COPPE/UFRJ

UMA ANÁLISE DA UTILIZAÇÃO DA METODOLOGIA DA PESQUISA-AÇÃO EM ENGENHARIA DE SOFTWARE

Paulo Sérgio Medeiros dos Santos

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Ciências em Engenharia de Sistemas e Computação.

Orientador(es): Guilherme Horta Travassos

Rio de Janeiro

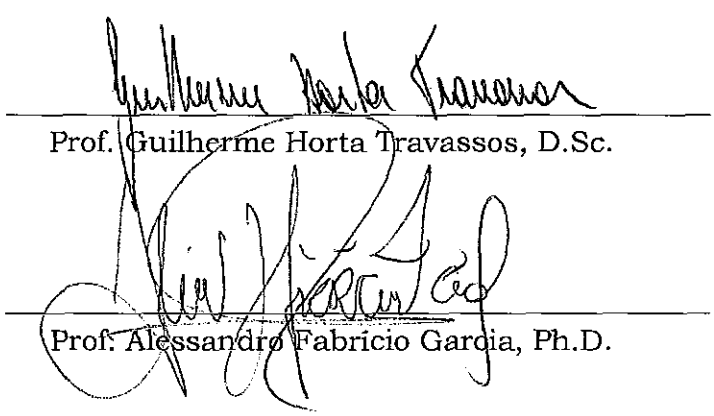
Dezembro de 2009

UMA ANÁLISE DA UTILIZAÇÃO DA METODOLOGIA DA PESQUISA-AÇÃO EM ENGENHARIA DE SOFTWARE

Paulo Sérgio Medeiros dos Santos

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:



Prof. Guilherme Horta Travassos, D.Sc.

Prof. Alessandro Fabricio Garcia, Ph.D.

Profª. Cláudia Maria Lima Werner, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

DEZEMBRO DE 2009

Santos, Paulo Sérgio Medeiros

Uma Análise da Utilização da Metodologia da Pesquisa-Ação em Engenharia de Software / Paulo Sérgio Medeiros dos Santos. – Rio de Janeiro: UFRJ/COPPE, 2009.

XI, 165 p.: il.; 29,7 cm.

Orientador: Guilherme Horta Travassos

Dissertação (mestrado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2009.

Referencias Bibliográficas: p. 132-141.

1. Pesquisa-Ação. 2. Engenharia de Software. 3. Metodologia de Pesquisa. I. Travassos, Guilherme Horta. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Titulo.

À minha família.

Agradecimentos

Ao Prof. Guilherme Travassos por me guiar desde o início da minha vida acadêmica, apresentando-me a experimentação como uma forma de pensar ciência em Engenharia de Software. Seu profundo conhecimento em experimentação, e sua confiança e apoio às minhas decisões resultaram em um tema relacionado à "meta-pesquisa" em Engenharia de Software Experimental. Ao longo destes anos, sua amizade e atenção durante os trabalhos desenvolvidos, puderam me proporcionar um amadurecimento profissional e pessoal fundamentais para a continuidade da minha carreira.

A Professora Claudia Maria Lima Werner pela contribuição ao meu aprendizado no decorrer do curso e por participar da banca examinadora da minha dissertação.

Ao Professor Alessandro Fabrício Garcia pela participação na banca examinadora da minha dissertação.

Aos amigos da COPPE/UFRJ. Arilo Claudio Dias Neto, Chessman Kennedy, Fernando Sampaio, Jobson Massolar, Leonardo Murta, Lucas Paes, Marcelo Schots, Marco Kalinowski, Marco Araujo, Priscila Pecchio, Rafael Cepeda, Rafael Mello, Rafael Santo, Rodrigo Murta, Rodrigo Santos, Rodrigo Spinola, Silvia Santa, Tayana Conte, Thiago Carvalho, Vitor Farias, Vitor Lopes, Wallace Martinho, Wladmir Chapetta, pelo agradável convívio ao longo destes anos.

Aos companheiros de trabalho onde os estudos foram conduzidos e que concederam a publicação dos resultados. A aos alunos da disciplina Engenharia de Software Orientado a Objetos (2008) pela participação no estudo e importantes sugestões para a avaliação da técnica de inspeção utilizada.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

UMA ANÁLISE DA UTILIZAÇÃO DA METODOLOGIA DA PESQUISA-AÇÃO EM ENGENHARIA DE SOFTWARE

Paulo Sérgio Medeiros dos Santos

Dezembro/2009

Orientador: Guilherme Horta Travassos
Programa: Engenharia de Sistemas e Computação

A Engenharia de Software, enquanto uma disciplina científica, ainda pode ser considerada uma área que busca alcançar maturidade na forma como estuda os efeitos da sua prática. A metodologia da pesquisa-ação, amplamente utilizada nas ciências sociais, possui características que podem permitir a condução de estudos relevantes em Engenharia de Software na medida em que permite que pesquisa e intervenções organizacionais sejam realizadas de forma simultânea. No entanto, a utilização de uma nova metodologia de pesquisa requer uma avaliação cuidadosa da forma como deve ser empregada na busca por resultados que sejam não apenas relevantes mas, principalmente, confiáveis.

Neste sentido, este trabalho apresenta uma análise de como a metodologia pode ser aplicada em Engenharia de Software, buscando inserir a pesquisa-ação como uma opção para investigações científicas na área. Para isto, dois estudos em ambientes reais utilizando a metodologia foram conduzidos nos domínios de refatoração, documentação e inspeção de software. Como resultado desta experiência, uma análise da pesquisa-ação é apresentada considerando as particularidades da Engenharia de Software como aspectos éticos, identificação de oportunidades de pesquisa, processo de condução da pesquisa e o papel do pesquisador, construção de teorias e um modelo para relato de estudos.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

AN ANALYSIS OF THE ACTION RESEARCH METHODOLOGY USE IN SOFTWARE ENGINEERING

Paulo Sérgio Medeiros dos Santos

December/2009

Advisor: Guilherme Horta Travassos

Departament: Engenharia de Sistemas e Computação

Software Engineering as a scientific discipline, can still be considered an area that seeks to reach maturity in the way that studies the effects of its practice. The action research methodology, widely used in the social sciences, has characteristics that may enable conducting relevant studies in Software Engineering since it research and organizational intervention are carried out simultaneously. However, the use of a new research methodology requires a careful assessment of how it should be used in pursuing results that are not only relevant but, above all, reliable.

Thus, this dissertation presents an analysis of how action research can be applied in Software Engineering, aiming to place the methodology as an option for scientific research in the area. For this, two studies in real environments were conducted in the areas of software refactoring, documentation and inspection. As a result of this experience, an analysis of action research is presented considering the particularities of Software Engineering such as ethical aspects, identification of opportunities for research, research process conduction and the researcher role, theory building and a model for studies reporting.

Sumário

Capítulo 1
Introdução

Neste capítulo são apresentadas as principais motivações e os objetivos que levaram à realização deste trabalho, juntamente com a forma como está organizado o texto desta dissertação.

1.1. Motivação

Um cientista, independentemente se um teorista ou experimentalista, invoca proposições, ou um sistema de proposições, e as testa passo a passo com objetivo de evoluir o conhecimento científico de sua área (Popper, 1959). Cada pesquisa é baseada firmemente em uma ou mais realizações científicas que a comunidade reconhece como a fundação para a prática de ações futuras. O desenvolvimento científico, conseqüentemente, pode ser caracterizado por um processo gradativo por meio do qual os fatos, teorias e métodos embutidos nas proposições são combinados em uma crescente reserva que constitui o corpo de conhecimento científico.

O estágio inicial de desenvolvimento de grande parte das ciências tem sido caracterizado por uma competição contínua entre um número distinto de diferentes visões sobre a natureza, cada uma parcialmente derivada de e todas aproximadamente compatíveis com o método científico (Kuhn, 1970). A Engenharia de Software, por ser uma disciplina recente se comparada à maioria das outras engenharias e áreas mais bem estabelecidas como Medicina, Física e Química, ainda busca por métodos científicos adequados a sua prática, os quais proporcionem resultados relevantes tanto para pesquisadores quanto para profissionais (Sjøberg *et al.*, 2007).

A Engenharia de Software foi definida em 1976 (Boehm, 1976) como a "aplicação prática do conhecimento científico no projeto e construção de programas de computadores e a documentação relacionada ao seu desenvolvimento, operação e manutenção". No entanto, apesar da sua concepção inicial já prever o uso do conhecimento científico, apenas em 1986, com Basili *et al.* (1986), o primeiro passo em direção ao estabelecimento de uma visão sistemática à investigação de fenômenos em Engenharia de Software foi dado. Desde este momento, o método científico em Engenharia de Software vem sendo progressivamente aperfeiçoado com trabalhos envolvendo

estudos controlados (Basili *et al.*, 1986), estudos secundários (Kitchenham *et al.*, 2004) e, recentemente, estudos de caso (Runeson e Höst, 2009).

Todos estes trabalhos adaptaram o conhecimento sobre o uso do método científico de disciplinas mais bem estabelecidas. A transposição destas metodologias para Engenharia de Software, entretanto, parece não ser direta, já que possui características específicas que a tornam uma ciência com aspectos tecnológicos, mas com forte influência dos fatores humanos. Na verdade, esta é uma das principais motivações destes trabalhos.

Neste sentido, este trabalho visa contribuir para esse cenário buscando analisar como a metodologia da pesquisa-ação[1], amplamente utilizada nas ciências sociais, pode ser utilizada em Engenharia de Software como meio de prover resultados relevantes ao corpo de conhecimento científico da área.

1.2. Metodologia de Pesquisa

Antes de descrever a metodologia de pesquisa deste trabalho é importante definir brevemente a pesquisa-ação. É possível caracterizar a pesquisa-ação por um conjunto de princípios. Primeiro, busca interligar teoria e prática alcançando objetivos tanto práticos quanto científicos (Susman e Evered, 1978); em segundo lugar, é uma metodologia de investigação crítica com foco na prática social em um contínuo processo deliberado de aprendizagem por meio da reflexão acerca das ações desempenhadas (Argyris *et al.*, 1985); e por fim, enfatiza a análise de julgamentos subjetivos gerados pelo pesquisador quando inserido em situações do dia-a-dia (Thiollent, 2007).

Neste sentido, a pesquisa-ação implica no intenso envolvimento na situação de pesquisa, com grandes oportunidades para o aprendizado – tanto prático quanto científico –, mas potencialmente em detrimento da objetividade. Conflitos podem surgir devido às dificuldades em atender as duas audiências com diferentes necessidades: os clientes e a comunidade de pesquisa. Um raciocínio reflexivo rigoroso e uma documentação apropriada do processo de pesquisa podem ser utilizados para minimizar estas dificuldades.

A metodologia de pesquisa deste trabalho pode ser traduzida por meio de uma analogia com a abordagem de aprendizagem baseada em problema (*problem-based learning*) (Albanese e Mitchell, 1993). Originada na clínica médica, a abordagem utiliza problemas comuns à pacientes como contexto

[1] Pesquisa-ação é o termo utilizado para traduzir *Action Research*.

para o aprendizado de habilidades de diagnóstico e solução de problemas, além de permitir a aquisição de conhecimento sobre ciências básicas e clínicas (Albanese e Mitchell, 1993). As suas principais vantagens estão associadas a razões cognitivas, motivacionais e funcionais (Bridges, 1992). Do ponto de vista cognitivo, a abordagem permite que o conhecimento prévio seja ativado, além do fato de que o contexto no qual a informação é aprendida assemelha-se com o contexto onde será posteriormente utilizada. Na perspectiva motivacional, a abordagem possibilita avaliar o *valor* (utilidade) da utilização bem sucedida de uma determinada técnica na resolução de um problema. Sob o prisma funcional, a abordagem proporciona a retratação de um cenário real de aplicação de uma técnica incluindo suas diferentes facetas (por exemplo, sociais, comunicação e tempo de utilização).

Traduzindo para o contexto desta pesquisa, a idéia é estudar a pesquisa-ação no contexto de Engenharia de Software através de sua aplicação em um contexto real. Com isto, espera-se que a sua real importância seja bem compreendida e que a descrição de seu uso sirva para posteriores aplicações da metodologia. Além disto, também tem-se a expectativa de que a partir desta aplicação seja possível depreender como a pesquisa-ação deve ser utilizada em Engenharia de Software.

Para isto, primeiramente examina-se em detalhes as características e o processo da metodologia da pesquisa-ação, que servirá de base para a descrição do uso da metodologia em um contexto real. A partir deste entendimento, dois estudos são descritos em detalhes onde a finalidade principal é exercitar e analisar a prática da metodologia. O primeiro estudo trata da utilização de refatoração de código como meio de documentar padrões de codificação de um projeto software e o segundo do uso de técnicas *checklist* de inspeção para melhora a compreensibilidade de modelos de casos de uso. Por fim, a partir desta experiência, elaboram-se recomendações para a condução de estudos de pesquisa-ação em Engenharia de Software, incluindo um guia para o relato deste tipo de estudo.

Ambos os estudos foram selecionados de forma oportunística pelo pesquisador, que atuava como projetista de *software* na organização, durante as suas atividades normais de projeto e desenvolvimento no contexto desta organização. Detalhes sobre o contexto de trabalho e motivações sobre cada estudo são apresentados no momento apropriado na descrição de cada um.

1.3. Objetivos

O objetivo deste trabalho é apresentar a metodologia da pesquisa-ação como uma opção viável à prática da pesquisa em Engenharia de Software, identificando os seus principais benefícios para este fim ao mesmo tempo em que se tenta examinar suas particularidades durante sua aplicação especificamente na área. A proposta base deste trabalho é, então, permitir que pesquisadores sejam capazes de avaliar situações em Engenharia de Software onde a metodologia possa ser bem aproveitada, além de prover recomendações para a sua prática na área.

Desta forma, o objetivo mais geral deste trabalho é fazer com que a pesquisa-ação seja apropriadamente entendida em Engenharia de Software. Utilizando o raciocínio de Sokal e Bricmont (1999) isto significa dizer que: (1) os conceitos e a terminologia científica relacionados à pesquisa-ação sejam usados em Engenharia de Software sabendo-se o que realmente representam na área e (2) a adaptação dos conceitos científicos sejam acompanhados minimamente de uma justificativa experimental ou conceitual. Assim, também tem-se como meta a descrição de estudos de pesquisa-ação em ambientes reais que suportem os objetivos estabelecidos.

Ainda que se objetive dar o maior aporte possível para pesquisadores que desejem utilizar a pesquisa-ação em Engenharia de Software, é importante apontar de antemão que, pela natureza "doutrinária" da metodologia, o discurso deste trabalho poderá ser compreendido, em alguns momentos, como de certa forma vago ou impreciso. Em outras palavras, na medida em que a metodologia da pesquisa-ação caracteriza-se por um conjunto de teorias e idéias gerais, este trabalho tende a manter este tipo de descrição já que não tem como objetivo tornar o processo da pesquisa-ação mais "claro" ou "detalhado" – ou seja, nosso objeto de estudo não é a metodologia em si, mas sim sua utilização em Engenharia de Software. Como conseqüência, a escolha da metodologia de pesquisa utilizada neste trabalho – *problem-based learning* – reflete esta questão onde a idéia é, já que a descrição da pesquisa-ação é de maneira geral "pouco precisa", interpretar esta descrição no contexto da Engenharia de Software aplicando-a em dois estudos reais. E, desta forma, estaremos indiretamente tornando o processo da pesquisa-ação mais "claro" e "detalhado" (no contexto da Engenharia de Software) a partir do momento em

que sua aplicação é descrita e discutida considerando a pesquisa em atividades de projetos reais de desenvolvimento de *software*.

Neste sentido, este trabalho busca mostrar, por meio das idéias e teorias da metodologia da pesquisa-ação, que a Engenharia de Software pode ser estudada cientificamente através de uma abordagem que não necessariamente preconize a separação entre pesquisador e participantes ou pesquisador e objeto de estudo, como ocorre em estudos controlados comumente utilizados em Engenharia de Software, e que seja capaz de produzir um novo conhecimento científico a partir de observações e análise de dados (qualitativos e quantitativos) da prática do dia-a-dia de Engenharia de Software. Desta forma, pretende-se mostrar também que apesar da pesquisa-ação parecer ser, em um primeiro momento devido a sua descrição, flexível e genérica, a sua prática a revela como uma metodologia fundamentada nos princípios científicos do conhecimento extraído da observação do mundo, por meio da análise cuidadosa e rigorosa dos fatos que levam de um problema até a sua solução.

1.4. Organização

Além desta introdução, esta dissertação é composta por mais cinco capítulos e mais dois apêndices.

No Capítulo 2, *Pesquisa em Engenharia de Software e a Metodologia da Pesquisa-Ação*, os principais conceitos do método científico são introduzidos, com ênfase particular àqueles relacionados à Engenharia de Software, em meio aos quais a pesquisa-ação é apresentada em detalhes.

No Capítulo 3, *Estudo de Pesquisa-Ação #1*, o primeiro estudo utilizando a metodologia é descrito. O estudo investiga o uso de técnicas de refatoração de código como meio para documentar padrões de codificação em um projeto de software. Uma teoria foi elaborada no contexto deste estudo, com intenso uso da abordagem *grounded theory*.

No Capítulo 4, *Estudo de Pesquisa-Ação #2*, o segundo estudo é detalhado. Neste estudo, inspeções de software são empregadas com objetivo da melhoria da compreensibilidade de modelos de casos de uso. Uma técnica *checklist* é utilizada e comparada a abordagem *ad-hoc*.

No Capítulo 5, *Utilizando a Pesquisa-Ação em Engenharia de Software*, a experiência na utilização da metodologia é analisada, recomendações para a

prática da metodologia são discutidas e um *template* para relato de estudos é apresentado.

O Capítulo 6, *Considerações Finais*, conclui a dissertação apontando suas contribuições e trabalhos futuros.

O Apêndice A, *Transcrição das Entrevistas*, contém, além da transcrição das entrevistas, o detalhamento de como a abordagem *grounded theory* foi aplicada a estes dados.

O Apêncice B, *Formulários e Questionários de Pesquisa*, contém os instrumentos utilizados durante os estudos conduzidos.

Capítulo 2

Pesquisa em Engenharia de Software e a Metodologia Pesquisa-Ação

Neste capítulo, a metodologia da pesquisa-ação é apresentada, considerando o contexto atual da pesquisa em Engenharia de Software.

2.1. Introdução

A Engenharia de Software tem como propósito fornecer meios para melhorar o desenvolvimento de software e sistemas. A prática da Engenharia de Software envolve a coordenação e integração de diferentes tarefas (análise, projeto, codificação, testes, garantia da qualidade, gerenciamento de projeto, dentre outras) as quais são intensamente dependentes da habilidade humana, muitas vezes no contexto do desenvolvimento de produtos inovadores. Dada a crescente complexidade dos projetos de Engenharia de Software, onde o software passa a fazer parte intrinseca da sociedade moderna, novas tecnologias e processos são propostos a todo o momento para apoiar as atividades dos diferentes estágios do processo de desenvolvimento e manutenção do software.

A experimentação em Engenharia de Software é fundamental neste contexto, pois é um método de investigação que fornece a base para que as propostas de novas tecnologias tenham os seus limites e aplicabilidade conhecidos, tornando a prática da Engenharia de Software previsível e economicamente viável. Por este motivo, a importância da experimentação vem crescendo significativamente ao longo dos últimos anos (Sjøberg *et al.*, 2005, Basili e Elbaum, 2006, Höfer e Tichy, 2007, Kampenes *et al.*, 2009). Ainda assim, a experimentação parece não ter alcançado a mesma importância como em outras disciplinas, por exemplo, a Medicina (Charters *et al.*, 2009). Um exemplo bastante representativo desta situação é que até hoje não existem evidências sólidas comparando tecnologias básicas à prática de Engenharia de Software como é o caso dos paradigmas de programação funcional e orientado à objetos, presentes em quase todos os projetos de software na indústria. Ambos certamente possuem vantagens e desvantagens. Na tentativa de mostrá-las à comunidade de Engenharia de Software, demonstrações ou heurísticas são elaboradas para seleção e utilização destas tecnologias. No entanto, raramente demonstrações produzem evidência sólida (Tichy, 1998).

Diferentes fatores podem ser atribuídos a este cenário. Inicialmente, a complexidade e o custo da experimentação eram, ou pareciam ser, os principais obstáculos a sua adoção (Juristo e Moreno, 2001). Principalmente, se considerarmos que o método experimental, quando introduzido por Basili *et al* (1986) em Engenharia de Software, teve forte influência de áreas como a Física e Medicina, onde existe o foco na condução de estudos controlados. No entanto, este tipo de estudo é difícil de ser controlado em projetos reais de Engenharia de Software devido ao grande número de variáveis não conhecidas (Basili, 1996) e, por isso, pode representar um elevado grau de risco a estes projetos. Além disto, estudos experimentais podem consumir muito tempo e gerar um grande volume de informação e conhecimento de difícil gerenciamento (Shull *et al.*, 2001). Não obstante, dada a importância das questões a serem investigadas em Engenharia de Software, esforços e recursos devem ser investidos para superar estes obstáculos (Tichy, 1998, Sjøberg *et al.*, 2007), assim como outras disciplinas o fizeram.

Em um segundo momento, já com uma maior consolidação da experimentação em Engenharia de Software, inclusive com o surgimento de livros dedicados exclusivamente ao assunto (Wohlin *et al.*, 2000, Juristo e Moreno, 2001), houve uma maior preocupação com a introdução de novos métodos e técnicas de pesquisa como estudos de caso, etnografia, pesquisas de opinião e técnicas de entrevista (Harrison *et al.*, 1999, Wohlin *et al.*, 2003, Zelkowitz, 2007, Easterbrook *et al.*, 2008). A busca por novos métodos de pesquisa tem como componente o fato de que apesar de estudos controlados e medição nos permitirem observar relacionamentos entre variáveis por meio de testes estatísticos, suas limitações podem restringir aquilo que podemos enxergar e, portanto, investigar (Pfleeger, 1999). Esta visão de Pfleeger (1999) parece ter sido coerente, já que anos mais tarde Kitchenham (2007) afirmou que a excessiva ênfase na utilização de estudos controlados em Engenharia de Software tem feito com que "nos encontremos investigando fenômenos que são resultados da abstração da tecnologia do seu contexto de utilização e não das características da tecnologia em si". Em outras palavras, a insistência prematura na precisão pode inibir o progresso levando cientistas a formular problemas de maneira que possam ser mensurados, mas têm limitada relevância em relação às características do problema (Argyris *et al.*, 1985). Por esta razão, corremos o risco de responsabilizar os profissionais da indústria por falharem no uso dos métodos propostos quando o real problema é a nossa

falha em entender a complexidade do contexto no qual as técnicas serão utilizadas (Kitchenham, 2007).

A utilização de métodos experimentais alternativos devido à necessidade de melhor observar e entender o contexto da prática da Engenharia de Software nos remete a um elemento central da pesquisa em Engenharia de Software: a *relevância* dos resultados científicos. A relevância está relacionada à utilidade dos resultados no contexto industrial (por exemplo, provendo resultados que permitam o desenvolvimento de guias) assim como no contexto acadêmico (por exemplo, permitindo um melhor entendimento de um fenômeno por meio da construção de teorias) (Sjøberg *et al.*, 2007). Vários outros pesquisadores já enfatizaram esta questão como Kitchenham *et al.* (2004) e Glass (2009). A própria indústria nos mostra a importância de aumentarmos a relevância dos estudos em Engenharia de Software. Trabalhos recentes (Rainer *et al.*, 2005, Kitchenham *et al.*, 2007) apontam que profissionais ignoram evidência científica em lugar à opinião de especialistas em grande parte das suas tomadas de decisões. Por este motivo, ainda é comum novas tecnologias serem adotadas sem nenhuma base ou critério científico.

Neste contexto, a metodologia da Pesquisa-Ação surge como uma alternativa para intensificar a realização de estudos relevantes (Baskerville e Wood-Harper, 1996) ao mesmo tempo em que permite a investigação da prática de Engenharia de Software em profundidade. Como será visto neste capítulo a pesquisa-ação é pouco explorada em Engenharia de Software, mas suas características sugerem que sua aplicação pode trazer benefícios à pesquisa em Engenharia de Software. Isto porque realiza de maneira simultânea pesquisa e ação. A ação está normalmente associada a alguma transformação em uma comunidade, organização ou programa, enquanto que a pesquisa é caracterizada por um maior entendimento do fenômeno transformador por parte do pesquisador (comunidade de pesquisa) ou interessado (cliente), ou ambos. Para Sjøberg et al. (2007), a pesquisa-ação representa "o tipo de estudo onde o cenário de pesquisa mais realístico é encontrado", uma vez que envolve um contexto real da indústria para investigar resultados de ações concretas. Além disto, para Avison *et al.* (1999) a pesquisa-ação enfatiza "mais o que os profissionais fazem do que naquilo

que dizem que fazem", isto é, a ação transformadora dentro da qual os profissionais estão imersos.

Este capítulo possui, além desta introdução, mais quatro seções. A seguir os principais paradigmas de pesquisa são apresentados de forma a posicionar a metodologia da pesquisa-ação frente a outras metodologias. Após, a metodologia é apresentada em detalhes. Em seguida, uma revisão da literatura é descrita a qual mostra o estado atual da utilização da pesquisa-ação em Engenharia de Software. Por fim, conclusões e direcionamentos para os próximos capítulos são apresentados.

2.2. Pesquisa em Engenharia de Software: Conceitos e Métodos

A Engenharia de Software é uma disciplina multidisciplinar, que envolve diferentes aspectos sociais e tecnológicos. Para entender como engenheiros de software mantém sistemas de software complexos, é necessário investigar não apenas ferramentas e processos que eles usam, mas também os processos cognitivos e sociais que os envolve. Isto requer o estudo das atividades humanas. Precisamos entender como engenheiros de software desenvolvem software individualmente, assim como equipes e organizações coordenam os seus esforços (Easterbrook *et al.*, 2008). Diferentes métodos de pesquisa podem ser utilizados para explorar estes diferentes aspectos da prática da Engenharia de Software. Para entender os objetivos, limitações e benefícios dos métodos de pesquisa, os principais critérios que os diferenciam serão abordados nesta seção. Esta seção não descreve estes métodos detalhadamente.

A distinção entre métodos de pesquisa pode ser feita utilizando o conceito de paradigma. Um paradigma pode ser visto como um conjunto de *verdades básicas* (que não se pode determinar sua veracidade) que lidam com princípios primários (axiomas e doutrinas). Representa uma *visão de mundo* que define, para aquele que vê, a natureza do "mundo", o lugar de cada indivíduo e o conjunto dos possíveis relacionamentos com este mundo e suas partes (Guba e Lincoln, 1994). Para Hathaway (1995), paradigmas científicos, na essência, atuam como lentes por meio das quais pesquisadores são capazes de observar e entender os problemas da sua área e as contribuições científicas para estes problemas. Paradigmas ditam o que pesquisadores consideram como dados, qual o seu papel em uma investigação científica, o que considerar como conhecimento e como a realidade é vista e acessada. Em resumo, trata-

se do conjunto de suposições do dia-a-dia do cientista sobre conhecimento, realidade e metodologias de investigação.

Paradigmas normalmente são caracterizados através dos conceitos de ontologia, epistemologia e metodologia (Guba e Lincoln, 1994). Epistemologia está relacionada com a natureza do conhecimento humano e como ele é entendido e comunicado, ontologia lida com as questões básicas da natureza do mundo independentemente de quem o tente observar e metodologia trata em como (por meio de quais métodos) obtemos conhecimento sobre o mundo.

A seguir os quatro paradigmas científicos predominantes na literatura são apresentados resumidamente (Guba e Lincoln, 1994, Healy e Perry, 2000, Easterbrook *et al.*, 2008):

- **Positivismo** afirma que todo o conhecimento deve ser baseado em inferência lógica a partir de um conjunto de fatos observáveis. Positivistas são reducionistas na medida em que estudam os acontecimentos quebrando-os em componentes mais simples. Isto corresponde a sua convicção de que o conhecimento científico é construído incrementalmente a partir de observações verificáveis e inferências baseadas nelas. Em outras palavras, os dados e sua análise são neutros e os dados não mudam devido ao fato de que estão sendo observados. No entanto, uma visão positivista é normalmente inapropriada para abordar fenômenos sociais quando envolvem humanos e suas experiências, já que ignora a sua habilidade em refletir sobre os problemas e agir sobre eles de maneira interdependente. O paradigma positivista está intimamente ligado à estudos controlados, mas pesquisas de opinião e estudos de caso também podem ser conduzidos sob esta perspectiva.
- **Construtivismo**, também conhecido como interpretativismo, rejeita a idéia de que o conhecimento científico pode ser separado do seu contexto humano. Construções não são mais ou menos "verdadeiras" de maneira absoluta, e sim simplesmente mais ou menos bem formadas e/ou sofisticadas, melhor informando sobre a complexidade do objeto estudado. Construtivistas concentram-se menos na verificação de teorias e mais no entendimento de como diferentes pessoas compreendem o mundo e como atribuem significado as suas ações. Teorias podem surgir neste processo, mas sempre ligadas ao contexto estudado, dando origem à chamada teoria local. O construtivismo está normalmente associado à etnografia, no

entanto muitas vezes pode estar associado a estudos de caso exploratórios, pesquisa-ação (Ludema *et al.*, 2001 apud Reason e Bradbury, 2001) e pesquisas de opinião desde que respeitem os princípios construtivistas.

- **Teoria crítica** julga o conhecimento científico a partir da sua habilidade de liberar pessoas de correntes de pensamento. Teóricos críticos argumentam que pesquisa é um ato político, já que o conhecimento dá poderes a diferentes grupos da sociedade ou fortalece estruturas de poderes existentes. Neste sentido, escolhem o que pesquisar baseado em quem irá ajudar e preferem abordagens participativas ou papéis de apoio. Desta forma, assume-se que o investigador e o objeto investigado estão interativamente ligados, com os valores do investigador inevitavelmente influenciando a investigação. Em Engenharia de Software esta filosofia pode ser associada, por exemplo, a comunidade de melhoria de processo e de métodos ágeis. Teóricos críticos utilizam muitas vezes estudos de caso para chamar atenção de algo que precisa ser modificado. No entanto, é a pesquisa-ação que melhor reflete este paradigma (Carr e Kemis, 1986 apud Reason e Bradbury, 2001).
- **Pragmatismo** reconhece que todo conhecimento é incompleto e aproximado, e seu valor depende do método utilizado para obtê-lo. Para pragmáticos, o conhecimento é julgado pelo quão útil é para solucionar problemas práticos. Desta forma, existe um certo grau de relativismo: o que é útil para uma pessoa pode não ser para outra e, assim, a verdade é relativa ao observador. Para superar críticas em relação a este aspecto, pragmáticos ressaltam a importância do consenso como "guardião" externo da objetividade como, por exemplo, no caso da tradição/prática (os resultados são coerentes com o conhecimento atual?) e a comunidade (editores, julgadores e profissionais). Em resumo, o pragmático adota a abordagem de engenharia para pesquisa, valorizando o conhecimento prático em vez do abstrato utilizando qualquer método apropriado para obtê-lo. Um método bastante utilizado, especialmente quando não há solução conhecida *a priori*, é a pesquisa-ação, pois há combinação de diferentes técnicas de pesquisa e coleção de dados (Greenwood e Levin, 1998 apud Reason e Bradbury, 2001).

Os paradigmas apresentados podem ser divididos em duas principais abordagens para obtenção e construção do conhecimento: quantitativa e

qualitativa. O positivismo está relacionado com a abordagem quantitativa enquanto que os outros paradigmas com a qualitativa (Healy e Perry, 2000). No entanto, muitas vezes, um paradigma faz uso das duas abordagens, mas sempre mantendo o foco em uma das duas de acordo com sua característica principal.

A abordagem quantitativa tem como objetivo medir e analisar as relações causais entre variáveis que representam indicadores do objeto observado. Neste sentido, a abordagem quantitativa opera com dados na forma numérica, coletados a partir de uma amostra representativa e normalmente analisados por meio de métodos estatísticos. O principal objetivo é identificar as variáveis dependentes e independentes reduzindo, desta forma, a complexidade do problema de maneira que as hipóteses iniciais possam ser confirmadas ou refutadas (Wohlin *et al.*, 2000). As principais desvantagens desta abordagem estão associadas ao fato de que seu caráter objetivo e reducionista limita um entendimento detalhado (descritivo) das propriedades e características do mundo real, na medida em que a observação de uma parte da realidade não pode ser realizada sem perder (parte da) a importância do fenômeno como um todo (Krauss, 2005).

A abordagem qualitativa é mais bem descrita avaliando como se difere da quantitativa (Näslund, 2002): (1) se aproxima da perspectiva do investigado por meio de entrevistas e observação; (2) tende a valorizar descrições detalhadas enquanto que a abordagem quantitativa está menos preocupada com este tipo de detalhamento; (3) dados qualitativos são representados por texto e imagens e, não, números. Para Seaman (1999), a principal vantagem do uso da abordagem qualitativa é que ela exige que o pesquisador imerja na complexidade do problema ao invés de abstraí-la. Desta forma, a abordagem qualitativa é também utilizada para responder "o porquê" para questões já tratadas por métodos quantitativos. No entanto, tem como desvantagem o fato dos resultados gerados serem considerados mais "vagos" e "fluidos", especialmente em uma comunidade técnica como Engenharia de Software (Seaman, 1999). Além disto, isto torna os resultados mais difíceis de serem agregados ou simplificados.

Para entender como as abordagens qualitativa e quantitativa se relacionam com os paradigmas apresentados, o gráfico da Figura 1 tenta ilustrar com qual intensidade os quatro paradigmas fazem uso das duas

abordagens. É interessante perceber, considerando todos os paradigmas em que a pesquisa-ação aparece como uma opção, o grande espectro de utilização da pesquisa-ação indo desde uma posição mais observadora do pesquisador no construtivismo, passando por uma posição de facilitador com a teoria crítica e chegando a uma postura mais atuante de resolução de problemas no caso do pragmatismo (Burns, 2005). Por este motivo, nem sempre é possível classificar cada estudo de pesquisa-ação em um único paradigma, ainda que o pesquisador adote mais fortemente uma das posições. Mas o que se tem como mais bem definido na pesquisa-ação é uma posição não-positivista, ou seja, o não distanciamento entre o observador e o objeto observado.

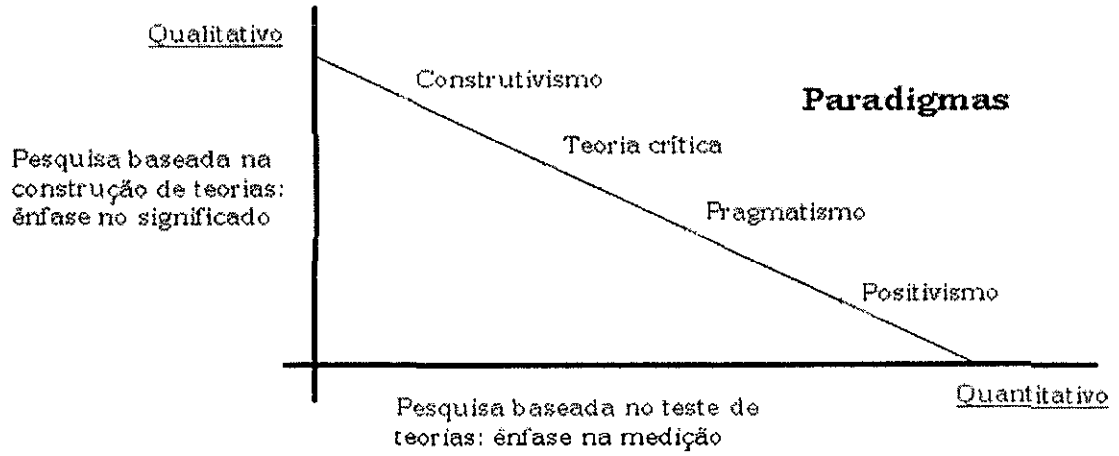


Figura 1 - Paradigmas científicos e as abordagens qualitativa e quantitativa – adaptado de (Healy e Perry, 2000)

2.2.1. Uma Outra Classificação de Metodologias de Pesquisa em Engenharia de Software

Os paradigmas de pesquisa apresentados são amplamente utilizados na literatura em geral, incluindo a Engenharia de Software. No entanto, uma taxonomia de estudos experimentais comumente utilizada pelo Grupo de Engenharia de Software Experimental da COPPE e pela comunidade de Engenharia de Software Experimental em geral, classifica os estudos em *in vivo* e *in vitro* (Basili, 1996). Posteriormente, esta classificação foi estendida por Travassos e Barros (2003) em mais dois tipos – *in virtuo* e *in silico* – que consideram a possibilidade do uso de modelos computacionais para simular o ambiente, o objeto sob análise e os participantes do estudo experimental. Os estudos *in vivo* e *in vitro*, neste sentido, classificam estudos "reais" ou não simulados.

Uma breve descrição da classificação dos estudos não simulados é apresentada a seguir (Travassos e Barros, 2003):

- **Estudos in vivo:** são aqueles que envolvem as pessoas em seus próprios ambientes. Tais estudos experimentais ocorrem durante o processo de desenvolvimento em circunstâncias reais. Apesar da necessidade de resultados experimentais, é difícil para muitas organizações de desenvolvimento de software adquirir tais resultados em ambientes de produção (*in vivo*) pelo fato de os estudos experimentais apresentarem riscos e custos elevados para sua realização;
- **Estudos in vitro:** são aqueles executados e controlados em ambientes tais como laboratórios ou comunidades controladas. Nestes estudos, pesquisadores criam situações próximas às reais em ambientes preparados ("laboratórios") para observar como os participantes executam uma determinada tarefa. Neste contexto, um ambiente preparado é aquele em que algumas variáveis podem ser controladas ou, pelo menos, observadas. Em Engenharia de Software, a maioria dos estudos in-vitro é realizada em universidades ou entre grupos selecionados de uma organização de desenvolvimento de software.

A pesquisa-ação situa-se dentre os estudos *in vivo*, ou seja, os estudos que ocorrem situações reais. Dentre outras metodologias de pesquisa usualmente utilizadas em Engenharia de Software para estudos *in vivo*, pode-se citar a Pesquisa de Opinião (*Survey*) que é aplicada para investigações realizadas em retrospecto quando, por exemplo, uma tecnologia é utilizada durante um período de tempo; Estudo de Caso utilizado para monitorar projetos ou atividades onde dados (quantitativos) são coletados para um objeto definido; e Estudo Experimental o qual é realizado com objetivo de manipular uma ou mais variáveis e controlar as demais a níveis fixos ou determinados.

Todas estas metodologias podem ser classificadas dentro do paradigma positivista. Ainda que estudos de caso, por exemplo, possam ser conduzidos com viés construtivista, estas metodologias mantém o distanciamento entre o pesquisador e o objeto observado, além de muitas vezes exigirem uma arranjo dos participantes em grupos distintos utilizando tecnologias diferentes para que seja possível uma análise estatística dos resultados. Este tipo de arranjo pode representar um elevado risco as organizações que não podem assumir o custo de uma mesma equipe sendo treinada e utilizando tecnologias diferentes que possivelmente não tragam os benefícios esperados – na verdade, normalmente, espera-se que uma delas tenha desempenho pior que a outra.

Neste sentido, devido a suas características, a pesquisa-ação apresenta-se como uma opção interessante para a Engenharia de Software para estudos *in vivo*, pois tende a trazer resultados relevantes à prática já que é conduzida em situações reais, ao mesmo tempo em que minimiza as limitações dos estudos citados acima, principalmente, os associados ao paradigma positivista. As características da pesquisa-ação são mais bem exploradas na seção seguinte.

2.3. A Metodologia da Pesquisa-Ação

A partir do entendimento do posicionamento da metodologia da pesquisa-ação no contexto dos paradigmas de pesquisa científica, o processo da pesquisa-ação é apresentado em detalhes o qual será utilizado em estudos descritos em capítulos posteriores. Antes, porém, um breve histórico da origem da metodologia é apresentado. E ao final algumas características importantes da pesquisa-ação são discutidas.

2.3.1. Histórico

A pesquisa-ação tem a sua origem associada às primeiras práticas intervencionistas realizadas por Kurt Lewin na década de 1940 em experimentos sociotécnicos. O estímulo inicial para o surgimento e concepção dos principais objetivos e aspirações da pesquisa-ação nasceu de uma dificuldade generalizada, na época, em se traduzir resultados da pesquisa social em ações práticas (Carr, 2006).

Normalmente, a história da pesquisa-ação é dividida em, no mínimo, duas fases (Reason e Bradbury, 2001). A primeira associada às práticas de Kurt Lewin, e a segunda associada ao ressurgimento do interesse na pesquisa-ação no contexto da pesquisa educacional no início da década de 1970, após sua rejeição inicial devido à predominância do paradigma positivista nas ciências sociais. Algumas razões para esta renovação do interesse na pesquisa-ação incluem, por exemplo, a reivindicação de que a profissionalização dos professores deveria ser melhorada atribuindo-lhes também um papel de pesquisador o qual os possibilitaria avaliar diretrizes curriculares na sala de aula e melhorar a prática pedagógica (Carr, 2006). Desta forma, visto desta maneira, a pesquisa-ação era transformada de um método através do qual profissionais aplicavam teorias científicas sociais na prática para um método que permitia aos profissionais avaliarem a adequação das suas próprias teorias tácitas durante a prática. Atualmente, a pesquisa-

ação é utilizada em diversas outras áreas como negócios, enfermagem e sistemas de informação (Dick, 2004).

Esta breve descrição da história da pesquisa-ação mostra como a metodologia passou (e vem passando) por mudanças, a partir de novas definições, interpretações e usos atribuídos pelos pesquisadores de diferentes áreas (Burns, 2005). Um exemplo nítido desta evolução é representado por Baskerville e Wood-Harper (1998) onde é identificada uma grande diversidade de "formatos" de pesquisa-ação – total de dez diferentes formatos – utilizados na pesquisa em sistemas de informação[2]. Por esta razão, na tentativa de ser o mais abrangente possível, o processo da pesquisa-ação apresentado a seguir é o definido por Davison *et al.* (2004) o qual é baseado no modelo de Susman e Evered (1978) que, segundo os autores, "tem sido amplamente adotado nas ciências sociais e por isso tem ganho a condição de processo 'canônico' ".

2.3.2. Processo

A pesquisa-ação pode ser definida como "um tipo de pesquisa social com base experimental que é concebida e realizada em estreita associação com uma ação ou com a resolução de um problema coletivo e no qual os pesquisadores e os participantes representativos da situação ou do problema estão envolvidos de modo cooperativo" (Thiollent, 2007). Em termos de processo, a essência por trás desta definição pode ser pensada, de maneira simplificada, em duas fases (Figura 2). A primeira é a fase de diagnóstico que envolve uma análise colaborativa da situação pelos pesquisadores e participantes da pesquisa. A partir disto, teorias em relação ao domínio de pesquisa são formuladas – neste caso, referimo-nos a teorias como sinônimo para conjecturas ou conhecimento especulativo e prático. Em uma segunda etapa, a fase terapêutica envolve experimentos (tentativas) colaborativos. Nesta fase, mudanças são introduzidas e seus efeitos estudados. Itera-se entre as duas fases até que o problema seja resolvido.

[2] É importante destacar neste momento que estamos diferenciando Engenharia de Software de Sistemas de Informação. Engenharia de Software aplica fundamentos da ciência da computação no desenvolvimento de sistemas de software. Sistemas de Informação, por sua vez, preocupa-se com as necessidades da comunidade de negócios em termos de computação, e especialmente informação. Desta forma, tanto Engenharia de Software quanto Sistemas de Informação possuem alguns elementos em comum – conceitos de computação, desenvolvimento de sistemas, e tecnologia de informação – mas têm claramente objetivos distintos (Ramesh *et al.*, 2004). Assim, ainda que a pesquisa-ação seja amplamente utilizada em Sistemas de Informação, entendemos que sua utilização em Engenharia de Software deve ser particularizada para este contexto.

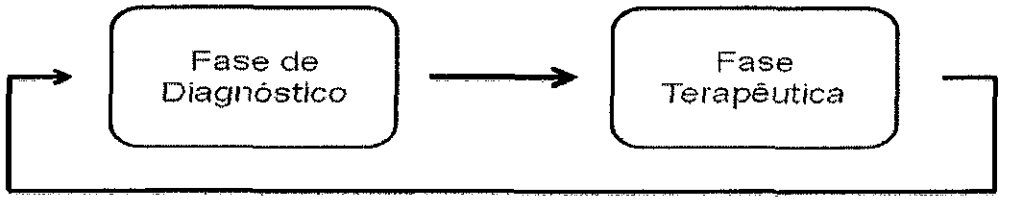


Figura 2 - Pesquisa-ação em um processo simplificado de duas fases (Baskerville, 2007)

Argyris *et al.* (1985) faz uma analogia de um processo semelhante a este com experimentação, afirmando que o conhecimento prático embutido na ação, representado pela identificação de fatores causais que podem ser manipulados para provocar as conseqüências desejadas dentro de uma conjunto de circunstâncias, é a hipótese sendo testada. Caso as conseqüências pretendidas ocorram, então a hipótese é confirmada, caso contrário, ela é refutada (ou as hipóteses auxiliares representadas pelas condições supostas do ambiente).

Ainda que não haja neutralidade na observação e manipulação do ambiente de estudo exigida para refutação de hipóteses, qualquer metodologia científica deve contribuir para o corpo de conhecimento científico de maneira válida e rigorosa. Além disto, a avaliação científica de qualquer afirmação dentro do corpo do conhecimento científico deve passar pela replicabilidade dos resultados encontrados. Como conseqüência, um processo de investigação semelhante ao apresentado na Figura 2 raramente é adotado na prática, pois não define qualquer tipo de orientação em relação a estas questões.

Para Checkland e Holwell (1998), qualquer modo de pesquisa pode ser pensado como vinculado aos seguintes elementos: um conjunto coeso de idéias formando um arcabouço conceitual F o qual é utilizado em uma metodologia M para investigar uma área de interesse A. Por exemplo, em Engenharia de Software poderia se pensar na seguinte pesquisa (simplificada) segundo a visão positivista: investigar se a manutenabilidade do *software* (área de interesse A) é melhorada por meio do uso de orientação a objetos (arcabouço conceitual F), através de estudos controlados (metodologia M). Ver parte superior da Figura 3.

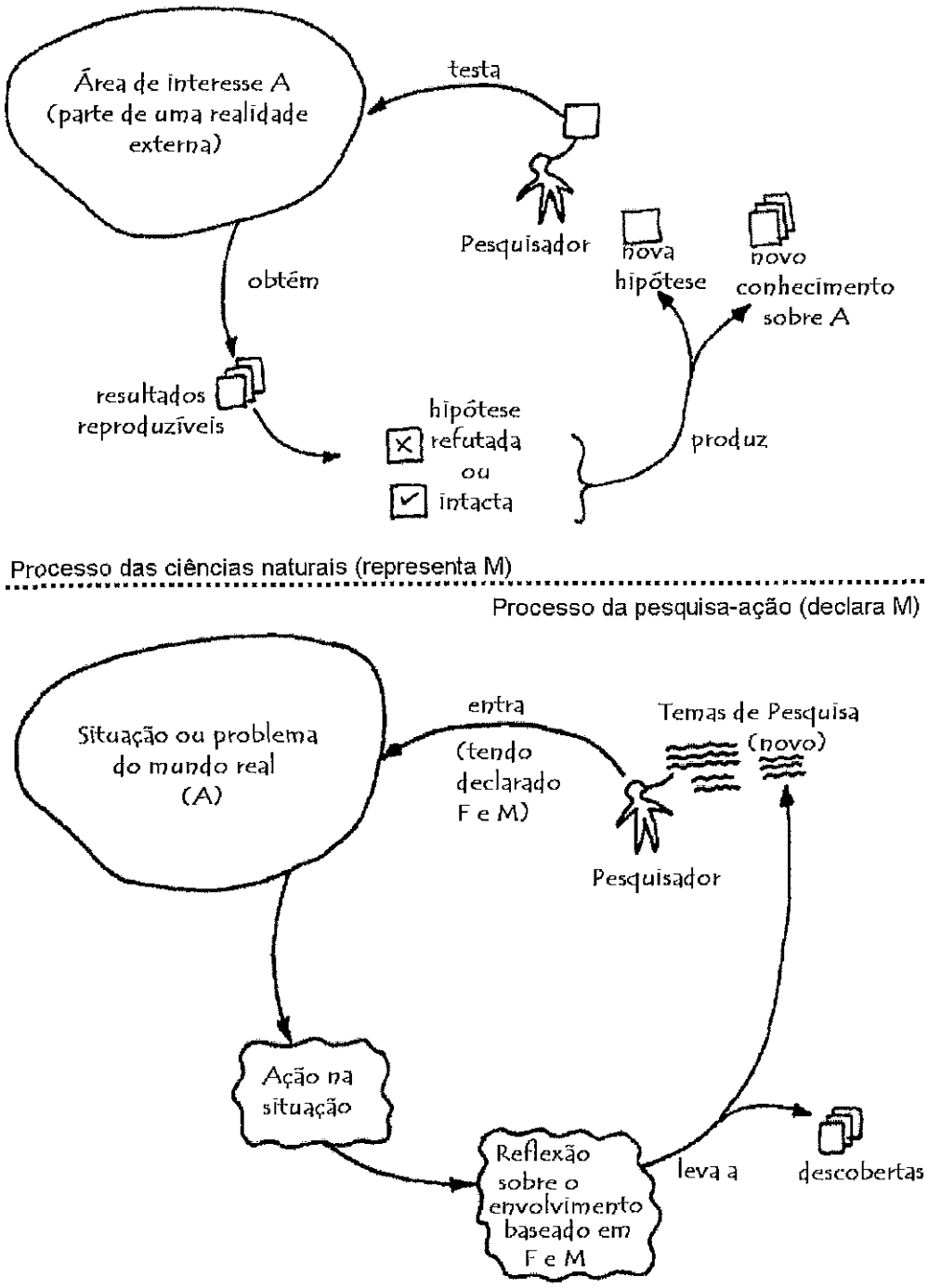


Figura 3 - Comparação entre os processos das ciências naturais e da pesquisa-ação – adaptado de Checkland e Holwell (1998)

No entanto, ainda segundo Checkland e Holwell (1998), a pesquisa-ação modifica o papel de F, M e até mesmo de A, na medida em que o pesquisador passa a estar envolvido no fluxo de transformações que ocorrem em uma determinada situação. Neste caso, o pesquisador interessado em um tema de pesquisa, declara F e M, e então entra em uma situação do mundo real em que aquele tema seja relevante. Como conseqüência, pesquisa-ação não apenas representa M, como nas ciências naturais, mas também deve declarar

um protocolo de pesquisa que permita que o processo de investigação seja *recuperável* por qualquer um interessado em submeter a pesquisa à análise crítica (parte inferior da Figura 3). Ainda que com isto a pesquisa-ação não atinja a repetibilidade alcançada nas ciências naturais, é suficiente para tornar claro os processos e modelos que permitiram ao pesquisador fazer suas interpretações e derivar suas conclusões. A importância de ter critérios explícitos antes que a pesquisa seja conduzida também é enfatizada por Avison *et al.* (1999) que afirmam que sem isto a avaliação dos resultados tende a ficar comprometida e, eventualmente, o que está sendo descrito possa ser ação (mas não pesquisa) ou pesquisa (mas não ação).

O processo apresentado na Figura 3 oferece indicações importantes de questões que devem ser tratadas para conferir rigor ao processo da pesquisa-ação, mas ainda assim não apresenta um guia explicativo de como a pesquisa deve ser conduzida. Com este propósito, outros pesquisadores expandiram a estrutura básica que guia o processo da pesquisa-ação. O processo mais comum, que segundo Davison *et al.* (2004) tem alcançado a condição de processo "canônico", é de Susman e Evered (1978), o qual é composto por cinco fases (Figura 4):

(1) Diagnóstico: consiste em descobrir o campo de pesquisa, os interessados e suas expectativas de uma perspectiva holística. Nesta etapa, existe ainda a definição do tema de pesquisa o qual é representado através da designação do problema prático e da área do conhecimento a ser abordada.

