Detecção de *Code Smells* em ambientes de nuvem

João Victor Silva de Amorim [1]*, Erica Teixeira Gomes de Souza[2]
joao.silvaamorim@ufrpe.br[1]*, erica.sousa@ufrpe.br[2] Universidade Federal Rural de Pernambuco (UFRPE), Brasil * autor correspondente

Resumo
Code smells são características do código que indicam possíveis problemas no design estrutural do software. Embora essas questões não causem falhas imediatas na funcionalidade do sistema, podem comprometer aspectos cruciais, como desempenho, escalabilidade, manutenibilidade e confiabilidade, especialmente em projetos de grande escala. Este trabalho apresenta uma ferramenta desenvolvida para detectar code smells em plataformas de computação em nuvem. A ferramenta foi projetada para identificar code smells relacionados a código repetitivo, código morto e funções contendo múltiplas instruções de retorno. A ferramenta proposta detectou nove tipos de code smells, analisando mais de 1.200 arquivos de código em Python e Java. No OpenStack, o código repetitivo foi o problema mais prevalente, representando 65,7% dos casos, enquanto no CloudStack, classes e métodos longos foram os mais recorrentes, totalizando 39,5% do total. Além disso, o code smell proposto, número mágico, se destacou ao representar 6,1% do total de code smells no CloudStack e 8,4% no OpenStack, evidenciando sua relevância para a legibilidade e manutenibilidade do código. A análise da detecção de code smells demonstra a importância da ferramenta *NimbusScan*, pois não apenas acelera o processo de revisão de código, mas também promove a modularidade e a clareza necessárias em sistemas complexos e dinâmicos. Essa análise conclui que soluções como a apresentada são essenciais para o desenvolvimento de software robusto e eficiente, capaz de atender às crescentes demandas do mercado de computação em nuvem, desempenhando um papel fundamental na melhoria contínua da qualidade do código e na longevidade dos sistemas.
Palavras-chave: análise estatística, cheiro de código, computação em nuvem, qualidade de software, refatoração de código.

Detection of Code Smells in Cloud Environments

Abstract
Code smells are code characteristics that indicate potential problems in the structural design of software. Although these issues do not cause immediate failures in system functionality, they can compromise crucial aspects such as performance, scalability, maintainability, and reliability, especially in large-scale projects. This work presents a tool developed for detecting code smells in cloud computing platforms. The tool was designed to identify code smells related to repetitive code, dead code, and functions containing multiple return statements. The proposed tool detected nine types of code smells, analyzing more than 1,200 code files in Python and Java. In OpenStack, repetitive code was the most prevalent issue, accounting for 65.7% of cases, while in CloudStack, long classes and methods were the most recurrent, making up 39.5% of the total. Additionally, the proposed code smell, magic number, represented 6.1% of the total code smells in CloudStack and 8.4% in OpenStack, highlighting its relevance to code readability and maintainability. The code smell detection analysis demonstrates the relevance of the Nimbus Scan tool, as it not only speeds up the code review process but also promotes the modularity and clarity required in complex and dynamic systems. This analysis concludes that solutions like the one presented are essential for developing robust and efficient software capable of meeting the growing demands of the cloud computing market, playing a fundamental role in continuously improving code quality and system longevity.
Keywords: *statistical analysis, code smells, cloud computing, software quality, code refactoring.*

1 Introdução

No desenvolvimento de *software*, simplicidade, clareza e facilidade de modificação do código são elementos fundamentais para garantir sua qualidade e sustentabilidade a longo prazo. Um código bem estruturado não apenas facilita a compreensão e a colaboração entre os membros da equipe, mas também permite que futuras manutenções e evoluções sejam realizadas de maneira eficiente. Contudo, o cenário real de desenvolvimento raramente oferece condições ideais.

A pressão por entregas rápidas, prazos curtos e cronogramas rigorosos frequentemente impõem desafios que dificultam a manutenção de padrões elevados de codificação. Nessas circunstâncias, os desenvolvedores, mesmo que involuntariamente, acabam introduzindo *code smells*, que são padrões no código que indicam potenciais problemas estruturais e de manutenção do *software* (Sommerville, 2019).

Embora não representem erros ou falhas imediatas, sua presença sugere um afastamento das boas práticas de design e desenvolvimento, tornando o código mais difícil de entender, modificar e expandir. Esses sinais podem surgir de diversas formas, como código excessivamente complexo, com funções e classes muito grandes que dificultam a leitura; duplicação de código, que aumenta o esforço de manutenção; má organização de classes e métodos, onde a falta de modularização compromete a reutilização; e uso inadequado de valores numéricos fixos que reduzem a clareza do código. Com o tempo, esses problemas podem se acumular, impactando a eficiência da manutenção e aumentando o risco de introdução de novos *code smells* a cada modificação. Assim, a identificação e a correção precoce de code smells são cruciais para assegurar que o software permaneça robusto, flexível e sustentável ao longo do seu ciclo de vida (Fowler, 2018).

Embora os *code smells* não sejam necessariamente prejudiciais, indicam possíveis desvios das boas práticas e princípios de design (Fowler, 2018). Eles não surgem instantaneamente, mas acumulam-se à medida que o software evolui. A identificação desses problemas pode ser complexa e depende diretamente da interpretação, do conhecimento e da habilidade do desenvolvedor. Ferramentas específicas podem auxiliar nesse processo, permitindo a identificação automatizada de *code smells* (Fowler, 2018). A remoção desses problemas deve ocorrer o mais cedo possível no ciclo de desenvolvimento do software, pois, se negligenciados, podem propagar-se e gerar problemas mais graves conhecidos como clusters de code smells (Thamid, 2016).

Os *code smell*s estão frequentemente associados a uma maior incidência de bugs, o que reduz a confiabilidade e a qualidade do software (Fowler, 2018). Assim, sua identificação e remoção constituem uma estratégia eficaz para prevenir falhas. Em ambientes de computação em nuvem, a natureza dinâmica e a rápida provisão de recursos tornam esses ambientes ideais para a execução de serviços distribuídos e aplicações que demandam alta capacidade de processamento. No entanto, a presença de code smells no código-fonte pode impactar negativamente o consumo de recursos nesses contextos (Imran, 2023). Estudos recentes têm explorado a relação entre code smells e o uso de recursos em nuvem, mas prever alterações no consumo devido à refatoração permanece um desafio (Imran, 2023).

A computação em nuvem caracteriza-se pela provisão dinâmica de recursos a partir de um pool compartilhado, alocados conforme a demanda dos usuários. Essa capacidade de rápida provisão tornou a nuvem uma plataforma atrativa para diversas aplicações, tanto comerciais quanto de código aberto. Organizações têm incorporado cada vez mais plataformas de nuvem de código aberto em sua infraestrutura de TI, buscando benefícios econômicos e suporte a aplicações complexas (Shah et al., 2023). Contudo, a presença de code smells nesses contextos representa um desafio para a manutenção e aceitação dessas plataformas. A refatoração de code smells consiste na remoção desses problemas por meio da reestruturação do código, mantendo sua funcionalidade. Ferramentas para detecção de code smells, aliadas a abordagens semânticas e à análise de *feedback* de usuários, podem ser empregadas nesse processo (Shah et al., 2023).

O objetivo deste artigo é investigar o impacto dos *code smells* no desenvolvimento de software, com foco na análise de plataformas de computação em nuvem, especificamente *OpenStack* e *CloudStack*. A pesquisa busca explorar como esses padrões problemáticos podem comprometer a qualidade e a manutenção do código em ambientes de nuvem, além de explorar detecção de code smells nas linguagens como *Python* e *Java*. O estudo visa analisar a presença desses problemas nessas plataformas e a importância da refatoração precoce para garantir o desempenho e a sustentabilidade do *software* a longo prazo.

