PONTIFÍCIA UNIVERSIDADE CATÓLICA DE SÃO PAULO

PUC-SP

Wagner Varalda

Engenharia Hermenêutica de Requisitos: uma abordagem técnica para melhorar a elicitação e a avaliação dos requisitos de software

DOUTORADO EM TECNOLOGIAS DA INTELIGÊNCIA E DESIGN DIGITAL

São Paulo

2018

Wagner Varalda

Engenharia Hermenêutica de Requisitos: uma abordagem técnica para melhorar a elicitação e a avaliação dos requisitos de software

Tese apresentada à Banca Examinadora da Pontifícia Universidade Católica de São Paulo, como exigência parcial para obtenção do título de Doutor em Tecnologias da Inteligência e Design Digital - Área de Concentração: Processos Cognitivos e Ambientes Digitais, sob a orientação do Prof. Dr. Ítalo Santiago Vega.

São Paulo

2018

Banca Examinadora

_____

_____

_____

_____

_____

Dedico este trabalho a meu pai (*in memorian*),
grande incentivador e exemplo de vida.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 88887.150110/2017-00.

AGRADECIMENTOS

A DEUS,
Pelo fôlego de vida, sem o qual nada mais seria possível.

À MINHA FAMÍLIA,
Pelo incentivo, apoio, estímulo, paciência e compreensão.

À MINHA MÃE,
Por sua incondicional compreensão em relação à minha dedicação a este trabalho.

AO PROFESSOR DR. ÍTALO SANTIAGO VEGA,
Pela sábia orientação do inicio ao fim deste trabalho.

AOS PROFESSORES DR. DANIEL COUTO GATTI E DR. SERGIO BASBAUM,
Pelas relevantes contribuições feitas durante o Exame de Qualificação.

À ASSISTENTE DE COORDENAÇÃO EDNA CONTI,
Por suas preciosas dicas e assistências.

AO GRUPO DE ESTUDOS EM MODELAGEM DE SOFTWARE (GEMS),
Por suas pertinentes sugestões.

À PROFESSORA DRA. SANDRA GAVIOLI PUGA,
Pelo incentivo, apoio e colaboração para eu ingressar neste Programa de Doutorado.

A MEU PRIMO JOSÉ DE ARITMATHÉA MARINHO,
Pela dedicação e empenho prestados para revisar, por completo, o texto desta tese.

“Pensar só começa no ponto em que chegamos a saber que a razão, glorificada há séculos, é o adversário mais obstinado do pensamento”.
(Martin Heidegger)

RESUMO

A Engenharia de Software tem por objetivo desenvolver software de maneira "sistemática, controlada e quantificável", por meio da aplicação de uma série de atividades combinadas e integradas. Para que se defina o que o software deverá fazer, inclui-se a execução da atividade Engenharia de Requisitos, que tem por finalidade identificar, examinar e especificar o contexto do software a ser desenvolvido. O desenvolvimento do software depende, primariamente, desta atividade. Porém, há um problema cada vez mais em evidência: compreender o contexto do software a ser desenvolvido. Essa tese tem por objetivo apresentar uma proposta que visa enfrentar este problema por meio da aplicação da Engenharia Hermenêutica de Requisitos, a qual é formada por dois instrumentos: Elicitação Hermenêutica de Requisitos e Teodolito Hermenêutico de Requisitos. A Elicitação Hermenêutica de Requisitos utiliza métodos hermenêuticos adequados especificamente para a Engenharia de Requisitos, os quais auxiliarão o engenheiro de requisitos a compreender melhor as necessidades originais dos negócios a serem atendidos. O Teodolito Hermenêutico de Requisitos é um instrumento composto por dois mecanismos: um que avalia e apresenta os níveis de compreensão e de dificuldade que o engenheiro de requisitos possui em relação ao domínio da aplicação, e outro que avalia e apresenta os graus de qualidade dos requisitos de software e os seus níveis de dificuldade. Assim, será possível estabelecer estratégias para melhorar a aplicação da Elicitação Hermenêutica de Requisitos. Com isso, a Engenharia Hermenêutica de Requisitos ajudará o engenheiro de requisitos a compreender melhor o contexto do software a ser desenvolvido e, assim, conseguir determinar e constituir melhor os requisitos de software.

Palavras-chave: Engenharia de Requisitos. Elicitação de Requisitos. Avaliação de Requisitos. Engenharia Hermenêutica de Requisitos. Elicitação Hermenêutica de Requisitos. Teodolito Hermenêutico de Requisitos.

ABSTRACT

Software Engineering aims to develop software in a "systematic, controlled and quantifiable" way, through the application of a series of combined and integrated activities. In order to define what the software should do, it includes the execution of the activity Requirements Engineering, whose purpose is to identify, examine and specify the context of the software to be developed. Software development depends primarily on this activity. However, there is a growing problem: to understand the context of the software to be developed. This thesis aims to present a proposal that faces this problem through the application of Hermeneutical Engineering of Requirements, which is formed by two instruments: Hermeneutical Elicitation of Requirements and Hermeneutical Theodolite of Requirements. The Hermeneutical Elicitation of Requirements uses hermeneutic methods appropriately specifically for Requirements Engineering, which will help the requirements engineer better understand the unique business needs to be met. The Hermeneutical Theodolite of Requirements is an instrument composed for two mechanisms: one that evaluates and presents the levels of understanding and difficulty that the requirements engineer has in relation to the application domain, and another that evaluates and presents the quality grades of software requirements and their difficulty levels. Thus, it will be possible to establish strategies to improve the application of Hermeneutical Elicitation of Requirements. With this, the Hermeneutical Engineering of Requirements will help the requirements engineer to better understand the context of the software being developed and thus be able to determine better constitute the software requirements.

Keywords: Requirements Engineering, Elicitation of Requirements, Assessment of Requirements, Hermeneutical Engineering of Requirements, Hermeneutical Elicitation of Requirements, Hermeneutical Theodolite of Requirements.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ACM: Association for Computing Machinery – Associação para Máquinas Informáticas. Reúne educadores, pesquisadores e profissionais de informática para inspirar diálogo, compartilhar recursos e enfrentar os desafios da área.

CMU: Carnegie Mellon University – Universidade de Carnegie Mellon, onde foi estabelecida a criação do Instituto de Engenharia de Software (SEI, Software Engineering Institute).

DoD: Department of Defense - Departamento de Defesa dos EUA. Iniciou a parceria com a CMU para estabelecer o Instituto de Engenharia de Software (SEI, Software Engineering Institute),

FAST: Acrônimo de Ferramenta de Apoio aos Serviços da Tekno. Projeto de Software desenvolvido para uma indústria siderúrgica chamada Tekno S/A, que foi apresentado nesta tese como um dos quatro estudos de casos em que foram aplicadas a Engenharia Hermenêutica de Requisitos.

IBM: International Business Machines – Máquinas de Negócios Internacionais. Atua praticamente em todos os nichos da área e contribui para o seu desenvolvimento desde o século XIX.

IEEE: Institute of Electrical and Electronics Engineers - Instituto de Engenheiros Eletricistas e Eletrônicos. É uma das mais importantes organizações profissionais que se dedica ao avanço da tecnologia em benefício da humanidade.

OMG: Object Management Group – Grupo de Gerenciamento de Objetos. Desenvolve e melhora padrões que são amplamente aceitos e adotados pela comunidade da Engenharia de Software.

OTAN: Organização do Tratado do Atlântico Norte. Patrocinou as duas primeiras conferências que deram origem à Engenharia de Software. A primeira, em 1968, na Alemanha e a segunda, em 1969, na Itália. Atualmente, enfatiza seu apoio à inovação da área junto à iniciativa privada.

PPCP: Acrônimo de Planejamento, Programação e Controle da Produção. É um processo industrial que abrange todo o ciclo produtivo e envolve, praticamente, todas as áreas de uma empresa, principalmente a comercial, a de engenharia de produtos, a de produção, a de gestão de materiais, a de qualidade, a de manutenção, a financeira, a fiscal, a de recursos humanos e a de expedição. O Projeto de Software FAST à automação do PPCP da Tekno S/A.

SEI: Software Engineering Institute – Instituto de Engenharia de Software. Desde sua fundação, vem contribuindo amplamente com os avanços e melhorias da Engenharia de Software, tanto na esfera da indústria, como na esfera da academia.

SEMAT: Software Engineering Method and Theory – Método e Teoria de Engenharia de Software. Organização formada para criar e promover uma estrutura comum para a definição de métodos e práticas de Engenharia de Software. Desenvolveu o *Essence*, que atualmente é mantido pelo OMG.

SOLO: Acrônimo de Structure of Observed Learning Outcome – Estrutura do Resultado de Aprendizagem Observado. Para mais informações, veja no Glossário: Taxonomia SOLO.

ULS: Ultra-Large-Scale Systems – Relatório desenvolvido pelo SEI sobre os Sistemas de Grandes Escalas. Explica o que são estes sistemas e quais são os desafios que eles oferecem à Engenharia de Software, apontando, também, as alternativas para enfrenta-los.

UML: Unified Modeling Language – Linguagem de Modelagem Unificada. É uma linguagem-padrão para a elaboração da estrutura de projetos de software. Atualmente, é mantida pelo OMG.

WGSEET: Working Group on Software Engineering Education and Training – Grupo de Trabalho sobre Educação e Treinamento em Engenharia de Software.

SUMÁRIO

1. INTRODUÇÃO

A Engenharia de Software tem por objetivo desenvolver software de maneira "sistemática, controlada e quantificável", por meio da aplicação de uma série de atividades combinadas e integradas[1]. No contexto deste trabalho, aquelas relacionadas com a identificação de requisitos assumem maior importância. Assim, no espectro de atividades da Engenharia de Software, encontra-se a Engenharia de Requisitos, que se propõe a estabelecer as metas a serem alcançadas pelo software a ser desenvolvido e, também, por definir o comportamento necessário deste software, juntamente com suas restrições e condições impostas, para que ele atinja as necessidades do negócio em que atuará. Para isto, a Engenharia de Requisitos organiza as tarefas a serem realizadas em: Elicitação de Requisitos, Análise de Requisitos, Especificação de Requisitos, Validação de Requisitos e Gestão de Requisitos. Diversas técnicas e ferramentas já foram desenvolvidas para suportar a realização de cada uma delas[2], embora ainda se deva explorar, detalhadamente, a questão de compreensão de domínio, relacionada com a Elicitação de Requisitos.

A figura 1 ilustra as tarefas da Engenharia de Requisitos, destacando a Elicitação de Requisitos por ser a que faz parte do escopo da Engenharia Hermenêutica de Requisitos.

Figura 1 - Tarefas da Engenharia de Requisitos

Fonte: produção do próprio autor

[1] Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. IEEE, 2015.
[2] Bourque, Pierre. & Fairley, Richard E. (Dick), eds. SWEBOK v3.0: Guide to the Software Engineering Body of Knowledge. IEEE Computer Society, 2014.

A Elicitação de Requisitos tem por objetivo ajudar o engenheiro de requisitos a desenvolver a compreensão do domínio da aplicação e, também, dos problemas a serem solucionados pelo software a ser desenvolvido e/ou das oportunidades de negócio a serem usufruídas com o seu apoio. Esta tarefa deve ser feita junto à comunidade de negócio (*stakeholders*), pois é ela quem possui os conhecimentos e as informações necessárias para a sua realização.

Neste sentido, visando colaborar para que o engenheiro de requisitos melhor compreenda o domínio da aplicação, técnicas inspiradas na Hermenêutica abrem novas perspectivas para a Elicitação de Requisitos. A compreensão de domínios de aplicações estabelece uma base conceitual primária e essencial utilizada ao especificar os requisitos de software. Isto é feito por meio de uma nova tarefa denominada Elicitação Hermenêutica de Requisitos.

Durante o processo de Elicitação de Requisitos, o engenheiro de requisitos adquire vários níveis de compreensão em relação ao domínio da aplicação. Inicialmente, seu nível de conhecimento é relativamente baixo e, conforme vai obtendo mais informações, seu nível de compreensão vai aumentando e, com isso, aumentando também a possibilidade de melhor extrair os requisitos de software. Esses níveis de compreensão do engenheiro de requisitos e esses graus de qualidade dos requisitos de software evoluem progressivamente de uma escala menor a uma escala maior, até atingirem níveis satisfatórios para que os requisitos de software possam ser bem especificados.

Visando colaborar para que sejam acompanhadas as evoluções de compreensão do engenheiro de requisitos em relação ao domínio da aplicação, como também os graus de qualidade dos requisitos de software, foram desenvolvidas técnicas inspiradas na *Taxonomia SOLO* e no *Essence*, denominadas Teodolito Hermenêutico de Requisitos.

A *Taxonomia SOLO* classifica o nível de aprendizagem de uma pessoa em relação a um determinado tema que visa aprender[3]. O *Essence* indica e acompanha a progressão do desenvolvimento de software por meio de estados a serem atingidos[4]. Desse modo, o Teodolito Hermenêutico de Requisitos consegue avaliar e revelar os graus de

[3] Ceia, Mário José Miranda. A taxonomia SOLO e os níveis de van Hiele. Escola Superior de Educação de Portalegre, 2002.
[4] OMG. Essence – Kernel and Language for Software Engineering Methods, Version 1.1. SMSC, 2015, p. 28-31.

qualidade dos requisitos de software e os níveis de compreensão e de dificuldade do engenheiro de requisitos em relação ao domínio da aplicação para o qual o software será desenvolvido. Dessa forma, com estes resultados em mãos, é possível estabelecer estratégias para melhorar a aplicação da Engenharia Hermenêutica de Requisitos.

Justificativas

Durante seus cinquenta anos de história[5], a Engenharia de Software evoluiu exponencialmente para sempre acompanhar a demanda de um mundo que se encontra em constantes transformações, sejam elas na esfera pessoal, social, ambiental, das artes, dos negócios ou das ciências. Em praticamente todos os setores o software stá sendo utilizado, mas também há muitas queixas em relação a ele. Apesar da Engenharia de Software oferecer um conjunto de atividades que auxiliam a construção de software de maneira "sistemática, controlada e quantificável", ainda há problemas preocupantes e desafiadores que precisam ser solucionados.

Pensando nisto, em 1985, foi criada a organização *The Standish Group*[6], dedicada a apontar os pontos críticos em relação aos projetos de software e fornecer conselhos de como os melhorar. Sua contribuição mais conhecida e utilizada como referência é o Relatório do Caos (*Chaos Report*)[7] que desde 1994 vem sendo lançado regularmente com informações que traduzem os sucessos e insucessos dos projetos de software, apontando os principais motivos que contribuíram para tal e as ações necessárias para melhorar os cenários apresentados.

Dados surpreendentes apresentados nestes relatórios dizem respeito ao retrabalho, onde 94% dos projetos que começam devem ser reiniciados devido às aplicações fracassarem em fornecer as características esperadas. Mesmo assim, dos projetos concluídos, apenas uma média de 37% dos recursos e funções inicialmente especificados foram entregues e, dos projetos que se encontram em desenvolvimento, apenas 12% deles estão dentro do prazo e do orçamento. Na maioria das vezes, o problema

[5] Naur P. & Randell, B., eds. Software Engineering: Report of a Conference Sponsored by the NATO Science Committee. Garmisch, Germany, October 7-11, 1968. Scientific Af-fairs Division, NATO, 1969.
[6] Grupo Standish, disponível em <www.standishgroup.com>.
[7] The Standish Group. Report: Chaos, ed. 1994, 1995, 2011, 2012, 2013, 2014 e 2015.

identificado é a dificuldade em extrair e compreender as necessidades a serem atendidas pelo software em construção e também em especificar os requisitos deste software.

Na literatura especializada[8], também são encontrados relatos sobre os problemas encontrados e enfrentados pela Engenharia de Software. Frequentemente, menciona-se que os sistemas não funcionam exatamente como esperado pela sua comunidade de usuários e que bilhões de dólares são gastos para desenvolvê-los e outros milhões de dólares são desperdiçados para corrigi-los. De maneira mais enfática, esta literatura especializada também critica a Engenharia de Software por ela ser inadequada para o desenvolvimento moderno de software, dizendo que mesmo os mais decantados processos apresentam falhas, ainda não superadas, e sugerindo que novas técnicas de Engenharia de Software precisam ser desenvolvidas para atender a essas novas demandas.

O Instituto de Engenharia de Software (SEI, *Software Engineering Institute*)[9] também afirma que a "complexidade dos sistemas está aumentando drasticamente" em escalas sem precedentes e sugere que novas demandas vêm surgindo e trazendo novos desafios à Engenharia de Software. E também afirma que a Engenharia de Software atual é incapaz de enfrentar os problemas apresentados e que há a necessidade de pesquisas inovadoras em conceitos, métodos e ferramentas, a serem realizadas de maneira multidisciplinar. Quanto aos requisitos de software, o SEI afirma que estes são um fator crítico para o sucesso dos projetos de software. Também afirma que esta criticidade vem aumentando consideravelmente com as atuais demandas de desenvolvimento de softwares mais complexos.

Diante desses cenários, justifica-se, então, propor a utilização da Engenharia Hermenêutica de Requisitos, que vem a ser a adequação conceitual de métodos hermenêuticos em uma abordagem técnica que auxilia o engenheiro de requisitos a conceber melhor os requisitos de software e também a avaliar e revelar seus níveis de compreensão (e de dificuldade) em relação ao domínio da aplicação e aos graus de qualidade dos requisitos de software.

---

[8] Muitos livros e periódicos especializados em Engenharia de Software mencionam os problemas encontrados e enfrentados por esta área. Alguns destes se encontram referenciados nesta tese.

[9] Disponível em <www.sei.cmu.edu>.

Objetivos

O objetivo geral desta tese é apresentar uma proposta que visa ajudar o engenheiro de requisitos a melhor compreender e interpretar o domínio da aplicação, juntamente com os problemas a serem solucionados pelo software a ser desenvolvido e as oportunidades de negócios a serem usufruídas com o seu apoio; como também a avaliar e revelar os graus de qualidade dos requisitos de software e os níveis de compreensão e de dificuldade do engenheiro de requisitos em relação ao domínio da aplicação e dos problemas e oportunidades.

Como objetivos específicos, destacam-se:

1. Adequar conceitualmente métodos hermenêuticos em uma abordagem técnica denominada Elicitação Hermenêutica de Requisitos, que auxilia o engenheiro de requisitos a conceber melhor os requisitos de software. Estes métodos hermenêuticos advêm da filosofia hermenêutica de Martin Heidegger (1889 – 1976).
2. Adequar conceitualmente a *Taxonomia SOLO* e o *Essence* em um instrumento nomeado Teodolito Hermenêutico de Requisitos, para que seja possível avaliar e revelar os níveis de compreensão (e de dificuldade) em que o engenheiro de requisitos se encontra em relação ao domínio da aplicação, como também avaliar e revelar os graus de qualidade dos requisitos de software.

Hipóteses

Ao aplicar a Elicitação Hermenêutica de Requisitos, o domínio da aplicação para o qual o software será desenvolvido poderá ser compreendido e interpretado de maneira mais abrangente e, com isto, será possível estabelecer o escopo do projeto em um nível satisfatoriamente adequado para realizar a especificação dos requisitos de software.

Ao aplicar a o Teodolito Hermenêutico de Requisitos, será avaliado e revelado o nível de compreensão (e de dificuldade) em que o engenheiro de requisitos se encontra em relação ao domínio da aplicação e também serão avaliados e revelados os graus de qualidade dos requisitos de software. Com isto, poderão ser estabelecidas estratégias para melhorar a aplicação da Engenharia Hermenêutica de Requisitos.

Percurso Metodológico

Tendo em vista o conteúdo exposto nesta introdução, serão feitos levantamentos bibliográficos para situar o estado da arte em:

1. Engenharia de Software e Engenharia de Requisitos;
2. Hermenêutica, mais especificamente a Hermenêutica Filosófica estabelecida por Martin Heidegger, principalmente o conceito *Dasein*;
3. *Taxonomia SOLO*, para que seja possível classificar o nível de compreensão (e dificuldade) em que o engenheiro de requisitos se encontra em relação ao domínio da aplicação;
4. *Essence*, no que tange a Engenharia de Requisitos, para que seja possível avaliar os graus de qualidade dos requisitos de software.

Também, adequar conceitualmente:

5. Os métodos hermenêuticos advindos da Filosofia Hermenêutica de Martin Heidegger em uma abordagem técnica, chamada Elicitação Hermenêutica de Requisitos, que vai auxiliar o engenheiro de requisitos a conceber melhor os requisitos de software;
6. A *Taxonomia SOLO* e o *Essence* ao Teodolito Hermenêutico de Requisitos, para que seja possível avaliar e revelar os níveis de compreensão (e de dificuldade) em que o engenheiro de requisitos se encontra em relação ao domínio da aplicação, como também avaliar e revelar os graus de qualidade dos requisitos de software.

E, finalmente:

7. Realizar testes de conceito através de estudos de casos de variados domínios de aplicação e com diversos níveis de complexidade, para comprovar a eficiência da Engenharia Hermenêutica de Requisitos.

Organização da tese

O Capítulo 1 apresenta de maneira introdutória o escopo dessa tese, juntamente com as justificativas, os objetivos, as hipóteses, o percurso metodológico e sua organização.

O Capítulo 2 traz como tema a Fundamentação Teórica, apresentando a Hermenêutica, descrevendo o seu propósito e trazendo um breve histórico sobre o seu surgimento e evolução. Em seguida, o foco desloca-se para o conceito *Dasein*, desenvolvido por Martin Heidegger, gênese da Elicitação Hermenêutica de Requisitos. Na sequência, apresenta a *Taxonomia SOLO*, que classifica e observa os níveis de de desenvolvimento cognitivo de um aprendiz, e o *Essence*, que avalia a evolução de uma equipe de desenvolvimento de software. Ambos foram adequados para a concepção do Teodolito Hermenêutico de Requisitos.

No Capítulo 3, encontra-se o tema Engenharia de Software, onde há uma breve explanação sobre o seu histórico e sua situação atual (estado da arte).

O Capítulo 4 apresenta a Engenharia de Requisitos e aponta alguns problemas atuais relacionados ao desenvolvimento de software, destacando os pontos fracos da própria Engenharia de Requisitos, que serão enfrentados por meio das propostas apresentadas nesta tese.

No capítulo 5, encontra-se o tema Engenharia Hermenêutica de Requisitos, composta pela Elicitação Hermenêutica de Requisitos e pelo Teodolito Hermenêutico de Requisitos, que são as duas propostas apresentadas nessa tese para auxiliarem a equipe de desenvolvimento de software a melhor conceber os requisitos de software. São apresentadas as adequações realizadas no conceito *Dasein*, para a composição da Elicitação Hermenêutica de Requisitos, e nos conceitos *Taxonomia SOLO* e *Essence*, para a composição do Teodolito Hermenêutico de Requisitos.

O capítulo 6 apresenta as aplicações da Engenharia Hermenêutica de Requisitos em quatro estudos de casos, onde são mostradas as aplicações bem-sucedidas das propostas apresentadas nesta tese. Com isso, foi possível verificar e constatar suas soluções e benefícios.

No Capítulo 7 encontram-se as considerações finais, as quais apresentam as conclusões desta tese e os trabalhos futuros a serem desenvolvidos como continuidade da pesquisa.

2. FUNDAMENTAÇÃO TEÓRICA

2.1. **Compreensão e interpretação hermenêuticas**

Não é o propósito dessa tese discorrer em detalhes sobre o surgimento e a evolução da Hermenêutica. Mesmo assim, alguns pontos são relevantes para sustentar a proposta nela apresentada. A Hermenêutica é a ciência que estabelece os princípios, as leis e os métodos do processo humano de interpretação. Relaciona-se com a compreensão e a interpretação, para analisar o significado no campo da intersubjetividade, compreendendo e decifrando o sentido das coisas, e também identificando e estudando as leis que regem estas coisas. A Hermenêutica estabelece um pensamento reflexivo que vai ao encontro da verdadeira interpretação e compreensão contextualizada daquilo que se pretende conhecer.

A origem etimológica da palavra "hermenêutica" ainda gera dúvidas, mas provavelmente seja a grega "*hermeneuo*", que quer dizer "eu interpreto", cujo sentido vai ao encontro do fazer com que algo que não esteja totalmente claro se torne inteligível. Sua origem também é associada ao deus da mitologia grega Hermes, que era o "deus intérprete" o qual transmitia as mensagens dos deuses para os homens, tornando-as inteligíveis a quem estivesse em condições de ouvi-las e, assim, obter conhecimento. No entanto, foi somente no século XVII que esta palavra aparece publicada pela primeira vez; foi no livro "*Hermenêutica sacra sive methodus exponendarum sacrarum litterarum*", de Johann Conrad Dannhauer (1603–1666), cujo teor da obra apresenta regras para a interpretação sólida da Bíblia, o qual foi amplamente utilizado pela Reforma Protestante para se defender dos ataques que sofria dos movimentos contrários ao seu, e também para afirmar sua ideia de que "o homem é um ser capaz de ser tocado e modificado pelo poder e efeito da palavra", quando ela é apropriada por alguém que, na sua leitura e interpretação que dela faz, encontra seu sentido transformador e o adota como estilo de vida[10].

Até o século XIX, a Hermenêutica desenvolveu-se como uma disciplina auxiliar às ciências, para orientá-las, de forma normativa, na compreensão e na interpretação de antigos clássicos literários. A partir daquele século, a Hermenêutica

[10] da Silva, Maria Luísa Portocarrero Ferreira. **Hermenêutica Filosófica**. Dissertação. Universidade de Coimbra, Coimbra: 2010. Portal Filovida (Filosofia e Vida), disponível em https://www.filovida.org/2016/06/hermeneutica-etimologia-e-significado/

adquiriu uma nova dimensão, que a desconectou das práticas e normas hermenêuticas existentes até então, passando a adquirir um caráter mais científico, cujo amadurecimento foi alcançado no século XX.

Com Friedrich Schleiermacher (1768–1834), o pensamento hermenêutico ajuda a tornar mais claro o sentido do texto que se quer conhecer. A ele interessa, fundamentalmente, mediar a relação que existe entre o escrever e o falar, bem como a compreensão que se faz do discurso. Aquele que interpreta tenta entender o discurso assim como o seu autor quis dizer e tem a condição de evoluir e compreender de forma ampliada o texto. Isto se torna possível devido ao acréscimo que se faz à interpretação gramatical dos aspectos psicológicos que moveram a intenção mental do autor.

Com Wilhelm Dilthey (1833–1911), as ciências do espírito e a Hermenêutica se unificam e são relacionadas ao conhecimento histórico, formando o núcleo de uma hermenêutica que tem como principal tarefa a compreensão e a descrição das leis da vida, que constituem o fundamento comum das diferentes ciências humanas. Assim, a Hermenêutica transforma-se na teoria universal da compreensão e interpretação das objetivações significativas do ser humano, tendo como núcleo fundamental a possibilidade da consciência histórica de reconstruir, a partir das significações da vida, a intenção e as circunstâncias originárias do autor.

Com Martin Heidegger (1889–1976), a Hermenêutica se renova, reforçando a compreensão do ser humano como a pedra angular do filosofar. Em sua obra “Ser e Tempo”, de 1927, o pensamento hermenêutico aparece ligado à exigência urgente de um retorno à questão do sentido do ser e sua temporalidade, os quais são essenciais para a ampla compreensão do ser humano. A partir deste ponto, a Hermenêutica adquire raízes ontológicas profundas e encontra-se em um novo e importante horizonte. O homem deixa de ser um sujeito eterno e passa a ser finito e histórico e, sabendo desta sua condição finita e história, ele deve se proteger das opiniões particulares e dos hábitos do pensamento para poder formar uma consciência hermenêutica capaz de conduzi-lo à novidade que o texto lhe tem a dizer.

Com Hans-Georg Gadamer (1900-2002), a Hermenêutica esclarece o fenômeno ontológico da compreensão. Em sua obra “Verdade e Método”, de 1960, a Hermenêutica vai além de uma ação da subjetividade. É um processo no seio do próprio

acontecer do conhecimento, tendo a linguagem como instrumento fundamental do ser, num mundo já dito ou significado. Compreender adquire a forma de traduzir e questionar o processo de transmissão espiritual que constitui a humanidade do humano. Interpretar não é reconstruir ou coincidir, chegar apenas à dimensão cognitiva do dito, mas compreender-se a si mesmo à luz do texto. É traduzir, para a atualidade, o sentido das questões a que responde o texto, ressignificando-o com os conceitos do presente.

Com Paul Ricoeur (1913–2005), a Hermenêutica se torna mais crítica, reforçando ainda mais a importância do fenômeno da linguagem como elemento fundamental à compreensão. A partir daí, surgem duas possibilidades conflitantes e opostas de interpretação: a Hermenêutica da Confiança, que acredita no poder perceptivo e revelador dos símbolos, e a Hermenêutica da Suspeita, que acentua o seu poder de dissimulação para efetuar uma interpretação redutora e arqueológica da simbologia humana. As ciências humanas também recebem atenção especial e, com isso, percebe-se que a compreensão, além de ser um momento propriamente semântico, também é um modo de ser.

Além desses autores aqui citados, vários outros contribuíram e contribuem de maneira igualmente importante para o estudo do fenômeno da compreensão e interpretação, mas que não foram aqui citados porque extrapolaria o contexto deste capítulo e consequentemente desta tese.

Mesmo assim, é possível intuir que o interesse primordial da Hermenêutica se encontra na compreensão; seu propósito é fazer com que o outro entenda o que lhe é transmitido através dos símbolos. E é também saber que só se interpreta, de fato, quando não há uma compreensão imediata, ou seja, que é necessário desistir de uma compreensão plena e total, quando já se tem uma clareza compreensiva daquilo que se pretende conhecer.

Assim sendo, assumimos que, a partir de Martin Heidegger, com sua profunda reflexão acerca do ser, a Hermenêutica atingiu o seu aspecto fundamental, mostrando-nos a urgente necessidade de compreendermos que sempre fazemos algo para alguém e, por isso, precisamos fazer com excelência. Desse modo, partirmos de seu conceito central, *Dasein¸* para analisarmos as possibilidades de sua aplicação à Elicitação de Requisitos, a partir de adequações a serem feitas especificamente para esta finalidade.

Dasein

No cerne da obra de Martin Heidegger encontra-se a questão: o que é o ser? Para (tentar) responder a esta indagação, alguns conceitos devem ser colocados e pontuados antes, a saber:

O homem é o único ente que pode perguntar-se sobre o ser. Mas, o que vem a ser este ente? Ente é tudo aquilo que existe e se apresenta no aqui e no agora. Por exemplo, assim como cada um de nós, uma pedra, uma árvore, um planeta, ambos são entes.

Entretanto, o homem está além do ente. À pessoa – homem, Heidegger nomeou como *Dasein*, que frequentemente é traduzido como presença ou "ser-aí", que expressa o sentido do ser único que reside em cada um de nós, entes, que está presente no agora, vivendo (sendo). Ou seja, o *Dasein* é o ente para o qual o ser foi revelado (apresentado, mostrado, descoberto).

> O *Dasein* é o ente que, sendo, *des-cobre*, revela o Ser (o quê e como algo é) a partir de sua condição existencial. O *Dasein* é o ente para o qual o Ser se mostra. (ROEHE, 2014, p. 107).

Sendo assim, vê-se que cada *Dasein* compreende o ser a partir de sua própria existência. Então, ao retornarmos à questão original (o que é o ser?), mesmo ainda não tendo a resposta do que ele é, sabemos o que ele não é: ser não é o *Dasein*, mas chega-se à sua compreensão a partir dele.

> A tarefa do *Dasein* fica totalmente orientada para a tarefa de guiar a elaboração sobre a questão do ser, a partir de uma analítica existencial, a qual não tem a pretensão de proporcionar uma ontologia completa a respeito do *Dasein*. Ela explicita o ser desse ente; o que lhe compete é liberar o horizonte para a mais originária das interpretações do ser. (COELHO, 2009, p. 3)

Conforme já mencionado, o *Dasein* é compreendido a partir de sua existência. Essa, por sua vez, é constituída de duas formas: a existência autêntica e a existência inautêntica.

A existência inautêntica descreve a vida cotidiana do *Dasein* e esta é marcada por três conceitos fundamentais: a facticidade, a existencialidade e a ruína.

A facticidade vem a ser o fato de o *Dasein* ter sido lançado no mundo sem a sua própria vontade ou participação. Ele se encontra inserido num contexto já pautado por condições geográficas, históricas, sociais e econômicas.

A existencialidade é o fato de o *Dasein* projetar-se para poder moldar sua existência diante das possibilidades que lhe são disponibilizadas em seu mundo e tornar-se o que deseja ser, conforme vai se apropriando das coisas, oportunidades e situações deste seu mundo.

A ruína é aquilo que faz o *Dasein* desviar-se de seu projeto, impedindo-o de colocá-lo em prática (tornar-se o que deseja ser). Sejam as preocupações cotidianas, as pressões sociais, ou quaisquer distrações ou perturbações da vida, o *Dasein* pode facilmente alienar-se e não realizar seu projeto individual.

A existência autêntica é a que possui a capacidade de conduzir o *Dasein* à sua totalidade para que ele se conheça inteiramente em sua dimensão mais profunda e a partir daí possa ser levado ao encontro de sua essência.

Essa condição (a existência autêntica) permite ao *Dasein* transcender a existência inautêntica e enriquecer o seu próprio mundo com uma nova dimensão e profundidade. Com isso, ele pode constantemente renovar-se e atribuir incessantemente sentido a si mesmo, projetando suas próprias possibilidades de ser, ampliando seu horizonte e, desse modo, vivenciar plenamente a sua existência.

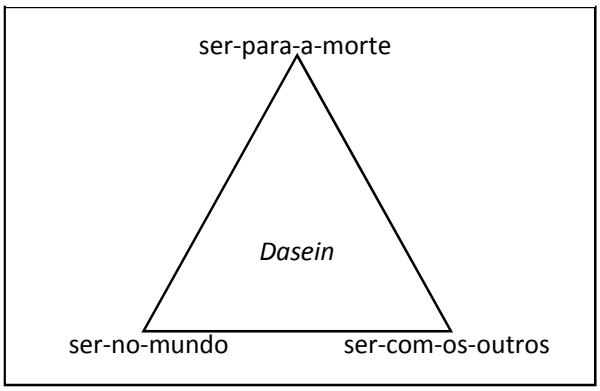
A transição da existência inautêntica para a existência autêntica pode acontecer por meio da utilização do vazio criativo (que às vezes ocorre nos estados de angústia), que é o sentimento que pode levar o *Dasein* a clarear novas possibilidades para o seu projeto.

> "Só o apelo sintonizado pela angústia possibilita que a presença se projete para o seu poder-ser mais próprio" (Heidegger, 1927/2006, p. 356). (ROEHE, 2014, p. 111).

O conceito do *Dasein* se desenvolve por meio de três conceitos-chave que permeiam estes dois grandes temas até aqui apresentados (vida inautêntica e vida autêntica). São eles: ser-no-mundo, ser-com-os-outros e ser-para-a-morte, os quais são apresentados a seguir.

A tríade do *Dasein* é ilustrada na figura 2:

Figura 2 - Tríade do Dasein



Fonte: produção do próprio autor

Ao adequar o conceito *Dasein* à Engenharia Hermenêutica de Requisitos, constituiu-se o conceito Elicitação Hermenêutica de Requisitos, apresentado no capítulo 5, o qual auxilia o engenheiro de requisitos a melhor compreender o domínio da aplicação para o qual o software será desenvolvido.

Ser-no-mundo

O *Dasein* é parte integrante do mundo e está sujeito ao que nele ocorre. Sua existência consiste em ser-no-mundo. Seu modo peculiar de se comportar e de se relacionar com as coisas e com as pessoas é marcado e definido por sua relação com o mundo, o qual já vem sendo contextualizado desde tempos remotos até o presente; é o mundo histórico, marcado e contextualizado pela temporalidade.

O *Dasein* não está dentro do mundo, mas envolvido por ele. Não se pode olhar para *Dasein* e mundo como duas entidades distintas que se relacionam, mas como partes constituintes de um todo. O mundo se funde ao *Dasein* e lhe dá o sentido de ser no mundo e não de estar nele. O mundo não é um espaço ou um lugar onde o *Dasein* se encontra lá dentro. O mundo é uma condição à existência do *Dasein* (e vice-versa) e possibilita a realização de suas atividades, seus modos de agir e, em última análise, sua existência.

> Em Ser e Tempo, homem e mundo aparecem como unidade ontológica original: não há homem sem mundo, nem mundo sem homem. (ROEHE, 2014, p. 107-108).

O *Dasein*, quando lançado no mundo, vivencia experiências que moldam e estruturam sua existência, dando-lhe sentido e contextualizando-o. A abertura que há nesta

relação lhe oferece as possibilidades de compreensão e interpretação de si e das coisas e pessoas que se encontram ao seu redor e estão à sua disposição. Com isso, é possível superar seus limites, remodelando e reestruturando sua existência, conforme ele vai se apropriando das novas possibilidades que lhe são desveladas (a esse conceito de "acontecimento apropriativo", Martin Heidegger chamou de *Ereignis*).

> Para Martin Heidegger, a acepção alemã *Ereignis* designa a noção de "acontecimento-apropriação" ou "acontecimento apropriador", já que remete o termo à sua dupla característica: situacional-temporal (acontecimento) e a de apropriação pelo homem. [...] A partir dessa acepção que o Ser se apropria do homem e o torna *Dasein*, o local da revelação do Ser. (SPINOLOA, 2012, p. 8-9).

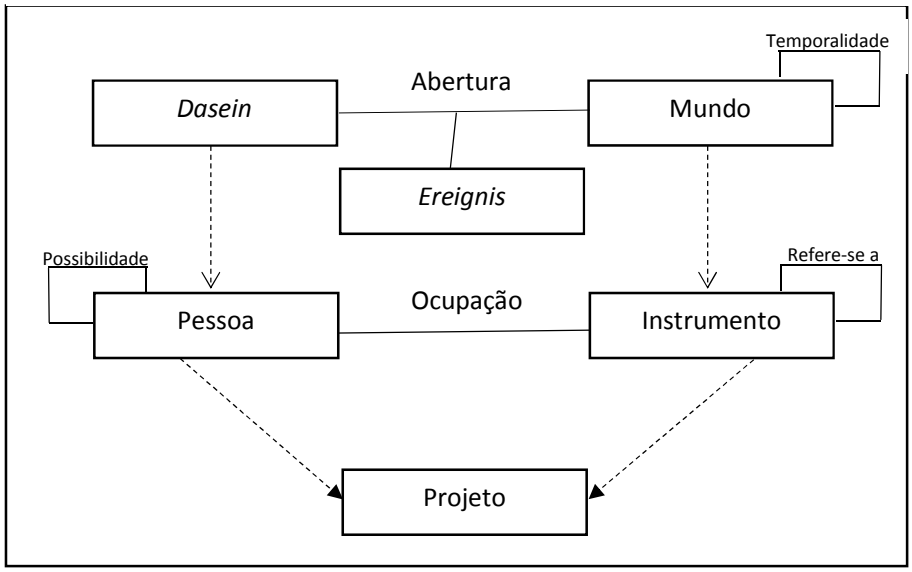
O mundo é o conjunto das relações que dão significado à existência do *Dasein*, constituindo a estrutura que lhe proporciona explicar e planejar suas ações para que ele possa viver sua vida autêntica, ao seu modo, de maneira mais plena e satisfatória. Qualquer mudança ocorrida em seu mundo pode ocasionar, também, mudança na estrutura do *Dasein*, pois suas escolhas e comportamentos estão inter-relacionados com este mundo.

A primeira relação do *Dasein* com o mundo se dá por meio da utilização e manuseio que ele faz dos instrumentos que lhe vêm ao encontro para que ele possa fazer suas tarefas e atividades. Essa relação acontece na forma de ocupação e esses instrumentos, inicialmente, são todos os objetos e pessoas que se encontram disponíveis para que seus projetos sejam realizados. Esses instrumentos sempre se referem a outros instrumentos e se integram a um todo instrumental que, em última análise, é o próprio mundo do *Dasein*.

Com isso, entende-se que o ser-no-mundo é o conceito que diz respeito ao olhar revelador e descobridor do *Dasein*, em sua condição imanente. É o olhar do *Dasein* para si mesmo, que se questiona a respeito de seu caráter ontológico para falar de si e encontrar-se com sua essência.

A figura 3 ilustra o conceito ser-no-mundo:

Figura 3 - *Dasein*: ser-no-mundo

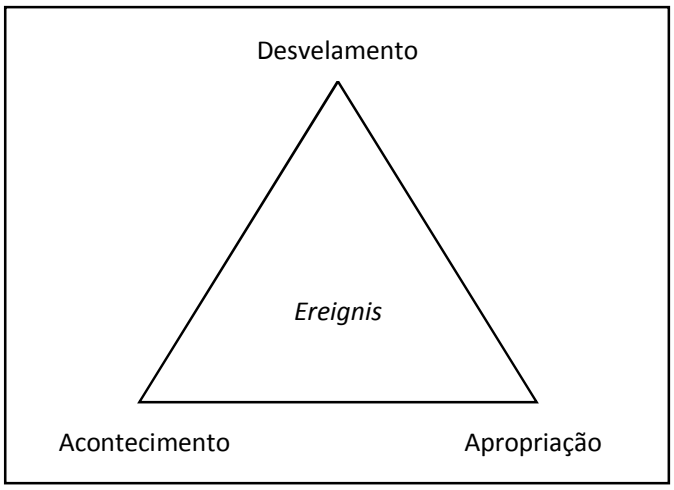


Fonte: produção do próprio autor

Ao adequar o conceito ser-no-mundo à Engenharia Hermenêutica de Requisitos, constituiu-se o conceito Identificação de Diferença Situacional, apresentado no capítulo 5, o qual auxilia o engenheiro de requisitos a identificar e compreender os problemas e/ou as oportunidades de negócio que se manifestam na Área de Negócio ou para a Comunidade de Negócio para o qual o software será desenvolvido.

A tríade do *Ereignis* é ilustrada na figura 4:

Figura 4 - Tríade do *Ereignis*



Fonte: produção do próprio autor

Ao adequar o conceito *Ereignis* à Engenharia Hermenêutica de Requisitos, constituiu-se o conceito Evento, apresentado no capítulo 5, o qual auxilia o engenheiro de requisitos a identificar e analisar os fatos e as informações que respondem pelas ocorrências das diferenças situacionais.

Ser-com-os-outros

Este conceito estabelece a condição de transcendência, a qual ocorre por meio das relações que são estabelecidas entre os entes. O outro do *Dasein* não é simplesmente o outro ente que com ele se encontra no mundo, mas sim aquele outro ente que partilha o mundo com ele. Esse outro sempre é percebido, pois estando no mundo, produz algo - realiza algo para alguém.

Sendo assim, o outro não pode ser visto como aquele ente que está presente em um mesmo espaço comum com os demais entes. Isso não faz dele o outro, pois esse espaço comum é o mundo cotidiano, onde, apesar dos entes aí estarem e se encontrarem, eles não compartilham nele o mesmo caráter ontológico do *Dasein*.

> No cotidiano, entretanto, o ser humano se coloca como mais um entre os entes, numa relação de identidade com as coisas que o cercam, relação esta na qual a característica ontológica fica encoberta. Na cotidianidade, o *Dasein* se mostra como sendo mais uma pessoa entre as outras pessoas, ou seja, vive sua vida como "fulano de tal" que tem um jeito particular de ser. Este nível cotidiano, da vida como sendo mais um entre os demais, é chamado ôntico. (ROEHE, 2014, p. 107).

Esse caráter ontológico do *Dasein*, chamado por Martin Heidegger de *Mitsein*, é que define o outro do *Dasein* e o possibilita dar sentido à sua existência. É nesse conceito que se encontra o mundo circunstancial, onde os entes se relacionam e se envolvem efetivamente para a realização de seus projetos.

> Como é notado, em Heidegger, o encontro com o outro não se dá através do estabelecimento das diferenças, ou mesmo, partindo de uma referência à exclusão de si mesmo. Seu encontro se origina a partir do mundo, onde o próprio *Dasein* se encontra de modo essencial. Tal expressão se denomina "mundo circunstancial". Nele *Dasein* não somente se encontra com o outro simplesmente dado, mas também, apropria-se da possibilidade de encontrar-se consigo mesmo. Tal encontro é sempre mediado pelo modo da "ocupação" com os entes intramundanos. (M. FILHO, p. 61).

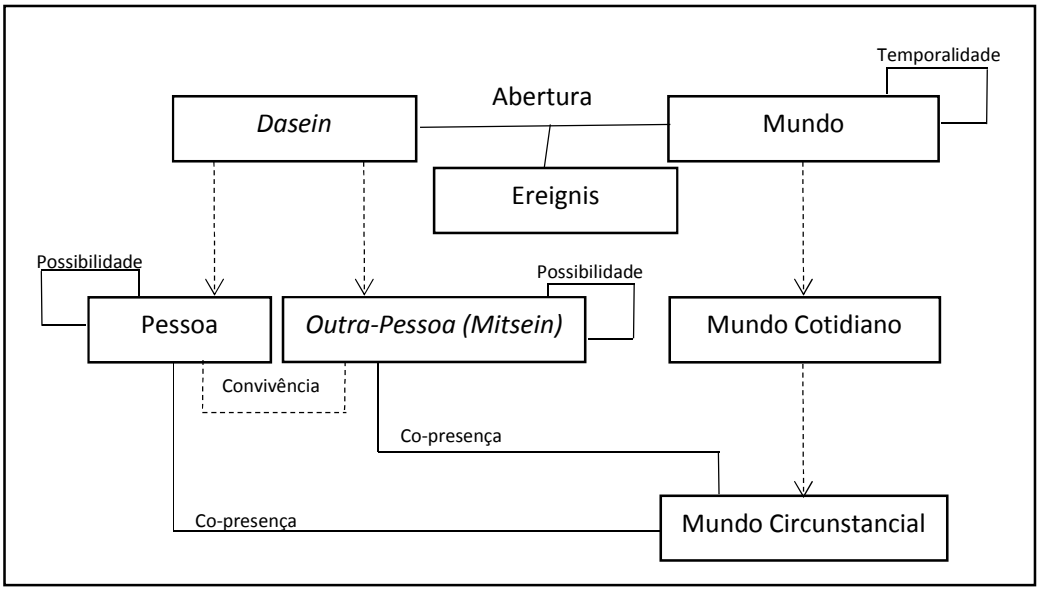
A abertura que há entre o *Dasein* e o mundo possibilita que o outro venha ao seu encontro. Mas esse encontro não ocorre por meio de uma proximidade física, mas por meio da co-presença, a qual estabelece a relação autêntica entre os entes.

Ser-com-os-outros significa uma relação de convivência entre os entes, onde os mesmos se correlacionam por meio de atitudes pautadas pela preocupação mútua entre eles, as quais fazem com que a compreensão do ser do *Dasein* passe também pela compreensão do outro.

> Em Heidegger, o encontro com o outro se estabelece por sua comum disposição no mundo; o outro me vem ao encontro no meu mundo circundante. Longe de guiar-se pelo pressuposto do eu ou do tu, tal encontro se define como convivência. O outro é aquele que é sempre e na maior parte das vezes um co-migo no mundo. (M. FILHO, p. 63).

A figura 5 ilustra o conceito ser-com-os-outros:

Figura 5 - *Dasein*: ser-com-os-outros



Fonte: produção do próprio autor

Ao adequar o conceito ser-com-os-outros à Engenharia Hermenêutica de Requisitos, constituiu-se o conceito Exame de Diferença Situacional, apresentado no capítulo 5, o qual auxilia o engenheiro de requisitos a identificar e compreender o ambiente em que a Área de Negócio e/ou a Comunidade de Negócio estão inseridos para, com isto, contextualizar amplamente as diferenças situacionais, para ter melhores condições de examiná-las e compreende-las.

Ser-para-a-morte

A condição de ser e existir do *Dasein* está intrinsecamente relacionada a este conceito, que serve como parâmetro fundamental para que ele desenvolva o seu projeto de vida.

O *Dasein*, quando lançado no mundo e em comunhão com os outros, vai conferindo sentido à sua existência. Ele pode escolher, por exemplo, sua profissão e área de estudo. Nas escolhas que faz, diante das inúmeras possibilidades, ele "está sendo", isto é, "existindo". Dentre essa variedade de possibilidades, existe uma que não há como fugir dela: a morte.

> A morte é um componente inevitável da temporalidade, ao qual não é dado ao homem evitar enquanto possibilidade. O fim é parte integrante e inevitável do ciclo existencial do *Dasein*, ocupando a morte seu papel de finitude na completude do Ser. A absorção do sentido da morte é essencial à experiência de apropriação do projeto de vida. Não há como essencializar o homem na verdade do Ser alheio à sua finitude. A morte compõe a outra face da mesma moeda em que a vida ocupa o seu anverso (presença). Sem qualquer das suas faces, não há valor existencial para a moeda. Na lição de Guimarães Rosa, no discurso de posse da Academia Brasileira de Letras – ABL foi ele peremptório: "a gente morre para provar que viveu". (SPINOLOA, 2012, p. 14).

A vida humana autêntica é aquela voltada para a possibilidade da morte – justamente aquela que não pode escolher – e não voltada para as possibilidades mundanas. Enquanto possibilidade, a morte é a mais própria do *Dasein*. Ela não tem relação alguma com o outro, muito menos com as coisas. A morte acontece com cada homem, e assim, individualmente, o homem experimenta-a; e um não pode interferir de modo algum na morte do outro. A morte marca a inexistência das outras possibilidades. Ela é a impossibilidade do próprio ser-no-mundo.

O "ser-para-a-morte" mostra ao *Dasein* que, um dia, ele não será mais um "ser-no-mundo" e nem um "ser-com-os-outros". É a visão de limite existencial, que permite testemunhar a totalidade do ser.

> Em Ser e Tempo, a morte como ser-para-o-fim é vista como a possibilidade mais própria do *Dasein*, pois é ele mesmo que dá a si essa possibilidade. A morte, como possibilidade própria, não está nas relações do ser-com ("preocupação" com os outros), nem no lidar com os entes não humanos ("ocupação" com as coisas). (ROEHE, 2014, p. 112).

O *Dasein* busca dar significado a certos fenômenos por ele percebidos, principalmente aos fenômenos da existência, da morte e da linguagem, sendo que cada ente

se relaciona de forma diferente para com estes fenômenos. A existência desperta curiosidade a ele, enquanto a morte lhe gera angústia. Em relação à linguagem, ela promove, por meio da comunicabilidade, a relação do "ser" com a essência do homem.