(2) Planejamento: etapa onde são definidas as ações para o quadro diagnosticado. As definições são guiadas pelas hipóteses as quais representam as suposições formuladas pelo pesquisador a respeito de possíveis soluções e resultados. As hipóteses, por sua vez, devem ser acompanhadas de uma formulação teórica científica.

(3) Tomada da ação: corresponde a implantação das ações planejadas. Um elemento central nesta etapa, segundo Thiollent (2007), é a técnica de seminário utilizada para examinar, discutir e tomar decisões acerca do processo de investigação.

(4) Avaliação: atividade onde se realiza a análise dos efeitos das ações frente ao apoio teórico utilizado como ponto de partida para a definição das ações.

(5) Reflexão: envolve a circulação de informação entre os participantes e outros setores da organização. A aprendizagem é facilitada pela colaboração temporária com especialistas em assuntos técnicos cujo conhecimento tenha sido útil ao grupo.

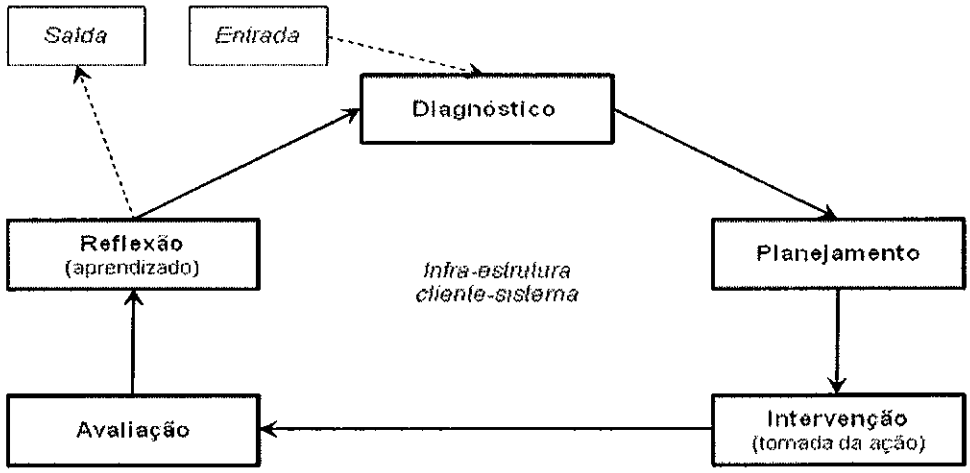


Figura 4 - Processo da pesquisa-ação "canônica" Davison *et al.* (2004)

Além destas atividades, o ambiente de pesquisa exige um contrato/acordo que legitima as ações e possíveis benefícios para ambas as partes (pesquisadores e organização), o qual compõe a chamada infra-estrutura cliente-sistema. Existem ainda duas etapas que não fazem parte diretamente do ciclo, e que dizem respeito ao início e ao fim do processo da pesquisa-ação. Para Avison *et al.* (2001), o início de um estudo de pesquisa-ação requer atenção, pois deve haver a percepção de uma real necessidade de melhoria por parte do cliente a qual demande um apoio científico, caso contrário, existe o risco de iniciar estudos para temas irrelevantes (em que não há perspectiva para o pesquisador de geração de conhecimento) ou estudos deixem de ser iniciados por conta de temas "iceberg" (onde o cliente não enxerga a necessidade de melhoria e, por isso, não busca ajuda de um pesquisador). Já a finalização de um estudo de pesquisa-ação deve se dar no momento em que ambas as partes estão satisfeitas com os resultados alcançados, o que ressalta a importância de um contrato/acordo prévio. Enquanto esta condição não é atingida, o processo da pesquisa-ação prevê iterações onde os resultados podem ser obtidos incrementalmente. Este aspecto iterativo é especialmente vantajoso quando o diagnóstico inicial não pode ser realizado de maneira plena.

Além do ciclo definido na Figura 4, apesar de não estar definido no processo de maneira explicita, existe a possibilidade de iteração e adaptação

entre as atividades. Por exemplo, pode-se retornar a etapa de diagnóstico após uma tentativa inicial de planejamento que não tenha sido concluída por falta de um maior detalhamento do problema ou a modificação do planejamento durante o curso das ações diante de situações inesperadas.

2.3.3. Características da Pesquisa-Ação

O processo descrito na seção anterior já antecipa algumas das características fundamentais da pesquisa-ação como um modelo de processo cíclico, a necessidade do acordo/contrato entre cliente e pesquisador, e o aprendizado através da reflexão. Davison *et al.* (2004) definem, além destas três, mais duas características essenciais a pesquisa-ação que os autores denominam "princípios da pesquisa-ação canônica":

- **Princípio da teoria:** pesquisadores precisam alicerçar-se sobre teorias como forma de guiar e focar suas atividades. Formuladas a partir de uma revisão da literatura, as teorias não só ajudam a conduzir a pesquisa, mas também a relatar os resultados posicionando-os frente à pesquisa existente. Teorias são normalmente da seguinte forma: em uma situação S que tem características evidentes F, G e H, os resultados X, Y e Z são esperados das ações A, B e C. Voltando ao processo da Figura 3, os elementos F e M representam as teorias utilizadas.
- **Princípio da mudança por meio da ação:** reflete a principal característica da pesquisa-ação que é a indivisibilidade entre pesquisa e ação. A falta de intervenção (ação) em uma situação de problema sugere a inexistência de um problema significativo. As ações planejadas devem ser desenhadas de forma a tratar os problemas observados, possibilitando o pesquisador a justificar cada ação como uma reparação de parte ou todo o diagnóstico realizado.

Todas estas características tornam a pesquisa-ação singular dentre as diferentes metodologias de pesquisa. A pesquisa-ação consegue se diferenciar da prática de rotina por ser pró-ativa e estrategicamente dirigida por teorias e técnicas de pesquisa, ao mesmo tempo em que se distingue da pesquisa científica "normal" por sua flexibilidade ao reconhecer a importância da colaboração entre pesquisador e participante e o valor da melhor evidência alcançável, mesmo que não se tenha boas medidas ou referências (baseline) (Tripp, 2005).

2.4. Estado Atual da Utilização da Pesquisa-Ação em Engenharia de Software

Para investigar o estado atual da utilização da pesquisa-ação em Engenharia de Software, uma extensa pesquisa da literatura foi conduzida. A pesquisa buscou artigos publicados em nove periódicos e três anais de conferências no período de 1993 a junho de 2009. Existem outras revisões da literatura realizadas por outros pesquisadores que buscaram especificamente sobre a utilização de uma metodologia de pesquisa como estudos controlados (Sjøberg *et al.*, 2005) e quasi-experimentos (Kampenes *et al.*, 2009), mas não foi identificada nenhuma especificamente sobre pesquisa-ação em Engenharia de Software. Foram identificados 16 artigos nos periódicos e conferências selecionados. Ainda que representem uma pequena fração da pesquisa sendo conduzida em Engenharia de Software, os artigos tratam diferentes contextos e, desta forma, são suficientes para ter uma indicação inicial do potencial da pesquisa-ação em Engenharia de Software.

Como forma de evitar viés durante a pesquisa da literatura, alguns princípios da revisão sistemática (Biolchini *et al.*, 2005) foram adotados como, por exemplo, a definição dos objetivos da pesquisa de forma antecipada, a organização da seleção dos artigos como um processo de várias fases e a documentação das razões para inclusão e exclusão dos artigos. A próxima subseção descreve parte deste processo e, após, os resultados são apresentados.

2.4.1. Método

Para conduzir esta revisão, seguiu-se o critério e alguns passos do método de pesquisa empregado em outras revisões da literatura (Lau, 1997, Sjøberg *et al.*, 2005). Os periódicos e conferências selecionados são os mesmo de Sjøberg *et al.* (2005) que, segundo os autores, configuram-se dentre os mais relevantes em Engenharia de Software. Os periódicos são: ACM Transactions on Software Engineering Methodology (TOSEM), Empirical Software Engineering (EMSE), IEEE Computer, IEEE Software, IEEE Transactions on Software Engineering (TSE), Information and Software Technology (IST), Journal of Systems and Software (JSS), Software Maintenance and Evolution (SME), e Software: Practice and Experience (SP&E). E as conferências: International Conference on Software Engineering (ICSE), the IEEE International Symposium on Empirical Software Engineering (ISESE), e the IEEE International Symposium

on Software Metrics (METRICS). Além destes, o International Symposium on Empirical Software Engineering and Measurement (ESEM) foi incluído nesta pesquisa, pois uniu as conferências METRICS e ISESE listadas acima.

Os termos utilizados para a busca foram selecionados de Baskerville e Wood-Harper (1998): "action research", "action learning", "action science", "reflective practice", "critical systems theory", "systems thinking" e "participative research". Dos 162 artigos retornados por estes termos, 138 foram eliminados pelo título e resumo e os outros 24 foram completamente lidos. Um dos principais julgamentos na seleção dos artigos foi avaliando se realmente se tratavam de pesquisa-ação. Muitos, por exemplo, se referiam a pesquisa-ação quando na verdade se tratavam de estudos de caso e vice-versa. Para fazer este julgamento, nossa decisão foi orientada pelos critérios de aceitação de estudos de pesquisa-ação definidos em Lau (1997) (que correspondem às características da pesquisa-ação discutidas na seção anterior) e os princípios de Davison *et al.* (2004), também apresentados na seção anterior. No entanto, prevendo a possibilidade de que encontrar um estudo de pesquisa-ação em Engenharia de Software que atendesse a todos estes critérios seria difícil, definiram-se *a priori* níveis de aderência à pesquisa-ação para classificar os estudos. A idéia de ter os níveis de aderência surgiu de um dos artigos selecionados onde os autores declararam explicitamente que o estudo conduzido por eles havia sido *inspirado* pela metodologia da pesquisa-ação (Abrahamsson e Koskela, 2004). Esta classificação e outras informações extraídas dos artigos estão sumarizadas na Tabela 1.

Tentou-se ser rigoroso em selecionar apenas artigos relacionados a pesquisa em Engenharia de Software, já que alguns periódicos e conferências também continham artigos relacionados a pesquisa em Sistemas de Informação. Para fazer esta distinção tomou-se como base Ramesh *et al.* (2004), conforme citado anteriormente. Finalmente, é importante mencionar que os artigos foram avaliados exclusivamente a partir do seu texto. Eventualmente, alguns deles poderiam ter sido avaliados de forma diferente se tivéssemos tido acesso ao relatório de pesquisa completo, particularmente quando alguns autores dão mais ênfase nos resultados em detrimento do processo de pesquisa empregado (Checkland e Holwell, 1998). Isto chama atenção para a importância em se ter diretrizes para relatar a pesquisa, talvez especificamente para pesquisa-ação em Engenharia de Software. Davison *et al.*

(2004) definiram critérios para tratar cada um dos cinco princípios da pesquisa-ação, citados na seção anterior, os quais podem ser pensados como diretrizes. Desta forma, foi esperado durante a pesquisa que os artigos selecionados atendessem a estes critérios como forma de demonstrar rigor e validade.

Tabela 1 - Informações extraídas dos artigos

Informação	Descição		Fonte de Referência
Problema	O problema tratado na pesquisa, normalmente relacionado a etapa de diagnóstico da pesquisa-ação.		(Lau, 1997)
Ação	Descrição da intervenção realizada.		(Lau, 1997)
Reflexão	Reflexão da ações implementadas e solução do problema tratado na pesquisa.		(Lau, 1997)
Taxonomia IEEE	Usado para classificar os tópicos de pesquisa.		(IEEE Keyword Taxonomy, 2002)
Aderência	Inspirado – quando existe o foco no aprendizado de pesquisadores na resolução de um problema do mundo real, realizando pesquisa em Engenharia de Software sem seguir os princípios da pesquisa-ação; Baseado – quando a metodologia da pesquisa-ação é modificada ou combinada com outras metodologias de pesquisa; Genuína – quando toda essência da metodologia da pesquisa-ação está presente.		
Tipo	Pesquisa-ação – foca na mudança e na reflexão; *Action Science* – tenta resolver os conflitos entre teorias propostas e aplicadas; Pesquisa-ação participatória – enfatiza a colaboração; *Action Learning* – possui instrução programada e aprendizado experiencial.		(Avison et al., 1999)
Tamanho	Duração do estudo.		(Sjøberg *et al.*, 2005)
Coleção de dados	Quantitativo ou qualitativo, incluindo que técnicas foram utilizadas para coleção de dados.		-
Estruturas de controle da pesquisa-ação	Inicialização	Pesquisador – estudo de campo; Profissional – geração clássica de um estudo de pesquisa-ação; Colaboração – evolui de uma interação existente;	(Avison *et al.*, 2001)
	Autoridade	Profissional – autoridade de consultor; Estagiado – migração da autoridade para o cliente; Identidade – profissional e pesquisador são os mesmos.	
	Formalização	Formal – contrato escrito especificamente para pesquisa; Informal – contrato amplo, eventualmente verbal; Evoluído – mudança de informal ou formal para a forma oposta.	
Ciclos da pesquisa-ação	Número de ciclos conduzidos.		(Davison *et al.*, 2004)

2.4.2. Resultados

Três resultados principais puderam ser compilados a partir da revisão. Primeiro, apresenta-se a distribuição das publicações entre os periódicos e conferências. Após, existem duas subseções sendo uma mostrando os domínios onde a pesquisa-ação tem sido empregada e a outra descrevendo como estes estudos foram executados.

A distribuição das publicações é apresentada na Tabela 2 (apenas periódicos e conferências a partir dos quais artigos foram selecionados são listados). O nível de aderência também é utilizado na distribuição dos artigos e aparece na tabela de maneira abreviada: Inspirado (I), Baseado (B) e Genuino (G). A partir da distribuição, é possível perceber um aumento no número de artigos publicados ao longo dos anos descrevendo estudos de pesquisa-ação. É importante notar que algumas conferências e periódicos ainda não publicaram em 2009 (por exemplo, ESEM). Além disto, o período 2004-2009 é menor que os outros dois, desta forma, existe a possibilidade de que a tendência de crescimento do número de artigos utilizando pesquisa-ação seja ainda maior.

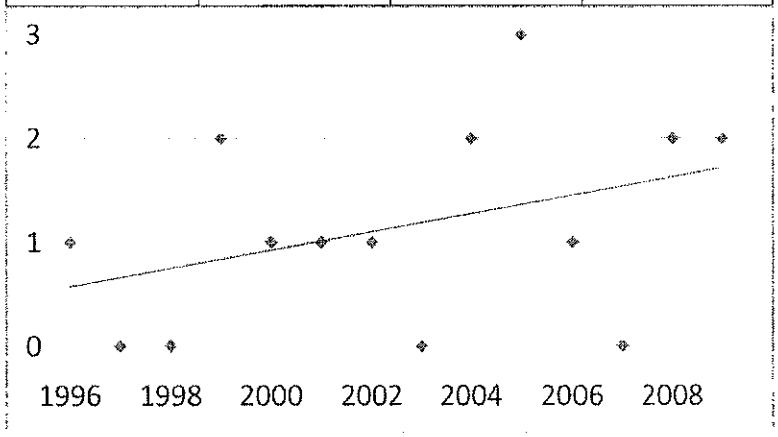
O número de artigos que foram classificados no nível inspirado foi relativamente alto, cerca de 30%. Isto significa que existe a necessidade do aumento do rigor nos estudos de pesquisa-ação em Engenharia de Software, se quisermos que investigações de pesquisa-ação possam formar uma base sólida para futuras pesquisas e aplicações na indústria em Engenharia de Software. Esta situação é ainda pior se considerarmos os vários estudos que mencionaram a utilização de pesquisa-ação, mas foram eliminados porque não puderam ser classificados nem no nível mais baixo (i.e., inspirado) de aderência.

Mais da metade dos periódicos e conferências utilizados para pesquisa não retornaram nenhum artigo. Considerando o número reduzido de estudos de pesquisa-ação que vêm sendo conduzidos em Engenharia de Software, não há muito que interpretar sobre este fato (caso houvesse um número grande de artigos concentrados em poucos periódicos/conferências, poder-se-ia interpretar que a pesquisa-ação ainda não é unanimemente aceita como uma metodologia de pesquisa para Engenharia de Software, mas não parece ser o caso). Uma nota, no entanto, é que o JSS retornou diversos artigos que não foram selecionados porque não eram do domínio de Engenharia de Software.

Ainda vale mencionar que dois dos três artigos do SP&E eram dos mesmos autores.

Tabela 2 - Distribuição dos artigos selecionados

Periódicos e Conferências	1993-1998			1999-2004			2005-2009		
	(I)	*(B)*	*(G)*	*(I)*	*(B)*	*(G)*	*(I)*	*(B)*	*(G)*
ESE				1			1		
ICSE						2			1
IEEE SW				1			1		
IST					1	1			3
SP&E	1					1		1	
TSE								1	
Totais	1			2	1	4	2	2	4
	1			7			8		



2.4.2.1. Tópicos e Contextos de Pesquisa

É interessante observar que, apesar do número reduzido, a pesquisa-ação vem sendo aplicada a uma ampla gama de domínios de pesquisa em Engenharia de Software (Tabela 3) indo desde o enfoque mais social (e.g., Gerência e Processo de Engenharia de Software) até o lado mais técnico (e.g., Construção de Software e Ambientes de Programação). O tópico com o maior número de estudos foi Processo de Engenharia de Software, e especificamente Implementação e Mudança de Processo. Este também foi o tópico onde o maior número de artigos no nível inspirado de aderência se concentrou, principalmente porque seus autores intercalaram a melhoria de processo de software com o processo da pesquisa-ação, deixando implícito quais eram as ações de pesquisa e de consultoria.

No que diz respeito à informação sobre diagnóstico/ação/reflexão extraída dos artigos, foi possível classificá-la em três grandes formatos: (1) investigação da introdução de tecnologias por meio de lições aprendidas (apresentando similaridades com estudos de caso), (2) concepção e/ou adequação de tecnologias com intensa colaboração e mudança com foco na intervenção (o formato que representa a essência da pesquisa-ação onde a

solução do problema é inicialmente desconhecida) e (3) facilitação e observação de atividades de Engenharia de Software (tendo um componente de consultoria).

Tabela 3 - Tópicos de pesquisa de acordo com a taxonomia IEEE

Taxonomia IEEE	# Artigos
Distribuição, Manutenção, e Melhoria	**2**
Documentação	(Lindvall e Sandahl, 1996)
Gerência da Manutenção	(Polo *et al.*, 2002)
Gerência	**2**
Modelagem e Controle do Projeto	(Canfora *et al.*, 2006)
Estimativa de Duração	(Staron e Meding, 2007)
Requisitos/Especificações	**2**
Métodos de Elicitação	(Maiden e Robertson, 2005)
Processo	(Kauppinen *et al.*, 2004)
Ambientes de Programação/Ferramentas de Construção	**1**
Ambientes para sistemas com múltiplos processadores	(Vigder *et al.*, 2008)
Arquitetura de Software	**2**
Arquiteturas especificas de domínio	(Bengtsson e Bosch, 1999)
Padrões	(Mattsson *et al.*, 2009)
Construção de Software	**2**
Gerenciamento e projeto de dados	(Fernández-Medina e Piattini, 2005)
Paradigmas de programação	(Lycett, 2001)
Processo de Engenharia de Software	**4**
Implementação e Mudança de Processo	(Fitzgerald e O'Kane, 1999) (Kautz *et al.*, 2000) (Salo e Abrahamsson, 2005)
Modelos de Processo de Software	(Abrahamsson e Koskela, 2004)
Outros	**1**
Bibliotecas de Software	(Staron *et al.*, 2009)

No geral, as iniciativas mais técnicas estavam relacionadas aos formatos (1) e (2). Por exemplo, um dos artigos relata uma metodologia de manutenção de software que foi criada no contexto de uma organização, enquanto que em outro caso a formalização de regras de projeto de arquitetura de software foi introduzida no contexto de desenvolvimento dirigido por modelos. Por outro lado, as pesquisas voltadas para os aspectos mais sociais estiveram mais relacionadas ao formato (3), como foi o caso dos artigos sobre melhoria de processo de software.

2.4.2.2. Detalhes de Execução dos Estudos

A extração de informações relacionadas aos detalhes de execução dos estudos foi problemática. Muitos artigos não descreviam as técnicas de coleção de dados utilizada, duração do estudo e o número de ciclos da pesquisa-ação. Além disto, quase todos os artigos não definiam explicitamente o tipo de pesquisa-ação empregado e as estruturas de controle. As estruturas de controle (Tabela 1) formam um importante componente da execução de um

estudo de pesquisa-ação, na medida em que tornam explícito o processo seguido e o porquê das decisões tomadas – possivelmente influenciadas por questões organizacionais. A definição do tipo de pesquisa-ação, por sua vez, torna claro o motivo pelo qual algumas características da pesquisa-ação foram enfatizadas. Ainda que o tipo muitas vezes não fosse explicitamente definido, em muitos casos foi possível deduzir implicitamente a partir da descrição geral do contexto e das ações de pesquisa – mas não se sabe se foi de fato a intenção do pesquisador ou se ele estava consciente desta decisão em relação ao tipo de pesquisa-ação.

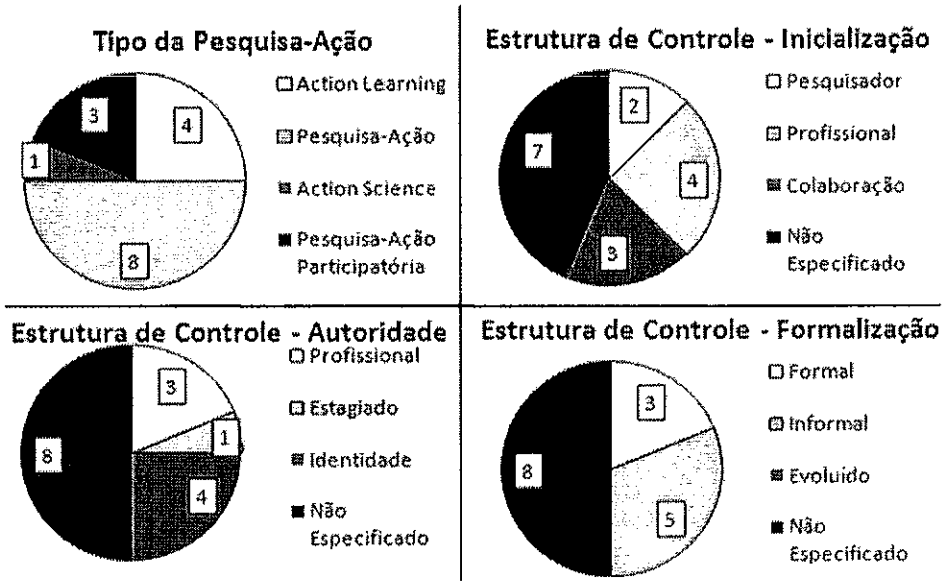


Figura 5 – Classificação dos artigos selecionados por tipo de pesquisa-ação e estruturas de controle

A Figura 5 mostra o número de artigos por tipo de pesquisa-ação e estruturas de controle. Examinando os gráficos da figura é possível perceber que grande parte dos estudos são do tipo Pesquisa-Ação, e são iniciados por profissionais os quais tem autoridade sobre a execução da pesquisa (autoridade do tipo identidade) que é conduzida sem nenhuma formalização. Este parece ser um resultado interessante já que possivelmente expressa que pesquisadores (que também tem o papel de profissional dentro da organização) estão conduzindo pesquisas com poucas restrições organizacionais. Ainda assim, vale reiterar o grande número de artigos que não mencionaram as suas estruturas de controle.

Todos os artigos fizeram uso intenso de dados qualitativos, confirmando esta intrínseca característica da pesquisa-ação na Engenharia de Software. Três artigos utilizaram dados quantitativos, indicando que pesquisa

quantitativa também é uma possibilidade, mesmo na pesquisa-ação. Observação foi de longe a técnica de coleção de dados mais mencionada seguida de perto de técnicas de entrevista. Para dados quantitativos, métricas foram utilizadas nos três artigos.

Por fim, a duração dos estudos variou entre 2 meses a 5 anos (a média foi de 21 meses e 16 foi o desvio padrão). Este resultado mostra que a pesquisa-ação é bastante flexível no que diz respeito a duração dos estudos e é mais influenciada pelo tópico de pesquisa estudado, tecnologia de software ou atividades envolvidas. Quatro estudos não especificaram a sua duração. Para o número de ciclos de pesquisa-ação, apenas dois artigos mencionaram explicitamente esta informação e ambos tiveram um ciclo. Mas, pela descrição linear dos outros artigos é possível esperar o mesmo número para grande parte deles. Estudos de pesquisa-ação em outras áreas (Davison *et al.*, 2004) não seguem esta tendência possuindo, normalmente, mais de um ciclo.

2.5. Conclusão

Neste capítulo, o papel e alguns desafios da pesquisa em Engenharia de Software foram discutidos e a metodologia da pesquisa-ação apresentada como uma opção viável de ser empregada para contribuir neste cenário. A pesquisa-ação pode ser empregada com diferentes focos (i.e., seguindo diferentes paradigmas de pesquisa), mas sempre tendo um forte componente da abordagem qualitativa. Além disto, os estudos de pesquisa-ação, por serem conduzidos em ambientes reais e se utilizarem do conhecimento prático por meio da intervenção conjunta entre pesquisadores e profissionais, normalmente permitem acessar as ricas camadas de significado criadas por atores sociais (Argyris *et al.*, 1985) e, por isto, tendem a resultar em conhecimento relevante para a comunidade científica.

Como conseqüência, a pesquisa-ação apresenta um grande potencial em Engenharia de Software. Uma revisão da literatura mostrou que o interesse na pesquisa-ação vem aumentando e, ainda que o número de estudos que empreguem esta metodologia de pesquisa seja bastante reduzido, reforça o seu potencial. Todos estes trabalhos identificados na revisão que empregaram pesquisa-ação estão, de alguma forma, relacionados com este trabalho na medida em que mostram a viabilidade da pesquisa-ação em Engenharia de Software. No entanto, uma discussão das particularidades da pesquisa-ação em Engenharia de Software ainda parece ausente na literatura,

e, devido ao crescente interesse, esta questão deve ser investigada com cuidado. A própria revisão da literatura revelou que o rigor no uso da metodologia deve aumentar e que existem pesquisadores que parecem ter interpretado a pesquisa-ação de maneira equivocada, inclusive confundindo pesquisa-ação com estudo de caso. Talvez, isto possa ser resultado da tentativa em se trazer uma metodologia de um domínio (ciências sociais) em que os estudos e metodologias costumam ser descritos utilizando uma linguagem diferente, não técnica. Novamente, isto indica a importância em se observar e relatar o comportamento da pesquisa-ação especificamente em Engenharia de Software.

Para permitir-nos tentar observar e tratar esta questão, dois estudos de pesquisa-ação foram conduzidos na indústria e são descritos nos dois capítulos seguintes. Ambos os estudos, apesar de terem sido conduzidos na mesma organização, são independentes. Além disto, o pesquisador, ao conduzí-los, teve como objetivo não só a questão de pesquisa tratada especificamente em cada estudo, mas, principalmente, observar e avaliar como o processo da pesquisa-ação se encaixa no processo de desenvolvimento de software o qual termina sendo afetado pelas intervenções de pesquisa.

Como mencionado no Capítulo 1, os estudos foram selecionados de forma oportunística, onde as situações do projeto foram identificadas e solucionadas por meio da metodologia da pesquisa-ação. A identificação de situações de projeto como oportunidades de pesquisa é uma característica importante da pesquisa-ação e, por isso, dissemos que os estudos foram selecionados de forma "oportunística". Um dos principais motivadores para utilizar a metodologia da pesquisa-ação e tê-la como mais apropriada para os estudos conduzidos foi a participação do pesquisador como membro da equipe do projeto de software desenvolvido pela organização. Além disto, tinha-se a expectativa desde o início em se atender todas as características da pesquisa-ação apresentadas neste capítulo.

Após os dois capítulos que descrevem os estudos, um capítulo será dedicado a discussão da aplicabilidade da pesquisa-ação em Engenharia de Software a partir das experiências na sua utilização.

Capítulo 3
Estudo de Pesquisa-Ação #1

O estudo descrito neste capítulo objetivou a externalização do conhecimento sobre estilos arquiteturais e padrões de codificação por meio de técnicas de refatoração de código em um projeto de software. Além disto, buscou investigar quais características de código afetam as decisões dos desenvolvedores para a refatoração. O capítulo está organizado com uma seção para cada fase da pesquisa-ação descrita anteriormente de forma a narrar todas as atividades de pesquisa.

3.1. Diagnóstico

O diagnóstico desta pesquisa colaborativa encontra-se dividido em três seções que descrevem o problema, o contexto de atuação e a definição do tema de pesquisa.

3.1.1.Descrição do Problema

Estilos de arquitetura e padrões de codificação são elementos essenciais da etapa de construção do desenvolvimento do *software*, aos quais estão associadas importantes características de qualidade, como legibilidade e manutenibilidade. Além disto, estes elementos têm fortes implicações sociais e gerenciais, pois servem de apoio à organização da equipe de projeto (Kazman e Bass, 2002).

A definição e documentação dos estilos arquiteturais e padrões de codificação, normalmente, devem ser realizadas *a priori*. Entretanto, em uma equipe com um número reduzido de integrantes estas questões podem ser facilmente trabalhadas por meio da comunicação direta. Por outro lado, na medida em que novos integrantes passam a fazer parte da equipe, incluindo equipes distribuídas, a comunicação direta passa a não ser o melhor meio para disseminar o conhecimento da equipe em relação à arquitetura e os estilos de código utilizados – considerando que o comprometimento da comunicação normalmente afeta a produtividade dos desenvolvedores e a qualidade do *software*. Dois fatores têm maior relevância neste contexto, (1) o aprendizado dos novos desenvolvedores sobrecarrega a comunicação e (2) equipes remotas trazem dificuldades à comunicação.

Para atenuar o impacto sobre a comunicação, o conhecimento antes disseminado diretamente entre integrantes da equipe de projeto deve ser externalizado e capturado explicitamente. Uma maneira imediata de proceder na externalização deste conhecimento é utilizar os desenvolvedores mais experientes no projeto. No entanto, devido a sua experiência, existe a possibilidade de que estes desenvolvedores não possam ser capazes de compreender as reais necessidades, em termos de informação, daqueles que não possuem experiência, já que estas necessidades não fazem parte do dia a dia destes desenvolvedores com maior experiência.

O problema tratado nesta pesquisa é, então, investigar mecanismos que permitam a externalização do conhecimento relacionado aos estilos arquiteturais e padrões de codificação de maneira a considerar as reais necessidades dos novos integrantes da equipe de projeto – e, também, de equipes distribuídas.

3.1.2.Contexto do Projeto/Trabalho

O projeto referente a esta pesquisa visa o desenvolvimento de um novo sistema de informação Web para gerenciamento das atividades de uma Fundação de Apoio a projetos de Ciência e Tecnologia. Trata-se de um projeto de grande porte que envolve diferentes setores da organização, como: recursos humanos, financeiro, contabilidade, gerenciamento de projetos e protocolo. O projeto pôde ser modularizado e desenvolvido seguindo um ciclo de vida incremental. Entre os fatores que justificaram esta decisão, podemos citar os dois seguintes: o cliente está interessado em (1) entregas parciais do produto e (2) na substituição gradual do sistema de informação atual pelo novo. Além disto, a Fundação possui um modelo de referência de processo baseado nos níveis G a C do MPS.br (SOFTEX, 2007).

O sistema é desenvolvido utilizando tecnologia Java. A arquitetura projetada é baseada no estilo arquitetural *Modelo-Visão-Controle* (MVC) onde a camada *Visão* é implementada pela tecnologia JavaServer Faces (JSF – *framework* baseado em componentes visuais para criação de interfaces Web) e as camadas *Modelo* e *Controle* utilizando classes Java simples. Atualmente, a equipe responsável pela codificação e projeto é distribuída, e possui três projetistas (sendo um atuando também como desenvolvedor) e dois desenvolvedores no local da Fundação (equipe L) e mais seis desenvolvedores remotos em uma cidade distante 200km (equipe R). A experiência média com

desenvolvimento da equipe L é de cerca de sete anos (variando entre 3 e 15 anos), enquanto que da equipe R cerca 2,5 anos (variando entre 1,5 e 4 anos). Existem ainda, na equipe L, mais três profissionais responsáveis pela especificação e atividades de validação, verificação e teste.

A construção de cada módulo é realizada em iterações, compostas por um conjunto de casos de uso conceitualmente relacionados, as quais seguem um processo cascata de construção. Em cada iteração, a etapa de projeto constitui da definição, pelos projetistas, das chamadas classes de negócio correspondendo à camada *Modelo* da arquitetura. Com o projeto liberado, os desenvolvedores ficam responsáveis, principalmente, pela codificação das camadas *Visão* e *Controle*. Além disto, a comunicação entre as duas equipes é realizada por meio de tecnologias de *e-mail*, mensagem instantânea e vídeo conferência, e as demandas do projeto – como defeitos, ajustes e melhorias – registradas e acompanhadas por meio da Web no sistema Trac (sistema de gerenciamento de mudanças – *issue tracking system*) (Trac, 2003).

A documentação detalhada referente aos estilos de arquitetura e padrões de codificação utilizados não é um artefato previsto no processo de desenvolvimento. A maior parte da definição destes estilos e padrões foi trazida da documentação oficial da Sun (Sun, 1999) e recomendações da indústria (Crupi *et al.*, 2004), mas houve algumas adaptações e modificações realizadas pela equipe de desenvolvimento. Como exemplo deste tipo de modificação, pode-se citar a criação de um mecanismo próprio para a validação de formulários de entrada de dados não utilizando, neste caso, os recursos oferecidos pela tecnologia JSF.

Em março de 2008, época do início de um novo módulo do projeto, foi integrada ao projeto a equipe R. Isto representou um aumento de oito para quatorze profissionais. Estes novos integrantes da equipe R receberam treinamento presencial durante dois dias. Durante o treinamento a tecnologia utilizada foi apresentada, e aspectos relacionados a procedimentos de construção foram discutidos. Os seguintes assuntos foram abordados:

- A tecnologia Java e JavaServer Faces;
- Padrão arquitetural *Modelo-Visão-Controle*, incluindo discussões sobre orientação a objetos, como encapsulamento e reutilização através de herança;

- Regras de nomenclatura e organização do código fonte – apresentados durante a (re)construção do caso de uso de um módulo anterior que foi utilizado como exemplo;
- Atividades de gerência de configuração e;
- Visão geral do processo de desenvolvimento, incluindo as atividades de especificação e teste.

Ainda que estas atividades de acompanhamento e treinamento tenham sido conduzidas, ao final da primeira iteração de construção do novo módulo, membros mais experientes da equipe L identificaram problemas relacionados à qualidade do código fonte construído pelos novos membros da equipe. Embora não desejado, este é um comportamento que geralmente se observa nos projetos de software e ocorre, dentre outros motivos, devido à inexperiência dos novos desenvolvedores no projeto (Bennet e Rajlich, 2000). Estes problemas estavam relacionados principalmente a não organização do código fonte de acordo com os padrões de codificação e estilos arquiteturais tácitos e esperados (já que não estavam documentados) pelos membros mais experientes.

É importante ilustrar ainda qual foi a organização adotada pela equipe com o objetivo de facilitar o diálogo e evitar falhas de comunicação. A Figura 6 representa a estrutura da equipe de construção. Para evitar a diálogo desordenado entre todos os membros entre si, foi criado o papel de responsável pela comunicação o qual deveria ser o encarregado por comunicar/responder qualquer necessidade a outra equipe. Quando alguma informação ou acerto em artefatos ficava pendente, o Trac era utilizado para registrar e acompanhar a resolução da demanda.

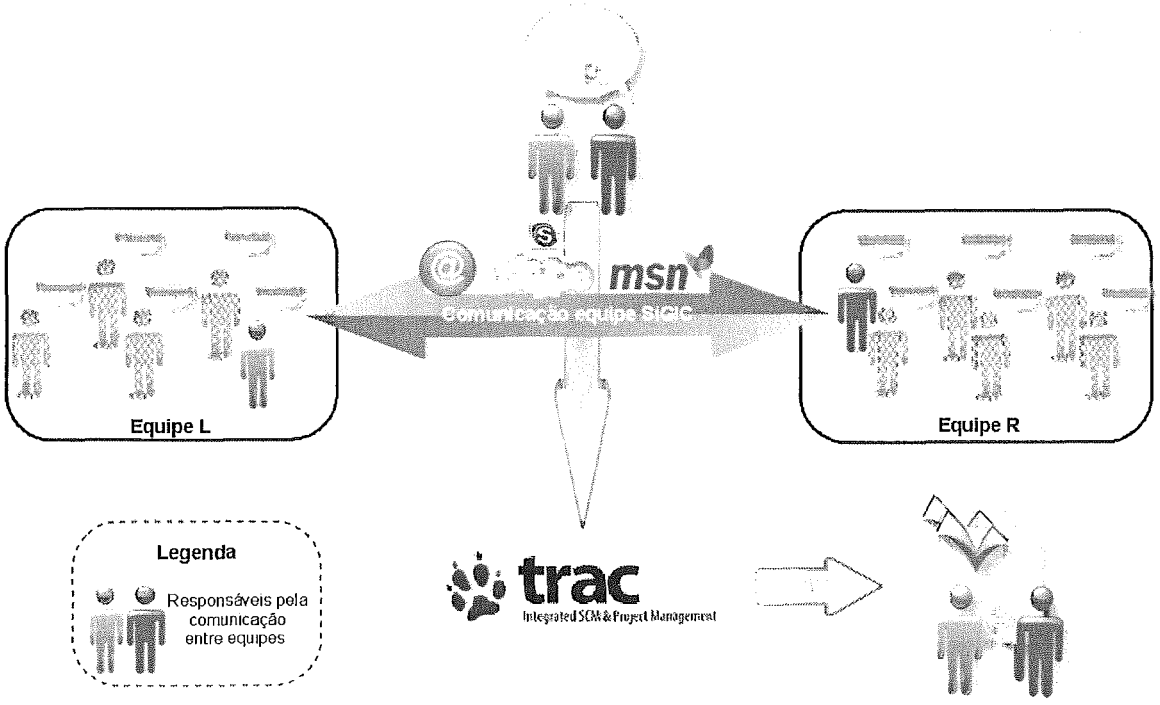


Figura 6 - Organização da Equipe

3.1.3.Tema de Pesquisa

O tema de pesquisa relacionado ao cenário apresentado acima pôde então, a partir dos problemas identificados, ser definido como: estratégias para detecção de "desvios" em código fonte relacionados à falta de conhecimento sobre padrões de codificação e estilos arquiteturais. A idéia é tentar utilizar técnicas de refatoração como forma de identificar desvios de código frente ao conhecimento tácito dos desenvolvedores experientes. Desta forma, estaremos trabalhando com a refatoração de código com objetivo de melhorar a sua qualidade de acordo com a percepção dos desenvolvedores mais experientes. Segundo Mens e Touwé (2004), a idéia central da refatoração é redistribuir ou reescrever classes, variáveis e métodos na hierarquia de classes a fim de facilitar futuras adaptações ou extensões. Em outras palavras, o objetivo da refatoração é melhorar a estrutura interna do software mantendo o seu comportamento externo inalterado.

Neste sentido, tem-se a expectativa de que será possível determinar com maior precisão, por meio dos defeitos ("anomalias") identificados no código produzido pelos desenvolvedores menos experientes, o conhecimento que deve ser explicitado. Por trás desta expectativa está sendo explorado o fato de que inspeções podem ser ferramentas úteis ao aprendizado de uma organização,

na medida em que bons padrões de desenvolvimento podem ser observados durante a refatoração.

3.2. Planejamento

Nesta seção encontra-se descrita a fase de planejamento, que é iniciada pela revisão da literatura técnica onde trabalhos relacionados ao tema de pesquisa são estudados. A partir disto, o enfoque da ação da pesquisa é fixado e as hipóteses associadas aos resultados esperados determinadas. Além disto, são definidos os instrumentos que serão utilizados, além de possíveis ferramentas e técnicas para a pesquisa.

3.2.1.Revisão da Literatura

Dado o tema de pesquisa, a busca por trabalhos publicados na literatura teve foco em técnicas de inspeção as quais pudessem guiar a leitura do código e revelar os desvios mencionados anteriormente. Um critério usado nesta revisão era que a técnica fosse de fácil aplicação, ou seja, que não necessitasse de treinamento dos desenvolvedores para aplicação, apenas a descrição dos seus passos deveria ser suficiente. Além disto, tentou-se pesquisar na literatura mecanismos que pudessem mostrar as motivações por trás da identificação dos defeitos – ou seja, uma taxonomia de defeitos que fosse adequada à categorização dos problemas associados aos estilos arquiteturais e padrões de codificação. A pesquisa foi conduzida de forma *ad-hoc* e os trabalhos selecionados por conveniência.

Dois trabalhos foram selecionados nesta revisão (Dunsmore *et al.*, 2003) e (Mäntylä, 2005), os quais foram utilizados para compor o procedimento elaborado para a leitura do código fonte (formulário da Tabela 4). O trabalho de Dunsmore *et al.* (2003) descreve uma técnica de inspeção baseada em casos de uso a qual permite a leitura de código orientado a objetos a partir do ponto de vista do seu modelo dinâmico. Os passos da técnica envolvem (1) a seleção de um caso de uso, (2) derivação de cenários de uso a partir do caso de uso (ex., "salvar protocolo", "cancelar protocolo") e (3) a leitura dos métodos das classes responsáveis pela execução do cenário.

Para categorizar as sugestões de refatoração foi utilizado o trabalho de Mäntylä e Lassenius (2006) – que é um trabalho posterior e diretamente relacionado à Mäntylä (2005). Nele, os autores descrevem um estudo experimental onde o objetivo foi entender o porquê desenvolvedores entendem

ser necessária a refatoração de código fonte. Trata-se de um estudo qualitativo onde foi requisitado aos participantes que registrassem o motivo para a necessidade de refatoração. Através da análise dos registros feitos pelos participantes foi elaborada uma taxonomia de defeitos. Para permitir a utilização da taxonomia dos desvios o seguinte passo foi incluído na técnica descrita anteriormente (Tabela 4): (4) para cada método, necessidades de refatoração devem ser registradas de acordo com o conhecimento prévio do desenvolvedor. Os autores sugerem um conjunto de métricas de código que poderiam ser utilizadas para avaliar algumas de suas propriedades do código refatorado. Estas métricas foram utilizadas por questões de reaplicação do estudo.

É importante observar, por fim, a existência de uma grande proximidade entre as atividades de um processo de inspeção e refatoração de código. Ambos possuem, de maneira geral, etapas de planejamento, detecção, retrabalho e avaliação. De fato, o processo de refatoração pode ser entendido como um tipo de inspeção. Assim, da mesma forma que a inspeção, o processo refatoração de código pode proporcionar um aprendizado aos envolvidos em relação às boas práticas de codificação no contexto onde a refatoração é aplicada.

3.2.2.Enfoque da Ação

3.2.2.1. Objetivos

O objetivo principal deste estudo, definido utilizando a abordagem *Goal-Question-Metric* (GQM) proposta por Basili *et al.* (1994):

> *analisar* a utilização de refatoração de código *com o propósito* de caracterizar *com respeito à* externalização do conhecimento associado aos estilos arquiteturais e padrões de codificação *do ponto de vista do* engenheiro de software *no contexto de* sistemas de informação Web

Existe ainda um objetivo adicional, o qual foi acrescentado para permitir a comparação dos resultados com o trabalho de Mäntylä (2005):

> *analisar* se características mensuráveis de código fonte podem explicar a avaliação subjetiva da necessidade de refatoração de código *com o propósito* de caracterizar *com respeito à* qualidade da predição[3] das

[3] Qualidade da predição foi a expressão usada para o termo inglês *predictability*.

medidas *do ponto de vista do* engenheiro de software *no contexto de* sistemas de informação Web

3.2.2.2. Questões de Pesquisa

O objetivo será alcançado quando respostas forem apresentadas às seguintes questões:

- Q.1. Quais são os fatores ou características do código fonte que um desenvolvedor utiliza como base para identificar necessidades de refatoração e qual o seu efeito sobre métricas de código? Esta questão objetiva o entendimento das motivações embutidas na identificação de necessidades de refatoração por parte de desenvolvedores experientes. Ou seja, a partir da análise das necessidades de refatoração identificadas, quais as características de qualidade do código foram observadas para que uma refatoração seja apontada como necessária.
- Q.2. Refatoração de código é útil para externalizar o conhecimento sobre estilos arquiteturais e padrões de codificação utilizados em um projeto, particularmente aqueles necessários para um novo desenvolvedor que esteja se integrando a equipe? A idéia desta questão é verificar se as necessidades de refatoração identificadas na questão 1 servem como entrada para a documentação dos estilos arquiteturais e padrões de codificação utilizados no projeto.

3.2.2.3. Resultados Esperados (coleção de dados)

A questão Q.1 possui as seguintes questões práticas associadas:

- Q.1.1 Qual é o efeito da refatoração sobre as métricas de código?
- Q.1.2 Quais características do código afetam a decisão dos desenvolvedores?
- Q.1.3 Quais tipos de refatoração são identificados apenas pelos desenvolvedores experientes?
- Q.1.4 É possível categorizar as necessidades de refatoração de forma que reflita as características de qualidade do código esperadas?
- Q.1.5 Qual a percepção dos desenvolvedores do efeito da refatoração sobre a qualidade do código?

As métricas de código sugeridas em Mäntylä (2005) (linhas de código, número de parâmetros, complexidade ciclomática, número de métodos remotos invocados, *fan-out* e *fan-in*) serão utilizadas para as questões Q.1.1. As métricas também serão utilizadas para a Q1.2 na medida em que representam

diferentes aspectos estruturais do código. As métricas foram escolhidas, segundo Mäntylä (2005), baseadas no seu reconhecimento na literatura técnica e no propósito de caracterizar os métodos de diferentes perspectivas. Cabe ainda uma descrição de como cada métrica foi coletada:

- Linhas de código: número de linhas que contém código fonte (incluindo comentários).
- Número de parâmetros: número de parâmetros de um método.
- Complexidade ciclomática: #Arestas – #Nós + #Componentes Conectados em um grafo de fluxo de controle.
- Número de métodos remotos invocados: número de métodos que são invocados dentro de um método.
- *Fan-in:* número de entradas que um método utiliza. Inclui parâmetros e variáveis globais.
- *Fan-out:* número de variáveis de saída que são modificadas. Incluindo parâmetros (comando "return") e variáveis globais.

Para a Q.1.4 foi realizada uma atividade de categorização junto aos desenvolvedores utilizando a taxonomia de Mäntylä e Lassenius (2006) (seção 3.4.1), com base nas descrições fornecidas no formulário da Tabela 4. Já a Q.1.3 e Q.1.5 serão respondidas por meio de entrevistas.

A questão Q.2 tem as seguintes questões práticas associadas:

- Q.2.1 A descrição da identificação das necessidades de refatoração deve conter detalhes sobre o que motivou a uma determinada identificação.
- Q.2.2 A refatoração do código deve ser realizada por aqueles que escreveram o código a fim de permitir o aprendizado por parte dos desenvolvedores.
- Q.2.3 Qual a melhor forma de formalizar o conhecimento externalizado?

A questão 2.1 deverá ser capturada no formulário de discrepâncias da Tabela 4. A questão 2.2 requer atividades de acompanhamento de projeto. A questão 2.3 será discutida entre os próprios membros da equipe L e R, tanto por aqueles que identificaram necessidades de refatoração quanto por aqueles que refatoraram o código.

3.2.3.Hipóteses (suposições)

- Existe um conhecimento tácito no projeto relacionado à organização do projeto segundo um estilo de arquitetura e padrões de codificação.
- A refatoração de código, utilizando a técnica inspeção de Dunsmore *et al.* (2003), permitirá externalizar o conhecimento relacionado ao estilo arquitetural e padrões de codificação utilizados no projeto.

3.2.4.Definições Operacionais

3.2.4.1. Técnicas

Entrevistas semi-estruturadas serão a principal forma de coleta de dados qualitativos durante esta pesquisa. Além disto, formulários de caracterização também deverão ser respondidos pelos envolvidos na pesquisa. Dados quantitativos serão extraídos diretamente do código fonte e analisados por meio de estatística descritiva e preditiva.

3.2.4.2. Instrumentos

A Tabela 1 apresenta as informações contidas no formulário de discrepâncias (sugestões de refatoração) utilizado pelos desenvolvedores para descrever as necessidades de refatoração do código. Além dos campos preenchidos pelos desenvolvedores, o formulário contém ainda as etapas da técnica que deveriam ser observadas no intuito de facilitar a leitura e entendimento do código – a compreensão do código é facilitada, pois a técnica auxilia a manter o foco da leitura em um caso de uso por vez (Dunsmore et al., 2003). Os passos 1, 2 e 3 foram retirados diretamente de Dunsmore et al. (2003) enquanto que os campos 5 e 6 trazidos de Mäntylä e Lassenius (2006). Os outros campos e passos foram criados especificamente no contexto deste trabalho.

Tabela 4 - Conteúdo do formulário de discrepâncias (defeitos)

Passos da técnica
1 - Selecione um caso de uso do projeto; 2 - Derive cenários de uso a partir do caso de uso (ex., "salvar protocolo", "cancelar protocolo"); 3 - Percorra os métodos das classes responsáveis pela execução do cenário. Verifique se os métodos corretos estão sendo invocados e se o estado do cenário está sendo mantido e manipulado da forma adequada e consistente pelo sistema; 4 - Para cada método, registre oportunidades de refatoração de acordo com o seu conhecimento prévio (ex., aninhamento excessivo, nomenclatura de variáveis, organização "visual" do código, dentre outros).
Campos do formulário
1. Caso de uso 2. Descrição do cenário 3. Nome da classe 4. Nome do método

5. Você refatoraria este método? (responda de 1 a 5 - 1 - Não; 2 - Dificilmente; 3 - Talvez; 4 - Sim, mas apenas depois que o método for evoluído; 5 - Sim, agora mesmo)
6. Explique sua decisão na questão acima. Se a refatoração é necessária, explique o quê deve ser mudado, como e o porquê. Se o método estiver OK, explicite as qualidades desejáveis que estão sendo atendidas incluindo padrões de projeto. Se sua resposta é talvez, também explique seu raciocínio. Se apropriado, mencione as linhas de código envolvidas na sua explicação.

Duas entrevistas foram planejadas para o estudo onde o objetivo é abordar os tópicos apresentados nas Tabela 5 e Tabela 6.

Tabela 5 – Entrevista semi-estruturada 1

Tópicos relacionados à refatoração e ao formulário preenchido[4]	
(i)	O procedimento sugerido e as perguntas foram úteis para focar a refatoração no propósito acordado?
(ii)	Como você classificaria cada explicação associada à motivação para a refatoração segundo os tipos citados em Mäntylä e Lassenius (2006)? (Q.1.4)
(iii)	Qual o efeito da refatoração sobre a qualidade do código? (Q.1.5)
(iv)	Quais tipos de refatoração são identificados apenas pelos desenvolvedores experientes? (Q.1.3)
(v)	Faltou alguma característica a ser capturada no formulário ou o espaço livre foi suficiente?

Tabela 6 - Entrevista semi-estruturada 2

Tópicos relacionados à externalização do conhecimento associado aos padrões de codificação e estilo arquitetural utilizados.	
(i)	De que maneira as necessidades de refatoração identificadas permitiu o aprendizado dos padrões de codificação e estilo arquitetural utilizados no projeto?
(ii)	Decidir como formalizar o conhecimento externalizado. (Q.2.3)
(iii)	Em sua opinião a refatoração foi útil como mecanismo para externalização deste conhecimento? Você entende que ainda está faltando externalizar algum conhecimento?

3.2.4.3. Ferramentas

Como ferramenta de apoio a pesquisa, pretende-se utilizar listas de discussão de e-mail. Para a medição do código a ferramenta "Understand – Source Code Analysis & Metrics" (http://www.scitools.com/products/understand/) será utilizada. E para análise dos dados quantitativos a ferramenta SPSS (http://www.spss.com/).

