Ciência de Dados e I.A.
Escola de Matemática Aplicada
Fundação Getúlio Vargas

Engenharia de Requisitos

Proposta de TCC

LLM para Engenharia de Requisitos

Aluno: Isabela Yabe
Orientador: Rafael de Pinho André
Escola de Matemática Aplicada, FGV/EMAp
Rio de Janeiro - RJ.

Rio de Janeiro, 2025

Sumário

1 Introdução

A engenharia de software estuda e avalia métodos capazes de aproximar o código fonte da linguagem natural. Essa busca se manifesta em duas vertentes complementares: a interação com o usuário final e a comunicação entre os próprios desenvolvedores.

Esse estudo fundamenta-se nas constribuições de Larry Constantine, Edward Yourdon, Ivar Jacobson e Alistar Cockburn, autores que defendem o desenvolvimento estruturado e orientado ao usuário. Isto é, projetado a partir da visão e das necessidades de quem o utiliza, e não apenas da estrutura interna ou das preferências de quem o desenvolve. Essa perspectiva deu origem a princípios de design centrados na função e no comportamento observável do sistema, enfatizando que a organização do código deve refletir a experiência do usuário e os fluxos de interação previstos. Yourdon e Constantine (1979) descrevem o processo tradicional de desenvolvimento de software como uma cadeia de tradução sucessiva. O diálogo ocorre entre o proprietário do produto, o usuário, e o analista responsável por capturar suas ideias, conhecimentos de usabilidade e o comportamento esperado do software. Em seguida, o engenheiro de requisitos traduz essa concepção em um conjunto de requisitos funcionais e não funcionais, que por sua vez são reinterpretados pelo designer de sistemas em estruturas lógicas e técnicas. Por fim, o programador concretiza essas decisões, implementando no código as interpretações que compreendeu a partir do trabalho do designer. Cada etapa dessa cadeia de traduções implica na perda ou distorção de parte do significado original do usuário, o que pode resultar em comportamentoa apenas próximo ao desejado.

Como solução desse problema, os autores apresentam o conceito de *projeto estruturado*,

entendido como “a arte de definir os componentes de um sistema e as inter-relações entre esses componentes da melhor forma possível”Yourdon e Constantine (1979). Esse processo tem início na clareza e na visibilidade das decisões e atividades envolvidas, promovendo uma compreensão compartilhada entre os membros da equipe e garantindo que o design reflita as intenções originais do sistema.

"Introduzindo uma atividade específica de design formal para descrever completamente, e com antecedência, todas as partes de um sistema e suas inter-relações, não criamos uma nova atividade no ciclo de desenvolvimento de programas. O design estruturado apenas consolida, formaliza e torna visíveis as atividades e decisões de design que inevitavelmente acontecem - e de forma invisível - no curso de todo projeto de desenvolvimento de sistemas. Em vez de ocorrerem por tentativa e erro, sorte e padrão, essas decisões podem ser abordadas deliberadamente como compensações técnicas."

"Reunindo todas as decisões que afetam a escolha de módulos e inter-relações em um sistema, inevitavelmente influenciamos a maneira como outras decisões são organizadas e resolvidas. Assim, algumas questões que tradicionalmente eram abordadas de uma certa forma durante a fase inicial de um projeto podem ter que ser tratadas de uma maneira completamente diferente em uma fase muito posterior, uma vez que o designer adota uma abordagem de design estruturado."

"O design estruturado é o processo de decidir quais componentes interconectados

de qual maneira resolverão algum problema bem especificado."

Para Constantine, propriedades de qualidade como coesão e acoplamento não são apenas métricas técnicas, mas indicadores de quão bem o software reflete o domínio do problema e promove a clareza de propósito de cada módulo. Um sistema altamente coeso e fracamente acoplado tende a reproduzir com maior fidelidade a lógica do usuário, tornando-se mais previsível, transparente e alinhado às suas necessidades operacionais.

Afim de alcançar o resultado de um software onde observamos o comportamento esperado, modelagem de *casos de uso* surge como uma estratégia orientada ao usuário. Introduzido por Ivar Jacobson em 1989, o conceito propõe que para compreendermos o que um sistema deve fazer, é preciso identificar quem o utilizará e quais objetivos esse usuário pretende alcançar.