Este artigo está organizado da seguinte forma: na Seção 2, são apresentados os conceitos básicos, incluindo definições de code smells, computação em nuvem e refatoração. A Seção 3 descreve a ferramenta proposta, detalhando sua estrutura, funcionalidades e os tipos de code smells detectados. Na Seção 4, são analisados os resultados obtidos com a ferramenta, destacando a detecção

de code smells em projetos desenvolvidos nas linguagens Python e Java. A Seção 5 discute os resultados, evidenciando as diferenças entre as plataformas OpenStack e CloudStack e o impacto das características das linguagens no desenvolvimento de software. Por fim, a Seção 6 conclui o artigo, ressaltando as contribuições do trabalho e propondo direções para pesquisas futuras.

2 Conceitos Básicos

Esta seção apresenta os fundamentos teóricos necessários para compreender o trabalho.

2.1 Computação em Nuvem

A computação em nuvem é um modelo que permite acesso conveniente e sob demanda a um conjunto compartilhado de recursos computacionais configuráveis, como redes, servidores, armazenamento, aplicativos e serviços. Esses recursos podem ser rapidamente provisionados e liberados com o mínimo de esforço de gerenciamento ou interação com o provedor do serviço, conforme a definição amplamente aceita pelo Instituto Nacional de Padrões e Tecnologia dos EUA (NIST) (Shah et al., 2023).

Entre as principais plataformas de computação em nuvem de código aberto estão o *OpenStack* e o *CloudStack*, amplamente utilizadas para gerenciar infraestruturas de nuvem em ambientes públicos, privados e híbridos. Ambas possibilitam que organizações criem e administrem recursos computacionais, como máquinas virtuais, redes e armazenamento, com alta eficiência e escalabilidade, embora suas abordagens e características apresentem diferenças significativas. O *OpenStack* é uma plataforma modular e robusta, projetada para gerenciar grandes *pools* de recursos computacionais em data centers. Desenvolvido inicialmente pela *NASA* e pela *Rackspace*, o *OpenStack* conta atualmente com uma vasta comunidade de desenvolvedores e empresas que colaboram para sua evolução contínua. A plataforma é composta por componentes que desempenham papéis específicos, como o gerenciamento de máquinas virtuais, o armazenamento de objetos, o armazenamento em blocos, as redes virtuais e a interface gráfica para administração. Essa modularidade confere ao *OpenStack* uma alta personalização, tornando-o particularmente popular em grandes empresas e ambientes que exigem flexibilidade e controle avançados (SHAH et al., 2023).

Por sua vez, o *CloudStack* é reconhecido por sua simplicidade e facilidade de uso, sendo ideal para organizações que desejam implementar uma infraestrutura de nuvem de forma rápida e eficiente. Desenvolvido originalmente pela *Cloud.com* e posteriormente adquirido pela *Citrix*, o *CloudStack* é compatível com diversos hipervisores, como *XenServer*, *VMware* e *KVM*. Ele oferece uma experiência consistente em diferentes ambientes e apresenta uma interface de usuário amigável e arquitetura prática, que facilitam a automação e a escalabilidade dos recursos computacionais sem a complexidade associada à configuração do *OpenStack*. Além disso, o CloudStack suporta uma ampla variedade de funcionalidades avançadas, como balanceamento de carga, firewalls e redes definidas por software (SDN), o que permite às organizações atender a demandas específicas de seus ambientes com maior eficiência e flexibilidade (Reynolds, 2014).

2.2 Code Smells

Code smells são considerados indícios de potenciais problemas no código-fonte de um *software*. Embora não causem erros imediatos ou comprometimentos funcionais diretos, podem degradar a qualidade estrutural do sistema ao longo do tempo. Eles funcionam como sinais de alerta que, se ignorados, podem gerar dificuldades futuras, principalmente relacionadas à manutenção, evolução e escalabilidade do *software*.

De acordo com Fowler (2018), code smells não são necessariamente bugs, mas padrões de má prática que afetam negativamente a legibilidade, clareza e modularidade do código. Essas características tornam o código mais propenso a erros futuros e dificultam o trabalho dos desenvolvedores na sua evolução. O conceito foi introduzido por Beck e Fowler (1999) e, desde então, tem sido amplamente estudado, resultando no desenvolvimento de diversas técnicas e ferramentas para sua detecção.

Os *code smells* podem manifestar-se de diferentes formas, dependendo de como o software foi projetado e da linguagem de programação utilizada. Exemplos comuns incluem código duplicado,

métodos excessivamente longos, classes extensas, condicionais complexas e o uso de números mágicos — valores literais inseridos diretamente no código sem uma explicação clara de seu significado (Fowler, 2019). Essas práticas indicam que o design do código pode ter sido inadequado, comprometendo sua manutenção a longo prazo.

Shah et al. (2023) ressalta que, em ambientes de computação em nuvem, esses problemas tornam-se ainda mais críticos. Plataformas como *OpenStack* e *CloudStack*, que lidam com sistemas de grande escala e alta complexidade, são especialmente suscetíveis aos impactos negativos dos code smells. A presença desses problemas em componentes de infraestrutura de nuvem pode resultar em aumento no consumo de recursos e na probabilidade de erros em ambientes de produção.

Uma característica importante dos code smells é sua natureza subjetiva e contextual. O que pode ser considerado um code smell em determinado contexto pode não ser em outro, dependendo da arquitetura do sistema e das práticas de desenvolvimento adotadas pela equipe. Por exemplo, classes longas são frequentemente vistas como code smells em sistemas orientados a objetos que priorizam modularidade e separação de responsabilidades. No entanto, em cenários específicos, pode ser preferível manter uma classe com múltiplas responsabilidades para evitar a fragmentação excessiva do código (Mens et al., 2004).

Essa subjetividade torna a detecção e a correção de *code smells* um processo complexo, que exige não apenas o uso de ferramentas automatizadas, mas também o julgamento e a experiência dos desenvolvedores. Fowler (2019) argumenta que a identificação de code smells é um passo essencial no processo de refatoração, cujo objetivo é melhorar o design do *software* sem alterar seu comportamento externo.

A refatoração é uma prática fundamental para garantir a qualidade do código ao longo do ciclo de vida do *software*, tornando-o mais claro, legível e sustentável. Durante esse processo, os desenvolvedores podem eliminar duplicações de lógica, aumentar a coesão de classes e funções e reduzir a complexidade de estruturas de controle. Esses aprimoramentos diminuem a probabilidade de erros e facilitam a incorporação de novas funcionalidades no futuro.

Quando o código é mal projetado ou adota práticas inadequadas, o desempenho do sistema pode ser comprometido por ineficiências na execução de operações ou pelo uso excessivo de recursos (Mens et al., 2004). Em plataformas de nuvem, onde a eficiência no uso de recursos é fundamental para reduzir custos e garantir escalabilidade, a eliminação de code smells torna-se ainda mais crucial.

Shah et al. (2023) destacam que em sistemas de grande escala a refatoração contínua do código é essencial para assegurar que o desempenho não seja prejudicado por práticas de codificação inadequadas. Por isso, a análise e a detecção de code smells configuram-se como práticas indispensáveis para a manutenção da qualidade do *software*.

Embora a identificação de code smells possa ser parcialmente automatizada, ela também exige uma análise crítica do código por parte dos desenvolvedores, que precisam considerar o contexto específico do sistema e suas demandas de manutenção a longo prazo. A combinação do uso de ferramentas adequadas com boas práticas de refatoração contribui para que o *software* se mantenha escalável, eficiente e fácil de manter, independentemente de sua complexidade ou da plataforma em que opera.

3 Trabalhos relacionados

Nesta seção, são apresentados trabalhos relacionados ao tema deste artigo, que abordam a detecção e mitigação de code smells em diferentes contextos, incluindo ambientes de nuvem, microsserviços e software tradicional.

