PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM ESTUDO EMPÍRICO SOBRE O USO DO BDD E SEU APOIO A ENGENHARIA DE REQUISITOS

LAURIANE CORRÊA PEREIRA MORAES

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientadora: Profa. Dra. Sabrina dos Santos Marczak

Porto Alegre
2016

Ficha Catalográfica

M827e Moraes, Lauriane Corrêa Pereira

Um estudo empírico sobre o uso do BDD e seu apoio a Engenharia de Requisitos / Lauriane Corrêa Pereira Moraes . – 2016.
139 f.
Dissertação (Mestrado) – Programa de Pós-Graduação em Ciência da Computação, PUCRS.

Orientadora: Profa. Dra. Sabrina dos Santos Marczak.

1. Behavior-Driven Development. 2. Engenharia de Requisitos. 3. Metodologias Ágeis. 4. Método Empírico. 5. Entrevistas. I. Marczak, Sabrina dos Santos. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da PUCRS com os dados fornecidos pelo(a) autor(a).

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada "Um estudo empírico sobre o uso do BDD e seu apoio a Engenharia de Requisitos" apresentada por Lauriane Correa Pereira Moraes como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, aprovada em 29 de março de 2016 pela Comissão Examinadora:

Profa. Dra. Sabrina dos Santos Marczak– Orientadora — PPGCC/PUCRS

Profa. Dra. Milene Selbach Silveira – — PPGCC/PUCRS

Prof. Dr. Cleidson Ronald Botelho de Souza - — UFPA

Homologada em...../...../....., conforme Ata No. ..... pela Comissão Coordenadora.

Prof. Dr. Luiz Gustavo Leão Fernandes
Coordenador.

PUCRS PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Campus Central
Av. Ipiranga, 6681 – P. 32 – sala 507 – CEP: 90619-900
Fone: (51) 3320-3611 – Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

“As coisas são como são. :)”
(Cleidson de Souza)

AGRADECIMENTOS

A Deus por me guiar e dar saúde e força para conquistar mais este sonho.

Aos meus pais e mentores, Célio Pereira e Laureci Corrêa, por todo o apoio, dedicação, esforço e liderança. As suas atitudes me conduziram a alcançar mais um sonho em minha vida.

À minha vó Gessi Pereira, irmã Celiane Pereira, Jean Moraes e seus pais e irmãs, pela paciência, compreensão, apoio, amizade e companheirismo.

À minha família: tias, tios, primos e aos amigos por todo apoio.

Aos Professores que participaram do meu processo de aprendizado e a Sabrina Marczak pelo seu apoio e conselhos.

À Josiane Kroll pela revisão do texto e pela paciência em fazer reflexões sobre está pesquisa.

Ao Prof. Cleidson de Souza por ceder o *debriefing* e pela colaboração.

Aos Profs. Michael Móra e Ricardo Bastos pelos *insights* gerais sobre está pesquisa.

Ao Gabriel Oliveira pelo interesse no tema e discussões ao longo do jornada.

Aos colegas do grupo, pelas conversas e contribuições durante o período do Mestrado, que me proporcionaram aprendizado.

Aos entrevistados que participaram deste estudo, sem eles este estudo não teria o mesmo valor.

UM ESTUDO EMPÍRICO SOBRE O USO DO BDD E SEU APOIO A ENGENHARIA DE REQUISITOS

RESUMO

A disciplina Engenharia de Requisitos centra-se na identificação das necessidades do cliente e especificação de requisitos de software para atender estas necessidades. No desenvolvimento ágil, esta etapa acontece no decorrer do ciclo de vida de desenvolvimento apoiada por um conjunto de práticas propostas pelos métodos ágeis. Dentre estas, tem-se *Behavior-Driven Development* (BDD) que integra uma linguagem ubíqua com *Test-Driven Development* e testes automatizados, projetada para auxiliar equipes a construírem e entregarem software mais eficientemente. O BDD centra-se na colaboração e descoberta do comportamento do sistema através de exemplos advindos da prática *Specification by Example*. Os cenários de negócio levantados para o BDD objetivam facilitar o compartilhamento da informação e redução do desalinhamento da comunicação, recorrentes da Engenharia de Requisitos. A automação dos cenários permitem a criação de documentação viva, o qual evita sua obsolescência, outro problema crítico da área. Esta dissertação apresenta um estudo empírico que visou identificar como o BDD é adotado na prática e como o mesmo apoia a Engenharia de Requisitos. Para isto, conduziu-se 24 entrevistas semi-estruturadas com profissionais da indústria explorando aspectos relacionados ao BDD, tais como processos, ferramentas de apoio, papéis envolvidos, benefícios, entre outros. Um conjunto de investigações futuras são apontadas para que pesquisadores possam explorar detalhadamente os aspectos identificados e uma reflexão é oferecida aos profissionais da indústria.

Palavras-Chave: Behavior-Driven Development, Engenharia de Requisitos, Metodologias Ágeis, Método Empírico, Entrevistas.

AN EMPIRICAL STUDY ON THE USE OF BDD AND ITS SUPPORT TO REQUIREMENTS ENGINEERING

ABSTRACT

The Requirements Engineering discipline focuses on identifying the customer needs and specifying the software requirements to attend these needs. In agile development, the related activities take place throughout the development lifecycle with the support of a set of agile practices. Behavior-Driven Development (BDD) is among one of these practices. BDD integrates an ubiquitous language with Test-Driven Development and automated tests. It was designed to help teams to build and deliver software more efficiently. It aims to promote collaboration and the discovery of the system behavior through examples following the *Specification by Example* agile practice. The business scenarios defined with BDD aim to facilitate information sharing and reduction of communication misalignment, recurring issues in Requirements Engineering. The automation of the scenarios allow the team to create living documentation, which in time avoid obsolete documentation, another critical issue in this discipline. This Dissertation presents an empirical exploratory study that aimed to identify how BDD is adopted in practice and how it supports Requirements Engineering. We conducted 24 semi-structured interviews with industry professionals in order to explore aspects related to BDD such as processes, tool support, associated roles, BDD benefits, among others. Future work is pointed out and a summary of benefits for industry is discussed.

Keywords: Behavior-Driven Development, Requirements Engineering, Agile Development, Empirical Study, Interview.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ER – Engenharia de Requisitos

BDD – Behavior-Driven Development

SBE – Specification by Example

XP – Extreme Programming

FDD – Feature-Driven development

JAD – Joint Application Development

IEEE – Institute of Electrical and Electronics Engineers

TDD – Test-Driven Development

DDD – Domain-Driven Design

ATDD – Acceptance-Test-Driven Development

ATDP – Acceptance Test-Driven Planning

GT – Grounded Theory

QP – Questão de Pesquisa

QA – Quality Assurance

PO – Product Owner

BA – Business Analyst

API – Application Programming Interface

SUMÁRIO

1. INTRODUÇÃO

Este capítulo apresenta uma visão geral desta Dissertação de Mestrado. A Seção 1.1 descreve o contexto, motivação e problemática deste estudo. A Seção 1.2 expõe os objetivos que o estudos se propôs a atender. A Seção 1.3 apresenta as principais contribuições do trabalho e, por fim, a Seção 1.4 apresenta como esta Dissertação está organizada.

1.1 Contexto, Motivação e Problemática

A Engenharia de Requisitos é uma disciplina importante na Engenharia de Software. Nesta, busca-se entender o que o software deveria fazer e como se comportar para atender as necessidades do cliente [10, 73]. No entanto, os usuários finais e os clientes frequentemente ficam incertos sobre suas reais necessidades, ou as alteram, junto as suas prioridades, ao longo do processo de desenvolvimento de software. Membros da equipe do projeto, inicialmente, não possuem o conhecimento dos conceitos e termos do negócio, ou não estão familiarizados com o problema do cliente, podendo causar o desalinhamento dos requisitos e o não entendimento de como o software deveria se comportar [80]. Ainda, culturas distintas e distância física geralmente agravam esses problemas [21, 34]. Isto significa que há mais oportunidades para as informações serem perdidas, mal entendidas, ou ignoradas. Dessa forma, a comunicação frequente e clara com uma linguagem que provenha um entendimento comum entre os envolvidos se torna crucial [3].

Atualmente, os *stakeholders*, as partes interessadas no projeto, precisam estar envolvidas, refletindo as atuais necessidades do negócio por estarem imersos a um mercado em constante mutação [11, 63]. Então, todos os *stakeholders* deveriam estar cientes das necessidades do negócio e como as mesmas impactariam nos requisitos. Diante deste contexto, as metodologias ágeis incentivam a tentativa de fazer tudo isto de maneira eficiente, se adaptando a indústria moderna [9].

Para acompanhar as tendências da indústria, o processo de desenvolvimento de software tornou-se mais dinâmico e adaptativo, havendo a constatação da maior aderência as metodologias ágeis na indústria nos últimos anos [78]. No entanto, os problemas de Engenharia de Requisitos (ER) são recorrentes há mais de 20 anos mesmo com o uso dessas práticas e metodologias. Neste contexto, surge o *Behavior-Driven Development* (BDD), ou o desenvolvimento orientado ao comportamento, como complemento ao desenvolvimento ágil. O BDD engloba especificações que permitem melhor engajamento da equipe do projeto com os *stakeholders*, objetivando o entendimento comum dos requisitos [2, 71], emergindo como uma potencial solução para os referidos problemas [64]. Esta abordagem está sendo cada vez mais adotada e tem ganhado atenção nos últimos anos, tanto da acade-

mia quanto da indústria [72]. A provável causa de adoção pela indústria está associada a simplicidade de uso da abordagem [50].

O BDD incentiva a discussão sobre as necessidades do cliente através de exemplos com o intuito de explorar e expressar os objetivos de negócio e as funcionalidades que devem ser implementadas. Esses exemplos seguem o que é proposto pela prática *Specification by Example* (SBE), ou especificação por exemplo, que representa o conjunto de padrões de processo para ajudar a equipe na construção de software [2]. O mesmo inicia-se pela Injeção de funcionalidade (do Inglês, *Feature Injection*), técnica de extração do escopo do projeto a partir das necessidades do negócio [54], podendo chegar às especificações executáveis [2]. O BDD vale-se do SBE e de outras práticas que auxiliam o entendimento do negócio do cliente e o desenvolvimento do software. Isto reflete em um conjunto de especificações que expressam o comportamento do sistema a ser desenvolvido, o qual pode estar atrelado ao código funcional a ser desenvolvido, fazendo com que a equipe do projeto se mantenha consciente de possíveis alterações no software e possa realizar a automatização do código [2, 71]. Estes exemplos são refletidos cenários que podem ser definidos em uma cláusula ou através de gherkin (cláusulas previamente definidas: Dado que-Quando-Então para definição do comportamento).

Apesar de gerar expectativas de resolução de problemas da ER, diante dos benefícios ressaltados que permeiam o uso do BDD, como: Alinhamento dos termos do negócio [71], melhora entendimento e comunicação dos *stakeholders* no projeto [23], melhora na qualidade do produto [71] e possibilidade de rastreamento [71]. Ainda há questões em aberto quanto ao BDD e as práticas relacionadas ao mesmo. Se o BDD requer um envolvimento contínuo com o cliente e a literatura aponta dificuldades em obter o envolvimento do cliente principalmente durante a elicitação dos requisitos [60] - como isto é realizado quando há o uso do BDD. Ainda, se o alinhamento do vocabulário é necessário para apoiar o entendimento dos requisitos, será que esta adoção tem auxiliado as equipes neste sentido?

Diante do exposto, há questionamentos com relação ao uso do BDD nesse processo de ER neste ambiente em constante mutação, no qual as empresas modernas se encontram. Relatos informais e superficiais em formatos de *blogs*, grupos, entre outros, da indústria sobre experiências particulares estão disponíveis, fomentando a tendência de adoção na indústria. Consequentemente, há a necessidade da identificação de como é realizado o BDD. Como um todo, pesquisadores investigam suas características gerais [72] e seu processo [51]. Entretanto, não foram encontrados estudos que discutam em detalhes os aspectos investigados neste estudo, os quais devem enaltecer a compreensão da forma que o BDD tem sido usado na prática e disponibilizar a profissionais e pesquisadores um entendimento consolidado e estruturado sobre o assunto.

Levando em conta este contexto, este estudo identificou como é adotado o fenômeno BDD e como o mesmo auxilia o processo de engenharia de requisitos através de um estudo qualitativo empírico, baseado em entrevistas semi-estruturadas.

1.2 Objetivo geral e específicos

O objetivo geral deste estudo foi identificar como o BDD é utilizado na prática, para que se possa identificar como o mesmo apoia a engenharia de requisitos e auxilia nos problemas de ER. Para que fosse possível atingir o objetivo geral, foram definidos os seguintes objetivos específicos:

- Obj1. Realizar o levantamento dos conceitos e problemas da engenharia de requisitos relatados na literatura;
- Obj2. Aprofundar o conhecimento sobre metodologias ágeis e suas práticas;
- Obj3. Aprofundar o conhecimento sobre o conceito de BDD;
- Obj4. Identificar como o BDD é utilizado na prática;
- Obj5. Identificar como o BDD auxilia na engenharia de requisitos e seus respectivos problemas.

1.3 Contribuições

A principal contribuição deste trabalho esta centrada em uma definição de como o BDD é utilizado na prática pela indústria, revelando como profissionais realizam o BDD como um todo, em qual tipo de projeto o mesmo é adotado, os envolvidos em sua utilização, ferramental de apoio para seu uso, entre tanto outros aspectos. Ainda, esta descrição também envolve o entendimento destes profissionais sobre como o BDD apoia a engenharia de requisitos, tal como a extensão do envolvimento de *stakeholders*, como se dá a escrita de exemplos e cenários que expressam o comportamento do sistema, etc, e quais problemas de ER são mitigados por BDD, como, por exemplo, redução de dificuldades de comunicação e facilitação do entendimento do escopo do sistema. Aproximar o cliente continua sendo um desafio deixado em aberto, mas existe um outro grande conjunto de benefícios que são relatados em detalhes neste estudo.

Os resultados desta pesquisa de Mestrado são importantes para se entender o uso real de uma abordagem promissora para a resolução de problemas conhecidos em desenvolvimento de software por mais de duas décadas. O usufruto destes resultados, tanto pela academia, ciência da computação como pela a indústria. Estes poderão auxiliar profissionais da área a definir seus processos e organizar suas equipes, entender a importância

do apoio ferramental, buscar definir estratégias para superar os desafios encontrados na adoção do BDD, entre tantos outros. Academia e indústria possuem a sua disposição um relato exploratório rico e diversificado que estabelece um ponto inicial de referência para ações como as citadas acima.

1.4 Estrutura da Dissertação

Esta dissertação está organizada em seis capítulos conforme segue: o **Capítulo 1 (atual capítulo)** descreve a motivação, o problema, os objetivos e principais contribuições deste estudo; o **Capítulo 2** apresenta o referencial teórico dos conceitos envolvidos neste estudo, quais sejam: Métodos ágeis, Engenharia de Requisitos e seus problemas, e BDD e conceitos relacionados. o **Capítulo 3** apresenta a metodologia de pesquisa adotada no estudo, descrevendo os métodos utilizados e como os mesmos foram utilizados; o **Capítulo 4** descreve os resultados encontrados com o estudo; o **Capítulo 5** apresenta a discussão sobre os resultados expostos no Capítulo 4, levando em consideração o que é apresentado na literatura (embasamento teórico e trabalhos relacionados); e, o **Capítulo 6** apresenta as considerações finais, incluindo as limitações do estudo, as conclusões e os trabalho futuros.

2. REFERENCIAL TEÓRICO

Este capítulo apresenta os conceitos que serviram de embasamento para este estudo. A Seção 2.1 introduz o tema de desenvolvimento ágil, destacando algumas das metodologias propostas e suas práticas. A Seção 2.2 discute a Engenharia de Requisitos, suas atividades e problemas relacionados. A Seção 2.3, por sua vez, discorre sobre *Behavior-Driven Development* e outros conceitos relacionados.

2.1 Metodologias Ágeis

Pressman (2010) reforça que os Métodos Ágeis se desenvolveram do esforço de sanar as fraquezas reais e perceptíveis da engenharia de software convencional, como, dificuldades de comunicação, acompanhamento da documentação dos requisitos que reflitam o negócio visto a necessidade de mudança dos mesmos, e outras. Kent Beck e outros 16 renomados desenvolvedores, autores e consultores da área de software, batizados de *Agile Alliance*, assinaram o Manifesto para desenvolvimento ágil de software, em 2001, que teve os seguintes pilares [9]:

- *Indivíduos e interações mais que processos e ferramentas;*
- *Software em funcionamento mais que documentação abrangente;*
- *Colaboração com o cliente mais que negociação de contratos;*
- *Responder a mudanças mais que seguir um plano.*

O desenvolvimento ágil oferece benefícios importantes, no entanto, não é indicado para todos os projetos, pessoas, situações e produtos [63]. A agilidade incentiva a estruturação e as atitudes em equipe, centrando na comunicação; enfatiza a entrega rápida do software operacional; diminui a importância dos artefatos intermediários; assume o cliente como parte da equipe de desenvolvimento; e reconhece que o planejamento em um mundo incerto tem seus limites e que o plano de projeto deve ser flexível [9, 63]. Nestas circunstâncias, a agilidade pode ser aplicada a qualquer processo de software que considere o desenvolvimento adaptativo como foco [63].

Diante disto, houve a criação de diversas metodologias e práticas para apoiar esta agilidade, conforme brevemente apresentado a seguir.

A) **Scrum:** Centra-se na gestão de projetos em situações onde há dificuldade em realizar planejamento com antecedência, ressaltando a importância dos mecanismos de *feedback* [66]. O Scrum objetiva, segundo Schwaber e Beedle [67], definir um processo para desenvolvimento de software orientado a objetos, que seja focado em pessoas e que seja

indicado para ambientes em que os requisitos surjam e mudem constantemente. A metodologia se baseia nos seguintes princípios: equipes pequenas entre cinco a nove pessoas [30], requisitos que são pouco estáveis ou desconhecidos e iterações curtas. Divide o desenvolvimento em intervalos de tempos de, no máximo, 30 dias, chamados *sprints*. Não requer ou fornece qualquer técnica ou método específico para a fase de desenvolvimento de software, apenas estabelece conjuntos de regras e práticas gerenciais que devem ser adotadas para o sucesso de um projeto. As práticas gerenciais do Scrum são: *product backlog, sprint backlog, increment* (soma de todo product backlog), *daily scrum, sprint, sprint planning meeting, sprint retrospective*, e *sprint review meeting* [67].

B) ***Extreme Programming* (XP):** Centra-se numa prática para melhora do desenvolvimento. Caracterizado por doze práticas descritas no *white book* de Kent Beck [8]. As doze práticas do XP são altamente sinérgicas e interdependentes, não sendo uma coleção aleatória de boas ideias como pode parecer em um primeiro momento. Após após tornar-se experiente com o XP 'padrão', pode-se optar por apagar ou alterar uma prática, mas esta personalização do XP deve acontecer somente [8]. As doze práticas são: pequenas *releases*, jogo de planejamento, refatoração, testes, programação em pares, ritmo sustentável, propriedade do código da equipe, padrão de codificação, *design* simples, metáforas, integração contínua, e cliente no local. A prática de refatoração é facilitada pelas práticas de programação em pares, *design* simples, propriedade coletiva, integração contínua e testes [8, 18].

C) **Kanban:** Propõe-se a ajudar uma equipe a melhorar sua forma de construção do software. Uma equipe que utiliza Kanban tem uma imagem clara de quais ações seus membros executam para construir software, como eles interagem com o resto da empresa, onde perdem por ineficiência e desnivelamento, e como melhorar ao longo do tempo com a remoção da causa principal deste desperdício [74]. Os princípios do Kanban são: visualização do fluxo de trabalho, limitação do trabalho em andamento, mensuração e gerenciamento do fluxo, tornando o fluxo explícito e melhorando a colaboração [4].

D) ***Lean software development*:** Em 2003, Mary e Tom Poppendiech publicaram o primeiro mapeamento dos valores Lean: eliminar o desperdício, ampliar o aprendizado, decidir o mais tarde possível, entregar o mais rápido possível, capacitar a equipe, construir integridade e ver "o todo"[62]. Esses valores ajudam a entrar na mentalidade de um pensamento enxuto, *lean*, e as ferramentas desse pensamento ajudam a equipe a identificar o desperdício e eliminá-lo [4].

E) ***Feature-Driven development* (FDD):** Jeff de Luca e Peter Coad apresentaram o FDD em 1997. O FDD combina o modelo dirigido ao desenvolvimento ágil com ênfase no desenvolvimento iterativo [31]. Esta é uma abordagem baseada em modelos que colocam ênfase na definição de um modelo global do sistema e uma lista de funcionalidades a serem inclusas no sistema antes de iniciar o esforço do projeto [16].

Dentre outras tantas diferenças entre o desenvolvimento tradicional e o desenvolvimento ágil, uma delas é a forma de coleta de requisitos e o tempo das atividades de ER [48], detalhadas na Seção 2.2. No desenvolvimento tradicional, os requisitos são levantados, analisados e especificados de forma linear e no início do projeto [60, 48, 73]. Já no desenvolvimento ágil, assume-se que os requisitos evoluem ao longo do tempo e que são descobertos durante todo o processo de desenvolvimento [9, 48, 64]. Assim, a ER no desenvolvimento ágil se torna uma atividade contínua. Desta forma, a comunicação informal e frequente entre os *stakeholders* de negócio e a equipe de software é a principal contribuição da ER ágil [11, 48, 64].

2.2 Engenharia de requisitos e seus problemas

Requisitos refletem as necessidades dos clientes para um sistema que serve uma determinada finalidade, como controlar um dispositivo ou selecionar informações. O processo de descobrir, analisar, documentar e verificar esses funções e suas restrições é chamado Engenharia de Requisitos (ER) [73]. O termo ER refere-se a todas as atividades do ciclo de vida relacionados com obtenção, definição e gerência dos requisitos [6].

O objetivo da ER é ajudar a saber o que construir antes do desenvolvimento do sistema começar, com o intuito de evitar retrabalho, o que implica em custo de tempo e dinheiro. Sendo assim, existem muitas técnicas disponíveis para garantir que os requisitos estejam completos, coerentes e relevantes no processo de ER [60, 73].

É comum que um ambiente de negócios, o qual motiva a necessidade de sistemas informatizados, esteja suscetível a constantes mutações. Isto torna-se um desafio para as abordagens tradicionais de desenvolvimento, tal como o modelo Cascata [11]. O desenvolvimento ágil propõe envolver as mudanças e acomodar melhor as necessidades diante das condições dos negócios atuais. A principal diferença entre o desenvolvimento tradicional diz respeito sobre como a ER ocorre, mais especificamente, quando o mesmo ocorre [53].

Quando *The Standish Group* pesquisou opiniões, com executivos da Tecnologia da Informação, sobre o fatores que impactam seus projetos. As três razões mais citadas, que auxiliam para o sucesso do projeto, são: o envolvimento do usuário, suporte de gestão executiva e uma declaração clara dos requisitos. Sem eles, a chance de falha aumenta drasticamente [40].

O mesmo grupo levantou razões que 'desafiam' um projeto, ou seja, projetos que foram concluídos, mas acima do orçamento ou acima da estimativa do tempo, ou que oferecem menor número de funcionalidades e funções originalmente especificadas. As três razões mais citadas foram [40]: falta de acesso ao usuário 12,8%, requisitos e especificações incompletas 12,3% e alterações de requisitos e especificações 11,8%. Já para os projetos que foram 'totalmente prejudicados', ou seja, projetos que foram cancelados em

algum momento durante o ciclo de desenvolvimento as seguintes razões foram citadas: requisitos incompletos 13,1%, falta de envolvimento com usuário 12,4% e falta de recursos 10,6%.

A Engenharia de Requisitos (ER) ágil não é centralizada em uma única fase antes do desenvolvimento. Ela está uniformemente distribuída ao longo do desenvolvimento do software [11]. A ER tradicional concentra-se em reunir todos os requisitos do software e preparar o documento de especificação de requisitos antes que se possa começar o *design* do sistema e enquanto a ER ágil acompanha mudanças nos requisitos, mesmo no final do ciclo de vida do desenvolvimento [53, 36].

As principais atividades que norteiam a ER são: elicitação, análise e negociação, documentação, validação e gerenciamento de requisitos [73]. Cada uma destas com as suas técnicas possuem as seguintes características:

1) Elicitação dos requisitos: Visa descobrir os requisitos e identificar fronteiras do sistema consultando os *stakeholders* (clientes, desenvolvedores, usuários finais). As fronteiras definem o escopo do sistema. Durante esta atividade, é essencial o entendimento do domínio da aplicação, das necessidades do negócio, das restrições do sistema, dos *stakeholders* e do problema a ser resolvido. Toda esta aquisição de conhecimento é fundamental para o correto desenvolvimento do sistema. As técnicas da elicitação mais utilizadas são: entrevistas, cenários/casos de uso, observação e análise social, grupo focado, *brainstorming*, e prototipagem [73].

2) Análise de requisitos: Objetiva verificar os requisitos quanto à sua necessidade (se é indispensável ao sistema), quanto à sua consistência (não deve ser contraditório), quanto à completude (nenhum serviço ou restrição esteja faltando), e quanto à realidade (realistas no contexto de custo e prazo do projeto). Os conflitos nos requisitos são resolvidos através da priorização e negociação com os *stakeholders* de negócio. Soluções para os problemas identificados são alinhados e um compromisso é firmado, considerando um conjunto de requisitos que foram acordados. Firesmith (2007) complementa refinando que é a verificação antecipada ao processo de desenvolvimento dos requisitos, para verificar se há qualidade suficiente para evitar consequências negativas resultantes de requisitos pobres [24]. As principais técnicas de análise e negociação são: sessões de *Joint Application Development* (JAD), priorização de requisitos e modelagem [73].

3) Especificação dos requisitos: A documentação de requisitos tem o objetivo de comunicar os requisitos entre os desenvolvedores e *stakeholders* de negócios sobre o sistema. Este documento é a base para avaliar produtos e processos subsequentes (projeto, teste, verificação e validação) e para controle de mudança. Um bom documento de requisitos não pode conter ambiguidades, tem que ser correto, entendível, consistente, conciso e realista.

Em alguns casos, a especificação dos requisitos pode fazer parte do contrato do projeto [73].

4) Validação de requisitos: Certifica que os requisitos são uma descrição aceitável do sistema a ser desenvolvido. As entradas para a atividade de validação são o documento de requisitos, os padrões e o conhecimento organizacional. As saídas são uma lista que contêm os problemas encontrados e as ações necessárias para resolver os problemas os mesmos. Firesmith (2007) ressalta que é uma tarefa imprescindível da ER. Ele afirma a importância de ter os *stakeholders* validando seus requisitos para garantir que os requisitos estão completos e corretos, especificando suas necessidades. Infelizmente, os requisitos nem sempre são devidamente validados pelos *stakeholders*. As técnicas usadas para validação de requisitos são revisão e teste de requisitos [73].

5) Gerenciamento dos requisitos: Objetiva capturar, armazenar, disseminar e gerenciar informação. Inclui todas as atividades preocupadas com mudanças, controle de versão e rastreamento de requisitos. Rastreamento de requisitos provê relacionamento entre requisitos, projeto e implementação de um sistema a fim de gerenciar suas mudanças [73]. Firesmith (2007) reportou que muitos projetos não geram adequadamente seus requisitos, pois os mesmos são armazenados em documentos de papel ou em planilhas simples. Tipos diferentes de requisitos também são armazenados separadamente, controlados por diferentes equipes, como a equipe de gerenciamento, a equipe de requisitos, e, especialmente as equipes de engenharia [24].

Por exemplo, Firesmith (2007) menciona que em muitos projetos os requisitos são documentados de forma incompleta, faltando ou documentando inadequadamente as tarefas, técnicas, funções e funcionalidades importantes. Contudo o processo de ER é frequentemente seguido de forma inconsistente [24]. Outro exemplo é o desafio para os engenheiros de requisitos de reconhecer que os clientes muitas vezes confundem os requisitos e metas do sistema [48]. As metas são objetivos de alto nível do negócio. Já um requisito é um objetivo que deve ser realizado pelo sistema.

Diante disso, faz-se importante representar os requisitos de forma que permita que o cliente e a equipe de projeto estabeleçam um entendimento comum sobre o funcionamento do sistema. Os requisitos podem ser representados através de linguagem natural ou em artefatos que os descrevam. Para isto, tem-se as seguintes opções:

- Funcionalidade: A IEEE classifica funcionalidade de software como característica distintiva de um item de software. É uma característica do software especificada na documentação de requisitos, como uma funcionalidade, restrições de desempenho, atributos, ou *design* [38]. É definida como um pedaço de software a ser entregue aos *stakeholders* de negócio, a fim de alcançar os objetivos do negócio [48, 14].

- Cenários: Cenário é uma sequência de interações que acontece sob certas condições, com a intenção de atingir o objetivo do ator primário, obtendo um resultado para este objetivo. As interações começam a partir de uma 'ação gatilho' e continuam até que o objetivo seja entregue ou abandonado, e o sistema completa as responsabilidades que tem em relação desta interação [17]. Os cenários podem ser particularmente úteis para adicionar detalhes na descrição do requisito. Eles são descrições de exemplos de interação. Cada cenário normalmente cobre um ou um pequeno número de possíveis interações. Diferentes cenários são desenvolvidos e eles provêm diferentes tipos de informações em diferentes níveis de detalhe sobre o sistema [73].

- Caso de Uso: Um caso de uso é uma descrição das possíveis sequências de interações entre o sistema e seus atores externos, relacionados a um determinado objetivo [17]. Podem ser documentados usando diagrama de caso de uso de alto nível. O conjunto de casos de uso representa todas as interações que descrevem os requisitos do sistema. Atores no processo podem ser humanos ou outros sistemas [73].

- *User story*: Uma *user story* é uma descrição escrita para planejar os detalhes sobre uma funcionalidade [18]. Estas substituem no ágil os requisitos que são tradicionalmente expressos como sentenças de requisitos ou casos de uso. Elas são a 'locomotiva' do desenvolvimento ágil [49]. A *user story* é uma definição de alto nível do requisito, a qual contêm informações suficientes para que os desenvolvedores possam produzir uma estimativa razoável do esforço para implementá-la [14, 16]. Assim, a funcionalidade é mais útil do ponto de vista do *stakeholder* de negócio e a história é mais útil do ponto de vista da equipe que entrega a funcionalidade [14].

- Épico: Um épico é uma grande *user story* que precisa ser dividida em *user stories* menores antes do início de uma iteração ágil [16, 71]. Não há nenhuma definição clara de como para fazer uma história épica, mas se uma história é tão grande que precisa de vários ciclos de desenvolvimento (*sprints*) para ser completada, normalmente ela é dividida em uma série de pequenas *user stories*.

- Critério de aceitação: Critérios de aceitação são as condições de satisfação para o sistema [49]. Estes permitem à equipe do projeto julgar se uma funcionalidade está implementada corretamente [29, 71].

Para lidar com a diversidade de tipos de requisitos, Sommerville (2009) sugere três níveis de abstração, quais sejam: *(i) Requisitos de usuário:* são declarações abstratas escritas em linguagem natural com acompanhamento de diagramas informais. Elas especificam quais serviços (funcionalidade do usuário) o sistema deve fornecer e suas restrições. Em muitas situações, *user stories* podem desempenhar o papel das necessidades dos utilizadores. *(ii) Requisitos de sistema:* são descrições detalhadas dos serviços e das respectivas

restrições. Requisitos do sistema são referidos como especificação funcional ou anexo técnico. Estes requisitos são derivados da análise das necessidades dos usuários. Eles agem como um contrato entre o cliente e o contratante, por isso, devem estar estruturados e precisos. Os casos de uso podem desempenhar o papel de requisitos do sistema em diversas situações. *(iii) Especificações de design de software:* surgem a partir da documentação de análise do projeto, utilizados na implementação pelos desenvolvedores. A especificação de *design* de software é derivada da análise dos requisitos do sistema.

A IEEE descreve as características de uma boa especificação de requisitos de software, listando oito medidas de qualidade, que envolve o requisito ser: correto, não ambíguo, completo, consistente, classificado por importância e/ou estabilidade, verificável, modificável, e rastreável [39]. No entanto sabendo da dificuldade de decidir exatamente o que construir no desenvolvimento de software [73], há a importância de ter uma boa percepção dos requisitos durante o processo de ER, na tentativa de mitigar problemas comumente enfrentados por equipes de software. Alguns dos problemas citados na literatura são listados a seguir e resumidos na Tabela 2.1:

1) **Identificação dos *stakeholders* do negócio:** Goguen (1993) reportou que em grandes organizações com departamentos rivais e uma hierarquia complexa pode não ser óbvio quem ou o que considerar 'o cliente' durante a ER [27]. Sharp *et al.* (1999) observaram que durante o processo de ER costuma-se dedicar pouca atenção à identificação dos *stakeholders*. Ainda Damian e Zowghi (2002) constataram que a dificuldade de identificação de *stakeholders* torna-se ainda mais crítica em ambientes de desenvolvimento distribuído. Sem a percepção presencial do ambiente onde o software em desenvolvimento será utilizado, a capacidade de descoberta das pessoas envolvidas fica prejudicada. Conflitos entre os *stakeholders* de negócio e suas exigências são comuns e quase inevitável. Além disso, os mesmo podem não querer comprometer ou priorizar seus requisitos quando ocorrem conflitos. Às vezes, *stakeholders* não sabem o que querem ou quais são suas reais necessidades e, portanto, são limitados em sua capacidade de apoiar a investigação de possíveis soluções [80].

Tabela 2.1 – Alguns problemas relacionados à Engenharia de Requisitos

	Problemas	**Referências**
1	Identificação dos *stakeholders* do negócio	[27][70][21] [80]
2	Dificuldade na comunicação	[27][3][12][33][37][21][34][80][11] [52]
3	Problemas da linguagem	[15][59][39][24][48][52][7]
4	Rastreamento	[65][24]
5	Volatilidade	[80][27][24]
6	Gestão do conhecimento	[15][12][45][37][79][24][52]
7	Diferença cultural	[12][37][21][52]

2) **Dificuldade na comunicação:** Desenvolvedores de sistemas podem frequentemente se beneficiar do contato direto com a equipe de requisitos, para descobrir as razões por trás das escolhas, para resolver inconsistências, para obter mais detalhes, etc. Infelizmente, muitos processos de desenvolvimento de sistemas não permitem esta comunicação contínua [27]. Dificuldades na comunicação advêm da natureza das notações de engenharia de software e metodologias, bem como as barreiras de comunicação causadas por fatores sociais e organizacionais [3]. A documentação formal dos requisitos não elimina a necessidade da comunicação frequente. Foi identificado que a comunicação intensa entre os desenvolvedores e clientes seria a prática mais importante da ER [11]. Comunicação face-a-face é a forma mais rica de comunicação. Quando pessoas se comunicam face-a-face, é possível se transmitir muito mais do que texto nas mensagens trocadas. Grande parte da mensagem está na linguagem corporal, tom de voz, e contexto [12]. Assim, equipes co-localizadas podem ampliar a credibilidade através de interações formais e informais, tornando a distância um fator que dificulta a construção de relações de credibilidade [12]. Damian e Zowghi (2002) complementam que o contato pessoal, o conhecimento das personalidades e valores são necessários para obter o engajamento, credibilidade e o espírito de equipe. Há ainda a dificuldade na articulação dos requisitos pelos *stakeholders*. Em alguns casos, isso pode ser resultado do analista e os *stakeholders* de negócio não compartilharem um entendimento comum dos conceitos e termos, ou o analista não estar familiarizado com o problema [80].

3) **Problemas da linguagem:** Christel (1992) reporta que durante a elicitação de requisitos os *stakeholders* de negócio nem sempre abordam as verdadeiras necessidades. Como consequência, tem-se requisitos ambíguos, incompletos, inconsistentes e até incorretos [15]. Firesmith (2007) observa que especificações ambíguas, não coesas, incompletas, inconsistentes, incorretas e obsoletas utilizam 'jargão' técnico ao invés da terminologia do usuário ou domínio do negócio/aplicação. Equipes variam muito no nível de conhecimento e tendem a descrever os requisitos de forma diferente [39]. A utilização de linguagem natural para especificação simplifica o entendimento por parte dos *stakeholders*, mas possibilita a ambiguidade na expressão de características e torna difícil o gerenciamento [59]. Problemas de linguagem natural resultam de ambiguidades e sensibilidade do contexto da linguagem (humana) natural. Problemas desta natureza são existentes em todos idiomas e para todos, não apenas para engenheiros de requisitos *et al.* [48]. Visto a dificuldade de se obter requisitos de boa qualidade, precisos e inequívocos [7], torna-se necessário haver uma linguagem de especificação comum ao pessoal técnico e não técnico do projeto, contribuindo com a clareza e redução de ambiguidades para especificação.

4) **Rastreamento:** O rastreamento dos requisitos refere-se a habilidade para descrever e acompanhar requisitos na concepção e desenvolvimento. É uma técnica fundamental no apoio às diversas atividades do projeto, assegurando que o sistema está coerente conforme

os requisitos [65]. Embora o valor do rastreio dos requisitos seja amplamente reconhecido, às vezes, até mesmo, obrigatório nos contratos e incluído nos métodos de ER, muitos requisitos ainda não são devidamente rastreados na prática [24].

5) **Volatilidade:** Os requisitos mudam ao longo do tempo. Então, o levantamento de requisitos deve ser feito de forma organizada, pois a volatilidade dos requisitos é inerente ao processo [15]. Durante o desenvolvimento do projeto, os *stakeholders* ficam familiarizados com o problema e o domínio da solução, assim os objetivos do sistema e as necessidades dos usuários são suscetíveis a mudança [80]. Lidar com as mudanças das necessidades do cliente é frequentemente exigido. Esta é uma consequência natural do aumento do entendimento pelo processo que envolve os requisitos. Alterações também podem surgir a partir de mudanças na organização do cliente ou seu contexto social (por exemplo, as leis fiscais, ou a sua concorrência) [27]. Assim, Firesmith (2007) ressalta que a maioria dos sistemas têm longos ciclos de desenvolvimento e ciclos de vida, então é óbvia a necessidade dos requisitos sofrerem alterações. Assim os sistemas necessitam acompanhar a evolução à medida que as necessidades de negócio mudam (por exemplo, com o advento de novos concorrentes e novas tecnologias) [24].

6) **Gestão do conhecimento:** Hanisch e Corbitt (2001) mencionam que todo volume de informações de requisitos precisa ser compartilhado com todos os *stakeholders* [37]. Uma equipe coesa exige experiências comuns e encontros face-a-face [12]. Os meios de comunicação utilizados (computadores, videoconferência, teleconferência) em geral não possuem mecanismos que permitam perceber interações não-verbais [45]. As dificuldades em manter *awareness*, coesão e coerência do conhecimento quando grupos diferentes tentam acessar as informações simultaneamente influenciam na gestão do conhecimento [52]. Os requisitos podem ser confusos, devido à quantidade de informações. Se os limites do sistema estão mal definidos ou os *stakeholders* de negócio especificam detalhes técnicos desnecessários podem confundir, ao invés de clarificar os objetivos do sistema [15]. Firesmith (2007) relata que muitos requisitos especificados não são realmente obrigatórios. Muitos são restrições de arquitetura, *design*, implementação e instalação/configuração que são desnecessariamente especificados como requisitos. Assim, acabam confundindo a implementação com especificação inadequada dos requisitos, de como construir o sistema ao invés de especificar o que sistema deve fazer ou como o sistema deve fazer [24].

7) **Diferença cultural:** Carmel (1999) relata que trabalhadores que realizam seus trabalhos em um único local, em uma cultura e com único idioma, tem a comunicação facilitada. Quando as tarefas são realizadas em locais dispersos no mundo, essa diversidade pode gerar choques culturais e mal-entendidos. Apesar do computador e da cultura do negócio internacional fortemente moldada por normas norte-americanas, as diferenças culturais que existem podem criar problemas para as equipes globais [12]. Damian e Zowghi (2002) observam que dificuldades causadas pelas diferenças culturais são exacerbadas diante da

distribuição geográfica [21]. López (2009) corrobora esta observação ao relatar que existem dificuldades decorrentes da interação entre grupos nos quais as pessoas têm origens culturais distintas. No desenvolvimento distribuído as barreiras éticas influenciam a expressão e priorização dos requisitos [52]. Assim, diferenças do idioma dos *stakeholders* e da cultura nacional afetam a colaboração global, e, também, as diferenças da cultura organizacional e funcional [37]. Entretanto, as diferenças culturais não advêm apenas da distância física. Existe também a distância entre os diferentes departamentos funcionais da organização (marketing, vendas, TI, etc) devido as imposições inerentes a hierarquias criadas para que uma organização possa ser funcional. Isto tem impacto significativo na obtenção do entendimento comum e na negociação dos requisitos [37].

O *Behavior-Driven development* (BDD) surge para auxiliar equipes a construir e entregar software de valor e qualidade de forma rápida [71]. Este objetiva promover o entendimento comum do domínio do negócio entre os *stakeholders* de negócio e a equipe do projeto, além de apoiar a automação de outras atividades do ciclo de desenvolvimento e manter a documentação atualizada para todos os envolvidos [23, 71]. Entende-se então que o BDD pode vir a auxiliar a mitigar alguns problemas observados da ER e no desenvolvimento de software como um todo.

Segundo Solis e Wang (2011), BDD engaja diferentes etapas para o desenvolvimento de software, como a etapa de planejamento dos comportamentos (correspondendo aos objetivos de negócios) e a etapa de análise dos objetivos de negócios; e fragmenta o conjunto de funcionalidades, capturando o comportamento do sistema [72]. Assim, observa-se que o BDD intrinsecamente está envolvido no processo de ER, em especial em relação às atividades de elicitação e análise dos requisitos. Bjarnason *et al.* (2015) corrobora esta observação e evidencia que o BDD incorpora aspectos de análise e documentação de requisitos, comunicação e testes de aceitação automatizados [22].

2.3 BDD e conceitos relacionados

Behavior-Driven Development, ou desenvolvimento orientado ao comportamento, foi inicialmente proposto por Dan North [58]. O BDD envolve práticas ágeis e foi projetado para torná-las mais acessíveis e eficazes para as equipes ágeis na entrega de um software. Com o tempo, o BDD cresceu para abranger o panorama de análise em ágil e testes de aceitação automatizados [58]. Ele também incorpora a definição de funcionalidades baseadas em especificação por exemplos, o denominado *Specification by Example*, proposto por Gojko Adzik [2].

O BDD reflete um conjunto de práticas de engenharia de software projetado para auxiliar equipes a construir e entregar software de maior valor e qualidade de forma mais

rápida [71]. Descreve como o sistema deve se comportar com linguagem ubíqua específica do domínio, facilmente compartilhada entre os *stakeholders* [28, 71]. Visa promover um entendimento comum do domínio do negócio entre os *stakeholders* de negócio e a equipe de desenvolvimento [23, 71], devido a descrever o comportamento do sistema da perspectiva dos *stakeholders* de todos os níveis [14], conforme ilustrado na Figura 2.1.

Ainda, o BDD objetiva promover a colaboração e descoberta de requisitos através de exemplos. O objetivo não é a geração de critérios de aceite ou arquivos de funcionalidades, não é automação e não é teste; embora o BDD contenha elementos de todos [71]. É uma combinação de abordagens como linguagem ubíqua, *Test-Driven Development* (TDD) e testes de aceitação automatizados, otimizando as conexões dessas abordagens para tirar o máximo proveito de cada uma [72]. Sua grande contribuição é esta integração, que permite que todos os envolvidos em um projeto tenham acesso a qualquer aspecto do sistema (requisito, código, testes) definido bem como resguardar que mudanças em qualquer um destes aspectos seja automaticamente refletido nos demais. O mesmo também promove a conexão de código para os requisitos, oferecendo um ambiente de gerenciamento do progresso da cobertura do desenvolvimento e dos requisitos durante todo o ciclo de vida [71].

Specification by example (SBE), especificação por exemplo, descreve o conjunto de práticas que emergiu do uso de exemplos e conversas para descobrir e descrever os requisitos [2]. Adzic escolheu este termo como o mais representativo para se referir a estas práticas. O uso de conversas e exemplos para especificar o comportamento esperado do sistema é cerne do BDD [71].

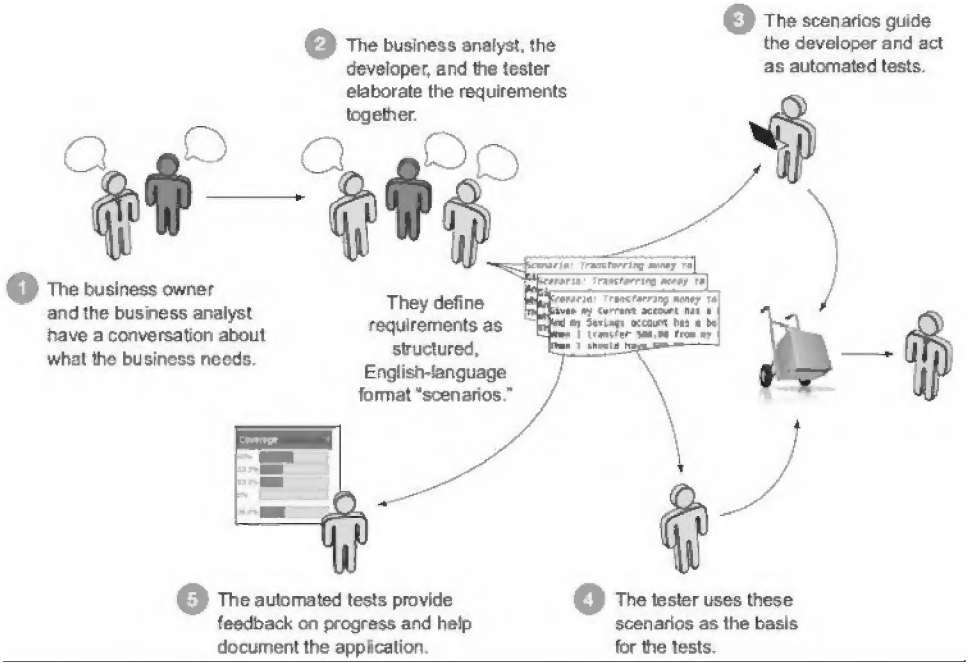


Figura 2.1 – BDD e o uso de conversas sobre exemplos (Fonte: [71])

Evans (2003) com *Domain-Driven Design* (DDD), ou projeto orientado a domínio, promove a linguagem ubíqua, uma linguagem universal, que promove a comunicação através de uma sintaxe própria, seguida de vários conceitos. Nesta, são utilizados termos que fazem parte das conversas entre os *stakeholders* de negócio e as equipes do projeto. Todos devem usar esses termos tanto na linguagem quanto no código escrito [23]. Esta linguagem ubíqua habilita os *stakeholders* a trabalharem com os mesmos termos, na tentativa de mitigar a ambiguidade [72]. Assim o BDD utiliza desta linguagem com o intuito de melhorar o entendimento entre todos os *stakeholders*.

Smart (2014) recomenda que BDD não seja utilizado em organizações que haja desenvolvimento 'em silo'. Ou seja, onde as especificações detalhadas sejam escritas por analistas de negócios e, em seguida, transferidas para as equipes de desenvolvimento, e depois para equipes de teste, verificar a qualidade do código.

North (2006) afirma que o BDD surgiu em resposta aos problemas de TDD. TDD é uma abordagem baseada em ciclos curtos de desenvolvimento e práticas de escrita de testes automatizados antes de escrever código funcional, refatoração e integração contínua [42]. Enquanto o ATDD [32, 46] dirigido por testes de aceitação, os quais são utilizados para representar as necessidades dos *stakeholders* [72].

Ainda, o BDD promove a escrita de especificações executáveis que orientam a implementação em todos os níveis de projeto. As metas do negócio definidas por analistas e *stakeholders* do negócio são chamadas especificações de alto nível. BDD de baixo nível, estente prática estabelecidas por TDD, é uma continuação natural dos princípios do BDD que são aplicados para os especificações de alto nível [71], conforme indica a Figura 2.2.

