UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM SISTEMAS DE INFORMAÇÃO

EDUARDO DE OLIVEIRA SOUSA

UMA EXTENSÃO DE ISTAR PARA MODELAGEM DE REQUISITOS PARA PREVENÇÃO DE CODE SMELLS

QUIXADÁ

2025

EDUARDO DE OLIVEIRA SOUSA

UMA EXTENSÃO DE ISTAR PARA MODELAGEM DE REQUISITOS PARA PREVENÇÃO DE CODE SMELLS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Orientador: Prof. Dr. Enyo José Tavares Gonçalves.

QUIXADÁ

2025

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S696e Sousa, Eduardo de Oliveira.
Uma extensão de iStar para modelagem de requisitos para prevenção de code smells / Eduardo de Oliveira Sousa. – 2025.
56 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2025.
Orientação: Prof. Dr. Enyo José Tavares Gonçalves.

1. code smells. 2. modelagem de requisitos. 3. extensão de istar. 4. qualidade de software. I. Título.
CDD 005

EDUARDO DE OLIVEIRA SOUSA

UMA EXTENSÃO DE ISTAR PARA MODELAGEM DE REQUISITOS PARA PREVENÇÃO DE CODE SMELLS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Sistemas de Informação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Sistemas de Informação.

Aprovada em: 07/03/2025

BANCA EXAMINADORA

Prof. Dr. Enyo José Tavares Gonçalves (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Marcos Antonio de Oliveira
Universidade Federal do Ceará (UFC)

Prof. Esp. Erlânio Freire Barros
EEEP Professor Plácido Aderaldo Castelo

À minha namorada, pelo apoio incondicional. À minha mãe, pela dedicação incansável. Ao meu pai, pela segurança e força em minha jornada.

AGRADECIMENTOS

Agradeço, primeiramente, à minha namorada, Antonia Erlene dos Santos, por estar sempre ao meu lado, me incentivando e apoiando em cada etapa desta jornada.

Aos meus pais, Marly de Oliveira Lima e Francinaldo Lourenço de Sousa, expresso minha mais profunda gratidão pelo esforço e dedicação em me proporcionar uma educação de qualidade, permitindo que eu chegasse até aqui.

Ao meu orientador, Prof. Dr. Enyo José Tavares Gonçalves, agradeço pela excelente orientação, pelo apoio constante e pelas contribuições fundamentais para o desenvolvimento deste trabalho.

Aos membros da banca examinadora, Prof. Dr. Marcos Antonio de Oliveira e Prof. Esp. Erlânio Freire Barros, agradeço pelas valiosas considerações e pelo olhar crítico que ajudaram a aprimorar esta pesquisa.

Por fim, expresso minha gratidão à Universidade Federal do Ceará e a todo o seu corpo docente, que contribuíram significativamente para minha formação acadêmica e profissional.

"Enquanto houver 1% de chance, teremos 99% de fé." (Neymar Jr, 2017.)

RESUMO

A qualidade do software é um dos fatores mais importantes para o sucesso de sistemas computacionais, e *code smells* são indicadores de problemas estruturais que podem comprometer essa qualidade. *Code smells* são padrões recorrentes no código que apontam para possíveis falhas no design ou implementação, impactando negativamente a manutenção e a evolução do software. Apesar da existência de práticas e ferramentas para sua identificação e correção, muitas vezes a prevenção de *code smells* não é adequadamente considerada nas etapas iniciais do desenvolvimento, como na engenharia de requisitos. A engenharia de requisitos é uma fase fundamental para a definição e especificação das funcionalidades e características desejadas de um sistema. Linguagens de modelagem baseadas em objetivos, como o iStar, são amplamente utilizadas para capturar as necessidades e motivações dos *stakeholders* durante essa fase. O *iStar* tem sido estendido para abordar especificidades de diversos domínios, sendo suportado por um processo estruturado para criação de extensões. Assim, este trabalho propõe uma extensão do *iStar* para representar requisitos voltados à prevenção de *code smells*. A extensão inclui três conceitos principais: *Quality Smell*, *Task Smell* e *Smell Metric*. Esses conceitos permitem modelar a prevenção de *code smells* de forma sistemática, contribuindo para o desenvolvimento de sistemas com maior qualidade e menor custo de manutenção.

Palavras-chave: *code smells*; modelagem de requisitos; extensão de *istar*; qualidade de software.

ABSTRACT

Software quality is one of the most important factors for the success of computational systems, and code smells are indicators of structural issues that can compromise this quality. Code smells are recurring patterns in code that point to potential design or implementation flaws, negatively affecting software maintenance and evolution. Despite the existence of practices and tools for their identification and correction, the prevention of code smells is often overlooked in the early stages of development, such as requirements engineering. Requirements engineering is a fundamental phase for defining and specifying the functionalities and desired characteristics of a system. Goal-oriented modeling languages, such as iStar, are widely used to capture the needs and motivations of stakeholders during this phase. iStar has been extended to address the specificities of various domains, supported by a structured process for creating extensions. Thus, this work proposes an iStar extension to represent requirements aimed at preventing code smells. The extension includes three main concepts: Quality Smell, Task Smell and Smell Metric. These concepts enable the systematic modeling of code smell prevention, contributing to the development of higher-quality systems with lower maintenance costs.

Keywords: code smells; requirements modeling; istar extension; software quality.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AHP	*Analytic Hierarchy Process*
BPMN	*Business Process Model and Notation*
ER	Engenharia de Requisitos
GORE	*Goal-Oriented Requirements Engineering*
KAOS	*Knowledge Acquisition Automated Specification*
MICMAC	*Matrice d'Impacts Croisés Multiplication Appliquée à un Classement*
PRISE	*Process to Support iStar Extensions*
PriseBot	*Chatbot* de apoio ao desenvolvimento de extensões iStar
SD	*Strategic Dependency*
SR	*Strategic Rationale*
TISM	*Total Interpretive Structural Modelling*

SUMÁRIO

1 INTRODUÇÃO

A Engenharia de Requisitos (ER) é uma sub-área da engenharia de software que busca identificar, validar, documentar funções e restrições que um software deve cumprir enquanto é desenvolvido e está em operação (Zave 1997).

Dentro do escopo da ER, o *Goal-Oriented Requirements Engineering (GORE)* desempenha um papel crucial ao focar nos objetivos e metas que o sistema deve alcançar. O GORE vai além das especificações funcionais tradicionais, ao considerar o contexto e as intenções dos *stakeholders*. Conforme (Lamsweerde 2001) explica, o GORE visa a modelagem e análise de objetivos para descobrir e refinar requisitos do sistema. Essa metodologia é fundamental para lidar com a complexidade crescente dos sistemas de software modernos, onde os requisitos muitas vezes envolvem diversas partes interessadas com necessidade e expectativas variadas (Yu 1997; Mylopoulos *et al.* 1999). Os métodos para melhorar a comunicação entre *stakeholders* e técnicas para gerenciar requisitos complexos e conflitantes são fundamentais para o sucesso da ER (Aurum e Wohlin 2005).

Na ER, diversas abordagens são utilizadas para a especificação de requisitos, cada uma adequada a diferentes contextos e necessidades. Métodos tradicionais incluem a especificação de requisitos em linguagem natural, diagramas de casos de uso e especificações formais. No entanto, o *framework iStar* se destaca por oferecer uma abordagem orientada a objetivos, permitindo a representação de atores, seus objetivos e as relações entre eles (Yu 1997). O *iStar* proporciona uma notação gráfica que facilita a visualização e análise das dependências e motivações dos *stakeholders*, promovendo uma compreensão mais abrangente dos requisitos do sistema (Dalpiaz *et al.* 2016). Essa adaptabilidade é essencial em um mercado volátil e é sustentada por estudos que enfatizam a importância de frameworks ágeis na engenharia de requisitos (Nuseibeh e Easterbrook 2000).

O *framework iStar*, consolidado na ER, tem várias extensões para adaptá-lo a diferentes cenários e demandas. A importância dessas extensões é destacada por (Goncalves *et al.* 2020), que afirmam que elas ampliam a aplicabilidade do iStar. Para o desenvolvimento organizado de extensões, foi proposto o PRISE, que utiliza a notação *Business Process Model and Notation* (BPMN) para modelar o processo. O PRISE orienta os desenvolvedores na criação de extensões alinhadas com os princípios do *iStar*, garantindo assim uma abordagem sistemática. Essa estruturação é crucial para manter a coerência das extensões e atender às necessidades específicas dos projetos de software (Horkoff *et al.* 2019).

No desenvolvimento de extensões *iStar*, o PriseBot é uma ferramenta inovadora projetada para aprimorar e refinar o processo de desenvolvimento seguindo o processo PRISE. Este *chatbot* desempenha um papel importante com relação ao auxilio ao desenvolvimento de extensões, uma vez que questionamentos de extensões relacionadas sobre o processo PRISE e extensões e construtores já desenvolvidos podem ser respondidos de forma ágil e rápido pelo *bot*, favorecendo a produtividade no desenvolvimento da extenão (Freire *et al.* 2024).

Na Qualidade de Software, *code smells* são indicativos de problemas no código que podem afetar negativamente sua qualidade e manutenibilidade. Para (Fowler 2018), *code smells* são certas estruturas no código que sugerem (mas não garantem) a possibilidade de refatoração. A complexidade introduzida pelos *smells* pode diminuir significativamente a eficiência do desenvolvimento, aumentando o custo de futuras modificações e correções (Valente 2020). Existem vários repositórios que catalogam *smells*, oferecendo exemplos detalhados e soluções de refatoração, como o *Code Smells Catalog*[1].

Diante disso, este trabalho busca incorporar uma modelagem de requisitos orientada à prevenção de *code smells* na ER, utilizando o *framework iStar*. A proposta inclui o desenvolvimento de uma extensão específica para *iStar*, seguindo o processo PRISE e apoiada pelo PriseBot, para abordar de forma sistemática os requisitos que previnem *code smells*.

1.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver uma extensão específica do *framework iStar* para a modelagem de requisitos orientada à prevenção de *code smells*, utilizando o processo PRISE e o PriseBot.

1.2 Objetivos Específicos

- Identificar e classificar os principais tipos de *code smells* que impactam a qualidade do software.
- Propor uma notação gráfica dentro do *framework iStar* para representar requisitos que previnem *code smells*.
- Validar a extensão proposta através de um estudo de caso, avaliando sua eficácia na fase de modelagem do software.

[1] https://luzkan.github.io/smells/ Acesso em: 24 de outubro 2024

2 FUNDAMENTAÇÃO TEÓRICA

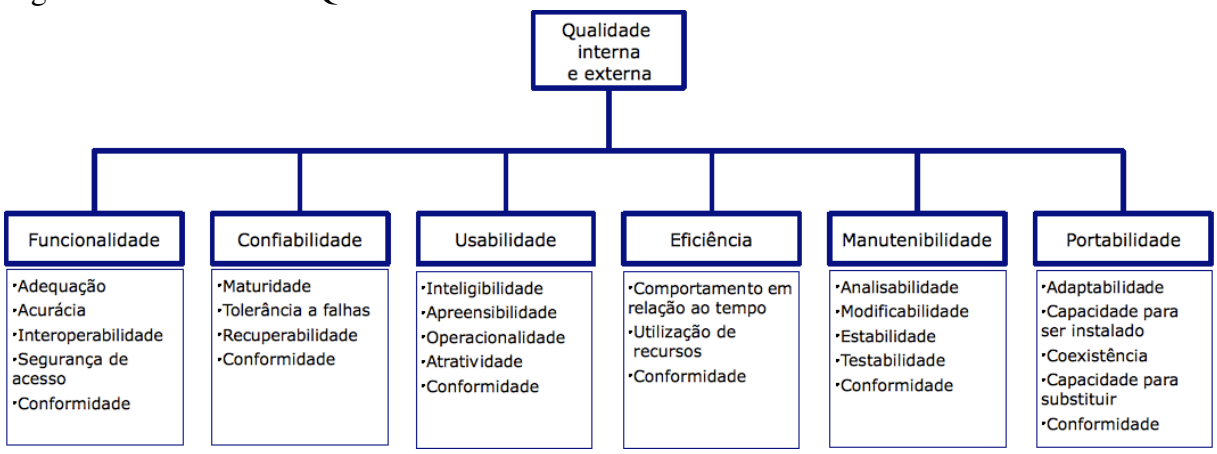
Neste capítulo, serão apresentados os principais conceitos fundamentais para o desenvolvimento deste trabalho. A Seção 2.1 trata dos aspectos relacionados à Qualidade de Software e aos requisitos de *smells*. Na Seção 2.2, discute-se a Engenharia de Requisitos. A Seção 2.3 oferece uma descrição detalhada sobre o *framework iStar* e suas extensões. Por fim, a Seção 2.4 aborda o processo PRISE e o PriseBot.

2.1 Qualidade de Software e *Code Smells*

A qualidade de software é um conceito fundamental que determina o valor e a eficácia de um produto de software. Ela engloba vários atributos importantes, como funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade e portabilidade, conforme definido pela (ISO/IEC 2024). Esses atributos asseguram que o software não apenas atenda às necessidades funcionais específicas dos usuários, mas também ofereça uma experiência de uso agradável e seja sustentável ao longo do tempo em termos de manutenção e atualizações.

Para ilustrar de maneira mais visual como a qualidade de software é avaliada, podemos visualizar, na Figura 1, uma decomposição dos atributos de qualidade. Cada categoria de qualidade de software, conforme a norma (ISO/IEC 2024), é dividida em subatributos específicos que detalham os critérios para avaliação. Por exemplo, a funcionalidade abrange adequação, acurácia, interoperabilidade, segurança de acesso e conformidade, enquanto a confiabilidade inclui maturidade, tolerância a falhas, recuperabilidade e conformidade. Essa visualização facilita a implementação de práticas que garantem a excelência do produto.

Figura 1 – Atributos de Qualidade de Software



Fonte: (ISO/IEC 2024)

Essas métricas formam a base para avaliar a qualidade do software, identificando áreas de melhoria. Seguir esses padrões é essencial para desenvolver software que atenda às expectativas dos *stakeholders* e mantenha a competitividade. Pesquisas realizadas por (Kitchenham e Brereton 2013) indicam que a aplicação consistente desses critérios pode aumentar a satisfação do cliente e promover a inovação nas empresas de software.

