seus usuários na avaliação da qualidade e segurança dos seus códigos, permitindo analisar o projeto em questão por completo, detectando todos os tipos de *code smells* presentes no mesmo (SONARSOURCE, s.d.).

Logo após, foi realizado uma classificação de forma geral dos tipos de *code smells* detectados, analisando seus diferentes tipos, suas vulnerabilidades e impactos negativos no projeto.

Por fim, com o auxílio do Resharper, foi feito a remoção dos *code smells* mapeados, através da refatoração dos seus códigos e apresentando alguns exemplos da refatoração feita por meio de figuras. O Resharper é uma extensão do editor de códigos Visual Studio para desenvolvedores .NET, que auxilia seus usuários no momento em que vão desenvolver seus códigos, tendo como uma de suas principais funcionalidades, oferecer várias soluções para a refatoração de diversos tipos de *code smells* (JETBRAINS, s.d.).

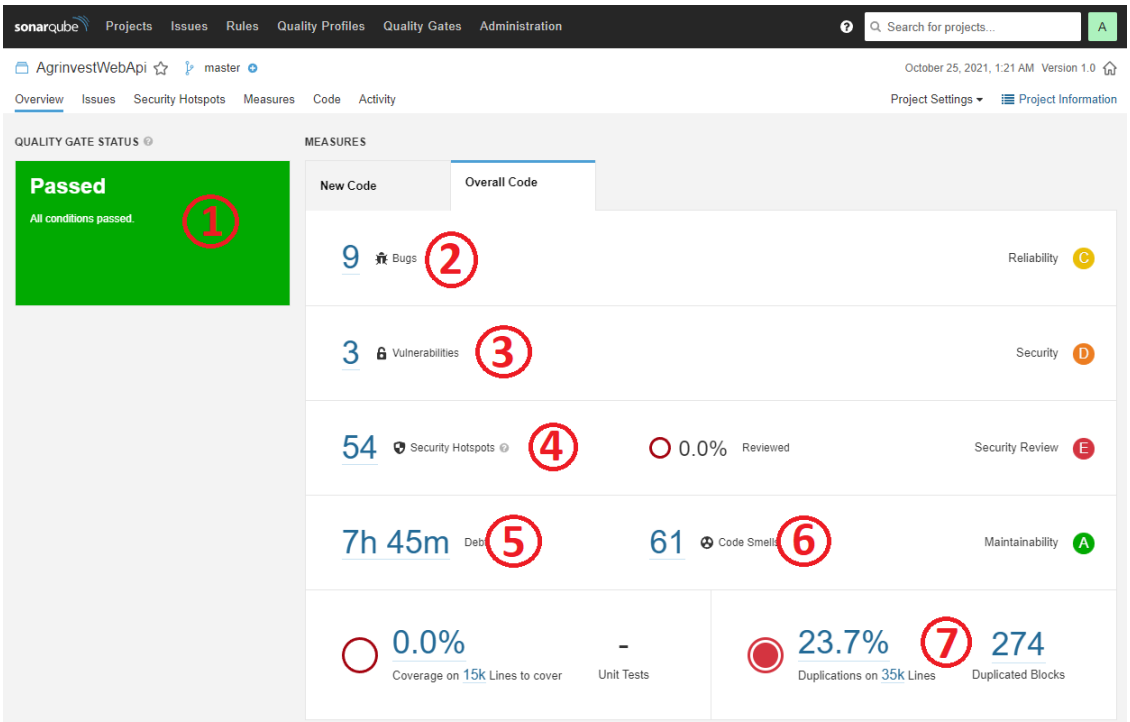
4 Resultados e Discussões

A partir da inspeção completa da aplicação Agrinvest *Web Api* por meio da ferramenta SonarQube. Foram encontrados um total de 61 *code smells* detectados na linguagem de programação C#. Levando em conta que o projeto analisado, possui uma quan-

tidade próxima a 20.000 linhas de códigos, e aproximadamente 2.000 funções distribuídas em 344 classes, de acordo as informações cedidas pela própria análise do SonarQube.