3.3. Ações

Inicialmente, a equipe R foi informada que problemas haviam sido observados no código fonte, ressaltando que isto se devia a falhas de comunicação e aprendizado com o desenvolvimento distribuído. Por esta razão, uma

[4] É importante diferenciar, neste momento, os três "tipos" de questões mencionados neste estudo. As questões de pesquisa estão associadas a questões práticas as quais indicam como a coleção de dados deve ser realizada. As questões listadas em algarismos romanos dizem respeito a condução da entrevista em si, sendo que algumas delas estão associadas a questões práticas.

refatoração no código seria realizada com intuito de tentar mostrar a equipe como o código deveria ser estruturado. Dois desenvolvedores (um projetista/desenvolvedor e um desenvolvedor), mais experientes e disponíveis, da equipe L foram selecionados para realizar a revisão do código e identificar necessidades de refatoração, ficando responsáveis por diferentes casos de uso.

O formulário de discrepâncias da Tabela 4 foi apresentado aos desenvolvedores que, então, receberam instruções sobre como preenchê-lo. As instruções incluíram o foco da revisão que em linhas gerais se deveria buscar por "desvios de código" segundo o que os desenvolvedores consideravam como padrão de codificação ou estilo arquitetural utilizado no projeto. Alguns exemplos foram discutidos utilizando o conhecimento prévio dos desenvolvedores sobre refatoração e conceitos sobre abstrações em orientação a objetos e linguagens de programação (herança, legibilidade, manutenibilidade, dentre outros). Após as instruções, a atividade foi iniciada e, após o seu término, os formulários preenchidos foram enviados àqueles que construíram os casos de uso.

Com base nas necessidades de refatoração identificadas e registradas nos formulários, a equipe R foi instruída a refatorar o código. Foi requisitado, ainda, que fosse feito o registro das versões (do sistema de controle de versão) dos arquivos fonte que estavam sendo refatorados e o tempo gasto com a refatoração de cada método. Os responsáveis pela construção dos casos de uso refatoraram o seu próprio código, mas poderiam discordar com as necessidades de refatoração apontadas.

Deram-se início, então, as entrevistas semi-estruturadas planejadas. A entrevista 1 e 2 foram realizadas junto aos desenvolvedores. A transcrição das entrevistas está no Anexo A. Foi solicitado aos desenvolvedores da equipe L, após a entrevista, que classificassem com o pesquisador as diferentes necessidades de refatoração apontadas por eles, utilizando como base a classificação apresentada por Mäntylä e Lassenius (2006).

Com isto as atividades de análise e aprendizado puderam ser conduzidas e são detalhadas nas próximas duas seções.

3.4. Avaliação e Análise

Neste estudo têm-se dois tipos de dados que serão explorados. As métricas do código que sofreu refatoração, visando atender as questões Q.1.1, Q.1.2 e

Q.1.4, e as entrevistas com os desenvolvedores com objetivo de responder às questões Q.1.3, Q.1.5 e Q.2.3. Neste sentido, esta seção apresenta a análise destes dados considerando as preocupações das questões práticas citadas. As questões de pesquisa (Q.1 e Q.2) às quais estas questões práticas (Q.1.1, Q.1.2, Q.1.3, Q.1.4, Q.1.5 e Q.2.3) estão associadas serão discutidas na seção 3.5 com base nos resultados apresentados e analisados nesta seção.

3.4.1.Analisando as Métricas de Código

Nesta seção serão analisadas as métricas de código de antes e depois de sua refatoração. A idéia é verificar quais características do código foram alteradas devido à refatoração do código. Além disto, tentar-se-á verificar a existência de alguma relação entre alguma característica do código manifestada por meio das métricas e a categorização das necessidades de refatoração. As métricas utilizadas aqui foram sugeridas em Mäntylä (2005) e foram coletadas para cada um dos métodos que sofreram algum tipo de refatoração. Além das métricas de código também será apresentado o tempo gasto para realizar a correção do código.

Antes de examinar o efeito da refatoração sobre as métricas de código os dados gerais são apresentados na Tabela 7. Um fato interessante que pode ser verificado através desta tabela é um alto percentual de métodos que, segundo os desenvolvedores, necessitavam de refatoração (67% e 88% para os desenvolvedores 1 e 2, respectivamente, e 79% no total – valores calculados dividindo-se o valor da coluna métodos com necessidade de refatoração e métodos revisados). Isto pode ser considerado um indício de que o diagnóstico feito inicialmente pela equipe L, que o código produzido pela equipe R necessitava adequar-se aos padrões tácitos do projeto, parece coerente.

Tabela 7 - Dados da refatoração de código

	Métodos Revisados	Métodos sem Necessidade de Refatoração	Métodos com Necessidade de Refatoração	Necessidades de Refatoração Sugeridas	Refatoração não realizada (falso-positivo)	Tempo (hs)
Desen. 1	21	7	14	19	1	12:15
Desen. 2	26	3	23	37	10	5
Totais	47	10	37	56	11	-

Como mencionado anteriormente, cada desenvolvedor descreveu o porquê de cada necessidade de refatoração que sugeriu. A partir dessa descrição, a taxonomia criada em Mäntylä e Lassenius (2006) foi, de forma análoga, aplicada e adaptada em cima das descrições relatadas pelos desenvolvedores. As categorias, assim como um resumo do efeito da refatoração sobre as métricas de código considerando cada categoria, são

apresentadas na Tabela 8. Duas categorias novas surgiram no contexto deste estudo, as categorias "baixa coesão" e "código não utilizado" (sublinhadas na Tabela 8). No estudo realizado em Mäntylä e Lassenius (2006), o código revisado pelos participantes foi criado especificamente, de maneira artificial, para o estudo. Desta forma, o surgimento destas duas novas categorias denota como o contexto de um projeto real pode revelar variações talvez não consideradas no estudo controlado. Por outro lado, diversas categorias identificadas em Mäntylä e Lassenius (2006) não se manifestaram no contexto deste projeto o que indica a particularidade de um projeto real. Ao todo, 13 categorias foram identificadas no contexto deste estudo e, em contrapartida, Mäntylä e Lassenius (2006) identificaram 22 categorias – ou seja, apenas 50%[5] das categorias foram identificadas neste trabalho. As categorias presentes em Mäntylä e Lassenius (2006) e não identificadas neste trabalho são: ortografia, nome de variável ruim, compreensibilidade ruim, total desorganização, instrução longa, localização incorreta do método, grande número de parâmetros, passar por parâmetros, aninhamento excessivo, linhas vazias insuficientes e agrupamento ruim.

Outro dado interessante da Tabela 8 é que apesar do número de métodos analisados ter sido maior (47 neste estudo contra 10 em Mäntylä e Lassenius (2006)), ainda assim o número de categorias identificadas foi menor, mostrando que os desenvolvedores que implementaram o código cometeram sempre os mesmos tipos de erros. Isto de certa forma pode ser um indício de que algum conhecimento tácito ao qual os desenvolvedores da equipe R não teve acesso parece de fato existir. Outro dado que fortalece esse indício é que quatro categorias (algoritmo ruim, organização interna ruim, pequenos problemas de estrutura e baixa coesão) respondem por quase 70% das necessidades de refatoração identificadas (Tabela 8).

Para analisar o efeito sobre as métricas é interessante dividir as sugestões em dois tipos: defeitos e melhorias – os dois tipos estão separados na Tabela 8 por uma linha vazia. Em cada linha da Tabela 8, os valores foram calculados subtraindo-se o valor da métrica antes e após a refatoração e somados por categoria. Desta forma, valores negativos representam decréscimo do valor da métrica. Apesar de não ter sido requisitado aos desenvolvedores, ambos detectaram defeitos no código fonte durante a sua revisão. Grande parte

[5] Para chegar a este valor consideraram-se as 11 categorias em comum entre os dois trabalhos.

destes defeitos, examinando os comentários feitos pelos desenvolvedores, estava relacionada a funcionalidades incompletas, ou seja, defeito do tipo omissão. Como conseqüência, é possível notar, por meio das métricas na Tabela 8, que a correção dos defeitos aumentou o tamanho do código e sua complexidade. Observando, agora, o efeito das sugestões de refatoração do tipo melhoria, percebe-se (totalização da Tabela 8) que o resultado foi inverso ao dos ajustes dos defeitos. Nesse caso, a refatoração representou uma redução do tamanho do código, de sua complexidade e acoplamento (neste último, considerando as métricas *fan-in*, métodos remotos invocados e *fan-out*).

Tabela 8 - Diferença das métricas de código antes e depois da refatoração

Categoria	Refatorações Sugeridas	Complexidade Ciclomática	Fan -In	Fan-Out	Linhas de Código	Métodos Remotos Invocados
Defeito	6	5	5	7	60	7
Algoritmo ruim	7 (17,9%)	-3	-1	-3	-14	1
Organização interna ruim	7 (17,9%)	-1	-11	-5	-1	2
Pequenos problemas de estrutura	7 (17,9%)	-4	-1	-5	-9	0
Baixa Coesão	6 (15,4%)	-1	1	6	-4	6
Código duplicado	3 (7,7%)	-8	-3	-23	-39	-21
Excesso de variáveis temporárias	2 (5,1%)	0	0	1	-2	0
Código não utilizado	1 (2,5%)	0	0	0	0	0
Legibilidade ruim	1 (2,5%)	0	0	0	1	0
Nome de método ruim	1 (2,5%)	0	0	0	0	0
Identação inadequada	1 (2,5%)	0	0	-1	18	0
Poucos comentários	1 (2,5%)	0	0	0	7	0
Organização de parâmetros ruim	1 (2,5%)	0	0	0	2	0
Método longo ou extrair método	1 (2,5%)	-21	-3	-57	-157	-57
TOTAIS (exceto categoria Defeito):	**39 (100%)**	**-38**	**-18**	**-87**	**-198**	**-69**

Para analisar este resultado da Tabela 8 com um nível de detalhe maior, o efeito sobre as métricas de código foi normalizado e a sua variação descrita através de percentuais. Os percentuais foram calculados da seguinte forma: o valor da variação da métrica dividido pelo valor da métrica antes da refatoração (ex: para uma redução de 4 linhas de código em um método com 20 linhas teríamos uma variação de -20%). Então, para cada categoria, os dados foram explorados através de *boxplots* criadas para cada métrica. Apenas as categorias mais freqüentes foram analisadas. Em cada *boxplot* os possíveis casos atípicos (*outliers*) foram mantidos para enriquecer a análise, já que não se têm muitos pontos para compor o *boxplot*.

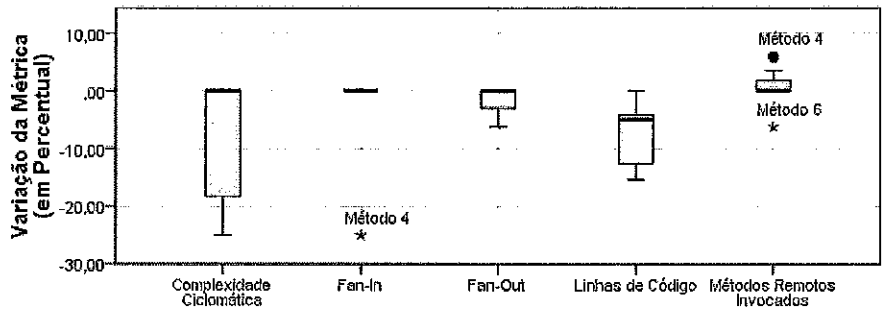


Figura 7 - Análise da Variação das Métricas para Categoria Algoritmo Ruim

Começando pela Figura 7, correspondente a categoria algoritmo ruim, observa-se que as métricas que mais sofreram efeito nesta categoria foram complexidade ciclomática e linhas de código. Isto sugere que a visão de algoritmo ruim por parte dos desenvolvedores esteve associada à questão do uso incorreto ou desnecessário de comandos de controle, o que é coerente com a concepção de um algoritmo que pode ser melhorado. Ilustrando de maneira expressiva esta visão, um dos desenvolvedores descreveu sua sugestão da seguinte forma: "Somente o primeiro 'if' é necessário, pois caso a busca não se dê pelo CPF, o 'else' deste 'if' cria a lista adequadamente pela instrução da linha 21". A instrução da linha 21 referida pelo desenvolvedor é um construtor de uma lista de clientes (ClientePessoaFisicaList). As listas normalmente possuem um construtor com vários parâmetros correspondentes aos diferentes atributos da entidade os quais são utilizados como filtro para uma busca no banco de dados – esses construtores possuem uma propriedade de que cada parâmetro só é utilizado quando um valor é passado (diferente de nulo). O 'if' utilizado pelo desenvolvedor estava verificando, desnecessariamente, se as variáveis que estavam sendo passadas como parâmetros eram nulas ou não para, então, decidir como iria realizar a busca. Desta forma, a propriedade dos construtores de lista descrita acima provavelmente não era de conhecimento do desenvolvedor que escreveu o código. Esta propriedade pode ser vista como um padrão tácito ao qual o desenvolvedor não teve acesso.

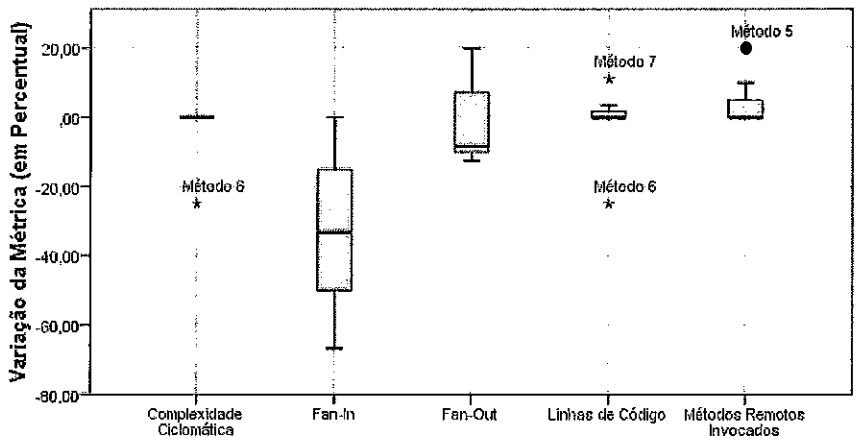


Figura 8 - Análise da Variação das Métricas para Categoria Organização Interna Ruim

A Figura 8 mostra que a métrica mais afetada pela refatoração na categoria organização interna ruim foi a *fan-in.* Esta métrica refere-se ao número de variáveis de entrada que um método usa (o que inclui parâmetros de função e atributos de classe). De fato, mais da metade das sugestões nessa categoria envolveu a recomendação do uso correto do escopo de variáveis onde, por exemplo, um desenvolvedor comentou "a variável de referência 'formBuscar' tem escopo pertinente apenas a este método, logo poderia ser declarada dentro deste método. A alteração é muito simples e poderia ser feita imediatamente." Outras sugestões incluíram inicialização de variáveis no construtor e o uso indevido de texto estático no corpo do método – existe um arquivo específico no projeto que deve ser utilizado para qualquer texto. A utilização incorreta do escopo de variáveis representa mais uma questão de boa prática de programação do que um padrão tácito do projeto, mas a utilização do arquivo, chamado no projeto de "resources", para os textos estáticos deveria ser documentada já que parece que os desenvolvedores não estavam familiarizados com este padrão tácito.

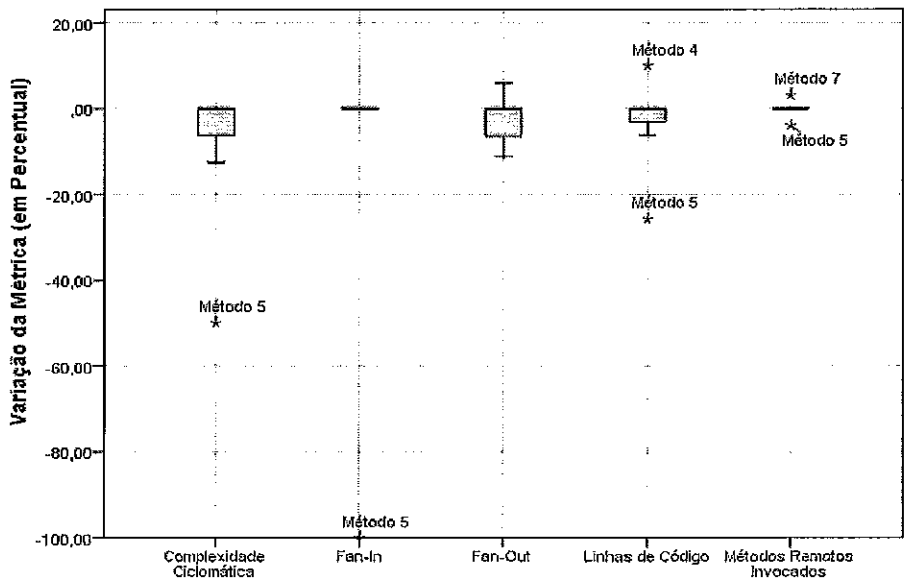


Figura 9 - Análise da Variação das métricas para categoria pequenos problemas de estrutura

A categoria pequenos problemas de estrutura incluiu questões gerais com a organização do código como, por exemplo, atribuição desnecessária de valor a variáveis, nas palavras do desenvolvedor "a atribuição a nulo é desnecessária neste caso" ou "não é adequado instanciar uma (classe) parcela sem qualquer informação e depois invocar seguidamente vários métodos 'set', o correto é criar um construtor que receba os parâmetros necessários". Em outro caso, o desenvolvedor comenta "a declaração 'if' é desnecessária, basta fazer o *return* e considerar logo a expressão condicional do 'if' apropriadamente" sobre o fato de se utilizar o comando 'if' para verificar, desnecessariamente, uma condição e retornar 'true' ou 'false' caso esta condição seja verdadeira ou falsa. Por ter um caráter mais genérico esta categoria possui o maior número de casos atípicos, tendo um total de seis casos em quatro métodos diferentes. Além disto, praticamente não possui nenhuma variação de métrica significativa (Figura 9). Ainda assim é possível perceber, novamente, tanto sugestões de refatoração mais preocupadas com boas práticas de programação, como o caso da atribuição desnecessária de valor nulo, quanto sugestões relacionadas a padrões tácitos do projeto como no caso da sugestão da criação do construtor.

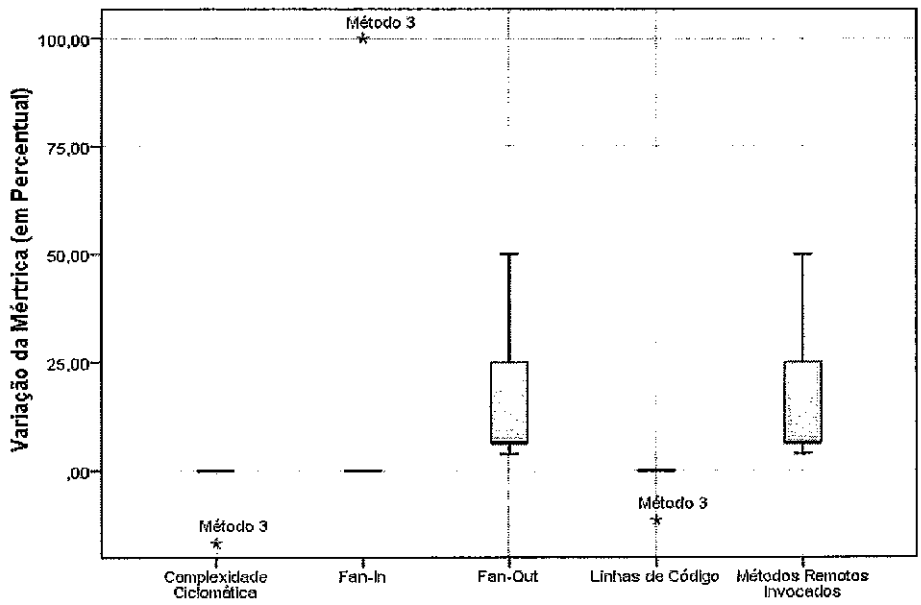


Figura 10 - Análise da variação das métricas para categoria baixa coesão

A categoria baixa coesão surgiu particularmente no contexto deste estudo já que não foi trazida de Mäntylä e Lassenius (2006). Dado o objetivo deste estudo – explorar a refatoração de código para explicitar padrões tácitos do projeto – esta categoria aparece como mais uma indicação de que a refatoração poderá revelar estes padrões, uma vez que a coesão de um método está associada ao quão relacionados estão as partes internas do método entre si e ao seu propósito geral. Neste sentido, a maior parte das sugestões de refatoração desta categoria girou em torno de violações do estilo arquitetural MVC. O seguinte comentário de um desenvolvedor ilustra este cenário: "o valor total das parcelas não deve ser somado pela classe de formulário. Isto deve ser de responsabilidade da classe 'ParcelaList'". A classe de formulário, referida pelo desenvolvedor, é uma classe da camada visão, responsável por manter em memória os valores preenchidos nos formulários. A Figura 10 mostra um acréscimo nas métricas *fan-out* e métodos remotos invocado. Praticamente nenhuma alteração no número de linhas de código foi registrada, mas um exame das refatorações mostrou que as linhas de código que estavam inadequadas foram substituídas por chamadas a métodos de outras classes. Como conseqüência, uma maior coesão foi atingida na medida em que os métodos deixaram de realizar funções que não eram de sua responsabilidade.

3.4.1.1. Fatores que Explicam as Sugestões de Refatoração

A análise da variação das métricas apresentada acima detalha o efeito sobre as métricas após a refatoração. Seguindo a abordagem de análise realizada em

Mäntylä (2005), esta seção investigará quais fatores podem explicar as decisões para a refatoração. Desta forma, características da estrutura do código fonte de antes da refatoração, representadas pelas métricas de código, serão exploradas para tentar explicar quais delas podem ter influenciado a decisão do desenvolvedor ao sugerir uma refatoração.

Análise de regressão foi utilizada para estudar como estas características afetaram as decisões. O método de regressão categórica foi empregado nesta análise e foi o mesmo utilizado em Mäntylä (2005). Segundo o autor, a regressão categórica possui vantagens sobre os métodos tradicionais como a habilidade de criar modelos de regressão combinados onde diferentes tipos de variáveis independentes podem existir simultaneamente e ainda a variável dependente pode estar em qualquer uma das três escalas nominal, ordinal ou razão. Além disto, ainda segundo os autores, a regressão categórica tem melhor desempenho em uma massa de dados que possui um número limitado de observações, várias variáveis e muitos valores diferentes por variável. Devido a estas características, a regressão categórica foi utilizada para conduzir esta análise. O nível de erro (*alpha*) utilizado para a análise de dados foi de 5%.

Para cada categoria analisada na seção anterior, um modelo de regressão foi criado. As métricas de código foram utilizadas como variáveis independentes e a decisão pela refatoração de cada método considerada como variável dependente (em escala nominal, "sim" ou "não"). Os métodos que não receberam nenhuma sugestão de refatoração foram considerados como "não" (total de 10 métodos) e aqueles que receberam foram considerados como "sim" (o total depende de cada categoria, ficando na faixa de 6-7 métodos). Além disto, métodos classificados exclusivamente como defeito foram utilizados para criação dos modelos de regressão como "não" na variável dependente. Isto porque o foco da refatoração não é a identificação de defeitos e, caso os defeitos não estivessem presentes, estes métodos não teriam nenhuma sugestão de refatoração de melhoria.

Tabela 9 - Modelo de regressão para Categoria Algoritmo Ruim

Adjusted R Square:	0,888	p-value:	0,018
Preditores do Modelo			
Preditores	Standardized Beta	p-value	Coeficiente de Correlação
Complexidade Ciclomática	-5,909	0,037	-0,847
Fan-In	-4,239	0,026	-0,828
Fan-Out	4,324	0,078	0,904
Linhas de Código	7,576	0,022	0,855
Métodos Remotos Invocados	-4,154	0,058	0,910

A Tabela 9 mostra que o modelo de regressão da categoria Algoritmo Ruim explica 88,8% das sugestões de refatoração. A partir da tabela é possível verificar que o preditor mais importante deste modelo é Linhas de Código seguido de Complexidade Ciclomática (estamos considerando o módulo do valor *standardized beta*, conforme recomendação do manual da ferramenta SPSS quando existem variáveis com escala nominal no modelo – da mesma forma, o módulo do coeficiente de correlação define a proporção de variação da variável dependente desconsiderando a influência entre as variáveis preditoras). Ainda que o modelo aponte também uma influência da métrica fan-in, ele está coerente com a Figura 7 a qual mostra que as mesmas métricas foram as mais afetadas na análise da variação após a refatoração.

Tabela 10 - Modelo de regressão para categoria Organização Interna Ruim

Adjusted R Square:	0,636	p-value:	0,047
	Preditores do Modelo		
Preditores	Standardized Beta	p-value	Coeficiente de Correlação
Complexidade Ciclomática	-0,998	0,783	-0,641
Fan-In	0,190	0,831	0,171
Fan-Out	3,160	0,091	0,603
Linhas de Código	2,666	0,337	0,727
Métodos Remotos Invocados	-5,115	0,012	-0,752

O modelo da Tabela 10 compreende 63,6% das sugestões de refatoração. O preditor mais importante é Métodos Remotos Invocados. Apesar do modelo apontar esta métrica como a mais influente nas sugestões dos desenvolvedores, a métrica mais afetada pela refatoração, segundo a análise da variação das métricas na Figura 8, foi a *Fan-In*. Para tentar identificar a origem desta diferença, o primeiro passo foi verificar que o fato do preditor *Fan-In* não influenciar no modelo da Tabela 10 seria um indício de que os métodos não refatorados e aqueles que foram refatorados deveriam possuir as mesmas características em termos dos valores da métrica *Fan-In*. Feito isto, na busca do porquê os valores desta métrica eram semelhantes, verificou-se que um dos desenvolvedores, em alguns casos, não identificou como problema métodos que faziam uso de variáveis com escopo maior do que o necessário – no caso, maiores que o escopo do método avaliado. Isto parece ter sido a origem da diferença entre o modelo da Tabela 10 e a análise da Figura 8. Ainda assim, o modelo conseguiu capturar a influência da métrica Métodos Remotos Invocados também identificado na Figura 8 ainda que em maior intensidade.

Tabela 11 - Modelo de regressão para Categoria Pequenos Problemas de Estrutura

Adjusted R Square:	-0,065	p-value:	0,581
	Preditores do Modelo		
Preditores	Standardized Beta	p-value	Coeficiente de Correlação
Complexidade Ciclomática	-0,724	0,939	-0,394
Fan-In	-0,747	0,714	-0,465
Fan-Out	1,156	0,916	0,480
Linhas de Código	1,867	0,807	0,551
Métodos Remotos Invocados	-1,694	0,838	-0,535

Não foi possível criar um modelo estatisticamente significativo para a categoria Pequenos Problemas de Estrutura (Tabela 11). De fato isto não chega a ser surpresa, já que esta categoria contabilizou o maior número de casos atípicos na análise de variação das métricas na Figura 9, devido a motivos já mencionados na seção anterior. Então, de certa forma, este modelo está coerente com a análise da Figura 9 na medida em que não é possível extrair conclusões precisas em nenhum dos dois casos.

Tabela 12 - Modelo de regressão para Categoria Baixa Coesão

Adjusted R Square:	0,676	p-value:	0,021
	Preditores do Modelo		
Preditores	Standardized Beta	p-value	Coeficiente de Correlação
Complexidade Ciclomática	0,581	0,916	0,421
Fan-In	-0,213	0,941	-0,191
Fan-Out	-4,401	0,033	-0,538
Linhas de Código	-1,417	0,605	-0,729
Métodos Remotos Invocados	5,215	0,020	0,596

As métricas mais importantes para as sugestões dos desenvolvedores relativas à categoria Baixa Coesão foram Métodos Remotos Invocados e *Fan-Out* (Tabela 12). O modelo é capaz de responder por 67,6% das necessidades de refatoração da categoria e está consoante com a análise da variação das métricas da Figura 10.

Como comentário final à análise, foi possível perceber diante da análise métricas de antes da refatoração (análise de regressão) que as decisões dos desenvolvedores pela refatoração ou não de um determinado método foram, de maneira geral, consistentes com as métricas afetadas após as modificações terem sido feitas. Desta forma, como as métricas de código estão representando características estruturais do código (seção 3.2.2.3), é possível observar que as mesmas características que influenciaram as decisões dos desenvolvedores foram aquelas que de fato foram afetadas após a realização das refatorações.

Diante deste cenário, e para entender melhor como a refatoração foi executada pelos desenvolvedores, a próxima seção trás uma análise qualitativa das entrevistas conduzidas após as atividades de refatoração.

3.4.2.Percepção sobre a Qualidade do Código e Aprendizado – Análise Qualitativa

Esta seção trás uma análise qualitativa do relato dos desenvolvedores durante as entrevistas semi-estruturadas (seção 3.2.4.2). Dado os objetivos do estudo, avaliou-se que as questões iii, iv, vi, vii e viii das entrevistas, seriam as mais significativas na medida em que estão diretamente alinhadas aos resultados esperados (questões Q.1 e Q.2).

A análise será conduzida por meio da utilização da abordagem *Grounded Theory* (Strauss e Corbin, 1990), com adaptações introduzidas por Baskerville e Pries-Heje (1999). Segundo Baskerville e Pries-Heje (1999), a pesquisa-ação normalmente modifica o papel dos elementos da *grounded theory* na medida em que tipicamente traz categorias pré-definidas, incluindo, possivelmente, a categoria central, as quais são obtidas a partir da definição do tema de pesquisa (seção 3.1.3) e da revisão inicial da literatura (seção 3.2.1). Desta forma, o caráter de emersão fundamentada de uma teoria não é genuinamente alcançado de acordo com os princípios da *grounded theory* canônica (Strauss e Corbin, 1990). Por isso, a chamada *grounded action research* (Baskerville e Pries-Heje, 1999) seleciona os componentes da *grounded theory* de acordo com os objetivos do estudo focando na codificação aberta e axial. A Figura 11 apresenta uma visão geral do processo de codificação canônico e sua aplicação neste estudo é detalhada no Anexo A.1. É importante destacar que este estudo vai até a etapa de codificação axial (representada pelas categorias na Figura 11).

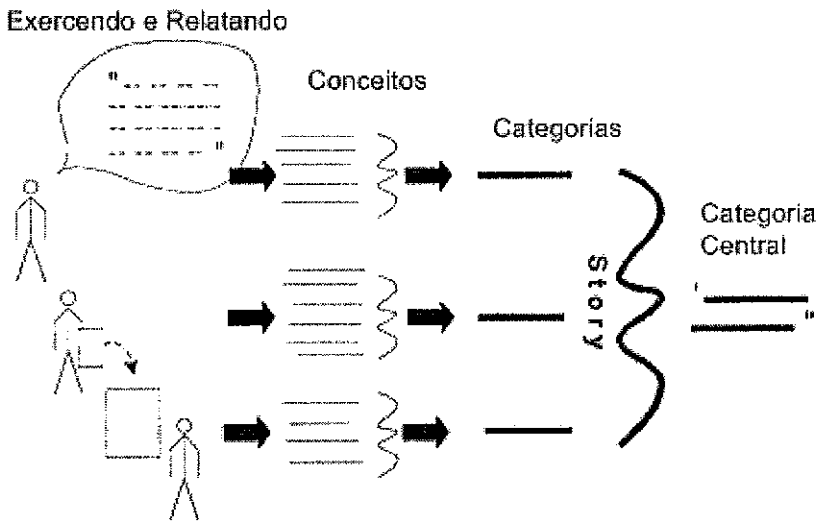


Figura 11 - Visão geral do processo de codificação (Baskerville e Pries-Heje, 1999)

A idéia principal da codificação aberta é revelar os conceitos essenciais embutidos nos dados – neste caso, nos depoimentos feitos pelos participantes. Com isto, as observações, que se encontram no ponto de vista dos participantes, são decompostas e discretizadas em eventos ou idéias as quais recebem rótulos (códigos) (Baskerville e Pries-Heje, 1999). Um segundo objetivo da codificação aberta é a categorização dos conceitos, sintetizando-os e ajudando à compreensão do fenômeno estudado.

Após este processo de codificação aberta, obteve-se o resultado apresentado na Tabela 13. Cada categoria foi relacionada às questões de pesquisa definidas na seção 3.2.2.2 como forma de representar suas repostas. A associação entre as categorias e as questões pode ser entendida como a codificação axial da *grounded theory* onde o relacionamento entre as categorias é dado pela questão que elas tratam. Por fim, a coluna "argumento" representa a ligação entre a categoria e os conceitos/códigos descobertos nos dados subjacentes (os conceitos/código identificados estão no Anexo A).

Do ponto de vista da validade de construção deste resultado, é importante mencionar que a partir das questões de pesquisa (seção 3.2.2.2) foram definidas questões práticas (seção 3.2.2.3) às quais algumas questões das entrevistas semi-estruturadas (Tabela 5 e Tabela 6) estavam associadas. Desta forma, observa-se, então, o rastro desde as questões de pesquisa até as transcrições utilizadas para a análise da *grounded theory* e, com isto, demonstra-se objetivamente como o resultado foi alcançado.

Tabela 13 – Categorias da Análise Qualitativa (Grounded Theory)

Id.	Questão	Categoria	Argumento
I1	-	O processo de refatoração de código é análogo ao de inspeção de código e de fácil aplicação quando realizado sob a perspectiva da dinâmica de execução do sistema segundo o modelo de casos de uso.	Desenvolvedores, quando habituados a utilizar este artefato, podem explorá-lo para a leitura mais eficiente de um código OO (Dunsmore et al., 2003).
I2	-	O processo de refatoração tem como benefício o aprendizado em relação aos padrões de defeitos que podem ser documentados por meio da análise dos resultados.	Diversos estudos apontam o aprendizado como um dos resultados da utilização de refatoração.
C2	Q.1	Os padrões tácitos de codificação estão mais presentes na arquitetura da solução e estruturação do código, representando um conhecimento essencial ao entendimento do código.	Segundo os desenolvedores, o conhecimento sobre o estilo arquitetural e padrões de codificação utilizados no projeto permitem saber de antemão onde encontrar a implementação das funcionalidades que em um código OO encontra-se distribuída em diferentes classes e pacotes.
C3	Q.1	Os desenvolvedores experientes conhecem e têm ciência da existência dos padrões tácitos do projeto e, por isso, podem identificar quando são negligenciados.	Os desenvolvedores que tiveram seu código refatorado salientaram que perceberam de forma nítida, com a ajuda das sugestões de refatoração, uma organização uniforme do código à qual, em muitos casos, não tinham atentado antes.
C4	Q.2	O aprendizado girou em torno da evidenciação dos motivos (tácitos) por trás da organização do código e da internalização dos padrões de codificação utilizados no projeto.	Resultado direto da utilização da refatoração: as descrições das sugestões de refatoração trouxeram as razões pelas quais o código deveria ser organizado de maneira específica e os participantes percebem os padrões de defeitos (neste caso, refatoração).
C5	Q.2	O conhecimento sobre o estilo arquitetural e padrões de codificação utilizados no projeto pode ser explicitado na forma de diretivas que guiarão os desenvolvedores a seguir um modelo único de desenvolvimento.	Os participantes entenderam que diretivas seriam a melhor maneira de guiar os desenvolvedores iniciantes de maneira objetiva, pontual e focada.
C6	Q.2	Todo o aprendizado deve ser documentado, desde que categorizado de forma a facilitar a busca e leitura das diretivas.	Como o conjunto de diretivas pode se tornar grande ao longo do tempo, a categorização ajudará a manter o documento focado no propósito necessário em um determinado contexto. (ex: diretivas relacionadas à nomenclatura em classes de domínio)

A tabela acima é auto-explicativa e não exige uma descrição detalhada. Alguns comentários são, no entanto, apropriados. O primeiro diz respeito ao fato de que as categorias da Q.2 foram fundamentais à condução da etapa de aprendizado deste estudo, na medida em que foram a base para elaboração da

documentação que servirá de apoio ao aprendizado da própria equipe em relação aos padrões, antes tácitos, de codificação – maiores detalhes sobre o aprendizado na seção 3.5.1. Além disto, a análise qualitativa, e conseqüentemente as categorias de maneira geral, permitiram explorar as diferentes camadas de significado construídas pelas diferentes ações conduzidas de forma a servir de entrada à construção de teorias locais (seção 3.5.2.1) (Sjøberg *et al.*, 2008).

3.5. Reflexões e Aprendizado

Esta seção explora os resultados deste estudo no sentido de condensá-los em algum aprendizado ao projeto, assim como examiná-los com objetivo de organizar e refletir sobre o conhecimento depreendido das ações.

3.5.1. Aprendizado

Após a entrevista semi-estruturada e a sua análise, definiu-se como o conhecimento tácito sobre o estilo arquitetural e padrões de codificação poderia ser externalizado. A externalização deste conhecimento representa um dos principais elementos de aprendizado no contexto deste estudo, na medida em que permitirá à equipe remota e futuros novos integrantes a se familiarizarem com os padrões adotados no projeto.

A externalização deu-se por meio da elaboração de diretivas a partir dos problemas identificados durante a refatoração. A idéia das diretivas é informar sobre a forma de estruturação do código e do projeto, onde cada uma cuida de um aspecto em particular. Além das diretivas, os participantes decidiram por um conjunto de informações que deveriam ser agregadas a cada diretiva a fim de torná-la prática ao uso. O conjunto de informações definido foi:

- Diretiva: descrição genérica (aplicável a diferentes contextos) de uma regra para organização do código, definindo um padrão de codificação.
- Motivação/Exemplo: fundamentação ou exemplificação da forma como a diretiva deve ser aplicada.
- Argumentos: explicação do porque a diretiva deve ser utilizada.
- Tipo: categoria da diretiva (relacionada à categorização da refatoração, porém sem o enfoque "negativo", ex.: legibilidade ruim passou a ser chamado de legibilidade.)
- Grau de impacto: avaliação subjetiva do impacto à qualidade do código caso a diretiva não seja aplicada. Três valores podem ser assumidos: 1 – baixo, 2 – médio e 3 – alto.

- Classes afetadas: tipos das classes onde as diretivas podem ser mais comumente aplicadas. Ex: classes "controller", de domínio ou utilitárias.

Um grupo inicial de diretivas foi definido pelo pesquisador e a partir daí evoluído por todos aqueles envolvidos no processo de refatorção. Houve várias sugestões de melhoria a este grupo inicial. Foram sugeridos novos tipos, exemplos, alterações das descrições, dentre outras alterações. Do total das 45 necessidades de refatoração detectadas na refatoração (incluindo defeitos), 43 foram utilizadas para definir 23 diretivas. A relação entre diretiva e necessidade de refatoração, como pode ser observado pelos números, não foi de um para um, pois algumas diretivas englobaram mais de uma necessidade de refatoração identificadas.

Após a definição e evolução deste grupo inicial, as diretivas foram disponibilizadas em um Wiki utilizado pelo próprio projeto, o Trac (Trac, 2003). O Trac, através de um plugin, permite a associação de categorias ("tags") as quais podem ser indexadas e ser alvo de buscas. O Wiki será uma tecnologia útil neste contexto, pois possibilitará a evolução contínua das diretivas pela equipe do projeto – lembrando, ainda, que apenas uma parte do código do projeto foi inspecionada, logo ainda devem surgir novas diretivas. Como o número de diretivas é relativamente grande a Tabela 14 apresenta apenas alguns exemplos e a Figura 12 uma diretiva cadastrada no Trac.

Tabela 14 – Exemplos de Diretivas

Diretiva	Motivação/Exemplo	Argumentos	Tipo	Grau de Impacto	Classes Afetadas

Métodos estáticos devem ser acessados estaticamente, isto é, sem instanciar a classe que os contém.	Incorreto:
	`Classe c = new NomeClasse();`
	`c.metodoInvocado();`
	
	Correto:
	`NomeClasse.metodoInvocado();`
		Não existe necessidade da criação da instância da classe para invocar um método estático. Além disso, um espaço na memória estaria sendo desnecessariamente ocupado.	Estrutura algorítmica	2	Todos

Verificações de segurança (acesso à facilidades) devem verificar se o perfil ao qual o usuário autenticado no sistema possui a funcionalidade desejada.	Verificações de acesso estavam verificando apenas o perfil do usuário (e não as funcionalidades associadas ao perfil). Existem perfis padrão no sistema que refletem as permissões dos atores do sistema – especificado no modelo de casos de uso. Este modelo define quais atores têm acesso às funcionalidades. A semelhança entre perfil e ator no contexto do projeto levou ao não entendimento destes conceitos.	O módulo de gestão de usuários prevê a manutenção de perfis, onde cada perfil pode ter um conjunto de funcionalidades às quais os usuários associados a este perfil podem ter acesso. Desta forma, diferentes perfis podem ser criados e terem acesso a uma mesma funcionalidade. Por isso, a verificação de acesso deve ser feita em cima das funcionalidades do perfil, e não em cima dos próprios perfis padrão.	Estruturação do Projeto	3	Controller /Domínio
Sempre que possível (e necessário) deve-se utilizar expressões booleanas como resultado de um método.	`return !(curso.isStatusEncerrado());`	Neste caso, a redução de linhas de código (caso se utilizasse a construção if/else) melhora a legibilidade.	Legibilidade	2	Todos
Todo texto estático deve ser declarado no arquivo UIResources.	`addErrorMessage(getResourceMessage("lancarRecebimentoParcela.valorMaiorParcela"), "iptValorRecebido")`	A utilização do UIResrouces permite a modificação de texto sem a necessidade de se recompilar o código, contribuindo, assim, para a manutenabilidade do projeto.	Estruturação do Projeto	2	Todos
O nome do método que realiza a validação dos dados de entrada do 'form' deve ser "validaDados".	`public Boolean validaDados()`	Regras de nomenclatura ajudam a ler e entender o código fonte, na medida em que uniformizam a sua apresenação.	Regra de Nomenclatura	1	Validator

Sempre que for necessário persistir mais de uma entidade de "uma só vez" (única transação) ou que uma classe que esteja sendo persistida for responsável por persistir outras classes associadas a ela, é necessário utilizar o gerenciador de transação.	`GerenteTransacaoBD gtBD = new GerenteTransacaoBD(new ITransaction[] {proposta, projeto});`
	`gtBD.iniciarTransacao();`
	`try {`
	`  proposta.save();`
	`  projeto.save();`
	`} catch (Exception e) {`
	`  gtBD.recuperarTransacao();`
	`  throw(e);`
	`}`
	`gtBD.finalizarTransacao();`
		Caso não haja controle e recuperação das transações em caso de falha dados inconsistentes poderão ser persistidos indevidamente.	Estruturação do Projeto	3	Controller

Figura 12 - Cadastro de diretivas no Wiki do projeto

Logo após o fechamento do primeiro conjunto de diretivas, os profissionais recém integrados a equipe foram recomendados a ler as diretivas. Um outro ponto interessante é que existe a previsão de que o projeto não seja mantido pela mesma equipe e, desta forma, há uma expectativa de que esta documentação seja útil a futura equipe de manutenção do projeto. Ainda assim, uma avaliação da utilidade deste tipo de documentação foge ao escopo deste estudo e deve ser avaliada em trabalhos futuros.

3.5.2.Reflexões

A principal base para a condução deste estudo foi o trabalho de Mäntylä e Lassenius (2006) e Mäntylä (2005), pois mostraram como o estudo poderia ser planejado e organizado para capturar e analisar os dados quantitativos (métricas de código) e qualitativos (descrição das sugestões de refatoração). Desta forma, cabe comparar, ou ao menos interpretar comparativamente já que este estudo não é estritamente uma repetição, os resultados obtidos. Um primeiro ponto a ser comentado é que a categorização, utilizada para

identificar os principais fatores para refatoração, coincidiu em grande parte nos dois estudos, ou seja, apenas 2 novas categorias surgiram (dentro do universo de 22 categorias identificadas em Mäntylä e Lassenius (2006)). Isto fortalece a categorização concebida – indicando que estas categorias representam apropriadamente as decisões de refatoração –, mas ao mesmo tempo indica a necessidade de outros estudos para verificar a possibilidade de novas categorias. Além disto, 11 categorias não foram identificadas neste trabalho, o que indica que a experiência dos desenvolvedores pode ter contribuído para este comportamento.

Os modelos de regressão elaborados nos dois estudos diferem em alguns aspectos e merecem ser detalhados. O modelo de regressão baseado nas métricas de código criado por Mäntylä (2005) para identificar as características do código que influenciaram as decisões de refatoração explicou apenas 30% das decisões e, na mesma direção, uma primeira tentativa em se criar o mesmo modelo no presente trabalho não gerou nenhum modelo estatisticamente significativo. No entanto, para um grupo dos participantes em Mäntylä (2005) foi dada uma lista pré-determinada de "code smells" (ex: método longo), onde questionou-se sobre a existência de cada "code smell" no código. Agrupando os modelos por "code smell", obteve-se desempenho significativamente melhor chegando a explicar mais de 70% das decisões para dois dos três "code smells". De forma análoga, após a categorização das decisões de refatoração, foi criado um modelo de regressão para algumas das categorias e, também, foi possível obter resultados significativamente melhores com modelos, explicando desde 63,6% até 88,8% em três das quatro categorias.

Ainda assim, existe um resultado conflitante com Mäntylä (2005). Neste estudo, métricas de código foram utilizadas com sucesso em um modelo de regressão como preditoras da decisão pela refatoração de código, enquanto que em Mäntylä (2005) o modelo baseado em métricas só foi estatisticamente significativo para a presença ou não de "code smells" (e não para a decisão pela refatoração de código). Um motivo para esta diferença pode residir no fato de que foi definido que os desenvolvedores deveriam ter experiência no contexto do projeto, já que existia uma expectativa de que este conhecimento prático seria útil à identificação de defeitos relacionados à desvios dos padrões de codificação e estilos arquiteturais. Desta forma, houve uma consistência

nos tipos de desvios de código identificados, o que possivelmente resultou em modelos de regressão significativamente melhores para as decisões de refatorção.

Este resultado abre espaço para a construção de ferramentas que apóiem esta tomada de decisão, mas para que isto ocorra os resultados indicam que as ferramentas precisariam ser calibradas com dados do projeto onde elas são utilizadas – dados estes que incluam algum histórico de avaliações subjetivas, conforme os modelos de regressão foram criados neste trabalho. Com isto, a sugestão de Fowler *et al.* (1999) de manter a avaliação humana na detecção de necessidades de refatoração seria mantida, ainda que indiretamente.

Existe um número expressivo de trabalhos que usam métricas para detectar a necessidade de refatoração (Mens e Touwé, 2004), mas estas tecnologias tendem a gerar muitos falso-positivos e, por isso, necessitam de uma avaliação humana em grande parte dos casos (Parnin e Görg, 2008). Desta forma, espera-se que os resultados aqui encontrados possam possibilitar a criação de ferramentas mais eficazes. De fato, o uso de modelos de regressão para diminuir o número de falso-positivos trazidos com análises baseadas em métricas de código mostrou-se útil em uma área relativamente próxima à estudada neste trabalho, detecção de defeitos. Ruthruff *et al.* (2008) mostram como defeitos reportados automaticamente por ferramentas que analisam o código estaticamente – incluindo métricas de código –, passam por uma triagem executada por meio de modelos de regressão, os quais identificam aqueles que têm mais chances de representar um defeito real. Estes modelos de regressão foram criados com bases históricas de triagens manuais.

O outro trabalho utilizado neste estudo (Dunsmore *et al.*, 2003) apoiou na definição de um procedimento sistemático para a leitura dos diferentes métodos presentes nas classes que deveriam ser refatoradas. Em ambos os estudos os desenvolvedores relataram que um ponto forte do procedimento é a possibilidade de pensar nos métodos no contexto de sua execução. Um outro aspecto importante, é o fato de que a técnica utiliza modelos de casos de uso como base a sua aplicação e, por isso, não houve necessidade de treinamento da equipe, pois esta já estava habituada à utilização deste artefato para as atividades de desenvolvimento.

Todavia, nem todos os resultados podem ser relacionados com os trabalhos utilizados no contexto deste estudo (Dunsmore *et al.*, 2003, Mäntylä, 2005, Mäntylä e Lassenius, 2006). Dois aspectos foram trabalhados de maneira diferenciada: (1) a utilização de refatoração de código como ferramenta para explicitar e documentar estilos arquiteturais e padrões de codificação e (2) a comparação dos modelos de regressão baseados nas métricas de código com o efeito sobre as métricas após a refatoração.

O primeiro aspecto foi, como dito anteriormente, o principal elemento de aprendizado deste estudo. Por isto, representa uma importante evidência de que de fato inspeções podem possibilitar a externalização e documentação de um conhecimento mantido tácito por uma equipe de desenvolvimento em relação ao estilo arquitetural e padrões de codificação. O processo de refatoração permitiu a equipe refletir sobre a sua experiência anterior e, ao descrever as necessidades de refatoração, explicitar a razão pela qual o código deveria ser organizado de uma certa maneira – ou seja, explicitar o conhecimento tácito. Além deste indício, em diferentes momentos da análise, tanto quantitativa quanto qualitativa, a idéia da existência do conhecimento tácito pôde ser observada. Por exemplo, a análise quantitativa, já considerando o segundo aspecto, mostrou que as métricas que influenciaram as decisões dos desenvolvedores foram as mesmas afetadas como resultado da realização da refatoração. Isto indica que os desenvolvedores estavam conscientes das decisões que estavam fazendo com base no seu conhecimento prévio (tácito), além de terem sido consistentes, pois decisões semelhantes afetaram um mesmo conjunto de métricas. Já a análise qualitativa, além de permitir a definição de como o aprendizado deveria ser formatado, revelou que, no contexto de uma equipe de desenvolvimento, o conhecimento tácito emerge e se forma no nível da arquitetura da solução e estruturação do código na medida em que é resultado do trabalho colaborativo.

3.5.2.1. Teorização

Com objetivo de permitir a organização e estruturação dos fatos e conhecimento relacionados a este estudo de forma sistemática e precisa, uma teoria sobre os conceitos tratados neste trabalho será criada. O uso de teorias em engenharia de software ainda não é comum, mas dada a riqueza das observações proporcionadas pela interpretação das ações, a construção de teoria pode facilitar a comunicação das principais idéias e conhecimento

gerados (Sjøberg *et al.*, 2008). Além disto, a elaboração de teorias é uma estratégia interessante para facilitar a agregação deste estudo em estudos secundários (Charters *et al.*, 2009), pois define detalhadamente a aplicabilidade e limitações da tecnologia investigada. Existem três níveis de sofisticação ou complexidade de teorias (Sjøberg *et al.*, 2008):

- Nível 1: pequenos relacionamentos estáveis concretos e baseados diretamente em observações.
- Nível 2: teorias de alcance médio que possuem algum grau de abstração, mas ainda estão intimamente ligadas às observações.
- Nível 3: teorias gerais que procuram explicar fenômenos no contexto de Engenharia de Software.

Estes níveis estabelecem marcos na geração de teorias, mas podem representar também teorias completas por si só. Para Sjøberg *et al.* (2008), o desenvolvimento de teorias em Engenharia de Software deve focar inicialmente nos níveis um e dois. A teoria elaborada nesta seção tem um perfil mais próximo do nível um.

Sjøberg *et al.* (2008) sugerem que a descrição de teorias deve ser dividida em quatro partes: constructos (quais são os elementos básicos), proposições (como os constructos se relacionam), explanações (porque as proposições foram especificadas) e o escopo (qual é o universo de discurso no qual a teoria é aplicável). Neste sentido, a construção de teorias deve envolver 5 passos: (1) definição dos constructos, (2) definição das proposições, (3) provimento de explanações para justificar a teoria, (4) determinação do escopo da teoria e (5) teste da teoria por meio de estudos experimentais. Ainda segundo Sjøberg *et al.* (2008), o uso da *grounded theory* facilita a identificação dos principais conceitos (constructos) e dos seus relacionamentos (proposições e explanações). Como consequência, os constructos que serão apresentados adiante foram baseados, principalmente, na análise *grounded theory* conduzida neste estudo.