A qualidade de software e os requisitos de *smells* estão intimamente ligados, pois *smells* são sinais de problemas no código que podem comprometer métricas como a manutenibilidade e a eficiência. Exemplos de *smells* incluem métodos longos e duplicação de código, que dificultam a manutenção e aumentam a probabilidade de falhas. Identificar e corrigir esses *smells* é crucial para garantir que o software seja sustentável e fácil de evoluir. Estudos destacam que a presença de *smells* está associada a dificuldades de manutenção e defeitos no software, tornando sua gestão essencial para o sucesso dos projetos de software (Yamashita e Moonen 2012; Cairo *et al.* 2018).

Antes de explorar os tipos específicos de *smells*, é crucial entender que esses são sinais de problemas estruturais no código que podem comprometer significativamente a qualidade do software (Marinescu 2004). Eles não apenas indicam má práticas de programação, mas também podem afetar diretamente a manutenibilidade e a eficiência do software. O reconhecimento e a correção desses *smells* são passos vitais para melhorar a saúde do código e assegurar a funcionalidade do sistema ao longo do tempo (Fowler 2018). Vamos explorar alguns dos mais comuns e impactantes tipos de *code smells* listados em (Fowler *et al.* 1999) e descritos por (Shatnawi e Li 2006), que serão abordados no presente trabalho:

- ***Long Method***: Métodos que são excessivamente longos e complexos, tornando-os difíceis de entender e manter.
- ***Large Class***: Classes que acumulam muitas responsabilidades, violando o princípio de responsabilidade única.
- ***Long Parameter List***: Métodos que recebem um número excessivo de parâmetros, dificultando o entendimento e a manutenção.
- ***Feature Envy***: Métodos que acessam mais dados de outras classes do que da própria, indicando um possível problema de encapsulamento.
- ***Data Clump***: Conjuntos de dados que frequentemente aparecem juntos, sugerindo que deveriam ser agrupados em uma estrutura de dados.

Para abordar *smells* de maneira eficaz, várias estratégias podem ser aplicadas. Uma

abordagem é a refatoração, que envolve modificar o código para melhorar sua estrutura interna sem alterar seu comportamento externo (Kerievsky 2005). Técnicas específicas incluem a extração de métodos para reduzir a complexidade de métodos longos e a aplicação do padrão de design *Strategy* para lidar com classes excessivamente grandes. Além disso, a ferramenta de software *JDeodorant* [1] pode ser usada para identificar e remover os principais tipos de *smells* automaticamente (Tsantalis *et al.* 2008). A integração de práticas de revisão de código e programação em par também é recomendada para prevenir a reintrodução de *smells* (Beck 2000).

Existem inúmeros repositórios de *code smells* disponíveis que oferecem uma ampla gama de recursos para identificar e corrigir problemas de design no código. Esses repositórios são fundamentais para desenvolvedores que buscam manter e melhorar a qualidade do software através de práticas de refatoração. Eles oferecem exemplos detalhados de *smells* e as respectivas soluções para cada caso, o que é extremamente valioso para a educação e prática contínua dos desenvolvedores (Tsantalis *et al.* 2008; Moha *et al.* 2009).

Entre esses repositórios, se destaca como o mais atual e abrangente, o *Code Smells Catalog*. Este repositório não só cataloga uma vasta quantidade de *smells* mas também sugere métodos de refatoração atualizados, adequados para enfrentar os desafios dos ambientes de desenvolvimento modernos. Ele serve como um guia prático para as equipes de desenvolvimento aplicarem as melhores práticas de refatoração de maneira sistemática e eficaz, garantindo software de alta qualidade e fácil manutenção (Kessentini *et al.* 2011).

2.2 Engenharia de Requisitos

A ER desempenha um papel fundamental no sucesso do desenvolvimento de software, garantindo que os produtos finais estejam alinhados com as expectativas dos usuários e as necessidades do negócio. Essa área envolve várias etapas críticas, começando pela elicitação e análise de requisitos, seguida pela sua documentação e validação contínua ao longo do ciclo de vida do software (Sommerville 2011).

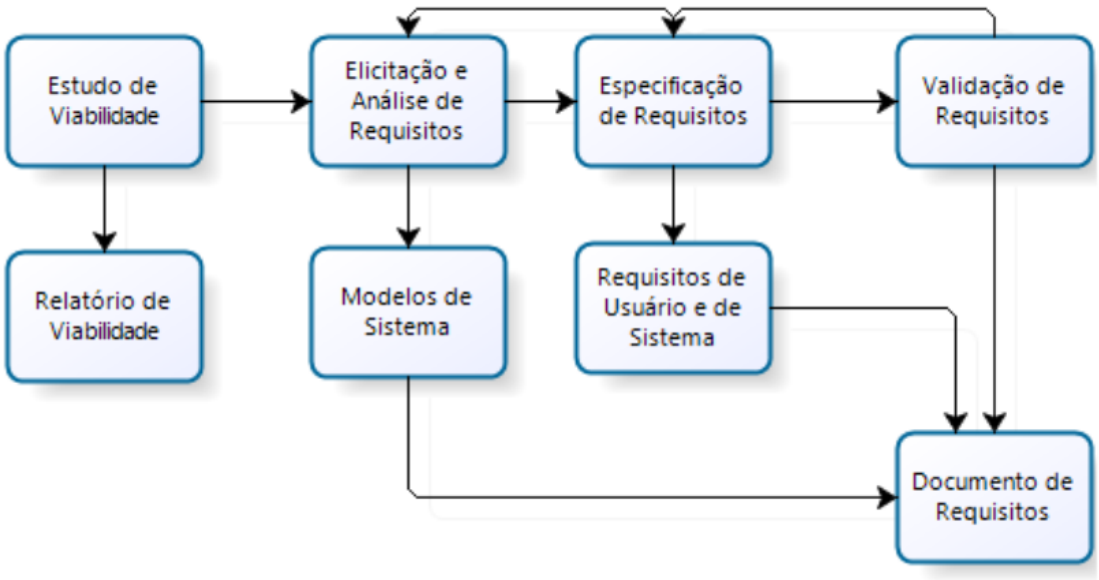
Os requisitos são categorizados em requisitos funcionais, que descrevem as funcionalidades específicas que o software deve realizar, e requisitos não funcionais, que abordam características como desempenho, segurança e usabilidade (Chung *et al.* 2012). A compreensão correta desses requisitos é vital, pois falhas na fase de requisitos podem levar a custos significativos de correção e manutenção após o desenvolvimento do software.

[1] https://github.com/tsantalis/JDeodorant Acesso em: 24 de outubro 2024

Modelos e abordagens como o GORE, são utilizados para capturar e analisar os objetivos e metas que o sistema deve satisfazer, utilizando ferramentas como *iStar* e *Knowledge Acquisition Automated Specification (KAOS)* para representar visualmente esses objetivos e suas inter-relações (Yu 1997). Essas representações ajudam a esclarecer as expectativas das partes interessadas e a definir metas claras e mensuráveis para o desenvolvimento do sistema.

A Figura 2 ilustra o processo iterativo e integrado da ER, destacando as fases de estudo de viabilidade, elicitação e análise de requisitos, especificação e validação de requisitos. Este processo não só facilita a detecção precoce de problemas, mas também suporta ajustes e melhorias contínuas, essenciais para responder às mudanças nas necessidades dos usuários e às evoluções no ambiente de negócios (Nuseibeh e Easterbrook 2000). A adaptabilidade proporcionada por este ciclo iterativo é crucial para a sustentabilidade do projeto, permitindo que as equipes de desenvolvimento reajam prontamente a novos desafios e oportunidades. Além disso, o engajamento constante com os *stakeholders* durante todas as fases do processo assegura que o desenvolvimento do software permaneça alinhado com as estratégias comerciais e as metas de mercado, maximizando assim o retorno sobre o investimento e reduzindo o risco de desalinhamento do projeto com as expectativas do mercado (Sommerville 2011).

Figura 2 – Processo de Engenharia de Requisitos



Fonte: (Sommerville 2011)

Portanto, a ER é uma área crítica que exige atenção detalhada e um método sistemá-

tico para garantir o completo entendimento, documentação e validação contínua dos requisitos, maximizando o sucesso do projeto e a satisfação do usuário.

2.3 *iStar* e extensões de *iStar*

O *framework iStar*, originado por (Yu 1995), é uma linguagem de modelagem voltada para a captura e representação de objetivos organizacionais e requisitos de software. Ele enfoca na delineação das intenções, motivações e estratégias dos diversos atores envolvidos, oferecendo uma estrutura robusta para a análise de relações interdependentes dentro de projetos complexos.

Uma ampla gama de estudos aplicou o *framework iStar* em contextos diversos, confirmando sua utilidade no manejo de sistemas adaptativos guiados por Requisitos, Design de Processos de Negócios, Engenharia de Requisitos de Segurança, entre outros. Essa aplicabilidade generalizada sublinha o papel do *iStar* em facilitar avaliações organizacionais e tecnológicas detalhadas e abrangentes (Cuesta *et al.* 2016).

Em 2016, a versão 2.0 do *iStar* foi introduzida (Dalpiaz *et al.* 2016), visando harmonizar as diversas variantes da linguagem que haviam se desenvolvido ao longo dos anos. Essa padronização foi uma resposta às barreiras encontradas na adoção do *framework* fora dos círculos de especialistas, facilitando sua aplicação mais ampla e aumentando sua acessibilidade para uma gama mais vasta de usuários e contextos aplicativos.

Os principais construtores presente na linguagem são:

- **Atores e tipos de atores**: No *iStar*, os atores são componentes cruciais que possuem ou são motivados por objetivos específicos. Há principalmente dois tipos de atores definidos: *Role* (Papel) e *Agent* (Agente). O *Role* representa uma abstração que define um conjunto de comportamentos ou responsabilidades esperadas de um ator em determinado contexto, enquanto o *Agent* refere-se a uma entidade concreta, que pode desempenhar um ou mais *Roles* ou operar de maneira independente. Em situações onde a diferenciação entre *Role* e *Agent* não é essencial, pode-se recorrer ao uso de um ator abstrato, uma figura genérica que serve como um placeholder para um papel ou agente ainda não determinado no modelo.
- **Associação de atores**: Os atores podem ser vinculados através de relações do tipo *is-a*, que indicam generalização ou especialização entre papéis, ou *participates-in*, usadas para expressar participações em atividades ou responsabilidades dentro de um contexto organizacional.
- **Dependências sociais**: Refletem as necessidades de cooperação entre atores, onde um ator

pode depender de outro para alcançar um objetivo, executar uma tarefa, satisfazer uma qualidade, ou fornecer um recurso necessário.

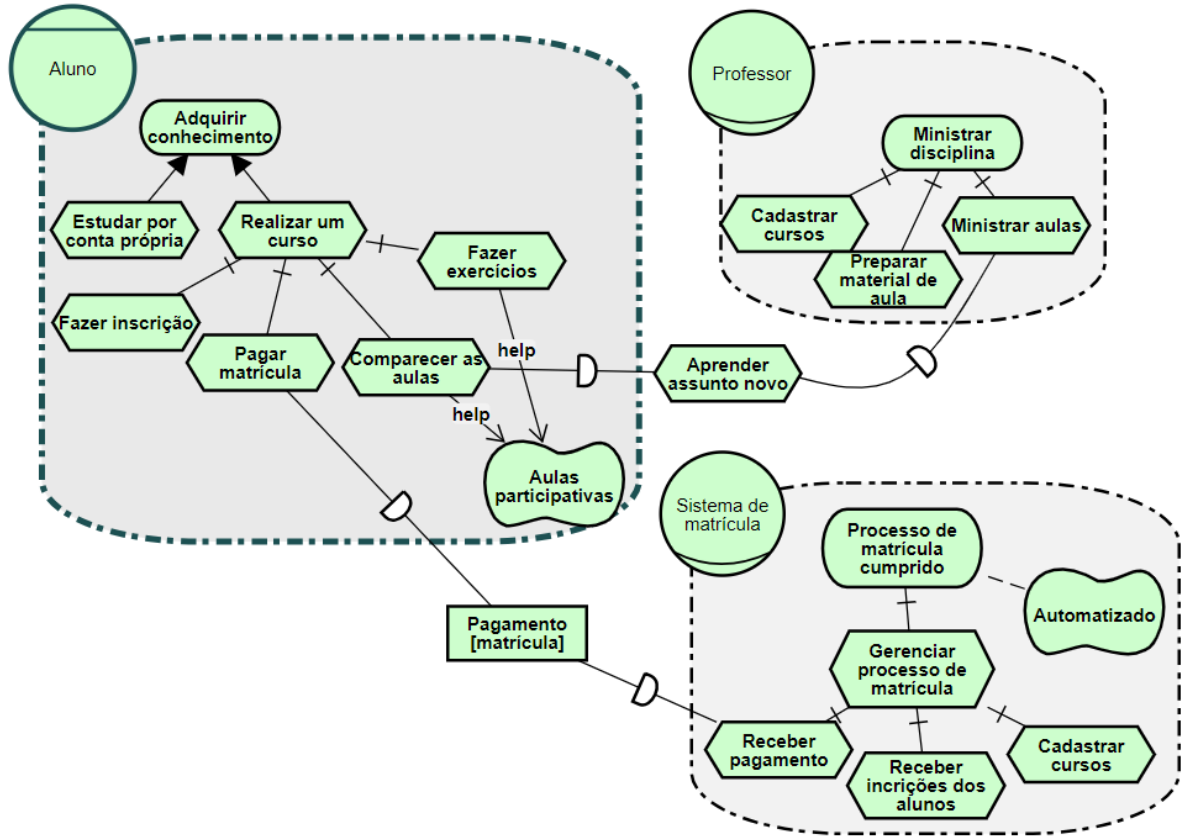
- **Elementos intencionais**: São os componentes que expressam as intenções dos atores, divididos em:
  - ***Goal* (Objetivo)**: Uma condição ou estado desejado que o ator pretende alcançar.
  - ***Quality* (Qualidade)**: Estabelece padrões de qualidade que os outros elementos devem satisfazer.
  - ***Resource* (Recurso)**: Elementos necessários, como informações ou objetos, que são utilizados para realizar tarefas.
  - ***Task* (Tarefa)**: Ações específicas que o ator empreende para alcançar seus objetivos.
- **Associação de elementos intencionais**: Descrição detalhada das interconexões entre os elementos dentro do sistema.
  - ***Refinement***: Estrutura hierárquica entre objetivos e tarefas, que pode ser representada como *AND* ou *OR*, indicando se a realização do elemento pai depende de todos ou apenas um dos elementos filhos.
  - ***Qualification***: Relaciona uma qualidade a um objetivo, tarefa ou recurso, especificando como esse elemento contribui para o alcance ou suporte da qualidade.
  - ***Contribution***: Indica o impacto de um elemento na realização de uma qualidade.
  - ***NeededBy***: Liga um recurso a uma tarefa, enfatizando a necessidade desse recurso para a execução bem-sucedida da tarefa.