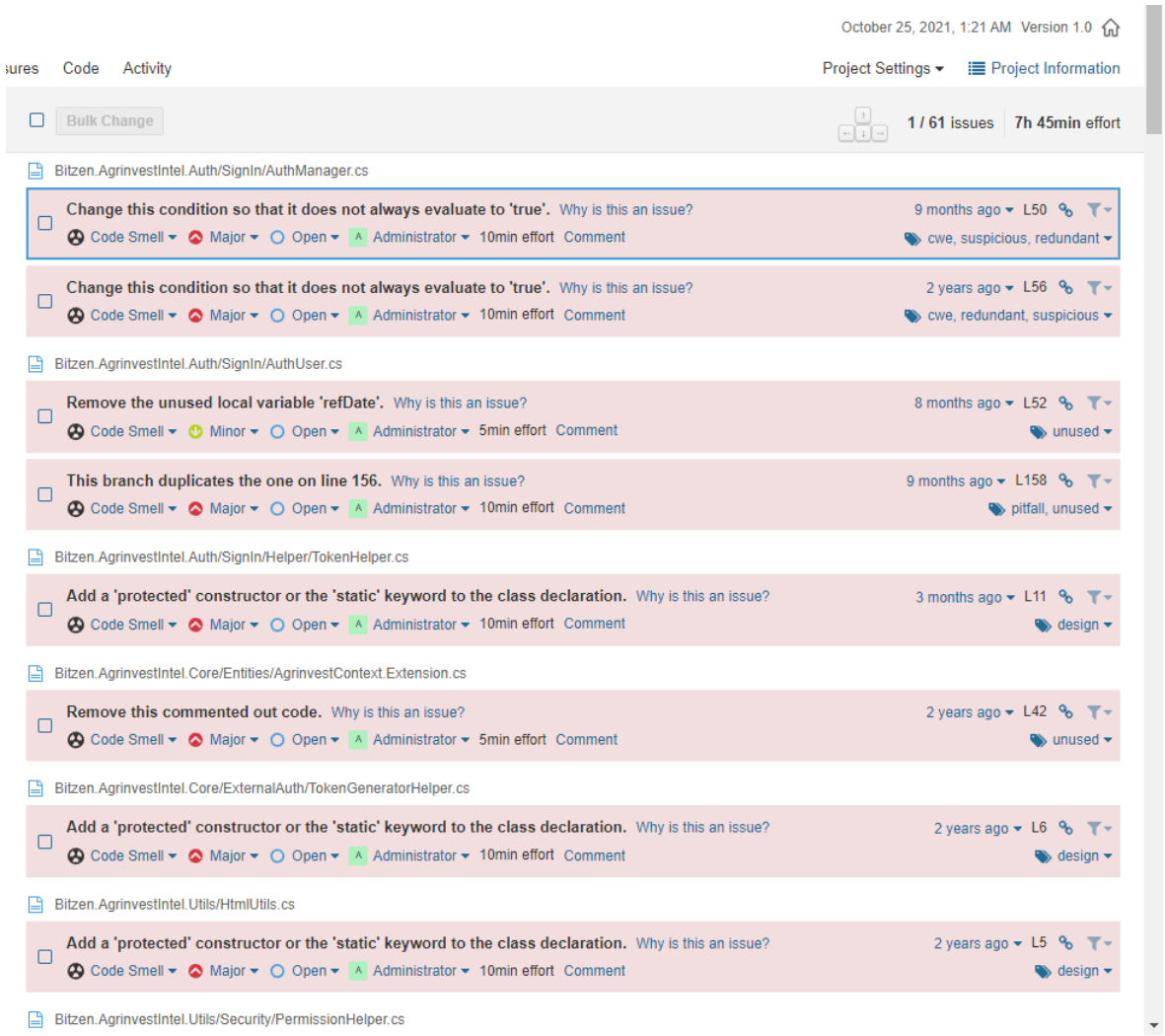
A Figura 1 apresenta os principais dados da análise feita pelo SonarQube no projeto em questão. O Elemento 1 mostra se o projeto está pronto para a produção, por meio de uma análise de um conjunto de condições baseados em medidas de qualidade. O Elemento 2 mostra a quantidade de *bugs* detectados pela ferramenta. O Elemento 3 mostra a quantidade de vulnerabilidades detectadas no projeto. O Elemento 4 mostra a quantidade de possíveis vulnerabilidades que podem estar ameaçando a segurança da aplicação. O Elemento 5 mostra a quantidade de tempo estimada para resolver todos os *code smells* presentes no projeto. O Elemento 6 mostra a quantidade total de *code smells* detectados. E por fim o Elemento 7 mostra a porcentagem e o número de blocos com linhas de códigos duplicadas no projeto.