Enquanto "ser-no-mundo" e "ser-com-os-outros" o *Dasein* projeta a si o seu modo de ser e existir e, por meio da curiosidade que sua existência lhe gera, vai descobrindo novas possibilidades que lhe permitem melhor compreensão de sua essência.

De todas as suas possibilidades, uma é inevitável: a possibilidade de morrer. E diante dessa possibilidade, o *Dasein* pode ser conduzido ao encontro de sua totalidade enquanto "ser" e alcançar um estado de consciência que lhe permite compreender que suas preocupações, desejos e ambições não passam de sentimentos efêmeros. Assim, nele se manifesta uma profunda necessidade de transcendência que possibilita superar as angústias cotidianas de sua existência inautêntica e centrar-se no "ser" que virá a ser em sua existência autêntica, de acordo com as possibilidades as quais projeta e realiza em si, tomando em suas mãos o controle de seu destino.

A compreensão da essência do ser passa obrigatoriamente pela linguagem, que funciona como uma espécie de farol à sua compreensão. O ser fala de si por meio da linguagem, principalmente a poética. De acordo com Martin Heidegger, "o ser habita na linguagem" e sua compreensão e interpretação se dá por meio dela através da comunicabilidade, a qual possibilita construir as relações com o mundo, de acordo com as experiências vivenciadas.

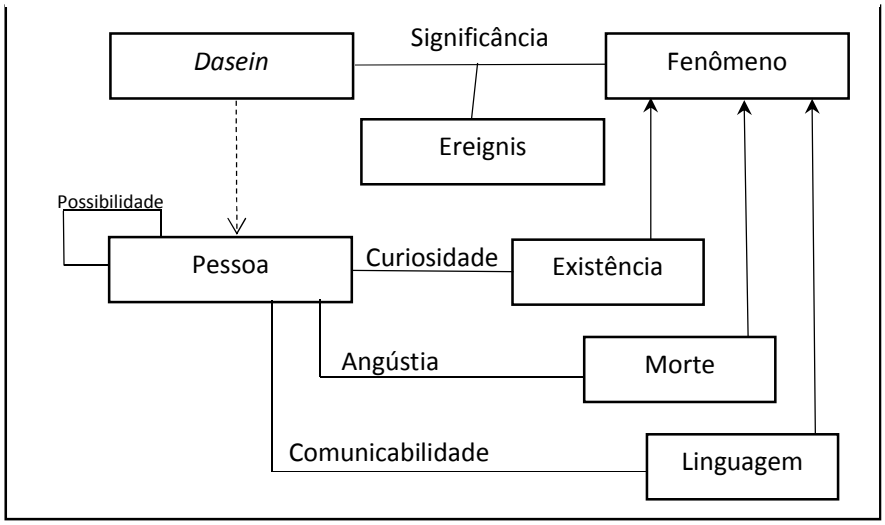
Através do "ser-para-a-morte" é possível determinar o sentido fundamental do *Dasein*, desde e a partir de sua origem, possibilitando compreender sua identidade, estabelecida ao longo de sua existência (temporalidade). Esta determinação funde o ser e o tempo numa unidade, para formar e destacar a base e essência do *Dasein*, em todas suas dimensões, para delimitá-lo. Com isso, o *Dasein* é desvelado em sua completude, conforme toda a sua existência até então compreendida, admitida e alcançada. Desse modo, a constituição ontológica do ser é estabelecida para dar testemunho de sua existência.

> A sua constituição ontológica deve se fazer visível com a elaboração da estrutura concreta da antecipação da morte. Como se cumpre a delimitação fenomenal dessa estrutura? Manifestamente no modo em que determinarmos os caracteres de sua abertura antecipadora a fim de se poder compreender puramente a possibilidade mais própria, irremissível, insuperável, certa e como tal

> indeterminada. Deve-se ainda atentar para o fato de que, primariamente, compreender não diz agarrar um sentido, mas compreender-se em suas possibilidades de ser, desentranhadas no projeto. [...] O que se busca é um poder-ser próprio do *Dasein* por ele mesmo, testemunhado em sua possibilidade existenciária. Antes de tudo é preciso que esse testemunho se deixe encontrar. E caso esse testemunho "se dê a compreender" para o *Dasein* em sua existência própria e possível, então ele deve ter suas raízes no ser do *Dasein*. A demonstração fenomenológica desse testemunho resguarda, pois, a comprovação de sua origem a partir da constituição ontológica do *Dasein*. (Ser e Tempo II, p.46, 52).

A figura 6 ilustra o conceito ser-para-a-morte:

Figura 6 - Dasein: ser-para-a-morte



Fonte: produção do próprio autor

Ao adequar o conceito ser-para-a-morte à Engenharia Hermenêutica de Requisitos, compõe-se o conceito Determinação Hermenêutica de Requisitos, apresentado no capítulo 5, o qual auxilia o engenheiro de requisitos a identificar e compreender os requisitos de software necessários para corresponderem ao Domínio da Aplicação.

2.2. **Taxonomia SOLO e os níveis de aprendizagem**

Desenvolvida por John Biggs e Kevin Collis em 1982, esta taxonomia tem por objetivo classificar os níveis de aprendizagem de uma pessoa em relação a um determinado tema que visa aprender, observando-se seu desenvolvimento por meio de um processo de aprendizagem adequado, estruturado em cinco níveis, onde cada nível representa um grau de progresso do aprendiz sobre seu conhecimento adquirido do objeto de estudo. SOLO vem a ser o acrônimo de *Structure of Observed Learning Outcome* (Estrutura do Resultado

de Aprendizagem Observado) e seus cinco níveis são: Pré-estrutural, Uni-estrutural, Multi-estrutural, Relacional e Estendido.

> Biggs' Solo (Structure of the Observed Learning Outcome) Taxonomy, is a systematic way of describing how a learner's performance develops from simple to complex levels in their learning. There are 5 stages, namely Prestructural, Uni structural, Multi-structural which are in a quantitative phrase and Relational and Extended Abstract which are in a qualitative phrase. (Chan, 2010, p.1).

Cada um desses níveis exige do aprendiz um determinado nível de esforço. Conforme aumenta o nível da Taxonomia SOLO, maiores são esses níveis de exigência.

> Each level of the SOLO taxonomy increases the demand on the amount of working memory or attention span. At the surface (unistructural and multistructural) levels, a student need only encode the given information and may use a recall strategy to provide an answer. At the deep (relational or extended abstract) levels, a student needs to think not only about more things at once, but also how those objects inter relate. (HATTIE, 2004, p.6).

Esses níveis de esforços exigidos são tratados na Taxonomia SOLO como "Capacidades", as quais representam as quantidades de memória de trabalho ou de tempo de atenção necessários para classificar o aprendiz em determinado nível.

No nível Pré-estrutural, o aprendiz está confuso em relação ao assunto que visa aprender (objeto de estudo). Ainda não possui conhecimento suficiente para encontrar respostas adequadas sobre o tema, se utiliza do senso comum para discorrer sobre ele e possui mínima capacidade para fazer sugestões a seu respeito.

Já no nível Uni-estrutural, o aprendiz possui capacidade para encontrar respostas sobre o objeto de estudo, apesar de ainda fazer generalizações a seu respeito, usando apenas um único aspecto e já conhecer alguns dados relevantes sobre o tema, que lhe permitem fazer fracas sugestões sobre ele.

O nível Multi-estrutural indica que o aprendiz já possui capacidade para encontrar respostas sobre o objeto de estudo, utilizando maior número de aspectos a seu respeito, apesar de ainda fazer generalizações com um número limitado de elementos relevantes conhecidos, os quais o limitam a fazer sugestões medianas acerca dele.

Estando no nível Relacional, o aprendiz possui grande capacidade para responder e sugerir sobre o objeto de estudo, podendo discorrer sobre ele de maneira contextualizada, utilizando diversos aspectos conhecidos, experimentados e relacionados.

Ao atingir o nível Estendido, o aprendiz possui total capacidade para responder e sugerir sobre o objeto de estudo, discorrendo de maneira contextualizada todos os aspectos relevantes conhecidos, estabelecendo inter-relações e elaborando hipóteses, onde suas habilidades lhe permitem fazer generalizações considerando situações ainda não experimentadas.

A tabela 1 apresenta a Taxonomia SOLO, destacando os seus cinco níveis e suas respectivas capacidades.

Tabela 1 - Taxonomia SOLO

Taxonomia SOLO Os cinco níveis com suas respectivas capacidades	
Nível	Capacidade
Pré-estrutural	Capacidade mínima de encontrar sugestões, dando respostas confusas.
Uni-estrutural	Fraca capacidade de encontrar sugestões ou dados relevantes.
Multi-estrutural	Capacidade mediana para descobrir sugestões e reconhecer os dados relevantes.
Relacional	Capacidade alta de encontrar sugestões, discorrer informação relevante e inter-relações.
Abstrato (Estendido)	Capacidade máxima de encontrar sugestões, discorrer informação relevante, estabelecer inter-relações e elaborar hipóteses.

Fonte: A taxonomia SOLO e os níveis de van Hiele, p. 245 – 246.

Ao adequar o conceito Taxonomia SOLO à Engenharia Hermenêutica de Requisitos, constituiu-se o mecanismo do Teodolito Hermenêutico de Requisitos, apresentado no capítulo 5, que auxilia o engenheiro de requisitos a avaliar e revelar seus níveis de compreensão (e também de dificuldade) em relação ao Domínio da Aplicação.

2.3. *Essence* e a Engenharia de Requisitos

Começou a ser desenvolvido no SEMAT (Software Engineering Method and Theory, Método e Teoria de Engenharia de Software) no final de 2009, por Ivar Jacobson, Bertrand Meyer e Richard Soley, com o apoio de outros colaboradores, com o propósito de

definir um núcleo e uma linguagem que possibilitam a criação, o uso e o aprimoramento dos métodos da Engenharia de Software.

> The work is based on the Semat initiative incepted at the end of 2009, which was envisioned by Ivar Jacobson, along with the other two Semat advisors Bertrand Meyer and Richard Soley. […]. This specification defines a kernel and a language for the creation, use, and improvement of software engineering methods. (OMG-Essence, 2015, p 7, 8).

O núcleo do *Essence* tem como objetivo definir uma base comum para o desenvolvimento de software, para determinar que práticas sejam elaboradas e aplicadas de forma independente, podendo ser combinadas para que sejam criados métodos específicos de Engenharia de Software, adaptados às necessidades específicas de uma comunidade, equipe, organização ou de um projeto particular de Engenharia de Software.

> The Software Engineering Kernel is a stripped-down, light-weight set of definitions that captures the essence of effective, scalable software engineering in a practice independent way. The focus of the kernel is to define a common basis for the definition of software development practices, one that allows them to be defined and applied independently. The practices can then be mixed and matched to create specific software engineering methods tailored to the specific needs of a specific software engineering community, project, team, or organization. (OMG-Essence, 2015, p 15).

No caso da linguagem, essa contém os elementos (e suas respectivas associações) para descreverem o conteúdo de um núcleo, fornecendo as coisas essenciais a serem feitas. Como a maioria das linguagens, ela também define e fornece a sintaxe, a semântica e a notação de seus elementos, listando-os e descrevendo-os, de maneira individualizada como também relacionada.

> This subclause serves as a narrative introduction to the most important language elements and illustrates their semantics on a coarse-grained level. […] are used to describe the contents of a Kernel. They provide the abstract and essential things to do, things to work with, and things to know in software engineering endeavors. […] As with most language specifications, this specification defines the elements included in the language (the abstract syntax), some rules for how these elements should be combined to create well -formed language constructs (the static semantics), and a description of the dynamic semantics of the language. In addition, for some of the elements or language constructs a concrete syntax (notation) is also provided. This subclause provides the abstract syntax and static semantics of the language by listing and describing the elements in the language and the relationships between them. (OMG-Essence, 2015, p 65-66).

Desse modo, o *Essence* abrange todas as atividades da Engenharia de Software com uma base sólida, consistente, madura e comprovada, que desde 2014 passou a ser um modelo padrão e vem sendo mantido pelo OMG.

Apesar de toda sua abrangência, devido ao escopo desta tese, a partir do próximo parágrafo será apresentado o *Essence* em relação à Engenharia de Requisitos.

Durante o desenvolvimento do software, seus requisitos progridem por meio de mudanças de estados, quais sejam: concebido, limitado, coerente, aceitável, endereçado e completo, onde a transição de um estado a outro ocorre conforme a compreensão da equipe de desenvolvimento do software vai melhorando em relação aos requisitos.

Inicialmente, os requisitos se encontram no estado concebido, onde a necessidade de um novo software foi percebida e acordada pelas partes interessadas, mas ainda não há um detalhamento de seu propósito. Mesmo assim, a oportunidade oferecida pelo novo software é conhecida.

Os requisitos passam para o estado delimitado quando as partes interessadas compartilham o propósito do novo software, as restrições a serem consideradas e o esquema de priorização, mesmo podendo ainda existir inconsistências de itens particulares e individuais.

Os requisitos atingem o estado coerente quando as características essenciais do novo software estiverem definidas de maneira clara. Esta coerência é alcançada por meio da elicitação, do refinamento, da análise, da negociação, da demonstração e da revisão dos requisitos, individualmente.

Os requisitos chegam ao estado aceitável quando estiverem suficientemente descritos a ponto das partes interessadas os aceitarem como uma solução inicial para o novo software.

O próximo estado é o endereçado, o qual indica que os requisitos abordam suficientemente o valor que o novo software resultará às partes interessadas, descrevendo de maneira precisa o que faz e também o que não faz parte de seu escopo, mas ainda não estão completamente descritos.

Ao progredirem para o estado completo, os requisitos já não possuem pendências, estão descritos completamente e as partes interessadas concordam que o software satisfará plenamente suas necessidades.

A tabela 2 apresenta os estados do OMG-*Essence* e seus respectivos progressos para a conclusão bem sucedida:

Tabela 2- Estados do *Essence*

Estados do OMG-*Essence*	
Estado	Progressos para a conclusão bem sucedida
Concebido	• O conjunto inicial das partes interessadas concorda que um sistema deve ser produzido. • As partes interessadas que usarão o novo sistema foram identificadas. • As partes interessadas que financiarão o trabalho inicial no novo sistema foram identificadas. • Existe uma clara oportunidade para o novo sistema abordar.
Delimitado	• As partes interessadas no desenvolvimento do novo sistema foram identificadas. • As partes interessadas concordam com o propósito do novo sistema. • Está claro qual é o sucesso para o novo sistema. • As partes interessadas têm uma compreensão compartilhada da extensão da solução proposta. • A maneira como os requisitos serão descritos é acordada. • Os mecanismos para gerenciar os requisitos estão em vigor. • O esquema de priorização é claro. • As restrições foram identificadas e consideradas. • As suposições foram claramente identificadas.
Coerente	• Os requisitos foram capturados e estão compartilhados com a equipe e as partes interessadas. • A origem dos requisitos é clara. • O raciocínio por trás dos requisitos é claro. • Os requisitos conflitantes foram identificados e atendidos. • Os requisitos comunicam as características essenciais do sistema a ser entregue. • Os cenários de uso mais importantes para o sistema podem ser explicados. • A prioridade dos requisitos é clara. • O impacto da implementação dos requisitos é entendido. • A equipe entende o que deve ser entregue e concorda em entrega-lo.
Aceitável	• As partes interessadas concordam que os requisitos descrevem uma solução aceitável.

	• A taxa de mudança para os requisitos acordados é relativamente baixa e está sob controle. • O valor fornecido pela implementação dos requisitos é claro. • As partes da oportunidade satisfeitas pelos requisitos estão claras. • Os requisitos são testáveis.
Endereçado	• O número suficiente de requisitos é endereçado para que o sistema resultante seja aceitável pelas partes interessadas. • As partes interessadas aceitam os requisitos como refletindo com precisão o que o sistema faz ou não faz. • O conjunto de itens de requisitos implementados fornece um claro valor para as partes interessadas. • O sistema que implementa os requisitos é aceito pelas partes interessadas.
Completo	• As partes interessadas aceitam que os requisitos capturam com precisão o que precisam para satisfazer plenamente a necessidade de um novo sistema. • Não há itens de requisitos pendentes que impeçam que o sistema seja aceito como satisfazendo totalmente os requisitos. • O sistema é aceito pelas partes interessadas como satisfazendo plenamente os requisitos.

Fonte: Essence – Kernel and Language for Software Engineering Methods, p. 37 – 38.

Ao adequar o conceito *Essence* à Engenharia Hermenêutica de Requisitos, constituiu-se o mecanismo do Teodolito Hermenêutico de Requisitos, apresentado no capítulo 5, que auxilia o engenheiro de requisitos a avaliar e revelar o graus de qualidade dos requisitos de software.

3. ENGENHARIA DE SOFTWARE: SURGIMENTO E ATUALIDADE

Neste capítulo apresenta-se uma visão geral da Engenharia de Software, desde seu surgimento até seu contexto atual e estado da arte.

3.1. **Apresentação da Engenharia de Software**

No início de 1967 o Comitê Científico da Organização do Tratado do Atlântico Norte (OTAN) reuniu cientistas do campo da ciência da computação, que representavam alguns de seus países membros, para discutirem sobre os problemas relacionados ao desenvolvimento de software. Como resultado dessas discussões, foi criado o Grupo de Estudo sobre Ciência da Computação que, no final daquele mesmo ano, recomendou a realização de uma conferência de trabalho sobre Engenharia de Software, a qual foi realizada em 1968, em Garmisch, Alemanha, entre os dias 7 e 11 de outubro. O termo Engenharia de Software foi escolhido tendo em vista a constatação de que o software deveria ser construído com base em fundamentos teóricos e práticos já estabelecidos pelas engenharias[11].

Com a participação de aproximadamente cinquenta especialistas de diversas áreas preocupadas com os problemas de software (desenvolvedores de software, universidades, fabricantes de hardware e usuários de computadores) foram discutidos problemas relacionados a três grandes tópicos: projeto de software, produção de software e serviço de software. Enquanto algumas discussões seguiram uma linha mais técnica, outras se focaram nas questões relacionadas à gestão e à metodologia da Engenharia de Software. Com isto, esta conferência da OTAN conseguiu abranger os principais pilares dos problemas enfrentados pelo desenvolvimento de software: atrasos de cronogramas, custos

[11] Naur P. & Randell, B., eds. Software Engineering: Report of a Conference Sponsored by the NATO Science Committee. Garmisch, Germany, October 7-11, 1968. Scientific Af-fairs Division, NATO, 1969.

superiores às estimativas, softwares não confiáveis e que não atendem às necessidades de seus usuários[12].

Logo no início da conferência, foram abordadas algumas preocupações acerca de uma tendência cada vez mais evidente: entregar software que não atinge em sua totalidade aquilo que se esperava dele. Este problema foi tratado por alguns membros da conferência como "crise do software" ou "hiato do software", em referência à diferença, cada vez maior e mais grave, do que é planejado e requerido, em comparação ao que é efetivamente realizado e entregue.

> There is a widening gap between ambitions and achievements in software engineering. This gap appears in several dimensions: between promises to users and performance achieved by software, between what seems to be ultimately possible and what is achievable now and between estimates of software costs and expenditures. The gap is arising at a time when the consequences of software failure in all its aspects are becoming increasingly serious. (OTAN 1968, 2001, p. 70).

Outra preocupação abordada logo no início desta conferência diz respeito à complexidade do software, que é tão importante quanto o hardware, mas bem menos compreendido. A combinação da crescente e urgente demanda pelo software e da inexperiência em desenvolvê-lo de acordo com os novos desafios que estavam surgindo resultou na necessidade de debater e encontrar novas soluções para superar seus atuais limites de desenvolvimento.

> Software is as vital as hardware, and in many cases much more complex, but it is much less well understood. It is a new branch of engineering, in which research, development and production are not clearly distinguished, and its vital role is often overlooked. There have been many notable successes, but recent advances in hardware, together with economic pressures to meet urgent demands, have sometimes resulted in this young and immature technology of software being stretched beyond its present limit. (OTAN 1968, 2001, p. 72).

Como resultado a esses debates sobre a "crise do software", chegou-se à conclusão de que é necessário melhorar as atuais técnicas utilizadas para construí-lo, mas que esta melhoria deve ser feita com cautela e de maneira constante.

> We can see no swift and sure way to improve the technology, and would view any claims to achieve this with extreme caution. We believe that the only way ahead lies through the steady development of the best existing techniques. (OTAN 1968, 2001, p. 73).

[12] Sommerville, Ian. Engenharia de Software. Tradução de Kalinka Oliveira e Ivan Bosnic. 9 ed. São Paulo: Pearson Basil, 2011, p. XI.

Assim, ao longo dos cinco dias desta conferência, diversos temas foram abordados e discutidos. Problemas de várias causas e efeitos apontados e reunidos em um relatório final (OTAN 1968) cuja finalidade é única: melhorar a maneira pela qual o software é desenvolvido. A partir de então, deu-se início a uma nova área do conhecimento humano: Engenharia de Software.

Um ano após, sequenciando este evento, a OTAN patrocinou outra conferência onde foram discutidos problemas mais relacionados aos aspectos técnicos, além da falta de comunicação e das dificuldades na formação dos estudantes da Engenharia de Software. Desta vez, a conferência foi realizada em Roma, Itália, entre os dias 27 e 31 de outubro de 1969[13].

Diferentemente da conferência anterior, os participantes desta conferência de Roma (aproximadamente setenta pessoas) já estavam habituados com o nome Engenharia de Software e já vinham trabalhando para o seu desenvolvimento e sua disseminação, apesar de saberem que muito trabalho ainda tinha pela frente, conforme se pode ver em um comentário feito ao final deste evento:

> One of the most interesting things that has been shown at this conference is that these projects don't all fail. It has been shown that some of them have been quite astonishingly successful. I think we ought to remember somebody or other's law, which amounts to the fact that 95 per cent of everything is rubbish. You shouldn't judge the contributions of computing science to software engineering on the 95 per cent of computing science which is rubbish. You shouldn't judge software engineering, from the high altitude of pure theory, on the 95 per cent of software engineering which is also rubbish. Let's try and look at the good things from the other side and see if we can't in fact make a little bridge. Let's see what the real difficulties are and whether we can do something to assist. (OTAN 1969, 2001, p. 9).

Este foi um comentário de merecido destaque, pois mostrou que a imatura Engenharia de Software, apesar de se encontrar em desenvolvimento, ainda carecia de muitas melhorias. Isto também ficou muito claro nas discussões sobre a educação em Engenharia de Software onde, logo no início dos debates, os participantes levantaram a questão: "existe diferença entre engenharia de software e ciência da computação?". Ao chegarem ao consenso que são duas áreas diferentes, as discussões passaram a ser sobre o que se deve ensinar aos estudantes de engenharia de software, reforçando a ideia de que

[13] Buxton J.N. & Randell, B, eds. Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee. Rome, Italy, October 27-31, 1969. Scientific Af-fairs Division, NATO, 1970.

deve ser um conteúdo que vai além do estado da arte, e que ainda seja útil e válido para os próximos vinte anos:

> [...] The premise is that it is clearly wrong to teach undergraduates the state of the art; one should teach them things which will still be valid in 20 years' time: the fundamental concepts and underlying principles. Anything else is dishonest. [...] Until we have a sufficient body of topics which are important, relevant and well developed we cannot call the subject a "science". I am quite convinced that in fact computing will become a very important science. But at the moment we are in a very primitive state of development; we don't know the basic principles yet and we must learn them first. If universities spend their time teaching the state of the art, they will not discover these principles and that, surely, is what academics should be doing. I do not for a moment underestimate the importance of the state of the art in engineering. [...] But, before teaching students we must get our basic principles right. (OTAN 1969, 2001, p. 49).

Além da parte técnica e da educação, a má comunicação também entrou na pauta das discussões e revelou a necessidade de se produzir uma boa e completa documentação capaz de abranger todas as etapas do processo de desenvolvimento do software.

Inicialmente, explicou-se a complexidade de desenvolver um grande sistema de software, o qual é um projeto caro, incerto e que requer muitas pessoas para a execução das tarefas, que não estão, necessariamente, preparadas para realizarem seus trabalhos e que também não têm a certeza de estarem fazendo o trabalho certo e da maneira certa, descobrindo se acertaram ou não somente na fase dos testes deste sistema, ficando evidente que esta situação não deve continuar como está e que deve, urgentemente, ser melhorada ou resolvida:

> One can conclude, then, that the large-scale system development process is complex, expensive and uncertain to a significant degree because large groups of people are utilized who are either not equipped to do their job or who may be doing the wrong job. Normally, of course, one doesn't recognize the difficulties of a systems development until the testing phase. Consequently, the people involved in this area bear the wrath of those who, until that time, assumed things were going well. In fact, the situation might have deteriorated quite badly by the time it got to the testing stage. It seems clear, however, that many of the difficulties are recognizable and perhaps some effort can improve the situation. (OTAN 1969, 2001, p. 101).

Na sequência, foi explicado que as pessoas responsáveis pelo desenvolvimento do software não entendem completamente as necessidades dos clientes e que há a necessidade de existir alguém da equipe que atue como uma espécie de orientador para intermediar e alinhar a comunicação entre todos os envolvidos, para que haja uma compreensão comum sobre a missão a ser atingida pelo software, apesar de saber da dificuldade de encontrar alguém que possua este perfil:

> In the early stages of the development of these systems, it is sometimes unclear that even the "customer" knows what he really wants. Of course, some subset of the customer group might have most of the information but the communication of these ideas is not easy. However, many venture into these systems quite unafraid, willing to figure it out after the work starts. Before allowing the build-up of the people and equipment necessary to implement the system, the customer and producer should be clear as to the mission to be performed. At least some subset of the producing organization should be operationally oriented. In addition that difficult-to-find operationally oriented but computer knowledgeable person should be available. (OTAN 1969, 2001, p. 101).

Por fim, para resolver o problema da má comunicação, concluiu-se que algumas pessoas devem atuar como comunicadores para documentarem as definições do software a ser desenvolvido, sob todos os seus aspectos. Apesar dessa tarefa já ser atribuída para alguns programadores, recomendou-se que ela seja executada por pessoas mais hábeis em escrever documentos de maneira mais abrangente, para que sejam produzidas as especificações completas e necessárias para o projeto:

> Communicators are needed. This includes people who can define the required system documentation. Then management must see to it that the documentation gets written. Frequently this is one of the chores left to programmers, who are in the main notoriously reluctant as well as not very skilled at writing. There are, however, individuals who understand programming and like to write. These people must be isolated and assigned the task of working with the operational and programming people to produce a complete series of operational, utility and test program specifications. (OTAN 1969, 2001, p. 102).

Ao final desta segunda conferência, tal como aconteceu no final da primeira, no ano anterior, foi produzido um relatório (OTAN 1969) o qual reuniu os diversos temas discutidos e registrou os problemas apontados e as respectivas recomendações para solucioná-los.

Logo na sequência desta conferência, houve a tentativa de se criar um Instituto Internacional de Engenharia de Software, mas os organizadores não conseguiram o financiamento pretendido para sua criação. Mesmo assim, sem as séries de conferências patrocinadas pela OTAN, diversos eventos foram realizados com o mesmo propósito e obtiveram bons resultados.

Em 1975 foi criada a Conferência Internacional de Engenharia de Software[14], que desde então "reúne pesquisadores, profissionais e educadores para apresentarem e discutirem as inovações, tendências, experiências e preocupações mais recentes no campo da Engenharia de Software". Seguindo um modelo parecido com o das conferências promovidas pela OTAN, esses eventos são realizados anualmente (salvo algumas

[14] www.icse-conferences.org

exceções: 1977, 1980, 1983 e 1986) e ocorrem em cidades e países diferentes com o intuito de promover a melhoria e o desenvolvimento da área.

No início da década de 1980, em 1981, foi fundada a *Rational Machines*, que posteriormente teve seu nome alterado para *Rational Software*, cuja missão era difundir o uso das práticas da Engenharia de Software. Com forte atuação no mercado, desenvolveu diversas ferramentas de apoio aos engenheiros de software e lançou importantes fundamentos para a área. Em 2003, a IBM comprou a *Rational Software*[15].

Também no início da década de 1980, o Departamento de Defesa dos Estados Unidos (DoD) sentiu a necessidade de mudar sua plataforma de desenvolvimento de novos sistemas, mas também estava consciente que estava pouco familiarizado com os conceitos da Engenharia de Software. Foi quando, em 1982, iniciou uma parceria com a Universidade Carnegie Mellon (CMU) onde foi estabelecida a criação do Instituto de Engenharia de Software (SEI, *Software Engineering Institute*)[16], fundado em 1984 e que, desde então, vem contribuindo amplamente com os avanços e melhorias da área, tanto na esfera da indústria, como na esfera da academia.

Em 1986, com o objetivo de fornecer produtos e serviços voltados ao desenvolvimento de software mais confiável e com custos e prazos reduzidos, foi fundada a *Hamilton Technologies, Inc*. (HTI)[17]. Suas contribuições estão sempre focadas no desenvolvimento de software onde as falhas não podem ocorrer.

Em 1989, com o objetivo de desenvolver padrões de tecnologia, foi fundado um consórcio internacional sem fins lucrativos, chamado Grupo de Gerenciamento de Objetos (OMG, *Object Management Group*)[18] o qual, com a "participação internacional de usuários finais, fornecedores, agências governamentais, universidades e instituições de pesquisas", vêm desenvolvendo e melhorando padrões que são amplamente aceitos e adotados pela comunidade da Engenharia de Software. Por ano, são realizadas quatro reuniões técnicas, onde são discutidas amplamente as deficiências e as oportunidades de diversos setores da tecnologia, oportunizando a antecipação aos problemas para evitá-los ou os mitigar. Além das reuniões técnicas, também são promovidos seminários, oficinas,

[15] www.ibm.com
[16] www.sei.cmu.edu
[17] www.htius.com
[18] www.omg.org

certificações e lançamento de livros e outros materiais que ajudam a disseminar o conhecimento de seus padrões.

Em 1990 foi fundado o Conselho Internacional de Engenharia de Sistemas (INCOSE, *International Council on Systems Engineering*)[19], uma associação sem fins lucrativos que representa estudantes, engenheiros, técnicos, gestores e organizações interessados no desenvolvimento e na disseminação dos "princípios e práticas interdisciplinares que permitem a realização de sistemas bem-sucedidos". Sua missão é "enfrentar desafios sociais e técnicos complexos, possibilitando, promovendo e avançando engenharia de sistemas e abordagens de sistemas". No caso do software, defende que o seu desenvolvimento não se concentra apenas na Engenharia de Software, mas que são necessários conceitos e métodos mais abrangentes, encontrados na engenharia de sistemas.

Em 1993, com a missão de "melhorar a qualidade de vida de todos que utilizam, desenvolvem e interagem com sistemas de software – usuários, desenvolvedores, gerentes, proprietários, educadores, estudantes e a sociedade como um todo", foi fundado, sem fins lucrativos, o Grupo *Hillside*[20], que atua para conscientizar os engenheiros de software a prestarem mais atenção aos fatores humanos, sem esquecerem-se das práticas da Engenharia de Software, para produzirem melhores sistemas de software. Acreditando ser o desenvolvimento de software uma das atividades humanas mais difíceis, este grupo promove uma série de atividades (oficinas, conferências e publicações) que ajudam os engenheiros de software a alcançarem este objetivo. Em 2001, foi fundado em Munique, Alemanha, o *Hillside Europe*[21], que também atua para divulgar os conhecimentos das melhores práticas em Engenharia de Software e a importância de se considerar os aspectos humanos da computação, por meio de conferências e publicações.

Em 1995 foi criado o Grupo de Trabalho sobre Educação e Treinamento em Engenharia de Software (WGSEET, *Working Group on Software Engineering Education and Training*) com aproximadamente oitenta profissionais internacionais da academia, indústria e governo, com o objetivo de orientar e apoiar o desenvolvimento de currículos

[19] www.incose.org
[20] www.hillside.net
[21] www.hillside.net/europlop/HillsideEurope

de graduação em Engenharia de Software. Em 1999, após intensos trabalhos, o WGSEET lança a primeira edição das Diretrizes para a Educação em Engenharia de Software[22].

Em 2001, duas grandes organizações de renome internacional: Associação para Máquinas Informáticas (ACM, *Association for Computing Machinery*)[23] e o Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE, *Institute of Electrical and Electronics Engineers*)[24] classificaram a Engenharia de Software como uma das quatorze áreas mais importantes do corpo de conhecimentos em informática. Este reconhecimento, mesmo que tardio, contribuiu com o desenvolvimento de currículos de graduação para a educação de profissionais da Engenharia de Software baseados nas diretrizes do WGSEET.

Em 2003 foi criado na Bulgária o Instituto Europeu de Software (ESI, *European Software Institute*)[25] que vem a ser uma importante instituição europeia focada em promover no continente a excelência em Engenharia de Software. Três anos após sua fundação, em abril de 2006, ele foi reconhecido por sua importante contribuição para o desenvolvimento de melhores práticas na área e, desde então, vem cooperando ativamente para a promoção, adoção e aperfeiçoamento da Engenharia de Software.

Em 2004, devido à crescente demanda por qualidade de software, foi fundada a Associação para Testes de Software (AST, *Association for Software Testing*)[26] a qual não possui fins lucrativos e é integrada por membros de mais de cinquenta países que se dedicam para a construção de práticas que avaliam a qualidade de software de maneira mais rápida, melhor e menos dispendiosa, aplicando e valorizando abordagens cientificas para o desenvolvimento e a avaliação de técnicas, processos e ferramentas.

Em 2010, foi fundado o Instituto de Sustentabilidade de Software (SSI, *Software Sustainability Institute*)[27], o qual reuniu para a sua formação especialistas em desenvolvimento de software, gerenciamento de projetos, pesquisa e publicidade das universidades de Edimburgo, Manchester, Oxford e Southampton, do Reino Unido. Posteriormente, o instituto passou a trabalhar com grupos de pesquisas de praticamente todas as áreas e localidades, os quais possibilitam "uma ampla compreensão dos problemas

[22] www.dtic.mil/get-tr-doc/pdf?AD=ADA370372
[23] www.acm.org
[24] www.ieee.org
[25] www.esicenter.bg
[26] www.associationforsoftwaretesting.org
[27] www.software.ac.uk

que os pesquisadores enfrentam ao usar software". Sabendo que atualmente o software se tornou também essencial para diversas áreas de pesquisas, o objetivo deste instituto é "cultivar e melhorar o software de pesquisa para apoiar a pesquisa de classe mundial".

Em 2012 foi fundada a Associação Internacional dos Arquitetos de Software (IASA, *International Association of Software Architects*)[28] que vem a ser um fórum internacional que atua para auxiliar os desenvolvedores de software em suas necessidades atuais, os ensinando sobre as mais recentes tecnologias e melhores práticas voltadas ao desenvolvimento de software, principalmente da Arquitetura de Software (que vem a ser uma das áreas da Engenharia de Software), mas não se limitando a ela.

Em 2013, com o apoio do SSI, foi fundada a Associação dos Engenheiros de Software de Pesquisa (RSE, *Research Software Engineers Association*)[29] onde são organizados regularmente eventos que permitem aos engenheiros de software de pesquisas se encontrarem para trocarem experiências, conhecimentos e colaborações. Há também atividades voltadas para fazer com que a academia reconheça oficialmente a área Engenharia de Software de Pesquisa. Conferências, oficinas, workshops e publicações também são outras atividades realizadas por esta associação para maior disseminação e melhor desenvolvimento da área. Em 2015, o Conselho de Pesquisa em Engenharia e Ciências Físicas (EPSRC, *Engineering and Physical Sciences Research Council*)[30], que é a principal agência do Reino Unido em financiamentos de pesquisas em engenharia e ciências físicas, ofereceu três bolsas de estudos especificamente para a Engenharia de Software de Pesquisa. Como houve procuras (duzentas e onze pessoas se inscreveram para estas três bolsas) o EPSRC aumentou o número de bolsas para sete. Em 2016, a RSE já contava com exatos 633 membros – engenheiros de software de pesquisa cadastrados em sua comunidade. Em 2017, participantes de quatorze países se reuniram no Museu de Ciência e Indústria, em Manchester, Inglaterra, entre os dias 6 e 8 de setembro para debaterem o estado da arte em software de pesquisa sustentável, discutirem os esforços que se encontram em andamento para o aprimoramento da área e identificarem os possíveis tópicos a serem melhorados.

[28] www.iasaglobal.org
[29] www.rse.ac.uk
[30] www.epsrc.ac.uk

Em setembro de 2014, na reunião de cúpula da OTAN realizada em Gales, Reino Unido, os líderes da aliança endossaram a Parceria Cibernética da Indústria (NICP, NATO Industry Cyber Partnership)[31], cujo objetivo vai ao encontro da defesa cibernética por meio da cooperação mútua, em particular com o compartilhamento de informações sobre ameaças para "prevenir, responder e se recuperar de ataques cibernéticos". Numa parceria com o setor privado, pois é onde se encontra o maior número dos sistemas de informação, este projeto visa melhorar, desenvolver e oferecer soluções técnicas para a defesa cibernética. Com isto, com o objetivo de manter sua vantagem tecnológica, a OTAN enfatiza seu apoio à inovação junto à iniciativa privada. Em 2016, foram investidos 750 milhões de euros em financiamentos de projetos, parcerias e novos negócios. Em 2017, este valor aumentou para 851 milhões de euros. Já em 2018, o valor investido pode alcançar os 950 milhões de euros. O planejamento para 2019 é que este valor atinja um bilhão de euros e chegue aos 1,1 bilhões de euros em 2020. Com todos estes investimentos, não só a Engenharia de Software, mas todos os setores relacionados às Tecnologias da Informação e Comunicação estão sendo beneficiados.

Este breve histórico apresentou resumidamente algumas instituições que foram criadas com a finalidade de atuarem em prol da Engenharia de Software, onde importantes contribuições foram e são feitas à área (exceto ACM, CMU, IBM, IEEE e OTAN, das quais já existiam antes do surgimento da Engenharia de Software), mas a história da Engenharia de Software abrange um espectro muito maior do que este aqui apresentado. Muitos outros nomes fazem parte de todo o seu contexto histórico. Porém, toda esta abrangência está além do contexto desta tese.

O resultado de todas estas contribuições (citadas ou não aqui) culminou no estado da arte da Engenharia de Software.

3.2. **Estado da arte da Engenharia de Software**

Em sua essência, a Engenharia de Software é formada por um conjunto de métodos, processos e ferramentas. Os métodos descrevem como aplicar as atividades e as técnicas que resultam no software desenvolvido. Os processos organizam os métodos para

[31] www.nicp.nato.int

que estes sejam aplicados de maneira ordenada e disciplinada, para que o software seja desenvolvido de maneira previsível e controlada. As ferramentas são os recursos computacionais que automatizam total ou parcialmente os métodos e os processos[32].

As atividades e as técnicas descritas nos métodos objetivam a realização da análise, do projeto, da construção, dos testes, da entrega e da manutenção do software, como também do gerenciamento do seu projeto de desenvolvimento e de evolução[33].

A análise (muitas vezes também chamada de Análise de Requisitos e, principalmente, Engenharia de Requisitos) consiste em definir o que o software deve fazer e também o que ele não vai fazer. Determina e especifica os seus requisitos. Para isso, é necessário compreender o contexto do mundo real em que ele atuará, juntamente com suas oportunidades de negócio e seus problemas a serem solucionados, para que sejam estabelecidas suas capacidades e restrições.

O projeto analisa os requisitos e produz a estrutura interna do software, sua arquitetura, que servirá de base e referência à sua construção. Com o foco nas oportunidades de negócio e nas soluções dos problemas identificados durante a análise, descreve os componentes, as interfaces e todas as características necessárias para que o software seja construído de modo que atenda fielmente aos seus requisitos.

A construção cria a versão operacional do software, conforme sua arquitetura definida no projeto. Esta atividade envolve a codificação, a verificação, a depuração e a integração dos itens de software produzidos.

Os testes consistem na avaliação dinâmica e permanente da qualidade de tudo aquilo que é gerado durante a aplicação da Engenharia de Software, constatando o que de fato está correto e identificando não apenas falhas, mas também possíveis pontos fracos, negligências, contradições, omissões e ambiguidades que possam resultar em erros quando o software entrar em operação.

A entrega disponibiliza o software para os usuários finais. Isto envolve garantir que ele está pronto para entrar em operação, seus manuais e todos os seus materiais de apoio e suporte estão concluídos e já podem ser acessados, seus usuários já sabem como

[32] Pressman, Roger S. Maxim, Bruce R. Engenharia de Software: uma abordagem profissional. Tradução de João Eduardo Nóbrega Tortello. 8 ed. Porto Alegre: AMGH, 2016. p. 14-16.
[33] Bourque, Pierre. & Fairley, Richard E. (Dick), eds. SWEBOK v3.0: Guide to the Software Engineering Body of Knowledge. IEEE Computer Society, 2014.

operá-lo e o ambiente em que ele será usado já se encontra configurado adequadamente para sua instalação.

A manutenção funciona como um período de garantia ou suporte após a entrega do software. É o período em que o software entra efetivamente em operação pelos seus usuários finais e onde correções e melhorias podem ser identificadas, resultando em modificações que possibilitem sua evolução e preservem sua integridade. Toda manutenção feita no software resulta também em modificação na documentação produzida durante o desenvolvimento deste software.

O gerenciamento é uma atividade técnica e administrativa que tem como objetivo planejar, controlar e medir sistematicamente as atividades da Engenharia de Software executadas, com o intuito de garantir que o software seja feito de acordo com os prazos, custos e qualidade estabelecidos e que atenda completamente seu propósito (e somente a ele).

Em relação aos processos, são eles que organizam e combinam a aplicação dos métodos. São roteiros que estabelecem a sequência do que deve ser feito, os artefatos que devem ser produzidos e os marcos de verificações. Eles fazem com que os métodos interajam entre si de modo que a Engenharia de Software seja aplicada eficazmente.

Atualmente, a demanda é que o software seja desenvolvido mais rapidamente, sem que sua qualidade seja afetada. Isso favorece a utilização dos chamados processos ágeis, os quais sequenciam a aplicação dos métodos de modo que eles sejam executados num menor intervalo de tempo (agilidade aqui não vem a ser pressa, mas otimização do tempo). Isso implica, necessariamente, na menor produção de documentos e na formação de equipes pequenas trabalhando localmente[34].

Por outro lado, o software vem se tornando cada vez mais complexo. Por isto, é natural que seu desenvolvimento consuma mais tempo e, por essa razão, a abordagem dos processos ágeis pode não ser a melhor maneira de aplicar a Engenharia de Software. Neste caso, a adoção de um processo formal pode ser a melhor escolha, pois permite uma maior

[34] Statpathy, Tridibesh. A Guide to the SCRUM BODY OF KNOWLEDGE (SBOK GUIDE). SCRUMstudy, 2016.

produção de documentos e a formação de grandes equipes trabalhando geograficamente distribuídas[35].

Qualquer que seja a abordagem adotada (ágil ou formal) o processo deve ser adaptado ao projeto. Deve-se compreender as necessidades do projeto para estabelecer o que do processo será aplicado e em qual nível de detalhamento.

No caso das ferramentas, estas são recursos computacionais que vão automatizar total ou parcialmente a utilização e aplicação dos métodos e dos processos para apoiar a produção do software. Atualmente, há ferramentas para todas as atividades da Engenharia de Software. Enquanto algumas trabalham em tarefas específicas, outras abrangem maior quantidade de tarefas. Há a possibilidade de integrar ferramentas para que estas apoiem todas as atividades da Engenharia de Software, formando assim a Engenharia de Software Auxiliada por Computador, que são as ferramentas CASE (*Computer-Aided Software Engineering*). No mercado são encontrados diversos tipos de ferramentas e fornecedores, onde os preços variam muito, havendo, inclusive, as versões gratuitas[36].

Junto a todo este espectro técnico da Engenharia de Software, com seus métodos, processos e ferramentas, encontram-se outras áreas do conhecimento mais voltadas aos fatores humanos, tais como os éticos, políticos, sociais, psicológicos, culturais e organizacionais, os quais vêm se incorporando cada vez mais à Engenharia de Software e contribuindo para que o software atinja seu principal objetivo que, em última análise, é: melhorar a vida das pessoas (físicas e jurídicas)[37].

---

[35] Tamburri , Damian A., Kruchteny, Philippe, Lago, Patricia, Vliet, Hans van. Orgaanizational Social Structures for Software Engineering. ACM Computing Surveys, Vol. 46, No. 1, Article 3. 2013, p. 2.

[36] O Anexo A contém informações mais detalhadas sobre algumas ferramentas de apoio à Engenharia de Requisitos. Mai informações sobre ferramentas também são encontradas ao longo das referências destacadas ao longo desta tese.

[37] Goguen, Joseph A. Formal Methods and Social Context in Software Development. Calhoun: The NPS Institutional Archive DSpace Repository, 1995. Cukierman, Henrique Luiz. Teixeira, Cássio. Prikladnicki, Rafael. Um Olhar Sociotécnico sobre a Engenharia de Software. RITA – Revista de Informática Teórica e Aplicada, Volume XIV, Número 2, 2007. Tamburri , Damian A., Kruchteny, Philippe, Lago, Patricia, Vliet, Hans van. Organizational Social Structures for Software Engineering. ACM Computing Surveys, Vol. 46, No. 1, Article 3, 2013. Tamburri , Damian A., Kruchteny, Philippe, Lago, Patricia, Vliet, Hans van. What Is Social Debt in Software Engineering? IEEE, CHASE 2013. LeBlanc, Rich & Sobel, Ann. Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering – A Volume of the Computing Curricula Series. IEEE Computer Society and Association for Computing Machinery, 2015. Trainer, Erik H. Kalyanasundaram, Arun. Herbsleb, James D. E-Mentoring for Software Engineering: A Socio-technical Perspective. ICSE – International Conference on Software Engineering, 2017, disponível em: http://www.cs.cmu.edu/~etrainer/papers/trainer-icse2017-ementoringSE.pdf.

4. ENGENHARIA DE REQUISITOS: APRESENTAÇÃO E PROBLEMAS

4.1. **Apresentação da Engenharia de Requisitos**

Para desenvolver um software, inicialmente deve-se estabelecer o seu propósito e definir o seu comportamento para que ele atinja as necessidades do negócio em que atuará. Isso implica em detalhar as tarefas que executará, as informações que utilizará e produzirá, as regras que obedecerá e as interações que fará com o seu ambiente externo (usuários finais, softwares e hardwares). Todas essas atividades são responsabilidades da Engenharia de Requisitos e resultam na visão do projeto do software.

O software deve ser projetado e construído de acordo com a visão de seu projeto e também deve ser avaliado e aprovado conforme os critérios estabelecidos por esta visão. Portanto, a qualidade do software vai refletir, no máximo, a qualidade de sua visão de projeto.

A primeira tarefa a ser realizada pela Engenharia de Requisitos é chamada de Elicitação de Requisitos, que visa compreender os problemas que o software deverá resolver e/ou as oportunidades de negócio a serem usufruídas com o seu apoio. Para isto, deve-se identificar quais são as pessoas interessadas no desenvolvimento do software (partes interessadas[38]) e, com elas, iniciar o processo de concepção dos requisitos de software, o qual é realizado por meio de uma comunicação efetiva com todas essas pessoas para que seja estabelecido o escopo do projeto, que descreve o software que será desenvolvido e a priorização das necessidades de negócio que devem ser atendidas e satisfeitas por ele.

Por envolver diversas pessoas, a Elicitação de Requisitos é considerada uma tarefa muito difícil de ser executada, pois requer que o engenheiro de requisitos obtenha uma visão completa das oportunidades de negócio e dos problemas a serem solucionados, a partir de informações fragmentadas e muitas vezes difíceis de serem identificadas porque algumas dessas pessoas não as têm documentadas, ou as realizam de maneira tácita, ou porque têm dificuldade em descrevê-las. Há também situações em que algumas pessoas,

[38] Pessoas e grupos de pessoas, podendo ser usuários, clientes, patrocinadores, analistas de mercado, procuradores, dentre outros, que interagem mais frequentemente com o engenheiro de requisitos. São as fontes dos requisitos e do financiamento do projeto de desenvolvimento do software.

por diversos motivos e razões, não têm interesse em colaborar e, por isto, passam informações erradas, conflitantes ou incompletas.

Para coletar e obter os esclarecimentos das informações necessárias dessas pessoas, o engenheiro de requisitos utiliza uma ou mais dessas técnicas durante a Elicitação de Requisitos: entrevistas (conversas individuais e espontâneas, de formatos livres, que descrevem as tarefas das pessoas), cenários (conversas individuais e de formatos estruturados que possibilitam roteirizar as tarefas das pessoas), protótipos (técnica que possibilita a representação ilustrativa das tarefas das pessoas), reuniões facilitadas (conversas realizadas com grupos de pessoas e mediadas por um facilitador que registra as decisões e depois as validam e as refinam novamente com este grupo de pessoas, ficando nesse processo até chegarem a um consenso), observações (onde o engenheiro de requisitos imerge no ambiente de trabalho das pessoas e aprende a realizar suas tarefas), histórias de uso (descrições curtas que expressam as expectativas das pessoas em relação ao software, mas que contém informações sobre os benefícios que obterão com a realização de suas tarefas), ou qualquer outra técnica que também ajude na compreensão das tarefas das pessoas[39].

Conforme os requisitos forem sendo concebidos, o escopo do projeto vai sendo elaborado, sendo essencial que nele estejam descritos: (1) o domínio da aplicação, que vem a ser a abordagem ontológica que servirá de base conceitual para todos os requisitos e que também ajudará na formulação do objetivo do projeto. (2) Todos os pontos de vista das pessoas interessadas, em relação aos problemas, impactos, soluções e oportunidades identificadas. (3) As regras do negócio que definirão e restringirão os aspectos estruturais e comportamentais do software. (4) O ambiente operacional em que o software será executado. Isto é importante porque ajuda a entender se haverá alguma restrição em relação ao projeto arquitetônico do software, além de ajudar a avaliar a viabilidade econômica de seu desenvolvimento. (5) O ambiente organizacional, para que processos de negócio, culturas e políticas internas, já existentes na organização, não sofram alterações não planejadas.