A Figura 13 esquematiza a teoria graficamente (e a Tabela 15 apresenta os elementos da teoria em detalhes). A semântica da notação utilizada foi definida em Sjøberg *et al.* (2008) e é descrita a seguir. Um constructo é representado como uma classe ou um atributo de uma classe. Uma classe é desenhada como uma caixa e o seu nome escrito no topo como, por exemplo, "Projeto distribuído". Uma classe pode ter uma subclasse (utilizando a seta de

generalização da UML) ou uma classe componente (desenhado como uma caixa dentro de outra caixa como, por exemplo, "Código fonte"). Normalmente, se um constructo é um valor em particular de uma variável, então o constructo é modelado como uma subclasse ou classe-componente. Por outro lado, se o foco é na variação de valores, então o constructo é uma variaval que é modelada como um atributo de uma classe como, por exemplo, "Esforço". Um atributo é escrito como um texto na parte inferior de uma caixa de uma classe (abaixo da linha horizontal). Todos os constructos estão sublinhados na Figura 13.

Um relacionamento é modelado com uma seta; uma seta de *A* para *B* indica que *A* afeta *B*, onde *A* é uma classe ou um atributo e *B* é um atributo. Em um relacionamento, *B* pode ser ainda um relacionamento em si, representado por uma seta. Neste caso, *A* é chamado de moderador, como no caso de "Experiência" na Figura 13. Isto significa que *A* afeta a direção e/ou a intensidade do efeito do relacionamento *B*. Os moderadores são, então, definidos como proposições (ver Tabela 15 proposição P7).

A base de onde todas as classes herdam são chamadas de classes arquétipos e são: ator, tecnologia, atividade e sistema de software. Segundo Sjøberg *et al.* (2008), a situação típica em engenharia de software se resume a um *ator* que aplica uma *tecnologia* para desempenhar algumas atividades em um *sistema de software* (planejado ou existente). Alguns exemplos para a classe-arquétipo *atividade* são criar, modificar e analisar (detalhes em Sjøberg *et al.* (2008)).

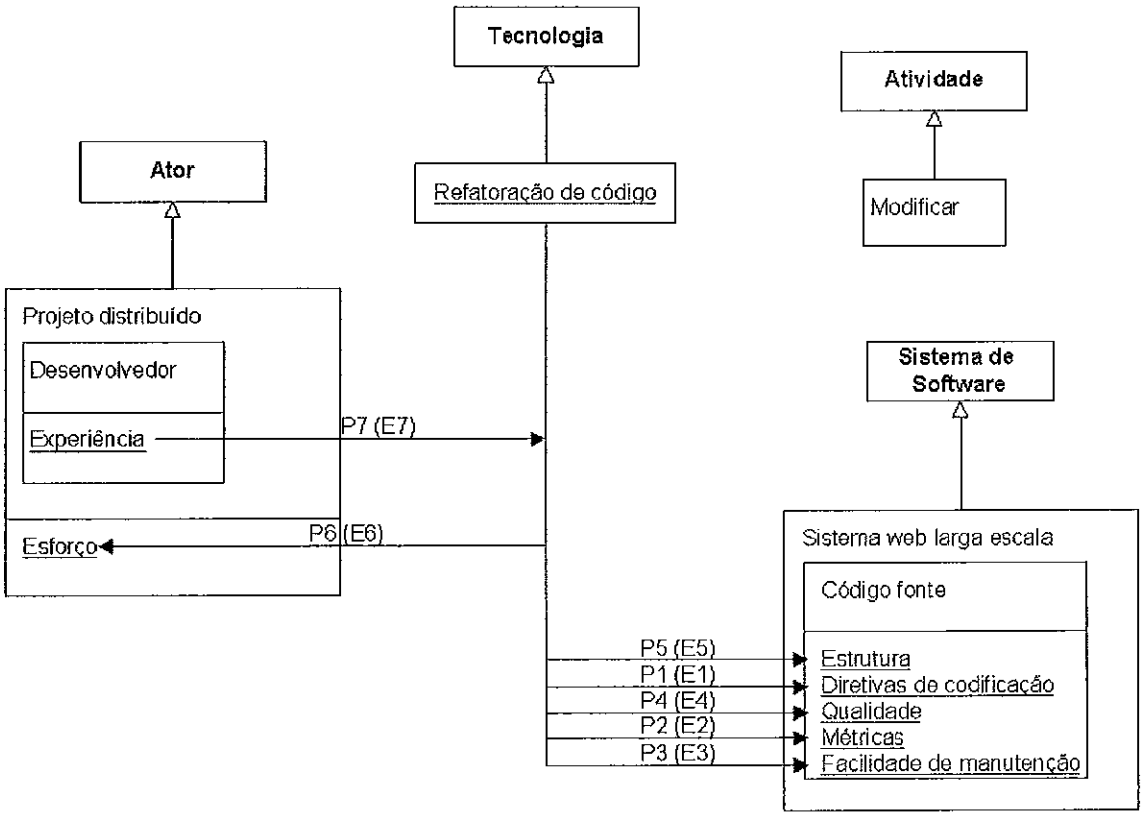


Figura 13- Teoria sobre o uso de refatoração de código para explicitar padrões de codificação em um projeto de *software* web larga escala

Tabela 15 - Elementos da teoria relacionada ao uso de refatoração de código para explicitar padrões de codificação em um projeto de *software* web larga escala

Constructos	
C1	*Refatoração de código* (baseado na técnica de leitura baseada em casos de uso [4] e registrando as sugestões de refatoração conforme [5])
C2	*Estrutura do Código* (propriedades estruturais perceptíveis do código, ex: legibilidade, estrutura algorítmica)
C3	*Qualidade do Código* (número de defeitos por linha de código)
C4	*Métricas de Código* (linhas de código, número de parâmetros, complexidade ciclomática, número de métodos remotos invocados, "fan-in" e "fan-out")
C5	*Diretivas de codificação* (padrões de codificação e estilos arquiteturais)
C6	*Facilidade de manutenção* (relacionado ao esforço necessário para atividades de manutenção do código fonte do sistema)
C7	*Experiência do desenvolvedor* (tempo de envolvimento no projeto onde o indivíduo revisará o código com objetivo de sugerir refatorações)
C8	*Esforço* (total de homens hora alocadas para atividade de projeto)
Proposições	
P1	A refatoração de código afeta positivamente as diretivas de codificação
P2	A refatoração do código afeta positivamente (reduz o valor) métricas de código
P3	A refatoração de código afeta positivamente (facilita) a manutenção do código
P4	A refatoração de código afeta positivamente a qualidade do código
P5	A refatoração de código afeta positivamente a estrutura do código
P6	A refatoração de código afeta negativamente (aumenta) o esforço da atividade de projeto
P7	Os efeitos positivos da refatoração de código são reduzidos caso o desenvolvedor não possua experiência prévia no contexto do projeto
Explanações	
E1	As diretivas de codificação podem ser construídas ou evoluídas • A descrição das sugestões de refatoração explicita a forma pela qual o código está estruturado.

- As diretivas são diretamente derivadas das sugestões de refatoração.

E2 Os valores das métricas de código se reduzem
- As medidas normalmente apresentam redução após a refatoração, principalmente, em virtude do objetivo principal da refatoração que é melhorar características do código como legibilidade e desempenho.

E3 A manutenção do código é facilitada
- As diretivas de codificação permitem entender *à priori* como o código está organizado e, por isso, facilitam o seu entendimento quando alguma atividade de manutenção é necessária.
- O código fonte tem sua estrutura mais homogeneizada podendo facilitar futuras manutenções.

E4 A qualidade do código melhora
- Desenvolvedores identificam defeitos no código (ainda que normalmente não seja o foco da refatoração de código e, sim, da inspeção de código).

E5 A estrutura do código melhora
- A estrutura do código fonte se homogeneiza ao longo de todo o projeto segundo o conhecimento prévio dos desenvolvedores
- O tamanho, complexidade e acoplamento do código diminuem

E6 O esforço do projeto aumenta
- É necessário a alocação de pessoas (homens/hora) para execução das atividades relacionadas à refatoração do código.

E7 A experiência dos desenvolvedores no projeto permite
- Capturar e depreender os padrões de codificação utilizados no projeto.
- Identificar desvios dos padrões de codificação durante a refatoração.
- Identificar defeitos em relação aos requisitos e conhecimento do domínio.
- Executar a atividade de refatoração em menor tempo.

Escopo

A teoria é supostamente aplicável em projetos com equipes remotas criando e modificando sistema de informação Web de média/larga escala em um processo de desenvolvimento incremental e baseado em atividades de VV&T.

3.6. Conclusão

Neste primeiro estudo conduzido o formato mais característico da pesquisa-ação pode ser explorado. Partiu-se de uma situação onde, a princípio, não se vislumbrava uma solução imediata e, perante a participação do pesquisador e colaboração dos participantes, foi possível avançar para um cenário satisfatório do ponto de vista dos problemas enfrentados inicialmente. Isto representa, de certa forma, uma das características essenciais da prática da engenharia, onde na falta do conhecimento científico apropriado para tratar diretamente o problema adota-se uma postura pragmática com base naquilo que "se conhece sobre o mundo".

Como resultado da assunção desta postura, uma teoria pôde ser elaborada e, apesar de restrita considerando a sua generalização, simboliza uma pequena evolução do conhecimento científico para o tema de pesquisa explorado. Neste sentido, assim como, por exemplo, a Engenharia Civil evoluiu no seu início, ainda que não se possua um entendimento formal sobre o problema – ou seja, controle e conhecimento avançados sobre as variáveis que

o influenciam – documenta-se as regras pragmáticas para os elementos recorrentes na situação enfrentada (Shaw, 1990).

Para permitir este tipo de documentação, diferentes facetas da pesquisa-ação puderam ser exploradas e, conseqüentemente, mais bem compreendidas. O uso intenso de dados qualitativos e do conhecimento tácito da prática dos profissionais da organização mostram como a aplicação da pesquisa-ação requer um planejamento e acompanhamento cuidadoso das atividades de pesquisa e da organização. Entretanto, em determinados momentos não é possível conciliar estes dois objetivos simultaneamente e, desta forma, o pesquisador deve buscar registrar os dados para posterior análise, já que a prioridade é as atividades da organização.

Um outro ponto importante que pôde ser observado na utilização da metodologia da pesquisa-ação foi a identificação de oportunidades de pesquisa – no caso, relacionada a refatoração de código fonte – em meio às atividades de uma organização real de software.

Capítulo 4
Estudo de Pesquisa-Ação #2

Neste estudo o objetivo foi avaliar a melhoria de qualidade em modelos de casos de uso por meio do uso de técnicas de inspeção de software. Uma técnica baseada em checklist é utilizada e comparada à abordagem ad-hoc. Um estudo controlado foi realizado previamente a aplicação no contexto real. Resultados foram positivos à técnica checklist em ambas as situações, mas no projeto real a diferença não foi tão expressiva quanto no estudo controlado e, além disto, uma avaliação qualitativa mostrou que os defeitos identificados não são relevantes na opinião de alguns inspetores e dos autores do artefato inspecionado. Assim como no estudo #1, o capítulo está organizado com uma seção para cada atividade etapa da pesquisa-ação de forma a narrar as atividades de pesquisa desempenhadas.

4.1. Diagnóstico

O diagnóstico encontra-se dividido em três seções que descrevem o problema, o contexto de atuação e a definição do tema de pesquisa.

4.1.1.Descrição do Problema

No contexto do desenvolvimento de software dirigido por modelos de caso de uso (diagrama e descrição), onde casos de uso são utilizados como base para planejamento do projeto, construção e comunicação com usuários, a qualidade e inteligibilidade destes modelos são fundamentais. Particularmente no escopo do desenvolvimento do *software* existem diferentes interesses. Por exemplo, é de interesse do gerente do projeto que os casos de uso abranjam todos os requisitos funcionais, ou para o projetista que a terminologia utilizada seja consistente ao longo de todo o documento (Anda e Sjøberg, 2002). Neste sentido, casos de uso representam o conhecimento daquilo que deve ser construído.

Casos de uso são intensamente utilizados ao longo de todo o projeto e, quando mal escritos, podem levar a construção de funcionalidades incorretas ou necessitar da interação e comunicação entre a equipe que ao longo do processo identifica e corrige omissões e incorreções no artefato. Uma maneira de prevenir estes problemas é a utilização de revisões ou inspeções de casos de uso.

Inspeções de artefatos de software têm sido utilizadas com sucesso na indústria como meio para evitar retrabalho e melhorar a qualidade do *software* (Fagan, 2001). Os principais fatores deste sucesso estão associados ao relativo baixo custo associado a sua utilização e a sua capacidade de encontrar defeitos tão logo são inseridos no processo. Além da detecção de defeitos e conseqüente melhoria da qualidade do *software,* existem outros benefícios associados à utilização de inspeções, dentre eles a integração entre os processos de detecção e prevenção de defeitos.

Diversos fatores podem influenciar no custo-eficiência de uma inspeção e na cobertura de defeitos encontrados. Grande parte deles tem influência da intervenção humana como, por exemplo, a experiência do inspetor ou a sua especialidade técnica no processo de desenvolvimento (analista, programador, testador dentre outros). Como conseqüência, a técnica de inspeção *ad-hoc*, onde não há nenhum tipo de controle sobre a intervenção humana, tem a sua produtividade (custo-eficiência) individualizada e não possui nenhuma garantia da cobertura do documento como um todo e dos tipos de defeitos encontrados (Porter e Votta, 1998). Com isto, pode apresentar um custo-eficiência adequado para inspetores com maior experiência, mas não para os inexperientes (Anda e Sjøberg, 2002). No entanto, mesmo com a utilização de inspetores experientes, a cobertura do documento como um todo pode ainda não ser alcançada já que cada inspetor utiliza a "sua técnica" – e, com isto, determinados tipos de defeitos podem ainda permanecer no documento (Figura 14).

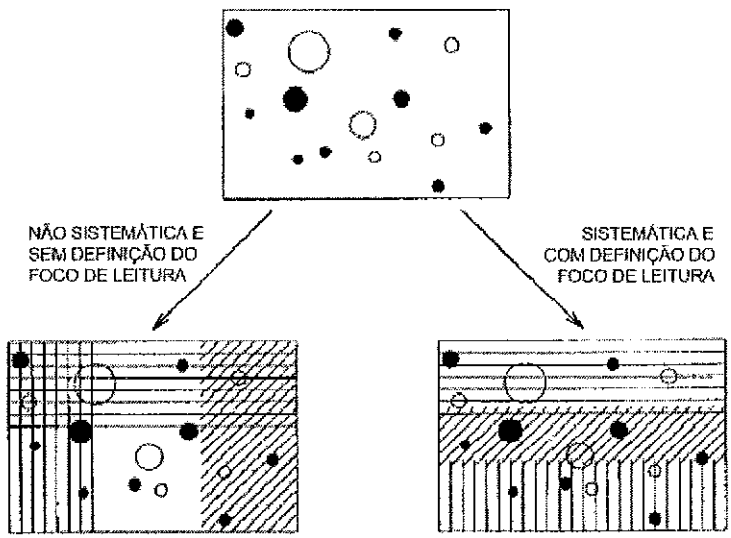


Figura 14 – Cobertura dos defeitos em uma inspeção não sistemática e sistemática – círculos e pontos representam defeitos e áreas tracejadas a cobertura de diferentes técnicas/inspetores – adaptado de (Porter e Votta, 1998)

Existem diferentes técnicas de inspeção as quais visam reduzir a influência do fator humano nos resultados de uma inspeção. Estas técnicas podem ser classificadas em *checklist* e técnicas de leitura. Ainda que não seja um tipo de técnica de inspeção estritamente sistemática, no tipo *checklist*, o inspetor segue uma lista de itens com características a serem revisadas, tendo a cobertura do documento inspecionado relacionada aos itens do *checklist* (Shull *et al.*, 2000). Técnicas de leitura, por sua vez, além de especificarem o que deve ser revisado em um documento, instruem como a leitura deve ser realizada (Shull *et al.*, 2000), possuindo assim um caráter mais sistemático. De qualquer maneira, independentemente da técnica de inspeção, um dos principais aspectos na utilização de técnicas de inspeção é observar a sua complementaridade com outras técnicas (Maldonado *et al.*, 2006).

O projeto onde este estudo foi conduzido utiliza, desde o seu início, a técnica de inspeção ad-hoc para modelos de casos de uso. Além disto, utiliza modelos de caso de uso para representar os requisitos funcionais do software. Vislumbrando uma maior complexidade dos casos de uso de um novo módulo do projeto que será iniciado, esta pesquisa investiga a utilização de técnicas de inspeção mais aprimoradas com objetivo de melhorar o processo de detecção de defeitos no sentido de ampliar a cobertura de identificação de defeitos.

4.1.2.Contexto do Projeto/Trabalho

O contexto deste estudo #2 é o mesmo referente ao estudo #1 que trata do desenvolvimento de um sistema de informação Web seguindo um ciclo de vida incremental onde o sistema é entregue em módulos. A equipe também é a mesma do estudo anterior, sendo uma equipe trabalhando remotamente (equipe R) e outra trabalhando na própria organização (equipe L). Para cada módulo do sistema as atividades de desenvolvimento são desempenhadas em cascata: *análise, projeto, codificação e testes. Em todas estas atividades* modelos de casos de uso são utilizados como base (Figura 15). Na etapa de projeto, modelos de alto nível são construídos e diagramas de classes e de sequência são elaborados a partir do modelo de caso de uso. Durante a codificação, o modelo de casos de uso é utilizado para implementar elementos da conversação entre o sistema e o usuário, como navegação, validação de formulários e apresentação de dados – a interface gráfica do sistema já é construída durante a etapa de análise por meio da utilização de prototipação. Testes estruturais e funcionais são extraídos do modelo de casos de uso (Dias

Neto *et al.*, 2007). Além disto, é importante mencionar que o responsável pelo planejamento e construção dos testes compõe a equipe de análise de requisitos. Isto torna mais simples a tarefa de extrair os casos de teste do modelo de casos de uso, já que este profissional possui conhecimento tácito sobre o domínio. A etapa uso refere-se à utilização do sistema em si, o qual deve contemplar todas as funcionalidades requeridas pelo cliente.

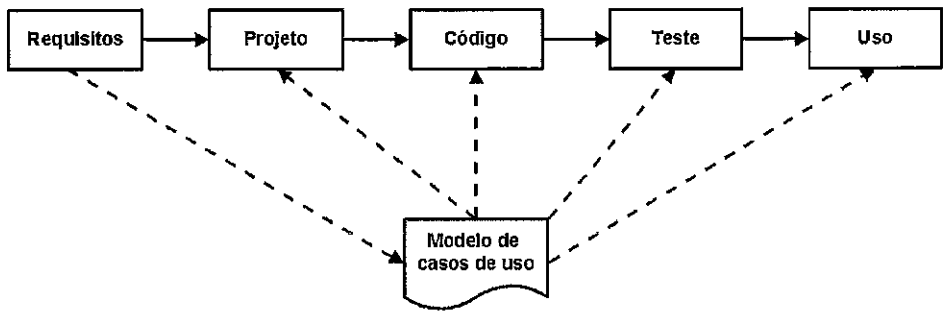


Figura 15 - Uso do modelo de casos de uso ao longo do ciclo de vida do software – adaptado de Shull (1998)

No momento desta pesquisa, o projeto contava com três módulos desenvolvidos (módulo de solicitações, gestão de usuários e protocolo, nesta ordem) e um parcialmente construído (módulo de acompanhamento de projetos – MAP) com a conclusão de duas iterações, sendo que toda a especificação do módulo já havia sido inspecionada. Um quinto módulo, o módulo financeiro (MFI), já possuía duas iterações especificadas, sendo que a primeira encontrava-se inspecionada.

Em uma análise quantitativa sobre a utilização de inspeção nos três primeiros módulos já desenvolvidos (Kalinowski *et al.*, 2007), pôde-se verificar que as inspeções permitiram encontrar um grande número de defeitos onde muitos deles foram encontrados por especialistas do domínio (clientes) que participaram das inspeções. Observou-se ainda uma redução contínua da densidade de defeitos (defeitos por página) entre as diferentes inspeções dos respectivos módulos – a densidade passou de 1,4 para 0,84 até chegar a 0,58 no terceiro módulo. Os dados das inspeções do MAP e MFI ainda não haviam sido consolidados. Dois fatores importantes podem ser atribuídos a este decréscimo; a mudança do processo de elicitação de requisitos que passou a considerar também nas entrevistas pessoas com perfil gerencial e não só operacional, e o aprendizado adquirido pela equipe com relação aos defeitos presentes nos modelos de casos de uso (Kalinowski *et al.*, 2007).

No entanto, mesmo com o bom histórico da utilização de inspeções nestes módulos do projeto, durante a construção do MAP as equipes de projetistas e desenvolvedores enfrentaram dificuldades no entendimento dos modelos de casos de uso, exigindo uma constante interação com os analistas para prosseguir com o desenvolvimento. A percepção da equipe em relação a esta situação foi a de que o domínio havia tornado-se maior e mais complexo e, associado a isto, a forma como as descrições dos casos de uso vinham sendo elaboradas estavam inadequadas para representar esta crescente complexidade. Por exemplo, cálculos financeiros eram apresentados de forma escrita – "para calcular o percentual da taxa X multiplique o valor orçado para o projeto pelo percentual de administração de projetos" – em vez do uso de uma fórmula algébrica.

É importante mencionar ainda que o analista responsável pela elicitação dos requisitos dos módulos anteriores ao MAP tinha também o papel de projetista. Desta forma, o MAP foi o primeiro módulo que foi especificado por membros da equipe que não tinham envolvimento com a construção do *software* (projeto e codificação). Como conseqüência, é possível que problemas nas descrições dos casos de uso estivessem presentes nos módulos anteriores, mas eram compensados pelo conhecimento tácito do projetista adquirido durante o desempenho de seu papel como analista.

4.1.3.Tema de Pesquisa

Diante das dificuldades no entendimento dos modelos de casos de uso do MAP e do histórico de bons resultados da utilização de inspeções no projeto surge, então, a questão do porquê os problemas relacionados à compreensão dos modelos não foram capturados. Devido às limitações das inspeções *ad-hoc*, esta pesquisa busca viabilizar e avaliar o uso de técnicas mais controladas (*checklist* ou técnicas de leitura) com o objetivo de melhorar a qualidade dos modelos dos casos de uso visando, principalmente, a compreensibilidade e inteligibilidade dos artefatos. A idéia é inicialmente aplicar as técnicas de inspeção na segunda interação do MFI composta por dez casos de uso.

O tema desta pesquisa é, então, a avaliação do uso de técnicas de inspeção em modelos de casos de uso considerando a sua compreensibilidade e inteligibilidade.

4.2. Planejamento

Nesta seção encontra-se descrita a fase de planejamento, a qual é iniciada pela identificação de trabalhos relacionados ao tema de pesquisa. A partir disto, o enfoque da ação da pesquisa é fixado e as hipóteses associadas aos resultados esperados determinadas. Além disto, são definidos os instrumentos que serão utilizados, além de possíveis ferramentas e técnicas para a pesquisa. Faz parte ainda do planejamento a condução de um estudo controlado com objetivo de conhecer os limites da tecnologia selecionada e sua aplicabilidade no projeto[6].

4.2.1.Seleção de Trabalhos

Neste estudo, diferentemente do estudo #1, a etapa de revisão da literatura não foi conduzida, pois a seleção deu-se em uma oportunidade no Simpósio Brasileiro de Qualidade de Software de 2008, onde o trabalho de Deboni e Gregolin (2008) foi apresentado e mostrou-se adequado para o contexto desta pesquisa – este trabalho corresponde à pesquisa realizada por Gregolin (2007).

Gregolin (2007) define um modelo de qualidade para diagrama e descrição de casos de uso. Nos modelos são definidos atributos de qualidade e regras de elaboração, e a partir desse modelo, *ckecklists* foram elaborados para verificar se os diagramas e descrições do caso de uso inspecionado atendem aos modelos de qualidade propostos. A proposta também define artefatos de registro de inspeção os quais são apresentados no anexo B.5.

O modelo de qualidade definido por Gregolin (2007) baseia-se em uma ampla pesquisa bibliográfica sobre boas práticas e diretrizes de elaboração de modelo de caso de uso, além de três modelos de qualidade de especificação de software (IEEE Std 830, 1998, Davis *et al.*, 1993, Fabbrini *et al.*, 2001). O modelo do Institute of Electrical and Electronics Engineers (IEEE Std 830, 1998), em suas práticas recomendadas para uma boa especificação de requisitos de software, define como características a correção, não ambigüidade, completeza, consistência, classificação por importância ou estabilidade, verificabilidade, modificabilidade e rastreabilidade. Em Davis *et al.* (1993) são definidos 18 atributos de qualidade muitos dos quais são análogos aos do modelo do IEEE Std 830 (1998), mas também apresentam preocupações diferentes incluindo, por exemplo, compreensibilidade e independência de ferramenta. O terceiro modelo de qualidade, de Fabrini *et al.*

[6] Este estudo inicial não foi conduzido no estudo #1.

(2001), é fundamentado em técnicas lingüísticas e visa, essencialmente, a testabilidade, consistência e compreensibilidade. Entre os indicadores de defeitos relacionados a estes atributos de qualidade de Fabrini *et al.* (2001) estão a imprecisão, subjetividade, presença de termos implícitos e média de palavras por sentença.

Por meio de um comparativo entre estes modelos Gregolin (2007) extraí atributos de qualidade e características que poderiam servir de regra de elaboração de um modelo de caso de uso e que poderiam ser verificáveis através de um processo de inspeção utilizando *checklists*. Os anexos B.3 e B.4 apresentam estes *checklists* que foram utilizados neste estudo e que foram retirados de Gregolin (2007).

Voltando ao contexto do projeto apresentado anteriormente, as características da proposta de inspeção de Gregolin (2007) apresentaram-se como uma alternativa interessante para os problemas diagnosticados. De fato, Anda e Sjøberg (2002) mostraram que diretrizes para elaboração de casos de uso são importantes para as descrições em termos do nível de detalhe utilizado, realismo (sequência lógica e completa) e consistência (uso correto de vocabulário). Tiveram como resultado que o uso de diretrizes é significantemente melhor para construir, e posteriormente compreender, casos de uso daqueles que não são elaborados com a utilização de diretrizes. Seguindo este raciocínio, é esperado que a utilização da proposta de Gregolin (2007) no contexto do projeto possa apresentar ganhos frente à inspeção *ad-hoc*, já que o seu foco está nos atributos de qualidade fundamentados nas diretrizes de elaboração de casos de uso dos três modelos de qualidade apresentados. Os atributos de qualidade são: compreensibilidade, completeza, precisão, não ambigüidade, consistência, independência de ferramenta e de interface, rastreabilidade, objetividade e representação de valor. Cada atributo de qualidade, detalhados em Gregolin (2007), possui associado uma ou mais questões do *checklist*.

Para melhor avaliar a decisão da escolha da tecnologia *checklist* de Gregolin (2007), entendeu-se necessária a condução de um estudo controlado. Este estudo é descrito na seção a seguir.

4.2.1.1. Estudo Inicial

Três objetivos principais foram traçados à execução do estudo controlado: (1) obter conhecimento e proficiência sobre a tecnologia que seria utilizada no projeto, no caso o *checklist*, (2) observar como a tecnologia se comportaria frente à técnica *ad-hoc* em um ambiente controlado, capturando indícios de seus benefícios assim como limitações, e (3) verificar a necessidade de alguma melhoria ou ajuste à tecnologia antes de aplicá-la no projeto.

Os participantes foram divididos aleatoriamente em três grupos, considerando a experiência individual de cada participante, a fim de equilibrar os grupos (a caracterização dos participantes foi feita por meio do formulário de caracterização do anexo B.1). O documento inspecionado pelos grupos era composto de dois casos de uso, requisitos funcionais e não-funcionais referentes aos casos de uso, além do glossário e uma descrição geral do sistema. Foram utilizados casos de uso reais do projeto referentes ao módulo MAP (módulo anterior ao MFI). No total, seis casos de uso foram selecionados e separados por complexidade (B – baixa, M – média e A – alta). Cada grupo utilizou inicialmente a técnica *ad-hoc* e, em seguida, a técnica *checklist*. A Tabela 16 sumariza o arranjo do estudo.

Tabela 16 - Arranjo do estudo controlado

	Grupo 1	Grupo 2	Grupo 3
Treinamento	Sem treinamento – foi apresentada a descrição do domínio e definido o foco da inspeção nos atributos de qualidade		
Inspeção Ad-hoc	Casos de Uso • B1 e A1	Casos de Uso • B2 e M1	Casos de Uso • M2 e A2
Treinamento	Com treinamento – foram apresentadas as regras do *checklist*		
Inspeção checklist	Casos de Uso • B2 e M1	Casos de Uso • M2 e A2	Casos de Uso • B1 e A1

Este estudo controlado foi executado entre os dias 14/08/2008 e 04/09/2008 com doze alunos de mestrado e doutorado – cada grupo ficou com quatro alunos. Durante a disciplina já havia sido ministrado aos alunos conceitos sobre inspeção e técnicas de inspeção (*checklist* e técnica de leitura). Os alunos também já haviam realizado uma inspeção *ad-hoc* no contexto da disciplina. A inspeção *ad-hoc* foi disparada no dia 14/08 e teve prazo de uma semana para execução, enquanto que a *checklist* foi disparada no dia 28/08 e também contou com uma semana para término. Para registro das

discrepâncias o formulário do anexo B.5 foi utilizado. A atividade de discriminação de defeitos, onde as discrepâncias são classificadas em falso-positivo e defeito, foi realizada pelo próprio pesquisador.

O método de análise de variância foi escolhido para a identificação de alguma diferença estatística significativa entre médias das medidas da inspeção *ad-hoc* e *checklist* como, por exemplo, número de defeitos. Cabe, portanto, descrever como a ferramenta JMP (JMP, 2007), utilizada para análise estatística, apresenta os seus resultados graficamente por meio do chamado diamante de média (Figura 16). A linha horizontal que cruza o diamante representa a média do grupo. A extensão vertical do diamante representa o intervalo de confiança de 95%. Marcas de sobreposição são traçadas em $(\sqrt{2} \times IC) \div 2$ acima e abaixo da média do grupo. Para amostras com tamanhos iguais, o que sempre ocorrerá neste estudo, marcas que sobreponham à linha que define média total dos grupos indicam que a média dos grupos não é significativamente diferente para o intervalo de confiança de 95%. Ou seja, caso a marca de sobreposição ultrapasse a linha da média total dos grupos não existe diferença significativa entre as médias. Na Figura 16, a marca de sobreposição inferior do diamante do grupo B ultrapassa a faixa da média total dos grupos, porém nada se pode afirmar sobre a significância do resultado na medida em que os grupos têm tamanho diferentes (representado pela largura dos diamantes).

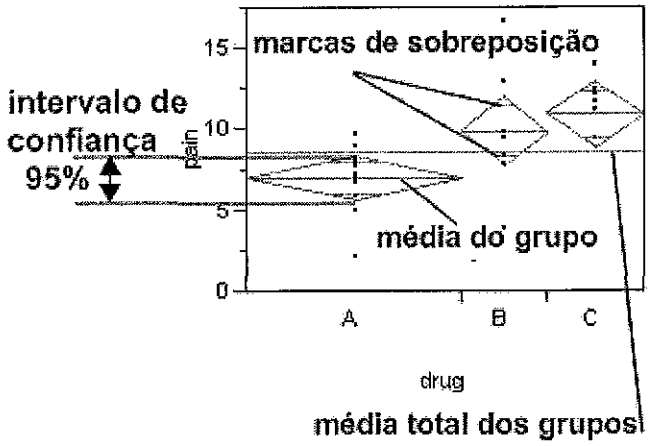
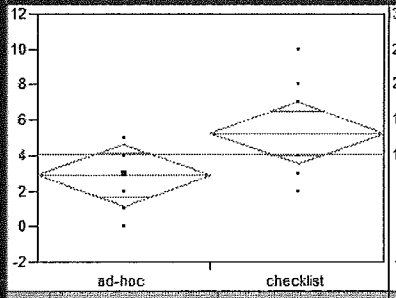
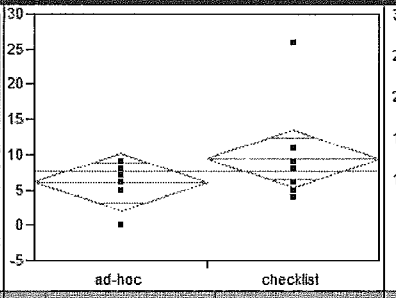
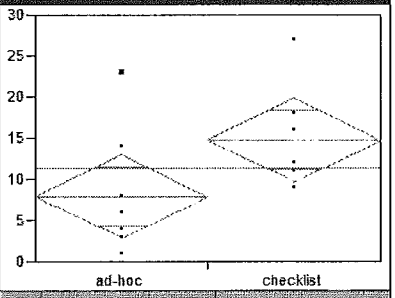
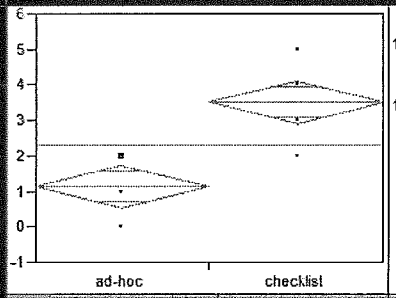
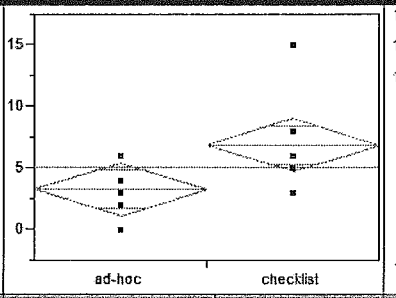
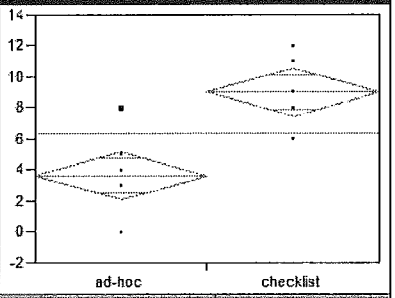


Figura 16 - Descrição do gráfico da ferramenta JMP

De posse desta descrição da representação dos dados na ferramenta JMP, a Tabela 17 consolida os resultados do estudo controlado conduzido. Conforme o arranjo descrito anteriormente, são apresentadas comparações entre as inspeções *ad-hoc* e *checklist,* agrupado pela complexidade dos casos

de uso. Com este agrupamento, têm-se oito observações (medidas) em cada inspeção. Por exemplo, para as observações da inspeção *checklist* dos casos de uso de baixa complexidade foram agrupados os grupos 1 (caso de uso B2) e 3 (caso de uso B1) – conforme Tabela 16. É importante mencionar que, apesar de a Tabela 17 só comparar o número de discrepâncias e o número de defeitos, tinha-se, inicialmente, a expectativa de comparar o tempo despendido em cada uma das inspeções, mas os participantes só foram instruídos a registrar o tempo total de inspeção e, desta forma, não foi possível computar o tempo por caso de uso – ou seja, por complexidade. Isto gerou um aprendizado para o planejamento do estudo de pesquisa-ação.

Tabela 17 - Resultados da inspeção

	Complexidade baixa		Complexidade média		Complexidade alta	
Discrepância						
	ad-hoc	checklist	ad-hoc	checklist	ad-hoc	checklist
$\bar{X}$	2,875	5,250	6,000	9,375	7,875	14,750
σ	1,727	2,764	2,725	7,090	7,279	5,994
α	0,0584		0,2294		0,0583	
Defeitos						
	ad-hoc	checklist	ad-hoc	checklist	ad-hoc	checklist
$\bar{X}$	1,125	3,500	3,250	6,875	3,625	9,000
σ	0,641	0,926	1,752	3,681	2,264	1,852
α	**0,0001**		**0,0248**		**0,0001**	

Analisando o resultado das inspeções nota-se uma diferença entre as médias, com um resultado superior à inspeção *checklist*, em todos os casos. No entanto, apenas as diferenças relacionadas à defeitos mostraram-se estatisticamente significativas ($\alpha < 0,05$). Isto representa um resultado positivo na medida em que mostra que a técnica *checklist* detectou um menor número de falso-positivos no contexto desta inspeção. Outro dado interessante é que a

inspeção *ad-hoc* surpreendentemente apresentou, em grande parte dos casos, um desvio padrão menor que *checklist* podendo ser um indício da influência de algum fator de confusão, como a experiência do inspetor. Por esta razão, uma análise foi feita buscando detectar alguma influência da experiência sobre o resultado. Para isto, os inspetores foram subdivididos em dois grupos, um com os mais experientes e outros com os menos. Não foi observada nenhuma mudança de comportamento nesta análise, ou seja, tanto para o grupo mais experiente quanto para o com menor experiência a técnica *checklist* foi superior no número de defeitos identificados, assim como a Tabela 17. No entanto, em um dos casos (complexidade média) não foi possível encontrar uma diferença estatisticamente significativa, mas isto pode ser atribuído ao pequeno número de observações que restaram em cada subgrupo[7].

Além dos dados quantitativos apresentados, dados qualitativos foram coletados por meio de questionários pós-estudo[8] (anexo B.2). Dois tipos de questões foram feitas aos inspetores: (1) questões relacionadas a condução e aplicação estudo controlado e (2) questões relacionadas a utilidade e qualidade da técnica *checklist*. Grande parte das questões estava em uma escala ordinal ou nominal e, por este motivo, a Tabela 18 consegue exprimir a avaliação dos inspetores. Para as questões discursivas tentou-se resumir a resposta em uma avaliação positiva ou negativa de forma a encaixar em uma escala nominal.

Tabela 18 - Resultado do questionário pós-estudo

Questão	**Opções e Resultados**				
Qualidade e utilidade do *checklist*					
Como você classifica o grau de dificuldade da aplicação do *checklist*?	Muito Difícil	Difícil	Médio	Fácil	Muito Fácil
	0	0	6	5	1
Você seguiu o checklist completamente?	Não	Parcialmente	Completamente		
	0	0	12		
As questões do checklist fizeram sentido como apoio à identificação de defeitos?	Não	Parcialmente	Completamente		
	0	4	8		
Considerando a perspectiva de qualidade	Obstáculo	Neutro	Auxiliou		

[7] Como cada agrupamento (por exemplo, inspeção *ad-hoc* dos casos de uso de complexidade baixa) tinha oito observações, a subdivisão dos agrupamentos em mais e menos experientes limitou a quatro observações para cada subagrupamento.

[8] As questões 8 e 9 foram elaboradas somente no estudo de pesquisa-ação, após este estudo inicial e, portanto, não são analisadas na Tabela 18.

adotada, como o checklist auxiliou na identificação de defeitos nos casos de uso?	0	0	12
Você utilizaria o checklist em inspeções futuras?	Não	Talvez	Sim
	0	3	9

Na sua opinião, como o checklist poderia ser melhorado?	Não	Sim
• Em relação à ordem das questões	2	10
• Questões para detecção de defeitos: Claras? Puderam ser entendidas?	3	9
• Número de questões (Alguma questão ausente ou excedente?)	7	5

Condução do estudo controlado

Como você avalia o treinamento conduzido?	Insuficiente	Neutro	Adequado
	0	2	10
Você achou o tempo de inspeção adequado?	Não	Sim	
	0	12	
Como você classifica a qualidade do documento de requisitos utilizado na inspeção?	Má qualidade	Neutra	Boa qualidade
	0	4	8

A avaliação qualitativa mostra um retorno positivo por parte dos participantes tanto da utilidade da técnica *checklist* quanto da condução do estudo controlado. Um dado que chama atenção é que um número considerável dos participantes enfrentou algum tipo de dificuldade na aplicação do *checklist*, como mostra a primeira questão da Tabela 18. Em discussões informais com alguns dos participantes a percepção foi de que um número elevado, mas necessário, de questões torna o *checklist* um pouco cansativo e até mesmo repetitivo já que muitas exigem a verificação de todo o documento. Ainda assim, quando perguntados se o número de questões era suficiente ou insuficiente, muitos participantes responderam com sugestões de novos tópicos ou questões que o *checklist* deveria cobrir.

Dado os objetivos definidos no início desta seção, os resultados (quantitativos e qualitativos) obtidos neste estudo controlado mostraram um comportamento benéfico no que diz respeito à detecção de defeitos segundo o foco da compreensibilidade dos modelos de casos de uso. Foi importante observar a utilidade do *checklist* tanto para inspetores experientes quanto para os inexperientes. Além disto, ainda segundo o escopo dos objetivos traçados, a condução do estudo permitiu ao pesquisador, durante atividades como preparação do material de treinamento e discriminação de defeitos da inspeção, um maior entendimento da técnica. Não foi observada, no entanto,

nenhuma necessidade de adaptação do *checklist* para a aplicação no projeto, avaliação esta bastante influenciada pela análise qualitativa da Tabela 18.

4.2.2.Enfoque da Ação

4.2.2.1. Objetivos

O objetivo principal deste estudo, definido utilizando a abordagem *Goal-Question-Metric* (GQM) proposta por Basili *et al.* (1994):

> *analisar* a inspeção de modelos de caso de uso *com o propósito de* caracterizar *com respeito à* eficiência da identificação de defeitos que dificultam a compreensibilidade de modelos de casos de uso *do ponto de vista* do engenheiro de software *no contexto de* desenvolvimento de sistemas de informação Web.

4.2.2.2. Questões de Pesquisa

O objetivo será alcançado quando respostas forem apresentadas às seguintes questões:

- Q.1. O uso da técnica de inspeção *checklist* encontra mais defeitos que a técnica *ad-hoc*? Esta questão tem como objetivo verificar, no contexto do projeto, se o custo-eficiência utilizando a técnica *checklist* é melhor do que a técnica *ad-hoc*.
- Q.2 A técnica de inspeção *checklist* encontra defeitos diferentes do que a técnica *ad-hoc*? A idéia aqui é comparar o resultado das duas técnicas, o que servirá de base para justificar a diferença na compreensibilidade dos modelos de caso de uso – caso esta diferença venha a se manifestar de fato (questão 3).
- Q.3 Caso a técnica de inspeção *checklist* encontre defeitos diferentes, os modelos de caso de uso inspecionados, e posteriormente corrigidos, são mais compreensíveis/inteligíveis? Esta última questão captura a expectativa da melhoria da qualidade dos modelos de caso de uso relacionada, principalmente, a sua compreensibilidade.

4.2.2.3. Resultados Esperados (coleção de dados)

A questão Q.1 possui as seguintes questões práticas associadas:

- Q.1.1 Qual é o número de defeitos encontrados utilizando cada uma das técnicas?
- Q.1.2 Qual é o tempo médio aplicado à inspeção pelos inspetores das duas técnicas?

Para as questões 1.1 e 1.2 serão analisados os dados informados pelos inspetores, após a etapa de discriminação de defeitos.

A questão Q.2 tem as seguintes questões práticas associadas:

- Q.2.1 Qual a distribuição do tipo de defeitos identificados?

Novamente, a questão 2.1 poderá ser respondida por meio da análise dos dados informados pelos inspetores.

A questão Q.3 tem as seguintes questões práticas associadas:

- Q.3.1 Com qual freqüência os projetistas e desenvolvedores interrompem suas tarefas por conta de um não entendimento de uma informação presente nos modelos de caso de uso?
- Q.3.2 O que leva a este não entendimento? Quais os fatores por trás deste não entendimento? Exemplos destes fatores poderiam ser a falta de informação, inconsistência, escrita ruim.

Os projetistas e desenvolvedores do projeto deverão registrar e descrever o que os levaram ao não entendimento do modelo. Isto deve ser realizado já para a primeira iteração do MFI que servirá de base de comparação para a segunda iteração.

4.2.3.Hipóteses (suposições)

- H.0 A técnica de inspeção *checklist* proposta por Gregolin (2007) encontra mais defeitos que a técnica *ad-hoc* utilizada no projeto[9].
- H.1 A técnica de inspeção *checklist* encontra defeitos diferentes da *ad-hoc*.
- H.2 Os defeitos capturados pela inspeção utilizando *checklist* contribuem para compreensibilidade/inteligibilidade dos modelos de caso de uso.

[9] É importante mencionar que vários estudos já foram executados com objetivo de analisar a inspeção *ad-hoc* frente à *checklist*. Os resultados ainda se mostram contraditórios com estudos apontando tanto uma superioridade da técnica *checklist* (Porter e Votta, 1998, Anda e Sjøberg, 2002) quanto favoráveis à abordagem *ad-hoc* (Wohlin *et al.*, 2002, Cox *et al.*, 2004). Desta forma, esta hipótese visa contribuir para este cenário com mais uma avaliação.

4.2.4.Definições Operacionais

4.2.4.1. Técnicas

Para permitir a coleção dos dados, diferentes formulários deverão ser preenchidos pelos inspetores, projetistas e desenvolvedores. Dados quantitativos serão analisados utilizando análise de variância (ANOVA) para testar a hipótese H.0. Estatística descritiva será utilizada para análise dos outros dados quantitativos como, por exemplo, a distribuição dos defeitos por inspeção (*ad-hoc* e *checklist*). Por fim, alguma avaliação qualitativa será realizada nos dados obtidos com o questionário pós-estudo (anexo B.2) e reuniões do projeto, porém sem a utilização de técnicas sistemáticas como, por exemplo, a *Grounded Theory* (Strauss e Corbin, 1990).

4.2.4.2. Ferramentas

A análise dos dados quantitativos será realizada com a ferramenta JMP (JMP, 2007). A inspeção será conduzida com apoio da ferramenta ISPIS (Kalinowski e Travassos, 2004). ISPIS possibilita a realização sistematizada de inspeções assíncronas, em artefatos produzidos ao longo do processo de desenvolvimento de *software*, com equipes geograficamente distribuídas. Além de uma série de facilidades necessárias para os inspetores registrarem as discrepâncias, informações da inspeção são disponibilizadas ao moderador da inspeção como, por exemplo, número total de defeitos, tempo gasto e distribuição dos defeitos por severidade e tipo.

4.2.4.3. Instrumentos

Devido ao uso da ferramenta ISPIS, alguns instrumentos estão embutidos na própria ferramenta. A Figura 17 apresenta o formulário de inclusão de discrepâncias utilizado pelos inspetores. Cada discrepância pode ser classificada em cinco tipos (Kalinowski, 2004):

- **Omissão:** (1) algum requisito importante relacionado à funcionalidade, ao desempenho, às restrições do projeto, ao atributo, ou à interface externa não foi incluído; (2) não está definida a resposta do software para todas as possíveis situações de entrada de dados; (3) faltam seções na especificação de requisitos; (4) faltam referências de figuras, tabelas e diagramas.
- **Ambigüidade:** um requisito tem várias interpretações devido a diferentes termos utilizados para uma mesma característica.
- **Inconsistência:** dois ou mais requisitos são conflitantes.

- **Fato incorreto:** um requisito descreve um fato que não é verdadeiro, considerando as condições solicitas para o sistema.
- **Informação estranha:** as informações fornecidas no requisito não são necessárias ou mesmo usadas.
- **Outros:** outros defeitos como a inclusão de um requisito em uma seção errada do documento.

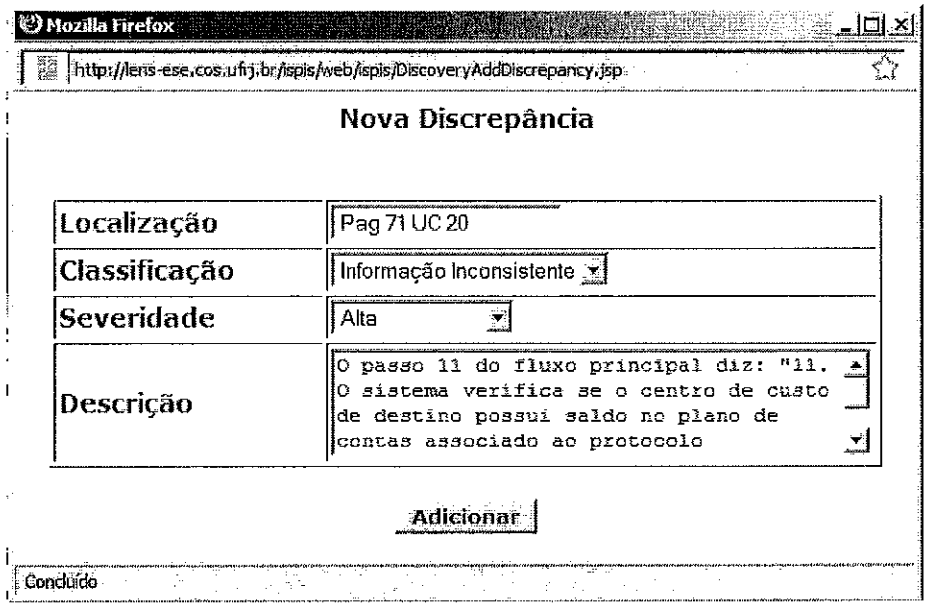


Figura 17 - Formulário de discrepâncias de ISPIS

Um segundo instrumento importante no contexto deste estudo é o de registro de dúvidas nos modelos de casos de uso que permitirá a coleta de dados para as questões Q.3.1 e Q.3.2. Este instrumento existe sob a forma de uma planilha eletrônica, onde cada linha representa uma dúvida e cada coluna um campo a ser preenchido. A Tabela 4 lista os campos da planilha e contém uma descrição de cada um deles.

Tabela 19 - Campos do formulário de registro de dúvidas em modelos de casos de uso

Nome do campo	Descrição
Localização	Módulo, UC e localização do trecho do UC (Passo, Fluxo alternativo/principal).
Data	Data e hora aproximada em que o problema foi detectado.
Descrição	Descrição detalhada da dúvida em relação ao caso de uso em questão.
Motivo ou origem da dúvida	Sugestão da causa do não entendimento. Exemplos: 1. Falta de conhecimento do domínio; 2. Problemas com a escrita do UC (ambiguidade, inconsistência, frases longas, nível de detalhe das descrições, falta de clareza etc); 3. Possível dependência com outros casos de uso os quais você não tem conhecimento.

Tempo gasto para resolução	Registro do tempo gasto (em minutos) aproximado para sanar a dúvida. Deve ser feito apenas quando discussões sejam realizadas para explicar conceito que não estavam explícitos na descrição do caso de uso. Caso seja uma resposta breve, ou uma resposta por email este campo deve ser ignorado.
Descrição da resposta	Relato breve de como a dúvida foi solucionada ou qual foi a resposta obtida.
Data da resposta	Data e hora aproximada em que a resposta foi obtida.
Forma de comunicação	Registro da forma de comunicação (email, chat, voip ou presencial).
Defeito no UC	A dúvida resultou na descoberta de um defeito no UC (sim/não).

A caracterização dos inspetores será realizada por meio do formulário de caracterização do anexo B.1. Por fim, o último instrumento trata-se do questionário pós-estudo que está disposto no anexo B.2.

4.2.4.4. Arranjo do Estudo

Como mencionado anteriormente, dez casos de uso referentes à segunda iteração do Módulo Financeiro estavam especificados e deveriam ser inspecionados, conforme o processo definido no projeto. Na medida em que atividades de inspeção não representam custos elevados ao projeto e para viabilizar resultados estatísticos significativos, definiu-se que quatro membros da equipe L (um projetista e três desenvolvedores) participariam da inspeção, e os seis membros da equipe R (todos desenvolvedores) também deveriam participar. Somando um total de dez inspetores para inspeção. Em inspeções anteriores, apenas os desenvolvedores mais experientes eram selecionados e, por este motivo, dos dez inspetores apenas três tem experiência prévia com inspeção no contexto do projeto – um na equipe L e dois na equipe R.

Tabela 20 - Arranjo do estudo

	Equipe L	Equipe R
Treinamento	Sem treinamento – foi apresentada a descrição do domínio e definido o foco da inspeção nos atributos de qualidade	
Inspeção Ad-hoc	Casos de Uso • B1, B2, M1, M2 e A1	Casos de Uso • B3, B4, M3, M4 e A2
Treinamento	Com treinamento – foram apresentadas as regras do *checklist*	
Inspeção checklist	Casos de Uso • B3, B4, M3, M4 e A2	Casos de Uso • B1, B2, M1, M2 e A1

O arranjo deste estudo foi fortemente baseado no estudo controlado inicial descrito anteriormente e é apresentado na Tabela 20. Para este estudo os casos de uso também foram classificados por complexidade (B – baixa, M –

média e A – alta). A complexidade foi avaliada, principalmente, pelo tamanho do caso de uso. Casos de uso de baixa complexidade tiveram em média 2.75 páginas, 5 fluxos alternativos e 4.5 regras de negócio. Os de complexidade média tiveram, na mesma ordem, 3.95, 6.5 e 8.75; e os de alta 6, 10 e 12.5.