Figura 1 – Dados da análise do SonarQube



Com os dados gerais obtidos do projeto, é possível visualizar todos os *code smells* em formato de lista. Que por meio das funcionalidades da ferramenta é apresentado soluções e informações mais detalhadas de cada caso detectado, que serão os dados utilizados neste estudo. Na Figura 2 abaixo são apresentados os *code smells* detectados pelo SonarQube.

Figura 2 – Listagem dos *code smells* detectados pelo SonarQube



Apesar do SonarQube conseguir detectar esses *code smells*, a ferramenta não os classifica de acordo aos seus tipos existentes. A aplicação realiza uma classificação dos itens por *tags*, só que não é devidamente precisa, por tanto a maior partes das *tags* não condiz com os tipos de *code smells* catalogados na atualidade. Sendo assim, foi realizado uma análise individual de cada *code smell* pertencente a lista, com intuito de conseguir classificá-los de acordo ao seu tipo.

Então os tipos de *code smells* encontrados e considerados para este estudo foram:

Dead code, *Duplicate Code*, *Indecent Exposure*, *Temporary Field*, *Long Method*, *Object-Orientation Abusers*, *Uncommunicative Name*. A Tabela 1 abaixo apresenta a classificação dos *code smells* encontrados no projeto de acordo com os tipos citados.

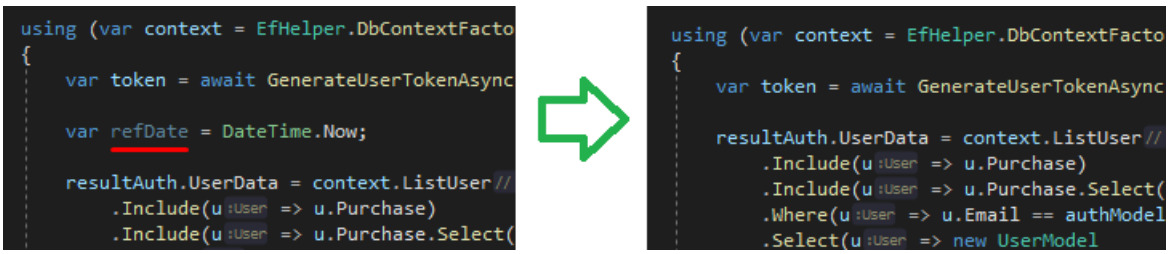
Dead Code	Duplicated Code	Indecent Exposure	Long Method	Temporary Field	OOP Abusers	Uncommunicative Name
27	3	8	12	1	4	6

Tabela 1 – Classificação dos tipos dos *code smells* encontrados

Os dados demonstram, que a maior parte dos *code smells* encontrados no projeto são do tipo *Dead Code* (27). O *Dead Code* basicamente são códigos que estão presentes na aplicação porém não são utilizados, geralmente porque se tornaram obsoletos. Aumentando o tamanho do código desnecessariamente, e tornando o suporte ao código mais complexo (FOWLER, 1999). Analisando os casos separadamente, é possível verificar alguns métodos que foram refeitos, porém suas versões anteriores não foram removidas, e uma grande quantidade de variáveis declaradas que não são utilizadas.

