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 em 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 comportamento apenas próximo ao desejado.

Diante desta situação, Yourdon e Constantine (1979) propõem o projeto estruturado, cujo ponto inicial é a clareza e a 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.

Com o mesmo intuito de tornar o comportamento do sistema visível e compreensível, surge a modelagem de casos de uso como um instrumento de unificação entre requisitos, design e usabilidade. Segundo Booch, Rumbaugh e Jacobson (1999), nenhum sistema existe isoladamente, todo sistema relevante interage com atores, humanos ou automáticos, que esperam comportamentos previsíveis. O caso de uso descreve essas interações por meio de um diagrama UML.

O diagrama de casos de uso coloca o usuário no centro do projeto, permitindo que analistas e desenvolvedores discutam o comportamento do sistema sem se prender aos detalhes da implementação. Essa representação funcional traduz a lógica de usabilidade em uma forma visual e verificável. Assim, os casos de uso oferecem uma linguagem comum para represetarmos comportamentos.

Autores posteriores ampliaram essa discussão ao nível do código, enfatizando a necessidade do código não ser apenas executável, mas também fácel de compreender. Trazendo as mesmas características que o desing estruturado: clareza e visibilidade. O código expressa intenções e decisões funcionando como um texto coletivo que deve ser lido e compreendido por outros desenvolvedores.

"Qualquer tolo escreve um código que um computador possa entender. Bons programadores escrevem códigos que os seres humanos podem entender."Fowler (2018)

Em projetos reais, frequentemente o código é o único artefato que sobrevive às

mudanças de equipe, tecnologia ou escopo. Embora o código contenha as decisões e comportamentos implementados, ele raramente comunica o motivo por trás dessas escolhas.

Wiegers e Beatty (2013) destacam que a documentação é um artefato fundamental da engenharia de requisitos, atuando como quem nos diz quem o software é, ou quem ele deveria ser. Ela traduz como o usuário pretende utilizar o sistema e quais são as características de negócio que o software deve refletir.

Quando essa camada de comunicação se perde, seja por ausência de documentação, desatualização ou inconsistências, o código torna-se a principal fonte de informações sobre o sistema. Essa situação, comum em projetos de longo ciclo de vida, motiva a necessidade de reconstruir a documentação a partir do próprio código-fonte, restaurando o elo entre o comportamento implementado e o comportamento pretendido.

1.1

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 capturar 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.

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

Este trabalho utiliza como base uma reimplementação de Rhodes (2010) em Python 3, que preserva o jogo original de Crowther e Don Woods, utilizando o arquivo de dados `advent.dat` Crowther e Woods (1977). O pacote permite jogar em dois modos, no *prompt* do Python e em terminal do sistema operacional. Além disso, disponibiliza *walkthroughs* automatizados na pasta de testes.

4.2 Descrição do jogo

Colossal Cave Adventure, também conhecido como *ADVENT* ou simplesmente *Adventure*, é amplamente reconhecido como o primeiro jogo de aventura baseado em texto da história, criado por Will Crowther em meados de 1975 e expandido por Don Woods em 1976.

Ambientado em uma caverna repleta de tesouros, criaturas e labiríntos, o jogador interage por comandos de texto, como *"GO NORTH"* ou *"GET LAMP"*. O sistema responde com descrições que narram as consequências das ações.

Como observa Dibbell (1998), o jogo automatiza o papel do mestre (*Dungeon Master*) característico de campanhas de *Dungeons and Dragons*. Suas descrições textuais simulam a fala do mestre ("*YOU ARE IN A MAZE OF TWISTY LITTLE PASSAGES, ALL ALIKE*").

"Como qualquer programa significativo, *Adventure* expressava a personalidade e o ambiente de seus autores." Levy (2010)

Will Crowther e sua ex-esposa, Patricia Crowther, ambos programadores e espeleólogos, participaram do mapeamento do sistema de cavernas *Mammoth Cave*. No verão de 1974, enquanto jogava campanhas de *Dungeons and Dragons*, Will começou o desenvolvimento do seu jogo utilizando o Fortran. O mapa utilizado no jogo foi inspirado diretamente nos levantamentos realizados pelo casal durante as expedições à Mammoth Cave, construindo no código a estrutura real da caverna.

Como o próprio Will Crowther relata, a ideia do jogo surgiu da combinação entre suas experiências em espeleologia e seu interesse por *Dungeons and Dragons*: "Eu estava envolvido em um jogo de interpretação de papéis... e tive uma ideia que combinasse o meu interesse por exploração de cavernas com algo que também fosse um jogo para as crianças..." Peterson (1983).

Levy (2010) conta como inicia a colaboração de Donald Woods, um pesquisador da *Stanford Artificial Intelligence Laboratory* (SAIL), em 1976. Após ter contato com uma prévia do jogo, Woods entrou em contato com Crowther, obteve sua permissão e passou a expandir o código. Sua versão incorporou novos puzzles, criaturas e elementos de fantasia inspirados na obra de Tolkien, além de um sistema de pontuação que estabelecia um objetivo ao jogador. A versão combinada de Crowther e Woods é um marco na história da interação humano-computador.

4.3

Como o jogo não possui documentação original, utilizei o artigo de Jerz (2007) como referência para compreender a estrutura e o funcionamento do código. O autor recupera e examina o código-fonte escrito por Will Crowther, a partir de um backup preservado no SAIL. Jerz descreve as seis tabelas centrais que organizam os dados do jogo: descrições longas, rótulos curtos das salas, dados de mapa, vocabulário agrupado, estados estáticos e eventos ou dicas.

Essa arquitetura de dados é mantida na reimplementação em Python, embora expandida para doze seções, resultado da integração da versão de Don Woods Rhodes (2010). A leitura e o processamento dessas tabelas ocorrem por meio do arquivo `advent.dat`, que preserva a semântica e a estrutura do código original.

As seis tabelas descritas por Crowther estruturam o mundo do jogo e suas interações:

1. **Long Descriptions**: textos descritivos longos que definem os ambientes e estados narrativos;

2. **Short Room Labels**: nomes curtos usados internamente para identificar locais e facilitar a navegação;
3. **Map Data**: conexões topológicas entre os ambientes e as direções de movimento possíveis;
4. **Grouped Vocabulary Keywords**: agrupamento de palavras-chave e comandos interpretados pelo sistema;
5. **Static Game States**: variáveis e condições fixas que controlam a lógica do jogo;
6. **Hints and Events**: mensagens de ajuda, eventos dinâmicos e respostas a situações específicas.

As outras seis adicionadas na versão em colaboração com Woods são:

1. *Object locations* — localização dos objetos;
2. *Action defaults* — mensagens padrão ligadas a verbos de ação;
3. *Liquid assets / flags* — `COND` por sala (luz, líquidos, restrições do pirata, bits de dicas);
4. *Class messages* — faixas de pontuação e mensagens de classificação do jogador;
5. *Hints* — dicas (turnos necessários, penalidade, pergunta e resposta);
6. *Magic messages* — mensagens de inicialização e manutenção.

Tabela 1 – Long Descriptions. A Tabela 1 contém descrições extensas dos ambientes do jogo. Com entradas identificadas de 1 a 140, ela define os textos apresentados ao jogador em diferentes locais. Cada linha representa uma sala ou estado narrativo. Parte dessas descrições refere-se diretamente a locais da caverna, como o trecho "*YOU ARE STANDING AT THE END OF A ROAD BEFORE A SMALL BRICK BUILDING*", enquanto outras descrevem situações de falha ou eventos inesperados, como "*YOU ARE AT THE BOTTOM OF THE PIT WITH A BROKEN NECK*".

Exemplos:

- 1 *AROUND YOU IS A FOREST. A SMALL STREAM FLOWS OUT OF THE BUILDING AND DOWN A GULLY.*
- 2 *YOU HAVE WALKED UP A HILL, STILL IN THE FOREST. THE ROAD SLOPES BACK DOWN THE OTHER SIDE OF THE HILL. THERE IS A BUILDING IN THE DISTANCE.*
- 3 *YOU ARE INSIDE A BUILDING, A WELL HOUSE FOR A LARGE SPRING.*

Tabela 2 – Short Room Labels. A Tabela 2 contém rótulos curtos correspondentes às localizações/ambientes do jogo. Com entradas numeradas de 1 a 130, nem todas as salas ou estados definidos em *Long Descriptions* possuem equivalentes resumidos.

Exemplos:

- 1 *YOU'RE AT END OF ROAD AGAIN.*
- 3 *YOU'RE INSIDE BUILDING.*
- 18 *YOU'RE IN NUGGET OF GOLD ROOM.*
- 19 *YOU'RE IN HALL OF MT KING.*

Tabela 3 – Map Data. A Tabela 3 codifica a topologia do mundo do jogo e as regras de navegação, funcionando como um grafo dirigido rotulado. A primeira coluna indica o ambiente em que o jogador se encontra, a segunda define o ambiente de destino, e as colunas subsequentes agrupam os vocabulários que podem ser utilizados para realizar a transição entre os dois pontos. O mapeamento dos vocabulários é definido na Tabela 4.

Em alguns casos, o valor do destino representa uma condição especial, e não uma simples sala. Se o número de destino for maior que 500, o jogo exibe uma mensagem da Tabela 6 e o jogador permanece no mesmo local; Se estiver entre 300 e 500, o valor indica um salto especial para um trecho de código do jogo.

Exemplos:

- 1 2 2 44 29: o jogador se desloca do ambiente 1 ao aombiente 2, se utilizados os comando 2, 44 ou 29.
- 3 1 3 11 32 44: o jogador se desloca do ambiente 2 ao ambiente 1 se utilizados os comando 3, 11, 32 ou 44.

Tabela 4 – Grouped Vocabulary Keywords. No código original em Fortran, toda entrada de texto era truncada nos cinco primeiros caracteres, de modo que o comando *"inventory"*, por exemplo, poderia ser digitado simplesmente como *"inven"*. A reimplementação em Python de Rhodes (2010) preserva essa lógica.

Os dados da tabela 4 são divididos em 4 grupos: o primeiro com id's entre 1 e 100 para movimento no jogo; com ids entre 1000 e 2000, trata de objetos manipuláveis ou características de cenário; com ids entra 2000 e 3000 são verbos de ação, se entre 3000 e 4000 são para casos especiais.

- 1–100: verbos de movimento, utilizados para navegação no espaço do jogo;
- 1000–2000: objetos e elementos de cenário manipuláveis;
- 2000–3000: verbos de ação (*carry*, *attack*, *drop*, etc.);
- 3000–4000: verbos de casos especiais, geralmente associados a eventos ou mensagens específicas definidas na Tabela 6.

Além dos comandos clássicos de navegação por bússola, "*EAST*"/"*E*", "*WEST*"/"*W*", "*NORTH*"/"*N*", "*SOUTH*"/"*S*", parte dos veros de movimentos são nomes de locais da caverna como "*BEDQU*"(truncamento de *Bedquilt*), "*HOUSE*", "*GATE*"e "*FORES*"(*forest*).

Exemplos:

- 2 *ROAD*
- 3 *ENTER*
- 3 *DOOR*
- 3 *GATE*
- 4 *UPSTR*
- 5 *DOWNS*
- 6 *FORES*

Palavras de mesmo sentido/sinônimos possuem mesmo id, como "*ENTER*", "*DOOR*"e "*GATE*".

Tabela 5 – Static Game States. A Tabela 5 armazena descrições curtas que representam estados do jogo, correspondendo às mudanças permanentes no ambiente. Cada linha contém um número e uma mensagem descritiva.

Quando o identificador está entre 1 e 100, a linha define a mensagem de inventário associada a um objeto, exemplo: "*SET OF KEYS*" se refere a "*KEYS*". Quando o identificador é um múltiplo de 100, a mensagem descreve uma propriedado do objeto.

Exemplos:

- 1 SET OF KEYS
- 000 THERE ARE SOME KEYS ON THE GROUND HERE.
- 2 BRASS LANTERN
- 000 THERE IS A SHINY BRASS LAMP NEARBY.
- 100 THERE IS A LAMP SHINING NEARBY.
- 3 *GRATE
- 000 THE GRATE IS LOCKED.
- 100 THE GRATE IS OPEN.

Tabela 6 – Hints and Events. A Tabela 6 reúne mensagens arbitrárias usadas como dicas e como descrições de eventos pontuais. Essas mensagens não estão relacionadas a um ambiente ou objeto específicos, elas são acionadas por outras estruturas do jogo, como as tabelas 3, 4, 8 e 11.

Exemplos:

1. 3 AXE AT YOU WHICH MISSED, CURSED, AND RAN AWAY.
2. 6 NONE OF THEM HIT YOU!
3. 13 I DON'T UNDERSTAND THAT!
4. 24 YOU ARE ALREADY CARRYING IT!
5. 33 I DON'T KNOW HOW TO LOCK OR UNLOCK SUCH A THING.

Tabela 7 – Object Locations. A Tabela 7 define onde cada objeto surge no mundo do jogo e se ele é móvel ou fixo. Cada linha possui o identificador do objeto, a sala inicial, e um campo opcional que indica imobilidade (-1) ou uma segunda sala quando o objeto existe simultaneamente em dois lugares

- Sala inicial = 0: o objeto não aparece no mundo no início e só será criado por algum evento ou ação do jogador.
- Terceiro campo = -1: o objeto está fixo naquela sala (não pode ser carregado).
- Terceiro campo = número de sala: o objeto está presente em duas salas ao mesmo tempo, objetos com duas localizações são tratados como imóveis.

Exemplos:

- 1 3: objeto 1 (1001 - KEY, KEYS) começam na sala 3 (INSIDE BUILDING).
- 2 3: objeto 2 (1002 - LAMP, HEADL, LANTE) começam na sala 3 (INSIDE BUILDING).
- 3 8 9: objeto 3 (1003 - grate) existe nas salas 8 e 9 simultaneamente (8 - YOU'RE OUTSIDE GRATE, 9 - YOU'RE BELOW THE GRATE.).
- (9 - DOOR) (94 - YOU ARE AT ONE END OF AN IMMENSE NORTH/SOUTH PASSAGE.)
- 9 94 -1: objeto 9 (1009 - DOOR) é fixo na sala 94 (94 - YOU ARE AT ONE END OF AN IMMENSE NORTH/SOUTH PASSAGE.).
- 15 0: objeto 15 (1015 - OYSTE) começa fora do mundo e aparece mais tarde.

Tabela 8 – Action Defaults. A Tabela 8 define o comportamento padrão dos verbos de ação, associando cada identificador de verbo ao índice da mensagem correspondente na Tabela 6. Cada linha contém dois valores: o primeiro é o número do verbo de ação, e o segundo é o identificador da mensagem padrão que deve ser exibida.

Exemplos:

- 1 24: o verbo de ação associado ao id 1 (2001 - CARRY, TAKE, KEEP, CATCH, STEAL, CAPTU, GET, TOTE) e a mensagem 24 da tabela 6 (YOU ARE ALREADY CARRYING IT!).
- 6 33: o verbo de ação associado ao id 6 (2006 - LOCK, CLOSE) e a mensagem 33 da tabela 6 (I DON'T KNOW HOW TO LOCK OR UNLOCK SUCH A THING.).
- 7 38: o verbo de ação associado ao id 7 (2007 - LIGHT, ON) e a mensagem 38 da tabela 6 (YOU HAVE NO SOURCE OF LIGHT.).

Tabela 9 – Liquid Assets, Etc. A Tabela 9 define os bits de condição associados a cada sala, controlando luz, líquidos, presença de inimigos e zonas de interesse para as rotinas de dicas. Cada linha contém um identificador de bit e uma lista de até vinte localizações nas quais esse bit é ativado. O jogo usa esses bits para determinar o comportamento dinâmico de cada ambiente.

- 0: indica que o ambiente está naturalmente iluminado.
- 1: tipo de líquido usado em conjunto com o bit 2. Quando o bit 2 está ativo, este bit diferencia óleo (1) de água (0).
- 2: marca as salas que contêm água ou óleo.
- 3: impede que o pirata apareça ali, exceto quando persegue o jogador.
- 4: jogador tentando entrar na caverna.
- 5: tentativa de capturar o pássaro.
- 6: interação com a cobra.
- 7: perdido no labirinto.
- 8: refletindo no quarto escuro.
- 9: na área final Witt's End.

Exemplos:

- 0 1 2 3 4 5 6 7 8 9 10 100 115 116 126: salas naturalmente iluminadas próximas à entrada.
- 2 1 3 4 7 38 95 113 24: presença de líquido (água ou óleo) nessas salas.
- 9 108: marca a área final do jogo, Witt's End.

Tabela 10 – Class Messages. A Tabela 10 contém as mensagens de classificação do jogador de acordo com a pontuação total atingida ao final da partida. Cada linha associa um limite superior de pontuação a uma mensagem que descreve o título ou o nível de habilidade alcançado.