Shah et al. (2023) propõem uma ferramenta para a detecção de code smells em ambientes de nuvem de código aberto, como o OpenStack. A ferramenta identifica oito tipos de code smells relevantes nesse contexto, utilizando uma abordagem baseada em regras adaptadas para códigos de nuvem. Sua eficácia foi comprovada por meio de avaliações práticas no OpenStack, demonstrando sua capacidade de melhorar a manutenibilidade do software. Além disso, o estudo sugere direções para futuras pesquisas, como a inclusão de refatorações automatizadas e a análise aprofundada das causas dos code smells, consolidando sua contribuição para a qualidade do software em nuvem.

Kumara (2020) apresenta uma abordagem baseada em conhecimento para a detecção semântica de code smells em códigos de infraestrutura de nuvem. O estudo desenvolve uma ontologia de desdobramento de aplicações em nuvem, fundamentada no padrão TOSCA, e define regras semânticas em SPARQL para identificar problemas de qualidade no código. Essa abordagem, que gera grafos de conhecimento RDF, foi validada por meio de protótipos e estudos de caso, destacando-se como uma ferramenta eficaz para aprimorar a qualidade do código em nuvem e trazendo maior consistência à análise semântica.

Fowler (2019) explora a detecção do *Temporary Field*, um *code smell* pouco estudado até então. O estudo propõe um método em três etapas: cálculo de métricas específicas, aplicação da refatoração avaliação da manutenibilidade do código após a refatoração. Os resultados mostram melhorias significativas na legibilidade e na manutenibilidade do código, apresentando uma abordagem sistemática para lidar com o *Temporary Field* e contribuindo para a qualidade geral do software.

Tahmid et al. (2016) investigam a evolução dos *code smells* em software, analisando suas inter-relações ao longo do tempo. A pesquisa utiliza o JUnit como estudo de caso, identificando clusters de code smells por meio da detecção de classes e métodos smelly, extração de suas relações e geração de gráficos representativos. Os resultados revelam que o número de clusters aumenta consistentemente entre versões, com uma média de 3,78 clusters por versão, sendo que o mega cluster representa 69,49% dos nós smelly. O estudo oferece insights valiosos sobre os desafios de manutenção e qualidade do software, além de prover ideias para novas pesquisas e a análise das dinâmicas de evolução dos code smells em diferentes contextos de software.

Walker et al. (2020) contribuem para a qualidade do desenvolvimento em microsserviços com uma ferramenta de código aberto para a detecção automatizada de code smells nesse ambiente. Essa ferramenta pode ser integrada ao ciclo de vida de desenvolvimento de software, permitindo a execução de testes de triagem automática antes da implantação e acelerando o processo de revisão de código. Como resultado, o esforço manual e os erros humanos são reduzidos, melhorando a qualidade do código e diminuindo o ciclo de lançamento de atualizações. O estudo também propõe uma abordagem para identificar microsserviços mal projetados, utilizando análises de similaridade de nomes e padrões de design, além de sugerir a ampliação da detecção para outras linguagens e padrões, aumentando a aplicabilidade da ferramenta.

Esses trabalhos relacionados evidenciam a crescente atenção da comunidade científica para a detecção e mitigação de code smells em diferentes contextos, incluindo nuvem, microsserviços e software tradicional. As contribuições destacam avanços tecnológicos, como ferramentas automatizadas e abordagens baseadas em conhecimento, que podem ser diretamente aplicáveis a projetos como o desenvolvido neste artigo. Juntos, fornecem uma base sólida para novas pesquisas e reforçam a importância de soluções práticas para melhorar a qualidade do código e facilitar a manutenção de sistemas complexos.

A Tabela 1 apresenta uma comparação entre (SHAH, 2023) e o *NimbusScan.*

Tabela 1 - Code smells detectados

Trabalho	**Tipo de abordagem**	**Aplicação na nuvem**	**OpenStack/ CloudStack**	**Python/ Java**	**Número mágico**
(SHAH, 2023)	Semântica	Sim	Não	Não	Não
NimbusScan	Semântica	Sim	Sim	Sim	Sim

Fonte: Elaborado pelo autor, 2025.

Em conclusão, a Tabela 1 ilustra a comparação entre diferentes abordagens para a detecção de code smells, com foco na aplicação em nuvem e suporte a plataformas específicas como OpenStack e CloudStack, além das linguagens Python e Java. A abordagem semântica adotada por SHAH (2023) não se aplica diretamente a essas plataformas ou linguagens, o que limita sua flexibilidade. Em contrapartida, o NimbusScan, também baseado em uma abordagem semântica, oferece suporte tanto

para as plataformas de nuvem OpenStack e CloudStack quanto para as linguagens Python e Java, tornando-se uma ferramenta mais versátil para a detecção de code smells.

4 NimbusScan

A ferramenta proposta, denominada NimbusScan, representa uma extensão do modelo original desenvolvido por (Shah et al, 2023), conhecido como *CloudScent*. As adaptações realizadas nos scripts de detecção visam ampliar a gama de *code smells* identificáveis e expandir o escopo semântico para abranger a linguagem de programação Java. Isso possibilita a aplicação do modelo de detecção em outras plataformas de nuvem, além das abordadas originalmente.

O *NimbusScan* pode identificar 9 code smells em plataformas de nuvem como *OpenStack* e *CloudStack*, desenvolvidas nas linguagens Python e Java, respectivamente. O trabalho de (Shah et al, 2023) descreve um conjunto de 8 code smells detectáveis; entretanto, este estudo propõe a inclusão de um novo tipo, o número mágico. A Tabela 1 apresenta os 9 code smells que podem ser detectados pela ferramenta *NimbusScan*.

Tabela 2 - Code smells detectados

Code Smell	**Descrição**
Linhas de Código Repetidas	Identificar e reportar todas as ocorrências de código repetitivo.
Código Morto	Detectar e relatar trechos de código morto após instruções return
Múltiplos Retornos	Listar funções com múltiplas instruções de retorno
Instruções Longas	Apontar declarações que excedem o limite recomendado de tamanho
Nomes de Funções Iguais	Detectar e relatar conflitos causados por funções com o mesmo nome
Classe/Método Longo	Indicar classes ou métodos que ultrapassam o tamanho recomendado
Condições/Laços Longos	Reportar blocos de loop considerados muito longos e blocos condicionais que excedem o tamanho esperado.
Parâmetros Longos	Apontar funções com listas de parâmetros excessivamente longas
Número Mágico	Detectar e sugerir substituição de números mágicos por constantes

Fonte: Elaborado pelo autor, 2025.

4.1 Estrutura e Funções das Classes

A estrutura do *NimbusScan* segue uma sequência de etapas lógicas para analisar os arquivos de código e identificar *code smells*. O processo tem início com a definição de uma estrutura inicial, chamada *codeSmellsStatistic*, que armazena as contagens iniciais para cada tipo de code smell a ser analisado. Em seguida, são configuradas variáveis e expressões regulares específicas para a detecção dos diferentes problemas.

Cada code smell é identificado por meio de uma função dedicada que é chamada durante a análise de cada linha do código. Essas funções são responsáveis por verificar padrões específicos e registrar as ocorrências de cada tipo de code smell, com base nas expressões regulares e nas regras definidas.

4.2 O número mágico

A função *checkMagicNumbers* visa identificar números mágicos no código, ou seja, valores numéricos que aparecem diretamente no código-fonte sem uma explicação clara de seu propósito. Esses números podem ser confusos e difíceis de entender, além de tornarem o código mais difícil de

manter, pois não há um contexto ou nome associado ao número. Como alternativa, é uma boa prática substituir números mágicos por constantes nomeadas, tornando o código mais legível e fácil de modificar.

O funcionamento da função para detecção do número mágico pode ser descrita da seguinte maneira:

Análise de linhas de código: A função percorre o código linha por linha e verifica se a linha atual contém uma estrutura condicional (*if, else if, else, while, for*). Para isso, a função busca a presença dessas palavras-chave, que indicam que a linha pertence a um bloco de controle de fluxo.