No caso dos *code smells* do tipo *Dead Code*, a solução para resolução do problema é bem simples. Basta localizar os *code smells* apontados pelo SonarQube, e com a utilização da extensão do Resharper, a ferramenta de refatoração indica que o código não está sendo utilizado, e tem como opção de ação para resolução do problema a exclusão do código em questão. A Figura 3 abaixo apresenta o código original do projeto com o *Dead Code*, e o código após sua refatoração.

Figura 3 – Exemplo de *Dead Code* antes e depois da refatoração realizada



Outra categoria de *code smell* encontrado foi o *Duplicated Code*, que de acordo a análise estão presentes cerca de 3 deles dentro do projeto estudado. O *Duplicated Code* se trata da duplicação de códigos, dados ou algoritmos em mais de um lugar no projeto, sendo trechos de códigos que podem até parecer diferentes mas no final acabam executando o mesmo trabalho. Causando um aumento desnecessário no código e aumentando sua complexidade para manutenções futuras (LANZA; MARINESCU, 2010).

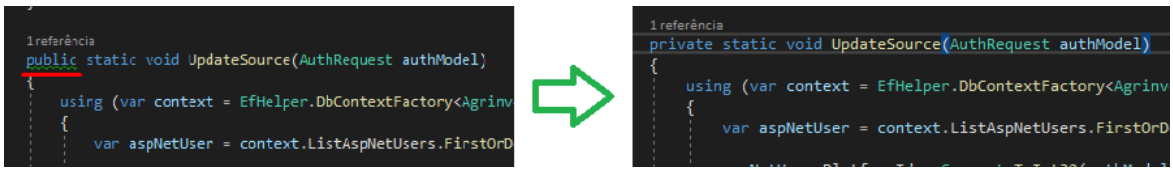
Analisando os três casos alertados pela ferramenta, foi possível verificar que as duplicações ocorreram em blocos de condições, onde foram feitas validações duplicadas dos mesmos dados, provavelmente causado por uma desatenção do desenvolvedor responsável. Como os nossos casos de *Duplicated Code* se trata só de validações duplicadas, ao localizar os *code smell* no arquivo, o Resharper alerta as duplicações de código e oferece como resolução a mudança dos dados validados, ou a remoção do bloco de condição. O exemplo de *Duplicated Code* detectado no projeto, pode ser visto na Figura 4 abaixo junto a sua refatoração.

Figura 4 – Exemplo de *Duplicated Code* antes e depois da refatoração realizada

Já o *Indecent Exposure* ou *Inappropriate Intimacy*, foi encontrado em 8 ocorrências no projeto. O *code smell* em questão, trata-se de uma exposição incorreta e desnecessária de atributos, campos ou métodos internos de uma classe, sendo necessário um bom motivo para esses itens serem públicos. Que pode ocasionar em uma maior desorganização dos códigos, e dificultando o suporte e a reutilização dos mesmos (KERIEVSKY, 2005).

Verificando os casos detectados, foi possível examinar que os erros encontrados foram nos modificadores de acesso escolhidos incorretamente em algumas classes, e métodos públicos que só são utilizados internamente nas suas próprias classes. Encontrando essas classes e métodos, o Resharper sugere como solução a troca dos modificadores de acesso para *private* ou *protected*. O código original e o código refatorado pode ser visto na Figura 5.

Figura 5 – Exemplo de *Indecent Exposure* antes e depois da refatoração realizada



O *Long Method*, foi detectado em 12 métodos presentes nos códigos da aplicação. Este tipo de *code smell* refere-se a um método que contém muitas linhas de códigos, podendo ocasionar geralmente em métodos com duplicação de códigos indesejados, e no aumento da dificuldade para entender e manter o mesmo (FOWLER, 1999).

Os *Long Methods* encontrados pelo SonarQube, são métodos que se estendem pois possuem muitas condições aninhadas e inseridas de forma desnecessária. A aplicação utilizada para refatoração, sugere como solução a mescla das condições em um mesmo *if*, de forma que combine as condições possíveis, mas mantendo a mesma lógica que já estava sendo utilizada no método e remoção das validações desnecessárias. A refatoração de um dos casos detectados, pode ser vista na Figura 6 abaixo.