2.3.1 Exemplo de modelagem de iStar

Na Figura 3, é apresentada uma modelagem *iStar* no contexto educacional, envolvendo os agentes "Aluno" e "Professor", além do sistema de "Matrícula". O aluno tem como objetivo principal "Adquirir conhecimento", que é refinado pelos objetivos de "Estudar por conta própria", "Realizar um curso" e "Fazer exercícios". Estes são interdependentes e essenciais para o sucesso acadêmico. Para se matricular em um curso, o aluno deve "Fazer inscrição" e "Pagar a matrícula", com o auxílio do sistema de matrícula que facilita essas transações. O professor, por sua vez, visa "Ministrar disciplina", decomposto em "Preparar material de aula" e "Ministrar aulas". O sistema de matrícula também suporta o professor com recursos automatizados para "Cadastrar cursos" e "Receber inscrições dos alunos". Este cenário ilustra a integração de múltiplos agentes e seus objetivos em um ambiente educacional, destacando a importância de

uma modelagem coerente para o suporte às atividades acadêmicas.

Figura 3 – Modelagem de sistema educacional



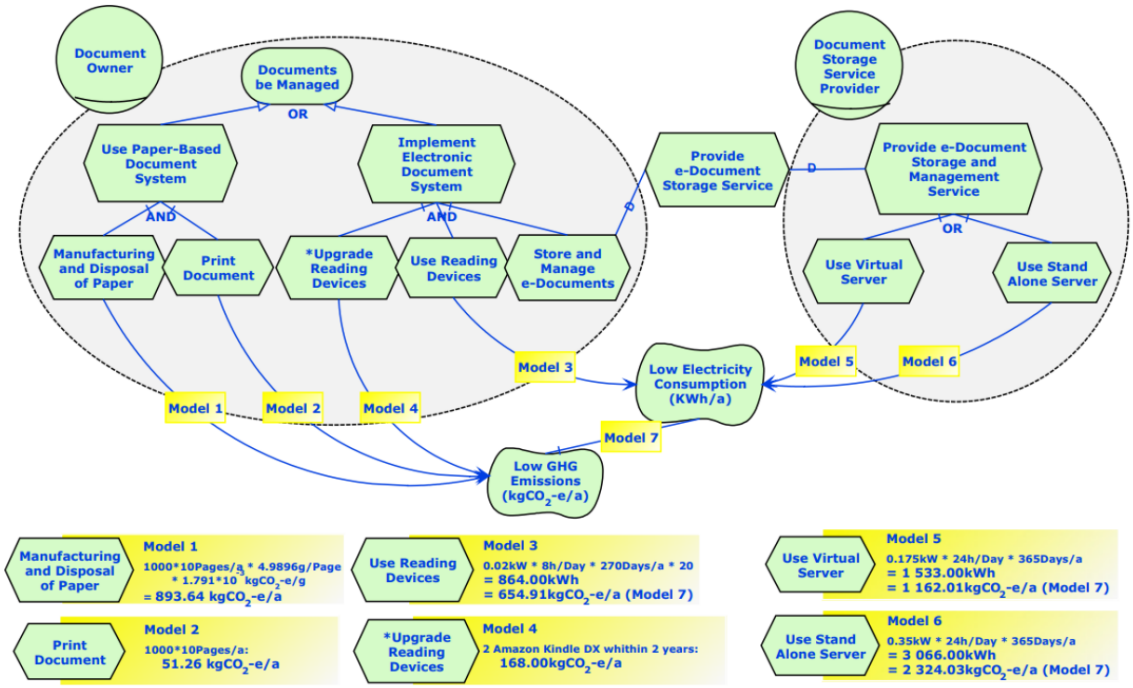
Fonte: Elaborada pelo autor

2.3.2 Exemplo de extensão de iStar

Na engenharia de software, frequentemente é necessário ajustar e expandir as tecnologias para que elas se encaixem melhor em contextos específicos de projetos e áreas de estudo. Isso também se aplica às linguagens de modelagem, onde muitas vezes é preciso estender a linguagem para adicionar novos construtores, modificar ou remover elementos existentes para que se adequem às necessidades específicas de determinados domínios (Whittle *et al.* 2013). Esta adaptação geralmente envolve a extensão de linguagens para incorporar novos construtores, além de modificar (Li *et al.* 2015).

Um exemplo notável disso é apresentado por (Zhang *et al.* 2011), onde uma nova extensão chamada *Model* foi adicionada à linguagem *iStar*. Esta relação é utilizada para representar modelos matemáticos que calculam impactos ambientais. Na Figura 4 esses modelos estão destacados em amarelo.

Figura 4 – Exemplo de uma extensão de iStar

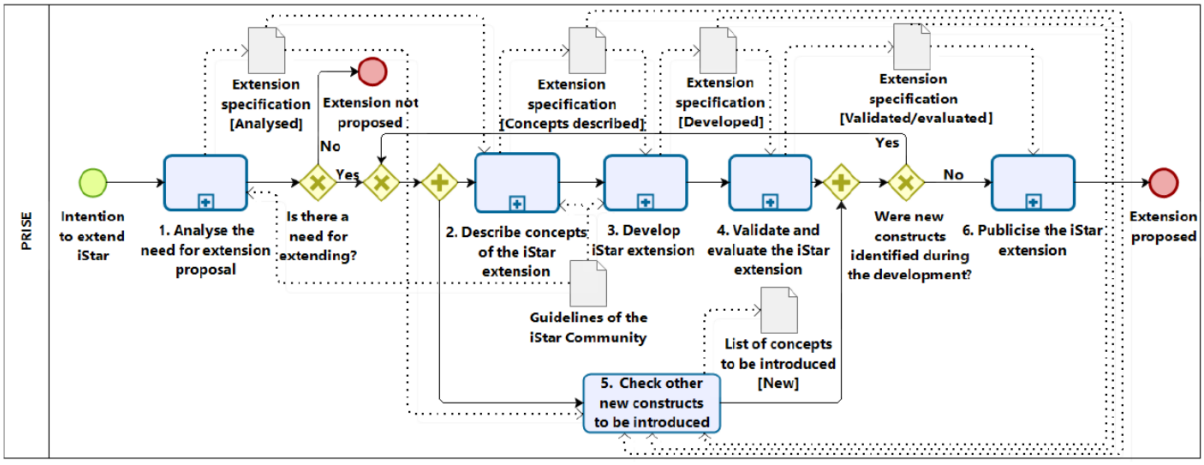


Fonte: (Zhang *et al.* 2011)

2.4 O processo PRISE e o PriseBot

As linguagens de modelagem frequentemente utilizam diagramas para representar sistemas e informações críticas, mas nem sempre conseguem abranger todos os conceitos de uma área específica. Para preencher essas lacunas, são criadas extensões, e para garantir que essas extensões sejam consistentes e livres de conflitos, foi desenvolvido um processo específico chamado PRISE (Goncalves *et al.* 2020). A Figura 5 apresenta uma visão geral do PRISE.

Figura 5 – Processo Principal do PRISE



Fonte: (Goncalves *et al.* 2020)

O PRISE é um método estruturado desenvolvido para apoiar a criação de extensões do *framework iStar*, garantindo consistência e integração adequadas. A seguir, detalharemos cada um dos subprocessos que compõem o PRISE, ilustrando como ele assegura a criação de extensões de alta qualidade para a linguagem *iStar* (Goncalves *et al.* 2020).

- ***Analyse the need for extension proposal*** (Analisar a necessidade de extensão): Este subprocesso inicial é crucial para determinar se realmente há uma necessidade de criar uma nova extensão para o *iStar*. A análise começa com a identificação das limitações e deficiências da versão atual do *iStar*. Isso pode envolver a realização de entrevistas com *stakeholders*, análise de casos de uso existentes e revisão de literatura. O objetivo é entender se os requisitos dos *stakeholders* não estão sendo atendidos pelas construções existentes. Se a análise concluir que uma extensão é necessária, a proposta é levada adiante. Caso contrário, a proposta de extensão é descartada. Esta fase garante que apenas extensões que realmente agregam valor sejam desenvolvidas, evitando esforços desnecessários.
- ***Describe concepts of the iStar extension*** (Descrever conceitos da extensão do *iStar*): Uma vez identificada a necessidade de extensão, a próxima fase é descrever detalhadamente os novos conceitos que a extensão deve introduzir. Isso inclui a definição de novos construtores, como atores, tarefas, objetivos e recursos, e como esses elementos se integrarão com os existentes na linguagem *iStar*. A descrição deve ser clara e completa, abrangendo todos os aspectos dos novos conceitos para garantir uma compreensão comum entre todos os envolvidos no desenvolvimento da extensão.
- ***Develop the iStar extension*** (Desenvolver a extensão do *iStar*): Com os conceitos bem definidos, a próxima fase é a implementação da extensão. Esta fase envolve a codificação dos novos elementos no *framework iStar* e a atualização de quaisquer ferramentas de suporte que possam ser necessárias. A implementação deve seguir as diretrizes da comunidade *iStar* para assegurar a compatibilidade e a integração adequada dos novos construtores. Durante esta fase, podem surgir questões técnicas que precisam ser resolvidas para garantir que a extensão funcione corretamente dentro do *framework iStar* existente. A implementação também deve ser acompanhada por testes iniciais para garantir que os novos elementos estejam funcionando conforme o esperado.
- ***Validate and evaluate the iStar extension*** (Validar e avaliar a extensão do *iStar*): Após a implementação, a extensão precisa ser rigorosamente validada e avaliada. Esta fase envolve a realização de estudos de caso e testes de funcionalidade para verificar se a

extensão atende aos objetivos definidos e resolve os problemas identificados inicialmente. O *feedback* dos usuários é coletado nesta fase e usado para realizar ajustes e melhorias na extensão. A validação deve incluir tanto testes técnicos quanto avaliações qualitativas para garantir que a extensão é útil e eficaz. O processo de validação é crucial para identificar quaisquer problemas ou deficiências antes que a extensão seja amplamente adotada.

- ***Check other new constructs to be introduced*** (Verificar outros novos construtores a serem introduzidos): Durante o desenvolvimento e a validação da extensão, pode ser que novos construtores sejam identificados como necessários. Esta fase visa revisar e verificar a necessidade de introdução desses novos elementos. Se forem considerados necessários, eles são descritos, desenvolvidos e validados seguindo o mesmo processo das fases anteriores. Esta fase garante que o processo de extensão é dinâmico e adaptável, permitindo a inclusão de novos conceitos conforme surgem novas necessidades e desafios.
- ***Publicise the iStar extension*** (Publicar a extensão do *iStar*): Na fase final, a extensão validada e avaliada é documentada e publicada. A documentação deve incluir uma descrição detalhada da extensão, exemplos de uso, benefícios, limitações e qualquer *feedback* relevante recebido durante a fase de validação. Além disso, o catálogo de extensões *iStar* [2] deve ser atualizado com a nova extensão. Esta documentação é então disponibilizada para a comunidade de usuários *iStar*, permitindo a adoção e utilização da nova extensão. Publicar a extensão de maneira abrangente e acessível é crucial para garantir que a comunidade iStar possa tirar proveito das novas funcionalidades e que a extensão seja amplamente adotada.

Para facilitar e otimizar o processo descrito pelo PRISE, foi desenvolvido o PriseBot [3], um *chatbot* inovador que auxilia no desenvolvimento de extensões para a linguagem *iStar*. O PriseBot oferece suporte automatizado aos desenvolvedores, guiando-os através das várias fases do PRISE e assegurando que as melhores práticas e diretrizes da comunidade *iStar* sejam seguidas (Freire *et al.* 2024).

A ferramenta *piStar-ext* [4], por sua vez, é a principal plataforma utilizada para realizar as modelagens no *iStar*, desempenhando um papel fundamental na aplicação prática das extensões desenvolvidas. Enquanto o PriseBot auxilia no entendimento e execução do processo PRISE, esclarecendo conceitos e artefatos específicos, o *piStar-ext* permite que os conceitos e

[2] https://istarextensions.cin.ufpe.br/catalogue/ Acesso em: 07 de setembro 2024
[3] http://prisebot.com/ Acesso em: 01 de dezembro 2024
[4] https://www.cin.ufpe.br/~ler/pistar-ext/ Acesso em: 27 de outubro 2024

elementos definidos sejam concretizados em diagramas visuais, tornando possível a validação das extensões no *framework iStar*.

A Figura 6 ilustra exemplos de interações do PriseBot, onde ele fornece explicações detalhadas sobre o processo PRISE e sobre artefatos específicos, como a "Lista de conceitos que serão introduzidos".

Figura 6 – PriseBot explicando o processo PRISE e artefatos relacionados

Fonte: (Freire *et al.* 2024)

Essa lista é um artefato essencial gerado no subprocesso 5 do PRISE, nomeado de *Check other new constructs to be introduced.* Ela registra todos os novos conceitos a serem introduzidos na extensão e é atualizada sempre que novos construtores precisam ser adicionados. O PriseBot detalha a importância dessa lista, explicando que ela é crucial para documentar todos os novos conceitos que serão incorporados, garantindo que a extensão esteja bem definida e completa.