[39] O Anexo B contém informações mais detalhadas sobre estas técnicas. Mai informações sobre elas também são encontradas em: Técnicas para levantamento de requisitos (https://www.devmedia.com.br/tecnicas-para-levantamento-de-requisitos/9151), Engenharia de Requisitos – Técnicas (https://brunobrum.wordpress.com/2011/04/27/principais-tecnicas-de-levantamento-de-requisitos-de-sistemas/) e Análise e Especificação de Requisitos (https://www.dimap.ufrn.br/~jair/ES/c4.html).

A segunda tarefa a ser realizada pela Engenharia de Requisitos é chamada Análise de Requisitos. Em linhas gerais, essa tarefa deve produzir a descrição dos requisitos com precisão suficiente para que sejam validados, a implementação do software verificada e os custos do projeto estimados. Ao final desta tarefa, os requisitos já deverão estar formalizados (se não todos, ao menos os mais críticos).

Durante a realização dessa tarefa, os requisitos devem ser priorizados (em relação aos seus graus de importância quanto às metas gerais do software, para atingir seus objetivos) e classificados como requisitos funcionais (que descrevem as funções que o software deverá executar – são os recursos que formarão as suas capacidades) e não funcionais (que descrevem os requisitos de qualidade e também as restrições impostas ao software – ou ao seu projeto de desenvolvimento).

A modelagem conceitual também faz parte desta tarefa, onde, primeiramente, deve-se elaborar um modelo contextual do software (o qual apresentará seu ambiente operacional e identificará suas interfaces com o meio ambiente). Para isto, são elaborados modelos que representam o domínio da oportunidade de negócio e do problema do mundo real a ser solucionado, demonstrando seus aspectos, relacionamentos e dependências, e também os conectando às soluções que serão dadas pelo software e estabelecendo a rastreabilidade entre estes. Todos esses modelos pertencem ao espectro da engenharia de software e, frequentemente, os mais utilizados pertencem à UML (*Unified Modeling Language*, Linguagem de Modelagem Unificada) sendo o Diagrama de Casos de Uso o mais adotado para a modelagem de requisitos, mas não se restringindo a este.

Durante a Alocação de Requisitos é feita a derivação da Arquitetura do Software. Não há a necessidade de realizar essa atividade nesse momento do projeto, mas em alguns casos, devido à sua complexidade, é importante sua realização antecipada a fim de se obter o mais cedo possível a associação entre a Engenharia de Requisitos e o Projeto de Software. Além disso, sua realização ajuda a detalhar mais (e melhor) os requisitos, existindo até a possibilidade de novos requisitos serem identificados. Assim como ocorre durante a modelagem conceitual, essa atividade é realizada elaborando-se diversos modelos onde, frequentemente, os mais utilizados também pertencem à UML.

Conforme os requisitos vão sendo descritos, poderão ocorrer alguns conflitos que deverão ser resolvidos. Para isso, deve-se realizar a atividade Negociação de

Requisitos, onde as partes interessadas devem chegar a um comum acordo como solucionarão esses casos. Não há critérios estabelecidos que ajudem nessa decisão, mas pode-se partir dos requisitos mais prioritários e/ou das soluções mais econômicas.

A terceira tarefa a ser realizada pela Engenharia de Requisitos é chamada Especificação de Requisitos, onde será produzido um documento que poderá ser sistematicamente revisado, avaliado e aprovado pelas pessoas interessadas no desenvolvimento do software. Este documento descreve a definição do sistema e dos seus requisitos (funcionais e não funcionais) a partir da perspectiva do domínio da aplicação, como também fornece uma base para estimar os custos, prazos e riscos do projeto, além de determinar os planos de verificação e validação do software. Dele são derivados e especificados os requisitos dos componentes a serem desenvolvidos para o software. Na maioria das vezes, esta especificação de requisitos faz parte do contrato de desenvolvimento do software celebrado entre o fornecedor e o cliente, o qual servirá de parâmetro para informar o que o software irá fazer e também o que ele não irá fazer.

Por se tratar de um documento que será lido por diversas pessoas de especialidades distintas, o comum é que ele seja escrito em uma linguagem natural, onde todos possam ter e compartilhar uma compreensão uniforme de seu conteúdo, com a maior precisão possível. Essa linguagem, no entanto, limita-se às habilidades e preferências dos autores e leitores deste documento.

Conforme cada requisito vai sendo especificado, é possível determinar seu tamanho e, a partir daí, estimar o tamanho do software que será desenvolvido. Essa estimativa de tamanho também pode ajudar a estimar o custo e o prazo do projeto de desenvolvimento do software. Atualmente, uma técnica muito utilizada para fazer essa estimativa é chamada (*Use Case Point*, Ponto por Caso de Uso), a qual se baseia nos casos de uso, atores e relacionamentos, além dos fatores técnicos e ambientais do projeto, para calcular e determinar as estimativas de tamanhos dos requisitos e, consequentemente, do software.

A quarta tarefa a ser realizada pela Engenharia de Requisitos é chamada Validação de Requisitos. Esta tarefa é executada para garantir que cada um dos requisitos foi entendido e descrito satisfatoriamente e que está compreensível, consistente e completo. Também deve-se avaliar se eles estão em conformidade com os padrões, cultura

e políticas do negócio em que será operado. Com isso, é possível garantir que o software está definido corretamente e que atingirá as necessidades para as quais deverá atender e suprir. Essa validação abrange todos os documentos produzidos durante a Engenharia de Requisitos; por isto, geralmente, ela é realizada paralelamente às três tarefas citadas anteriormente (Elicitação de Requisitos, Análise de Requisitos e Especificação de Requisitos).

A quinta e última tarefa a ser realizada é chamada Gestão de Requisitos. A Engenharia de Requisitos, assim como todo desenvolvimento de software, não é uma atividade estática. Sua dinâmica implica em constantes alterações que podem ocorrer a qualquer momento. Por isso, os requisitos estão sempre em evolução e suas atualizações devem ser sempre monitoradas para que os impactos das mudanças sejam avaliados (isso envolve avaliar custo, prazo e escopo do projeto) e para garantir que o software será desenvolvido de acordo com suas especificações. Geralmente, a Gestão de Requisitos é executada paralelamente à execução das demais tarefas já citadas anteriormente (Elicitação de Requisitos, Análise de Requisitos, Especificação de Requisitos e Validação de Requisitos).

4.2. **Problemas Relacionados ao Desenvolvimento de Software**

A Engenharia de Software é uma área do conhecimento humano relativamente nova (cinquenta anos). Mesmo assim, já contribuiu (e continua a contribuir) bastante para que o software seja produzido de maneira "sistemática, controlada e quantificável", por meio da aplicação de seu espectro de atividades combinadas e integradas. É notório o quanto o software evoluiu (e continua a evoluir) a partir das contribuições da Engenharia de Software. Por outro lado, também é notório o quanto o software precisa evoluir (e, consequentemente, a Engenharia de Software).

A partir de 1994, a organização *The Standish Group*[40] vem lançando regularmente o Relatório do Caos (*Chaos Report*)[41], o qual é uma referência muito conhecida e utilizada para se obter as informações que apresentam os motivos de maiores

[40] www.standishgroup.com

[41] The Standish Group Report: Chaos, ed. 1994, 1995, 2011, 2012, 2013, 2014 e 2015. Jean-Sébastien Vachon. Why Projects Fail. Inteloom, October 21, 2016.

destaques que contribuíram para levar os projetos de software ao sucesso ou ao insucesso, como também as ações necessárias para que os cenários apresentados sejam melhorados (nos casos dos insucessos) ou copiados (nos casos dos sucessos).

Já em sua primeira edição, este relatório alertou que 31,1% dos projetos foram cancelados antes de serem concluídos. Dos que foram concluídos, 52,7% extrapolaram, em média, 189% as estimativas de custos, alcançando trilhões de dólares em perdas. Em relação às especificações dos requisitos de software, estas não são refletidas completamente na maior parcela dos projetos concluídos e entregues. Quanto às falhas apresentadas, 48% dos executivos de TI confirmaram que, atualmente, elas são maiores e mais preocupantes, em comparação há cinco anos.

> The Standish Group research shows a staggering 31.1% of projects will be canceled before they ever get completed. Further results indicate 52.7% of projects will cost 189% of their original estimates. The cost of these failures and overruns are just the tip of the proverbial iceberg. The lost opportunity costs are not measurable, but could easily be in the trillions of dollars. One just has to look to the City of Denver to realize the extent of this problem. The failure to produce reliable software to handle baggage at the new Denver airport is costing the city $1.1 million per day. Based on this research, The Standish Group estimates that in 1995 American companies and government agencies will spend $81 billion for canceled software projects. These same organizations will pay an additional $59 billion for software projects that will be completed, but will exceed their original time estimates. Risk is always a factor when pushing the technology envelope, but many of these projects were as mundane as a driver's license database, a new accounting package, or an order entry system. On the success side, the average is only 16.2% for software projects that are completed on-time and onbudget. In the larger companies, the news is even worse: only 9% of their projects come in on-time and on-budget. And, even when these projects are completed, many are no more than a mere shadow of their original specification requirements. Projects completed by the largest American companies have only approximately 42% of the originally-proposed features and functions. Smaller companies do much better. A total of 78.4% of their software projects will get deployed with at least 74.2% of their original features and functions. This data may seem disheartening, and in fact, 48% of the IT executives in our research sample feel that there are more failures currently than just five years ago. The good news is that over 50% feel there are fewer or the same number of failures today than there were five and ten years ago. (Chaos Report, 1994, p. 2).

Outro dado surpreendente apresentado neste relatório diz respeito ao retrabalho, onde 94% dos projetos que começam devem ser reiniciados devido às aplicações fracassarem em fornecer as características esperadas. Mesmo assim, dos projetos concluídos, apenas uma média de 37% dos recursos e funções inicialmente especificados foram entregues e, dos projetos que se encontram em desenvolvimento, apenas 12% deles estão dentro do prazo e do orçamento.

> The figures for failure were equally disheartening in companies of all sizes. Only 9% of projects in large companies were successful. At 16.2% and 28%

> respectively, medium and small companies were somewhat more successful. [...] The most projects, 37.1%, were impaired and subsequently canceled [...] One of the major causes of both cost and time overruns is restarts. For every 100 projects that start, there are 94 restarts. This does not mean that 94 of 100 will have one restart; some projects can have several restarts. For example, the California Department of Motor Vehicles project, a failure scenario summarized later in this article, had many restarts. [...] Currently, the 365 companies have a combined 3,682 applications under development. Only 431, or 12%, of these projects are on-time and on-budget. (Chaos Report, 1994, p. 3-4).

Já para obter sucesso nos projetos de software, este relatório concluiu que os três principais fatores são: o envolvimento dos usuários, o suporte da gerência executiva e uma plena declaração dos requisitos.

> The most important aspect of the research is discovering why projects fail. To do this, The Standish Group surveyed IT executive managers for their opinions about why projects succeed. The three major reasons that a project will succeed are user involvement, executive management support, and a clear statement of requirements. There are other success criteria, but with these three elements in place, the chances of success are much greater. Without them, chance of failure increases dramatically. (Chaos Report, 1994, p. 5).

De acordo com este relatório, os dez motivos pelos quais um projeto terá sucesso são destacados na tabela 3, em ordem de importância:

Tabela 3 - Os dez principais fatores de sucesso para os projetos de software

Relatório do Caos 1994 Os dez principais fatores de sucesso para os projetos de software	
Fator de sucesso	% de responsabilidade
1. Envolvimento dos usuários	15.9%
2. Suporte da gerência executiva	13.9%
3. Clara declaração dos requisitos	13.0%
4. Planejamento adequado	9.6%
5. Expectativas realistas	8.2%
6. Marcos menores do projeto	7.7%
7. Equipe competente	7.2%
8. Propriedade	5.3%
9. Visão e objetivos claros	2.9%
10. Trabalhador diligente, pessoal focado	2.4%
Outros	13.9%

Fonte: Chaos Report, 1994, p. 5.

Esta entrevista também revelou os dez principais motivos que contribuem para que os projetos de software sejam concluídos com orçamento e prazo excedidos, e ainda ofereçam menos recursos e funções dos que foram originalmente especificados. Essas informações são apresentadas na tabela 4, em ordem de importância:

Tabela 4 - Os dez principais fatores para exceder estimativas e entregar menos funções

Relatório do Caos 1994 Os dez principais fatores que contribuem para os projetos excederem orçamento e prazo e oferecerem menos recursos e funções dos que foram originalmente especificados	
Fatores que aumentam orçamento e prazo	% de Responsabilidade
1. Falta de envolvimento dos usuários	12.8%
2. Requisitos e especificações incompletos	12.3%
3. Mudanças dos requisitos e especificações	11.8%
4. Falta de suporte da gerência executiva	7.5%
5. Incompetência tecnológica	7.0%
6. Falta de recursos	6.4%
7. Expectativas irrealistas	5.9%
8. Objetivos pouco claros	5.3%
9. Estimativas de prazo irrealistas	4.3%
10. Nova tecnologia	3.7%
Outros	23%

Fonte: Chaos Report, 1994, p. 5.

Essa entrevista também revelou os dez principais motivos que contribuem para o cancelamento antecipado do projeto de software. A tabela 5 os apresenta em ordem de importância:

Tabela 5 - Os dez principais fatores para o cancelamento antecipado dos projetos

Relatório do Caos 1994 Os dez principais fatores que contribuem para o cancelamento antecipado dos projetos	
Fator de cancelamento	% de responsabilidade
1. Requisitos incompletos	13.1%
2. Falta de envolvimento dos usuários	12.4%
3. Falta de recursos	10.6%
4. Expectativas irrealistas	9.9%
5. Falta de suporte da gerência executiva	9.3%
6. Mudança dos requisitos e especificações	8.7%
7. Falta de planejamento	8.1%
8. Não precisou mais	7.5%
9. Falta de gerenciamento de TI	6.2%
10. Analfabetismo tecnológico	4.3%
Outros	9.9%

Fonte: Chaos Report, 1994, p. 6.

Por fim, essa primeira edição do Relatório do Caos encerra sua contribuição informando que os problemas relacionados aos projetos de software são assustadores e como solução sugere a realização de projetos menores e menos complexos, a serem executados em menores intervalos de tempo, fazendo com que o software evolua conforme pequenos elementos (componentes) vão sendo construídos.

> Notwithstanding, this study is hardly in-depth enough to provide a real solution to such a daunting problem as the current project failure rates. Application software projects are truly in troubled waters. In order to make order out of the chaos, we need to examine why projects fail. Just like bridges, each major software failure must be investigated, studied, reported and shared. Because it is the product of the ideas of IT managers, the "Success Potential" chart can be a useful tool for either forecasting the potential success of a project or evaluating project failure. Research at The Standish Group also indicates that smaller time frames, with delivery of software components early and often, will increase the success rate. Shorter time frames result in an iterative process of design, prototype, develop, test, and deploy small elements. This process is known as "growing" software, as opposed to the old concept of "developing" software. Growing software engages the user earlier, each component has an owner or a small set of owners, and expectations are realistically set. In addition, each software component has a clear and precise statement and set of objectives. Software components and small projects tend to be less complex. Making the projects simpler is a worthwhile endeavor because complexity causes only confusion and increased cost. There is one final aspect to be considered in any degree of project failure. All success is rooted in either luck or failure. If you begin with luck, you learn nothing but arrogance. However, if you begin with

> failure and learn to evaluate it, you also learn to succeed. Failure begets knowledge. Out of knowledge you gain wisdom, and it is with wisdom that you can become truly successful. (Chaos Report, 1994, p. 9).

Em suas edições subsequentes, o Relatório do Caos continuou apontando praticamente os mesmos problemas e soluções apontados em sua primeira edição, com algumas variações pontuais, mostrando sinais de melhorias em uns aspectos e sinais de retrocessos em outros aspectos.

Em 2015 os dados continuaram sendo preocupantes e desafiadores. Ao avaliar mais de cinquenta mil projetos, apenas 29% foram bem-sucedidos (foram entregues dentro do prazo, custo e objetivos), 19% foram cancelados antecipadamente e 52% excederam orçamento e prazo e ofereceram menos recursos e funções dos que foram originalmente especificados e, por isto, tiveram seus projetos originais modificados. A título de comparação, vê-se que na edição de 1994 esses percentuais são, respectivamente, 16.2%, 31.1% e 52.7%, mostrando uma importante evolução em relação aos projetos bem-sucedidos e aos projetos cancelados antecipadamente, mas mantendo praticamente a mesma taxa de projetos modificados (ou seja, que excederam o orçamento e prazo e ofereceram menos recursos e funções dos que foram originalmente especificados).

A tabela 6 apresenta as variações dessas mesmas taxas ocorridas entre os anos 2012 e 2015, onde se pode perceber que há uma estabilidade das informações ao longo dos anos:

Tabela 6 - Taxas de projetos bem-sucedidos, modificados e cancelados, de 2012 a 2015

Relatório do Caos, edições 2012 a 2015 Taxas de projetos bem-sucedidos, modificados e cancelados				
Projetos	2012	2013	2014	2015
Bem-sucedidos	27%	31%	28%	29%
Modificados	56%	50%	55%	52%
Cancelados	17%	19%	17%	19%

Fonte: Inteloom, 2016, p. 2.

Dos fatores que contribuem para o projeto de software ser bem-sucedido, desde 2011, de maneira bem destacada, surgiram dois novos: Maturidade Emocional e Processo Ágil.

Maturidade Emocional: Abrange o estado emocional do ambiente do projeto. Os projetos são resolvidos dentro do ecossistema. Um ecossistema saudável produz projetos mais bem-sucedidos. Equipes com alta inteligência emocional são três vezes mais propensas a ter sucesso do que aquelas com habilidades moderadas ou quase nulas. Equipes que podem se comunicar e colaborar eficazmente são melhores nas resoluções dos problemas.

> We must consider that emotional maturity is the ability and capacity to perceive, assess, manage, and direct the emotions and actions of the project stakeholders .It is the ability to rise above petty politics and provide both balance and a transparent view .It is the ability to identify and remove unnecessary requirements, as well as the aptitude to deliver bad news and accept critical feedback .It is the skill to recognize and deal with the Five Deadly Sins, which are over ambition, arrogance, ignorance, abstinence, and fraudulence .It is the ability to acknowledge and endorse the CHAOS Commandments of project management, which are community, honor, awareness, objective, and superior. The Five Deadly Sins are part of all project ecosystems, healthy and unhealthy .In fact, a project cannot be successful without them .For example, every project will have an impatient visionary, which results in over ambition the first deadly sin .It is how you deal with each of these sins that will determine the success or failure of a project .The majority of IT executives believe that overcoming the Five Deadly Sins is too difficult .For example, two-thirds of IT executives believe that overcoming abstinence skills is slightly difficult or not difficult for them to master .But what does give them difficulty is recognizing this in time to prevent the project from failing. The CHAOS Commandments are proactive techniques used to promote maturity in emotions within project teams .By taking a proactive approach the organization avoids the gnarly problem of recognizing and overcoming the Five Deadly Sins .The five CHAOS Commandments let the project team set the agenda, manage expectations, and move the process along .The CHAOS Commandments start and end with managing not only the real project's outcome, but also the perceived project's outcome .It is also knowing when and when not to do something. CHAOS Commandments help the organization improve the skills to be self-aware, socially aware, self-managed, and to manage relationships. (Chaos Report, 2011, p. 17).

Processo Ágil: Os processos ágeis aumentam o engajamento e estão focados na entrega de software que atenda às expectativas, em períodos mais curtos de tempo. No geral, houve uma melhoria expressiva e significativa nas taxas de sucesso dos projetos de software que adotaram processos ágeis, a saber: grandes projetos: de 3% para 18%, projetos médios: de 7% para 27% e pequenos projetos: de 44% para 58%. Com isto, pode-se dizer que o processo ágil é a melhor solução para as falhas dos projetos de desenvolvimento de software.

> The agile process is the universal remedy for software development project failure. Software Applications developed through the agile process have three times the success rate of the traditional waterfall method and a much lower percentage of time and cost overruns. The secret is the trial and error and delivery of the iterative process. Software should be built in small, iterative steps with small, focused teams. The project team delivers functionality in small bites or steppingstones. A steppingstone, also known as an iteration, is a small but

> significant deliverable .A steppingstone activity allows for tangible inspection, either visually or hands-on. Steppingstones are a key driver for the success of the agile and iterative software development process. Steppingstones lead to more deliverables or indicate the project is not on the right track .Steppingstones are powerful because they allow for rapid feedback, creation of feature velocity, and accelerated user training and acceptance. Time is the enemy of all projects, but the agile process turns time into an ally. The agile process does this by using standard time intervals or time boxes .Time boxing sets deadlines and establishes a fixed amount of time in which to complete a steppingstone or iteration. A set number of steppingstones or iterations are then grouped into a micro project. Project initiatives that are rigid run the greatest risk of dissolution .Yet, having a flexible, formal process has been proven to greatly improve the success rate .A good project management methodology builds in interaction among team members as well as the user community, and an agile process improves this interaction .Quickness and velocity are vital to an agile process, and that encompasses feedback. However, a major advantage of the agile process is the closeness of the executive sponsor, or in agile terms, the owner. The owner's role is to own the responsibility for the resolution of the project. (Chaos Report, 2011, p. 25).

Em relação aos requisitos de software (tema central de discussão desta tese) eles estiveram sempre em evidência, tanto em relação aos fatores de sucesso (plena declaração dos requisitos) como em relação aos fatores que contribuem para os projetos excederem orçamento e prazo e oferecerem menos recursos e funções das que foram originalmente especificadas (requisitos e especificações incompletos e mudanças dos requisitos e especificações) e também em relação aos fatores que levam ao cancelamento antecipado dos projetos de software (requisitos incompletos).

> Poorly defined requirements can be a guaranteed path to project failure. A lack of detailed and / or accurate descriptions of what needs to be delivered and when, leads to a myriad of issues in a project. Poorly defined requirements also leads to the dreaded "scope creep". Since there was a lack of understanding of the deliverables in the first place, projects will begin to mutate into unrecognizable beasts that no one had planned on. (Inteloom, 2016, p. 4).

Além de serem bem definidos, os requisitos também devem ser bem gerenciados para garantirem a qualidade total do projeto de software.

> It is absolutely essential for the project manager to ensure that the primary requirements for the project are as detailed as possible with all relevant agents. The GOOM[42] model emphasizes the importance of approaching requirement gathering as an iterative process, as the requirements will evolve as the project evolves. They need to be reviewed during the execution phase to ensure that any changes are captured and adjusted for as soon as possible, and that the output satisfies GOOM's commitment to total quality. (Inteloom, 2016, p. 7).

[42] [42] GOOM (Gestion Orientée Objet, Gestão Orientada a Objetos) ou OOM (Object Oriented Management, Gerenciamento Orientado a Objetos) é um modelo de gerenciamento de projetos que visa a qualidade total. São marcas registradas da Inteloom Incorporated (www.inteloom.com).

Além dos relatórios do *Standish Group*, os problemas da Engenharia de Software são encontrados frequentemente na literatura, ora de maneira mais genérica, ora de maneira mais específica.

Quando de maneira mais genérica, mencionam-se os efeitos causados pelos softwares mal construídos, os quais funcionam de maneira insatisfatória e suas estimativas de custos e prazos são extrapoladas:

> Frequentemente, os sistemas funcionam, mas não exatamente como se esperava. Sempre ouvimos histórias de sistemas que funcionam de maneira insatisfatória. [...]. No começo da década de 1.980, a Receita Federal dos Estados Unidos (United States Internal Revenue Service – IRS) contratou a empresa Sperry Corporation para construir um sistema automatizado de processamento de formulários de impostos federais. De acordo com o jornal americano Washington Post, "o sistema se mostrou inadequado à carga de trabalho, custou cerca de duas vezes o esperado e deve ser logo substituído" (Sawyer, 1985). Em 1985, foi necessário adicionar US$ 90 milhões aos US$ 103 milhões que já haviam sido pagos pelos equipamentos da Sperry. Além disso, o problema acarretou o atraso das restituições da IRS aos contribuintes, o que forçou a IRS a pagar US 40,2 milhões em juros e US$ 23 milhões em horas extras para os funcionários que tentaram compensar o estrago. Em 1996, a situação não havia melhorado. O jornal americano Los Angeles Times publicou, em 29 de março daquele ano, que ainda não havia nenhum plano para a modernização dos computadores da IRS, somente um relatório técnico de 6000 páginas. O congressista americano Jim Lightfoot chamou o projeto de "um fiasco de quatro bilhões de dólares, que está afundando devido ao planejamento inadequado". (Lawrence, 2004, p. 5-6).

As falhas de funcionamento que os softwares apresentam também são mencionadas frequentemente:

> A comunidade médica ficou chocada quando o Therac-25, uma máquina de terapia de radiação e de raio X, causou a morte de diversos pacientes por uma falha em seu funcionamento. Os projetistas do software não haviam previsto o uso de algumas teclas de seta de maneiras não convencionais; como consequência, o software manteve os ajustes mais altos e emitiu uma dose altamente concentrada de radiação quando, na verdade, se pretendia liberar baixos níveis de radiação (Leveson e Turner, 1993). (Lawrence, 2004, p. 6).

Os problemas relacionados às características e funcionalidades dos softwares (ou seja, questões relacionadas aos requisitos de software) são outras preocupações geralmente citadas quando o assunto é "qualidade do software":

> A necessidade de maior qualidade de software surgiu realmente a partir do momento que o software ficou cada vez mais integrado a todas as atividades de nossas vidas. Na década de 1990, as principais empresas reconheciam que bilhões de dólares por ano eram desperdiçados em software que não apresentava as características e as funcionalidades prometidas. Pior ainda, tanto o governo quanto as empresas estavam cada vez mais preocupados com o fato de que uma falha grave de software poderia inutilizar importantes infraestruturas, aumentando o custo em dezenas de bilhões. Na virada do século, a CIO Magazine anunciou em manchete: "Chega de desperdiçar US$ 78 bilhões por ano", lamentando o fato de que "as empresas americanas gastavam bilhões em

> software que não fazia o que supostamente deveria fazer" [Lev01]. (Pressman, 2016, p. 412).

Quando de maneira mais específica, evidenciam-se os pontos fracos a serem superados, conforme os autores comentam sobre os métodos e os processos da Engenharia de Software como sendo os grandes vilões das "falhas dos softwares":

> Ainda existem muitos relatos e projetos de software que deram errados e resultaram em "falhas de software". A engenharia de software é criticada por ser inadequada para o desenvolvimento moderno de software. [...] conforme novas técnicas de engenharia de software nos auxiliam a construir sistemas maiores e mais complexos, as demandas mudam. Os sistemas têm de ser construídos e entregues mais rapidamente; sistemas maiores e até mais complexos são requeridos; sistemas devem ter novas capacidades que antes eram consideradas impossíveis. Como os métodos de engenharia de software existentes não conseguem lidar com isso, novas técnicas de engenharia de software precisam ser desenvolvidas para atender a essas novas demandas. (Sommerville, 2011, p. 2-3).

Neste sentido, o que se sugere é o desenvolvimento de novas soluções para a construção de software:

> Todos os modelos de construção de software que apareceram foram ruindo no resultado final e, ainda hoje, mesmo os mais decantados processos, apresentam falhas, ainda não superadas. [...]. Somos capazes de apontar os motivos, mas não somos tão felizes em apontar as soluções para esta problemática. A fórmula secreta da boa construção de software ainda não foi desenvolvida, e isso ocorre em detrimento de toda a tecnologia existente. (Ernani, 2010, p. 1-2).

Entre os anos 2017 e 2018, foram publicados artigos científicos que alertaram sobre a importância crucial da Engenharia de Requisitos para o desenvolvimento de software e enfatizaram problemas críticos ainda existentes nesta atividade, principalmente os relacionados à Elicitação de Requisitos.

Em 2017, a dificuldade de elicitar os requisitos de software foi evidenciada como um problema sério da Engenharia de Software, tendo despertado o interesse de muitos estudiosos para solucioná-los.

> Requirement elicitation is a process of determining the problems and needs of the customer, so that software developers can construct a system that actually resolve customer problems and address their needs. Understanding requirements is a difficult task because it involves natural language to interact with the end-users, and end users may provide incomplete and ambiguous requirements. Requirements are volatile in nature, and they change over the period of time. There are some social issues also, that affect requirement elicitation process [...]. Number of authors study about requirements elicitation problem, and they confirm that the problem is at much larger scale. Surveys revealed that one third of the projects started were never completed, and one half of them succeeded only partially. The reason behind such failure is poor requirement elicitation. (Saurabh, 2017, p. 2).

Em 2018, a Elicitação de Requisitos continuou sendo destacada como sendo uma atividade muito complexa e uma das mais críticas para o desenvolvimento de software.

> Requirements elicitation is a process of searching, revealing, acquiring and detailing of requirements for computer based system. It is a complex process involving many activities with a variety of available techniques, approaches, and tools for performing them. During this activity, the client and developers define the purpose of the system to be developed. Requirements elicitation is the first and one of the most critical activities during software development process. Quality requirements are essential to the success of any software development project, regardless of whether it is based on agile or traditional methodologies. If this activity is not performed in a well-controlled manner and error remains in this phase, the final product may not satisfy the needs of the client and end users which will ultimately make the system unacceptable. (Theodros, 2018, p. 118).

Além dos relatórios do *Standish Group* e da literatura sobre Engenharia de Software, o SEI também afirma que a "complexidade dos sistemas está aumentando drasticamente" em escalas sem precedentes e sugere que novas demandas vêm surgindo e trazendo novos desafios à Engenharia de Software.

> "Given the issues with today's software engineering, how can we build the systems of the future that are likely to have billions of lines of code?" (Site oficial do SEI[43]).

Buscado responder a esta questão, foi elaborado o relatório *Ultra-Large-Scale Systems* (ULS[44], Sistemas de Grandes Escalas) cujo resultado confirma a existência de "lacunas fundamentais" na Engenharia de Software, que precisam ser resolvidas por meio de novas ideias e perspectivas diferentes das conhecidas e praticadas atualmente para se desenvolver softwares.

> Fundamental gaps in our current understanding of software and software development at the scale of ULS systems present profound impediments to the technically and economically effective achievement of the DoD goal of deterrence and dominance based on information superiority. These gaps are strategic, not tactical. They are unlikely to be addressed adequately by incremental research within established categories. Rather, we require a broad new conception of both the nature of such systems and new ideas for how to develop them. We will need to look at them differently, not just as systems or systems of systems, but as socio-technical ecosystems.We will face fundamental challenges in the design and evolution, orchestration and control, and monitoring and assessment of ULS systems. These challenges require breakthrough research. (ULS, 2006, p.ix).

[43] http://resources.sei.cmu.edu/library/asset-view.cfm?assetID=30519

[44] Northrop, Linda. Ultra-Large-Scale Systems: The Software Challenge of the Future. Software Engineering Institute. Pittsburg, PA, 2006.

Além dos sistemas ULS possuírem bilhões de linhas de código, outras características também se encontram em escalas muito grandes: número de pessoas que utilizam esses sistemas para diferentes finalidades, quantidade de informações armazenadas, acessadas e manipuladas, número de conexões e interdependências entre os componentes de software e número de elementos de hardware, além de tudo isto se encontrar geograficamente distribuído. Consequentemente, estes sistemas mudarão as atuais práticas de construção, aquisição, implantação, gerenciamento, documentação, utilização e evolução de software.

> The U. S. Department of Defense (DoD) has a goal of information dominance — to achieve and exploit superior collection, fusion, analysis, and use of information to meet mission objectives. This goal depends on increasingly complex systems characterized by thousands of platforms, sensors, decision nodes, weapons, and warfighters connected through heterogeneous wired and wireless networks. These systems will push far beyond the size of today's systems and systems of systemsby every measure: number of lines of code; number of people employing the system for different purposes; amount of data stored, accessed, manipulated, and refined; number of connections and interdependencies among software components; and number of hardware elements. They will be ultra-largescale (ULS) systems. [...]. The sheer scale of ULS systems will change everything. ULS systems will necessarily be decentralized in a variety of ways, developed and used by a wide variety of stakeholders with conflicting needs, evolving continuously, and constructed from heterogeneous parts. People will not just be users of a ULS system; they will be elements of the system. Software and hardware failures will be the norm rather than the exception. The acquisition of a ULS system will be simultaneous with its operation and will require new methods for control. These characteristics are beginning to emerge in today's DoD systems of systems; in ULS systems they will dominate. Consequently, ULS systems will place unprecedented demands on software acquisition, production, deployment, management, documentation, usage, and evolution practices. (ULS, 2006, p.ix).

Este relatório afirma que a Engenharia de Software atual é incapaz de enfrentar os desafios apresentados pelos sistemas ULS e que há a necessidade de pesquisas inovadoras, a serem realizadas de maneira multidisciplinar, por disciplinas que envolvam, por exemplo, a engenharia de software, a economia, os fatores humanos, a psicologia cognitiva, a sociologia, a engenharia de sistemas e a política de negócios.

> The proposed research does not supplant current, important software research but rather significantly expands its horizons. [...]. As a first step, we recommend the funding and establishment of a ULS System Research Startup Initiative, which over the course of the next two years would, among other things, work with others to conduct new basic research in key areas; oster the growth of a community of informed stakeholders and researchers; and formulate and issue an initial Broad Agency Announcement (BAA) to attract researchers with proven expertise in the diverse set of disciplines (e.g., software engineering, economics, human factors, cognitive psychology, sociology, systems engineering, and business policy) that are collectively required to meet the challenge of ULS systems. (ULS, 2006, p. xi).

O relatório também afirma que a atual compreensão em desenvolvimento de software apresenta profundos impedimentos para a realização técnica e econômica dos objetivos do DoD:

> The problem that the DoD now faces is clear. Fundamental gaps in our understanding of software and software development at the scale of ULS systems present profound impediments to the technically and economically effective achievement of the DoD goal of deterrence and dominance based on information superiority. (ULS, 2006, p. 4).

Por isto, o relatório continua a insistir e a enfatizar a necessidade de se investir em pesquisas inovadoras sobre conceitos, métodos e ferramentas para o desenvolvimento de software, pois nem mesmo as abordagens atuais de maiores destaques são suficientes para serem aplicadas aos sistemas ULS:

> This report presents a new perspective on problem formulations and an initial research agenda that we believe has the potential to lead to the required breakthroughs. What this report avoids is any suggestion that adequate solutions will be found solely by a straightforward extrapolation from today's technology, including high-visibility concepts such as service-oriented architecturesand model-based development. The depth of the gaps in current knowledge demands not just the incremental extension of existing work but also innovation, ranging from new conceptual models of the problem space to revolutionary solution approaches. (ULS, 2006, p. 4).

O desenvolvimento ágil também foi ponto de discussão diante às novas realidades impostas pelos sistemas ULS. Neste sentido, foram recomendadas pesquisas inovadoras sobre análise computacional, verificação de especificações, arquiteturas, projetos e implementações dos componentes destes tipos de sistemas que, por suas naturezas, são altamente descentralizados e distribuídos, indo contra as características atuais dos softwares desenvolvidos por meio das abordagens ágeis:

> The scale and complexity of problems to be solved by ULS systems mean that, in many cases, the requirements to be satisfied by the systems will not be adequately known until the systems are in use. Even then, as the system is put into operation, perceptions of the problem it is solving will change. Each attempt at a solution will give a deeper understanding of what the problem really is, leading to yet another attempt at a solution. Some problems addressed by ULS systems are likely to be so complex that there can be no fully satisfactory solution; requirements will never really converge. (ULS, 2006, p. 14).

Quanto aos requisitos de software, o relatório afirma que estes são um fator crítico para o sucesso dos projetos de software e que as deficiências e dificuldades nesta atividade são as principais causas de falhas nos projetos de software. Também afirma que essa criticidade aumenta consideravelmente com os sistemas ULS, pois as complexidades dos problemas a serem abordados por eles excederão as capacidades intelectuais humanas

de compreendê-los e dominá-los e, por isto, recomenda que sejam feitas pesquisas sobre este tema.

> Formulating system requirements is a critical success factor in software engineering, and shortcomings in this dimension are one leading cause of software project failures. Determining or discovering the requirements is also the software engineering activity that is the most domain specific, the least capable of being automated and formalized, and the least scalable. ULS systems will make these problems worse because of the scope of application domains that exceed the limits of human intellectual capabilities, the complexity and fragmentation of socio-economic processes and organizations that are highly decentralized and autonomous, and the sheer complexity of the problems being addressed. (ULS, 2006, p. 85).

Não só a dificuldade em determinar e descrever os requisitos de software foi citada como questão crítica quando o assunto é requisitos de software, mas o gerenciamento deles também recebeu especial atenção e, dada a complexidade de realizar esta tarefa, a sugestão ficou por conta da necessidade de automatizar todo o processo de gerenciamento dos requisitos de software:

> How will requirements reflect the changing nature of people within the ULS system over decades-long durations? While it is inevitable that requirements will change over time, requirements must be monitored and managed to ensure that no individual or organization can appreciably change the system without understanding, and perhaps getting approval from, the other participants. This implies the need for research into the archiving and retrieval of requirements (which will become enormous repositories over time), and more important, for research into automated support for analyzing these requirements so that the implications of new requirements can be assessed and requirements trends can be tracked. (ULS, 2006, p. 63).

Seguindo ainda a problemática do gerenciamento dos requisitos de software, as constantes mudanças que estes sofrerão também foi assunto de merecido destaque:

> Requirements can be known in advance and change slowly as experience with a system grows.We know that requirements for systems today are never completely understood in advance of building and using a system. Even so, most systems today are developed on the assumption that key requirements are sufficiently well understood that if a system is built to meet these requirements, it will be useful. But ULS systems are likely to encounter so-called wicked problems in which requirements are neither knowable in advance (because there is no agreement about what the problem is) nor stable (because each solution changes the problem, and no solution is considered to have "solved" the problem. In ULS systems, even more so than today, system development, operation, and usage will have to be based on a premise of continual change and (re)negotiation of user needs. (ULS, 2006, p. 14).

Ao olhar a combinação determinar, descrever e gerenciar os requisitos de software, o SEI conclui que a solução para tal complexidade jamais será totalmente satisfatória e que, por isto, tais requisitos nunca vão convergir a um estado imutável, pois sempre mudarão e sempre uma nova compreensão acerca deles será conhecida:

> The scale and complexity of problems to be solved by ULS systems mean that, in many cases, the requirements to be satisfied by the systems will not be adequately known until the systems are in use. Even then, as the system is put into operation, perceptions of the problem it is solving will change. Each attempt at a solution will give a deeper understanding of what the problem really is, leading to yet another attempt at a solution. Some problems addressed by ULS systems are likely to be so complex that there can be no fully satisfactory solution; requirements will never really converge. (ULS, 2006, p. 14).

Dada à complexidade dos sistemas ULS, eles ficarão em operação por muito tempo e, por isto, a tendência é que eles cresçam cada vez mais em tamanho e em complexidade, até se tornarem impossíveis de serem substituídos ou aposentados. Consequentemente, novos recursos serão adicionados a eles, para que sejam atendidos novos requisitos ou incorporadas novas tecnologias. Por esta razão, regras e políticas locais e gerais deverão ser criadas para acomodarem as mudanças e evoluções que estes sistemas sofrerão e suportarão:

> Another consequence of size is that ULS systems will be in service for a long time. Their size will make it impractical to replace or retire them. Instead, like very large systems today, they will continuously evolve to meet new and modified requirements and to incorporate new technologies. But we envision a different type of evolution than is typical of today's very large systems. By evolution, we mean change that is guided and constrained by rules and policies that allow local needs to be satisfied in local ways without destroying the integrity and value of the overall system. The evolution of a ULS system will be supported by different subgroups of stakeholders, each subgroup seeking to achieve a solution that fits its own needs, but under the guidance of general economic, technical, and political rules that limit the impact (both positive and negative) of any given change. (ULS, 2006, p. 15).

Dez anos após sua primeira edição, em Dezembro de 2016, o SEI lançou um novo relatório sobre os sistemas ULS[45], em que, em sua essência, o cenário inicialmente exposto não apresentou grandes evoluções, conforme se pode verificar em duas afirmações aqui destacadas.

A primeira afirmação encontra-se logo no início do relatório, a qual diz que o atual conhecimento em desenvolvimento de software apresenta sérios impedimentos para o desenvolvimento dos sistemas ULS e que, diante desta situação, é necessário investir em pesquisas inovadoras para o enfrentamento destes desafios que surgem a partir destes sistemas:

> Fundamental gaps in our current understanding of software and software development at the scale of ULS systems present profound impediments to the technically and economically effective achievement of information superiority. These gaps are strategic, not tactical. They are unlikely to be addressed adequately by incremental research within established categories. Rather, we

45 HISSAM, Scott...[et al.]. Ultra-Large-Scale (ULS) Systems: Socio-Adaptive Systems. Software Engineering Institute. Pittsburg, PA, 2016.

> require a broad new conception of both the nature of such systems and new ideas for how to develop them. We will need to look at them differently, not just as systems or systems of systems, but as socio-technical ecosystems. We will face fundamental challenges in the design and evolution, orchestration and control, and monitoring and assessment of ULS systems. These challenges require breakthrough research. (ULS, 2016, p. 2).

Na segunda afirmação, em seu glossário, o relatório diz que é problemático construir sistemas ULS utilizando técnicas e processos de desenvolvimento de software atuais:

> Ultra-large-scale (ULS): A system at least one of whose dimensions is of such a large scale that constructing the system using development processes and techniques prevailing at the start of the 21st century is problematic. ULS systems exhibit the following characteristics: decentralization; conflicting, unknowable, and diverse requirements; continuous evolution and deployment; heterogeneous and changing elements; erosion of the people/system boundary; and normal failures of parts of the system. (ULS, 2016, p. 16).

Desse modo, por meio dos relatórios do grupo Standish, da literatura sobre Engenharia de Software e dos relatórios dos sistemas ULS, os problemas relacionados à Engenharia de Software (e também da Engenharia de Requisitos) foram aqui apresentados e evidenciados. Com isso, viu-se que tais problemas são críticos e suas soluções requerem urgência.

5. ENGENHARIA HERMENÊUTICA DE REQUISITOS: APRESENTAÇÃO

Este capítulo apresenta o tema central de discussão desta tese: Engenharia Hermenêutica de Requisitos, que vem a ser a composição de dois instrumentos: Elicitação Hermenêutica de Requisitos e Teodolito Hermenêutico de Requisitos.

Inicialmente, é apresentada a Elicitação Hermenêutica de Requisitos, a qual é o instrumento que oferece ao Engenheiro de Requisitos mecanismos que o capacitam a melhor compreender e interpretar o Domínio da Aplicação e, assim, conseguir determinar e constituir com maior e melhor precisão os requisitos de software.

Na sequência, é apresentado o Teodolito Hermenêutico de Requisitos, que vem a ser o instrumento que revela os níveis de compreensão (e de dificuldade) em que o Engenheiro de Requisitos se encontra em relação ao Domínio da Aplicação e, também, dos graus de qualidade dos requisitos de software.

5.1. **Elicitação Hermenêutica de Requisitos**

A Elicitação Hermenêutica de Requisitos tem por finalidade ajudar o Engenheiro de Requisitos a adquirir, de forma suficiente, consciência e conhecimentos mais abrangentes em relação ao Domínio da Aplicação para o qual o software será desenvolvido e, assim, estabelecer de maneira mais sólida e consistente sua base primária e primordial (sua pedra angular) que lhe dará a sustentação conceitual para o seu desenvolvimento.

Ao aplicar a Elicitação Hermenêutica de Requisitos, o Engenheiro de Requisitos é levado a pensar de forma mais reflexiva em relação ao sentido do contexto para o qual o software será desenvolvido, o qual é o elemento essencial para a compreensão dos problemas e/ou das oportunidades de negócio que deverão ser resolvidos e/ou usufruídas com o seu apoio.

Para isso, alguns conceitos advindos da filosofia hermenêutica de Martin Heidegger, os quais se encontram descritos na seção 2.1, foram usados como fundamentação teórica para a criação e composição da Elicitação Hermenêutica de Requisitos. Para tal, foram realizadas neles algumas adequações, que se encontram

relatadas nesta seção, que os personalizaram para serem aplicados especificamente na Engenharia Hermenêutica de Requisitos.

O Conceito *Dasein* fundamenta teoricamente o conceito Elicitação Hermenêutica de Requisitos que, em sua essência, busca compreender e interpretar o sentido ontológico do Domínio da Aplicação para o qual o software será desenvolvido. Isso se dá a partir do conhecimento prévio que o Engenheiro de Requisitos já possui e dos novos conhecimentos que ele passará a adquirir acerca do contexto organizacional e/ou do negócio que o origina, juntamente com os problemas a serem solucionados pelo software a ser desenvolvido e/ou as oportunidades de negócio a serem usufruídas com o seu apoio.

A Elicitação Hermenêutica de Requisito revela ao Engenheiro de Requisitos o que é como é o Domínio da Aplicação. A partir deste ponto, é descoberto o contexto organizacional e/ou do negócio e o propósito a ser atingido pelo software será desenvolvido.

Assim, a Elicitação Hermenêutica de Requisitos anuncia ao Engenheiro de Requisitos a essência do Domínio da Aplicação, o orientando na execução desta tarefa que, em última análise, proporcionará que lhe seja revelado o seu horizonte originário que o contextualiza.

Desse modo, o Engenheiro de Requisitos será conduzido a conhecer o Domínio da Aplicação em suas diversas dimensões, podendo aprofundar seu conhecimento até o nível de compreensão que for necessário e suficiente para, a partir deste ponto, poder conceber os requisitos de software.

A Elicitação Hermenêutica de Requisitos possibilita ao Engenheiro de Requisitos a identificação de novas dimensões do Domínio da Aplicação, como também o aprofundamento de seu conhecimento em dimensões já identificadas. Isso quer dizer que o nível de detalhamento e a profundidade do conhecimento que o Engenheiro de Requisitos deve alcançar em relação ao Domínio da Aplicação dependem, exclusivamente, de suas necessidades, como também das necessidades do projeto; ou seja, não é uma decisão que cabe à Elicitação Hermenêutica de Requisitos determinar se o Engenheiro de Requisitos deve ou não continuar aprofundando seus conhecimentos acerca do Domínio da Aplicação

A tarefa da Elicitação Hermenêutica de Requisitos é orientar e guiar o Engenheiro de Requisitos a determinar os requisitos do software a ser desenvolvido, sem a

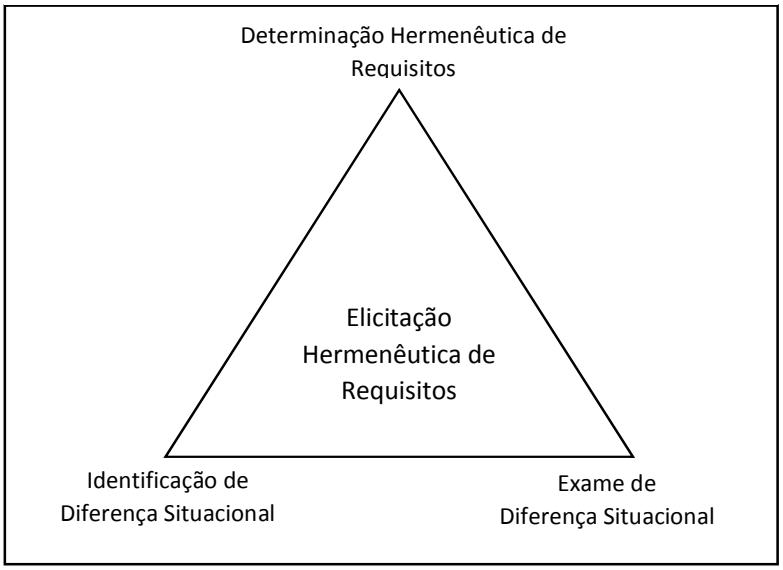
pretensão de proporcionar uma ontologia completa sobre eles, embora isto seja possível, conforme as necessidades exclusivas de cada projeto.

Desse modo, conclui-se que quanto melhor for a compreensão e a interpretação deste contexto organizacional e/ou deste negócio (juntamente com seus problemas e suas oportunidades), melhor será a definição do Domínio da Aplicação e, consequentemente, quanto melhor for a compreensão e interpretação do Domínio da Aplicação, melhor será a especificação dos requisitos de software e, por fim, quanto melhor for a compreensão e interpretação dos requisitos de software, melhor será o software a ser desenvolvido.

A tríade do *Dasein* fundamenta teoricamente a tríade da Elicitação Hermenêutica de Requisitos, que é composta pelos conceitos-chave "Identificação de Diferença Situacional", "Exame de Diferença Situacional" e "Determinação Hermenêutica de Requisitos".

A figura 7 ilustra a tríade da Elicitação Hermenêutica de Requisitos:

Figura 7 - Tríade da Elicitação Hermenêutica de Requisitos



Fonte: produção do próprio autor

Identificação de Diferença Situacional

O conceito "ser-no-mundo", apresentado na seção 2.1, fundamenta teoricamente esse conceito aqui apresentado (Identificação de Diferença Situacional), cuja finalidade é a compreensão e a interpretação do Domínio da Aplicação, que vem a ser a base conceitual primária e essencial utilizada para especificar os requisitos de software.

A Identificação de Diferença Situacional vem a ser o olhar voltado para uma Comunidade de Negócio e/ou uma Área de Negócio com o propósito de compreendê-las e, a partir daí, identificar e investigar os fatos e as informações que resultam em algum comportamento não adequado ou fora do padrão e/ou do planejamento.

Logo, a Diferença Situacional está sujeita ao que ocorre com a Comunidade de Negócio e/ou na Área de Negócio, e sua identificação se dá mediante a análise que se faz dos eventos que ocorrem entre elas. Por isso, não se pode analisar a Diferença Situacional separadamente de sua Comunidade de Negócio e/ou Área de Negócio, pois ambas se contextualizam e se complementam. Não é possível existir Diferença Situacional sem Comunidade de Negócio e/ou Área de Negócio, como também não é possível existir Comunidade de Negócio e/ou Área de Negócio sem que haja Diferença Situacional.

Portanto, a existência da Diferença Situacional consiste no modo peculiar de seu relacionamento com a Comunidade de Negócio e/ou Área de Negócio e sua contextualização também é marcada por esta relação. Isto significa que Comunidade de Negócio e/ou Área de Negócio são condições à existência da Diferença Situacional (e vice-versa). Portanto, a compreensão da estrutura e da dinâmica deste relacionamento consiste na compreensão do sentido ontológico da Diferença Situacional.

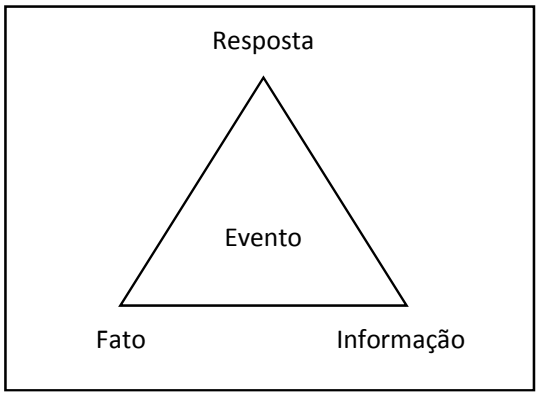
Não se pode olhar para Diferença Situacional e Comunidade de Negócio / Área de Negócio como duas coisas distintas que se relacionam, mas como partes constituintes de um todo, que se fundem para formarem o sentido de suas existências, numa unidade ontológica própria.

Sendo assim, a abertura que há nessa relação possibilita que sejam identificados os elementos que a compõem, suas disposições e os eventos que ocorrem entre si. Estes eventos são constituídos pelos fatos, pelas informações e pelas respostas dadas por suas interações, num sentido de causa e efeito, onde na causa se encontram os fatos e as informações e no efeito encontram-se as respostas e, também, as informações.

A tríade do *Ereignis*, apresentada na seção 2.1, fundamenta teoricamente esta tríade do Evento (formada por Fatos, Informações e Respostas) a qual possibilita ao Engenheiro de Requisitos a apropriação da compreensão contextualizada da Diferença Situacional.

A figura 8 ilustra a Tríade do Evento:

Figura 8 - Tríade do Evento



Fonte: produção do próprio autor

Através da aplicação contínua desta tríade do Evento, o Engenheiro de Requisitos passa a compreender melhor os conceitos até então já conhecidos, como também a identificar novos conceitos que se fazem necessários para uma melhor compreensão do Domínio da Aplicação.

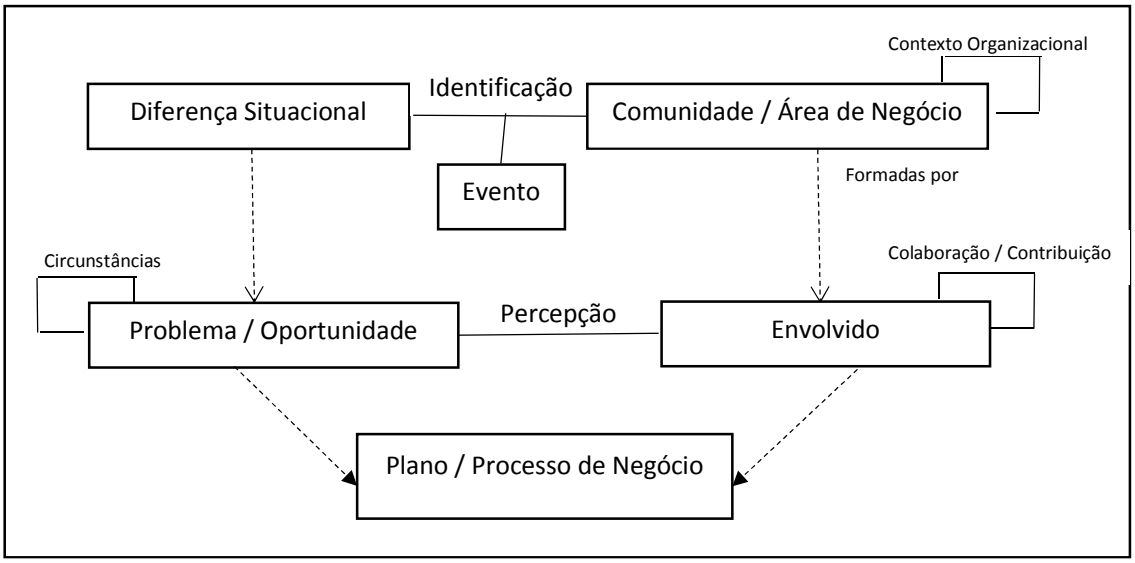
A Comunidade de Negócio e a Área de Negócio, que compõem o contexto organizacional para o qual o software será desenvolvido, são formadas pelos "Envolvidos", que são as pessoas físicas e/ou jurídicas que possuem alguma relação com a Diferença Situacional. Esses Envolvidos podem interagir entre si de maneira colaborativa e/ou contributiva para a realização de suas atividades e tarefas.

A Diferença Situacional, dependendo das circunstâncias, se manifesta como problemas e/ou oportunidades, os quais são percebidos pelos Envolvidos de acordo com os Eventos que ocorrem entre eles. Cada Envolvido possui uma relação particular com os problemas e/ou oportunidades que percebe e, devido a isto, possui expectativas e também experiências particulares de como lidar com eles, pois também é afetado e/ou beneficiado de maneira particular com suas ocorrências.

Os contextos das relações entre os Envolvidos e os problemas e/ou oportunidades geralmente são descritos em planos e/ou processos de negócios que mapeiam como eles ocorrem em detalhes, descrevendo as diretrizes de como proceder diante dos problemas e/ou oportunidades, como também as sugestões de como os resolver, mitigar e/ou reproduzir, conforme o conjunto de possibilidades (e seus respectivos benefícios) conhecido.

A figura 9 ilustra o conceito Identificação de Diferença Situacional.

Figura 9 - Conceito Identificação de Diferença Situacional

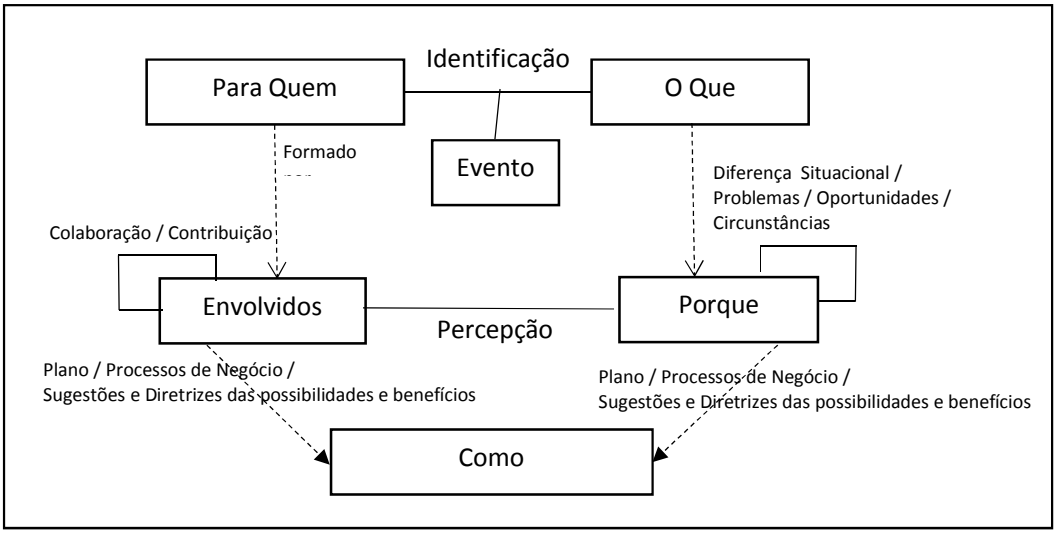


Fonte: produção do próprio autor

Durante a aplicação da Identificação de Diferença Situacional são identificados, compreendidos e interpretados o que está acontecendo, para quem está acontecendo, porque está acontecendo e como está acontecendo. Por se tratar do Domínio da Aplicação, todos esses elementos dizem respeito ao contexto do negócio para o qual o software será desenvolvido.

A figura 10 ilustra a aplicação da Identificação de Diferença Situacional.

Figura 10 - Aplicação da Identificação de Diferença Situacional



Fonte: produção do próprio autor

Não há uma obrigatoriedade ou exigência, mas essas informações podem ser organizadas em uma simples tabela, conforme apresentada na tabela 7:

Tabela 7 - Exemplo de como organizar o registro da Identificação de Diferença Situacional

Identificação de Diferença Situacional	
O que fazer?	
Para quem fazer?	
Por que fazer?	
Quem são os envolvidos?	
Como funciona o processo?	

Fonte: produção do próprio autor

Ou então, cada um desses componentes pode ser registrado separadamente em documentos individuais, ou da forma que julgar mais conveniente. Porém, essa decisão não cabe à Identificação de Diferença Situacional, pois é uma responsabilidade exclusiva de projeto e, portanto, deve ser tomada pelos seus gestores.

A Identificação de Diferença Situacional possibilita ao Engenheiro de Requisitos obter melhor conhecimento sobre o Domínio da Aplicação. Com isso, ele vai se tornando mais capacitado a especificar os requisitos de software, à medida que vai se apropriando mais ampla e profundamente de cada um de seus elementos aqui explanados.

Com isso, tal qual o “ser-no-mundo”, entende-se que a Identificação de Diferença Situacional é o conceito que diz respeito ao olhar revelador e descobridor do Domínio da Aplicação, em sua condição imanente, que questiona a respeito de seu sentido ontológico para falar de si e encontrar-se com sua essência.

Exame de Diferença Situacional

O conceito “ser-com-os-outros”, explanado na seção 2.1, fundamenta teoricamente esse conceito aqui apresentado (Exame de Diferença Situacional), cuja finalidade é a compreensão e interpretação do cenário, do ambiente e dos utensílios (ou insumos) que fazem parte, são usados e/ou produzidos e contextualizam o Domínio da Aplicação e o negócio para o qual o software será desenvolvido.

O Exame de Diferença Situacional vem a ser o olhar voltado para o ambiente em que a Comunidade de Negócio e/ou a Área de Negócio se encontram para realizarem suas atividades e tarefas e produzirem seus utensílios (ou insumos) e, desse modo, compreender com maior e melhor precisão seus processos de trabalho.

O caráter ontológico da Diferença Situacional é examinado por meio da compreensão combinada e relacionada do ambiente que o compõe e dos utensílios (ou insumos) que o forma.

Para compreender o ambiente ao qual a Comunidade de Negócio e/ou a Área de Negócio estão inseridos é necessário que seja compreendida a cultura que as envolve e rege suas atitudes. As regras, as políticas, as metas, as estratégias, os processos, as práticas, as informações, os padrões e os mecanismos de controle que moldam os comportamentos desse ambiente são estabelecidos por essa cultura.

Por sua vez, esse ambiente é influenciado por seu meio-ambiente, ou seja, por todos os fatores externos que possuem algum modo de relacionamento com ele, que o afetam diretamente e provocam mudanças e adaptações em sua cultura. Esses fatores externos são dos mais variados tipos, podendo ser, mas não se limitando a esses: clientes, concorrentes, fornecedores, entidades reguladoras, leis e políticas externas, economia, demografia, tecnologia e ambiente sociocultural.

Assim como o ambiente, o meio-ambiente também é influenciado por fatores externos. Nesse caso, pela natureza do negócio em si. Toda a Comunidade de Negócio e Área de Negócio tem a sua natureza própria, de onde se originou e foi determinada sua finalidade raiz. Todo o sistema que permeia a Comunidade de Negócio e a Área de Negócio é o resultado da natureza de seu negócio. Por exemplo, uma indústria automobilística e uma indústria de calçado, ambas são da mesma natureza: indústria, logo, seus meio-ambientes são impactados pelo sistema industrial. Uma escola de ensino infantil e uma universidade, ambas são da mesma natureza: educação, logo, seus meio-ambientes são impactados pelo sistema educacional. Portanto, a compreensão do cenário da Comunidade de Negócio e/ou da Área de Negócio depende da compreensão de sua natureza.

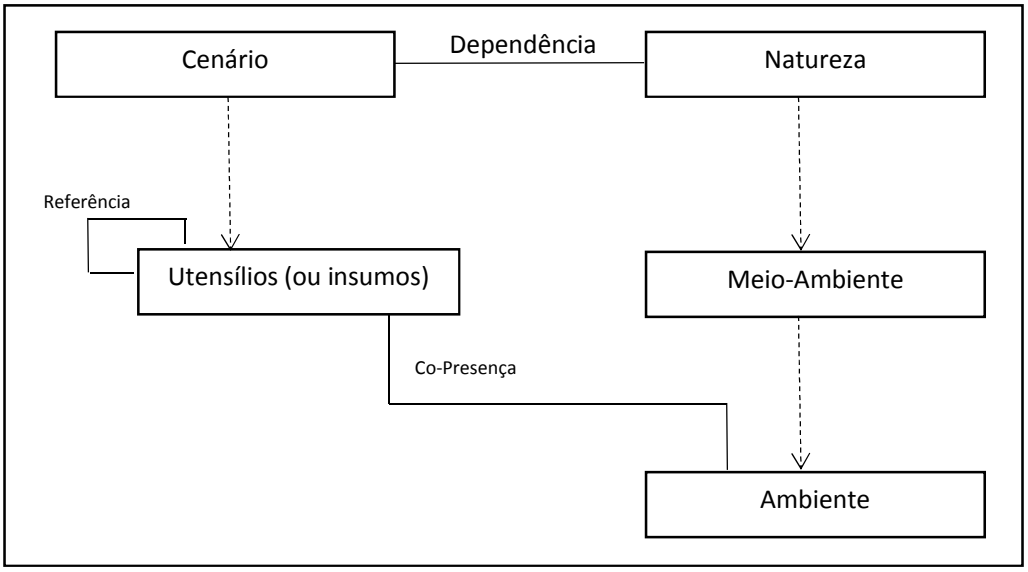
Os utensílios (ou insumos) que se encontram inseridos no ambiente e o formam são os elementos que determinam o cenário do Domínio da Aplicação e do Negócio para o qual o software será desenvolvido. Esses utensílios (ou insumos) referem-se uns aos outros, numa relação de colaboração mútua, para formarem o todo instrumental deste cenário.

Através da aplicação do Exame de Diferença Situacional, o Engenheiro de Requisitos passa a conhecer e a compreender de maneira mais ampla e geral o caráter ontológico do Domínio da Aplicação.

Logo, o Exame de Diferença Situacional é o conceito que orienta o Engenheiro de Requisitos a ampliar sua compreensão acerca do Domínio da Aplicação, ao encontrar os outros fatores que o contextualiza. Fatores, esses, que vão além daqueles apresentados na Identificação de Diferença Situacional.

A figura 11 ilustra o Exame de Diferença Situacional:

Figura 11 - Exame de Diferença Situacional



Fonte: produção do próprio autor

Não há uma obrigatoriedade ou exigência, mas essas informações podem ser organizadas em uma simples tabela, conforme apresentada na tabela 8:

Tabela 8 - Exemplo de como organizar o registro do Exame de Diferença Situacional

Exame de Diferença Situacional	
Cenário	
Natureza	
Meio-ambiente	
Ambiente	
Utensílios (ou insumos)	

Fonte: produção do próprio autor

Ou então, cada um desses componentes pode ser registrado separadamente em documentos individuais, ou da forma que julgar mais conveniente. Porém, essa decisão não cabe ao Exame de Diferença Situacional, pois é uma responsabilidade exclusiva de projeto e, portanto, deve ser tomada pelos seus gestores.

Com isso, ao aplicar o Exame de Diferença Situacional, o Engenheiro de Requisitos eleva o seu nível de maturidade em relação ao Domínio da Aplicação, expandindo ainda mais sua compreensão acerca de seu sentido ontológico e, portanto, capacita-se melhor para especificar os requisitos de software.

Assim como o “ser-com-os-outros”, entende-se que o Exame de Diferença Situacional é o conceito que diz respeito ao olhar transcendental que possibilita a ampliação da compreensão a cerca do Domínio da Aplicação, tornando seu sentido ontológico mais elevado e revelando de maneira estendida os aspectos de sua existência.

Determinação Hermenêutica de Requisitos

O conceito “ser-para-a-morte”, explanado na seção 2.1, fundamenta teoricamente essa definição aqui apresentada (Determinação Hermenêutica de Requisitos), cuja finalidade é, a partir da compreensão e interpretação do Domínio da Aplicação, declarar as necessidades originais dos Envolvidos (e suas respectivas expectativas), produzir a especificação aceitável e constituir os requisitos de software.

A Determinação Hermenêutica de Requisitos vem a ser o olhar voltado para o Domínio da Aplicação, compreendido e interpretado por meio da aplicação dos conceitos Identificação de Diferença Situacional e Exame de Diferença Situacional e fazer uso dessa compreensão e interpretação para constituir os requisitos de software.

A condição de ser e existir dos requisitos de software está intrinsicamente relacionada a este conceito, que serve a eles como parâmetro fundamental para determiná-los e constituí-los, conforme o Engenheiro de Requisitos vai compreendendo mais e melhor o Domínio da Aplicação, que os confere sentido à existência.

A Determinação Hermenêutica de Requisitos é o componente do ciclo da Elicitação Hermenêutica de Requisitos (vide figura 7 – Tríade da Elicitação Hermenêutica de Requisitos) onde ocorre a transição do Domínio da Aplicação à constituição dos requisitos de software, dando-lhes valor autêntico a eles, ou seja, determinando-os de maneira fidedigna aos propósitos pelos quais o software deverá ser desenvolvido.

A Determinação Hermenêutica de Requisitos funde os resultados obtidos durante as aplicações dos conceitos Identificação de Diferença Situacional e Exame de Diferença Situacional para formar e destacar a base e a essência de cada requisito de software, em todas as suas dimensões desveladas, para delimitar seu escopo essencial, de acordo com seu conceito e contexto até então compreendidos, admitidos, alcançados e compreendidos. Desse modo, a constituição ontológica de cada requisito de software é estabelecida para dar testemunho de sua existência.

Essa constituição ontológica do requisito de software é a elaboração da estrutura concreta de sua delimitação essencial. É a manifestação de sua essência própria, que possibilita a compreensão de seu sentido conforme a sua determinação.

Por meio de interações entre o Engenheiro de Requisitos e os envolvidos da Comunidade de Negócio e/ou da Área de Negócio, que se darão através de uma comunicação pautada pelo Contexto da Diferença Situacional, as Necessidades Originais (e as respectivas Expectativas) são declaradas, a Especificação Aceitável é produzida e os Requisitos de software constituídos.

O Contexto da Diferença Situacional é a aplicação conjunta e combinada da Identificação de Diferença Situacional e do Exame de Diferença Situacional, apresentados anteriormente, cujos seus respectivos resultados servem como parâmetro fundamental para conferir sentido à existência dos requisitos de software.

As Necessidades Originais (e as respectivas Expectativas) demonstram as preocupações e necessidades dos Envolvidos da Comunidade de Negócio e/ou da Área de Negócio, concebidas e analisadas durante as aplicações da Identificação de Diferença Situacional e do Exame de Diferença Situacional.

A Especificação Aceitável é extraída das Necessidades Originais (e as respectivas Expectativas) e contém o escopo geral do software a ser desenvolvido.

A Constituição dos Requisitos de Software são as descrições das tarefas que serão executadas pelo software, das informações que serão processadas por ele, das regras que ele obedecerá, das interações a serem feitas com seu ambiente externo, as quais estabelecem o que o software a ser desenvolvido deve fazer e como ele deve se comportar.

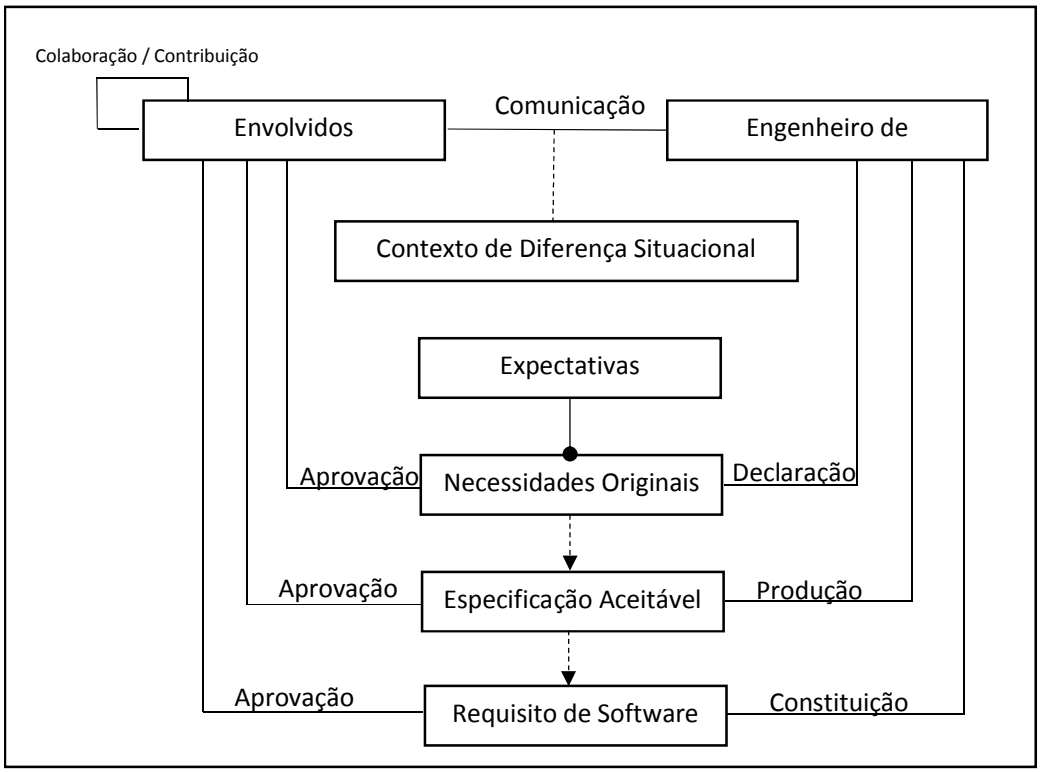
Enquanto, por um lado, o Engenheiro de Requisitos elabora esses artefatos (Necessidades Originais, Expectativas dos Envolvidos, Especificação Aceitável e Requisitos de Software), por outro lado, os Envolvidos da Comunidade de Negócio e/ou da Área de Negócio os validam e os aprovam, ou solicitam ajustes e melhorias.

A aplicação da Determinação Hermenêutica de Requisitos resulta em requisitos de software melhor concebidos e possibilita melhor qualidade de definição da visão do projeto do software a ser desenvolvido, a qual conterá informações suficientemente claras,

completas, e corretas, condizentes e compatíveis com o Domínio da Aplicação do qual foi originada e gerou sua condição de existência.

A figura 12 ilustra a Determinação Hermenêutica de Requisitos:

Figura 12 - Determinação Hermenêutica de Requisitos



Fonte: produção do próprio autor

Não há uma obrigatoriedade ou exigência, mas essas informações podem ser organizadas em duas simples tabelas, conforme apresentadas a seguir:

Tabela 9, para o caso das Necessidades Originais, das Expectativas dos Envolvidos e da Especificação Aceitável (tanto a descrição como a dinâmica).

Tabela 9 - Exemplo de como organizar a Determinação Hermenêutica de Requisitos

Determinação Hermenêutica de Requisitos	
Necessidades Originais	
Expectativas	
Especificação Aceitável (descrição)	
Especificação Aceitável (dinâmica)	

Fonte: produção do próprio autor

Tabela 10, para o caso da Constituição dos Requisitos de Software.

Tabela 10 - Exemplo de como organizar a Constituição dos Requisitos de Software

Determinação Hermenêutica de Requisitos Requisitos de Software (RS)	
RS-1: <nome do requisito de software 1>	
RS-2: <nome do requisito de software 2>	
RS-N <nome do requisito de software N>	

Fonte: produção do próprio autor

Ou então, cada um desses componentes pode ser registrado separadamente em documentos individuais, ou da forma que julgar mais conveniente. Porém, essa decisão não cabe à Determinação Hermenêutica de Requisitos, pois é uma responsabilidade exclusiva de projeto e, portanto, deve ser tomada pelos seus gestores.

Assim, a Determinação Hermenêutica de Requisitos capacitará o Engenheiro de Requisitos a conceber com melhor precisão os requisitos do software a ser desenvolvido, independentemente do processo de desenvolvimento de software que esteja sendo utilizado.

5.2. **Teodolito Hermenêutico de Requisitos**

O Engenheiro de Requisitos, durante o desenvolvimento de um software, atinge diferentes níveis de compreensão e de dificuldade em relação ao Domínio da Aplicação. O mesmo ocorre com os requisitos de software, os quais evoluem individualmente e possuem graus de qualidade diferentes, conforme o Engenheiro de Requisitos obtém melhor compreensão em relação a eles.

Diante desses fatos, surge uma questão: como identificar, mensurar, revelar e analisar estes diferentes níveis de compreensão e de dificuldade em relação ao Domínio da Aplicação e aos graus de qualidade dos requisitos de software?

Para responder a essa questão, propõe-se o uso do Teodolito Hermenêutico de Requisitos, o qual é formado por dois mecanismos: um que avalia e apresenta os níveis de compreensão e de dificuldade relacionados ao Domínio da Aplicação, e outro que avalia e apresenta os graus de qualidade dos requisitos de software.

Para avaliar e apresentar os níveis de compreensão e de dificuldade em que o Engenheiro de Requisitos se encontra em relação ao Domínio da Aplicação, o Teodolito

Hermenêutico de Requisitos usa como fundamentação teórica a Taxonomia SOLO, desenvolvida por John Biggs e Kevin Collis (explanada na seção 2.2), que possui cinco níveis de classificação de aprendizagem de uma pessoa em relação a um determinado tema que visa aprender, os quais foram adaptados especificamente para o Teodolito Hermenêutico de Requisitos para formar os seguintes cinco níveis de conhecimento do Engenheiro de Requisitos: Pré-Conceitual, Conceitual, Contextual, Sistêmico e Holístico.

A avaliação de cada um desses níveis de conhecimento passa pela verificação do conhecimento obtido pelo Engenheiro de Requisitos em relação ao Domínio da Aplicação. Desse modo, também é conhecida a capacidade para realizar a constituição dos requisitos de software.

No nível Pré-Conceitual, o Engenheiro de Requisitos ainda não possui um conceito bem estabelecido sobre o Domínio da Aplicação, apesar de já identificar a Diferença Situacional e a Comunidade de Negócio ou Área de Negócio, mas sem ainda identificar a relação que ocorre entre elas. Neste nível de conhecimento, sua capacidade para realizar a constituição dos requisitos de software é muito fraca.

Já no nível Conceitual, o Engenheiro de Requisitos identifica a Diferença Situacional, a Comunidade de Negócio ou Área de Negócio e também a relação que ocorre entre elas, mas ainda não possui detalhes sobre os eventos que acontecem nesta relação. Neste nível de conhecimento, sua capacidade para constituir os requisitos de software é fraca.

O nível Contextual indica que o Engenheiro de Requisitos identifica a Diferença Situacional, a Comunidade de Negócio ou Área de Negócio, a relação que ocorre entre elas e também já conhece os detalhes sobre os eventos que acontecem nesta relação. Neste nível de conhecimento, sua capacidade para realizar a constituição dos requisitos de software é razoável.

Estando no nível Sistêmico, o Engenheiro de Requisitos identifica os Envolvidos que formam a Comunidade de Negócio ou Área de Negócio e como eles interagem entre si, de forma colaborativa e/ou contributiva. Identifica, também, como cada Envolvido percebe a Diferença Situacional, dependendo das circunstâncias, na forma de Problema ou Oportunidade, e como ele lida, é afetado ou beneficiado por ela. Neste nível de conhecimento, sua capacidade para constituir os requisitos de software é forte.

Ao atingir o nível Holístico, o Engenheiro de Requisito identifica as possibilidades (e seus respectivos benefícios) que podem ser adotadas pelos Envolvidos da Comunidade de Negócio ou Área de Negócio para resolverem (ou mitigarem) os problemas e usufruírem das oportunidades. Também é identificado o Ambiente que contextualiza todo o cenário da Diferença Situacional. Neste nível de conhecimento, sua capacidade para realizar a constituição dos requisitos de software é muito forte.

A tabela 11 apresenta os cinco níveis de conhecimento do Engenheiro de Requisitos em relação ao Domínio da Aplicação e suas respectivas capacidades para constituir os requisitos de software.

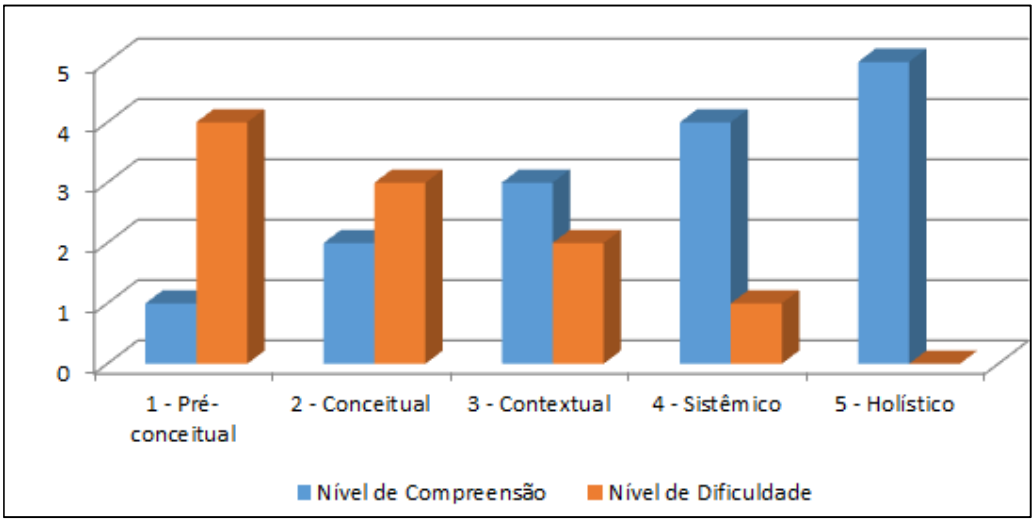
Tabela 11- Os níveis de conhecimento do Engenheiro de Requisitos

Teodolito Hermenêutico de Requisitos Os cinco níveis de conhecimento do Engenheiro de Requisitos	
Nível	Conhecimento em relação ao Domínio da Aplicação e suas Capacidades para constituir os requisitos de software
1 - Pré-conceitual	Identifica a Diferença Situacional e a Comunidade de Negócio / Área de Negócio, mas não identifica a relação que ocorre entre elas. Capacidade para constituir os requisitos de software: muito fraca.
2 – Conceitual	Identifica a Diferença Situacional, a Comunidade de Negócio / Área de Negócio, a relação que ocorre entre elas, mas não conhece os eventos que acontecem nesta relação. Capacidade para constituir os requisitos de software: fraca.
3 – Contextual	Identifica a Diferença Situacional, a Comunidade de Negócio / Área de Negócio, a relação que ocorre entre elas e conhece os eventos que acontecem nesta relação. Capacidade para constituir os requisitos de software: razoável.
4 – Sistêmico	Identifica os Envolvidos que formam a Comunidade de Negócio / Área de Negócio e como eles interagem entre si. Também identifica os Problemas / Oportunidades, juntamente com suas circunstâncias, e sabe como cada Envolvido percebe, lida, é afetado ou beneficiado por eles. Capacidade para constituir os requisitos de software: forte.
5 - Holístico	Identifica as possibilidades (e seus respectivos benefícios) que podem ser adotadas pelos Envolvidos da Comunidade de Negócio ou Área de Negócio para resolverem (ou mitigarem) os problemas e usufruírem das oportunidades. Também é identificado o Ambiente que contextualiza todo o cenário da Diferença Situacional. Desse modo, sabe o que está acontecendo, para quem está acontecendo, porque está acontecendo e como está acontecendo. Capacidade para constituir os requisitos de software: muito forte.

Fonte: produção do próprio autor

De acordo com esses níveis de conhecimento do Engenheiro de Requisitos em relação ao Domínio da Aplicação e suas respectivas capacidades para constituir os requisitos de software, o Teodolito Hermenêutico de Requisitos exibirá os seguintes resultados conforme a escala apresentada na figura 13:

Figura 13 - Níveis de compreensão em relação ao Domínio da Aplicação



Fonte: produção do próprio autor

Desse modo, é possível visualizar e acompanhar ao longo da aplicação da Elicitação Hermenêutica de Requisitos os níveis de conhecimento (e também de dificuldade) em que o Engenheiro de Requisitos se encontra em relação ao Domínio da Aplicação.

Assim, o Teodolito Hermenêutico de Requisitos ajuda a revelar o quão maduro está o nível do conhecimento (e de dificuldade) do Engenheiro de Requisitos em relação ao Domínio da Aplicação e, de acordo com estas observações, determinar estratégias para melhorar a aplicação da Elicitação Hermenêutica de Requisitos.

Já para avaliar e apresentar os graus de qualidade dos requisitos de software, o Teodolito Hermenêutico de Requisitos usa como base teórica o *Essence* (explanado na seção 2.3), formado por um conjunto de estados e sub-estados que apresenta o quanto cada requisito de software evoluiu e o quanto cada um ainda tem por evoluir.

Esta adequação realizada no *Essence* para o Teodolito Hermenêutico de Requisitos resultou na criação de cinco estados e para cada estado quatro sub-estados, onde

o requisito de software só estará em um determinado estado quando seus quatro sub-estados forem alcançados.

Inicialmente, os requisitos de software devem atingir o estado Identificado. Para isto, cada um deles deve ser identificado individualmente, classificado de maneira clara, ter suas regras de negócio e restrições conhecidas e possuir uma descrição resumida concisa.

O requisito de software atingirá o estado Concebido quando ele comunicar suas características essências, for priorizado, não possuir conflito com outro requisito e poder ser rastreado.

Para chegar ao estado Descrito, o requisito de software deve estar claro em relação ao seu escopo, consistente em relação às expectativas dos Envolvidos, aceito pelos Envolvidos por ter capturado com precisão o que ele faz e o que ele não faz e ter sido alocado.

O próximo estado a ser atingido é o Declarado, onde o requisito deve estar em conformidade com os padrões exigidos, fornecer um claro valor agregado aos Envolvidos, estar constituído de maneira suficiente e possibilitar ser testado e avaliado.

Ao progredir ao estado Aprovado, o requisito de software está satisfatoriamente completo e consistente, não possui omissões e nem ambiguidades, não possui itens pendentes que impeçam seu aceite pelos Envolvidos e foi aceito pelos Envolvidos como satisfazendo plenamente suas necessidades.

A tabela 12 apresenta os cinco estados, com seus respectivos sub-estados, que serão avaliados pelo Teodolito Hermenêutico de Requisitos para determinar os graus de qualidade dos requisitos de software.

Tabela 12 - Os estados dos requisitos de software e seus respectivos sub-estados

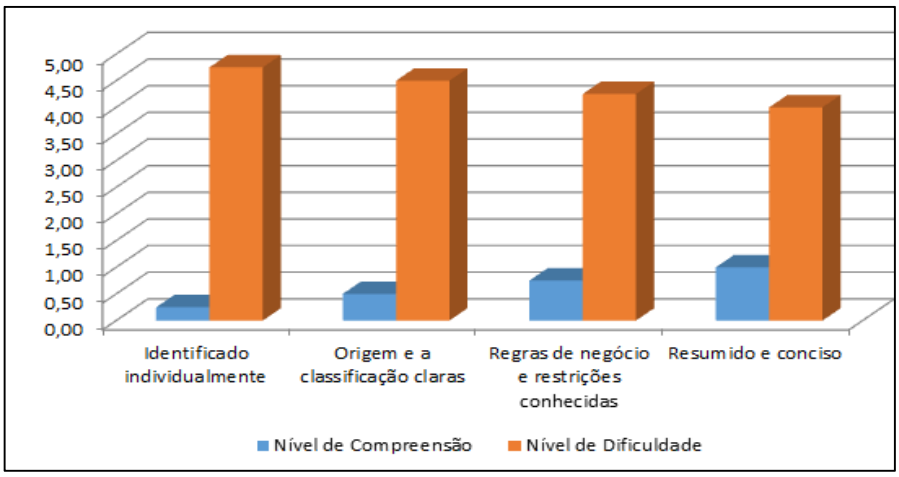
Teodolito Hermenêutico de Requisitos Os estados dos requisitos de software e seus respectivos sub-estados	
Estado	Sub-estados
1. Identificado	1. O requisito foi identificado individualmente para as Envolvidos. 2. A origem e a classificação do requisito são claras. 3. As regras de negócio e as restrições impostas ao requisito são conhecidas. 4. O requisito foi descrito de forma resumida e concisa.
2. Concebido	1. O requisito comunica suas características essenciais. 2. O requisito foi priorizado. 3. O requisito não possui conflito com outro requisito. 4. É possível rastrear o requisito.
3. Descrito	1. O requisito está claro em relação ao seu escopo. 2. O requisito está consistente em relação às expectativas dos Envolvidos. 3. As partes interessadas aceitam que o requisito captura com precisão o que ele faz e não faz. 4. A alocação do requisito foi feita.
4. Declarado	1. O requisito está em conformidade com os padrões exigidos. 2. O conjunto de itens do requisito fornece um claro valor para as partes interessadas. 3. O requisito foi constituído de maneira suficiente. 4. É possível testar e avaliar o requisito.
5. Aprovado	1. O requisito está satisfatoriamente completo e consistente. 2. O requisito não possui omissões e nem ambiguidades. 3. Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas. 4. O requisito foi aceito pelas Envolvidos como satisfazendo plenamente suas necessidades.

Fonte: produção do próprio autor

Levando em consideração esses estados e sub-estados dos requisitos de software, o Teodolito Hermenêutico de Requisitos exibirá os seguintes resultados conforme a escala apresentada nas figuras a seguir, organizadas por estado de requisito de software:

A figura 14 ilustra as escalas referentes ao Estado “Identificado”, onde os resultados indicam que os níveis de dificuldade estão muito acima dos de compreensão.

Figura 14 - Estado Identificado - Resultados do Teodolito Hermenêutico de Requisitos

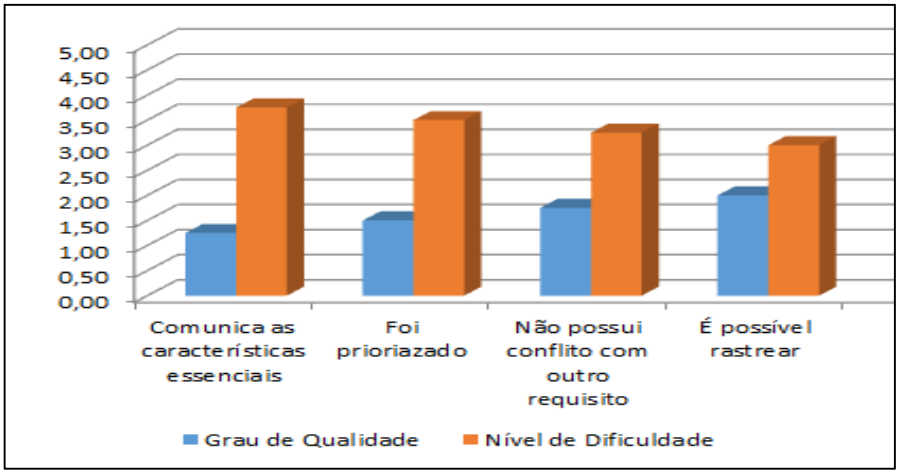


Fonte: produção do próprio autor

Esse estado revela que os níveis de dificuldade do Engenheiro de Requisitos são muito superiores aos de compreensão. Logo, o grau de qualidade do requisito de software é muito baixo.

A figura 15 ilustra as escalas referentes ao Estado "Concebido", onde os resultados indicam que os níveis de compreensão evoluíram em relação ao estado anterior, mas ainda os níveis de dificuldade estão acima dos de compreensão.

Figura 15 - Estado Concebido - Resultados do Teodolito Hermenêutico de Requisitos

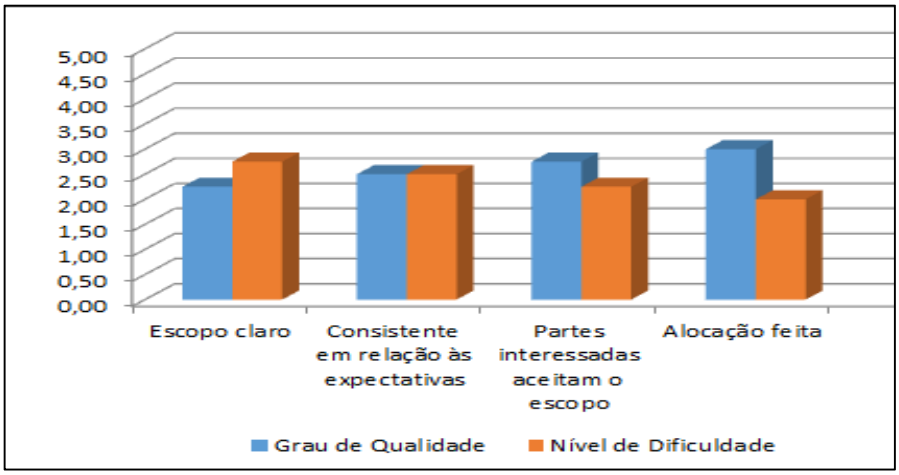


Fonte: produção do próprio autor

Esse estado revela que os níveis de dificuldade do Engenheiro de Requisitos são superiores aos de compreensão. Logo, o grau de qualidade do requisito de software é baixo.

A figura 16 ilustra as escalas referentes ao Estado “Descrito”, onde os resultados indicam que os níveis de compreensão evoluíram em relação ao estado anterior e começam a superar os níveis de dificuldade.

Figura 16 - Estado Descrito - Resultados do Teodolito Hermenêutico de Requisitos

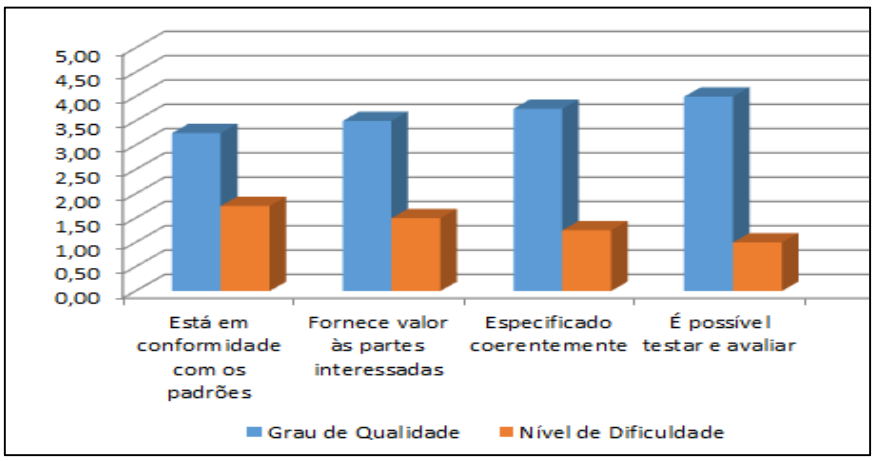


Fonte: produção do próprio autor

Esse estado revela que os níveis de dificuldade do Engenheiro de Requisitos são praticamente os mesmos aos de compreensão. Logo, o grau de qualidade do requisito de software é relativamente baixo.

A figura 17 ilustra as escalas referentes ao Estado “Declarado”, onde os resultados indicam que os níveis de compreensão evoluíram bem em relação ao estado anterior e também aos níveis de dificuldade.

Figura 17 - Estado Declarado - Resultados do Teodolito Hermenêutico de Requisitos

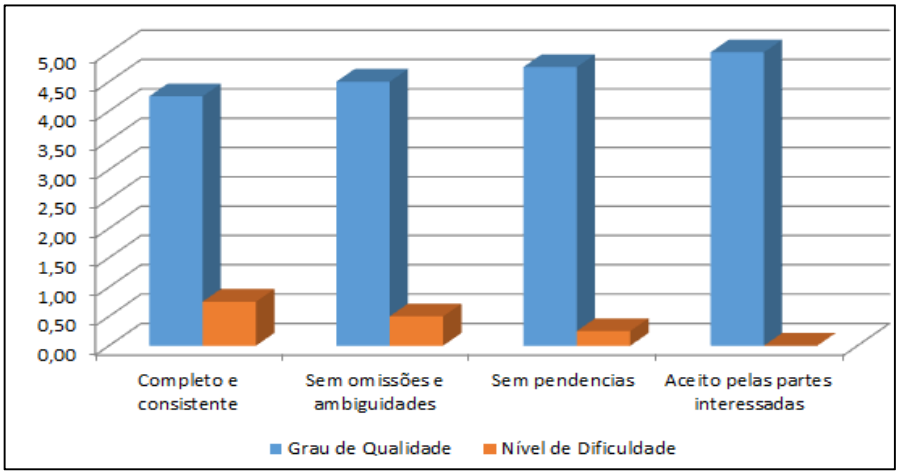


Fonte: produção do próprio autor

Esse estado revela que os níveis de compreensão do Engenheiro de Requisitos são superiores aos de dificuldade. Logo, o grau de qualidade do requisito de software é relativamente alto.

A figura 18 ilustra as escalas referentes ao Estado "Aprovado", onde os resultados indicam que os níveis de compreensão evoluíram bastante em relação ao estado anterior e estão bem acima dos níveis de dificuldade.

Figura 18 - Estado Aprovado - Resultados do Teodolito Hermenêutico de Requisitos



Fonte: produção do próprio autor

Esse estado revela que os níveis de compreensão do Engenheiro de Requisitos são bem superiores aos de dificuldade. Logo, o grau de qualidade do requisito de software é alto.

Por meio da aplicação do Teodolito Hermenêutico de Requisitos é possível acompanhar o progresso dos requisitos de software, individualmente, através das várias mudanças de estados e sub-estados, desde suas concepções até suas determinações e constituições.

O Teodolito Hermenêutico de Requisitos é um instrumento que ajuda a equipe de desenvolvimento de software (e a todos os demais Envolvidos) a conhecer o estado atual de cada requisito de software, e suas progressões subsequentes e necessárias, revelando o que é preciso fazer para que essa evolução ocorra. Isso é importante para melhorar os graus de qualidade dos requisitos de software e decidir o quão eles estão suficientes completos e sem ambiguidades.

Desse modo, é possível visualizar e acompanhar ao longo da aplicação da Engenharia de Requisitos os graus de qualidade dos requisitos de software. Assim, o Teodolito Hermenêutico de Requisitos ajuda a revelar o quão maduro está cada requisito de software e, desse modo, também determinar o nível de maturidade do escopo do projeto.

Com isto, foi visto que o Teodolito Hermenêutico de Requisitos é um instrumento capaz de avaliar e revelar os níveis de compreensão (e de dificuldade) em que o Engenheiro de Requisitos se encontra em relação ao Domínio da Aplicação, como também avaliar e revelar os graus de qualidade dos requisitos de software. Com esses resultados em mãos, é possível estabelecer estratégias para melhorar a aplicação da Engenharia Hermenêutica de Requisitos.

Assim, a Engenharia Hermenêutica de Requisitos, por meio de seus da aplicação combinada e coordenada de seus instrumentos Elicitação Hermenêutica de Requisitos e Teodolito Hermenêutico de Requisitos, vai ajudar o Engenheiro de Requisitos a obter uma compreensão compartilhada sobre o que o software a ser desenvolvido deve efetivamente fazer.

Desse modo, o Engenheiro de Requisitos terá sua capacidade de compreensão ampliada e melhorada, pois a Engenharia Hermenêutica de Requisitos foi concebida e projetada com base em sólidos e consistentes conceitos hermenêuticos, que tratam as questões relacionadas à compreensão e interpretação daquilo que se pretende conhecer; acadêmicos, que classificam e avaliam os níveis de aprendizagem dos aprendizes em relação àquilo que se pretende conhecer; da Engenharia de Requisitos, que determina as tarefas que devem ser executadas para que os requisitos de software sejam bem concebidos e especificados; e da Elicitação de Requisitos, que vem a ser a primeira tarefa a ser executada durante a aplicação da Engenharia de Requisitos, que tem por objetivo ajudar o Engenheiro de Requisitos a compreender os problemas que o software deverá resolver e/ou as oportunidades de negócio que devem ser usufruídas com o seu apoio.

Com isto, é realizada a determinação e consequente constituição dos requisitos de software, contemplando e formalizando a necessária e suficiente abrangência de seus propósitos, de seus benefícios, de suas funções, de suas restrições, de suas regras e de suas informações usadas e geradas.

Por fim, a Engenharia Hermenêutica de Requisitos auxilia o Engenheiro de Requisitos a realizar um estudo abrangente do Domínio da Aplicação para o qual o software será desenvolvido, possibilitando que seus requisitos sejam determinados e constituídos de forma consistente e compatível aos problemas que deverão ser solucionados com o seu auxilio e às oportunidades de negócios que deverão ser usufruídas com o seu apoio.

6. ENGENHARIA HERMENÊUTICA DE REQUISITOS: APLICAÇÃO

Este capítulo apresenta quatro estudos de casos, os quais mostram as aplicações da Engenharia Hermenêutica de Requisitos (Elicitação Hermenêutica de Requisitos e Teodolito Hermenêutico de Requisitos), que evidenciam suas soluções e benefícios.

6.1. **Jogo da Velha**

Nesta seção, é apresentada a aplicação da Engenharia Hermenêutica de Requisitos em uma situação que não envolve o contexto organizacional, mas sim o de entretenimento (game), mais especificamente o do Jogo da Velha.

Jogo da Velha: Aplicação da Elicitação Hermenêutica de Requisitos

A tabela 13 o resultado da aplicação da Identificação de Diferença Situacional:

Tabela 13 - Jogo da Velha: Identificação de Diferença Situacional

Identificação de Diferença Situacional	
O que fazer?	Software de Jogo da Velha.
Para quem fazer?	Jogadores de Jogo da Velha.
Por que fazer?	Promover o entretenimento entre os envolvidos.
Quem são os envolvidos?	A cada partida, dois Jogadores de Jogo da Velha.
Como se joga o Jogo da Velha?	O jogo consiste em uma disputa entre dois jogadores que se revezam alternadamente marcando as casas livres de um tabuleiro tipo grade 3x3 (#). Enquanto um jogador marca as casas livres com “X”, o outro jogador as marca com “O”. Cada jogador só pode marcar uma única casa livre por vez. O jogador que conseguir fazer primeiro três de suas marcas em uma linha horizontal, vertical ou diagonal vence a partida. Se nenhum jogador conseguir essa proeza, a partida é considerada empatada (diz-se “deu velha”).