4.3. Ações

A equipe R foi a primeira a inspecionar os documentos entre os dias 17/02/2009 e 19/02/2009. O pesquisador deslocou-se até a cidade da equipe R para acompanhar as atividades. Inicialmente os desenvolvedores foram informados de que a inspeção, apesar de uma atividade normal de projeto, também seria utilizada para pesquisa e, por isto, as atividades seriam um pouco diferenciadas das inspeções anteriores, como por exemplo, aplicação de treinamento.

Os desenvolvedores foram instruídos sobre o foco da inspeção (mesmos atributos de qualidade do *checklist*) e sobre o uso da ferramenta ISPIS. Feito isto, a inspeção *ad-hoc*, com cinco casos de uso, foi iniciada em ISPIS. Como alguns inspetores ainda não haviam trabalhado com a ferramenta, eles foram auxiliados na sua utilização. A ferramenta é de uso bastante simples, especialmente na visão do inspetor, onde praticamente só existe facilidade para o cadastro de discrepâncias. A inspeção *ad-hoc* pôde ser concluída em um dia. No dia seguinte, o treinamento previsto para a inspeção *cheklist* foi aplicado e, logo em seguida, a inspeção *checklist*, com os outros cinco casos de uso, iniciada em ISPIS. A inspeção *checklist* foi concluída em dois dias, devido à necessidade do treinamento um pouco mais elaborado.

Para a equipe L a inspeção foi executada durante os dias 20/02/2009 e 09/03/2009. O treinamento conduzido foi o mesmo, mas, diferentemente da inspeção da equipe R, cada inspeção teve prazo de cerca de uma semana já que os desenvolvedores desta equipe trabalham em regime de tempo parcial e estavam em meio a outras atividades do projeto. Em nenhuma inspeção houve qualquer tipo de interrupção das atividades por conta de falha técnica de ISPIS ou não entendimento dos procedimentos apresentados aos inspetores. Ao final da inspeção o questionário pós-estudo foi aplicado e os seus resultados apresentados na próxima seção.

Paralelamente a condução das inspeções, os projetistas da equipe registraram as questões/dúvidas relacionadas aos modelos de casos de uso no instrumento que lhes foi fornecido (Tabela 19). Entretanto, próximo ao fim do

projeto da primeira iteração do Módulo Financeiro, um dos projetistas teve um problema com a sua estação de trabalho e terminou perdendo os seus registros. Isto inviabilizou uma análise detalhada para a Questão 3 neste estudo. Ainda assim, a definição do instrumento da Tabela 19 e das questões 3.1 e 3.2 foram mantidas no caso de uma futura avaliação destes aspectos em estudos posteriores. A título de curiosidade, o número de registros feitos pelo outro projetista chegou a 17. Considerando que o projeto da iteração ainda não tinha terminado e que cada projetista era responsável aproximadamente pela metade da carga de trabalho, é possível perceber que o registro destas questões/dúvidas poderia prover um resultado útil à investigação de pesquisa relacionadas à questão Q.3. Até o momento da escrita do relato deste estudo, a segunda iteração já tinha sete dos seus casos de uso projetados. A percepção até este momento foi a de que existiu um menor número de questões/dúvidas nesta etapa, entretanto apenas com o registro seria possível precisar melhor se isto de fato ocorreu e a sua real causa – por exemplo, tentando avaliar um possível aprendizado do domínio por parte dos projetistas ou a melhoria da compreensibilidade dos modelos de casos de uso devido à aplicação do *checklist*.

4.4. Avaliação e Análise

Conforme definido anteriormente, a análise dos resultados segue o mesmo formato da apresentada no estudo inicial. Como uma lição aprendida na condução do estudo inicial (seção 4.2.1.1), atentou-se para a necessidade do registro do tempo gasto por caso de uso já que foram divididos por complexidade como apresentado na Tabela 20. Esta seção está dividida em duas partes, inicialmente concentra-se na análise quantitativa relacionada às questões de pesquisa 1 e 2 cujo principal objetivo é avaliar o desempenho da técnica *checklist* com relação a quantidade e qualidade (distribuição) dos defeitos encontrados. A segunda parte descreve sucintamente a avaliação qualitativa do estudo com base, principalmente, no questionário pós-estudo, mas trazendo também alguns comentários sobre a percepção da utilização da técnica no contexto do projeto.

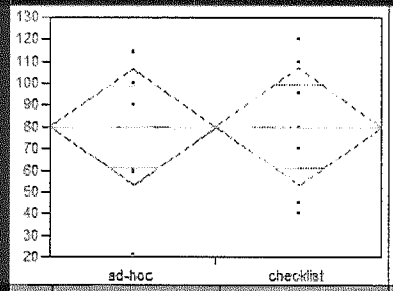
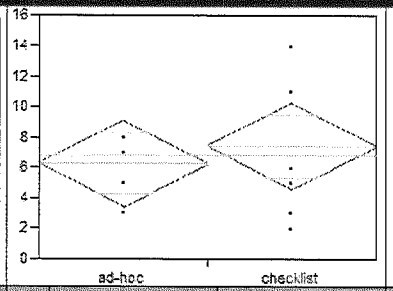
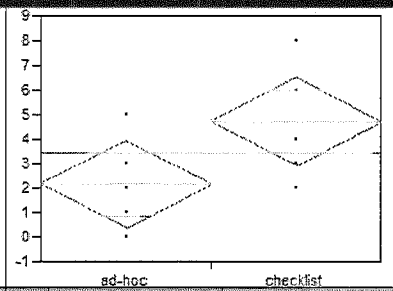
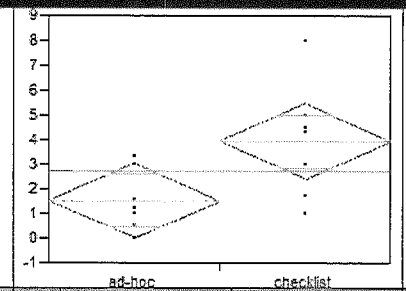
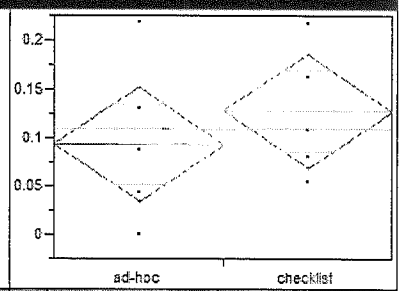
Como uma primeira etapa à análise dos dados foi feita a remoção de possíveis casos atípicos (*outliers*). Foram identificados dois casos atípicos. O primeiro está relacionado ao desenvolvedor que participou das últimas cinco inspeções do projeto e é um dos desenvolvedores mais experientes da equipe.

Em sua inspeção reportou em média 66 discrepâncias enquanto que o inspetor mais próximo identificou 26. Na outra ponta, o segundo caso atípico está relacionado a um desenvolvedor recém integrado à equipe (3 meses) e com pouca experiência em desenvolvimento de *software*. Em média reportou 3 discrepâncias enquanto que o inspetor com o número mais próximo reportou 9.

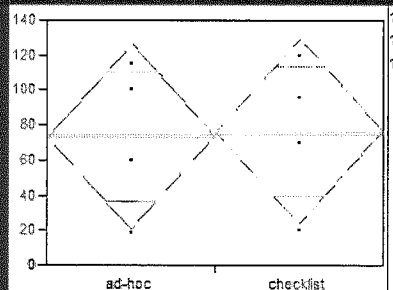
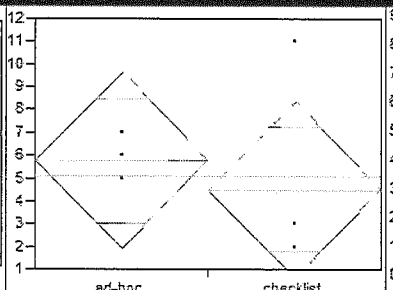
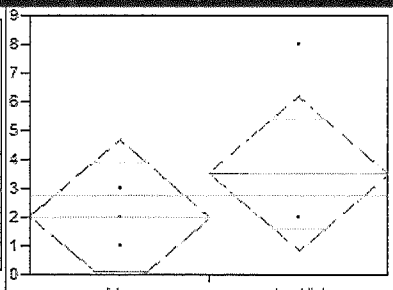
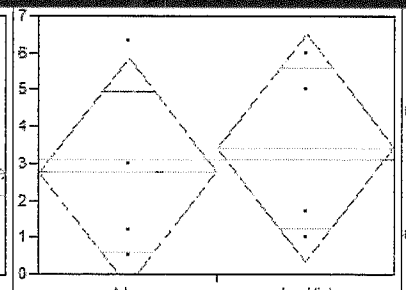
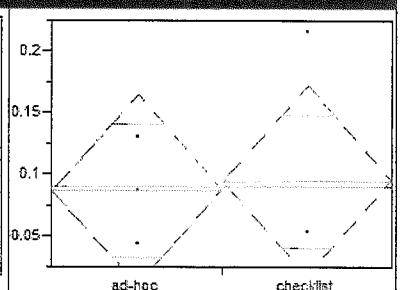
A seguir os resultados da inspeção são apresentados por complexidade dos casos de uso. Ao final, os resultados considerando o total somando todos os resultados anteriores. No estudo inicial esta totalização não foi possível, pois havia três grupos que inspecionaram casos de uso de apenas duas complexidades diferentes. Além do registro do tempo mencionado anteriormente, o custo-eficiência da inspeção pôde ser computado, e é representado pelo número de defeitos por hora. Outra medida nova apresentada, em relação ao estudo inicial, é a cobertura-eficiência que traz a porcentagem de defeitos identificados por cada inspetor em relação ao total de defeitos identificados por todos os inspetores. Também são apresentados resultados considerando a experiência do inspetor. A experiência foi avaliada com base no formulário de caracterização (anexo B.1) considerando o tempo como desenvolvedor, participações em outras inspeções, tempo de atuação no contexto do projeto, dentre outros. Cada subgrupo (experiente e inexperiente) ficou com quatro indivíduos já desconsiderando os casos atípicos.

Tabela 21 - Resultados da inspeção

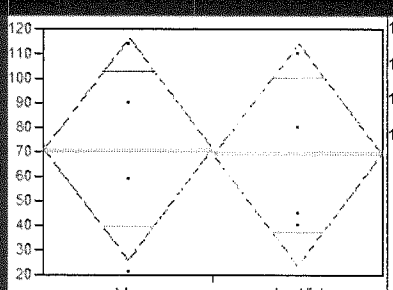
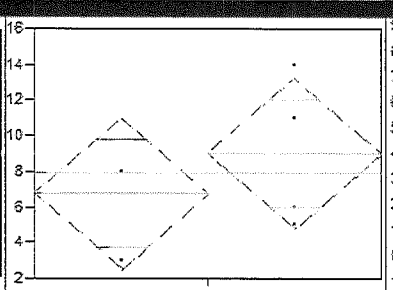
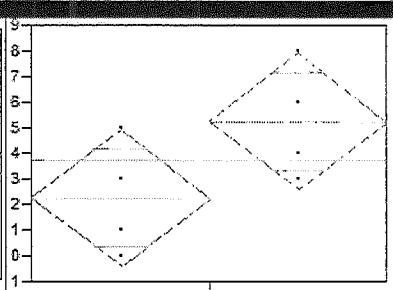
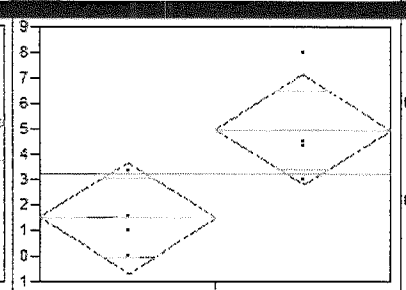
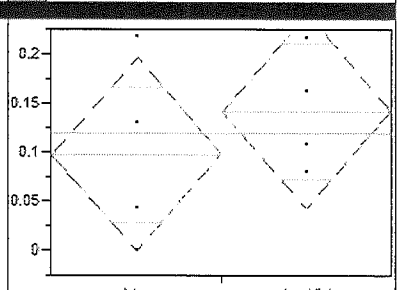
Complexidade Baixa (total)

	Tempo	Discrepância	Defeito	Custo-Eficiência	Cobertura-Eficiência
					
	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist
$\overline{X}$	79,857 / 80,143	6,286 / 7,429	2,143 / 4,714	1,521 / 3,939	0,093 / 0,127
σ	34,648 / 30,770	1,976 / 4,577	1,676 / 2,628	1,234 / 2,331	0,073 / 0,071
α	0,9873	0,5555	**0,0496**	**0,0320**	0,3905

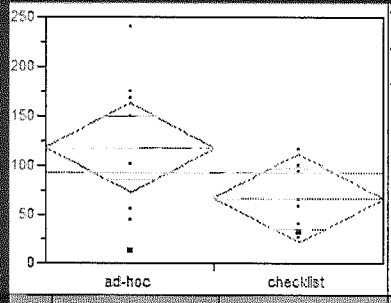
Complexidade Baixa (experiente)

					
	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist
$\overline{X}$	73,500 / 76,500	5,750 / 4,500	2,000 / 3,500	2,760 / 3,428	0,087 / 0,095
σ	43,116 / 42,844	0,957 / 4,359	0,816 / 3,000	2,594 / 2,445	0,035 / 0,081
α	0,9246	0,5956	0,3719	0,7209	0,8684

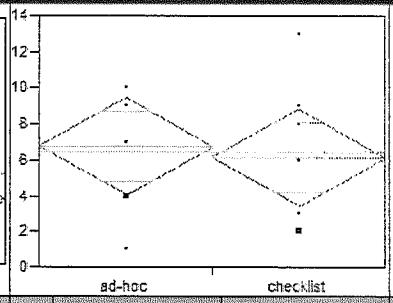
Complexidade Baixa (inexperiente)

					
	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist
$\overline{X}$	71,000 / 68,750	6,750 / 9,000	2,250 / 5,250	1,482 / 4,965	0,097 / 0,142
σ	40,224 / 32,755	2,500 / 4,243	2,217 / 2,217	1,395 / 2,133	0,096 / 0,060
α	0,9337	0,3961	0,1047	**0,0341**	0,4669

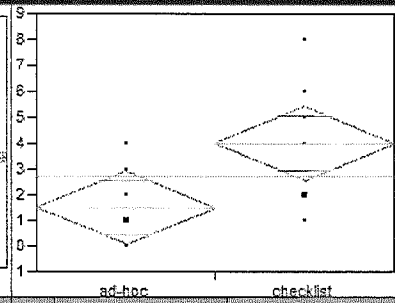
Complexidade Média (total)



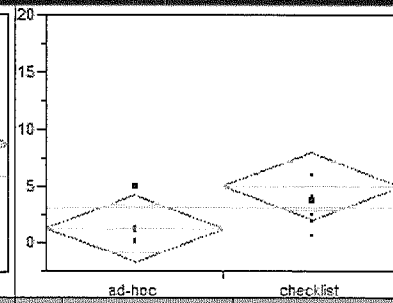
	Tempo	
	ad-hoc	checklist
$\bar{X}$	118,375	66,750
σ	77,785	33,830
α	0,1072	



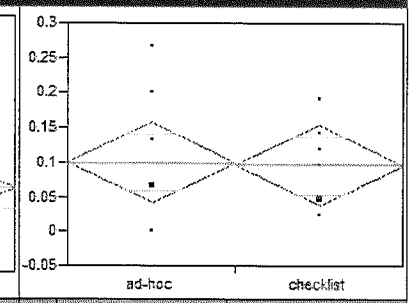
	Discrepância	
	ad-hoc	checklist
$\bar{X}$	6,750	6,125
σ	3,370	3,833
α	0,7342	



	Defeito	
	ad-hoc	checklist
$\bar{X}$	1,500	4,000
σ	1,414	2,330
α	**0,0212**	

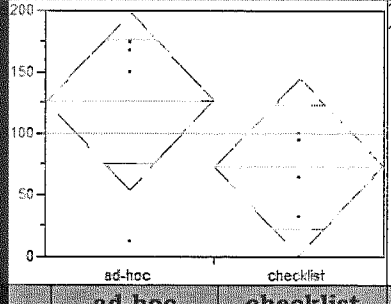


	Custo-Eficiência	
	ad-hoc	checklist
$\bar{X}$	1,279	5,039
σ	1,616	5,392
α	0,0798	

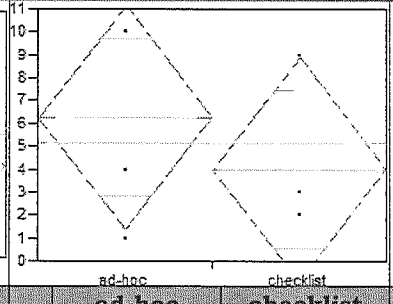


	Cobertura-Eficiência	
	ad-hoc	checklist
$\bar{X}$	0,100	0,095
σ	0,094	0,055
α	0,9033	

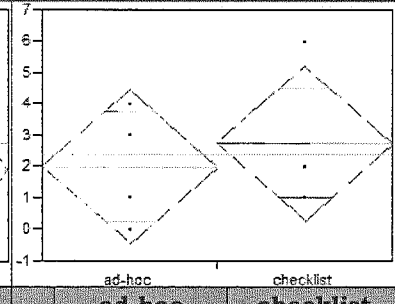
Complexidade Média (experiente)



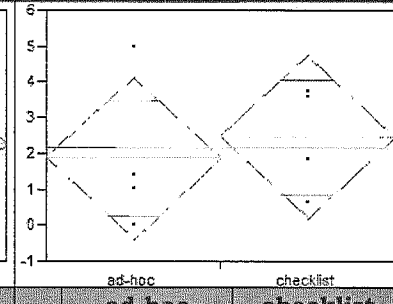
	Tempo	
	ad-hoc	checklist
$\bar{X}$	126,250	73,000
σ	76,891	31,401
α	0,2471	



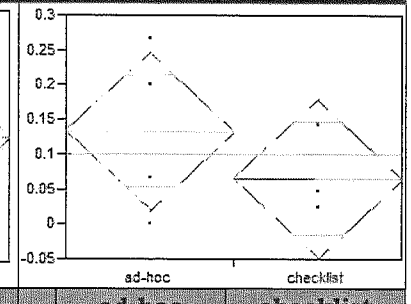
	Discrepância	
	ad-hoc	checklist
$\bar{X}$	6,250	4,000
σ	4,500	3,367
α	0,4538	



	Defeito	
	ad-hoc	checklist
$\bar{X}$	2,000	2,750
σ	1,826	2,217
α	0,6202	

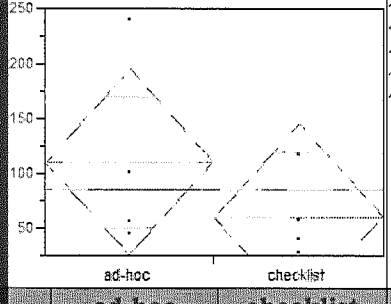


	Custo-Eficiência	
	ad-hoc	checklist
$\bar{X}$	1,865	2,458
σ	2,175	1,492
α	0,6691	

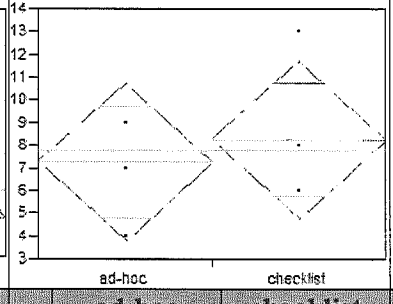


	Cobertura-Eficiência	
	ad-hoc	checklist
$\bar{X}$	0,133	0,065
σ	0,122	0,053
α	0,3457	

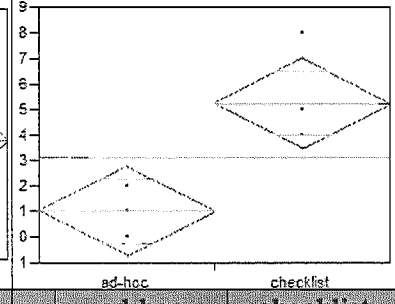
Complexidade Média (inexperiente)



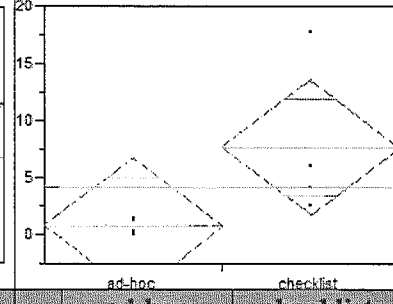
	Tempo	
	ad-hoc	checklist
$\bar{X}$	110,500	60,500
σ	89,668	39,753
α	0,3473	



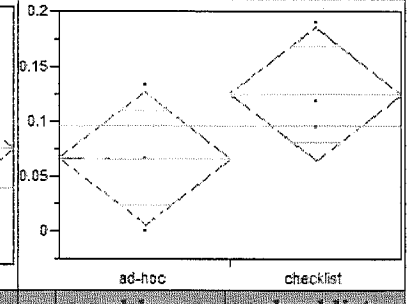
	Discrepância	
	ad-hoc	checklist
$\bar{X}$	7,250	8,250
σ	2,363	3,304
α	0,6400	



	Defeito	
	ad-hoc	checklist
$\bar{X}$	1,000	5,250
σ	0,817	1,893
α	**0,0062**	

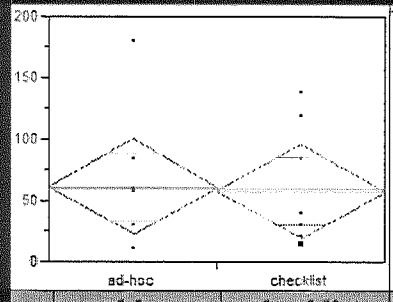


	Custo-Eficiência	
	ad-hoc	checklist
$\bar{X}$	0,693	7,620
σ	0,666	6,918
α	0,0932	

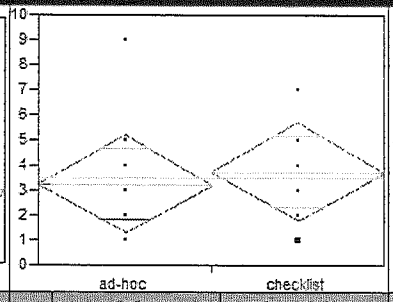


	Cobertura-Eficiência	
	ad-hoc	checklist
$\bar{X}$	0,067	0,125
σ	0,054	0,045
α	0,1501	

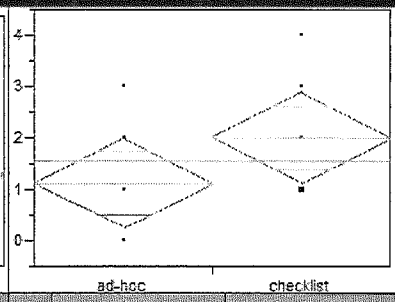
Complexidade Alta (total)



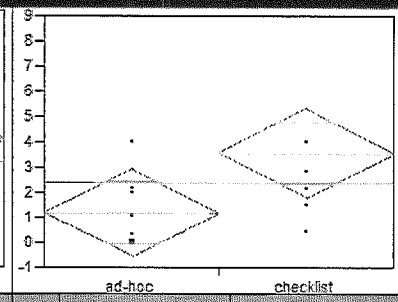
	ad-hoc	checklist
$\bar{X}$	61,625	57,750
σ	54,329	49,239
α	0,8833	



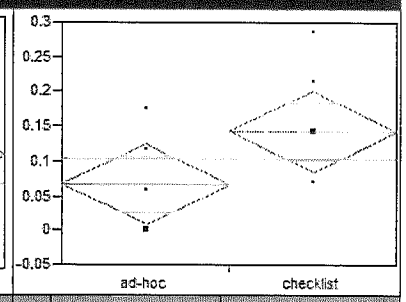
	ad-hoc	checklist
$\bar{X}$	3,250	3,750
σ	2,765	2,435
α	0,7068	



	ad-hoc	checklist
$\bar{X}$	1,125	2,000
σ	1,126	1,195
α	0,1540	

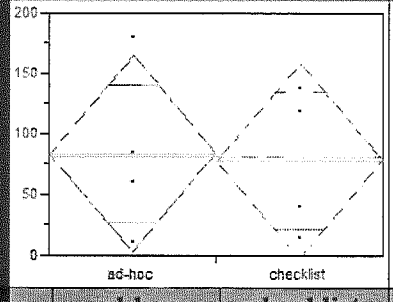


	ad-hoc	checklist
$\bar{X}$	1,187	3,555
σ	1,438	2,937
α	0,0598	

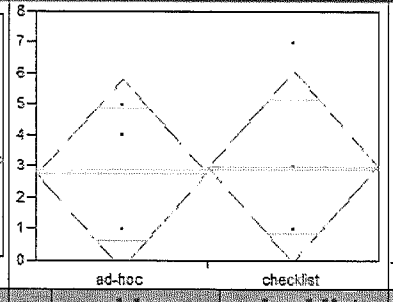


	ad-hoc	checklist
$\bar{X}$	0,066	0,143
σ	0,066	0,085
α	0,0644	

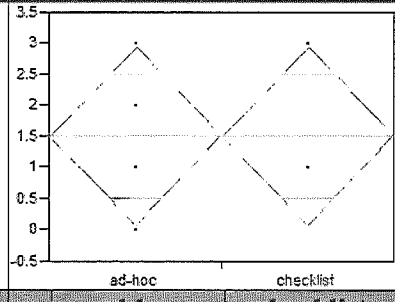
Complexidade Alta (experiente)



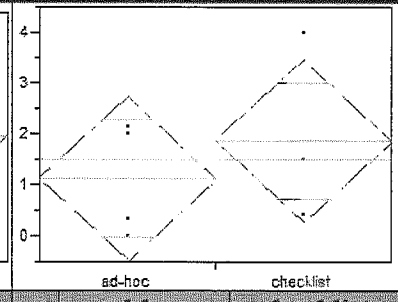
	ad-hoc	checklist
$\bar{X}$	83,750	78,00
σ	70,995	59,705
α	0,9054	



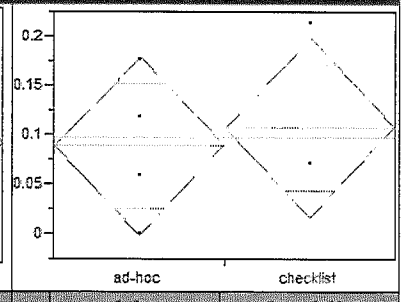
	ad-hoc	checklist
$\bar{X}$	2,750	3,000
σ	2,061	2,828
α	0,8911	



	ad-hoc	checklist
$\bar{X}$	1,500	1,500
σ	1,291	1,000
α	1,0000	

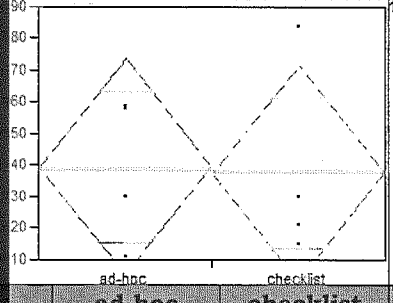


	ad-hoc	checklist
$\bar{X}$	1,118	1,860
σ	1,109	1,514
α	0,4590	

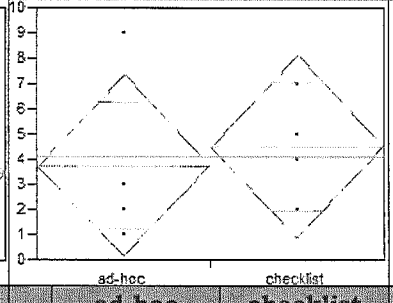


	ad-hoc	checklist
$\bar{X}$	0,088	0,107
σ	0,075	0,071
α	0,7294	

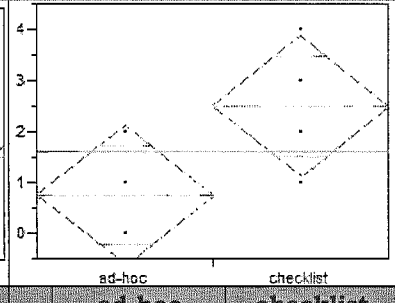
Complexidade Alta (inexperiente)



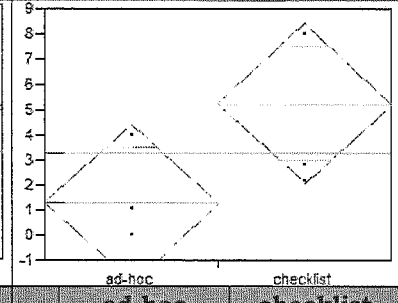
	ad-hoc	checklist
$\bar{X}$	39,500	37,500
σ	23,274	31,607
α	0,9222	



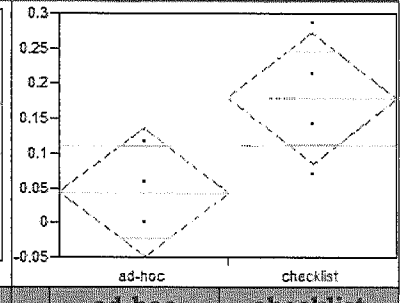
	ad-hoc	checklist
$\bar{X}$	3,750	4,500
σ	3,594	2,082
α	0,7304	



	ad-hoc	checklist
$\bar{X}$	0,750	2,500
σ	0,957	1,291
α	0,0723	

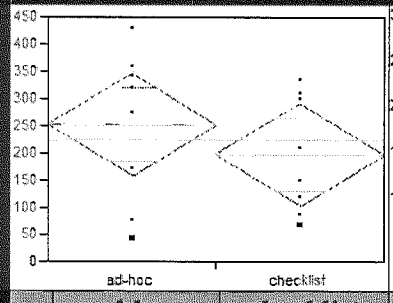
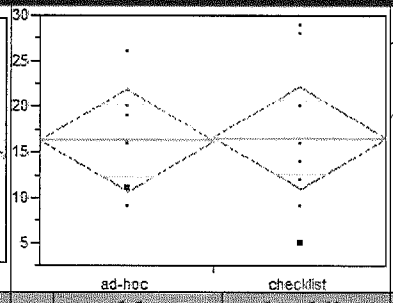
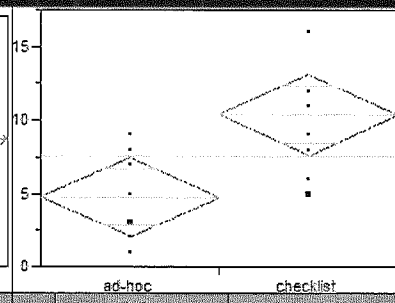
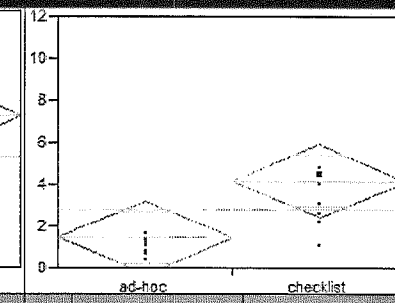
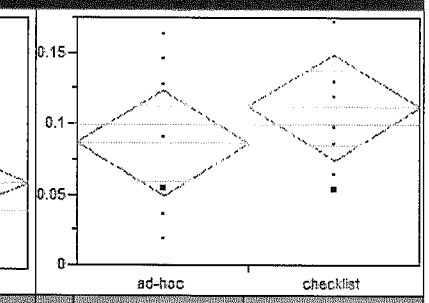


	ad-hoc	checklist
$\bar{X}$	1,258	5,250
σ	1,891	3,189
α	0,0747	

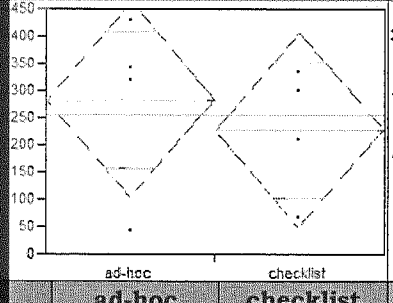
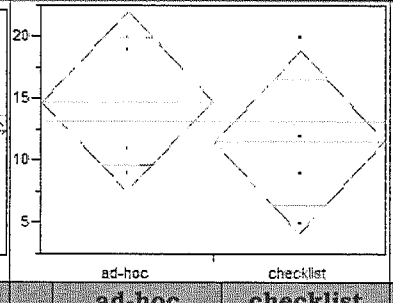
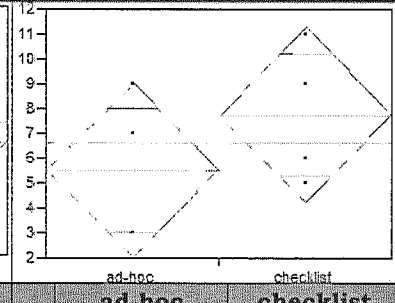
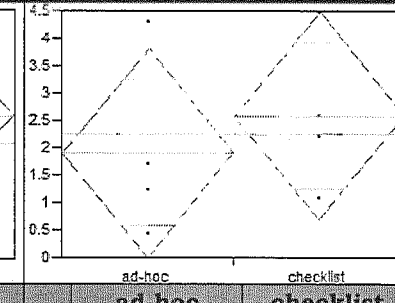
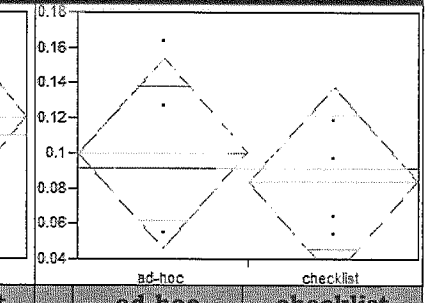


	ad-hoc	checklist
$\bar{X}$	0,044	0,178
σ	0,056	0,092
α	**0,0472**	

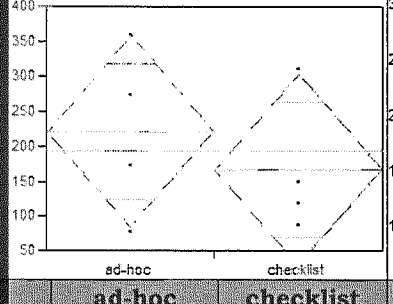
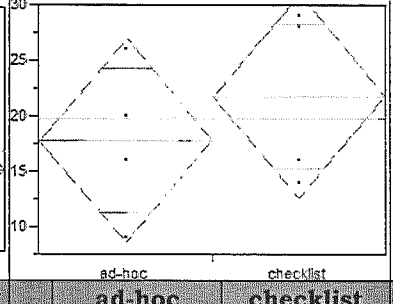
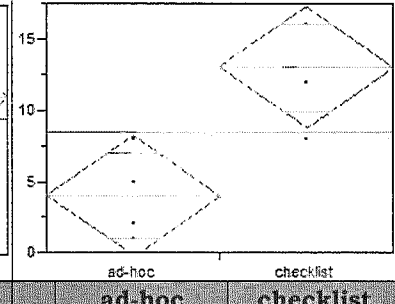
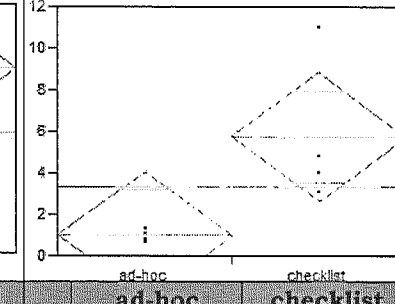
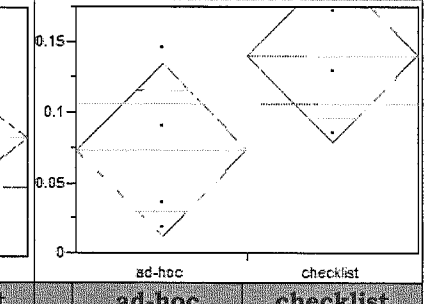
Agregado (total)

	Tempo	Discrepância	Defeito	Custo-Eficiência	Cobertura-Eficiência
					
	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist
$\bar{X}$	252,250 / 197,125	16,250 / 16,650	4,750 / 10,375	1,439 / 4,161	0,086 / 0,112
σ	140,187 / 106,658	6,135 / 8,585	2,964 / 4,173	1,219 / 3,040	0,054 / 0,045
α	0,3910	0,9214	**0,0077**	**0,0339**	0,3268

Agregado (experiente)

	Tempo	Discrepância	Defeito	Custo-Eficiência	Cobertura-Eficiência
					
	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist
$\bar{X}$	283,500 / 227,500	14,750 / 11,500	5,500 / 7,750	1,905 / 2,585	0,100 / 0,083
σ	167,914 / 119,053	5,560 / 6,351	3,000 / 2,754	1,674 / 1,414	0,055 / 0,030
α	0,6060	0,4705	0,3115	0,5577	0,6115

Agregado (inexperiente)

	Tempo	Discrepância	Defeito	Custo-Eficiência	Cobertura-Eficiência
					
	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist	ad-hoc / checklist
$\bar{X}$	221,000 / 166,750	17,750 / 21,750	4,000 / 13,000	0,973 / 5,738	0,073 / 0,140
σ	122,706 / 99,547	7,136 / 7,848	3,162 / 3,830	0,294 / 3,597	0,058 / 0,041
α	0,5179	0,4792	**0,0110**	**0,0385**	0,1069

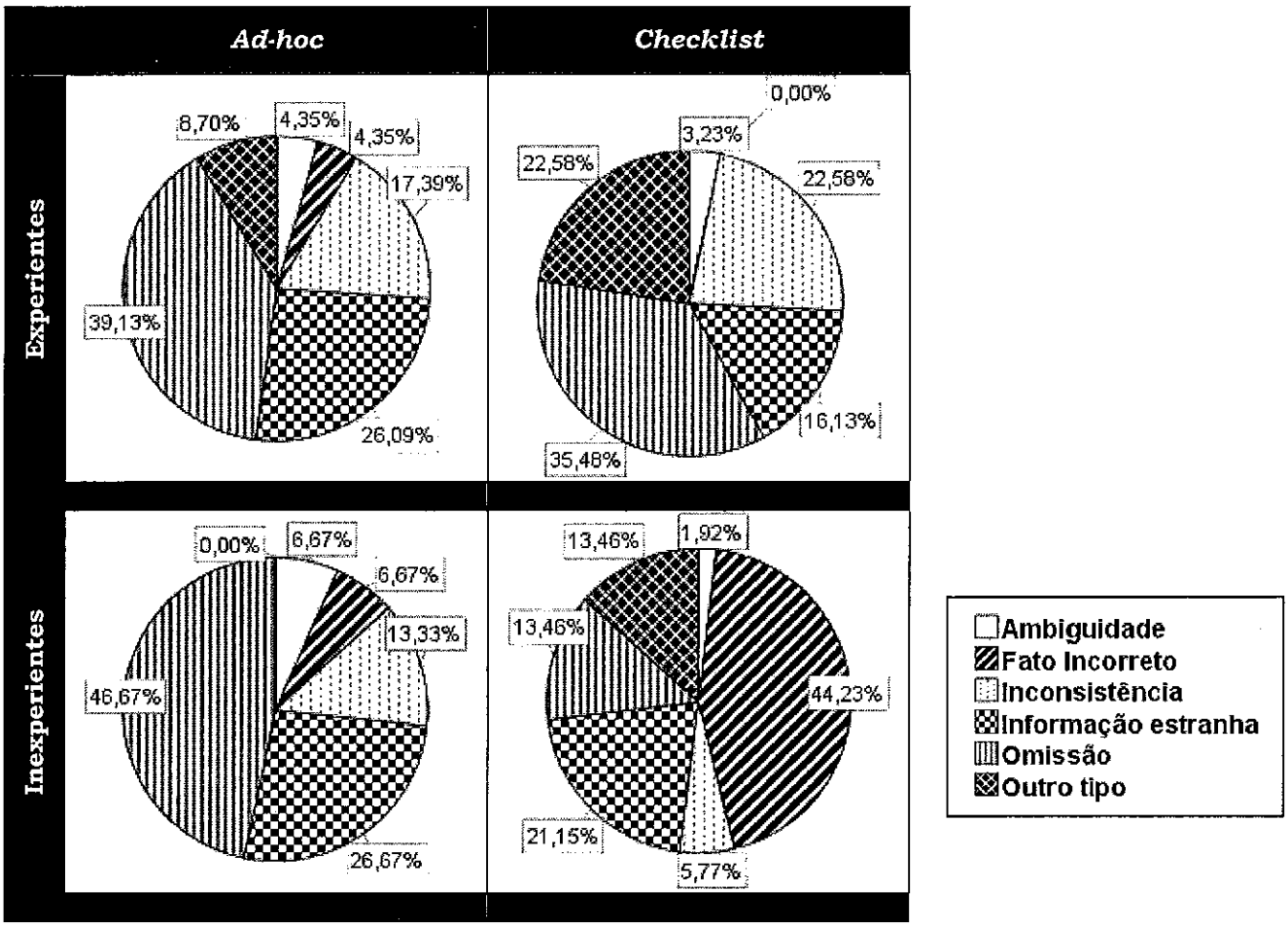
Os resultados apresentados acima mostram, assim como no estudo inicial conduzido, uma superioridade da técnica *checklist* na identificação de defeitos em relação à *ad-hoc* ainda que aponte, em média, o mesmo número de discrepâncias. Não foi possível perceber nenhuma diferença significativa para o tempo gasto entre as técnicas, o que é um indício de que a técnica *checklist* não afeta intensamente a forma de leitura dos modelos de casos de uso ainda que exista um foco maior nas questões tratadas no *checklist*. Em função do registro do tempo pôde ser calculado o custo-eficiência, onde também se observou um bom desempenho da técnica *checklist* influenciado em grande parte pelo um maior número de defeitos identificados. Em resumo, estes resultados mostram que *cheklist* identifica um maior número de defeitos, resultante de um menor número de falso-positivos apontados, em um mesmo intervalo de tempo em comparação com a técnica *ad-hoc*. Para a cobertura-eficiência, apesar de não estatisticamente significativo, identifica-se uma tendência a uma maior cobertura por parte da técnica *checklist* o que representa um maior número de duplicatas (defeitos identificados por mais de um inspetor) conseqüência, possivelmente, do maior foco da técnica *checklist*.

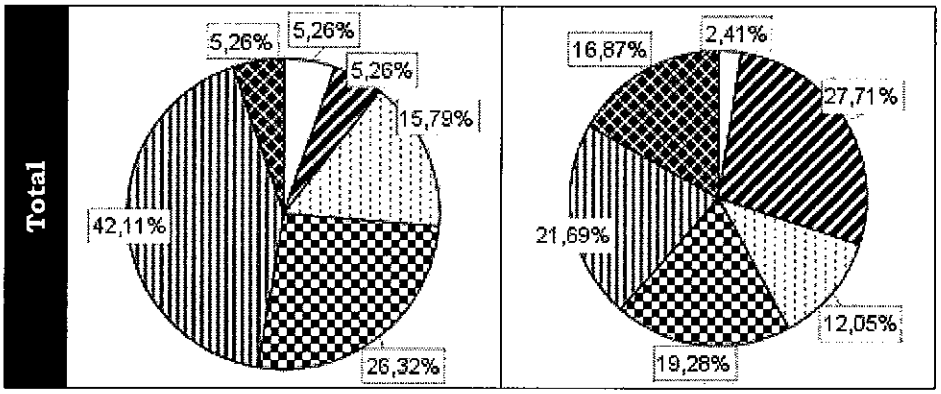
Considerando a classificação por complexidade dos casos de uso não houve, assim como no estudo inicial, nenhuma diferença expressiva entre os três diferentes grupos. De maneira geral, os três grupos apresentam uma tendência semelhante. Entretanto, para casos de uso de complexidade alta não foi possível obter um resultado estatisticamente significativo para a variável defeito. Isto pode sugerir uma limitação da técnica quanto ao tamanho do caso de uso inspecionado, mas existe um possível fator de confusão associado a esta situação, que é o fato de que apenas um caso de uso de complexidade alta ter sido inspecionado em cada rodada – contra dois casos de uso das outras complexidades (Tabela 20) – além do fato do reduzido número de participantes. Outro caso de destaque que destoa dos outros é o tempo para casos de uso de complexidade média. Neste caso, existiu uma diferença grande de comportamento em relação às complexidades baixa e alta, pois se observa um tempo menor para a técnica *checklist*. Não há base sólida, diante dos resultados apresentados, para avaliar a origem desta diferença, mas possíveis hipóteses são o tamanho do caso de uso e características como número de regras de negócio e fluxos alternativos que também estão associadas à idéia de tamanho. Particularmente no contexto deste estudo, os

casos de uso com complexidade média caracterizavam-se por um grande número (relativo) de fluxos alternativos pequenos e regras de negócio simples.

Por fim, é necessário discutir o resultado mais característico em relação ao estudo inicial que é a existência de uma diferença significativa de comportamento entre as técnicas quando a análise é feita considerando a experiência dos inspetores. Nesta perspectiva é possível perceber que a técnica *checklist* é mais útil aos inspetores menos experientes do que aqueles com mais experiência. Isto parece ser um indício de que os inspetores mais experientes consideraram todas ou grande parte das preocupações presentes nas questões do *checklist* durante a realização da inspeção *ad-hoc*. No sentido contrário, outro fator que pode explicar este resultado é que a experiência do inspetor pode tê-lo feito apontar um número idêntico de defeitos em ambas as técnicas, mas defeitos diferentes. Tenta-se avaliar esta questão, adiante, por meio da análise das distribuições dos defeitos.

Tabela 22 - Distribuição de defeitos das inspeções *ad-hoc* e *checklist* classificado pela experiência dos inspetores





Dois fatos interessantes podem ser observados através da distribuição de defeitos apresentada na Tabela 22. O primeiro é que, olhando apenas para a coluna *ad-hoc*, percebe-se que tantos os mais experientes quanto os menos têm distribuições semelhantes, sugerindo que apesar da inspeção ter sido *ad-hoc* o foco foi bastante homogêneo entre todos os inspetores. O segundo, agora comparando *ad-hoc* e *checklist* (as duas colunas da tabela), é que para os mais experientes a distribuição manteve-se razoavelmente similar enquanto que para os menos experientes muda sensivelmente. No caso dos menos experientes, os tipos "fato incorreto" e "outro tipo" ampliaram expressivamente em detrimento, principalmente, dos defeitos do tipo "omissão". Em uma análise qualitativa superficial dos defeitos percebeu-se que os defeitos do tipo "omissão" estavam mais associados ao domínio do negócio e do sistema, enquanto que "fato incorreto" e "outro tipo" foram os tipos mais usados para classificar questões apontadas diretamente pelo *checklist*. Este mesmo padrão de diferença entre as distribuições *ad-hoc* e *checklist* pode ser percebido para os mais experientes, porém de forma bastante tênue, mas neste caso os mais experientes usaram o tipo "inconsistência" para classificar questões apontadas pelo *checklist* em grande parte dos casos em vez de "fato incorreto". Parece que os menos experientes entenderam que um não atendimento ao *checklist* representava um fato incorreto enquanto que os mais experientes classificaram corretamente.

Em suma, diante da análise dos resultados apresentados, é possível argumentar que a técnica *checklist* auxilia na identificação defeitos singulares (diferentes da abordagem *ad-hoc*) em modelos de casos de uso, tornando-os em princípio mais compreensíveis.

Cabe agora, por fim, analisar a avaliação feita pelos participantes em relação à técnica empregada. Esta avaliação foi capturada em um questionário pós-estudo – basicamente o mesmo aplicado no estudo inicial, porém com

duas questões adicionais (questões 8 e 9) visando investigar a percepção dos inspetores em relação ao aprendizado com o uso da técnica *checklist* para a detecção de defeitos, de forma a complementar os dados quantitativos analisados até o momento. A Tabela 23 sintetiza os dados qualitativos organizando-os em escala nominal, excetuando-se as questões adicionais que são discursivas e serão discutidas posteriormente.

Tabela 23 - Resultado do questionário pós-estudo

Questão	Opções e Resultados				
Qualidade e utilidade do *checklist*					
Como você classifica o grau de dificuldade da aplicação do *checklist*?	Muito Difícil	Difícil	Médio	Fácil	Muito Fácil
	0	1	2	7	0
Você seguiu o checklist completamente?	Não	Parcialmente	Completamente		
	0	0	10		
As questões do checklist fizeram sentido como apoio à identificação de defeitos?	Não	Parcialmente	Completamente		
	0	5	5		
Considerando a perspectiva de qualidade adotada, como o checklist auxiliou na identificação de defeitos nos casos de uso?	Obstáculo	Neutro	Auxiliou		
	1	0	9		
Você utilizaria o checklist em inspeções futuras?	Não	Talvez	Sim		
	0	3	7		
Na sua opinião, como o checklist poderia ser melhorado?	Não	Sim			
• Ordem das questões adequada?	1	9			
• As questões foram claras? Puderam ser entendidas?	2	8			
• Número de questões adequado?	5	5			
Condução do estudo controlado					
Como você avalia o treinamento conduzido?	Insuficiente	Neutro	Adequado		
	0	0	10		
Você achou o tempo de inspeção adequado?	Não	Sim			
	0	10			
Como você classifica a qualidade do documento de requisitos utilizado na inspeção?	Má qualidade	Neutra	Boa qualidade		
	0	1	9		

O resultado acima é praticamente idêntico ao identificado no estudo inicial e assim, de maneira geral, as mesmas observações se aplicam. Todos informaram ter seguido completamente as instruções do *checklist* o que torna mais sólido e consistente os resultados quantitativos analisados. Um fato, entretanto, que tem um significado importante, foi a visão, por um dos

inspetores, de que o *checklist* atuou como obstáculo à identificação de defeitos. Este inspetor está entre os mais experientes. Com relação a isto, o pesquisador pôde presenciar durante reuniões de projeto comentários entre os inspetores demonstrando alguma insatisfação no uso da técnica *checklist*, quando ao segui-la completamente abdicaram da possibilidade de apontar alguns possíveis defeitos não capturados no *checklist*. Talvez isto denote uma incompreensão destes inspetores de que de fato a idéia do *checklist* é justamente focar a inspeção em alguns aspectos tratados pelo próprio *checklist*. Mas, por outro lado, reafirma que a experiência é um fator preponderante ao desempenho do inspetor mesmo com uso de técnicas mais sistemáticas. De fato, este é um comportamento já observado em estudos anteriores (Shull *et al.*, 2000) onde inspetores mais experientes tendem a aplicar heurísticas adquiridas anteriormente combinando-as com possíveis novos aspectos da técnica utilizada. Para equipes experientes, o mais apropriado parece ser utilizar técnicas de inspeção como mecanismo de aprendizado para aspectos específicos de interesse do projeto e, a partir disto, retornar a utilização da técnica *ad-hoc*. Este tema será mais bem elaborado na próxima seção, onde as duas questões adicionais do questionário pós-estudo do anexo B.2 mencionadas anteriormente serão exploradas.

4.5. Reflexões e Aprendizado

Esta seção avalia os resultados deste estudo sob a perspectiva do aprendizado obtido pelo projeto com a utilização da técnica de inspeção *checklist*, assim como do ponto de vista científico onde, com respaldo das questões de pesquisa estabelecidas, se tenta posicionar os fenômenos investigados frente a trabalhos da literatura técnica.

4.5.1.Aprendizado

O aprendizado, como resultado da utilização de inspeção no processo de desenvolvimento de *software*, é um efeito reconhecido em diversos trabalhos (Porter e Votta, 1998, Briand et al., 2000, Shull et al., 2000, Fagan, 2001). Particularmente com o uso de técnicas de inspeção não utilizadas anteriormente pelos inspetores, este aprendizado torna-se ainda mais significativo.

Neste sentido, na medida em que os desenvolvedores do projeto não haviam utilizado a técnica *checklist*, existia a expectativa de que o aprendizado, como etapa prevista de um estudo de pesquisa-ação, estivesse

associado a este aspecto. Por esta razão, a questão 9 do questionário pós-estudo foi elaborada como forma de avaliar este aprendizado. Nela, o principal questionamento foi quais dos atributos de qualidade do *checklist* os inspetores mais internalizaram e que, possivelmente, aplicariam no futuro. É importante reiterar que ao responder o inspetor poderia citar mais de um atributo de qualidade. A Tabela 24 computa as citações feitas pelos inspetores em suas respostas.