Verificação de números mágicos: Após identificar que a linha está em um bloco condicional ou de *loop*, a função usa uma expressão regular para procurar números mágicos. A expressão regular busca números que aparecem isoladamente (por exemplo, "5", "100") ou em operações lógicas (por exemplo, "5 < 10").

Registro de code smell: Se a expressão regular encontrar números ou expressões que correspondem a números mágicos, a função chama a função *describeCodeSmell* para registrar a ocorrência do code smell. A função registra a linha onde o número foi encontrado, e dependendo do contexto (se é um número isolado ou uma operação), pode fazer uma distinção entre números encontrados diretamente ou em operadores lógicos.

Em resumo, a função *checkMagicNumbers* procura identificar números que aparecem arbitrariamente no código dentro de blocos condicionais ou *loops*. O objetivo é promover a melhoria da legibilidade e manutenção do código, sugerindo a substituição desses números por constantes nomeadas, o que torna o código mais claro e fácil de modificar.

4.3 O código

O código foi projetado para analisar arquivos de código-fonte em Java ou Python, percorrendo linha por linha e aplicando sequencialmente as funções de detecção de code smells. A cada arquivo processado, o sistema gera uma contagem das ocorrências de cada tipo de code smell identificado. Essas informações são então armazenadas em um dicionário chamado codeSmellsStatistic, que armazena as estatísticas de cada code smell encontrado durante a análise.

Esse dicionário codeSmellsStatistic é uma estrutura de dados essencial para o funcionamento da ferramenta, pois permite o acesso rápido às contagens e a geração de relatórios detalhados. Cada tipo de code smell detectado (como números mágicos, código duplicado, métodos longos, entre outros) possui uma chave única no dicionário, e o valor associado a cada chave é o número de ocorrências desse code smell no arquivo analisado; essas estatísticas são úteis para fornecer *insights* sobre a qualidade do código, auxiliando os desenvolvedores a identificar áreas críticas que necessitam de refatoração e melhoramento.

Uma vez que a contagem é gerada, o sistema pode produzir um relatório, destacando os tipos de code smells encontrados, suas localizações e a frequência de ocorrência, facilitando assim a tarefa de refatoração e otimização do código-fonte.

4.4 Fluxo Geral

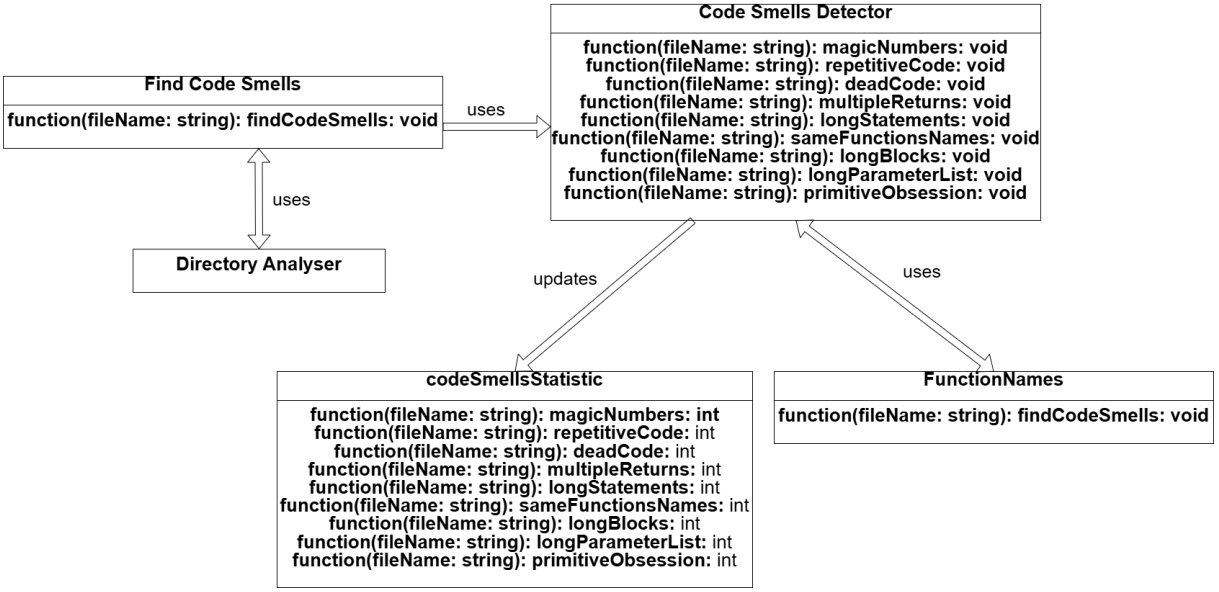
O processo de análise do *NimbusScan* segue um fluxo bem definido, começando pelo módulo directoryAnalyser, que explora os diretórios especificados e identifica os arquivos de código-fonte a serem analisados. Após essa identificação dos arquivos do código-fonte, a função principal, *findCodeSmells*, localizada no módulo Find *Code Smells* e coordena o processo de análise ao orquestrar a execução das funcionalidades disponíveis no módulo *codeSmellsDetector*.

O *codeSmellsDetector* contém diversas funções especializadas para identificar diferentes tipos de code smells, como: números mágicos que são valores numéricos inseridos diretamente no código sem explicação clara; código repetitivo que corresponde a trechos duplicados que podem ser abstraídos; código morto o qual representa partes do código que não são utilizadas; múltiplos retornos que significa quando funções apresentam mais de um ponto de saída; blocos longos o qual contém trechos extensos de código em um único bloco de controle; e nomes de funções repetidos que geram ambiguidade ao utilizarem nomes idênticos ou muito semelhantes.

Cada uma dessas funções é projetada para receber o nome do arquivo como parâmetro e realizar uma análise específica, aplicando técnicas como expressões regulares e parsing de código. Os resultados obtidos pelas funções são enviados para o módulo *codeSmellsStatistic*, que mantém um registro centralizado das contagens de cada tipo de code smell identificado.

O fluxo completo funciona de maneira coordenada com a identificação dos arquivos de código-fonte, a análise realizada pelas funções especializadas e o registro das estatísticas finais no *codeSmellsStatistic*. Essa abordagem sistemática garante que o sistema consiga identificar e registrar *code smells* de forma eficiente e escalável, facilitando o trabalho dos desenvolvedores na manutenção e refatoração do código.

Figura 1 - Descrição dos módulos



Fonte: Elaborado pelo autor, 2025.

4.5 A detecção de smells

O processo de análise inicia com a leitura do arquivo de código-fonte linha por linha. Durante essa leitura, dependendo da estrutura identificada, como blocos de funções, classes, loops e condicionais, funções especializadas são acionadas para detectar possíveis problemas. O sistema também monitora o nome das funções e verifica a ocorrência de múltiplos retornos ou parâmetros excessivamente longos. Após a análise de todos os arquivos, as informações sobre os code smells identificados são armazenadas em um arquivo CSV, proporcionando um relatório detalhado sobre a qualidade do código. O fluxo do programa foi otimizado para explorar as pastas e subpastas do projeto, garantindo que todos os arquivos escritos em Java ou Python sejam analisados de forma abrangente.

O *NimbusScan* realiza a análise do código-fonte visando identificar diversos *code smells* comuns que podem comprometer a qualidade e a manutenção do *software*. A detecção é conduzida por meio de funções especializadas, cada uma responsável por verificar aspectos específicos do código. Os resultados são organizados e apresentados de forma estruturada, facilitando a identificação de áreas que necessitam de aprimoramentos. Para cada code smell encontrado, o sistema fornece uma descrição detalhada de sua localização, indicando as linhas de código impactadas.