O auxílio do PriseBot com o PRISE exemplifica como ferramentas automatizadas podem aprimorar processos complexos, promovendo eficiência e consistência no desenvolvimento de extensões de *iStar*.

3 TRABALHOS RELACIONADOS

Este capítulo descreve os trabalhos identificados na literatura que estarão relacionados ao presente trabalho. Foram identificados três (3) trabalhos, a citar:

- **TR1: Modelling and measuring code smells in enterprise applications using TISM and two-way assessment** (Gupta *et al.* 2016);
- **TR2: Management of quality requirements in agile and rapid software development: A systematic mapping study** (Behutiye *et al.* 2020);
- **TR3: Hybrid analytic hierarchy process-based quantitative satisfaction propagation in goal-oriented requirements engineering through sensitivity analysis** (Sumesh e Krishna 2020)

3.1 TR1: Modelling and measuring code smells in enterprise applications using TISM and two-way assessment

No estudo realizado por (Gupta *et al.* 2016) é proposto um *framework* para a modelagem e medição de *code smells* em aplicações empresariais, utilizando o *Total Interpretive Structural Modelling* (TISM). O trabalho se concentra em identificar e controlar os *code smells* nas fases de *design* e desenvolvimento, a fim de melhorar a manutenibilidade e a qualidade do código.

O TISM é utilizado para modelar as inter-relações entre diferentes *code smells*, oferecendo uma abordagem interpretativa que permite entender as dependências e influências entre eles. Esse processo permite que os *code smells* sejam estruturados de maneira hierárquica, revelando como cada um impacta a manutenibilidade do sistema.

Ao estruturar os *code smells* de maneira interpretativa, o *framework* proposto ajuda a identificar aqueles que possuem maior potencial de deteriorar a qualidade do código ao longo do tempo, facilitando a tomada de decisões no processo de refatoração. A análise *Matrice d'Impacts Croisés Multiplication Appliquée à un Classement* (MICMAC) complementa essa abordagem ao classificar os *code smells* em *clusters*, com base no poder de condução (*driving power*) e no grau de dependência de cada um.

A combinação do TISM com a análise MICMAC fornece um mapa visual e quantitativo das influências e dependências entre os *smells*, o que aumenta a precisão e eficiência na otimização da manutenibilidade.

A diferença fundamental entre o trabalho de (Gupta *et al.* 2016) e o presente estudo é que o primeiro foca na modelagem e mensuração de *code smells* para aprimorar a manutenibilidade de código em aplicações empresariais, enquanto este trabalho propõe uma extensão da linguagem *iStar* para a modelagem de requisitos que previnam *code smells*, buscando uma abordagem sistemática desde a fase de ER para evitar futuros problemas de manutenibilidade no código.

3.2 TR2: Management of quality requirements in agile and rapid software development: A systematic mapping study

Em (Behutiye *et al.* 2020), é apresentada uma análise sistemática da gestão de requisitos de qualidade em ambientes de desenvolvimento ágil e rápido. O trabalho foca na importância de gerenciar eficazmente os requisitos de qualidade em contextos onde a rapidez na entrega de software é fundamental, como em metodologias ágeis e práticas de desenvolvimento contínuo.

A pesquisa identifica e classifica estratégias, práticas e desafios enfrentados na gestão de requisitos de qualidade, destacando como esses requisitos, se não forem devidamente tratados, podem comprometer a qualidade final do software.

Para estruturar a análise, os autores realizaram um mapeamento sistemático da literatura, identificando 156 estudos primários que abordam a gestão de requisitos de qualidade em ambientes de desenvolvimento ágil e rápido. Dentre as práticas e estratégias identificadas, destacam-se as abordagens para garantir manutenibilidade e a eficiência, que foram os tipos de requisitos de qualidade mais frequentemente mencionados. O estudo também revela que a falta de atenção adequada aos requisitos de qualidade durante as fases iniciais de desenvolvimento pode levar à criação de dívida técnica, impactando negativamente a estabilidade e a manutenção do software.

A pesquisa realizada por (Behutiye *et al.* 2020) é relevante para o campo da ER, pois fornece uma visão abrangente sobre os desafios e práticas na gestão de qualidade de requisitos em contextos de desenvolvimento acelerado.

Este trabalho difere do estudo de (Behutiye *et al.* 2020) no que diz respeito ao foco e à abordagem. Enquanto o estudo de (Behutiye *et al.* 2020) explora estratégias e desafios na gestão de qualidade de software em ambientes ágeis e rápidos, o presente trabalho concentra-se na formalização e prevenção de *code smells* através da modelagem de requisitos específicos no

iStar, buscando garantir a qualidade do código desde a concepção do sistema.

3.3 TR3: Hybrid analytic hierarchy process-based quantitative satisfaction propagation in goal-oriented requirements engineering through sensitivity analysis

Em (Sumesh e Krishna 2020) é proposto um método híbrido para análise quantitativa de satisfação em ER utilizando o GORE. O trabalho foca em melhorar o processo de tomada de decisão durante a seleção de alternativas de requisitos de qualidade.

A proposta combina o *Analytic Hierarchy Process* (AHP), uma técnica utilizada para hierarquizar e priorizar alternativas com base em critérios específicos, com a análise de metas no *framework iStar*, permitindo uma avaliação quantitativa das alternativas. O AHP atribui pesos específicos a cada alternativa, com base na importância relativa dos *softgoals*, e esses pesos são então propagados ao longo da hierarquia de objetivos dentro do *iStar*, facilitando a comparação entre as alternativas.

Além disso, o método permite que os objetivos de qualidade sejam priorizados de forma mais eficaz, considerando as interdependências entre os diferentes atores do sistema modelado no *iStar*. Esses objetivos de qualidade representam os critérios que o sistema precisa alcançar para atender aos requisitos esperados, permitindo uma análise detalhada de como cada alternativa contribui para a satisfação desses objetivos. Isso proporciona uma visão clara de como cada alternativa contribui para a satisfação dos objetivos de qualidade, permitindo uma análise mais profunda das interações entre atores e requisitos.

A diferença fundamental entre o estudo de (Sumesh e Krishna 2020) e o presente estudo é que o primeiro concentra-se na avaliação quantitativa de alternativas para atingir metas de qualidade usando AHP e *iStar*, enquanto que o presente trabalho busca estender o *framework iStar* para modelar requisitos orientados à prevenção *code smells*.

3.4 Comparação entre os trabalhos relacionados

Uma comparação entre os trabalhos é apresentada no Quadro 1. As colunas do quadro comparam os seguintes critérios: **Suporte para linguagem de modelagem**, que se refere à utilização de alguma linguagem de modelagem para auxiliar o processo de ER; **Utiliza o *framework iStar* para modelagem de requisitos**, que avalia se o trabalho faz uso do *framework iStar* para a modelagem de requisitos em seus processos; **Trata de requisitos orientada à**

prevenção de *code smells*, que considera se o trabalho aborda diretamente requisitos que previnam a ocorrência de *code smells* no desenvolvimento de software; e, por fim, **Aplicável a áreas diversas**, que avalia a flexibilidade do trabalho em relação à sua aplicabilidade em diferentes domínios e áreas de software, indicando se a abordagem proposta pode ser utilizada em variados contextos, além de sua área de estudo original.

Quadro 1 – Análise comparativa deste trabalho aos trabalhos relacionados

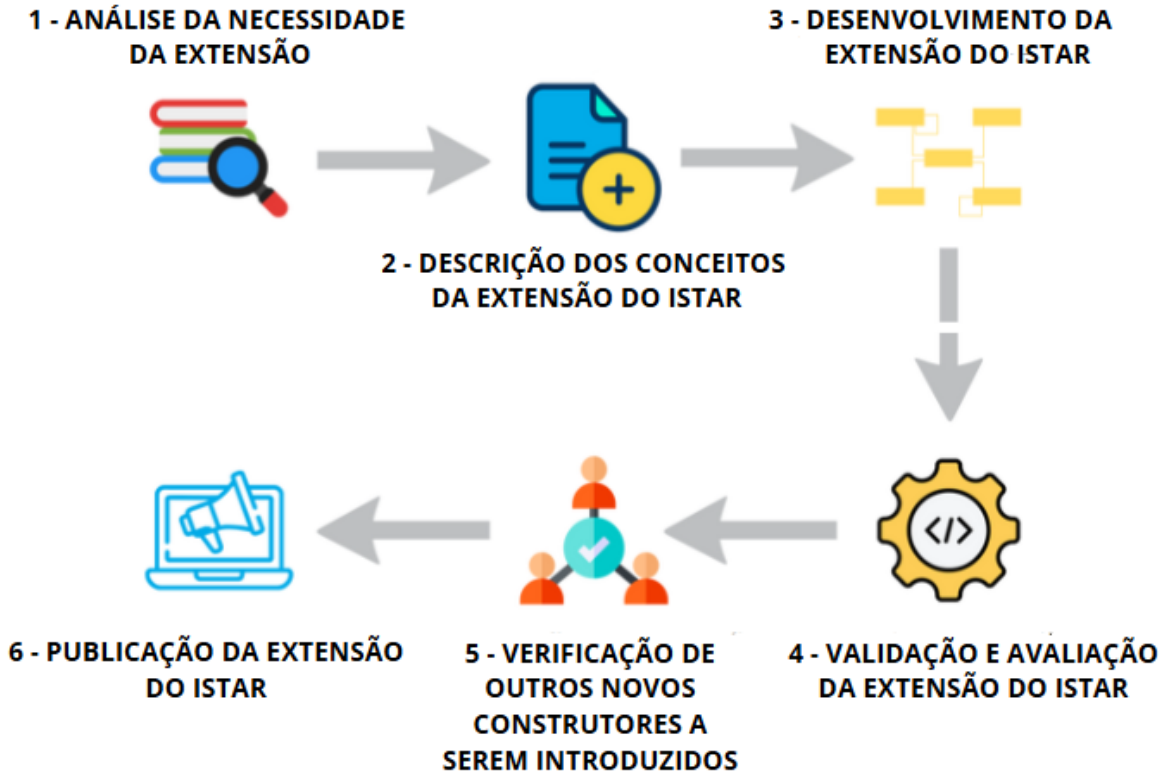
Trabalho	**Suporte para linguagem de modelagem**	**Utiliza o *framework iStar* para modelagem de requisitos**	**Trata de requisitos orientada à prevenção de *code smells***	**Aplicável a áreas diversas**
TR1	Sim	Não	Não	Não
TR2	Não	Não	Sim	Sim
TR3	Sim	Sim	Sim	Não
Este trabalho	Sim	Sim	Sim	Sim

Fonte: Elaborado pelo autor.

4 METODOLOGIA

Para este trabalho, seguimos o processo PRISE (Goncalves *et al.* 2020). Este processo fornece uma abordagem estruturada para o desenvolvimento de extensões na linguagem *iStar*, utilizando uma série de subprocessos organizados. A seguir, descrevemos a aplicação de cada subprocesso no presente trabalho. Os resultados dessas etapas são discutidos na Seção 5.

Figura 7 – Metodologia usada no desenvolvimento da extensão



Fonte: Elaborado pelo autor

4.1 Análise da necessidade da extensão

Nesta etapa, conduzimos uma análise abrangente da literatura para entender as lacunas existentes na elicitação e modelagem de requisitos que previnam *code smells*. Os principais trabalhos relacionados estão descritos na Seção 3.

Os conceitos da extensão serão definidos e refinados ao longo do seu próprio processo de desenvolvimento. Iniciamos com uma revisão detalhada da literatura para identificar os *code smells* a níveis de requisitos que necessitam de representação. Posteriormente, realizamos diversas iterações de modelagem para testar e aprimorar a forma como esses conceitos podem

ser incorporados na linguagem *iStar*.

Estabelecemos diálogo contínuo e marcamos reuniões semanais com o orientador deste trabalho, especialista em extensões *iStar* para aprofundar nossa compreensão das necessidades do setor. Esses encontros proporcionaram *insights* valiosos que foram fundamentais para orientar o desenvolvimento da extensão proposta.

Utilizamos a ferramenta *piStar-ext* para realizar a modelagem dos conceitos identificados. Investigamos a existência de extensões que abordassem conceitos similares aos propostos. Realizamos buscas no repositório oficial de extensões *iStar* e conduzimos uma revisão da literatura para identificar extensões *iStar* relacionadas a *code smells*.

Geramos a especificação da extensão, que inclui os resultados da pesquisa, uma lista de referências, pesquisadores consultados, os conceitos a serem introduzidos, além de modelagens e observações pertinentes. Essa especificação servirá como um guia detalhado para a implementação e aplicação da extensão, facilitando sua adoção por outros pesquisadores e profissionais da área.

4.2 Descrição dos conceitos da extensão do *iStar*

Inicialmente, conduzimos pesquisas para identificar construtores existentes que possam ser reutilizados na extensão proposta. Essa tarefa envolverá a pesquisa de extensões e construtores relacionados a *code smells* no catálogo de extensões *iStar* (Gonçalves *et al.* 2018).

Em seguida, os conceitos centrais da extensão foram descritos detalhadamente. Cada conceito identificado foi documentado, junto com os possíveis construtores reutilizáveis, para garantir que todos os elementos necessários estejam claramente definidos e integrados ao *framework iStar*.

Após a definição dos conceitos, realizamos uma análise para integrar os novos construtores da extensão com aqueles já presentes no *iStar*. Essa etapa inclui experimentações com diferentes combinações de construtores e a aplicação de estereótipos ou outras técnicas para especializar os elementos já existentes na linguagem *iStar*.

Consultas regulares com o orientador deste trabalho, especialista em *iStar*, também foram realizadas durante todo o processo, visando identificar e mitigar possíveis problemas na integração dos novos conceitos.

Finalmente, todos os artefatos gerados durante este subprocesso foram adicionados ao documento de especificação detalhada da extensão.