Tabela 24 - Avaliação do aprendizado por atributo de qualidade[10]

	Experiente	Inexperiente	Total
Compreensibilidade	4	3	7
Completeza	2	3	5
Precisão	1	3	4
Não ambigüidade	-	1	1
Consistência	-	2	2
Independência de ferramenta e interface	-	2	2
Rastreabilidade	2	-	2
Objetividade	-	1	1
Representação de valor	1	-	1
Média de atributos citados por grupo[11]	2	3	

A Tabela 24 apresenta um padrão distinto para inspetores com maior e menor experiência, fortalecendo os resultados quantitativos analisados na seção 4.4. O primeiro aspecto a se observar é que os inspetores menos experientes citaram em média mais atributos de qualidade que os mais experientes (ver ultima linha da tabela), o que sugere que tiveram uma maior oportunidade de aprendizado – o que já era de certa forma esperado. Outro aspecto interessante é que a maior parte dos atributos foi citada apenas por um dos grupos. Para os mais experientes é possível argumentar que os

[10] Para compor os dados da tabela não foram descartados, como para a análise quantitativa, os dois casos atípicos – sendo considerados, portanto, todas as dez respostas dos inspetores. Isto porque, apesar destes dois casos terem sido significativamente distintos quanto ao comportamento observado nos dados quantitativos da inspeção, entende-se que são úteis à análise qualitativa onde o objetivo é capturar a percepção destes indivíduos.

[11] O cálculo é feito dividindo-se o total de citações (soma da coluna experiente/inexperiente) e dividindo por cinco (total de inspetores de cada grupo – experiente/inexperiente).

atributos não citados são potencialmente aqueles que já eram considerados em suas inspeções *ad-hoc*, de fato um deles citou isto explicitamente em sua resposta. Por este motivo, é coerente concluir que estes mesmos atributos (não citados pelos mais experientes) sejam aqueles que recebam maior importância pelos os inspetores menos experientes em um primeiro momento. Desta forma, quando se tornarem experientes também já estarão atentos a estes atributos em inspeções *ad-hoc*.

Cabe ainda mencionar os atributos mais citados considerando todos os inspetores, que foram os atributos compreensibilidade, completeza e precisão. Do ponto de vista do desenvolvedor, papel desempenhado no projeto pela maioria dos inspetores, estes parecem ser os atributos de qualidade que mais afetam as suas atividades quando utilizam os modelos de casos de uso.

Como comentário final, considerando a etapa de aprendizado em um estudo de pesquisa-ação, foi importante observar que todos os envolvidos tiveram a oportunidade de aprimorar a sua capacidade técnica como inspetores. Além disto, do ponto de vista da equipe e do projeto como um todo, os padrões de defeitos identificados, relacionados principalmente a compreensibilidade, completeza e precisão, tenderão a ocorrer em menor intensidade em modelos de casos de uso elaborados futuramente e, mesmo que ocorram, inspeções *ad-hoc* em que alguns dos inspetores que utilizaram a técnica participem poderão identificar estes defeitos mais facilmente.

4.5.2.Reflexões

As questões de pesquisa deste estudo estiveram associadas à investigação do desempenho da técnica de inspeção *checklist* proposta por Gregolin (2007) frente a inspeção *ad-hoc*.

As questões Q.1 e Q.2 puderam ser avaliadas por meio da análise quantitativa, onde foi possível perceber que a técnica *checklist* foi capaz não apenas de identificar um maior número de defeitos, mas, sobretudo, diferentes daqueles identificados em inspeções *ad-hoc* realizadas até então no contexto do projeto. Esta era a principal expectativa em relação à utilização da técnica no projeto. Ainda assim, apesar de em princípio este objetivo ter sido alcançado, a avaliação qualitativa do questionário pós-estudo, particularmente relacionadas às respostas da questão 8, mostrou que uma parcela dos inspetores, principalmente os experientes, sentiu-se limitada pela aplicação do

checklist. Na verdade, a percepção do pesquisador em relação a esta idéia de limitação por parte dos inspetores, é que ela esteve associada ao fato de que o *checklist* foca mais questões sintáticas do que semânticas. No entanto, um modelo de casos de uso semanticamente aderente, mas com deficiências sintáticas como, por exemplo, imprecisões e inconsistências, pode provocar tantos defeitos em etapas seguintes do processo de desenvolvimento quanto um modelo de casos de uso com características opostas.

O fator experiência revelou-se neste estudo como elemento principal da análise dos resultados. Os menos experientes puderam ser mais eficientes com o uso da técnica *checklist* do que os mais experientes. De qualquer maneira, em termos de aprendizado, todos tiveram algum aproveitamento, o que mostra, como mencionado anteriormente, que o emprego de técnicas de inspeção pode ser uma ferramenta útil mesmo que não seja aplicado em inspeções posteriores.

De fato, este resultado é coerente com hipóteses sugeridas por Shull *et al.* (2000) e Anda e Sjøberg (2002), onde se aponta o fator experiência para explicar a não diferença entre técnicas *ad-hoc* e *checklist* em estudos anteriores (Cheng e Jeffery, 1996, Lanubile e Visaggio, 1996, Miller *et al.*, 1998). Da mesma forma, esta hipótese também deveria ser avaliada quando há diferença a favor de *checklist* como outros estudos mostram (Wohlin *et al.*, 2002, Cox *et al.*, 2004). No entanto, em nenhum destes dois trabalhos (Shull *et al.* 2000, Anda e Sjøberg, 2002) foi feita uma avaliação experimental desta questão. No trabalho de Anda e Sjøberg (2002) esta hipótese surgiu devido à observação de que no estudo executado por eles, onde nenhuma diferença entre *ad-hoc* e *checklist* foi identificada, todos os participantes eram experientes.

Todavia, ao contrário de Anda e Sjøberg (2002), o grupo de inspetores deste estudo tinha experiência bastante heterogênea, o que permitiu a avaliação experimental que culminou na confirmação da hipótese. Não foi possível identificar na literatura técnica nenhum trabalho que tenha investigado esta questão como foi neste estudo. Em Maldonado *et al.* (2006) o fator experiência foi avaliado experimentalmente, mas considerando técnicas de leitura e não *checklist*. Nele, não é identificada diferença de desempenho para a técnica de leitura PBR (*Perspective-based reading*) (Basili *et al.*, 1996) entre inspetores de diferentes níveis de experiência.

A questão de pesquisa Q.3 não pôde ser avaliada neste estudo, pois não foi possível capturar dados que permitissem estudar se a melhoria da compreensibilidade[12]/inteligibilidade dos modelos de casos de uso ocorre como efeito do uso do *checklist* de Gregolin (2007). É claro que o fato de defeitos terem sido identificados e, a princípio, removidos dos modelos de caos de uso, sugere que houve alguma melhoria de qualidade nos modelos. Mas, avaliar se a compreensibilidade/inteligibilidade dos modelos foi afetada exige uma investigação mais precisa e focada como, por exemplo, a planejada nas questões Q.3.1 e Q.3.2 além do instrumento da Tabela 19. Ainda que isto não tenha sido possível, os resultados da Tabela 24 indicam os atributos de qualidade que mais afetam os desenvolvedores que utilizam modelos de casos de uso são compreensibilidade, completeza e precisão. Como estes atributos parecem ter sido focados pelos inspetores e, ao mesmo tempo, defeitos associados a estes atributos foram identificados, usando o raciocínio anterior – que remoção de defeitos melhora a qualidade – é razoável supor que a compreensibilidade/inteligibilidade dos modelos de casos de uso tenha melhorado (do ponto de vista dos desenvolvedores). Entretanto, isto é uma hipótese que deve ser avaliada em estudos futuros.

4.6. Conclusão

Este segundo estudo foi significativamente diferente do primeiro em termos metodológicos, mas ainda assim ambos caracterizam-se como estudos de pesquisa-ação. Primeiramente, fez-se uso mais intenso de dados quantitativos para avaliar os efeitos da tecnologia adotada. Além disto, ao contrário do primeiro estudo, a tecnologia empregada já se encontrava disponível na literatura técnica, resultando na necessidade da execução de um estudo controlado para permitir uma avaliação preliminar.

Olhando sob o prisma da experiência na utilização da metodologia, este estudo envolveu um número maior de participantes diretos com diferentes responsabilidades, exigindo uma atuação mais diplomática, ou mesmo política, por parte do pesquisador. Neste sentido, na medida em que diferentes percepções sobre o problema estavam envolvidas – por exemplo, os autores dos documentos minimizarem o problema da compreensibilidade dos modelos

[12] Neste caso, refere-se à compreensibilidade como características gerais necessárias à facilitação do entendimento de modelos de casos e não especificamente das características estabelecidas por Gregolin (2007) para o atributo de qualidade compreensibilidade.

de casos de uso – o pesquisador buscou manejar estas visões no sentido de trazer todos à mesma perspectiva para figurar a importância e alcance da situação enfrentada, especialmente, considerando a organização distribuída da equipe.

Estas questões e outras relacionadas a ambos os estudos serão discutidas no próximo capítulo.

Capítulo 5
Utilizando a Pesquisa-Ação em Engenharia de Software

Neste capítulo, a metodologia da pesquisa-ação é examinada no contexto de Engenharia de Software a partir da análise de sua aplicação nos capítulos anteriores.

5.1. Considerações Iniciais

Se bem planejada e executada, a pesquisa-ação, com o seu duplo objetivo de melhoria dos problemas organizacionais e geração de conhecimento científico, leva a um cenário "ganha-ganha" tanto para os profissionais (e organização) quanto para o pesquisador. Um modelo "canônico" para a condução da pesquisa-ação foi apresentado no Capítulo 2. De maneira geral, segundo este modelo – o qual serviu de base para os estudos #1 e #2 –, a pesquisa se inicia com a determinação das dimensões do problema em um contexto particular, e com base na evidência disponível uma possível solução é formulada a qual é, então, transformada em ação com objetivo de resolver o problema. Por fim, pesquisa é utilizada para investigar e avaliar as ações implementadas.

O objeto central da pesquisa-ação é o próprio processo de mudança o qual o pesquisador não só observa, mas também participa. Isto faz da pesquisa-ação especialmente apropriada para investigar fenômenos que não são homogêneos ao longo do tempo (Checkland e Holwell, 1998), ou seja, não se mantêm exatamente iguais cada vez que ocorrem e, portanto, não são reproduzíveis. Normalmente, estes tipos de fenômenos são eventos sociais como, por exemplo, grande parte das atividades de Engenharia de Software. Neste sentido, para que a pesquisa-ação seja bem conduzida e proporcione resultados relevantes, impõe ao pesquisador conhecimento e habilidades adicionais (Baskerville, 1999). Primeiro, é necessário ao pesquisador um profundo conhecimento dos processos atuais da organização além da cultura organizacional. Em segundo lugar, o pesquisador deve ser hábil em termos da interpretação e entendimento das observações de campo; o planejamento e condução das intervenções; a coleção, análise e interpretação dos dados após as intervenções; a elaboração de conceitos e teorias, e a construção de explanações teóricas; e a formação da colaboração com as pessoas e organização, lidando, inclusive, com questões éticas.

Grande parte deste conhecimento e habilidades é exigida a todo o momento – e "em tempo real" – ao longo da condução da pesquisa na medida em que o desempenho da solução proposta depende diretamente das constantes decisões do pesquisador. Estas decisões são cruciais tanto para permitir a geração de um conhecimento genuinamente científico quanto para conciliar este objetivo com as necessidades de negócio da organização. São ainda mais decisivas se considerarmos que na pesquisa-ação o cenário de uma intervenção se dá em um ambiente real onde os resultados da pesquisa afetam diretamente a organização. Acrescenta-se a isto o fato de que, ao contrário dos estudos controlados, o objeto observado não se encontra inicialmente pronto, mas é construído ao longo da pesquisa junto com os participantes. Desta forma, ainda que o planejamento da pesquisa-ação seja fundamental, o seu alcance é limitado pelas necessidades de improvisação feitas pelo pesquisador no dia a dia.

Para McKay e Marshall (2007), estas constantes decisões enfrentadas pelo pesquisador não são insignificantes, pois apesar de num primeiro momento parecerem sem conseqüência, cumulativamente têm impacto, possivelmente negativos, aos esforços de pesquisa ou na solução do problema. Um exemplo deste tipo de decisão, no estudo #1, foi gravar e transcrever as entrevistas e utilizar a abordagem *grounded theory* para entender os efeitos da refatoração na explicitação do conhecimento tácito em relação aos padrões de codificação no contexto da organização investigada. No início, só havia a expectativa em se fazer anotações durante as entrevistas para então tentar produzir algum conhecimento científico a partir da análise deste material. Posteriormente, a decisão pelo uso da abordagem *grounded theory* mostrou-se muito útil à construção da teoria.

No entanto, para um pesquisador, apenas estudar os vários diagramas e descrições sobre pesquisa-ação, assim como aqueles apresentados no Capítulo 2, pode deixá-lo com uma visão distante da realidade daquilo que uma intervenção de pesquisa-ação representa (McKay e Marshall, 2007). De fato, alguns pesquisadores argumentam que a pesquisa-ação está mais próxima de uma estratégia do que metodologia ou técnica (Heatwole *et al.*, 1976) e questionam se a pesquisa-ação não poderia representar um paradigma de pesquisa (Lau, 1999). Como conseqüência, a pesquisa-ação é normalmente definida e apresentada na forma de recomendações gerais –

como, por exemplo, realização simultânea de pesquisa e ação, pesquisa colaborativa e aprendizado por meio da reflexão – que deixam em aberto questões importantes em termos do conhecimento e habilidades necessários à prática da pesquisa-ação. Outro ponto importante, ainda do ponto de vista do pesquisador, é que em disciplinas que tradicionalmente utilizam-se de métodos quantitativos de pesquisa, existe um maior risco de que uma metodologia como a pesquisa-ação seja má interpretada devido ao viés naturalmente introduzido e, desta forma, aplicada de maneira incorreta.

Neste sentido, este capítulo visa discutir a aplicabilidade da pesquisa-ação em Engenharia de Software e apresentar uma adaptação da metodologia provendo um guia claro sobre como aplicá-la no contexto de Engenharia de Software, considerando as questões discutidas anteriormente. Tomar-se-á como base para isto, a experiência na condução dos estudos #1 e #2 descritos nos capítulos anteriores. Além disto, para orientar a decisão das principais questões a serem abordadas, serão utilizados trabalhos da literatura técnica que também trataram a adaptação da pesquisa-ação, porém em disciplinas diferentes. Diversas áreas já trabalharam este tema incluindo Enfermagem (Holter e Schwartz-Barcott, 1993, Meyer, 1993), Ciências Políticas (Heatwole *et al.*, 1976), Administração (Ottosson, 2003), Gerência de Produção e Operações (Westbrook, 1995), Logística (Näslund, 2002), Marketing (Ballantyne, 2004, Kates e Robertson, 2004) e Comunicação (Hearn e Foth, 2004).

De maneira geral, estes trabalhos podem ser divididos em dois tipos. Aqueles que argumentam sobre os potenciais benefícios que a utilização da pesquisa-ação pode trazer as suas respectivas disciplinas (Heatwole *et al.*, 1976, Holter e Schwartz-Barcott, 1993, Näslund, 2002, Kates e Robertson, 2004, Hearn e Foth, 2004), o que de certa forma foi feito no Capítulo 2 e será complementado neste capítulo. E aqueles que propõem algum tipo de adaptação/adequação segundo as particularidades de suas áreas (Meyer, 1993, Ottosson, 2003, Westbrook, 1995, Ballantyne, 2004), que será o foco maior deste capítulo. É bastante interessante perceber como cada disciplina se utiliza da pesquisa-ação com um determinado foco. Por exemplo, Meyer (1993) preocupa-se com questões éticas, relacionadas ao consentimento da participação na pesquisa, em um contexto caracterizado por um grande número de especialistas diferentes e uma intensa rotatividade de profissionais como é o caso de muitas enfermarias; Ballantyne (2004) aponta a importância

do estabelecimento do espírito colaborativo por meio da pesquisa-ação – em um ambiente marcado por atividades que dependem da criatividade – como base para elaboração e disseminação sobre estratégias de *marketing* de uma organização entre seus funcionários; e Hearn e Foth (2004) destacam a capacidade da pesquisa-ação em revelar o conhecimento tácito embutido em uma organização, que deseja se expor em novas mídias de comunicação, de forma a capturar a sua identidade.

Ainda que cada disciplina tenha focado em aspectos particulares da pesquisa-ação, no conjunto, os trabalhos tratam um número limitado de características. Por esta razão, estas características serão utilizadas como linha base para a argumentação sobre como pensar a pesquisa-ação em Engenharia de Software. Os seguintes tópicos serão abordados no restante deste capítulo:

1) Quais são as habilidades essenciais necessárias aos pesquisadores?
2) Quais aspectos relevantes no gerenciamento do processo de pesquisa?
3) Como a cultura da organização influencia a reflexão e a colaboração?
4) Quais os possíveis conflitos éticos gerados pela condução da pesquisa?
5) Como novo conhecimento é gerado?
6) Quais são os benefícios da pesquisa-ação à construção de teorias?
7) Como o aprendizado pode ser extraído das ações?
8) É possível pensar na prática da Engenharia de Software baseada em evidência com a pesquisa-ação?
9) Quais são os principais formatos que a pesquisa-ação pode assumir em Engenharia de Software?

A próxima seção traz uma síntese da experiência do autor na iniciação e condução dos estudos de pesquisa-ação, considerando fatores como aprendizado e dificuldades, com objetivo de alertar sobre as possíveis armadilhas e aspectos importantes na condução da pesquisa. Em seguida, apresenta-se um roteiro detalhado sobre como a pesquisa-ação pode ser aplicada em Engenharia de Software. Em uma seção posterior, a organização utilizada nos estudos anteriores é explorada como um modelo para relatar estudos de pesquisa-ação. Por fim, conclui-se o capítulo com comentários sobre a aplicabilidade da pesquisa-ação em Engenharia de Software.

5.2. Avaliação dos Estudos Conduzidos

5.2.1.Aprendizado e Dificuldades na Utilização da Metodologia

Iniciar um estudo de pesquisa-ação em Engenharia de Software não é uma atividade trivial. Um primeiro obstáculo que o pesquisador deve transpor é a ausência de material específico acerca da pesquisa-ação em Engenharia de Software. Na falta de trabalhos técnicos específicos sobre o tema, o pesquisador encontra-se em uma situação desconfortável, onde deve não apenas buscar compreender como utilizar a metodologia, mas também encontrar uma forma de aplicá-la em Engenharia de Software. É importante lembrar, neste contexto, que a pesquisa-ação surgiu e é predominantemente utilizada em ciências sociais e ainda que a Engenharia de Software possua um forte componente social, a realidade de pesquisa é bastante distinta, bem mais voltada para o lado técnico e tecnológico. Como já foi mencionado anteriormente, um dos principais objetivos deste capítulo e deste trabalho como um todo é justamente tratar esta questão.

Após compreender os principais benefícios e limitações da metodologia, o pesquisador deve então avaliar se é adequada aos propósitos da pesquisa. Neste momento, o mais importante é certificar-se de que os princípios da pesquisa-ação poderão ser cumpridos: mudança por meio da ação e aprendizado através da reflexão. Todavia, em certos casos, pode ser difícil diferenciar um estudo de pesquisa-ação de um estudo de caso. Isto parece ser particularmente mais evidente para os estudos de introdução de novas tecnologias – como foi o estudo #2 –, pois há um foco maior na observação do que na ação em alguns momentos. De qualquer maneira, se houver intenção da mudança na cultura da organização e uma preocupação com a conscientização ampla dos seus membros sobre os problemas enfrentados e quais soluções foram adotadas, pode-se adotar a pesquisa-ação como metodologia de investigação.

No entanto, a análise da adequação aos propósitos da pesquisa pode não ser possível antes de um diagnóstico preciso e, eventualmente, até mesmo do início do planejamento. É apenas durante a etapa de diagnóstico que o pesquisador tem a oportunidade de examinar em detalhes o problema a ser tratado e, durante o planejamento, idealizar soluções. Por exemplo, durante as reuniões de projeto do estudo #1, das quais o autor participava no papel de profissional e pesquisador, os desenvolvedores mais experientes do projeto

mostraram alguma insatisfação em relação à qualidade do código produzido pela equipe remota que havia sido recentemente integrada ao projeto. Diante de situações como esta, o pesquisador deve ter sensibilidade apurada para perceber uma possível oportunidade de pesquisa. Do ponto de vista científico, esta sensibilidade está associada ao conhecimento do estado da arte sobre o tema investigado, além de um entendimento dos fundamentos do método científico, principalmente no que se refere ao pensamento crítico e reflexivo (e.g., formulação de hipóteses e suas possíveis respostas que levarão a forma como a intervenção será dirigida). Já do ponto de vista da prática, está associada a um conhecimento técnico refinado sobre o problema a ser tratado de forma a possibilitar o pesquisador adotar uma postura pragmática baseada em sua experiência. Talvez, a exigência deste grande número de aptidões faz com que o número de estudos que explorem a pesquisa-ação como metodologia ainda seja baixo em Engenharia de Software – como foi visto na revisão da literatura apresentada no Capítulo 2 –, ainda mais pelo fato de que engenheiros de software não têm histórico de serem reflexivos sobre suas ações (Rus e Lindvall, 2002).

Desta forma, cabem às atividades de diagnóstico e planejamento da pesquisa permitir ao pesquisador evoluir o seu sentimento inicial sobre a necessidade de investigação do problema na medida em que possibilitam a identificação das causas primárias da situação enfrentada pela organização e o arranjo de uma possível solução, respectivamente. No exemplo do estudo #1 citado anteriormente, o diagnóstico permitiu a hipotetização da existência do conhecimento tácito a partir da análise da estruturação das equipes (local e remota) e as suas implicações à comunicação de maneira remota entre seus membros, que possivelmente levou a situação diagnosticada (má qualidade do código). Contudo, apenas durante a etapa de planejamento, na revisão da literatura, é que se teve a possibilidade de verificar que o tema ainda havia sido pouco explorado. Na verdade, nenhum trabalho mostrou-se tratar especificamente o problema enfrentado, mas, por meio do conhecimento técnico do pesquisador ao qual nos referimos anteriormente e diante dos trabalhos mais próximos ao tema de pesquisa, foi possível optar pela refatoração de código como um caminho viável. É importante notar que até este momento do planejamento o tema de pesquisa ainda não havia sido completamente definido (i.e., utilizar refatoração de código para externalizar o conhecimento tácito em relação aos padrões de codificação) e,

conseqüentemente, o diagnóstico precisou ser revisitado – principalmente, na definição do tema de pesquisa. Isto caracteriza uma propriedade importante do processo da pesquisa-ação, que é a possibilidade de interatividade entre as etapas. Por fim, retornando a questão da falta do hábito dos engenheiros de software da reflexão sobre suas próprias ações, pode-se depreender, do que foi discutido em relação às atividades de diagnóstico e planejamento, que o aprendizado da pesquisa-ação pelos profissionais pode ser uma ferramenta interessante para estimulá-los sobre a importância deste hábito. De fato, a experiência mostrou que o exercício da pesquisa-ação pode ser benéfico ao tornar o pesquisador mais reflexivo e crítico em relação aos problemas enfrentados no dia a dia de um projeto de software e, caso profissionais da indústria assumissem um papel de "investigador", a pesquisa-ação poderia ser uma ferramenta para isto. Além disto, o foco pragmático da pesquisa-ação a torna mais próxima da prática do profissional. Esta questão abre espaço para uma análise do papel do profissional na Engenharia de Software Baseada em Evidência (Kitchenham *et al.*, 2004) e será discutida adiante neste capítulo.

Um outro aspecto que merece atenção durante o planejamento é a coleção de dados. A programação prévia dos dados que serão coletados alivia a exposição do pesquisador a eventos inesperados, permitindo-o focar no acompanhamento das ações e participar das intervenções. É claro que situações imprevistas irão ocorrer, mas a metodologia da pesquisa-ação acomoda estes tipos de desvios. No entanto, o pesquisador precisa registrar estes eventos para que depois permita justificá-los, atribuindo assim um maior rigor à pesquisa e relevância dos resultados. A utilização de um diário de pesquisa pode ser bastante útil, neste contexto. Um exemplo deste tipo de desvio no estudo #1 ocorreu quando, após inicialmente planejar que cada desenvolvedor iria revisar todos os métodos listados para revisão, verificou-se que a melhor estratégia seria dividir o trabalho entre os dois desenvolvedores para poupar esforço do projeto. Esta divisão só foi possível pois a categorização das sugestões de refatoração da revisão faria com que os dados fossem agregados independentemente do revisor (desenvolvedor). Desta forma, é importante notar que esta decisão só foi tomada após a avaliação de que não teria impacto à análise dos dados.

Ainda em relação à coleção de dados, deve-se também observar a maneira pela qual os dados serão coletados. As formas automáticas de coleta

(e.g., gravação de voz) garantem tempo adicional para o pesquisador conduzir as ações. Ainda que em um primeiro momento pareça uma questão trivial, no estudo #1 isto não foi previsto inicialmente e fez com que a entrevista com um dos participantes tomasse um tempo maior que o necessário, que seria mais bem utilizado nas atividades do projeto. Em estudos com um grande número de participantes, esta questão torna-se ainda mais sensível.

Como comentário final em relação ao aprendizado, foi interessante perceber como a pesquisa-ação acomoda a utilização de diferentes técnicas de análise de dados, ainda que fosse uma expectativa mesmo antes do início dos estudos #1 e #2 devido aos diferentes objetivos da metodologia com a solução do problema, aprendizado da organização e pesquisa.

5.2.2. Gerenciando as Ações de Pesquisa: Colaboração + Conhecimento Tácito + Reflexão = (Possibilidade de) Novo Conhecimento

Devido as suas características, a pesquisa-ação é intensamente dependente da participação colaborativa dos envolvidos, de maneira que seja possível evidenciar o conhecimento tácito relacionado ao *know-how* das ações desempenhadas. Assim, por meio da análise/reflexão destas ações, pode-se identificar comportamentos relevantes que mostrem o inter-relacionamento – dificilmente de causa-efeito – entre a solução e o problema.

Diferentes fatores podem afetar a colaboração em um estudo. Por ser uma pesquisa com e para os profissionais da organização investigada, o gerenciamento das ações de pesquisa exige atenção especial para a seleção dos participantes. Os participantes precisam estar motivados para participar da pesquisa de forma a permiti-los colaborar genuinamente para o resultado final. Também é conveniente um bom relacionamento entre eles que possibilite criar um ambiente menos constrangedor para a sugestão de novas propostas e que facilite a comunicação espontânea na disseminação do conhecimento ao resto da organização. Evidentemente, nem sempre estas condições podem ser satisfeitas, sendo importante neste momento a habilidade política do pesquisador em conseguir apoio da alta gerência para que as ações sejam implementadas. Segundo Ballantyne (2004), é importante, ainda neste contexto, que o pesquisador mostre comprometimento com os objetivos buscados atuando no mesmo nível e condições dos outros participantes para tentar criar um ambiente de confiança e que lhes dê confiança frente à alta gerência – na medida em que mostra que o pesquisador está do mesmo "lado"

dos participantes. No contexto dos estudos isto foi alcançado naturalmente já que o pesquisador atuava na organização anteriormente.

Todos os participantes entram na pesquisa-ação como aprendizes, ainda que sejam especialistas em algumas áreas. Não obstante, uma variedade de conhecimento e habilidades dos profissionais é crucial e pode incluir não só a sua experiência direta, mas também indireta (e.g., observando outros profissionais). Este tipo de conhecimento é comumente denominado conhecimento tácito e representa o *know-how* que os profissionais trazem para desempenhar suas atividades. Rus e Lindvall (2002) argumentam que o desenvolvimento de software é um processo de *design* onde todos os envolvidos têm um grande número de decisões a serem feitas e, para desenvolver sistemas de software complexos, devem se comunicar de forma que o conhecimento (tácito) individual possa ser disseminado e utilizado no projeto e pela organização como um todo. Neste sentido, o diálogo tem papel importante durante a pesquisa, pois é o veículo condutor do conhecimento tácito e que permite a colaboração entre os participantes.

Durante os estudos conduzidos todos estes elementos, colaboração, conhecimento tácito e diálogo, foram extensamente explorados. O estudo #1 começou com a hipótese da existência de um conhecimento tácito sobre os padrões de codificação (i.e., conhecimento da prática) e que poderia ser explicitado por meio de um processo de refatoração de código. Com isto, o primeiro passo foi fazer com que os participantes – desenvolvedores da organização – refletissem indiretamente sobre a sua prática na medida em que identificavam e registravam desvios dos padrões estabelecidos por eles de maneira não planejada. Esta questão teve que ser tratada com bastante cuidado, pois a equipe remota cujo código estava sendo avaliado tinha acabado de ser integrada ao projeto. Desta forma, tentou-se evitar um clima hostil entre as equipes, informando a equipe remota que práticas visando à qualidade do produto eram comuns e que a responsabilidade do problema não era exclusivamente deles, mas sim do aprendizado de ambas as equipes de se trabalhar de forma remota. Desta forma, como é possível observar, o pesquisador deve sempre estar atento a este tipo de questões organizacionais. Devido a isto e aos fatores anteriormente citados, o problema foi enfrentado naturalmente como mais uma atividade de projeto e o diálogo entre os participantes sobre o problema ocorreu de forma espontânea. A base para o

diálogo foi o registro dos desvios e teve objetivo de tentar solucionar o problema, ou seja, explicitar o conhecimento sobre os padrões de codificação. Com este registro, as equipes puderam interagir colocando o seu ponto de vista e, junto com o pesquisador, decidiram como explicitar o conhecimento e quais os desvios eram os mais relevantes. Tudo isto também foi registrado (e.g., grau de impacto das diretivas de codificação). Resumindo em poucas palavras a importância do diálogo e colaboração, colocado por Schein (1994), o diálogo é a forma de "pensar em conjunto para construir significados compartilhados".

A partir de toda esta manipulação do conhecimento tácito utilizado pelos profissionais e da evidenciação dos seus comportamentos durante a prática das atividades, a reflexão, representada nos estudos pelo aprendizado, torna-se resultado direto das ações praticadas. Claro que deve haver, em alguns casos, um esforço adicional para formatar este resultado de forma a permitir a sua disseminação – como foi o caso do uso do wiki Trac (Trac, 2003) no estudo #1. Mas o mais importante é que a organização tenha consciência de como o problema foi solucionado e que, com isto, possa ser capaz de evitá-lo no futuro. Além disto, cabe ao pesquisador informar aos participantes sobre os resultados da pesquisa e como se relacionam com o aprendizado. Estes resultados da pesquisa buscam mostrar como os efeitos alcançados representam um conhecimento novo ao contrastá-los com trabalhos anteriores da literatura técnica, sempre explorando as ricas camadas de significado do conhecimento tácito e colaboração que a pesquisa-ação é capaz de proporcionar.

No caso do estudo #2, os elementos citados anteriormente parecem não estar presentes à primeira vista, principalmente o conhecimento tácito. Como dito anteriormente, nos casos de introdução de nova tecnologia pode não ser imediato distinguir a pesquisa-ação de estudo de caso e, baseado na experiência com o estudo #2, cabem algumas indicações para se assegurar que os princípios da pesquisa-ação estejam sendo seguidos. Primeiramente, a colaboração entre os participantes da pesquisa ficou entremeada pelo processo básico de inspeção. De fato, não havia sentido planejar outras atividades de coordenação de pesquisa se a própria inspeção já trás consigo um processo bem estruturado. De qualquer maneira, a colaboração e o diálogo estiveram presentes, ainda que em intensidade menor que o estudo #1, principalmente

nos treinamentos iniciais sobre a utilização do *checklist* e, posteriormente, entre inspetores e autores dos documentos inspecionados discutindo sobre a eficácia da tecnologia utilizada e os seus impactos para inspeções futuras. Esta disseminação do aprendizado e uma mudança da cultura da organização, que passou a entender a importância da compreensibilidade dos modelos de casos de uso, são alguns dos aspectos essenciais para caracterizar um estudo de pesquisa-ação. O conhecimento tácito, menos evidente ainda neste estudo #2, serviu como pano de fundo para toda a análise de pesquisa na comparação entre a eficiência do uso do *checklist* pelos desenvolvedores mais e menos experientes. Nesta análise, é como se o próprio conhecimento tácito tivesse sido posto a prova e testado se era pivô da forma como o *checklist* foi utilizado.

Diante dos cenários discutidos, é possível perceber como a colaboração, o conhecimento tácito e a reflexão sobre as ações são elementos importantes para a prática da pesquisa-ação e que permitem a construção de um novo conhecimento. De fato, Rus e Lindvall (2002) comentam que o conhecimento em uma organização pode ser criado/originado do aprendizado, resolução de problemas, inovação e criatividade. A pesquisa-ação pode ser pensada, neste sentido, como um meio de viabilizar este processo de criação do conhecimento trabalhando-o não apenas no contexto da organização, mas tentando também tratá-lo sob uma perspectiva científica. A construção de teorias pode ser útil para o segundo objetivo, como foi feito no estudo #1, e será abordada posteriormente.

5.2.3.Questões Éticas

Por ser uma pesquisa colaborativa e em um contexto real, as questões éticas envolvidas em um estudo de pesquisa-ação tornam-se ainda maiores. Do ponto de vista do pesquisador, pode ser difícil perseguir o seu objetivo paralelo de pesquisa, principalmente considerando os requisitos para publicação em periódicos indexados. Neste sentido, é uma questão de ética, já que o acordo inicial (formal ou não) sempre visa tratar um problema, que a solução do problema nunca seja colocada em segundo plano. Para isto, o pesquisador precisa tentar ao máximo encaixar as tarefas de pesquisa como atividades normais do projeto, e o que não for possível deve ser registrado para posterior análise. Caso isto não seja feito, existe uma chance do comprometimento dos dados gerados para a própria organização. Isto reforça a discussão anterior

sobre a importância da utilização de formas automáticas de coleta de dados e diários de pesquisa, que permitam o pesquisador concentrar-se na resolução do problema. Desta forma, ainda que a pesquisa-ação seja realizada "em tempo real", grande parte da análise é feita em retrospectiva. Ambos os estudos se utilizaram intensamente de formas automáticas ou mesmo "implícitas" de coleta – já que existiriam independentemente de serem coletadas ou não –, como gravação de voz, métricas de código e ferramenta de registro de discrepâncias (no caso da inspeção).

Uma segunda questão é o consentimento à participação das pessoas na pesquisa. Por ser realizada com objetivo da resolução de um problema da organização onde atuam, os profissionais podem se sentir sem outra alternativa já que precisam desempenhar as suas atribuições profissionais. Para amenizar esta situação, também deve ser oferecida aos profissionais a possibilidade de que seus dados não sejam publicados, ainda que sua participação termine sendo, em termos práticos, obrigatória. Neste sentido, o consentimento de livre arbítrio não é de fato alcançado e uma vez que o projeto tenha sido iniciado torna-se ainda mais difícil a este profissional se sentir em uma situação confortável de desistir desta escolha, já que todos os envolvidos inicialmente estão juntos perseguindo o objetivo de mudança. Por este motivo, vale reforçar que o pesquisador deve tentar atuar no mesmo nível e condições dos outros participantes para que se sintam apoiados em suas atividades.

Além destas questões, o pesquisador não descarta das questões éticas comuns a todo esforço de pesquisa como honestidade, privacidade dos participantes na publicação dos resultados, respeito básico aos direitos da pessoa, dentre outras.

5.3. Pesquisa-Ação na Engenharia de Software

Nesta seção, não se pretende propor a adoção da pesquisa-ação como a solução para a pesquisa em Engenharia de Software, mas apenas discutir a sua utilidade neste cenário, ao mesmo tempo em que se tenta mostrar como pode ser aplicada em Engenharia de Software de maneira geral nas organizações de desenvolvimento de software.

5.3.1.Engenheirando Software e Construindo Teorias

Ainda hoje em dia, é comum ao longo do desenvolvimento de software, muitas vezes por conta da pressão do tempo ou por desconhecimento, que soluções *ad-hoc* sejam adotadas. Infelizmente, isto vai ao encontro à definição inicial de Engenharia de Software dada por Barry Boehm em 1976 (Boehm, 1976): "aplicação prática do conhecimento científico no projeto e construção de programas de computadores e a documentação relacionada ao seu desenvolvimento, operação e manutenção". Contudo, a pesquisa-ação oferece a oportunidade de se atender esta expectativa inicial embutida em sua definição, mesmo em contextos reais onde há pressão por resultados rápidos, devido ao seu duplo objetivo de pesquisa e ação.

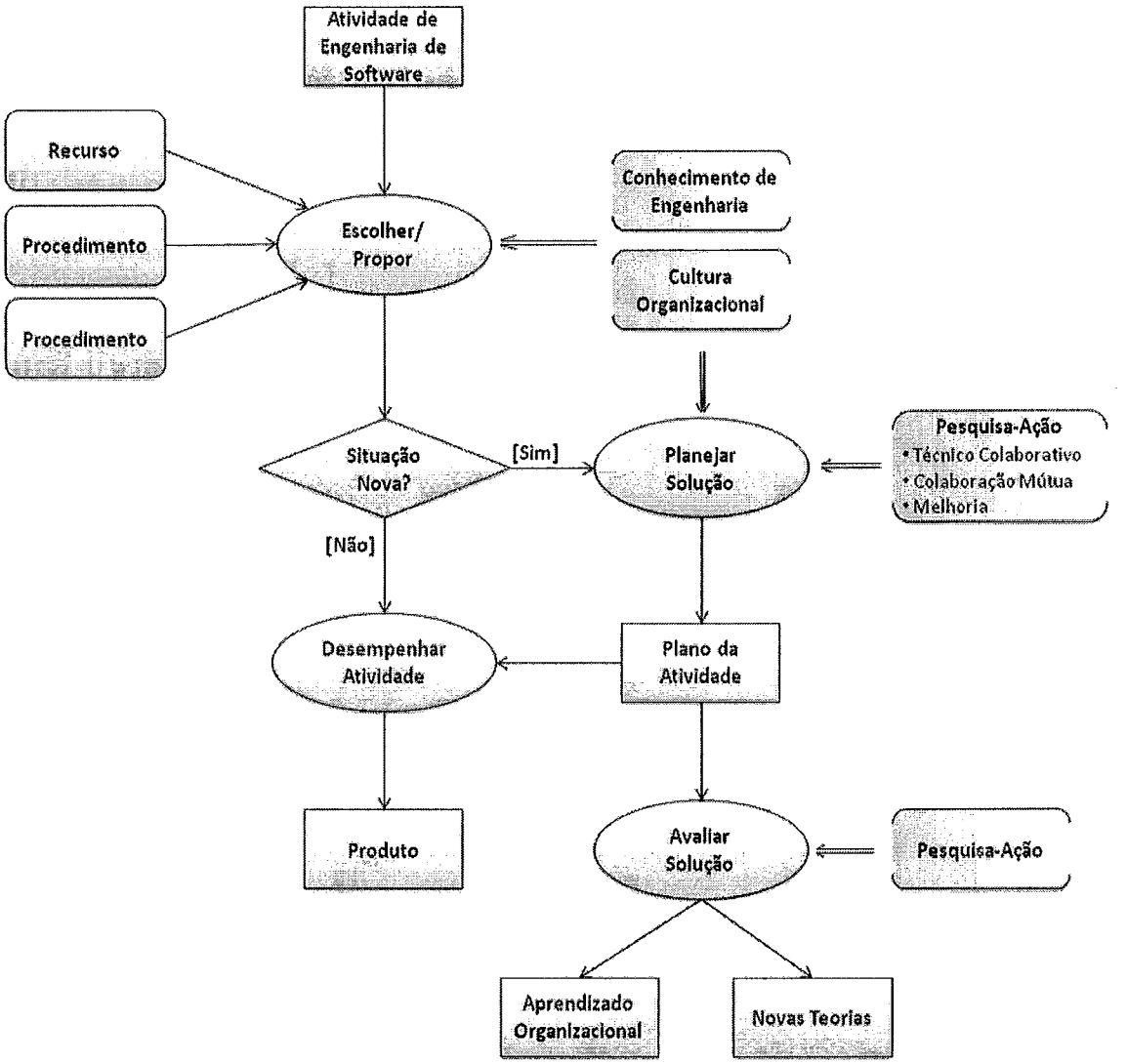


Figura 18 – Geração de novas teorias em um ambiente de Engenharia de Software por meio da pesquisa-ação

A pesquisa-ação permite que os resultados de pesquisa, ainda que em estágios preliminares, possam ser usados na prática para um retorno imediato sobre a sua utilidade. Mesmo quando nenhum resultado científico é diretamente adequado à situação enfrentada, a pesquisa-ação permite que uma solução *ad-hoc* seja elaborada colaborativamente, porém, ao contrário da prática comum em Engenharia de Software, dando atenção especial à reflexão sobre os resultados alcançados. A Figura 18 ilustra este processo.

Segundo Falbo (1998), um processo de software pode ser caracterizado pelas atividades a serem realizadas e os recursos, procedimentos e artefatos a serem utilizados. Em cada atividade, os recursos, procedimentos e artefatos precisam ser definidos, para que o produto desejado seja gerado por meio de um raciocínio de engenharia e influenciado pela cultura organizacional – a Tabela 25 detalha as dimensões desta prática.

Tabela 25 - Termos-chave das dimensões da prática de Engenharia de Software adaptados de Higgs *et al.* (2001)

Prática baseada em evidência pode ser pensada como fundamentar as decisões de engenharia na melhor evidência disponível.
Raciocínio de engenharia é o processo de tomada de decisão associado à aplicação de diferentes técnicas e princípios científicos com o propósito de definir uma tecnologia de *software* que seja detalhada de forma apropriada e viável economicamente de maneira que permita a sua concepção final.
Tomada de decisão de engenharia é uma série de julgamentos realizados pelos engenheiros na interação com o seu ambiente considerando suas restrições (recursos, tempo, perícia, dentre outros).
Julgamento em engenharia consiste em ponderar evidências disponíveis diante do conhecimento relevante do contexto e do domínio.
Conhecimento proposicional é derivado da pesquisa e teoria.
Conhecimento de perícia profissional origina-se de uma rigorosa apreciação e processamento da experiência profissional.
Conhecimento pessoal resulta da experiência de vida pessoal/geral.

Contudo, em algum momento, o conhecimento da Engenharia de Software pode não ser suficiente para uma determinada situação (e.g., pode-se não saber qual o melhor procedimento a ser adotado). Nesta circunstância, a pesquisa-ação pode ser utilizada como meio para tratar o problema. Inicialmente, planeja-se como a atividade será conduzida, por exemplo, buscando-se um procedimento novo na comunidade científica ou optando-se por elaborar o procedimento na própria organização. Depois, a atividade é desempenhada normalmente ao mesmo tempo em que se avaliará a sua execução buscando um aprendizado organizacional e a geração de novas teorias (Figura 18). Os três formatos de pesquisa-ação identificados em Engenharia de Software na revisão do Capítulo 2 foram intitulados de técnico

colaborativo, colaboração mútua e melhoria com base na nomenclatura dada por Holter e Schwartz-Barcott (1993). O primeiro tem o foco em testar uma tecnologia ou uma metodologia em um cenário real; no segundo pesquisador e participantes se unem para identificar problemas, suas causas e possíveis intervenções; e o último visa à facilitação e observação de atividades de Engenharia de Software. Tentando classificar os estudos conduzidos, o estudo #1 teria um perfil próximo ao formato de colaboração mútua e o #2 tem grande similaridade com o técnico colaborativo. Um exemplo do formato melhoria seriam os estudos de melhoria de processo (Kautz et al., 2000). É importante ressaltar dos três formatos identificados, como o papel do pesquisador muda, atuando mais como observador no primeiro, participativo no segundo (transformando-se quase em um "funcionário" da organização, se ainda não for) e um papel de facilitador no terceiro. Se classificássemos o grau de colaboração do pesquisador teríamos o seguinte resultado: técnico colaborativo < melhoria < colaboração mútua.

No entanto, identificar novos problemas ou, em outras palavras, detectar uma oportunidade genuína de pesquisa pode não ser tarefa fácil. Diante do constante fluxo de eventos e variáveis de contexto presentes em um ambiente real, estas oportunidades podem estar ocultas ou mesmo serem identificadas indevidamente. Como regra básica, o tema de pesquisa investigado não deve possuir solução trivial, ou seja, deve ser algo que não esteja disponibilizado na indústria como uma solução comum ao problema. Em ambos os estudos é possível caracterizar este fato. No primeiro, a documentação do conhecimento sobre padrões de codificação e estilos arquiteturais do projeto não era algo que poderia ser realizado automaticamente por uma ferramenta, já que o conhecimento era tácito; além disto, procedimentos sistemáticos para realização desta documentação não foram encontrados. No segundo estudo, apesar de inspeções de software já estarem disponíveis e terem sido estudadas há algum tempo, a questão da compreensibilidade de modelos de casos de uso ainda precisa ser mais bem investigada em cenários reais, já que novas propostas continuam surgindo.

A cultura organizacional também pode ser um fator impossibilitador da identificação de novas oportunidades de pesquisa ou motivo para resistência a mudanças. Normalmente, está relacionada a crenças, tradições e históricos que sustentam as perspectivas, ações e práticas dos indivíduos em uma

organização (Figura 19 – as setas com duplo sentido representam implicação recíproca) (Kates e Robertson, 2004). No estudo #2, por exemplo, muitos problemas apontados no modelo de casos de uso pelo *checklist* – introduzido no contexto da pesquisa – foram inicialmente rejeitados, pois, segundo os próprios autores, já vinha sendo há muito tempo especificado daquela forma e, por isso, não viam problema naquilo que havia sido apontado. Desta forma, mais do que dificultar a identificação de novas oportunidades de pesquisa, a própria cultura organizacional pode ser a origem de um problema enfrentado na organização na medida em que se torna base para afirmações incorretas.

Figura 19 - Contexto Organizacional em Engenharia de Software adaptado de McKay e Marshall (2007)

O pesquisador deve lidar com este contexto utilizando o conhecimento tácito e os relacionamentos em seu benefício para enfrentar os desafios e promover as (novas) ações na organização, sem esquecer o histórico da organização para tentar não criar atritos. O papel do pesquisador, neste sentido, é conduzir as mudanças tentando não causar grandes rompimentos ou da maneira mais natural possível. Caso consiga cumprir este papel, conseguirá manipular a visão de mundo dos participantes em torno dos objetivos propostos durante as intervenções e terá acesso a dados fiéis aos da prática normal do dia-a-dia da organização, contribuindo para uma maior relevância dos resultados tanto para organização quanto para a sua pesquisa. Para que tudo isto ocorra da forma "mais natural possível", o pesquisador deve atentar-se para as dimensões da prática de Engenharia de Software citadas

anteriormente. A fim de auxiliar o pesquisador nesta questão, o seguinte mapeamento entre as dimensões e a pesquisa-ação é sugerido na Tabela 26.

Tabela 26 - Mapeamento entre pesquisa-ação e as dimensões da prática de Engenharia de Software[13]

Atividades da Pesquisa-Ação	*Termos-chave da prática de Engenharia de Software*
Diagnóstico	Conhecimento de perícia profissional e pessoal
Planejamento	Prática baseada em evidência, Raciocínio de engenharia, Conhecimento proposicional e pessoal
Tomada da Ação	Raciocínio de engenharia, Conhecimento de perícia profissional, pessoal e proposicional
Avaliação	Conhecimento proposicional
Aprendizagem	Conhecimento de perícia profissional, pessoal e proposicional

A Figura 20 ilustra como este processo de pesquisa se encaixa no contexto organizacional. Conforme já discutido anteriormente, após a coleção dos dados, o pesquisador os estrutura para análise e, em seguida, realiza a análise utilizando-se de alguma técnica que permita gerar algum conhecimento que deverá ser revertido em aprendizado para própria organização e em algum conhecimento científico, possivelmente, no formato de uma teoria. Em relação à construção de teorias, a Figura 20 mostra a origem principal dos constructos, proposições e explanações que compõem uma teoria (definido pelas setas tracejadas). Durante a manipulação e estruturação dos dados, o pesquisador pode perceber os principais conceitos e objetos utilizados e afetados durante a pesquisa, os quais se tornarão os candidatos imediatos à constructos. Na medida em que grande parte dos dados coletados durante a pesquisa-ação são dados qualitativos, as técnicas de análise de dados não permitem o estabelecimento de relações de causa-efeito precisas (numéricas), mas ainda assim permitem, ao menos, a definição das proposições entre os constructos. As explanações da teoria têm origem nas observações do pesquisador sobre as intervenções, interpretadas pela sua orientação teórica e valores.

[13] Os termos tomada de decisão de engenharia e julgamento em engenharia foram omitidos, pois estão embutidos na definição de raciocínio de engenharia.

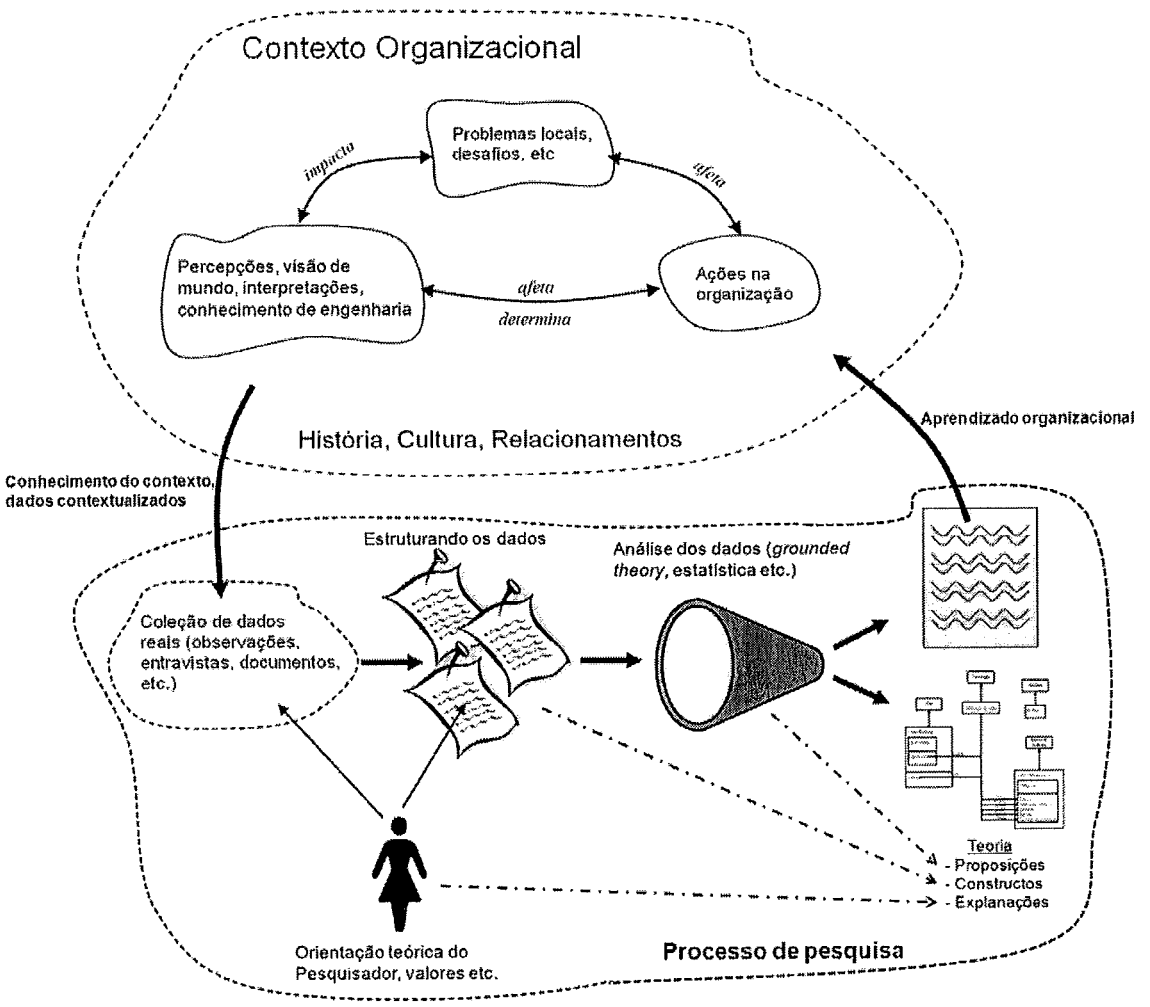


Figura 20 – O processo da pesquisa-ação na geração de teorias evoluído de McKay e Marshall (2007)

Devido a forma como são construídas na pesquisa-ação, as teorias tendem a ser de nível 1 (que apresentam relações causais mínimas que são concretas e baseadas diretamente na observação) e nível 2 (representadas pelas teorias de alcance médio as quais envolvem alguma abstração mas estão ainda próximas das observações de campo). A teoria construída no estudo #1 está classificada no nível 1. A construção de teorias em Engenharia de Software é uma busca recente dos pesquisadores que vêm tentando fornecer algum tipo de guia para este objetivo (e.g., Sjøberg *et al.*, 2008). Diante da discussão nesta seção, a pesquisa-ação parece uma metodologia adequada para a geração e avaliação de teorias em Engenharia de Software, mas atenção especial deve ser dada ao rigor da pesquisa para que resultados relevantes sejam produzidos.