Exemplos:

- 35: YOU ARE OBVIOUSLY A RANK AMATEUR. BETTER LUCK NEXT TIME.
- 100: YOUR SCORE QUALIFIES YOU AS A NOVICE CLASS ADVENTURER.
- 130: YOU HAVE ACHIEVED THE RATING: 'EXPERIENCED ADVENTURER'.
- 200: YOU MAY NOW CONSIDER YOURSELF A 'SEASONED ADVENTURER'.
- 250: YOU HAVE REACHED 'JUNIOR MASTER' STATUS.
- 300: MASTER ADVENTURER CLASSES C.
- 330: MASTER ADVENTURER CLASSES B.
- 349: MASTER ADVENTURER CLASSES A.
- 9999: ALL OF ADVENTUREDOM GIVES TRIBUTE TO YOU, ADVENTURER GRANDMASTER!

Tabela 11 – Hints. A Tabela 11 associa dicas contextuais a condições determinadas de jogo. Cada linha contém cinco valores:

- O primeiro valor vincula a dica a uma condição definidos na Tabela 9.
- O segundo valor define quantos turnos o jogador deve gastar no mesmo estado antes da dica ser oferecida.
- O terceiro valor representa a penalidade subtraída da pontuação total ao aceitar a ajuda.
- Os dois últimos valores apontam para mensagens da Tabela 6: a pergunta inicial e a resposta.

Exemplos:

- 4 4 2 62 63 — Bit 4 (entrada da caverna): após 4 turnos no local, o jogo exibe a pergunta 62 (Do you need help getting inside?) e, se aceita, mostra a resposta 63 (Perhaps you should explore the grate.), descontando 2 pontos.
- 6 8 2 20 21 — Bit 6 (cobra): depois de 8 turnos, o jogador recebe uma dica para resolver o enigma da serpente.

- 7 75 4 176 177 — Bit 7 (labirinto): após 75 turnos perdido, é oferecida uma dica de saída, com penalidade de 4 pontos.
- 8 25 5 178 179 — Bit 8 (quarto escuro): a dica surge depois de 25 turnos, custando 5 pontos.

Tabela 12 – Magic Messages. A Tabela 12 contém as chamadas *Magic Messages*, um conjunto de mensagens reservadas utilizadas pelos modos de inicialização, manutenção e administração do jogo. Embora seu formato seja idêntico ao da Tabela 6, elas são separadas para facilitar o acesso e o controle das rotinas especiais do sistema. Cada linha contém um identificador e um texto associado.agens internas do sistema.

Exemplos

- 1 *A LARGE CLOUD OF GREEN SMOKE APPEARS IN FRONT OF YOU... HE MAKES A SINGLE PASS OVER YOU WITH HIS HANDS, AND EVERYTHING FADES AWAY INTO A GREY NOTHINGNESS.*
- 2 *EVEN WIZARDS HAVE TO WAIT LONGER THAN THAT!*
- 3 *I'M TERRIBLY SORRY, BUT COLOSSAL CAVE IS CLOSED. OUR HOURS ARE:*
- 4 *ONLY WIZARDS ARE PERMITTED WITHIN THE CAVE RIGHT NOW.*

Referências

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

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. *The Unified Modeling Language User Guide*. [*S. l.*]: Addison-Wesley, 1999. (Addison-Wesley object technology series).

FOWLER, M. *Refactoring: Improving the Design of Existing Code*. [*S. l.*]: Pearson Education, 2018. (Addison-Wesley Signature Series (Fowler)).

WIEGERS, K.; BEATTY, J. *Software Requirements*. [*S. l.*]: Pearson Education, 2013. (Developer Best Practices).

RHODES, B. *Adventure (Python 3 port) — a faithful port of Crowther and Woods's 1977 FORTRAN Adventure*. [*S. l.: s. n.*], 2010.

CROWTHER, W.; WOODS, D. *Original Adventure sources (FORTRAN) and data*. [*S. l.: s. n.*], 1977. Archive of original sources. Linked from the historical page curated by Rick Adams.

DIBBELL, J. *My Tiny Life*: Crime and Passion in a Virtual World. New York: Holt, 1998.

LEVY, S. *Hackers: Heroes of the Computer Revolution - 25th Anniversary Edition*. [*S. l.*]: O'Reilly Media, 2010.

PETERSON, D. *Genesis II, Creation and recreation with computers*. [*S. l.*]: Reston Publishing Company, 1983.

JERZ, D. G. Somewhere Nearby is Colossal Cave: Examining Will Crowther's Original "Adventure" in Code and in Kentucky. Versão inglesa. *Digital Humanities Quarterly*, 2007.