4.6 Validação e Testes

Testes foram conduzidos em cada função para assegurar a precisão dos algoritmos implementados. A validação envolveu uma análise comparativa entre as linguagens Python e Java, garantindo a eficácia do sistema em ambos os contextos. O processo incluiu uma análise sintática detalhada, considerando que cada tipo de code smell possui um padrão característico que pode variar conforme a linguagem de programação.

Por exemplo, a função *checkDeadcodeAfterReturn* é projetada para identificar a presença de código desnecessário (*dead code*) logo após uma instrução de retorno (*return statement*). O processo consiste em percorrer as linhas subsequentes ao *return*, verificando se elas estão no mesmo nível de indentação. Caso sejam encontradas, essas linhas são classificadas como código morto e marcadas para análise.

5 Estudo de Caso

O estudo analisou a detecção de code smells na plataformas *OpenStack* e *CloudStack* desenvolvidos em Python e Java, respectivamente, utilizando a ferramenta *NimbusScan*, projetada para identificar práticas de codificação prejudiciais à legibilidade, manutenção e eficiência do código. O processo incluiu a leitura sistemática do código-fonte para identificar estruturas como funções e *loops*, a detecção automatizada de problemas como código repetitivo, números mágicos e funções com múltiplas instruções de retorno, a geração de relatórios detalhados em formato CSV e a validação da eficácia da ferramenta por meio de testes comparativos entre as linguagens, com análise sintática detalhada.

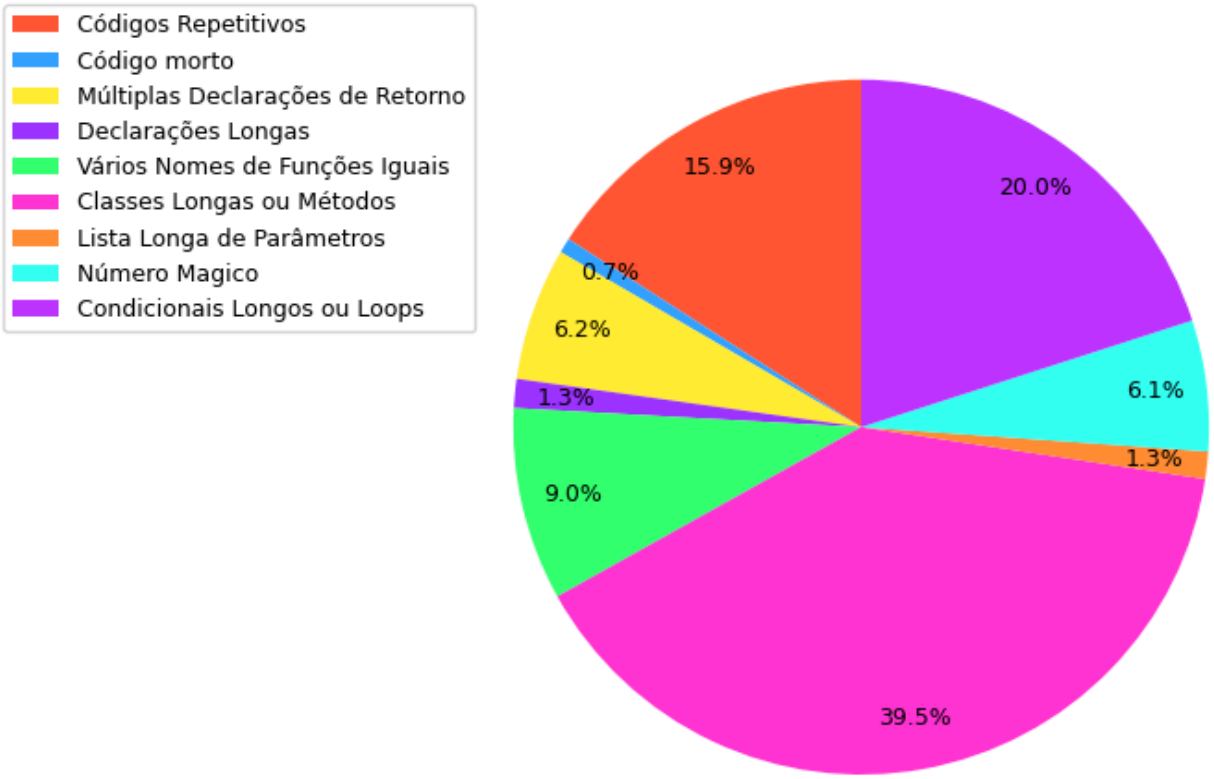
5.1 CloudStack

A análise de *code smells* no *CloudStack* revelou desafios significativos relacionados à manutenção e qualidade do código.

5.1.1 Distribuição de code smells no CloudStack

A Figura 2 apresenta a distribuição percentual dos *code smells* identificados no Cloudstack. Os mais prevalentes foram classes longas e métodos extensos, correspondendo a 39,5% do total. Em seguida, destacaram-se condicionais e loops longos, com uma incidência de 20%. O *code smell* número mágico, introduzido como uma contribuição deste estudo, representou 6,1% do total analisado. Esses resultados evidenciam os principais desafios na manutenção do código da plataforma, ressaltando a necessidade de abordar tais problemas para melhorar a qualidade, a legibilidade e a sustentabilidade do sistema a longo prazo.

Figura 2 - Code smells no CloudStack

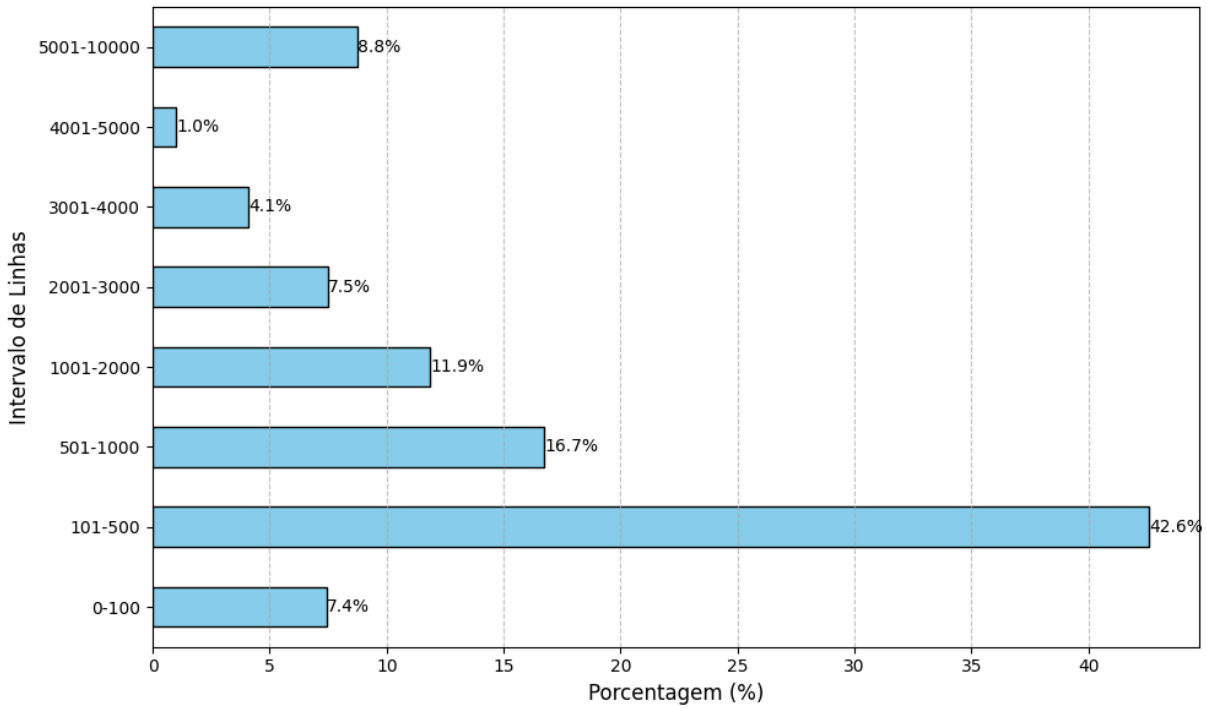


Fonte: Elaborado pelo autor, 2025.

No contexto do *CloudStack*, a prevalência de classes longas e métodos extenso representam 39,5% dos *code smells* identificados, isso evidencia um possível desalinhamento com práticas que priorizam a modularidade e a separação de responsabilidades, características essenciais em sistemas orientados a objetos como mencionado em (Mens et al., 2004). Embora, em alguns casos, manter classes com múltiplas responsabilidades possa ser uma decisão consciente para evitar a fragmentação excessiva do código, a elevada incidência desse tipo de *code smell* no *CloudStack* sugere que essa prática pode estar contribuindo para aumentar a complexidade do sistema, dificultando sua manutenção e evolução. Isso reforça a necessidade de uma análise criteriosa do contexto de cada classe longa ou método extenso, para determinar se sua estrutura atual é justificada ou se uma refatoração poderia melhorar a qualidade e a sustentabilidade do código.

A Figura 3 apresenta a distribuição percentual de *code smells* no *CloudStack* em diferentes intervalos de tamanho de arquivos, considerando a quantidade de linhas de código. Observa-se que a maior incidência de code smells ocorre em arquivos com 101 a 500 linhas, representando 42,6% do total; vale ressaltar que para esse code smells numa plataforma desenvolvida majoritariamente em Java se aplica as características essenciais em sistemas orientados a objetos apresentados em (Mens et al., 2004), muitas dessas ocorrências servem para evitar a fragmentação do código o que deixaria de se enquadrar como code smells.

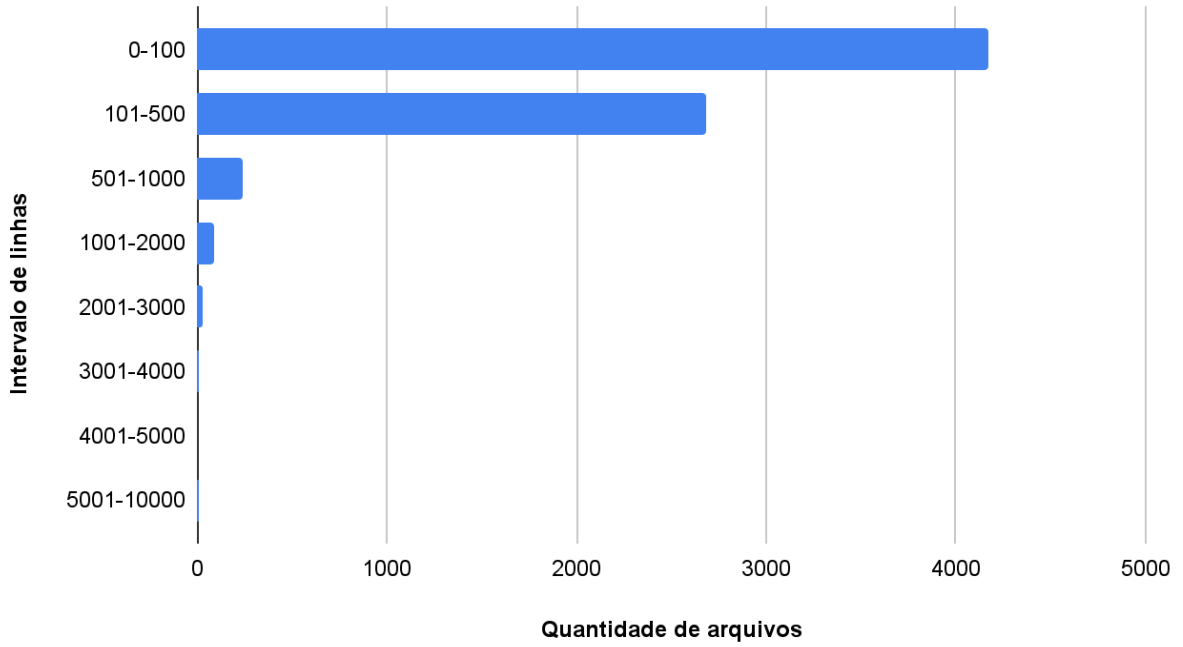
Figura 3 - Code smells por intervalo de linhas no CloudStack



Fonte: Elaborado pelo autor, 2025.

A Figura 4 ilustra a distribuição da quantidade de arquivos no *CloudStack* em relação ao intervalo de linhas de código por arquivo.

Figura 4 - Distribuição da quantidade de linhas por arquivo



Fonte: Elaborado pelo autor, 2025.

A análise revela que o projeto *CloudStack* é composto majoritariamente por arquivos pequenos, com até 500 linhas e que concentram a maior incidência de *code smells*, como pode ser observado nas

Figuras 3 e 4. Embora arquivos maiores, com mais de 500 linhas, apresentem menos de 6% do total de arquivos, acabam por representar cerca de 33% dos code smells, destacando sua relevância para a manutenção do código. Esses resultados apontam para a necessidade de estratégias específicas tanto para lidar com a frequência de arquivos com code smells, tanto em arquivos pequenos quanto para arquivos maiores.

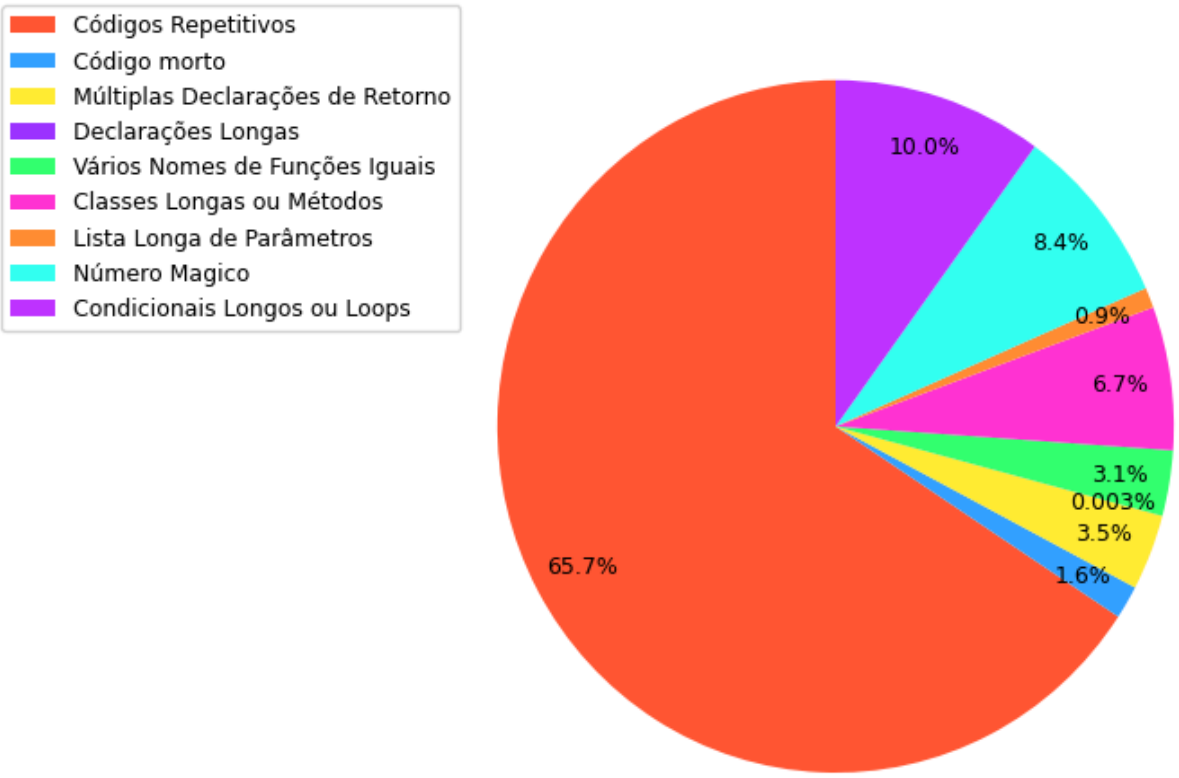
5.2 Openstack

Essa seção apresenta o resultado da detecção dos code smells na plataforma OpenStack.

5.2.1 Distribuição de code smells no OpenStack

A Figura 5 apresenta a distribuição percentual dos *code smells* identificados no *OpenStack*. O *code smell* mais recorrente identificado foi o código repetitivo, representando 65,7% do total, com 25084 ocorrências. Em seguida, destacaram-se as condicionais longas e loops, com 10%, e 2553 ocorrências, enquanto o número mágico correspondeu a 8,4% dos code smells. Notavelmente, as declarações longas apresentaram uma incidência mínima, representando apenas 0,003% do total de code smells.

Figura 5 - Code smells no OpenStack

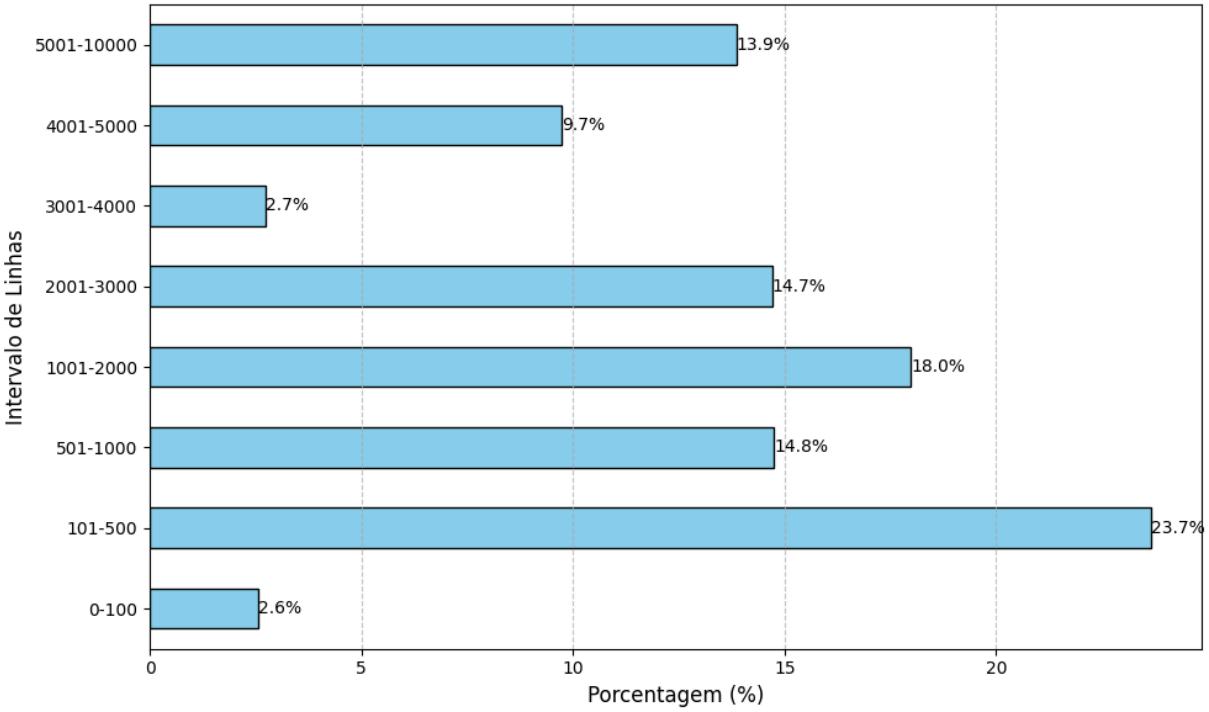


Fonte: Elaborado pelo autor, 2025.

5.4.2 Distribuição de code smells no OpenStack

A Figura 6 retrata a distribuição percentual de *code smells* no *OpenStack* por intervalo de linhas. Arquivos no intervalo de 101 a 500 linhas possuem a maior concentração de *code smells*, representando 23,7% do total, sendo 7157 ocorrências, seguindo por arquivos entre 1001 e 2000 linhas com 18% dos *code smells,* um total de 5437 ocorrências.

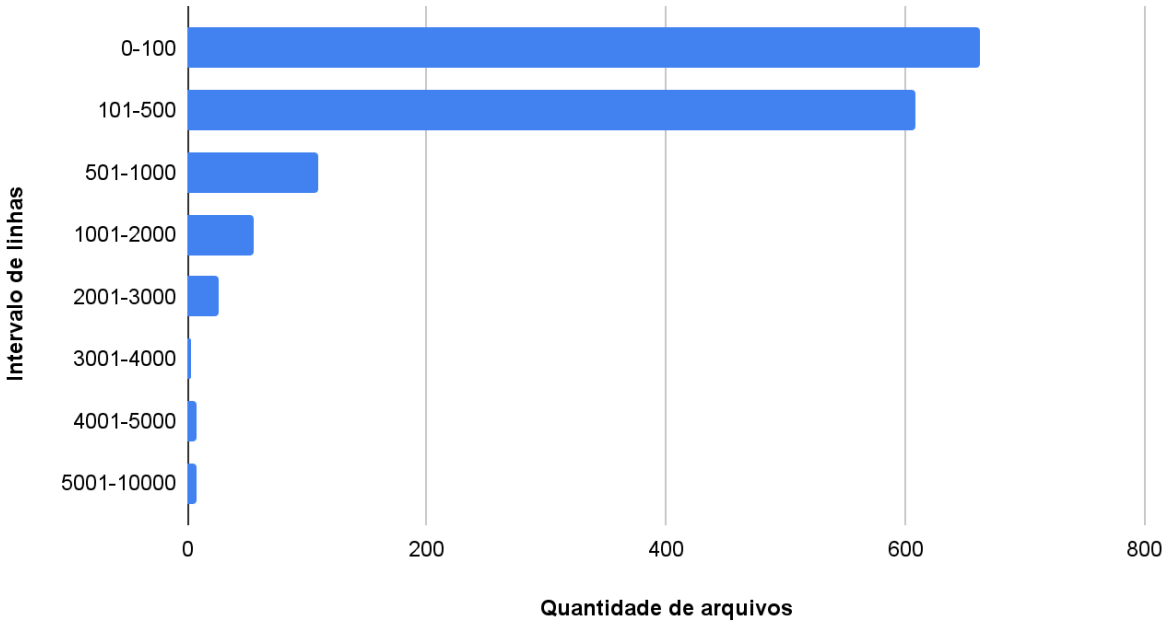
Figura 6 - Code smells por intervalo de linhas no OpenStack



Fonte: Elaborado pelo autor, 2025.

A Figura 7 apresenta a distribuição da quantidade de arquivos por intervalo de linhas no *OpenStack*.

Figura 7 - Distribuição da quantidade de linhas por arquivo



Fonte: Elaborado pelo autor, 2025.

Observa-se que arquivos maiores, com mais de 500 linhas, são menos frequentes no projeto *OpenStack*, conforme apresentado na Figura 7. No entanto, a análise indica que esses arquivos concentram uma quantidade significativa de *code smells*, totalizando 22.297 ocorrências, como

mostrado na Figura 6. Esse dado evidencia que, embora representem uma parcela menor do total de arquivos, esses arquivos extensos possuem um impacto desproporcional na incidência de *code smells*. Assim, torna-se fundamental adotar estratégias específicas para sua manutenção, garantindo que sua complexidade não comprometa a qualidade do código e a sustentabilidade do sistema ao longo do tempo.

A ferramenta *NimbusScan* se destaca como uma solução eficaz na detecção de *code smells*, demonstrando sua importância na manutenção da qualidade do código, especialmente em ambientes de nuvem. A análise dos resultados obtidos durante a implementação da ferramenta em plataformas como *OpenStack* e *CloudStack* revela sua capacidade de identificar, de forma precisa e sistemática, problemas estruturais que podem comprometer a performance e a manutenibilidade do *software*. A diversidade de *code smells* detectados, aliada a uma taxa significativa de identificação, facilita o processo de revisão de código e permite aos desenvolvedores priorizar as áreas que exigem refatoração. Ao adotar ferramentas automatizadas como a *NimbusScan*, os desenvolvedores dão um passo importante na busca por melhores práticas de desenvolvimento, promovendo uma abordagem mais proativa na melhoria contínua da qualidade do código.

Dentro dos *code smells* identificados, o número mágico se destaca como um problema crítico que afeta a legibilidade e manutenibilidade do código. Este tipo de *code smell* envolve o uso de valores numéricos diretamente no código, sem contexto ou explicação, o que pode gerar confusão e dificultar a compreensão do propósito do código. A detecção de números mágicos pela *NimbusScan* é crucial para promover melhores práticas, como a substituição desses valores por constantes nomeadas, proporcionando clareza e contexto ao código. Ao adotar essa prática, o código se torna mais fácil de entender e futuras modificações podem ser realizadas com maior facilidade. A inclusão da detecção de números mágicos na análise automatizada da *NimbusScan* ressalta a necessidade de uma abordagem consciente nas práticas de codificação, visando a melhoria contínua da qualidade do software, especialmente em projetos complexos.

6 Discussão de resultados

A quantidade de linhas de código está diretamente correlacionada ao número de *code smells* identificados, evidenciando a importância de adotar design patterns e práticas de modularização para evitar a criação de arquivos excessivamente grandes. A separação do código em módulos não só melhora a organização e a manutenção, mas também contribui para a legibilidade, tornando o código mais fácil de entender e modificar ao longo do tempo.

Além disso, observa-se uma diferença significativa na quantidade de código repetitivo entre as plataformas analisadas. O *CloudStack* que foi desenvolvido em Java, apresenta uma incidência menor de código repetido em comparação com o *OpenStack*, que utiliza Python. Esse contraste pode ser atribuído às características intrínsecas de cada linguagem.

O Java é uma linguagem fortemente orientada a objetos, assim favorece o uso de conceitos como herança, encapsulamento e polimorfismo, que incentivam a reutilização de código e reduzem a duplicação. Por outro lado, o Python, embora também suporte programação orientada a objetos, oferece maior flexibilidade, permitindo a utilização de diferentes paradigmas de programação. Essa flexibilidade, embora benéfica em muitos casos, pode resultar em padrões de código menos rígidos e, consequentemente, em uma maior ocorrência de repetições, especialmente quando boas práticas de design não são seguidas de forma consistente.

Essa disparidade entre as plataformas ressalta a influência do paradigma de programação e da linguagem escolhida na organização e manutenção do código, especialmente em projetos de grande escala como o *OpenStack* e o *CloudStack*. Portanto, é fundamental adotar boas práticas de desenvolvimento para mitigar o impacto desses problemas, independentemente da linguagem utilizada.

7 Conclusão

Este trabalho apresentou como uma das principais contribuições o desenvolvimento da ferramenta *NimbusScan*, projetada para a detecção automatizada de code smells em ambientes de computação em nuvem. O trabalho também mapeou um novo code smell para ambientes de nuvem, o

número mágico, como uma métrica relevante para melhorar a qualidade do código. Além disso, esse artigo apresentou uma análise comparativa da ocorrência de code smells em duas plataformas de nuvem, OpenStack, desenvolvida em Python, e CloudStack, em Java.

A análise de code smells no OpenStack e *CloudStack* revelou aspectos importantes sobre a qualidade e a manutenção do código em grandes projetos de software. A distribuição de code smells demonstrou que a quantidade de linhas de código está intimamente relacionada à complexidade e à ocorrência de problemas de qualidade, destacando a necessidade de práticas como a adoção de *design patterns* e a modularização do código. Esses recursos são essenciais para evitar a criação de arquivos excessivamente grandes e promover uma melhor organização, legibilidade e manutenção do código ao longo do tempo.

Além disso, as diferenças observadas na incidência de código repetitivo entre as duas plataformas, Java (*CloudStack*) e Python (*OpenStack*), destacam o impacto dos paradigmas de programação e das características das linguagens utilizadas no desenvolvimento de *software*. O Java, com sua forte orientação a objetos, promove a reutilização e diminui a duplicação de código, enquanto o Python, mais flexível em seus paradigmas, pode resultar em maior ocorrência de repetições quando as boas práticas não são seguidas.

Esses resultados reforçam a importância de adotar estratégias adequadas de desenvolvimento, que considerem tanto as características da linguagem quanto os padrões de design, para mitigar a ocorrência de code smells e, assim, melhorar a qualidade do *software*. Projetos de grande porte como o *OpenStack* e o *CloudStack* exigem constante vigilância e refatoração para garantir que o código permaneça sustentável e fácil de manter ao longo do tempo, independentemente das diferenças tecnológicas entre as plataformas.

Em futuras pesquisas, será realizada a análise sobre a relação entre a modularização, o uso de *design patterns* e a ocorrência de code smells, além de explorar maneiras de automatizar a detecção e correção de problemas em código de forma mais eficiente.

Referências

SOMMERVILLE, I. **Engenharia de Software.** 10. ed. Pearson, 2019.

FOWLER, M. **Refactoring: Improving the Design of Existing Code; Addison-Wesley Longman Publishing Co.**, Inc.:Boston, MA, USA, 2018.

IMRAN, M.; KOSAR, T. **Impact of Code Smells on Cloud-Based Systems: A Systematic Review.** Cloud Computing and Applications, v. 12, n. 3, p. 45-67, 2022.

SHAH, M.; et al. **Detecting Code Smells in Cloud Computing Environments: Challenges and Approaches.** IEEE Transactions on Cloud Computing, v. 11, n. 2, p. 234-246, 2023.

REYNOLDS, Sébastien Goasguen. *60 Recipes for Apache CloudStack: Using the CloudStack Ecosystem*. 1. ed. Sebastopol: O'Reilly Media, 2014.

BECK, K.; FOWLER, M. **Refactoring: Improving the Design of Existing Code.** Addison-Wesley, 1999.

MENS, Tom et al. A Survey of Software Refactoring. IEEE Transactions on Software Engineering, v. 30, n. 2, p. 126-139, 2004.

KUMARA, Indika et al. Towards semantic detection of smells in cloud infrastructure code. In: **Proceedings of the 10th International Conference on Web Intelligence, Mining and Semantics**. 2020. p. 63-67.

TAHMID, Ahmad; NAHAR, Nadia; SAKIB, Kazi. Understanding the evolution of code smells by observing code smell clusters. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE, 2016. p. 8-11.

Walker, A., Das, D., & Cerny, T. (2020). Automated Code-Smell Detection in Microservices. *Application Sciences*, 10(7800), 1-20.

ALLARD, A. 31 **Code Smells all software engineers must watch out for.** Pragmatic Ways. 2023. Disponível em: https://pragmaticways.com/31-code-smells-you-must-know/. Acesso em 10 de out de 2024.

GUPTA, S.; SINGH, A. **Code Smells in Python: A Study of Open Source Repositories.** Journal of Software Engineering Research and Development, v. 8, n. 1, p. 1-18, 2020.

MARTINS, J. S. **INVESTIGANDO O IMPACTO DAS COOCORRÊNCIAS DE CODE SMELLS NOS ATRIBUTOS INTERNOS DE QUALIDADE.** UFCE. PCOMP. 2021.