5.3.2.O Rigor da Pesquisa e a Relevância dos Resultados

A relevância dos resultados da pesquisa-ação para aplicação prática é evidente. No entanto, nenhum resultado é de fato relevante se os meios para obtê-lo não forem legítimos. Para mostrar a legitimidade dos resultados é necessário um grau mínimo de rigor da pesquisa que mostre que os efeitos descritos nas explanações foram obtidos como efeito da solução proposta.

Recomendações e estratégias para conferir rigor à pesquisa-ação já foram amplamente discutidas na literatura (Avison *et al.*, 2001, Davison *et al.*, 2004). Inclusive, as chamadas estruturas de controle da pesquisa-ação de Avison *et al.* (2001) foram utilizadas no Capítulo 2 para avaliar o rigor dos estudos publicados em Engenharia de Software. Existem pelo menos dois aspectos fundamentais que caracterizam o rigor de um estudo. Um é o grau de controle exercido na condução do estudo com objetivo de minimizar o viés e a influência de outras variáveis sobre os resultados alcançados. Os trabalhos citados tratam esta questão. O segundo aspecto, particularmente mais evidente na pesquisa-ação, é a forma como o conhecimento teórico é mantido explícito durante as ações de pesquisa (Baskerville e Pries-Heje, 1999). Para tratar esta segunda questão, na falta de abordagens disponíveis na literatura, as etapas de planejamento e análise da pesquisa-ação foram trabalhadas por meio do uso da abordagem GQM (*Goal-Question-Metric*) (Basili *et al.*, 1994).

A abordagem GQM baseia-se na suposição de que para se medir de maneira eficaz, alguns objetivos devem ser estabelecidos para que estes sirvam de rota para o estabelecimento de questões que orientarão a definição de métricas em um contexto particular. É caracterizada por dois processos (Latum *et al.*, 1998): um refinamento de cima para baixo (*top-down*) dos objetivos em questões e, após, em métricas, e uma análise de baixo para cima (*bottom-up*) dos dados coletados.

A abordagem GQM foi utilizada em ambos os estudos conduzidos. A nomenclatura foi levemente modificada para melhor se encaixar no formato de relato de estudo definido e com as características da pesquisa-ação. Primeiramente, os objetivos da pesquisa são definidos no nível conceitual considerando o objeto estudado, o ponto de vista adotado e o contexto do estudo. Depois, as questões de pesquisa são definidas no nível operacional onde o objeto de estudo é caracterizado segundo alguma característica de qualidade selecionada para investigação. Por fim, as questões práticas –

expressão utilizada no lugar de métricas – são determinadas e buscam fixar quais os dados devem ser coletados em função das questões de pesquisa.

A partir da definição do GQM, foi possível perceber, nos estudos conduzidos, que a etapa de análise da pesquisa-ação torna-se mais focada, pois a interpretação dos dados já é pensada inicialmente. Além disto, é possível rastrear os resultados alcançados ao objetivo inicial definido para pesquisa e conferindo-lhe um maior rigor. Particularmente no estudo #1, o GQM foi útil no mapeamento das categorias da *grounded theory* e as questões de pesquisa, servindo de base para a teoria construída.

Como comentário final, recomendamos o uso da abordagem GQM em estudos de pesquisa-ação como forma de estruturação e análise da pesquisa.

5.3.3.A Engenharia de Software Baseada em Evidência com a Pesquisa-Ação

Como último tópico a ser discutido na aplicação da pesquisa-ação em Engenharia de Software, abordamos um tema que tem ganhado destaque recentemente em Engenharia de Software Experimental que é a Engenharia de Software Baseada em Evidência (ESBE). A idéia é tentar mostrar como a ESBE pode ser facilitada por meio da pesquisa-ação, especialmente para os profissionais da indústria que venham a tentar ter um papel mais atuante no contexto acadêmico, estreitando a ligação entre academia e indústria.

O conceito da ESBE foi adaptado da Medicina onde é comum que o conhecimento científico produzido a partir de métodos experimentais oriente o desenvolvimento de novas pesquisas e apóie importantes decisões na indústria. O papel dos profissionais (médicos) na Medicina é fundamental, pois eles atuam com base no conhecimento do estado da arte, em grande parte disponibilizado por meio de Revisões Sistemáticas, e contribuem retornando as suas experiências à comunidade científica (Kreder, 1999). Neste sentido, a idéia de se trazer a prática baseada em evidência para Engenharia de Software teve forte influência de um cenário onde grande parte da evidência utilizada pelos profissionais baseia-se exclusivamente em sua própria experiência ou opinião de especialistas (Dybå *et al.*, 2005).

O objetivo da ESBE é "prover os meios que permitam que a melhor evidência atual de pesquisa possa ser integrada com experiência prática e valores humanos num processo de tomada de decisão relativo ao

desenvolvimento e manutenção de software" (Kitchenham et al., 2004). A ESBE consiste da execução de 5 passos:

1. Converter uma necessidade de informação em uma pergunta respondível;
2. Identificar a melhor evidência que responda à questão;
3. Avaliar a evidência de maneira crítica considerando sua validade, impacto e aplicabilidade;
4. Integrar esta avaliação ao conhecimento do domínio e aos valores dos interessados;
5. Avaliar a efetividade e eficiência na execução dos passos 1-4 e buscar maneiras de melhorá-los.

É possível traçar um relacionamento entre o processo da pesquisa-ação com as etapas da ESBE. A Tabela 27, uma evolução da Tabela 26, mapeia as atividades da ESBE com atividades do processo da pesquisa-ação e termos chaves da prática de Engenharia de Software associados às responsabilidades do profissional indústria. A idéia da tabela não é ser precisa em relação a este mapeamento, mas ter a percepção da essência de cada etapa.

Tabela 27 - Mapeamento entre ESBE, pesquisa-ação e termos-chave

Etapas da ESBE	*Atividades da Pesquisa-Ação*	*Termos-chave da prática de Engenharia de Software*
1. Fazer uma questão respondível	Diagnóstico	Conhecimento de perícia profissional e pessoal
2. Encontrar a melhor evidência	Planejamento	Prática baseada em evidência
3. Avaliar a evidência	Planejamento	Prática baseada em evidência, Raciocínio de engenharia, Conhecimento proposicional e pessoal
4. Aplicar a evidência	Tomada da Ação	Raciocínio de engenharia, Conhecimento de perícia profissional, pessoal e proposicional
5. Avaliar o desempenho	Avaliação, Aprendizagem	Conhecimento de perícia profissional, pessoal e proposicional

Por este mapeamento é possível perceber que a ESBE pode ser traduzida em atividades da pesquisa-ação, dando possibilidade de que a ESBE seja mais amplamente utilizada. Argumentamos também que a pesquisa-ação pode ser uma ferramenta útil como meio de aprendizado da ESBE por ter práticas mais próximas aos do profissional da indústria além de, pela própria experiência no aprendizado da metodologia, poder tornar o profissional mais reflexivo em relação as suas práticas.

5.4. Um *Template* para Relatar Estudos

Ao iniciar o trabalho com a pesquisa-ação um vasto material sobre o tema pôde ser encontrado – ainda que com as características "genéricas" mencionadas anteriormente. No entanto, um ponto que não é tratado em nenhum deles, mas que representou uma dificuldade inicial para aplicação da metodologia, é a forma de relatar os estudos.

Neste sentido, esta seção descreve um *template* elaborado no contexto deste trabalho para reportar estudos em pesquisa-ação (Tabela 28). Este *template* foi utilizado nos estudos conduzidos e, como pode ser visto pelas suas seções, sua elaboração foi guiada pelo próprio processo da pesquisa-ação, possuindo uma seção para cada etapa.

Tabela 28 - Seções do *Template*

1)		**Diagnóstico**
	a)	Descrição do Problema
	b)	Contexto do Projeto/Trabalho
	c)	Tema de Pesquisa
2)		**Planejamento**
	a)	Revisão da Literatura
		i)Estudo Inicial (Opcional)
	b)	Enfoque da Ação
		i)Objetivos
		ii)Questões de Pesquisa
		iii)Resultados Esperados
	c)	Hipóteses
	d)	Definições Operacionais
		i)Técnicas
		ii)Ferramentas
		iii)Instrumentos
		iv)Arranjo do Estudo (Opcional)
3)		**Ações**
4)		**Avaliação e Análise**
5)		**Reflexões e Aprendizado**
	a)	Aprendizado
	b)	Reflexões

Diagnóstico

A descrição da etapa de diagnóstico foi dividida em três seções: descrição do problema, contexto do projeto e o tema de pesquisa. A descrição do problema deve descrever a situação a ser enfrentada de maneira a destacar a sua importância. Em seguida, esta situação deve ser inserida no contexto no qual onde ela esta ocorrendo completando a descrição do diagnóstico. O tema de pesquisa trás apenas uma síntese do problema a ser tratado fazendo um vínculo com o resto das seções.

Planejamento

A seção de planejamento é uma das mais detalhadas e serve de apoio para execução da pesquisa. Diante do diagnóstico realizado, o primeiro passo é descrever a revisão da literatura para dar base ao resto do planejamento. Nesta seção, já deve haver alguma indicação dos aspectos importantes de cada trabalho que serão aproveitados para o restante da pesquisa. Se os resultados destes trabalhos ainda forem muito incipientes, existe a possibilidade da execução de um estudo inicial que também deve ser descrito na seção de planejamento. Em seguida, a seção chamada "enfoque da ação" define os objetivos da pesquisa por meio da abordagem GQM conforme descrito anteriormente. Em cima desta definição, algumas hipóteses são determinadas, mostrando algumas expectativas em relação aos resultados que serão alcançados. Por fim, devem ser consideradas algumas definições operacionais como ferramentas a serem utilizadas, técnicas de análise e qualquer outro tipo de recurso necessário a realização da pesquisa. Opcionalmente, como também foi feito no estudo #2, pode-se programar algum tipo de arranjo dos participantes durante as atividades.

Ações

Esta seção é uma das mais simples. A idéia é descrever as atividades desempenhadas ao longo da pesquisa. A regra é quanto mais detalhes melhor, especialmente detalhes como datas e duração das atividades.

Avaliação e Análise

O objetivo desta seção é, basicamente, descrever o processo de análise de dados. Um ponto importante durante a análise dos dados é tentar manter sempre um vínculo com as questões de pesquisa estabelecidas de modo a garantir um maior rigor à análise na medida em que torna os resultados "rastreáveis" em relação aos problemas enfrentados.

Reflexões e Aprendizado

A última seção tem um duplo objetivo, assim como os da pesquisa-ação. Primeiramente, deve-se examinar os resultados alcançados frente ao estado da arte disponível na literatura técnica. Durante esta análise, é importante dar ênfase aos trabalhos utilizados na seção "revisão da literatura" para comparar os resultados alcançados, caso seja possível. Em uma segunda parte, a descrição do aprendizado deve mencionar não apenas o material "físico" gerado no contexto da e para a organização, mas também tentar retratar o

aprendizado vivido pelos participantes e como possivelmente influenciou a cultura organizacional.

5.5. Conclusão

A análise da experiência na condução dos estudos descritos nos capítulos anteriores e a discussão sobre como a pesquisa-ação deve ser explorada em Engenharia de Software feitas neste capítulo nos oferecem uma base que nos permite ter alguma indicação sobre a aplicabilidade desta metodologia em Engenharia de Software.

A pesquisa-ação tem características que se encaixam com naturalidade à prática de Engenharia de Software unindo e evoluindo o conhecimento científico com o conhecimento tácito da prática. Desta forma, aparece como uma opção viável para contribuir para um cenário de pesquisa em Engenharia de Software com uma maior quantidade de resultados e com maior relevância por garantir acesso direto ao *know-how* que muitas vezes pesquisas de opinião e estudos controlados, por exemplo, não permitem alcançar. Como posto por Polanyi (1966), "nós sabemos mais do que podemos dizer".

Capítulo 6
Conclusão

Este capítulo descreve as principais contribuições, limitações e alguns possíveis trabalhos futuros.

6.1. Considerações Finais

Nesta dissertação foi discutida a necessidade de estudos experimentais relevantes em Engenharia de Software. Por ser uma atividade com forte componente social, estudos controlados normalmente são executados em pequena escala e ambientes não reais, pois representam um alto grau de risco em projetos reais. Como conseqüência, ainda que produzam resultados importantes para a pesquisa em Engenharia de Software, seus resultados são limitados e muitas vezes com pouca relevância para aplicação direta na indústria.

Existem diferentes alternativas para execução de estudos em ambientes reais e que tratam o componente social da prática em Engenharia de Software. Dentre as diferentes alternativas, avaliou-se a metodologia da pesquisa-ação com um grande potencial para aplicação em Engenharia de Software devido as suas características. Além disto, uma pesquisa nos principais periódicos e conferências da área mostrou que o interesse na pesquisa-ação vem crescendo, mas ainda longe de ser amplamente utilizada. Foi possível identificar também, nesta pesquisa, a aplicação inadequada da metodologia em alguns casos. De maneira geral, mais do que um aumento da utilização da pesquisa-ação em Engenharia de Software, observa-se uma maior receptividade e apreço pela pesquisa qualitativa como uma alternativa apropriada para pesquisa em Engenharia de Software, principalmente enquanto ainda não somos capazes de construir modelos sofisticados como outras engenharias.

Diante deste cenário, buscou-se, então, analisar a aplicabilidade da pesquisa-ação em Engenharia de Software ao mesmo tempo em que se tentou aprimorar o conhecimento do uso da metodologia especificamente em Engenharia de Software. Para isto, foram conduzidos dois estudos de pesquisa-ação onde foi dada atenção especial a maneira como a metodologia

pode ser utilizada em Engenharia de Software. A partir desta experiência, diferentes aspectos particulares à utilização da metodologia em Engenharia de Software puderam ser abordados, e representam um conjunto de indicações importante para guiar futuros estudos de pesquisa-ação na área. Podemos citar como exemplo destas indicações particulares ao contexto de Engenharia de Software:

- A avaliação do aprendizado da pesquisa-ação por pesquisadores em Engenharia de Software que não tenham tido contato prévio com a metodologia.
- A importância do conhecimento tácito, intrínseco a qualquer atividade dependente da perícia humana como a Engenharia de Software, na condução de estudos *in vivo*, especialmente em estudos de pesquisa-ação.
- Questões éticas em estudos de pesquisa-ação em Engenharia de Software.
- Os elementos principais que definem o reconhecimento de uma situação real de um projeto de software que pode ser traduzida em um tema de pesquisa.
- O cuidado que deve ser dado ao rigor nos estudos de pesquisa-ação, onde se utilizou uma abordagem comumente utilizada em experimentação em Engenharia de Software – o GQM –, chamando a atenção para o fato de que apesar da pesquisa-ação parecer ser "demasiadamente flexível" o rigor da pesquisa deve ser mantido.

É importante mencionar, por fim, que a pesquisa-ação distingue-se de forma evidente de abordagens e práticas relacionadas à consultoria e gestão de conhecimento, por exemplo. Só para consultoria, podemos citar um conjunto de cinco aspectos chaves:

1. Motivação. A pesquisa-ação é motivada por interesses científicos. Consultoria é normalmente motivada por benefícios comerciais, incluindo lucros e conhecimento proprietários sobre soluções para problemas da organização.

2. Comprometimento. A pesquisa-ação se compromete com a comunidade científica para a produção de conhecimento científico, assim como para o cliente. Em uma situação de consultoria, o comprometimento é para com o cliente apenas.

3. Abordagem. Como visto a colaboração é peça chave em estudos de pesquisa-ação. Consultoria tipicamente valoriza o ponto de vista imparcial e "de fora", provendo uma perspectiva objetiva aos problemas organizacionais.

4. Embasamento para recomendações. A pesquisa-ação utiliza predominantemente um arcabouço teórico-científico para a solução dos problemas organizacionais. Consultores, por outro lado, normalmente sugerem soluções que, na sua experiência, provaram ser bem sucedidas em situações similares.

5. Essência do entendimento organizacional. Na pesquisa-ação, o entendimento organizacional é fundamentado em práticas bem sucedidas de mudanças experimentais na organização. Consultores, por sua vez, tendem a desenvolver um entendimento baseado na sua análise crítica da situação.

Neste sentido, deve-se reforçar a singularidade da pesquisa-ação enquanto metodologia científica que, pela análise do capítulo anterior, mostra-se apropriada para condução de estudos *in vivo* em Engenharia de Software podendo contribuir para a geração de resultados relevantes para a pesquisa na área.

6.2.Contribuições

Nesta seção, algumas contribuições identificadas neste trabalho de pesquisa são descritas.

Revisão da literatura. O Capítulo 2 descreve conceitos relacionados à pesquisa em Engenharia de Software apontando diferentes tipos de metodologias e abordagens, e a metodologia da pesquisa-ação em detalhes. Além disto, a pesquisa envolvendo os artigos de nove periódicos e três conferências mostrou o estado atual da utilização da pesquisa-ação em Engenharia de Software e seu resultado foi publicado em Santos e Travassos (2009).

Execução de estudos de pesquisa-ação. Dois estudos foram executados do início ao fim em ambientes reais e seus resultados podem ser utilizados para contribuir para a área nos tópicos de refatoração, documentação e inspeção de software. Além disto, estes estudos representam exemplos concretos do uso da pesquisa-ação e de técnicas como a *grounded theory* e a construção de teorias. Destacam-se ainda os seguintes pontos:

- A proposta de uma teoria para explicar os benefícios da refatoração de código na explicitação do conhecimento tácito relacionado a padrões de codificação.
- A adaptação e avaliação de uma tecnologia proposta pela comunidade científica – um *checklist* para inspeção de modelos de casos de uso –, tendo como um resultado importante a indicação do efeito da experiência dos inspetores na utilização da abordagem *checklist* para inspeções de software.

Utilização da pesquisa-ação em Engenharia de Software. Os estudos executados possibilitaram a avaliação do uso da metodologia sob diferentes aspectos:

- Questões relacionadas à prática da metodologia, como aprendizado e dificuldades, controle da pesquisa e aspectos éticos.
- Descrição do processo da pesquisa-ação no contexto da Engenharia de Software, mostrando como a construção de teorias pode servir como ferramenta de análise e considerando fatores como o rigor e a relevância da pesquisa.
- Discussão de uma possível integração entre a Engenharia de Software Baseada em Evidência e a pesquisa-ação, como meio de ampliar a prática baseada em evidência. A idéia inicial desta integração foi publicada em Santos e Travassos (2008).

Definição de um *template* para relato de estudos de pesquisa-ação. O *template* foi elaborado no contexto dos estudos, o que indica a sua utilidade prática em estudos de pesquisa-ação em Engenharia de Software. Por meio do uso do *template*, espera-se que a utilização da pesquisa-ação em Engenharia de Software seja facilitada.

6.3.Limitações

A proposta deste trabalho foi descrever a forma como a pesquisa-ação pode ser aplicada em Engenharia de Software a partir da experiência prévia na condução de dois estudos-exemplo. Neste sentido, sua principal limitação é justamente o fato de que a experiência na aplicação da metodologia pode não proporcionar o contato com todos os fatores importantes para a sua utilização. No entanto, tentamos minimizar esta questão baseando nossa discussão em trabalhos que tiveram o mesmo propósito que este, mas em áreas distintas.

Isto acrescenta uma base mais sólida à discussão na medida em que os tópicos abordados também foram em outros trabalhos, ao mesmo tempo em que nos permite afirmar com uma maior segurança que a abordagem para tratar o tema – por meio da experiência na utilização da metodologia – foi adequada a proposta deste trabalho.

6.4.Trabalhos Futuros

Entre os trabalhos futuros sugeridos no intuito de dar continuidade a esta pesquisa podemos citar:

- A expansão da revisão da literatura por meio de uma revisão sistemática. Neste trabalho, a revisão da literatura foi limitada aos principais periódicos e conferências. Assim, uma revisão mais ampla da literatura pode trazer resultados interessantes.
- Utilização da pesquisa-ação em outros contextos, caracterizando melhor seu uso em situações particulares.
- Executar um estudo que permita avaliar a pesquisa-ação frente a outras metodologias, como estudos controlados. A idéia seria ter algo como um meta-estudo, onde pesquisadores utilizariam diferentes abordagens para estudar um único tema. Aspectos a serem avaliados incluem o aprendizado, esforço e nível de impacto ao ambiente estudado (que risco o estudo representa). Uma avaliação como esta solidificaria os resultados apresentados neste trabalho ou, caso os contrarie, permitiria delimitar melhor o escopo de aplicação de cada uma.

Referências

Abrahamsson, P., Koskela, J. (2004) "Extreme Programming: A Survey of Empirical Data from a Controlled Case Study". In: Proc. of International Symposium on Empirical Software Engineering, pp. 73-82.

Albanese, M. A., Mitchell, S. (1993) "Problem-based learning: A review of literature on its outcomes and implementation issues". Academic Medicine, 68, 52–81.

Anda, B. and Sjøberg, D.I.K. (2002). "Towards an Inspection Technique for Use Case Models". In: Fourteenth International Conference on Software Engineering and Knowledge Engineering (SEKE), Ischia, Italy, July 15-19.

Argyris, C., Putnam, R., Smith, D.M. (1985) "Action Science". In: Jossey-Bass Social and Behavioral Science Series, 1st edition.

Avison, D.E., Baskerville, R., Myers, M. (2001) "Controlling action research projects".In: Information Technology and People, 14, 28–45.

Avison, D. E., Lau, F., Myers, M. D., and Nielsen, P. A. (1999). "Action research". Communications of the ACM, 42(1):94–97.

Ballantyne, D. (2004) "Action research reviewed: a market-oriented approach". In: European Journal of Marketing, Vol. 38 No.3/4, pp.321-37.

Basili, V. (1996) "The role of experimentation: Past, current, and future". In Proceedings of the 18th International Conference on Software Engineering, pages 442–450.

Basili, V.R., Caldiera, G., Rombach, H.D. (1994) "The Goal Question Metric approach". In the Encyclopedia of Software Engineering, vol. 2, pp. 528-532, John Wiley & Sons, Inc.

Basili, V., Elbaum, S. (2006) "Empirically Driven SE Research: State of the Art and Required Maturity". Invited Talk, ICSE 2006, Shanghai.

Basili, V., Green, S., Laitenberger, O., Shull, F., Sorumgaard, S., Zelkowitz, M. (1996) "The empirical investigation of perspective based reading". In: Empir Softw Eng-Int J. 1:133–164.

Basili, V.R., Selby, R.W., Hutchens, D.H. (1986) "Experimentation in Software Engineering," IEEE Trans. Software Eng., pp. 733-743, July.

Baskerville, R. L. (1999) "Investigating information systems with action research". In: Communications of the Association for Information Systems, volume 2.

Baskerville, R. (2007) "Educing Theory from Practice". In N. Kock (Ed.) Information Systems Action Research: An Applied View of Emerging Concepts and Methods. Springer, New York.

Baskerville, R., Pries-Heje, J. (1999). "Grounded action research: A method for understanding IT in practice". In: Accounting, Management and Information Technologies, 9:1-23.

Baskerville, R., Wood-Harper, A.T. (1996) "A critical perspective on action research as a method for information systems research". In: Journal of Information Technology, 11, pp. 235–246.

Baskerville, R., Wood-Harper, A. T. (1998) "Diversity in Information Systems Action Research Methods". In: European Journal of Information Systems, (7) 2, pp. 90-107.

Bengtsson, P., Bosch, J. (1999) "Haemo Dialysis Software Architecture Design Experiences". Proc. 21st ICSE, pp. 516-525.

Bennet, K. H. and Rajlich, V. T. (2000) "Software maintenance and evolution: A roadmap", In: The Future of Software Engineering, Ed. ACM Press, 73–87.

Biolchini, J., Mian, P. G., Natali, A. C. C., Travassos, G. H. (2005) "Systematic Review in Software Engineering", COPPE/UFRJ, PESC, Technical Report ES 679/05, 31p.

Briand, L. C., Freimut, B., Vollei, F. (2000) "Assessing the Cost-Effectiveness of Inspections byCombining Project Data and Expert Opinion". In: Proceedings of the 11th International Symposium on Software Reliability Engineering, 124-135.

Boehm, B.W. (1976), "Software Engineering," IEEE Trans. Computers, pp. 1.226 -1.241.

Bridges, E. M. (1992) "Problem-Based Learning for Administrators" ERIC Clearinghouse on Educational Management, Eugene, OR.

Burns, A. (2005) "Action research: An evolving paradigm?" Language Teaching 38(2), pp. 57–74.

Canfora, G., Garcia, F., Piattini, M., Ruiz, F., Visaggio, C.A. (2006) "Applying a framework for the improvement of the software process maturity in a software company". Journal Software Practice and Experience 36 (3) 283–304.

Carr, W. (2006) "Philosophy, Methodology and Action Research". Journal of Philosophy of Education, Special Issue: Philosophy, Methodology and Educational Research Part 1, 40.2, pp. 421–437.

Carr, W., Kemmis, S. (1986) "Becoming Critical: Education, Knowledge and Action Research". Basingstoke: Falmer Press.

Charters, S. M., Budgen, D., Turner, M., Kitchenham, B., Brereton, P., Linkman, S. G. (2009) "Objectivity in Research: Challenges from the Evidence-Based Paradigm". Australian Software Engineering Conference, pp. 73-80.

Checkland, P., Holwell, S. (1998). "Action Research: Its Nature and Validity". In: Systemic Practice and Action Research 11(1): 9-21.

Cheng, B., Jeffery, R. (1996) "Comparing Inspection Strategies for Software Requirement Specifications". In: Proceedings Australian Software Engineering Conference. IEEE Comput. Soc, Los Alamitos, CA, USA.

Cox, K., Aurum, A., Jeffery, R. (2004) "An experiment in inspecting the quality of use case descriptions". In: Journal of Research and Practice in Information Technology 36(4):211–229.

Crupi, J., Alur, D., Malks, D. (2001). "Core J2EE Patterns: Best Practices and Design Strategies", Prentice Hall PTR.

Davis, A, Overmyer, S, Jordan, K. (1993). "Identifying and Measuring Quality in a Software Requirements Specification". Proceedings of the First International Software Metrics Symposium, Baltimore.

Davison, R. M., Martinsons, M. G., Kock, N. (2004). "Principles of canonical action research". In: Information Systems Journal, 14(1), pp. 65–86.

Deboni, J. E. Z., Gregolin, R. (2008). "Inspeção de Qualidade em Descrições de Casos de Uso: Uma Proposta de Modelo e Artefatos". In: VII SBQS, Florianópolis – SC, Brasil.

Dias Neto, A. C., Sinola, R. O., Bott, A., Travassos, G. H. (2007). "Estratégia de Teste de Software no Desenvolvimento Incremental de um Sistema de Informação". In: Workshop on Systematic and Automated Software Testing, 2007, João Pessoa – PB, Brasil.

Dick, B. (2004). "Action research literature: Themes and trends". In: Action Research, v. 2, pp. 425-444.

Dunsmore, A., Roper, M., and Wood, M. (2003). "Practical code inspection techniques for object-oriented systems: an experimental comparison". In: IEEE Software, 21-29.

Dybå, T., Kitchenham, B.A., Jørgensen, M. (2005). "Evidence-Based Software Engineering for Practitioners". In: IEEE Software, v. 22, pp. 58-65.

Easterbrook, S., Singer, J., Storey, M-A., Damian, D. (2008) "Selecting empirical methods for software engineering research". In: Advanced Topics in Empirical Software Engineering, F. Shull, J. Singer, and D.I.K. Sjøberg, eds. Springer-Verlag.

Fabbrini, F., Fusani, M., Gnesi, S., Lami, G. (2001). "An Automatic Quality Evaluation for Natural Language Requirements". 7th International Workshop on Requirements Engineering (REFSQ'01).

Fagan, M. (2001) "A History of Software Inspections", Sd&m Conference, Disponível em: http://www.mfagan.com/resources.html. Acessado em 06 de janeiro de 2009.

Falbo, R. A. (1998) "Integração de Conhecimento em um Ambiente de Desenvolvimento de Software", Tese de Doutorado, Engenharia de Sistemas e Computação, COPPE/UFRJ.

Fernández-Medina, E. and Piattini M. (2005) "Designing secure databases". In: Information & Software Technology 47(7), pp. 463-477.

Fitzgerald, B., O'Kane, T. (1999) "A Longitudinal Study of Software Process Improvement". IEEE Software, 16, 3, pp. 37-45.

Fowler, M., Beck, K., Brant, J., Opdyke, W. and D. Roberts. (1999), "Refactoring: Improving the Design of Existing Code", Addison Wesley, 1st edition.

Gregolin, R. (2007). "Uma proposta de inspeção em modelos de caso de uso". Dissertação de Mestrado em Engenharia da Computação, Instituto de Pesquisas Tecnológicas do Estado de São Paulo, São Paulo, 108 f.

Greenwood, D. J., Levin, M. (1998) "Introduction to Action Research: Social Research for Social Change". Thousand Oaks, CA: Sage Publications.

Guba, E. G., Lincoln, Y. S. (1994) "Competing paradigms in qualitative research". In: N. K. Denzin & Y. S. Lincoln (Eds.), Handbook of qualitative research (pp. 105-117). Thousand Oaks, CA: Sage.

Harrison, R., Badoo, N., Barry, E., Biffl, S., Parra, A.,Winter, B., Wuest, J. (1999) "Directions and Methodologies for Empirical Software Engineering Research". In: Empirical Software Engineering, 4(4), 405–410.

Hathaway, R. S. (1995) "Assumptions underlying quantitative and qualitative research: implications for institutional research". In: Research in Higher Education 36(5):535–562.

Healy, M., Perry, C. (2000) "Comprehensive criteria to judge validity and reliability of qualitative research within the realism paradigm". Qualitative Market Research: An International Journal, Vol. 3 No.3, pp.118-26.

Hearn, G., Foth, M. (2005) "Action Research in the Design of New Media and ICT Systems". In K. Kwansah-Aidoo (Ed.), Topical Issues in Communications and Media Research, New York, NY: Nova Science.

Heatwole, C. G., Keller, L. F., Wamsley, G. L. (1976) "Action Research and Public Policy Analysis: Sharpening the Political Perspectives of Public Policy Research". In: Political Research Quarterly, Vol. 29, pp. 597 – 609.

Higgs, J., Burns, A., Jones, M. (2001). "Integrating clinical reasoning and evidence-based practice". AACN Clinical Issues 12, pp. 482–490.

Höfer, A., Tichy, W.F. (2007) "Status of Empirical Research in Software Engineering". In: Basili et al. (eds), Experimental Software Engineering Issues: Assessment and Future Directions, Springer-Verlag, LNCS 4336.

Holter, I.M., Schwartz-Barcott, D. (1993) "Action Research: What is it? How has it been used and how can it be used in nursing?". In: Journal of Advanced Nursing, pp. 298-304.

IEEE Std 830-1998. (1998). "IEEE Recommended Practice for Software Requirements Specifications". Software Engineering Standards Committee of the IEEE Computer Society.

IEEE Keyword Taxonomy. (2002) http://www2.computer.org/portal/web/publications/acmsoftware.

JMP (2007) "JMP 7" Disponível em: http://www.jmp.com/software/jmp7/. Acessado em: 13 de setembro de 2009.

Juristo, N., Moreno, A. M. (2001) "Basics of Software Engineering Experimentation". Kluwer Academic Publisher, USA.

Kalinowski, M. (2004). "Infra-estrutura Computacional de Apoio ao Processo de Inspeção de Software". Dissertação de Mestrado em Engenharia de Sistemas e Computação, Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 120 f.

Kalinowski, M., Travassos, G.H. (2004) "A Computational Framework for Supporting Software Inspections", 19th IEEE International Conference on Automated Software Engineering (ASE 2004), Linz, Austria.

Kalinowski, M., Spínola, R.O., Dias Neto, A.C., Bott, A., Travassos, G. H. (2007) "Inspeções de Requisitos de Software em Desenvolvimento Incremental: Uma Experiência Prática", In: VI SBQS, Porto de Galinhas – PE, Brasil

Kampenes, B., Dybå, T., Hannay, J.E., Sjøberg, D.I.K. (2009) "A systematic review of quasi-experiments in software engineering", Inform. Software Technol. 51 (1), pp. 71-82.

Kates, S., Robertson, J. (2004) "Adapting action research to marketing". European Journal of Marketing, Vol. 38 No. 3/4, pp.418-32.

Kauppinen, M., Vartiainen, M., Kontio, J., Kujala, S., Sulonen, R. (2004) "Implementing requirements engineering processes throughout organizations: success factors and challenges". In: Information and Software Technology, vol. 46, pp. 937-953.

Kautz, K, Hansen, H.W., Thaysen, K. (2000) "Applying and Adjusting a Software Process Improvement Model in Practice: The Use of the IDEAL Model in a Small Software Enterprise". Proc. 22nd Int'l Conf. Software Eng., IEEE CS Press, pp. 626–633.

Kazman, R., Bass, L. (2002). "Making Architecture Reviews Work in the Real World," IEEE Software, vol. 19, No. 1, pp. 62-73.

Kitchenham, B.A. (2004) "Procedures for Performing Systematic Reviews," Technical Report TR/SE-0401, Keele University, NICTA.

Kitchenham, B. A., Dybå, T., and Jørgensen, M. (2004). "Evidence-Based Software Engineering". In: ICSE 2004, 273-281, IEEE Computer Society Press.

Kitchenham, B. (2007) "Empirical Paradigm – The Role of Experiments" In: Basili et al. (eds), Experimental Software Engineering Issues: Assessment and Future Directions, Springer-Verlag, LNCS 4336.

Kitchenham, B., Budgen, D., Brereton, P., Turner, M., Charters, S., Linkman, S., (2007) "Large-scale software engineering questions - expert opinion or empirical evidence?". In: IET Software, vol.1, no.5, pp.161-171.

Krauss, S.E. (2005) "Research Paradigms and Meaning Making: A Primer". In The Qualitative Report, 10 (4), 758-770.

Kreder, H.J. (1999) "Evidence-based surgical practice: what is it and do we need it?" In: World J. Surg., 23:1232–1235.

Kuhn, T. S. (1970) "The Structure of Scientific Revolutions". Second edition. Chicago: University of Chicago Press.

Lanubile, F., Visaggio, G. (1996) "Assessing defect detection methods for software requirements inspections through external replication". In: ISERN-96-01, January.

Latum, F., Solingen, R., Hoisl, B., Oivo, M., Rombach, H.D., Ruhe, G. (1998) "Adopting GQM-based measurement in an industrial environment". In: IEEE Software, pp. 78-86.

Lau, F. (1999) "Toward a framework for action research in information systems studies". Information, Technology & People, vol. 12, pp. 148-175.

Lau, F. (1997) "A Review On The Use of Action Research in Information Systems Studies". In: A. Lee, J. Liebenau, and J. DeGross, (eds.) Information Systems and Qualitative Research, Chapman & Hall, pp. 31-68.

Lindvall, M. and Sandahl, K. (1996) "Practical Implications of Traceability". Journal of SP&E, 26(10):1161–1180, October.

Ludema, J. D., Cooperrider, D. L., Barret, F. J. (2001) "Appreciative Inquiry: the Power of the Unconditional Positive Question". In P. Reason & H. Bradbury (Eds.), Handbook of action research: Participative inquiry and practice (pp. 1–14). Thousand Oaks, CA: Sage.

Lycett, M. (2001) "Understanding 'Variation' in component-based development: Case findings from practice". Information and Software Technology, 43(3), 203–213.

Maiden, N., Robertson, S. (2005) "Developing use cases and scenarios in the requirements process". Proc. ICSE 2005, 561-570.

Maldonado, J., Carver, J., Shull, F., Fabbri, S., Doria, E., Martimiano, L., Mendonça, M., and Basili, V. (2006) "Perspective-Based Reading: A Replicated Experiment Focused on Individual Reviewer Effectiveness". In: Empirical Software Engineering: An International Journal. Volume 11, Number 1.

Mäntylä, M.V. (2005) "An experiment on subjective evolvability evaluation of object-oriented software: explaining factors and interrater agreement", In: International Symposium on Empirical Software Engineering, pp. 277-286.

Mäntylä, M. V. and Lassenius, C. (2006) "Drivers for software refactoring decisions", In: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering (ISESE'06), pages 297-306, New York, NY, USA.

Mattsson, A., Lundell, B., Lings, B., Fitzgerald, B. (2009) "Linking Model-Driven Development and Software Architecture: A Case Study Software Engineering". IEEE Transactions on 35, 83–93.

McKay, J., Marshall, P. (2007) "Driven By Two Masters, Serving Both". In N. Kock (Ed.) Information Systems Action Research: An Applied View of Emerging Concepts and Methods. Springer, New York.

Mens, T. and Tourwe, T. (2004) "A survey of software refactoring", In: IEEE Transactions on Software Engineering, 30(2):126–139.

Meyer, J. E. (1993) "New paradigm research in practice: The trials and tribulations of action research". In: Journal of Advanced Nursing, 18(7), 1066-1072.

Miller, J., Wood, M., Roper, M. and Brooks, A. (1998) "Further Experiences with Scenarios and Checklists". Empirical Software Engineering, Vol. 3(1), pp. 37-64, January.

Näslund, D. (2002) "Logistics needs qualitative research – especially action research". In: International Journal of Physical Distribution & Logistics Management, Vol. 32 No. 5, pp. 321-328.

Ottosson, S. (2003) "Participation action research: a key to improved knowledge of management". In: The International Journal of Technological Innovation, Entrepreneurship and Technology Management (Technovation), Vol. 23, No. 2, pp. 87-94.

Parnin, C. and Görg, C. (2008) "A catalogue of lightweight visualizations to support code smell inspection", In: Proceedings of the ACM Symposium on Software Visualization, pp. 77-86.

Pfleeger, S. H. (1999) "Albert Einstein and empirical software engineering". IEEE Computer, 32(10), 32–37.

Polo, M., Piattini, M., Ruiz, F., (2002) "Using a Qualitative Research Method for Building a Software Maintenance Methodology". In: Software Practice & Experience, 32(13), pp. 1239-1260.

Popper, K. R. (1959) "The logic of scientific discovery". London: Hutchinson.

Porter A. and Votta L. (1998) "Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment Using Professional Subjects". In: Empirical Software Eng. J., vol. 3, no. 4, pp. 355-379.

Rainer, A., Jagielska, D. and Hall, T. (2005) "Software Practice versus evidence-based software engineering research". In: Proceedings of the Workshop on Realising Evidence-based Software Engineering, ICSE-2005.

Reason, P., Bradbury, H. (2001) "Introduction: Inquiry and participation in search of a world worthy of human aspiration". In P. Reason & H. Bradbury (Eds.), Handbook of action research: Participative inquiry and practice (pp. 1–14). Thousand Oaks, CA: Sage.

Runeson, P. and Höst, M. (2009) "Guidelines for conducting and reporting case study research in software engineering", Empirical Software Engineering 14 (2), pp. 131–164.

Rus, I., Lindvall, M. (2002) "Knowledge Management in Software Engineering" IEEE Software (19:3), pp. 26-38.

Ruthruff, J. R., Penix, J., Morgenthaler, J. D., Elbaum, S., Rothermel, G. (2008) "Predicting accurate and actionable static analysis warnings: an experimental approach", In: Proc. of the 30th International Conference on Software Engineering, pp. 341–350.

Salo, O., Abrahamsson, P. (2005) "Integrating Agile Software Development and Software Process Improvement: a Longitdinal Case Study". ISESE, Autralia, Noosa Heads.

Santos, P. S. M. e Travassos, G. H. (2008) "Colaboração entre Academia e Indústria: Oportunidades para Utilização da Pesquisa-Ação em Engenharia de Software". In: 5th Experimental Software Engineering Latin American Workshop, v. 1, p. 1-10.

Santos, P.S.M., Travassos, G. H. (2009) "Action Research Use in Software Engineering: an Initial Survey" 3nd International Symposium on Empirical Software Engineering and Measurement, Orlando, USA, to appear.

Schein, E.H. (1994) "The process of dialogue: creating effective communication". In: The Systems Thinker, Vol. 5 No. 5, pp. 1-4.

Seaman, B.C. (1999) "Qualitative Methods in Empirical Studies of Software Engineering". In: IEEE Transactions on Software Engineering, vol. 25(4), pp. 557-572.

Shaw, M. (1990) "Prospects for an Engineering Discipline of Software". IEEE Software, 7(6): 15-24.

Shull, F. (1998). "Developing Techniques for Using Software Documents: A Series of Empirical Studies". PhD Thesis, Department of Computer Science, University of Maryland, USA.

Shull, F., Carver, J., Travassos, G.H., 2001, "An Empirical Methodology for Introducing Software Processes". In: 8th European Software Engineering Symposium and 9th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-9) and 8th European Software Engineering Conference (ESEC), Vienna, Austria, September.

Shull, F., Rus, I., and Basili, V. (2000). "How Perspective-Based Reading Can Improve Requirements Inspections". IEEE Software , July 73-79.

Sjøberg, D. I. K., Dybå, T., Anda, B.C.D. and Hannay, J. E. (2008) "Building Theories in Software Engineering". Advanced Topics in Empirical Software Engineering, F. Shull, J. Singer, and D.I.K. Sjøberg, eds. Springer-Verlag.

Sjøberg, D.I.K., Dybå, T., Jørgensen, M. (2007) "The future of empirical methods in software engineering research". In FOSE '07: Future of Software Engineering, pages 358-378, Washington, DC, USA.

Sjøberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanović, A., Liborg, N.-K., Rekdal, A.C. (2005) "A Survey of Controlled Experiments

in Software Engineering". IEEE Transactions on Software Engineering, 31(9): 733–753.

SOFTEX (2007) "MPS.BR: Melhoria de Processo do Software Brasileiro", Guia Geral Versão 1.2, Campinas, SP, SOFTEX.

Sokal, A., and Bricmont, J. (1999) "Fashionable nonsense: Postmodern intellectuals' abuse of science". New York: Picador.

Staron, M., Meding, W. (2007) "Predicting weekly defect inflow in large software projects based on project planning and test status". Information and Software Technology.

Staron, M., Meding, W., Nilsson, C. (2009) "A framework for developing measurement systems and its industrial evaluation". In: Information and Software Technology, 51(4):721–737, April.

Strauss, A., Corbin, J. (1990). Basics of qualitative research: Grounded theory procedures and techniques. Newbury Park, CA: Sage.

Sun (1999). "Code Conventions for the Java Programming Language", Sun Microsystems. Disponível em: http://java.sun.com/docs/codeconv/. Acessado em: 28 de outubro de 2008.

Susman, G.L., Evered, R.D. (1978) "An assessment of the scientific merits of action research". In: Administrative Sciences Quarterly, 23, pp. 582–603.

Vigder, M., Vinson, N. G., Singer, J., Stewart, D., Mews, K. (2008) "Supporting Scientists' Everyday Work: Automating Scientific Workflows". IEEE Software, 25, 52–58.

Travassos, G.H., Barros, M.O. (2003) "Contributions of In Virtuo and In Silico Experiments for the Future of Empirical Studies in Software Engineering". In: Proc. of the WSESE03, Fraunhofer IRB Verlag, Roma.

Tichy, W. F. (1998) "Should Computer Scientists Experiment More?" IEEE Computer, pp. 32-40, May.

Thiollent, M. (2007) "Metodologia da Pesquisa-Ação", Cortês Editora, 15° edição.

Trac (2003) "Trac: Integrated Software Configuration and Project Management" Disponível em: http://trac.edgewall.org/. Acessado em: 13 de setembro de 2009.

Tripp, D. (2005). "Action Research: A methodological introduction," Educacao e pesquisa, (31:3), pp. 443-466.

Westbrooke, R. (1995), "Action research: a new paradigm for research in production and operations management", International Journal of Operations and Production Management, Vol. 15, No.12, pp 6-20.

Wohlin, C., Aurum, A., Petersson, H., Shull, F., Ciolkowski, M. (2002) "Software inspection benchmarking - a qualitative and quantitative comparative opportunity". In: Proceedings of 8th International Software Metrics Symposium, 118-130.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000) "Experimentation in Software Engineering: An Introduction". Kluwer Academic Publishers.

Wohlin, C., Höst, M., Henningsson, K. (2003) "Empirical Research Methods in Software Engineering". In Lecture Notes in Computer Science: Empirical Methods and Studies in Software Engineering: Experiences from ESERNET, edited by A. I. Wang and R. Conradi, Springer Verlag.

Zelkowitz, M.V. (2007) "Techniques for Empirical validation" In: Basili et al. (eds), Experimental Software Engineering Issues: Assessment and Future Directions, Springer-Verlag, LNCS 4336.

Anexo A – Transcrição das Entrevistas

A.1. Utilização da *Grounded Theory*

Nesta seção, é apresentado o procedimento de análise qualitativa *Grounded Theory* em cima da transcrição das entrevistas semi-estruturadas relativas ao capítulo 3. O processo inicia-se com a codificação aberta, apresentado ao lado das transcrições, prossegue com o agrupamento e sintetização dos principais conceitos presentes nos códigos e termina com a definição das categorias a partir dos grupos. Após, o relacionamento entre as categorias é estudada por meio da codificação axial.

A.1.1. Transcrição das Entrevistas

Algumas questões, i, ii e v, focaram apenas em como certas atividades do estudo deveriam ser conduzidas e, por isto, não serão analisadas por meio da Grounded Theory.

A.1.1.1. Questão iii

Questão: Qual o efeito da refatoração sobre a qualidade do código? (relacionada à questão prática Q.1.1)

A.1.1.1.1. Desenvolvedor 1 (equipe L)

Os maiores efeitos sobre o código foram sobre a sua estrutura visual (identação, posição de métodos e variáveis, por exemplo) e nomenclatura. Com isso, o maior benefício na percepção do desenvolvedor é para a fase de manutenção, já que a refatoração uniformizou um único padrão de codificação e, por isso, deve facilitar a sua leitura no futuro. Poucos defeitos foram encontrados na refatoração, mas não foi o foco adotado (nem o sugerido) para inspeção.	Efeito sobre a representação visual do código Benefício à manutenção do código Efeito sobre a legibilidade do código

A.1.1.1.2. Desenvolvedor 2 (equipe L)

[Entrevistador perguntou: Olhando o código e com ferramentas de métricas de código nós já teremos uma idéia sobre a qualidade do código, mas o que eu queria nessa questão é a sua percepção em relação ao efeito da refatoração sobre o código. Desenvolvedor 2 responde:] Falando muito por "alto", pois o que é a qualidade de código você deve estar sabendo mais do que eu, já que você vem conduzindo a pesquisa. *[Entrevistador enfatiza: A idéia é uma descrição em alto nível mesmo, uma percepção sua em relação à quais aspectos a refatoração poderá melhorar a qualidade do código. Desenvolvedor 2 continua:]* A refatoração contribuiu para melhorar a representação visual do código no que diz respeito à aspectos "cosméticos". Ou	Efeito sobre a representação visual do

seja, a maneira ou a forma como se codifica - o que está ligado à questão de quão fácil de entender é o código (legibilidade, compreensibilidade). Um dos aspectos que melhora é nesse sentido, os aspectos "visuais". Alguns exemplos, identação ou uma linha de código muito grande.	código Efeito sobre a legibilidade do código
[Entrevistador pergunta: Esse seria o maior efeito que vem a sua cabeça, então? Desenvolvedor 2 responde:] Lembro-me de ser numeroso sim, mas não são os mais importantes. O maior efeito seria a melhora do algoritmo, por exemplo. Aí, estaríamos pensando na estrutura "da coisa". Como a solução é pensada e expressa no código. Eu acho que vai além da sintaxe do código... Teve aquele clássico, sobre o qual X (um dos projetistas da equipe) até comentou, em se que estava buscando no banco de dados item a item e depois adicionando a uma lista, mas sabemos que está longe de ser a melhor maneira de se extrair um bom desempenho da solução.	Efeito sobre a estrutura do código e otimização do algoritmo
[Entrevistador pergunta: Então, quando não se segue o que seria uma "boa prática" na sua percepção, qual seria o efeito sobre o código?] Na legibilidade, considerando o que foi falado no início em relação à identação, comentários ... a questão de como se codifica "visualmente". Teria também o aspecto do desempenho o qual está mais relacionado à estrutura, ou seja, a construção da solução.	Os principais efeitos são sobre a compreensibilidade e estrutura do código.
[Entrevistador afirma: Você entende, então, que problemas na estrutura e desempenho, não necessariamente afetariam a compreensibilidade? Desenvolvedor 2 comenta:] Sim, aliás eu acho que se pode ter um código que tem uma boa legibilidade, mas que peca na forma como usa os recursos computacionais.	Estrutura algorítmica nem sempre esta associada a idéia de compreensibilidade
[Entrevistador pergunta: Mudando de assunto, e com relação a documentação e comentários do código? Você percebeu a sua ausência? Ou o código de maneira geral é simples e não exige documentação? Desenvolvedor 2 responde:] Eu quando fui inspecionar meio que eu já sabia mais ou menos como que a coisa funcionava. Então, no meu ponto de vista, o comentário para mim, em alguns pontos, não fez muita falta. Mas eu acredito que poderia fazer falta para qualquer outra pessoa que não estivesse ligada ao projeto, porque eu vi muitas partes onde não tinha comentário. Posso estar sendo, então, enviesado. Inclusive, nas partes que eu não tinha tido contato o comentário fez falta diretamente. De qualquer maneira, tem muitas partes em que o código é relativamente simples e não requer comentários, e tem outras partes em que eu senti falta de comentário e documentação, principalmente, nas classes de domínio.	A organização do código (arquitetura da solução), quando conhecida pelo desenvolvedor, é mais útil que outros tipos de documentação no entendimento do código. Classes de domínio necessitam de um maior cuidado com relação à documentação.
[Desenvolvedor 2, por iniciativa própria, tentam resumir tudo que havia dito nesta questão:] Assim, eu acho que o código na medida em que se aplicam as modificações (refatorações), melhora na legibilidade, ajusta a estrutura para melhorar o desempenho e/ou legibilidade. De maneira geral, acho que também facilita a manutenção do código, ou seja, facilita a vida de quem está fazendo a manutenção ou ainda de quem vai depurar o código. Considerando que quem precisa depurar o código precisa lê-lo.	Benefício à manutenção do código

A.1.1.1.3. Desenvolvedor 3 (equipe R)

[Entrevistador perguntou: Qual foi a sua percepção em relação ao resultado da refatoração sobre a qualidade do código? Você teve a percepção, de maneira geral, de que melhorou ou piorou? Em quais características de qualidade? Desenvolvedor respondeu:]	
Eu acho que melhorou e destacaria os aspectos da compreensão e legibilidade do código. Ainda, a refatoração vai impactar para o próprio desenvolvimento no futuro quando a gente já vai poder aplicar as idéias das necessidades de refatoração identificadas.	Efeito sobre a legibilidade do código Aprendizado
[Entrevistador perguntou: Quando você menciona "o código ficou mais claro e compreensível" a que você se refere exatamente? Por exemplo, ao uso mais adequado das classes de negócio, à API do JSF, à estrutura do código ou em termos de nomes de classe ou método. Desenvolvedor respondeu:]	
Eu destacaria mais a estrutura do código e utilização das classes de negócio. Foi possível perceber durante a inspeção como o código deveria ser organizado em termos de quais classes (ou camadas) deveriam responder a determinadas responsabilidades.	Efeito sobre a estrutura do código Classes de domínio necessitam de um maior cuidado com relação à documentação.