4.3 Desenvolvimento da extensão do *iStar*

No desenvolvimento da extensão *iStar*, a primeira etapa consiste na definição do metamodelo da extensão, que foi criado com base no metamodelo original do *iStar*. Este metamodelo servirá como a estrutura principal que guiará a incorporação dos novos conceitos relacionados a *smells* dentro do *framework iStar*.

Após a criação do metamodelo, foram estabelecidas regras de validação que expressam as restrições que não podem ser capturadas diretamente pelo metamodelo. Essas regras foram detalhadas e documentadas de maneira precisa, garantindo que qualquer inconsistência ou problema de integridade seja identificado e corrigido durante o processo de modelagem.

Em seguida, foi desenvolvida a sintaxe concreta da extensão, onde serão criados exemplos visuais que demonstram como os novos conceitos introduzidos pela extensão serão representados na modelagem. Além disso, realizamos uma verificação minuciosa para assegurar a completude, consistência e ausência de conflitos na extensão.

Durante todo o processo, o desenvolvimento da extensão utilizou o PriseBot para auxiliar nas diversas etapas, assegurando que as tarefas sejam realizadas de maneira eficiente e consistente.

Após a definição e validação dos conceitos e regras, a extensão foi implementada na ferramenta online *piStar-ext*.

Finalmente, todos os artefatos gerados durante este processo foram anexados ao documento de especificação detalhada da extensão. Essa especificação inclui o metamodelo, as regras de validação, a lista de representação da sintaxe concreta, *checklist* para verificação de problemas e *link* para *download* da ferramenta de modelagem.

4.4 Validação e avaliação da extensão do *iStar*

A extensão desenvolvida foi implementada em um ambiente de teste para modelar um sistema representativo, o que nos permitiu identificar ajustes e melhorias necessárias na modelagem de requisitos que previnem *code smells*.

Realizamos sessões de *feedback* com o orientador deste trabalho, especialista em *iStar*, para discutir o desempenho da extensão. Essas interações foram fundamentais para detectar oportunidades de melhoria e refinar os construtores utilizados.

Compilamos uma documentação abrangente que descreve a extensão em detalhes,

incorporando as melhorias sugeridas e as alterações implementadas. Isso garante que a extensão está pronta para a aplicação em projetos futuros, com uma base sólida de evidências de sua eficácia.

Além disso, para obter uma avaliação mais ampla da extensão, foi aplicado um questionário direcionado a pesquisadores e profissionais com experiência em *iStar* e *code smells*. O questionário buscou avaliar aspectos como a clareza dos conceitos introduzidos, a facilidade de modelagem e a aplicabilidade da extensão em diferentes domínios.

Os resultados dessa avaliação forneceram *insights* valiosos sobre a utilidade da abordagem proposta e oportunidades para refinamentos futuros.

4.5 Verificação de outros novos construtores a serem introduzidos

Durante as etapas de desenvolvimento, foi mantido um processo contínuo de avaliação para identificar a necessidade de novos construtores que poderiam ser integrados à extensão.

4.6 Publicação da extensão do *iStar*

Após a validação da extensão, ela foi submetida ao catálogo de extensões iStar, garantindo sua disponibilidade para a comunidade e promovendo seu uso em projetos futuros.

5 RESULTADOS

Neste capítulo, são expostos os resultados alcançados ao longo deste estudo.

5.1 Análise da necessidade da extensão

A representação de *code smells* a níveis de requisitos desempenha um papel essencial na garantia da qualidade e manutenibilidade dos sistemas desde as fases iniciais do desenvolvimento. Ao definir e modelar corretamente os requisitos para a prevenção de *code smells*, é possível evitar problemas como ambiguidade, complexidade excessiva e dependências inadequadas, que poderiam comprometer o sucesso do projeto e aumentar significativamente os custos e o tempo de desenvolvimento.

Esses problemas estão frequentemente relacionados a requisitos de qualidade como manutenibilidade e eficiência, já que *code smells* podem impactar a clareza e a organização do código, dificultando sua evolução ao longo do tempo e, em alguns casos, limitando o uso eficiente de recursos.

Apesar da relevância desse tema, as extensões propostas para a linguagem *iStar* até o momento não abordam diretamente a questão dos requisitos voltados para a prevenção de *code smells*. Em (Gonçalves *et al.* 2018), foram analisadas 96 extensões da linguagem *iStar*, e nenhuma delas foi categorizada como uma extensão dedicada à representação de requisitos para a prevenção de *code smells*.

Além disso, uma busca minuciosa realizada no repositório oficial de extensões *iStar* (iStar Repository 2016) confirmou a ausência de extensões que tratem diretamente da modelagem de requisitos voltados para a prevenção de *code smells*, evidenciando uma lacuna significativa na literatura e na prática de desenvolvimento de sistemas.

Além das extensões, foram também analisados diversos estudos de caso e exemplos práticos disponíveis na literatura, que reforçaram a ausência de abordagens estruturadas para a representação de *smells* a níveis de requisitos.

Estudos como o de (Boehm e Basili 2007) sobre práticas de redução de defeitos e o trabalho de (Kitchenham *et al.* 1995) sobre medição de qualidade desde as fases iniciais de desenvolvimento destacam métodos para identificar e mitigar problemas de qualidade desde a fase de requisitos. No entanto, essas abordagens ainda não tratam diretamente da prevenção de *code smells* a nível de requisitos, evidenciando uma lacuna significativa na literatura para

práticas específicas de prevenção nesse estágio.

5.2 Descrição dos conceitos da extensão

Nesta etapa, são descritos os conceitos a serem adicionados à linguagem *iStar* por meio da extensão proposta. Como mencionado na seção 5.1, não foram encontradas extensões voltadas especificamente para a área de prevenção de *code smells*. Diante disso, foi realizada uma análise dos construtores existentes na linguagem original, buscando maneiras de integrar os novos conceitos de forma coerente e alinhada aos princípios do *iStar*. O Quadro 2 apresenta os conceitos definidos para a extensão e suas respectivas descrições.

Quadro 2 – Descrição dos conceitos da extensão

Etapas	**Descrição**
Quality Smell	Refere-se à ausência de *smells* e à aplicação de boas práticas na modelagem de requisitos.
Task Smell	Representa uma tarefa que pode estar associada a um *code smell*, indicando possíveis problemas na forma como determinada funcionalidade é modelada.
Smell Metric	Define métricas utilizadas para quantificar e monitorar a presença de *smells*, auxiliando na avaliação da qualidade da modelagem de requisitos.

Fonte: Elaborado pelo autor.

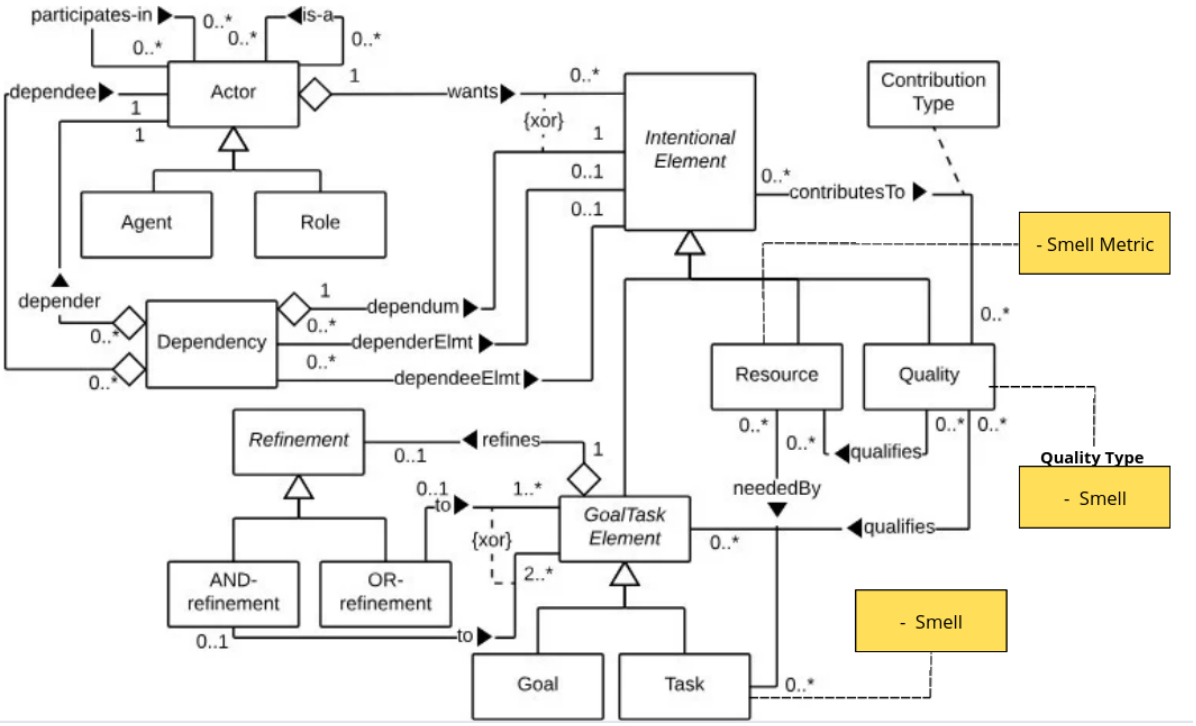
5.3 Desenvolvimento da extensão

Após a identificação dos conceitos, o próximo passo concentra-se no desenvolvimento da extensão, que é dividido em dois subprocessos principais: a construção do metamodelo, responsável por descrever a sintaxe abstrata e estabelecer as regras da extensão proposta; e a definição da sintaxe concreta (ou gráfica), que apresenta os elementos da extensão e como serão representados visualmente. Para garantir consistência e evitar problemas de incompatibilidade ou incoerência, foi realizada uma verificação desses aspectos durante o processo. Como resultado, essa etapa documenta a especificação e o desenvolvimento completo da extensão.

5.3.1 Definição do Metamodelo da extensão

Os conceitos identificados na lista de conceitos foram incorporados ao metamodelo do *iStar*. O resultado pode ser visualizado na Figura 8. As novas metaclasses introduzidas pela extensão estão destacadas na cor amarela.

Figura 8 – Metamodelo da extensão



Fonte: Elaborado pelo autor

Os novos conceitos introduzidos nesta extensão são derivados de metaclasses existentes na linguagem *iStar* e são representados por estereótipos textuais.

O conceito *Quality Smell* é uma especialização de Qualidade, representando aspectos que podem comprometer a qualidade do sistema desde a fase de requisitos. Já o *Smell Metric* herda de Recurso, funcionando como um suporte analítico para a avaliação dos *smells*, permitindo o monitoramento e a quantificação da presença de problemas na modelagem. Da mesma forma, o *Task Smell* especializa Tarefa, indicando atividades que podem estar associadas a *code smells* e necessitam de refinamento para evitar impactos negativos no sistema.

Todas as novas metaclasses preservam as relações e características dos elementos originais dos quais derivam.

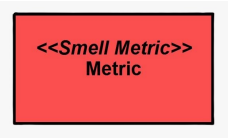
Nas próximas seções, será detalhado como esses conceitos são representados graficamente e como podem ser utilizados na modelagem de requisitos voltados à prevenção de *code smells*.

5.3.2 Representação Gráfica dos Conceitos da Extensão

Nesta seção, são apresentadas as representações visuais dos conceitos definidos pela extensão, conforme descrito na Seção 5.2. O Quadro 3 exibe cada conceito, sua representação

gráfica, uma breve descrição e indica se o construtor foi reutilizado ou não.

Quadro 3 – Lista da sintaxe concreta dos conceitos da extensão

Nome do Conceito	Representação Gráfica	Descrição	Reutilizado?
Quality Smell		Representa uma qualidade associada à prevenção de *smells*. Esse deve ter um tipo de *smell* definido.	Não
Task Smell		É uma tarefa que pode estar associada a um *code smell*, ou seja, uma atividade no sistema que pode apresentar problemas estruturais.	Não
Smell Metric		Representa um elemento analítico, usado para definir métricas quantitativas que ajudam a medir a presença e o impacto dos *code smells* na modelagem.	Não

Fonte: Elaborado pelo autor.

5.3.3 Utilização da extensão

Para utilizar a extensão proposta, na representação *Strategic Dependency* (SD) do *iStar*, que destaca os atores e as dependências entre eles, é necessário criar um modelo que represente os agentes, atores e papéis envolvidos, bem como as relações de dependência entre eles. Pelo menos um agente associado ao conceito *Smell* deve ser incluído no modelo, representando sistemas ou subsistemas onde problemas de *code smells* foram identificados ou mitigados.

Na representação *Strategic Rationale* (SR) do *iStar*, que complementa a SD ao detalhar os elementos intencionais e as estratégias adotadas pelos atores, o agente associado a um *Smell* deve ter algum elemento relacionado a uma instância de *Quality Smells*. Esse conceito deve conter o estereótipo "«Smell»", e o tipo específico de *smell*, como por exemplo, o *Long Method*.

O conceito *Quality Smell* pode ser conectado a diversos elementos intencionais, como *Task*, *Goal*, *Resource* ou *Quality*, de forma similar à base da linguagem *iStar*. Para documentar as informações associadas a cada *Quality Smells*, recomenda-se o uso de um quadro que inclua os seguintes campos: identificação do elemento *iStar* relacionado, a relação com requisitos

elicitados previamente, as principais características do *smell* e a solução ou técnica aplicada para resolver ou mitigar o problema. O Quadro 4 apresenta um modelo para essa documentação.

Quadro 4 – Modelo de quadro relacionado à Quality Smell

Identificação do elemento *iStar*	[Tipo de *Smell*]
Relação com requisitos	[Lista de requisitos relacionados]
Principais problemas associados	[Descrição dos problemas associados ao *smell*]
Tipo de *Smell*	**Solução ou Refatoração Aplicada**
[Tipo de *Smell*]	[Solução ou técnica utilizada]

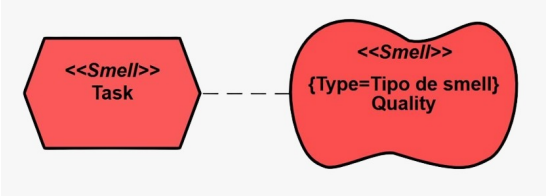
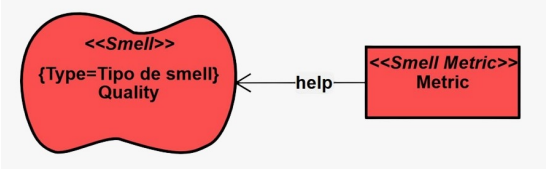
Fonte: Elaborado pelo autor.