Segundo Jacobson e Cockburn, um caso de uso "conta a história completa", uma sequência de eventos que se inicia com uma necessidade e termina com a geração de valor, incluindo também alternativas, desvios e exceções. Essa narrativa funcional estabelece o elo entre o domínio do problema e a estrutura do software, permitindo que analistas e desenvolvedores compartilhem uma mesma linguagem de entendimento sobre o sistema.

Os princípios subjacentes aos casos de uso refletem diretamente o ideal do projeto estruturado defendido por Constantine e Yourdon. Ambos valorizam a decomposição do sistema em unidades compreensíveis e interdependentes, com ênfase na clareza das intenções e na previsibilidade das interações. Enquanto o projeto estruturado busca representar a estrutura do sistema de forma coerente e modular, os casos de uso descrevem o comportamento dessa estrutura na perspectiva do usuário.

Autores posteriores ampliaram essa discussão ao destacar o papel humano da comunicação no próprio ato de programar. Autores como Frederick P. Brooks Jr., Martin Fowler, Kent Beck e Robert C. Martin ressaltam que o desenvolvimento de software não é apenas um esforço técnico, mas também um ato de comunicação entre mentes humanas mediado pelo código. A produtividade e a clareza de um sistema dependem, portanto, não apenas da habilidade individual do programador, mas da forma como o conhecimento é transmitido, compreendido e preservado entre diferentes desenvolvedores ao longo do tempo.

Nesse sentido, a documentação, a estrutura do código, a nomenclatura e o comportamento esperado funcionam como pontes entre a intenção original do autor e a compreensão dos futuros mantenedores. Sommerville e Wiegers e Beatty destacam que a documentação é um artefato essencial da engenharia de requisitos, atuando como quem nos diz quem é o software, ou quem deveria ser. Ela traduz como o usuário pretende utilizar o sistema e quais são as características de negócio que o software deve refletir. A ausência de clareza nesse elo resulta em sistemas de difícil manutenção, inconsistências entre requisitos e implementação e perda de rastreabilidade, um problema recorrente em ambientes de desenvolvimento complexos e colaborativos.

No cenário contemporâneo, o avanço dos modelos de linguagem e das técnicas de representação semântica, como os *embeddings*, oferece novas possibilidades. Trabalhos recentes exploram como representações vetoriais de elementos de software, tais como identificadores, comentários, *issues*, *commits* e *docstrings*, podem captu-

rar aspectos semânticos úteis à engenharia de requisitos e à geração automatizada de artefatos. Essa evolução permite observar o código não apenas como uma estrutura sintática, mas como um repositório de intenções comunicativas, em que cada elemento textual contribui para reconstruir a lógica, o propósito e o comportamento esperados do sistema.

É nesse contexto que se insere o presente trabalho, propondo uma abordagem de engenharia reversa orientada por aprendizado de máquina, fundamentada em *embeddings* e *Retrieval-Augmented Generation* (RAG). O objetivo é investigar como informações semânticas presentes em comentários, identificadores e estruturas de código podem ser utilizadas para recuperar a intenção do desenvolvedor e derivar artefatos de engenharia de requisitos, em especial casos de uso. Ao conectar o código-fonte a uma base de conhecimento composta por obras clássicas e boas práticas de engenharia de software, busca-se contribuir para a construção de um processo de documentação e elicitação de casos de uso mais preciso, compreensível e automatizável.

1.1 Fundamentos da Engenharia de Software e Projeto Estruturado

Desde as décadas de 1970 e 1980, a engenharia de software vem se consolidando como um campo preocupado em reduzir a distância entre o comportamento esperado do sistema e sua implementação técnica. Yourdon e Constantine introduziram os princípios de projeto estruturado, destacando a importância da coesão e do baixo acoplamento entre módulos como indicadores de qualidade.

Posteriormente, Jacobson, Booch, Rumbaugh propuseram a UML e os casos de uso como meio de representar o comportamento do sistema a partir da perspectiva do usuário. Essa abordagem complementa o projeto estruturado ao oferecer uma visão funcional do software, alinhada às intenções e objetivos do usuário.

Autores como Brooks Jr., Fowler e Martin reforçaram a dimensão humana do desenvolvimento de software, tratando o código como um meio de comunicação entre desenvolvedores e um reflexo das intenções de design. Essas obras clássicas sustentam a premissa deste trabalho: o código-fonte contém, em sua estrutura e documentação, pistas da intenção do autor que podem ser extraídas e formalizadas por técnicas de engenharia reversa.

2 Metodologia de Mapeamento Sistemático

2.1 Capítulo 17 CASOS DE USO - UML

Nenhum sistema existe isoladamente. Todo sistema interessante interage com atores humanos ou autômatos que utilizam esse sistema para algum propósito e esses atores esperam que o sistema se comporte de acordo com as maneiras previstas. Um caso de uso especifica o comportamento de um sistema ou de parte de um sistema e é uma descrição de um conjunto de seqüências de ações, incluindo variantes realizadas pelo sistema para produzir um resultado observável do valor de um ator. Os casos de usos podem ser aplicados para captar o comportamento pretendido do sistema que

está sendo desenvolvido, sem ser necessário especificar como esse comportamento é implementado. Os casos de uso fornecem uma maneira para os desenvolvedores chegarem a uma compreensão comum com os usuários finais do sistema e com os especialistas do domínio. Além disso, os casos de uso servem para ajudar a validar a arquitetura e para verificar o sistema à medida que ele evolui durante seu desenvolvimento. À proporção que você implementa o seu sistema, esses casos de uso são realizados por colaborações cujos elementos trabalham em conjunto para a execução de cada caso de uso. Casos de uso bem-estruturados denotam somente o comportamento essencial do sistema ou subsistema e não são amplamente gerais, nem muito específicos.

Um caso de uso executa alguma quantidade tangível de trabalho. Sob a perspectiva de um determinado ator, um caso de uso realiza algo que é de valor para um ator, como o cálculo de um resultado, a geração de um novo objeto ou a modificação do estado de outro objeto. Por exemplo, na modelagem de um banco, o processamento de um empréstimo resulta na entrega de um empréstimo aprovado, manifestada como uma pilha de dinheiro entregue nas mãos do cliente.

Você poderá aplicar os casos de uso a todo o seu sistema. Também pode aplicá-los a uma parte do sistema, incluindo subsistemas e até interfaces e classes individuais. Em cada situação, os casos de uso não apenas representam o comportamento desejado desses elementos, mas também podem ser utilizados como a base de casos de teste para esses elementos, à medida que evoluem durante o desenvolvimento. Casos de uso aplicados aos subsistemas são excelentes fontes de testes de regressão; casos de uso aplicados a todo o sistema são excelentes fontes de testes de sistema e de integração. A UML fornece a representação gráfica de um caso de uso e de um ator, conforme mostra a Figura 17.1. Essa notação permite visualizar um caso de uso em separado de sua realização e no contexto com outros casos de uso.

3 Escolha da linguagem e porque OO

4 Escolha de repositório

4.1 Colossal Cave Adventure

Um casal de programadores e espleólogos, Will Crowther e sua ex Patricia Crowther, estavam mapeando o maior sistema de cavernas de Flint Ridge. No verão de 1974, Will jogava campanhas de Dungeons And Dragons e virava noites programando em FORTRAN o primeiro jogo de ficção interativa, "ADVENT".

Eu estava envolvido em um jogo de interpretação de papéis não-computadorizado chamado Dungeons and Dragons na época, e também vinha explorando ativamente cavernas — especialmente a Mammoth Cave, no Kentucky. De repente, eu tive uma ideia que combinasse o meu interesse por exploração de cavernas com algo que também fosse um jogo para as crianças, e talvez tivesse alguns elementos de Dungeons and Dragons, que eu vinha jogando. Minha ideia era que fosse um jogo de computador que não intimidasse pessoas que não usavam computadores, e esse foi um dos motivos pelos quais eu fiz com que o jogador dirigisse o jogo por meio

de comandos em linguagem natural, em vez de comandos mais padronizados. Meus filhos acharam que foi muito divertido.

Colossal Cave Adventure, inicialmente nomeado como "ADVENT", foi criado por Will Crowther em 1976 e expandido por Don Woods em 1977. Foi o primeiro jogo de ficção interativa, onde o jogador explora um mundo apenas por texto. O jogo foi inspirado em Dungeons and Dragons e também na caverna Mammoth no Kentucky. Escrito originalmente em FORTRAN para o computador PDP-10.

Referências

YOURDON, E.; CONSTANTINE, L. L. *Structured design: fundamentals of a discipline of computer program and systems design*. [*S. l.*]: Prentice-Hall, Inc., 1979.