O BDD pode facilmente caber em pequenas iterações ágeis ou processo baseado em fluxo, de modo que a informação é produzida antecipadamente ao momento do desenvolvimento [2]. Os testes para BDD são escritos com estrutura de especificações

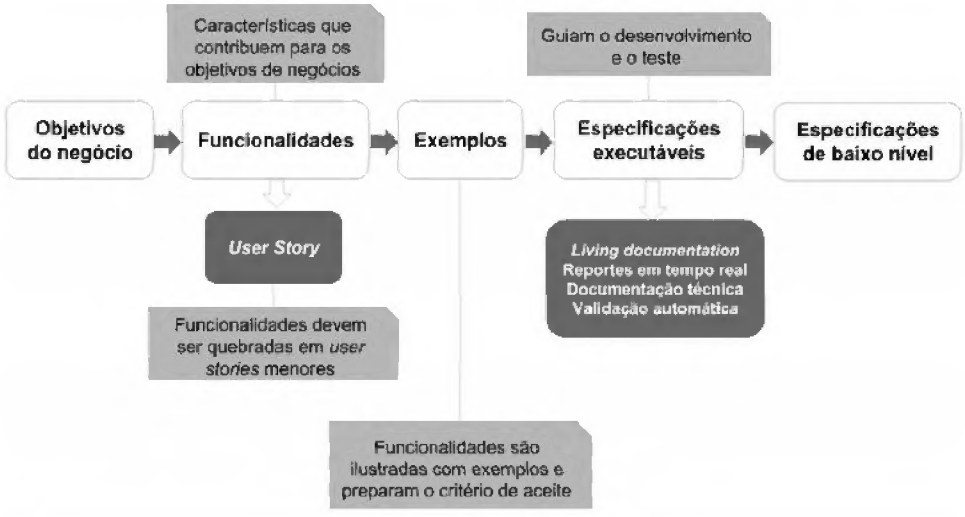


Figura 2.2 – Das necessidades de negócios até especificações executáveis (Fonte: [71])

executáveis, chamadas requisitos, e não como testes de unidade. Os requisitos se concentram no comportamento da aplicação, utilizando testes simples para expressar e verificar o comportamento do sistema [71]. North também observa que os testes escritos desta forma são mais fáceis de manter, devido a clareza da intenção [58].

Desta forma, o BDD inicia na identificação dos objetivos de negócio e funcionalidades de software que irão cobrir esses objetivos. Colaborando com o cliente, os praticantes de BDD usam exemplos concretos para ilustrar essas funcionalidades. As funcionalidades podem ser divididas em pequenos pedaços, chamadas *user stories*. Os exemplos definidos podem ser automatizados sob a forma de especificações executáveis que seguem um formato estruturado chamado 'cenários'. A Figura 2.2 ilustra a decomposição dos objetivos de negócio em especificações executáveis. É importante destacar que o BDD também pode ser utilizado com sucesso para descobrir e verificar requisitos não funcionais [71].

Os principais princípios do BDD recentemente divulgados por Smart [71] e discutido por Solis e Wang [72] são: *i*) Descrever o comportamento, não especificar soluções; *ii*) Descobrir o comportamento que entregue valor de negócio; *iii*) Usar conversas e exemplos para explorar o que um sistema deve fazer. Detalhando estes princípios, tem-se [71]:

- Foco nas funcionalidades: Há um constante foco em dar valor ao negócio. É por isso que em vez de tentar reunir todos os requisitos de uma única vez, as equipes que utilizam BDD participam de conversas com os usuários finais e outros *stakeholders* para construir progressivamente um entendimento comum de quais funcionalidades eles devem construir.

- Trabalho em equipe para especificar as funcionalidades: Os analistas de negócios, desenvolvedores e testadores trabalham em conjunto com os usuários finais para definir e especificar as funcionalidades, e os membros da equipe constroem ideias a partir de sua experiência individual e '*know-how*'.

- Abraça a incerteza: Uma maneira efetiva de saber se os usuários gostaram de uma funcionalidade é construí-la e mostrar-lhes o mais rapidamente possível.

- Ilustra as funcionalidades com exemplos concretos: Ao implementar uma funcionalidade a equipe que pratica BDD trabalha em conjunto com os usuários e outros *stakeholders* para definir histórias e cenários.

- Não escreve testes unitários, escreve especificações de baixo nível: O BDD apoia desenvolvedores para escrita de código confiável, sustentável e documentado. As especificações executáveis são similares aos testes unitários convencionais, mas são escritas de forma a comunicar a intenção do código e fornecer exemplo real de como o código deve ser usado. Escrever especificações executáveis de baixo nível (*low-level*) é escrever documentação detalhada do projeto.

- Entrega documentação atualizada: A documentação está sempre atualizada e requer pouca ou nenhuma manutenção manual. Os relatórios produzidos pelas especificações executáveis não são relatórios técnicos para desenvolvedores, mas efetivamente se tornam uma documentação do produto para toda equipe, expressa com vocabulário familiar aos usuários.
- Usa a documentação atualizada para apoiar o trabalho de manutenção: Um projeto desenvolvido usando as práticas de BDD também é significativamente mais fácil e menos caro para manter, pelos benefícios da documentação estar sempre atualizada e especificações executáveis não acabarem ao final do projeto.

Com isto, o BDD apoia a conexão de código para os requisitos (especificações executáveis) que, quando executadas, reportam se alguma alteração foi realizada nas especificações, para que todos integrantes da equipe se mantenham atualizados. Para tal, existem *frameworks* que apoiam a realização do BDD, dentre eles, tem-se: JBehave [57], Cucumber [35], RSpec [5], Behat [47], entre outros. Os *frameworks* apoiam alguns dos princípios acima, conforme resumido na Tabela 2.2. No entanto, Solis e Wang (2011) indicam que não há *framework* que suporte todo processo de decomposição iterativo [72].

Existem ainda *frameworks* de BDD usados para teste de unidade, como RSpec, NSpec, e Jasmine, que tem ênfase na escrita especificações executáveis de baixo nível. Já no estilo de codificação descritiva há ferramentas de teste unitário como JUnit e NUnit.

Smart [71] lista os seguintes benefícios na adoção BDD: Redução de gastos, concentrando o esforço do desenvolvimento na descoberta e entrega de funcionalidades que proverão valor ao negócio; Redução dos custos, consequência da redução dos gastos, em dinheiro, para o projeto; Mudança fáceis e seguras, com *living documentation*, ou documentação viva, gerada das especificações executáveis que utilizam termos que os *stakeholders* estão familiarizados; e, *Releases* mais rápidas, com testes automatizados também aceleram o ciclo de *release* consideravelmente). Há ainda algumas dificuldades do BDD, quais sejam: Requer alto engajamento e colaboração do cliente (com base em conversas e *feedback*); Funciona melhor em contexto ágil ou iterativo; e, não funciona bem em um silo

Apoio de características BDD		Família xBehave		Família xSpec		StoryQ	Cucumber	SpecFlow
		JBehave	NBehave	RSpec	MSpec			
Definição de linguagem ubíqua		×	×	×	×	×	×	×
Processo iterativo de decomposição		×	×	×	×	×	×	×
Edição de texto baseado em	Formato de User story	√	√	×	×	×	√	√
	Formato de Cenário	√	√	×	×	×	√	√
Teste de aceitação automatizado com mapeamento de regras		√	√	×	×	×	√	√
Codigo de especificação orientado a comportamento legível		√	√	√	√	√	×	√
Dirigido a comportamento em diferentes fases	Planejamento	×	×	×	×	×	×	×
	Análise	√	√	×	×	×	√	√
	Implementação	√	√	√	√	√	×	√

Nota: √ - O *framework* apoia a característica de BDD, × - O *framework* não apoia a característica de BDD

Tabela 2.2 – As características do BDD em seus *frameworks* de apoio (Fonte: [72]).

Tabela 2.3 – Ilustração de um exemplo usando *Gherkin* (Adaptado de: [71])

Funcionalidade: Comprando livros com o cartão da livraria
No intuito de comprar livros **Como um** cliente loja de livros **Eu quero** pagar meus livros escolhidos
Cenário 1: Pagando com saldo positivo
Dado que meu cartão tem um saldo de R$ 300,00 E meu cartão da loja de 15% de desconto
Quando eu compro 100,00 do meu carrinho
Então eu deveria pagar R$ 85,00 E eu deveria ter R$ 215,00 em meu cartão da loja de livros
Cenário 2: Pagando sem saldo suficiente
Dado que meu cartão tem um saldo de R$ 50,00 E meu cartão da loja de 15% de desconto
Quando eu compro 100,00 do meu carrinho
Então eu deveria receber uma mensagem 'saldo insuficiente' E eu deveria continuar com R$ 50,00 em meu cartão da loja de livros

(em muitas organizações grandes, uma abordagem de desenvolvimento em silos ainda é a regra).

Gherkin em BDD

A Tabela 2.3 ilustra a notação definida para escrever um exemplo que representa uma determinada funcionalidade. A notação Dado que-Quando-Então é chamada de *Gherkin*. Com ferramentas apropriadas, cenários escritos desta forma podem ser automatizados e executados automaticamente [71].

Os exemplos em *Gherkin* de uma determinada funcionalidade são agrupados em um único arquivo de texto chamado *feature file* (arquivo de funcionalidade). O *feature file* contém uma breve descrição da funcionalidade, seguido por um número de cenários, ou exemplos formalizados de como a funcionalidade funciona [2, 71, 56].

A narrativa da funcionalidade segue o formato proposto por Chris Matts [55] no contexto de *Feature Injection* (Injeção de Funcionalidades) afirmando: qual é o valor de negócio que a funcionalidade pretende proporcionar, quem precisa da funcionalidade que está sendo proposta, e qual é a ação a ser feita.

Cada cenário *Gherkin* é composto por um conjunto de cláusulas pré-definidas, Dado que 'contexto'...Quando 'evento'...Então 'resultado'. E e Mas são outras duas cláusulas comumente também utilizadas. Mais detalhadamente:

- **Dado que** descreve as pré-condições para o cenário e prepara o ambiente de teste. Serve para colocar o sistema no estado conhecido antes que o usuário (ou sistema externo) comece a interagir com o sistema.
- **Quando** descreve a ação-chave que o usuário executa, ou estado de transição.
- **Então** é usado para descrever os resultados esperados. As observações devem estar relacionados com o valor de negócios/benefício na descrição da funcionalidade.
- **E** e **Mas** são cláusulas adicionais usados para unir as cláusulas anteriores e proporcionar uma forma mais legível para especificar a funcionalidade.

Specification by Example

Acceptance-Test-Driven Development (ATDD), ou desenvolvimento orientado aos testes de aceitação, é um sinônimo amplamente utilizado para a *Specification by Example* (SBE), ou Especificação por exemplo. A prática já existe de várias formas desde o início da década de 90. Kent Beck e Martin Fowler mencionaram o conceito em 2000, embora observaram a dificuldade de implementar critérios de aceitação na forma de testes de unidade convencionais no início do projeto [71].

SBE requer uma participação ativa dos *stakeholders* de negócio e da equipe de desenvolvimento, incluindo desenvolvedores, testadores e analistas [2]. ATDD envolve a colaboração com *stakeholders* nos testes de aceitação antes da escrita do código [14]. *Acceptance Test-Driven Planning* (ATDP), ou planejamento guiado por teste de aceitação, é a ideia de definir critérios de aceitação para uma funcionalidade para conduzir melhor a estimativa [71].

A diferença entre ATDD e ATDP é o que ATDD especifica a escrita dos testes de aceitação antes do código, mas não especifica quando deve escrevê-los. Já o ATDP especifica quando os testes de aceitação são escritos, se durante ou antes, mas não depois do desenvolvimento. É uma reunião do planejamento da iteração. O mesmo considera o critério de aceitação na estimativa, a qual melhora a habilidade de planejamento da iteração [14].

Verificar critérios de aceitação manualmente para cada alteração de código pode ser demorado e ineficiente. Isto poderia retardar o *feedback*, o qual deixaria o processo de desenvolvimento lento. Sempre que possível, as equipes transformam esses critérios de aceitação em testes de aceitação automatizados ou, mais precisamente, especificações executáveis [71]. No SBE, os requisitos, as especificações e os testes de aceitação funcionais são a mesma coisa [2].

Os padrões de processo-chave do SBE são derivados do escopo de metas do negócio, especificação colaborativa, ilustração de especificações usando exemplos, refi-

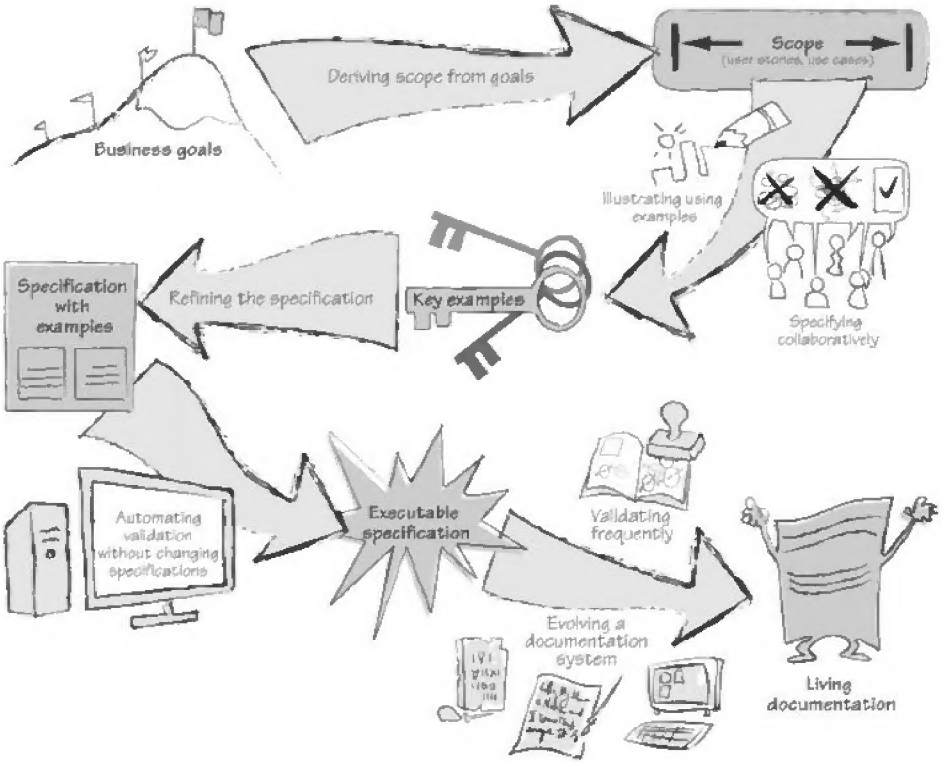


Figura 2.3 – Processo padrão do SBE (Fonte: [2])

namento das especificações, automação da validação sem alterações das especificações, validação do sistema frequentemente, e o envolvimento da documentação atualizada (*living documentation*), conforme ilustrado na Figura 2.3 [2]. O SBE tem as seguintes características:

- Derivando escopo das metas: As equipes começam a delinear o objetivo de negócio com o cliente. Em seguida, os membros colaboram para definir o escopo que permita atingir esse objetivo. A equipe trabalha com os usuários do negócio para determinar a solução mais barata, mais rápida e mais fácil de entregar ou manter do que a que os usuários de negócio chegariam por conta própria.
- Especificação colaborativa: Permite aproveitar o conhecimento e a experiência de toda a equipe. Isto também cria uma propriedade coletiva de especificações, fazendo com que todos fiquem mais envolvidos no processo de entrega.
- Ilustração de especificações usando exemplos: Exemplos chaves definem efetivamente o que o software precisa fazer.
- Refinamento das especificações: Remove informações irrelevantes e criam um contexto concreto e preciso para desenvolvimento e teste. Eles definem o alvo com a quantidade certa de detalhes para implementação e verificação. Exemplos refinados podem ser utilizados como critérios de aceitação para a entrega.

- Automação da validação sem alterações das especificações: Uma especificação com exemplos automatizados, compreensível e acessível a todos os membros da equipe, se torna uma especificação executável. Pode ser usada como um alvo para o desenvolvimento e a verificação do que foi acordado para o sistema, e pode ser usado esse mesmo documento para obter esclarecimentos de usuário de negócio.
- Documentação atualizada: Todas as especificações para todas as funcionalidades implementadas serão validadas frequentemente, a maioria por processo de compilação automatizado, ajudando a evitar problemas de regressão funcional, enquanto que as especificações estiverem atualizadas.

De acordo com Adzic (2011), o SBE incentiva pequenos *workshops*, conhecido como reunião dos 'Três amigos', envolvendo um desenvolvedor, um testador e um analista de negócios. Frequentemente demonstra-se suficiente para obter um bom *feedback* de diferentes perspectivas.

Há o incentivo à escrita em pares, em que os analistas fornecem o comportamento esperado e os desenvolvedores encontram a melhor forma de escrever os testes para serem automatizáveis, se encaixando na documentação atualizada. Com o intuito de obter melhor resultado, o *stakeholder* do negócio deve se envolver, colaborando durante as especificações [2].

No SBE, equilíbrio entre o trabalho realizado na preparação e o trabalho realizado durante a colaboração depende de vários fatores, como, maturidade do produto, nível de domínio do conhecimento da equipe do projeto, complexidade da solicitação de alteração, gargalos do processo, e disponibilidade de usuários de negócio [2].

3. METODOLOGIA DE PESQUISA

Neste capítulo é apresentada a metodologia utilizada para realizar a pesquisa apresentada neste estudo. A Seção 3.1 apresenta as questões de pesquisa que delinearam o estudo. A Seção 3.2 apresenta o desenho de pesquisa contendo as respectivas etapas que esta pesquisa foi organizada. A Seção 3.3 apresenta uma discussão sobre os aspectos metodológicos abordados e como os mesmos se relacionam com este estudo. E, a Seção 3.4, apresenta o detalhamento do Estudo de Campo.

3.1 Questões de Pesquisa

As questões de pesquisa (QP) que nortearam este estudo de natureza empírica bem como a discussão da contribuição do mesmo foram as seguintes:

- QP1. Como o BDD é usado durante todo ciclo de vida do projeto?
- QP2. Como o BDD apoia a engenharia de requisitos?
- QP3. Quais são os benefícios, dificuldades e recomendações em relação a adoção de BDD?

Diante dos objetivos deste estudo o mapeamento com QPs que delinearam este estudo está ilustrado na Tabela 3.1.

Tabela 3.1 – Mapeamento QP e Objetivos

QP	Objetivo
QP1	Identificar como o BDD é utilizado na prática
QP1 e QP2	Identificar como BDD apoia a engenharia de requisitos
QP1 e QP3	Identificar quais problemas da ER são mitigados pelo uso do BDD

3.2 Desenho da pesquisa

Foi conduzido um estudo de campo exploratório com o intuito de investigar os aspectos de interesse quanto à adoção do BDD. Esses aspectos estão relacionados ao ciclo do BDD, como por exemplo, as atividades realizadas, os integrantes da equipe que estão envolvidos, a definição dos cenários, etc. Também, quais os benefícios e dificuldades percebidas, levando-se ao entendimento das vantagens da adoção do mesmo e como este resolve problemas de ER e desenvolvimento de software como um todo.

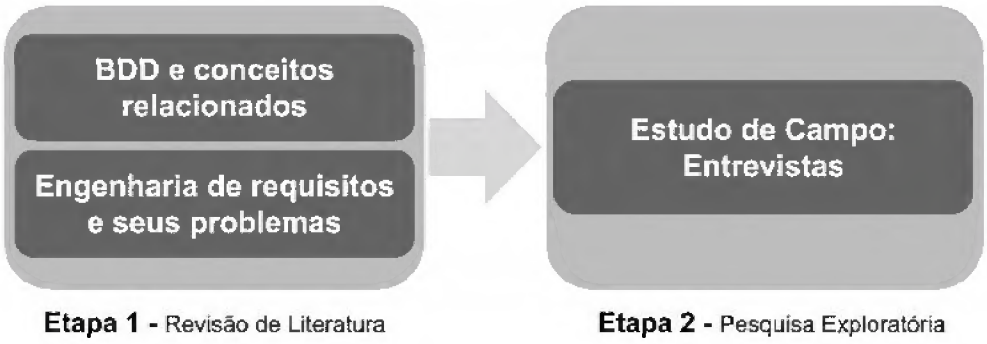


Figura 3.1 – Desenho da Pesquisa

Com isto, este estudo foi organizado em duas grandes etapas, conforme apresentado na Figura 3.1.

• **Etapa 1 - Revisão de Literatura:** Creswell afirma que revisão de literatura significa localizar e/ou resumir os estudos em determinado tópico. Ele também observa que não há uma única maneira de realizar uma revisão da literatura [20]. Na Engenharia de Software Empírica, adota-se visões sistemáticas da literatura [43], mapeamento sistemático [61], ou simplesmente um levantamento não estruturado da literatura como um todo, conforme o pesquisador identifica a necessidade de buscar conhecimento como a ponte ao desenvolvimento do seu trabalho. Assim, a revisão não estruturada da literatura teve como objetivo construir a base teórica desta pesquisa, apoiando o desenvolvimento da pesquisa como um todo, mais especificamente a revisão de literatura teve o objetivo de entender os conceitos inerentes a este estudo, observando estudos que compõe as metodologias ágeis, o BDD e a engenharia de requisitos. Identificou-se apenas alguns estudos estudos científicos sobre BDD. No entanto, descobriu-se termos de estudos paralelos ao BDD, como *Specification by Example*, ATDD, entre outros, apresentados na Seção 2.3. Assim, identificou-se que estudos empíricos sobre BDD, como o que foi conduzido nesta pesquisa de Mestrado, ainda não foram encontradas.

• **Etapa 2 - Pesquisa Exploratória:** O estudo exploratório foi conduzido baseado na abordagem qualitativa (vide Seção 3.3) através de um estudo de campo baseado em entrevistas. Esta fase visou levantar dados gerais sobre o uso do BDD na prática, sob o ponto de vista de profissionais da Tecnologia da Informação que utilizavam desta abordagem no dia a dia em seus projetos. Para isto, foram conduzidas entrevistas semi estruturadas [68] com perguntas abertas. Os dados provenientes das entrevistas foram analisados utilizando métodos de *Grounded Theory* [75]. A coleta de dados era dinâmica, ou seja, a definição dos próximos entrevistados era feita de acordo com a teoria emergente (theoretical sampling [25]). Ao final da análise de dados, pode-se identificar como BDD é adotado pelos entrevistados, quem são os integrantes envolvidos, quais ferramentas e metodologias são

utilizadas comumente ao BDD, quais os benefícios e dificuldades no uso de BDD. Cada uma das fases da pesquisa é detalhada a seguir.

3.3 Pesquisa qualitativa e Métodos coletados

O termo 'pesquisa qualitativa' denota um tipo de pesquisa que produz resultados que não resultam de procedimentos estatísticos ou outros que se quantificam [76]. Este refere-se a pesquisa sobre a vida das pessoas, experiências vividas, comportamentos, emoções, e sentimentos, bem como funcionamento organizacional, movimentos sociais, fenômenos culturais, e interações entre povos [76]. Assim, a pesquisa qualitativa esta envolvida na interpretação dos assuntos tratados, estudando as coisas em suas configurações naturais na tentativa de interpretar o fenômeno conforme relatos trazidos pelas pessoas participantes do mesmo [77].

Segundo Seaman (2008), a vantagem deste tipo de pesquisa está em forçar o pesquisador a se aprofundar na complexidade do problema ao invés de abstraí-la. Assim, os dados são mais ricos e informativos, ajudando a responder questões que envolvem variáveis que são difíceis de quantificar, mas essenciais quando se trata de atividades com forte influência de fatores humanos [68].

De acordo com Creswell (2007), pesquisa qualitativa começa com suposições, uma visão de mundo e o estudo de problemas que investigam os significados individuais ou grupos de um problema social ou humano. Para estudar este problema, os pesquisadores qualitativos usam a abordagem qualitativa para investigação, a coleta de dados em um ambiente natural para as pessoas, e a análise de dados que induzem e estabelecem padrões ou temas. O documento final escrito ou apresentado inclui vozes, ou transcrições, dos participantes, reflexões dos pesquisadores e uma descrição complexa e uma interpretação do problema. Ainda, se estende a literatura ou sinaliza uma chamada para a ação [19].

Diante deste estudo de natureza empírica, tem-se um estudo qualitativo, visto que se buscou identificar como o BDD é realizado na prática e como o mesmo auxilia a ER.

Método de coleta de dados

Um estudo de campo se caracteriza como o estudo dos praticantes reais de uma determinada atividade e de como os mesmos resolvem problemas reais. Assim um dos métodos de coleta de dados comumente utilizado é a condução de entrevistas, o método utilizado nesta pesquisa. Entrevistas são utilizadas por pesquisadores com o intuito de entender informações gerais sobre processos, conhecimento pessoal, entre outros [41].

Há três tipos de categorias para entrevistas, são elas [68]: estruturadas, não estruturadas e semi-estruturadas. Nas estruturadas, o pesquisador tem objetivos muito específicos e faz com que as perguntas sejam muito específicas, para que o foco esteja no tipo de informações que ele procura. Já nas entrevistas não estruturadas, o objetivo é coletar o máximo de informações possíveis. Acontece quando se deseja coletar o máximo sobre um certo tópico de forma ampla. As entrevistas semi-estruturadas, compreendem um misto de perguntas específicas fechadas ou abertas, permite elicitar os tipos de informações de interesse do pesquisador e também aqueles inesperados [68]. Como se pode observar o que varia no tipo das entrevistas é o nível de controle que o pesquisador tem e o que se deseja saber sobre o tópico em pesquisa.

Questões abertas permite a investigação em profundidade de respostas sobre as experiências pessoais, percepções, opiniões, sentimentos e conhecimento do respondente. Os dados consistem em citações literais com um contexto suficiente para ser interpretável [77]. Desta forma, este estudo seguiu entrevistas semi-estruturadas com perguntas abertas com o intuito de buscar mais detalhes sobre o fenômeno estudado.

Método de análise de dados

O termo *Grounded Theory* (GT), foi introduzido por Glaser e Strauss como a descoberta da teoria dos dados sistematicamente obtidos e analisados na pesquisa social [25]. O *Grounded Theory Institute*, coordenado por Glaser, um dos fundadores de GT, define-o da seguinte forma:

> *"Toda a investigação é 'fundamentada' nos dados, mas poucos estudos produzem uma 'teoria fundamentada'. GT é uma metodologia indutiva. Embora muitos chamem GT de método qualitativo, não é. É um método geral. É a geração sistemática da teoria da pesquisa sistemática. É um conjunto de rigorosos procedimentos de investigação que levam ao surgimento de categorias conceituais. Estes conceitos/categorias estão relacionados uns aos outros como uma explicação teórica das ações que continuamente resolvem a principal preocupação dos participantes na área. GT pode ser utilizado tanto com dados qualitativos ou quantitativos" [26].*

Strauss e Corbin (1990) definem GT, ou Teoria Fundamentada em Dados, como uma metodologia geral, uma maneira de pensar e conceituar dados [75]. Creswell (2007) reforça que GT provê um procedimento para o desenvolvimento de categorias de informação (codificação aberta), interligando as categorias (codificação axial) e a construção de uma 'história', que liga as categorias (codificação seletiva), terminando com um discursivo conjunto teóricos de proposições [19]. Desta forma, o GT consiste em 3 estágios

Figura 3.2 – Análise dos dados

de codificação [75]: codificação aberta, codificação axial, e codificação seletiva, conforme detalhado a seguir:

- **Codificação aberta:** Trata-se de obtenção de dados (por exemplo, transcrições de entrevistas) e segmentação em categorias de informação. A codificação aberta inclui comparação entre informações em termos de similaridade e diferenças, dando rótulos conceituais as informações, e agrupando esses conceitos em categorias [75].
- **Codificação axial:** Pesquisadores criam categorias com suas subcategorias, testam as relações com os dados [75]. Neste sentido, Creswell (2007) sugere que uma estrutura final possa ser incorporada através de um 'diagrama lógico', o qual o pesquisador apresenta a teoria de forma visual. Os elementos desta estrutura são identificadas pelo pesquisador na fase de codificação axial. A 'história' na codificação axial é uma versão narrativa disto [19].
- **Codificação seletiva:** Esta é a fase final de codificação da informação. O pesquisador captura o fenômeno central e sistematicamente relaciona com outras categorias, validando as relações e preenchendo as categorias que necessitam de refinamento e desenvolvimento [75]. É possível o desenvolvimento da 'história' para narrar as categorias e apresentar as inter-relações [19].

Charmaz (2006) reforça o processo de codificação inicial, o qual é semelhante a codificação aberta, em que o pesquisador desenvolve categorias de informações. A codificação focada é um processo destinado a estreitar os códigos iniciais até códigos importantes e frequentes. A codificação teórica é um processo utilizado para encontrar relacionamentos entre códigos e categorias; tem o potencial de dar origem a uma teoria [13].

Desta forma, este estudo utilizou GT e segue o fluxo apresentado na Figura 3.2. As setas da figura representam a comparação constante entre os casos (entrevistas por entrevistado), os dados obtidos e a teoria. Isto com o intuito de evitar o viés, que é uma ameaça a natureza deste tipo de estudo.

3.4 Detalhamento do Estudo de Campo

O objetivo deste estudo de campo foi identificar como o BDD é realizado na prática, identificando como o mesmo pode auxiliar a engenharia de requisitos (ER) e a resolução dos problemas comuns ao desenvolvimento de software. Para isto, observou-se como a abordagem é adotada pelos entrevistados, quem são os integrantes envolvidos, quais ferramentas e metodologias são utilizadas comumente ao BDD, quais os desafios enfrentados, benefícios observados e recomendações do BDD, entre outros aspectos. Diante disto, a Figura 3.3 apresenta os principais passos adotados para a preparação e realização do estudo de campo desta pesquisa (Etapa 2 da Figura 3.1). Estes passos estão organizados em três grandes etapas, conforme apresentado em detalhes a seguir.

Etapa 2.1 - Planejamento do Estudo de Campo

Esta etapa envolveu a elaboração e validação do roteiro de entrevistas, bem como a realização de 2 entrevistas piloto para refinamento do roteiro e a identificação de potenciais respondentes. O resultado desta etapa esta no Protocolo de estudo de campo, apresentado como Apêndice A.

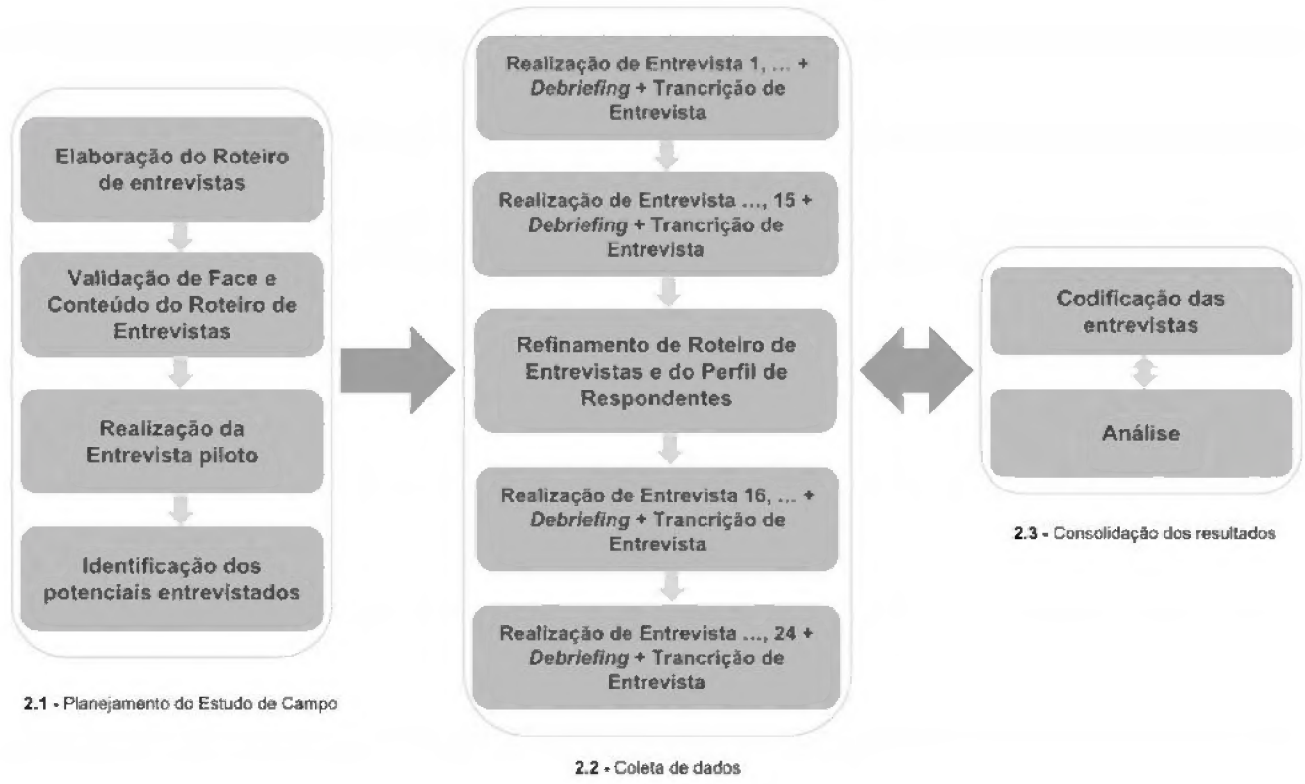


Figura 3.3 – Processo de pesquisa

Para a coleção dos entrevistados, foi realizada uma busca preliminar de possíveis candidatos em empresas do Tecnopuc; em listas de discussão profissionais, como, Guts, rioagile, gts-ce, teste-de-software-salvador, na rede profissional LinkedIn, e na comunidade *skillsmatter*, a qual preside uma conferência de BDD. Foram enviadas *e-mails* convite para os nomes verificados, apresentando a pesquisa, confirmando se o mesmo tinha o perfil desejado e se o mesmo teria disponibilidade de participar da entrevista. O perfil buscado se resume ao profissional tenha participado de pelo menos um projeto que utilizasse de BDD na indústria. De um total de 150 convites enviados, foram recebidas respostas de 100 profissionais (taxa de respondentes = 67%), que voluntariamente aceitaram participar deste estudo. Dentre estes 100 profissionais que se disponibilizaram inicialmente a contribuir com este estudo, 24 no total foram entrevistados pelo entrevistador (com o apoio da orientadora) terem percebido a saturação dos resultados, ou seja, os mesmos começaram a se repetir após este ponto.

Após alinhamento e clareza na definição dos objetivos deste estudo, elaborou-se o roteiro de entrevistas (vide Apêndice A). Este roteiro elenca um conjunto de perguntas abordadas durante as entrevistas, listadas na Tabela 4.1 (vide Seção 4,) para facilitar a compreensão do escopo da investigação.

Uma vez elaborado o roteiro de entrevistas, partiu-se para validação do mesmo. Foram realizadas diversas iterações entre os colaboradores deste estudo (orientadora, professor Cleidson de Souza da UFPA, colegas do grupo de pesquisa, etc). Após a concretização da versão que se acreditava estar condizente com o que se havia planejado, foram realizadas validações do roteiro de entrevistas. Assim, foi realizada a validação face-conteúdo para refinar as questões de pesquisa, para verificar se este roteiro seria entendido pelos entrevistados e avaliar a validade interna do roteiro de entrevistas, validação conteúdo [44]. Para validação do roteiro de entrevistas buscou-se a perspectiva de três pesquisadores, dentre os quais dois possuem experiência teórica de BDD, professores que colaboram na linha de Engenharia de Software da Faculdade de Informática da PUCRS, que participaram da validação de conteúdo; e, um que possui 6 anos de experiência na indústria e 3 anos de experiência com BDD, que participou da validação de face. Vale ressaltar que estes três integrantes foram profissionais escolhidos por conveniência e que a realização destas validações foram realizadas com um profissional de cada vez, considerando observações realizadas por eles no roteiro de entrevistas.

Após a validação de face e de conteúdo [44] do roteiro de entrevistas, foram realizados duas entrevistas piloto com praticantes de BDD na indústria. A primeira entrevista piloto foi com um profissional, uma Engenheira de Testes, com 8 anos de experiência no cargo e 3 anos em BDD, indicada por uma colega de pesquisa. A segunda entrevista piloto foi com um profissional, um Projetista, com 6 anos de experiência no cargo e 2 anos em BDD, o qual estava listado entre os profissionais que participariam deste estudo. Ambas en-

trevistas piloto tiveram duração média de 45 minutos e foram realizadas com profissionais localizados em Porto Alegre, RS.

As entrevistas piloto foram devidamente transcritas e analisadas para observar se as respostas estavam condizentes com o que era perguntando, isto com o intuito de garantir a clareza das perguntas e o entendimento dos entrevistados. Além disso, os pilotos serviram para a entrevistadora buscar o domínio do instrumento de coleta e refletir sobre como conduzir a entrevista. Os pilotos realizados tiveram duração com tempo médio de 45 minutos. Os dados coletados durante os pilotos não foram utilizados. Em paralelo às validações do roteiro de entrevistas, foi confeccionado um Termo de Consentimento (vide Apêndice B), para garantir aos entrevistados a confidencialidade dos dados fornecidos e integridade de suas identificações.

Etapa 2.2 - Coleta de Dados

Após validação do roteiro de entrevistas e dos pilotos, deu-se início ao processo de coleta dos dados. As entrevistas foram conduzidas baseadas no roteiro previamente elaborado. Todas as entrevistas foram devidamente gravadas com autorização prévia dos respondentes e transcritas posteriormente. Durante as entrevistas, os participantes foram estimulados a falar livremente enquanto respondiam às perguntas feitas. As entrevistas foram realizadas pessoalmente em prédios do campus central da PUCRS, em Porto Alegre, incluindo os prédios nos quais as empresas do Parque Tecnológico (Tecnopuc) se encontram hospedadas ou nos locais de trabalho dos entrevistados, em alguma outra localização da cidade de Porto Alegre. Quando as entrevistas eram realizadas com profissionais de fora de Porto Alegre ou do Brasil, as entrevistas eram realizadas via Skype/Hangouts.

Imediatamente, ou em até 24 horas após a realização de cada uma das entrevistas, que duraram de 35 a 76 minutos, foi redigido o *debriefing*. Este documento (Anexo A), gentilmente cedido pelo professor Cleidson de Souza da UFPA, visa a reflexão sobre a entrevista realizada, buscando resumir as percepções e aprendizados adquiridos com a entrevista. A ideia do uso do instrumento veio como sugestão do professor Cleidson após uma breve conversa do mesmo com a orientadora deste trabalho de Mestrado sobre como melhorar o processo de coleta e análise de em pesquisas qualitativas. Passou-se então a compartilhar as anotações do *debriefing* com a orientadora e o próprio professor Cleidson para uma breve discussão dos dados coletados, garantindo uma contínua discussão e refinamento da forma de conduzir as entrevistas. Após as discussões, se considerava o que poderia ser investigado, em profundidade, nas entrevistas posteriores.

Visto a natureza cíclica de Teoria Fundamentada em Dados, a qual promove a coleta e a análise dos dados [75, 19], foi possível se observar a saturação do entendi-

mento do fenômeno. Complementarmente, considerou-se que a coleta ocorreu em dois grandes momentos. Ao se consolidar os resultados parciais até a entrevista 15 para uma apresentação ao grupo de pesquisa ao qual a primeira autora está inserida, percebeu-se que era necessário entender mais profundamente o viés do uso de métodos ágeis anterior a adoção do BDD e também averiguar se profissionais atuando fora do Brasil possuíam percepções semelhantes as encontradas até então. Assim, 9 novas entrevistas foram realizadas com o maior representatividade de profissionais localizados fora do Brasil. Para este segundo momento de coleta, adicionou-se uma nova pergunta ao roteiro de entrevistas visando identificar como o BDD é impactado pelas metodologias ágeis. Esta pergunta, Tabela 3.3, adicional foi repassada aos 15 entrevistados do primeiro momento de coleta, sendo que 9 deles retornaram suas respostas.

Cada uma das entrevistas foi devidamente gravada e depois da realização e discussão do *debriefing*, transcrita com um aluno de apoio de iniciação científica e revisada pela aluna autora deste trabalho. As entrevistas duraram de 35 a 76 minutos, com uma média de 55 minutos. No total, obteve-se 22 horas de entrevistas gravadas que representaram mais de 500 páginas de transcrição.

Etapa 2.3 - Consolidação dos Resultados

Por se tratar de uma pesquisa qualitativa, a análise dos dados é dita interpretativa. Para isto, as entrevistas foram analisadas com o auxílio de um software de análise qualitativa, denominado Atlas.ti [1]. Para esta codificação a análise foi feita questão a questão, ou seja, uma questão por vez para todos os entrevistados. Ao final desta análise, vantagens do BDD, desvantagens, entre outros aspectos foram identificados. A partir da reflexão sobre os mesmos, observou-se que alguns entrevistados "agrupavam"estes aspectos.

Durante a codificação através do Atlas.ti, foram analisadas questão a questão, ou seja, uma questão por vez de todos os entrevistados, perguntas nas Tabelas 3.2 e 3.3. Posteriormente a codificação inicial, que transformou citações do entrevistados em códigos, foram formadas redes, exemplificada na Figura 3.4. Assim pode-se consolidar as respostas dos entrevistados em um único local, local que refletiu uma rede de agrupamento dos códigos (respostas = citações dos entrevistados) das perguntas.

Primeiramente, na etapa de codificação aberta, realizou-se a separação dos dados em categorias e subcategorias. Posteriormente, a codificação axial foi realizada com intuito de obter-se a compreensão do que os dados significavam. Para isto, utilizou-se de códigos e famílias utilizando-se de uma funcionalidade que permite visualizar graficamente o resultado da codificação. Um exemplo é apresentado na Figura 3.4. Este exemplo refere-se a codificação feita para a pergunta 1 do roteiro de entrevistas.

Tabela 3.2 – Perguntas do primeiro momento da coleta

ID	Pergunta	QP
1	Como você define o BDD?	0
2	Como você tipicamente utiliza o BDD nas atividades diárias?	1
3.1	Em qual tipo de projeto você frequentemente utiliza BDD?	0
3.2	Quantos projetos você se envolveu com o uso do BDD?	1
3.3	Qual tipo de tecnologia foi usada?	0
3.4	Qual linguagem de programação?	1
3.5	Quais tipos de metodologias ágeis foram utilizadas conbinadas com a abordagem BDD?	0
3.6	A equipe está distribuída? Qual o nível de distribuição?	0
4.1	Quantos membros se envolvem durante o uso do BDD?	1
4.2	Quais são os papéis que se envolvem durante o ciclo BDD?	1
5	Há algum cliente envolvido durante o BDD?	2
6.1	Quais *frameworks* são adotados para apoiar as atividades do BDD?	1
6.2	Quais as ferramentas são utilizadas para apoiar a gestão dos cenários e histórias?	1
7	Você gera algum artefato quando utiliza BDD? Se sim, qual(is)?	1
8.1	Quem é responsável por criar os cenários do BDD?	2
8.2	Como é coletada a informação para a escrita dos cenários?	2
8.3	Qual tipo de informação é usada como referência para a escrita dos cenários? Quem provê estas informações?	2
8.4	Quando, no ciclo de vida, você define os cenários?	2
8.5	O cliente se encontra com a equipe enquanto os cenários são definidos?	2
8.6	Como os cenários são compartilhados com a equipe?	1
8.7	Quem garante o alinhamento dos cenários com as necessidades do negócio?	2
8.8	Você acredita que as necessidades dos clientes ficam mais claras e são melhor entendidas quando há o uso do BDD para expressar os requisitos de software?	2
8.9	Você acredita que o BDD apoia a especificação dos requisitos?	2
9.1	Você acredita que haja benefícios no uso do BDD? Se sim, quais?	3
9.2	Você acredita que o BDD contribui para a melhora da qualidade do produto de software desenvolvido?	3
9.3	Em uma visão geral, você acredita que os problemas frequentemente enfrentados pela equipe de desenvolvimento são reduzidas com o uso do BDD?	3
9.4	Você recomendaria o BDD para outros? Em quais situações?	3
10.1	Você encontra dificuldades com o uso do BDD? Se sim, quais?	3
10.2	Qual sua sugestão para a melhora da abordagem BDD?	3
10.3	O que você acredita que esteja faltando para uma maior adoção do BDD na comunidade?	3
11.1	Porque você ou sua equipe decidiram adotar o BDD?	0
11.2	Como você aprendeu BDD?	0

Tabela 3.3 – Pergunta adicional do segundo momento de coleta

ID	Questão	QP
12	Imagine a seguinte situação: Você esta usando somente cerimônias ágeis, sem o BDD, conforme as cerimônias citadas anteriormente. Você acredita que continuaria tendo os mesmos benefícios?	3

A Figura 3.5 exemplifica a codificação realizada utilizando o software Atlas.ti e ilustra trechos de uma entrevista já categorizada, com o conjunto de códigos, conforme descrito a seguir:

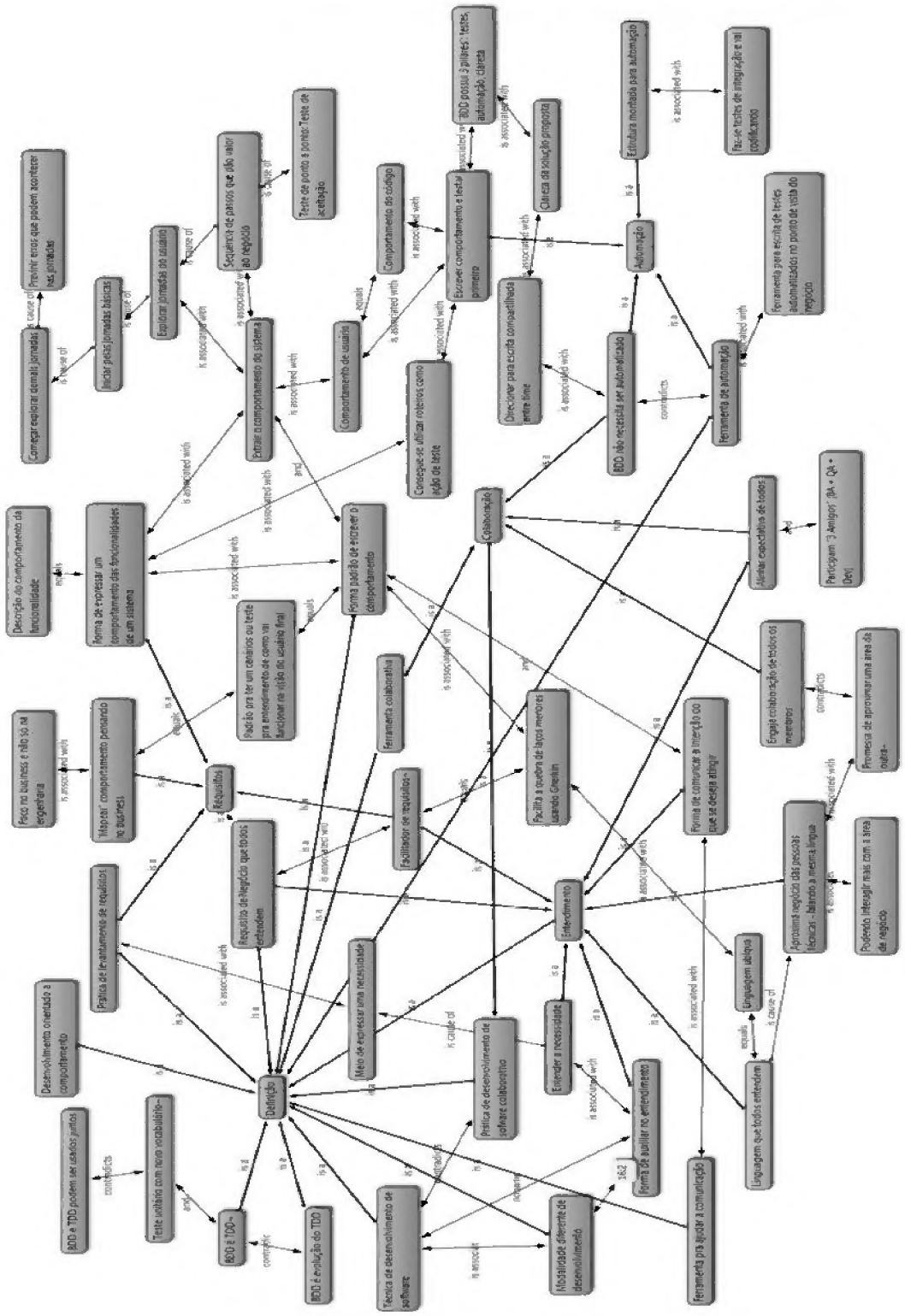


Figura 3.4 – Exemplo de Rede do Atlas.ti

- A Seção 1 (indicada pela cor laranja) da tela possui a listagem das transcrições para a categorização, que são utilizadas para a análise (chamados Documentos Primários) ou serve para explorar os objetos, como códigos de toda a unidade de análise (chamada Unidade Hermenêutica), citações, notas do documento, e visualização e edição de redes.

- A Seção 2 (cor verde) apresenta o trecho de transcrição, chamado Campo de Documentos Primários.
- A Seção 3 (cor roxa) exibe informações e categorizações relacionadas ao documento em exibição, no Campo de Documentos Primários (chamada Área Marginal). As marcações coloridas em laranja e verde são as categorizações realizadas sobre o documento.

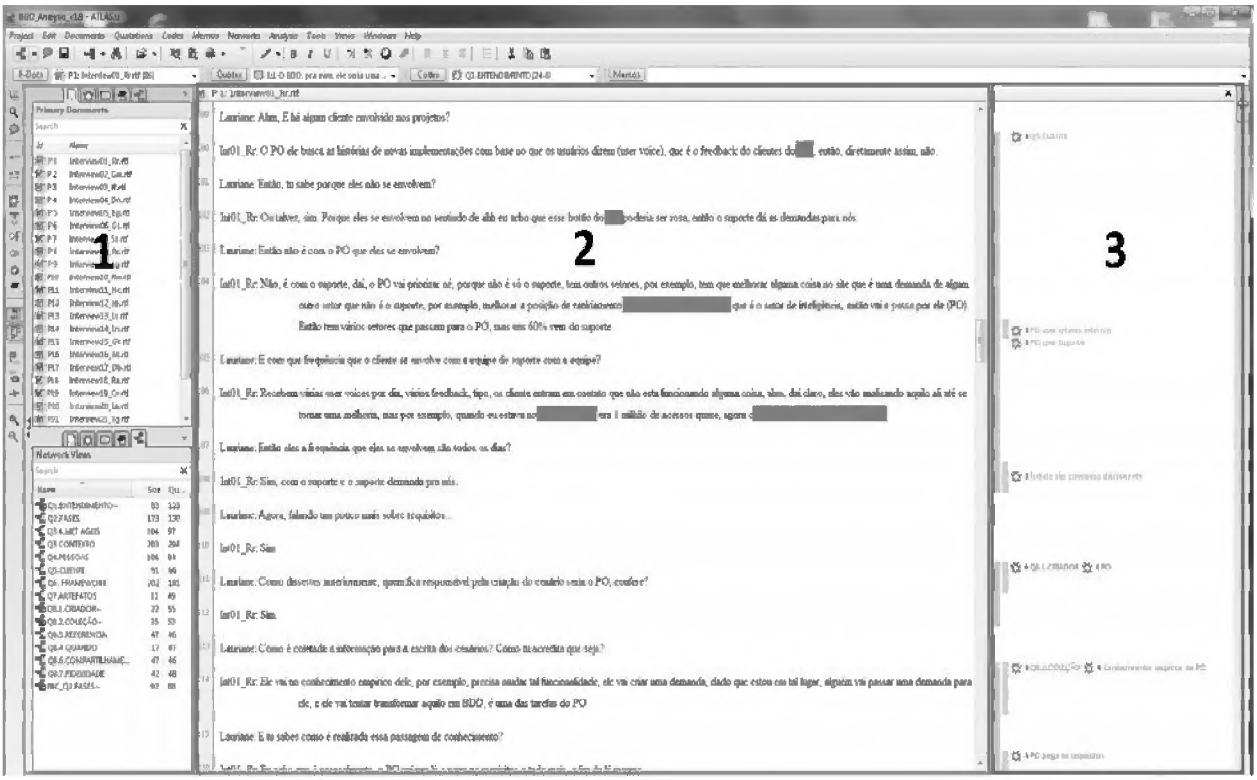


Figura 3.5 – Exemplo de uso do software Atlas.ti

Após a categorização e análise das entrevistas, obteve-se a codificação seletiva, a ser descrita no Capítulo 4, o qual apresenta os resultados deste estudo. Com isto, GT forneceu a perspectiva pela qual foi possível examinar como o BDD é realizado na prática e como o mesmo apoia a ER.

4. RESULTADOS DO ESTUDO DE CAMPO

Este capítulo apresenta os resultados do estudo de campo. A Seção 4.1 apresenta a caracterização dos entrevistados. A Seção 4.2 apresenta os resultados de perguntas gerais, ou seja, que serviram apenas o propósito de auxiliar no entendimento geral do tema e que não estão relacionadas às questões de pesquisa (QP) deste estudo. A Seção 4.3 apresenta os resultados referentes à QP1. A Seção 4.4 apresenta os resultados para a QP2 e a Seção 4.5 apresenta os resultados para a QP3.

Assim, os resultados discorridos neste capítulo refletem 24 entrevistas com profissionais da área de Tecnologia da Informação, distribuídos entre diversos papéis tais como Analistas, Desenvolvedores e Testadores. Todos entrevistados tinham experiência prática na utilização de BDD na indústria, conforme critério de seleção dos mesmos. Observou-se o uso do BDD tanto para desenvolvimento co-localizado quando para desenvolvimento distribuído de software. A aplicação do mesmo foi vista nos seguintes contextos: manutenção de software, criação de novos projetos e produtos de software, para o compartilhamento de informações de projeto ou a automação de software e tanto para aplicações *web*, *mobile* e *desktop*, entre outros.

A forma de apresentação dos resultados está organizada segundo as questões de pesquisa que guiaram este trabalho. Assim, cada QP é composta de resultados advindos de um conjunto de perguntas conforme mapeadas umas as outras na Tabela 3.2 e Tabela 3.3. Ainda, os resultados são apresentados em uma sequência lógica para facilitar a narrativa e compreensão do leitor de acordo com as indicações para as devidas perguntas do roteiro de entrevistas mencionadas no início de cada Seção.

4.1 Caracterização dos Entrevistados

A caracterização dos 24 profissionais está na Tabela 4.1. Esta está definida com os seguintes atributos:

- (ID) número de identificação do respondente;
- (Cidade) cidade em que o respondente estava alocado;
- (País) a sigla do país em que o mesmo estava alocado;
- (Ind) anos do profissional na indústria, os indicadores de '+' e '-' utilizados se referem é 'mais de' e 'menos de' respectivamente;
- (Cargo) anos do cargo de atuação;
- (Ágil) anos que tem de experiência com metodologias ágeis, o '-' que há nos campos representa que o respondente não preencheu este campo;

Tabela 4.1 – Caracterização dos entrevistados

ID	Cidade	País	Cargo	Ind	Cargo	Ágil	BDD	Equipe	D?	Tam	TI?
R1	Porto Alegre	BR	Analista de Testes	8	1	8	5	6	N	M	S
R2	Porto Alegre	BR	Agile Coach	10	3+	7	4+	15	S	GM	N
R3	São Paulo	BR	Desenvolvedor	8	1-	-	4+	20	N	GN	S
R4	Porto Alegre	BR	Desenvolvedor	3	3	-	2	6	N	GM	S
R5	Porto Alegre	BR	Engenheiro de Software	9	1-	-	5	7	N	GN	N
R6	Porto Alegre	BR	Desenvolvedor	5	5	-	1-	7	N	GN	S
R7	Rio de Janeiro	BR	Analista de Qualidade	4	4	6	1	5 - 12	N	GN	N
R8	Sorocaba	BR	Coordenador de Qualidade		8	-	1	2 - 15	N	GM	S
R9	Manaus	BR	Analista de Testes	3	3	-	2	6	N	GN	N
R10	Porto Alegre	BR	Consultor	10	6	6	5-	15	S	GM	S
R11	Porto Alegre	BR	Analista de Negócios	14	13	7	5	6 - 12	S	GM	S
R12	Porto Alegre	BR	Analista de teste	4	2	2	2-	7	N	M	S
R13	Porto Alegre	BR	Engenheiro de Software	12	10	5	2	4	S	GN	S
R14	Porto Alegre	BR	Analista de Qualidade	4	3	4	3	12	S	GM	S
R15	Nova Hamburgo	BR	Arquiteto de Software	6	2	6	5	13	S	P	S
R16	Estocolmo	SE	Engenheiro Backend	16	5	5	3	6	S	M	S
R17	Auckland	NZ	Engenheira de Testes	6	1-	5	1-	8	S	GM	S
R18	Londres	UK	Líder QA	8	1-	5	3	3	S	M	S
R19	Berlim	AL	Desenvolvedor	6	1	6	6	5 - 10	S	GN	N
R20	Belo Horizonte	BR	Engenheiro QA	5	5	5	5	6	S	M	S
R21	Nova York	US	Engenheiro de Software	9	1	6	4	10	S	P	S
R22	Belo Horizonte	BR	Engenheiro QA	17	7	5	2	7	N	M	N
R23	Porto Alegre	BR	Analista de Sistemas	9	3	1,5	1,5	5	N	GN	S
R24	São Paulo	BR	Analista de Sistemas	7+	4+	5+	3+	7	S	GM	S

- (BDD) anos de experiência com BDD;
- (Equipe) quantidade de pessoas que há na equipe que o respondente geralmente está inserido, quando utiliza de BDD;
- (D?) 'S', sim, significa que o profissional participa de projetos distribuídos e 'N', não, ou seja, o mesmo trabalha em equipes co-localizadas;
- (Tam), indicando qual é o tamanho da empresa em que o entrevistado estava imerso. Esta coluna possui as seguintes definições: (P) para pequena empresa, que possui entre 20 e 99 empregados; (M) para média empresa, que possui, entre 100 e 499 empregados; (GN) para grande empresa nacional com mais de 500 empregados; e, (GM) para grande empresa multinacional com mais de 500 empregados, localizada em diversos países [69]; e,
- (TI?) 'S', sim, se a atividade principal do negócio é Tecnologia da Informação e 'N', não, caso não pertença.

4.2 Observações gerais sobre o BDD

1) Definição do termo BDD

De acordo com a Tabela 3.2, a pergunta 1 teve como objetivo identificar como os entrevistados definem o BDD. Isto com o intuito de verificar se os respondentes tem o mesmo entendimento do que é BDD comparado com o que há na literatura atual. Assim, observou-se a seguinte rede de códigos extraída da análise desta questão, ilustradas no Apêndice D.

Os respondentes definiram o BDD como sendo: uma forma de extrair o comportamento do sistema, uma linguagem que todos entendem, uma ferramenta de automação, uma ferramenta para ajudar na comunicação, a evolução do TDD, o desenvolvimento orientado ao comportamento, uma prática de levantamento de requisitos, uma prática de desenvolvimento de software colaborativo, uma ferramenta colaborativa, uma técnica de desenvolvimento de software, uma modalidade de desenvolvimento, TDD, um complemento de documentação, e, teste de aceitação.

Assim, observa-se que não há um consenso sobre a definição do termo BDD. Conforme relato: BDD *"é um conceito (...), ninguém concorda na definição do BDD. Algumas pessoas pensam em ferramentas, outros pensam meio técnica ou mindset, uma coisa assim. Mas para mim, eu sempre foco nos results [resultados], no comportamento final do sistema"* (R11). O mesmo informou que vê o BDD como uma **ferramenta pra ajudar na comunicação**. BDD também foi definido como uma **modalidade diferente de desen-**

volvimento (R16). Também há relatos de que BDD é uma técnica de desenvolvimento de software (R10, R21). Sendo que há adição de que BDD também seja uma prática de desenvolvimento de software colaborativo (R21). No entanto há a definição de que BDD é uma prática de levantamento de requisitos (R24). Há definição de BDD como meio de expressar uma necessidade (R2, R23).

Em contrapartida, há a definição de que **BDD seria o TDD com um novo vocabulário**(R5):

> *"Para mim BDD é o TDD. O unit-testing com o novo vocabulário. Eu não vejo diferença nenhuma além dessa, sinceramente." (R5)*

Em contrapartida foi reportado que BDD e TDD podem ser usados juntos (R3). Além disso, entrevistados informaram que entendem **BDD como uma evolução do TDD**, conforme abaixo:

> *"BDD é uma evolução do TDD, porque no fim das contas quando tu tinha o TDD tu tinhas um entendimento básico do que tinha que desenvolver do software, então tu vinhas com abordagens para testar o software antes de desenvolver. Então, tu colocarias ali os testes, o que tu achas que os componentes do software devem fazer, já pensando na arquitetura e tudo mais, então tu desenvolves os testes, vê que eles não passam, desenvolve a funcionalidade o mais rápido possível para os testes passarem e aí depois vai lá, refatora o código e tal para depois ir fazendo descobertas ao longo do caminho e aí tu vai fazendo a cobertura de testes. Então, acho que o BDD é uma evolução em cima disso porque tu não pegas, não partes de componentes da arquitetura, tu parte de funcionalidades que são expressas pelos requisitos." (R17)*

> *"A ideia do BDD é que ele evoluiu do TDD para incluir os analistas de negócio no teu desenvolvimento, trazer eles para mais perto possível que eles escrevam os próprios cenários de teste em BDD." (R19)*

Contudo, o que foi frequentemente mencionado pelos entrevistados ao serem inquiridos sobre a definição do BDD foi entendimento (R9, R21), como uma **forma de auxiliar no entendimento**, conforme:

> *"Forma de auxiliar o entendimento e a cobertura, o rastreamento de cenários para aplicação." (R9)*

> *"Então a gente vota [faz estimativa de pontos] e o legal do BDD ou dos cenários, dos critérios de aceite nessa parte, que isso facilita a visualização do problema ou do entendimento daquela funcionalidade." (R21)*

Há quem ressaltou que BDD auxilia **entender a necessidade**:

"Através dos requisitos que eu adquiri do cliente, eu faço entrevistas com os clientes, para entender a necessidade e a partir disto eu crio uma documentação, não crio somente o BDD." (R23)

Este entendimento esta associado a **linguagem comum** que é apoiada pelo BDD, chamada de linguagem que todos entendem, linguagem ubíqua, OU *Gherkin*, citado por grande parte dos entrevistados, dentre os quais estavam analistas, desenvolvedores e testadores, declararam, alguns declararam que BDD apoia **linguagem que todos entendem, linguagem ubíqua**:

"Se a gente usa, por exemplo, alguma linguagem natural para expressar algum cenário, ele é escrito de forma... da mesma forma que a gente escreve um texto e isso auxilia que mesmo que uma pessoa que não seja técnica tenha um entendimento do que aquilo representa, e aí por trás tem todo um lado mais técnico, digamos assim, que de fato implementa e corresponde a esse mapeamento que é a linguagem natural e a execução daquele cenário." (R2)

"O principal objetivo do BDD é ter uma linguagem que todas as pessoas envolvidas no projeto consigam entender descrevendo quais são os comportamento da aplicação dada às funcionalidades." (R7)

"Porque como o BDD emprega a linguagem ubíqua ele então, assim, é de fácil entendimento." (R9)

Alguns entrevistados descreveram que *Gherkin*, na descrição de cenários, contribui para o entendimento dos integrantes da equipe. Um deles mencionou que isto facilita a quebra em laços menores, com *Gherkin*, fazendo que todos envolvidos entendam os requisitos (R1).

"Um Gerente de produto, alguém quer definir como a gente vai desenvolver uma certa funcionalidade, consegue quebrar isso em histórias, em laços menores usando o BDD que é o Gherkin (...) Isso acaba facilitando tipo, como o pessoal fazia a tempo, dez anos atrás de quebrar em bullet points, escrever user keys e essas coisas que ficavam complicadas." (R21)

"Facilitador de requisitos [Dado que...Quando...Então], é uma linguagem aonde que o PO entende, que a gente utiliza ágeis, que o cliente lá na ponta entende, eu, como testador entendo, então é tipo é uma linguagem de requisitos aonde tecnicamente se entende as coisas de forma universal." (R1)

"Um requisito de negócio de uma forma que todas as pessoas envolvidas consigam entender." (R2)

Assim como há entrevistado que ressalte que o BDD faça que os requisitos de negócio sejam entendidos por todos (R11). No entanto o mesmo complementou que isto é algo que possibilita que os desenvolvedores escolham o jeito deles para desenvolver o código, pois o foco é o negócio, os comportamentos, os resultados do sistema.

Assim, foi relatado que o BDD permite uma forma de comunicar a intenção do que deseja atingir (R4, R12). Diante desta linguagem comum há a concepção de que haja uma aproximação de pessoas técnicas com pessoas de negócios conforme os seguintes relatos: **aproxima negócio das pessoas técnicas - falando a mesma língua, Podendo interagir mais com a área de negócio, Promessa de aproximar uma área da outra** (R21, R15, R5).

Alguns dos entrevistados informaram ainda que há o entendimento dos requisitos devido haver o foco no comportamento do sistema e no negócio, por BDD permitir **mapear comportamento pensando no negócio** (R15). Há quem ressalte que o BDD centre no negócio e não só na engenharia (R16). Diante disto, o mesmo informou que isto evidencia que o BDD é uma modalidade diferente de desenvolvimento (R16). No entanto, um entrevistado ressaltou que há a padronização de cenários ou testes para entendimento de como vai funcionar na visão do usuário final (R18).

Diante da linguagem ubíqua apoiada pelo BDD entende-se que há uma forma padrão de escrever o comportamento (R4, R18). Há a definição que BDD ajuda sendo uma forma de expressar um comportamento das funcionalidades de um sistema (R2). Similarmente, há a definição de que **BDD é a descrição do comportamento da funcionalidade**:

"BDD é a definição de comportamento do software em si, ele tem que identificar como ele funciona, como que ele anda, como que pausa na visão do usuário final, qual que é o comportamento que ele deve ter, esse é o principal foco dele tá." (R8)

"Tu parte de funcionalidades que são expressas pelos requisitos. Então, pegas comportamentos do software, quando eu tenho um pausa quando um cliente vai e pede um reembolso no caixa, o que tem que acontecer? Eu tenho que devolver a mercadoria para o estoque, tenho que devolver o dinheiro para o cliente e eu tenho que sei lá, registrar isso em algum outro sistema. Então, tu defines isso em outro comportamento e a partir daí, então, tu vais desenvolver os componentes de software para entender esse comportamento, mas tu partes dos requisitos." (R17)

BDD é definido como uma estratégia para extrair o comportamento do sistema (R14). Isto foi reforçado, por outros entrevistados que relataram que há uma sequência de

passos, em BDD, que dão valor ao negócio (R14, R3, R20). Uma entrevistada ressaltou que explora jornadas do usuário, iniciando pelas jornadas básicas até as demais (R14), e que isto consequentemente ajudará a prevenir erros que podem acontecer nestas jornadas (R14).

Há quem mencione que BDD representa escrever comportamento e testar primeiro (R3). No entanto, há quem ressalte que BDD auxilia na definição do comportamento de usuário (R15), conforme a respectiva citação:

> *"Ao invés de pegar e criar um teste unitário para uma porção de código, eu passava a pegar um comportamento do usuário, o comportamento de uma regra, então escrevia um teste específico para o comportamento." (R15)*

Também, há entrevistados que reportaram que o foco do BDD está na definição do **comportamento do código**, conforme a seguir:

> *"Você foca no comportamento que você quer que o código tenha, então você, normalmente, vai tentar descrever um teste ou antes ou depois, depende, varia um pouco, eu pessoalmente prefiro antes para descrever o comportamento que se espera que o código tenha e você pode fazer um teste automatizado ou não, vai depender também da situação, daí você começa a escrever o código para atender aquele comportamento." (R10)*

> *"Então a gente sentava com a pessoa do negócio e definia em inglês mesmo, na ferramenta que a gente usava, e aí depois a gente traduzia isso para código, executava, fazia o código passar e a área de negócio para implementar. Então, partia do comportamento do usuário, como usuário estaria usando o aplicativo, no caso um aplicativo web." (R19)*

Além disso, há quem reporte que com BDD se consiga utilizar roteiros como ações de teste (R7). Assim os roteiros, ou passos descritos para o BDD podem ser utilizados também como teste.

Ainda, há entrevistado que menciona que BDD é visto como uma ferramenta de automatização por muitos usuários (R22). Também, há quem relate que utiliza da ferramenta para escrita de testes automatizados do ponto de vista do negócio (R19). Há quem mencione que usufrua da estrutura do BDD para automação (R3, R8). No entanto, há entrevistados que ressaltem que BDD não necessite ser automatizado, pois o grande valor está centrado na clareza da solução (R9, R10, R12).

Adicionando um entrevistado cita que 'BDD possui 3 pilares': testes, automação, clareza: "*testes para avaliar se as definições estão alinhadas com o negócio, a automação vem de resultado final, e a clareza para deixar claro as necessidades de projeto à todos*

envolvidos no mesmo"(R8). Um entrevistado relata que o BDD proporciona a **clareza da solução proposta** e que está é a maior vantagem de uso do BDD, por direcionar a escrita compartilhada entre a equipe (R13). Assim como BDD é definido como ferramenta colaborativa a qual permite a **colaboração de todos os membros**:

> *"Então, para mim é realmente um desenvolvimento ambientado a um comportamento mais que um lugar assim vários embasamento entre todos stakeholders envolvidos em um projeto assim, desde QA interrupção com a visão mais macro que alguns casos que eu usei tipo diretorias e empresas também chegaram, li as histórias, sabia o que estava acontecendo, consegui entender, então tem essa questão que eu acho de engajar colaborações de todos os membros envolvidos num projeto." (R22)*

2) Contexto de projetos que usufruem de BDD

De acordo com a Tabela 3.2, a perguntas 3.1 e 3.3 tiveram como objetivo entender a qual contexto de projetos os entrevistados estavam imersos, para verificar os contextos aos quais o BDD está inserido.

Uma diversidade de contextos foram reportados. O BDD é aplicado em sistemas da área da saúde que atendam clínicas farmacêuticas, pesquisas e medicamentos e sistemas hospitalares (R16, R19). Também é usufruído para projetos de aviação e televisão (R19).

Também, o BDD é usado para **projetos de *e-commerce*, e desenvolvimento de lojas online** (R20, R24, R11). E para projetos de sistemas de **gerenciamento de conteúdo**, revistas online (R12, R24), **sistemas de controle administrativo** (R23), **educacionais** (R9) e **projetos que tenha interação com o consumidor** (R22). E, ainda, para construção de **ferramentas de análise** (R11) e **projetos que envolvem *Big Data*** (R17).

BDD é utilizado para **documentar regras e apoiar o entendimento** (R2) e **validar regras de negócio e mapear cenários** (R2), ou para **definir um comportamento que já existe** (R4). Além disso foi mencionado que BDD usufruído para **banco de dados** (R20), **integração de servidores** (R18), e desenvolvimento do ***serverside*** (R5). Há o uso de BDD para **testar a interface gráfica** (R5, R12). Há que mencione que utiliza de BDD para aplicações *Desktop*.

3) Metodologias ágeis usadas com BDD

De acordo com a Tabela 3.2, a pergunta 3.5 teve como objetivo entender quais metodologias ágeis são utilizadas junto ao BDD visando identificar quais práticas servem de apoio, complemento ou contextualização para o mesmo.

Uma variedade dos entrevistados indicou que utiliza Scrum e/ou Lean. Há quem tenha relatado que utiliza **mais de Lean do que Scrum** (R5), enquanto outros entrevistados mencionaram que o **Kanban** é predominante em seus projetos(e.g., (R3, R11, R19)) .

Ainda de forma geral, em outro conjunto de práticas de outros métodos também é utilizado: o Scrum, algumas cerimônias utilizadas concomitantemente ao uso de BDD, como em: **Planning, Daily, Review e Restrospective** (e.g., R1, R7, R12). Em contra-partida há quem reporte que praticam o **Scrum não tão formal**, que é quando não se utiliza todas as cerimônias que o Scrum recomenda (R4, R6, R9, R21).

Há o planejamento, a ***Planning*** para apoio ao desenvolvimento (e.g. (R16, R18, R22, R24)) ou ***Grooming*** (R16, R17, R22, R24) o qual no planejamento é pensando os cenários para o BDD. No ***Reality check*** (R8) há a reflexão da equipe quanto ao desenvolvimento dos cenários. No ***Kick off*** (R11, R14) se repassa todos os critérios de aceite da *user story* para haver o entendimento da equipe. No ***Dual-track Agile*** (R2) há o desenvolvimento do cenário junto a implementação. Na ***Daily*** (e.g. (R15, R16, R17, R18)) ou **Stand Up** (R20) se acompanha o andamento do projeto. No ***Desk check*** (R11, R19), conhecida também como **Coffee talk** (R19) se valida se os cenários estão implementados corretamente. Na ***Review*** (R15), ***Show-case*** (R10, R14, R19), ou ***Demo*** (R17) validam entrega com o cliente. Na ***Retrospective*** (e.g. (R10, R11, R16, R19)) se reflete sobre o projeto com a equipe. É frequentemente observado que o BDD é inserido nas seguintes cerimônias de BDD: ***Planning***, ***Kick off***, ***Grooming***, ***Desk check***, **Show-case**.

Há quem relate que utilize ***Scrum of Scrums*** (R17) enquanto diversos entrevistados (e.g., (R8, R13, R19, R21)) ressaltaram o uso do **XP**. Das práticas do XP, utiliza-se o ***Pair programming*** para o desenvolvimento (R9, R14, R19, R23) ou para escrita de Critérios de Aceite para o BDD (R14). Há quem ressalte o uso de **TDD** (R3, R21, R19). Há quem tenha mencionado o uso do ***Specification by example*** com ***Workshop*** **de escrita de *user stories*** (R14).

4) Níveis de distribuição das equipes que utilizam BDD

De acordo com a Tabela 3.2, a pergunta 3.6 teve como objetivo entender se o BDD é utilizado por equipes fisicamente distribuídas e entender qual o nível de dispersão das

mesmas visando identificar se o uso do mesmo é factível em desenvolvimento distribuído de software.

Os entrevistados relataram tanto o uso em equipes dispersas em **um mesmo estado** ou **entre diferentes estados** do país. Por exemplo, há quem indique que usa BDD com parte da equipe localizada em Porto Alegre (capital) e outra em Caxias do Sul (interior) (R23), ambas cidades no estado do Rio Grande do Sul. Há quem relate que usa com equipes dispersas entre Porto Alegre (RS) e São Paulo (SP), a capital. Outro que reporte que também segue a distribuição entre os dois estados, RS e SP, tendo sua equipe distribuída nas cidades de Campinas e Porto Alegre (R24).

Há distribuição entre **duas localizações**, mas foram relatados também casos em que as equipes estão em **três localizações** como as citadas a seguir. Há quem tenha sua equipe distribuída entre São Paulo (SP), Natal (RN) e Campinas (SP) (R13) e quem possua a distribuição entre São Paulo, Rio de Janeiro (RJ) e Porto Alegre (R11). Teve-se ainda o relato em que a equipe localiza-se distribuída entre duas cidades próximas no RS, Porto Alegre e Novo Hamburgo (localizada na região metropolitana da capital Riograndense), e Belo Horizonte (MG) (R15).

O mesmo nível de dispersão entre duas ou três localizações foi relatado entre países os quais estão localizados no **mesmo continente** ou em **continentes distintos** com mínima ou nenhuma sobreposição de horas de trabalho síncronas conforme exemplificado a seguir. A distribuição entre Brasil e USA foi diversas vezes reportada (R2, R4, R10, R19). Um deles também já trabalhou usando BDD com equipes distribuídas entre USA e Haiti (R19). Um dos entrevistados, atualmente localizado em Nova Iorque, USA, informou que já trabalhou com desenvolvedores localizados em entre Nova Iorque e São Paulo, e também distribuídas entre Brasil, Índia e USA (R21).

Um entrevistado reportou a distribuição entre Nova Iorque, Tóquio e Japão ou Nova York, Tóquio e Londres, Inglaterra (R16). Um dos entrevistados ressaltou que já participou, também, de equipes distribuídas entre Brasil, USA e Índia (R2) enquanto um outro entrevistado participou de equipes distribuídas entre duas regiões dos USA (Califórnia e Arizona) e Auckland, Nova Zelândia (R17). Há quem mencionou a distribuição entre dois países localizados na Europa–Inglaterra e França, e um na Ásia–Singapura (R18). Houve ainda o relato de distribuição entre Brasil, USA e Índia (R20).

Há quem esteve engajado em um projeto distribuído entre **quatro localizações** (R3): Brasil, USA, Índia e China. Além disso, há equipes que trabalham co-localizadas, ou seja, **sem distribuição** dos membros (e.g., R1, R2, R12, R22).

5) Adoção e aprendizado do BDD

De acordo com a Tabela 3.2, as perguntas 11.1 e 11.2 tiveram como objetivo entender a razão da adoção do BDD e como os praticantes aprenderam sobre o mesmo, visando entender sobre a motivação da adoção do BDD e como os mesmos adquiriram proficiência nesta abordagem.

Alguns entrevistados indicaram que o BDD **já era utilizado** quando eles entraram na empresa (R8, R9). Desta forma, eles desconhecem o motivo da adoção. No entanto, quando conheciam o motivo, vários entrevistados indicaram que a adoção aconteceu para seguirem a tendência do mercado (e.g., (R7, R14)) ou por notarem o uso crescente na comunidade *open source* (R2).

Outros indicaram que a adoção partiu **do incentivo da empresa**. Por exemplo, há quem mencione que BDD teve um experimento motivado pelas iniciativas da empresa de se buscar aprender algo novo (R6). Há relatos que houve a sugestão da empresa (R13, R23). Há entrevistado que ressalte que a crença da diretoria em ágil e suas práticas foi o que levou a adoção de BDD (R22). Há quem mencine que a adoção veio da percepção dos possíveis ganhos para a empresa (R16).

Outra razão interessante para a adoção foi pela **cultura de qualidade**, presente na empresa, proposta pelo pessoal de qualidade (R24). Um entrevistado mencionou que houve recomendação veio do departamento de engenharia para que se pudesse obter **qualidade no código** (R2). Também houve relatos de que a motivação para adoção foi visando **a melhoria da qualidade** como um todo (R19, R20, R23) ou para **reduzir o número de incidentes** em produção (R3).

Contudo, há quem tenha adotado BDD para disciplina da equipe (R11), de forma que todos pudessem definir os cenários juntos. No entanto há que ressalte que a adoção do BDD veio para **melhoria da comunicação entre a equipe** técnica e o negócio (R21). Ou para **melhoria da comunicação com o cliente** (R14). Todavia há quem tenha mencionado que a adoção veio para **padronizar as intenções do *product owner*** (R12).

O BDD também foi adotado pela equipe para **prezar a visibilidade do negócio** (R14), **buscar clareza com o cliente** (R19) e **vislumbrar interação com o mesmo** (R16). Ou ainda, com o intuito de se **buscar entender melhor o sistema**, ilustrado abaixo:

> *"Por que eles querem seguir todas as metodologias ágeis possíveis, porque é uma startup, então eles estão tentando agilizar tudo, então, eles falaram que com BDD podia entender melhor quais são os requisitos e tudo, melhor do que ficar lendo um monte de texto." (R18)*

Um dos entrevistados argumentou que sua equipe decidiu usar apenas para a **definição dos cenários** (R11), de forma a permitir o entendimento dos mesmos, sem visar a

automação. Há quem tenha adotado por **ser de fácil entendimento** (R7) e por **facilitar a previsão de estimativas** do projeto (R21). Outras razões foram: para **facilitar o desenvolvimento** (R15), **ter segurança nas entregas** (R3) e para **melhorar cenários de testes que estavam ruins**, conforme ilustrado a seguir:

> *"Foi a questão dos cenários de testes que estavam muito ruins, estavam difíceis de entender e não estavam cobrindo bem as funcionalidades, então a gente resolveu experimentar o BDD." (R10)*

Além de tudo, houveram ainda relatos da adoção pela **necessidade do cliente em ter documentação**, conforme os seguintes relatos:

> *"No segundo projeto foi mais essa questão da necessidade da documentação mesmo que deu início, mas o benefício maior que a gente viu foi na comunicação, apesar de não ter sido o foco inicial [da adoção]." (R10)*

> *"A documentação em si, a empresa não tem os sistemas documentados e hoje não se tem ideia quando se é um desenvolvedor novo, ou quando se tem uma nova funcionalidade, não se tem ideia do que se tem feito, então de tentar fazer um esforço nesse sentido, e tem uma validação mais rápida pelo fato de ter os testes automatizados, que hoje é difícil fazer testes de regressão, custa mais." (R16)*

Quanto à forma de aprendizado, foi recorrente a indicação de que o conhecimento de BDD é adquirido ao se iniciar a trabalhar com a abordagem (e.g., (R1, R12, R18, R21)), ou seja, com a prática no dia-a-dia em si. Desta forma, alguns entrevistados mencionaram que aprenderam errando (e.g., R12) visto que o projeto tinha a necessidade de utilização (e.g., R10, R11).

Também foi relatado que o aprendizado vem da **inserção em uma comunidade**, como a comunidade de Ruby (R4), por exemplo. Há situações em que o conhecimento vem de se observar outros projetos (R16) e ao se misturar BDD com TDD (R5).

Ainda, diante da necessidade de se ter conhecimento para colocar BDD em prática, alguns entrevistados relataram que acabaram **estudando por conta própria** (R3, R14), **ler livros** (R2, R4, R11, R15) ou, ainda, **participar de treinamentos**, sejam eles treinamento oferecidos por uma instituição especializada (e.g., *Cucumber Foundation* (R17)), cursos *online* (R20), canais de apoio à formação da empresa (R17), vídeos *online* (R3, R6), ou palestras como um todo (R1, R2).

4.3 BDD no ciclo de vida do projeto de software (QP1)

1) Ciclo de Vida de desenvolvimento com BDD

De acordo com a Tabela 3.2, a pergunta 2 teve como objetivo explorar como o BDD se encaixa no ciclo de desenvolvimento de software como um todo, visando identificar na adoção na prática, ilustrado na Figura 4.1. Há situações em que **todos desenvolvedores criam os cenários de BDD após definidos os critérios de aceite**, assim eles geram os **testes de aceitação automatizados** (R2).

Além disso, há o uso do DD como se fosse TDD, em que o **Desenvolvedor cria exemplos a partir de documentação escrita por Analista de Negócios** (R15). Assim como o uso de **BDD para construção de software** (R5), o mesmo denominou-se como *Coder*, que seria o cargo de Engenheiro do desenvolvimento com responsabilidade de Testador. Um outro entrevistado mencionou que utiliza o BDD para entendimento de comportamento (R4) tanto para definir funcionalidades novas de um sistema legado quanto para verificar um comportamento já existente no sistema.

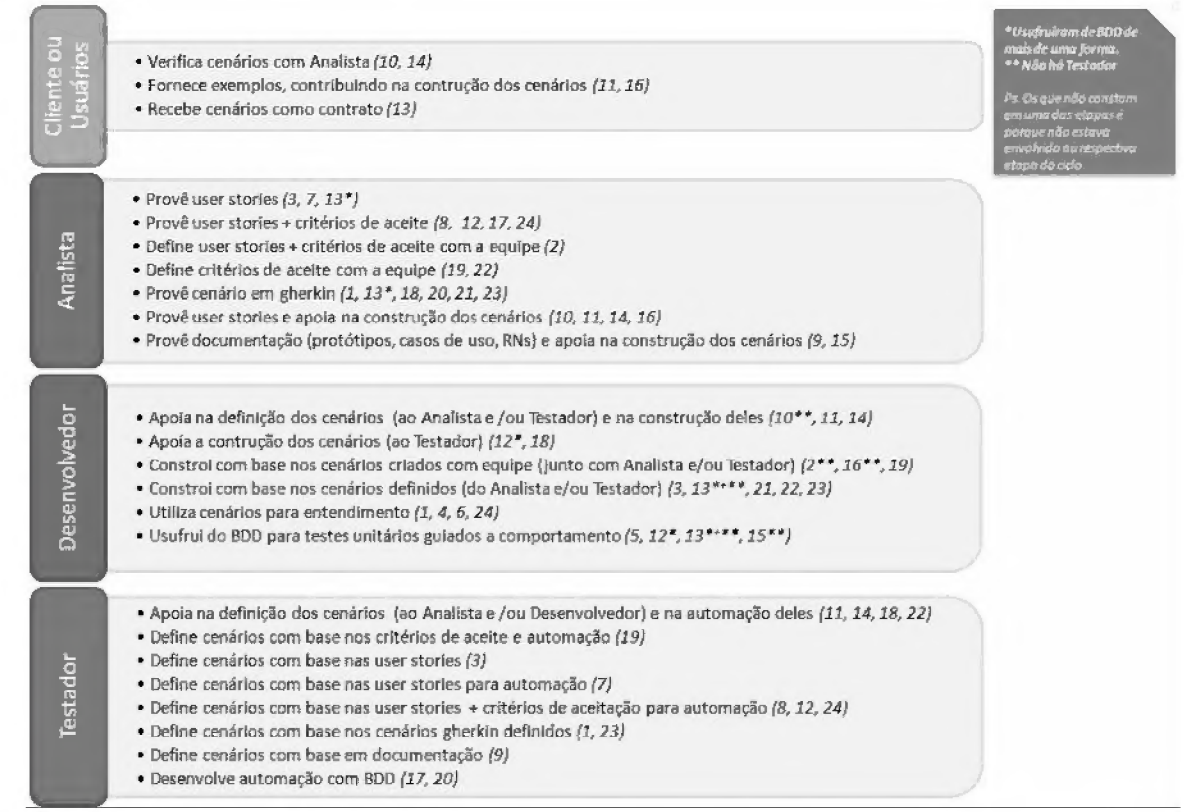


Figura 4.1 – Ciclos que utilizam BDD

Uma Analista de Teste utiliza da **estrutura do BDD para escrever histórias fornecidas pelo PO** (*Product Owner*) (R12). Para isto o PO descrevia a funcionalidade e os

critérios de aceite no *card*. Após há a revisão dos Desenvolvedores, ajustes são realizados e **cenários eram automatizados de acordo com ROI - mínimo para validar o máximo na automação**. Ou ainda outra opção é o **BDD realizado somente por Desenvolvedores, fazendo testes unitários guiados por comportamento**.

Também há **Analista de Teste passando *user story* para formato de BDD**, no entanto o **Desenvolvedor não utiliza desses cenários**. A equipe define quais histórias devem ser automatizadas, e o **Analista de Teste automatiza os cenários** (R7).

Há o uso de *Gherkin* para validação das funcionalidades (R17). O mesmo relatou que o PO define *user story* bem ampla e que os '3 amigos' se encontram raramente, **não há reunião para definição conjunta do *Gherkin***. O Testador lida com Cucumber , sendo que o software e a cobertura de testes são desenvolvidos paralelamente, com o Desenvolvedor fazendo a parte funcional do software e o Testador fazendo a automação dos testes.

Em contrapartida, há a situação em que o **Analista de Teste cria cenários que atendam todas regras de negócio, cenários alternativos e para comportamento da tela**, e compartilha estes cenários com toda equipe, viabilizando uma possível leitura para os desenvolvedores (R9). Este complementou que recebe os **protótipos, casos de uso e regras de negócio do PO** e que conta com apoio do PO para criar seus cenários.

Há, algumas vezes, o interesse pela descoberta dos critérios de aceite e pela automação, diante disso o **Gerente de produto define *backlog* com critérios de aceite (com *Gherkin*)** (R21). Assim tem-se a inserção do **BDD desde a análise dos itens**, estes critérios de aceite contemplam *happy paths* dos cenários. Estes são usados em *Planning poker*, mas sem as cartas. Após a criação de todos os cenários de aceite o código é refatorado pela equipe. Também, um **QA escreve os cenários com base em critérios de aceite** (R8) e desenvolvem a respectiva automação. Também, há **somente QA utilizando de BDD** (R20), com base nos requisitos, critérios de aceite, criados pelo PO. O QA automatiza os testes que validarão o sistema.

No entanto há situações em que o QA escreve cenários de aceitação e o **Desenvolvedor faz estes cenários passarem** (R3). Há situações em que se centra a atenção nos critérios de aceitação (R22), em que após definição de *user stories* pelo PO, antes da *Planning*, o PO e QA definem alguns critérios de aceite, com intuito de que na *Planning* os Desenvolvedores e QAs façam o máximo de perguntas para gerar os critérios de aceite, relatou um QA.

No entanto, há relato que consta a interação entre Desenvolvedores, QAs e Analistas de Negócio para definição dos critérios de aceite servindo para a **escrita dos cenários de teste baseados nos critérios de aceite** (R19).

Há situações em que é realizada discussão de *user stories*, algumas vezes com BDD escrito/cenários, assim o **QA coloca variações de cenários** (R24). Isto devido o PO ver BDD de forma simplista, sem detalhes, fazendo com que a equipe usasse BDD em sua

melhor forma, encontrando situações possíveis de erro. No entanto os **Desenvolvedores usam cenários para desenvolvimento, sem *framework* de BDD** e o QA conversa com PO sobre os cenários e os automatiza. Em alguns projetos PO escreve *user stories* e critérios de aceite, estes critérios de aceite definem características de que a história esta pronta.

Há ciclo em que o **PO passa requisitos para cenários com apoio de QA e Desenvolvedores, colocando estes em *tickets*** (R18), quando são **funcionalidades novas PO e Desenvolvedores dão *reviewed* para QA** ou quando são **funcionalidades existentes Desenvolvedores dão *reviewed* para QA**. A regressão é usada para estes cenários, havendo os que são realizados de forma manual.

O **BDD faz parte desde a escrita dos requisitos até os testes** (R16). Com isto PO e representantes do negócio podem contribuir com os cenários. Estes **cenários têm a estrutura *Gherkin* implementada** com código iterativo usando TDD.

Similarmente há a definição de *user stories pelo BA e os critérios de aceite pelo QA* (R14), com isto, **faz-se a compilação dos cenários para transformar em código, as jornadas** e o último passo envolve a apresentação dos cenários para os clientes. Também há o (BA, Analista de Negócios) criando *user stories* com a equipe adicionando detalhes (R11), os mesmos praticam '3 amigos', com Desenvolvedor, Analista de Qualidade ou Negócio e alguém do Negócio, sendo que o **usuário fornece exemplos para o Analista de Teste**.

Desenvolvedores e Analistas de Negócios escrevem funcionalidades e cenários e que os cenários são verificados com o cliente, sendo que quando há a conclusão da funcionalidade é valida a coerência com os cenários (R10).

Há ciclos no qual o **Analista cria documento *.feature* com cenários**, posteriormente o Analista de Teste refina esses cenários do Analista de Negócios, colocando cenários de interface e o Desenvolvedor utiliza do BDD para codificar, caso necessário efetua alterações (R23).

Há também situações em que o **PO escreve na estrutura do BDD, colocando-o no Kanban** (R1). Com isto, o Desenvolvedor pega este cartão chamado *ticket* e desenvolve, sem utilizar de *framework* de BDD. Após isto o Testador pega o *ticket* e verifica necessidade de automação, diante da frequência de uso, posteriormente a isto volta para o PO fazer a validação.

Por fim, a situação em que os **cenários são usados como contrato com o cliente** e com isto os Desenvolvedores pegam cenários e detalham para utilizar no desenvolvimento (R13). O processo ágil desde o início utiliza dos cenários. O mesmo Engenheiro reportou que em outro projeto utilizou de **BDD como parte do processo ágil, no Scrum**, assim durante a *planning* pegavam funcionalidades e definiam cenários a serem implementados.

2) Composição da equipe

De acordo com a Tabela 3.2, as perguntas 4.1 e 4.2 tiveram como objetivo entender a composição da equipe que se envolve durante o BDD. Isto para verificar quais os profissionais que realizam as atividades relacionadas ao uso desta abordagem.

Encontrou-se diversas configurações de equipes durante o BDD. Frequentemente foi mencionada a composição de equipe com **Desenvolvedor, QA e PO**. No entanto há quem ressalte que a conversa sobre dos cenários é realizada entre **Desenvolvedores e QAs** (R10). Há quem relate que o **Desenvolvedor esta fora do ciclo que envolve BDD** (R20).

Houve relatos de uso de BDD com diversos integrantes da equipe, como por exemplo **Desenvolvedores, QA, PO e *Designers***, com média de 6 pessoas integrando durante o BDD (e.g. (R12, R13, R18)). Similarmente a outra composição com **PO, Testador, Desenvolvedor e *Scrum Master***, com média de 7 pessoas interagindo da equipe (e.g. (R1, R3, R7, R9)).

No entanto, poucas vezes o cliente é inserido junto a equipe de projeto, conforme mencionado por entrevistados, os quais possuem suas equipes compostas por **Analista de Negócio, Analista de Teste, Desenvolvedores e clientes** (e.g. (R14, R23)), estes possuem uma equipe média de 7 integrantes. Muito similar a composição de equipe reportada que contém **Gerente de Produto (cliente), Analista de Negócio e Testadores** (R21).

Em contrapartida, teve-se também relatos de equipes em que apenas um papel usufruía de BDD como, em uma equipe que havia somente ***Coders*** (R5), ou seja, pessoas que faziam papel de Desenvolvedor e Testador ao mesmo tempo. Nesta mesma linha haviam equipes que se denominavam **Multifuncionais** (e.g. (R2, R11)), ou seja, pessoas tinham diversas tarefas dentro do projeto e não somente a função de desenvolver ou testar, assim todos integrantes da equipe usufruíam de BDD. Apesar de ter-se relatos de diversos papéis se envolvendo em BDD, algumas vezes o PO não se envolve diretamente com BDD, como, a composição de **Líder do Time, Engenheiro de Teste e Desenvolvedores** (R17).

3) *Frameworks* e linguagens de programação utilizados para BDD

De acordo com a Tabela 3.2, as perguntas 3.4 e 6.1 tiveram como objetivo identificar o que esta sendo utilizado de *frameworks* e linguagens de programação no apoio a realização do BDD.

Dentre os *frameworks* relatados, os mais utilizados pelos respondentes são **Cucumber** (e.g. (R2, R3, R5, R8, R11, R12)), **RSpec** (e.g. (R13, R14, R15, R20)),**Specflow** (e.g. (R2, R10, R16)) e **JBehave** (e.g. (R10, R11, R14)).

Outros *frameworks* também são utilizados, tais como: **DASpec** (R11), **Cucumber JS** (R4), **Jasmine, Protactor** (R14), **NBehave** (R2), **BehavePro** (R24), **Behave-django** (R13), **FitNesse** (R11), **Lettuce** (R14), **Calabache** (R12), **Capybara, MetaRuby e Watcher** (R2), **Spinach** (R3), **PyUnit** (R21), e **Demoiselle** (R1).

Contudo, as linguagens de programação mencionada pelos entrevistados foram: **Ruby** (e.g. (R2, R4, R13, R16)), **Java** (e.g. (R14, R18, R20, R23)), **.Net** (R1, R11, R16, R20), **C#** (e.g. (R11, R16, R20, R23)), **ASP** (R8), **JavaScript** (R4), **AngulasJS** (R22), **Python** (e.g. (R5, R7, R13, R21)), **NodeJS** (R3), **Groovy** (R19), **Scala** (R17), **Struts 2** (R15), **Java Android** (R3), **Android Studio** (R1), **objectiveC iOS** (R3), e **xCode** (R1).

4) Recursos técnicos utilizados junto ao BDD

De acordo com a Tabela 3.2, as perguntas 6.2, 7 e 8.6 tiveram como objetivo identificar qual ferramental e artefatos são utilizados por equipes que usufruem de BDD em seus projetos.

Para compartilhar *user stories* e cenários frequentemente são utilizados os *dashboards*. Estes usufruem de **Kanban**, quadro físico Kanban, para compartilhar seus cenários (R1, R21), **JIRA** para compartilhar as *user stories*, tarefas, ou critérios de aceite (e.g. (R7, R20, R22, R24)). Complementando há quem ressalte o uso do **JIRA** para controle de cenários do BDD (R18). Além destes, foram mencionados também os seguintes *dashboards*: **Confluence** (R16, R20, R22, R24), **Pivotal** (R5, R13), **Trac** (R6), **Tablero, Mingle, Eiffel** (R14), **Kanbanery** (R8), **Redmine** (R23), **Trello** (R4), **Gemini** (R10), e **Asana** (R21).

Para apoio na comunicação visual da equipe foram mencionados o uso de **protótipos de tela** (R5, R9, R23) e **mock-ups** (R4). Houveram relatos de uso de ferramentas como **Word** para edição de texto (R1, R9, R13), **Excel** para gerenciamento de cenários (R1, R2) e até mesmo **Enterprise Architect** para compartilhar as regras do negócio do projeto (R6). Além destes, editores de texto são frequentemente utilizados para criação de cenários, como, **Notepad** (R9, R23), **Sublime** (R11), **Textmate** (R11), **Subway** (R13), e **GoogleDocs** (R8, R13). Há, também, o uso, de **papel e caneta** (R22), **quadro branco** (R2, R12), e, ainda, **mapa mental** para entendimento das atividades (R1, R8).

Além destas ferramentas, há o uso de ferramentas de apoio a comunicação entre a equipe, como o **Slack** (R22) e **videoconferência e câmeras** (R2).

Para automação dos testes do BDD há relato de uso de **Appium** (R12), **Selenium UpDriver** (R7, R8), **Selenium Grid** (R8) e **JMeter** (R8) para testes de performance. Para o

controle do versionamento do código há o uso de **repositório** (e.g. (R13, R17, R21, R22)), como, **GitHub** (R2, R11, R13, R24), **GitLab** (R8, R9), **GoogleDrive** (R12), **Mercurial** (R11), e **Subversion** (R2, R7, R11).

Para integração contínua de código foi mencionado o uso de **Team city** (R2) e **Jenkins** (R7, R8, R20, R21). O Jenkins gera um relatório de progresso do projeto (e.g. (R12, R14, R16, R24)) com *status* dos testes que foram automatizados. Além disso, observou-se relatos de uso de **living documentation** (R11, R14, R16) e uso de html do *Pickles* (R2, R10) que produz esta documentação. Há ainda o relato frequente de que **não há o uso de *living documentation*** pela equipe do projeto (e.g. (R5, R15, R17, R19, R23)).

4.4 BDD e seu suporte a engenharia de requisitos (QP2)

1) Envolvimento do cliente

De acordo com a Tabela 3.2, as perguntas 5 e 8.5 tiveram como objetivo identificar o nível do envolvimento do cliente em relação aos requisitos ou cenários e quando a demanda da necessidade é interna, ou seja, quando o desenvolvimento de software não é realizado para um cliente externo, **há um *proxy* mapeando esta demanda interna** para a equipe. Este *proxy* geralmente é representado pelo **PO** (e.g. (R1, R12, R20, R24)), **Gerente de produto** (R21), ou **Analista** (R7). Ou ainda, é tratado por representantes como **UX (*User Experience Design*) e PO**, ***Agile Coach*** (R3), ou **PO e Diretoria** (R22). Um entrevistado mencionou que **quando demanda era interna havia contato com clientes** (R16).

Também, nestas situações houve relatos, algumas vezes, da **necessidade de não se ter um cliente** (R24) porque o mesmo poderia atrapalhar ao invés de ajudar no projeto. Assim como há quem tenha utilizado as definições dos cenários para projeto legado para entendimento do sistema existente e documentação deste comportamento para que o mesmo se mantenha ao longo do tempo (R4). Também um outro entrevistado ressaltou que utilizou de BDD, de forma experimental, somente entre os Desenvolvedores para entendimento da abordagem (R6).

Houve diversos relatos de demanda externa, quando as necessidades vem de um cliente externo a própria empresa. Nesta situação, diversos casos como **não haver a relação com o cliente, mas apenas com o *proxy*** (e.g. (R15, R19, R23)), como, POs (R9, R16) ou 1 UX e 1 PO (R18).

Entretanto, há situações em que o **cliente participa da confecção dos cenários** (R10, R11). Também, algumas vezes os **clientes participam, mas depende da complexi-**

dade do cenário (R2) para haver essa interação. Além disso, situações em que **os clientes geram exemplos** (R14) para serem utilizados nos cenários. Algumas vezes o **encontro do cliente com a equipe de desenvolvimento e teste é eventual**, como por exemplo, encontro apenas na *demo* que ocorre a cada 10 dias (R17).

Há circunstâncias em que o ***proxy* utiliza de filtragem de informações fornecidas por usuários** para transformar isto em demanda para a equipe de projeto, por haver um número exorbitante de Usuários/cliente. Assim, há um sistema que coleta de possíveis sugestões e um *proxy* captura estas possíveis necessidades, através da coleta do Suporte (R8) ou via *Tickets* e *User voices* (R1), aberta pelos usuários, ilustrado na Figura 4.1.

2) Papéis envolvidos na criação dos cenários

De acordo com a Tabela 3.2, a pergunta 8.1 teve como objetivo identificar quais membros se envolvem na criação dos cenários que são usados no BDD.

Há situações em que BDD é inserido em somente uma etapa do desenvolvimento de software e como tal usado **por apenas um papel**, como o uso somente para os testes por QAs (R3, R20, R24) ou por Analistas de Teste (R7, R9, R12) os quais centram-se na criação de cenários para teste do sistema. Também há o uso somente por Desenvolvedores (e.g. (R3, R4, R5, R13, R15, R17)), os quais criam os cenários para validar se os seus desenvolvimentos conferem com o comportamento do software desejado.

Há situações que a **criação de cenários envolvem dois papéis**, como engajamento de Analistas de Negócio ou Qualidade com Desenvolvedores (R10). Também, há a interação entre QA com PO mesmo que dos Desenvolvedores não participarem ativamente do processo de criação dos cenários (R22). Além destes, há situações em que os Desenvolvedores e os Analistas de Negócio definem os cenários com base nos critérios de aceites que eles mesmos definiram (R19).

Há poucas situações em que **três papéis se envolvem para a criação dos cenários**, como a situação que engaja Analista, Desenvolvedor e QA - conhecida como '3 amigos' (R11). Neste sentido entrevistados mencionaram que **todos da integrantes da equipe participam da criação dos cenários** (R2, R16). Numa situação similar há o envolvimento entre **BAs e QAs (*Quality Assurance*) escrevendo cenários com Desenvolvedor validando esta escrita, em *Gherkin*** (R14). Assim outro entrevistado reportou que há **Analista de Teste conversando com PO e Desenvolvedor - '3 amigos'**, ou seja, a criação dos cenários é realizada primeiramente pelo Analista de Teste da equipe, após isto é realizada a conversa com PO e Desenvolvedor para melhorar estes cenários ou fazer alterações dos mesmos (R8).

Há a **reescrita de cenários entre os papéis que não se envolvem durante a construção dos cenários** (R23). Esta disse que Analista de Negócios escreve cenários e que os Analistas de Teste complementam estes cenários com detalhes de interface, assim como os Desenvolvedores. Também, há relatos que PO efetua a criação dos cenários para os demais integrantes consumirem estes cenários, fazendo as alterações necessárias (R1, R18). Assim como há a situação em que Gerente de Produto e Engenheiros de Software escrevem os cenários, sendo que a primeira versão dos cenários é escrita pelo Gerente de Produto, chamada de critério de aceite, contendo *Gherkin*, e, posteriormente, os Engenheiros complementam com detalhes para a realização da implementação destes cenários (R21).

3) Coleta e referência para a escrita de cenários

De acordo com a Tabela 3.2, as perguntas 8.2 e 8.3 tiveram como objetivo identificar como é feita a coleta dos cenários para o BDD, qual tipo de informação e como estas informações ser tornam em cenários para o BDD, para identificar como esta sendo realizado este processo de escrita desses cenários na prática.

Há estímulo para **encontros com regularidade, através de reuniões**. Estas são realizas entre diversas configurações, como reunião com equipe de negócio para a coleta das informações para a escrita dos cenários (R21). Assim como, Desenvolvedores com PO (R13), Desenvolvedor com Analista de Negócios (R19) e Desenvolvedor com Analista de Negócios (R4). Isto proporciona um conhecimento do negócio para a equipe, (R2, R16).

Há situações em que **diversos integrantes da equipe participam de encontros para a coleção e refinamento dos cenários** com interação entre QA, PO e Desenvolvedor (R8). Assim como foi mencionado que QA reúne-se com PO para pensar nos caminhos felizes (R22). Há situações em que o PO reúne-se com setores internos para averiguar a necessidade da demanda interna (R1). E, também, há situações em que o **PO coleta do Suporte as possíveis alterações do sistema** (R1, R8). Algumas vezes, **há o engajamento do cliente** através de entrevista entre PO e com o cliente (R17).

Há diversas práticas de incentivo à coleção de cenários, como *workshop* de escrita com os clientes lançando ideias para a equipe (R14), e, também, *backlog grooming* com BA com as *story cards* (R16). Complementando quando o software está em desenvolvimento há a interação entre o Engenheiro de Testes com Desenvolvedor e Testador para alinhamento de cenários (R16). Há situações em que a equipe utiliza ***Planning* para coletar os cenários** (R24).

Outra prática de coleta é a apresentação de **exemplos com a equipe** (R10, R11, R14). Estes buscavam a captura de informações dos usuários e exemplos reais do negócio.

Neste sentido, um dos entrevistados ressaltou que o **Analista de negócio escreve *user stories* e os cenários vem da conversa posterior a esta definição, gerando cenários com todos da equipe** (R10).

Há um movimento forte para a escrita colaborativa de *user stories*, para entendimento dos testes unitários e dos testes de aceitação (usando o BDD), através de conversas e reuniões para buscar entendimento sobre as necessidades do sistema (R2):

> *"A partir do trabalho de escrita das user stories e dos requisitos colaborativamente amadurecendo as informações ao longo do processo, então no início tinha uma, essas reuniões para entender, de entendimento, de refinamento, de user story e depois se iniciava um trabalho de desenvolvimento da escrita de testes unitários e de testes de aceitação utilizando o BDD." (R2)*

Diversos relatos que usufruem de **critérios de aceites já definidos para escrita dos cenários** (e.g. (R18, R20, R24)). Um projeto experimental com BDD usufruía de tópicos fornecidos por Analista (R6). Também, os **critérios de aceite são definidos também por PO e UX** (R7, R12, R18). Há, também situações em que é realizado um **comportamento de alto nível reportado em (*user stories*)fornecido pelo PO e que a partir destas são extraídos cenários** (R5, R17, R19).

No entanto, há situações em que o Analista de Negócios interage com um Comitê e **gera documentação em linguagem natural e alguns exemplos, dando o embasamento para desenvolver os cenários a partir desta documentação** (R15). Neste sentido há situações em que os **cenários para o BDD são coletados a partir de definições de cenários principais gerados anteriormente por Analista de Negócios** (R23).

Também, há o uso de **protótipos e Regras de Negócios do sistema** (R9). E, também, há situações que tem-se que definir cenários para um software já é existente, realizando a **definição para um comportamento de software existente** (R4).

4) Momento de definição dos cenários

De acordo com a Tabela 3.2, a pergunta 8.4 teve como objetivo descobrir em qual momento do ciclo de vida do projeto os cenários para o BDD são definidos. Isto com intuito de identificar o momento que as definições dos cenários estão sendo realizadas.

Mesmo a Engenharia de Requisitos ser contínua quando imersa em contexto Ágil, é frequentemente mencionado que **os cenários para o uso no BDD são definidos antes do desenvolvimento** (R3, R5, R10, R11, R20). Sendo que ocorre o planejamento desses cenários antes de efetuar o desenvolvimento dos mesmos.

Por exemplo, a apresentação dos cenários são realizadas em *Planning*, ou seja, há a escrita dos mesmos anteriormente pelo PO (R1). Assim como há está escrita no início do ciclo (R18). Além destes, foi declarado que a **confecção dos cenários tinha início durante o planejamento do ciclo**:

> *"Desde a tomada inicial de como o projeto vai ser feito, ele [PO] já começava a desenhar histórias com aquela nomenclatura com uso de expressões muito parecidas com a que usa no BDD clássico, de escrever "dado um cenário, como um sistema deve se comportar." (R13)*

Há situações que a **confecção dos cenários para o BDD se adaptam aos ciclos, sendo realizados na etapa anterior o planejamento para a próxima etapa**, como a criação dos cenários na semana anterior ao desenvolvimento (R22), na *sprint* anterior com cliente (R23), em encontros como *Desk-check* que são dedicados para confecção dos cenários (R14, R19).

Também, houve o relato de que os **cenários são confeccionados simultaneamente ao desenvolvimento** (e.g. (R2, R6, R7, R17)). Entrevistados complementaram informando que quando há a liberação da demanda pelo PO já se inicia o desenvolvimento dos cenários (R8, R24).

Há situações em que há a **definição dos cenários em dois momentos, no planejamento e ao londo do desenvolvimento**, antes da *sprint* e no *backlog grooming* (durante o desenvolvimento), (R16). Também um entrevistado reportou que a criação dos cenários é realizada no momento que o item é adicionado e depois durante o desenvolvimento (R21).

Há quem tenha mencionado que descreve cenários **após o software desenvolvido** (R4), como:

> *"Hoje eu [desenvolvedor] estou usando o BDD, no sentido de fazer, duas coisas, uma para descrever o comportamento do sistema, mas, também, para ter um entendimento melhor, e para esse entendimento continue mais para a frente" (R4)*

5) Verificação dos cenários

De acordo com a Tabela 3.2, a pergunta 8.7 teve o objetivo de verificar quem fica responsável por garantir que os cenários descritos estão realmente alinhados com as necessidades do negócio. Isto para identificar como é realizada a verificação dos cenários que serão descritos para o BDD.

Há situações em que **somente um papel fica responsabilizado por garantir que os cenários estão alinhados com as necessidades do negócio**. Por exemplo, alguns QAs validam os cenários (e.g. (R20, R22, R24)) ou Analistas de Teste (R7, R9). No entanto, estes não chegam a verificar os cenários, somente validam se histórias estão implementadas:

> *"Eles [QAs] testam várias vezes isso com base nos critérios de aceite que foram escritos, depois disso é feito um review que a gente chama o nosso PO que é quem aceita a história, tipo, 'é isso mesmo'. E a base disso são critérios de aceite sobre a história que foi escrita." (R3)*

Além da validação, há Desenvolvedores (R6, R13) ou Líderes Técnicos da equipe (R5) que ficam responsáveis pela verificação dos cenários. Estes desenvolvedores também contam com a validação do PO ou Analista responsável, de maneira similar a situação mencionada anteriormente (R6):

> *"O PO não cita os cenários próprios que a gente [desenvolvedores] escreva, que estavam lá no Git. Ele não cita esses cenários aqui é X, esse aqui é Y, não existe esse tipo de conversa. Ele [PO] usa como base, mas ele faz uma tradução para uma linguagem que talvez fique um pouco mais fácil dos clientes entenderem." (R6)*

Além de somente uma pessoa verificar cenários, também há situações em que **há mais de uma papel responsabilizado por esta atividade**. Como, por exemplo, QAs e posteriormente PO e UX validando cenários (R18), ou Analista de Negócio verificando os cenários (R10, R14, R19) ou 1 PO e 1 BA (R16), ou, ainda, um PO verificando a escrita (R1, R12):

> *"Depois que a gente [equipe] acaba de escrever os cenários a gente faz o test checking, chama o Analista de Negócio para revisar e meio que aprova 'é isso mesmo'." (R19)*

Há **situações em que o cliente se envolve para verificação e validação dos cenários** (R23,R10). Há também relatos de situações em que **todos membros da equipe validam cenários** (R4, R17). Há a responsabilidade da garantia dos cenários atribuída ao PO, QA e Desenvolvedor (R8) e QA, Desenvolvedor e BA (R11). Os cenários automatizados garantem validação dos cenários (R2):

> *"A gente utiliza esses cenários para garantir que a aplicação continua funcionando ao longo do desenvolvimento para ter certeza que a gente desenvolveu exatamente o que foi solicitado pelo negócio ou pelo cliente então não tem uma etapa de validação após a criação dos cenários." (R2)*

6) BDD como apoio a especificação de requisitos

De acordo com a Tabela 3.2, as perguntas 8.8 e 8.9 tiveram como objetivo identificar as opiniões dos entrevistados quanto a apoio do BDD nas especificações realizadas por eles.

O BDD apoia a especificação dos requisitos. Foi reportado que com BDD há **melhor entendimento dos requisitos** e auxilio em **expressar os requisitos** de software (e.g. (R3, R18, R23, R24)). Há relatos de que a linguagem usada no BDD facilita o entendimento (R1, R7, R19, R20), até mesmo o entendimento do cliente (R23). Conforme ilustrado a seguir:

> *"BDD é muito bom pra entender o que que o cliente quer, é uma linguagem que todo mundo vai entender, inclusive é muito mais fácil de você olhar o cenário que você tem que ter, o que você está fazendo e o que que você tem que esperar, do que você ler uma carta praticamente, uma bíblia, você lê um livro para entender o que o cliente quer, então fica muito mais fácil mesmo." (R18)*

> *"Para o cliente também, eu acho que é importante, por que as vezes um simples requisito escrito ali fica muito difícil dele entender onde ele está dentro do sistema, e o BDD soluciona esse problema. Eu acho que a comunicação, a interpretação fica muito mais fácil no uso dele." (R23)*

O BDD expressa as necessidades do cliente quando há o alinhamento do cliente com a equipe (R9). Um entrevistado ressaltou que precisa haver a **comunicação e colaboração para que o BDD expresse as necessidades do negócio** (R22). Complementando, um outro entrevistado ressaltou que **ao longo do projeto o cliente vê valor do BDD para expressar suas necessidades** (R14).

Em contrapartida, há quem mencione que BDD não expresse as necessidades do cliente, e que ressalte que BDD é mais uma questão de estilo para escrever os cenários de teste (R5). Enquanto, ou outro entrevistado, relatou que o BDD expressa os requisitos, e complementou que TDD não abrange o que o BDD abrange (R13).

BDD gera uma **documentação fácil para criar software** (R17), por apoiar a padronização da escrita (R13), criando uma documentação de leitura fácil (R12). Conforme ilustrado a seguir:

> *"(...) gera uma documentação que fica muito fácil de traduzir para o software, na minha opinião." (R17)*

Um entrevistado reforçou que para o BDD devem ser usadas **palavras do negócio** (R11). Assim como outro entrevistado informou que **às vezes, a escrita torna a compreensão difícil** (R4):

"(...) deixando um teste tão específico que você não consegue reutilizá-lo, e talvez até torna difícil a compreensão para as pessoas que estão mais ligadas as negócio." (R4)

Foi ressaltado que **BDD ajuda na especificação dos requisitos** (R13, R24). Complementando a **escrita dos cenários com todos integrantes juntos já ajuda** (R24). No entanto houveram relatos que só há suporte se os cenários forem **bem escritos, usando *domain language*** (R1, R4, R11). E, um entrevistado, complementou, ainda, que **não é prático a definição de cenários para todo o sistema** (R10).

Há relato que BDD suporte a especificação dos cenários quando se mapeia para a funcionalidade e não para uma tarefa (R16). Além disso, foi complementado que **BDD apoia a especificação, mas conta com fragmentos de *user story*** (R8, R19, R20). Um entrevistado mencionou que BDD apoia a especificação, mas depende do que esta especificado, pois Desenvolvedores sentem falta de informações de arquitetura (R20).

Um dos entrevistados ressaltou que **BDD apoia a especificação, ainda mais quando contém o cliente envolvido durante especificação** (R14). Também, um outro entrevistado reportou que **BDD apoia a especificação quando há necessidade de processo de qualidade do produto** (R2):

"É sempre os mesmos exemplos, hospitalar ou algo que controla tráfego aéreo, essas aplicações se beneficiam de uma série de processos de qualidade que qualificam o produto em função do contexto que elas existem então a utilização de BDD, de TDD, de ter uma estratégia de teste bem definida." (R2)

Foi reportado que para **casos mais complicados, é possível realizar a definição dos requisitos** (R23). Entrevistados relataram que BDD apoia a especificação da camada não funcional (R5), requisitos funcionais (R15), e requisitos de *frontend*, funcionais e não funcionais (R21).

No entanto, houve relato de que **requisitos de performance são complicados de definir com BDD** (R21). Assim como foi relatado que **quanto mais baixo nível mais complicado** (Q22):

"Só que você conversar com alguém de firmware é tão baixo nível quanto você falar 'olha, o buffer de memória, memória do hardware'. Eu acho que são coisas tão técnicas que eu não acho legal, porque às vezes quem vai demandar aquilo, vai bater o olho na escrita em um nível tão baixo." (R22)

4.5 Benefícios, dificuldades e recomendações sobre o BDD (QP3)

1) Benefícios do BDD

De acordo com a Tabela 3.2, as perguntas 9.1, 9.2 e 9.3 tiveram como objetivo identificar os benefícios do BDD, como o BDD contribui para a melhoria na qualidade do produto de software e como o BDD auxilia nos problemas enfrentados no desenvolvimento de software, assim visa-se identificar quais as contribuições que o BDD trás consigo.

Há diversos benefícios levantados quando há uso de BDD para o processo de desenvolvimento de software. Entre eles: o **entendimento** (e.g. (R1, R7, R16, R23)) que é proporcionado pelo suporte do BDD ao conhecimento da aplicação (R2), a **clareza do que deve ser feito** (e.g. (R8, R15, R18, R20)), tornando objetivo o que o projeto deve fazer (R13), a **maior visibilidade do negócio** (R14), alinhando com a expectativa da entrega do projeto (R2). Isto, possibilita, também, a **capacitação mais rápida de novos integrantes na equipe**:

> *"O Desenvolvedor saber o que ele tá fazendo, isso é muito importante, com isso ele vai fazer o trabalho dele muito melhor, do que simplesmente se um trabalho mecânico" (R23)*

> *"Tem a questão de saber se o valor que é esperado é entregue e tem a questão de saber o que está sendo entregue corresponde ao que foi pedido porque muitas vezes a gente tem a brincadeira de telefone sem fio, de uma pessoa passar a necessidade para outra e acabar entregando algo completamente diferente." (R2)*

Além de proporcionar o entendimento, há também a **documentação viva** (e.g. (R11, R16, R17, R22)). Tem-se ainda relato de **maior colaboração** (R9, R21), com **compartilhamento entre visões da equipe** (R18, R19), e quando o **contexto é distribuído, o mesmo oferece comunicar mais na hora de criação do cenários** (R10). Além disso, verificou-se a **facilidade de comunicação com o cliente** (e.g. (R1, R4, R7, R12)):

> *"A comunicação com o cliente fica muito mais fácil, porque quando você tenta falar com o cliente com termos técnicos a conversa não flui." (R14)*

Também, há a **melhora da conversa entre Desenvolvedores e Analistas** (R10),. Assim como ilustrado a seguir:

"(...) traz mais a colaboração, ele meio que tira o 'buraco' que tem entre o Analista de Teste e o PO, entre o Analista de Teste e o cliente, o Analista de Teste e o desenvolvimento." (R7)

Além disso, há o **rastreamento dos cenários** (R18) listado como benefício, e organização dos testes de alto nível (R11). Também, foi constatado o que **em relação ao TDD, o BDD é mais produtivo** (R15). Também, há o aumento da produtividade e segurança quanto ao funcionamento do sistema (R24), **aumentando a confiança no sistema** (R4). Visto que o uso de teste em si traz este benefício (R5), diminuindo a chance de erro do sistema (R3), ajudando na prevenção dos defeitos (R7). A **automação que o BDD proporciona se torna uma vantagem** (R15, R16, R23). Assim, consequentemente diminui-se o número de *bugs* (R14, R24), conforme ilustrado:

"(...) eliminação de desperdício de tempo das pessoas, eliminação/redução de bug por não conformidade do software." (R24)

Diante da melhora na qualidade do produto, constatou-se a **redução de *bugs*** (e.g. (R3, R8, R9, R14, R20)), corroborando para **redução do retrabalho** (e.g. (R2, R7, R16, R24)), com a prevenção de problemas (R21). Complementando um entrevistado mencionou que mesmo **sem automação, há antecipação dos problemas** (R23).

Além disso, a **automação garante a qualidade** (R5, R22, R23), **ajudando na regressão** (R13). Assim, os cenários criados no BDD são úteis para automação (R7, R21), conforme ilustrado:

"Ajuda muito em regressão, como eu usava pra testes e era um teste de nível de aceitação, nível já do navegador, automaticamente fazendo as interações que a gente definia nos cenários, nas funcionalidades de BDD, ele pegava coisas que o TDD não conseguia pegar no meu projeto." (R13)

Assim, há a redução de problemas frequentemente enfrentados pelo time de desenvolvimento, como a **redução de documentação** (R7, R8), evitando o desperdício de tempo investido para gerar documentação (R14, R24). Há o uso de **documentação atualizada** (R9, R10, R11, R18), *living documentation* do BDD. Um entrevistado ressaltou que esta documentação é de leitura uniforme, fazendo que haja uma única forma de leitura dos requisitos que estão sendo implementados (R16).

Assim, um entrevistado expôs que com BDD há a **padronização da escrita** e isto viabiliza o entendimento de todos (R23). Além disso, um entrevistado declarou que BDD é interessante para **evitar a ambiguidade** do entendimento dos requisitos, que ocorre entre Desenvolvedor e PO, pois desde o início é trabalhada a criação dos cenários com a equipe (R18).

Há o **entendimento dos requisitos** (R1, R2, R5, R8, R23, R24), promovendo a clareza da necessidade (R11, R17, R18) e pensamento do comportamento antes do desenvolvimento do software (R4, R7, R10, R18, R19).

Também foi reportado que **antes do BDD havia muitas falha na comunicação** (R1, R17). Dois entrevistados mencionaram que com BDD há **quebra de barreiras para a comunicação, pois há envolvimento dos integrantes da equipe** (R20, R22). Assim mencionaram:

> *"Havia muita falha na comunicação, tipo, 'o que tu quis dizer com isso e isso', dava um determinado problema neste sentido, agora não mais." (R1)*

> *"Já tinha visto do PO especificar e mudar a documentação no meio da sprint e a gente acabar atendendo uma coisa que não é mais verdade." (R17)*

Reforçando, há diversos relatos de que BDD **melhora na forma de comunicar** (R1, R7, R10, R12, R19), conforme ilustrado:

> *"Critérios de aceitação teriam Given, When, Then porque nós como equipe acreditávamos que esse formato melhorava a comunicação entre a equipe." (R19)*

Além disso há a **facilidade de identificar o problema** (R2, R18), conforme:

> *"Como a gente consegue usar ele [cenário] no meio do código e as pessoas conseguem rastrear os cenários e o código da regressão, então acaba ficando muito mais fácil pra vida de todo mundo, no geral. Porque antes do BDD você criava o código dos testes lá e os cenários ficavam perdidos em outro lugar, então agora você pode ter tudo num lugar só" (R18)*

O uso do BDD trás também o **auxílio do entendimento para manutenção futura** (R15), diante do **incentivo a clareza do escopo** (R21), conforme relatado:

> *"Se ele [PO] está mais envolvido na definição dos cenários, comportamentos, fica muito mais fácil da equipe de desenvolvimento acertar o que ele está querendo dizer. Porque no fim das contas ele [PO] entende mais que todo mundo o que as coisas devem fazer." (R17)*

> *"Especificações via cenários, critérios de aceite, ajudam não só a esclarecer o escopo sem muita margem para diferentes interpretações, mas também ajudam a dirigir o fluxo de desenvolvimento, automação dos testes e a homologação das histórias" (R21)*

Além disso, o entrevistado R17 ressaltou que o BDD **incentiva a colaboração entre os papéis da equipe**:

> *"O BDD é uma ferramenta que acaba eliminando muitas das barreiras ainda presentes nas equipes ágeis, por exemplo, colaboração dentre os diferentes papéis da equipe o tempo todo, ou melhor, o mais cedo possível no processo de desenvolvimento." (R21)*

2) Benefícios advém de metodologias ágeis ou do BDD

De acordo com a Tabela 3.3, a pergunta 12 teve o objetivo de identificar se os benefícios listados advém de metodologias ágeis ou da adoção do BDD em si.

Há um entrevistado que ressaltou que ao **realizar o BDD a comunicação se torna melhor** (R19). Uma outro expôs que **sem o BDD não haveria a linguagem comum entre todos os integrantes da equipe** devido o BDD exigir conversas sobre o comportamento do sistema (R7, R11):

> *"BDD exige tu falar sobre o comportamento da sistema, sem precisar pensar em como tu vais implementar. Isto ajuda o fluxo da comunicação entre pessoas, focado no sistema e pessoas focado em negócio. O foco é nos usuários e nos benefícios." (R11)*

Um entrevistado expôs que **comunicação efetiva veio com uso do BDD**, conforme:

> *"(...) começamos nos projetos do dia 1 com cerimônias [práticas] Ágeis, mas BDD só adicionamos depois. Então ficou claro que os benefícios [como: comunicação efetiva veio com uso do BDD] eram do BDD e não das cerimônias." (R10)*

Assim, há a opinião de que o BDD **estreita a comunicação, mantendo o fluxo de informações** (R17). Complementando um outro entrevistado reportou que BDD proporciona uma **documentação que sempre fica atualizada** (R11). Assim como, há ressaltado que **sem o BDD seria mais custoso para o projeto, pois teria que ter muito mais comunicação para contextualizar equipe** (R23). Há relatos que **antes do BDD não havia padronização da documentação, assim a comunicação era onerosa** (R12).

Além disso um outro entrevistado reportou que BDD facilita na comunicação com o cliente e viabiliza a geração de documentação fácil e enxuta pra cliente (R14). No entanto

um entrevistado informou que ele conseguiria ter os mesmos benefícios sem o BDD, dependendo do contexto, mas no projeto o qual ele esteve alocado, de auditoria e de melhoria da qualidade dos testes, ele só vê os benefícios associados ao uso de BDD pelos seus contextos (R10).

Um entrevistado ressaltou que sem o BDD haveria **maior número de *bugs*, dúvidas e tempo para discutir as coisas** (R24). Complementando um outro entrevistado reportou que sem o BDD poderia ter os mesmos benefícios que se tem com o BDD, no entanto, teria mais defeitos por falta de entendimento (R11). Adicionando alguns entrevistado ressaltaram que o BDD ajuda na clareza, levando o melhor entendimento da equipe (R13), faz que haja clareza nas regras (R15), compartilhando as regras com todos integrante, fazendo que todos tenham a mesma visibilidade do comportamento do sistema (R19).

Há outro relato que **sem o BDD se conseguiria manter os benefícios, mas não com tanta eficácia** (R1). Neste sentido um entrevistado reportou que **critérios de aceite não são suficientes** (R24). Um outro expôs que teria alguns benefícios **similares ao de BDD se utilizar TDD bem escrito** poderá se atender algumas necessidades (R2). Neste sentido há quem tenha informado que **utilizaria outros artefatos, mais técnicos, para ter essa clareza do BDD** caso não houvesse o BDD (R22). Assim como outros entrevistados ressaltaram que **se não houvesse BDD teriam que criar outra forma de especificar requisitos** (R18, R24).

Contudo, um entrevistado ressaltou que **Ágil não se preocupa com o que BDD dá ênfase** (R16):

> *"Creio que não Ágil não tem os mesmos benefícios de BDD, porque eu creio que o BDD faz parte de complementar o Ágil nessa parte, que é a parte que o Ágil não se preocupa de maneira nenhuma com essas partes que o BDD foca. O Ágil é muito mais focado em mudanças e como gerenciar, como planejar, em si, do que, digamos, a descrição, do que esta sendo feito, por isto que o Ágil não é só usado para desenvolvimento de software, mas qualquer projeto em si." (R16)*

Houve relato de que **Scrum e BDD são complementares**, e que BDD é complementar as cerimônias, práticas ágeis, fazendo com que BDD dê mais agilidade ao processo de desenvolvimento de software (R1). No mesmo sentido outro entrevistado reportou que **BDD dá suporte para 'atender' os princípios do manifesto ágil**:

> *"Os princípios do manifesta ágil são muito claros, porém sem utilizar algumas práticas diárias como o BDD fica muito difícil para uma equipe atender os princípios do manifesto. Um dos principais problemas do Scrum, na minha opinião é o fato de ele não recomendar nenhuma prática técnica, mas sim só o arcabouço de projeto e reuniões levando muitas equipes a continuar usando as mesmas técnicas de desenvolvimento cascata dentro de iterações de 2-4 semanas." (R21)*

Complementando, **Ágeis continham problemas de comunicação e com BDD isto melhorou bastante**, conforme ilustrado:

> *"Os métodos ágeis já eram utilizados faziam 5 anos, mas mesmo assim existiam problemas de comunicação nos projetos. Depois da adoção de BDD no projeto de Intranet, essa comunicação entre a área cliente, Desenvolvedores e Testadores melhorou bastante." (R7)*

4) Dificuldades para uso do BDD

De acordo com a Tabela 3.2, as perguntas 10.1, 10.2 e 10.3 tiveram como objetivo identificar dificuldades e melhorias da adoção do BDD no desenvolvimento de software.

Há dificuldades mencionadas para o BDD, como a carência em **mostrar valor do BDD** (R24), em **especial para o cliente** (R3, R14), e quando há **resistência de adoção em projetos legados, pelo custo de automação** (R16); e, também, por **deixar o desenvolvimento mais lento, devido ter as definições** (R3, R19), e isto, implica em tempo de desenvolvimento e consequentemente em custo. Também há a dificuldade na **capacitação do cliente de como fazer cenários** (R2), na **compreensão da equipe do que é BDD** (R15), na **distinção entre TDD, BDD e teste unitário** (R15); em **encontrar material/relatos de uso** (R6), pois muitas coisas não são claras, havendo diversas formas de fazer o BDD (R18).

Tem que saber definir o que será automatizado, devido a lentidão em rodar toda suíte de testes (R2, R20). Diante disto, um Desenvolvedor, reportou que a manutenção do suíte de testes ao longo do tempo deve ser realizada para que não se torne um suíte muito demorado para execução (R21).

Outra dificuldade é **na escrita dos cenários** (R1, R21, R23), para formato BDD (R12), tendo dificuldade para a **definição de *Gherkin*** (R16). Há relatos que alguns integrantes da equipe tiveram dificuldade em ver o comportamento (R7, R11).

> *"mudar essa dinâmica de se for desenvolvedor, do analista de qualidade, do analista de sistemas, todo mundo conversar junto, sentar para conversar realmente, gastar o tempo que tem que gastar para entender os cenários e mudar um pouquinho aquela questão de jogar documento pra cá e pra lá e eu faço a minha parte e tu faz a tua parte" (R10)*

> *"algum dos elementos do time não usou ainda BDD, parece um pouco mais difícil de convencer eles a usar, de seguir, de dar manutenção." (R13)*

"Talvez a dificuldade era fazer a tradução dos steps no começo pra ações no código. Quando tu desenha o cenário, antes de meter a mão no código, tá tudo bem. Mas como é que eu traduzo isso?" (R12)

"dificuldade, as vezes, principalmente no começo, digamos, de criar features e tentar utilizar o Gherkin e não ter experiência disso e não ter a visão de como quebrar, como olhar para um requisito" (R16)

"Outra dificuldade é fazer o cliente entender o valor que tá ali por trás, por a gente estar lá sentando para pensar em cenários, de pensar em exemplos, isso é muito difícil no começo" (R14)

"Para o analista de negócio do projeto é muitas vezes difícil escrever cenários que podem ser facilmente automatizados pela equipe técnica." (R21)

Além dessas, ainda há a dificuldade em **definir quais cenários são úteis para escrever** (R14), ou seja, maturidade para entender os cenários (R22), pois os cenários podem se tornar repetitivos (R23), e tem que ver as situações nas quais os mesmos possam ser reaproveitados (R13). Sendo que foi considerado que há diferença dos cenários de análise para a implementação (R14), assim há a **tradução dos *steps* para o código** (R13), além de um dos entrevistados ressaltar que o **uso de *mock-up* para definição de cenários inviabiliza a refatoração** (R15).

Devido aos cenários ficarem dentro do código, permitindo a edição a qualquer momento deve-se ter cuidado para que os Desenvolvedor não alterem os cenários, "*fazendo com que os mesmos fiquem desalinhados ao que foi definido pelo negócio*"(R11).

Descrição distorcida de cenários, fazendo com que os mesmos não façam sentido (R4, R5), podendo ocasionar o não entendimento do que vale ser testado (R5, R11). Além disso, há a **má utilização das ferramentas** (R15), como, por exemplo, encarar o **BDD como Selenium** (R4).

Uma dificuldade foi a questão de **cultura, em manter a equipe dinâmica** (R10, R4). Outra dificuldade sentida foi em **fazer que as pessoas leiam toda a documentação** pois há situações em que só uma parte da equipe está engajada com o uso do BDD (R12).

Assim, é **difícil fazer que todos usufruam de BDD** (R13), fazendo que o Desenvolvedor esteja do lado do QA (R8). Ainda, que haja a **participação da equipe do negócio** para construção de cenários (R7, R17).

Além disso para o **PO a definição do Desenvolvedor já é muito técnica** havendo ainda a distância entre as áreas (R13). Também há quem ressalte que o PO negligencie a definição dos cenários e escreva histórias superficiais (R24).

5) Sugestões de melhorias para o uso do BDD

Há diversas sugestões de melhorias, como **maior divulgação do BDD** (R3, R6), **ter mais estudos sobre ele** (R23), **estudo que mensurem benefícios do BDD** (R24), em especial divulgação de como o BDD interage com outras metodologias (R16).

Neste sentido, para a melhora do BDD, observou-se que seria interessante encontrar forma de cenários estarem detalhados, mas não longos (R18), e quando BDD estiver muito grande, ter alguma forma de reduzi-lo, com reaproveitamento (R1).

Também, foi mencionado que uma melhoria seria o **estímulo do pessoal para contribuir com cenários** (R9, R22), como, por exemplo, levar o PO para o lado do Desenvolvedor (R5). Complementando, um dos entrevistado ressaltou que **cada equipe vai ter uma adaptação** para utilizar do BDD (R8).

Além disso, **seria interessante entender em qual contexto se utiliza de BDD, auxiliando entender quando realmente é aplicável** (R2). Também, entender qual cenário seria importante para uso do BDD, centrando no fluxo dos usuários no software ressaltou um dos entrevistados (R21).

Sugere-se que se **esclareça alguns pontos sobre a aplicabilidade do BDD**, com **divulgação sobre BDD** (R6, R17, R12, R4). Foi mencionado que há uma **carência em debater o assunto BDD** (R24) fazendo que haja o entendimentodo que realmente é o BDD (R9, R22). Também foi mencionada a carência de **material em português** (R6). No mesmo sentido constatou-se carência de **treinamentos de BDD** (R17), que permita capacitar o pessoal para entender a escrita para o BDD (R12), com recomendações de como escrever os cenários não funcionais (R21), auxiliando também na capacitação, pois foi reportado que diversas pessoas pessoas confundem TDD, FDD com BDD (R2).

Outra melhoria esta relacionada aos ***frameworks* que apoiam as atividades do BDD**, como: **mudança de palavras BUT e AND** (R14), **não restringir somente ao *Gherkin*** (R10), compressão da execução dos testes, cobrindo diversos cenários (R19). Neste sentido, um dos entrevistados ressaltou que a ferramenta DASpec tenta misturar *Gherkin*, fitnesse e concordian, mas sem seus problemas (R11). Foi mencionado que **dá a necessidade de se ter entendimento sobre ferramentas** (R14) e entendimento do valor de automação (R15). Assim, há a **questão cultural que deve reaver suas condutas para adoção do BDD** (R3, R23), eliminando preconceitos entre negócio e desenvolvimento (R13, R10).

6) Recomendações a quem deseja adotar o BDD

De acordo com a Tabela 3.2, a pergunta 9.4 teve como objetivo entender quais as recomendações feitas pelos profissionais entrevistados a quem deseja utilizar o BDD.

O BDD é recomendado em diversas situações, como quando se deseja ter **tradução simultânea do negócio para código** (R13), devido proporcionar **visibilidade do negócio** (R13) a equipe. Além disso, recomenda-se para **facilitar a colaboração** (R21) entre os membros da equipe, fazendo com que haja pessoas do negócios mais próximas do projeto (R24, R14, R19). Já um dos entrevistados ressaltou que seria interessante o uso de BDD em projetos que tenham usuários conhecidos.

Também é recomendado o uso de BDD **quando há muitos *bugs* por falta de entendimento** (R12, R18). Assim como, foi recomendado o uso do **BDD quando há dificuldade em entender o problema** (R6), ilustrado a seguir:

> *"Eu recomendaria principalmente se tu tiveres tendo muitos bugs por falta de não entendimento dos requisitos que foi o nosso caso de sucesso, assim ó, tinha bastante não entendimento dos requisitos e depois que a gente usou o BDD, todos entenderam melhor o que era pra fazer." (R12)*

> *"Então havia muitos bugs que a gente acabava pegando porque justamente ninguém queria ler esses documentos de 4 páginas e o BDD acabou melhorando isto, as pessoas estão lendo mais, lendo realmente o que é que tem que ser feito, tentando fazer focado." (R18)*

Recomenda-se o uso do BDD **quando há cenários complexos** como em cotação de seguro (R11). Além disso foi recomendado que se deve utilizar **BDD para cenários** e não para todas as funcionalidades (R2). Além dele, um outro entrevistado expôs a importância de **evitar o uso de BDD a nível de interface** para que o desenvolvimento não fique só a nível de interface (R10). Um dos entrevistados recomenda **foco em *Given-When-Then* executável** para ser útil para o código em si (R19).

Outro entrevistado mencionou que é saudável o uso de testes (R3), e um outro relatou que BDD é **útil como técnica de teste** e que é importante conhecer qual a técnica de teste é melhor para a equipe (R5). Além disso, um outro entrevistado complementou dizendo que **testes normais não são tão bem escritos do que os que são escritos para o BDD** (R13), conforme:

> *"Eu sinto falta daquela forma, porque quando tu faz com testes normais, por assim dizer, tu não tem aquele cenário tão bem escrito quando tu tem no BDD." (R13)*

Outra recomendação feita é que **se deve tomar cuidado com a ferramenta que será utilizada para o BDD**. É interessante o uso do Cucumber quando se tem cliente interessado no ciclo (R14), abaixo:

> *"Cucumber, se o seu cliente esta interessado em saber cobertura de testes, dos seus cenários, aí sim, você toma o seu tempo ali pra escrever o BDD direitinho, implementar aquilo da melhor maneira possível. Se não tem ninguém que vá olhar pra cobertura de testes, reports, nem nada depois, não perde tempo escrevendo cenários de BDD pro seu código, porque não vai ter valor nenhum." (R14)*

Complementando, diante do incentivo à colaboração, o BDD deve ser utilizado nem que seja só como **ferramenta de conversa entre analistas e desenvolvedores** (R10).

Além disso, as características para aplicaç ão de BDD em projetos foram as seguintes: em **projetos que tem iterações** (R7, R16), pois o BDD reflete o comportamento dessas iterações; em **projetos grandes** (R4, R6), pois um projeto pequeno não se beneficiaria tanto do que o BDD proporciona; para **melhoria e atualização de software** (R20), para manter essas informações de atualização; e, em **projetos novos** (R6), para ter início com o projeto.

Também, há recomendação quando há a necessidade de **gerar documentação do software** ainda mais se precisar passar para ao cliente (R12). Um dos entrevistados ressaltou que o uso do BDD é um **ganho para a empresa em qualidade** (R16). Assim como um outro entrevistado mencionou que BDD é **prática para melhoria do software** (R24).

Contudo, há quem ressaltou que para uso de BDD **deve-se mudar questão cultural** (R8). Um entrevistado reforçou que para adoção desta abordagem a **equipe tem que ter maturidade** (R22). Por outro lado, foi reportado que o BDD serve **para a evolução da equipe** (R9).

Há situações em que não é recomendado o uso do BDD, como: **projetos sem analistas e com usuários desconhecidos** (R19), ou **quando requisitos são flutuantes** (R16), ou um produto de inovação (R10), ou para **pequenas provas de conceito** (R21), ou **em teste a nível de API** (R15), pelo nível de especificação, ou em **equipes menores do que Scrum prevê** (R1), ou ainda **para definir CRUDs, a não ser que estes saiam do convencional** (R11). Ou seja, o BDD não agregada se o comportamento é conhecido como na Criação, Leitura, Atualização e Deleção de registros.

5. DISCUSSÃO DOS RESULTADOS

Este capítulo apresenta uma discussão sobre os resultados discorridos no Capítulo 4. Assim procurou-se analisar criticamente e relacionar com a literatura, quando possível, seja em relação ao embasamento teórico ou aos trabalhos relacionados, os aspectos identificados e descritos anteriormente. A discussão está organizada em três etapas, respondendo as três questões de pesquisa que delinearam este estudo. Acredita-se que com isto possa-se auxiliar equipes de software e pesquisadores da área a compreender melhor os aspectos que possam ser determinantes para adoção do BDD e apontar para novas direções para a pesquisa.

5.1 BDD no ciclo de vida do projeto de software (QP1)

O BDD é utilizado em projetos de natureza ágil. As equipes que usufruem do mesmo estão alocados de forma local e distribuída, sendo que há indícios do BDD melhorar a eficácia e entendimento da equipe, até mesmo para a forma distribuída. Constatou-se que BDD é aplicado em diversos tipos de plataforma (*Web*, *Desktop* e *Mobile*), no entanto o maior uso é para aplicações *Web* e posteriormente *Mobile*. Para sistemas para pesquisas, saúde, *e-commerce*, gestão de conteúdo, administrativo, educação, intranet, sistemas legados, integração de servidores, teste de interface, descrição de APIs. Há uma vasta variedade de *frameworks* disponíveis para apoio ao BDD para diversas linguagens de programação, como Java, C#, Ruyb, Python, entre outras.

O uso do BDD na prática se difere do que há na teoria [58, 71]. Os praticantes em geral não entraram em um único consenso sobre a definição do BDD, visto que temos diversas definições para o mesmo [72, 28, 71]. Observou-se que há a confusão que BDD é igual a TDD, ATDD, ou mesmo que visa apenas a realização de testes ou definição de critérios de aceite do sistema. Por não haver consenso no entendimento, foi mencionado, pelos entrevistados, que o BDD é uma forma de extração do comportamento do sistema, uma linguagem que pessoas entendem, uma ferramenta de automação, uma ferramenta de comunicação, uma modalidade de desenvolvimento, entre outras definições. Assim, constatou-se que a maioria dos praticantes entrevistados não possuem uma visão completa sobre a extensão do BDD [58].

O BDD frequentemente está associado com outras metodologias como Scrum, XP, Kanban, e desenvolvimento de software Lean, utiliza práticas como *Scrum of scrums*, programação em pares, *workshops de escrita*, entre outras. Algumas destas práticas, encapsuladas por um *framework* conceitual compõem o que é o BDD em si. Desta forma, observou-se que, quando o BDD for usado na íntegra por profissionais de desenvolvimento

de software, os mesmos ressaltaram que quando todos integrantes estavam envolvidos durante o ciclo evitava retrabalho, pois as definições de cenários que estavam sendo utilizas era de consenso de todos.

Identificou-se ainda que dificilmente analistas, desenvolvedores e testadores estão envolvidos durante todo o processo de desenvolvimento do software. Foi verificado que as vezes o uso do BDD está centrado nos testadores ou desenvolvedores para realização do entendimento do comportamento do sistema ou validações do produto de software, ilustrado na Seção 4.3 Figura 4.1. Também, foi constatado que há pequenas variações na formação da equipe que utiliza do BDD com Scrum Master ou *Designer*. Raramente o cliente está envolvido junto a equipe do projeto, quando o mesmo participa se envolve com a definição dos cenários e verificação da entrega, garantindo que todos os cenários estão implementados. O desenvolvimento ágil prima pelo envolvimento mais extenso do cliente e o BDD herda esta característica [71]. Entretanto, este estudo revelou que ainda é preciso buscar aproximar ainda mais o cliente para se obter um maior alinhamento dos cenários com a realidade do negócio.

As ferramentas de apoio ao BDD são diversas, desde o aplicativo de escrita de documentos da Microsoft, como *Word*, e *dashboards* para definição de cenários até versionamento e integração contínua para o código. Além destas, há *frameworks* para uso de BDD como: RSpec, Cucumber, Specflow, JBehave, DASpec, Cucumber JS, NBehave, BehavePro, Behave-django,FitNesse, Lettuce, Calabache, Capybara, MetaRuby, Watcher, Spinach, PyUnit e Demoiselle. Isto revela a variedade de recursos disponíveis na industria, sugerindo que os profissionais identifiquem critérios para seleção dos mesmos, conforme observado por Solis e Wang [72].

Foi também identificado que frequentemente o BDD é centrado em somente uma etapa do desenvolvimento de software, ou para o time desenvolvimento ou para o time da validação de software. No entanto, quando há mais de um papel envolvido realizando o BDD observa-se que há distorções na forma de adoção em relação ao que a teoria propõe. Por alguns acreditarem que o BDD centra-se na descoberta dos critérios de aceite ou na automação destes critérios, teve-se diversos relatos do uso exigir a 'tradução' de requisitos escritos em formatos diversos para este conjunto de itens. Dan North concebeu o BDD de forma que o mesmo apoie o uso de conversas e exemplos para especificar o comportamento do sistema, conforme apresentado em [71], substituindo a necessidade de se escrever documentos separados ou registrar os requisitos em formatos que o cliente desconheça, como a UML por exemplo. Os resultados indicam que, em sua grande parte, a indústria ainda não utiliza desta 'integração' entre diferentes visões de um sistema (e.g., requisitos, código, teste) e que há espaço para a busca dos benefícios prometidos pelo BDD.

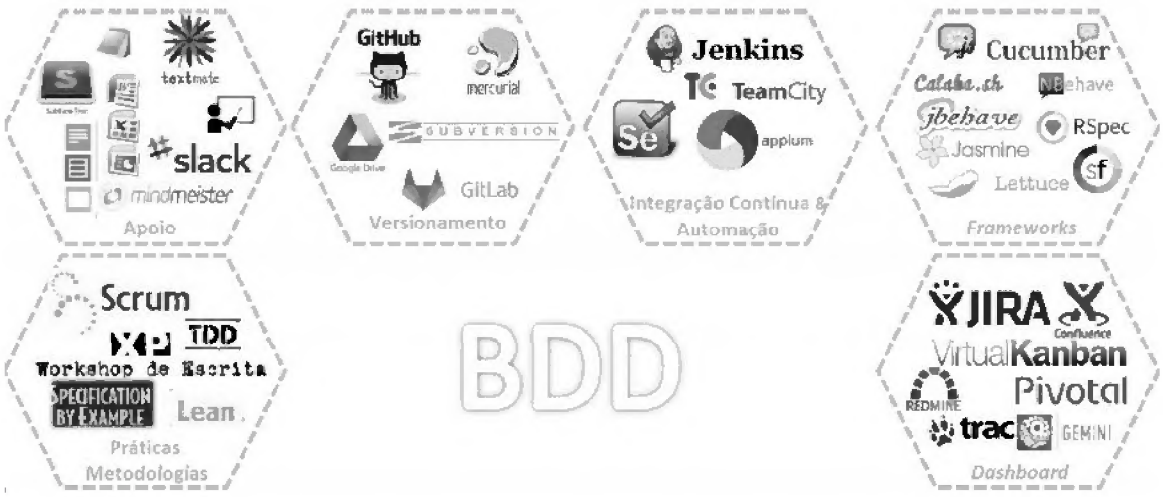


Figura 5.1 – Apoio ferramental quando utilizado BDD

Constatou-se ainda que quanto maior o número de integrantes da equipe alinhados durante a definição dos cenários, menor é o retrabalho para desenvolvimento de software. Discute-se mais sobre os benefícios na Seção 5.3

5.2 BDD e seu suporte à engenharia de requisitos (QP2)

Constatou-se que o cliente raramente está envolvido com a equipe, geralmente o mesmo é representado por um *proxy*. Na teoria, é recomendado que as especificações detalhadas não sejam escritas por analistas de negócios e, em seguida, transferidas para as equipes de desenvolvimento [71]. No entanto, isto é frequentemente reportado na prática, assim o envolvimento frequentemente através de um *proxy* é comum, independentemente se a demanda do negócio for interna ou externa a empresa que desenvolverá o projeto de software para esta demanda.