Além disso, o conceito *Smell Metric* pode estar relacionado a uma instância de *Quality Smell*, indicando que métricas específicas são utilizadas para monitorar e prevenir *smells*.

No caso de *Task Smell*, essas tarefas podem estar associadas a um *Quality Smell*, indicando que determinadas atividades no sistema apresentam possíveis problemas estruturais que podem afetar a qualidade do software.

O Quadro 5 ilustra exemplos das interações entre os elementos introduzidos pela extensão e os elementos da linguagem base.

Quadro 5 – Exemplo das relações entre elementos da extensão

Descrição	**Representação Gráfica**
Relação entre *Task Smell* e *Quality Smell*: Representa a relação entre um *Task Smell* e um *Quality Smell*, indicando que uma tarefa pode estar associada a um *smell* que afeta a qualidade do sistema. No exemplo, a **Tarefa 1** está vinculada a um *Quality Smell*, que especifica um tipo de *smell* identificado	
Relação entre *Quality Smell* e *Smell Metric*: Demonstra como um *Smell Metric* pode auxiliar na avaliação e mitigação de um *Quality Smell*, fornecendo indicadores quantitativos para monitorar a presença e o impacto de *smells* na modelagem de requisitos.	

Fonte: Elaborado pelo autor.

5.3.4 Como implementar a extensão na ferramenta piStar-ext

Nesta seção, apresenta-se o processo de utilização do *piStar-ext* para modelagem baseada na extensão desenvolvida.

Para iniciar a modelagem, a ferramenta permite a criação de grupos de construtores, facilitando a organização dos estereótipos utilizados na extensão. Esse recurso é útil para a modelagem de requisitos para a prevenção de *code smells*, pois possibilita agrupar os conceitos introduzidos de forma estruturada. Na barra lateral do *piStar-ext*, deve-se selecionar a aba "*Grouper*", definir um nome para o grupo de construtores e, em seguida, selecionar os elementos desejados segurando a tecla "*Ctrl*".

Após a criação do grupo, é necessário cadastrar os estereótipos na aba "*Stereotype*". Para manter a organização da modelagem, os estereótipos devem ser configurados conforme os construtores definidos na extensão. O estereótipo «*Smell*» deve ser utilizado para representar tanto um sistema ou subsistema no qual *code smells* foram identificados quanto elementos específicos, como *Task Smell* e *Smell Metric*, unificando a representação e permitindo uma modelagem mais coesa e organizada.

Opcionalmente, é possível personalizar a cor dos elementos da extensão por meio da aba "*Style*", destacando os *code smells* e as tarefas associadas à sua mitigação no modelo. Com essas configurações, a modelagem utilizando a extensão se torna mais intuitiva e organizada, possibilitando uma representação clara dos requisitos voltados à prevenção de *code smells* e suas estratégias de mitigação dentro do processo da ER.

5.3.5 Verificação de Conflitos, Consistência e Completude

Nesta seção, é realizada uma análise da completude, avaliando se todos os conceitos da extensão estão devidamente definidos em seus três níveis: conceitual, metamodelo e sintaxe concreta. Também é verificada a consistência, assegurando que os novos construtores foram representados corretamente nesses níveis. Além disso, examinam-se possíveis conflitos com os elementos já existentes no *iStar*, garantindo que a extensão mantenha compatibilidade com a linguagem original.

A análise realizada demonstra que a extensão é completa e consistente, uma vez que todos os conceitos foram incorporados nos três níveis de modelagem. Como a extensão segue uma abordagem conservativa, a estrutura base do *iStar* foi preservada, garantindo que os

construtores originais permaneçam inalterados e possam coexistir sem conflitos com os novos elementos introduzidos.

O Quadro 6 apresenta os resultados dessa verificação, demonstrando a presença dos conceitos introduzidos e sua relação com os construtores da linguagem *iStar*.

Quadro 6 – Conflitos, Consistência e Completude da Extensão

Completude				
[X] Definição dos conceitos [X] Metamodelo [X] Regras de boa formação (opcional) [X] Sintaxe Concreta				
Consistência				
Conceito	**Descrição**	**Metamodelo**	**Sintaxe concreta**	**Conflitos**
Quality Smell	[X]	[X]	[X]	0
Task Smell	[X]	[X]	[X]	0
Smell Metric	[X]	[X]	[X]	0

Presença de construtores e relações (*links*) da sintaxe *iStar*		
Conceito	**Metamodelo**	**Sintaxe concreta**
Goal	[X]	[X]
Quality	[X]	[X]
Resource	[X]	[X]
Task	[X]	[X]
Actor	[X]	[X]
Agent	[X]	[X]
Role	[X]	[X]
Refinement	[X]	[X]
Qualification	[X]	[X]
Contribution (Make, Help, Hurt, Break)	[X]	[X]
NeededBy	[X]	[X]
Dependency	[X]	[X]
Association Links (is a, participates in)	[X]	[X]

Fonte: Elaborado pelo autor.

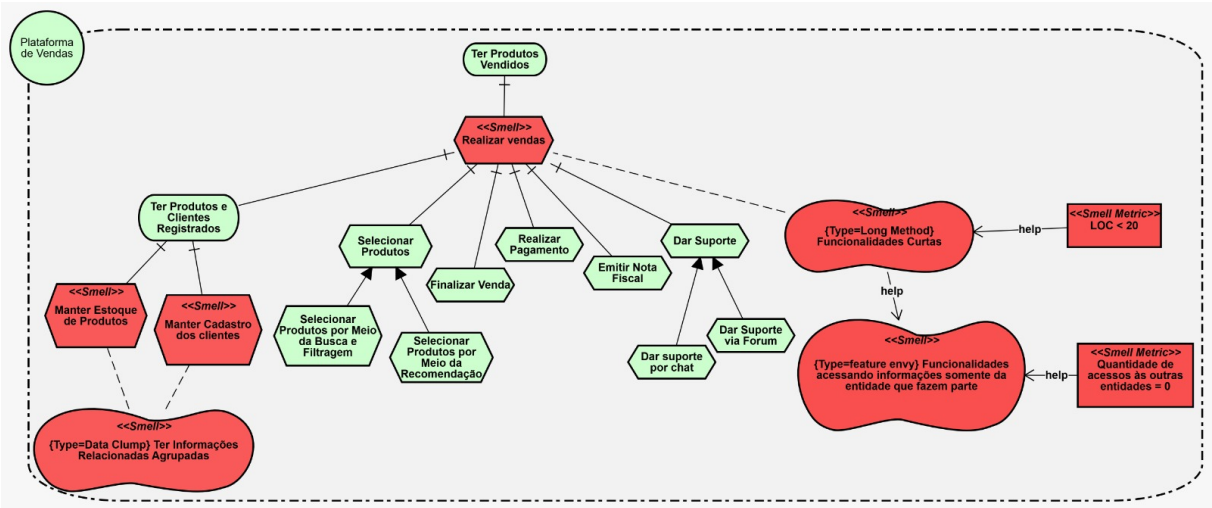
5.4 Exemplificação da Utilização dos Construtores da Extensão

Para demonstrar a aplicabilidade da extensão e sua capacidade de representar os conceitos propostos, foi desenvolvida uma modelagem que simula um sistema de plataforma de vendas. Essa modelagem serve como uma prova de conceito, ilustrando como os novos construtores podem ser utilizados na prática dentro do processo de ER, possibilitando a visualização clara das interações entre os elementos e a identificação de possíveis melhorias estruturais

no sistema. Além disso, a modelagem permite verificar como os construtores introduzidos auxiliam na detecção e mitigação de *code smells*, garantindo maior qualidade na especificação dos requisitos e facilitando futuras refatorações.

A Figura 9 apresenta a representação gerada a partir da extensão, destacando os diferentes construtores utilizados e suas relações dentro da plataforma modelada.

Figura 9 – Modelagem de exemplo



Fonte: Elaborado pelo autor

A modelagem apresentada aplica a extensão do *iStar* para representar requisitos relacionados à prevenção de *code smells* em uma plataforma de vendas. O diagrama mostra como as tarefas do sistema podem estar associadas a possíveis problemas estruturais e como métricas quantitativas podem ser utilizadas para monitorar e mitigar esses problemas desde a fase de modelagem dos requisitos.

No centro do modelo, o *task smell* "realizar vendas"está destacada como um ponto crítico e foi identificada como uma potencial fonte de *smell*. A partir dela, várias subtarefas são derivadas, incluindo "Selecionar Produtos", "Realizar Pagamento", "Emitir Nota Fiscal" e "Dar Suporte". Algumas dessas tarefas foram identificadas como associadas a *code smells*, como "Manter Estoque de Produtos" e "Manter Cadastro dos Clientes", que estão vinculadas ao *Data Clump* como uma qualificação, indicando que informações relacionadas estão agrupadas de maneira inadequada. Esse *smell* pode impactar a manutenção do sistema, dificultando a separação e reutilização de dados.

Outro aspecto modelado é a presença do *Long Method*, associado à tarefa "realizar vendas". Esse problema indica que a funcionalidade pode se tornar excessivamente complexa, comprometendo a clareza e manutenibilidade do código. Para auxiliar na mitigação desse

problema, A modelagem propõe uma métrica que define um limite de linhas de código inferior a 20, permitindo que a equipe de desenvolvimento monitore esse aspecto ao longo do projeto. Esse limite foi estabelecido com base em critérios adotados na modelagem da extensão. Além disso, a modelagem representa uma relação entre a resolução desse problema e a introdução de um novo *code smell*. Sempre que um *Long Method* é resolvido, a tendência é que ele seja fragmentado em métodos menores, muitas vezes gerando *Feature Envy*, pois as novas funções criadas podem acessar apenas informações da entidade original sem interagir adequadamente com outras partes do sistema. Esse efeito colateral foi representado na modelagem, mostrando que a extração de métodos pode gerar novas dependências e que a mitigação de um *smell* pode levar ao surgimento de outro.

Além disso, a modelagem identifica a presença do *Feature Envy*, um *code smell* que ocorre quando funcionalidades acessam informações apenas dentro da entidade a que pertencem, sem interagir com outras partes do sistema. Esse problema foi identificado nas tarefas de suporte ao usuário, sugerindo um encapsulamento excessivo das informações. Para monitorar esse comportamento, foi estabelecida uma métrica que verifica se a quantidade de acessos a outras entidades é igual a zero, servindo como um indicador do isolamento inadequado da funcionalidade.

As relações entre os elementos do modelo reforçam a conexão entre tarefas, *code smells* e métricas. A tarefa "realizar vendas" está diretamente ligada a problemas estruturais que podem comprometer a qualidade do sistema, enquanto os *code smells* identificados possuem métricas associadas para quantificar sua ocorrência e impacto. Dessa forma, a modelagem permite uma análise detalhada dos requisitos, incorporando mecanismos de monitoramento e prevenção que podem ser utilizados desde as primeiras etapas do desenvolvimento.

O conceito de *Quality Smell* foi cuidadosamente associado aos elementos intencionais da linguagem *iStar*, possibilitando a identificação detalhada de problemas que podem impactar negativamente a qualidade do código durante o desenvolvimento do sistema. Essa abordagem permite que tais problemas sejam reconhecidos de maneira estruturada, facilitando sua mitigação ainda na fase de modelagem dos requisitos. Dentro desse contexto, três *Quality Smell* principais foram destacados como particularmente relevantes para a análise: *Long Method*, um problema que compromete significativamente tanto a compreensão quanto a manutenção do código ao torná-lo excessivamente extenso e difícil de gerenciar; e *Feature Envy* e *Data Clump*, que, por sua vez, evidenciam o uso inadequado e desorganizado de dados dentro da estrutura do

sistema, podendo resultar em um alto nível de acoplamento entre os componentes e dificultar a modularidade do software.

O Quadro 7 apresenta os tipos de *smells* e as técnicas utilizadas para mitigá-los.

Quadro 7 – Quadro de relação de construtores de *Quality Smells*

Identificação do elemento *iStar*	*Feature Envy*
Relação com requisitos	O sistema deve ter classes bem definidas
Principais problemas associados	Métodos que acessam mais dados de outras classes do que da própria classe
Tipo de Smell	**Solução ou Refatoração Aplicada**
Feature Envy	Mover dados para a classe mais apropriada
Identificação do elemento *iStar*	*Data Clump*
Relação com requisitos	Dados semelhantes devem ser agrupados
Principais problemas associados	Dados frequentemente utilizados juntos são mantidos separadamente, dificultando a coesão
Tipo de *Smell*	**Solução ou Refatoração Aplicada**
Data Clump	Agrupar os dados em uma estrutura comum
Identificação do elemento *iStar*	*Long Method*
Relação com requisitos	Os métodos do sistema devem ter no máximo 30 linhas
Principais problemas associados	Métodos longos dificultam a compreensão e reutilização do código
Tipo de *Smell*	**Solução ou Refatoração Aplicada**
Long Method	Refatorar métodos longos e quebrar o método em métodos menores e mais específicos

Fonte: Elaborado pelo autor.

5.5 Resultados da Avaliação da Extensão

Os seguintes tópicos descrevem a metodologia utilizada na avaliação e os resultados obtidos a partir da análise das respostas dos especialistas.

5.5.1 Metodologia de Avaliação da Extensão

Para avaliar a aplicabilidade e usabilidade da extensão proposta, foi elaborado um questionário estruturado, destinado a pesquisadores e profissionais com experiência em *iStar*,

extensões de *iStar* e *code smells*. O objetivo desse questionário foi coletar percepções sobre a clareza dos conceitos introduzidos pela extensão, sua integração ao *iStar*, sua facilidade de uso e sua contribuição para a prevenção de *code smells* na fase de requisitos.

Para garantir uma avaliação qualificada e representativa, o questionário foi enviado por *e-mail* a um grupo de 50 especialistas e permaneceu disponível para respostas entre os dias 17 e 27 de fevereiro de 2025. Desses, 26 especialistas participaram da pesquisa. Os respondentes foram selecionados a partir de uma amostragem criteriosa, contemplando tanto pesquisadores acadêmicos quanto profissionais com experiência prática na ER, na modelagem com *iStar* e na detecção de *code smells*. A seleção dos participantes foi realizada por meio de redes acadêmicas, eventos especializados da área e convites direcionados a grupos de pesquisa com atuação relevante no tema. Dessa forma, buscou-se garantir que os participantes tivessem conhecimento suficiente para fornecer avaliações embasadas e pertinentes à proposta da extensão.

Antes de responderem ao questionário, os participantes foram instruídos a assistir a um vídeo explicativo, elaborado com o propósito de apresentar, de forma clara e detalhada, os principais construtores da extensão proposta. Nesse vídeo, foram abordadas as representações gráficas dentro do modelo *iStar*, explicando como cada novo elemento se integra à notação original e de que maneira contribui para a prevenção de *code smells* na fase de requisitos. Além disso, foi demonstrado um exemplo prático de aplicação da extensão, permitindo que os participantes visualizassem sua utilização em um cenário realista. Essa etapa foi cuidadosamente planejada para assegurar que todos os especialistas tivessem um entendimento uniforme dos conceitos envolvidos e do funcionamento da extensão antes de realizarem sua avaliação. Dessa forma, buscou-se minimizar possíveis interferências que poderiam surgir devido a diferenças no nível de familiaridade dos participantes com a modelagem proposta, garantindo que as respostas fossem mais consistentes e refletissem uma análise objetiva da proposta apresentada.

A estrutura do questionário foi composta por afirmações quantitativas e qualitativas, incluindo uma seção inicial para caracterização do perfil dos participantes. Nessa etapa, foram coletadas informações sobre a formação acadêmica, a ocupação profissional ou vínculo com a pesquisa, a experiência no desenvolvimento de extensões de *iStar* e a atuação na prevenção de *code smells*, proporcionando um melhor contexto para a análise das respostas subsequentes.

As afirmações quantitativas foram formuladas em uma escala *Likert* de 1 a 7, na qual 1 representa "Discordo totalmente", 2 representa "Discordo parcialmente", 3 representa "Discordo", 4 representa "Neutro", 5 representa "Concordo parcialmente", 6 representa "Concordo"e 7

representa "Concordo totalmente", permitindo mensurar o nível de concordância dos participantes em relação aos critérios analisados.

Além disso, o questionário incluiu três (3) perguntas abertas, com o intuito de capturar percepções mais detalhadas sobre possíveis dificuldades na compreensão dos conceitos, sugestões de melhorias e a aplicabilidade prática da extensão em diferentes tipos de projetos.

O Quadro 8 apresenta os resultados das médias das avaliações obtidas para cada critério analisado.

Quadro 8 – Resultados da Avaliação da Extensão

Critério avaliado	**Média**
S1 - A extensão proposta facilita a modelagem de *code smells* no *iStar*	5.85
S2 - O uso da extensão melhora a clareza na identificação e mitigação de *code smells*	5.85
S3 - A extensão se integra bem aos construtores originais do *iStar*	5.85
S4 - Os conceitos introduzidos são fáceis de entender e aplicar	5.90
S5 - A representação gráfica dos construtores ajuda a visualizar corretamente os problemas de *code smells*	6.00
S6 - A ferramenta *piStar-ext* oferece suporte adequado para a modelagem utilizando a extensão	6.15
S7 - O uso da extensão pode beneficiar equipes de desenvolvimento ao mapear e prevenir *code smells*	5.92
S8 - A inclusão de *Smell Metric* contribui para uma melhor rastreabilidade dos problemas	5.92
S9 - A estrutura da extensão é flexível o suficiente para ser utilizada em diferentes domínios de software	6.15
S10 - Em projetos futuros, eu consideraria utilizar essa extensão na modelagem de requisitos	6.23
S11 - Comparado à modelagem tradicional no *iStar*, a extensão facilita a representação de *code smells*	6.31
S12 - A extensão cobre adequadamente os principais problemas relacionados a *code smells*	6.31

Fonte: Elaborado pelo autor.

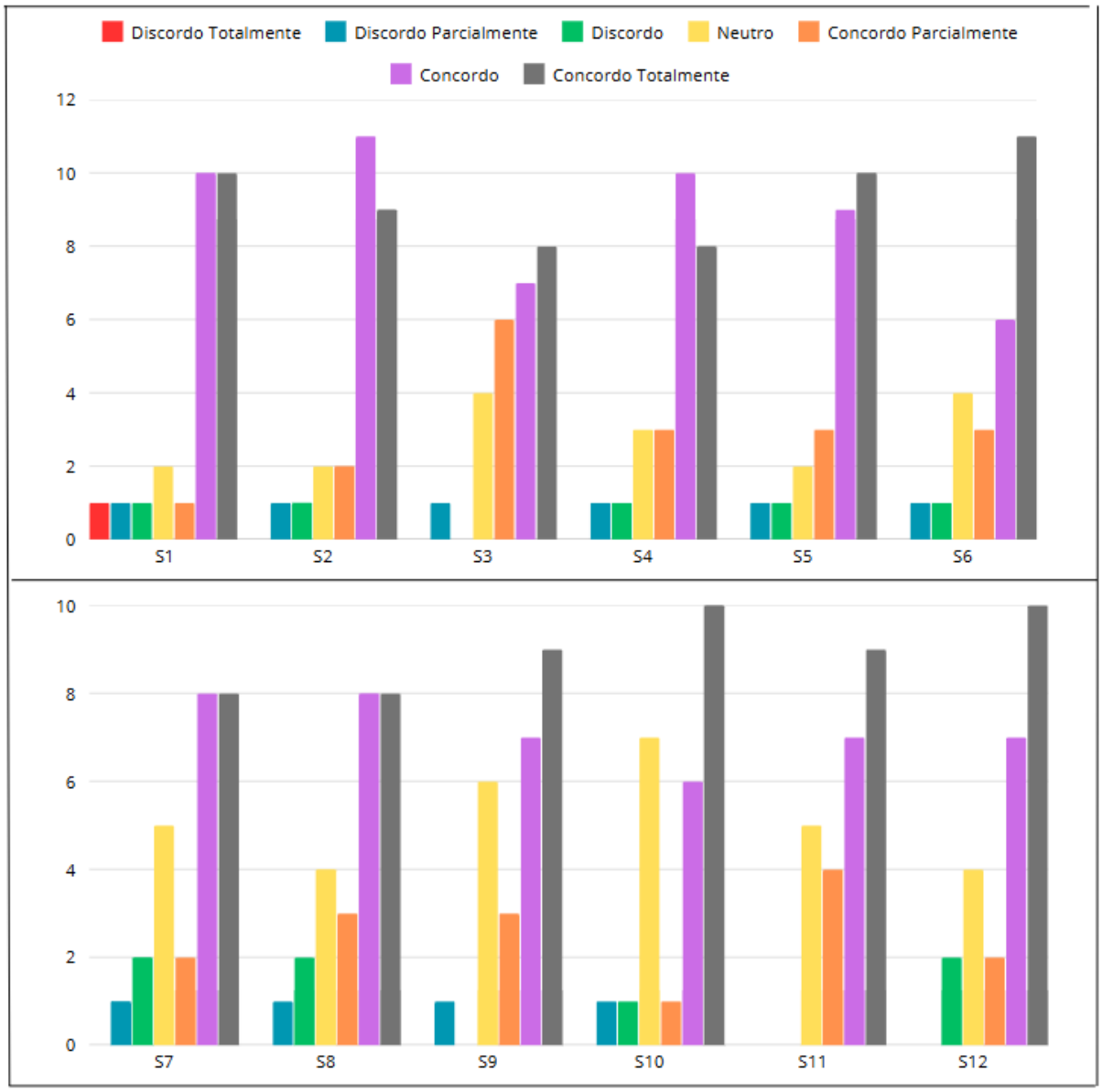
As afirmações do questionário foram elaboradas para avaliar a usabilidade, integração e benefícios da extensão no *iStar*. As questões iniciais (**S1** a **S3**) verificam se a extensão facilita a modelagem, melhora a clareza na identificação de *code smells* e se integra bem à notação original, garantindo sua coerência. As perguntas seguintes (**S4** a **S6**) analisam a compreensão e usabilidade dos novos conceitos, avaliando a clareza dos elementos, a eficácia da representação gráfica e o suporte da ferramenta *piStar-ext*.

Já as questões **S7** a **S9** investigam o impacto da extensão no desenvolvimento de software, verificando se auxilia equipes a mapear e prevenir *code smells*, se as métricas melhoram a rastreabilidade dos problemas e se a estrutura da extensão é flexível o suficiente para diferentes domínios. Por fim, as perguntas **S10** a **S12** medem a aceitação da extensão, analisando seu potencial de adoção em projetos futuros, sua vantagem sobre a modelagem tradicional e sua

cobertura dos principais problemas de *code smells*. Esse conjunto de perguntas permitiu uma avaliação ampla e estruturada da extensão, garantindo insights para futuras melhorias.

Na Figura 10, detalhamos a distribuição das respostas para cada afirmação, permitindo uma melhor análise das percepções dos participantes.

Figura 10 – Resultados detalhados da avaliação



Fonte: Elaborado pelo autor

5.5.2 Análise dos Resultados da Avaliação

A avaliação da extensão contou com a participação de 26 especialistas, distribuídos entre diferentes níveis de formação acadêmica. Entre os respondentes, 15 possuem título de doutor, 7 são mestres e 4 possuem especialização. A qualificação dos participantes assegura a

robustez da análise realizada, proporcionando uma avaliação fundamentada por especialistas com experiência significativa na área.

Com base nas respostas desses especialistas, a análise dos resultados indica que, de maneira geral, a extensão foi considerada intuitiva e útil para representar requisitos voltados à prevenção de *code smells* dentro do *iStar*. A extensão recebeu avaliações positivas na maioria dos critérios analisados, destacando-se especialmente pela sua representação gráfica e pela capacidade de auxiliar no rastreamento e mitigação de *code smells* ainda na fase de requisitos.

O critério mais bem avaliado foi a representação facilitada dos code smells em comparação à modelagem tradicional no *iStar*, que obteve uma média de 6.31. Dentre os participantes, 9 pessoas avaliaram esse critério como "Concordo Totalmente", enquanto 7 pessoas responderam como "Concordo". O mesmo valor foi atribuído à cobertura da extensão em relação aos principais problemas associados a *code smells*, sugerindo que a extensão atende bem à necessidade de modelagem desses problemas desde a elicitação de requisitos. Além disso, outro critério com avaliação elevada foi a ajuda da representação gráfica na visualização de *code smells*, que obteve uma média de 6.00, com 10 pessoas avaliando como "Concordo Totalmente" e 9 pessoas como "Concordo", evidenciando que os especialistas consideraram essa abordagem útil para identificar problemas estruturais de software logo nas primeiras etapas do desenvolvimento.

Outros critérios que também apresentaram médias altas incluem o suporte oferecido pela ferramenta *piStar-ext* para modelagem utilizando a extensão (6.15) e a flexibilidade da extensão para ser utilizada em diferentes domínios de software, que obteve o mesmo valor (6.15). No caso do suporte da ferramenta, 11 pessoas responderam como "Concordo Totalmente", enquanto 6 pessoas avaliaram como "Concordo". Já a flexibilidade da extensão para diferentes domínios foi considerada adequada por 9 pessoas que selecionaram "Concordo Totalmente", enquanto 7 pessoas marcaram "Concordo", indicando que os especialistas perceberam valor prático na aplicação da extensão, tanto no contexto acadêmico quanto no profissional, reforçando sua aplicabilidade na melhoria da qualidade do software.

O potencial da extensão para ser utilizada em projetos futuros na modelagem de requisitos também foi bem avaliado, com uma média de 6.23, com 10 pessoas respondendo "Concordo Totalmente" e 6 pessoas indicando "Concordo", demonstrando que os participantes consideram sua adoção viável.

A extensão também foi bem avaliada no que se refere à sua contribuição para equipes de desenvolvimento no mapeamento e prevenção de *code smells*, que recebeu 5.92, com 8

pessoas marcando "Concordo Totalmente"e 8 pessoas selecionando "Concordo". Da mesma forma, a inclusão de *Smell Metric* obteve a mesma média (5.92), indicando que os especialistas reconheceram a importância da extensão na rastreabilidade dos problemas de código, auxiliando na mitigação dos *code smells* desde a fase de requisitos.

Por outro lado, os critérios com menores médias foram a facilidade na modelagem de *code smells* no *iStar* (5.85), a clareza na identificação e mitigação de *code smells* (5.85) e a integração da extensão aos construtores originais do *iStar* (5.85). Esses valores indicam que, apesar das avaliações gerais positivas, alguns especialistas perceberam desafios na adoção da extensão.

Os participantes também responderam a três (3) perguntas abertas, permitindo uma análise mais detalhada sobre a percepção da extensão. As perguntas e os principais destaques das respostas obtidas são apresentados a seguir:

- **Você encontrou alguma dificuldade ao compreender os conceitos da extensão? Se sim, quais?** A maioria dos participantes não relatou dificuldades significativas na compreensão dos conceitos da extensão. Aqueles que apontaram desafios mencionaram principalmente a necessidade de um período inicial de familiarização com os novos construtores, mas ressaltaram que o vídeo explicativo e a documentação fornecida foram suficientes para entender a proposta.
- **Que melhorias poderiam ser feitas na extensão para facilitar seu uso?** Entre as sugestões de melhorias, os participantes mencionaram a possibilidade de adicionar exemplos mais detalhados e documentações complementares, além da integração de um guia rápido de referência para auxiliar novos usuários. Alguns especialistas sugeriram que a extensão poderia explorar ainda mais a relação entre diferentes tipos de *code smells*, tornando a modelagem ainda mais intuitiva.
- **Você acredita que a extensão poderia ser utilizada na prática? Em quais tipos de projetos?** Todos os especialistas concordaram que a extensão tem potencial de aplicação prática, especialmente em projetos de desenvolvimento de software que adotam modelagem baseada em requisitos. Foram citados casos de uso em equipes que desejam reduzir a ocorrência de *code smells* desde as primeiras fases do ciclo de vida do software, bem como em projetos acadêmicos e de pesquisa voltados à melhoria da qualidade do código.