Fonte: produção do próprio autor

A tabela 14 apresenta o resultado da aplicação do Exame de Diferença Situacional:

Tabela 14 - Jogo da Velha: Exame de Diferença Situacional

Exame de Diferença Situacional	
Cenário	Jogo eletrônico do tipo Jogo da Velha.
Natureza	O Jogo da Velha é um jogo do gênero tradicional.
Meio-ambiente	Jogo de tabuleiro tipo grade 3x3 (#).
Ambiente	Virtual.
Utensílios (ou insumos)	• Um Jogador de Jogo da Velha; • Um Jogador Virtual; • Um Tabuleiro do tipo grade 3x3 (#); • Cinco peças "X"; • Cinco peças "O"; • Um Placar para indicar o número de vitórias de cada jogador e de empates.

Fonte: produção do próprio autor

A tabela 15 apresenta o resultado da aplicação da Determinação Hermenêutica de Requisitos:

Tabela 15 - Jogo da Velha: Determinação Hermenêutica de Requisitos

Determinação Hermenêutica de Requisitos	
Necessidades Originais	Desenvolver um jogo virtual do tipo Jogo da Velha, do gênero tradicional, com o intuito de promover entretenimento aos jogadores de Jogo da Velha.
Expectativas	Conseguir jogar partidas de jogo da velha em ambientes virtuais.
Especificação Aceitável (descrição)	O Jogador de Jogo da Velha disputará partidas de Jogo da Velha com o Jogador Virtual. O Jogador do Jogo da Velha poderá escolher ser o "Jogador Um" (o primeiro jogador a fazer a jogada) ou o "Jogador Dois" (o segundo jogador a fazer a jogada). O "Jogador Um" utiliza as peças "X" e o "Jogador Dois" utiliza as peças "O". O placar marcará o número de vezes vencidas pelo "Jogador Um", pelo "Jogador Dois" e também o número de empates ocorridos. Esses números começarão com zero, assim que o Jogo da Velha for carregado.
Especificação Aceitável (dinâmica da partida Jogo da Velha)	1. Ao iniciar a partida, o tabuleiro estará com todas as casas livres.

	2. O “Jogador Um” realiza a jogada marcando com uma peça “X” uma casa livre do tabuleiro. 3. Após o “Jogador Um” realizar sua jogada, o “Jogador Dois” realiza a jogada marcando com uma peça “O” uma casa livre do tabuleiro. 4. Após cada jogador realizar sua jogada: 1. Se houver três peças “X” ou “O” sequenciais na vertical, horizontal ou diagonal, o respectivo jogador da peça venceu a partida, um ponto é adicionado ao seu placar e a partida é finalizada. 2. Se nenhum jogador venceu a partida e não houver mais casa livre no tabuleiro, a mesma é finalizada e um ponto é adicionado ao placar do empate. 3. Se nenhum jogador venceu a partida e houver mais casa livre no tabuleiro, a vez de realizar a jogada é do jogador que não fez a última jogada.

Fonte: produção do próprio autor

A tabela 16 apresenta o resultado da aplicação da Determinação Hermenêutica de Requisitos, para as descrições gerais dos requisitos de software:

Tabela 16 - Jogo da Velha: Determinação Hermenêutica de Requisitos

Determinação Hermenêutica de Requisitos Requisitos de Software (RS)	
RS-1: Escolher Jogador	O Jogador do Jogo da Velha escolhe jogar como “Jogador Um” (indicado com o “X”) ou como “Jogador Dois” (indicado com o “O”). Essa escolha não pode ser feita com a partida em andamento. Os resultados do placar, caso haja mudança no uso das peças, são influenciados por essa escolha, ou seja, o resultado do placar do “Jogador Um” é substituído pelo resultado do placar “Jogador Dois” e vice-versa. O resultado do placar de empates continua o mesmo.
RS-2: Zerar Placar	O Jogador do Jogo da Velha escolhe zerar o placar. Essa ação faz com que os placares do “Jogador Um”, do “Jogador Dois” e do Empate sejam zerados. Essa ação pode ser realizada a qualquer momento da partida e o Jogador do Jogo da Velha deve confirmar essa ação antes dos placares serem zerados.

RS-3: Iniciar Partida	O Jogador do Jogo da Velha escolhe iniciar uma nova partida. Essa ação pode ser realizada a qualquer momento e o Jogador do Jogo da Velha deve confirmar essa ação. Ao iniciar a nova partida, todas as casas do tabuleiro serão liberadas e os placares permanecerão os mesmos.
RS-4: Marcar “X”	Após o início de uma nova partida ou após o “Jogador Dois” ter feito sua jogada, o “Jogador Um” analisa uma casa livre do tabuleiro para realizar sua jogada. Ao escolher a casa livre, a mesma é preenchida com uma peça “X”. Dentre suas opções, ele deve escolher uma casa livre que combine três peças “X” na vertical, na horizontal ou na diagonal. Caso ocorra uma dessas combinações, o placar do “Jogador Um” é acrescido de um ponto e a partida é finalizada. Caso não seja possível uma dessas combinações, ele deve escolher uma casa livre que impeça a vitória do “Jogador Dois” (combinação de três peças “O” na vertical, na horizontal ou na diagonal). Caso só tenha uma casa livre disponível, após essa ser escolhida, o placar do Empate é acrescido de um ponto e a partida é finalizada.
RS-5: Marcar “O”	Após o “Jogador Um” ter feito sua jogada, o “Jogador Dois” analisa uma casa livre do tabuleiro para realizar sua jogada. Ao escolher a casa livre, a mesma é preenchida com uma peça “O”. Dentre suas opções, ele deve escolher uma casa livre que combine três peças “O” na vertical, na horizontal ou na diagonal. Caso ocorra uma dessas combinações, o placar do “Jogador Dois” é acrescido de um ponto e a partida é finalizada. Caso não seja possível uma dessas combinações, ele deve escolher uma casa livre que impeça a vitória do “Jogador Um” (combinação de três peças “X” na vertical, na horizontal ou na diagonal). Caso só tenha uma casa livre disponível, após essa ser escolhida, o placar do Empate é acrescido de um ponto e a partida é finalizada.
RS-6: Salvar Placar	O Jogador do Jogo da Velha escolhe salvar o placar e, após confirmar essa ação, os resultados de vitórias dos jogadores Um e Dois e também o número de empates são gravados para posterior recuperação. Essa ação pode ser acionada a qualquer momento.

RS-7: Recuperar Placar	O Jogador do Jogo da Velha escolhe recuperar o placar e, após confirmar essa ação, os últimos resultados de vitórias dos jogadores Um e Dois e também do número de empates gravados são recuperados, substituindo os resultados apresentados até o momento no placar. Essa ação pode ser acionada a qualquer momento.
RS-8: Explicar o Jogo	O Jogador do Jogo da Velha escolhe explicação sobre o jogo. Essa ação pode ser acionada a qualquer momento, fazendo com que a seguinte mensagem seja apresentada: "O Jogo da Velha é um jogo tradicional, simples de ser jogado. Curiosamente, ele foi o primeiro jogo a ser desenvolvido para computador, por Alexander Sandy Douglas., na Universidade de Cambridge, em 1952, para provar suas teorias de interação homem-computador que estava escrevendo em sua tese de doutorado. O jogo consiste em uma disputa entre dois jogadores que se revezam alternadamente marcando as casas livres de um tabuleiro tipo grade 3x3 (#). Enquanto um jogador marca as casas livres com "X", o outro jogador as marca com "O". Cada jogador só pode marcar uma única casa livre por vez. O jogador que conseguir fazer primeiro três de suas marcas em uma linha horizontal, vertical ou diagonal vence a partida. Se nenhum jogador conseguir essa proeza, a partida é considerada empatada (diz-se "deu velha"). Especificamente nesse jogo virtual, o jogador do jogo da velha deve escolher jogar como "X" ou como "O". O jogador "X" sempre realiza a primeira jogada. O oponente sempre será o jogador virtual. O jogo é acompanhado por um placar eletrônico que indica o número de vezes vencidas por cada jogador, como também o número de empates. O Jogador do Jogo da Velha pode escolher: jogar como "Jogador Um" (indicado com "X") ou "Jogador Dois" (indicado com "O"), zerar o placar, iniciar nova partida e efetuar jogada (colocar uma peça "X" ou "O" numa casa livre do tabuleiro). Para escolher jogador, escolha a ação "Escolher Jogador" e indique com qual jogador pretende jogar ("Jogador Um", indicado com "X" ou "Jogador Dois", indicado com "O"). Para zerar o placar, escolha a ação "Zerar Placar" e confirme-a se realmente desejar que o placar seja zerado. Para iniciar uma nova partida, escolha a ação "Iniciar Partida". Para realizar uma jogada, escolha uma casa livre do tabuleiro e a selecione. Ao realizar essa ação, a respectiva casa do tabuleiro será marcada com "X" ou com "O" de

	acordo com o jogador que estiver jogando (“Jogador Um” ou “Jogador Dois”, respectivamente). Para salvar o placar, escolha “Salvar Placar” e confirme essa ação para que os atuais resultados apresentados no placar sejam gravados para posterior recuperação. Para recuperar o placar, escolha “Recuperar Placar” e confirme essa ação para que os atuais resultados do placar sejam substituídos pelos resultados gravados pela última vez.

Fonte: produção do próprio autor

Jogo da Velha: Aplicação do Teodolito Hermenêutico de Requisitos

Os requisitos de software para o Jogo da Velha Virtual foram definidos e compostos em quatro ciclos, conforme detalhados abaixo:

Ciclo 1: Durante esse ciclo foram Identificados e descritos: o que deve ser feito, para quem, porque e como. Também foram identificados os seguintes requisitos de software: “Escolher Jogador”, “Iniciar Partida”, “Marcar X” e “Marcar O”, mas os mesmos não foram definidos e compostos.

Ao aplicar o Teodolito Hermenêutico de Requisitos durante o ciclo 1, foram obtidos os seguintes resultados:

Tabela 17 - Jogo da Velha Virtual: ciclo 1 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
Jogo da Velha Tradicional	4 – Sistêmico
Jogo da Velha Virtual	2 – Conceitual

Fonte: produção do próprio autor

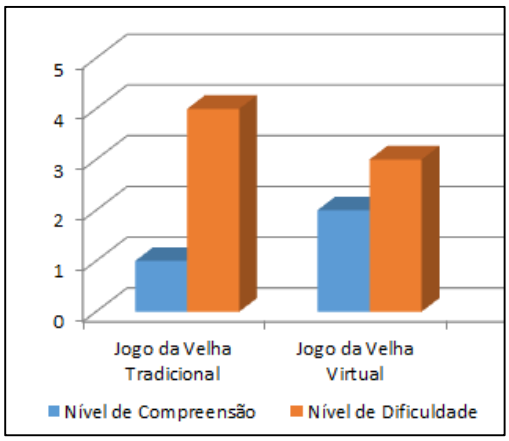
Tabela 18 - Jogo da Velha Virtual: ciclo 1 - Requisitos de Software

Requisito de Software	Grau de Qualidade
Escolher Jogador	2.3 – O requisito não possui conflito com outro requisito.
Iniciar Partida	2.3 – O requisito não possui conflito com outro requisito.
Marcar “X”	2.3 – O requisito não possui conflito com outro requisito.
Marcar “O”	2.3 – O requisito não possui conflito com outro requisito.

Fonte: produção do próprio autor

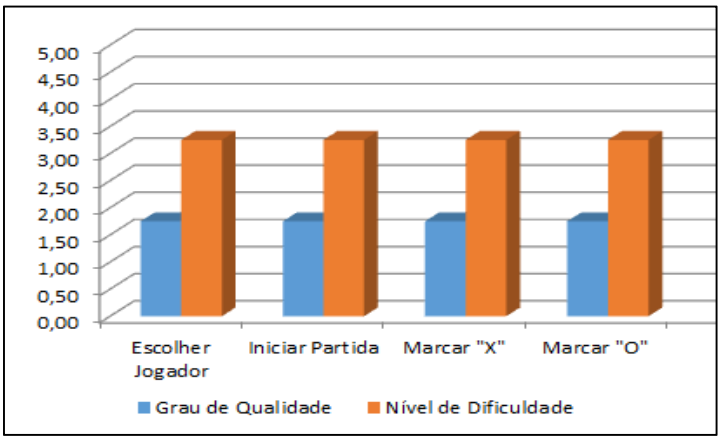
Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 19 - Jogo da Velha Virtual: ciclo 1 - leituras do domínio da aplicação



Fonte: produção do próprio autor

Figura 20 - Jogo da Velha Virtual: ciclo 1 - leituras dos requisitos de software



Fonte: produção do próprio autor

Ciclo 2: Durante esse ciclo, compreendeu-se de maneira contextualizada o cenário, o ambiente e os utensílios (ou insumos) do Jogo da Velha Tradicional, como também do Jogo da Velha Virtual. Foram definidos e compostos os requisitos de software “Escolher Jogador”, “Iniciar Partida”, “Marcar X” e “Marcar O” e identificados, mas não definidos e compostos, os requisitos de software: “Zerar Placar”, “Salvar Placar”, “Recuperar Placar” e “Explicar Jogo”.

Ao aplicar o Teodolito Hermenêutico de Requisitos durante o ciclo 2, foram obtidos os seguintes resultados:

Tabela 19 - Jogo da Velha Virtual: ciclo 2 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
Jogo da Velha Tradicional	5 – Holístico
Jogo de Velha Virtual	5 – Holístico

Fonte: produção do próprio autor

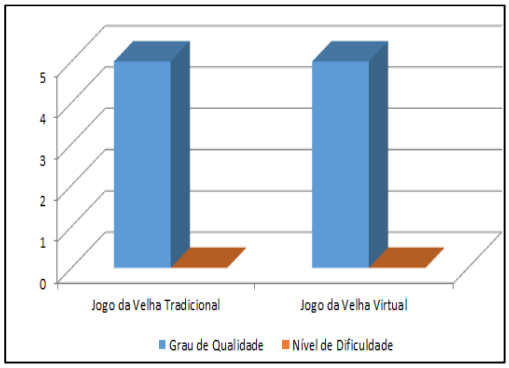
Tabela 20 - Jogo da Velha Virtual: ciclo 2 - Requisitos de Software

Requisito de Software	Grau de Qualidade
Escolher Jogador	5.3 - Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Iniciar Partida	5.3 - Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Marcar "X"	5.3 - Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Marcar "O"	5.3 - Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Zerar Placar	2.3 – O requisito não possui conflito com outro requisito.
Salvar Placar	2.3 – O requisito não possui conflito com outro requisito.
Recuperar Placar	2.3 – O requisito não possui conflito com outro requisito.
Explicar Jogo	2.3 – O requisito não possui conflito com outro requisito.

Fonte: produção do próprio autor

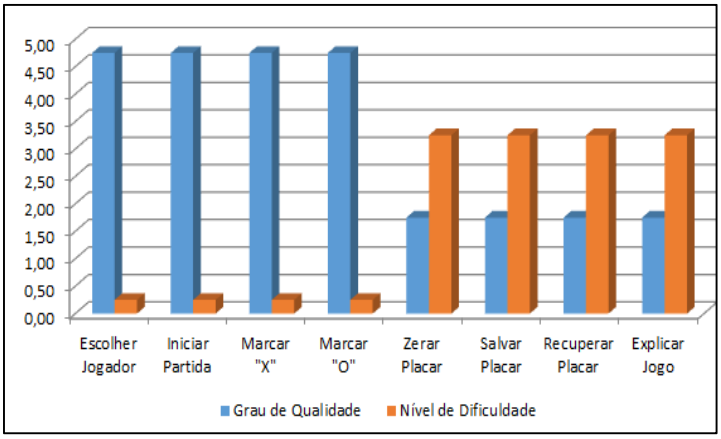
Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 21 - Jogo da Velha Virtual: ciclo 2 - leituras do domínio da aplicação



Fonte: produção do próprio autor

Figura 22 - Jogo da Velha Virtual: ciclo 2 - leituras dos requisitos de software



Fonte: produção do próprio autor

Ciclo 3: Durante esse ciclo, foram apresentadas as definições e composições dos requisitos de software “Escolher Jogador”, “Iniciar Partida”, “Marcar X” e “Marcar O” aos jogadores de Jogo da Velha, onde os mesmos as avaliaram e as aprovaram e também foram definidos e compostos os requisitos de software “Zerar Placar”, “Salvar Placar”, “Recuperar Placar” e “Explicar Jogo”.

Ao aplicar o Teodolito Hermenêutico de Requisitos durante o ciclo 3, foram obtidos os seguintes resultados:

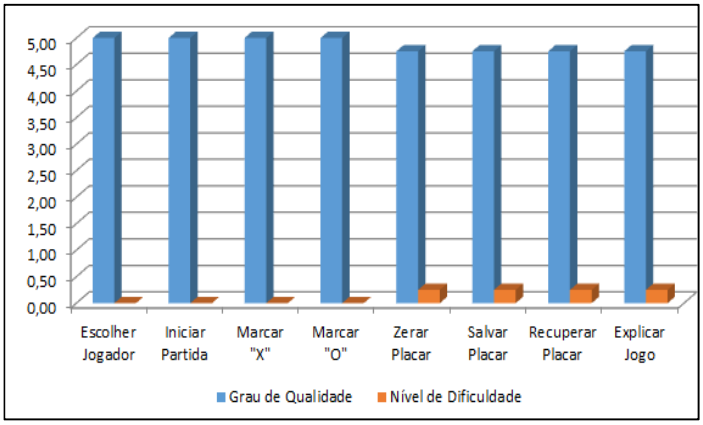
Tabela 21 - Jogo da Velha Virtual: ciclo 3 - Requisitos de Software

Requisito de Software	Grau de Qualidade
Escolher Jogador	5.4 – Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Iniciar Partida	5.4 – Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Marcar X	5.4 – Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Marcar O	5.4 – Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Zerar Placar	5.3 – Satisfatoriamente completo e composto, mas ainda não aprovado pelas partes interessadas.
Salvar Placar	5.3 - Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Recuperar Placar	5.3 - Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Explicar Jogo	5.3 - Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.

Fonte: produção do próprio autor

Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 23 - Jogo da Velha Virtual: ciclo 3 - leituras dos requisitos de software



Fonte: produção do próprio autor

Ciclo 4: Durante esse ciclo, foram apresentadas as definições e composições dos requisitos de software “Zerar Placar”, “Salvar Placar”, “Recuperar Placar” e “Explicar Jogo” aos jogadores de Jogo da Velha, onde os mesmos as avaliaram e as aprovaram.

Ao aplicar o Teodolito Hermenêutico de Requisitos durante o ciclo 4, foram obtidos os seguintes resultados:

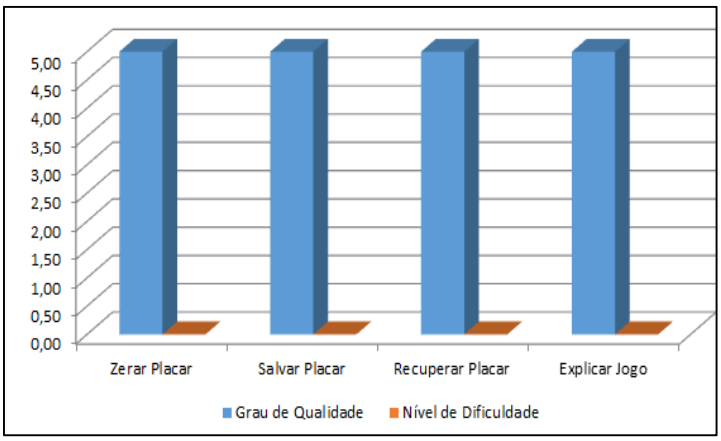
Tabela 22 - Jogo da Velha Virtual: ciclo 4 - Requisitos de Software

Requisito de Software	Grau de Qualidade
Zerar Placar	5.4 – Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Salvar Placar	5.4 – Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Recuperar Placar	5.4 – Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Explicar Jogo	5.4 – Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.

Fonte: produção do próprio autor

Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 24 - Jogo da Velha Virtual: ciclo 4 - leituras dos requisitos de software



Fonte: produção do próprio autor

6.2.**FAST**

Nesta seção, é apresentada a aplicação da Engenharia Hermenêutica de Requisitos em uma situação real, vivenciada pelo autor desta tese durante os anos 1996 e 2000, em um projeto de software desenvolvido para uma indústria siderúrgica chamada Tekno S/A, cuja sede localiza-se na cidade de São Paulo / SP e seu parque industrial situa-se em Guaratinguetá / SP. O projeto em si, aqui denominado FAST (Ferramenta de Apoio aos Serviços da Tekno), foi especificamente desenvolvido no parque industrial para automatizar seu processo de PPCP (Planejamento, Programação e Controle da Produção).

Em linhas gerais, um PPCP vem a ser um processo industrial que abrange todo o ciclo produtivo e envolve praticamente todas as áreas de uma empresa, principalmente a comercial, a de engenharia de produtos, a de produção, a de gestão de materiais, a de qualidade, a de manutenção, a financeira, a fiscal, a de recursos humanos e a de expedição.

Visão geral da FAST

A tabela 23 apresenta a descrição do problema exposto pela comunidade de negócio da Tekno e destaca também o seu impacto, a quem ele afeta e qual seria uma boa solução.

Tabela 23- FAST - Descrição do Problema

Descrição do Problema Informações apresentadas pela comunidade de negócio	
O problema de	Falta de controle das Ordens de Fabricação (OF) que entram e saem das Linhas de Pintura.
Afeta	Os clientes e a empresa como um todo, principalmente as áreas Comercial, PPCP, Planejamento e Administração de Materiais, Produção e Expedição.
Cujo impacto é	Clientes insatisfeitos, lucros reduzidos, desperdícios de matéria-prima, acúmulo dos estoques dos materiais de segunda e sucata e má qualidade do material acabado.
Uma boa solução seria	Desenvolver mecanismos de software para acompanhar e controlar todo o processo de pintura das bobinas de aço, permitindo tomar ações para evitar os desperdícios de matéria-prima e a má qualidade do material acabado, reduzir o tempo de produção e baixar os níveis dos estoques de material de segunda e de sucata.
Isto resultaria em	Clientes mais fiéis e satisfeitos, melhor controle e acompanhamento de todo o processo de pintura das bobinas de aço, menos desperdício de matéria-prima, baixos estoques de material de segunda e de sucata e maior lucro financeiro para a empresa.

Fonte: produção do próprio autor

A tabela 24 apresenta as principais necessidades expostas pela comunidade de negócio da Tekno e também destaca suas preocupações, as soluções atuais e para o futuro.

Tabela 24 - FAST - Principais Necessidades

Principais Necessidades Informações apresentadas pela comunidade de negócio			
Necessidade	**Preocupações**	**Soluções Atuais**	**Soluções Propostas**
Visualizar a programação das OF	• Não ter exatamente a sequência da produção a ser realizada. • Não atender adequadamente às necessidades dos clientes.	• Diariamente, a programação das OF, feita pelo PPCP, é impressa e entregue aos encarregados das Linhas de Pintura, que se encarregam de distribuí-la aos respectivos operadores das estações de trabalho, para que estes a sigam e	• Disponibilizar terminais de computadores nas Linhas de Pintura para que os operadores das estações de trabalho visualizem diretamente a programação das OF e possam interagir com o PPCP caso surjam

		façam os devidos apontamentos de produção. • Por uma questão prática das Linhas de Pintura, muitas vezes esta programação não é seguida, impactando em desperdício de matéria prima e atraso da produção.	necessidades de fazer quaisquer alterações. • Procedendo desta forma, serão tomadas ações coordenadas que evitarão os desperdícios de matéria-prima e o atraso da produção.
Necessidade	**Preocupações**	**Solução Atual**	**Solução Proposta**
Correto apontamento da produção	• Manter uma comunicação uniforme e constante entre os operadores das estações de trabalho durante o processo de produção. • Acompanhar os serviços realizados pelas Linhas de Pintura, para que o cliente esteja sempre bem informado sobre o processo de produção. • Melhorar o processo de programação das OF, que está deficiente devido à falta de histórico do planejado x realizado. • Orientar os operadores das estações de trabalho a	Os serviços realizados são diretamente apontados nos relatórios das OF Programadas. Esses apontamentos, na maioria das vezes, não condizem com a realidade, pois os operadores das estações de trabalho se preocupam em não mostrar valores muito diferentes dos programados. Isso gera a falsa informação de que a programação da OF foi realizada corretamente, com pouca margem de erro.	Disponibilizar serviços de software para que seja automatizada a coleta de dados e sejam registradas as verdadeiras ocorrências das Linhas de Pintura.

	seguirem e apontarem corretamente a programação das OF.		

Fonte: produção do próprio autor

FAST: Aplicação da Elicitação Hermenêutica de Requisitos

A tabela 25 apresenta o resultado da aplicação da Identificação de Diferença Situacional:

Tabela 25 - FAST: Identificação de Diferença Situacional

Identificação de Diferença Situacional	
O que está acontecendo?	Falta de controle das Ordens de Fabricação que entram e saem das Linhas de Pintura.
Para quem está acontecendo?	PPCP e Linha de Produção da Tekno.
Por que está acontecendo?	A Linha de Produção não sabe exatamente a sequência da produção a ser realizada para atender adequadamente às necessidades dos clientes. Não é possível estabelecer e manter uma comunicação uniforme e constante entre os operadores das estações de trabalho durante o processo de produção. Não é possível acompanhar os serviços realizados pela Linha de Produção. Não é possível atender aos clientes de maneira clara e adequada, devido à falta de histórico do que foi planejado pelo PPCP e o que foi realmente realizado pela Linha de Produção.
Quem são os envolvidos?	Os clientes e a empresa como um todo, principalmente as áreas Comercial, PPCP, Planejamento e Administração de Materiais, Produção e Expedição.
Como está acontecendo?	Os operadores das estações de trabalho não estão cumprindo a programação de ordens de fabricação definida pelo PPCP e estão apontando inadequadamente os serviços realizados. A análise feita pelo PPCP está sendo feita com base em dados alienados, portanto, dificultando a programação das ordens de fabricação do dia seguinte.

Fonte: produção do próprio autor

A tabela 26 apresenta o resultado da aplicação do Exame de Diferença Situacional:

Tabela 26 - FAST: Exame de Diferença Situacional

Exame de Diferença Situacional	
Cenário	FAST
Natureza	Indústria Siderúrgica
Meio-ambiente	Todos os setores envolvidos com a Linha de Produção de pintura de bobinas de aço, a saber: o comercial, o de engenharia de produtos, o de produção, o de gestão de materiais, o de qualidade, o de manutenção, o financeiro, o fiscal, o de recursos humanos e o de expedição.
Ambiente	Linha de Produção, formada por duas Linhas de Pintura.
Utensílios (ou insumos)	• Estações de Trabalho, onde cada Linha de Pintura contém: o Uma seção de entrada. o Uma seção de tratamento químico. o Uma seção de pintura. o Uma seção de inspeção do processo. o Uma seção de saída. • Um operador de cada estação de trabalho. • Segmentos da Ordem de Fabricação, específicos às necessidades de cada estação de trabalho. • Conforme as necessidades de produção: o Bobinas de aço e de proteção. o Produtos químicos e tambores de tinta.

Fonte: produção do próprio autor

A tabela 27 apresenta o resultado da aplicação da Determinação Hermenêutica de Requisitos:

Tabela 27 - FAST: Determinação Hermenêutica de Requisitos

Determinação Hermenêutica de Requisitos	
Necessidades Originais	Desenvolver mecanismos de software para acompanhar e controlar todo o processo de pintura das bobinas de aço, permitindo tomar ações para evitar os desperdícios de matéria-prima e a má qualidade do material acabado, reduzir o tempo de produção e baixar os níveis dos estoques de material de segunda e de sucata.
Expectativas	• Que sejam disponibilizados terminais de computadores nas Linhas de Pintura para que os operadores das estações de trabalho visualizem diretamente a programação das ordens de fabricação e

	possam interagir com o PPCP caso surjam necessidades de fazer quaisquer alterações, pois, procedendo desta forma, serão tomadas ações coordenadas que evitarão os desperdícios de matéria-prima e o atraso da produção. • Manter uma comunicação uniforme e constante entre os operadores das estações de trabalho durante o processo de produção. • Acompanhar os serviços realizados pelas Linhas de Pintura, possibilitando melhorar o atendimento aos clientes. • Melhorar o processo de programação das ordens de fabricação, devido ao fácil acesso do real histórico do que foi planejado pelo PPCP e realizado pela Linha de Produção.
Especificação Aceitável (Descrição do FAST)	Serão disponibilizados serviços de software que farão a coleta dos dados de maneira automatizada para que sejam registradas as verdadeiras ocorrências das Linhas de Pintura.
Especificação Aceitável (Dinâmica do FAST)	1. O PPCP programa as ordens de fabricação, conforme pedidos de compra dos clientes e histórico da produção. 2. Todas as estações de trabalho visualizam as ordens de fabricação programadas, de acordo com as informações que necessitam. 3. A seção de entrada inicia o processo de produção. 4. A seção de tratamento químico prepara os banhos químicos necessários para tirar as impurezas das bobinas de aço para que as tintas sejam nelas aplicadas. 5. A seção de pintura aplica as tintas e as proteções nas bobinas de aço. 6. A seção de inspeção de processo avalia a qualidade da aplicação das tintas e das proteções nas bobinas de aço. 7. A seção de saída mede a velocidade da produção e o peso das bobinas de aço pintadas e protegidas.

Fonte: produção do próprio autor

A tabela 28 apresenta o resultado da aplicação da Determinação Hermenêutica de Requisitos, para as descrições gerais dos requisitos de software:

Tabela 28 - FAST: Determinação Hermenêutica de Requisitos

Determinação Hermenêutica de Requisitos Requisitos de Software (RS)	
RS-1: Iniciar Produção	Assim que uma bobina de aço é iniciada pela Seção de Entrada, o software acusa essa ocorrência marcando sua respectiva OF como "em produção" e um aviso sobre essa marcação é disparado para todas as estações de trabalho dessa Linha de Pintura.
RS-2: Finalizar Produção	Assim que todas as bobinas de aço e de proteção programadas para a OF foram concluídas pela Seção de Saída e as que não foram utilizadas tenham sido retornadas e/ou devolvidas pela Seção de Entrada, é verificada se há alguma pendência a ser resolvida para a OF. Caso haja, a produção não pode ser finalizada. Caso não haja, a OF é marcada como "concluída" e um aviso sobre essa ocorrência é disparado para todas as estações de trabalho dessa Linha de Pintura.
RS-3: Cancelar Produção	Durante a execução de uma OF, caso ocorra algum problema que impeça definitivamente a continuação de sua produção, a Seção de Entrada retorna todas as matérias-primas programadas para esta OF aos seus respectivos estoques, explica o problema ocorrido e a OF é marcada como "cancelada". Nesse momento, todas as estações de trabalho dessa Linha de Pintura são notificadas sobre essa ocorrência, como também o PPCP.
RS-4: Inverter Produção	Durante a execução de uma OF, a sequência de produção de suas bobinas de aço pode ser invertida, por uma questão de praticidade e dinamismo da Linha de Pintura. Nesse caso, a Seção de Entrada inverte a sequência de execução das bobinas de aço programadas para uma OF, registra o motivo pelo qual a inversão está sendo feita e todas as estações de trabalho dessa Linha de Pintura são notificadas sobre essa ocorrência. Há também o caso de inverter a sequência de execução de OF. Nesse caso, para as OF que ainda não estejam em execução. Quando a sequência de OF é invertida pela Seção de Entrada, todas as estações de trabalho dessa Linha de Pintura são notificadas sobre essa ocorrência.
RS-5: Congelar Produção	Durante a execução de uma OF, caso ocorra algum problema que impeça temporariamente a continuação de sua produção, a Seção de Entrada marca a OF como "congelada" e explica o problema ocorrido. Nesse momento, todas as estações de

	trabalho dessa Linha de Pintura são notificadas sobre essa ocorrência e nenhuma ação poderá ser feita nessa OF.
RS-6: Descongelar Produção	Assim que os problemas da OF congelada forem resolvidos, a Seção de Entrada a descongela, liberando-a para produção. Todas as estações de trabalho dessa Linha de Pintura são notificadas sobre essa ocorrência.
RS-7: Iniciar Bobina de Aço	Inicia o processo de pintura do aço. Essa ação é realizada pela Seção de Entrada. Como uma OF pode conter várias bobinas de aço programadas, nem sempre o início do processo de pintura do aço coincide com o início de uma OF. Quando isto ocorre, automaticamente a OF para a qual a bobina está programada é iniciada. O Operador da Seção de Entrada deve registrar a espessura e a largura da Bobina de Aço para que a mesma seja marcada como "em produção".
RS-8: Gerar Ponta de Bobina	Quando é verificado que haverá ponta de bobina, a Seção de Saída pesa esta sobra, etiqueta-a e a retorna para o estoque de bobinas de aço. Nesse momento, todas as estações de trabalho da Linha de Pintura são notificadas sobre essa ocorrência.
RS-9: Gerar Sucata	Quando a produção de uma OF e/ou bobina de aço não passa satisfatoriamente pela avaliação feita pela Seção de Inspeção do Processo, a Seção de Saída, pesa, etiqueta e retorna para o estoque de sucata as bobinas de aço reprovadas. Nesse momento, todas as estações de trabalho da Linha de Pintura são notificadas sobre essa ocorrência, inclusive o PPCP.
RS-10: Apontar Tinta Aplicada	Conforme a Seção de Pintura for aplicando nas bobinas de aço as tintas e as bobinas de proteção, a mesma registra as quantidades aplicadas, as velocidades das aplicações e sob quais temperaturas foram aplicadas. Todas as estações de trabalho da Linha de Pintura são notificadas sobre essa ocorrência.
RS-11: Abrir RNC	Durante a avaliação dos níveis de qualidade, realizada pela Seção de Inspeção do Processo, caso haja alguma irregularidade, a mesma é registrada como "não conforme" e um relatório dos testes é registrado no relatório de não conformidade e enviado ao PPCP e à Seção de Saída. De acordo com a gravidade registrada no relatório de não conformidade, as bobinas de aço marcadas como "não conforme" serão devolvidas aos estoques de sucata ou de material de segunda. Esses retornos serão realizados pela Seção de Saída.

Fonte: produção do próprio autor

FAST: Aplicação do Teodolito Hermenêutico de Requisitos

Os requisitos de software para a FAST foram definidos e compostos em diversos ciclos ao longo de seu período de desenvolvimento. A título de demonstração, aqui serão apresentados dois recortes desse momento, chamados de ciclo 1 e ciclo 2, onde o ciclo 1 refere-se ao início do projeto e o ciclo 2 se encontra em um estágio mais avançado do mesmo.

Ciclo 1: Durante esse ciclo foram Identificadas as áreas de negócio PPCP e Linha de Produção, mas não há nenhum conhecimento detalhado sobre as mesmas. Em relação aos requisitos de software, ainda não há condições de saber algo sobre eles.

Ao aplicar o Teodolito Hermenêutico de Requisitos durante o ciclo 1, foram obtidos os seguintes resultados:

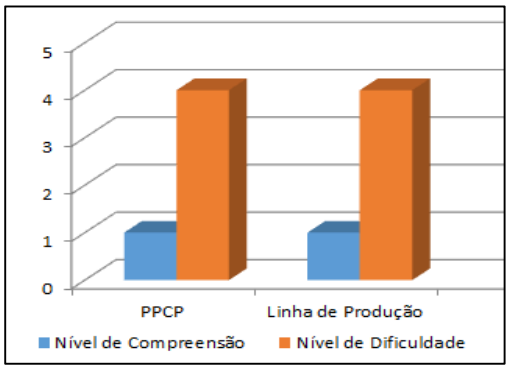
Tabela 29 - FAST: ciclo 1 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
PPCP	1 – Pré-conceitual
Linha de Produção	1 – Pré-conceitual

Fonte: produção do próprio autor

Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 25 - FAST: ciclo 1 - leituras do domínio da aplicação



Fonte: produção do próprio autor

Ciclo 2: Durante esse ciclo o PPCP foi melhor compreendido e foram detalhadas as estações de trabalho das Linhas de Pintura da Linha de Produção. Em relação aos requisitos de software, foram identificados e analisados os seguintes: iniciar produção, finalizar produção, cancelar produção, inverter produção, congelar produção, descongelar produção, iniciar bobina de aço, gerar ponta de bobina, gerar sucata, apontar tinta aplicada e abrir RNC.

Ao aplicar o Teodolito Hermenêutico de Requisitos durante o ciclo 2, foram obtidos os seguintes resultados:

Tabela 30 - FAST: ciclo 2 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
PPCP	3 – Contextual
Seção de Entrada	5 – Holístico
Tratamento Químico	3 – Contextual
Aplicação da Tinta	4 – Sistêmico
Inspeção do Processo	4 – Sistêmico
Seção de Saída	5 – Holístico

Fonte: produção do próprio autor

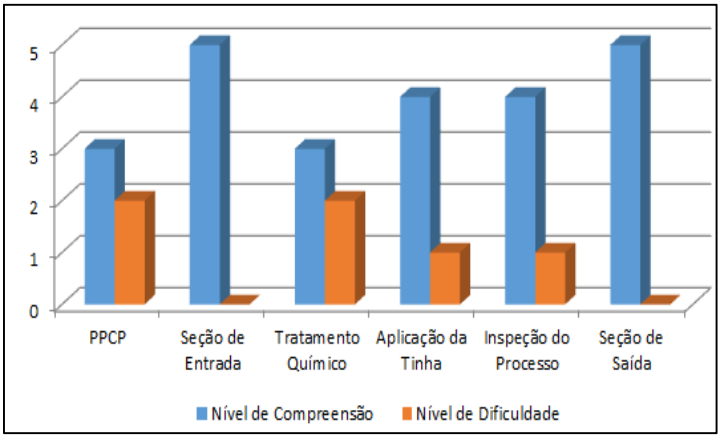
Tabela 31 - FAST: ciclo 2 - Requisitos de Software

Requisito de Software	Grau de Qualidade
Iniciar Produção	4.3 – O requisito foi satisfatoriamente composto.
Finalizar Produção	4.1 – O requisito está em conformidade com os padrões exigidos.
Cancelar Produção	3.3 – As partes interessadas aceitam que o requisito captura com precisão o que ele faz e não faz.
Inverter Produção	4.2 – O conjunto de itens do requisito fornece um claro valor para as partes interessadas.
Congelar Produção	5.1 – O requisito está satisfatoriamente completo e consistente.
Descongelar Produção	5.1 – O requisito está satisfatoriamente completo e consistente.
Iniciar Bobina de Aço	5.3 – Não há itens pendentes no requisito, impedindo seu aceite pelas partes interessadas.
Gerar Ponta de Bobina	3.4 – A alocação do requisito foi feita.
Gerar Sucata	3.3 – As partes interessadas aceitam que o requisito captura com precisão o que ele faz e não faz.
Apontar Tinta Aplicada	4.1 – O requisito está em conformidade com os padrões exigidos.
Abrir RNC	3.2 – O requisito está consistente em relação às expectativas das partes interessadas.

Fonte: produção do próprio autor

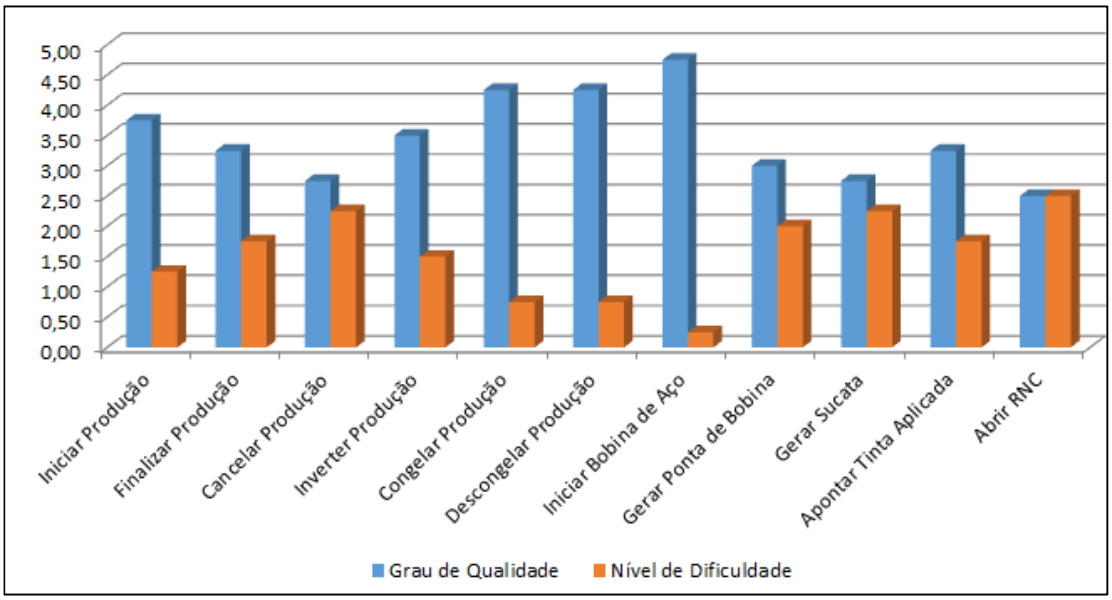
Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 26 - FAST: ciclo 2 - leituras do domínio da aplicação



Fonte: produção do próprio autor

Figura 27 - FAST: ciclo 2 – Graus de Qualidade dos Requisitos de Software



Fonte: produção do próprio autor

6.3. **Estudos de Casos extraídos da literatura**

A partir desta seção, apresentam-se as aplicações da Engenharia Hermenêutica de Requisitos em duas aplicações extraídas da literatura, para que seja possível analisar, de maneira comparativa, os resultados obtidos por ambas as abordagens e verificar a eficácia

e a eficiência das propostas apresentadas nesta tese e, assim, reforçar ainda mais as suas soluções e os seus benefícios.

A escolha da literatura

Após analisar algumas obras de alcance e importância internacionais, que aplicam seus conteúdos em situações práticas, optou-se por selecionar "*Object-Oriented Analysis and Design With Applications*", de Grady Booch, segunda e terceira edições.

Essa obra foi muito premiada em suas duas primeiras edições e se tornou referência essencial e indispensável àqueles que decidem aprender e aplicar corretamente o paradigma Orientado a Objetos para desenvolver sistemas de softwares.

A segunda edição é de 1994 e aborda o Método Booch para fazer a modelagem dos sistemas. Já a terceira edição é de 2007 e, então, a modelagem dos sistemas passou a ser feita com a UML.

Em ambas as edições, a primeira parte de cada obra fornece orientações práticas sobre o desenvolvimento de sistemas orientados a objetos, com o propósito de ensinar como aplicar na realidade o paradigma Orientado a Objetos. Em sua última parte, exemplifica o conteúdo da primeira parte o aplicando em situações consideradas de alta complexidade.

> This book provides practical guidance on the analysis and design of object oriented systems. (One of) its specific goals (is) the following: To teach the realistic application of object-oriented analysis and design within a variety of problem domains. [...] Section I examines the inherent complexity of software and the ways in which complexity manifests itself. [...] Section III offers a collection of five non trivial examples encompassing a diverse selection of problem domains. [...] We have chosen these particular problem domains because they are representative of the kinds of complex problems faced by the practicing software engineer. (BOOCH, 2007, p. XV - XVII).

A obra também afirma que, apesar de já existirem soluções técnicas para lidar com os sistemas de software complexos (sugerindo como melhor alternativa o paradigma Orientado a Objetos), ainda existem sérios problemas para desenvolvê-los com sucesso. Neste ponto, afirma que, neste caso, as limitações fundamentais se encontram na capacidade humana de lidar com a complexidade.

> If we know what the design of complex software systems should be like, then why do we still have serious problems in successfully developing them? This

> concept of the organized complexity of software (whose guiding principles we call the object model) is relatively new. However, there is yet another factor that dominates: the fundamental limitations of the human capacity for dealing with complexity. (BOOCH, 2007, p. 17 - 18).

O foco, então, passa a ser a análise desses sistemas de software (Engenharia de Requisitos). Como a complexidade vem aumentando cada vez mais e nossa capacidade humana de lidar com ela é limitada, surge a necessidade de melhorar a maneira de conceber estes sistemas.

> As we first begin to analyze a complex software system, we find many parts that must interact in a multitude of intricate ways, with little perceptible commonality among either the parts or their interactions; this is an example of disorganized complexity. As we work to bring organization to this complexity through the process of design, we must think about many things at once. [...] We are thus faced with a fundamental dilemma. The complexity of the software systems we are asked to develop is increasing, yet there are basic limits on our ability to cope with this complexity. How then do we resolve this predicament? (BOOCH, 2007, p. 18).

A solução proposta, sugerida então, é disciplinar a forma de como conceber estes sistemas para, assim, conseguir lidar com a complexidade, a ponto de dominá-la.

> Certainly, there will always be geniuses among us, people of extraordinary skill who can do the work of a handful of mere mortal developers, the software engineering equivalents of Frank Lloyd Wright or Leonardo da Vinci. These are the people whom we seek to deploy as our system architects: the ones who devise innovative idioms, mechanisms, and frameworks that others can use as the architectural foundations of other applications or systems. However, “The world is only sparsely populated with geniuses. There is no reason to believe that the software engineering community has an inordinately large proportion of them”. Although there is a touch of genius in all of us, in the realm of industrial-strength software we cannot always rely on divine inspiration to carry us through. Therefore, we must consider more disciplined ways to master complexity. (BOOCH, 2007, p. 18).

Em outra oportunidade, a obra afirma que “a complexidade do software não é uma propriedade acidental, mas essencial” e que há quatro elementos determinantes a ela que devem ser considerados: a complexidade do domínio do problema, a dificuldade em gerenciar o processo de desenvolvimento do software, a flexibilidade do software e os problemas de caracterização do comportamento de sistemas discretos.

> As Brooks suggests, "The complexity of software is an essential property, not an accidental one". We observe that this inherent complexity derives from four elements: the complexity of the problem domain, the difficulty of managing the developmental process, the flexibility possible through software, and the problems of characterizing the behavior of discrete systems. (BOOCH, 1994, p. 3).

Em relação à complexidade do domínio do problema, destacam-se os vários elementos que o software deverá considerar, os quais, muitas vezes são concorrentes e/ou

contraditórios entre si. Isto faz com que a concepção dos requisitos se torne extremamente complexa.

> The problems we try to solve in software often involve elements of inescapable complexity, in which we find a myriad of competing, perhaps even contradictory, requirements. Consider the requirements for the electronic system of a multi-engine aircraft, a cellular phone switching system, or an autonomous robot. The raw functionality of such systems is difficult enough to comprehend, but now add all of the (often implicit) nonfunctional requirements such as usability, performance, cost, survivability, and reliability. This unrestrained external complexity is what causes the arbitrary complexity about which Brooks writes. (BOOCH, 1994, p. 3 - 4).

Esta extrema complexidade também ocorre devido à "incompatibilidade de impedância" que existe entre os *stakeholders* de um sistema de software e seus desenvolvedores. Isso ocorre porque ambos possuem experiências distintas, que os fazem ter diferentes perspectivas sobre a natureza do problema e, consequentemente, da sua solução.

> This external complexity usually springs from the "impedance mismatch" that exists between the users of a system and its developers: users generally find it very hard to give precise expression to their needs in a form that developers can understand in extreme cases, users may have only vague ideas of what they want in a software system. This is not so much the fault of either the users or the developers of a system; rather, it occurs because each group generally lacks expertise in the domain of the other. Users and developers have different perspectives on the nature of the problem and make different assumptions regarding the nature of the solution. (BOOCH, 1994, p. 4).

Mesmo que não ocorra esta "incompatibilidade de impedância" entre os *stakeholders* de um sistema de software e seus desenvolvedores, há poucos instrumentos que auxiliam a captura precisa de seus requisitos. Muitas vezes, são produzidos documentos difíceis de serem compreendidos, que descrevem mais os elementos do projeto, em vez de descreverem os requisitos em sua essência.

> Actually, even if users had perfect knowledge of their needs, we currently have few instruments for precisely capturing these requirements. The common way of expressing requirements today is with large volumes of text, occasionally accompanied by a few drawings. Such documents are difficult to comprehend, are open to varying interpretations, and too often contain elements that are designs rather than essential requirements. (BOOCH, 1994, p. 4).

Outro fator que contribui para com esta complexidade é a mutabilidade dos requisitos de software, que geralmente provoca alterações nas regras dos problemas (e consequentemente de suas soluções). Por outro lado, essas mudanças também ajudam os desenvolvedores do software a compreenderem melhor o domínio destes problemas.

> A further complication is that the requirements of a software system often change during its development, largely because the very existence of a software development project alters the rules of the problem. Seeing early products, such as design documents and prototypes, and then using a system once it is installed and operational, are forcing functions that lead users to better understand and articulate their real needs. At the same time, this process helps developers master the problem domain, enabling them to ask better questions that illuminate the dark comers of a system's desired behavior. (BOOCH, 1994, p. 4).

Mesmo quando um sistema complexo de software entra em operação, com o tempo de uso, seus requisitos continuam a sofrer mudanças que fazem parte de sua evolução comum e necessária. Apesar de este ser o cenário comum, esperado e requerido, o que está ocorrendo é um gasto de energia e recurso para manter o mesmo antigo e decadente software em operação.

> Because a large software system is a capital investment, we cannot afford to scrap an existing system every time its requirements change. Planned or not, large systems tend to evolve over time, a condition that is often incorrectly labeled software maintenance. To be more precise, it is maintenance when we correct errors; it is evolution when we respond to changing requirements; it is preservation when we continue to use extraordinary means to keep an ancient and decaying piece of software in operation. Unfortunately, reality suggests that an inordinate percentage of software development resources are spent on software preservation. (BOOCH, 1994, p. 4).

Desse modo, vê-se que o ponto crucial para se alcançar o êxito no desenvolvimento de software é atingir, principalmente, uma boa compreensão e interpretação do domínio da aplicação, de onde serão extraídos e compostos os requisitos de software, que serão utilizados como guias para direcionar todas as demais atividades do projeto de desenvolvimento do software.

A escolha das aplicações

Uma vez selecionadas as obras, passou-se a analisá-las para escolher as aplicações nelas descritas e optou-se por uma aplicação de cada obra: “Computação Cliente / Servidor: Acompanhamento de Inventário”, da segunda edição e “Inteligência Artificial: Criptoanálise”, da terceira edição. Essas aplicações foram selecionadas por serem de propósitos mais abrangentes.

Aplicação “Cliente / Servidor: Acompanhamento de Inventário”

Projeto de desenvolvimento de uma aplicação cujo propósito geral é rastrear o inventário de um centro de distribuição que possui armazéns regionais relativamente autônomos em relação ao gerenciamento de seus estoques e processamentos dos pedidos.

> As part of its expansion into several new and specialized markets, a mail-order catalog company has decided to establish a number of relatively autonomous regional warehouses. Each such warehouse retains local responsibility for inventory management and order processing. To target niche markets efficiently,each warehouse is tasked with maintaining inventory that is best suited to the local market. The specific product line that each warehouse manages may differ from region to region; furthermore, the product line managed by any one region tends to be updated almost yearly to keep up with changing consumer tastes. For reasons of economies of scale, the parent company desires to have a common inventory- and order-tracking system across all its warehouses. (BOOCH, 1994, p. 381).

Para atingir seu objetivo, essa aplicação deverá oferecer, principalmente, as seguintes funções: acompanhar o inventário, rastrear os pedidos, gerar guias de embalagens, gerar faturas, acompanhar as contas a pagar e receber, acompanhar as tendências de vendas, identificar clientes e fornecedores (problemáticos e bem avaliados) e realizar programas promocionais especiais.

> The key functions of this system include: (BOOCH, 1994, p. 381 - 382):
>
> - Tracking inventory as it enters the warehouse, shipped from a variety of suppliers.
> - Tracking orders as they are received from a central but remote telemarketing organization; orders may also be received by mail, and are processed locally.
> - Generating packing slips, used to direct warehouse personnel in assembling and then shipping an order.
> - Generating invoices and tracking accounts receivable.
> - Generating supply requests and tracking accounts payable.
>
> In addition to automating much of the warehouse's daily workflow, the system must provide a general and open-ended reporting facility, so that the management team can track sales trends, identify valued and problem customers and suppliers, and carry out special promotional programs.

Outras necessidades para o sistema envolvem, principalmente, questões relacionadas à operação e controle do software; por exemplo, a equipe de contabilidade poderá realizar consultas gerais, o departamento de compras poderá consultar os registros contábeis relativos aos pagamentos de fornecedores e um estoquista não poderá emitir cheques. Isto indica que o sistema deverá possuir mecanismos de controle à rede para restringir e/ou conceder direitos a determinadas informações e funcionalidades.

> There should be nothing in our software architecture that constrains a specific PC to only one activity: the accounting team should be able to perform general queries, and the purchasing department should be able to query accounting records concerning supplier payments. In this manner, as changing business conditions dictate, management can add or reallocate computing resources as needed to balance the daily workflow. Of course, security requirements dictate that some management discipline is needed: a stockperson should not be allowed to send out checks. We delegate responsibility for these kinds of constraints as an operational consideration, carried out by general network access-control

> mechanisms that either constrain or grant rights to certain data and applications. (BOOCH, 1994, p. 384).

Nesta mesma linha de necessidades relacionadas à operação do sistema, foi solicitado que os estoquistas trabalhem com computadores portáteis sem fio para que as operações de suas tarefas se tornem mais práticas e eficientes, pois estes estão sempre em movimento pelo armazém e não podem possuir pontos fixos para terem acesso à aplicação.

> Basically, our plan will be to give each stockperson a handheld PC. As new inventory is placed in the warehouse, they use these devices to report the fact that the stock is now in place, and also notify the system where it is located; as orders for the day are assigned to be filled, packing orders are transmitted to these devices, directing workers where to find certain stock, as well as how many of each to retrieve to pass on to shipping. (BOOCH, 1994, p. 384).

Em se tratando, ainda, à operação do sistema, outra preocupação que foi identificada diz respeito às interfaces com os usuários, onde, por exemplo, para os clientes, é melhor que eles interajam com a aplicação por meio de *menus*, já as funções de planejamento, compras e estoques ficam mais interativas com interfaces baseadas em *janelas* e os estoquistas devem possuir interfaces mais simples para tornarem suas atividades mais práticas.

> In fact, it is possible (and highly desirable) to permit a variety of user interfaces for this system. For example, a simple, interactive, menu-oriented interface is most likely adequate for customers who submit their own orders. Modern, window-based interfaces are likely best for the planning, purchasing, and accounting functions. Hardcopy reports may best be generated in a batch environment, although some managers may wish to use a graphic interface to view trends interactively. Stockpersons need an interface that: is simple; mouse-driven windowing systems don't work well in the industrial environment of a warehouse, and furthermore, training costs are an issue to consider. (BOOCH, 1994, p. 385 - 386).

Após apresentar o escopo do sistema, foram analisados os cenários de uso, onde foram enumerados e relacionados de acordo com os vários elementos funcionais identificados para a aplicação:

- Um cliente faz um pedido.
- Um cliente consulta o status de um pedido.
- Um cliente adicionar e/ou remove itens em um pedido existente.
- Um estoquista recebe uma ordem de embalagem para separar os produtos de um pedido de um cliente.
- A expedição recebe um pedido de cliente para envia-lo pelos correios.

- A contabilidade emite nota fiscal ao cliente.
- Realizam-se compras locais para obter um novo inventário.
- A compra adiciona um novo fornecedor ou remove um já existente.
- Compras consulta o status de uma ordem de fornecedor existente.
- O recebimento aceita uma remessa de um fornecedor.
- Um estoquista coloca novos estoques em inventário.
- A contabilidade emite cheque em relação a um pedido de compra de novo estoque.
- O departamento de planejamento gera um relatório de tendências, mostrando a atividade de vendas para vários produtos.
- Para fins de relatórios fiscais, o departamento de planejamento gera um resumo que mostra os níveis atuais do inventário.

> Now that we have established the scope of our system, we continue our analysis by studying several scenarios of its use. We begin by enumerating a number of primary use cases, as viewed from the various functional elements of the system: (BOOCH, 1994, p. 388).
>
> - A customer phones the remote telemarketing organization to place an order.
> - A customer mails in an order.
> - A customer calls to find out about the status of an order.
> - A customer calls to add items to or remove items from an existing order.
> - A stockperson receives a packing order to retrieve stock for a customer order.
> - Shipping receives an assembled order and prepares it for mailing.
> - Accounting prepares a customer invoice.
> - Purchasing places an order for new inventory.
> - Purchasing adds or removes a new supplier.
> - Purchasing queries the status of an existing supplier order.
> - Receiving accepts a shipment from a supplier, placed against a standing purchase order.
> - A stockperson places new stock into inventory.
> - Accounting cuts a check against a purchase order for new inventory.
> - The planning department generates a trend report, showing the sales activity for various products.
> - For tax-reporting purposes, the planning department generates a summary showing Current inventory levels.

Para cada um desses cenários principais, foi identificada uma série de secundários:

- Um item solicitado por um cliente está fora de estoque ou em atraso.
- O pedido de um cliente está incompleto ou menciona números de produtos incorretos ou obsoletos.

- Um cliente pretende consultar ou alterar um pedido, mas não consegue lembrar o que exatamente foi solicitado, por quem, ou quando.
- Um estoquista recebe uma ordem de embalagem para separar os produtos, mas estes não podem ser encontrados (ou algum deles).
- A expedição recebe um pedido de cliente para enviá-lo pelos correios, mas o mesmo está incompleto.
- Um cliente não pagou uma fatura.
- Compras coloca um pedido de inventário novo, mas o fornecedor saiu do mercado ou já não possui o item.
- O recebimento aceita uma remessa incompleta de um fornecedor.
- O recebimento aceita uma remessa de um fornecedor para o qual nenhum pedido de compra pode ser encontrado.
- Ocorrem alterações de código de imposto de negócios, exigindo que o departamento de planejamento gere uma série de novos relatórios de inventário.

> For each of these primary scenarios, we can envision a number of secondary ones: (BOOCH, 1994, p. 388 - 389).
> - An item a customer requested is out of stock or on backorder.
> - A customer's order is incomplete, or mentions incorrect or obsolete product numbers.
> - A customer calls to query about or change an order, but can't remember what exactly was ordered, by whom, or when.
> - A stockperson receives a packing order to retrieve stock, but the item cannot be found. Shipping receives an incompletely assembled order.
> - A customer fails to pay an invoice.
> - Purchasing places an order for new inventory, but the supplier has gone out of business or no longer carries the item.
> - Receiving accepts an incomplete shipment from a supplier.
> - Receiving accepts a shipment from a supplier for which no purchase order can be found.
> - A stockperson places new stock into inventory, only to discover that there is no space for the item.
> - Business tax code changes, requiring the planning department to generate a number of new inventory reports.

Devido à complexidade desta aplicação, explicou-se que dezenas de cenários primários e muitos outros cenários secundários seriam identificados, mas como esse processo levaria semanas para ser concluído, sugeriu-se a aplicação da “regra de 80%” para evitar a “paralisia da análise”, pois nenhuma quantidade de tempo será suficiente para elaborar uma lista completa de todos os cenários.

> In fact, this part of the analysis Process would probably take several weeks to complete to any reasonable level of detail. For this reason, we strongly suggest applying the 80% rule of thumb: don't wait to generate a complete list of scenarios (no amount of time will be sufficient). (BOOCH, 1994, p. 389 - 390).

A “regra de 80%” diz respeito ao estudo de cerca de 80% dos cenários de uso principais mais interessantes e uma rápida prova de conceito sobre estes para verificar se o processo está fluindo bem.

> [...] But rather, study some 80% of the (scenarios) interesting ones, and if possible, try a quick-and-dirty proof of concept to see if this part of analysis is in the right track. (BOOCH, 1994, p. 390).

Quanto à “paralisia da análise”, a obra explica e sugere, em uma nota de rodapé, que o ciclo da análise do software não deve demorar mais do que o tempo em que a janela de oportunidade para o negócio em que ele aturará estiver aberta, para que o projeto não seja abandonado antes de seu fim e, também, para que o desenvolvedor de software continue atuando no mercado de trabalho.

> But beware of analysis paralysis: if the software analysis cycle takes longer than the window of opportunity for the business, then abandon hope, all ye who follow this path, for you will eventually be out of business. (BOOCH, 1994, p. 390).

Após toda esta explanação e justificativas, explicou-se que, para os propósitos da referida obra, foram elaborados apenas dois dos cenários principais do sistema.

> For the purposes of this chapter, let's elaborate upon two of the system's primary scenarios. (BOOCH, 1994, p. 390).

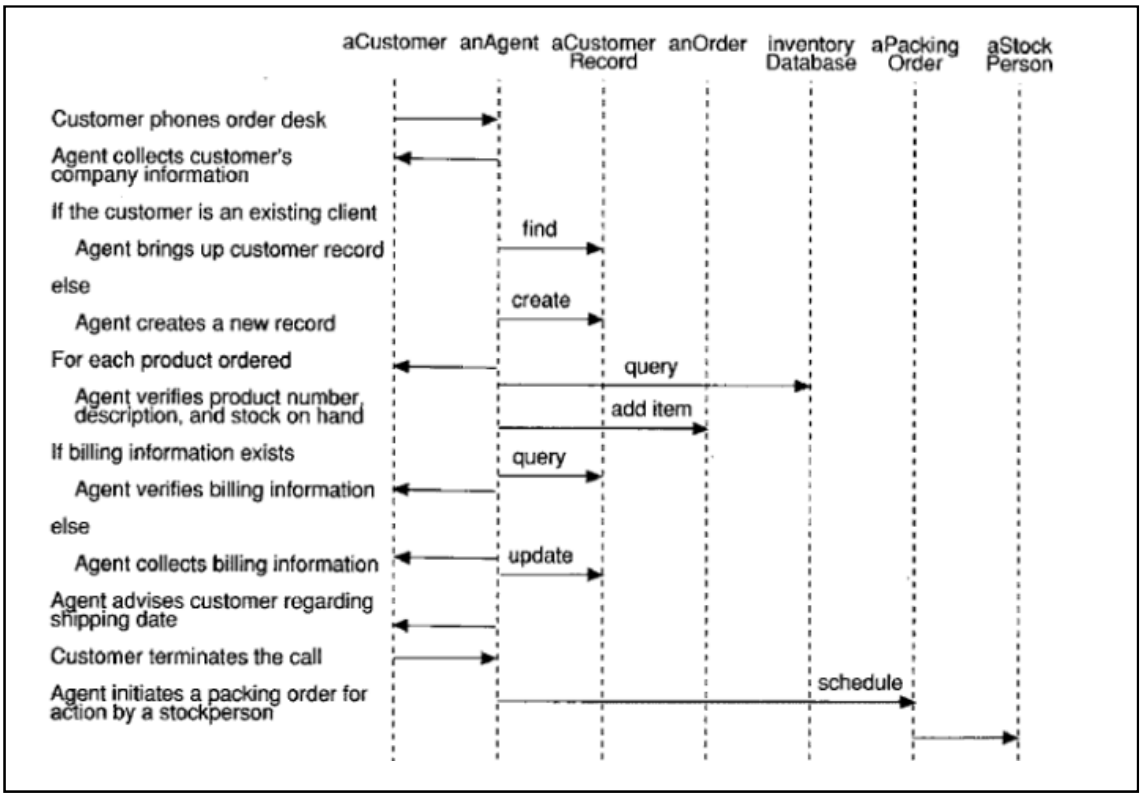
Para o cenário referente ao pedido de compra feito remotamente pelo cliente, o fluxo de sequencia descrito é o seguinte:

1. Cliente entra em contato com armazém via telefone.
2. A Agente recolhe informações da empresa do cliente.
3. Se o cliente é um cliente existente.
    - O agente recupera o registro do cliente.
4. Senão
    - O agente cria um novo registro para o cliente.
5. Para cada produto solicitado
    - O agente verifica o número do produto, a descrição e o estoque disponível.
6. Se houver informações de cobrança
    - O agente verifica as informações de cobrança.

7. Senão
    - O agente coleta as informações de cobrança.
8. O agente informa o cliente sobre a data de envio.
9. O cliente encerra a chamada.
10. O agente inicia uma ordem de embalagem junto a um estoquista

Este fluxo de sequencia foi modelado usando o diagrama de sequencia, conforme ilustrado na figura 28:

Figura 28 - Pedido de Compra feito remotamente pelo cliente



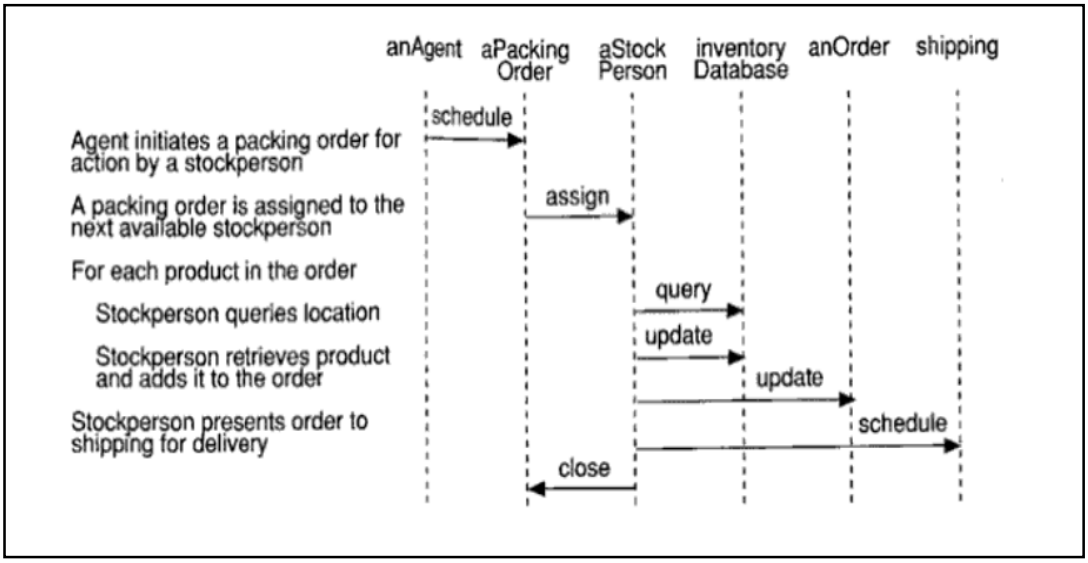
Fonte: BOOCH, 1994, p. 390

Para o cenário referente à ordem de embalagem, o fluxo de sequencia descrito é o seguinte:

1. Agente inicia uma ordem de embalagem.
2. Uma ordem de embalagem é atribuída ao próximo estoquista disponível.
3. Para cada produto na ordem de embalagem
    - O estoquista consulta a localização do produto no estoque.
    - O estoquista recupera o produto e o adiciona à ordem.
4. Estoquista envia a ordem para expedição.

Este fluxo de sequencia foi modelado usando o diagrama de sequencia, conforme ilustrado na figura 29:

Figura 29 - Ordem de Embalagem



Fonte: BOOCH, 1994, p. 39

Durante a análise dessa aplicação, foram identificadas as pessoas que vão interagir com o sistema: cliente, fornecedor, agente de pedidos, contador, agente de transportes, estoquista, agente de compras, agente de recebimentos e planejador. Essa identificação é importante para conhecer os diferentes *papéis* que vão interagir com a aplicação e também para restringir ou conceder acesso às informações e funcionalidades do sistema, de acordo com o perfil de cada um desses *papéis*, como também poder controlar melhor o uso da aplicação.

> By anthropomorphizing our abstractions in this manner, for each of the system's function points we eventually come to discover many of the interesting high-level objects within out system. Specifically, our analysis leads us to discover the following abstractions. First, we list the various people that interact with the system: Customer, Supplier, Order Agent, Accountant, Shipping Agent, Stoffierson, Purchasing Agent, Receiving Agent and Planner. It is important for us to identify these classes of people, because they represent the different roles that people play when interacting with the system. lf we desire to track the who, when, and why of certain events that took place within our system, then we must formalize these roles. For example, when resolving a complaint, we might like to identify what people within the company had recently interacted with the unhappy customer, and only by making this a part of our enterprise model do we retain enough information to make an intelligent analysis. in addition to serving an outwardly visible role, it is important for us to distinguish among these classes of people for the purpose of operationally restricting or granting access to parts of the system's functionality. With an open network, this form of centralized control is a reasonably effective way to control accidental or malicious misuse. (BOOCH, 1994, p. 391 - 392).

Durante a análise dessa aplicação, também foram identificadas as abstrações-chave que representam algumas informações a serem manipuladas pela aplicação: registro do cliente, registro do produto, registro do fornecedor, pedido do cliente, ordem de compra, fatura, ordem de embalagem, ordem de estocagem e etiqueta de remessa.

> Our analysis also reveals the following key abstractions, each of which represents some information manipulated by the system: Customer Record, Product Record, Supplier Record, Order, Purchase Order, Invoice, Packing Order, Stocking Order and Shipping Label. (BOOCH, 1994, p. 392).

A fase de análise da aplicação foi completada com a definição dos vários tipos de consultas que os usuários poderão gerar. Essas consultas serão feitas por um mecanismo de relatório geral, o qual permitirá que novos relatórios sejam adicionados, pois todos eles compartilharão comportamentos e estruturas comuns, além de uma aparência homogênea.

> We include the abstraction Report to denote the base class of all the various kinds of hardcopy and online queries users might generate. Our detailed analysis by scenario will probably discover many of the concrete kinds of reports that our workflow demands, but because of the open-ended nature of our system, we are best advised to develop a more general reporting mechanism, so that new reports can be added in a consistent fashion. Indeed, by identifying the commonality among reports, we make it possible for all such reports to share common behavior and structure, thereby simplifying our architecture as well as allowing our system to present a homogeneous look and feel to its users. (BOOCH, 1994, p. 394).

Nesse momento da fase de análise, apesar dela ainda estar incompleta, percebeu-se que já é possível iniciar o projeto arquitetônico dessa aplicação e, então, findou-se a seção que aborda sua concepção.

> Our list of things in the system is by no means complete, but we have sufficient information at this point to begin to move on to architectural design. (BOOCH, 1994, p. 394).

Acompanhamento de Inventário: Aplicação da Elicitação Hermenêutica de Requisitos

O resultado da aplicação da Elicitação Hermenêutica de Requisitos no projeto de Acompanhamento de Inventário é apresentado nas tabelas a seguir.

A Tabela 32 apresenta o resultado da aplicação da Identificação de Diferença Situacional no projeto de Acompanhamento de Inventário.

Tabela 32 - Acompanhamento de Inventário: Identificação de Diferença Situacional

Identificação de Diferença Situacional	
O que fazer?	Sistema de Rastreamento de Inventário
Para quem fazer?	Central de Distribuição de Produtos
Por que fazer?	Para que cada armazém local possa fazer de forma autônoma o gerenciamento de seu estoque e o processamento de seus pedidos, mantendo o inventário mais adequado ao mercado local e fazendo atualizações de suas linhas de produtos, para que seja possível acompanhar as mudanças dos gostos dos consumidores.
Quem são os envolvidos?	Central de Distribuição de Produtos, Armazéns Locais, Clientes, Fornecedores, Agentes de Pedidos, Contadores, Agentes de Transportes, Estoquistas, Agentes de Compras, Agentes de Recebimentos e Planejadores.
Como é o dia a dia de um armazém local?	De modo geral, as atividades operacionais de cada armazém local de maior destaque são: 1. Entrada de encomenda: responsável por aceitar pedidos de clientes e por responder a consultas de clientes sobre o status de um pedido. 2. Contabilidade: Responsável pela geração e envio das faturas e acompanhamento das contas a receber, bem como pela geração e rastreamento das contas a pagar. 3. Envio: responsável pela montagem de pacotes para embarque em apoio ao preenchimento de pedidos de clientes. 4. Estoque: Responsável por colocar novos inventários em estoque, bem como para recuperar inventário em suporte para preencher pedidos de clientes. 5. Compras: responsável pela encomenda de fornecedores de estoque e acompanhamento de envios de fornecedores. 6. Recebimento: Responsável pelo aceite e registro de estoque de fornecedores. 7. Planejamento: Responsável por gerar relatórios para gerenciamento e estudar tendências em níveis de inventário e atividade do cliente. Para isso, as principais tarefas realizadas no dia a dia são: • Acompanhamento do inventário à medida que ele entra no armazém, enviado de uma variedade de fornecedores.

	• Rastreamento de pedidos à medida que são recebidos de uma organização de telemarketing central, mas remota; Os pedidos também podem ser recebidos por correio e são processados localmente. • Geração de embalagem direcionada ao pessoal do armazém na montagem e envio de um pedido. • Geração de faturas e acompanhamento de contas a receber. • Geração de solicitações de fornecimento e acompanhamento de contas a pagar. • Acompanhamento das tendências de vendas, análise de clientes e fornecedores bem avaliados e também os problemáticos. • Realização de programas promocionais especiais. Para que essas tarefas sejam realizadas com maior eficiência, todos têm acesso a um espectro maior de informações, além das de seu setor. Por exemplo, a equipe de contabilidade pode acessar informações gerais e o departamento de compras pode acessar as informações da contabilidade para saber sobre os pagamentos a fornecedores, mas cada qual só pode executar seu conjunto de tarefas. Por exemplo: um estoquista não pode emitir um cheque. Ao longo do tempo, novas linhas de produtos são gerenciadas por cada armazém, novos clientes e fornecedores são adicionados aos respectivos catálogos e alguns antigos são removidos. De modo mais específico, o dia a dia de um armazém local se ocupa com as seguintes situações: • Um cliente telefona para a organização remota de telemarketing para fazer um pedido. • Um cliente envia mensagens por ordem de pedido. • Um cliente entra em contato com o armazém para descobrir o status de um pedido. • Um cliente entra em contato com o armazém para adicionar itens ou remover itens de um pedido existente. • Um cliente entra em contato com o armazém e faz uma queixa. • Um estoquista recebe uma ordem de embalagem para recuperar o estoque para uma ordem de pedido do cliente. • O agente de transportes recebe um pedido montado e o prepara para enviar por correio. • A contabilidade prepara uma nota fiscal do cliente.

	• A compra obtém um novo item para o inventário local, conforme demanda da região. • A compra adiciona ou remove do catálogo um fornecedor. • Compras questiona o status de uma ordem de fornecedor existente. • O agente de recebimentos aceita uma remessa de um fornecedor. • Um estoquista coloca novos estoques em inventário. • A contabilidade reduz um cheque em relação a um pedido de compra de novo estoque. • O departamento de planejamento gera um relatório de tendências, mostrando a atividade de vendas para vários produtos. • Para fins de relatórios fiscais, o departamento de planejamento gera um resumo que mostra os níveis atuais do inventário.

Fonte: produção do próprio autor

A Tabela 33 apresenta o resultado da aplicação do Exame de Diferença Situacional no projeto de Acompanhamento de Inventário.

Tabela 33 - Acompanhamento de Inventário: Exame de Diferença Situacional

Exame de Diferença Situacional	
Cenário	Sistema de Informações Gerenciais: Acompanhamento de Inventário
Natureza	Centro de Distribuição de Produtos
Meio-ambiente	Armazém Central e Mercado Local
Ambiente	Armazém Regional
Utensílios (ou insumos)	• Catálogo (e linha) de produtos • Estoque (armazenamento de itens / produtos e suas localizações internas) • Inventário • Composição de custos (administrativos e fiscais) • Cliente • Fornecedor • Avaliação de clientes • Avaliação de fornecedores • Embalagem • Pedido de compra • Pedido de embalagem

	• Pedido de estocagem • Ordem de expedição (com rótulo de remessa) • Nota Fiscal • Contas (a pagar e a receber) • Ordem de cobrança • Fatura • Programas promocionais • Gostos dos consumidores (sugestões / preferências / lista de desejos) • Tendências de vendas • Rastreamento • Queixas • Mecanismo de relatórios

Fonte: produção do próprio autor

A Tabela 34 apresenta o resultado da aplicação da Determinação Hermenêutica de Requisitos no projeto de Acompanhamento de Inventário.

Tabela 34 - Acompanhamento de Inventário: Determinação Hermenêutica de Requisitos

Determinação Hermenêutica de Requisitos	
Necessidades Originais	A empresa matriz (Central de Distribuição) deseja ter um sistema comum de rastreamento de estoques e pedidos em todos os seus armazéns e, assim, poder acompanhar o inventário de cada armazém como também de todo o conjunto de armazéns. Desse modo, será possível funcionar como uma eficiente central de distribuição de produtos.
Expectativas	Nosso plano será dar a cada estoquista um PC portátil. À medida que um novo inventário é colocado no armazém, eles usam esses dispositivos para anunciar o fato de que o estoque está agora no local e também notificar o sistema onde ele está localizado; À medida que os pedidos para o dia são designados para serem processados, os pedidos das embalagens são transmitidos para esses dispositivos, direcionando os trabalhadores para encontrar determinado estoque, como também para a expedição fazer seu planejamento. Gostaríamos de gerar uma etiqueta de remessa para o cliente associado, para permitir a navegação desta ordem de volta para o cliente. Dada uma ordem de embalagem, gostaríamos de navegar de volta ao cliente e ao agente de pedidos, para anunciar o fato dos itens estarem nos pedidos; isso exige que naveguemos da

	ordem de embalagem de volta ao pedido e, em seguida, voltemos ao cliente e ao agente de pedidos. Queremos saber quais os produtos que os clientes mais compram em determinadas épocas do ano. Esta consulta exige que possamos navegar do cliente de volta para todos os pedidos pendentes e anteriores. Nossas regras de negócios estabelecem que cada ordem de embalagem seja exclusiva de uma determinada ordem de pedido. No entanto, suponha que o armazém esteja fora de estoque para certos itens referenciados na ordem original: temos que agendar uma segunda ordem de embalagem, cujos itens estão de volta ao estoque. As interfaces com os usuários devem ser simples e interativas, projetadas de acordo com os perfis e tarefas dos usuários.
Especificação Aceitável (descrição)	Com o Sistema de Acompanhamento de Inventário será possível: acompanhar o inventário desde sua entrada no Armazém, criar e gerenciar o catálogo (e linha) de produtos, rastrear os pedidos à medida que eles são criados, indicar aos estoquistas a localização do produto no estoque, gerar (ou agendar) embalagens e direcioná-las ao pessoal do armazém durante a montagem e envio de um pedido, planejar e gerenciar a expedição, gerar os rótulos de remessas, compor os custos administrativos e fiscais, gerar as faturas, notas fiscais e ordens de cobrança, acompanhar as contas a pagar e receber, gerar as solicitações de fornecimento e os pedidos de estocagem, acompanhar as tendências de vendas, avaliar os clientes e fornecedores, capturar e analisar os gostos dos consumidores, realizar programas promocionais, receber, analisar, resolver e acompanhar as queixas dos clientes e gerar consultas gerais de todas as informações contidas no sistema, conforme as necessidades de seus usuários.
Especificação Aceitável (dinâmica)	Em relação ao fluxo geral do acompanhamento de inventário: 1. O armazém gera o catálogo (e a lista) de produtos. 2. O armazém gera e gerencia o seu inventário. 3. O armazém rastreia todos os pedidos dos

	clientes. 4. O armazém analisa as tendências de vendas. 5. O armazém analisa os gostos dos consumidores. 6. O armazém compra produtos (novos e de reposição). Em relação ao fluxo geral de um pedido de compra: 1. O cliente faz um pedido de compra. 2. O estoquista localiza e separa os produtos. 3. O estoquista gera uma embalagem para os produtos (ou agenda a geração de embalagens). 4. O sistema gera uma ordem de expedição. 5. O sistema gera o rótulo de remessa. 6. A expedição envia os pedidos aos clientes. 7. O cliente rastreia seus pedidos, desde o momento de suas gerações. Em relação ao fluxo geral do relacionamento com o cliente: 1. Os consumidores registram os seus gostos. 2. Os clientes registram suas queixas. 3. O armazém realiza campanhas promocionais. Em relação ao fluxo geral da contabilidade: 1. A contabilidade gera as notas fiscais. 2. A contabilidade gera a ordem de cobrança 3. A contabilidade gera e envia a fatura ao cliente. 4. A contabilidade gerencia as contas a receber. 5. A contabilidade gerencia as contas a pagar. 6. A contabilidade realiza pagamentos a fornecedores (e, eventualmente, a clientes). 7. A contabilidade compõe os custos administrativos e fiscais do armazém. Em relação ao fluxo geral das estratégias para a melhoria do armazém: 1. O armazém analisa as queixas dos clientes, acompanhando-as e solucionando-as. 2. O armazém avalia os clientes. 3. O armazém avalia os fornecedores. 4. Os usuários do sistema elaboram consultas

	de acordo com suas necessidades.

Fonte: produção do próprio autor

A Tabela 35 apresenta o resultado da aplicação da Determinação Hermenêutica de Requisitos no projeto de Acompanhamento de Inventário, para constituir os requisitos de software.

Tabela 35 - Acompanhamento de Inventário: Descrições dos Requisitos de Software

Determinação Hermenêutica de Requisitos Requisitos de Software (RS)	
RS-1: Adicionar Produto ao Catálogo de Produtos	Os estoquistas adicionam produtos ao Catálogo de Produtos, informando suas características gerais e os indicando como ativados ou desativados.
RS-2: Ativar Produto do Catálogo de Produtos	Os estoquistas indicam como "ativados" alguns produtos do Catálogo de Produtos. A partir desta ação, esses produtos serão visualizados por todos que têm acesso ao Catálogo de Produtos. Produtos também podem ser agendados para ficar ativos entre um intervalo de datas.
RS-3: Desativar Produto do Catálogo de Produtos	Os estoquistas indicam como "desativados" alguns produtos do Catálogo de Produtos. A partir desta ação, esses produtos não serão visualizados no Catálogo de Produtos. Produtos serão desativados automaticamente, caso não haja mais itens deles no estoque.
RS-4: Atualizar Produto do Catálogo de Produtos	Os estoquistas modificam algumas informações pertinentes às características dos produtos, conforme necessidades.
RS-5: Buscar Produto no Catálogo de Produtos	Todos os usuários do sistema poderão pesquisar produtos no Catálogo de Produtos, informando uma ou mais características do produto. Todos os produtos que possuem as mesmas características informadas serão apresentados como resultado da busca.
RS-6: Gerar Inventário	Os planejadores, os estoquistas (e quem mais tiver autorização) poderão, a qualquer momento, gerar o inventário atualizado, onde será indicada a situação atual de cada item estocado.
RS-7: Rastrear Pedidos (de clientes)	Os planejadores, os agentes de pedidos (e quem mais tiver autorização) rastreiam pedidos, podendo ser por cliente, por data, por região, por produto, ou qualquer outra informação e/ou combinação de

	informações.
RS-8: Analisar Tendências de Vendas	Os planejadores (e quem mais tiver autorização) analisam as tendências de vendas, podendo ser por cliente, por data, por região, por produto, ou qualquer outra informação e/ou combinação de informações. As tendências de vendas são analisadas fazendo projeções futuras sobre os históricos das vendas.
RS-9: Analisar os Gostos dos Consumidores	Os planejadores (e quem mais tiver autorização) analisam as sugestões, preferências e listas de desejos indicadas pelos consumidores (clientes ou não).
RS-10: Comprar (item de) Produto	Os agentes de compras dão entrada dos itens no estoque, indicando fornecedores, lotes e quantidades.
RS-11: Gerar Pedido de Compra	O cliente indica quais produtos do Catálogo de Produtos ele quer comprar. Ele indica também as quantidades de cada produto, além das informações de cobrança. Ao finalizar, o sistema calcula e indica ao cliente a data que seu pedido será entregue pela expedição. Se o cliente for um cliente novo, ele deverá, antes de fazer o seu pedido, cadastrar-se no sistema.
RS-12: Indicar Localização do Produto	Os estoquistas verificam os endereços dos produtos dentro dos estoques.
RS-13: Gerar Pedido de Embalagem	A aplicação envia aos estoquistas mais próximos aos produtos as ordens de preparação de embalagens.
RS-14: Gerar Ordem de Expedição	O sistema envia aos agentes de transportes as ordens de expedição para os pedidos de compras.
RS-15: Gerar Rótulo de Remessa	O sistema gera rótulos de remessas, possibilitando a navegação dos pedidos de compras para os clientes. Também será possível navegar de volta aos clientes, a partir dos pedidos de embalagens.
RS-16: Enviar Pedido ao Cliente	Os agentes de transportes indicam as entregas dos pedidos de compras aos clientes.
RS-17: Acompanhar Pedido de Compra (cliente verifica seus pedidos)	O cliente, a qualquer momento, verifica a situação atual de um determinado pedido de compra ou de um grupo de pedidos de compras que estão em seu nome.
RS-18: Registrar Gostos dos Consumidores	Clientes e não clientes registram sugestões, preferências e listas de desejos acerca de produtos que pretendem comprar.
RS-19: Registrar Queixa	Clientes podem, a qualquer momento, registrar queixas de quaisquer naturezas, indicando fatos que vivenciou de maneira insatisfatória.
RS-20: Criar Campanha Promocional	Os planejadores registram as campanhas promocionais, nomeando-as e indicando seu objetivo, suas regras e seu intervalo de validade. Ao registrar uma campanha promocional, a mesma

	estará indicada como “desativada”.
RS-21: Ativar Campanha Promocional	Os planejadores ativam determinadas Campanhas Promocionais. Nesse momento, confirmam os seus intervalos de validade. Estando nesse intervalo de datas, a Campanha Promocional será visualizada por todos os usuários do sistema.
RS-22: Desativar Campanha Promocional	Os planejadores desativam determinadas Campanhas Promocionais. Automaticamente, nenhum usuário do sistema as visualizará, exceto os planejadores.
RS-23: Gerar Nota Fiscal	Os contadores geram as notas fiscais de vendas e de expedições, de acordo com os pedidos de compras que são gerados pelos clientes.
RS-24: Gerar Ordem de Cobrança	Os contadores geram as ordens de cobranças referentes aos pedidos de compras feitos pelos clientes.
RS-25: Enviar Fatura Para Cliente	Os contadores geram e enviam as faturas para os clientes, quando estes, durante as gerações de seus pedidos, indicaram a forma de pagamento “boleto”.
RS-26: Gerar Conta a Receber	Os contadores registram as contas a receber, indicando as contas, os devedores, os valores e as datas para recebimento.
RS-27: Gerar Conta a Pagar	Os contadores registram as contas a pagar, indicando as contas, os credores, os valores e as datas para pagamento.
RS-28: Pagar Fornecedor	Os contadores efetuam os pagamentos devidos aos fornecedores de produtos. Essa ação tira automaticamente as referidas contas da situação “conta a pagar”.
RS-29: Compor Custos Administrativos e Fiscais	Os contadores efetuam a composição dos custos administrativos e fiscais para os produtos do armazém.
RS-30: Analisar Queixas	Os planejadores (e quem mais tiver autorização) analisam as queixas registradas pelos clientes e as direciona para os profissionais do armazém resolvê-las, de acordo com as características de cada queixa.
RS-31: Resolver Queixas	Os profissionais responsáveis por resolverem as queixas, as solucionam e registram as soluções dadas a elas.
RS-32: Acompanhar Queixas	Os planejadores acompanham as situações das queixas em aberto, podendo redirecioná-las para outros profissionais do armazém ou indica-las como insolúveis no momento. Em acompanhamentos futuros, essas queixas indicadas como insolúveis poderão ser ativadas novamente, direcionando-as aos profissionais do armazém solucioná-las.
RS-33: Avaliar Cliente	Os planejadores (e quem mais tiver autorização) registram as avaliações dos clientes, podendo ser problemáticos, regulares ou bons.
RS-34: Avaliar Fornecedor	Os planejadores (e quem mais tiver autorização)

	registram avaliações dos fornecedores, podendo ser problemáticos, regulares ou bons.
RS-35: Gerar Consulta Personalizada	Os usuários do sistema, conforme suas necessidades específicas podem gerar consultas de acordo com as informações a que tem acesso. Essas consultas geradas ficam gravadas no sistema para posterior utilização.

Fonte: produção do próprio autor

Acompanhamento de Inventário: Aplicação do Teodolito Hermenêutico de Requisitos

Devido a sua complexidade, os requisitos do sistema de acompanhamento de inventário necessitam de diversas iterações para serem definidos e compostos satisfatoriamente. Sendo assim, com intuito de demonstração, aqui serão apresentados quatro momentos não sequenciais das aplicações do Teodolito Hermenêutico de Requisitos para o desenvolvimento deste sistema, chamados respectivamente de ciclo 1, ciclo 2, ciclo 3 e ciclo 4, onde o ciclo 1 refere-se ao início do projeto e os demais ciclos se encontram em estágios mais avançados do mesmo.

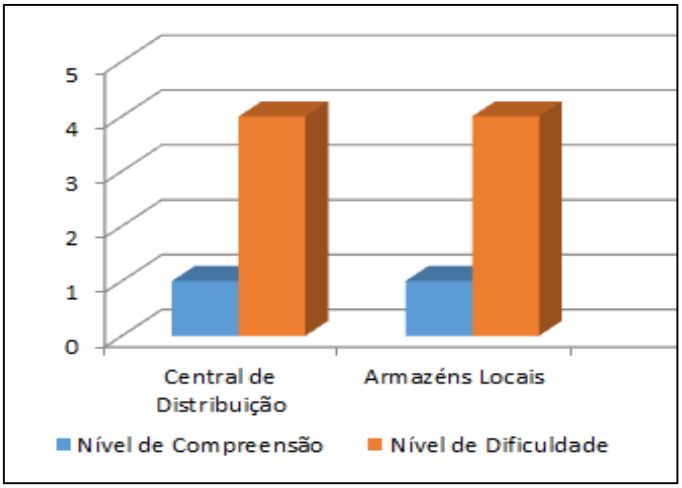
Ciclo 1: durante esse ciclo, foram identificadas as áreas de negócio Central de Distribuição e Armazéns Locais, mas as mesmas não foram detalhadas a ponto de ser possível identificar os requisitos de software. Sendo assim, ao aplicar o Teodolito Hermenêutico de Requisitos, foram obtidos os seguintes resultados e leituras:

Tabela 36 - Acompanhamento de Inventário: ciclo 1 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
Central de Distribuição	1 – Pré-conceitual
Armazéns Locais	1 – Pré-conceitual

Fonte: produção do próprio autor

Figura 30 - Inventário: ciclo 1 - leituras do domínio da aplicação



Fonte: próprio autor

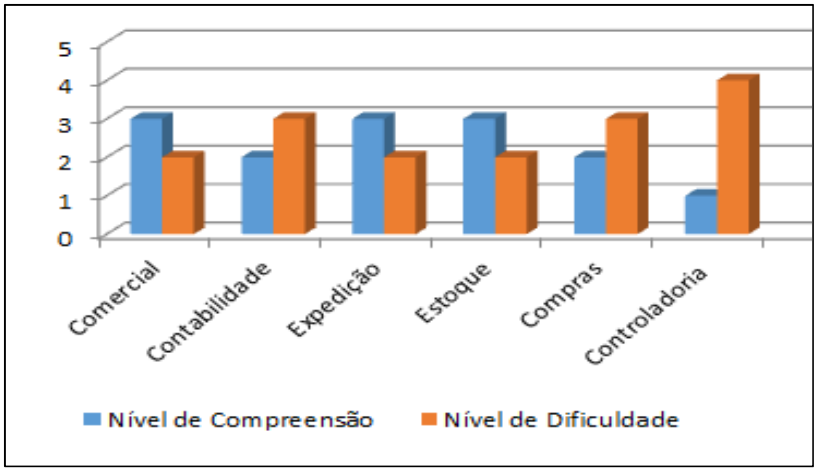
Ciclo 2: durante esse ciclo, as áreas de negócio Central de Distribuição e Armazéns Locais foram melhor compreendidas, possibilitando a identificação e produção de alguns requisitos de software. Com isso, ao aplicar o Teodolito Hermenêutico de Requisitos, foram obtidos os seguintes resultados e leituras:·

Tabela 37 - Acompanhamento de Inventário: ciclo 2 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
Comercial	3 – Contextual
Contabilidade	2 – Conceitual
Expedição	3 – Contextual
Estoque	3 – Contextual
Compras	2 – Conceitual
Controladoria	1 – Pré-conceitual

Fonte: produção do próprio autor

Figura 31 - Inventário: ciclo 2 - leituras do domínio da aplicação



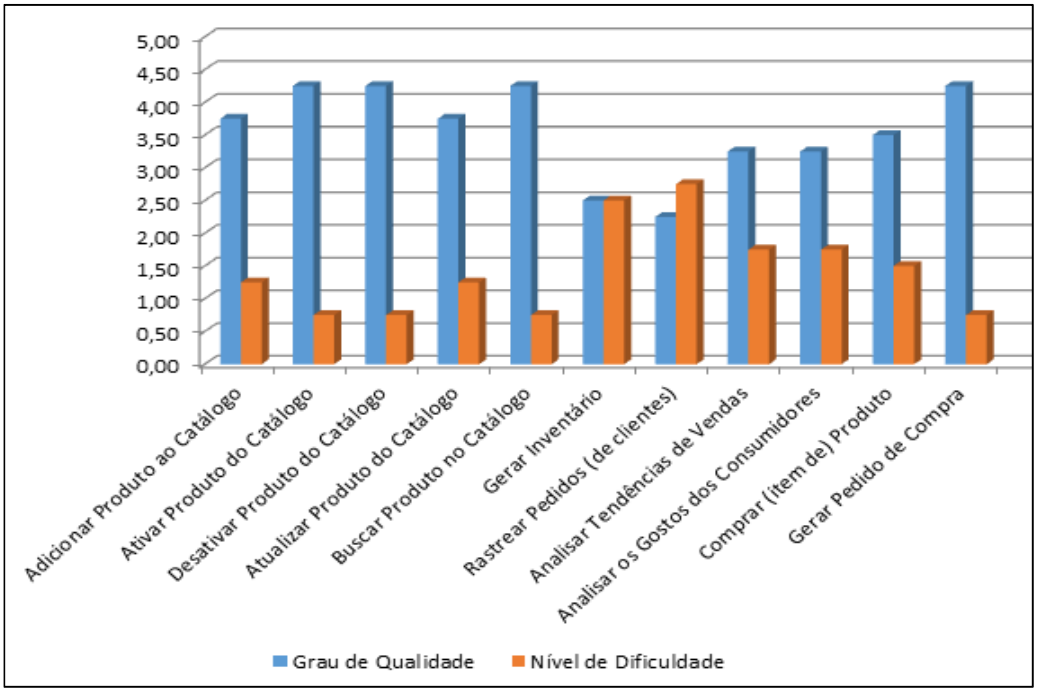
Fonte: produção do próprio autor

Tabela 38 - Acompanhamento de Inventário: ciclo 2 - Requisitos de Software

Requisito de Software	Grau de Qualidade
Adicionar Produto ao Catálogo	4.3 - O requisito foi composto de maneira suficiente.
Ativar Produto do Catálogo	5.1 - O requisito está satisfatoriamente completo e consistente.
Desativar Produto do Catálogo	5.1 - O requisito está satisfatoriamente completo e consistente.
Atualizar Produto do Catálogo	4.3 - O requisito foi composto de maneira suficiente.
Buscar Produto no Catálogo	5.1 - O requisito está satisfatoriamente completo e consistente.
Gerar Inventário	3.2 - O requisito está consistente em relação às expectativas das partes interessadas.
Rastrear Pedidos (de clientes)	3.1 - O requisito está claro em relação ao seu escopo.
Analisar Tendências de Vendas	4.1 - O requisito está em conformidade com os padrões exigidos.
Analisar os Gostos dos Consumidores	4.1 - O requisito está em conformidade com os padrões exigidos.
Comprar (item de) Produto	4.2 - O conjunto de itens do requisito fornece um claro valor para as partes interessadas.
Gerar Pedido de Compra	5.1 - O requisito está satisfatoriamente completo e consistente.

Fonte: produção do próprio autor

Figura 32 Inventário: ciclo 2 – Graus de Qualidade dos Requisitos de Software



Fonte: produção do próprio autor

Ciclo 3: durante esse ciclo, a compreensão das áreas de negócio evoluiu, possibilitando a identificação e produção de outros requisitos de software. Dessa forma, ao aplicar o Teodolito Hermenêutico de Requisitos, foram obtidos os seguintes resultados e leituras:

Tabela 39 - Acompanhamento de Inventário: ciclo 3 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
Comercial	4 – Sistêmico
Contabilidade	3 – Contextual
Expedição	4 – Sistêmico
Estoque	4 – Sistêmico
Compras	3 – Contextual
Controladoria	2 – Conceitual

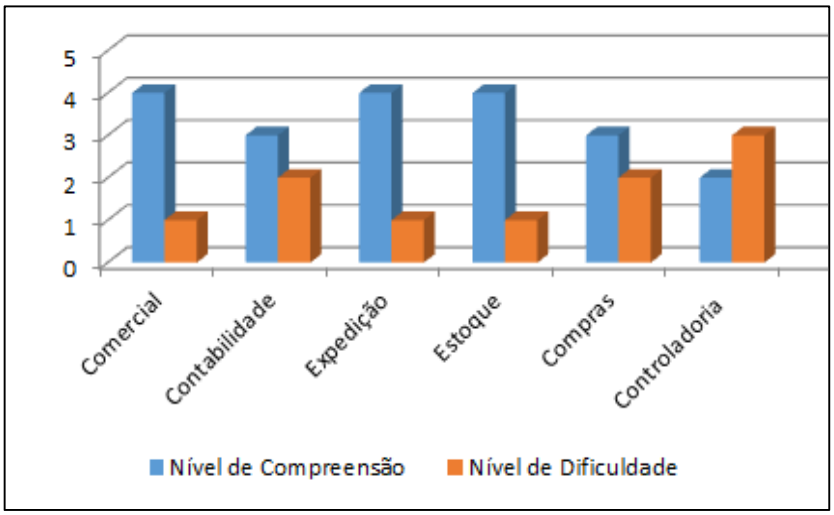
Fonte: produção do próprio autor

Tabela 40 - Acompanhamento de Inventário: ciclo 3 - Requisitos de Software

Requisito de Software	Grau de Qualidade
Indicar Localização do Produto	4.4 - É possível testar e avaliar o requisito.
Gerar Pedido de Embalagem	3.4 - A alocação do requisito foi feita.
Gerar Ordem de Expedição	3.4 - A alocação do requisito foi feita.
Gerar Rótulo de Remessa	3.2 - O requisito está consistente em relação às expectativas das partes interessadas.
Enviar Pedido ao Cliente	4.4 - É possível testar e avaliar o requisito.
Acompanhar Pedido de Compra	5.1 - O requisito está satisfatoriamente completo e consistente.
Registrar os Gostos dos Consumidores	5.2 - O requisito não possui omissões e nem ambiguidades.
Registrar Queixas	5.2 - O requisito não possui omissões e nem ambiguidades.
Realizar Campanha Promocional	3.4 - A alocação do requisito foi feita.
Avaliar Campanha Promocional	4.2 - O conjunto de itens do requisito fornece um claro valor para as partes interessadas.
Desativar Campanha Promocional	4.2 - O conjunto de itens do requisito fornece um claro valor para as partes interessadas.
Gerar Nota Fiscal	4.1 - O requisito está em conformidade com os padrões exigidos.