Observou-se que quando há envolvimento do cliente, mesmo sendo não tão frequente, há uma troca de informações pertinentes ao negócio, com uma discussão dos exemplos e definição de cenários. Diante disso, quando há este envolvimento há a mitigação de haver divergência na perspectiva do negócio. O BDD não é recomendado quando o cliente não quer se comprometer a dedicar tempo para o envolvimento no projeto [2], mas este estudo revelou que o mesmo tem sido adotado em situações adversas e que mesmo assim ainda se enxerga valia na adoção do mesmo. Vislumbra-se então o potencial de benefícios caso o mesmo fosse adotado na 'íntegra' conforme proposto.

Observou-se que frequentemente a coleta para definição dos cenários do BDD é realizada através de reuniões, *workshops* e cerimônias ágeis como o *planning*. Além disso, também foi observada a realização de estimativas que usufruem da definição dos cenários. Quando há o envolvimento do cliente, há encontro com o mesmo para alinhar o enten-

dimento, buscando exemplos para maior entendimento do negócio, que seria o SBE. Há o incentivo a prática do '3 amigos', um desenvolvedor, um testador, e um analista de negócios ou PO (*Product Owner*), se reúnem para discutir funcionalidades e elaborar exemplos. Para que isto funcione bem, os três precisam estar razoavelmente familiarizados com o problema [71, 2]. '3 amigos' incentivada pela teoria dificilmente foi constatada na prática.

Mesmo sendo a Engenharia de Requisitos contínua quando imersa em contexto Ágil, frequentemente observa-se que os cenários para o BDD são definidos anteriormente à etapa de desenvolvimento da *sprint/release*. Também, em poucas vezes, constatou-se situações em que o desenvolvimento e a definição dos cenários acontecem de forma paralela. Diante disto, frequentemente foi observado que a verificação ou validação dos cenários é realizada por um único papel, ou desenvolvedores ou testadores. Em algumas situações mais de um papel é responsabilizado pela garantia de alinhamento dos cenários com as necessidades do negócio, geralmente um desenvolvedor e um analista, ou um testador e um analista, conforme ilustrado na Figura 5.2, na Figura há arestas indicando a intensidade (frequência) de relação entre os integrantes conectados. Assim busca-se mitigar o viés de má interpretação dos cenários.

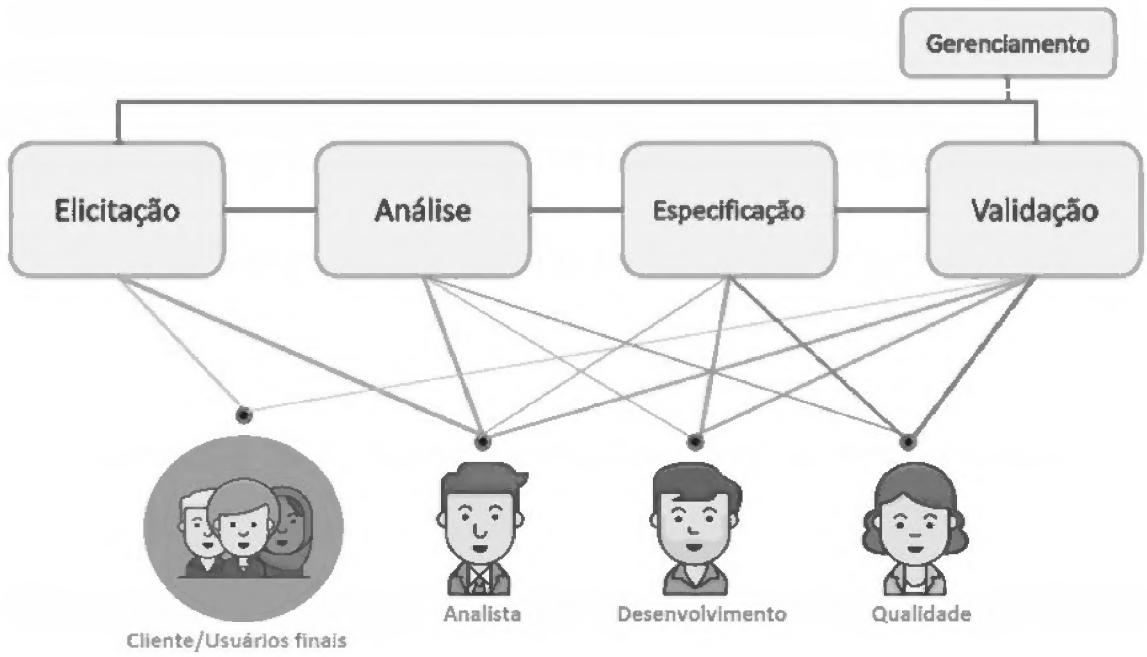


Figura 5.2 – Engenharia de Requisitos em BDD

Observou-se o uso de *living documentation*, que é automaticamente gerada quando o BDD é usado para o código. Não de forma vasta, mas as empresas que se beneficiam da mesma. Esta documentação pode ser considerada de grande valia dentro do processo de engenharia de requisitos, pois representa uma especificação viva do projeto. Ainda, grande parte dos entrevistados relataram que usufruem de BDD para o entendimento do comportamento do sistema, utilizando na verdade de SBE, que é um conjunto de práticas que emergiu do uso de exemplos e conversas para descobrir e descrever os requisitos [2].

Constatou-se também que em algumas situações os cenários levantados durante este processo servem como contratos ou documentação para comunicar o time das atividades que devem ser realizadas. No entanto, estas documentações acabam rapidamente ficando obsoletas, tirando o grande benefício do BDD de manter uma documentação viva do projeto. Assim, constatou-se que não há, muitas vezes, o entendimento e o ambiente propício para o uso do BDD, fazendo com que os mesmos usufruam somente dos cenários para troca de conhecimento com a equipe. Esta documentação atualizada requer pouca ou nenhuma manutenção manual. Os relatórios produzidos pelas especificações executáveis não são relatórios técnicos para desenvolvedores, mas efetivamente se tornam uma documentação do produto para toda equipe, expressa com vocabulário familiar aos usuários [71].

Frequentemente foi observado que o BDD viabiliza a expressão dos requisitos do negócio, fornecendo a melhora do entendimento da equipe. Assim, com um padrão (ex. Gherkin com as cláusulas Dado que-Quando-Então) de escrita faz-se que haja uma documentação fácil para criar software, devido esta escrita. No entanto, verificou-se que BDD apoia o alinhamento do entendimento das necessidades do negócio com a equipe do projeto, fazendo com que os cenários reflitam o negócio. Além disso, contatou-se que BDD apoia a especificação de diversos tipos de requisitos, como, requisitos de *frontend*, funcionais e não funcionais. Ainda, o BDD demonstra-se útil quando há necessidade de um processo que visa resguardar a qualidade do produto. Observou-se que o processo de escrita dos cenários, incentivado pelo SBE, auxilia no apoio à especificação dos requisitos como um todo.

5.3 Benefícios, dificuldades e recomendações sobre o BDD (QP3)

Constatou-se uma diversidade de benefícios listados associados ao uso do BDD, como: visibilidade do negócio, melhora do entendimento, facilidade de vocabulário para a comunicação entre todos *stakeholders*, evita a ambiguidade, diminuição dos problemas da linguagem, melhora da comunicação devido a padronização da escrita, incentiva a colaboração e compartilhamento da equipe, capacitação de novos integrantes, rastreio dos cenários, evita desperdício de documentação, provê documentação atualizada, automação, melhoria da qualidade com redução de defeitos e retrabalho, havendo aumento da produtividade. Smart (2014) listou em seu livro os seguintes possíveis benefícios do BDD: redução de gastos, concentrando o esforço do desenvolvimento na descoberta e entrega de funcionalidades que proverão valor ao negócio; redução dos custos, como consequência desta redução dos gastos é redução de custos; mudança fáceis e seguras (documentação viva) é gerada das especificações executáveis que utilizam termos que os *stakeholders* estão familiarizados); e, *releases* mais rápidas, por causa que testes automatizados também aceleram o ciclo de *release* consideravelmente [71].

Diante disto, verificou-se que, na prática, as *releases* mais rápidas não foram observadas. Já a visibilidade do negócio e a diminuição de retrabalho, reduzindo o gastos de tempo e custo foram constatadas. A linguagem facilitando a comunicação e padronizada com vocabulário útil foi frequentemente ressaltada quando há o uso do BDD.

Contatou-se que benefícios do BDD são adicionais a aqueles advindos das metodologias ágeis, pois frequentemente observou-se relatos de que o desenvolvimento ágil não se preocupa com o que o BDD está preocupado. Observou-se que se não houvesse a adoção do BDD seria muito mais custoso para a equipe, pois teria que haver maior comunicação para contextualizar equipe, já que o grande valor do BDD esta na melhora da comunicação, padronizando os termos, o vocabulário.

Foram constatadas diversas recomendações de uso em BDD, como: quando se quer ter um vocabulário que permita tradução simultânea do negócio para código, quando há muitos defeitos por falta de entendimento, para estimular a equipe a contribuir com cenários, em projetos que tenham iterações, em projetos que tenham usuários envolvidos, para melhorias e atualizações em software, para obter melhoria da qualidade, quando há cenários complexos, para evitar de fazer cenários a nível de interface, quando há a necessidade do cliente ter auditoria em ter documentação, e para equipe entender o que é BDD e o que as ferramentas associadas a ele proporcionam.

Também se identificou que há uma certa carência na divulgação em profundidade sobre o BDD e do entendimento de para que servem as suas ferramentas de apoio, fazendo com que haja diversos praticantes de BDD utilizando da abordagem de forma distorcida. Assim, foi observado que há a necessidade de divulgação do BDD, como, estudos mensurando seus benefícios, e, também, necessidade de treinamentos no assunto, conforme ilustrado na Figura 5.3.

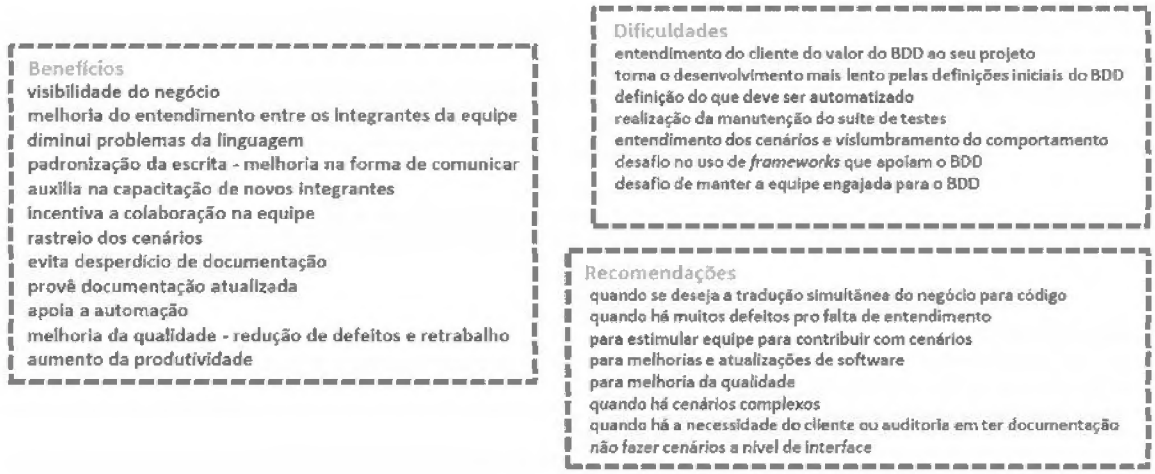


Figura 5.3 – Benefícios e dificuldades de BDD

Constatou-se que há o desejo de melhorias em *frameworks* que apoiam o BDD, como alteração em palavras que possam fazer com que o sentido se torne ambíguo, em gherkin, alteração em palavras com *AND* e *BUT*. Notou-se uma maior adoção do BDD na

comunidade há necessidade de mudança cultural, pela natureza de ser uma abordagem ágil, fazendo com que diminua as distâncias entre os papéis.

Contatou-se diversas desvantagens associadas ao uso do BDD, como: fazer com que o cliente entenda o valor do BDD ao seu projeto pelo custo maior, faz que o desenvolvimento se torne mais lento pela necessidade das definições iniciais, definir o que será automatizado, realizar a manutenção do suíte de testes, entender os cenários e vislumbrar o comportamento, utilizar as ferramentas de forma correta, manter a equipe dinâmica e encontrar material de BDD. Na teoria, foi relatado que BDD requer alto engajamento e colaboração do cliente (com base em conversas e *feedbacks*); BDD funciona melhor em contexto ágil ou iterativo; e, BDD não funciona bem em um silo (em muitas organizações grandes, uma abordagem de desenvolvimento em silos ainda é a regra) [71]. No entanto, observou-se que quando o BDD funciona em um silo há diversos benefícios em seu uso, como a melhora do entendimento da equipe; observou-se que o BDD estava associado a contextos ágeis; quanto ao cliente foi observado que o mesmo não vê valor no BDD, pois isto irá custar mais pra ele, devido ao tempo a mais que leva para a realização do uso do BDD, mas, com o tempo, o mesmo entente este valor agregado em seu sistema.

Diante disto, observou-se a adoção do BDD na indústria tem sido ocasionada por uma tendência do mercado, pois diversos setores de engenharia e qualidade, e as próprias empresas, tem incentivado a adoção do BDD visto as melhorias que o mesmo tem potencial a oferecer. Constatou-se que a maioria dos praticantes estão imersos em projetos que usufruem de BDD diante desta tendência, observando que a adoção do BDD está em expansão. Também, observou-se a adesão do BDD em projetos *open-source*. Frequentemente a adesão do BDD está associada a melhora da qualidade do sistema, pelos testes e pelo entendimento do sistema, gerando uma sensação de segurança na entrega, mitigando o número de incidentes que possam ir para produção.

6. CONSIDERAÇÕES FINAIS

Este capítulo apresenta as resoluções finais e conclusões deste estudo. A Seção 6.1 detalha as limitações quanto a forma de execução deste estudo. A Seção 6.2 apresenta a síntese dos resultados apresentados no Capítulo 5. Por fim, a Seção 6.3 apresenta os possíveis trabalhos futuros.

6.1 Limitações

Como em todos os estudos de natureza empírica, houveram ameaças que podem afetar a validade dos resultados. Assim, observou-se as seguintes limitações:

- Há a limitação do método escolhido para coleta de dados, as entrevistas. Estas obtém relatos dos respondentes sobre suas perspectivas, pensamentos e conclusões. No entanto, é possível que os relatos não sejam absolutamente fidedignos com as atividades que eles desempenham na realidade. Devido ser um conhecimento tácito os respondentes acreditam que fatos não são relevantes, ou deixam de reportar certas informações por algum receio, ou relatem as atividades esperam que sejam realizas. Assim, buscou-se seguir recomendações de Singer *et al.* (2008) conforme apresentada na Seção 3.3.
- Há a limitação por parte do pesquisador e o seu entendimento sobre resultados do estudo. Assim, buscou-se embasamento teórico para melhor interpretação dos resultados pelo pesquisador, e, além disso, utilizou-se de métodos de coleta e análise conhecidos com o intuito de mitigar o viés do pesquisador.
- Outra ameaça para a validade dos resultados é a subjetividade da classificação de dados, uma vez que a análise qualitativa foi realizada por apenas um pesquisador com a revisão dos códigos criados e mapeados em redes pela orientadora, conforme Apêndice D, sem a codificação do todo ou parte das entrevistas por outro colega. Assim, *Grounded Theory* foi utilizada a fim de diminuir essa ameaça, uma vez que este método requer que toda a análise seja baseada nos dados coletados, conforme Strauss e Corbin (1990) propõe [75], e sugere o processo cíclico de coleta-análise-reflexão, conforme seguido neste estudo.

6.2 Conclusões

Esta pesquisa apresentou um estudo empírico qualitativo, através de um estudo de campo exploratório com o intuito de identificar como o BDD é adotado na Indústria. Assim,

identificou-se como o BDD é utilizado na prática,como o mesmo apoia a engenharia dos requisitos, e seus benefícios, dificuldades e recomendações.

Diante dos problemas recorrentes da ER, narrados na literatura há mais de 20 anos, mesmo nesse contexto de necessidades de mudança para se adaptar ao mercado atual, impactando na forma de desenvolver software, o qual adere as metodologias ágeis (as quais recomendam entregas contínuas, equipes menores, entre outros) continua-se a enfrentar os mesmos problemas mencionados na teoria (vide Seção 2.2). E, visto que BDD se propõe, na teoria, alinhar dos termos do negócio [71], melhorar entendimento e comunicação dos *stakeholders* do projeto [23], melhorar a qualidade do software e rastrear os requisitos [71].

Foi explorado e identificado, através da experiência de 24 profissionais, como é o ciclo de desenvolvimento com BDD, como o mesmo apoia a ER e auxilia nos problemas da ER. Assim, apresentou-se resultados com base em 24 profissionais da área de Tecnologia da Informação.

O método de análise de dados utilizado neste estudo foi a Teoria fundamentada em dados (*Grounded Theory*), baseada em [75]. Assim, observou-se o uso do BDD tanto para desenvolvimento local quando para desenvolvimento distribuído de software, e constatou-se que o mesmo apoia, com êxito, a engenharia dos requisitos. A aplicação de BDD foi observada em manutenção de software, criação de novos projetos e produtos de software, para o compartilhamento de informações de projeto ou a automação de software, tanto para aplicações *web*, *mobile* e *desktop*, entre outros.

Constatou-se que houve similaridades nos benefícios/desafios mencionados pelos entrevistados deste estudo e nas práticas e metodologias adotadas ágeis adotadas pelas mesmas durante o processo de desenvolvimento de software. Houve semelhança nas atividades desempenhadas aos diferentes papéis (cargos) que usufruíam de BDD, por influência das práticas ágeis combinadas ao BDD.

Assim, pode-se constatar que o ciclo de desenvolvimento que possui uma configuração propicia ao uso de BDD tem um *feedback* quanto ao êxito de entrega de um software que atenda as necessidades do cliente e de qualidade, ou seja, auxilia a mitigar alguns problemas da engenharia de requisitos, como dificuldade na comunicação, gestão do conhecimento, problemas de linguagem e rastreamento. Diante disso, dentre das características observadas, a síntese de resultados deste estudo é representada pela Figura 6.1.

O estudo deste fenômeno reforça os poucos estudos correlatos identificados, de uma perspectiva empírica da prática do BDD. O mesmo centrou-se em auxiliar equipes e pesquisadores da área em identificar como é adotado o BDD na indústria, atualmente, auxiliando-os a compreender melhor os aspectos que podem ser determinantes para adoção do BDD e novas direções para a pesquisa, em detalhas na Seção 6.3.

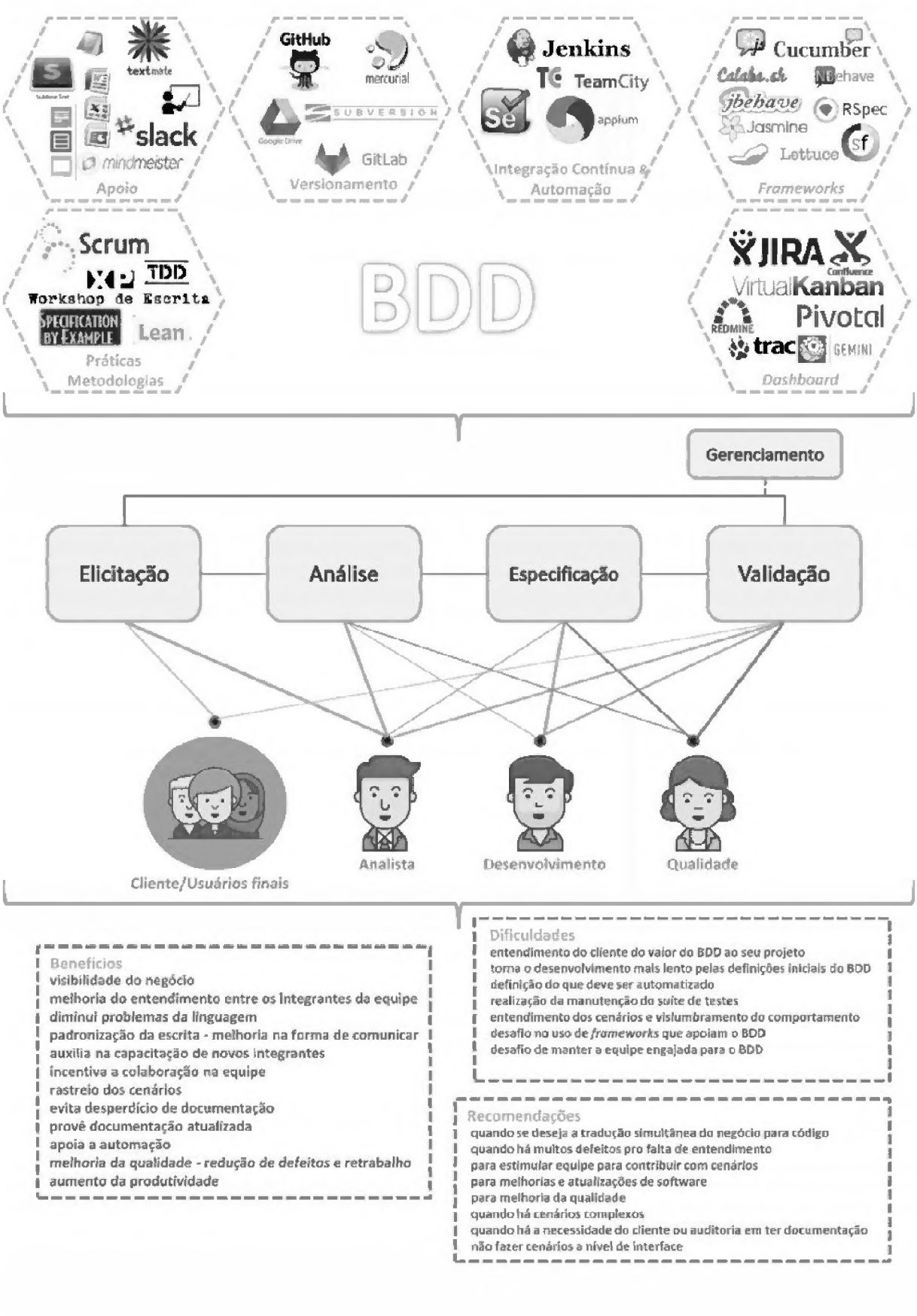


Figura 6.1 – Síntese das respostas para QP1, QP2 e QP3

6.3 Trabalhos Futuros

Identificou-se grande potencial de crescimento do assunto abordado neste estudo, criando as seguintes oportunidades para investigação:

(i) Entendimento de como as ferramentas auxiliam em determinadas etapas da engenharia de requisitos. *(ii)* Mensurar como os benefícios impactam nos projetos que usufruem de BDD. Em andamento por estudante de Mestrado pela mesma professora orientadora desta pesquisa de Mestrado. *(iii)* Relacionar BDD com demais abordagens, como, ATDD, SBE, entre outras. *(iv)* Formular conjunto de recomendações para uso de BDD. Em andamento por estudante de Mestrado pela mesma professora orientadora desta pesquisa de Mestrado. *(v)* Entender mais profundamente quais tipos de especificações são apoiadas pelo BDD. *(vi)* Analisar impacto dos *frameworks* de BDD e suas limitações. *(vii)* Generalizar, através de um *survey* em incluir, como o BDD está sendo realizado na prática. Conduzido e finalizado em Trabalho de Conclusão de Curso em 2016/2 na Faculdade de Informática da PUCRS. No entanto este *survey* está centrado somente nos praticantes de BDD localizados no Brasil. *(viii)* Generalizar, através de um *survey* em inglês, como o BDD está sendo realizado na prática, para identificar como o BDD está sendo adotado em um âmbito global. *(ix)* Propor *framework* que apoie maior cobertura das características do BDD.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] "Atlas.ti: The qualitative data analysis". Capturado em: http://atlasti.com/, 2016.

[2] Adzic, G. "Specification by Example: How Successful Teams Deliver the Right Software". Connecticut, USA: Manning Publications, 2011.

[3] Al-Rawas, A.; Easterbrook, S. "Communication problems in requirements engineering: A field study". In: Proceedings of the Conference on Professional Awareness in Software Engineering, NASA, London, UK, 1996, pp. 1–12.

[4] Anderson, D. "Kanban: Successful Evolutionary Change for Your Technology Business". Washington DC, USA: Blue Hole Press, 2010.

[5] Astels, D.; *et al.* "Behaviour-driven development for ruby. making tdd productive and fun". Capturado em: http://rspec.info/, 2015.

[6] Aurum, A.; Wohlin, C. "Requirements engineering: setting the context". In: *Engineering and managing software requirements*, New Jersey, USA: Springer-Verlag New York, 2005, pp. 1–15.

[7] Bano, M.; Zowghi, D. "User involvement in software development and system success: A systematic literature review". In: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering, 2013, pp. 125–130.

[8] Beck, K. "Extreme Programming Explained: Embrace Change". Boston, USA: Addison-Wesley Longman Publishing, 2000.

[9] Beck, K.; *et al.* "Manifesto for agile software development". Capturado em: http://www.agilemanifesto.org/, 2014.

[10] Bourque, P.; Fairley, R. (Editores). "SWEBOK: Guide to the Software Engineering Body of Knowledge". Los Alamitos, CA: IEEE, 2014, 3rd ed.

[11] Cao, L.; Ramesh, B. "Agile Requirements Engineering Practices: An Empirical Study", *IEEE Software*, vol. 25–1, 2008, pp. 60–67.

[12] Carmel, E. "Global Software Teams: Collaborating Across Borders and Time Zones". New Jersey, USA: Prentice Hall PTR, 1999.

[13] Charmaz, K. "Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis". Lincoln, USA: SAGE Publications, 2006.

[14] Chelimsky, D. "The RSpec Book: Behaviour-driven Development with RSpec, Cucumber, and Friends". Pragmatic Bookshelf, 2010, 1st ed.

[15] Christel, M.; Kang, K. "Issues in requirements elicitation", Relatório Técnico CMU/SEI-92-TR-012, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1992.

[16] Cobb, C. "The Project Manager's Guide to Mastering Agile: Principles and Practices for an Adaptive Approach". New Jersey, USA: Wiley, 2015, 1st ed.

[17] Cockburn, A. "Writing Effective Use Cases". Boston, USA: Addison-Wesley Professional, 1999.

[18] Cohn, M. "User Stories Applied: For Agile Software Development". California, USA: Addison Wesley Longman Publishing, 2004.

[19] Creswell, J. "Qualitative Inquiry and Research Design: Choosing Among Five Approaches". Lincoln, USA: SAGE Publications, 2007.

[20] Creswell, J. "Research Design: Qualitative, Quantitative and Mixed Methods Approaches". Lincoln, USA: Sage Publications, 2009.

[21] Damian, D.; Zowghi, D. "The impact of stakeholders' geographical distribution on managing requirements in a multi-site organization". In: International Requirements Engineering Conference, 2002, pp. 319–330.

[22] Elizabeth Bjarnason, Michael Unterkalmsteiner, E. E. M. B. "An industrial case study on test cases as requirements", *Springer International Publishing Switzerland*, 2015, pp. 27–39.

[23] Evans, E. "Domain-Driven Design: Tacking Complexity In the Heart of Software". Boston, MA, USA: Addison-Wesley, 2003.

[24] Firesmith, D. "Common requirements problems, their negative consequences, and the industry best practices to help solve them.", *Journal of Object Technology*, vol. 6–1, 2007, pp. 17–33.

[25] Glaser, B.; Strauss, A. "The Discovery of Grounded Theory: Strategies for Qualitative Research". Chicago, USA: Aldine Publishing Company, 1967.

[26] Glaser, B. G. "What is grounded theory?" Capturado em: http://www.groundedtheory.com/what-is-gt.aspx, 2015.

[27] Goguen, J. A. "Social issues in requirements engineering". In: Proceedings of IEEE International Symposium on Requirements Engineering, IEEE, San Diego, CA, 1993, pp. 194–195.

[28] Gohil, K.; Alapati, N.; Joglekar, S. "Towards behavior driven operations (bdops)". In: International Conference on Advances in Recent Technologies in Communication and Computing, 2011, pp. 262 –264.

[29] Goldsmith, R. "Discovering Real Business Requirements for Software Project Success". London, UK: Artech House, 2004.

[30] Goldstein, I. "Scrum Shortcuts without Cutting Corners: Agile Tactics, Tools, & Tips". Boston, USA: Pearson Education, 2013.

[31] Goyal, S. "Agile techniques for project management and software engineering", Relatório Técnico, Major Seminar On Feature Driven Development, 2007.

[32] Gärtner, M. "ATDD by Example: A Practical Guide to Acceptance Test-Driven Development, First Edition". Boston, USA: Addison-Wesley Professional, 2012.

[33] Hanisch, J. "Requirements engineering in virtual software development: Achieving balance". In: Proceedings of the International Conference on Inforrmation Resoures ciation, Managing Information Technology in a Global Economy, Hershey, USA, 2001, pp. 1–6.

[34] Hanisch, J.; Corbitt, B. "Impediments to requirements engineering during global software development.", *European Journal of Information Systems*, vol. 16–6, 2007, pp. 793–805.

[35] Hellesøy, A. "Cucumber - making bdd fun". Capturado em: http://cukes.info/, 2008.

[36] Helmy, W.; Kamel, A.; Hegazy, O. "Requirements engineering methodology in agile environment", *International Journal of Computer Science Issues*, vol. 9–5, 2012, pp. 293–300.

[37] Herbsleb, J. D.; Moitra, D. "Global software development", *IEEE Software*, vol. 18–2, Mar 2001, pp. 16–20.

[38] IEEE. "Ieee standard glossary of software engineering terminology", *IEEE Std 610.12-1990*, Dec 1990, pp. 1–84.

[39] IEEE. "Ieee recommended practice for software requirements specifications", *IEEE Std 830-1998*, 1998, pp. 1–40.

[40] International, T. S. G. "The chaos manifesto". Capturado em: http://www.standishgroup.com/, 2013.

[41] Janice Singer, Susan E. Sim, T. C. L. "Software Engineering Data Collection for Field Studies". London, UK: Springer London, 2008.

[42] Janzen, D.; Saiedian, H. "Test-driven development: Concepts, taxonomy, and future direction", *Computer*, vol. 38–9, 2005, pp. 43–50.

[43] Kitchenham, B.; Pearl Brereton, O.; Budgen, D.; Turner, M.; Bailey, J.; Linkman, S. "Systematic literature reviews in software engineering - a systematic literature review", *Inf. Softw. Technol.*, vol. 51–1, Jan 2009, pp. 7–15.

[44] Kitchenham, B.; Pfleeger, S. L. "Principles of survey research part 4: Questionnaire evaluation", *SIGSOFT Software Engineering Notes, New York, USA*, vol. 27–3, Maio 2002, pp. 20–23.

[45] Kogut, Bruce; Meitu, A. "Open source software development and distributed innovation", *Oxford Review of Economic Policy*, vol. 17–2, 2001, pp. 248–264.

[46] Koskela, L. "Test Driven: Practical TDD and Acceptance TDD for Java Developers". Connecticut, USA: Manning Publications, 2008.

[47] Kudryashov, K.; *et al.* "Writing features". Capturado em: http://docs.behat.org/en/v2.5/, 2014.

[48] Laplante, P. A. "Requirements Engineering for Software and Systems". CRC Press,Auerbach Publications, 2009, 2nd ed.

[49] Leffingwell, D. "Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise". Boston, USA: Addison-Wesley Professional, 2011, 1st ed.

[50] Lerner, R. M. "At the forge: Cucumber", *Linux Journal*, vol. 2010–189, Jan 2010.

[51] Lopes, J. H. "Evaluation of behavior-driven development", Master thesis, Delft University of Technology, Delft, the Netherlands, 2012.

[52] Lopez, A.; Carrillo-de Gea, J.; Toval, A. "Risks and safeguards for the requirements engineering process in global software development". In: International Conference on Global Software Engineering, IEEE, Limerick, 2009, pp. 394–399.

[53] Lucia, A. D.; Qusef, A. "Requirements engineering in agile software development", *Journal of Emerging Technologies in Web Intelligence*, vol. 2–3, 2010, pp. 212–220.

[54] Matts, C.; Adzic, G. "Feature injection: three steps to success". Capturado em: http://www.infoq.com/articles/feature-injection-success, 2014.

[55] Matts, C.; Adzic, G. "Feature injection: three steps to success". Capturado em: http://www.infoq.com/articles/feature-injection-success, 2014.

[56] McDonald, K. "Beyond Requirements: Analysis with an Agile Mindset". Boston, USA: Pearson Education, 2015.

[57] North, D. "What is jbehave?" Capturado em: http://jbehave.org/, 2003.

[58] North, D. "Introducing bdd". Capturado em: http://dannorth.net/introducing-bdd/, 2006.

[59] Osborne, M.; Macnish, C. "Processing natural language software requirement specifications". In: Proceedings of the Second International Conference on Requirements Engineering, IEEE, Colorado Springs, CO, 1996, pp. 229–236.

[60] Paetsch, F.; Eberlein, A.; Maurer, F. "Requirements engineering and agile software development". In: Proceedings of the International Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises, IEEE, Linz, Austria, 2003, pp. 1–6.

[61] Petersen, K.; Vakkalanka, S.; Kuzniarz, L. "Guidelines for conducting systematic mapping studies in software engineering: An update", *Information and Software Technology*, vol. 64, 2015, pp. 1–18.

[62] Poppendieck, M.; Poppendieck, T. "Lean Software Development: An Agile Toolkit". Boston, USA: Addison-Wesley Professional, 2003.

[63] Pressman, R. "Software Engineering: A Practitioner's Approach". New York, USA: McGraw-Hill, 2010, 7th ed.

[64] Ramesh, B.; Cao, L.; Baskerville, R. "Agile requirements engineering practices and challenges: an empirical study", *Information Systems Journal*, vol. 20–5, 2010, pp. 449–480.

[65] Ramesh, B.; Powers, T.; Stubbs, C.; Edwards, M. "Implementing requirements traceability: a case study." In: Proceedings of the International Conference on Requirements Engineering, IEEE, 1995, pp. 89–99.

[66] Rising, L.; Janoff, N. S. "The scrum software development process for small teams", *IEEE Software*, vol. 17–4, Jul 2000, pp. 26–32.

[67] Schwaber, K.; Sutherland, J. "Scrum guide". Capturado em: www.scrumguides.org/scrum-guide.html, 2014.

[68] Seaman, C. B. "Guide to Advanced Empirical Software Engineering". London, UK: Springer London, 2008, cap. Qualitative Methods, pp. 35–62.

[69] Sebrae. "Critérios de classificação de empresas". Capturado em: http://www.sebrae-sc.com.br/leis, 2015.

[70] Sharp, H.; Finkelsteiin, A.; Galal, G. "Stakeholder identification in the requirements engineering process". In: Proceedings of the 10th International Workshop on Database & Expert Systems Applications, Washington, USA, 1999, pp. 387–393.

[71] Smart, J. "BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle". New York, USA: Manning Publications, 2014.

[72] Solis, C.; Wang, X. "A study of the characteristics of behaviour driven development". In: Conference on Software Engineering and Advanced Applications, Washington DC, USA, 2011, pp. 383–387.

[73] Sommerville, I. "Software Engineering". Harlow, England: Addison-Wesley, 2010, 9th ed.

[74] Stellman, A.; Greene, J. "Learning Agile: Understanding Scrum, XP, Lean, and Kanban". California, USA: O'Reilly Media, 2014.

[75] Strauss, A.; Corbin, J. "Basics of qualitative research: grounded theory procedures and techniques". Thousand Oaks, USA: Sage Publications, 1990, 1st ed.

[76] Strauss, A.; Corbin, J. "Basics of qualitative research: techniques and procedures for developing Grounded Theory". Thousand Oaks, USA: Sage Publications, 1998, 2nd ed.

[77] Trumbull, M. "Integrating Quantitative and Qualitative Methods in Research". Maryland, USA: University Press of America, 2005.

[78] Versione. "State of agile survey", Relatório Técnico 9, VersiOne Agile Made Easier, 2015.

[79] Zowghi, D. "Does global software development need a different requirements engineering process?" In: Proceedings of the International Workshop on Global Software Development, Florida, USA, 2012, pp. 56–58.

[80] Zowghi, D.; Coulin, C. "Requirements Elicitation: A Survey of Techniques, Approaches". New Jersey, USA: Springer-Verlag New York, 2005.

APÊNDICE A – ROTEIRO DE ENTREVISTAS

Interview Script

Goal of this study: *The study is focused on understanding the Behavior-Driven Development approach in practice. We want to have an understanding of what BDD means to you, how it is done, the benefits and difficulties you face when using it, among other aspects.*

[Concept] P1: In your opinion, how would you define BDD?
(E.g. Model, method, framework, tool, approach, technique, methodology)

[Phases] P2: How do you typically use BDD in your daily activities? *[Phases, transition, activities, identify the Agile practices]*

In which phases do you use BDD? *(Analysis, Dev, Test, All cycle)*
(I.e. To identify the business goals, to elaborate the features, to create the examples, to define acceptance criteria | to develop the code | to test the code | to create living documentation, to generate reports and/or technical documentation)

[Project-based] P2.1: In which type of projects do you often use BDD?

- **How many projects you are involved with use BDD?**
- **Technology type - solution type?** *(Web, Desktop or Mobile)*
- **Programming language?** *(E.g. Java, C#, ...)*
- **Which agile methodologies are you using combined with BDD approach?** *(E.g. TDD, Scrum, XP, ...)*
- **Is the team distributed?** *(Yes or no)*
  - **If <YES>** *How is the distribution?*

[People involved] P3: How many members get involved during the usage of BDD?

[Team-based] P3.1: Who are the people involved *[role/professional profile]* **during the cycle?**
(E.g. Business owner, client, customer, business analyst, product owner, developer, tester, etc)

[Client] P3.2: Are there any client involved? *[Note: Don't consider P.O.]*

- **If <YES>**
  - How is the client [in/outside team] involved during BDD?
  - How often the client is involved? *(E.g. Once a week, twice a week, everyday)*
- **If <NO>**
  - Do you know why they do not get involved?
    - *If <YES> Why?*
    - *If <NO> Ok*
  - Do you think they should get involved?
    - *If <YES> How do you think they could get involved during the BDD cycle?*
    - *If <NO> Why not?*

[Framework] P4: Which tool do you adopt to support the BDD activities?
Cucumber (Ruby, Java), JBehave (Java), NBehave and SpecFlow (C#), Freshen (Python), Behat (PHP), JSpec (JavaScript), CSpec (C), ...

[Environment] P4.1: Do you adopt any other tool to support the management *[creation, update, etc]* **of scenarios or stories? If so, which ones?**

(E.g. Word, Excel, Requirements management tool, Kanban, etc)

[Artifacts] P5: Do you use or generate any artifact when using BDD? Which ones? [Note: Here consider the artifacts proposed by BDD, *(E.g. Living documentation, Progress reports)]*

Give me an example of how do you use the referred artifacts?

[Requirements] P6: About the scenarios...

[Scenarios creator] P6.1: Who is responsible in creating the scenarios in BDD?

(E.g. All members, BA, SA, PO, etc)

[Scenarios collection] P6.2: How is collected the information to write the scenarios?

(E.g. Meeting, workshop, only via document, etc)

[Scenarios info] P6.3: What information type is used as reference to write the scenarios? Who provides them?

(E.g. Mock-ups, stories, acceptance criteria, etc)

[Scenarios info] P6.4: When (in the lifecycle) do you define the scenarios?

(E.g. After user stories definition, once a sprint, before next sprint, etc)

[Client participation] P6.5: Does the client meet with the development and/or test team while defining the scenarios? *(Note: Make sure if the client doesn't participate with team)*

[Team sharing] P6.6: How are the scenarios shared with the team?

(E.g. Repository, meeting, workshop, only a document)

[Fidelity] P6.7: Who is responsible to guarantee that the scenarios are aligned and attending the business goals?

(E.g. All members, BA, SA, PO, etc)

[Customer wants] P6.8: In your opinion, do you believe that what the customer wants is clearly defined and better understood when using BDD to express the software requirements?

Do you believe that BDD supports requirements specification?

- If <NO> Why not?

[Benefits] P7: Do you believe that there are benefits using BDD? If so, which are they? *[Note: Remember the requirements]*

[Product] P7.1: Do you believe that the BDD contribute to improve the quality of the developed software product?

- If <YES>
  What are the perceived improvements?
  (E.g. less defects, increased features coverage, reduced misunderstandings about what the software should do, less rework, etc)
- If <NO> Why not?

[Process] P7.2: Overall, do you think that often-face issues by the development team are reduced when using BDD?
(E.g. such as miscommunications, obsolete documentation, etc)

[Recommendation] P7.3: In your experience, would you recommend BDD to others? In which situations?

[Difficulties] P8: Have you faced difficulties using of BDD? If so, which are they?
What are, in your opinion, the main challenges to adopt BDD?

[Change] P8.1: What do you suggest to improve the BDD approach?
What do you believe is missing for a broader adoption of BDD in the community?

[Why to adopt] P9: Why you/your team decided to start adopting it?

[How to adopt] P9.1: How did you learned about BDD?

[Refletion] P10: Please imagine the following situation: You are using only agile cerimonies (without BDD), as you mention earlier.

[Agile or BDD] Do you believe that you would have the same benefits you mentioned before?

Background Information

Subject:

S1. Full Name _____

S2. Education level _____

S3. Country and City _____

S4. Current job position and a brief description _____

S5. Years of experience in the current job position _____

S6. Years at Org _____

S7. Years of working experience in industry _____

S8. Years of experience using Agile _____

S9. Years of experience using BDD _____

Company:

C1.Main software methodology adopted?
Agile [ ] Traditional [ ]

C2. Number of people in the team: _____

C3. Number of employees at the company: _____

C4. Is IT the company's core business?
Yes [ ] No [ ]
If No, which one is it? _____

APÊNDICE B – TERMO DE CONSENTIMENTO

Faculdade de Informática /PUCRS
Programa de Pós-Graduação em Ciência da Computação
Avenida Ipiranga, 6681 – Prédio 32 - 90619-900 – Porto Alegre – RS

Termo de Consentimento Livre e Esclarecido

A PUCRS, através das equipes de Engenharia de Software da Faculdade de Informática, agradece a todos os participantes de entrevistas realizados sob sua responsabilidade, a inestimável contribuição que prestam para o avanço da pesquisa sobre Engenharia de Requisitos.

O objetivo desta pesquisa é compreender como está sendo realizado o ciclo do BDD (Behavior-Driven Development) na prática. Para isto, serão realizadas entrevistas com profissionais que estão ou estiveram envolvidos em projetos que fizeram o uso do BDD, atuantes na indústria. Com isto visamos identificar como ocorrem as atividades correspondentes ao uso desta abordagem. Além disso, visa-se descobrir se é utilizado apoio ferramental ou algum outro, quais as vantagens e desvantagens do BDD. Espera-se que os resultados desta pesquisa possam contribuir com dados para a consolidação do entendimento sobre o uso geral do BDD.

Lembramos que o objetivo deste estudo **não é** avaliar o entrevistado, **mas sim** entender como está sendo usado o BDD. O uso que se faz dos registros efetuados durante a entrevista é **estritamente** limitado às atividades de pesquisa e desenvolvimento, garantindo-se para tanto que:

1. O anonimato dos participantes será preservado em todo e qualquer documento divulgado em foros científicos (tais como conferências, periódicos, livros e assemelhados) ou pedagógicos (tais como apostilas de cursos, *slides* de apresentações, e assemelhados).
2. Todo entrevistado terá acesso às cópias destes documentos após a publicação dos mesmos.
3. Todo entrevistado que se sentir constrangido ou incomodado durante uma situação da entrevista pode interromper a entrevista e estará fazendo um favor à equipe se registrar por escrito as razões ou sensações que o levaram a esta atitude. A equipe fica obrigada a descartar a entrevista para fins da avaliação a que se destinaria.
4. Os entrevistados que forem menores de idade terão, obrigatoriamente, que apresentar o consentimento de seu responsável, para participação no estudo, o qual será declarado ciente do estudo a ser realizado através de sua assinatura no presente Termo de Consentimento.
5. Todo entrevistado tem direito de expressar por escrito, na data da entrevista, qualquer restrição ou condição adicional que lhe pareça aplicar-se aos itens acima enumerados (1, 2, 3 e 4). A equipe se compromete a observá-las com rigor e entende que, na ausência de tal manifestação, o participante concorda que rejam o comportamento ético da equipe somente as condições impressas no presente documento.
6. A equipe tem direito de utilizar os dados das entrevistas, mantidas as condições acima mencionadas, para quaisquer fins acadêmicos, pedagógicos e/ou de desenvolvimento contemplados por seus membros.

[a ser preenchido pelo entrevistador]
Forma:_____ Data: __ / __ / ____

Condições especiais (caso não haja condições especiais, escreva "nenhuma"):

_____
_____
_____
_____
_____
_____

☐ continua no verso

Por favor, indique sua posição em relação aos termos acima:

☐ Estou de pleno acordo com os termos acima.

☐ Em anexo registro condições adicionais para este teste.

_____
Assinatura do participante

_____
Assinatura do responsável
(caso o participante seja menor de idade)

_____
Assinatura do pesquisador

Nome do Participante:

Pesquisador Responsável: – Faculdade de Informática - PUCRS

APÊNDICE C – PAPER ELA-ES

Furthering Knowledge on How Behavior-Driven Development Can Support Requirements Elicitation

Lauriane Correa[1], Sabrina Marczak[1], Cleidson R. B. de Souza[2]

[1]Computer Science School – Pontifícia Universidade Católica do Rio Grande do Sul
90619-900 – Porto Alegre – RS – Brasil

[2]Instituto Tecnológico Vale e Universidade Federal do Pará
66055-080 – Belém – PA – Brasil

lauriane.moraes@acad.pucrs.br, sabrina.marczak@pucrs.br

cleidson.desouza@acm.org

Abstract. *Requirements elicitation major's challenge is establishing common ground with the customer. Behaviour-Driven Development (BDD), inspired on the concept of 'Specification by Example', proposes a structured, English-based format named scenario to state the desired behavior for the software to be built. We aim to understand BDD usage in the field by first exploring the topic through a multiple case study and later by confirming the preliminary findings in a large-scale survey. While we are still conducting the exploratory study, we have also started planning the survey. In this paper we introduce our research plan to conduct the survey aiming to promote discussion on how to better acquire comprehension about the phenomena.*

1. Introduction

Requirements elicitation tries to discover the application domain, business needs, requirements and system constraints by consulting stakeholders [Sommerville 2010]. Requirements analysts and stakeholders often do not share a common understanding of related concepts and terms. Such lack of common ground can cause misalignment of the elicited requirements [Zowghi and Coulin 2005]. Stakeholders also often have difficulties expressing their needs, making it harder to define the software expected behaviors.

A recurrent reported issue in literature is the difficulty of software teams to communicate clearly [International 2013], causing projects to go over budget or fail. That is why Behavior-driven Development (BDD) emerges as a promessing approach. BDD is the name given to a set of methods and techniques put together aiming to help teams to focus their efforts on identifying, understanding, and building valuable features that matter to businesses, and to ensure that these features are well designed and implemented [Smart 2014]. The way the pieces are tied together aims to ensure consistency and traceability of requirements throughout the development life cycle, to allow for timely communication with anyone involved in the project, including the customer. Communication aspects become less important since BDD uses a structured form.

Despite the promissed benefits and the anedoctal reports of how much BDD can do for a software team, there is little empirical evidence of the extent that it can specifically support requirements elicitation. To fill in this gap, we posed the following research

question: *How can BDD support requirements elicitation in practice?*, and designed an empirical study to answer it. We set to first explore how BDD is used in practice through a case study and then later confirm the preliminary findings in a large-scale survey. Our interest in a large-scale study is to better understand the contexts in which BDD can support requirements engineering (e.g., when the analyst writes the specification, when there is an internal customer), aiming for generalization of our findings. While we briefly introduce our entire research plan to provide context, the goal of this paper is to present the survey design aiming to collect feedback to help us ensure we are in the right path.

2. Behavior-Driven Development in a Nutshell

Behavior-driven Development (BDD) was designed to help teams build and deliver more valuable, higher-quality software faster [Smart 2014]. It was initially proposed by Dan North as a way to teach Test-driven Development (TDD) [North 2006]. It is composed of a set of practices from agile methodologies, such as TDD, automated acceptance testing, and continuous building [Smart 2014]. It also incorporates the definition of features or requirements based on examples as proposed by [Adzic 2011].

BDD provides a connection from code to the requirements, offering a better environment for managing project progress. This is done through scenarios that define a way to describe how the system should behave based in a language that is native to the stakeholder, promoting a common understanding of the business domain between stakeholders and development team [Evans 2003]. BDD starts by identifying relevant business goals and software features that will cover these goals. Collaborating with the customer, BDD practitioners use concrete examples to illustrate the features. These features can be broken down into smaller chunks, named user stories, when more than one aspect composes them. The defined examples are then automated in the form of executable specifications that follow a structured format named 'scenarios'.

Figure 1 illustrates the notation defined to write an example that represents a certain feature. This feature is composed of two scenarios, each composed of a set of steps marked by pre-defined clauses as explained next. A *Feature* is a descriptive text of what is desired by the customer. This description provides context to those reading and

Feature title: *Buying books with the bookstore card*
Narrative: **In order to** buy the books **As a** bookstore client **I want to** pay my chosen books with my bookstore card
Scenario 1: *Paying with a positive balance*
Given my bookstore card has a balance of 300.00 And my bookstore card give me 15% discount **When** I buy 100.00 from my cart in books **Then** I should pay 85.00 And I should have 215.00 left in my bookstore card
Scenario 2: *Paying without sufficient balance*
Given my bookstore card has a balance of 50.00 And my bookstore card give me 15% discount **When** I buy 100.00 from my cart in books **Then** I should receive an 'insufficient balance' error message And I should still have 50.00 in my bookstore card

Figure 1. Illustration of a Feature and Its Scenarios using BDD

using a feature definition and describes the business value of the feature to the software as a whole [North 2006]. A feature usually contains a list of *Scenarios* [Smart 2014]. Each scenario is composed by a set of pre-defined clauses, namely: 'Given', 'When', and 'Then'. *Given* describes the preconditions for the scenario and prepares the test environment. *When* describes the key action the user performs, or state transition. *Then* is used to describe outcomes. The observations should inspect the output of the system (a report, user interface, message, command output). *And* and *But* are additional clauses used to join the previous clauses and provide a more readable way to specify the feature.

3. Proposed Survey in the Context of Our Long-Term Research Plan

Given its novelty, there is little empirical evidence how BDD is used in practice and none, to the best of our knowledge, on how it addresses requirements elicitation issues. To better understand this phenomena and explain the benefits and challenges of BDD adoption, we are currently conducting an exploratory study organized in two smaller steps as indicated in Figure 2, Phase 2, and are preppering for applying a survey (Phase 3) as previously mentioned. We briefly introduce the exploratory studies to provide context.

Our goal conducting the Interview step was to develop a initial understanding about how BDD is defined and used in practice. We have conducted ten semi-structured interviews before the book 'BDD in Action' [Smart 2014] has been released. This book provides a consolidated description of BDD and its related techniques. While analyzing data from the interviews, we started observing team members of a project at a large agile IT company with development centers located in five continents. The case was selected based on convenience and access to the company's office in Brazil. In this project, analysts discussed the needs for a software solution with the customer and wrote the elicited features down using user stories. Later, these stories were transformed into scenarios with the development team's help and then automated by a tool that supports BDD. We have just recently joined the company on site. An iteration is about to be completed soon, allowing us to get familiar with all activities of an interation cycle.

Although we are still conducting the Case Study, we have already started designing the Survey study (Phase 3). Our goal with the survey is to aim for generalization of our findings from Phase 2. So, for now, we have decided that our population is IT professionals located in any place around the world who adopt BDD. We will follow a snowballing sampling to select our sample, that should be as large as possible given the two months we plan to keep the survey open. We will ask the ten participants of our interview step to indicate colleagues and request them to indicate other people. We have also already started to look for additional respondents by inspecting discussion groups in social media websites such as LinkedIn. Eight groups of interest located in Latin America, Europe, and Asia have been identified so far.

Figure 2. Proposed Research Method

The survey will be made available online and will be non-supervised. We have already tested the Qualtrics survey tool to make the questionnaire instrument available but we learned that the license our University owns is limitated to 200 respondents. We are looking for a reliable, free-of-cost alternative solution at this moment.

The survey instrument will be composed of closed questions only, in a Likert-scale format. Response choices will be designed based on the findings from the literature review, our interviews, and the insights from our case study. We have an initial draft but this work will be refined and completed as the case study is finished and data analyzed. The general constructs we are considering to design the questions are as follows: BDD concept, BDD activities, roles involved in the process, artifacts used, tools adopted, benefits of adoption and challenges faced when using BDD. Requirements engineering related specific contructs are as follows: requirements elicitation issues, requirements quality, product quality, communication issues, common ground establishment, and obsolete documentation. Our survey design is following Kitchenham's and Pfleeger's guidelines [Kitchenham and Pfleeger 2002].

4. Final Remarks

BDD aims to promote collaboration and to facilitate communication among stakeholders and the software team. We aim to further our knowledge about BDD usage in the wild by conducting a large-scale survey to confirm findings from our previous initial exploratory studies. This paper presented our high level plan to conduct the survey. We hope that our about-to-come-soon findings will motivate additional practitioners to adopt BDD and researchers to explore other aspects related to this topic.

Acknowledgment

This work is sponsored by the PDTI Program, financed by Dell Computers of Brazil Ltd. (Law 8.248/91).

References

Adzic, G. (2011). *Specification by Example: How Successful Teams Deliver the Right Software*. Manning Publications Co., Greenwich, CT, USA.

Evans, E. (2003). *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Addison-Wesley, Boston, MA, USA.

International, T. S. G. (2013). The chaos manifesto.

Kitchenham, B. A. and Pfleeger, S. L. (2002). Principles of survey research part 2: Designing a survey. *SIGSOFT Softw. Eng. Notes*, 27(1):18–20.

North, D. (2006). Introducing BDD. `http://dannorth.net/introducing-bdd/`.

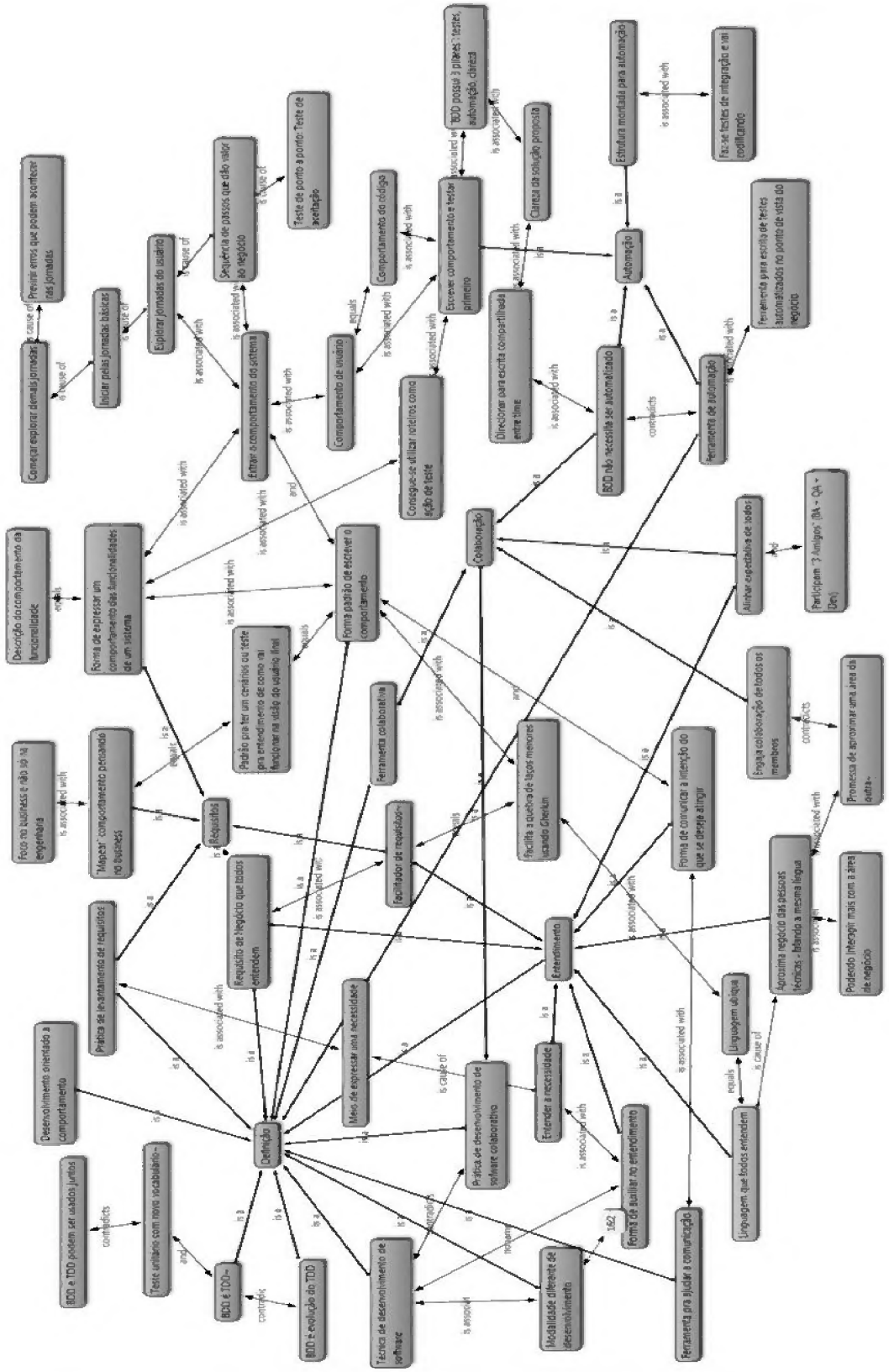
Smart, J. (2014). *BDD in Action: Behavior-Driven Development for the Whole Software Lifecycle*. Manning Publications, Shelter Island, NY.

Sommerville, I. (2010). *Software Engineering*. Addison-Wesley, England, 9 edition.

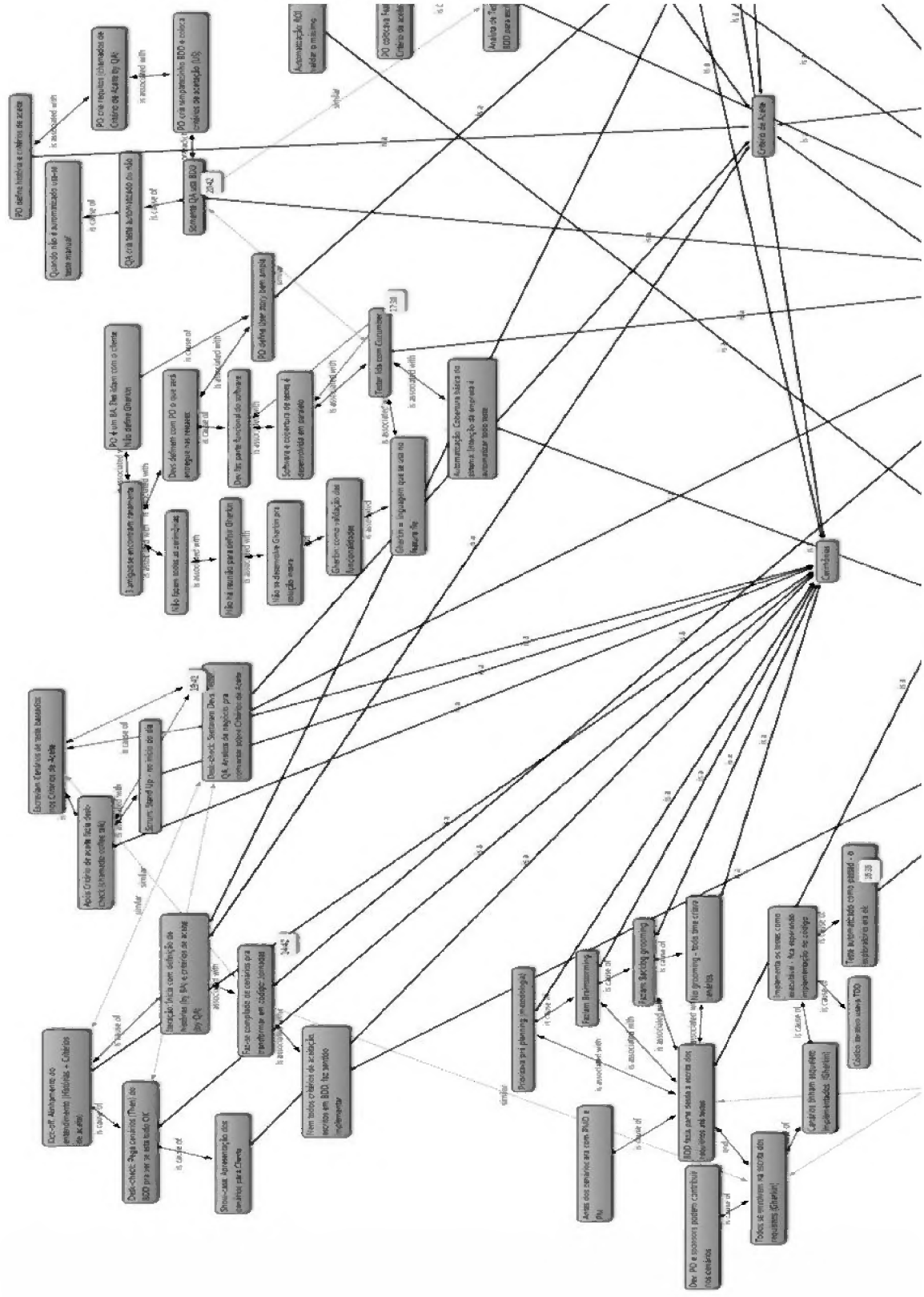
Zowghi, D. and Coulin, C. (2005). *Requirements Elicitation: A Survey of Techniques, Approaches*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

APÊNDICE D – REDES DO ATLAS.TI

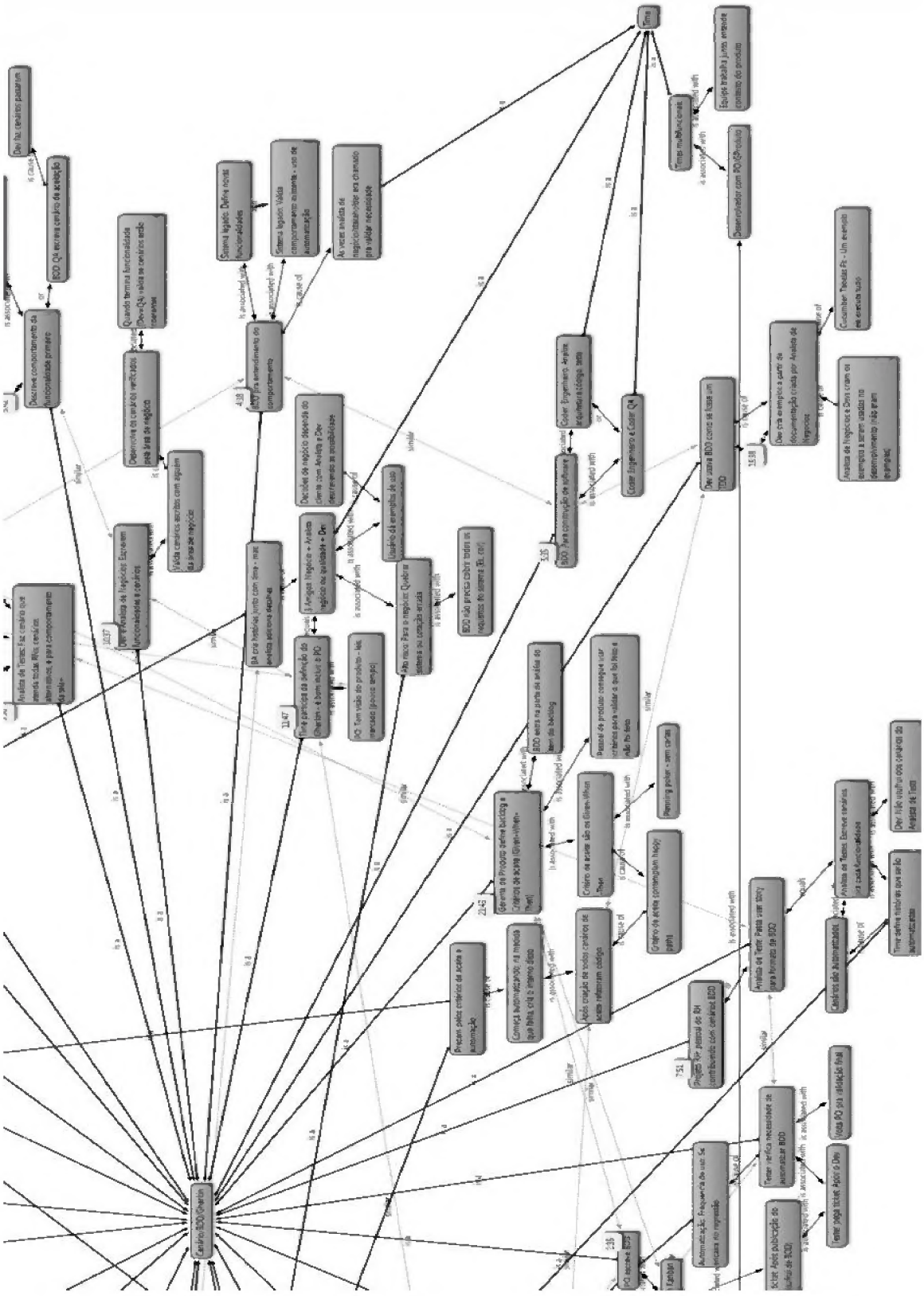
Questão 1 do Roteiro de entrevistas



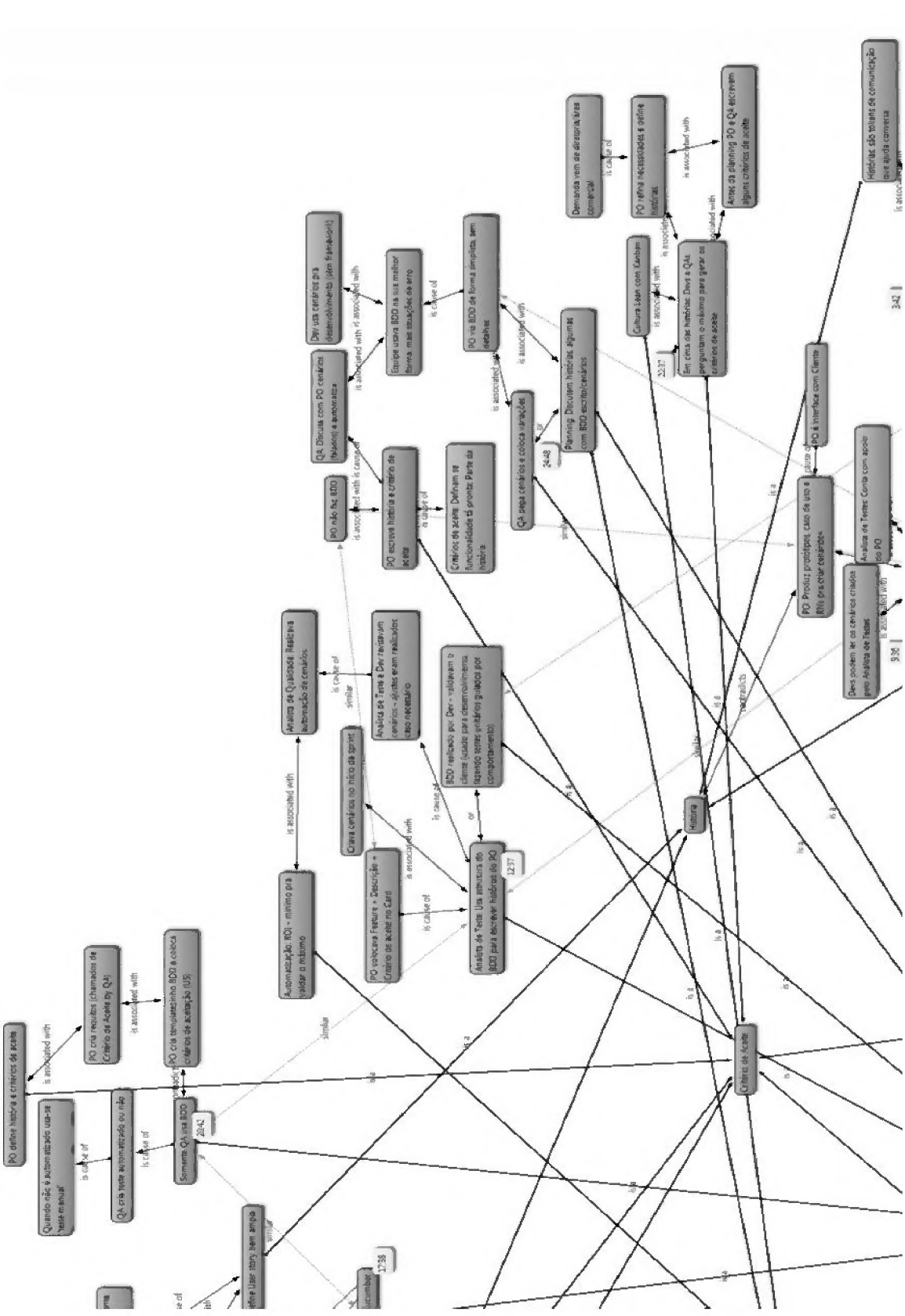
Questão 2 (parte 1/2 - esquerda) do Roteiro de entrevistas



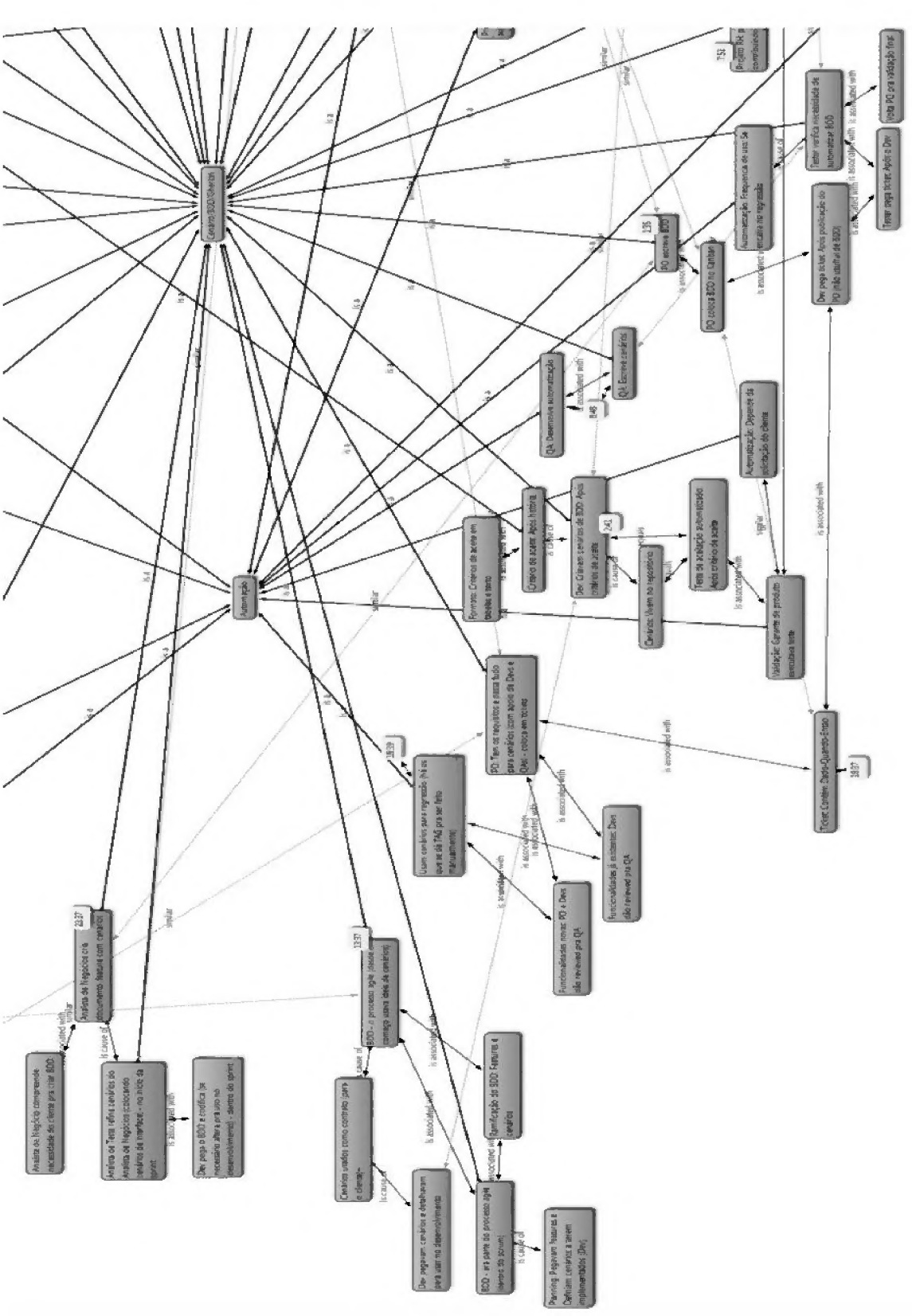
Questão 2 (parte 2/2 - esquerda) do Roteiro de entrevistas

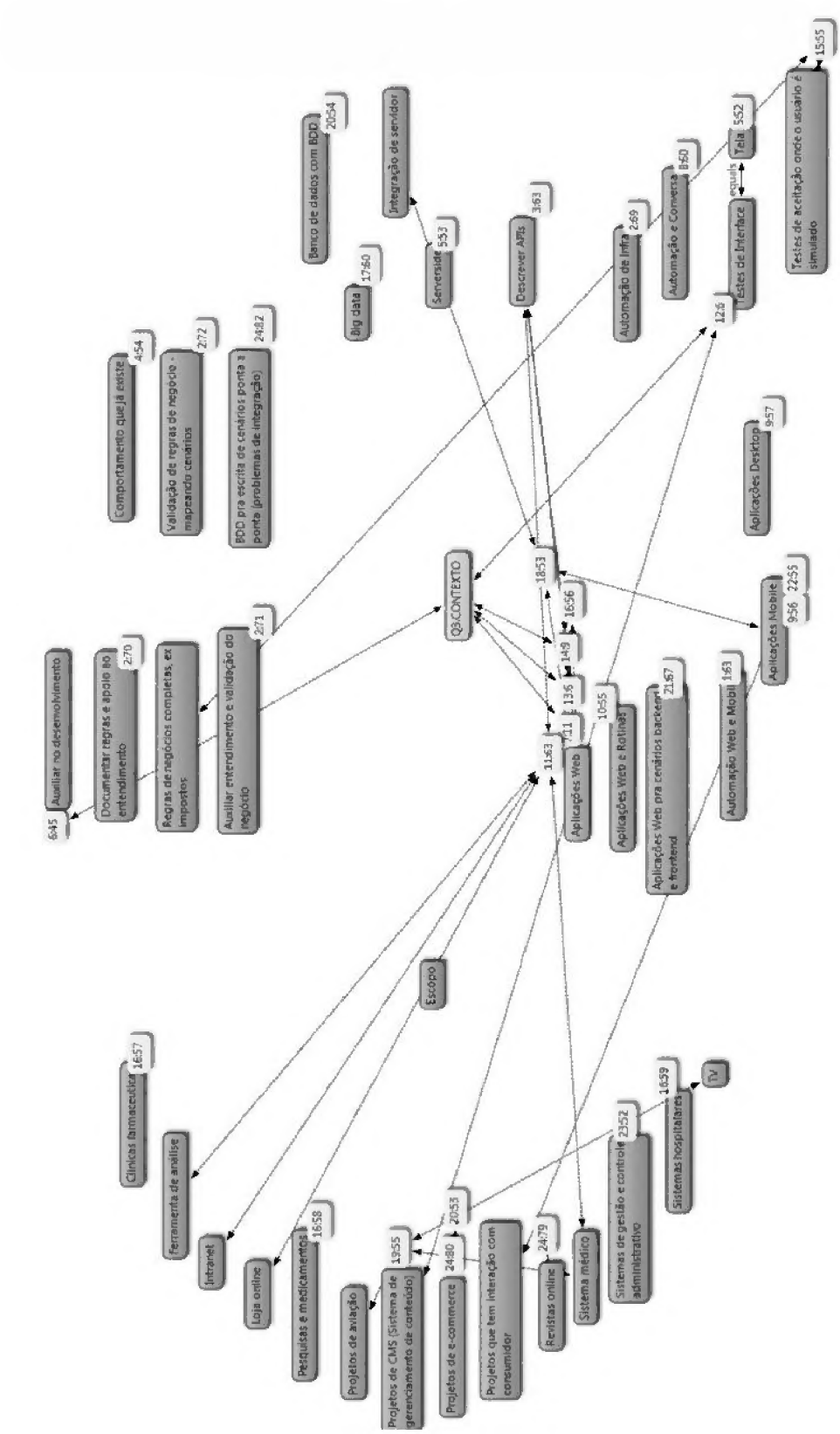


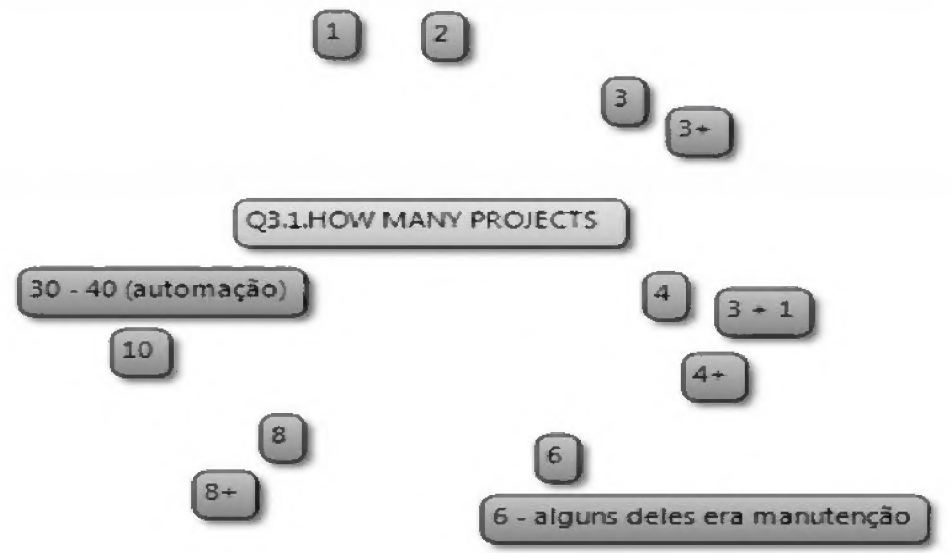
Questão 2 (parte 1/2 - direita) do Roteiro de entrevistas

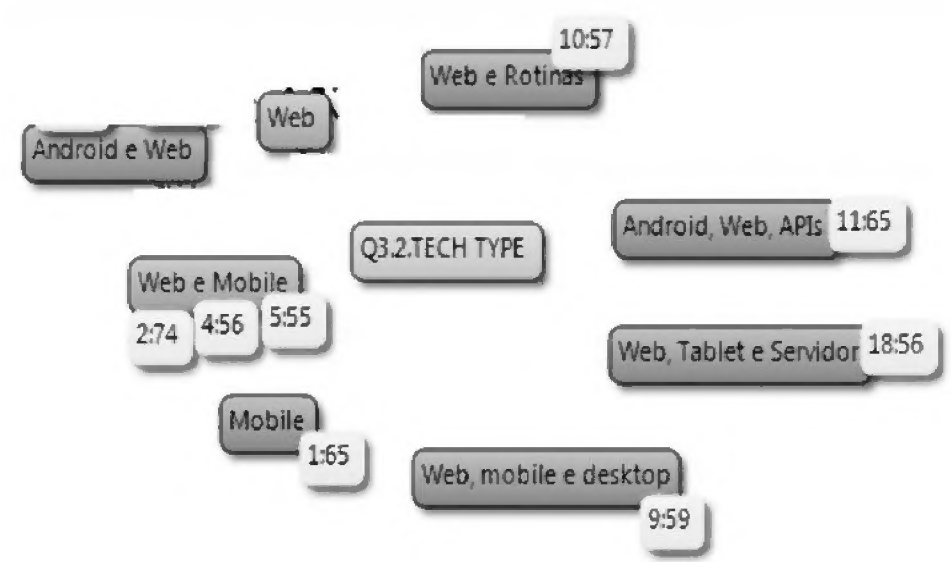


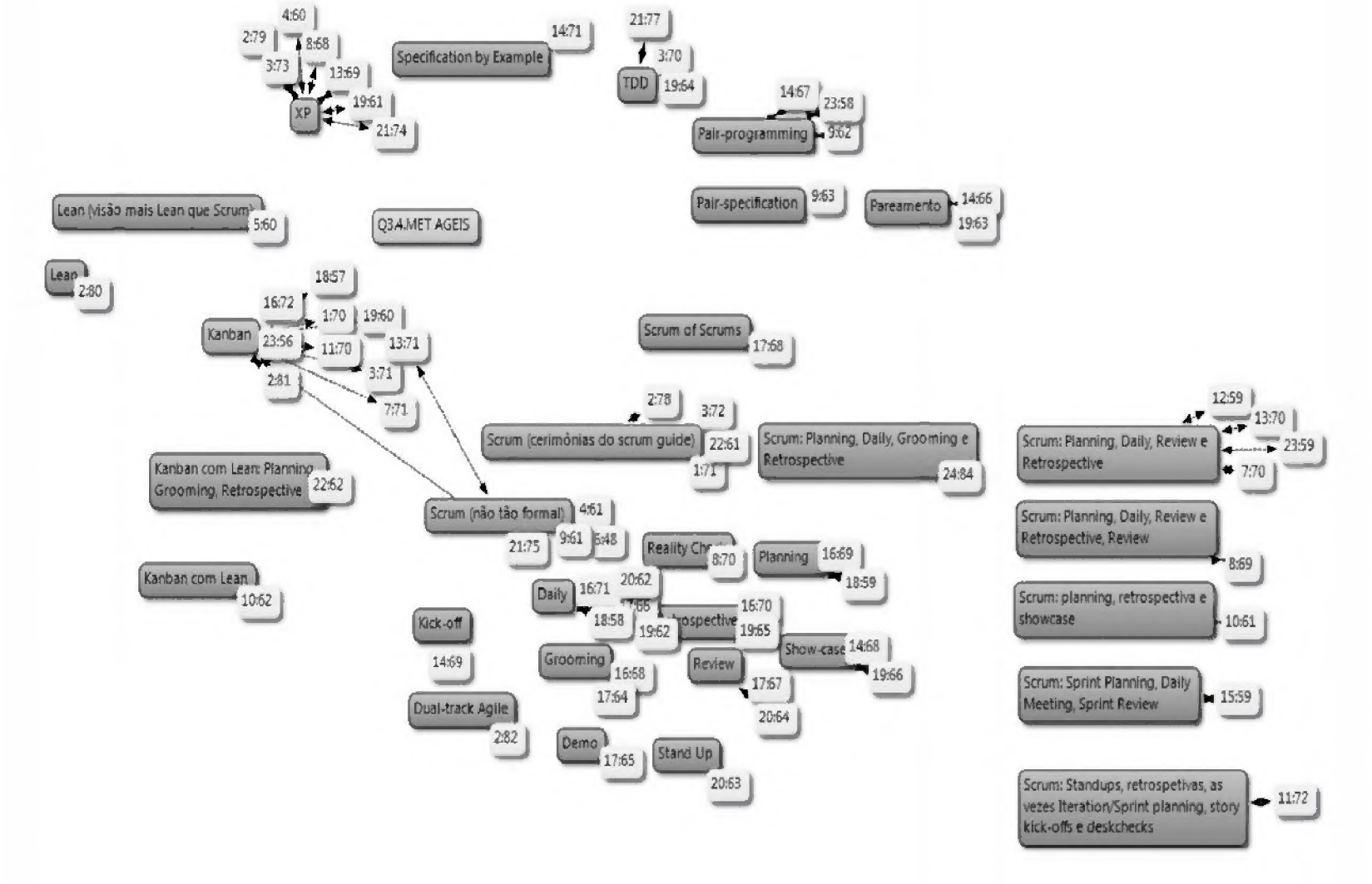
Questão 2 (parte 2/2 - direita) do Roteiro de entrevistas

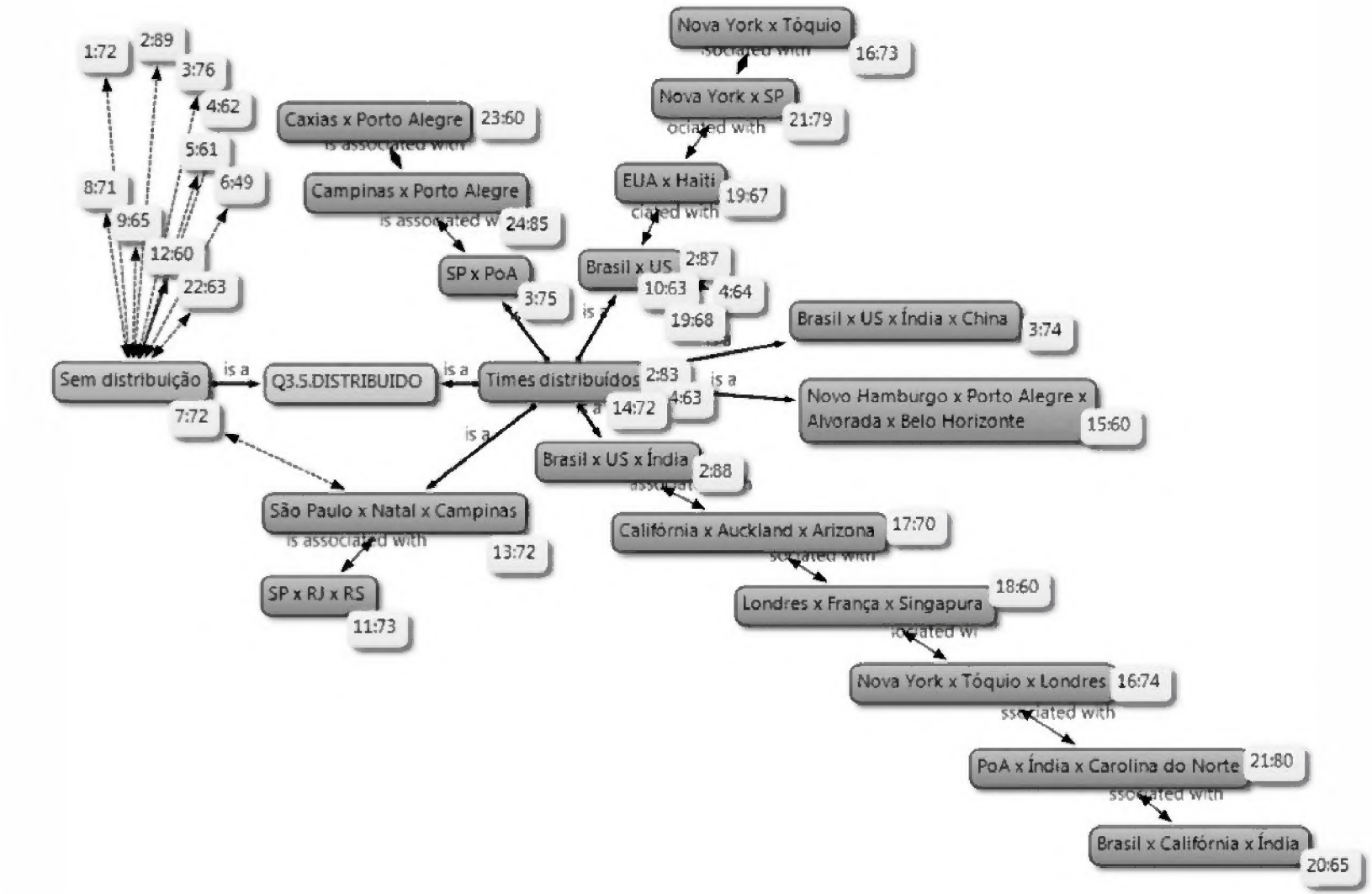


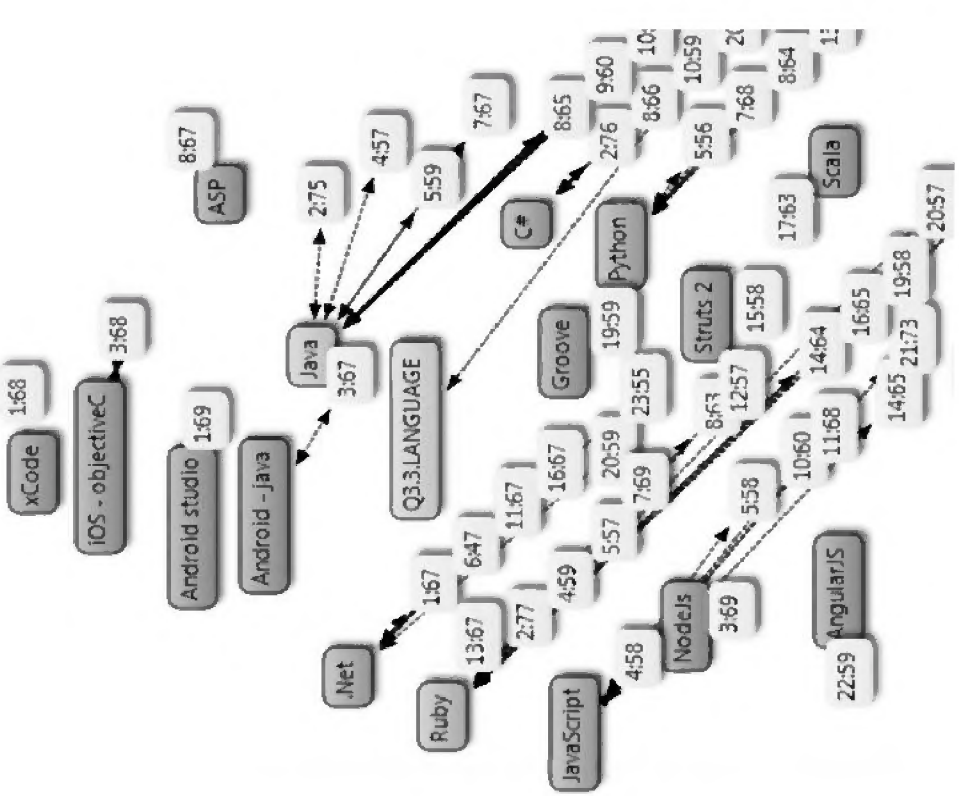

Auxiliar no desenvolvimento
645
Documentar regras e apoio no entendimento
270
Regras de negócios completas, ex importos
Auxiliar entendimento e validação do negócio
271
Comportamento queija existe
454
Validação de regras de negócio - mapeando cenários
272
BDD pra escrita de cenários ponta a ponta (problemas de integração)
2482
Banco de dados com BDD
2054
Big data
1760
Integração de servidor
Serverside
533
Descrever APIs
363
Automação de Infra
269
Automação e Comercial
850
Testes de Interface
Tela
552
igual
126
Testes de aceitação onde o usuário é simulado
1555
Aplicações Desktop
957
Aplicações Mobile
956
2255
Automação Web e Mobil
163
Aplicações Web pra cenários backend e frontend
2157
Aplicações Web e Rotinas
1055
Aplicações Web
711
1363
136
149
1656
1853
QB.CONTEXTO
Escopo
Clínicas farmacêuticas
1657
Ferramenta de análise
Intranet
Loja online
Pesquisas e medicamentos
1658
Projetos de aviação
Projetos de CMS (Sistema de gerenciamento de conteúdo)
1955
Projetos de e-commerce
2480
2053
Projetos que tem interação com consumidor
Revistas online
2479
Sistema médico
Sistemas de gestão e controle administrativo
2352
Sistemas hospitalares
1639
TV

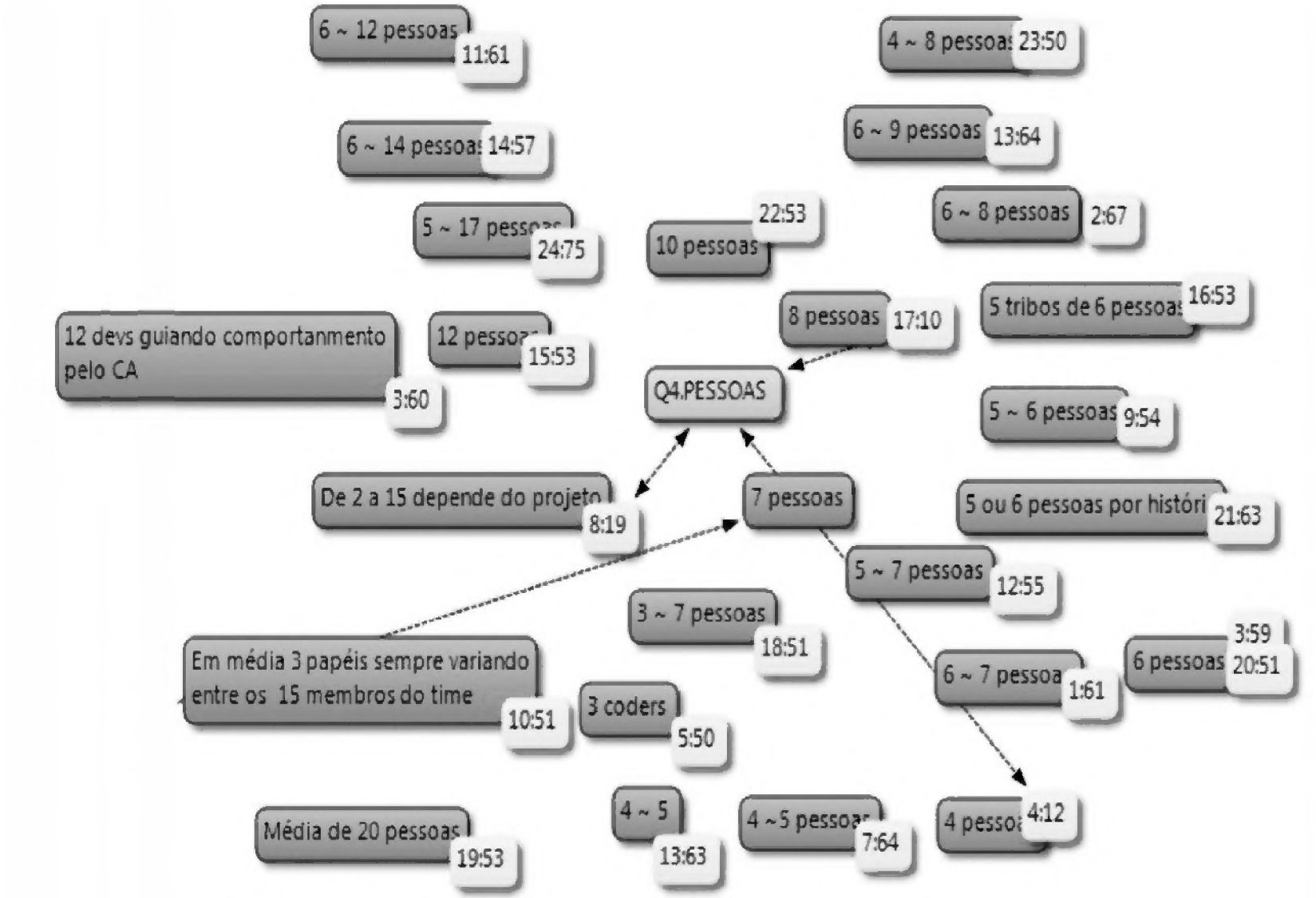

1
2
3
3+
Q3.1.HOW MANY PROJECTS
30 - 40 (automação)
10
4
3 + 1
4+
8
8+
6
6 - alguns deles era manutenção

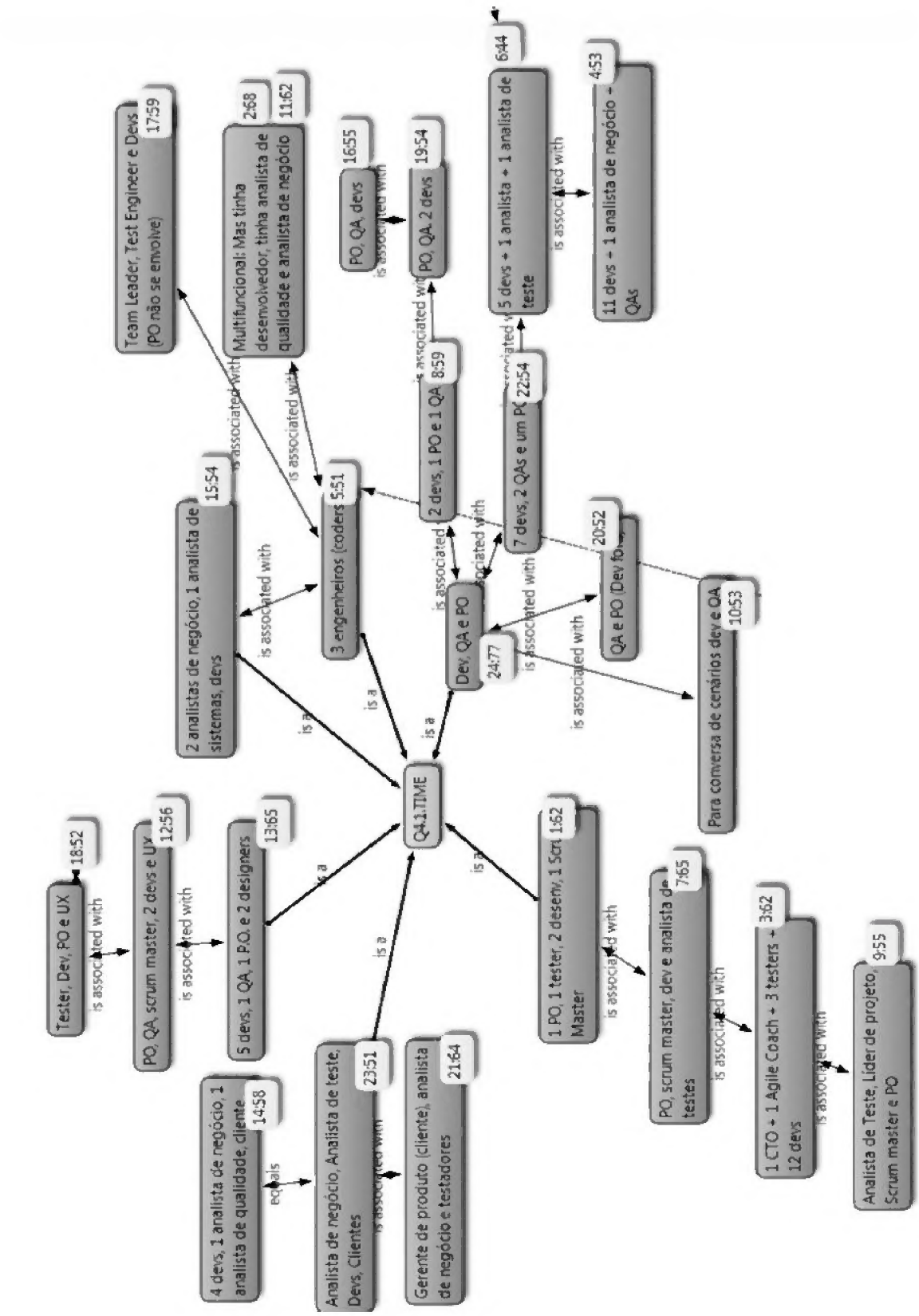

10:57
Web e Rotinas
Web
Android e Web
Android, Web, APIs
11:65
Q3.2.TECH TYPE
Web e Mobile
2:74
4:56
5:55
Web, Tablet e Servidor
18:56
Mobile
1:65
Web, mobile e desktop
9:59

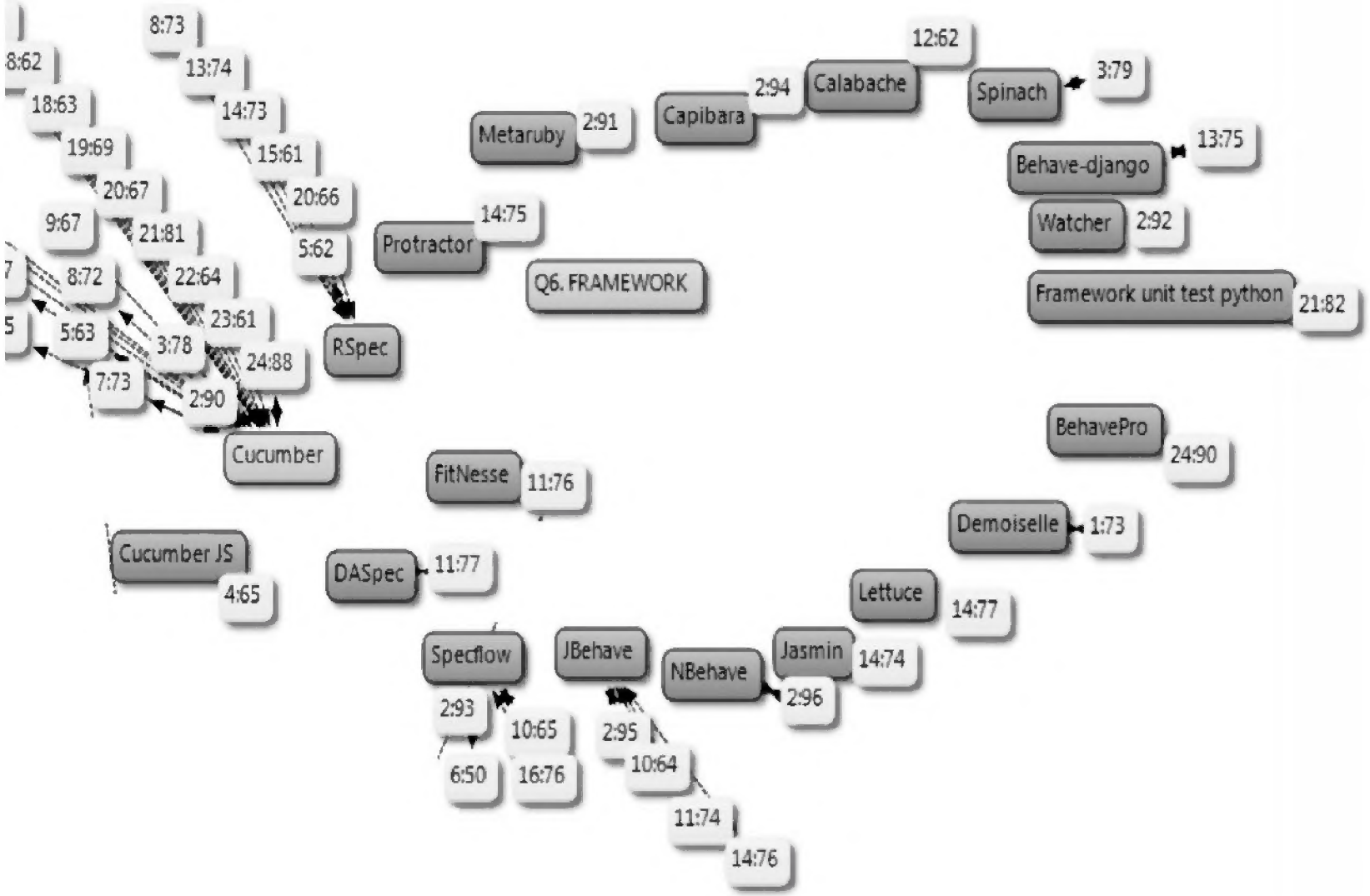

4:60
2:79
8:68
3:73
13:69
XP
19:61
21:74
Specification by Example
14:71
21:77
3:70
TDD
19:64
14:67
23:58
Pair-programming
9:62
Pair-specification
9:63
Pareamento
14:66
19:63
Lean (visão mais Lean que Scrum)
5:60
Q3.4.MET AGEIS
Lean
2:80
18:57
16:72
1:70
19:60
Kanban
23:56
11:70
13:71
3:71
2:81
7:71
Scrum of Scrums
17:68
2:78
3:72
Scrum (cerimônias do scrum guide)
22:61
1:71
Scrum: Planning, Daily, Grooming e Retrospective
24:84
12:59
13:70
Scrum: Planning, Daily, Review e Retrospective
23:59
7:70
Kanban com Lean: Planning Grooming, Retrospective
22:62
Scrum (não tão formal)
4:61
21:75
9:61
5:48
Scrum: Planning, Daily, Review e Retrospective, Review
8:69
Kanban com Lean
10:62
Reality Ch
8:70
Planning
16:69
18:59
Daily
16:71
20:62
18:58
19:62
16:70
19:65
Scrum: planning, retrospectiva e showcase
10:61
Kick-off
14:69
Grooming
16:68
17:64
Review
17:67
20:64
Show-case
14:68
19:66
Scrum: Sprint Planning, Daily Meeting, Sprint Review
15:59
Dual-track Agile
2:82
Demo
17:65
Stand Up
20:63
Scrum: Standups, retrospetivas, as vezes Iteration/Sprint planning, story kick-offs e deskchecks
11:72

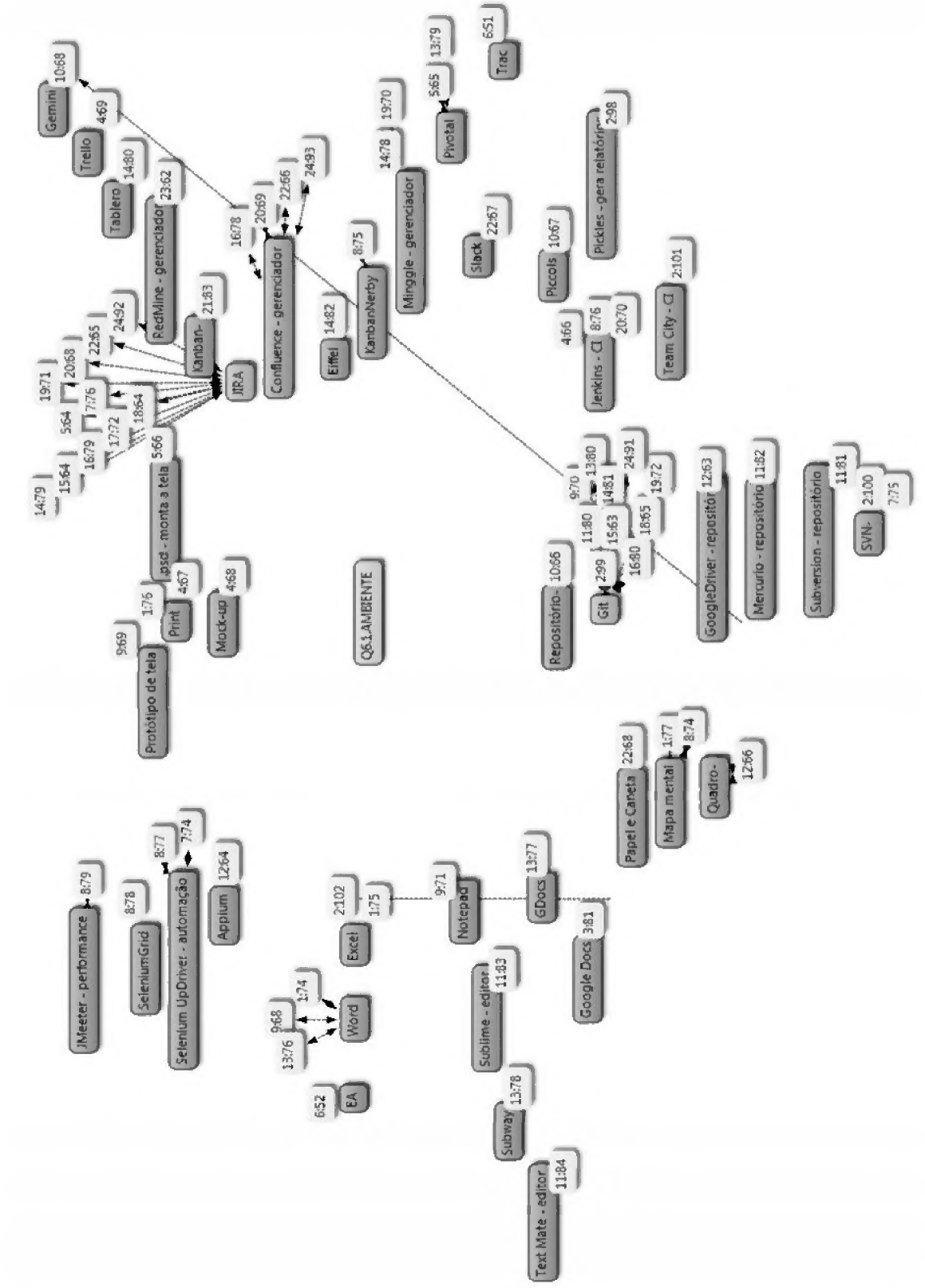

Nova York x Tóquio
16:73
Nova York x SP
21:79
Caxias x Porto Alegre
23:60
Campinas x Porto Alegre
24:85
EUA x Haiti
19:67
SP x PoA
3:75
Brasil x US
2:87
10:63
4:64
19:68
Brasil x US x Índia x China
3:74
1:72
2:89
3:76
4:62
5:61
6:49
8:71
9:65
12:60
22:63
Sem distribuição
is a
Q3.5.DISTRIBUIDO
Times distribuídos
2:83
4:63
14:72
Novo Hamburgo x Porto Alegre x Alvorada x Belo Horizonte
15:60
7:72
Brasil x US x Índia
2:88
São Paulo x Natal x Campinas
13:72
Califórnia x Auckland x Arizona
17:70
SP x RJ x RS
11:73
Londres x França x Singapura
18:60
Nova York x Tóquio x Londres
16:74
PoA x Índia x Carolina do Norte
21:80
Brasil x Califórnia x Índia
20:65

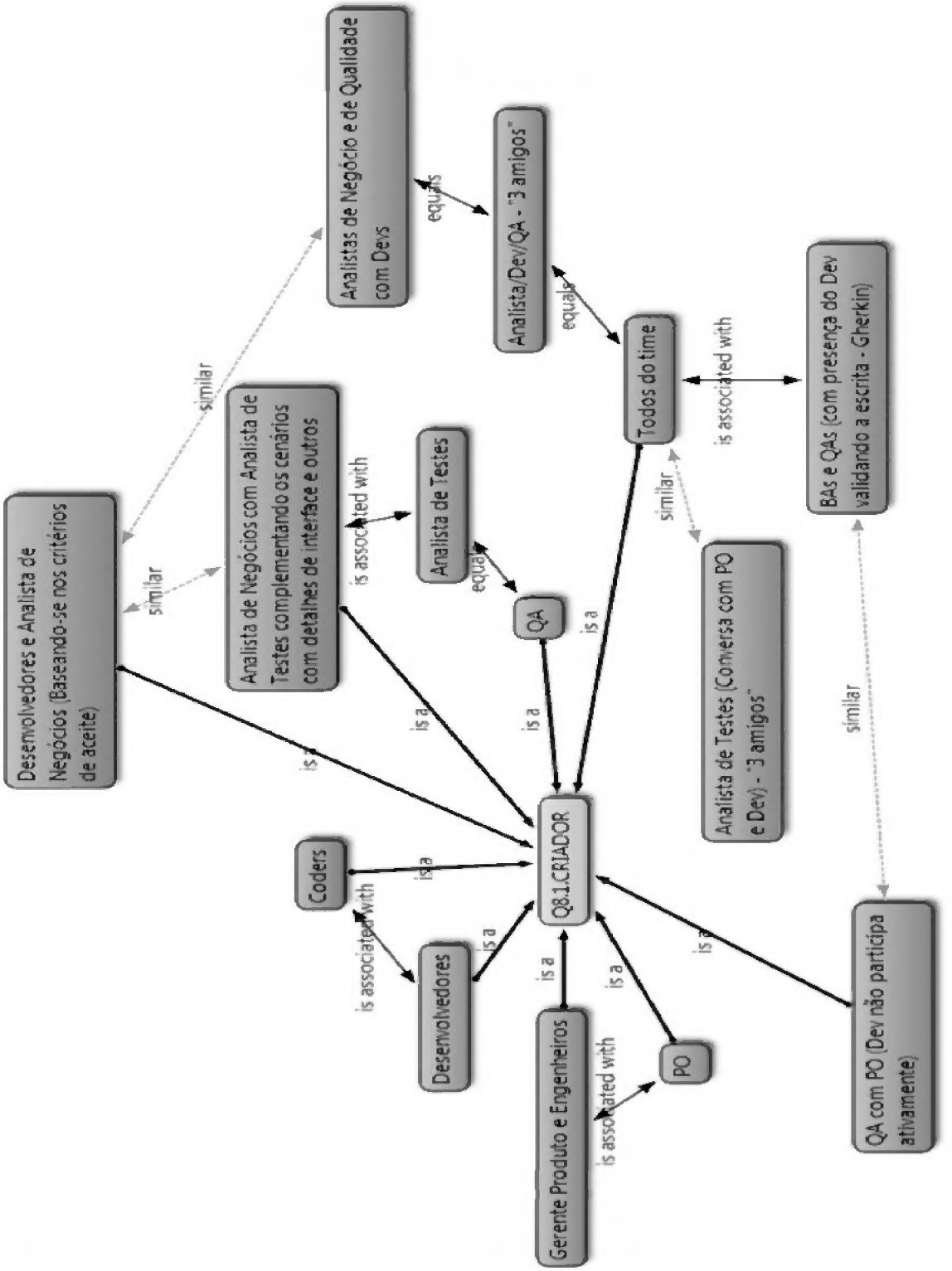

xCode 1:68
iOS - objectiveC 3:68
Android studio 1:69
Android - java
ASP 8:67
Java 3:67
2:75
4:57
5:59
7:67
C# 8:65
2:76
9:60
8:66
10:59
5:56
7:68
8:64
Python 19:59
Groove
Struts 2
15:58
Scala 17:63
20:57
19:58
21:73
16:65
14:65
14:64
11:68
10:60
Q3.3.LANGUAGE
23:55
12:57
7:69
20:59
16:67
11:67
5:57
4:59
2:77
6:47
1:67
.Net
Ruby 13:67
JavaScript 4:58
NodeJs 3:69
5:58
AngularJS 22:59

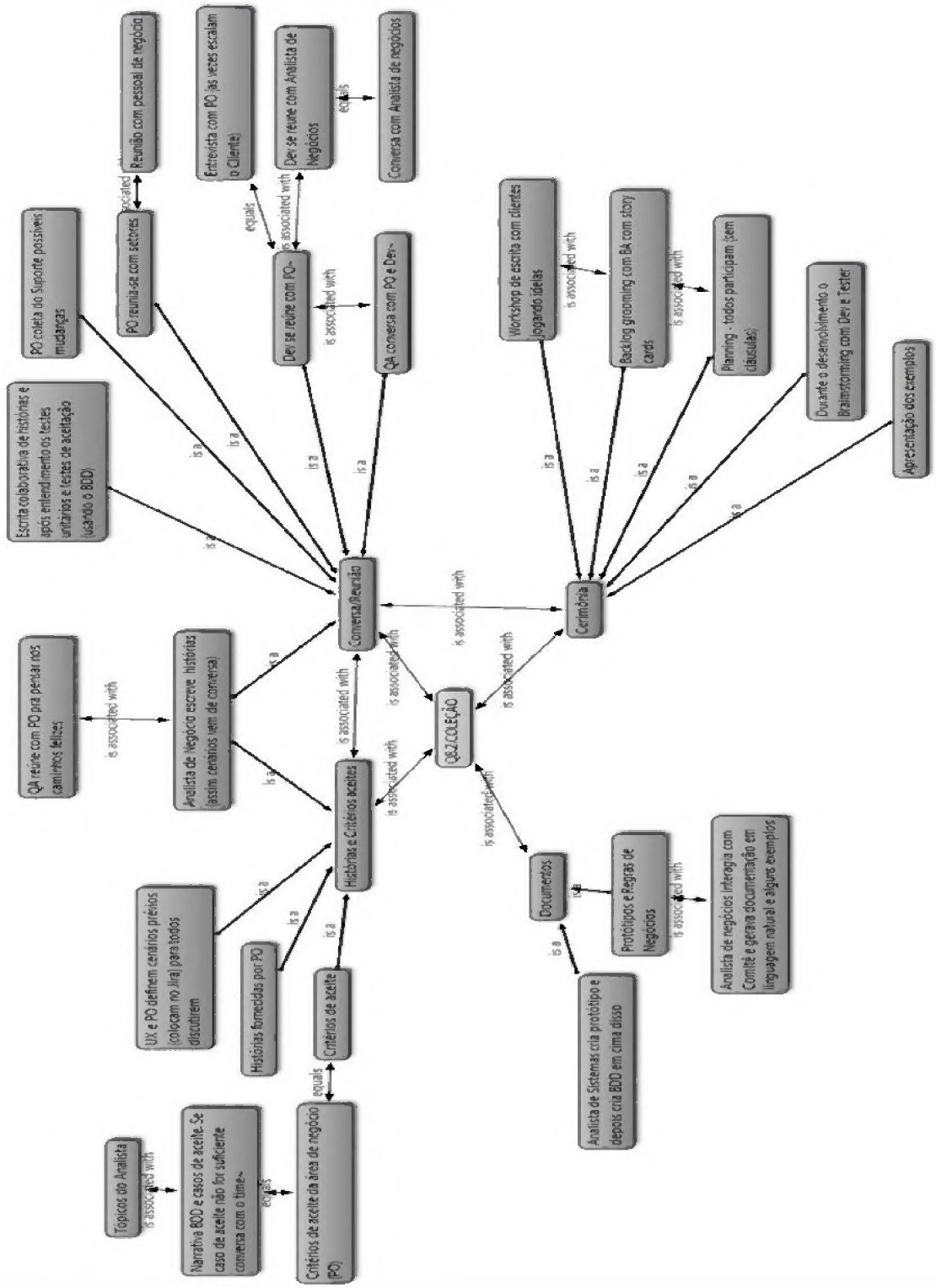

6 ~ 12 pessoas
11:61
4 ~ 8 pessoas
23:50
6 ~ 14 pessoas
14:57
6 ~ 9 pessoas
13:64
5 ~ 17 pessoas
24:75
22:53
10 pessoas
6 ~ 8 pessoas
2:67
8 pessoas
17:10
5 tribos de 6 pessoas
16:53
12 devs guiando comportanmento pelo CA
3:60
12 pessoas
15:53
Q4.PESSOAS
5 ~ 6 pessoas
9:54
De 2 a 15 depende do projeto
8:19
7 pessoas
5 ou 6 pessoas por histórí
21:63
5 ~ 7 pessoas
12:55
3 ~ 7 pessoas
18:51
3:59
6 pessoas
20:51
Em média 3 papéis sempre variando entre os 15 membros do time
10:51
3 coders
5:50
6 ~ 7 pessoa
1:61
4 ~ 5
13:63
4 ~5 pessoas
7:64
4 pessoa
4:12
Média de 20 pessoas
19:53

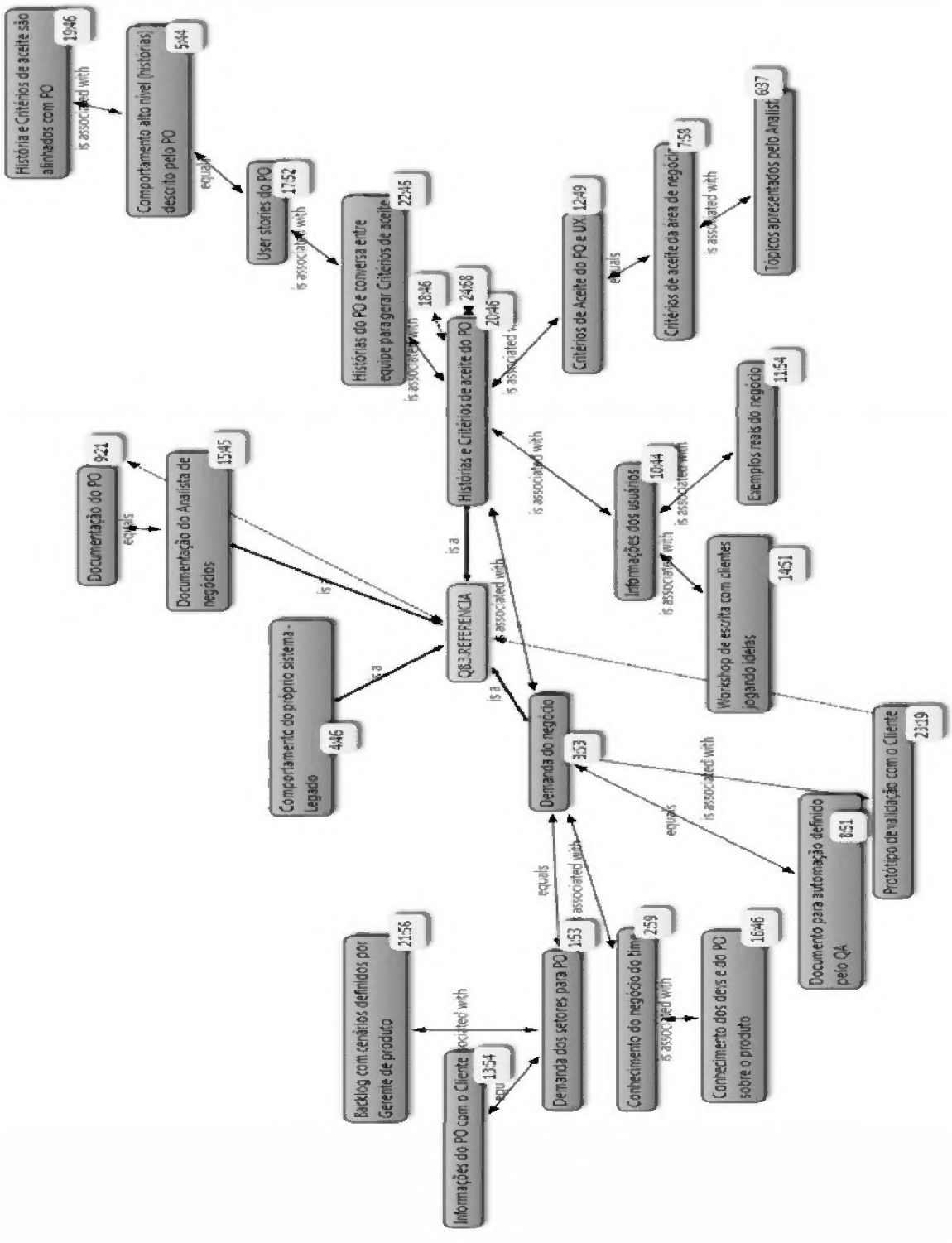

Q4.1.TIME
Tester, Dev, PO e UX 18:52
PO, QA, scrum master, 2 devs e UX 12:56
5 devs, 1 QA, 1 P.O. e 2 designers 13:65
4 devs, 1 analista de negócio, 1 analista de qualidade, cliente 14:58
Analista de negócio, Analista de teste, Devs, Clientes 23:51
Gerente de produto (cliente), analista de negócio e testadores 21:64
2 analistas de negócio, 1 analista de sistemas, devs 15:54
3 engenheiros (coders 5:51
Team Leader, Test Engineer e Devs (PO não se envolve) 17:59
Multifuncional: Mas tinha desenvolvedor, tinha analista de qualidade e analista de negócio 2:68 11:62
2 devs, 1 PO e 1 QA 8:59
PO, QA, devs 16:55
PO, QA, 2 devs 19:54
Dev, QA e PO 24:77
7 devs, 2 QAs e um P 22:54
5 devs + 1 analista + 1 analista de teste 6:44
11 devs - 1 analista de negócio - QAs 4:53
QA e PO (Dev fo 20:52
Para conversa de cenários dev e QA 10:53
1 PO, 1 tester, 2 desenv, 1 Scru 1:62 Master
PO, scrum master, dev e analista de testes 7:65
1 CTO + 1 Agile Coach + 3 testers + 12 devs 3:62
Analista de Teste, Líder de projeto, Scrum master e PO 9:55
is a
is associated with
equals

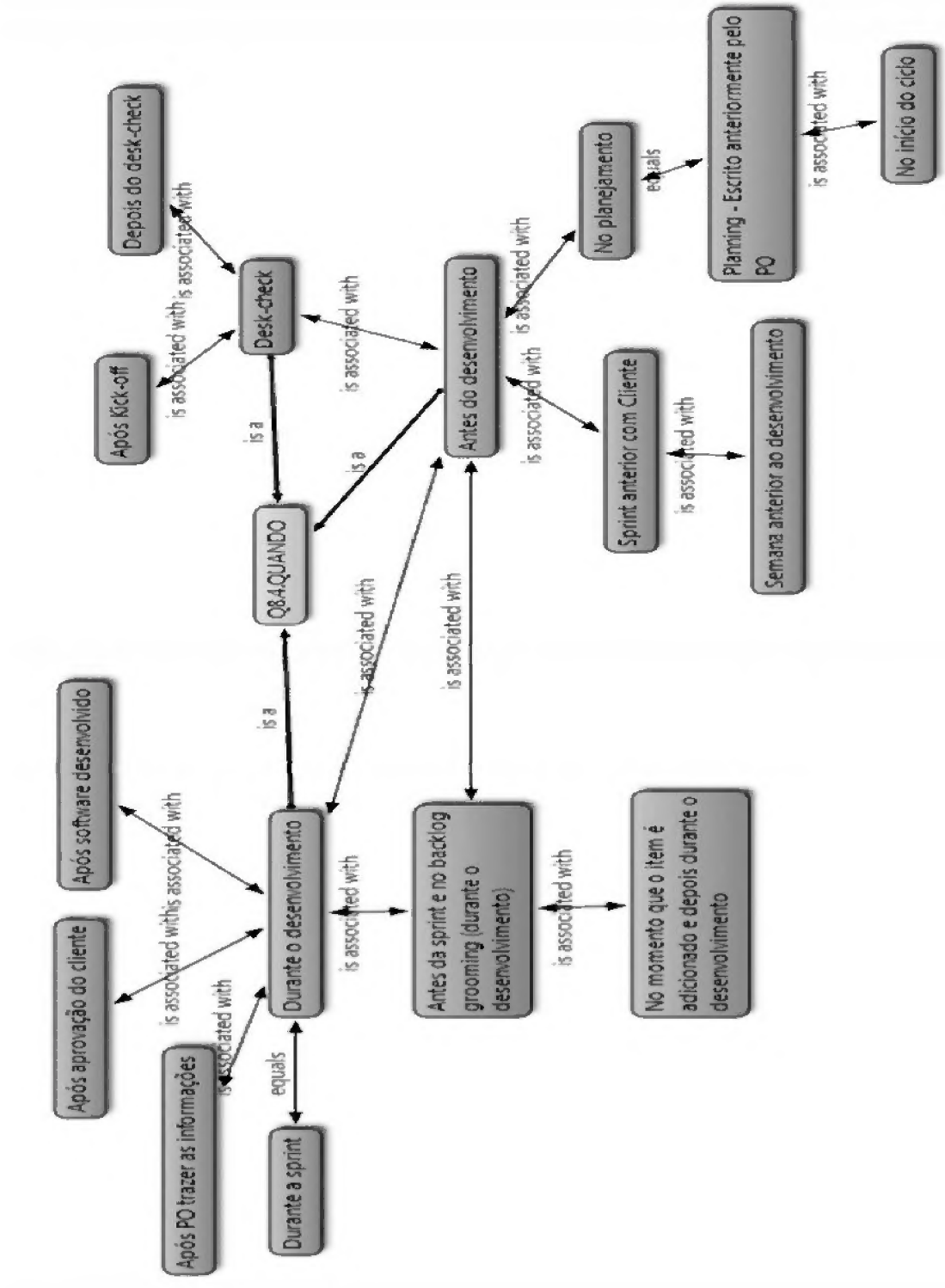

8:73
13:74
14:73
15:61
20:66
5:62
RSpec
Cucumber
2:90
7:73
3:78
24:88
23:61
22:64
21:81
20:67
19:69
18:63
8:62
9:67
8:72
5:63
Metaruby
2:91
Protractor
14:75
Q6. FRAMEWORK
Capibara
2:94
Calabache
12:62
Spinach
3:79
Behave-django
13:75
Watcher
2:92
Framework unit test python
21:82
BehavePro
24:90
Demoiselle
1:73
Lettuce
14:77
Jasmin
14:74
2:96
NBehave
JBehave
2:95
10:64
11:74
14:76
Specflow
2:93
10:65
6:50
16:76
DASpec
11:77
FitNesse
11:76
Cucumber JS
4:65

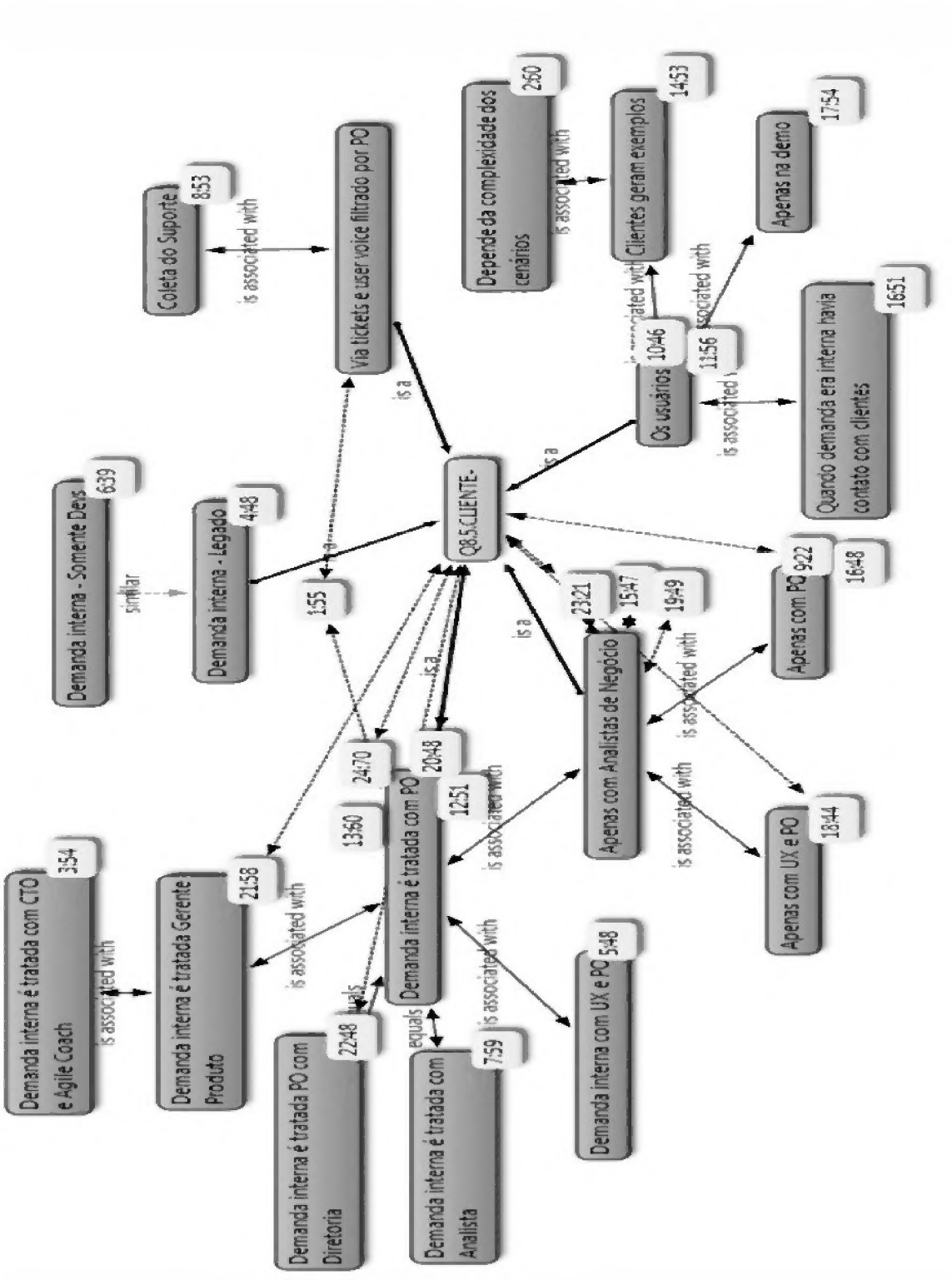

Gemini 10:68
Trello 4:69
Tablero 14:80
RedMine - gerenciador 23:62
Kanban- 21:83
JIRA
Confluence - gerenciador 20:69
Eiffel 14:82
KanbanNerby 8:75
Minggle - gerenciador 14:78
Pivotal 5:65
Trac 6:51
Slack 22:67
Piccols 10:67
Pickles - gera relatórios 2:98
Jenkins - CI 8:76
Team City - CI 2:101
Repositório- 10:66
Git 2:99
GoogleDriver - repositór 12:63
Mercurio - repositório 11:62
Subversion - repositório 11:81
SVN- 2:100
Prototipo de tela 9:69
Print 4:67
Mock-up 4:68
Q61.AMBIENTE
JMeeter - performance 8:79
SeleniumGrid 8:78
Selenium UpDriver - automação 7:74
Appium 12:64
Excel 2:102
Word 1:74
EA 6:52
Notepad 9:71
GDocs 13:77
Google Docs 3:81
Sublime - editor 11:83
Subway 13:78
Text Mate - editor 11:84
Papel e Caneta 22:68
Mapa mental 1:77
Quadro- 8:74
12:66

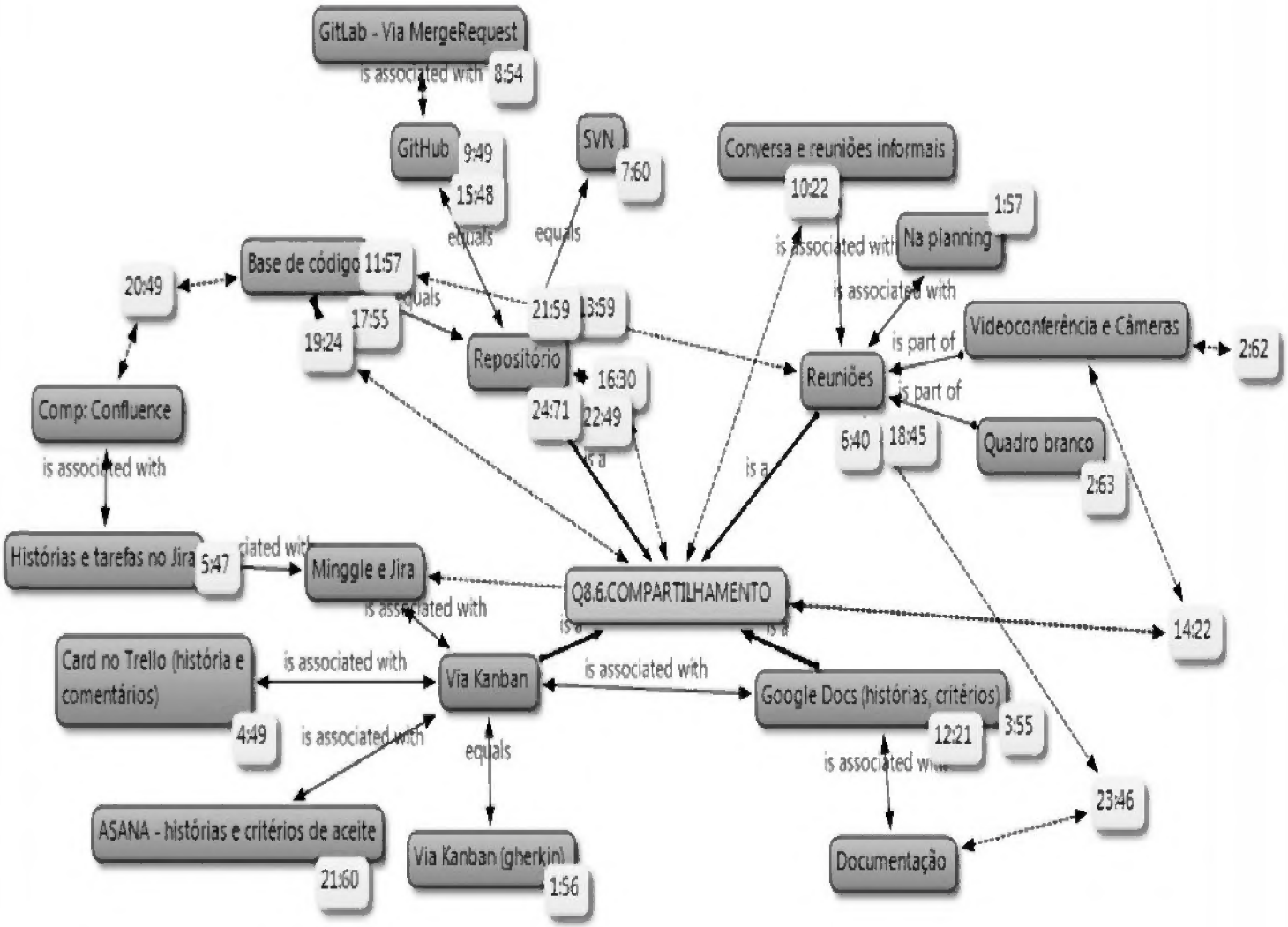

Desenvolvedores e Analista de Negócios (Baseando-se nos critérios de aceite)
Analistas de Negócio e de Qualidade com Devs
Analista/Dev/QA - "3 amigos"
Analista de Negócios com Analista de Testes complementando os cenários com detalhes de interface e outros
Analista de Testes
QA
Todos do time
BAs e QAs (com presença do Dev validando a escrita - Gherkin)
Analista de Testes (Conversa com PO e Dev) - "3 amigos"
Coders
Q8.1.CRIADOR
Desenvolvedores
Gerente Produto e Engenheiros
PO
QA com PO (Dev não participa ativamente)
similar
equals
is associated with
is a

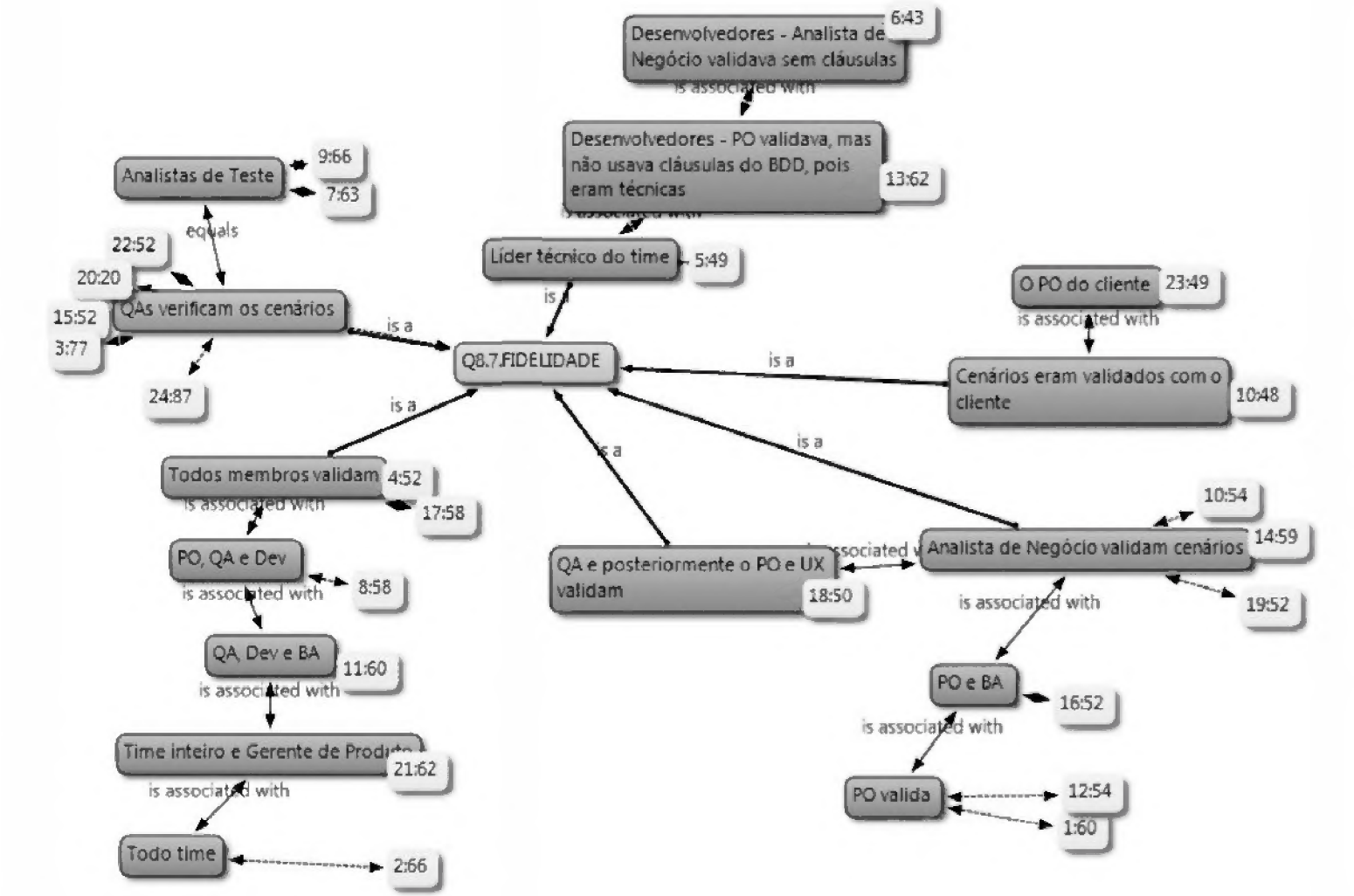

Reunião com pessoal de negócio
Entrevista com PO (as vezes escalam o Cliente)
Dev se reúne com Analista de Negócios
Conversa com Analista de negócios
PO coleta do Suporte possíveis mudanças
PO reuinia-se com setores
Dev se reúne com PO-
QA conversa com PO e Dev-
Escrita colaborativa de histórias e após enlerdimento os testes unitários e testes de aceitação (usando o BDD)
Workshop de escrita com clientes (jogando ideias
Backlog grooming com BA com story cards
Planning - todos participam (sem cláusulas)
Durante o desenvolvimento o Brainstorming com Dev e Tester
Apresentação dos exemplos
QA reúne com PO pra pensar nos cenarinhos felizes
Analista de Negocio escreve histórias (assim cenarios vem de conversa)
Conversa/Reunião
Cerimônia
Histórias e Critérios aceites
QBZ COLEÇÃO
UX e PO definem cenários prévios (colocam no Jira) para todos discutirem
Histórias fornecidas por PO
Critérios de aceite
Documentos
Protótipos e Regras de Negócios
Analista de negócios interagia com Comitê e gerava documentação em linguagem natural e alguns exemplos
Analista de Sistemas cria protótipo e depois cria BDD em cima disso
Tópicos do Analista
Narrativa BDD e casos de aceite. Se caso de aceite não for suficiente conversa com o time-
Critérios de aceite da área de negócio (PO)
is a
is associated with
equals

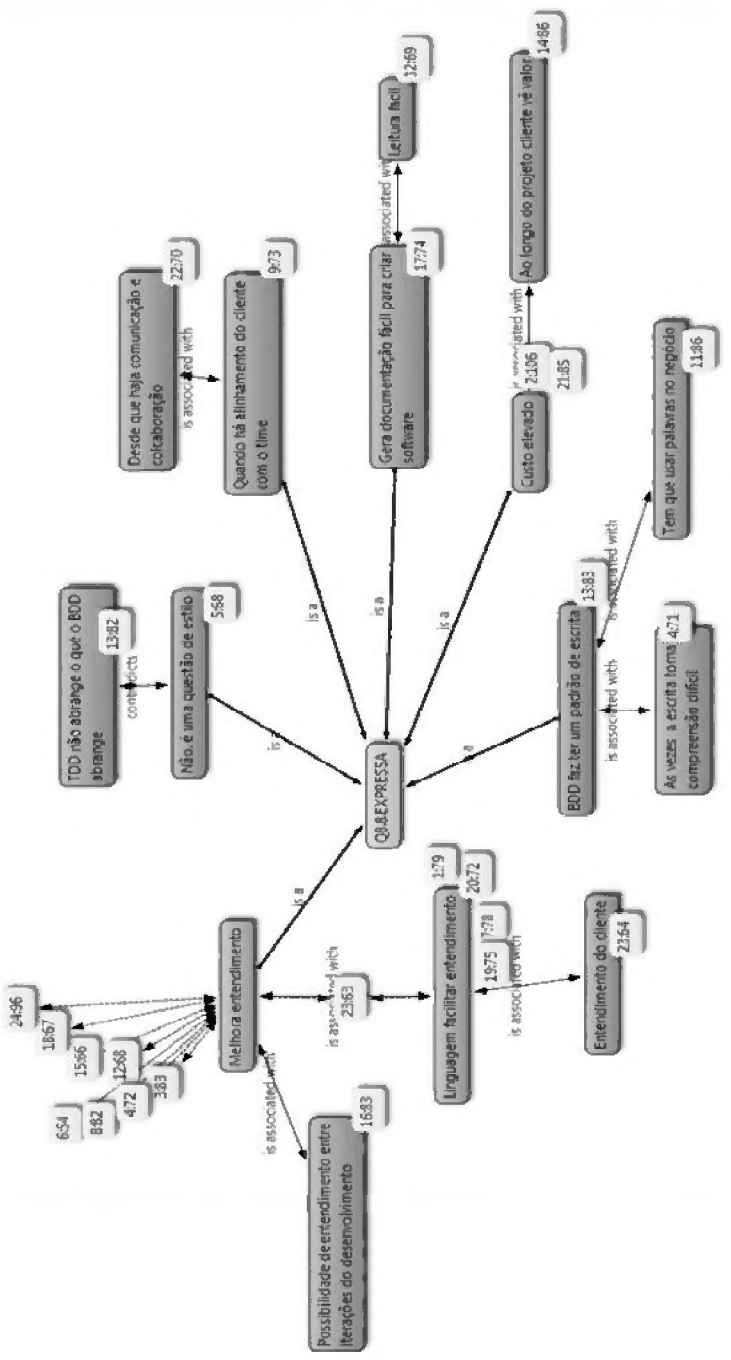

História e Critérios de aceite são alinhados com PO
19:46
is associated with
Comportamento alto nivel (histórias) descrito pelo PO
5:44
equals
User stories do PO
17:52
is associated with
Histórias do PO e comersa entre equipe para gerar Critérios de aceite
22:46
is associated with
18:46
Histórias e Critérios de aceite do PO
24:68
20:46
is associated with
Critérios de Aceite do PO e UX
12:49
equals
Critérios de aceite da area de negócio
7:59
is associated with
Tópicos apresentados pelo Analista
6:37
Documentação do PO
9:21
equals
Documentação do Analista de negócios
15:45
is a
Comportamento do próprio sistema - Legado
4:46
is a
QB3.REFERENCIA
is associated with
is a
is associated with
Informações dos usuários
10:44
is associated with
Exemplos reais do negócio
11:54
Workshop de escrita com clientes jogando ideias
14:51
Demanda do negócio
3:53
equals
is associated with
Protótipo de validação com o Cliente
23:19
Documento para automação definido pelo QA
8:51
Backlog com cenários definidos por Gerente de produto
21:56
Informações do PO com o Cliente
13:54
equals
Demanda dos setores para PO
1:53
associated with
Conhecimento do negocio do time
2:59
is associated with
Conhecimento dos devs e do PO sobre o produto
16:46

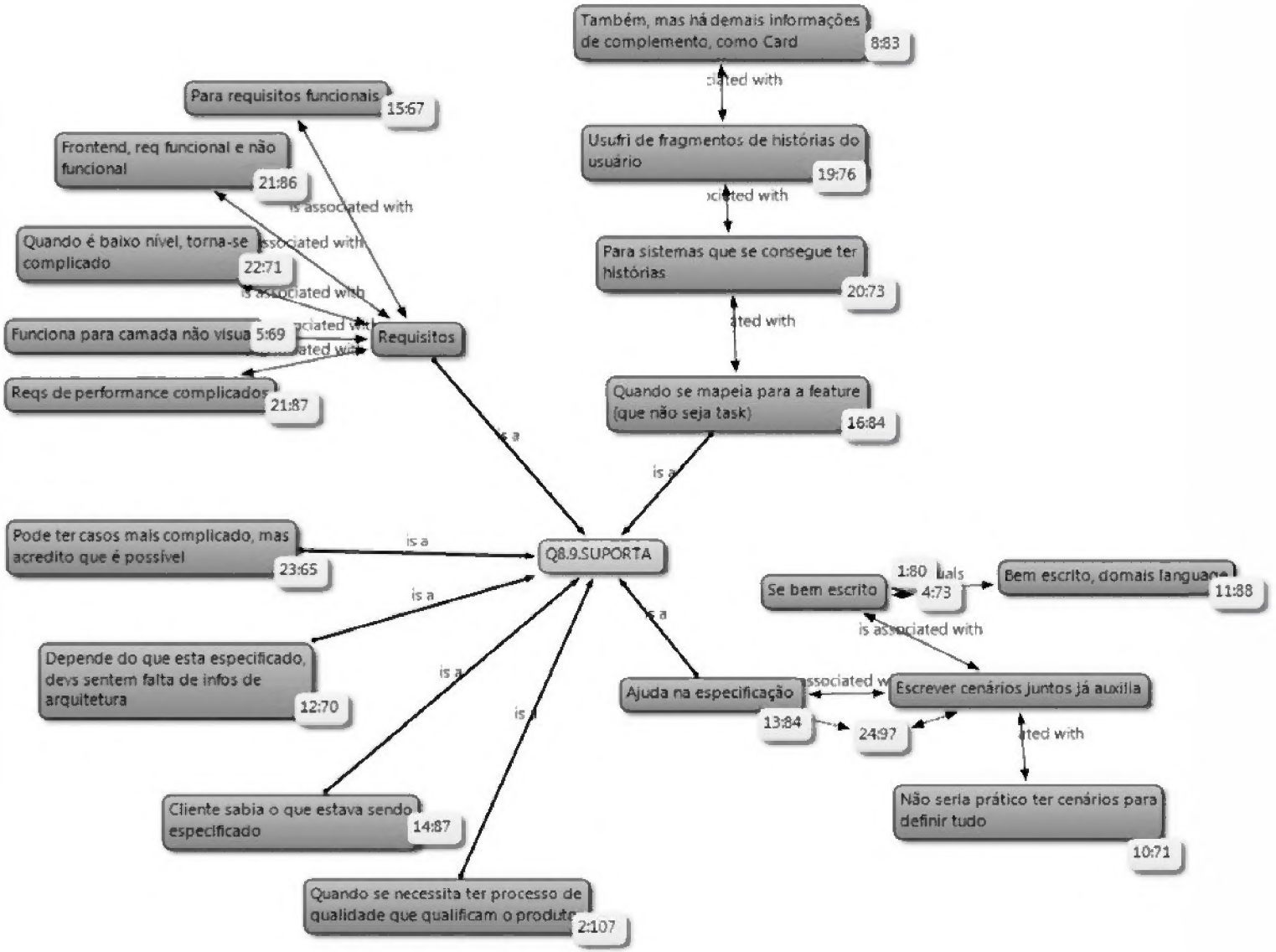

Depois do desk-check
Após Kick-off
Desk-check
Q8.4.QUANDO
Antes do desenvolvimento
No planejamento
Planning - Escrito anteriormente pelo PO
No início do ciclo
Sprint anterior com Cliente
Semana anterior ao desenvolvimento
Após software desenvolvido
Após aprovação do cliente
Após PO trazer as informações
Durante o desenvolvimento
Durante a sprint
Antes da sprint e no backlog grooming (durante o desenvolvimento)
No momento que o item é adicionado e depois durante o desenvolvimento
is associated with
is a
equals

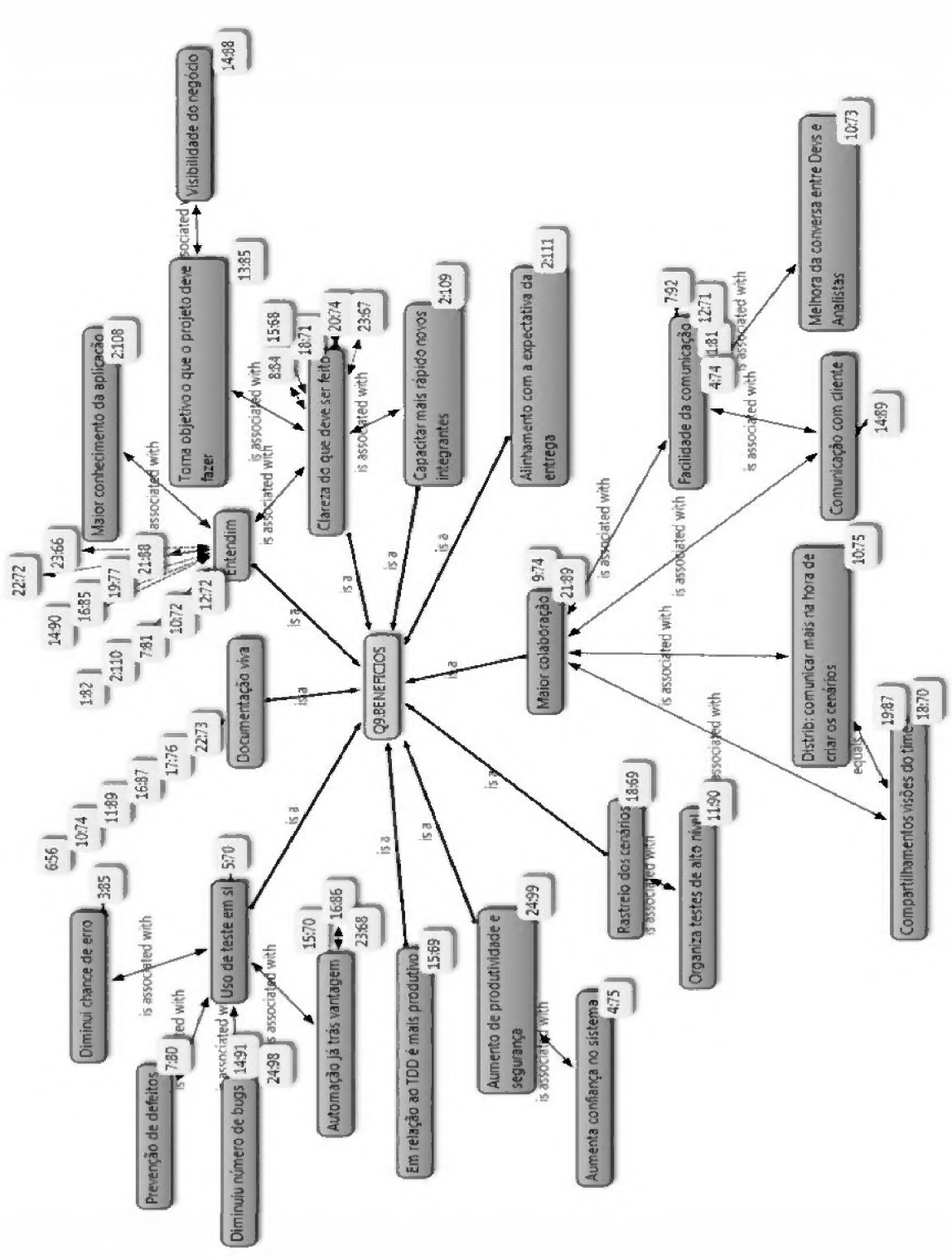

Demanda interna é tratada com CTO e Agile Coach
354
is associated with
Demanda interna é tratada Gerente Produto
2158
is associated with
Demanda interna é tratada PO com Diretoria
2248
equals
Demanda interna é tratada com Analista
759
is associated with
Demanda interna é tratada com PO
2048
1360
2470
1251
is associated with
Demanda interna com UX e PO
548
Demanda interna - Somente Devs
639
similar
Demanda interna - Legado
448
155
Coleta do Suporte
853
is associated with
Via tickets e user voice filtrado por PO
is a
Depende da complexidade dos cenários
260
is associated with
Clientes geram exemplos
1453
Os usuários
1046
1156
is associated with
Apenas na demo
1754
Quando demanda era interna havia contato com clientes
1651
Q8.5.CLIENTE-
is a
2321
1547
1949
Apenas com Analistas de Negócio
is associated with
Apenas com PO
922
1648
Apenas com UX e PO
1844

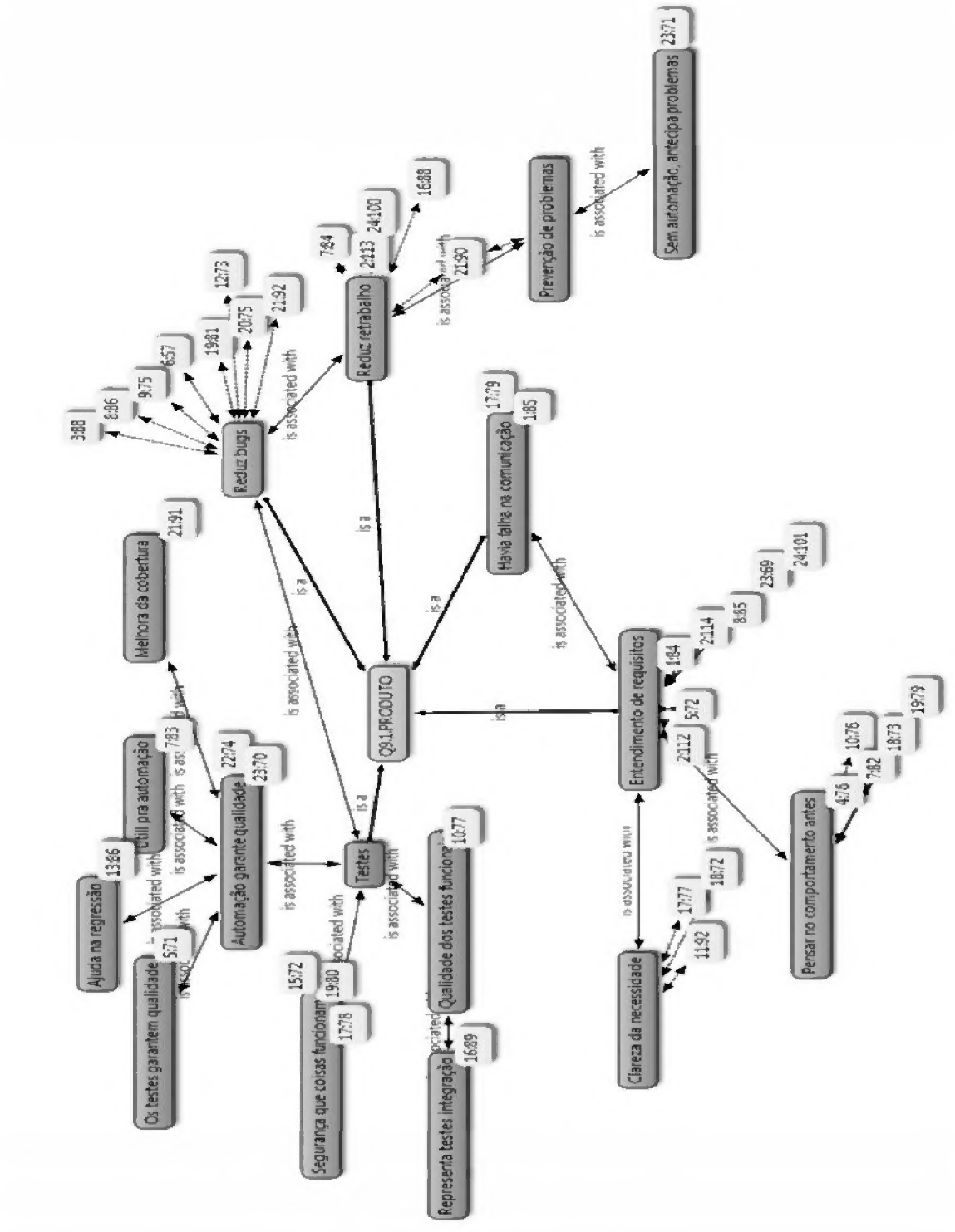

GitLab - Via MergeRequest
is associated with
8:54
GitHub
9:49
15:48
equals
SVN
7:60
equals
Conversa e reuniões informais
10:22
is associated with
Na planning
1:57
is associated with
Base de código
11:57
20:49
17:55
19:24
21:59
13:59
Repositório
16:30
24:71
22:49
Reuniões
is part of
Videoconferência e Câmeras
2:62
is part of
Quadro branco
2:63
6:40
18:45
Comp: Confluence
is associated with
Histórias e tarefas no Jira
5:47
Minggle e Jira
Q8.6.COMPARTILHAMENTO
14:22
Card no Trello (história e comentários)
4:49
is associated with
Via Kanban
is associated with
Google Docs (histórias, critérios)
12:21
3:55
is associated with
equals
ASANA - histórias e critérios de aceite
21:60
Via Kanban (gherkin)
1:56
Documentação
23:46

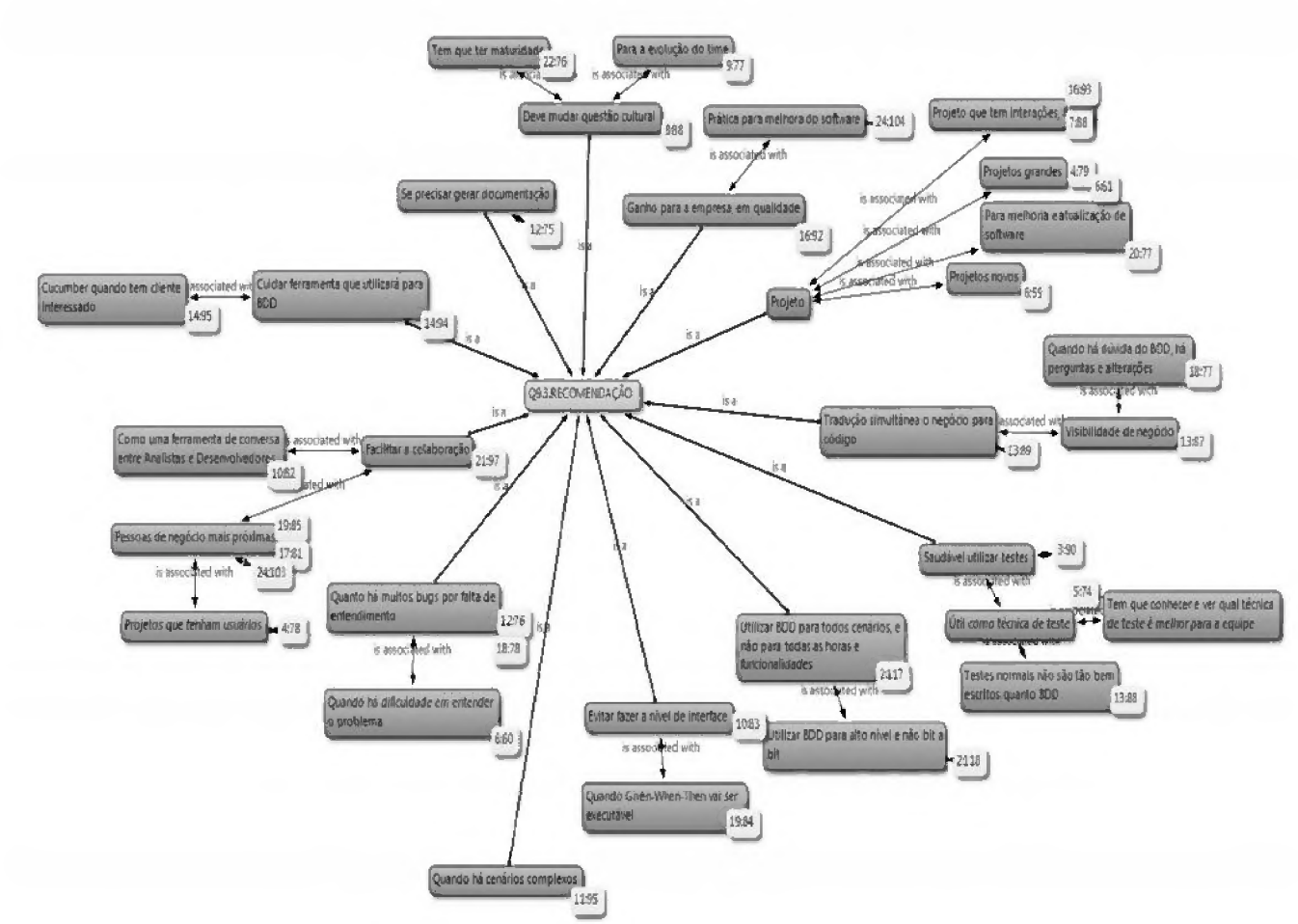

Desenvolvedores - Analista de Negócio validava sem cláusulas
6:43
is associated with
Desenvolvedores - PO validava, mas não usava cláusulas do BDD, pois eram técnicas
13:62
Líder técnico do time
5:49
Analistas de Teste
9:66
7:63
equals
22:52
20:20
QAs verificam os cenários
15:52
3:77
24:87
is a
Q8.7 FIDELIDADE
O PO do cliente
23:49
is associated with
Cenários eram validados com o cliente
10:48
Todos membros validam
4:52
17:58
is associated with
PO, QA e Dev
8:58
is associated with
QA, Dev e BA
11:60
is associated with
Time inteiro e Gerente de Produto
21:62
is associated with
Todo time
2:66
QA e posteriormente o PO e UX validam
18:50
Analista de Negócio validam cenários
10:54
14:59
19:52
is associated with
PO e BA
16:52
is associated with
PO valida
12:54
1:60

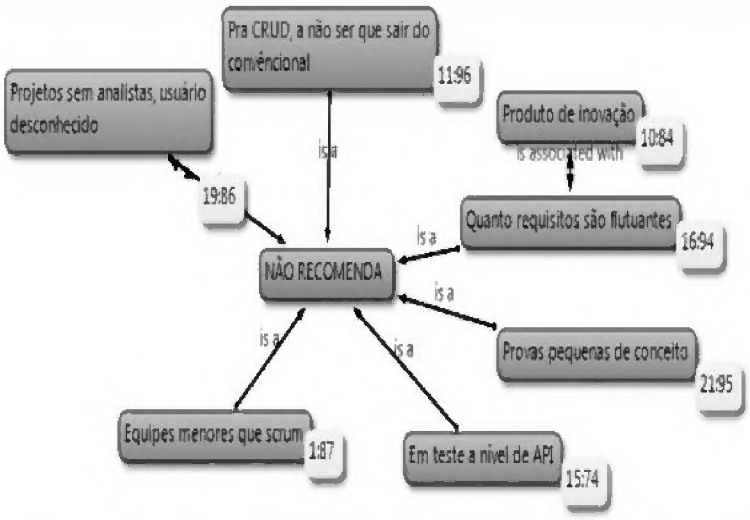

QB4EXPRESSA
Melhora entendimento
Linguagem facilitar entendimento
Entendimento do cliente
Possibilidade de entendimento entre iterações do desenvolvimento
TDD não abrange o que o BDD abrange
Não. é uma questão de estilo
Desde que haja comunicação e colaboração
Quando há alinhamento do cliente com o time
Gera documentação fácil para criar software
Leitura fácil
Custo elevado
Ao longo do projeto cliente vê valor
BDD faz ter um padrão de escrita
Tem que usar palavras no negócio
Ás vezes a escrita torna a compreensão difícil
is a
is associated with
contradicts

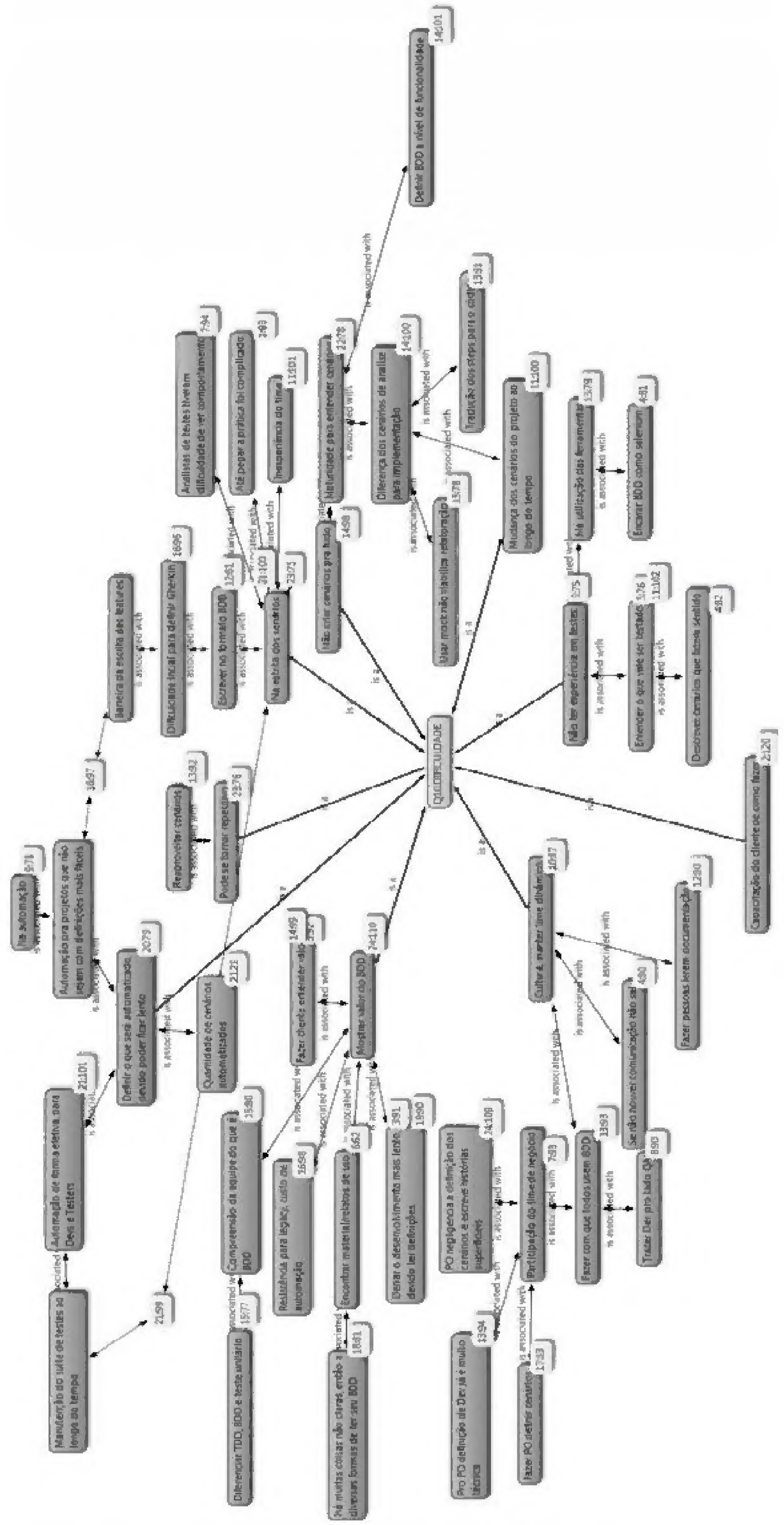

Também, mas há demais informações de complemento, como Card
8:83
Para requisitos funcionais
15:67
Usufri de fragmentos de histórias do usuário
19:76
Frontend, req funcional e não funcional
21:86
Para sistemas que se consegue ter histórias
20:73
Quando é baixo nível, torna-se complicado
22:71
Funciona para camada não visua
5:69
Requisitos
Quando se mapeia para a feature (que não seja task)
16:84
Reqs de performance complicados
21:87
Pode ter casos mais complicado, mas acredito que é possível
23:65
Q8.9.SUPORTA
Se bem escrito
4:73
Bem escrito, domais languag
11:88
Depende do que esta especificado, devs sentem falta de infos de arquitetura
12:70
Ajuda na especificação
13:84
Escrever cenários juntos já auxilia
24:97
Cliente sabia o que estava sendo especificado
14:87
Não seria prático ter cenários para definir tudo
10:71
Quando se necessita ter processo de qualidade que qualificam o produto
2:107
is a
is associated with

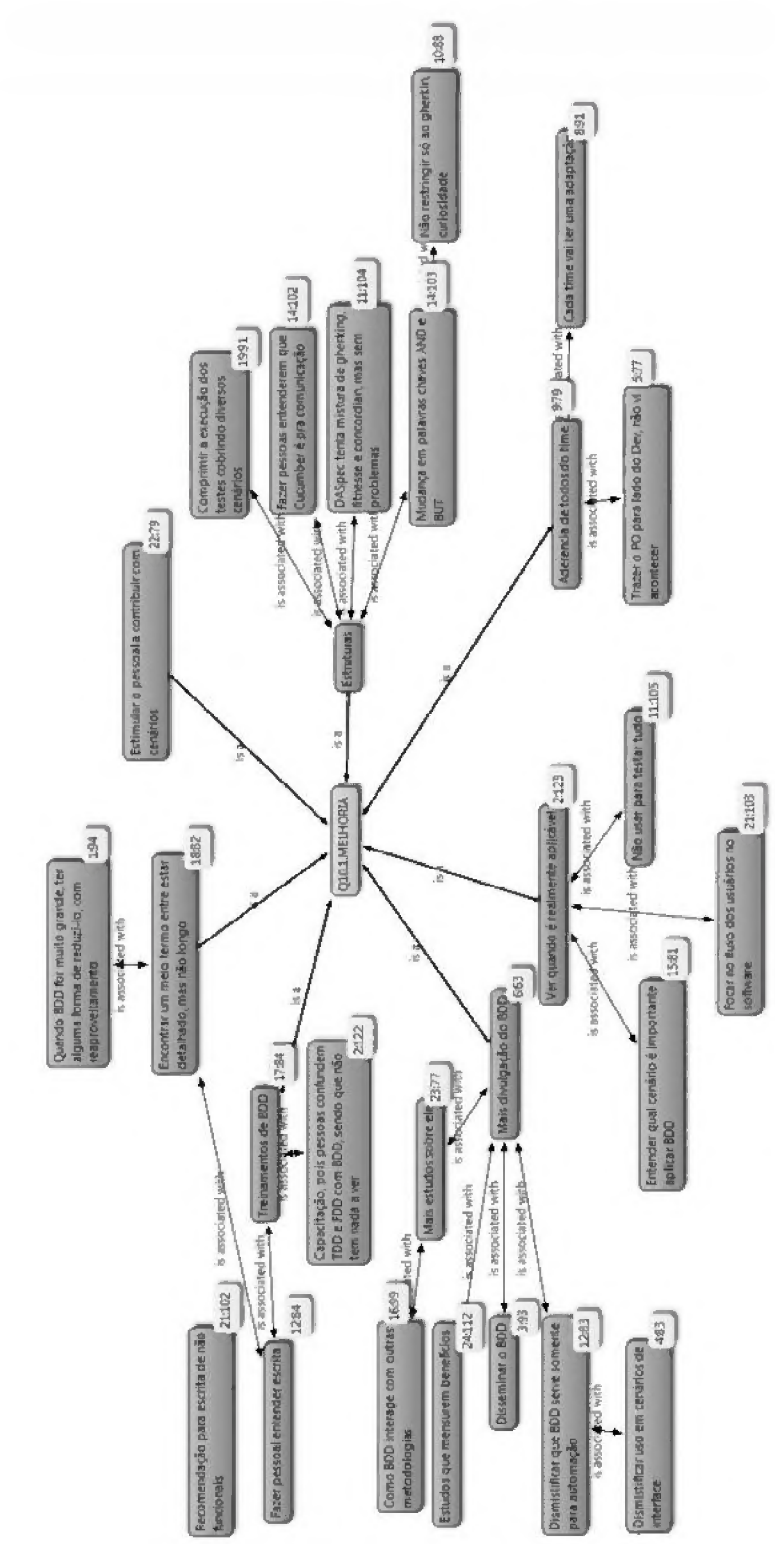

Q9:BENEFICIOS
is a
Visibilidade do negócio
14:08
Toma objetivo o que o projeto deve fazer
13:85
Maior conhecimento da aplicação
2:108
Entendim
Clareza do que deve ser feito
Capacitar mais rápido novos integrantes
2:109
Alinhamento com a expectativa da entrega
2:111
Documentação viva
Maior colaboração
9:74
21:89
Facilidade da comunicação
7:92
Comunicação com cliente
14:89
Melhora da conversa entre Devs e Analistas
10:73
Distrib: comunicar mais na hora de criar os cenários
10:75
Compartilhamentos visões do time
19:87
18:70
Rastreio dos cenários
18:69
Organiza testes de alto nível
11:90
Uso de teste em si
5:70
Diminui chance de erro
3:85
Prevenção de defeitos
7:80
Diminuiu número de bugs
14:91
Automação já trás vantagem
16:86
23:68
Em relação ao TDD é mais produtivo
15:69
Aumento de produtividade e segurança
24:99
Aumenta confiança no sistema
4:75
is associated with
equals

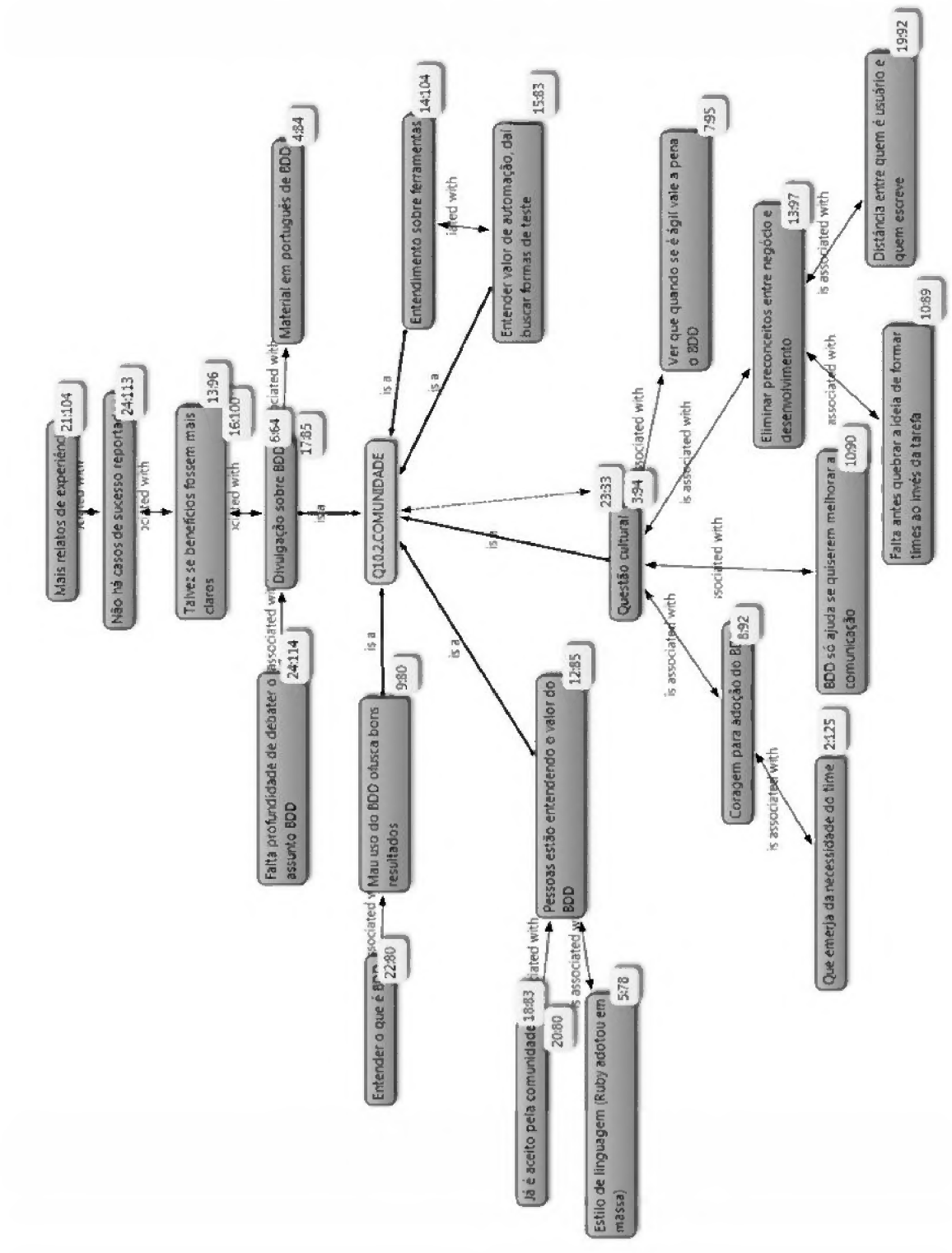

Q9.1.PRODUTO
Reduz bugs
Reduz retrabalho
Prevenção de problemas
Sem automação, antecipa problemas
Havia falha na comunicação
Melhora da cobertura
Automação garante qualidade
Testes
Qualidade dos testes funciona
Representa testes integração
Segurança que coisas funcionam
Os testes garantem qualidade
Útill pra automação
Ajuda na regressão
Entendimento de requisitos
Clareza da necessidade
Pensar no comportamento antes
is a
is associated with

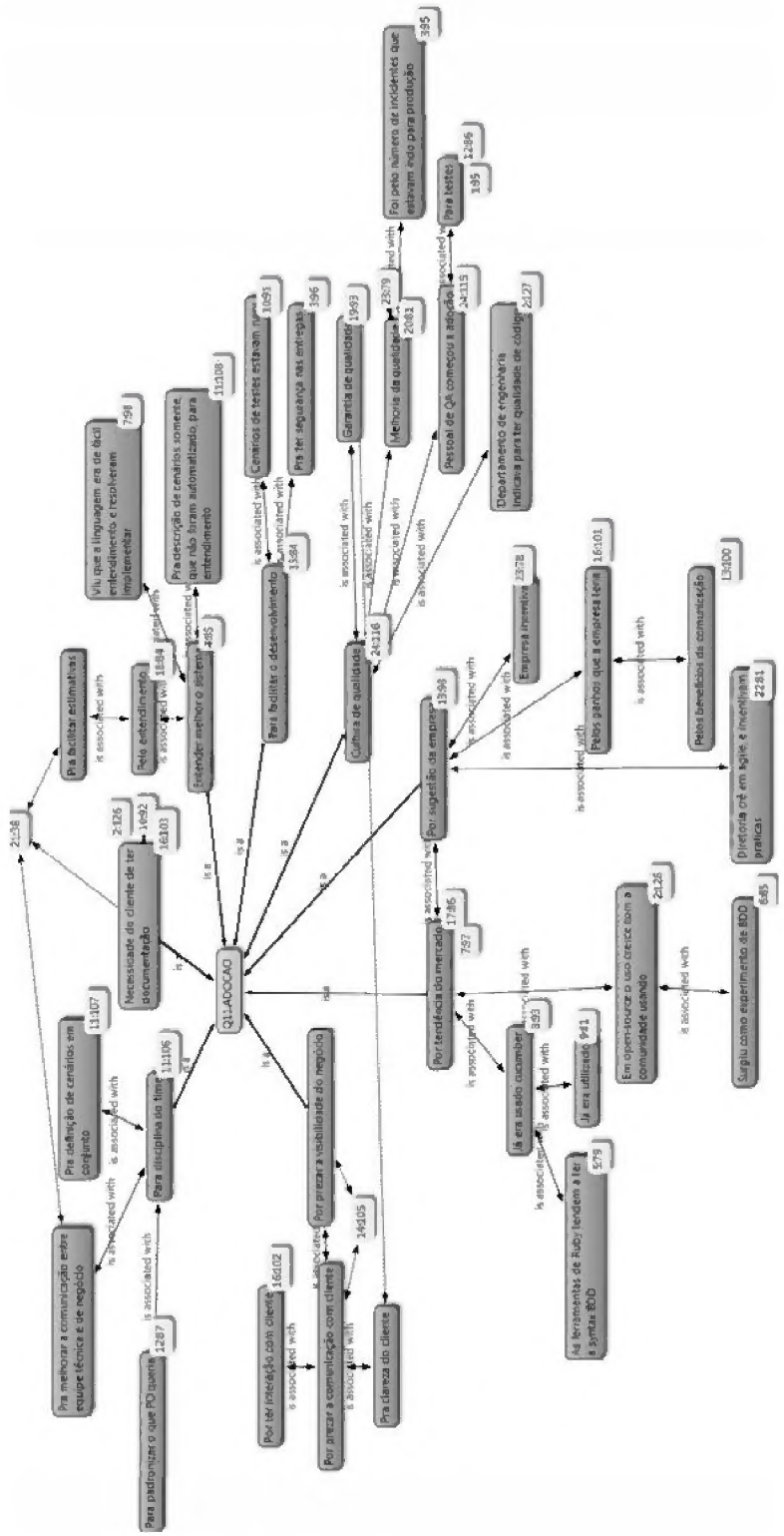

Tem que ter maturidade
22:76
Para a evolução do time
9:77
Deve mudar questão cultural
9:88
Prática para melhora do software
24:104
Projeto que tem interações
16:93
7:88
Projetos grandes
4:79
6:61
Para melhoria e atualização de software
20:77
Projetos novos
6:55
Se precisar gerar documentação
12:75
Ganho para a empresa em qualidade
16:92
Cucumber quando tem cliente interessado
14:95
Cuidar ferramenta que utilizará para BDD
14:94
Projeto
Q93.&RECOMENDAÇÃO:
Quando há dúvida do BDD, há perguntas e alterações
18:77
Visibilidade de negócio
13:87
Tradução simultânea o negócio para código
13:89
Como uma ferramenta de conversa entre Analistas e Desenvolvedores
10:62
Facilitar a colaboração
21:97
Pessoas de negócio mais próximas
19:85
17:81
24:103
Projetos que tenham usuários
4:78
Quanto há muitos bugs por falta de entendimento
12:76
18:78
Quando há dificuldade em entender o problema
6:60
Saudável utilizar testes
3:90
Útil como técnica de teste
5:74
Tem que conhecer e ver qual técnica de teste é melhor para a equipe
Testes normais não são tão bem escritos quanto BDD
13:88
Utilizar BDD para todos cenários, e não para todas as horas e funcionalidades
24:17
Utilizar BDD para alto nível e não bit a bit
24:18
Evitar fazer a nível de interface
10:83
Quando Given-When-Then vai ser executável
19:84
Quando há cenários complexos
11:95

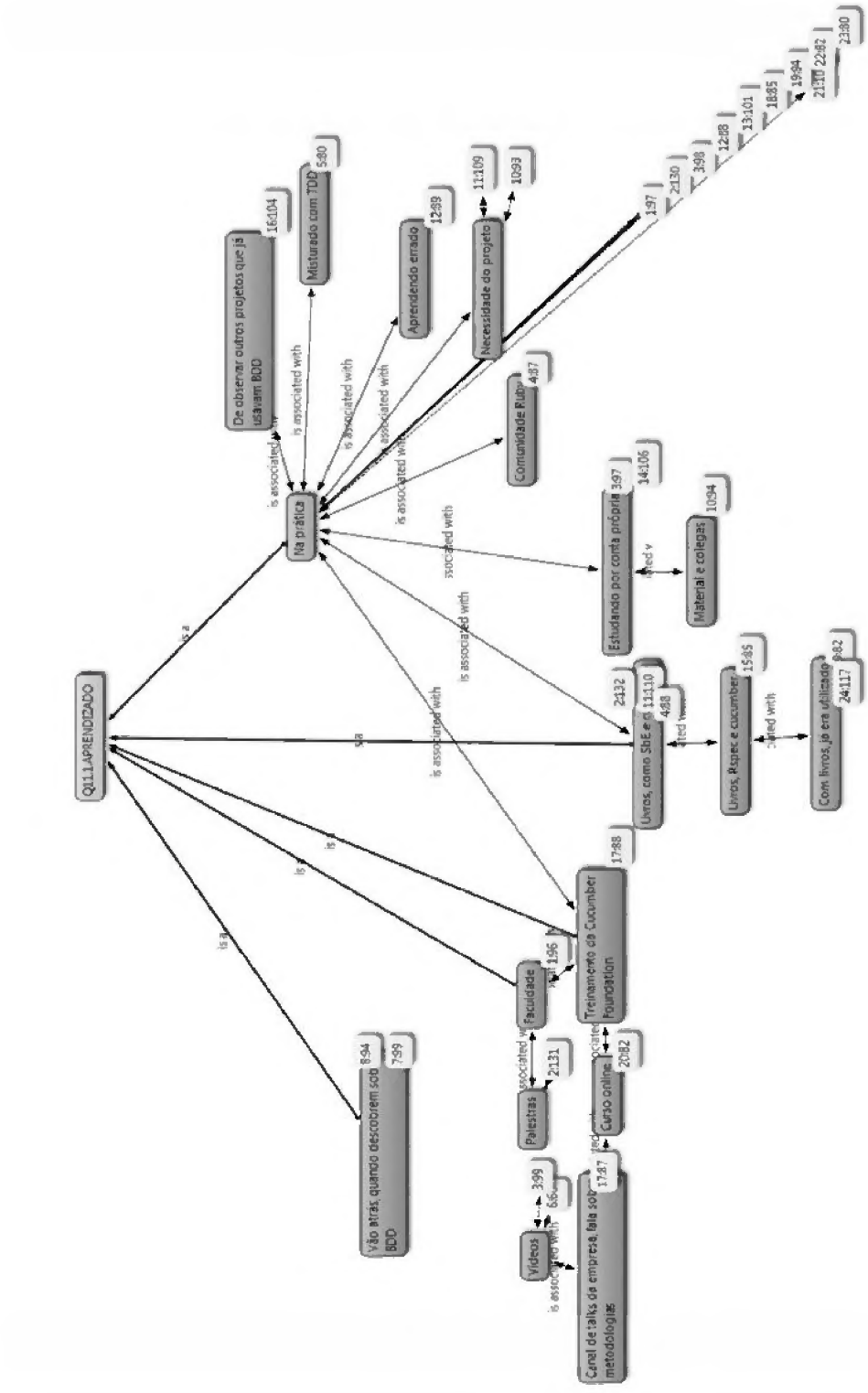

Pra CRUD, a não ser que sair do convêncional
11:96
Projetos sem analistas, usuário desconhecido
19:86
Produto de inovação
10:84
Quanto requisitos são flutuantes
16:94
NÃO RECOMENDA
Provas pequenas de conceito
21:95
Equipes menores que scrum
1:87
Em teste a nível de API
15:74


Q10DIFICULDADE
Definir BDD a nível de funcionalidade
Mostrar valor do BDD
Na escrita dos cenários
Cultura, manter time dinâmico
Mudança dos cenários do projeto ao longo do tempo
Diferença dos cenários de analise para implementação
Não ter experiência em testes
Capacitação do cliente de como fazer
Automação de forma efetiva para Devs e Testers
Manutenção do suíte de testes ao longo do tempo
Definir o que será automatizado, devido poder ficar lento
Quantidade de cenários automatizados
is associated with
is a


Q161:MELHORIA
Estimulam
Estimular a pessoalia contribuir com cenários
2279
Compreender a execução dos testes cobrindo diversos cenários
1991
Fazer pessoas entenderem que Cucumber é pra comunicação
14102
DASpec tenta mistura de gherking (teste e concordian, mas tem problemas
11104
Mudança em palavras chaves AND e BUT
14103
Não restringir só ao gherkin, curiosidade
1088
Adicencia de todos do time
979
Casa time vai ter uma adaptação
891
Trazer o PO para lado do Dev, não vi acontecer
3077
Ver quando é realmente aplicável
2123
Não usar para testar tudo
11105
Focar no fluxo dos usuários no software
21103
Entender qual cenário é importante aplicar BDD
1581
Mais divulgação do BDD
663
Mais estudos sobre ele
2377
Treinamentos de BDD
1784
Encontrar um meio termo entre estar detalhado, mas não longo
1882
Quando BDD for muito grande, ter alguma forma de reduzi-lo, com refatoretalamento
194
Capacitação, pois pessoas confundem TDD e FDD com BDD, sendo que não tem nada a ver
2122
Como BDD interage com outras metodologias
1699
Estudos que mensurem benefícios
24112
Disseminar o BDD
393
Dismistifiar que BDD serve somente para automação
1283
Dismistificar uso em cenarios de interface
483
Recomendação para escrita de não funcionais
21102
Fazer pessoal entender escrita
1284
is a
is associated with


Mais relatos de experiênd 21:104
Não há casos de sucesso reportar 24:113
Talvez se benefícios fossem mais claros 13:96
16:100
Divulgação sobre BDD 6:64
17:85
Material em português de BDD 4:84
Falta profundidade de debater o assunto BDD 24:114
Q102.COMUNIDADE
Entendimento sobre ferramentas 14:104
Entender valor de automação, dai buscar formas de teste 15:83
Mau uso do BDD ofusca bons resultados 9:80
Entender o que é BDD 22:90
Questão cultural 3:94
23:33
Ver que quando se é ágil vale a pena o BDD 7:95
Pessoas estão entendendo o valor do BDD 12:85
Coragem para adoção do BDD 8:92
Eliminar preconceitos entre negócio e desenvolvimento 13:97
BDD só ajuda se quiserem melhorar a comunicação 10:90
Falta antes quebrar a ideia de formar times ao invés da tarefa 10:89
Distância entre quem é usuário e quem escreve 19:92
Já é aceito pela comunidade 18:83
20:80
Estilo de linguagem (Ruby adotou em massa) 5:78
Que enteija da necessidade do time 2:125
is a
is associated with


QUADRANTE
Necessidade do cliente de ter documentação
Para disciplina do time
Pra definição de cenários em conjunto
Pra melhorar a comunicação entre equipe técnica e de negócio
Para padronizar o que PO queria
Por ter interação com cliente
Por prezar a comunicação com cliente
Por prezar visibilidade do negócio
Pra clareza do cliente
Pra facilitar estimativas
Pelo entendimento
Entender melhor o sistema
Viu que a linguagem era de difícil entendimento e resolveram implementar
Pra descrição de cenários somente que não foram automatizado, para entendimento
Para facilitar o desenvolvimento
Cenários de testes estavam quebrando
Pra ter segurança nas entregas
Cultura de qualidade
Garantia de qualidade
Melhoria da qualidade
Pessoal de QA começou a adoção
Foi pelo número de incidentes que estavam indo para produção
Para testes
Departamento de engenharia indicava para ter qualidade de código
Por sugestão da empresa
Empresa investiu
Pelos ganhos que a empresa teve
Pelos benefícios da comunicação
Diretoria crê em ágile e inventaram práticas
Por tendência do mercado
Já era usado cucumber
Já era utilizado
Já ferramentas de Ruby tendem a ter a syntax BDD
Em open-source o uso cresce bem a comunidade usando
Surgiu como experimento de BDD


Q111.APRENDIZADO
is a
Na prática
De obtenar outros projetos que já usavam BDD
16104
Misturado com TDD
580
Aprendendo errado
1259
Necessidade do projeto
11109
1093
Comunidade Ruby
437
is associated with
197
2130
398
1288
13101
1835
1994
2197
2282
2380
Estudando por conta própria
397
14106
Material e colegas
1094
Livros, como SbE e os
2132
13110
438
Livros, Rspec e cucumber
1585
Com livros, já era utilizado
24117
882
Treinamento do Cucumber Foundation
17188
Faculdade
196
Palestras
2131
Curso online
2082
Vídeos
399
Canal de talks da empresa, fala sobre metodologias
1787
Vão atrás quando descobrem sobre BDD
894
799

ANEXO A – Debriefing - cedido por Prof Cleidson de Souza

Debriefing Questions

Local:
Date: Duration:
Interviewer:
Interviewee:

1. What were the key points observed about the focus?

2. What did you find to be most surprising about this observation?

3. What did you see or hear that was pretty much what you expected (or like other sites that you have seen)?

4. What did you learn about the problem and "fixes" that you didn't know before? That you did?

5. What would you ask if we could go back? Would you ask the next participant this as well?

6. What worked really well?

7. What didn't work so well or what should be changed?

8. Other comments?