Com base nesses resultados, conclui-se que a extensão cumpre seu objetivo de fornecer uma abordagem estruturada para a modelagem de requisitos voltados à prevenção

de *code smells*, demonstrando aplicabilidade e relevância para a comunidade acadêmica e profissional. No entanto, melhorias podem ser exploradas para aumentar sua flexibilidade e aplicação em diferentes contextos de software.

5.6 Limitações

Esta seção discute as restrições e desafios enfrentados no desenvolvimento da extensão, seguindo as diretrizes do processo PRISE e contando com o suporte do PriseBot.

Uma das principais limitações está na avaliação da extensão. Embora etapas do PRISE tenham sido seguidas, incluindo revisões conduzidas pelo próprio desenvolvedor e por especialistas na área de *code smells* e extensões *iStar*, a validação empírica mais ampla ainda não foi realizada. A ausência de testes práticos aplicados a cenários reais de modelagem pode impactar a compreensão do impacto e da aplicabilidade da extensão.

Outro aspecto relevante é o suporte ferramental. A ferramenta *piStar-ext*, utilizada para modelagem *iStar*, precisou ser adaptada para incorporar os novos conceitos introduzidos pela extensão. A implementação desses recursos foi essencial para garantir que a modelagem dos requisitos para a prevenção de *code smells* pudesse ser realizada de maneira adequada dentro do ambiente da ferramenta. No entanto, melhorias adicionais ainda podem ser exploradas para tornar a integração mais fluida e otimizar a usabilidade da extensão, proporcionando uma experiência mais intuitiva aos usuários.

Além disso, o PriseBot, que desempenhou um papel fundamental no suporte ao processo de desenvolvimento, apresentou limitações em termos de personalização e aprofundamento das respostas fornecidas. Embora tenha auxiliado na definição e estruturação dos conceitos da extensão, sua atuação ainda depende de refinamentos para permitir interações mais avançadas e alinhadas às necessidades específicas do projeto.

Por fim, a disseminação e validação científica da extensão ainda não foram concluídas. Apesar de seu desenvolvimento seguir metodologias consolidadas, a proposta ainda não foi publicada em conferências ou periódicos, o que limita sua adoção.

5.7 Ameaças à Validade

Atualmente, não existem extensões no *iStar* voltadas especificamente para a modelagem de *code smells*, o que dificulta a comparação direta da extensão proposta com outras

abordagens existentes. Dessa forma, sua validação foi conduzida considerando apenas a fundamentação teórica e a adequação ao processo PRISE.

Além disso, a avaliação da extensão foi realizada por um especialista em extensões *iStar*, que atuou como orientador do trabalho, e por um especialista na área de *code smells*. No entanto, não foram conduzidos testes empíricos para verificar sua aplicabilidade prática em projetos reais.

Portanto, avaliações futuras são essenciais para analisar o impacto da extensão em projetos de software, verificando sua eficácia na modelagem de requisitos para a prevenção de *code smells* e sua aceitação pela comunidade acadêmica e profissional.

6 CONCLUSÕES E TRABALHOS FUTUROS

Este trabalho abordou a importância da identificação e prevenção de *code smells* durante o desenvolvimento de software, destacando a necessidade de ferramentas que auxiliem na elicitação de requisitos relacionados à qualidade do código. Foi realizada uma revisão de trabalhos existentes sobre modelagem de requisitos para a *code smells* e identificada a ausência de uma extensão específica para essa finalidade na linguagem *iStar* 2.0.

Para preencher essa lacuna, foi desenvolvida uma extensão para *iStar* 2.0, seguindo as diretrizes do processo PRISE e contando com o suporte do PriseBot. Durante o desenvolvimento, foram definidos os conceitos que compõem a extensão, além da criação do metamodelo e da sintaxe concreta, permitindo a modelagem de elementos relacionados a *code smells* e suas estratégias de mitigação. Além disso, foi demonstrado como a extensão pode ser utilizada dentro da ferramenta *piStar-ext*, garantindo suporte ferramental à modelagem.

A extensão introduziu conceitos como *Task Smell*, *Quality Smell* e *Smells Metric*, permitindo que *code smells* sejam representados no contexto de requisitos de software e possibilitando a modelagem de estratégias para evitá-los ou corrigi-los. Esses conceitos fornecem uma abordagem estruturada para a análise de qualidade do código, integrando elementos de prevenção e mitigação no processo de ER.

A avaliação inicial da extensão foi realizada por especialistas em extensões *iStar* e *code smells*, garantindo uma primeira validação conceitual. No entanto, testes empíricos ainda não foram conduzidos para verificar a aplicabilidade da extensão em cenários reais de desenvolvimento.

Como trabalhos futuros, considera-se a aplicação da extensão em projetos reais ou experimentos controlados para avaliar seu impacto na modelagem de requisitos. Além disso, pretende-se aprimorar a integração da extensão à ferramenta *piStar-ext*, tornando seu uso mais intuitivo e acessível.

Dessa forma, a extensão proposta representa um avanço na modelagem de requisitos relacionados à qualidade do código no *iStar*, fornecendo uma estrutura inicial que pode ser expandida e aprimorada para atender melhor às necessidades da comunidade de desenvolvimento de software.

REFERÊNCIAS

AURUM, A.; WOHLIN, C. **Engineering and managing software requirements**. [*S. l.*]: Springer, 2005. v. 1.

BECK, K. **Extreme programming explained**: embrace change. [*S. l.*]: addison-wesley professional, 2000.

BEHUTIYE, W.; KARHAPÄÄ, P.; LÓPEZ, L.; BURGUÉS, X.; MARTÍNEZ-FERNÁNDEZ, S.; VOLLMER, A. M.; RODRÍGUEZ, P.; FRANCH, X.; OIVO, M. **Management of quality requirements in agile and rapid software development**: A systematic mapping study. Information and software technology, Elsevier, v. 123, p. 106225, 2020.

BOEHM, B.; BASILI, V. R. **Software defect reduction top 10 list**. Software engineering: Barry W. Boehm's lifetime contributions to software development, management, and research, v. 34, n. 1, p. 75, 2007.

CAIRO, A. S.; CARNEIRO, G. d. F.; MONTEIRO, M. P. **The impact of code smells on software bugs**: A systematic literature review. Information, MDPI, v. 9, n. 11, p. 273, 2018.

CHUNG, L.; NIXON, B. A.; YU, E.; MYLOPOULOS, J. **Non-functional requirements in software engineering**. [*S. l.*]: Springer Science & Business Media, 2012. v. 5.

CUESTA, L. L.; AYDEMIR, F. B.; DALPIAZ, F.; HORKOFF, J. **An empirical evaluation roadmap for iStar 2.0**. In: iStar 2016: Proceedings of the Ninth International i* Workshop, co-located with the 24rd International Requirements Engineering Conference (RE 2016): Beijing, China, September 12-13, 2016. [*S. l.*: *s. n.*], 2016. p. 55–60.

DALPIAZ, F.; FRANCH, X.; HORKOFF, J. **istar 2.0 language guide**. arXiv preprint arXiv:1605.07767, 2016.

FOWLER, M. **Refactoring**: improving the design of existing code. [*S. l.*]: Addison-Wesley Professional, 2018.

FOWLER, M.; BECK, K.; BRANT, J.; OPDYKE, W.; ROBERTS, D.-R. **Improving the design of existing code**. Addisonâ^'Wesley, v. 6, 1999.

FREIRE, E.; GONÇALVES, E.; OLIVEIRA, M.; LEITE, L. G. M.; SANTOS, S. K. C. **PriseBot-A chatbot to assist in the development of iStar**. In: Proceedings of the 20th Brazilian Symposium on Information Systems. [*S. l.*: *s. n.*], 2024. p. 1–10.

GONCALVES, E.; ARAUJO, J.; CASTRO, J. **PRISE**: A process to support istar extensions. Journal of Systems and Software, Elsevier, v. 168, p. 110649, 2020.

GONÇALVES, E.; HEINECK, T.; ARAÚJO, J.; CASTRO, J. **CATIE**: a catalogue of istar extensions. Cadernos do IME-Série Informática, v. 41, p. 23–37, 2018.

GUPTA, V.; KAPUR, P. K.; KUMAR, D. **Modelling and measuring code smells in enterprise applications using TISM and two-way assessment**. International Journal of System Assurance Engineering and Management, Springer, v. 7, p. 332–340, 2016.

HORKOFF, J.; AYDEMIR, F. B.; CARDOSO, E.; LI, T.; MATÉ, A.; PAJA, E.; SALNITRI, M.; PIRAS, L.; MYLOPOULOS, J.; GIORGINI, P. **Goal-oriented requirements engineering**: an extended systematic mapping study. Requirements engineering, Springer, v. 24, p. 133–160, 2019.

ISO/IEC. **Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — Quality model overview and usage**. 2024. International Organization for Standardization. ISO/IEC 25002:2024.

iStar Repository. **iStar Repository**. 2016. Disponível em: https://istarextensions.cin.ufpe.br/catalogue/. Acesso em: 15 ago. 2024.

KERIEVSKY, J. **Refactoring to patterns**. [*S. l.*]: Pearson Deutschland GmbH, 2005.

KESSENTINI, M.; KESSENTINI, W.; SAHRAOUI, H.; BOUKADOUM, M.; OUNI, A. **Design defects detection and correction by example**. In: IEEE. 2011 IEEE 19th International Conference on Program Comprehension. [*S. l.*], 2011. p. 81–90.

KITCHENHAM, B.; BRERETON, P. **A systematic review of systematic review process research in software engineering**. Information and software technology, Elsevier, v. 55, n. 12, p. 2049–2075, 2013.

KITCHENHAM, B.; PFLEEGER, S. L.; FENTON, N. **Towards a framework for software measurement validation**. IEEE Transactions on software Engineering, IEEE, v. 21, n. 12, p. 929–944, 1995.

LAMSWEERDE, A. V. **Goal-oriented requirements engineering**: A guided tour. In: IEEE. Proceedings fifth ieee international symposium on requirements engineering. [*S. l.*], 2001. p. 249–262.

LI, J.; WOODSIDE, M.; CHINNECK, J.; LITIOU, M. **Adaptive cloud deployment using persistence strategies and application awareness**. IEEE Transactions on Cloud Computing, IEEE, v. 5, n. 2, p. 277–290, 2015.

MARINESCU, R. **Detection strategies**: Metrics-based rules for detecting design flaws. In: IEEE. 20th IEEE International Conference on Software Maintenance, 2004. Proceedings. [*S. l.*], 2004. p. 350–359.

MOHA, N.; GUÉHÉNEUC, Y.-G.; DUCHIEN, L.; MEUR, A.-F. L. **Decor**: A method for the specification and detection of code and design smells. IEEE Transactions on Software Engineering, IEEE, v. 36, n. 1, p. 20–36, 2009.

MYLOPOULOS, J.; CHUNG, L.; YU, E. **From object-oriented to goal-oriented requirements analysis**. Communications of the ACM, ACM New York, NY, USA, v. 42, n. 1, p. 31–37, 1999.

NUSEIBEH, B.; EASTERBROOK, S. **Requirements engineering**: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering. [*S. l.*: *s. n.*], 2000. p. 35–46.

SHATNAWI, R.; LI, W. **An investigation of bad smells in object-oriented design**. In: IEEE. Third International Conference on Information Technology: New Generations (ITNG'06). [*S. l.*], 2006. p. 161–165.

SOMMERVILLE, I. **Software engineering (ed.)**. America: Pearson Education Inc, 2011.

SUMESH, S.; KRISHNA, A. **Hybrid analytic hierarchy process-based quantitative satisfaction propagation in goal-oriented requirements engineering through sensitivity analysis**. Multiagent and Grid Systems, IOS Press, v. 16, n. 4, p. 433–462, 2020.

TSANTALIS, N.; CHAIKALIS, T.; CHATZIGEORGIOU, A. **JDeodorant**: Identification and removal of type-checking bad smells. In: IEEE. 2008 12th European conference on software maintenance and reengineering. [*S. l.*], 2008. p. 329–331.

VALENTE, M. T. **Engenharia de software moderna**. Princípios e Práticas para Desenvolvimento de Software com Produtividade, v. 1, n. 24, 2020.

WHITTLE, J.; HUTCHINSON, J.; ROUNCEFIELD, M.; BURDEN, H.; HELDAL, R. **Industrial adoption of model-driven engineering: Are the tools really the problem?** In: SPRINGER. Model-Driven Engineering Languages and Systems: 16th International Conference, MODELS 2013, Miami, FL, USA, September 29–October 4, 2013. Proceedings 16. [*S. l.*], 2013. p. 1–17.

YAMASHITA, A.; MOONEN, L. **Do code smells reflect important maintainability aspects?** In: IEEE. 2012 28th IEEE international conference on software maintenance (ICSM). [*S. l.*], 2012. p. 306–315.

YU, E. S. **Towards modelling and reasoning support for early-phase requirements engineering**. In: IEEE. Proceedings of ISRE'97: 3rd IEEE International Symposium on Requirements Engineering. [*S. l.*], 1997. p. 226–235.

YU, E. S. K. **Modelling Strategic Relationships for Process Reengineering**. PhD thesis, University of Toronto, 1995.

ZAVE, P. **Classification of research efforts in requirements engineering**. ACM Computing Surveys (CSUR), ACM New York, NY, USA, v. 29, n. 4, p. 315–321, 1997.

ZHANG, H.; LIU, L.; LI, T. **Designing IT systems according to environmental settings**: A strategic analysis framework. The Journal of Strategic Information Systems, Elsevier, v. 20, n. 1, p. 80–95, 2011.