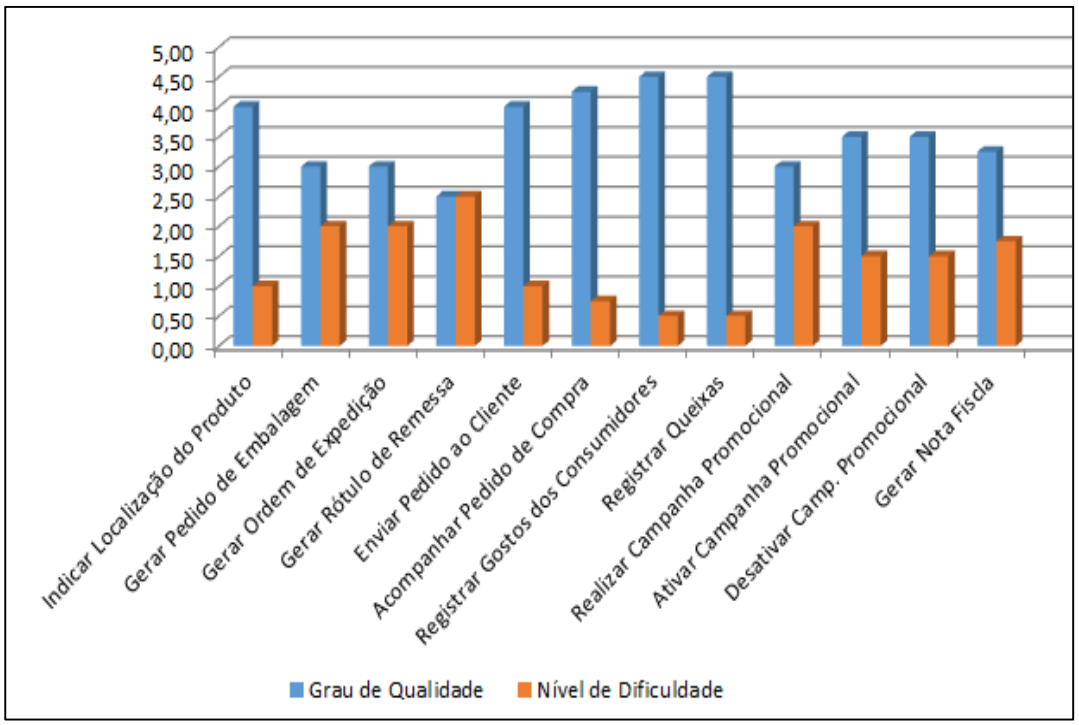
Fonte: produção do próprio autor

Figura 33 - Inventário: ciclo 3 - leituras do domínio da aplicação



Fonte: produção do próprio autor

Figura 34 - Inventário: ciclo 3 – Graus de Qualidade dos Requisitos de Software



Fonte: produção do próprio autor

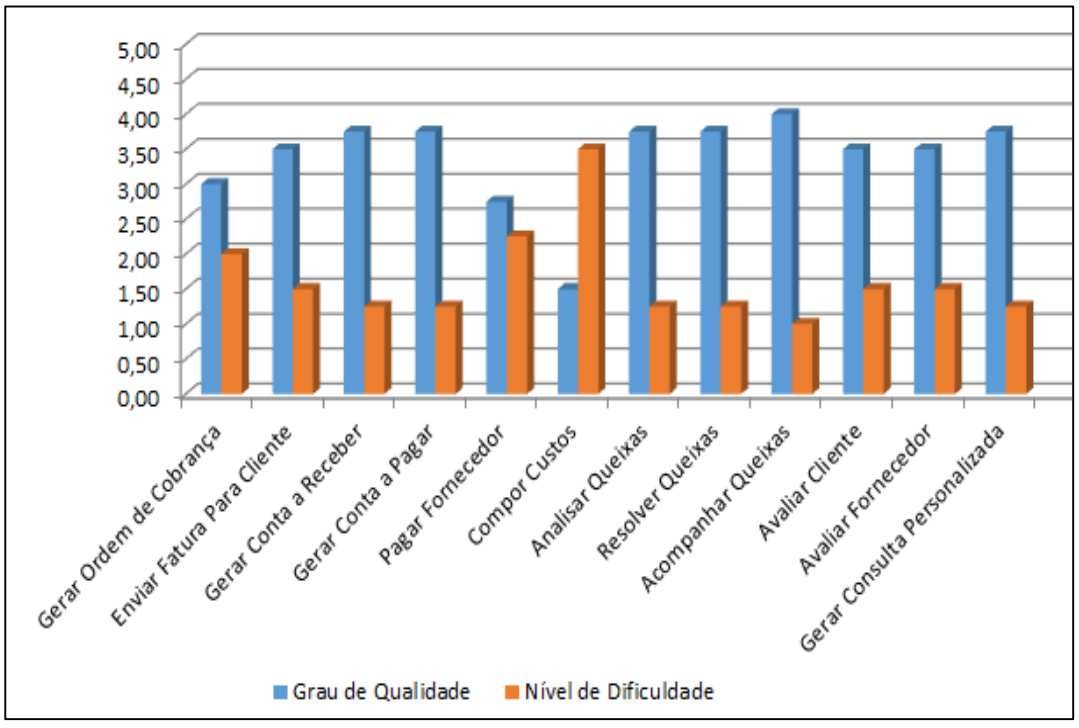
Ciclo 4: durante esse ciclo, ainda com o mesmo nível de compreensão das áreas de negócio alcançado no ciclo 3, foi possível identificar e produzir outros requisitos de software. Dessa forma, ao aplicar o Teodolito Hermenêutico de Requisitos, foram obtidos os seguintes resultados e leituras:

Tabela 41 - Acompanhamento de Inventário: ciclo 4 - Requisitos de Software

Requisito de Software	Grau de Qualidade
Gerar Ordem de Cobrança	3.4 - A alocação do requisito foi feita.
Enviar Fatura Para Cliente	4.2 - O conjunto de itens do requisito fornece um claro valor para as partes interessadas.
Gerar Conta a Receber	4.3 - O requisito foi composto de maneira suficiente.
Gerar Conta a Pagar	4.3 - O requisito foi composto de maneira suficiente.
Pagar Fornecedor	3.3 - As partes interessadas aceitam que o requisito captura com precisão o que o ele faz e não faz.
Compor Custos Administrativos e Fiscais	2.2 - O requisito foi priorizado.
Analisar Queixas	4.3 - O requisito foi composto de maneira suficiente.
Resolver Queixas	4.3 - O requisito foi composto de maneira suficiente.
Acompanhar Queixas	4.4 - É possível testar e avaliar o requisito.
Avaliar Cliente	4.2 - O requisito é claro em relação ao seu escopo.
Avaliar Fornecedor	4.2 - O requisito é claro em relação ao seu escopo.
Gerar Consulta Personalizada	4.3 - O requisito foi composto de maneira suficiente.

Fonte: produção do próprio autor

Figura 35 - Inventário: ciclo 4 – Graus de Qualidade dos Requisitos de Software



Fonte: produção do próprio autor

Aplicação "Inteligência Artificial: Criptoanálise"

Visão Geral da Aplicação

O objetivo deste projeto é automatizar o processo de decodificação de criptogramas, o qual é considerado um desafio muito complexo até mesmo para as técnicas mais sofisticadas.

> Our problem is one of cryptanalysis, the process of transforming cipher text back to plain text. In its most general form, deciphering cryptograms is an intractable problem that defies even the most sophisticated techniques. Happily, our problem is relatively simple because we limit ourselves to single substitution ciphers. (BOOCH, 2007, p. 414).

A criptografia tem por objetivo tornar ininteligíveis as mensagens dos textos que só podem ser compreendidas por quem estiver autorizado a compreendê-las. Tais textos são decodificados por algoritmos criptográficos, que os transformam em suas formas simples empregando, por exemplo, a "cifra de substituição". Essa técnica mapeia cada letra do texto por uma letra diferente. Por exemplo, o texto "este exemplo foi simples de entender" foi cifrado por "ftuf fyfnqmp gpj tjnqmft ef foufoefs", onde cada letra do texto foi substituída por sua subsequente letra do alfabeto. Desse modo, basta aplicar o inverso desta regra, que o texto retornará à sua forma simples.

Mas há outra forma de utilizar a cifra de substituição para tornar a decodificação do texto mais complexa. Por exemplo, substitui-se a letra "A" pela letra "D", a letra "B" pela letra "H", a letra "C" pela letra "L", e assim por diante. Desse modo, o texto "esse exemplo não foi simples de entender" é cifrado por "txxt trtzlvh ddh xhj xjzlvtx pt tdbtdptt". Neste caso, se não for conhecida a regra de mapeamento utilizada para codificar cada letra, sua decodificação fica muito difícil de ser feita, e não faz sentido tentar substituí-las por tentativa e erro, pois, considerando o alfabeto português, que possui 26 letras, existem aproximadamente 4.03×10^{26} combinações possíveis

> Trying an exhaustive search is pretty much senseless. Assuming that the plaintext alphabet encompasses only the 26 uppercase English characters, there are 26 (approximately 4.03 x1 0^{26}) possible combinations. (BOOCH, 2007, p. 416).

Outro fator que impacta nessa decodificação é o conhecimento que se tem do idioma utilizado tanto para codificar como para decodificar o texto. No exemplo apresentado, foi usado para ambos os casos o idioma português. Sendo assim, vemos que, por exemplo, a terceira palavra do texto codificado, "ddh" inicia com duas letras iguais

“dd”. Portanto, sabendo que em português não existe palavra de três letras que inicie com duas letras iguais, logo, percebe-se que cada “d” está codificando uma letra diferente do alfabeto português. O mesmo ocorre, por exemplo, com a primeira e quarta palavras: “txxt” e “xhj”, onde a letra “x” está codificando duas letras diferentes do alfabeto português. Portanto, além do mapeamento, deve-se utilizar o conhecimento que se tem sobre a estrutura e as regras do idioma para o qual o texto será decodificado.

> [...]Thus, we must try something other than a brute force attack. An alternate technique is to make an assumption based on our knowledge of sentence, word, and letter structure and then follow this assumption to its natural conclusions. Once we can go no further, we choose the next most promising assumption that builds on the first one, and so on, as long as each succeeding assumption brings us closer to a solution. If we find that we are stuck, or we reach a conclusion that contradicts a previous one, we must backtrack and alter an earlier assumption. [...] (BOOCH, 2007, p. 416).

Sendo assim, compreendemos que decifrar criptogramas é uma tarefa bem complexa, que não pode ser resolvida por meio de soluções algorítmicas triviais. Em relação ao nosso problema específico, ele vai ao encontro de transformar criptogramas de volta ao original, assumindo que apenas uma simples cifra de substituição foi empregada para sua codificação.

> It is a vastly simplifying assumption to know that only a substitution cipher was employed to encode a plaintext message; nevertheless, deciphering the resulting cryptogram is not an algorithmically trivial task. (BOOCH, 2007, p. 415).

A título de exemplo de como solucionar este problema, o livro apresentou o seguinte cenário (em inglês):

> Q AZWS DSSC KAS DXZNN DASNN. (BOOCH, 2007, p. 416).

A dica dada para a solução desse problema é que a letra “W” representa a letra “V”:

> As a hint, we note that the letter W represents the plaintext V. (BOOCH, 2007, p. 416).

A solução a este criptograma se deu da seguinte maneira:

1. De acordo com a sugestão, podemos substituir diretamente a letra “W” pela letra “V”.

   > Q AZVS DSSC KAS DXZNN DASNN (BOOCH, 2007, p. 416).

2. A primeira palavra é pequena (“Q”), então é bem provável que ela seja um “A” ou um “I”; vamos supor que é um “A”.

A AZVS DSSC KAS DXZNN DASNN (BOOCH, 2007, p. 416).

3. A terceira palavra precisa de vogal, e é provável que sejam vogais duplas. Provavelmente não é nem “II” nem “UU”, e não pode ser “AA” porque já usamos um “A”. Assim, podemos tentar “EE".

A AZVE DEEC KAE DXZNN DAENN (BOOCH, 2007, p. 416).

4. A quarta palavra é de três letras e termina em um “E”; é provável que seja a palavra “THE”.

A HZVE DEEC THE DXZNN DHENN (BOOCH, 2007, p. 416).

5. A segunda palavra precisa de uma vogal, mas apenas de uma, “I”, “O” ou “U” (já usamos “A” e “E”). Apenas o “I” dá sentido à palavra.

A HIVE DEEC THE DXINN DHENN (BOOCH, 2007, p. 416).

6. Existem poucas palavras de quatro letras que têm um “E” duplo, incluindo “DEER”, “BEER” e “SEEN”. Nosso conhecimento de gramática sugere que a terceira palavra deva ser um verbo; então nós selecionamos “SEEN”.

A HIVE SEEN THE SXINN SHENN (BOOCH, 2007, p. 416).

7. Esta frase não faz nenhum sentido, por isso, provavelmente, fizemos um mau presságio em algum lugar ao longo do caminho. O problema parece estar com a vogal na segunda palavra, então podemos considerar a reversão de nossa suposição inicial.

I HAVE SEEN THE SXANN SHENN (BOOCH, 2007, p. 417).

8. Vamos resolver a última palavra. As letras duplas não podem ser “SS” (usamos “S” e, além disso, “SHESS” não faz sentido), mas “LL” forma uma palavra significativa.

I HAVE SEEN THE SXALL SHELL (BOOCH, 2007, p. 417).

9. A quinta palavra é a parte de uma frase nominal e, portanto, é provavelmente um adjetivo (“STALL”, por exemplo, é rejeitado nessa frase). Procurando por palavras que se encaixam no padrão “S?ALL”, encontra-se a palavra “SMALL”.

I HAVE SEEN THE SMALL SHELL (BOOCH, 2007, p. 417).

Na sequência, foram feitas as seguintes observações acerca do processo utilizado para a solução deste problema:

- Aplicamos muitas fontes diferentes de conhecimento, como reconhecimento de gramática, ortografia e vogais.
- Registramos nossas premissas em um lugar central e aplicamos nossas fontes de conhecimento a essas suposições para raciocinar sobre suas consequências.
- Nós raciocinamos oportunamente. Às vezes, raciocinamos de regras genéricas a regras específicas (se a palavra tem três letras e finaliza em "E", é bem provável que seja "THE"), e em outras vezes, raciocinamos do específico para o geral ("?EE?" pode ser "DEER", "BEER" ou "SEEN", mas como a palavra deve ser um verbo e não um substantivo, somente o "SEEN" satisfaz nossa hipótese).

A partir dessas observações, foram identificadas três abstrações importantes: múltiplas fontes de conhecimento, um lugar central para suposições ou hipóteses e um componente de controle da resolução de problemas.

> From these problem-solving observations, we can identify some key abstractions. Key abstractions are analysis elements of our solution that begin to establish the initial architectural framework. The three bullets identify multiple knowledge sources, a central place for assumptions or hypotheses, and a control component that opportunistically controls the problem solving. (BOOCH, 2007, p. 417).

Após essas observações, foi explicada a abordagem de solução de problemas que foi aplicada no cenário apresentado. Tal abordagem é conhecida como *Blackboard Model* (Modelo Quadro-Negro), que provou ser eficaz em domínios relacionados à representação do conhecimento declarativo (exemplos: reconhecimento de fala, interpretação de sinais, modelagem de estruturas moleculares tridimensionais, compreensão de imagens e planejamento):

> What we have described is a problem-solving approach known as a blackboard model. The blackboard model was first proposed by Newell in 1962 and later incorporated by Reddy and Erman into the Hearsay and Hearsay II projects, both of which dealt with the problems of speech recognition. The blackboard model proved to be useful in this domain, and the framework was soon applied successfully to other domains, including signal interpretation, the modeling of three-dimensional molecular structures, image understanding, and planning. Blackboard frameworks have proven to be particularly noteworthy with regard to the representation of declarative knowledge and are space and time efficient when compared with alternate approaches (BOOCH, 2007, p. 417).

Em seguida, explicou resumidamente que o *Blackboard Model* é um padrão arquitetural que pode ser representado utilizando-se o paradigma Orientado a Objetos para solucionar nosso referido problema:

> The blackboard framework is an architectural pattern that can be appliedasa result of the analysis of our problem-solving algorithm. The framework can be represented in terms of classes and mechanisms that describe how instances of those classes collaborate. (BOOCH, 2007, p. 418).

E de maneira mais detalhada, explicou-se mais sobre o *Blackboard Model*, fazendo uma analogia com pessoas reunidas em uma sala, cada uma segurando peças de quebra-cabeça. Inicialmente, algumas pessoas, voluntariamente, colocam no quadro-negro algumas peças e, a partir daí, as demais pessoas avaliam suas peças e verificam quais delas podem ajudar a completar o quebra-cabeça, atualizando o quadro-negro com tais peças, fazendo com que a solução do problema evolua, até que o resultado final seja alcançado:

> Imagine a room with a large blackboard and around it a group of people each holding over-size jigsaw pieces. We start with volunteers who put on the blackboard (assume it's sticky) their most "promising" pieces. Each member of the group looks at his pieces and sees if any of them fit into the pieces already on the blackboard. Those with the appropriate pieces go up to the blackboard and update the evolving solution. The new updates cause other pieces to fall into place, and other people go to the blackboard to add their pieces. It does not matter whether one person holds more pieces than another. The whole puzzle can be solved in complete silence; that is, there need be no direct communication among the group. Each person is self-activating, knowing when his pieces will contribute to the solution. No a priori established order exists for people to go up to the blackboard. The apparent cooperative behavior is mediated by the state of the solution on the blackboard. If one watches the task being performed, the solution is built incrementally (one piece at a time) and opportunistically (as an opportunity for adding a piece arises), as opposed to starting, say, systematically from the left top corner and trying each piece. (BOOCH, 2007, p. 418).

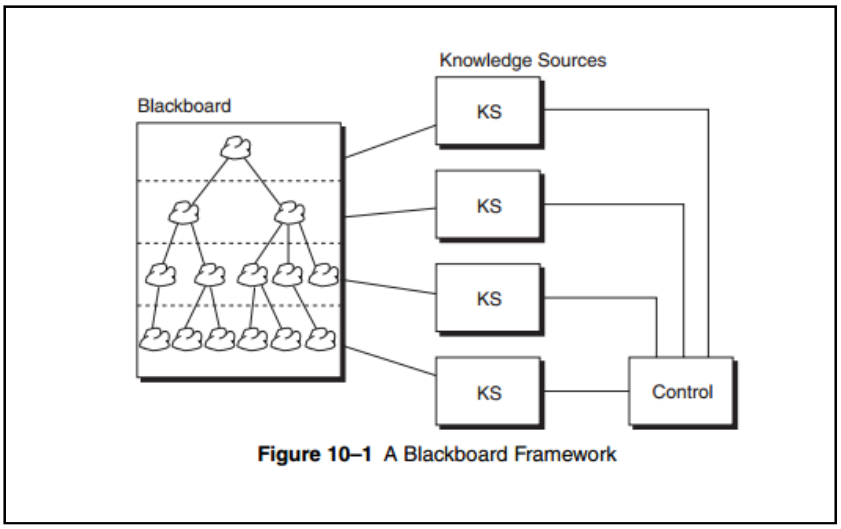
Na sequência, citou-se uma explicação dada pelos próprios elaboradores deste conceito, os quais disseram que a resolução do problema ocorre por meio de fontes de conhecimento separadas e independentes que contribuem com informações referentes aos seus domínios específicos para a solução do problema, de acordo com as atuais informações apresentadas no quadro-negro. Essas fontes de conhecimento são representadas por procedimentos, conjuntos de regras ou asserções lógicas específicas de seus domínios. Além das fontes de conhecimento, há também o controlador que funciona como uma espécie de moderador para a solução do problema, ou seja, quando a fonte do conhecimento encontra uma provável contribuição a fazer para a solução do problema, ela avisa ao controlador e este autoriza ou não a sua participação (similarmente a alguém

levantar a mão para solicitar permissão para sua participação e o moderador escolher aquele que provavelmente possui a melhor contribuição a fazer):

> As Englemore and Morgan explain, "The domain knowledge needed to solve a problem is partitioned into knowledge sources that are kept separate and independent. The objective of each knowledge source is to contribute information that will lead to a solution to the problem. A knowledge source takes a set of current information on the blackboard and updates it as encoded in its specialized knowledge. The knowledge sources are represented as procedures, sets of rules, or logic assertions". Knowledge sources, or KSs for short, are domain-specific. In speech recognition systems, knowledge sources might include agents that can reason about phonemes, morphemes, words, and sentences. In image recognition systems, knowledge sources would include agents that know about simple picture elements, such as edges and regions of similar texture, as well as higher-level abstractions representing the objects of interest in each scene, such as houses, roads, fields, cars, and people. Generally speaking, knowledge sources parallel the hierarchical structure of objects on the blackboard. Furthermore, each knowledge source uses objects at one level as its input and then generates and / or modifies objects at another level as its output. For instance, in a speech recognition system, a knowledge source that embodies knowledge about words might look at a stream of phonemes (at a low level of abstraction) to form a new word (at a higher level of abstraction). Alternately, a knowledge source that embodies knowledge about sentence structure might hypothesize the need for a verb (at a high level of abstraction); by filtering a list of possible words (at a lower level of abstraction), this knowledge source can verify the hypothesis. These two approaches to reasoning represent forward chaining and backward chaining, respectively. Forward chaining involves reasoning from specific assertions to a general assertion, and backward chaining starts with a hypothesis, then tries to verify the hypothesis from existing assertions. This is why we say that control in the blackboard model is opportunistic: Depending on the circumstances, a knowledge source might be selected for activation that uses either forward- or backward-chaining. Knowledge sources usually embody two elements, namely, preconditions and actions. The preconditions of a knowledge source represent the state of the blackboard in which the knowledge source shows an interest. For example, a precondition for a knowledge source in an image recognition system might be the discovery of a relatively linear region of picture elements (perhaps representing a road). Triggering a precondition causes the knowledge source to focus its attention on this part of the blackboard and then take action by processing its rules or procedural knowledge. Under these circumstances, polling is unnecessary: When a knowledge source thinks it has something interesting to contribute, it notifies the blackboard controller. Figuratively speaking, it is as if each knowledge source raises its hand to indicate that it has something useful to do; then, from among eager knowledge sources, the controller calls on the one that looks the most promising. (BOOCH, 2007, p. 418-420).

Para ilustrar esta explanação, foi apresentada a figura 36:

Figura 36 – *Blackboard Model*



Fonte: BOOCH, 2007, p. 419

A partir deste ponto, voltou-se ao problema específico (transformar criptogramas de volta ao original, assumindo que apenas uma simples cifra de substituição foi empregada para sua codificação) para identificar as fontes de conhecimento necessárias para a sua solução, onde foram encontradas treze relevantes:

- Prefixos comuns: Início de palavras comuns, como "re", "anti" e "un".
- Sufixos comuns: terminações de palavras comuns, como "ly", "ing", "es" e "ed".
- Consoantes: palavras sem vogal.
- Substituição Direta: Dicas dadas como parte da declaração do problema.
- Letras duplas: letras duplas comuns, como "tt", "ll" e "ss".
- Frequência de letras: Probabilidade do aparecimento de cada letra.
- *Strings* legais: combinações legais e ilegais de letras, como "qu" e "zg", respectivamente.
- Correspondência de padrões: palavras que correspondem a um padrão especificado de letras.
- Estrutura da frase: Gramática, incluindo os significados das frases nominais e verbais.
- Palavras pequenas: Correspondências possíveis para palavras de uma, duas, três e quatro letras.

- Resolvido: se o problema foi resolvido ou não, ou se nenhum progresso adicional pode ser feito.
- Vogais: palavras não concordantes.
- Estrutura da palavra: a localização das vogais e a estrutura comum de substantivos, verbos, adjetivos, advérbios, artigos, conjuntivas e assim por diante.

> Let's return to our specific problem and consider the knowledge sources that can contribute to a solution. As is typical with most knowledge-engineering applications, the best strategy is to sit down with an expert in the domain and record the heuristics that this person applies to solve the problems in the domain. For our present problem, this might involve trying to solve a number of cryptograms and recording our thinking process along the way. (BOOCH, 2007, p. 420).
>
> Our analysis suggests that 13 knowledge sources are relevant; they appear with the knowledge they embody in the following list: (BOOCH, 2007, p. 420).
>
> - Common prefixes: Common word beginnings such as "re", "anti", and "un".
> - Common suffixes: Common word endings such as "ly", "ing", "es", and "ed".
> - Consonantes: Nonvowel letters.
> - Direct substitution: Hints given as part of the problem statement.
> - Double letters: Common double letters, such as "tt","ll", and "ss".
> - Letter frequency: Probability of the appearance of each letter.
> - Legal strings: Legal and illegal combinations of letters, such as "qu" and "zg", respectively.
> - Pattern matching: Words that match a specified pattern of letters.
> - Sentence structure: Grammar, including the meanings of noun and verb phrases.
> - Small words: Possible matches for one-, two-, three-, and four letter words.
> - Solved: Whether or not the problem is solved, or if no further progress can be made.
> - Vowels: Nonconsonant letters.
> - Word structure: The location of vowels and the common structure of nouns, verbs, adjectives, adverbs, articles, conjunctives, and so on.

Por fim, foram dadas duas sugestões arquiteturais para a elaboração do projeto da aplicação. A primeira sugere que cada uma das treze fontes de conhecimento representa uma classe, onde cada uma de suas instâncias incorpora algum conhecimento do domínio, exibe certo comportamento específico e é exclusivamente identificável:

> From an object-oriented perspective, each of these 13 knowledge sources represents a candidate class in our architecture: Each instance embodies some state (its knowledge), each exhibits certain class-specific behavior (a suffix knowledge source can react to words suspected of having a common ending), and each is uniquely identifiable (a small-word knowledge source exists independent of the pattern-matching knowledge source). (BOOCH, 2007, p. 421).

Já a segunda sugestão arquitetural organiza as fontes de conhecimento de maneira hierárquica, onde algumas delas operam em frases, outras em palavras, outras em sequências de letras e outras em letras individuais:

> We may also arrange these knowledge sources in a hierarchy. Specifically, some knowledge sources operate on sentences, others on letters, still others on contiguous groups of letters, and the lowest-level ones on individual letters. Indeed, this hierarchy reflects the objects that may appear on the blackboard: sentences, words, strings of letters, and letters. (BOOCH, 2007, p. 421).

Assim, encerrou-se a seção que trata a Concepção da aplicação e iniciou-se a que aborda a sua correspondente Elaboração de projeto.

Criptoanálise: Aplicação da Elicitação Hermenêutica de Requisitos

O resultado da aplicação da Elicitação Hermenêutica de Requisitos no projeto de Criptoanálise é apresentado nas tabelas a seguir.

A Tabela 42 apresenta o resultado da aplicação da Identificação de Diferença Situacional no projeto de Criptoanálise.

Tabela 42 - Criptoanálise: Identificação de Diferença Situacional

Identificação de Diferença Situacional	
O que fazer?	Criptoanálise: processo de transformar texto cifrado de volta para o texto simples (decodificação de criptograma).
Para quem fazer?	Especialistas em determinada área do conhecimento.
Por que fazer?	Para tornar legíveis as mensagens dos textos que foram cifrados com o intuito de evitar que pessoas não autorizadas compreendam suas mensagens.
Quem são os envolvidos?	Todas as pessoas que estão autorizadas a acessarem e compreenderem as mensagens cifradas por meio de algoritmos criptográficos, que transformaram os textos empregando a técnica conhecida como "cifra de substituição".
Como funciona o processo de decodificação de criptograma?	Para o nosso caso, vamos utilizar o *Blackboard Model*. Este modelo, conforme explicado por seus idealizadores (Englemore e Morgan) se dá da seguinte forma: "O conhecimento de domínio necessário para resolver um problema é particionado em fontes de conhecimento que são mantidas separadas e independentes. O objetivo de cada fonte de conhecimento é contribuir com informações que

	levarão a uma solução para o problema. Uma fonte de conhecimento usa um conjunto de informações atualizadas no quadro-negro e as atualiza de acordo com seu conhecimento especializado. As fontes de conhecimento são representadas por procedimentos, conjuntos de regras ou asserções lógicas". As fontes de conhecimento, ou abreviadamente são específicas do domínio. Nos sistemas de reconhecimento de fala, as fontes de conhecimento podem incluir agentes que podem argumentar sobre fonemas, morfemas, palavras e sentenças. Nos sistemas de reconhecimento de imagem, as fontes de conhecimento deveriam incluir agentes que conheçam elementos simples de imagem, como bordas e regiões de textura semelhante, bem como abstrações de nível mais elevado, representando os objetos de interesse em cada cena, como casas, estradas, campos, carros e pessoas. De um modo geral, as fontes de conhecimento paralelizam a estrutura hierárquica do objeto no quadro-negro. Além disso, cada fonte de conhecimento usa objetos em um nível como entrada e, em seguida, gera e / ou modifica objetos em outro nível, como saída. Por exemplo, em um sistema de reconhecimento de fala, uma fonte de conhecimento que incorpora o conhecimento sobre as palavras pode olhar para um fluxo de fonemas (em um nível baixo de abstração) para formar uma nova palavra (em um nível mais alto de abstração). Alternativamente, uma fonte de conhecimento que incorpora o conhecimento sobre a estrutura da sentença pode fazer hipóteses sobre a necessidade de um verbo (em um alto nível de abstração); filtrando uma lista de possíveis palavras (em um nível inferior de abstração), essa fonte de conhecimento pode verificar a hipótese. Essas duas abordagens ao raciocínio representam encadeamento direto e encadeamento atrasado, respectivamente. O encadeamento direto envolve o enfoque de asserções específicas para uma asserção geral, e o encadeamento atrasado começa com uma hipótese, e depois tenta verificar a hipótese a partir de asserções existentes. Por isso, dizemos que o controle no *Blackboard Model* é oportuno: dependendo das circunstâncias, uma fonte de conhecimento pode ser selecionada para ativação que seja encadeada para frente ou para trás. As fontes de conhecimento geralmente formam

	dois elementos, a saber, pré-condições e ações. As pré-condições de uma fonte de conhecimento representam o estado do quadro-negro em que a fonte de conhecimento mostra um interesse. Por exemplo, uma pré-condição para uma fonte de conhecimento em um sistema de reconhecimento de imagem pode ser a descoberta de uma região relativamente linear de elementos de imagem (talvez representando uma estrada). O desencadeamento de uma pré-condição faz com que a fonte de conhecimento concentre sua atenção nessa parte do quadro-negro e, em seguida, atue processando suas regras ou conhecimento procedural. Nessas circunstâncias, a pesquisa é desnecessária: quando uma fonte de conhecimento acha que tem algo interessante para contribuir, ela examina o controlador do quadro-negro. Figurativamente falando, é como se cada fonte de conhecimento levantasse sua mão para indicar que tem algo útil para fazer; então, dentre as fontes de conhecimento ansiosas, o controlador chama o que parece ser o mais promissor.

Fonte: produção do próprio autor

A Tabela 43 apresenta o resultado da aplicação do Exame de Diferença Situacional no projeto de Criptoanálise.

Tabela 43 - Criptoanálise: Exame de Diferença Situacional

Exame de Diferença Situacional	
Cenário	Transformar texto cifrado de volta para sua forma original.
Natureza	Criptoanálise
Meio-ambiente	Cifra de substituição e idioma utilizado
Ambiente	Sistema Especialista em *Blackboard Model*
Utensílios (ou insumos)	• Pessoa autorizada • Criptograma • Cifra de Substituição • Idioma (inicialmente, Português) • Quadro-Negro • Especialistas no idioma (Fontes de Conhecimento) • Controlador

Fonte: produção do próprio autor

A Tabela 44 apresenta o resultado da aplicação da Determinação Hermenêutica de Requisitos no projeto de Criptoanálise.

Tabela 44 - Criptoanálise: Determinação Hermenêutica de Requisitos

Determinação Hermenêutica de Requisitos	
Necessidades Originais	Conceber um sistema que, dado um criptograma, o transforme de volta ao seu texto original, assumindo que apenas uma cifra simples de substituição foi empregada para codifica-lo.
Expectativas	• Automatizar o processo de decodificação de criptogramas. • Somente pessoas autorizadas poderão decodificar os criptogramas. • O algoritmo utilizado para a decodificação não deve utilizar o recurso de tentativa e erro, mas munir-se de diferentes fontes de conhecimento, como reconhecimento de gramática, ortografia e vocais, para fazer seus raciocínios e tomar as decisões corretas para a decodificação do criptograma. • Projetar a aplicação conforme o *Blackboard Model*.
Especificação Aceitável (descrição)	A pessoa que quiser decodificar um criptograma, após se identificar na aplicação, informa o texto cifrado a ser decodificado e, na sequência, a aplicação o transforma em sua forma original e o apresenta à pessoa autorizada. A aplicação utilizará o *Blackboard Model* em seu algoritmo para realizar a decodificação do criptograma.
Especificação Aceitável (dinâmica)	1. Ao abrir a aplicação, a pessoa deve informar sua permissão de acesso. 2. Se a pessoa não tiver permissão de acesso à aplicação, ela será informada de que seu acesso foi negado. 3. A pessoa tendo permissão de acesso à aplicação, os próximos passos são executados. 4. A pessoa informa o criptograma a ser decodificado. 5. Se a aplicação verificar que o criptograma informado não é válido, ela avisa à pessoa que o criptograma informado não pode ser decodificado e retorna ao passo <4>. Caso contrário (o criptograma informado é válido), o passo <6> é executado. 6. Aplicando algoritmos criptográficos que empregam o *Blackboard Model*, o

	criptograma é transformado em seu texto original. 7. A aplicação exibe o resultado obtido no passo <6>. 8. A pessoa encerra a aplicação ou informa outro criptograma a ser decodificado. 9. Se a pessoa optar por encerrar a aplicação, a mesma será fechada e seu uso desautorizado, até que o passo <1> seja realizado novamente. 10. Se a pessoa optar por informar outro criptograma, os passos <4> em diante serão realizados novamente. 11. Se a pessoa não fizer nenhuma opção após ler o texto em sua forma original, a aplicação será automaticamente fechada e seu uso desautorizado, até que o passo <1> seja realizado novamente.

Fonte: produção do próprio autor

A Tabela 45 apresenta o resultado da aplicação da Determinação Hermenêutica de Requisitos no projeto de Criptoanálise, para constituir os requisitos de software.

Tabela 45 - Criptoanálise: Determinação Hermenêutica de Requisitos

Determinação Hermenêutica de Requisitos Requisitos de Software (RS)	
RS-1: Identificar-se na Aplicação	Assim que alguém abrir a aplicação, será solicitada sua identificação de acesso. Caso a identificação de acesso informada não seja válida, a aplicação não autorizará o seu uso, avisará que o acesso à aplicação foi negado e solicitará novamente a identificação de acesso. Se a identificação de acesso informada for válida, a aplicação autorizará o seu uso.
RS-2: Abrir Arquivo de Criptograma para Decodificação	A pessoa informa um arquivo de criptograma a ser decodificado e aplicação o verifica: 1. Se o arquivo de criptograma informado não puder ser aberto pela aplicação, um alerta será exibido informando que não é possível abrir o arquivo de criptograma informado e um novo arquivo de criptograma a ser decodificado será solicitado. 2. Se o arquivo de criptograma informado puder ser aberto, a aplicação abre este arquivo e verifica se o seu conteúdo é válido.

	a. Se o conteúdo do arquivo não for válido, um alerta será exibido informando que o conteúdo não é válido, o arquivo de criptograma será fechado e um novo arquivo de criptograma a ser aberto será solicitado. b. Se o conteúdo do arquivo de criptograma estiver válido, o seu conteúdo será exibido à pessoa.
RS-3 – Informar Criptograma para Decodificação	A pessoa informa (escrevendo ou falando) um criptograma a ser decodificado e a aplicação verifica se o seu conteúdo é válido. Se não for válido, um alerta será exibido informando que o conteúdo informado não é válido e um novo criptograma a ser decodificado será solicitado. Se for válido, o conteúdo a ser decifrado será exibido à pessoa.
RS-4: Decodificar Criptograma	Após um conteúdo de criptograma válido for exibido à pessoa: 1. O criptograma válido é transcrito no quadro-negro. 2. O controlador avalia o quadro-negro. 3. As fontes de conhecimento avaliam o quadro-negro. 4. As fontes de conhecimento fazem suposições sobre a cifra de substituição, de acordo com seus conhecimentos. 5. As fontes de conhecimento que inferirem sobre uma contribuição, avisam ao controlador que têm uma sugestão interessante a dar. 6. O controlador avalia as sugestões dadas pelas fontes de conhecimento. 7. O controlador seleciona a sugestão mais promissora à solução do problema. 8. O controlador ativa a fonte de conhecimento referente à sugestão selecionada. 9. A fonte de conhecimento ativada pelo controlador atualiza o quadro-negro com sua sugestão. 10. Se o criptograma ainda não foi resolvido, o processo é repetido a partir do passo <2>. 11. Se o criptograma foi resolvido, o texto original decodificado é exibido no quadro-negro.
RS-5: Encerrar Aplicação	A pessoa opta por encerrar a aplicação. Nesse momento, o conteúdo apresentado no quadro-negro é apagado, a aplicação desautoriza o seu uso e é fechada.

	Se a pessoa não tomar nenhuma ação após o texto original decodificado for apresentado no quadro-negro, automaticamente o seu conteúdo será apagado, a aplicação desautorizará o seu uso e fechará.

Fonte: produção do próprio autor

Criptoanálise: Aplicação do Teodolito Hermenêutico de Requisitos

Os requisitos da aplicação de criptoanálise foram definidos e compostos em cinco iterações, chamadas respectivamente de ciclo 1, ciclo 2, ciclo 3, ciclo 4 e ciclo 5, conforme detalhados a seguir.

Ciclo 1: durante esse ciclo, foi identificada a diferença situacional, mas o processo de decodificação de criptograma não foi suficientemente compreendido, impedindo a identificação dos requisitos de software. Sendo assim, ao aplicar o Teodolito Hermenêutico Requisitos, foram obtidos os seguintes resultados:

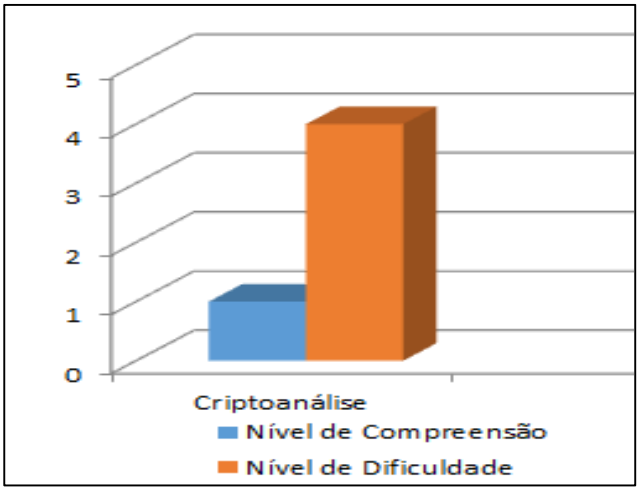
Tabela 46 - Criptoanálise: ciclo 1 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
Criptoanálise	1 – Pré-conceitual

Fonte: produção do próprio autor

Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 37 - Criptoanálise: ciclo 1 - leituras do domínio da aplicação



Fonte: produção do próprio autor

Ciclo 2: durante esse ciclo, foi identificada e descrita a diferença situacional e o processo de decodificação de criptograma foi razoavelmente compreendido, possibilitando a identificação e breve descrição dos requisitos de software “Identificar-se na Aplicação”, “Abrir Arquivo de Criptograma para Decodificação”, “Informar Criptograma para Decodificação” e “Encerrar Aplicação”. Sendo assim, ao aplicar o Teodolito Hermenêutico Requisitos, foram obtidos os seguintes resultados:

Tabela 47 - Criptoanálise: ciclo 2 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
Criptoanálise	3 – Contextual
Processo de Decodificação de Criptograma	2 – Conceitual

Fonte: produção do próprio autor

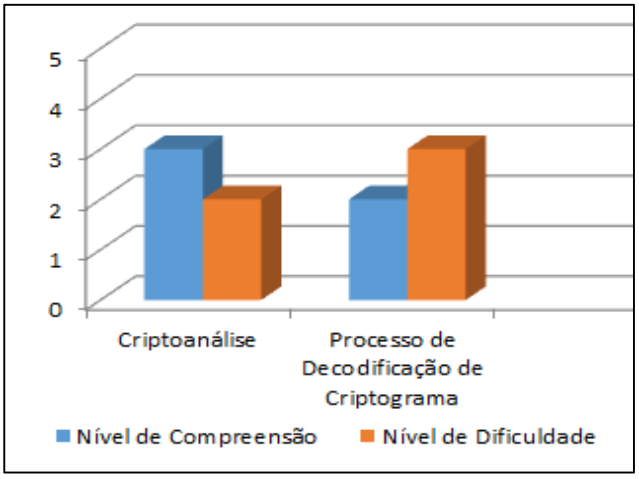
Tabela 48 - Criptoanálise: ciclo 2 - requisitos de software

Requisito de Software	Grau de Qualidade
Identificar-se na Aplicação	3.1 – O requisito está claro em relação ao seu escopo.
Abrir Arquivo de Criptograma para Decodificação	3.1 – O requisito está claro em relação ao seu escopo.
Informar Criptograma para Decodificação	3.1 – O requisito está claro em relação ao seu escopo.
Encerrar Aplicação	3.1 – O requisito está claro em relação ao seu escopo.

Fonte: produção do próprio autor

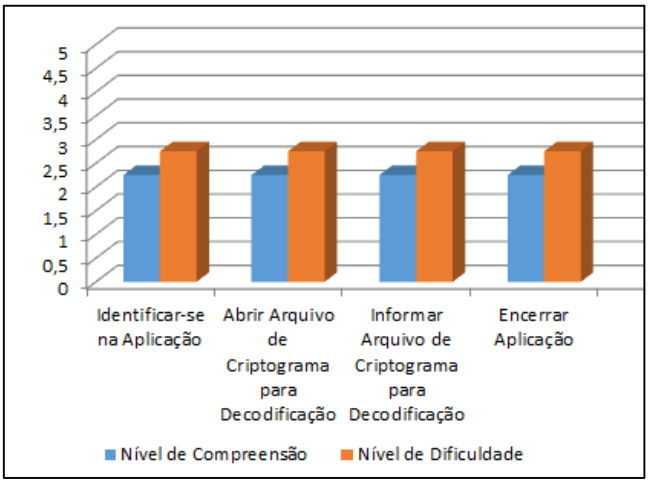
Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 38 - Criptoanálise: ciclo 2 - leituras do domínio da aplicação



Fonte: produção do próprio autor

Figura 39 - Criptoanálise: ciclo 2 - leituras dos requisitos de software



Fonte: produção do próprio autor

Ciclo 3: durante esse ciclo, foi descrita completamente a Identificação de Diferença Situacional e o processo de decodificação de criptograma foi bem compreendido, possibilitando a finalização das descrições dos requisitos de software “Identificar-se na Aplicação”, “Abrir Arquivo de Criptograma para Decodificação”, “Informar Criptograma para Decodificação” e “Encerrar Aplicação” e a identificação e breve descrição do requisito de software “Decodificar Criptograma”. Ainda durante esse ciclo, foi feito o Exame de Diferença Situacional, onde foram identificados e descritos o

Cenário, a Natureza, o Meio-Ambiente e os Utensílios (ou insumos) da aplicação de criptoanálise. Sendo assim, ao aplicar o Teodolito Hermenêutico Requisitos, foram obtidos os seguintes resultados:

Tabela 49 - Criptoanálise: ciclo 3 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
Criptoanálise	4 – Sistêmico
Processo de Decodificação de Criptograma	3 – Contextual

Fonte: produção do próprio autor

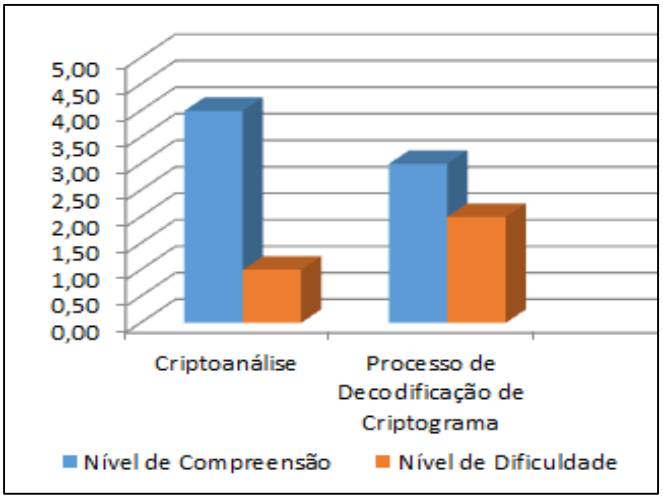
Tabela 50 - Criptoanálise: ciclo 3 - requisitos de software

Requisito de Software	Grau de Qualidade
Identificar-se na Aplicação	5.4 – O requisito foi aceito pelas partes interessadas como satisfazendo plenamente suas necessidades.
Abrir Arquivo de Criptograma para Decodificação	5.4 – O requisito foi aceito pelas partes interessadas como satisfazendo plenamente suas necessidades.
Informar Criptograma para Decodificação	5.4 – O requisito foi aceito pelas partes interessadas como satisfazendo plenamente suas necessidades.
Encerrar Aplicação	5.4 – O requisito foi aceito pelas partes interessadas como satisfazendo plenamente suas necessidades.
Decodificar Criptograma	3.1 – O requisito está claro em relação ao seu escopo.

Fonte: produção do próprio autor

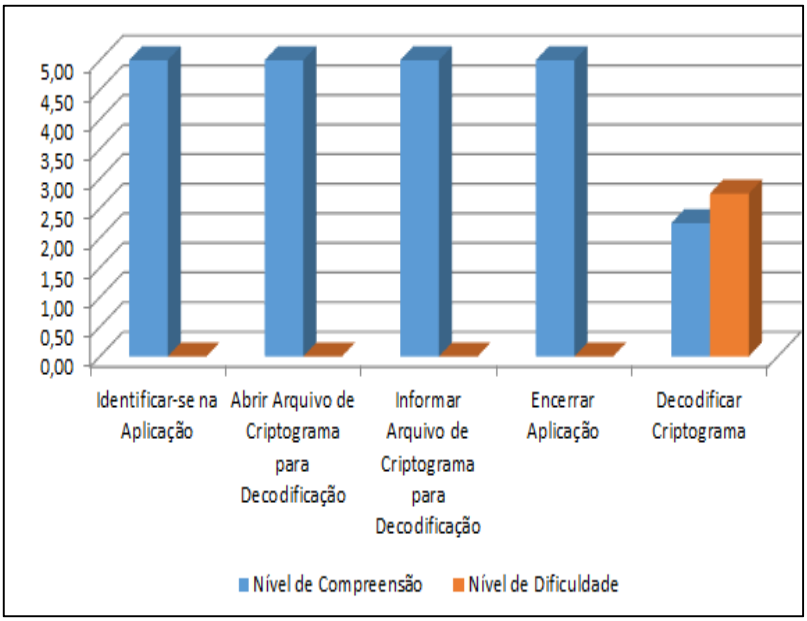
Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 40 - Criptoanálise: ciclo 3 - leituras do domínio da aplicação



Fonte: produção do próprio autor

Figura 41 - Criptoanálise: ciclo 3 - leituras dos requisitos de software



Fonte: produção do próprio autor

Ciclo 4: durante esse ciclo, foi feita a Determinação Hermenêutica de Requisitos, onde foram descritas as necessidades originais, as expectativas e as especificações aceitáveis (descrição e dinâmica). Com isso, foi obtida uma boa compreensão do processo de decodificação de criptograma. Sendo assim, ao aplicar o Teodolito Hermenêutico de Requisitos, foram obtidos os seguintes resultados:

Tabela 51 - Criptoanálise: ciclo 4 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
Criptoanálise	5 – Holístico
Processo de Decodificação de Criptograma	4 – Sistêmico

Fonte: produção do próprio autor

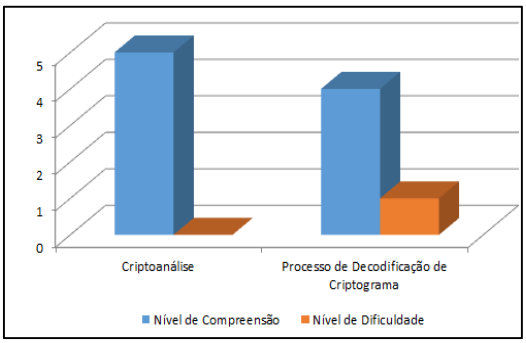
Tabela 52 - Criptoanálise: ciclo 4 - requisitos de software

Requisito de Software	Grau de Qualidade
Decodificar Criptograma	4.2 – O conjunto de itens do requisito fornece um claro valor para as partes interessadas.

Fonte: produção do próprio autor

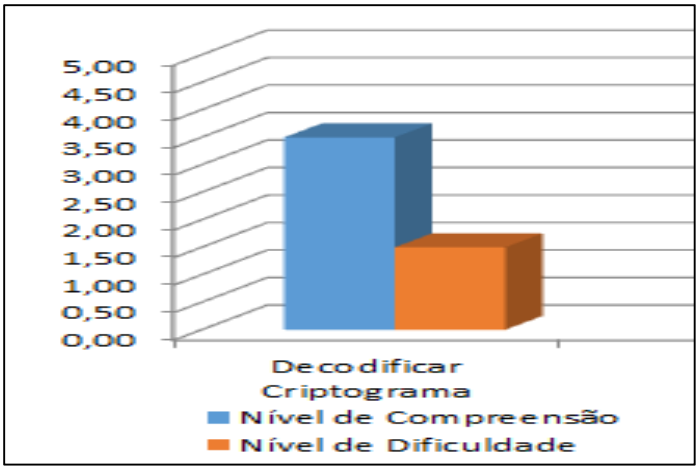
Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 42 - Criptoanálise: ciclo 4 - leituras do domínio da aplicação



Fonte: produção do próprio autor

Figura 43 - Criptoanálise: ciclo 4 - leituras dos requisitos de software



Fonte: produção do próprio autor

Ciclo 5: durante este ciclo, foi finalizada a Determinação Hermenêutica de Requisitos, onde todos os requisitos de software foram definidos e compostos completamente. Sendo assim, ao aplicar o Teodolito Hermenêutico de Requisitos, foram obtidos os seguintes resultados:

Tabela 53 - Criptoanálise: ciclo 5 - domínio da aplicação

Domínio da Aplicação	Nível de Compreensão
Criptoanálise	5 – Holístico
Processo de Decodificação de Criptograma	5 – Holístico

Fonte: produção do próprio autor

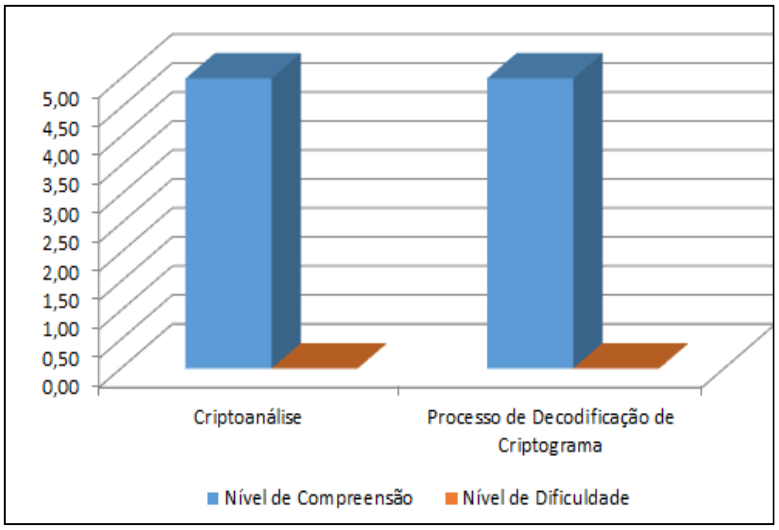
Tabela 54 - Criptoanálise: ciclo 5 - requisitos de software

Requisito de Software	Grau de Qualidade
Decodificar Criptograma	5.4 – O requisito foi aceito pelas partes interessadas como satisfazendo plenamente suas necessidades.

Fonte: produção do próprio autor

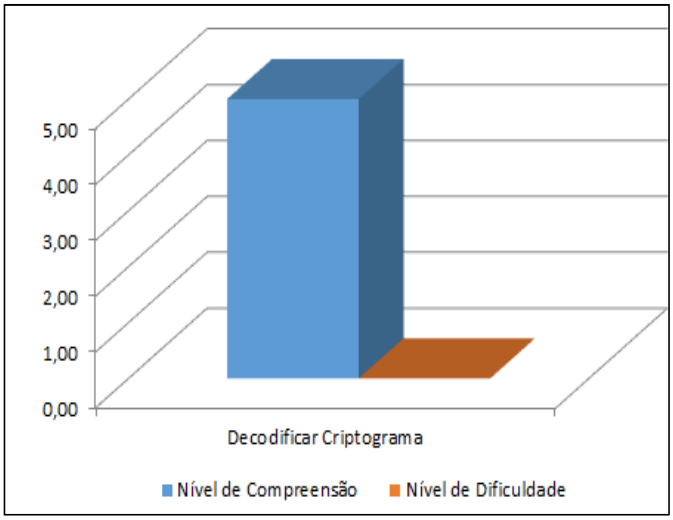
Assim, as leituras do Teodolito Hermenêutico de Requisitos foram as seguintes:

Figura 44 - Criptoanálise: ciclo 5 - leituras do domínio da aplicação



Fonte: produção do próprio autor

Figura 45 - Criptoanálise: ciclo 5 - leituras dos requisitos de software



Fonte: produção do próprio autor

7. CONSIDERAÇÕES FINAIS

Aos requisitos estão associados os principais problemas do desenvolvimento de software. Requisitos que não refletem as necessidades originais dos usuários, incompletos ou inconsistentes, mudanças de requisitos já acordados e a dificuldade para conseguir um entendimento comum entre todos os envolvidos são as principais dificuldades identificadas em um projeto no qual o sistema de software assume um papel determinante. Estas dificuldades identificadas geram retrabalho, atraso de cronograma, aumento de custo e a insatisfação dos usuários e clientes, isto quando o projeto não é cancelado precocemente.

Esta tese apresentou uma proposta para a solução aos problemas relacionados à Elicitação de Requisitos, que é a primeira tarefa a ser realizada durante a Engenharia de Requisitos, cujo objetivo é ajudar o engenheiro de requisitos a desenvolver sua compreensão sobre o domínio da aplicação, como também sobre os problemas a serem solucionados pelo software a ser desenvolvido e/ou as oportunidades de negócio a serem usufruídas com o seu apoio. Essa proposta é a Elicitação Hermenêutica de Requisitos.

A Elicitação Hermenêutica de Requisitos é o resultado da adequação do conceito *Dasein* proposto por Martin Heidegger. Ela trata os requisitos de software, cada um, em sua dimensão direta e imediata para compreender sua estrutura e sua essência, como também em sua dimensão ampla para compreender o cenário que o envolve e o fundamenta.

Outra proposta apresentada nesta tese (Teodolito Hermenêutico de Requisitos) vai ao encontro da avaliação e revelação dos níveis de compreensão do engenheiro de requisitos acerca do domínio da aplicação, assim como a avaliação e revelação dos graus de qualidade dos requisitos de software. Ao obter esses resultados, é possível estabelecer um plano estratégico para melhorar a aplicação da Engenharia Hermenêutica de Requisitos.

Estas duas propostas apresentadas nesta tese compõem a Engenharia Hermenêutica de Requisitos, que é a abordagem técnica que visa melhorar a elicitação e a avaliação dos requisitos de software que, por sua vez, foi verificada por meio da realização de testes de conceito para comprovar seu propósito. Estes testes de conceitos foram feitos

através de quatro estudos de casos, os quais demonstraram a eficiência da Engenharia Hermenêutica de Requisitos.

Desse modo, conclui-se que é viável aplicar a Engenharia Hermenêutica de Requisitos em projetos de desenvolvimento de software, obtendo melhores resultados na identificação, compreensão e interpretação do domínio da aplicação, como também na especificação dos requisitos de software, independentemente de seus graus de complexidade.

Com estes estudos de casos também foi possível constatar a independência da Engenharia Hermenêutica de Requisitos em relação ao processo de desenvolvimento de software, como também às abordagens, técnicas e ferramentas adotadas e utilizadas no projeto.

A Engenharia Hermenêutica de Requisitos mostrou ser uma abordagem técnica bastante eficiente e útil ao disponibilizar ao engenheiro de requisitos dois instrumentos que possibilitam melhorar sua capacidade de conceber os requisitos de software: Elicitação Hermenêutica de Requisitos e Teodolito Hermenêutico de Requisitos.

Por fim, ao aplicar a Engenharia Hermenêutica de Requisitos, o engenheiro de requisitos pode compreender melhor o domínio da aplicação para o qual o software será desenvolvido (juntamente com os problemas a serem solucionados por este software e/ou as oportunidades de negócio a serem usufruídas com o seu apoio), podendo, também, avaliar em que nível está sua compreensão acerca desses fatores, como, ainda, os graus de qualidade dos requisitos de software, tendo, assim, melhores condições de evoluir seus níveis de entendimento sobre esses itens, até alcançar um estágio satisfatório de compreensão para, com isso, estar mais capacitado a descrever os requisitos de software.

Portanto, com os requisitos de software descritos de maneira fidedigna ao domínio da aplicação, os engenheiros de software terão mais capacidade de desenvolverem o software que, em última análise, cumprirá com o seu objetivo primário: ajudar a melhorar a vida das pessoas, oferecendo a elas os suportes adequados e necessários à realização de suas tarefas.

Ao longo do desenvolvimento desta tese, percebeu-se que algumas outras contribuições podem ser desenvolvidas futuramente, conforme são mencionadas nos parágrafos que se seguem.

Como trabalho futuro, vislumbra-se a continuidade da adequação dos métodos hermenêuticos às demais tarefas da Engenharia de Requisitos: Análise de Requisitos, Especificação de Requisitos, Validação de Requisitos e Gestão de Requisitos, como sequência natural das propostas apresentadas nesta tese.

Futuramente, deseja-se desenvolver a Arquitetura Hermenêutica de Software, a qual objetiva dar continuidade ao ciclo de desenvolvimento de software, estabelecendo a estrutura de sua construção a partir das atividades realizadas e dos artefatos produzidos durante a aplicação da Engenharia Hermenêutica de Requisitos. Desse modo, a associação entre a Engenharia de Requisitos e o Projeto de Software ocorrerá de maneira mais eficaz e homogênea, possibilitando a rastreabilidade entre eles de forma mais fácil, prática e eficiente.

Outro novo trabalho vai ao encontro do desenvolvimento de um software para auxiliar a aplicação da Engenharia Hermenêutica de Requisitos e possibilitar o suporte automatizado para a Engenharia de Requisitos como um todo, não só para a Elicitação e Avaliação dos Requisitos de Software, como também para a Análise de Requisitos, a Especificação de Requisitos, a Validação de Requisitos e a Gestão de Requisitos.

O aprofundamento dos estudos no campo da filosofia hermenêutica (e outras áreas correlatas) também se mostra oportuno como trabalho futuro relacionado à elaboração de novas propostas que visam criar a Engenharia Hermenêutica, que poderá ser aplicada em diversas áreas do conhecimento onde a compreensão e interpretação se fazem necessárias para o desenvolvimento de seus produtos finais.

A Educação Hermenêutica vem a ser outra proposta de trabalho futuro a ser desenvolvido por meio do aprofundamento dos estudos no campo da filosofia hermenêutica (e outras áreas correlatas). Nesse caso, o foco é melhorar o processo de aprendizagem das pessoas em seus campos de estudos, através da aplicação de uma abordagem técnica voltada à construção do conhecimento de tal modo que sua compreensão seja e esteja mais consistente e melhor organizada.

REFERÊNCIAS BIBLIOGRÁFICAS

Almeida, Rogério da Silva. **O cuidado, o ser do ser-aí (Dasein) em Ser e Tempo**. Existência e Arte – Revista Eletrônica do Grupo PET – Ciências Humanas, Estética e Arte da Universidade Federal de São João Del-Rei – Ano IV, Número IV: Janeiro a Dezembro de 2008. p. 1-16.

Association for Computing Machinery. Disponível em <www.acm.org>. Acesso em 12/08/2017.

Association for Software Testing. Disponível em <www.associationforsoftwaretesting.org>. Acesso em 15/08/2017.

Barbosa, Márcio Ferreira. **A noção de ser no mundo em Heidegger e sua aplicação na psicopatologia**. Psicologia: Ciência e Profissão, V.18, n 3. Versão Impressa ISSN 1414-9893. Brasília: 1998.

BOOCH, Grady. **OBJECT-ORIENTED ANALYSIS AND DESIGN WITH APPLICATIONS**, 2 ed. Santa Clara, California: Addison-Wesley, 1994, p. 3 – 5; 381 - 394.

BOOCH, Grady...[et al.]. **OBJECT-ORIENTED ANALYSIS AND DESIGN WITH APPLICATIONS**, 3 ed. Boston, Massachusetts: Pearson Education, 2007, p. XV-XVII; 17-18; 27-28; 414-421.

Bourque, Pierre. & Fairley, Richard E. (Dick), eds. **SWEBOK v3.0: Guide to the Software Engineering Body of Knowledge**. IEEE Computer Society, 2014.

Buxton J.N. & Randell, B, eds. **Software Engineering Techniques: Report on a Conference Sponsored by the NATO Science Committee**. Rome, Italy, October 27-31, 1969. Scientific Af-fairs Division, NATO, 1970.

Castro, Paula Roberta de. **Heidegger e o problema das noções de sujeito e mundo em confronto com o problema da realidade em Ser e Tempo**. Dissertação. Universidade Federal de Goiás, Goiânia: 2009.

Ceia, Mário José Miranda. **A taxonomia SOLO e os níveis de van Hiele**. Escola Superior de Educação de Portalegre, 2002.

Chan C (2010) Assessment: Blooms' Taxonomy, Assessment Resources @HKU, University of Hong Kong [http://ar.cetl.hku.hk/large_class.htm]: Available: Accessed: 07/26/2018.

Coelho, José Henrique. **A questão do ser segundo Heidegger**. Pensamento Extemporâneo: filosofia a qualquer tempo: 2017. Disponível em http://pensamentoextemporaneo.com.br/?p=147. Acesso em 04/03/2017.

Cukierman, Henrique Luiz. Teixeira, Cássio. Prikladnicki, Rafael. **Um Olhar Sociotécnico sobre a Engenharia de Software**. RITA – Revista de Informática Teórica e Aplicada, Volume XIV, Número 2, 2007.

Engineering and Physical Sciences Research Council. Disponível em <www.epsrc.ac.uk>. Acesso em 15/08/2017.

European Software Institute. Disponível em <www.esicenter.bg>. Acesso em 15/08/2017.

Figueira, Anderson Marques da Silva. **Análise das técnicas de levantamento de requisitos para desenvolvimento de software nas empresas de Vitória da Conquista – BA**. Universidade Estadual do Sudoeste da Bahia – Departamento de Ciências Exatas. Vitória da Conquista – Bahia: 2012.

Filho, José Reinaldo Felipe Martins. **O outro, quem é ele? Considerações em torno da fenomenologia de Hussel, Heidegger e Lévinas. Griot** – Revista de Filosofia, v.1, n.1. Amargosa, Bahia: Julho de 2010.

Goguen, Joseph A. **Formal Methods and Social Context in Software Development**. Calhoun: The NPS Institutional Archive DSpace Repository, 1995.

Guidelines for Software Engineering Education. Version 1.0. Disponível em <www.dtic.mil/get-tr-doc/pdf?AD=ADA370372>. Acesso em 12/08/2017.

Hamilton Technologies. Disponível em <www.htius.com>. Acesso em 15/03/2017.

Hattie, J.A.C., & Brown, G.T.L. (2004, September).**Cognitive processes in asTTle: The SOLO taxonomy**. asTTle Technical Report #43, University of Auckland/Ministry of Education.

Heidegger, Martin. **Ser e Tempo – Parte I**. Tradução de Marcia Sá Cavalcante Schuback. 15 ed. Petrópolis, RJ: Vozes, 2005.

Heidegger, Martin. **Ser e Tempo – Parte II**. Tradução de Marcia Sá Cavalcante Schuback. 13 ed. Petrópolis, RJ: Vozes, 2005.

HISSAM, Scott...[et al.]. **Ultra-Large-Scale (ULS) Systems: Socio-Adaptive Systems**. Software Engineering Institute. Pittsburg, PA, 2016.

IBM. Disponível em <www.ibm.com>. Acesso em 12/08/2017.

Institute of Electrical and Electronics Engineers. Disponível em <www.ieee.org>. Acesso em 12/08/2017.

Intellom. Disponível em <www.inteloom.com>. Acesso em 15/08/2017.

International Converence on Software Engineering. Disponível em <www.icse-conferences.org>. Acesso em 15/08/2017.

International Council on Systems Engineering. Disponível em <www.incose.org>. Acesso em 12/08/2017.

Jean-Sébastien Vachon. **Why Projects Fail**. Inteloom, October 21, 2016.

Lawrence, Shari Pfleeger. **Engenharia de Software: teoria e prática**. Tradução de Dino Franklin. 2 ed. São Paulo: Prentice Hall, 2004, p. 5-6.

LeBlanc, Rich & Sobel, Ann. **Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering** – A Volume of the Computing Curricula Series. IEEE Computer Society and Association for Computing Machinery, 2015.

Medeiros, Ernani. **Desenvolvendo Software com UML 2.0: Definitivo**. 1 ed. São Paulo: Pearson Makron Books, 2004, reimp. 2010. p. 1-2.

NATO Industry Cyber Partnership. Disponível em <www.nicp.nato.int>. Acesso em 12/08/2017.

Naur P. & Randell, B., eds. **Software Engineering: Report of a Conference Sponsored by the NATO Science Committee**. Garmisch, Germany, October 7-11, 1968. Scientific Af-fairs Division, NATO, 1969.

Northrop, Linda. **Ultra-Large-Scale Systems: The Software Challenge of the Future**. Software Engineering Institute. Pittsburg, PA, 2006.

Object Management Group. Disponível em <www.omg.org>. Acesso em 12/08/2017.

Oliveira, Lauro Ericksen C. de. **O ser-com como compartilhamento da verdade do ser-aí**. Revista Saberes, v.3, número especial. Natal, RN: Dezembro de 2010.

OMG. Essence – **Kernel and Language for Software Engineering Methods**, Version 1.1. SMSC, 2015, p. 28-31.

OMG – Object Management Group. **Kernel and Language for Software Engineering Methods (Essence)**, v1.1. OMG Headquarters, 2015.

Portal Citações e frases famosas: Frases de Martin Heidegger. Disponível em https://citacoes.in/autores/martin-heidegger. Acesso em 22/09/2018.

Portal Filovida (Filosofia e Vida). Disponível em https://www.filovida.org. Acesso em 27/06/2018.

Pressman, Roger S. Maxim, Bruce R. **Engenharia de Software: uma abordagem profissional**. Tradução de João Eduardo Nóbrega Tortello. 8 ed. Porto Alegre: AMGH, 2016. p. 14-16.

Research Software Engineers Association. Disponível em <www.iasaglobal.org>. Acesso em 15/08/2017.

Roehe, Marcelo Vial, Dutra, Elza. **Dasein, o entendimento de Heidegger sobre o modo de ser humano**. Universidade Federal do Rio Grande do Norte: 2014. Doi: dx.doi.org/10.12804/apl32.1.2014.07.

Samaridi, Isadora. **O ser-no-mundo e suas possibilidades existenciais num contexto atual**. Anais do Congresso de Fenomenologia da região Centro-Oeste. Caderno de Textos – IV Congresso de Fenomenologia da região Centro-Oeste – 19-21 de Setembro de 2011.

Saurabh, Tiwari, Santosh, Singh Rathore. **A Methodology for the Selection of Requirement Elicitation Techniques**. arXiv:1709.08481v1. cs.SE. 25 Sep 2017.

Schach, Stephen R. **Engenharia de Software: Os Paradigmas Clássico & Orientado a Objetos**. Tradução de Ariovaldo Griesi. 7 ed. Porto Alegre: AMGH, 2010.

Silva, Maria Luísa Portocarrero Ferreira da. **Hermenêutica Filosófica**. Dissertação. Universidade de Coimbra, Coimbra: 2010.

Silva, Samuel Fabiano Barbosa. **Engenharia de Requisitos: Uma análise das técnicas de levantamento de requisitos**. Universidade FUMEC – Faculdade de Ciências Empresariais (FACE). Belo Horizonte: 2012.

Software Engineering Institute. Disponível em <www.sei.cmu.edu>. Acesso em 12/08/2017.

Software Sustainability Institute. Disponível em <www.software.ac.uk>. Acesso em 15/08/2017.

Sommerville, Ian. **Engenharia de Software**. Tradução de Kalinka Oliveira e Ivan Bosnic. 9 ed. São Paulo: Pearson Brasil, 2011, p. XI.

Spinola, Marcello Eloy Mendes. **A neo-ontologia do pensamento do *ereignis* na apropriação da linguagem humanista em Martin Heidegger**. Revista Vozes dos Vales da UFVJM: Publicações Acadêmicas, N. 2, Ano I. Minas Geais, Brasil: 2012.

Statpathy, Tridibesh. **A Guide to the SCRUM BODY OF KNOWLEDGE (SBOK GUIDE)**. SCRUMstudy, 2016.

Stein, Ernildo. **Coleção "Os Pensadores": Heidegger – Vida e Obra**. São Paulo, SP: Editora Nova Cultural, 2000.

Tamburri , Damian A., Kruchteny, Philippe, Lago, Patricia, Vliet, Hans van. **Organizational Social Structures for Software Engineering**. ACM Computing Surveys, Vol. 46, No. 1, Article 3, 2013.

Tamburri , Damian A., Kruchteny, Philippe, Lago, Patricia, Vliet, Hans van. **What Is Social Debt in Software Engineering?** IEEE, CHASE 2013.

Theodros, Tiruneh, Manisk, Kumar Mishra. **Analysis and Performance Evaluation of Requirement Elicitation Techniques**. International Journal of Computer Sciences and Engineering. V. 6, N. 4. 2018.

The Hillside Group. Disponível em <www.hillside.net>. Acesso em 15/08/2017.

The Standish Group. Disponível em <www.standishgroup.com>. Acesso em 12/08/2017.

The Standish Group. **Report: Chaos**, ed. 1994, 1995, 2011, 2012, 2013, 2014 e 2015.

Trainer, Erik H. Kalyanasundaram, Arun. Herbsleb, James D. **E-Mentoring for Software Engineering: A Socio-technical Perspective**. ICSE – International Conference on Software Engineering, 2017, Disponível em: <http://www.cs.cmu.edu/~etrainer/papers/trainer-icse2017-ementoringSE.pdf>. Acesso em 19/08/2017.

VARALDA, Wagner. **Engenharia de Software Orientada a Objetos**. 2003. Dissertação (Mestrado em Gerenciamento em Sistemas de Informação) – Pontifícia Universidade Católica de Campinas.

GLOSSÁRIO

Área de Negócio: Área do conhecimento para a qual o software será desenvolvido. Possui alguma relação com a Diferença Situacional.

Blackboard Model: Padrão arquitetural, desenvolvido em 1.962 que, inicialmente, lidava com os problemas de reconhecimento de fala e, após mostrar-se útil nesse domínio, passou a ser aplicado mais amplamente em outros domínios, incluindo interpretação de sinais, modelagem de estruturas moleculares tridimensionais, compreensão de imagens e planejamento. Demonstrou ser particularmente notável no que diz respeito à representação do conhecimento declarativo e eficiente em termos de desempenho, quando comparado com abordagens alternativas.

Brainstorming: Técnica para explorar a potencialidade criativa de uma pessoa (ou de um grupo de pessoas) colocando-a a serviço de objetivos pré-determinados. Esta técnica propõe que o grupo se reúna e utilize a diversidade de pensamentos e experiências para gerar soluções inovadoras, sugerindo qualquer pensamento ou ideia que vier à mente a respeito do tema tratado. Com isso, espera-se reunir o maior número possível de ideias, visões, propostas e possibilidades que levem a um denominador comum e eficaz para solucionar problemas e entraves que impedem um projeto de seguir adiante.

Comunidade de Negócio: Grupo de pessoas físicas e/ou jurídicas inseridas na Área de Negócio que interagem entre si de maneira colaborativa e contributiva para realizarem suas atividades / tarefas e produzirem seus utensílios (ou insumos). Possui alguma relação com a Diferença Situacional.

Dasein: É o tema central da filosofia desenvolvida por Martin Heidegger. Expressa a essência e o sentido do ser único que reside em cada ente.

Determinação Hermenêutica de Requisitos:	Constituição dos Requisitos de Software com base nos resultados obtidos através das aplicações da Identificação de Diferença Situacional e do Exame de Diferença Situacional.
Diagrama de Casos de Uso:	Um dos diagramas da UML. Utilizado para capturar e modelar os requisitos de software.
Diferença Situacional:	É percebida pelos Envolvidos como, dependendo da situação, manifestação de problemas ou oportunidades de negócio que ocorrem na Área de Negócio ou para a Comunidade de Negócio.
Domínio da Aplicação:	Base conceitual primária e essencial utilizada para especificar todos os requisitos de software, como também para formular o objetivo do projeto. Diz respeito ao contexto do negócio para o qual o software será desenvolvido.
Elicitação de Requisitos:	Primeira tarefa a ser realizada pelo Engenheiro de Requisitos, durante a aplicação da atividade Engenharia de Requisitos. Tem por finalidade compreender o Domínio da Aplicação, juntamente com os problemas a serem resolvidos pelo software a ser desenvolvido e as oportunidades de negócio a serem usufruídas com o seu apoio.
Elicitação Hermenêutica de Requisitos:	Métodos hermenêuticos, advindos da filosofia de Martin Heidegger, mais especificamente do *Dasein*, adaptados especificamente para ajudarem o Engenheiro de Requisitos a desenvolver melhor sua compreensão sobre o Domínio da Aplicação para o qual o software será desenvolvido, juntamente com os problemas a serem resolvidos por ele e as oportunidades de negócio a serem usufruídas com o seu apoio.

Engenharia de Requisitos: Uma das atividades da Engenharia de Software. Tem por objetivo estabelecer as metas a serem alcançadas pelo software a ser desenvolvido e, também, definir o comportamento necessário deste software, juntamente com suas restrições e condições impostas, para que ele atinja as necessidades do negócio em que atuará. Está organizada nas seguintes tarefas: Elicitação de Requisitos, Análise de Requisitos, Especificação de Requisitos, Validação de Requisitos e Gestão de Requisitos.

Engenharia de Software: Objetiva desenvolver software de maneira sistemática, controlada e quantificável, por meio da aplicação de uma série de atividades combinadas e integradas.

Engenharia Hermenêutica de Requisitos: Adequação conceitual de métodos hermenêuticos em uma abordagem técnica que auxilia o Engenheiro de Requisitos a conceber melhor os requisitos de software e também a avaliar e revelar seus níveis de compreensão (e de dificuldade) em relação ao Domínio da Aplicação e aos graus de qualidade dos requisitos de software. É formada pela Elicitação Hermenêutica de Requisitos e pelo Teodolito Hermenêutico de Requisitos.

Engenheiro de Requisitos: Um dos papéis da Engenharia de Software. É responsável por executar a atividade Engenharia de Requisitos.

Ente: Tudo aquilo que existe e se apresenta no aqui e no agora. De acordo com Martin Heidegger, o único ente capaz de questionar sobre o ser é o homem (a pessoa humana).

Envolvidos: Pessoas físicas e/ou jurídicas que formam tanto a Comunidade de Negócio como a Área de Negócio e que possuem alguma relação com a Diferença Situacional.

Ereigns: Ocorrência de um evento observável, que vem à tona e pode ser experimentado ou vivenciado. É o conceito por onde ocorre o desvelamento e a apropriação daquilo que até então estava oculta.

Especificação de Requisitos: Artefato que captura todos os requisitos de software para o sistema ou para uma parte do sistema.

Essence: Núcleo e linguagem que possibilitam a criação, o uso e o aprimoramento dos métodos de Engenharia de Software, fornecendo o ponto comum para definir as práticas de desenvolvimento de software. Foi desenvolvido no SEMAT e, atualmente, é mantido pelo OMG.

Graus de Qualidade: Níveis de maturidade dos requisitos de software, avaliados e revelados através do Teodolito Hermenêutico de Requisitos.

Hermenêutica: Ciência que estabelece os princípios, as leis e os métodos do processo humano de interpretação. Relaciona-se com a compreensão e a interpretação, para analisar o significado no campo da intersubjetividade, compreendendo e decifrando o sentido das coisas, e também identificando e estudando as leis que regem estas coisas. Estabelece um pensamento reflexivo que vai ao encontro da interpretação e compreensão contextualizada daquilo que se pretende conhecer.

Níveis de Compreensão: Graus de maturidade que indicam o quão o Engenheiro de Requisitos evoluiu em relação à sua compreensão em relação ao Domínio da Aplicação para o qual o software será desenvolvido.

Mitsein: Caráter ontológico do *Dasein*. É o que ocorre na relação do Ente com os outros. Tem a ver com comunhão e solidariedade.

Ontologia: Reflexão a respeito do sentido abrangente do ser. Trata, de maneira ampla, a natureza, a realidade e a existência dos entes.

Qualidade de Software:	Refere-se às características desejadas de um produto de software, à extensão em que um produto de software em particular possui essas características e aos processos, ferramentas e técnicas que são usadas para garantir essas características. No sentido mais geral, é a gestão efetiva aplicada de modo a criar um produto de software útil, que atende ou excede os requisitos acordados, que é feito dentro dos prazos e custos previstos, e que forneça valor mensurável para aqueles que o produzem e para aqueles que o utilizam.
Requisitos de Software:	São as capacidades, condições e restrições necessárias que o software deverá possuir para satisfazer ou alcançar seus objetivos.
Stakeholder:	Pessoa que é afetada, direta ou indiretamente, por um projeto, de forma positiva ou negativa. Por isto, possui grande interesse por sua execução. Tem conhecimentos e influências fundamentais que devem ser reunidos, avaliados e compreendidos para a determinação dos requisitos de software.
Taxonomia SOLO:	Classifica o nível de aprendizagem de uma pessoa em relação a um determinado tema que visa aprender, observando-se seu desenvolvimento por meio de um processo de aprendizagem adequado, estruturado em cinco níveis, onde cada nível representa um grau de progresso do aprendiz sobre seu conhecimento adquirido do objeto de estudo.
Teodolito Hermenêutico de Requisitos:	Métodos que avaliam e revelam os níveis de compreensão (e de dificuldade) do Engenheiro de Requisitos em relação ao Domínio da Aplicação e os graus de qualidade dos requisitos de software. Estes métodos são adequações da Taxonomia SOLO e do *Essence*.

ANEXO A – FERRAMENTAS DE APOIO À ENGENHARIA DE REQUISITOS

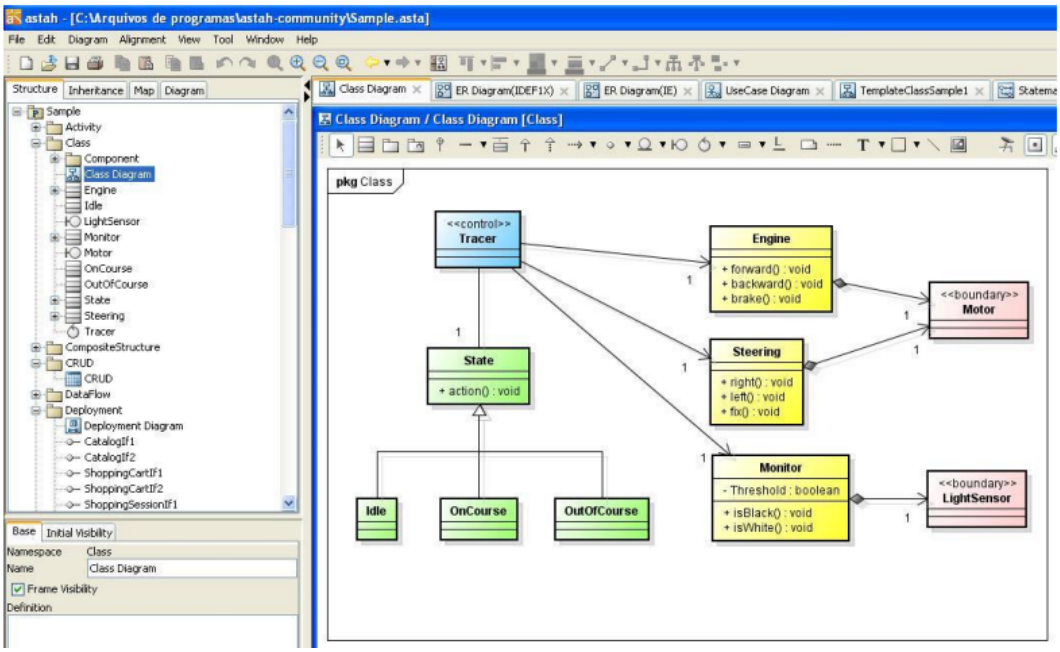
A seguir, são apresentadas algumas ferramentas utilizadas para apoiar algumas tarefas da Engenharia de Requisitos.

Astah

Ferramenta amplamente utilizada e muito simples de ser operada para a criação dos diagramas da UML. No site do fabricante (www.change-vision.com) pode ser encontrada uma versão para teste, já que ela é uma ferramenta proprietária.

A figura 46 ilustra o uso desta ferramenta na criação do Diagrama de Classes.

Figura 46 - Ferramenta ASTAH



Fonte: Figueira, p. 34

RequisitePro

Desenvolvida pela empresa Rational (atualmente, IBM – www.ibm.com) é uma ferramenta que objetiva o gerenciamento de requisitos de software. Ela possui um

banco de dados centralizado onde são alocados os requisitos de software, permitindo o acesso a eles de forma mais adequada pela a organização.

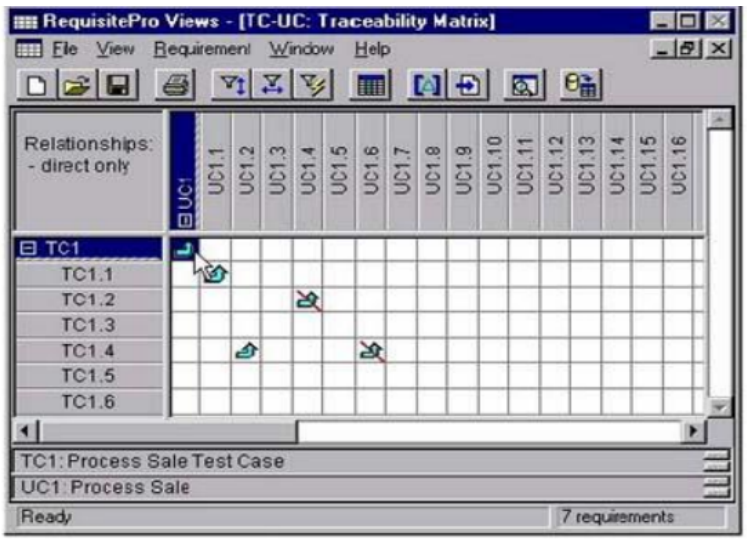
Possui integração com *Microsoft Word*, permitindo que os documentos sejam ligados ao seu banco de dados dinamicamente.

Dentre as diversas características que a ferramenta possui, algumas são:

- Organização dos requisitos através de parametrizações;
- Gerenciamento de características específicas dos requisitos, através de definição de atributos;
- Controle de relacionamentos entre requisitos;
- Poder visualizar o impacto nas alterações dos requisitos;
- Controlar o histórico de mudanças dos requisitos;
- Gerar informações estatísticas de requisitos;
- Desenvolver um projeto originário com base em um existente no banco de dados, podendo usufruir dos mesmos tipos de documentos, tipos de requisitos, atributos, e tipos de segurança do projeto de origem;
- Permite relacionar os requisitos a grupos de segurança;
- Permite o acesso distribuído dos requisitos.

A figura 47 ilustra uma das interfaces da ferramenta RequisitePro.

Figura 47 - Ferramenta RequisitePro



(Fonte: Figueira, p. 36

OSRMT

A ferramenta OSRMT, “Open Source Requirements Management Tool” (ou, Ferramenta de Código Aberto para Gerência de Requisitos) foi desenvolvida na *linguagem Java*, tendo como finalidade, apoiar o processo de gerência de requisitos.

Licenciada sob os termos da GPL (*General Public License*), pode ser adquirida livremente no site sourceforge.net.

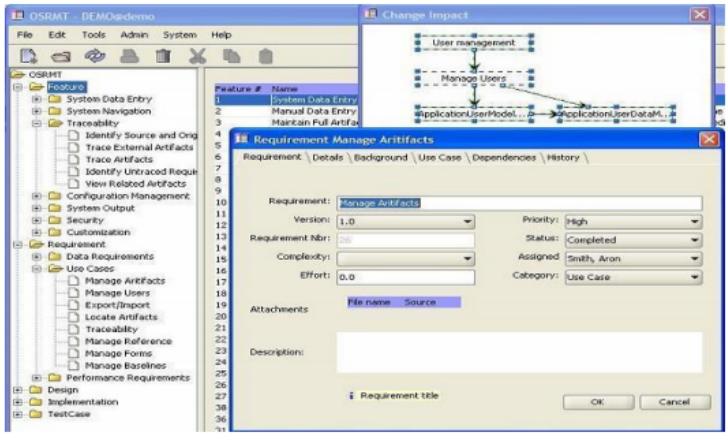
Uma das metas principais desta ferramenta é permitir uma completa rastreabilidade do ciclo de vida de desenvolvimento de software em relação aos seus requisitos.

Abaixo, são destacadas algumas funções desta ferramenta:

- Registro do autor, origem e motivo da necessidade de cada requisito;
- Registro de casos de uso, status e origem de cada requisito;
- Rastreabilidade (através de gráficos que identificam todas as dependências entre requisitos);
- Definição e organização de artefatos e entrada de dados;
- Possibilidade de gerar relatórios em formato PDF (Portable Document Format) de forma padronizada.

A figura 48 seguir ilustra uma das interfaces da ferramenta OSRMT.

Figura 48 - Ferramenta OSRMT



Fonte: Figueira, p. 37

CaliberRM

Ferramenta de gerenciamento de requisitos desenvolvida pela empresa *Borland* (www.microfocus.com).

Com esta ferramenta, os *stakeholders* podem acompanhar todos os requisitos de software encontrados e modificados durante todo o ciclo de vida do desenvolvimento do software.

Possui representação das rastreabilidades entre os requisitos de software através de matriz e diagramas.

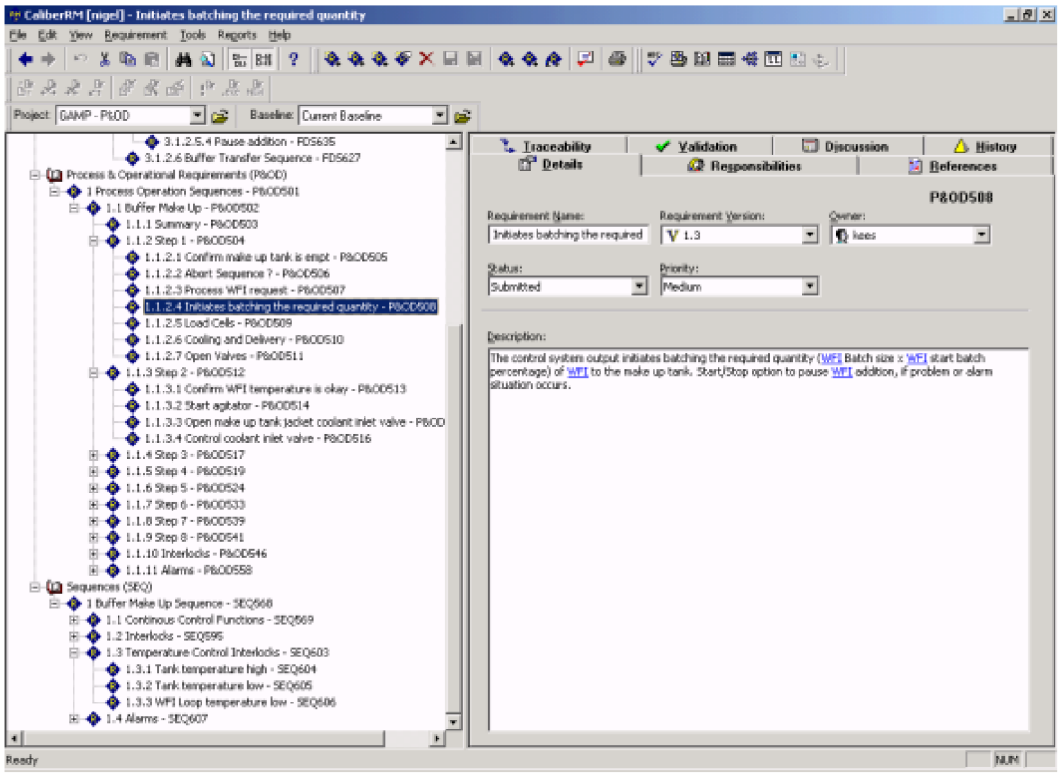
Segundo a *Borland* (2012) a ferramenta possui outras características, dentre as quais estão:

- Banco de dados centralizado e permite que sejam definidas matrizes e rastreabilidade entre os artefatos;
- Permite relacionamentos bidirecionais, em que é possível identificar o requisito de software que prova impacto e aquele que sofre a ação;
- Permite visualizar relacionamentos indiretos entre requisitos de software;
- Relacionamentos de artefatos modificados são marcados como suspeitos, cabendo aos usuários decidir se esse relacionamento ainda é válido;

- Permite o relacionamento de vários tipos de documentos aos requisitos de software;
- Mantém o rastreamento entre os requisitos de software de projetos distintos.

A figura 49 ilustra uma das interfaces da ferramenta CaliberRM.

Figura 49 - Ferramenta CaliberRM



Fonte: Figueira, p. 38

Comentário Final

Existem diversas ferramentas que são utilizadas para apoiar as tarefas da Engenharia de Requisitos (e a Engenharia de Software como um todo), as quais algumas delas foram apresentadas neste anexo, que não teve, de forma alguma, a intensão de apresentá-las a ponto de esgotar o assunto. Além destas, outras ferramentas são mencionadas nas referências destacadas ao longo da tese.

ANEXO B – TÉCNICAS PARA ELICITAR REQUISITOS

A seguir, são apresentadas algumas técnicas utilizadas durante a Elicitação de Requisitos, as quais foram brevemente citadas no capítulo 4. São técnicas utilizadas para facilitar e auxiliar a comunicação entre o Engenheiro de Requisitos e os Envolvidos.

Entrevista

É uma das técnicas mais utilizadas, a qual usa de questões que, após suas respostas serem filtradas, poderão se tornar em possíveis requisitos do software.

Basicamente existem dois tipos de entrevistas: as entrevistas fechadas, onde perguntas são definidas previamente e os Envolvidos irão respondê-las da forma que foi concebida, e as entrevistas abertas, em que não há um roteiro predefinido de questões, o Engenheiro de Requisitos explora diversos assuntos com a finalidade de obter uma maior compreensão sobre as necessidades dos Envolvidos.

As vantagens da utilização dessa técnica vão ao encontro da flexibilidade. As desvantagens vão ao encontro da grande quantidade de tempo e esforço necessários para que o Engenheiro de Requisitos entreviste diversos Envolvidos e, em seguida, analise suas respostas para chegar a um denominador comum em relação à definição dos requisitos de software.

Cenários

Um dos meios para se fazer um levantamento de informações visando desenvolver um novo software é por meio da construção de cenários. Essa técnica corresponde a uma descrição parcial do comportamento da aplicação. Ela pode ocorrer em um dado momento e em uma situação específica.

Utiliza-se de uma coleção de narrativas de situações referentes ao domínio do negócio para levantar as informações, identificar os problemas e, com isso, resolvê-los antecipadamente.

Os cenários devem ser elaborados durante as reuniões realizadas entre o Engenheiro de Requisitos e os Envolvidos e, de modo geral, deve possuir: uma descrição do que os Envolvidos esperam do software, uma descrição dos eventos do cenário; uma descrição do que pode ocasionar em erro e como isso será tratado e uma descrição do estado do software quando o cenário for finalizado.

Protótipo

Protótipo, ou Prototipação, é uma técnica que visa construir um protótipo inicial do software proposto. É uma versão inicial deste software, usada para demonstrar conceitos, experimentar opções de projeto, conhecer mais sobre o problema e suas possíveis soluções.

O protótipo permite aos Envolvidos a experimentação do software e, assim, verifiquem como o futuro software vai auxiliar em suas atividades. Além disso, durante esta experimentação, o protótipo pode revelar erros e omissões nos requisitos de software propostos, como também permitir aos Envolvidos a solicitação de novos recursos para o software.

Outros benefícios do protótipo são a possibilidade de validação dos requisitos de software encontrados e a verificação da viabilidade do projeto proposto (é um meio em que os Envolvidos avaliam se o software a ser desenvolvido será muito custoso, acarretando na inviabilidade de seu projeto).

O protótipo também auxilia na criação do projeto de interface com os usuários, pois estes, por exemplo, podem dizer sobre a dificuldade de utilizar o software em relação a encontrar funções, a complexidade dos menus, a linguagem complicada do software, botões que não representam o que efetivamente executam e dentre outras dificuldades com relação à interface.

Reuniões Facilitadas

Joint Application Development

Nos anos 70 a IBM canadense desenvolveu uma técnica para a Elicitação de Requisitos chamada Jo*int Application Dev*elopment (JAD). Ela visa criar sessões de

trabalho estruturadas, através de uma dinâmica de grupo e recursos visuais, em que os *stakeholders* envolvidos trabalham no intuito de encontrar os requisitos básicos do software a ser desenvolvido, além do *layout* de suas telas e relatórios. Com isso, JAD promove a cooperação e o entendimento entre os desenvolvedores e os Envolvidos.

Os componentes de uma sessão JAD variam entre oito e quinze pessoas de diversas áreas envolvidas com a produção do software. Dentre elas a um facilitador que tem a função de envolver os participantes nas reuniões, mediando às discussões e fazendo indagações a procura dos requisitos de software ainda não relatados.

Essa técnica se baseia em quatro princípios básicos: dinâmica de grupo, a qual possui um facilitador para estimular a capacidade dos indivíduos envolvidos; técnicas audiovisuais, que visam melhorar a comunicação e o entendimento entre os participantes; manutenção do processo organizado e racional; e a documentação padrão, em que os participantes da reunião preenchem e assinam.

JAD possui duas grandes etapas: planejamento, cuja finalidade é levantar e especificar os requisitos de software; e projeto, em que se lida com o projeto do software.

Existem determinados fatores que são responsáveis em obter sucesso utilizando a técnica JAD. É importante que o facilitador use sua experiência e habilidade, capture os participantes corretos, e defina seus papéis e responsabilidades. Deve também conter uma agenda detalhada e criar rapidamente um documento final. É imprescindível a presença de pessoas que tenham o poder de decisão, evitando assim contestação de decisões tomadas durante a sessão. As sessões devem ocorrer fora do local de trabalho, a fim de retirar o participante de sua rotina diária para evitar interrupções.

As experiências com JAD que resultaram em sucesso mostraram poucas mudanças de requisitos de software e uma redução de tempo e esforço gastos em seus levantamentos. Isso torna essa técnica relevante para o processo de aquisição de requisitos de software.

Observações

A observação é uma técnica de grande utilidade para aquisição de requisitos de software. Ela possibilita a inserção do desenvolvedor no ambiente de trabalho em que o usuário ou grupo de usuários trabalham, para observar as tarefas executadas por eles, sem interferir no ambiente. Ou seja, seu intuito é obter requisitos de software através da análise das tarefas realizadas pelos usuários.

Uma das técnicas de observação utilizadas para a Elicitação de Requisitos, muito empregada na área das ciências sociais, é a etnografia, que é uma técnica de observação que pode ser usada para compreender os requisitos sociais e organizacionais implícitos de um sistema. Ou seja, o desenvolvedor observa o dia-a-dia com a finalidade de anotar as tarefas reais dos Envolvidos.

Sendo assim, a técnica de observação tem como fator positivo a capacidade de coletar as atividades desempenhadas pelos Envolvidos de forma natural, obtendo como resultado final requisitos de software mais próximo de suas necessidades.

Por outro lado, essa técnica não é apropriada para a descoberta de novos requisitos de software, pois não é possível observar aquilo que não existe até o momento. Outra desvantagem da técnica é que o Envolvido, devido a presença de uma pessoa externa, pode não agir naturalmente na execução de suas atividades.

Histórias de Usuários

Histórias de Usuários são descrições feitas pelos próprios Envolvidos (usuários), como funções que gostariam que o software realizasse. Elas também servem para estimar o plano de entrega das versões do software e para a criação dos testes de avaliação do software para verificar se ele cumpre o que foi requisitado pelos Envolvidos.

Outras Técnicas

Todas essas técnicas citadas até este ponto são muito utilizadas durante a Elicitação de Requisitos, mas estas não são as únicas e tampouco devem ser utilizadas

isoladamente (os resultados obtidos melhoram bastante quando as técnicas são bem combinadas).

A seguir, algumas outras técnicas utilizadas durante a Elicitação de Requisitos serão apresentadas. Além dessas, outras também poderiam ser listadas, mas estenderia demais e desnecessariamente o propósito deste anexo e não agregaria valor à tese.

Pontos de Vista

As abordagens orientadas a pontos de vista levam em consideração as percepções dos *stakeholders*, sabendo que cada de um deles tem uma visão diferente do software proposto, obtém diferentes benefícios com relação ao sucesso do desenvolvimento do software e também está exposto a diferentes riscos, caso o software não venha atingir as expectativas aguardadas.

O Engenheiro de Requisitos tem a função de criar e organizar uma lista com os diferentes pontos de vista fornecidos de cada *stakeholder* à medida que forem surgindo os requisitos de software.

Os pontos de vista podem ser divididos em três tipos genéricos: pontos de vista de interação (são pessoas ou sistemas que interagem com o sistema), pontos de vista indiretos (*stakeholders* que não utilizam o sistema diretamente, mas tem influencia sobre os requisitos do software) e pontos de vista de domínio (são características e restrições que afetam os requisitos do software).

Após a coleta dos diferentes pontos de vista dos *stakeholders*, os requisitos de software resultantes podem ser inconsistentes ou conflitantes. O Engenheiro de Requisitos, então, tem a tarefa de categorizar essas informações e fazer prevalecer os requisitos de software que agregam consistência ao software.

Questionário

Uma das técnicas de Elicitação de Requisitos que pode abranger um grande número de pessoas é o questionário. O seu uso é essencial quando se deseja obter informações de inúmeras pessoas. Além disso, justifica-se a sua aplicabilidade

quando há indisponibilidade física, dispersão das pessoas envolvidas no projeto ou, até mesmo, quando há necessidade de um levantamento estatístico das pessoas que utilizarão o software.

Esta técnica permite adquirir informações de várias pessoas afetadas pelo software. Pode-se obter um *feedback* sobre problemas ou identificar possíveis melhorias em relação ao sistema.

O processo de criação do questionário não é tão simples como pode parecer. É preciso usar uma metodologia para a formulação de suas questões, de acordo com o perfil de usuários que irão respondê-lo. O planejamento adequado sobre o conteúdo, formato, ordem, clareza e não ambiguidade das questões são fatores relevantes a serem considerados para a sua.

A flexibilidade de horário para o Envolvido responder o questionário é um fator positivo, como também é a sua possibilidade de ser aplicada a um grande número de pessoas a um baixo custo.

Uma desvantagem desta técnica é a sua inflexibilidade, pois impossibilita a análise de questões subjetivas, as quais podem ser de grande importância para o entendimento do problema.

Casos de Uso

A técnica de modelagem através de casos de uso está sendo utilizada cada vez mais pelos Engenheiros de Requisitos. Idealizada na década de 1970, por Ivar Jacobson, e posteriormente incorporada à UML, tem sido de grande utilidade para a documentação dos requisitos funcionais de um sistema.

A modelagem baseada em casos de uso possui uma notação gráfica relativamente simples e uma descrição em linguagem natural que facilitam a comunicação entre o desenvolvedor e os Envolvidos.

Os modelos de casos de uso são compostos por atores, casos de uso e relacionamentos entre eles e descrevem o software ou o sistema sob os pontos de vista de seus usuários.

Um caso de uso é a representação de uma sequência de interações entre um sistema e os atores (os atores são os agentes externos que interagem com esse sistema) e é descrito de forma narrativa para detalhar seus possíveis comportamentos quando entrar em execução.

Análise de Documentos

Essa técnica é aplicada através da dedução de conhecimentos já expressos, ou melhor, escritos dentro do universo de informações. Para a modelagem de requisitos, ela é muito útil, pois ajuda a definir os objetos que irão compor o modelo.

Com a análise de documentos, o Engenheiro de Requisitos passa a conhecer melhor o vocabulário utilizado no domínio do problema e, com isto, a criação do glossário do projeto é facilitada.

Brainstorming

É uma das técnicas de reuniões em grupo mais conhecida para levantamento de requisitos de software. Ela consiste em uma reunião de especialistas de diversos setores, sendo que cada componente tem a função de estimular ao outro a criação de ideias, visando à resolução do problema em questão. Normalmente, aplica-se essa técnica na fase inicial do desenvolvimento do software, quando ainda se conhece pouco sobre o propósito do projeto.

As ideias obtidas não devem ser criticadas, pois o seu objetivo é gerar novas ideias através da liberdade dos participantes de criar e aceitar as ideias sugeridas pelo grupo. Com isso, uma sessão bem sucedida de *brainstorming* resulta em um conjunto de boas ideias e os participantes sentem que contribuíram de alguma maneira para a solução inovadora do problema.

Comentário Final

Existem diversas técnicas que são utilizadas para elicitar requisitos de software, as quais algumas delas foram apresentadas neste anexo, que não teve, de forma alguma, a intensão de apresentá-las a ponto de esgotar o assunto. Mesmo assim, foi possível constatar que são instrumentos importantes e eficientes para coletar e documentar as necessidades dos Envolvidos e também para facilitar e auxiliar a comunicação entre eles e o Engenheiro de Requisitos.