Figura 6 – Exemplo de um trecho de *Long Method* antes e depois da refatoração realizada

```
var section = context.ListSections.FirstOrDefault(p :Section => p.Id == sectionId);
if (section.HasntValue())
    return listPosts;

var qry = (IQueryable<Post>)context.Posts // DbSet<Post>
    .Include(p :Post => p.PostCategory)
    .Include(p :Post => p.PostSubCategory)
    .Include(p :Post => p.Author)
    .Include(p :Post => p.ListPostRating); // IQueryable<Post>

qry = qry.OrderByDescending(p :Post => p.PublishDate);

switch (section.SearchType)
{
    case ESearchType.StartsWith:
        if (section.SecondComparative != null)
            qry = qry.Where(p :Post => p.Title.StartsWith(section.Comparative) || p.Title.
        else
            qry = qry.Where(p :Post => p.Title.StartsWith(section.Comparative));
        break;

    case ESearchType.Contains:
        if (section.SecondComparative != null)
            qry = qry.Where(p :Post => p.Title.Contains(section.Comparative) || p.Title.Co
        else
            qry = qry.Where(p :Post => p.Title.Contains(section.Comparative));
        break;

    case ESearchType.EndsWith:
        if (section.SecondComparative != null)
            qry = qry.Where(p => p.Title.EndsWith(section.Comparative) || p.Title.EndsWi
        else
            qry = qry.Where(p => p.Title.EndsWith(section.Comparative));
        break;
}

if (lastPostId.HasValue && lastPostId > 0)
    qry = qry.Where(p => p.Id < lastPostId);
```

```
var section = context.ListSections.FirstOrDefault(p :Section => p.Id == section

var qry = (IQueryable<Post>)context.Posts // DbSet<Post>
    .Include(p :Post => p.PostCategory)
    .Include(p :Post => p.PostSubCategory)
    .Include(p :Post => p.Author)
    .Include(p :Post => p.ListPostRating); // IQueryable<Post>

qry = qry.OrderByDescending(p :Post => p.PublishDate);

switch (section.SearchType)
{
    case ESearchType.StartsWith:
        qry = section.SecondComparative != null ? qry.Where(p :Post => p.Title
        break;

    case ESearchType.Contains:
        qry = section.SecondComparative != null ? qry.Where(p :Post => p.Title
        break;

    case ESearchType.EndsWith:
        qry = section.SecondComparative != null ? qry.Where(p :Post => p.Title
        break;
}

if (lastPostId > 0)
    qry = qry.Where(p :Post => p.Id < lastPostId);
```

No projeto analisado, foi encontrado somente uma ocorrência de *code smell* do tipo *Temporary Field*. O *Temporary Field* trata-se de um objeto que contém muitos campos opcionais ou que não estão sendo utilizados, dificultando muito seu entendimento pois espera-se que estes campos possua dados mas quase sempre estão vazios, perdendo a clareza e organização do código (FOWLER, 1999).

Verificando o código do erro detectado pela análise, é encontrado um objeto que possui várias propriedades que são opcionais ou não estão sendo utilizadas em lugar algum do código. Para a refatoração, o Resharper sugere remover os campos não utilizados, gerar uma nova classe para extrair as propriedades temporárias, ou passar os campos por parâmetros dos métodos em que são utilizados. Estes *Temporary Fields* e sua refatoração pode ser visualizada na Figura 7.

Figura 7 – Exemplo de um objeto com *Temporary Field* antes e depois da refatoração realizada

```
public class PostCardResponse
{
    23 referências
    public int Id { get; set; }
    1 referência
    public int? RecommendationId { get; set; }
    1 referência
    public string Recommendation { get; set; }
    1 referência
    public int? FavoriteId { get; set; }
    15 referências
    public string Title { get; set; }
    18 referências
    public DateTime PublishDate { get; set; }
    13 referências
    public string Category { get; set; }
    20 referências
    public int? SubCategoryId { get; set; }
    13 referências
    public string SubCategory { get; set; }
    6 referências
    public decimal? Rating { get; set; }
    4 referências
    public bool? CanView { get; set; }
    19 referências
    public bool? Exclusive { get; set; }
    15 referências
    public string Description { get; set; }
    13 referências
    public int? ViewsCount { get; set; }
    13 referências
    public int? SharedCount { get; set; }
    15 referências
    public string Author { get; set; }
    12 referências
    public int? CommentsCount { get; set; }
    19 referências
    public int? CategoryId { get; set; }
    2 referências
    public string Content { get; set; }
    5 referências
    public string Image { get; set; }
    2 referências
    public bool? ShowStamp { get; set; }
    1 referência
    public decimal? UserRating { get; set; }
    1 referência
    public bool? IsFavorite { get; set; }
    0 referências
    public bool? IsSubCategoryFavorite { get; set;
}
```

```
75 referências
public class PostCardResponse
{
    22 referências
    public int Id { get; set; }
    15 referências
    public string Title { get; set; }
    17 referências
    public DateTime PublishDate { get; set; }
    15 referências
    public string Description { get; set; }
    15 referências
    public string Author { get; set; }
    2 referências
    public string Content { get; set; }
    5 referências
    public string Image { get; set; }
}
```

Já o *Object-Orientation Abusers* está presente em 4 trechos nos códigos do projeto, estes tipo de *code smell* trata-se de códigos que não seguem os princípios da programação orientada a objetos, praticando o uso incorreto dos conceitos de heranças, polimorfismos ou encapsulamentos. Podendo gerar problemas no código, diminuindo a facilidade de manutenção e a reutilização do mesmo (LANZA; MARINESCU, 2010).

Analisando os trechos de códigos identificados como *code smell* do tipo *OOP Abusers*, é possível identificar um problema de encapsulamento de campos públicos dentro das classes, ao qual é alertado pelo Resharper. Como solução de refatoração, a aplicação sugere tornar os campos privados e encapsulá-los em propriedades públicas, para seguir as regras de encapsulamento e garantir que os dados não sofram acessos indevidos. A Figura 8 abaixo, apresenta um caso de *OOP Abuser* no código original do projeto.

Figura 8 – Exemplo de *Object-Orientation Abusers* antes e depois da refatoração realizada

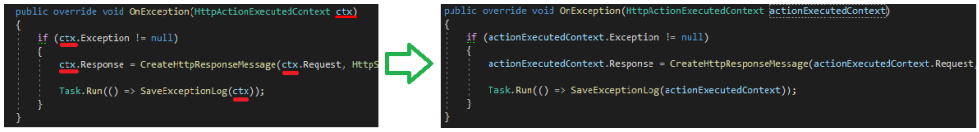
```
1 referência
public static List<City> CacheListCities
{
    get
    {
        var listCities = CacheHelper.GetCachedValue<List<City>>(key: "ListCities");
        if (listCities == null)
        {
            using (var context = EfHelper.DbContextFactory<AgrinvestEntities>())
            {
                listCities = context.ListCities.OrderBy(x:City => x.Name).ToList();
                CacheHelper.SetChachedValue("ListCities", listCities);
            }
        }
        return listCities;
    }
}
#endregion
```

```
public static List<City> ListCities => CacheListCities;
1 referência
private static List<City> CacheListCities
{
    get
    {
        var listCities = CacheHelper.GetCachedValue<List<City>>(key: "ListCities");
        if (listCities == null)
        {
            using (var context = EfHelper.DbContextFactory<AgrinvestEntities>())
            {
                listCities = context.ListCities.OrderBy(x:City => x.Name).ToList();
                CacheHelper.SetChachedValue("ListCities", listCities);
            }
        }
        return listCities;
    }
}
```

E os dados demonstram também a detecção de *code smells* do tipo *Uncommunicative Name* (6), que são variáveis, métodos ou classes que foram nomeados de forma não comunicativa ou esclarecedora. O nome dos itens citados ao serem criados, devem descrever sucintamente sua função no código ou o que significam, caso ao contrário irá dificultar muito a legibilidade e a manutenção do código (FOWLER, 1999).

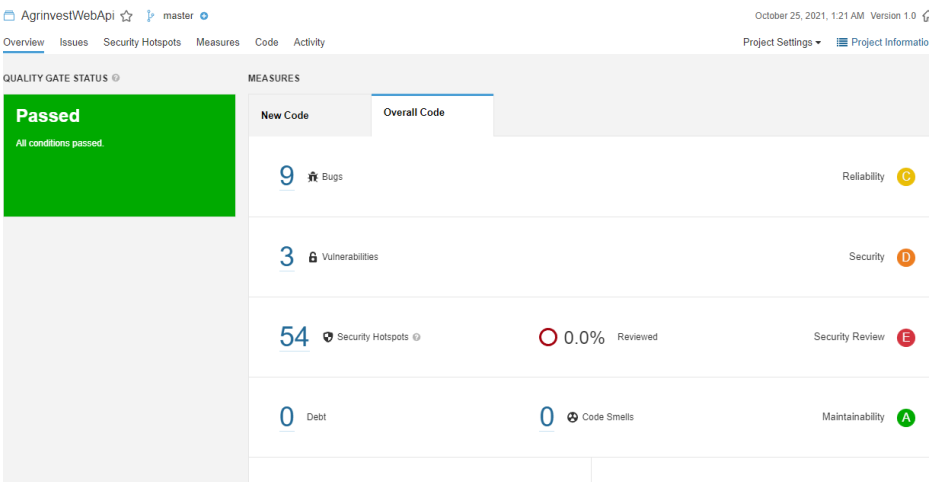
Verificando os seis casos encontrados, é possível ver que cinco são variáveis que são passadas por parâmetros em métodos, sendo o último caso o nome de uma classe. A ferramenta de refatoração sugere e oferece opções para a renomeação dos itens em questão, para que não seja utilizado abreviaturas e correspondam a sua classe base, declaração da interface ou no caso da classe que termine com o tipo ao qual está sendo herdado. Um destes exemplos de *Uncommunicative Name* pode ser visto na Figura 9 abaixo, junto a sua refatoração.

Figura 9 – Exemplo de *Uncommunicative Name* antes e depois da refatoração realizada



Por fim, após a realização da refatoração de todos os casos detectados, por meio da utilização do Resharper e seguindo as métricas do *code smell*. Foi realizado outra inspeção completa do projeto por meio do SonarQube, obtendo como resultado nenhum *code smell* detectado no código-fonte da aplicação. Na Figura 10, é possível ver os dados da análise feita pela ferramenta e os resultados obtidos.

Figura 10 – Dados da análise do SonarQube após a refatoração feita



5 Conclusão

Sabendo que os *code smells* são indícios de que o código-fonte pode ser melhorado, e que futuramente podem representar problemas a aplicação se não tratados a tempo, levando então a necessidade da refatoração do código (FOWLER, 1999). Neste trabalho, foi conduzido um estudo de caso sobre *code smells* em códigos de linguagem de programação C#, realizando uma análise de vulnerabilidades e a refatoração dos mesmos. Com intuito de remover as vulnerabilidades, e aprimorar a qualidade do código final do projeto utilizado como objeto de estudo para a realização do trabalho.

Para avaliar a presença dos *code smells* no projeto em questão, foi utilizado a ferramenta SonarQube, realizando uma análise completa de seu código-fonte. Tendo como resultado um total de 61 *code smells* detectados nos códigos da aplicação, e por meio da classificação realizada é possível perceber uma predominância considerável do *Dead Code*.

Foram apresentadas todas as vulnerabilidades e impactos negativos que cada tipo desses *code smells* mapeados, estão causando ou poderiam causar futuramente a aplicação. Com o auxílio da ferramenta Resharper, foi feito a refatoração desses *code smells* detectados, removendo os possíveis problemas e apresentando alguns exemplos de códigos antes e depois da refatoração feita. Por fim, foi realizado novamente uma inspeçaõ completa do projeto com o SonarQube, obtendo como resultado nenhum *code smell* detectado no código-fonte do *software*.

Sendo assim, foi alcançado como resultado final no projeto, uma melhoria geral na qualidade do código-fonte seguindo as métricas do *code smell*, sendo elas: diminuição das vulnerabilidades; melhoria na organização dos códigos; remoção de códigos desnecessários; facilitamento do entendimento do código e sua manutenção; e o beneficiamento na reusabilidade.

Referências

BARRA, Daniela Couto Carvalho; NASCIMENTO, Eliane Regina Pereira do; JESUS MARTINS, Josiane de; ALBUQUERQUE, Gelson Luiz; ERDMANN, Alacoque Lorenzini. Evolução histórica e impacto da tecnologia na área da saúde e da enfermagem. *Revista Eletrônica de Enfermagem*, v. 8, n. 3, 2006.

FLAUZINO, Matheus Antonio; CARVALHO, Eduardo Gomes; FERNANDES, Hebert Rausch; SILVA, Lázaro Eduardo; LAGO, Daniel Guimarães. Pull Request e Code Smells: uma análise em aplicações móveis. *Brazilian Journal of Development*, v. 7, n. 8, p. 80542–80550, 2021.

FOWLER, Martin. *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley, 1999. ISBN 0-201-48567-2.

FREITAS, Mydiã Falcão. *Investigando a relação entre code smells e problemas de design: estudos qualitativos em microempresas de software*. 2017. Diss. (Mestrado) – Universidade Federal da Bahia, Instituto de Matemática, Brasil.

GOMES, Felipe Gustavo de S; MENDES, Thiago Souto; SPINOLA, Rodrigo O; MENDONÇA, Manoel; FARIAS, Mário. Uma análise da relação entre code smells e dıvida técnica auto-admitida. In: SBC. ANAIS do VII Workshop on Software Visualization, Evolution and Maintenance (VEM). [S.l.: s.n.], 2019. P. 37–44.

JETBRAINS. *Recursos do ReSharper*. [S.l.]: JetBrains. `https://www.jetbrains.com/pt-br/resharper/features/`. Acesso em: 21 set. 2021.

JUNIOR, Anderson Hilario; FARAH, Paulo Roberto. Avaliação de Code Smells e Padrões de Projeto em Aplicações com Angular e React. In: SBC. ANAIS da III Escola Regional de Engenharia de Software. [S.l.: s.n.], 2019. P. 81–88.

KERIEVSKY, Joshua. Smells to Refactorings Cheatsheet. Industrial Logic, 2005.

LANZA, Michele; MARINESCU, Radu. *Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems*. 1st. [S.l.]: Springer Publishing Company, Incorporated, 2010. ISBN 3642063748.

MACIEL, Igor; JÚNIOR, Methanias Colaço; MENDONÇA, Manoel; CARNEIRO, Glauco. Um Experimento na Indústria para Mineração Visual de Padrões de Interações de Programadores. Universidade Federal de Sergipe (UFS), 2012.

MEDINA, Marco; FERTING, Cristina. *Algoritmos e programação: teoria e prática*. [S.l.]: Novatec Editora, 2006.

PÁDUA PAULA FILHO, Wilson de. *Engenharia de software*. [S.l.]: LTC, 2003. v. 2.

REGIS, Luiz Felipe Muniz; OLIVEIRA, Roberto Felıcio. *A arte da pesquisa bibliográfica na busca do conhecimento sobre a adoção de boas paáticas no desenvolvimento de software*. 2018. Diss. (Mestrado) – Universidade Estadual de Goiás (UEG) - Câmpus Posse, Brasil.

SONARSOURCE, SA. *Documentação SonarQube*. [S.l.]: SonarQube. `https://docs.sonarqube.org/latest/`. Acesso em: 21 set. 2021.