A.1.1.1.4. Agrupando Códigos

Grupo	Código (# de ocorrências entre parênteses)
Os principais efeitos da refatoração sobre o código foram na legibilidade do código, destacando a sua representação visual, e na organização estrutural do código, considerando os algoritmos empregados, o que poderá beneficiar à manutenção do código.	Efeito sobre a representação visual do código (2/3) Efeito sobre a legibilidade do código (3/3) Efeito sobre a estrutura do código e otimização do algoritmo (2/3) Estrutura algorítmica nem sempre esta associada à idéia de compreensibilidade (1/3) Benefício à manutenção do código (2/3)
O principal conhecimento necessário ao entendimento do código é o da arquitetura da solução. Ainda assim, no contexto da arquitetura MVC, a camada *Model* deve ser documentada.	A organização do código (arquitetura da solução), quando conhecida pelo desenvolvedor, é mais útil que outros tipos de documentação no entendimento do código. (1/3) Classes de domínio necessitam de um maior cuidado com relação à documentação. (2/3)
-	Aprendizado (1/3)

A.1.1.2. Questão iv

Questão: Quais tipos de refatoração são identificados apenas pelos desenvolvedores experientes? (relacionada à questão prática Q.1.3)

A.1.1.2.1. Desenvolvedor 1 (equipe L)

Depende do tipo de defeito. Tiveram defeitos que o próprio IDE (ambiente de desenvolvimento integrado) detectou, mas o desenvolvedor que escreveu o código não se atentou para os problemas. Outros relacionados ao uso de recursos computacionais – métodos de validação de formulário eram

executados após a inicialização de algumas variáveis, sendo que caso o formulário não fosse válido as variáveis não seriam utilizadas. Mesmo um desenvolvedor inexperiente poderia detectar estes defeitos.

No entanto, grande parte dos defeitos apenas desenvolvedores experientes no projeto e em desenvolvimento poderiam detectá-los:

- Emprego de algoritmos simples com o uso adequado de tomadas de decisão ('ifs') e iteração ('for' e 'while').
- Regras de nomenclatura usadas atualmente no projeto e que não estão explícitas.
- Uso inadequado da API (Interface de Programação de Aplicativos), segundo os padrões adotados no projeto, da tecnologia de componentes de interface Web JavaServer Faces. Ou seja, como estruturar a solução de projeto utilizando a tecnologia.
- O uso apropriado do estilo arquitetural MVC (model–view–controller) considerando as respectivas responsabilidades de cada camada no contexto do projeto.

Apenas desenvolvedores mais experientes conhecem os padrões de codificação utilizados no contexto do projeto
Algoritmos mal elaborados

Regras de Nomenclatura

Padrões tácitos estão mais presentes na forma da estruturação da solução
- Estrutura do código

Padrões tácitos estão mais presentes na forma da estruturação da solução
- Arquitetura

A.1.1.2.2. Desenvolvedor 2 (equipe L)

[Entrevistador perguntou: Partindo da premissa que o pessoal mais experiente no projeto, não necessariamente em desenvolvimento mas terminou sendo também, identificaria certos tipos de problemas no código que nós queríamos ajustar. No caso, problemas no uso da tecnologia, problemas no código e de estrutura, não uso das "boas práticas" do projeto, dentre outros. Problemas esses que a gente tinha percebido olhando o código da recém integrada equipe de Juiz de Fora. Você conseguiria dizer, primeiro, se esta premissa foi verdadeira? E se sim, quais seriam os tipos de defeitos que são melhor identificados pelos mais experientes? Desenvolvedor 2 respondeu:]
Em relação a primeira pergunta eu acho que sim. O que me leva a crer que seja importante ter os mais experientes atuando nesse tipo de atividade é que eles têm mais condições de detectar os problemas (necessidades de refatoração) e defeitos relacionados à algoritmos mal elaborados, à estrutura do código ou à coisas que estão ligadas a construção e lógica (aqui se referindo aos padrões de codificação e estilo arquitetural) do código. Existem construções de código que funcionam (estão corretas) ou que possuem desempenho adequado, mas que você (um desenvolvedor mais experiente) vê que não é o correto, pois não respeitam o que seria uma "boa prática" no contexto do projeto.

Algoritmos mal elaborados
Estrutura do código

Apenas desenvolvedores mais experientes conhecem os padrões de codificação utilizados no contexto do projeto

[Entrevistador perguntou: Com base nisso que você falou, você teve a perceção que já trouxe consigo alguns padrões que você já assumia sendo adotados no projeto e quando você se deparou com o código construído pela equipe de Juiz de Fora percebeu que esse código que não estava de acordo com esses padrões? Desenvolvedor 2 respondeu:]
Aconteceram duas coisas. Uma foi que, como você falou, eu fui realizando a inspeção me deparando com problemas que achavam que deveriam estar corretas segundo a minha percepção de como o projeto vem sendo construído até então.
[Entrevistador interrompeu: Em outras palavras, você

percebe, de maneira geral, que existem padrões de construção no projeto, padrões de codificação e estilo arquitetural, que nós adotamos tacitamente? Desenvolvedor 2 comenta:]

Exatamente. São padrões tácitos que não estão documentados, mas que eu trouxe à inspeção preconcebidos. Mas aconteceu também uma outra coisa (voltando a pergunta anterior). Aconteceu que durante a inspeção aspectos que transcendem a questão dos padrões (padrões de codificação e estilo arquitetural) utilizados no nosso projeto, como boas práticas de programação OO, que eu não vim com isso na cabeça previamente, mas durante a inspeção estes aspectos também vieram à tona. Inclusive, tiveram momentos em que eu pude observar que o código estava ruim (necessitaria de uma refatorção), mas não feria nenhum padrão dentre os que eu tinha preconcebido antes da inspeção já que nós não tínhamos nenhum padrão para aquilo. Vi que, nesse caso, seria uma boa oportunidade de se propor uma melhoria e de se colocar também como um novo padrão nosso.

Existem padrões de codificação que são tácitos e conhecidos pelos desenvolvedores

Novos padrões de codificação puderam ser depreendidos

[Entrevistador comenta: Parece, então, que cada um depositou a sua experiência (padrões de codificação e estilo arquitetural) no projeto e que após esse conhecimento ter se sedimentado, isto ficou acordado entre a equipe (antes da integração da equipe de Juiz de Fora). Mas ainda assim, existem alguns padrões de sua experiência de fora do projeto que ainda não estão presentes no projeto - como você mesmo falou, existiram situações novas em que você observou a necessidade da refatoração mesmo quando um padrão "tácito" do projeto não estava sendo negligenciado. Então, acredito que será interessante como aprendizado e documentação dos padrões de codificação e estilo arquitetural do projeto, que tenhamos a oportunidade tanto de documentar o que seria o conhecimento "tácito" já sedimentado no projeto quanto sugerir novos padrões a serem seguidos. Desenvolvedor 2 argumenta:]

E um fato que é tranquilo de notar é que da para perceber nitidamente que o desenvolvedor X (um desenvolvedor de Juiz de Fora) tinha um jeito de codificar e o Y (outro desenvolvedor de Juiz de Fora) tinha outro. Então é como se cada um desses novos desenvolvedores estivesse utilizando o "seu padrão", segundo a sua experiência prévia.

Heterogeneidade no estilo de codificação entre desenvolvedores

[Entrevistador interrompeu: Você acredita, então, que inicialmente, já que tinham poucas pessoas, a equipe conseguiu formalizar um padrão próprio com base na comunicação acessível da equipe e na leitura constante do código entre os membros da equipe, e quando os novos membros foram integrados cada um deles terminou utilizando o seu próprio "padrão" em vez do "tácito" existente na até então? Desenvolvedor 2 responde:]

Eu acho que sim, apesar de ser bastante difícil de se garantir que todo o código fique homogêneo, pois em algum momento a individualidade dos desenvolvedores fica registrada. Mas se pegarmos só os fundamentos do que seria o nosso padrão tácito, eu percebi que esse consegue se permear entre o código dos diferentes desenvolvedores (que compunham a equipe antes da de Juiz de fora ser integrada). Pelo menos nos alicerces da coisa, na maneira como a solução é implementada (padrões de codificação) e distribuída pelo código (estilo arquitetural). Ainda assim, existem diferenças, principalmente, em um nível de mais detalhe, como, por exemplo, identação do código ou agrupamento de código.

Padrões tácitos estão mais presentes na forma da estruturação da solução

- Estrutura do código
- Arquitetura (estilo arquitetural)

Padrões tácitos não atingem a representação visual do código

[Entrevistador voltou a perguntar a segunda parte da questão anterior: Considerando estes dois níveis que você comentou agora, você vê que a equipe de Juiz de Fora teve diferenças nestes dois níveis? (o nível em que o conhecimento tácito conseguiu se permear - que seria na implementação da solução considerando os padrões de codificação e estilo arquitetural - e o nível em que a individualidade manifesta-se mais intensamente - que seria a forma de "escrita" do código considerando a sua identação e seu agrupamento, por exemplo. Desenvolvedor 2 completa:]
Exatamente. O nível que reflete mais a forma de escrever o código é bastante evidente (foi comentado anteriormente). Já no nível de como implementar a solução, considerando os nossos padrões tácitos, aí fica também evidente (respondeu isto revisando a sua planilha de discrepâncias).

A.1.1.2.3. Desenvolvedor 3 (equipe R)

Esta questão não era direcionada a equipe R, apenas aqueles que identificaram necessidades de refatoração (no caso, equipe L).

A.1.1.2.4. Agrupando Códigos

Grupo	Código (# de ocorrências entre parênteses)
Existem padrões de codificação sobre os quais os desenvolvedores experientes têm consciência, ainda assim novos padrões puderam ser conhecidos.	Existem padrões de codificação que são tácitos e conhecidos pelos desenvolvedores (1/2) + 1 (questão iv) --- Novos padrões de codificação puderam ser depreendidos (1/2)
Desenvolvedores mais experientes têm condições de identificar padrões relacionados a regras de nomenclatura, algoritmos mal elaborados e estrutura do código.	Apenas desenvolvedores mais experientes conhecem os padrões de codificação utilizados no contexto do projeto (2/2) --- Regras de Nomenclatura (1/2) --- Algoritmos mal elaborados (2/2) --- Estrutura do código (1/2)
Os padrões tácitos de codificação estão mais inveterados no nível da arquitetura da solução e estruturação do código do que no nível de representação visual do código, onde há uma grande diversidade de estilos.	Padrões tácitos estão mais presentes na forma da estruturação da solução (2/2) • Estrutura do código (2/2) • Arquitetura (2/2) --- Padrões tácitos não atingem a representação visual do código (1/2) --- Heterogeneidade no estilo de codificação entre desenvolvedores (1/2)

A.1.1.3. Questão vi

Questão: De que maneira a refatoração permitiu o aprendizado dos padrões de codificação e estilo arquitetural utilizados no projeto?

A.1.1.3.1. Desenvolvedor 1 (equipe L)

Os problemas encontrados refletem a necessidade de se adequar a padrões de codificação e estilos arquiteturais tácitos no projeto. No entanto, tem-se a percepção de que não foram todos os padrões que foram encontrados. Estes deverão ser mencionados durante a criação da documentação dos padrões e estilos utilizados no projeto.

Evidenciação do raciocínio por trás dos padrões de codificação

A.1.1.3.2. Desenvolvedor 2 (equipe L)

[Entrevistador perguntou: Você teve a percepção de houve algum tipo de aprendizado com a execução da inspeção, em relação aos padrões (padrão de codificação e estilo arquitetural) no projeto? Em outras palavras, detectando os problemas que os desenvolvedores menos experientes de Juiz de Fora tiveram em relação a que você concebia como padrão, foi possível detectar essas diferenças e, a partir daí, visualizar o que você entendia ser o padrão? Desenvolvedor 2 respondeu:]

Sim, eu acredito que sim. A inspeção permitiu que eu pudesse observar e contrastar o que eles (equipe de Juiz de Fora) implementaram e o que a gente tem como padrão de desenvolvimento. E isso se deve ao fato de que é muito oportuno, durante uma inspeção, poder ver na prática com um exemplo de um código fora do padrão e ver nesse código diretamente a razão pela qual faz sentido o nosso padrão. Antes da inspeção eu já trouxe o meu conhecimento prévio sobre o projeto para tentar encontrar lugares no código onde uma refatoração seria necessária segundo aos padrões preconcebidos que eu tinha em mente. Então, a questão do aprendizado esteve muito relacionada à oportunidade que eu tive em poder ver um código que não satisfaz o padrão e, no ato de descrever a necessidade de refatorção (defeito no contexto da inspeção), entender e aprender com aquele defeito.

Contraste entre o "certo" e o "errado"

Evidenciação do raciocínio por trás dos padrões de codificação

Descrição do não atendimento aos padrões

A.1.1.3.3. Desenvolvedor 3 (equipe R)

[Entrevistador perguntou: De que maneira você acha que a refatoração permitiu o aprendizado em relação aos padrões de codificação e estilo arquitetural utilizados no projeto? Desenvolvedor respondeu:]

Eu acho que ficaram claros para mim os padrões que vocês utilizam aí (referindo-se a equipe do Rio de Janeiro). Ainda mais nestes casos de uso, quando nós tínhamos acabado de começar no projeto, então nós de fato não tínhamos tido tempo de capturar esses padrões. E eu percebi que os padrões, apesar de serem tácitos, não são bem definidos e a inspeção serviu para aprender casos em que eles poderiam ser utilizados melhor.

Existem padrões de codificação que são tácitos e conhecidos pelos desenvolvedores

Evidenciação do raciocínio por trás dos padrões de codificação

[Entrevistador perguntou: Mas você teve a percepção de que realmente você desconhecia estes padrões por trás das necessidades de refatorção apontadas ou você já sabia que este padrão existia mas ele não foi respeitado por conta de uma "falha" do desenvolvedor? Desenvolvedor responde:]

Uma parte, de fato, eram padrões que nós terminamos não seguindo por conta do dia-a-dia corrido. Mas tiveram coisas que realmente eram padrões que nós não conhecíamos como,

Internalização dos padrões utilizados

por exemplo, o uso da classe de Funcionalidade para acomodar a verificação de acesso por um determinado ator especificado no modelo de caso de uso em vez do uso da classe Perfil.

[Entrevistador comentou: Então, você entende que a gente conseguiu, por meio da refatoração, capturar e formalizar alguns padrões que provavelmente não foram passados para vocês (equipe de Juiz de Fora) durante o treinamento inicial? Desenvolvedor respondeu:]

Sim, vai ser importante ter isso formalizado, porque assim nós podemos mostrar a futuros integrantes da equipe. Inclusive, me lembro de alguns casos, durante a refatoração, em que você já tinha me dito aquilo, mas que por vezes por conta da distância e das atribulações do dia-a-dia terminou passando.

A formalização dos padrões ajudará a não negligenciá-los durante o desenvolvimento

A.1.1.3.4. Agrupando Códigos

Grupo	Código (# de ocorrências entre parênteses)
A refatoração permitiu entender a razão da organização do código por meio das descrições das necessidades de refatoração as quais indicavam as diferenças em relação ao padrão praticado no projeto.	Evidenciação do raciocínio por trás dos padrões de codificação (3/3) --- Contraste entre o "certo" e o "errado" (1/3) --- Descrição do não atendimento aos padrões (1/3)
A refatoração permitiu a internalização dos padrões utilizados no projeto, evitando que se repitam no futuro.	A formalização dos padrões ajudará a não negligenciá-los durante o desenvolvimento (1/3) --- Internalização dos padrões utilizados (1/3)

A.1.1.4. Questão vii

Questão: Decidir como formalizar o conhecimento externalizado. (relacionada à questão prática Q.2.3)

A.1.1.4.1. Desenvolvedor 1 (equipe L)

A melhor maneira de se externalizar o conhecimento adquirido com a refatoração é por meio do uso de modelos onde exemplos de código fonte corretos seriam listados e discutidos. É importante identificar os erros mais importantes e listar apenas estes a fim de que a lista de exemplos mantenha-se em um limite razoável, para facilitar a leitura desta lista, e até mesmo para facilitar a consulta quando utilizada como referência, a sua categorização ajudaria a focar naquilo que se procura. Talvez utilizando a própria categorização feita em cima das sugestões de refatoração identificadas durante a refatoração.

Utilização de exemplos

Categorização dos problemas
- Categorização por impacto

A utilização de categorias deve facilitar a leitura do documento

Basear nas categorias criadas a partir da descrição das discrepâncias

A.1.1.4.2. Desenvolvedor 2 (equipe L)

[Entrevistador perguntou: Nós já fizemos a refatoração e já tivemos as entrevistas. Agora a etapa final seria tentarmos, a partir do nosso aprendizado, formalizar alguma documentação em relação aos nossos padrões tácitos de codificação. Você sugeriria alguma forma de tentar capturar

e documentar estes padrões? Desenvolvedor respondeu:]
Primeiramente, olhando para o formulário de discrepância, para explicitar este conhecimento valeria separar JSPX e código Java. Tentar identificar a natureza dos problemas que acometeram a JSPX e, assim, tentar caracterizar de alguma maneira o problema e verificar a solução que foi dada. Daí, tentar mapear se estes problemas se repetem e verificar se as soluções são iguais. Então, nesse momento, estabelecido este padrão (categoria ou cenário) de problema inferir um texto que resuma a solução de maneira suficiente para que qualquer um possa aplicar para qualquer cenário semelhante, já que as descrições das sugestões estão em termos de um método específico.

Estabelecimento de cenários de problemas, verificando se a solução é sempre a mesma

Descrição da solução do problema de maneira genérica

[Neste momento, o desenvolvedor olha as suas sugestões no formulário de discrepâncias buscando um exemplo representativo do que ele tinha acabado de relatar:]
"a variável de referência formBuscar tem escopo pertinente apenas ao método buscarCursos()" (sugestão #10 do Desenvolvedor 2) ... Aí é aquele negócio, esta sugestão é de padrão de programação Java, mas ainda assim deveria ser incluído como padrão "nosso". Tentando extrair um cenário desta sugestão seria "respeitar a declaração das variáveis dentro do escopo pertinente a elas, então se você vai utilizar apenas dentro de um método declare-as nele". Poderia até acompanha de um exemplo ilustrativo, quando for um pouco difícil de visualizar a descrição dada.

Padrões de programação também devem ser incluídos como regras

Utilização de exemplos

[Entrevistador tentar resumir o que foi falado até então: Acho que a idéia foi capturada, então basicamente seria uma recomendação ou uma regra genérica ... Desenvolvedor interrompe e continua o resumo:]
É, pega a descrição da sugestão, que é especifica, e tenta extrair dela uma regra dentro de um cenário genérico o suficiente que descreveria problemas iguais só que em funcionalidades diferentes. Utilizar um exemplo quando pertinente.

Descrição da solução do problema de maneira genérica

[Entrevistador pergunta: Então você considera estas informações suficientes para externalizar o nosso conhecimento? Desenvolvedor responde, olhando novamente a lista de sugestões:]
Eu estava aqui olhando para a última coluna do formulário e lembrei que nós tínhamos categorizado as sugestões. Tem aqui, por exemplo, "poor readability". Nós poderíamos fazer uma seção de regras que tenham a ver com, por exemplo, a legibilidade de código. É porque o nome das categorias do formulário são "negativas", por exemplo, identação errada ou organização interna ruim. Então, uma sugestão seria na hora de organizar as regras colocar as diferentes categorias, mas com o nome no sentido positivo, como por exemplo, regras que dizem respeito à modularidade do código.

Categorização dos problemas
- Categorização por tipo

Basear nas categorias criadas a partir da descrição das discrepâncias
- Retirar o aspecto negativo

Chamar a descrição do problema + solução de regra

[Desenvolvedor continua pensando em outras informações para compor o documento:]
Não sei se para cada regra valeria a pena dizer os artefatos no código que são passiveis de se observar aquela regras. Por exemplo, possivelmente ou fortemente ocorrem em classes do tipo "controllers".

Categorização dos problemas
- Categorização por arquivos afetados (controllers, forms etc)

[Entrevistador comenta: Então isto seria, de certa forma, um outro tipo de categorização, onde, por exemplo, poderíamos ter "regras de legibilidade para controllers". Desenvolvedor comenta:]
Isso seria bem interessante. Até para uma pessoa que esteja

iniciando, quando estivesse implementando um controller poderia fazer a seguinte pergunta: "quais são as opções para um controlador?" As duas categorizações seriam excelentes para ajudar na leitura do documento e entendimento das regras.

A utilização de categorias deve facilitar a leitura do documento

[Entrevistador pergunta: Você acha que existiria um número ou quantidade ideal para estas regras? Ou seja, o tamanho do documento. Desenvolvedor responde:]

Poderíamos fazer uma alusão ao documento de casos de uso. Nele quanto maior pior, dependendo, é claro, de sua complexidade. Então, acredito que no final das contas quanto menos regras melhor. Mas isto também depende do domínio ou o tipo de conteúdo do documento. Por exemplo, uma pessoa que esteja começando no projeto, mas já sabe Java. Nesse caso, um documento grande não causaria tanto quanto alguém que não tivesse esse conhecimento.

Quanto maior o documento pior para o entendimento.

[Entrevistador questiona: Mas, ainda assim, é possível que um documento grande contenha tanta informação que o desenvolvedor não consiga absorver, concorda? Nesse caso, talvez, uma priorização poderia ser interessante? Desenvolvedor responde:]

Eu acho, então, que seria interessante determinar o impacto da não conformidade com as regras. Por exemplo, um nível "catastrófico" – não implementar seguindo uma regra deste nível acarretará em retrabalho, etc. Claro, é importante dar destaque àquelas regras com maior impacto. Mas também é importante que não se deixe de capturar alguma regra por conta do fato de ter um impacto baixo. Porque tirar alguma regra far-nos-ia perder conhecimento e, por isso, não valeria a pena.

Categorização dos problemas
- Categorização por impacto

Independentemente do tamanho do documento todas as regras devem ser capturadas

A.1.1.4.3. Desenvolvedor 3 (equipe R)

[Entrevistador pergunta: Você conseguiria pensar em alguma forma de tentar formatar uma documentação que explicitasse aquilo que nós aprendemos nesta refatoração? Ou seja, algumas coisas que eram feitas de maneira incorreta e que vocês não sabiam que não estavam em conformidade com o padrão tácito de codificação. Desenvolvedor responde:]

Uma lista com a forma certa e o problema de não fazer desta forma (por exemplo, se você fizer desta forma isto vai estar fora do padrão ou isto vai gerar problema para fazer algo amanhã). Seria interessante colocar no documento itens que não estão explícitas em nenhum lugar (citando como exemplo a sugestão #6 do desenvolvedor 1). Existem outras que estão mais relacionadas com a nossa dificuldade em utilizar a tecnologia.

Descrição da solução do problema

Priorização dos problemas relacionados ao não entendimento do projeto (e não da tecnologia)

[O desenvolvedor volta a comentar sobre o que seria interessante ter na listagem:]

O porquê utilizar desta forma, o que impacta se não usar e porque que usa desta forma. Acho que bastaria falar: "olha a forma certa é essa e se você não fizer isto, pode acontecer aquilo ou seu código vai ficar ruim ...".

Categorização dos problemas
- Categorização por impacto

Argumentar sobre o porquê respeitar uma regra

[Entrevistador questiona: Em relação ao tamanho do documento você teria alguma preocupação em relação a isto? Desenvolvedor responde:]

Eu acho que não mais do que quatro páginas. Porque a gente conhece, as pessoas começam a ler e em um determinado momento fica um pouco estressante. Mas talvez, pensando bem, o documento poderia ser organizado com um índice para

Quanto maior o documento pior para o entendimento

que possa ser utilizado como consulta.

Utilização de índice para facilitar consulta => categorias são um tipo de indexação

A.1.1.4.4. Agrupando Códigos

Grupo	Código (# de ocorrências entre parênteses)
Os problemas identificados na refatoração devem ser estudados e combinados em regras genéricas que indiquem sua aplicação em diferentes contextos, incluam motivação para sua utilização e, quando necessário, apresentem exemplos.	Estabelecimento de cenários de problemas, verificando se a solução é sempre a mesma (1/3) - Descrição da solução do problema de maneira genérica (3/3) - Utilização de exemplos (2/3) - Padrões de programação também devem ser incluídos como regras (1/3) - Chamar a descrição do problema + solução de regra (1/3) - Argumentar sobre o porquê respeitar uma regra (1/3)
O documento deve capturar todas as regras depreendidas da refatoração ainda que o seu tamanho comprometa o seu entendimento.	Quanto maior o documento pior para o entendimento. (2/3) - Independentemente do tamanho do documento todas as regras devem ser capturadas (1/3)
Para facilitar a leitura e consulta ao documento, deve ser realizada a categorização das regras a qual pode ser por impacto, tipo ou arquivos afetados.	Categorização dos problemas (3/3) • Categorização por impacto (3/3) • Categorização por tipo (1/3) • Categorização por arquivos afetados (controllers, forms etc) (1/3) - A utilização de categorias deve facilitar a leitura do documento (2/3) - Basear nas categorias criadas a partir da descrição das discrepâncias (2/3) • Retirar o aspecto negativo (1/3) - Utilização de índice para facilitar consulta => categorias são um tipo de indexação
	Priorização dos problemas relacionados ao não entendimento do projeto (e não da tecnologia) (1/3)

A.1.2. Codificação Aberta Inicial (revisão da literatura)

Dado o tema de pesquisa, a busca por trabalhos publicados na literatura teve foco em técnicas de inspeção as quais pudessem guiar a leitura do código e revelar os desvios mencionados anteriormente. Um critério usado nesta revisão era que a técnica fosse de fácil aplicação, ou seja, que não necessitasse de treinamento dos desenvolvedores para aplicação, apenas a descrição dos seus passos deveria ser suficiente. Além disto, tentou-se pesquisar na literatura mecanismos que pudessem

Sem necessidade de treinamento

Avaliação qualitativa

mostrar as motivações por trás da identificação dos defeitos – ou seja, uma taxonomia de defeitos que fosse adequada à categorização dos problemas associados aos estilos arquiteturais e padrões de codificação. A pesquisa foi conduzida de forma *ad-hoc* e os trabalhos selecionados por conveniência.

Dois trabalhos foram selecionados nesta revisão (Dunsmore *et al.*, 2003) e (Mäntylä, 2005), os quais foram utilizados para compor o procedimento elaborado para a leitura do código fonte (formulário da Tabela 4). O trabalho de Dunsmore *et al.* (2003) descreve uma técnica de inspeção baseada em casos de uso a qual permite a leitura de código orientado a objetos a partir do ponto de vista do seu modelo dinâmico. Os passos da técnica envolvem (1) a seleção de um caso de uso, (2) derivação de cenários de uso a partir do caso de uso (ex., "salvar protocolo", "cancelar protocolo") e (3) a leitura dos métodos das classes responsáveis pela execução do cenário.

Para categorizar as sugestões de refatoração foi utilizado o trabalho de Mäntylä e Lassenius (2006) – que é um trabalho posterior e diretamente relacionado à Mäntylä (2005). Nele, os autores descrevem um estudo experimental onde o objetivo foi entender o porquê desenvolvedores entendem ser necessária a refatoração de código fonte. Trata-se de um estudo qualitativo onde foi requisitado aos participantes que registrassem o motivo para a necessidade de refatoração. Através da análise dos registros feitos pelos participantes foi elaborada uma taxonomia de defeitos. Para permitir a utilização da taxonomia dos desvios o seguinte passo foi incluído na técnica descrita anteriormente (Tabela 4): (4) para cada método, necessidades de refatoração devem ser registradas de acordo com o conhecimento prévio do desenvolvedor. Os autores sugerem ainda algumas métricas de código que poderiam ser utilizadas para avaliar algumas de suas propriedades antes e após a refatoração.

É importante observar, por fim, a existência de uma grande proximidade entre as atividades de um processo de inspeção e refatoração de código. Ambos possuem, de maneira geral, etapas de planejamento, detecção, retrabalho e avaliação. De fato, o processo de refatoração pode ser entendido como um tipo de inspeção. Assim, da mesma forma que a inspeção, o processo refatoração de código pode proporcionar um aprendizado aos envolvidos em relação às boas práticas de codificação no contexto onde a refatoração é aplicada.

- Descrição das necessidades de refatoração
- Categorização

Leitura do código segundo a perspectiva do modelo de casos de uso

Derivação de cenários

Avaliação qualitativa
- Descrição das necessidades de refatoração

Avaliação quantitativa
- Métricas de código

Processo de refatoração = inspeção

Aprendizado em relação aos padrões dos defeitos

A.1.2.1.1. Agrupando Códigos

Grupo	Código
A refatoração de código embute um processo de inspeção, sendo de fácil aplicação.	Processo de refatoração = inspeção --- Sem necessidade de treinamento
A avaliação qualitativa será utilizada para investigar as necessidades de refatoração e a quantitativa para avaliar o efeito sobre o código.	Avaliação qualitativa • Descrição das necessidades de refatoração • Categorização --- Avaliação quantitativa • Métricas de código

A técnica consiste, resumidamente, na derivação de cenários a partir do modelo de casos de uso, leitura do código segundo estes cenários e a descrição do motivo da necessidade de refatoração.	Leitura do código segundo a perspectiva do modelo de casos de uso --- Derivação de cenários --- Descrição das necessidades de refatoração
Inspeções permitem proporcionam aprendizado aos envolvidos na medida em que os capacita na identificação de padrões de defeitos.	Aprendizado em relação aos padrões dos defeitos --- Avaliação qualitativa • Descrição das necessidades de refatoração

A.1.3. De Conceitos e Grupos a Categorias

Categoria	Grupos (origem)
A refatoração de código é benéfica à sua legibilidade e organização estrutural, e melhora a manutenabilidade, pois o formata de uma só maneira.	Os principais efeitos da refatoração sobre o código foram na legibilidade do código, destacando a sua representação visual, e na organização estrutural do código, considerando os algoritmos empregados, o que poderá beneficiar à manutenção do código. (q.iii) --- O principal conhecimento necessário a leitura do código é o da arquitetura da solução. Ainda assim, no contexto da arquitetura MVC, a camada Model deve ser documentada. (q.iii) --- A refatoração permitiu a internalização dos padrões utilizados no projeto, evitando que se repitam no futuro. (q.vi)
Os padrões tácitos de codificação estão mais presentes na arquitetura da solução e estruturação do código, representando um conhecimento essencial ao entendimento do código.	O principal conhecimento necessário ao entendimento do código é o da arquitetura da solução. Ainda assim, no contexto da arquitetura MVC, a camada Model deve ser documentada. (q.iii) --- Os padrões tácitos de codificação estão mais inveterados no nível da arquitetura da solução e estruturação do código do que no nível de representação visual do código, onde há uma grande diversidade de estilos. (q.iv)
Os desenvolvedores experientes conhecem e têm ciência da existência dos padrões tácitos do projeto e, por isso, podem identificar quando são negligenciados.	Existem padrões de codificação sobre os quais os desenvolvedores experientes têm consciência, ainda assim novos padrões puderam ser conhecidos. (q.iv) --- Desenvolvedores mais experientes têm condições de identificar padrões relacionados a regras de nomenclatura, algoritmos mal elaborados e estrutura do código. (q.iv) --- Os padrões tácitos de codificação estão mais inveterados no nível da arquitetura da solução e estruturação do código do que no nível de representação visual do código, onde há uma grande diversidade de estilos. (q.iv)

O aprendizado girou em torno da evidenciação dos motivos (tácitos) por trás da organização do código e da internalização dos padrões de codificação utilizados no projeto.	A refatoração permitiu entender a razão da organização do código por meio das descrições das necessidades de refatoração as quais indicavam as diferenças em relação ao padrão praticado no projeto. (q.vi) - A refatoração permitiu a internalização dos padrões utilizados no projeto, evitando que se repitam no futuro. (q.vi)
O conhecimento sobre o estilo arquitetural e padrões de codificação utilizados no projeto pode ser explicitado na forma de diretivas que guiarão os desenvolvedores a seguir um modelo único de desenvolvimento.	Os problemas identificados na refatoração devem ser estudados e combinados em regras genéricas que indiquem sua aplicação em diferentes contextos, incluam motivação para sua utilização e, quando necessário, apresentem exemplos. (q.vii) - O documento deve capturar todas as regras depreendidas da refatoração ainda que o seu tamanho comprometa o seu entendimento. (q.vii)
Todo o aprendizado deve ser documentado, desde que categorizado de forma a facilitar a busca e leitura das diretivas.	O documento deve capturar todas as regras depreendidas da refatoração ainda que o seu tamanho comprometa o seu entendimento. (q.vii) - Para facilitar a leitura e consulta ao documento, deve ser realizada a categorização das regras a qual pode ser por impacto, tipo ou arquivos afetados. (q.vii)
O processo de refatoração de código é análogo ao de inspeção de código e de fácil aplicação quando realizado sob a perspectiva da dinâmica de execução do sistema segundo o modelo de casos de uso.	A refatoração de código embute um processo de inspeção, sendo de fácil aplicação. (rev. lit.) - A técnica consiste, resumidamente, na derivação de cenários a partir do modelo de casos de uso, leitura do código segundo estes cenários e a descrição do motivo da necessidade de refatoração. (rev. lit.)
O processo de refatoração tem como benefício o aprendizado em relação aos padrões de defeitos que podem ser documentados por meio da análise dos resultados.	A avaliação qualitativa será utilizada para investigar as necessidades de refatoração e a quantitativa para avaliar o efeito sobre o código. (rev. lit.) - Inspeções permitem proporcionam aprendizado aos envolvidos na medida em que os capacita na identificação de padrões de defeitos. (rev. lit.)

A.2. Transcrição das Entrevistas (questões não utilizadas na grounded theory)

A.2.1. Desenvolvedor 1 (equipe L)

Questão (i): O procedimento apresentado foi adequado e útil para as atividades propostas. No entanto, houve um foco maior nas classes Java do que nos arquivos de apresentação (.jspx), ainda que tivesse sido indicado durante o treinamento que ambos deveriam ser inspecionados. Além disto, para facilitar o entendimento e aplicação do procedimento, seria interessante um modelo com o que deveria ser considerado código correto.

Questão (ii): A lista de discrepâncias foi revista e tipos foram atribuídos aos defeitos segundo as categorias apontadas em Mäntylä e Lassenius (2006).

Questão (v): Para o correto preenchimento do formulário, o treinamento inicial deveria ter enfatizado a necessidade de descrever quais atributos de qualidade estavam presentes nos métodos que não precisaram de refatoração. O desenvolvedor apontou a importância de se mostrar previamente quais os tipos de defeitos deveriam ser procurados, no entanto foi argumentado pelo pesquisador que esse era exatamente o intuído da inspeção (buscar as necessidades de refatoração de código segundo a experiência do desenvolvedor no projeto). Fora estas questões, o inspetor julgou o formulário de discrepâncias apropriado para a tarefa e sugeriu que o campo "linha de código do defeito" fosse acrescido ao formulário.

A.2.2. Desenvolvedor 2 (equipe L)

Questão (i): *[Entrevistador perguntou: O procedimento sugerido, presente na planilha que continha o formulário de discrepâncias, e as perguntas que deveriam ser respondidas para cada discrpância encontrada, foram úteis para focar a inspeção nas oportunidades de refatoração de código? Desenvolvedor 2 responde:]*

Acho que, primeiro, foi útil sim e eu vejo que a razão que levou a ser bastante útil no que diz respeito a focar a inspeção foi a aplicação da idéia de se estabelecer um cenário (o estabelecimento de um cenário é um dos passos do procedimento de inspeção elaborado). Imagine, um caso de uso pode possuir várias funcionalidades, então se você procurar estabelecer um cenário que cerque uma funcionalidade apenas você não tem chance de divergir na sua inspeção ou de tentar avaliar outros métodos que não tem nada a ver. Então acho que isso facilita muito ... ainda, embora em um cenário eu possa avaliar o mesmo método que no outro cenário, por serem cenários diferentes talvez isso dê *insights* diferentes... em outras palavras, coisas que passaram desapercebidas em um determinado cenário em outro cenário o defeito passou a ser "perceptível".

[Entrevistador comentou: Esse procedimento eu retirei de um estudo da literatura, onde três métodos para realizar inspeção de código eram comparados de se realizar a inspeção. Eu achei este adequado, pois o desenvolvimento tem forte base nos modelos de caso de uso. Desenvolvedor 2 comenta:]

Acho que se deve a isso então, se fosse aplicar novamente seguir este "padrão" seria interessante.

Questão (ii): A lista de discrepâncias foi revista e tipos foram atribuídos aos defeitos segundo as categorias apontadas em [5].

Questão (v): *[Entrevistador perguntou: A idéia desta questão é verificar se o formulário atendeu as expectativas na hora do preenchimento, você imaginava ou mesmo sentiu falta de informação que deveria ser capturada? Ou o espaço livre, onde deveria se descrever o porquê a refatoração seria necessária, foi suficiente? Desenvolvedor 2 respondeu:]*

Sim, o formulário atendeu plenamente às atividades conduzidas.

A.2.3. Desenvolvedor 3 (equipe R)

Questão (i): questão apenas para equipe L.

Questão (ii): questão apenas para equipe L.

Questão (iv): questão apenas para equipe L.

Questão (v): *[Entrevistador perguntou: Com relação ao formulário utilizado, você conseguiu entender bem o que eles (desenvolvedores) descreveram como problemas? E o espaço (um campo no formulário para justificar o porquê a refatoração havia sido realizada ou não) que foi reservado a você (responsável pela refatoração) estava adequado? Desenvolvedor respondeu:]*

Eu achei adequado. E gostei muito da forma como as necessidades de refatoração foram descritas, pois foram muito precisas em relação à localização e ao problema presente no código.

Anexo B – Formulários e Questionários de Pesquisa (estudo #2)

B.1. Formulário de Caracterização

Nome _____Nível (Ms.c/D.Sc.): ____

Formação Geral

Qual é sua experiência anterior com desenvolvimento de software na prática ? (marque aqueles itens que melhor se aplicam)
__ nunca desenvolvi software.
__ tenho desenvolvido software para uso próprio.
__ tenho desenvolvido software como parte de uma equipe, relacionado a um curso.
__ tenho desenvolvido software como parte de uma equipe, na indústria.

Por favor, explique sua resposta. Inclua o número de semestres ou número de anos de experiência relevante em desenvolvimento (E.g. "Eu trabalhei por 10 anos como programador na indústria")

Experiência em Desenvolvimento de Software

Por favor, indique o grau de sua experiência nesta seção seguindo a escala de 5 pontos abaixo:
1 = nenhum
2 = estudei em aula ou em livro
3 = pratiquei em 1 projeto em sala de aula
4 = usei em 1 projeto na indústria
5 = usei em vários projetos na indústria

Experiência com Requisitos

- Experiência escrevendo requisitos 1 2 3 4 5
- Experiência escrevendo casos de uso 1 2 3 4 5
- Experiência revisando requisitos 1 2 3 4 5
- Experiência revisando casos de uso 1 2 3 4 5
- Experiência modificando requisitos para manutenção 1 2 3 4 5

Experiência em Projeto

- Experiência em projeto de sistemas 1 2 3 4 5
- Experiência em desenvolver projetos a partir de requisitos e casos de uso 1 2 3 4 5
- Experiência criando projetos Orientado a Objetos 1 2 3 4 5
- Experiência lendo projetos Orientado a Objetos 1 2 3 4 5
- Experiência com Unified Modeling Language (UML) 1 2 3 4 5

- Experiência alterando projeto para manutenção 1 2 3 4 5

Outras Experiências

- Experiência com gerenciamento de projeto de software? 1 2 3 4 5
- Experiência com inspeções de software? 1 2 3 4 5
- Experiência com planejamento de inspeções de soft.? 1 2 3 4 5
- Experiência com testes de integração de software? 1 2 3 4 5
- Experiência desenhando interfaces com o usuário? 1 2 3 4 5
- Experiência com avaliação de usabilidade de software? 1 2 3 4 5

Experiência em Contextos Diferentes

Nós usaremos esta seção para compreender quão familiar voce está com vários sistemas que poderão ser utilizados como exemplos ou para exercícios durante o curso.

Por favor, indique o grau de experiência nesta seção seguindo a escala de 3 pontos abaixo:
1 = Eu não tenho familiriadade com a área. Eu nunca fiz isto.
3 = Eu utilizo isto algumas vezes, mas não sou um especialista.
5 = Eu sou muito familiar com esta área. Eu me sentiria confortável fazendo isto.
Quanto voce sabe sobre...

- Utilizar sistema de gerenciamento financeiro? 1 3 5
- Utilizar um sistema de submissão de artigos científicos? 1 3 5

B.2. Questionário Pós-Estudo

Identificação: _____
Data: _____

Por favor, responda o questionário abaixo de forma a nos permitir o registro de sua opinião sobre a aplicação do checklist de inspeção em casos de uso, além dos procedimentos envolvidos na condução do estudo experimental. A sua opinião é muito importante para nós.

Sobre o *Checklist* de Inspeção em Casos de Uso

1.Como você classifica o grau de dificuldade da aplicação do *checklist*?

___Muito Fácil ___Fácil ___Médio ___Difícil ___Muito Difícil

2.Na sua opinião, quais aspectos do checklist tornam sua aplicação fácil/difícil de usar?

_____
_____
_____
_____
_____

3.Você seguiu o checklist completamente?

___ Não. Eu o ignorei completamente.
___ Parcialmente. Eu tentei, mas não segui o checklist na maior parte do tempo.
___ Completamente. Segui o checklist questão a questão.

4.As questões do checklist fizeram sentido como apoio à identificação de defeitos?

___ Não. As questões focam atributos de qualidade dos casos de uso que não representam defeitos.
___ Parcialmente. As questões apontam atributos de qualidade dos casos de uso que devem ser observados, mas a confirmação de que representavam defeitos dependeram da minha interpretação.
___ Completamente. As questões focam atributos de qualidade dos casos de uso que representam defeitos, e permitiram a identificação imediata de defeitos.

5.Considerando a perspectiva de qualidade adotada, como o checklist auxiliou na identificação de defeitos nos casos de uso?

___ Negativamente. O checklist atuou como um obstáculo. O meu desempenho teria sido melhor se eu não o tivesse utilizado.
___ Neutro. Acho que encontraria os mesmos defeitos caso não tivesse utilizado o checklist.
___ Positivamente. O checklist me auxiliou na detecção de defeitos. Talvez, não teria detectado alguns defeitos, caso não tivesse utilizado o checklist.

6.Você utilizaria o checklist em inspeções futuras?

___Não. ___ Talvez. ___ Sim.

7.Na sua opinião, como o checklist poderia ser melhorado?

Em relação à ordem das questões:_____

Questões para detecção de defeitos (Claras? Puderam ser entendidas?): _

Número de questões (Quantidade adequada? Alguma questão ausente ou excedente?):

Outros Aspectos:

8.Descreva detalhadamente quais as diferenças você identificou na utilização da abordagem ad-hoc e a abordagem checklist empregada nesta inspeção. (sugestões de alguns tópicos a serem discutidos: facilidade, foco da inspeção, tipos e severidade das discrepâncias identificadas)

9. Considerando a sua experiência e aprendizagem durante as atividades de inspeção, quais atributos de qualidade ou questões do checklist você internalizou mais? Por que você acha que especificamente estes atributos/questões foram os que você internalizou mais e não os outros atributos/questões? Qual técnica você experimentou um maior aprendizado, ad-hoc ou checklist? Por quê?

<u>Sobre os Procedimentos do Experimento</u>

10.Como você avalia o treinamento conduzido?

___ Insuficiente.
___ Neutro.
___ Adequado.

11.Você achou o tempo de inspeção adequado?

___ Não. O tempo de inspeção não foi suficiente para a aplicação do checklist.
___ Sim. O tempo de inspeção foi suficiente para a aplicação do checklist.

12.Como você classifica a qualidade do documento de requisitos utilizado na inspeção?

___ Má qualidade.
___ Neutra.
___ Boa qualidade.

13.Por favor, registre quaisquer comentários que julgar pertinente

_____
_____
_____
_____
_____
_____
_____

Muito obrigado por sua participação!

B.3. Checklist de Inspeção em Diagrama de Caso de Uso

Item de inspeção	Questão	Ref. Regra do Modelo	Impacto
Nome do ator	O nome do ator reflete o seu papel no sistema, evitando títulos de cargos, nomes de áreas ou atividades relacionadas com a estrutura da organização?	2.1	M
Nome do caso de uso	O nome do caso de uso utiliza verbo seguido de substantivo(s), podendo o substantivo ter adjetivo, e o verbo se encontra no modo infinitivo ou no modo indicativo em tempo presente, e utiliza a voz ativa ao invés da passiva? (Ex: "Imprimir atestado" ou "Imprime atestado" e não "Impressão de atestado" ou "Atestado impresso")	2.2	M
	O nome do caso de uso é de fácil compreensão e evita a utilização de verbos ou substantivos específicos de sistemas, de forma a tornar difícil a compreensão para o leitor, clientes e usuários?	2.3	M
	O nome do caso de uso reflete funções de forma agrupada com um resultado de valor para o usuário, sem derivar-se em múltiplos casos de uso que poderiam ser agrupados?	1.1	A
	Os casos de uso evitam a utilização de especificação de múltiplos relatórios ou CRUD (Create, Retrieve, Update, Delete ou Criar,Consultar com finalidade de cadastro,Atualizar, Deletar), havendo no máximo um caso de uso genérico para todos os cadastros? (Ex: "Gerenciar Cadastros", "Visualizar Relatórios") (Considerar 1 erro para cada caso de uso de cadastro para o mesmo ator e subtrair 1 da Qtde no Registro de Inspeção).	1.2	B
Relacionamento entre atores e casos de uso	Cada ator tem relacionamento com ao menos um caso de uso e é representado por uma linha sólida, podendo ou não ser seguida de uma flecha aberta indicando a direção da invocação inicial do relacionamento?	2.4	A
Relacionamento entre casos de uso e casos de uso (se houver)	Se houver relacionamentos entre casos de uso e casos de uso, são um entre estes: Relacionamento de Inclusão: Representado por uma linha hifenizada, seguida de flecha aberta apontando para o caso de uso incluído, sendo este o oposto do caso de uso base, com estereótipo que indique a inclusão <<include>>? Relacionamento de Extensão: Representado por uma linha hifenizada, seguida de uma flecha aberta apontando para o caso de uso base, seguida de estereótipo que indique a extensão <<extend>>? Relacionamento de Generalização: Representado por uma linha sólida, seguida de uma flecha fechada, sendo na origem o caso de uso especializado e no destino o caso de uso genérico?	2.5	A
	Se houver relacionamentos entre os casos de uso e se removidos todos os relacionamentos entre eles, isto é, os casos de uso incluídos, os de extensão e os de generalização, o propósito do modelo de caso de uso continua claro no diagrama?	2.6	A
Relacionamento entre casos de uso e atores	Cada caso de uso tem relacionamento com ao menos um ator (excetuando-se casos de uso incluídos, de extensão de generalização) e é representado por uma linha sólida, podendo ou não ser seguida de uma flecha aberta indicando a direção da invocação inicial do relacionamento?	2.7	A
Relacionamento de inclusão (se houver)	Se houver caso de uso incluído, ele utilizado para dividir suas funcionalidades, portanto tem relacionamento de inclusão com mais de um caso de uso?	2.8	A
Relacionamento de extensão (se houver)	Se houver caso de uso de extensão, ele é utilizado sob as seguintes situações: funcionalidades adicionadas a um produto existente, funcionalidades que podem ser compradas separadamente, descrições de exceções que são tão complexas que se tornam maiores do que o caso de uso de base, funcionalidades que serão introduzidas em versões posteriores?	2.9	A
Relacionamento de generalização (se houver)	Se houver caso de uso especializado, advindo de um relacionamento de generalização, ele é utilizado para representar o reuso de um caso de uso em famílias de sistemas com funcionalidades variáveis?	2.10	A
Relacionamento entre atores e atores (se houver)	Se houver relacionamento entre atores e atores, ele é de generalização, representado por uma linha sólida seguida de uma flecha fechada, sendo na origem o ator especializado e no destino o ator generalizado?	2.11	A
	Se houver relacionamento entre os atores e atores, ele é utilizado apenas para representar que o ator especializado tem características comuns ao ator generalizado, mas também acessa casos de uso adicionais, evitando representar hierarquias como as de permissão de segurança ou hierarquias funcionais na organização?	2.12	A
Breve descrição	Há presença de breve descrição de atores, que especifica de forma clara o que eles representam?	2.13	B
	Há presença de breve descrição de casos de uso, que especifica de forma clara o seu propósito?	2.14	B

B.4. Checklist de Inspeção em Diagrama de Caso de Uso

Item de inspeção	Questão	Ref. Regra do Modelo	Impacto
Forma e Organização	A descrição de caso de uso é a de um caso de uso representado no diagrama?	1.1	A
	A descrição de caso de uso contém nome do caso de uso, nome do ator, fluxo básico e alternativo? (Se não houver um fluxo alternativo definido, considerar se ele está especificado dentro do fluxo básico)	1.2	A
	A descrição de caso de uso com mais de uma página contém índice e paginação?	2.5	B
	Se houver tabelas ou figuras, elas têm explicação adicional de forma que fiquem compreensíveis para o leitor?	2.6	B
	Se houver referências, essas são numeradas ou nomeadas da mesma forma na origem (descrição de caso de uso) e no destino (a própria referência)?	5.2	B
Escrita	As frases representam um diálogo entre ator e sistema, evidenciando a ação do ator e a resposta do sistema?	2.1	M
	As frases se utilizam de subtítulos para comunicar as idéias chaves dos fluxos de forma mais clara?	2.4	B
	As frases são construídas em voz ativa? (ex.: "Sistema valida a quantia informada" em vez de "A quantia informada deve ser validada pelo sistema").	2.2	B
	As frases utilizam o tempo presente?	2.3	B
Termos utilizados	São evitados termos sem quantificação precisa, como "muito", "pouco", "adequado", "claro", "fácil" "longo", "curto", "rápido" "etc"?	3.1	M
	São evitados termos que indicam opção, como "possivelmente", "alternativamente", "no caso", "se", etc, sem especificar um fluxo alternativo?	3.2	M
	Os termos passíveis de mais de uma interpretação constam em glossário, com clara definição?	4.1	A
	Uma vez utilizado um termo, ele é mantido para referenciar-se ao mesmo elemento?	5.1	M
	São evitados termos que indicam a prematura especificação de interface, tais como "clicar" "botão" etc?	6.1	B
Conteúdo	As funcionalidades se restringem ao quê o sistema deve fazer e não em como, evitando a definição explícita de código na especificação?	2.7	M
	A descrição evita requisitos de negócio sem ação direta ao sistema?	2.8	M
	Há presença de breve descrição ou resumo no início da descrição de caso de uso, que especifique de forma clara o seu propósito?	2.11	B
	O fluxo básico está aparentemente completo, isto é, há inexistência de evidências claras de incompleteza na especificação?	1.3	A
	O fluxo alternativo está aparentemente completo, isto é, há inexistência de evidências claras de incompleteza na especificação?	1.3	A
	As frases são numeradas para que possibilitem a rastreabilidade?	7.1	M
	As frases procuram ser objetivas, evitando redundâncias ou presença de informações evidentemente desnecessárias?	8.1	M
Artefatos de suporte	O caso de uso é acompanhado de protótipo de interface a fim de aumentar a sua compreensibilidade?	2.9	A
	O caso de uso é acompanhado de especificação de requisitos não funcionais separadas do fluxo de eventos do caso de uso ou em documento de especificação suplementar?	2.9	M
	O caso de uso é acompanhado de modelo de domínio, mostrando os relacionamentos entre os principais conceitos do sistema, a fim de aumentar a sua compreensibilidade?	2.9	M
	Se houver regras de negócios associadas, estas estão separadas dos fluxos de evento do caso de uso ou em documento de especificação de regras de negócios?	2.9	B

B.5. Formulário de Registro de Discrepâncias

REGISTRO DE INSPEÇÃO EM DESCRIÇÃO DE CASO DE USO

Nome do projeto:
Data da inspeção:
Responsável pela inspeção:
Hora inicial:
Hora final:
Tempo despendido: (hh:mm)
Quantidade de casos de uso inspecionados:
Quantidade de atores inspecionados:

Nº. da questão	Impacto (A/M/B)	Registros de não-conformidades				
		N/A	Sim	Não	Qtde	Descrição (Localização, obs.)

Observações: