Universidade Federal do Pampa
Thiago Cassio Krug

Um Sistema Multiagente para Suporte à Inspeção de Software

Alegrete
2014

Thiago Cassio Krug

Um Sistema Multiagente para Suporte à Inspeção de Software

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. Me. João Pablo Silva da Silva

Alegrete

2014

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

K094s Krug, Thiago Cassio
Um sistema multiagente para suporte à inspeção de software / Thiago Cassio Krug.
92 p.

Trabalho de Conclusão de Curso(Graduação)-- Universidade Federal do Pampa, ENGENHARIA DE SOFTWARE, 2014.
"Orientação: João Pablo Silva da Silva".

1. Inspeção de Software. 2. Sistema Multiagente. 3. Tropos. 4. Qualidade de Software. I. Título.

Thiago Cassio Krug

Um Sistema Multiagente para Suporte à Inspeção de Software

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Trabalho de Conclusão de Curso defendido e aprovado em 24 de março de 2014

Banca examinadora:

Prof. Me. João Pablo Silva da Silva
Orientador

Prof. Me. Jean Felipe Patikowski Cheiran
UNIPAMPA

Prof. Me. Sam da Silva Devincenzi
UNIPAMPA

Este trabalho é dedicado à minha família,
namorada, colegas e amigos.

Agradecimentos

Agradeço primeiramente a minha família, que me apoia em todos os momentos da vida, me trazendo conforto e força.

Ao meu orientador João Pablo Silva da Silva pela paciência, dedicação, ajuda e conselho, que foram fundamentais durante o desenvolvimento deste trabalho e durante o curso.

Um agradecimento especial para a minha namorada, Rubia Cavalheiro, por me apoiar e me aturar durante esse período de muitos estudos e dedicação.

Agradeço também aos meus colegas e amigos Bruno Vicelli, Mateus Dal Forno, Helison Teixeira, Rafhael Cunha, Rafael Amorim, Nasser Rahman, Marcelo Lopes, Wolmir Nemitz, Juliano Rodovalho e Renan Uchôa, pela amizade e companheirismo. Eu só tenho uma coisa para falar para vocês: Só o ouro! =D.

Agradeço a dedicação e profissionalismo dos professores que me acompanharam durante a graduação, compartilhando seu tempo, paciência e conhecimento.

Por fim, agradeço a Universidade Federal do Pampa (UNIPAMPA) pela oportunidade de um ensino de qualidade de forma gratuita.

Obrigado à todos!

"O insucesso é apenas uma oportunidade
para recomeçar com mais inteligência."
(Henry Ford)

Resumo

O setor de tecnologia da informação é considerado estratégico para empresas que buscam a melhoria de seus produtos e serviços. A compra de produtos de software com qualidade é requisitada cada vez mais. Empresas desenvolvedoras de software podem atingir altos níveis de qualidade através da inspeção de software, a qual permite a detecção precoce de falhas. Entretanto sua adoção ainda enfrenta problemas. Foi criado um projeto de pesquisa para auxiliar no suporte às inspeções de software, no qual o presente trabalho faz parte. Este trabalho apresenta um sistema de apoio ao processo de inspeção de software da norma IEEE-1028, baseado em multiagentes, que objetiva a automação de algumas atividades desse processo. O sistema é formado por cinco agentes que implementam as regras necessárias para a automação das atividades de enviar o artefato para a inspeção, reunir e distribuir os materiais de revisão, além da notificação do time de inspeção e do inspetor líder. A modelagem e projeto do sistema multiagente são guiados através da metodologia Tropos, enquanto a implementação é realizada por meio da plataforma JADE. O sistema é verificado através de testes unitários e de integração, e após testado em conjunto com o sistema de suporte às inspeções de software. Conclui-se que o sistema multiagente atende a todos os objetivos propostos e é funcional, porém não é provado que o sistema multiagente aumenta a produtividade em ambientes reais de inspeção de software. Espera-se que o sistema multiagente contribua na otimização e melhoria das atividades da inspeções de software, possibilitando às empresas se beneficiarem dos resultados da inspeção de software.

Palavras-chave: Inspeção de Software, Sistema Multiagente, Tropos, Qualidade de Software.

Abstract

The information technology sector is considered strategic for companies who seeks to improve their products and services. Purchase of software products with quality is required more and more. Software development companies may achieve quality high levels through software inspection, that enables early fault detection. However its adoption still faces problems. A research project to help support the software inspections, in which the present work is part, was created. This paper presents a system to support software inspection process from IEEE-1028 standard, based on multiagents, which aims at automating some activities in this process. The system consists of five agents that implement the necessary rules for the automation of sending the artifact for inspection activity, assemble and distribute the review materials activities, in addition to notification of the inspection team and inspection leader. The multiagent system modeling and design are guided by the Tropos methodology, while the implementation is performed through the JADE platform. The system is verified through individual and integrated testing, and further tested in combination with the software inspections support system. It is concluded that the multiagent system caters to all proposed goals and is functional, but is not proved that the multiagent system increases productivity in real software inspection environments. It is expected that the multiagent system contributes to the optimization and improvement of software inspection activities, enabling companies to benefit from the results of the software inspection.

Key-words: Software Inspection, Multiagent Systems, Tropos, Software Quality.

Lista de ilustrações

Lista de tabelas

Lista de siglas

ACL *Agent Communication Language*

API *Application Programming Interface*

CMMI *Capability Maturity Model Integration*

FIPA *Foundation for Intelligent Physical Agents*

IEEE *Institute of Electrical and Electronics Engineers*

J2ME *Java Platform, Micro Edition*

JADE *Java Agent Development*

JSP *Java Server Pages*

LESA Laboratório de Engenharia de Software Aplicada

LGPL *Library Gnu Public License*

PBR *perspective-based reading*

PPQA *Process and Product Quality Assurance*

XAF *XQuery-based Analysis Framework*

Sumário

1 Introdução

Empresas buscam cada vez mais qualidade na compra de produtos de software. Essas empresas consideram a tecnologia da informação um setor estratégico para a melhoria e inovação de seus produtos e serviços (BARTIÉ, 2002). Logo, os fornecedores de software precisam adaptar-se a esse ambiente e atingir um alto nível de maturidade. Nesse cenário se destaca a engenharia de software, que engloba um processo, métodos e ferramentas para o desenvolvimento de software economicamente viável e eficiente (PRESSMAN, 2006). Além disso, a engenharia de software atua em todas as características da produção de software, cobrindo desde a especificação até a manutenção do sistema (SOMMERVILLE, 2007).

Uma maneira de se desenvolver um software economicamente viável e eficiente é através da detecção precoce de falhas. Segundo Pressman (2006), o custo para a correção de um erro aumenta gradativamente durante o ciclo de vida de desenvolvimento. Ao passo que uma falha é inserida durante o levantamento de requisitos, a correção dessa falha custará por volta de 10 vezes durante o processo de codificação, chegando por volta de 40 a 1000 vezes no processo de produção. Logo, quanto menor for o intervalo entre a inserção e a detecção de uma falha, menor será custo envolvido na correção. Uma forma de detectar essas falhas com antecedência é através de revisões de software.

A *Institute of Electrical and Electronics Engineers* (IEEE) separa as revisões de software em revisões sistemáticas e revisões não-sistemáticas. O termo sistemático significa bem definido, no qual há a participação do time, documentação dos resultados e procedimentos para a realização da revisão. Revisões que não seguem tais características da norma IEEE-1028 são consideradas revisões não-sistemáticas (IEEE, 2008).

Segundo IEEE (2008), a inspeção de software é um processo sistemático para verificar se o produto de software satisfaz suas especificações, exibe os atributos de qualidade especificados, segue as regulamentações, padrões, orientações, planos, especificações e procedimentos, entre outros. Desde sua criação em 1976 por Michael E. Fagan (FAGAN, 1999), muitas empresas têm obtido resultados positivos com a adoção da inspeção de software. Há casos que o seu uso detectou uma taxa de 60 a 90 por cento de todos os defeitos existentes do sistema durante o ciclo de vida de desenvolvimento (FAGAN, 1986).

Apesar dos benefícios relatados, a adoção de inspeções de software ainda enfrenta problemas. De acordo com Denger e Shull (2007) é difícil de medir o esforço despendido em uma inspeção de software e relacionar com o ganho na qualidade final do produto. Além disso, o processo de inspeção não é suficientemente adaptado para o contexto do projeto e dos membros do time de desenvolvimento atuais. Outro fator que dificulta a

adoção é o investimento inicial parecer muito alto, pois não há ligação entre as atividades da inspeção com as atividades usuais desempenhadas pelos envolvidos. Dessa forma, os participantes da inspeção se sentem como se fossem retirados de suas responsabilidades primárias.

A coleta de materiais necessários para a inspeção caracteriza uma dificuldade a mais para o processo. É necessário que o responsável pela inspeção reúna os itens de revisão, padrões e orientações referentes ao artefato a ser inspecionado. Além disso, a distribuição desse material para os pares custa tempo. Enfim, após a reunião de inspeção, os artefatos revisados precisam ser monitorados pelo líder da inspeção. Dessa maneira, ele precisa estar pró-ativamente em busca de informações e sobre o andamento das correções.

Para dar suporte ao processo de inspeção de software e possibilitar um aumento em sua utilização foi criado um projeto de pesquisa vinculado ao Laboratório de Engenharia de Software Aplicada (LESA)[1] da Universidade Federal do Pampa. Esse projeto tem por objetivo prover suporte à aplicação do processo de inspeção de software possibilitando o aumento da produtividade e controle dos materiais e dados gerados durante uma inspeção. Além disso, as informações referentes à inspeção ficam centralizadas e podem ser consultadas pelos membros a qualquer momento em qualquer lugar, possibilitando até que sejam realizadas inspeções com times distribuídos. Por fim, o projeto de pesquisa também tem o propósito de possibilitar a redução do impacto da aplicação do processo de inspeção de software durante o processo de desenvolvimento de software.

1.1 Motivação

O projeto de pesquisa possui como um de seus objetivos possibilitar o aumento da produtividade nas inspeções de software. Uma forma de viabilizar o aumento de produtividade da execução do processo de inspeção é através da automatização de atividades, de maneira que o esforço seja direcionado a análise dos artefatos, e não para a coleta e distribuição dos materiais envolvidos no processo. Uma alternativa para a automatização de atividades é através do uso de agentes de software, que são programas computacionais dotados de autonomia, pró-atividade e sociabilidade (RUSSELL; NORVIG, 2009).

Dessa maneira, a motivação deste trabalho está no uso de agentes para possibilitar o aumento da produtividade através da automatização de atividades do processo de inspeção de software. Esta motivação é derivada dos objetivos do projeto de pesquisa em que o presente trabalho faz parte. Assim, a realização deste trabalho contribui tanto para

[1] O LESA é um grupo de pesquisa que desenvolve projetos no intuito de prover soluções que maximizem a colaboração e produtividade em projetos de desenvolvimento de software, e que acarretem em um aumento da qualidade dos produtos de software desenvolvidos. http://porteiras.s.unipampa.edu.br/lesa/.

o projeto de pesquisa quanto para a engenharia de software através da possibilidade de uma maior utilização do processo de inspeção de software.

1.2 Objetivo Geral e Específicos

O presente trabalho tem por objetivo geral desenvolver uma solução para automatizar atividades do processo de inspeção de software para possibilitar uma maior produtividade. Como objetivos específicos são definidos:

- modelar os objetivos e intenções do processo de inspeção de software para evidenciar o domínio e as relações entre as entidades;
- identificar quais atividades do processo de inspeção de software podem ser automatizadas;
- integrar a solução desenvolvida em sistema de suporte para a realização das verificações e validações.

1.3 Principais Contribuições

As principais contribuições deste trabalho são:

- a modelagem do domínio do processo de inspeção de software, no qual pode ser aproveitada para a implementação de outros sistemas multiagentes;
- a utilização e documentação do processo de inspeção de software utilizando a metodologia de engenharia de agentes Tropos;
- a estrutura de uma estratégia para a automatização de atividades do processo de inspeção de software;
- o projeto e a implementação de um sistema multiagente que automatiza as atividades de: enviar um artefato para a inspeção, reunir e distribuir os materiais de inspeção para os inspetores, notificar os membros do time de inspeção sobre a nova inspeção de software, e verificar as correções das anomalias;

1.4 Organização do Trabalho

O restante do documento está organizado da seguinte maneira:

- Capítulo 2: apresenta os conceitos necessários para o entendimento da solução proposta. São descritos o processo de inspeção de software, os conceitos de um sistema multiagente, uma metodologia de engenharia de agentes, e a plataforma de desenvolvimento de agentes.
- Capítulo 3: são apresentados os trabalhos relacionados obtidos através de uma revisão sistemática da literatura, relacionando-os ao presente trabalho.
- Capítulo 4: são apresentados o sistema multiagente para suporte às inspeções de software, a modelagem realizada do domínio e da solução, além da implementação.
- Capítulo 5: são apresentados os testes realizados para a verificação do sistema desenvolvido e uma análise dos dados.
- Capítulo 6: relata as considerações finais sobre a execução do trabalho, sendo descrito o atendimento ao objetivo proposto, e apresenta as potenciais evoluções através dos trabalhos futuros.

2 Fundamentação Teórica-Tecnológica

Neste capítulo é apresentada a fundamentação teórica-tecnológica que serve de base para os conceitos e tecnologias utilizadas para o desenvolvimento deste trabalho. Dessa forma, é descrito um entendimento básico que viabiliza a compreensão das soluções. Na seção 2.1 é apresentado o processo de inspeção de software padronizado pela Norma IEEE Std 1028™-2008, e um exemplo fictício da realização do processo de inspeção. Na seção 2.2 é abordado sobre agentes e sistemas multiagentes. Além disso, é apresentada a metodologia de engenharia de software orientada a agentes chamada Tropos. Em seguida é relatado o *framework* JADE para desenvolvimento de sistemas multiagentes. Por fim, na seção 2.3 é realizado o fechamento do capítulo, retomando os conceitos abordados e evidenciando suas características.

2.1 Inspeção de Software

A inspeção de software é reconhecida pelos benefícios que traz para o aumento da qualidade e a redução do custo em projetos de desenvolvimento de software. Das abordagens de inspeção de software existentes, o projeto de pesquisa selecionou o processo de inspeção de software padronizado através da Norma IEEE Std 1028™-2008, que descreve de forma detalhada e sistemática como o processo deve ser aplicado.

Segundo IEEE (2008) a inspeção de software é um processo sistemático para a detecção e identificação de anomalias no produto de software. É um exame visual conduzido por membros imparciais com treinamento nas técnicas de inspeção. Uma anomalia é qualquer condição que desvia das expectativas baseadas em documentos já estabelecidos, como requisitos, padrões, etc., ou da percepção e experiência de alguma pessoa.

Uma vantagem da inspeção de software é que a mesma não é direcionada para apenas um tipo de artefato de software, possibilitando seu uso em vários outros produtos que resultam do processo de desenvolvimento de software, desde que satisfaçam os critérios de entrada da inspeção. Pode-se citar os artefatos como: documento de requisitos, código-fonte, plano de testes, etc.

De acordo com Sommerville (2007), pode-se destacar que a utilização de inspeções ajuda na detecção de muitos erros em uma única instância de inspeção. A inspeção de software ajuda a encontrar problemas relacionados com a semântica dos artefatos, validando-os de acordo com os padrões estabelecidos. Além disso, o sistema não precisa estar completamente desenvolvido para realizar uma inspeção.

2.1.1 Processo de Inspeção

Seguindo a Norma IEEE-1028, a entrada da inspeção deve possuir um conjunto de itens: uma homologação dos objetivos da inspeção, o produto de software a ser inspecionado, os procedimentos da inspeção documentados, os formulários para reportar as decorrências da inspeção, a lista de anomalias ou de problemas, e documentação utilizada como fonte pelo autor para desenvolver o produto de software sob análise. Podem ser incluídos alguns itens para a inspeção que não são obrigatórios, tais como: *checklists* de inspeção, critérios de qualidade para uma reinspeção, um produto de software previamente inspecionado e aprovado, qualquer regulamentação, padrão, orientação, plano, especificação e procedimento que o produto de software contra o qual possa ser verificado, especificação de hardware, instrumentação, ou outra especificação de produto de software, dados de performance, e categorias de anomalias[1] (IEEE, 2008).

As inspeções devem ser registradas em um documento de planejamento do projeto. Elas são planejadas no plano das apreciações do projeto, onde são documentadas as análises que serão realizadas na fase de projeto. Inspeções adicionais podem ser executadas durante o ciclo de vida do desenvolvimento de software (PAULA FILHO, 2009).

Uma inspeção só deve ser conduzida quando as entradas da inspeção relevantes estiverem disponíveis. De acordo com IEEE (2008), é necessário alguns critérios de entrada mínimos para que ocorra um processo de inspeção de software. Um critério é se o produto de software é completo e segue as normas e padrões definidos, de acordo com a determinação do inspetor líder. Outro critério é o uso de ferramentas automatizadas para detecção de erros, para eliminar erros antes da inspeção. Os marcos anteriores que o produto de software depende precisam estar prontos de acordo com o plano do projeto. Por fim, a documentação de suporte disponível estabelece mais um critério para a entrada da inspeção de software (IEEE, 2008).

Para a realização da inspeção de software, os papéis de inspetor líder, escrivão, leitor, autor e inspetor devem estar presentes. Todos os participantes da inspeção são inspetores, porém, o autor não pode atuar como inspetor líder, leitor ou escrivão. Os outros papéis podem ser divididos entre os membros do time. Além disso, os participantes da inspeção podem assumir mais de um papel. Podem ser descritas as responsabilidades de cada papel (IEEE, 2008):

- Inspetor líder: é responsável pelo planejamento e organização da inspeção, determinar os componentes de produto software que serão inspecionados durante a inspeção, garantir que a inspeção seja conduzida de forma ordenada e atinja os objetivos seus objetivos, garantir que a os dados da inspeção sejam coletados, e publicar a saída da inspeção.

[1] IEEE Std 1044™-1993, IEEE Standard Classification for Software Anomalies.

- Escrivão: possui a responsabilidade de documentar as anomalias, decisões, recomendações, e levantamentos feitos pelo time de inspeção, e registrar os dados da inspeção necessários para o processo de análise. O inspetor líder pode assumir o papel de escrivão.

- Leitor: é responsável por conduzir o time de inspeção através do produto de software de uma forma compreensiva e lógica, interpretando seções e focando em aspectos importantes. O produto de software pode ser divido e designado a diferentes leitores para a redução do tempo de preparação.

- Autor: é responsável pelo produto de software satisfazer os critérios de entrada da inspeção, contribuir para a inspeção através do entendimento do produto de software, e realizar qualquer retrabalho necessário para que o produto de software atinja os critérios de saída da inspeção.

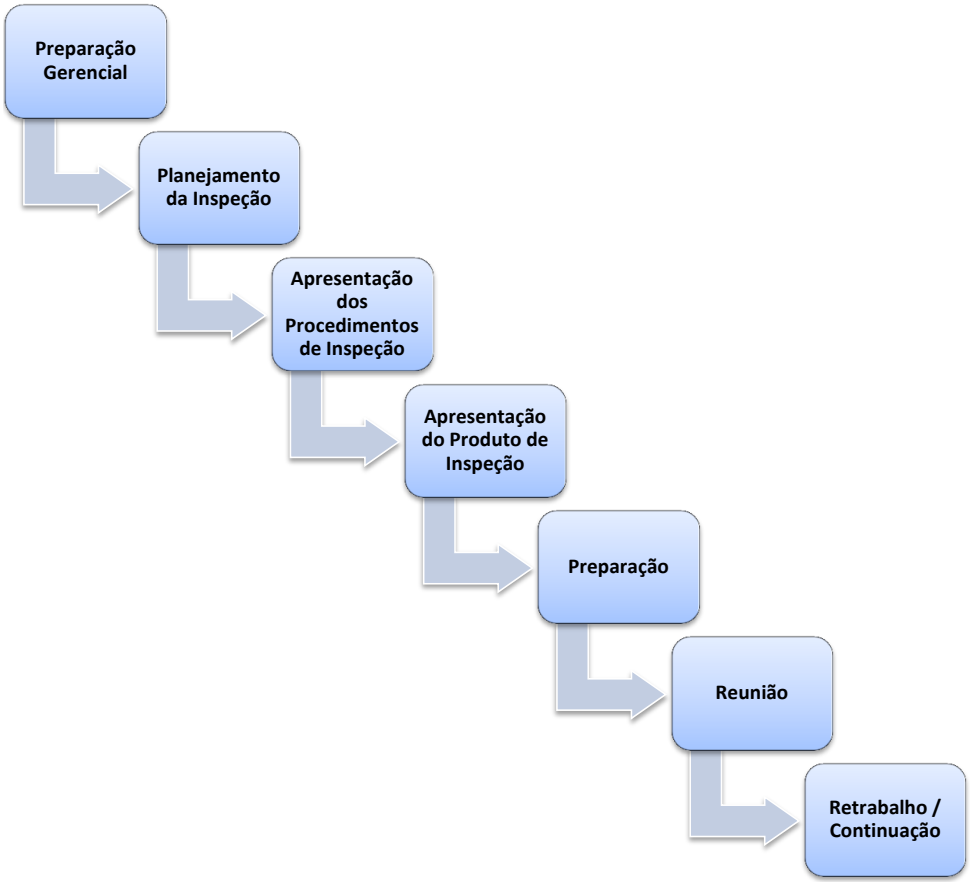
- Inspetor: possui a responsabilidade de identificar e descrever as anomalias do produto de software. Os inspetores devem ser escolhidos de acordo com sua perícia e representar diferentes pontos de vista durante a reunião. Além disso, alguns inspetores devem ser selecionados para garantir a cobertura de acordo com tópicos específicos, como a conformidade por padrões específicos, sintaxe ou precisão das figuras, entre outros.

O processo de inspeção pela Norma IEEE-1028 está dividido em sete fases como são apresentadas na Figura 1. A fase de preparação gerencial é dedicada para o ajuste do ambiente e treinamento dos gerentes. Na fase de planejamento da inspeção é feita a escolha e treinamento do time de inspeção, reunido o material para a inspeção e enviado para os inspetores. Em seguida, na fase de apresentação dos procedimentos da inspeção, o líder da inspeção esclarece as dúvidas em relação ao artefato a ser inspecionado.

Seguindo o processo da Figura 1, na apresentação do produto da inspeção, o autor descreve o artefato sob inspeção. Na fase de preparação os inspetores realizam uma avaliação do artefato e devolvem para o inspetor líder. Na reunião são discutidas as anomalias encontradas e registradas na lista de anomalias, e ao final decidido se o artefato deve ser reinspecionado. Por fim, na fase de retrabalho / continuação a lista de anomalias é repassada para o autor corrigir as falhas, no qual o artefato pode passar por um novo processo de inspeção ou apenas ser inspecionado por uma pessoa designada.

O processo de inspeção é considerado completo quando possui um conjunto de documentos de evidência que demostram os resultados da inspeção. Compõem esse conjunto o documento do projeto do produto de software, os membros do time de inspeção, dados sobre a reunião de inspeção, o produto de software inspecionado, a lista de anomalias, os objetivos da inspeção, informações sobre o tempo gasto pelos membros do time de

Figura 1 – Processo de inspeção da Norma IEEE-1028.



inspeção, e o total de tempo de retrabalho. Alguns dados que se mostram importantes são a sumarização das anomalias, o tempo de esforço estimado para o retrabalho e se esse esforço foi significante, além do ganho em dinheiro pela correção dos itens inspecionados comparados com seu custo se identificados em um estágio posterior (IEEE, 2008).

Os dados coletados durante a inspeção são utilizados para avaliar a qualidade do produto e a eficiência e eficácia do processo de inspeção. Para possibilitar essa análise, as anomalias precisam ser classificadas, categorizadas e ordenadas. O processo de inspeção deve ser analisado regularmente para identificar problemas e benefícios em seu uso. Dessa forma a inspeção atingirá seus objetivos com eficiência e eficácia (IEEE, 2008).

2.1.1.1 Preparação Gerencial

Nesta fase os gerentes são preparados para garantir que a inspeção seja realizada de acordo com as normas e padrões aplicáveis. Ao final de sua preparação, os gerentes devem realizar uma série de procedimentos para garantir que a inspeção será realizada com sucesso. Alguns procedimentos são planejar o tempo e recursos despendidos pelas inspeções, prover os recursos, infraestrutura e instalações necessárias para planejar, exe-

cutar e gerenciar as inspeções, além de prover treinamento e orientação nos procedimentos da inspeção do projeto, etc (IEEE, 2008).

2.1.1.2 Planejamento da Inspeção

Na fase de planejamento da inspeção, o autor deve reunir os materiais necessários para a inspeção e entregar ao inspetor líder. Os materiais devem compreender o produto de software a ser inspecionado, assim como os artefatos que são relacionados a ele, como padrões, planos e orientações (IEEE, 2008).

O líder da inspeção deve identificar as pessoas que devem fazer parte do time de inspeção, sempre observando se o membro possui habilidades e conhecimento o suficiente para poder avaliar os artefatos. Após, ele designa responsabilidades específicas para cada membro do time (IEEE, 2008).

O líder da inspeção escolhe o dia, hora e local para a reunião de inspeção e notifica os membros do time. Conseguinte, ele distribui o material da inspeção para os participantes da reunião, reservando um tempo adequado para a preparação. Seleciona um horário para a distribuição do material e do *feedback* dos participantes, e após envia-os para o autor (IEEE, 2008).

Depois o inspetor líder especifica o escopo da inspeção, incluindo as partes prioritárias que serão revisadas na reunião. Por fim, é responsabilidade do líder da inspeção criar uma taxa de inspeção, a qual regula a velocidade da inspeção por artefato revisado (IEEE, 2008).

2.1.1.3 Apresentação dos Procedimentos de Inspeção

Na apresentação dos procedimentos de inspeção, o inspetor líder deve responder a qualquer dúvida referente aos *checklists* da inspeção e aos papéis e responsabilidades dos membros da equipe de inspeção. Além disso, deve apresentar os dados como o relatório de tempo para a avaliação dos membros, o número de anomalias encontradas anteriormente, a taxa de inspeção recomendada, etc (IEEE, 2008).

2.1.1.4 Apresentação do Produto da Inspeção

Na apresentação do produto da inspeção o autor expõe as características do artefato a ser inspecionado. Dessa forma, os inspetores obterão conhecimento do produto de software. Essa apresentação pode ser assistida por outros membros de outros projetos. Assim haverá uma comunicação entre as equipes e a expansão do conhecimento (IEEE, 2008).

2.1.1.5 Preparação

Os artefatos de software sob análise são avaliados e as anomalias encontradas são repassadas para o inspetor líder. Cada anomalia deve ser classificada e verificado se o tempo da inspeção será utilizado de forma eficiente. Caso existam muitas falhas com nível de seriedade muito alto o inspetor líder pode cancelar a reunião de inspeção até que o produto a ser avaliado tenha um nível que satisfaça os critérios de entrada para um processo de inspeção. Todas as anomalias encontradas são repassadas ao autor para que sejam corrigidas (IEEE, 2008).

A leitura do material deve ser planejada pelo inspetor líder ou pelo leitor do material. A forma da leitura do material deve colaborar com a inspeção. Algumas formas organização para a leitura são através da sequência do artefato, hierarquia, fluxos de dados, fluxo de controle, *bottom up*, *top down*, etc (IEEE, 2008).

O leitor deve estar preparado para a apresentação do material na reunião de inspeção, mas é de responsabilidade do inspetor líder garantir que o leitor e os inspetores estejam preparados para a reunião. Caso os membros não estejam prontos para inspeção, o inspetor líder deve reagendar a reunião (IEEE, 2008).

O líder da inspeção deve obter o tempo de preparação de cada inspetor e registrá-lo no documento de inspeção (IEEE, 2008).

2.1.1.6 Reunião

No início da reunião, o inspetor líder faz alguns esclarecimentos sobre as intenções da inspeção. Ele descreve os papéis e responsabilidades dos membros envolvidos, o propósito da inspeção, e o esforço na busca de anomalias. O inspetor líder deve esclarecer que as considerações em relação ao artefato inspecionado devem ser direcionadas para o escrivão, e não para o autor. Por fim, as discussões que se tornem muito extensas devem ser discutidas ao final da reunião ou em uma nova reunião para esse fim (IEEE, 2008).

Após, são apresentadas as anomalias encontradas no produto de software na fase de preparação e registradas pelo escrivão para compor a lista de anomalias resultante da reunião (IEEE, 2008). Em seguida, o artefato é mostrado para os membros da inspeção pelo leitor, e esses membros devem examinar objetivamente e cuidadosamente buscando por possíveis problemas. O escrivão registra cada falha encontrada na lista de anomalias, colocando o local onde ela está presente, sua descrição e classificação. As dúvidas referentes ao produto de software são esclarecidas pelo autor, contribuindo assim com a detecção de problemas. Caso ocorra um desacordo sobre alguma anomalia encontrada, ela deve ser registrada e revisada ao final da reunião de inspeção para que não se desperdice tempo (IEEE, 2008).

Ao final da reunião, o inspetor líder verifica a lista de anomalias descobertas. Os

possíveis desacordos sobre as anomalias são discutidos, buscando seu esclarecimento e não sua possível resolução. As discordâncias que não forem facilmente resolvidas devem ser registradas no relatório de anomalias (IEEE, 2008).

Enfim, é decidido se o produto de software atendeu aos critérios de saída e de qualidade aos quais ele está relacionado. Qualquer retrabalho, correção ou ajuste deve ser descrito e registrado. Por fim, o time deve classificar o produto de software entre três opções: aceito sem nenhum retrabalho ou verificação, aceito com algum retrabalho ou verificação, ou reinspecionar o produto de software (IEEE, 2008).

2.1.1.7 Retrabalho / Continuação

Caso exista algum retrabalho a ser realizado, a lista de anomalias é disponibilizada para que o autor resolva os problemas identificados. Após a resolução das anomalias, o inspetor líder ou algum inspetor designado verifica se tais correções atendem com os critérios estabelecidos para a correção do artefato (IEEE, 2008).

No caso de reinspeção, é agendada uma nova reunião para que se verifique as correções realizadas e se inspecione novamente o produto de software (IEEE, 2008).

2.1.2 Simulação do Processo de Inspeção de Software

Para a exemplificar a execução do processo de inspeção foi realizada a simulação de uma inspeção de software. O cenário é composto de um projeto de desenvolvimento de software que utiliza o processo de desenvolvimento OpenUp[2]. Estão envolvidos nesse projeto seis pessoas: um gerente do projeto, um engenheiro de implantação, um analista, um gerente de implantação, um escrivão técnico, e um testador. O projeto se trata de um sistema de controle financeiro pessoal, que deve ser implantado em dez clientes diferentes. Por fim, o projeto está atualmente na fase de construção.

O processo se inicia quando o gerente do projeto realiza o plano inicial da inspeção, informando os locais que estão disponíveis para a realização das reuniões. É também registrado os materiais (como projetor, quadro, cadeiras, mesa, etc) que ficarão disponíveis para as reuniões de inspeção. Além disso, é planejado a duração máxima das inspeções, calculando-se o quanto será investido para que a inspeção seja realizada com sucesso. Por fim, o gerente escolhe o analista para ser o inspetor líder dessa inspeção, pois o analista já havia realizado outras inspeções como inspetor e inspetor líder.

Em seguida, a fase de planejamento da inspeção se inicia. O inspetor líder escolhe quais artefatos devem ser inspecionados nesse processo. Ele verifica que, nessa fase do projeto, o artefato a ser inspecionado é o plano de implantação do sistema, informando o

[2] OpenUp é uma versão mais leve do Processo Unificado, que utiliza uma abordagem iterativa e incremental para o desenvolvimento de software. http://epf.eclipse.org/wikis/openup/

autor do artefato sobre a escolha. O engenheiro de implantação, que é o autor do artefato, o localiza dentro dos documentos do projeto, e encontra um plano de implantação padrão que serve como guia. Ambos, artefato e guia, são entregues ao inspetor. Após, o inspetor líder cria um *checklist* a partir de uma lista de critérios de qualidade para planos de implantação que o processo OpenUp provê. Alguns exemplos de questões que integram o *checklist* são: quando a implantação será completada e testada? Quais componentes estão incluídos neste pacote de entrega? Como o time saberá se o produto foi terminado e quando a implantação está finalizada?

Com os materiais de revisão em mãos, o inspetor líder seleciona as pessoas aptas para fazer parte do time de inspeção, designando um papel para cada uma. São as pessoas selecionadas: o gerente de implantação no papel de inspetor, o escrivão técnico no papel de escrivão e o testador no papel de leitor.

De acordo com a agenda de cada membro do time, o inspetor escolhe a data e local, seguindo o planejamento realizado pelo gerente do projeto, para a reunião e para as apresentações dos procedimentos de inspeção e do produto da inspeção. Por questões de logística, ambas as apresentações serão realizadas no mesmo dia e local. Após, o líder da inspeção escolhe um dia em que todos os inspetores entregarão a lista de anomalias encontradas para que assim possa organizar a reunião. Por fim, ele replica o artefato e os materiais de revisão para cada membro do time escolhidos para essa inspeção, para que os membros possam realizar a inspeção, encerrando-se a fase de planejamento. Como esta inspeção tem apenas um artefato, não são definidas partes prioritárias a serem revisadas.

No dia das apresentações os integrantes do time de inspeção são apresentados aos procedimentos de inspeção. As dúvidas referentes ao *checklist*, papéis e responsabilidades são esclarecidas. Em seguida, o autor explica as características do plano de implantação, descrevendo cada item do artefato. Enfim, as apresentações são finalizadas e se inicia a fase de preparação.

A partir desse momento, o time de inspeção realiza as inspeções no plano de implantação, buscando por inconsistências que o artefato possa conter. O inspetor encontra três anomalias, o escrivão, que também é um inspetor, localiza apenas uma e o leitor, também um inspetor, não encontra nenhuma. As listas de anomalias são repassadas para o inspetor líder, que as classifica de acordo com a gravidade. Das quatro anomalias encontradas, a anomalia informada pelo escrivão é igual a uma das anomalias encontradas pelo inspetor. Após, o leitor se prepara para a reunião, escolhendo a forma de leitura *top down*.

No dia da reunião, o time de inspeção, o autor e o inspetor líder se encontram no local reservado para a realização dessa fase. O inspetor líder faz alguns esclarecimentos sobre a inspeção e as responsabilidades dos membros. Em seguida, as anomalias encontradas na fase de preparação são registradas pelo escrivão. Enfim é iniciada a reunião,

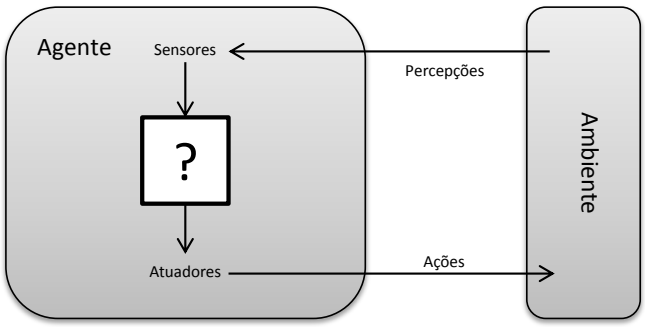
o leitor realiza a leitura do artefato, passando item por item do plano de implantação. Durante a reunião são encontrados somente erros ortográficos e gramaticais. Ao final da reunião, o inspetor líder verifica a lista de anomalias, e solicita a classificação do artefato pelo time de inspeção. Pelo nível de gravidade das três anomalias encontradas ser baixo, o time de inspeção resolve aceitar com correções o plano de implantação. Enfim, o inspetor líder é designado para verificar as correções dessas anomalias durante a fase de retrabalho / continuação, encerrando-se assim a reunião.

Na fase de retrabalho / continuação, o inspetor líder repassa para o autor as anomalias encontradas para que sejam corrigidas. O autor recebe as anomalias informadas e realiza a correção de cada uma. Após a correção, o autor informa o inspetor líder para que o mesmo possa verificar as anomalias do artefato. Como as três anomalias informadas foram corrigidas, o inspetor líder finaliza a inspeção.

2.2 Sistema Multiagente

Para compreender um sistema multiagente, é necessário saber o que é um agente de software. Ainda não há um consenso sobre a definição de agente de software. Segundo Wooldridge e Jennings (1995), um agente é um sistema computacional que está situado em algum ambiente, no qual é capaz de realizar ações autônomas neste ambiente para atingir seus objetivos. Uma definição mais simples feita por Russell e Norvig (2009) é que um agente é qualquer coisa que consegue perceber o ambiente através de sensores e atuar neste ambiente através de atuadores. Essa interação é mostrada na Figura 2.

Figura 2 – Agentes atuando com o ambiente através de sensores e atuadores.



Fonte: Russell e Norvig (2009, p. 35)

Pode-se destacar que uma característica comum às duas definições de agentes é a autonomia. Mas para um agente ser considerado inteligente, é necessário que ele possua mais propriedades. Wooldridge e Jennings (1995) sugere que são esperados que um agente inteligente possua:

- Reatividade: agentes inteligentes são capazes de perceber o seu ambiente e respondê-lo em um tempo hábil para modificar aquela ocorrência para satisfazer seus objetivos.

- Proatividade: agentes inteligentes são capazes de exibir comportamento direcionado à objetivos através da iniciativa, para assim satisfazer seus objetivos.

- Habilidade social: agentes inteligentes são capazes de interagir com outros agentes (e possivelmente humanos) para satisfazer seus objetivos.

Existe uma grande variedade de ambientes, porém se pode identificar poucas dimensões nos quais os ambientes podem ser categorizados. Essas dimensões determinam o projeto de agente apropriado e a aplicabilidade das técnicas principais de implementação de agentes. As dimensões são (RUSSELL; NORVIG, 2009):

- Completamente observável *versus* parcialmente observável: um ambiente é completamente observável quando um sensor de um agente consegue obter um estado completo de um ambiente em qualquer ponto no tempo. Entretanto, um ambiente é parcialmente observável quando há ruídos, sensores imprecisos ou porque partes do estado do ambiente foram perdidos do sensor.

- Agente único *versus* multiagentes: um ambiente é agente único quando há apenas um agente percebendo e interagindo com o ambiente de forma a atingir seus objetivos. Um ambiente multiagente se caracteriza pela existência de vários agentes com percepções e ações distintas, os quais interagem para atingir seus objetivos definidos.

- Determinístico *versus* estocástico: um ambiente é determinístico caso se o próximo estado do ambiente é determinado pelo estado atual e pela ação executada pelo agente. Caso não seja satisfeita esta condição, o ambiente é estocástico.

- Episódico *versus* sequencial: em um ambiente episódico, as experiências do agente são divididas em episódios atômicos. Em cada episódio o agente recebe uma percepção e executa uma única ação. O próximo episódio não depende das ações realizadas no episódio anterior. Um ambiente é sequencial caso as decisões atuais afetarem as ações futuras do agente.

- Estático *versus* dinâmico: se um ambiente pode mudar enquanto um agente está ponderando, então o ambiente é dinâmico para esse agente, caso contrário, ele é estático.

- Discreto *versus* contínuo: A diferença entre os ambientes discreto e contínuo se aplica no estado do ambiente, na maneira que o tempo é manipulado, e nas percepções e ações do agente. Um ambiente discreto tem um conjunto finito de estados, percepções e ações distintas. Porém um ambiente contínuo não é possível descrever todos os estados, percepções e ações envolvidas.

- Conhecido *versus* desconhecido: esta distinção refere-se ao estado de conhecimento do agente em relação ao ambiente. Um ambiente é conhecido para o agente quando todos os resultados de todas as ações são divulgados. No caso de um ambiente desconhecido, o agente precisa aprender como ele funciona para que possa decidir corretamente.

Segundo Russell e Norvig (2009), a estrutura de um agente é composta pela agregação de uma arquitetura com um programa agente. Uma arquitetura é descrita como um dispositivo computacional com sensores e atuadores. Um programa agente é a implementação das funcionalidades de um agente, ou seja, o mapeamento das percepções para ações. O mesmo autor descreve quatro tipos de programas agentes que possuem os princípios existentes na maioria dos sistemas inteligentes. São eles (RUSSELL; NORVIG, 2009):

- Agentes Reativos Simples: é o tipo de agente mais simples. Ele atua com base em sua percepção atual, não utilizando o resto do histórico de percepções.

- Agentes Reflexivos Baseados em Modelos: este tipo de agente armazena um estado interno que é depende do histórico de percepções, dessa forma, ele reflete pelo menos algum aspecto não observado do estado atual. Esse tipo de conhecimento sobre os possíveis estados é chamado de modelo do mundo.

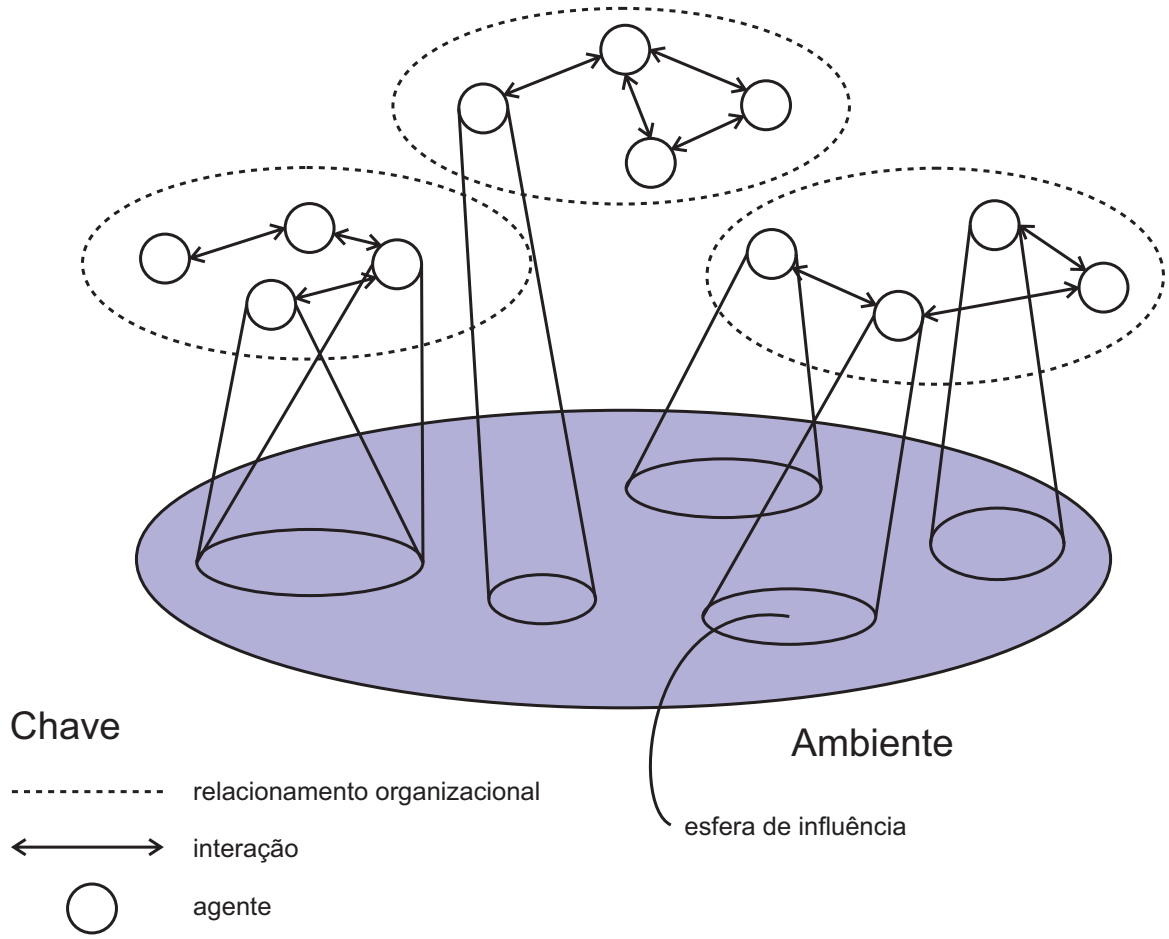
- Agentes Baseados em Objetivos: este agente possui um objetivo que descreve as situações que são desejáveis, pois nem sempre é possível decidir conhecendo os estados que ele pode atuar.

- Agentes Baseados em Utilidade: os objetivos sozinhos não são o suficiente para gerar um comportamento de alta qualidade em vários ambientes. Objetivos trazem apenas duas distinções de estados, que pode ser “feliz” ou “triste”. Com o uso de uma função de utilidade que compara os estados de acordo com o grau de “felicidade”, ele pode tomar suas decisões escolhendo sempre o estado que deixa o agente mais “feliz”, ou seja, o estado mais funcional.

Para Wooldridge (2002), um sistema multiagente consiste em um conjunto de agentes que interagem uns com os outros, cada um com seu objetivo e motivação para atingir suas metas.

A Figura 3 mostra a estrutura típica de um sistema multiagente. O sistema contém um número de agentes que interagem entre si através de uma comunicação. Cada agente pode atuar no ambiente, onde cada agente possui uma "esfera de influência" diferente, possibilitando o controle de diferentes partes do ambiente. Em certos casos, agente podem atuar sobre a mesma esfera de influência, o que pode trazer uma relação de dependência, como por exemplo dois robôs de limpeza, um encerando e outro varrendo a mesma sala (WOOLDRIDGE, 2002).

Figura 3 – Estrutura típica de um sistema multiagente.



Fonte: Wooldridge (2002, p. 106)

A *Foundation for Intelligent Physical Agents* (FIPA) é uma organização da IEEE *Computer Society Standarts* que promove a tecnologia baseada em agentes e a interoperabilidade de seus padrões com outras tecnologias (FIPA, 2013).

As especificações FIPA representam uma coleção de padrões com a intenção de promover a interoperação de agentes heterogêneos e serviços que eles podem representar

(FIPA, 2013).

Em 1995, a FIPA começou a trabalhar no desenvolvimento de padrões para sistemas de agentes. O resultado dessa iniciativa foi a *Agent Communication Language* (ACL). Essa linguagem permitiu representar crenças, desejos e incertezas dos agentes, assim como as ações que os agentes realizam (WOOLDRIDGE, 2002).

2.2.1 Engenharia de Agentes

A programação orientada a agentes é motivada pela necessidade de arquiteturas abertas em constante mudança para acomodar novos requisitos e componentes. Em sua abordagem é falado em estados mentais e crenças em vez de máquina de estados, planos e ações no lugar de procedimentos e métodos, em comunicação, negociação e habilidade social em vez de interação e funcionalidades de entrada e saída, além de objetivos, desejos, etc. Para a implementação dessas características que a programação orientada a agentes proporciona, pode-se utilizar da metodologia de desenvolvimento chamada Tropos (BRESCIANI et al., 2004).

Tropos é uma metodologia de engenharia de software orientada a agentes que cobre todo o processo de desenvolvimento do software. Seu propósito é a construção do sistema e seu ambiente, no qual é refinado e estendido incrementalmente (BRESCIANI et al., 2004).

A metodologia Tropos possui quatro fases de desenvolvimento:

- Requisitos Preliminares: é estudado o sistema como um todo, são identificadas as pessoas, relacionamentos, planos e objetivos que fazem parte do domínio. É possível estabelecer o *"por que"*, juntamente com o *"o que"* e *"como"*, das funcionalidades (BRESCIANI et al., 2004).

- Requisitos Finais: é descrito como o sistema deve ser. É adicionado outro ator que representa o sistema e relacionado com os outros atores identificados. As dependências encontradas definem os requisitos funcionais e não funcionais (BRESCIANI et al., 2004).

- Arquitetura do Projeto: o sistema é descrito em subsistemas interconectados através de fluxo de controle e de dados. Cada subsistema representa um ator, e as conexões representam as dependências. Essa fase pode ser realizada em três passos: definição da arquitetura, identificação das capacidades que os atores devem possuir para atingir seus objetivos, e a definição de um conjunto de agentes com uma ou mais capacidades (BRESCIANI et al., 2004).

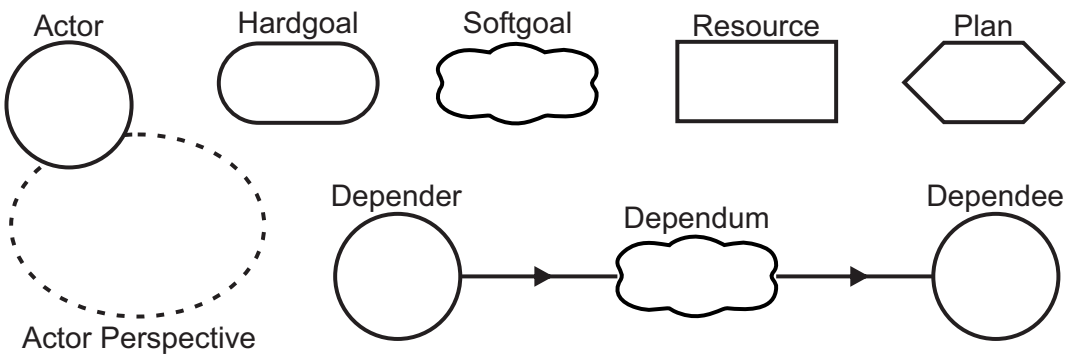
- Projeto Detalhado: é realizada a especificação detalhada de cada componente da arquitetura. São descritas em detalhes as capacidades, objetivos e crenças de cada

agente. Além disso, as interações entre os agentes são delineadas (BRESCIANI et al., 2004).

A Tropos utiliza a notação oferecida pelo *framework* de modelagem *i**, do trabalho de Eric Yu's. Esse *framework* propõe conceitos como ator, dependências sociais entre atores, incluindo dependência de objetivo, de tarefa e de recurso (BRESCIANI et al., 2004).

Ator é uma entidade que possui objetivos e intenções no sistema. Ele pode representar uma pessoa ou agente de software assim como um papel ou posição. Sua representação gráfica se dá através de um círculo, como mostrado na Figura 4 através do *Actor* (BRESCIANI et al., 2004). Por não haver uma tradução direta para *Depender*, *Dependum* e *Dependee* e para padronizar o idioma na Figura 4, foi mantido o idioma original.

Figura 4 – Conceitos básicos da modelagem Tropos.



Fonte: adaptado de Bresciani et al. (2004)

Cada ator possui uma perspectiva, no qual agrega os objetivos, planos, recursos e tarefas relacionados à ele. Os objetivos fora dessa perspectiva não são desejados pelo ator. Sua representação gráfica se dá por uma elipse tracejada, como apresentado na Figura 4 através do *Actor Perspective* (BRESCIANI et al., 2004).

O objetivo representa os interesses do ator. Os objetivos podem ser derivados em *"hardgoals"* e *"softgoals"*. Os *hardgoals* são descritos de forma que podem ser satisfeitos, assim como os requisitos funcionais. A representação gráfica dos *hardgoals* é feita através de uma forma oval. Já os *softgoals* são utilizados geralmente para modelar requisitos não funcionais. A Figura 4 apresenta sua representação gráfica através da forma de uma nuvem (BRESCIANI et al., 2004).

O recurso representa uma entidade física ou de informação, como um documento ou uma foto. Através da Figura 4 é mostrada a representação gráfica do recurso, no qual possui a forma de um retângulo através da entidade *Resource* (BRESCIANI et al., 2004).

O plano, ou tarefa, é uma abstração da maneira de realizar algo. A execução do plano pode satisfazer um *hardgoal* ou *softgoal*. Sua representação gráfica se dá através de um hexágono, como é demostrada na Figura 4 através da entidade *Plan* (BRESCIANI et al., 2004).

Dependência é a relação entre dois atores, no qual um depende do outro para a realização de algum objetivo, execução de um plano, ou a entrega de algum recurso. A representação gráfica se dá pela ligação entre dois atores através de duas linhas com flechas, conectadas por um símbolo gráfico, que pode ser um *hardgoal*, um *softgoal*, um plano, ou um recurso. O ator que depende é chamado *depender*, o ator que provê é chamado de *dependee*, e o símbolo gráfico que conecta os atores é chamado de *dependum*, assim como é apresentado na Figura 4 (BRESCIANI et al., 2004).

A capacidade é a habilidade do ator de definir, escolher e executar um plano para atingir um objetivo. Pode ser disparado quando existir uma condição ou evento específico (BRESCIANI et al., 2004).

Por fim, a crença representa o conhecimento que o ator possui do mundo (BRESCIANI et al., 2004).

Várias atividades de modelagem contribuem para a aquisição do modelo de requisitos preliminares, seu refinamento e sua evolução para modelos subsequentes. São elas a modelagem do ator, de dependência, de objetivos, do plano, e da capacidade (BRESCIANI et al., 2004).

Na modelagem do ator há a identificação e análise dos atores do ambiente e dos atores e agentes do sistema. Na fase de requisitos preliminares os colaboradores do domínio são modelados como atores, possuindo intenções e objetivos. Na fase de requisitos finais essa modelagem foca em quem e como serão os atores do sistema. Na fase de arquitetura do projeto são descritas as estruturas do ator através de subsistemas interconectados por dados e controle de fluxos. Na fase de projeto detalhado são modelados todos os agentes e suas descrições necessárias para a implementação (BRESCIANI et al., 2004).

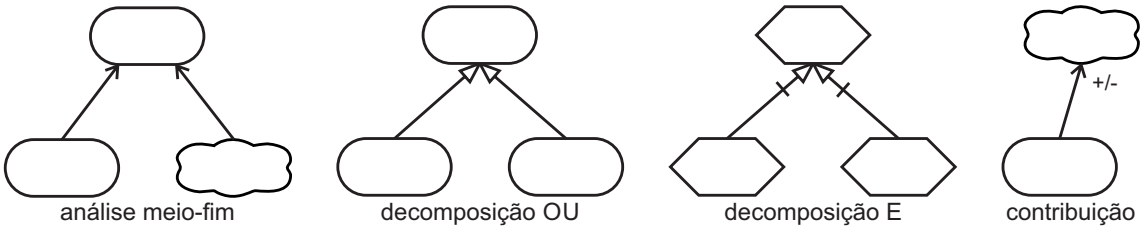
A modelagem de dependência foca na identificação dos atores que dependem de outros para atingir seus objetivos, realizar seus planos e obter recursos. Na fase de requisitos preliminares é realizada a modelagem das dependências dos objetivos entre os atores da organização. Na fase de requisitos finais há a análise das dependências dos atores do sistema. Na fase de arquitetura do projeto as dependências dos controles de fluxo e dados entre os "subatores" do sistema são modelados. Essa modelagem juntamente com o mapeamento dos agentes do sistema servirá de base para o modelo de capacidade (BRESCIANI et al., 2004).

Na modelagem de objetivos é realizada a análise dos objetivos do ator a partir do seu ponto de vista, usando três tipos de técnicas: análise meio-fim, análise de contribuição,

e decomposição E/OU (BRESCIANI et al., 2004).

A análise meio-fim busca identificar planos recursos e *softgoals* que proveem meios para atingir objetivos. A Figura 5 apresenta um *hardgoal* que pode ser atingido por um novo *hardgoal* e um *softgoal* (BRESCIANI et al., 2004).

Figura 5 – Relações da modelagem Tropos.



Fonte: adaptado de Bresciani et al. (2004)

A análise de contribuição identifica objetivos que contribuem de forma positiva e negativa para realização do objetivo. Sua representação gráfica é demonstrada na Figura 5, no qual um objetivo pode colaborar positivamente para a satisfação de um *softgoal* (BRESCIANI et al., 2004).

A decomposição E/OU reúne as decomposições de um objetivo em "subobjetivos". Dessa forma se pode encontrar novas dependências nas fases de requisitos preliminares e finais, e contribui na decomposição de atores do sistema em "subatores". É demostrada na Figura 5 as duas decomposições, no qual a decomposição OU deriva um *hardgoal* em dois novos *hardgoals*, e a decomposição E deriva um plano em dois novos planos. Na decomposição OU, o *hardgoal* inicial será satisfeito se qualquer um dos *hardgoals* derivados for satisfeito. Porém na decomposição E, o plano original só será cumprido caso os dois planos derivados forem realizados (BRESCIANI et al., 2004).

A modelagem do plano pode ser considerado uma técnica complementar de análise da modelagem de objetivos, no qual se baseia de forma equivalente às técnicas de análise da modelagem de objetivos (BRESCIANI et al., 2004).

Por fim, a modelagem da capacidade é iniciada no fim da fase de arquitetura do projeto quando os "subatores" já possuem seus objetivos e dependências com os outros atores do sistema. Cada "subator" do sistema deve fornecer uma capacidade individual para conseguir definir, escolher e executar o plano para atingir seu objetivo. Objetivos e planos modelados anteriormente se tornam parte integral das capacidades. Por fim, na fase de arquitetura detalhada as capacidades de cada agente são refinadas (BRESCIANI et al., 2004).

2.2.2 Plataforma JADE

O *framework Java Agent Development* (JADE) foi iniciado em 1998 pela Telecom Italia, como propósito de validar a recém-criada especificação FIPA. Em 2000, o JADE teve seu código-fonte aberto sob a licença *Library Gnu Public License* (LGPL). JADE é uma plataforma de desenvolvimento que provê uma camada *middleware* de funcionalidades básicas, que são independentes das especificação da aplicação e na qual simplifica o desenvolvimento de aplicações distribuídas que utilizam agentes de software. Sua última versão é a 4.3.1 e se encontra disponível no site oficial[3]. JADE foi implementada para prover aos programadores as seguintes funcionalidades (BELLIFEMINE; CAIRE; GREENWOOD, 2007):

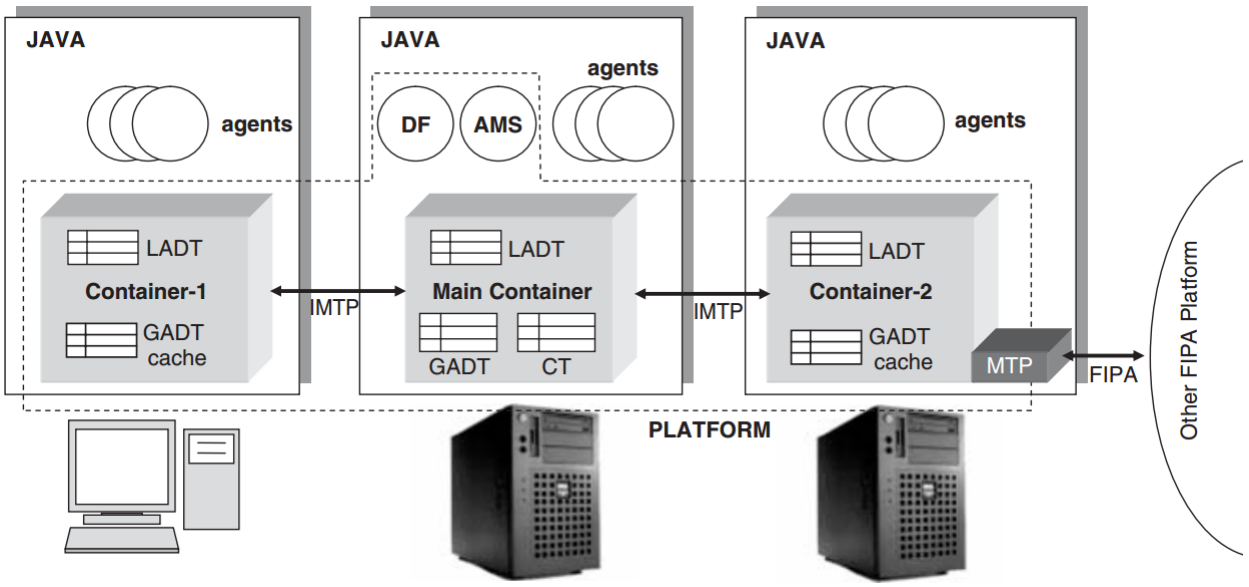
- um sistema completamente distribuído habitado por agentes, cada um executando uma *thread* separadamente, potencialmente em máquinas remotas, e capaz de se comunicar transparentemente;
- compatibilidade total com as especificações FIPA, participando com sucesso em todos os eventos de interoperabilidade;
- transporte eficiente de mensagens assíncronas por uma *Application Programming Interface* (API) transparente de localização, na qual seleciona o melhor meio disponível de comunicação;
- implementação de páginas brancas e páginas amarelas, onde sistemas podem ser implementados para representar domínios e subdomínios como um grafo de diretórios;
- gerenciamento do ciclo de vida do agente simples e efetivo através de ferramentas gráficas e APIs simples;
- suporte para mobilidade de agentes, possibilitando o agente ser movido de processo e de máquina de forma transparente;
- um mecanismo de inscrição de agentes que desejam de inscrever em uma plataforma para ser notificado de todos os eventos dessa plataforma;
- um conjunto de ferramentas gráficas para dar suporte no monitoramento e depuração;
- suporte para ontologias e linguagens de conteúdo, no qual a verificação e a codificação é realizada automaticamente, possibilitando que o desenvolvedor escolha as ontologias e linguagens de conteúdo de sua preferência;

[3] http://jade.tilab.com/

- uma biblioteca de protocolos de interação que modelam os padrões orientados à comunicação para atingir um ou mais objetivos;
- integração com várias tecnologias baseadas na *web*, incluindo *Java Server Pages* (JSP), *servlets*, *applets* e *Web services*;
- suporte para a plataforma *Java Platform, Micro Edition* (J2ME) e ambientes sem fio;
- uma interface de processos para iniciar e controlar a plataforma e seus componentes distribuídos através de aplicação externa;
- um *kernel* extensível projetado para permitir que programadores estendam sua funcionalidade através da adição de níveis de *kernel*.

A plataforma JADE é composta por contêineres que podem estar distribuídas pela rede, como é apresentado na Figura 6. Esses contêineres são processos Java disponibilizados pelo *run-time* JADE, que provê todos os serviços necessários para a hospedagem e execução dos agentes que ali vivem. O contêiner *main* representa o ponto de inicialização da plataforma. Ele é o primeiro a ser iniciado e todos os outros contêineres devem se registrar ao contêiner *main*. É possível que exista a interação entre a plataforma que JADE em execução com outras plataformas através da interface FIPA provida (BELLIFEMINE; CAIRE; GREENWOOD, 2007).

Figura 6 – Relação entre os elementos da arquitetura principal.



Fonte: Bellifemine, Caire e Greenwood (2007, p. 32)

JADE é utilizada amplamente nos últimos anos por muitas organizações acadêmicas e da industria. Podem ser encontradas formas de sua utilização desde tutoriais a protótipos industriais (BORDINI et al., 2006). É possível encontrar a documentação[4] da plataforma, exemplos de aplicação[5], artigos e apresentações, *links* relacionados[6], e um *Journal* somente com trabalhos sobre JADE.

2.3 Fechamento do Capítulo

Neste capítulo foram apresentados os conceitos necessários para o entendimento do trabalho realizado. Através da inspeção de software se pode realizar verificações dos produtos de trabalho gerados durante o processo de desenvolvimento de software. Porém, sua aplicação não é trivial, pois demanda a cooperação e comprometimento de várias pessoas durante o processo.

Agentes de software são uma boa alternativa de ferramenta para a construção de sistemas inteligentes, seguindo suas características de autonomia, pró-atividade e habilidade social. A modelagem e construção de agentes são fatores críticos para o sucesso de um sistema eficaz. Neste contexto a metodologia Tropos contribui para o projeto de sistemas multiagentes, através de uma forma sistemática desde a modelagem até a implementação. Para a sua aplicação é necessário o entendimento dos conceitos relacionados a crenças, comportamentos e incertezas, que podem confundir e dificultar a implementação da solução. Por fim, a plataforma JADE traz uma maneira rápida e prática para a construção de sistemas multiagentes, demandando bons conhecimentos sobre orientação a objetos do desenvolvedor.

[4] http://jade.tilab.com/doc/index.html
[5] http://jade.tilab.com/papers-examples.htm
[6] http://jade.tilab.com/papers-links.htm

3 Trabalhos Relacionados

Neste capítulo são apresentados os trabalhos relacionados com o propósito de tornar válido o tema exposto. Eles contribuem para dar uma pequena amostra de onde está sendo investido o esforço em inspeções de software. Na seção 3.1 é relatado o método de pesquisa utilizado para a busca e seleção dos trabalhos relacionados. A seção 3.2 mostra os artigos obtidos através do processo de revisão, de forma sumarizada e divididos em dois grupos. Na seção 3.3 é feita uma análise dos artigos, apontando suas características e contribuições para o presente trabalho. Por fim, na seção 3.4 é realizado um fechamento do capítulo.

3.1 Metodologia de Revisão

Para a realização da pesquisa de artigos relacionados foi utilizado um processo de revisão sistemática da literatura (SAMPAIO, 2007). Esse tipo de técnica auxilia na busca de publicações relevantes ao trabalho proposto. Foi definida uma pergunta para guiar a pesquisa: *Quais soluções existem para dar suporte às atividades de inspeções de software de forma a aumentar a produtividade da equipe?* Dessa pergunta, foram derivadas as palavras-chave: *software inspection tool, inspection tool, inspection optimization, software quality tool, quality assurance, software engineering, software inspection improvement, agent.* Foram selecionadas as palavras-chave em inglês porque se esperava que os trabalhos resultantes fossem mais significativos que utilizando os termos em outro idioma. A partir da ferramenta *Google Scholar* foram extraídos vários artigos, dos quais foram selecionados aqueles que possuíam no mínimo seis páginas, estivessem disponíveis para *download*, possuíssem resultados, fossem de até cinco anos, e contivessem soluções de engenharia. Em seguida, os artigos eram lidos e avaliados de acordo com a sua relevância para o presente trabalho. Por fim, os artigos selecionados foram sumarizados. Na próxima seção são apresentados os trabalhos encontrados e discutido sua importância para o presente trabalho.

3.2 Resultados da Revisão da Literatura

Foram encontrados oito trabalhos relacionados como resultado da pesquisa utilizando o processo de revisão sistemática da literatura. Eles foram divididos em soluções orientadas à automação do processo, e em soluções orientadas à melhoria do processo.

Na subseção 3.2.1 são descritos as soluções que atuam na automação do processo. Cada trabalho propõe uma forma de automatização de alguma fase, etapa ou atividade

de controle e garantia da qualidade, substituindo em parte a intervenção humana.

Na subseção 3.2.2 são relatadas as soluções que atuam na melhoria do processo. Cada trabalho apresenta um processo de inspeção de software modificado e adaptado a seu contexto. Todas as soluções são apoiadas por uma ferramenta de suporte a seu processo de inspeção personalizado.

3.2.1 Soluções Orientadas à Automação do Processo

O trabalho de Silva et al. (2013) apresenta um sistema de suporte às inspeções de garantia da qualidade de software, que busca a automação de atividades específicas em um processo de inspeção. O sistema utiliza uma ontologia que descreve o domínio das inspeções da garantia da qualidade. Além disso, ele possui um conjunto de três agentes do tipo reativo simples, que são acionados pela interface do sistema. Através de um experimento em uma fábrica de software, o sistema mostrou que é capaz de automatizar algumas atividades envolvidas na inspeção de garantia da qualidade, levando à uma maior cobertura de artefatos inspecionados e o custo para a utilização foi mínimo.

Lee e Wang (2009) propõem um sistema multiagente baseado em ontologias para a avaliação do *Capability Maturity Model Integration* (CMMI) que realiza a sumarização de relatórios sobre o *Process and Product Quality Assurance* (PPQA) do CMMI automaticamente. Ele se justifica pelo CMMI ser composto pelas melhores práticas que envolvem o desenvolvimento de software. Uma ontologia é construída baseada no domínio de conhecimento do CMMI na área de processo de garantia da qualidade do processo e produto. A ontologia está dividida em oito camadas, e três tipos de agentes. O agente de processamento de linguagem natural que é um etiquetador e filtro de termos. O agente de raciocínio ontológico obtém o resultado do agente anterior e relaciona as forças dos termos com os conceitos do CMMI. O agente de sumarização encontra conceitos, extrai e filtra os caminhos das sentenças, gera sentenças, e calcula a taxa de compressão.

Os mesmos autores do artigo anterior, no trabalho (WANG; LEE, 2008) é proposto um *web service* inteligente para a área de processo da PPQA do CMMI, que reporta relatórios de não conformidade e de controle de qualidade, e mantém registro das avaliações do PPQA e CMMI. Baseado na PPQA o *web service* possui uma interface de usuário, um serviço de banco de dados, um serviço de produtos de trabalho e processos de avaliação, um agente de raciocínio ontológico, um prestador de serviço de visão objetiva, e um serviço de notificação de mensagens. O sistema auxilia os gerentes quanto à aderência aos processos definidos.

É apresentado por Nödler, Neukirchen e Grabowski (2009) o *XQuery-based Analysis Framework* (XAF), um *framework* baseado em XML e XQuery para a automatização da análise de artefatos de software. Com um alto nível de expressividade na inserção

de regras de análise, o XAF consegue inspecionar produtos de software diferentes sem a necessidade de readequação das regras. Ele se aproveita da característica do XML em possuir independência de formatos, assim podendo mapear os artefatos em forma de XML. Os experimentos realizados mostraram que a ferramenta consegue a independência do objeto de análise, e a extensibilidade para novas classes e alvos de análise. Por fim, não foi necessário o aprendizado de nenhuma linguagem proprietária para sua utilização.

3.2.2 Soluções Orientadas à Melhoria do Processo

Mishra e Mishra (2012) apresentam um processo de inspeção de software para o desenvolvimento de software distribuído, além de uma ferramenta de suporte a esse processo. É motivado pela atual mudança nos times de desenvolvimento, nos quais podem se encontrar distribuídos em países e continentes diferentes. São esperados a melhora na eficiência do tempo de desenvolvimento, a possibilidade de ficar mais perto dos clientes, e ter flexibilidade no acesso à recursos maiores e menos onerosos. O modelo de processo é assíncrono e pode incluir artefatos recém gerados. A inspeção pode ser automatizada pela ferramenta possibilitando a obtenção de dados vistos anteriormente.

O objetivo do artigo de Lucia et al. (2011) é apresentar um processo de inspeção distribuído geograficamente, modificado do método de Fagan. Esse processo encoraja os membros da inspeção a realizar uma discussão assíncrona após a fase de preparação e antes da reunião. São utilizadas as ferramentas de inspeção WAIT e ADAMS em conjunto. Foi realizado um aprimoramento na ferramenta ADAMS para trabalhar com inspeções distribuídas, assim, todo o ciclo de vida do processo é coberto pelo software. O processo de inspeção possui sete fases, sendo quatro dela opcionais. Logo, uma fase só é utilizada caso seja necessário para o artefato analisado.

O trabalho proposto por Mishra e Mishra (2009) é a criação de um processo de inspeção de software mais simples de ser utilizado. Assim, empresas que não dispõem de muitos empregados podem adotar esse processo. A inspeção simplificada se dá de forma parecida com a tradicional, com a diferença de que na fase de preparação os inspetores devem informar em um software *web* o resultado da inspeção. Assim, o autor insere seu comentário sobre essa inspeção realizada. A reunião de inspeção apenas ocorre para se verificar os artefatos inspecionados, pois a discussão já aconteceu no software de gerenciamento, sem que fosse necessária uma reunião para isso. Comparando esse processo com o padrão da IEEE e o da NASA, a inspeção simplificada é mais rápida de ser executada.

Liu et al. (2012) apresentam um método de inspeção sistemático e rigoroso, no qual é suportado por uma ferramenta que utiliza especificações formais. O objetivo é determinar se os caminhos dos cenários funcionais derivados da especificação são corretamente implementados pelos caminhos no software, e se qualquer caminho do software contribui para a especificação. O método proposto foi comparado com o método *perspective-based*

reading (PBR). Os resultados mostraram que o método abordado é mais efetivo que o método PBR em encontrar defeitos relacionados à função, mas menos efetivo para encontrar defeitos relacionados à implementação.

3.3 Análise dos Resultados

Os trabalhos (SILVA et al., 2013), (LEE; WANG, 2009), (WANG; LEE, 2008) e (NöDLER; NEUKIRCHEN; GRABOWSKI, 2009) contribuem na forma de automatizar fases, etapas ou procedimentos de controle e garantia da qualidade. Exceto no trabalho de (NöDLER; NEUKIRCHEN; GRABOWSKI, 2009), em todos os trabalhos a ferramenta desenvolvida utiliza múltiplos agentes, o que reforça a sua utilização de para a automatização de atividades que envolvem o controle e a garantia da qualidade. Enquanto o artigo em (NöDLER; NEUKIRCHEN; GRABOWSKI, 2009) apresenta uma ferramenta de automatização da busca de falhas em artefatos de software, o que pode auxiliar e substituir em algumas partes o trabalho dos inspetores. Essa abordagem traz uma solução para a produtividade das inspeções de software. Porém, ela não oferece um suporte ideal para realizar o gerenciamento do processo de inspeção de software como um todo.

Em (MISHRA; MISHRA, 2012), (LUCIA et al., 2011), (MISHRA; MISHRA, 2009) e (LIU et al., 2012) são propostos novos processos de inspeção de software. Todos os trabalhos são apoiados por uma ferramenta de suporte à essa nova abordagem. Os trabalhos demonstram que a adaptação do processo de inspeção de software e o suporte desse processo com um sistema de apoio melhoram a qualidade e a produtividade das inspeções de software. Além disso, em alguns trabalhos foi possível a realização da inspeção em times de desenvolvimento distribuídos. Apesar das ferramentas oferecerem suporte e maior produtividade às inspeções, nenhuma delas consegue ser pró-ativa ao ambiente, fornecendo apenas uma interface de visualização para o que está acontecendo. Além disso, nenhuma abordagem sugere a utilização de agentes como tecnologia a ser utilizada para o suporte ao processo.

O presente trabalho possui relações com os trabalhos desenvolvidos por Silva et al. (2013), Lee e Wang (2009), Wang e Lee (2008) e Nödler, Neukirchen e Grabowski (2009), pois todos tem por finalidade automatizar atividades do processo de inspeção de software. Complementarmente, em Silva et al. (2013), Lee e Wang (2009) e Wang e Lee (2008) são utilizados sistemas multiagentes para essa automatização, igualmente como é proposto pelo presente trabalho. Os trabalhos realizados por Mishra e Mishra (2012), Lucia et al. (2011), Mishra e Mishra (2009) e Liu et al. (2012) comparam-se com o presente trabalho através do suporte à inspeção de software utilizando um sistema computacional. Entretanto, o trabalho em questão não possui como objetivo a criação de um novo processo de inspeção de software.

3.4 Fechamento do Capítulo

Neste capítulo foi apresentado o uso da revisão sistemática da literatura que possibilitou alcançar uma maior cobertura na busca de trabalhos relacionados relevantes para o tema exposto. Verificou-se que existem vários trabalhos que abordam o tema de inspeção de software, e por intermédio deles é possível identificar que há mais de uma maneira de se possibilitar uma maior produtividade na inspeção de software.

Através da análise realizada no trabalho, foi possível verificar que a automatização de atividades vem sendo resolvida através da utilização de sistemas multiagentes. Isso é um indício que agentes de software são uma boa solução computacional para a realização de processos no lugar de pessoas e em apoio à essas pessoas. Quanto a melhoria do processo de inspeção, é possível afirmar que mesmo havendo uma modelagem do processo de inspeção aplicado à um contexto, o processo ainda necessita ser apoiado por alguma ferramenta computacional para obter um maior desempenho. Finalmente, o conhecimento obtido por essa revisão contribuiu para a validação dos objetivos do presente trabalho, onde pode ser verificado que a produtividade e a qualidade das inspeções de software estão em foco.

4 Sistema Multiagente para Suporte à Inspeção de Software

Neste capítulo é apresentada uma ferramenta para auxiliar o processo de inspeção de software, através da automação de algumas atividades e dessa maneira possibilitar o aumento da produtividade nas inspeções. Na seção 4.1 é relatado contexto atual das inspeções de software e uma visão geral da solução de software, através de sua arquitetura e funcionalidades. Na seção 4.2 são relatadas as fase da metodologia Tropos utilizadas para a implementação do sistema multiagente. A seção 4.3 descreve como cada componente do módulo multiagente foi construído. Na seção 4.4 mostra como é a interação do sistema multiagente com o ambiente e com a aplicação *web*. Por fim, na seção 4.5 é realizado o fechamento do capítulo.

4.1 Visão Geral

A inspeção de software ainda enfrenta problemas para ser amplamente adotada. Podem ser citadas como motivos a complexidade do processo de inspeção, a falta de aderência da inspeção ao processo de desenvolvimento de software e a baixa produtividade na sua aplicação. Além disso, há um esforço adicional na coleta e distribuição dos materiais referentes à inspeção em andamento.

Para minimizar os problemas e prover suporte ao processo de inspeção de software, foi criado um projeto de pesquisa vinculado ao LESA. O projeto de pesquisa possui uma proposta de um sistema de suporte às inspeções de software para apoiar e gerenciar os materiais relacionados à inspeção. Assim, minimizando o impacto da aplicação de uma inspeção de software.

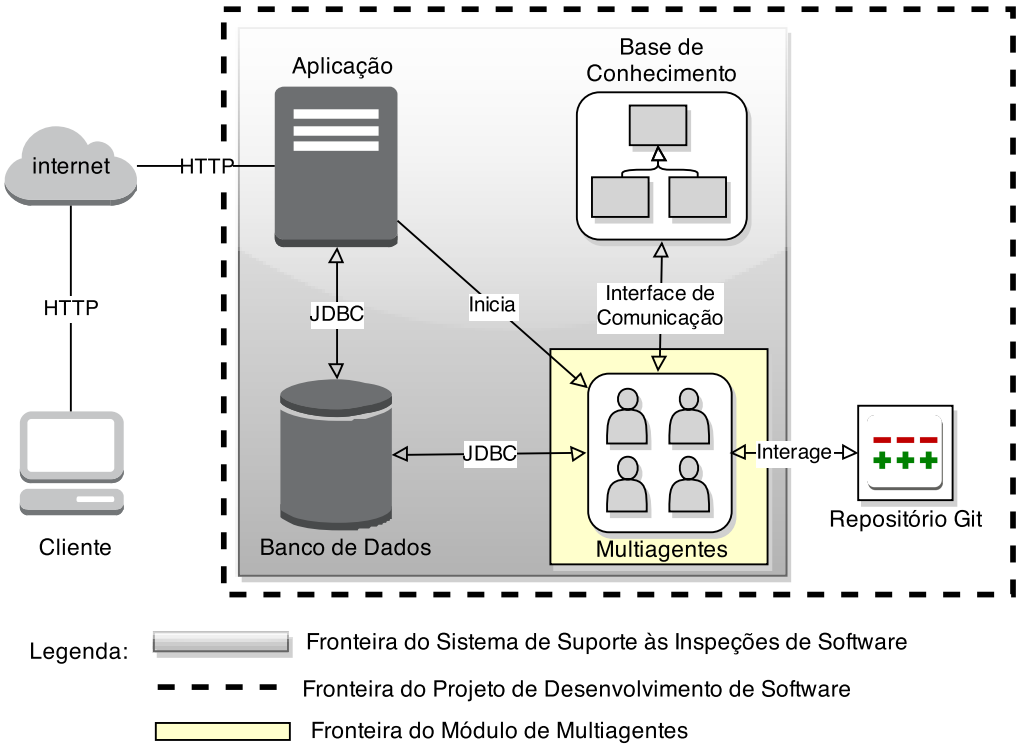
O presente trabalho traz como solução um sistema multiagente que realiza atividades do processo de inspeção de software. Essa solução faz parte do sistema de suporte às inspeções de software do projeto de pesquisa, em que contribui através da redução de atividades realizadas pelas pessoas envolvidas no processo e possibilita uma maior produtividade.

4.1.1 Arquitetura da Solução

Conforme apresentado na Figura 7 o sistema de suporte às inspeções de software é composto por uma base de conhecimento, um módulo de aplicação *web* e um banco de dados. O módulo multiagente interage com a base de conhecimento, na qual representa

computacionalmente o conhecimento necessário para auxiliar a tomada de decisões durante o processo de inspeção. Os agentes também interagem com uma base de dados, em que são registrados os dados relevantes ao processo. Os dados podem ser apresentados posteriormente pela aplicação *web* que interage com o usuário, relatando o andamento do processo de inspeção, assim como o registro de artefatos e anomalias. Por fim, no momento que a aplicação *web* é inicializada, o módulo multiagente também é iniciado para que ambos possam ser utilizados pelo processo de inspeção em andamento.

Figura 7 – Fronteiras da aplicação.



O módulo multiagente foi construído partindo de algumas premissas que devem estar presentes no projeto de desenvolvimento de software em andamento, como pode ser visualizado na Figura 7. Considera-se que o projeto possui um repositório sob o sistema de controle de versão Git[1], contendo todos os arquivos do projeto. Esse repositório possui um ramo específico para o envio dos artefatos prontos para a inspeção. O mesmo ramo recebe os envios dos artefatos que são corrigidos por seus autores. Além disso, a base de conhecimento deve representar todo o conhecimento do processo de desenvolvimento em andamento e seguir as restrições do projeto.

É também pressuposto que exista uma aplicação *web* para que os membros do time de inspeção possam interagir com os dados da própria inspeção, como é demonstrado na

[1] Git é um sistema de controle de versões distribuído feito para manipular desde projetos pequenos até grandes com velocidade e eficiência. http://git-scm.com/.

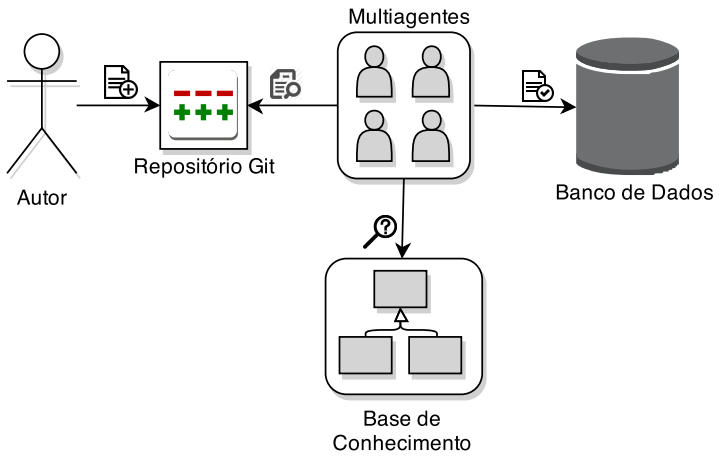
Figura 7. E além disso, deve haver uma base de dados para que os dados possam ser registrados e obtidos.

A estrutura do sistema multiagente é composta por cinco agentes com responsabilidades distintas, estruturalmente definida como reativo simples. Apesar disso, o módulo multiagente atende à todas as outras propriedades de sistemas multiagentes inteligentes: é *autônomo*, quando consulta por dados no ambiente e na base de conhecimento; *pró-ativo*, pois identifica, reúne e distribui os materiais de inspeção; e é *sociável*, pois se relaciona com outros agentes para a obtenção de dados referentes à inspeção.

O sistema multiagente possui cinco funcionalidades principais: enviar o artefato para a inspeção, reunir os materiais de inspeção, distribuir os materiais de inspeção, possibilitar o inspetor analisar o artefato, e notificar o inspetor líder das correções das anomalias.

Na Figura 8 é demonstrada a funcionalidade de enviar o artefato para a inspeção. Ao finalizar o artefato, o autor realiza o envio do artefato no ramo das inspeções. Após, o sistema multiagente detecta que esse artefato foi adicionado e consulta a base de conhecimento para descobrir qual o tipo desse artefato. Por fim, o artefato é inserido no banco de dados para que fique disponível para a inspeção.

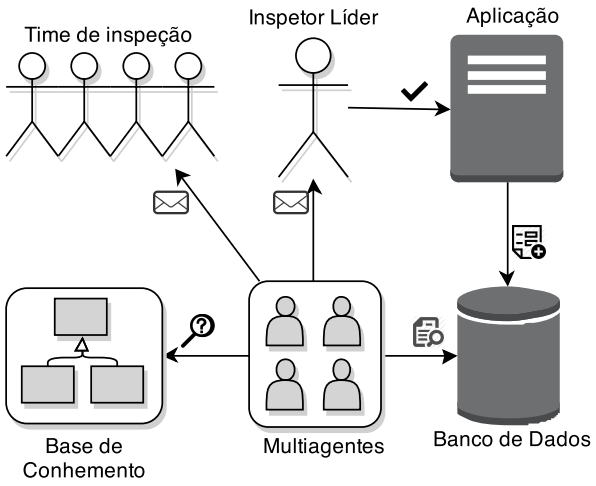
Figura 8 – Funcionalidade de enviar o artefato para a inspeção.



A Figura 9 apresenta três funcionalidades: reunir os materiais de inspeção, distribuir os materiais de inspeção e possibilitar o inspetor analisar o artefato. O inspetor líder acessa a aplicação *web*, seleciona os artefatos disponíveis para a inspeção, seleciona os membros do time de inspeção e autoriza a inspeção. Em seguida, o sistema multiagente verifica a adição de uma nova inspeção, e reúne os membros do time e os artefatos selecionados pelo inspetor líder.

Conseguinte, o módulo multiagente consulta a base de conhecimento para a obter

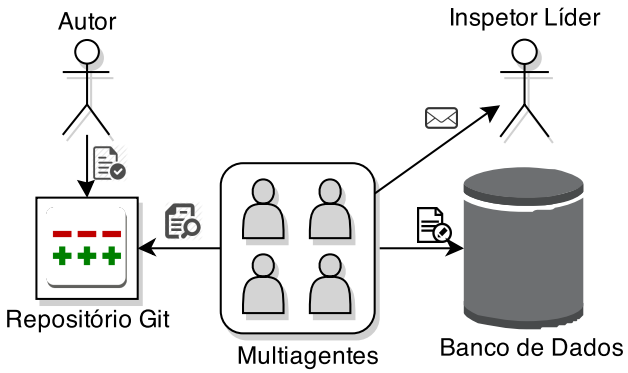
Figura 9 – Funcionalidades de reunir e distribuir os materiais de inspeção e possibilitar o inspetor analisar o artefato.



os itens de revisão que pertencem àquele artefato. Por fim, o módulo envia um *e-mail* para cada membro do time de inspeção informando sobre a nova inspeção, e fornecendo um *link* para a página que se encontra o *checklist* de cada membro. Dessa forma, possibilitando o inspetor analisar o artefato de software, como demonstrado na Figura 9.

A última funcionalidade se trata da verificação das correções das anomalias, como pode ser visualizada na Figura 10. O autor corrige a anomalia encontrada pelos inspetores e realiza o envio dessas correções para o repositório Git. O autor pode informar qual o estado da anomalia que ele está corrigindo na mensagem de envio, que pode ser "Novo", "Em Andamento", "Resolvido" e "Fechado". Após, o módulo multiagente verifica as mensagens de envio realizadas e encontra a modificação do estado da anomalia. Em seguida ele encontra e modifica o estado da anomalia de acordo com a mensagem enviada pelo autor, salvando o novo estado da anomalia e enviando um *e-mail* para o inspetor líder.

Figura 10 – Funcionalidade de verificar as correções das anomalias.



As funcionalidades descritas pelo módulo multiagente foram selecionadas a partir da análise do ambiente do processo de inspeção de software. Para isso, foi utilizada a metodologia de engenharia de agentes Tropos. Na seção de Análise (seção 4.2) é apresentado como ocorreu o processo de desenvolvimento do sistema multiagente.

4.2 Análise

Para realizar a automatização das atividades específicas da inspeção de software, foi necessário definir quais atividades seriam automatizadas. Uma maneira de realizar essa definição é através da modelagem dos objetivos e intenções que envolvem o processo de inspeção de software. Este trabalho utilizou a metodologia de engenharia de agentes Tropos para abstrair o conhecimento envolvido no processo de inspeção. Todos os quatro passos da metodologia Tropos foram realizados, os quais direcionaram o desenvolvimento desde sua concepção até os projeto dos agentes de software.

4.2.1 Requisitos Preliminares

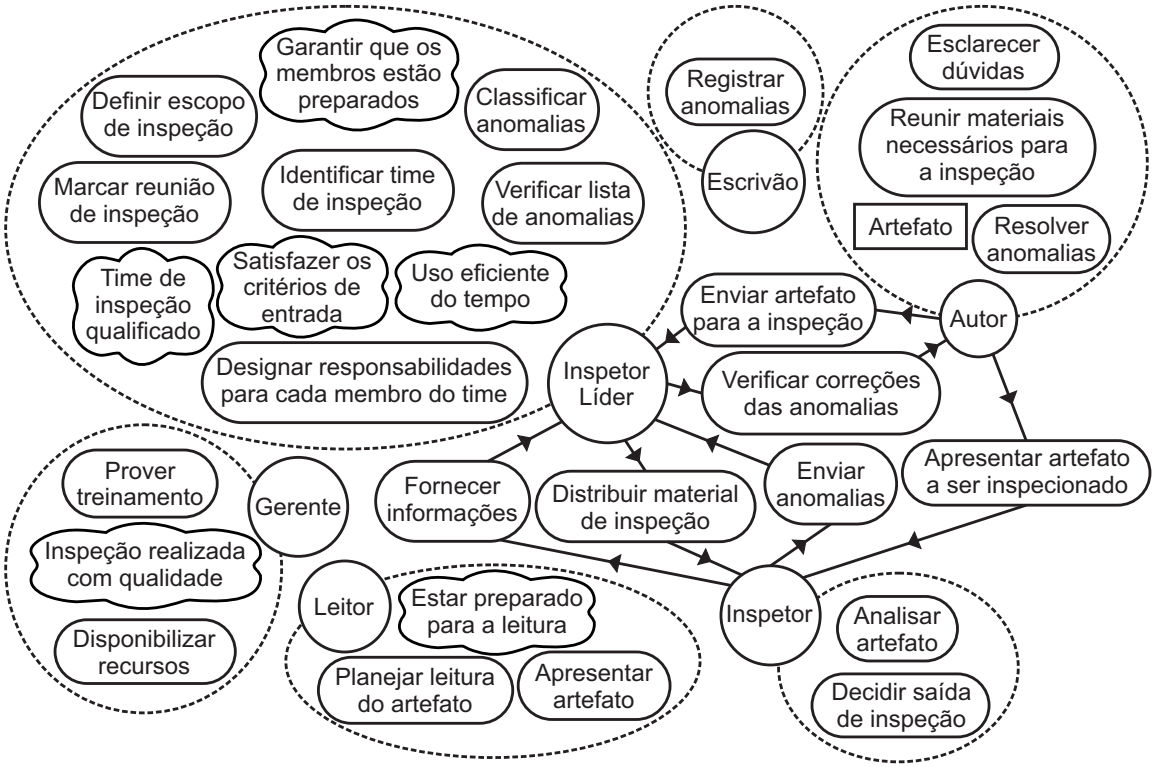
Na fase de requisitos preliminares, foi gerado um modelo de dependência, no qual destaca os atores envolvidos no processo de inspeção de software, assim como seus objetivos e dependências, como é apresentado na Figura 11. São evidenciados seis atores do domínio: inspetor líder, autor, inspetor, leitor, escrivão e gerente. Cada um deles possui um conjunto de objetivos, que é constituído de *hardgoals* e *softgoals*, e um conjunto de dependências com outros atores.

A Figura 11 apresenta o inspetor líder, no qual foram elicitados alguns *hardgoals* e *softgoals*. Como se pode observar, o inspetor líder possui responsabilidades durante todo o processo de inspeção de software, desde a identificação do time de inspeção até a verificação das correções das anomalias. Além desses objetivos, o inspetor líder busca manter a inspeção de forma produtiva, através do uso eficiente do tempo, da seleção de pessoas qualificadas para fazer parte do time de inspeção, e da satisfação dos critérios de entrada.

Pode-se evidenciar na Figura 11 a relação de dependência entre o inspetor líder e os atores inspetor e autor. Nesse caso o inspetor líder depende do autor para a realização das verificações das anomalias, e do inspetor para a distribuição do material de inspeção. O autor necessita do inspetor líder para enviar o artefato para a inspeção, e do inspetor para apresentar o artefato a ser inspecionado. Da mesma forma, o inspetor precisa do inspetor líder para a solicitação de informações e o envio das anomalias.

O autor apresentado na Figura 11 possui alguns objetivos específicos, principalmente em relação ao artefato. Ele realiza a coleta dos materiais necessários para a inspeção, enviando para o inspetor líder, e esclarecendo as dúvidas relacionadas ao artefato.

Figura 11 – Modelo de dependências entre atores e seus objetivos.



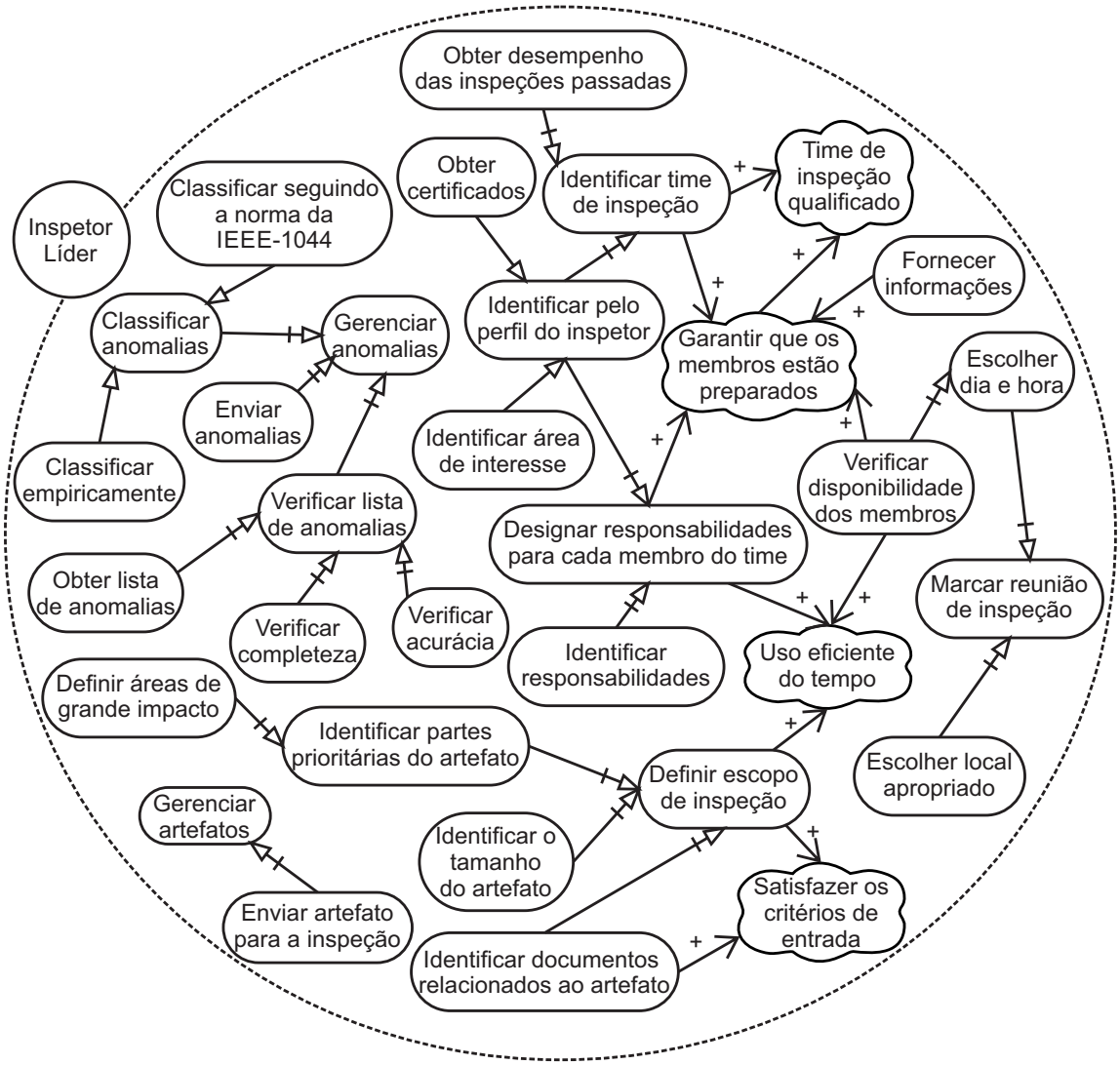
O inspetor enfatiza a análise do artefato, buscando informações e enviando as anomalias para o inspetor líder. Além disso, o leitor tem objetivos direcionados à apresentação do artefato durante a reunião de inspeção. O gerente por sua vez, possui metas quanto a qualidade da inspeção a ser realizada. Por fim, o escrivão tem o objetivo de registrar as anomalias encontradas, como demonstrado na Figura 11.

Após a identificação dos atores e suas dependências, é expandida a perspectiva de cada ator, no qual os objetivos são decompostos em "subobjetivos". A Figura 12 apresenta a perspetiva expandida do ator inspetor líder. Além das decomposições de *hardgoals* em novos *hardgoals*, também há a identificação de novos *softgoals*.

Pode-se observar na Figura 12 que o inspetor líder possui a maior quantidade de objetivos no processo de inspeção de software. Ele deve realizar as atividades relativas à identificação dos membros do time de inspeção, gerenciamento das anomalias, gerenciamento dos artefatos, definição do escopo e o agendamento do dia da inspeção.

A identificação dos membros e a designação das responsabilidades contribuem positivamente para a garantia de que os membros estão preparados, afinal os objetivos do inspetor líder passam pela pesquisa dados relacionados aos inspetores, tanto de desempenho quanto de interesse na área. A divisão das responsabilidades entre os membros faz com que aumente a velocidade da inspeção, aumentando assim o uso eficiente do tempo.

Figura 12 – Perspectiva expandida do inspetor líder.



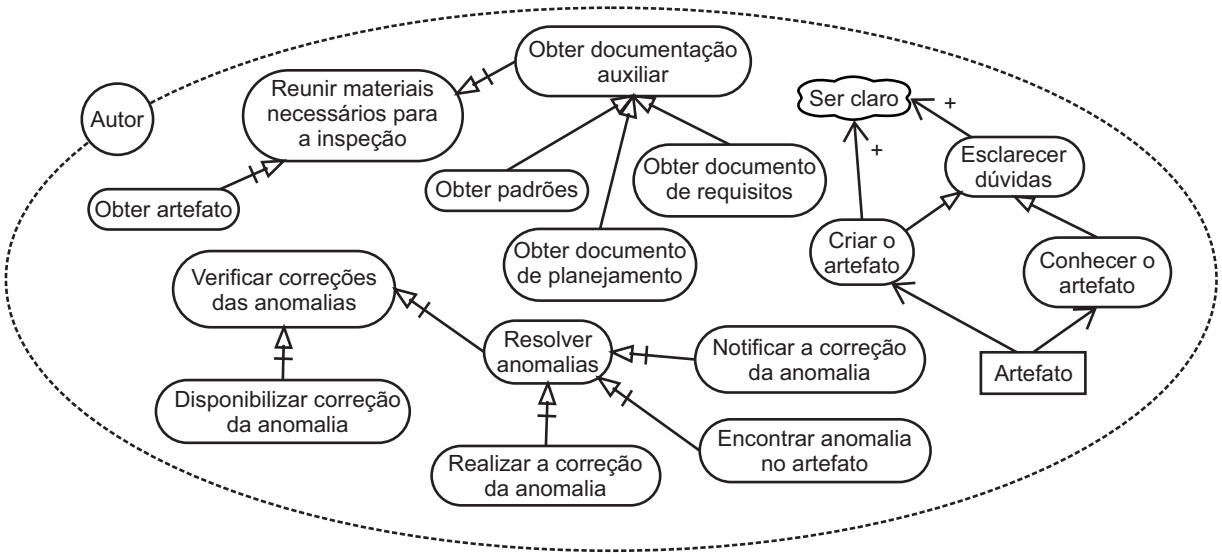
A identificação dos materiais e do escopo que serão avaliados contribuem para que os critérios de entrada do processo sejam satisfeitos, evitando artefatos incompletos que resultam em perca de tempo. Além disso, o gerenciamento dos artefatos e das anomalias devem ser realizadas para que o processo de inspeção ocorra normalmente.

Alguns objetivos são derivados do gerenciamento das anomalias. O inspetor líder deve verificar a lista de anomalias e classificá-las corretamente, seja de forma empírica ou seguindo algum padrão estabelecido. Além disso, ele deve fornecer uma maneira que os inspetores possam lhe enviar a lista de anomalias. Da mesma forma, o inspetor deve fornecer um mecanismo para que o autor possa enviar o artefato para a inspeção, como é relatado na Figura 12.

A Figura 13 apresenta a perspectiva expandida do autor. Seus objetivos são decompostos em algumas metas que ele deve realizar para cumprir o seu objetivo derivado.

No caso dos materiais é necessário que ele obtenha os documentos relacionados à criação e alinhamento do artefato durante seu desenvolvimento. Além disso, ele precisa ser claro para esclarecer as dúvidas que os inspetores possuirão. Uma forma é ter sido o criador do artefato, outra é conhecer profundamente ele.

Figura 13 – Perspectiva expandida do autor.

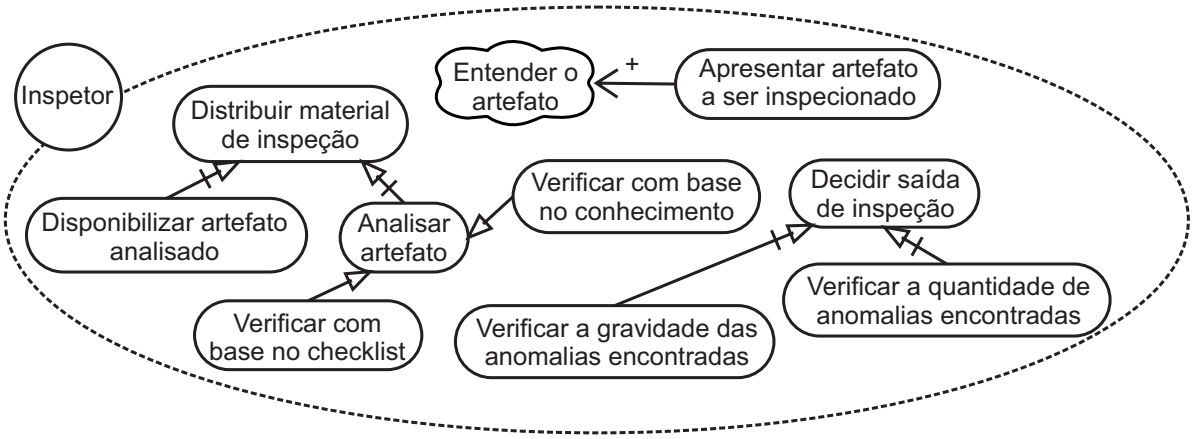


Uma meta de destaque é a resolução das anomalias encontradas durante a inspeção. Para isso é necessário atingir os objetivos de encontrar a anomalia apontada, realizar a correção e notificar o inspetor líder dessa correção. Pode-se notar que o autor deve disponibilizar de alguma maneira a correção das anomalias, para que o inspetor líder possa realizar a verificação de que a anomalia foi solucionada. Além disso, tanto os *hardgoals Esclarecer dúvidas* e *Criar o artefato* impactam positivamente no *softgoal Ser claro.*

Pode ser observada na Figura 14 a perspectiva expandida do inspetor. Ele tem o principal objetivo de analisar o artefato, seja utilizando um *checklist* ou analisando através de seu próprio conhecimento. Após essa análise, o inspetor necessita disponibilizar as anomalias encontradas para o inspetor líder. Através da apresentação do autor, o inspetor pode entender melhor sobre o artefato para poder inspecioná-lo. E após a reunião, os inspetores decidem se o artefato será aceito ou reinspecionado, que pode ser através da quantidade ou da gravidade das anomalias encontradas.

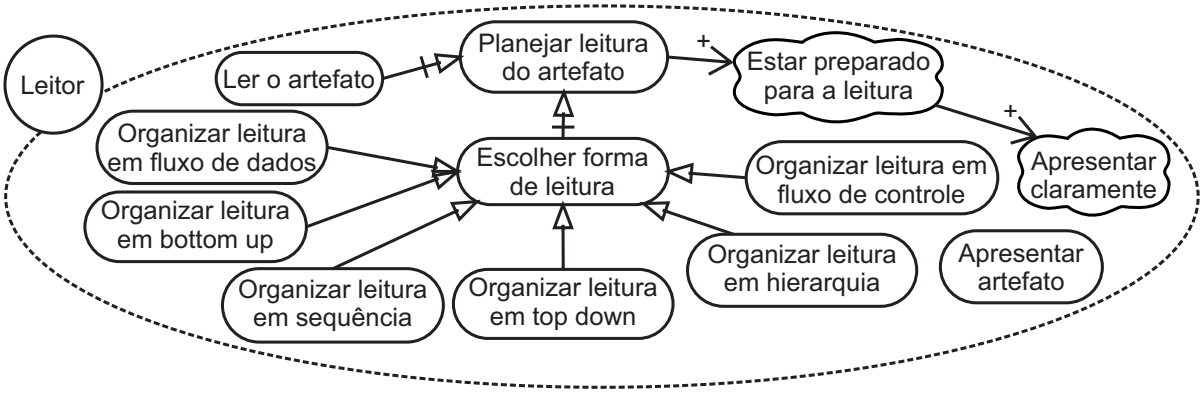
A Figura 15 apresenta a perspectiva do leitor. Sua principal meta é apresentar o artefato durante a reunião de inspeção. Porém, isso só é possível caso ele tenha realizado um planejamento sobre o artefato. A forma de leitura do material na reunião pode contribuir para que os inspetores possam encontrar novas anomalias. Além disso,

Figura 14 – Perspectiva expandida do inspetor.



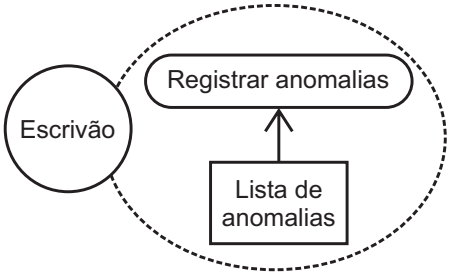
o planejamento da leitura contribui para que o leitor esteja preparado, levando à uma apresentação clara.

Figura 15 – Perspectiva expandida do leitor.



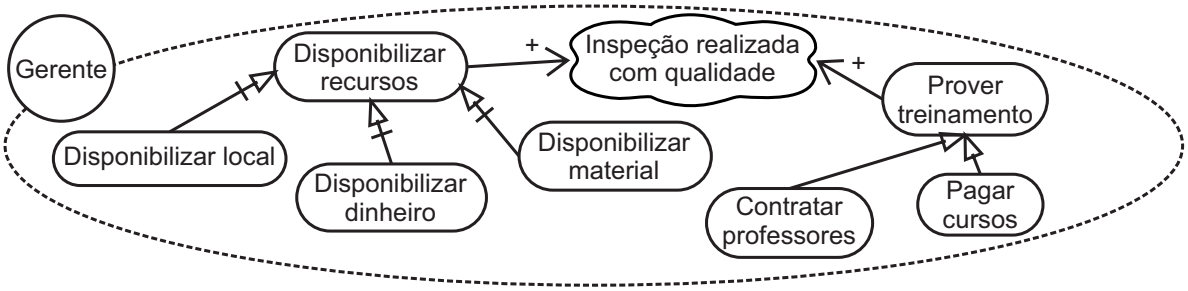
É mostrada através da Figura 16 o principal objetivo do escrivão, que é o registro das anomalias, resultando na lista de anomalias.

Figura 16 – Perspectiva expandida do escrivão.



Enfim, o gerente deve se preocupar com a qualidade e a eficiência do processo inspeção de software, como á apresentado na Figura 17. Para ajudar na qualidade, ele pode prover o treinamento através de cursos especializados, ou mesmo de professores ou consultores. Por fim, a disponibilização de material, local e dinheiro contribui de forma essencial à realização de uma inspeção com qualidade.

Figura 17 – Perspectiva expandida do gerente.



4.2.2 Requisitos Finais

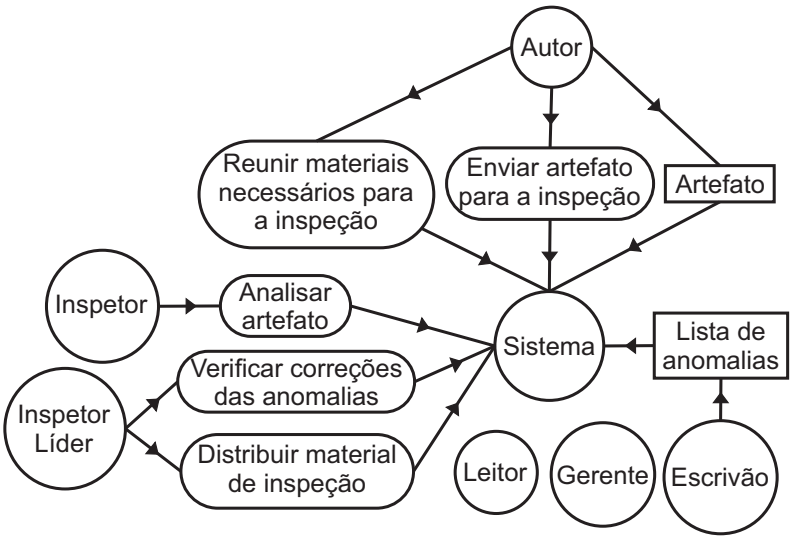
Na segunda fase do processo Tropos é adicionado um novo ator chamado "sistema", o qual se relaciona com os outros atores através de dependências. Os atores ligados ao ator sistema dependem dele para a satisfação de seus objetivos. Essas dependências definem os requisitos funcionais e não funcionais do sistema.

Do conjunto total de objetivos identificados na fase Requisitos Preliminares (subseção 4.2.1), foram escolhidos cinco para ser implementados pelo sistema multiagente. São eles: *Enviar artefato para a inspeção*, *Reunir materiais necessários para a inspeção*, *Distribuir material de inspeção*, *Analisar artefato* e *Verificar correções das anomalias*. Esses objetivos foram escolhidos por fazerem parte do fluxo principal do processo de inspeção. Com isso, as atividades manuais desempenhadas durante a inspeção serão reduzidas e assim contribuindo para o aumento da produtividade.

A Figura 18 apresenta o modelo de dependência entre os atores do processo de inspeção de software e o sistema. Os atores inspetor, inspetor líder e autor possuem um conjunto de objetivos que dependem do sistema. Pode-se notar que os atores gerente e leitor não possuem nenhum objetivo relacionado. Isso acontece pois a participação do gerente no processo somente se dá pela preparação da inspeção através de recursos e treinamento, os quais são independentes do sistema. E quanto ao leitor, somente é envolvido para a visualização e apresentação do artefato, que, da mesma forma que o gerente, são atividades independentes do sistema.

Os objetivos do inspetor líder que dependem do sistema são:

Figura 18 – Modelo de dependência dos atores com o sistema.



- *Verificar correções das anomalias*: o sistema informa as anomalias corrigidas pelo autor para que o inspetor líder avalie o andamento das correções. Dessa forma, o autor não precisa informar o inspetor líder diretamente sobre o andamento das correções das anomalias. E da mesma maneira, o inspetor líder não precisa verificar diretamente com o autor sobre as correções.

- *Distribuir material da inspeção*: o sistema envia o material da inspeção para que os inspetores inspecionem o artefato em busca de anomalias. Assim, o inspetor líder não precisa criar uma cópia do material para passar para cada membro do time. Sendo o sistema que realiza a busca dos itens de revisão do artefato, e os envia para cada membro.

O objetivo do inspetor que depende do sistema é:

- *Analisar artefato*: o sistema informa o inspetor sobre a inspeção, para que o mesmo analise o artefato e possa informar as anomalias encontradas. Assim, o inspetor é informado quando uma nova inspeção é criada, já repassando os itens de revisão para que possa verificar em busca de anomalias.

Os objetivos do autor que dependem do sistema são:

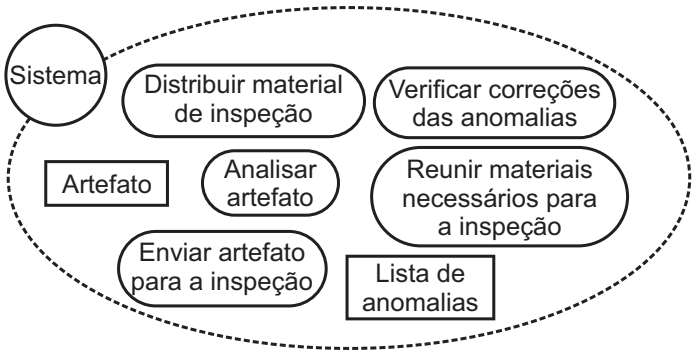
- *Enviar artefato para a inspeção*: o sistema permite que o autor envie o artefato a ser inspecionado pelo processo de inspeção de software. Dessa maneira, o autor não precisa parar a sua atividade atual para obter o artefato e entregá-lo para o inspetor líder.

- *Reunir materiais necessários para a inspeção*: o sistema busca os itens de revisão relacionados ao artefato para que posteriormente o próprio sistema possa enviá-los para os inspetores. Poupando assim o trabalho do autor de encontrar os materiais referentes ao artefato para entregar ao inspetor líder.

Os recursos *Artefato* e *Lista de anomalias*, do autor e escrivão respectivamente, são gerenciados pelo sistema. Dessa forma, há uma relação de dependência entre seus atores e o sistema.

Por fim, é apresentada a perspectiva do sistema através da Figura 19. Neste caso, os objetivos não estão relacionados por decomposição, dessa forma não há ligação entre eles.

Figura 19 – Perspectiva expandida do sistema.



4.2.3 Arquitetura do Projeto

Nesta fase foi realizado o refinamento da perspectiva do sistema, evidenciando-se os "subatores"[2] do ator Sistema e conectando-os através de suas dependências.

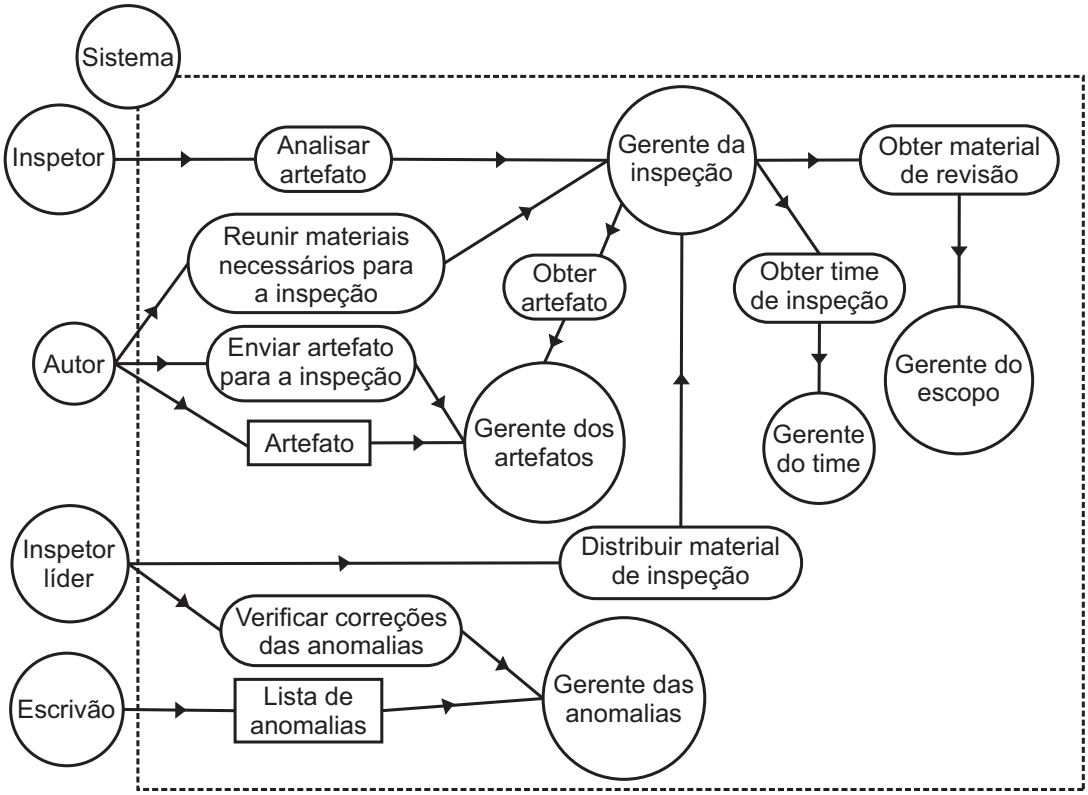
Dessa forma, foram introduzidos cinco "subatores", cada um para ser responsável por um ou mais objetivos, como é apresentado na Figura 20. São eles: *Gerente da inspeção*, *Gerente dos artefatos*, *Gerente das anomalias*, *Gerente do time* e *Gerente do escopo*.

Para que o objetivo *Reunir materiais necessários para a inspeção* do autor seja satisfeito, ele depende do "subator" do sistema chamado *Gerente da inspeção*. Para que esse "subator" consiga realizar o objetivo do autor, ele requer outros três objetivos, que são *Obter artefato*, *Obter time de inspeção* e *Obter material de revisão*.

O objetivo *Obter artefato* é realizado pelo "subator" *Gerente dos artefatos*, no qual busca os artefatos da inspeção no banco de dados e retorna para o *Gerente da inspeção*.

[2] Neste momento do projeto, os agentes ainda são "subatores".

Figura 20 – Projeto arquitetural.



Já o objetivo *Obter material de revisão* é realizado pelo "subator" *Gerente do escopo*, que consulta os itens de revisão do artefato na base de conhecimento e retorna-os para o *Gerente da inspeção*. Por fim, o objetivo *Obter time de inspeção* é realizado através do *Gerente do time*, no qual obtém os inspetores selecionados pelo inspetor líder. São necessários os membros do time nessa etapa para se criar um pacote de inspeção contendo os artefatos, itens de revisão e o time de inspeção. Desse forma, é criada uma relação de dependência entre o *Gerente da inspeção* e o *Gerente dos artefatos*, entre o *Gerente da inspeção* e o *Gerente do escopo*, e entre o *Gerente da inspeção* e o *Gerente do time*.

Para o autor *Enviar o artefato para a inspeção* ele necessita do "subator" *Gerente dos artefatos*. O objetivo é realizado quando o autor adiciona o artefato no repositório onde se localizam os artefatos. Após, o *Gerente dos artefatos* coleta esse artefato, identifica o tipo do artefato de acordo com os critérios vindos da base de conhecimento, e o adiciona no banco de dados. Além disso, o recurso *Artefato* também se torna gerenciado pelo *Gerente dos artefatos*. Assim, a realização do objetivo do autor cria uma relação de dependência entre eles.

Para que o inspetor possa *Analisar o artefato*, ele necessita que o "subator" *Gerente da inspeção* distribua o material de inspeção e envie um *e-mail* para o inspetor informando a localização dos *checklists* de revisão de cada artefato.

O objetivo do inspetor líder *Distribuir material de inspeção* é satisfeito quando o *Gerente da inspeção* distribui os materiais de revisão para cada inspetor. Para isso ele verifica se há artefatos na inspeção, se há membros do time selecionados pelo inspetor líder, e se há itens de revisão da base de conhecimento.

Há também o objetivo *Verificar correções das anomalias* do ator inspetor líder, que é satisfeito pelo "subator" *Gerente das anomalias*. Quando o autor corrige o artefato, o mesmo envia uma alteração do arquivo no repositório. O *Gerente das anomalias* verifica esses envios e através da mensagem que foi enviada, ele altera o estado da anomalia no banco de dados e envia uma notificação para o inspetor líder informando-o que a anomalia foi alterada.

Como as anomalias são gerenciadas pelo *Gerente das anomalias*, o recurso *Lista de anomalias* é dependente do "subator" *Gerente das anomalias*.

4.2.4 Projeto Detalhado

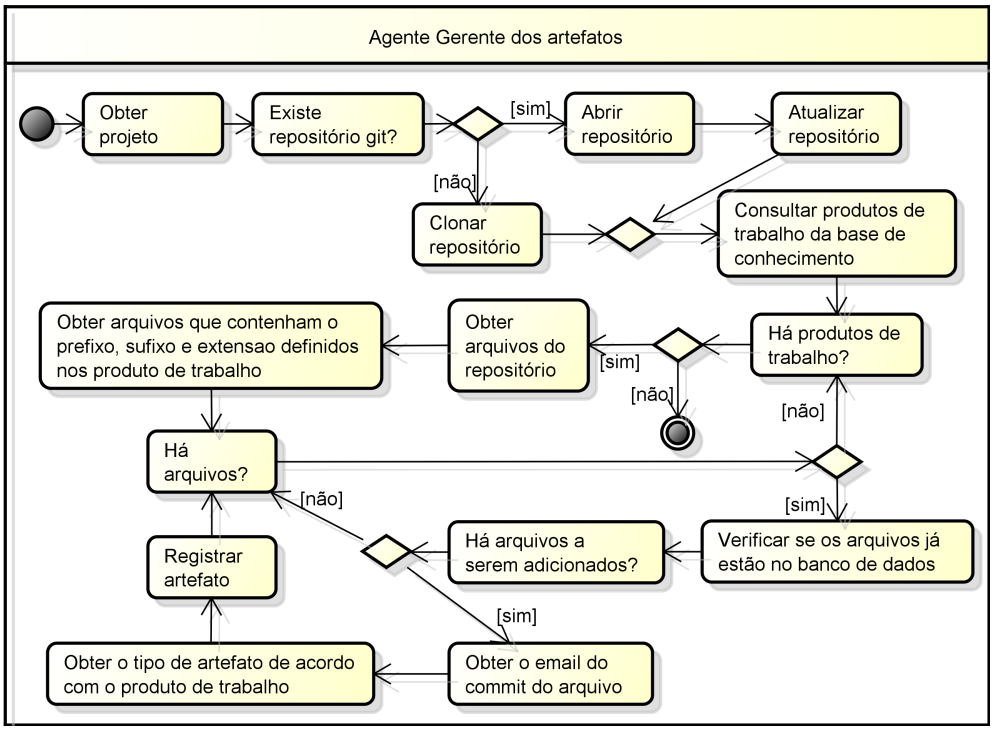
Nesta última etapa da metodologia Tropos é realizado o detalhamento de cada componente da arquitetura. Primeiro, os "subatores" do Sistema foram transformados em agentes, e para cada agente, foram criados os diagramas de suas capacidades. Através dos diagramas de capacidade também são ilustradas as interações com outros agentes.

O agente *Gerente dos artefatos* possui a capacidade de enviar o artefato para a inspeção, como apresentado na Figura 21. Para a realização dessa capacidade, ele realiza uma série de passos.

Primeiro, o agente *Gerente dos artefatos* obtém o projeto que contém o *link* do repositório Git. Após, ele verifica se o repositório já existe no computador, e, caso exista, ele abre o repositório e o atualiza, caso contrário ele clona o repositório. Conseguinte, o agente consulta os produtos de trabalho da base de conhecimento, que são quem define o tipo de cada artefato (plano do projeto, *script* de teste, etc). Para cada produto de trabalho, ele obtém os arquivos adicionados no repositório, filtrando aqueles que o contém o sufixo, prefixo e extensão definidos pelo produto de trabalho. Em seguida, para cada arquivo filtrado, o agente *Gerente dos artefatos* verifica se o mesmo está no banco de dados. Caso não esteja, o agente busca o *e-mail* do autor que adicionou o arquivo no repositório, obtém o tipo de artefato que define o arquivo, e o registra como um artefato pronto para a inspeção.

Houve a identificação da capacidade do agente *Gerente da inspeção* de reunir os materiais necessários para a inspeção, na qual é demostrada na Figura 22. Essa é a única capacidade que há interação entre agentes, onde o agente *Gerente da inspeção* faz requisições para os agentes *Gerente dos artefatos*, *Gerente do time* e *Gerente do escopo*, dessa forma, evidenciando a troca de mensagens entre agentes para a satisfação de seus

Figura 21 – Capacidade de enviar o artefato para a inspeção.



objetivos.

Para efetuar essa capacidade, o agente *Gerente da inspeção* obtém as inspeções que estão na fase de planejamento, e cria um pacote de inspeção para cada uma. Caso a inspeção não esteja pronta, ou seja, sem inspetores e itens de revisão, ele requisita a lista de artefatos da inspeção desse pacote. Essa requisição é enviada para o agente *Gerente dos artefatos*, que retorna a lista de artefatos da inspeção. Caso existam artefatos, o agente *Gerente da inspeção* faz uma requisição para obter os membros do time de inspeção, e outra requisição para obter os materiais de revisão do artefato. Em seguida, o agente *Gerente da inspeção* recebe o time de inspeção do agente *Gerente do time*, e o material de revisão do agente *Gerente do escopo*. Por fim, o time e o material de revisão são adicionados no pacote de inspeção.

A Figura 23 apresenta a capacidade do agente *Gerente dos artefatos* de obter os artefatos da inspeção. Para isso, o agente recebe a inspeção que os artefatos estão inclusos. Em seguida, acessa o banco de dados buscando pelos artefatos da inspeção. Após obter a lista de artefatos, o gerente dos artefatos envia essa lista para o agente que a solicitou.

A capacidade de obter o time de inspeção, como pode ser visto na Figura 24, é realizada pelo agente *Gerente do time*. O primeiro passo consiste em receber a inspeção de software no qual se deseja saber os inspetores. Conseguinte, o agente acessa o banco de dados em busca dos inspetores. Ao final, ele obtém os inspetores e envia-os para o

Figura 22 – Capacidade de reunir materiais necessários para a inspeção.

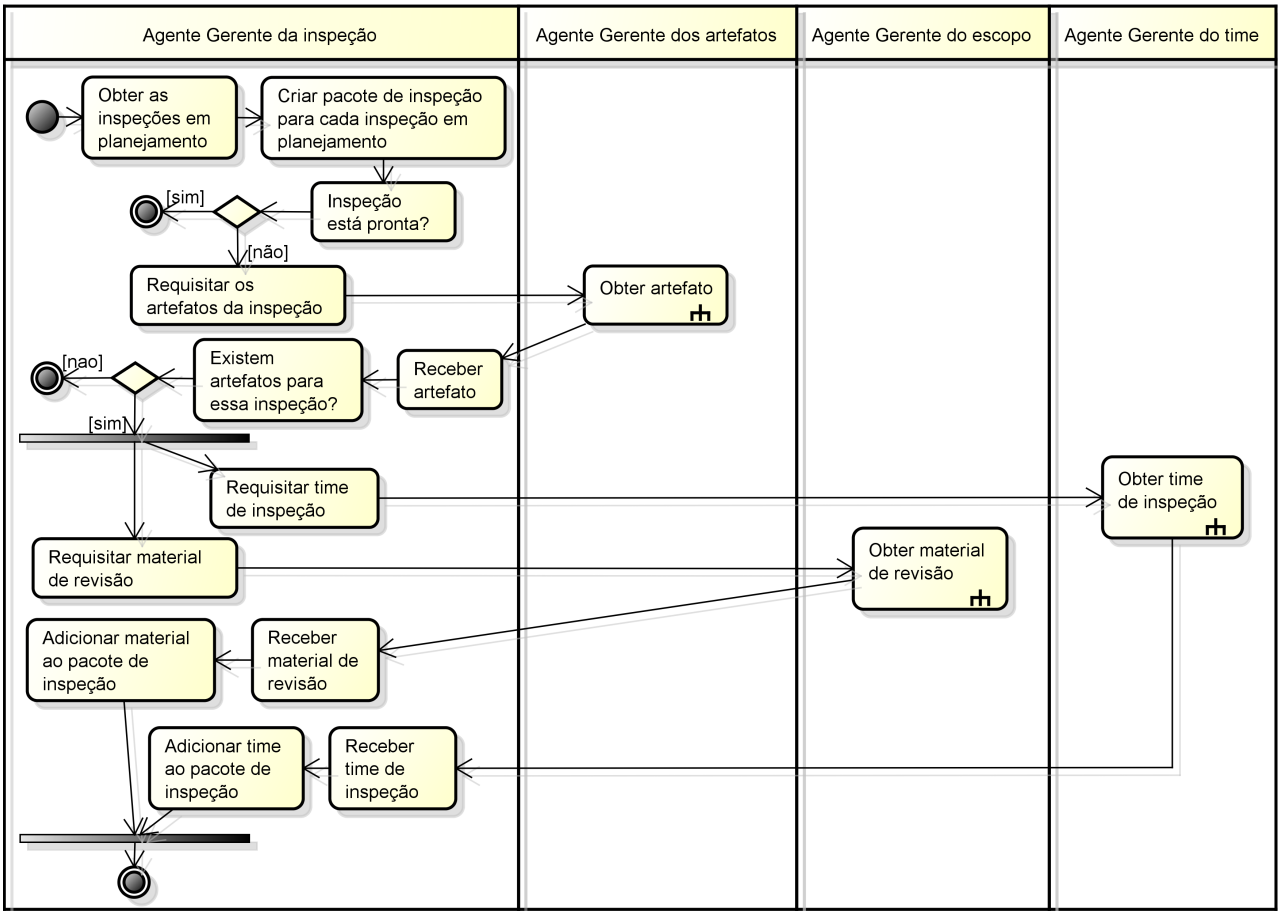
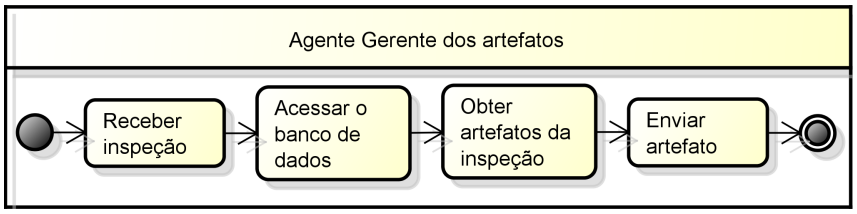


Figura 23 – Capacidade de obter o artefato.



agente solicitante.

Na Figura 25 é apresentado a capacidade do agente *Gerente do escopo* de obter o material de revisão. O agente recebe o artefato no qual devem ser consultados os itens de revisão, e após acessa a base de conhecimento. Ele realiza a consulta dos itens de revisão daquele artefato, obtém a lista de itens e os envia para o agente que realizou a solicitação.

Foi identificada a capacidade do agente *Gerente da inspeção* de distribuir o material da inspeção, como é apresentada na Figura 26. Em primeiro lugar, o agente *Gerente da inspeção* obtém os pacotes de inspeção em andamento, e verifica se cada pacote de

Figura 24 – Capacidade de obter o time de inspeção.

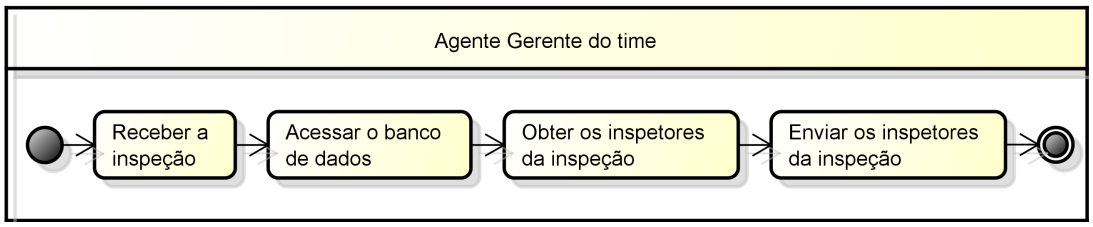
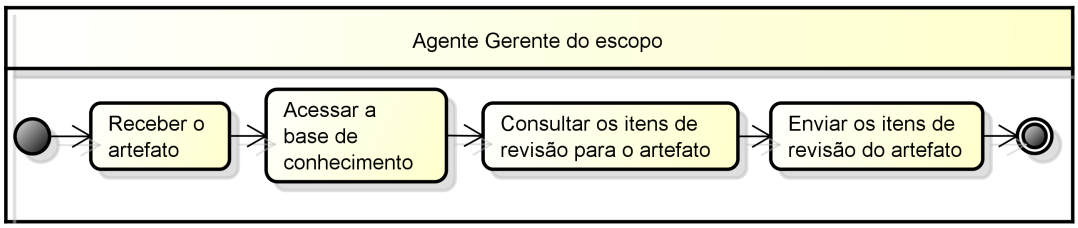
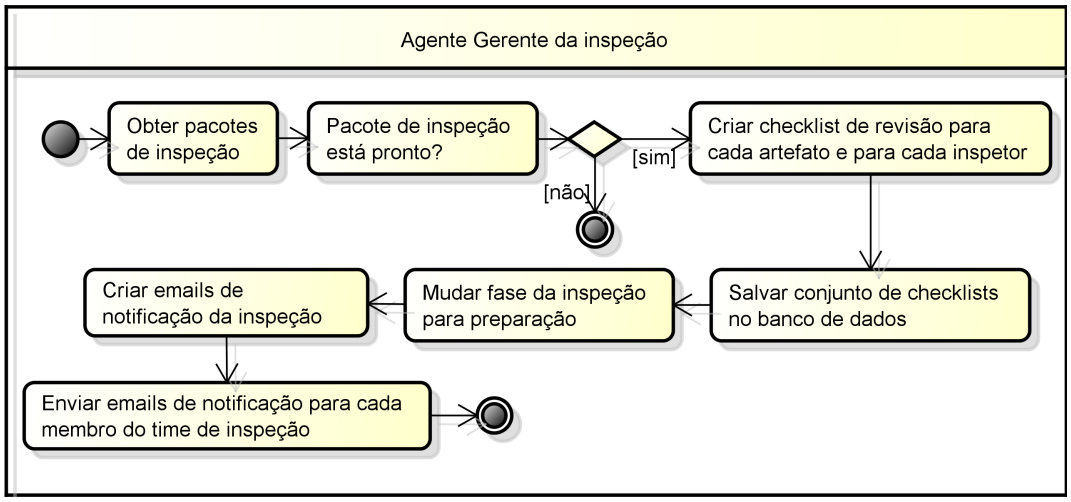


Figura 25 – Capacidade de obter o material de revisão.



inspeção está pronto, ou seja, com artefatos, itens de revisão e time de inspeção. Caso positivo, o agente cria um *checklist* de revisão de cada artefato para cada membro do time de inspeção. Em seguida, ele insere esse conjunto de *checklists* no banco de dados, ficando assim disponível para os inspetores acessarem e inspecionarem os artefatos. Após, o agente *Gerente da inspeção* altera a fase da inspeção para Preparação e a salva. No final, ele cria e envia os *e-mails* de notificação para os membros do time de inspeção.

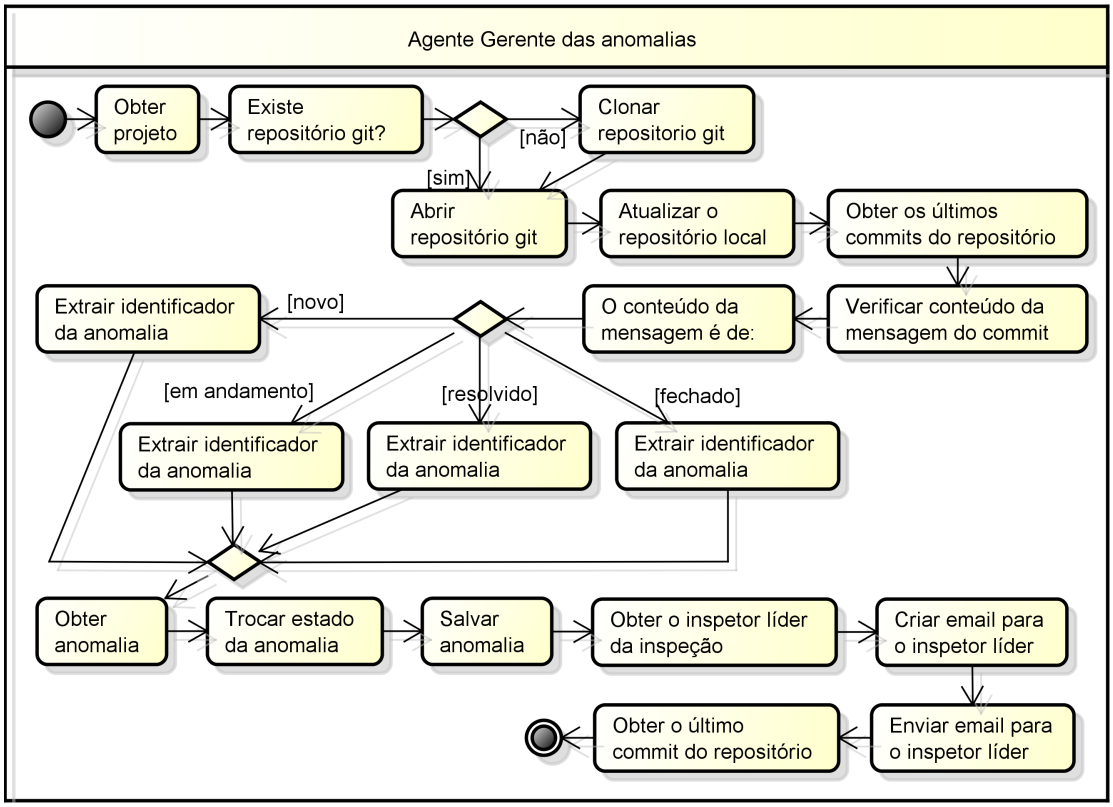
Figura 26 – Capacidade de distribuir o material de inspeção.



A última capacidade se trata da verificação das correções das anomalias, que é realizada pelo agente *Gerente das anomalias*. A Figura 27 apresenta essa capacidade, que é iniciada pela obtenção do projeto. Em seguida, é verificado se o repositório do

projeto já existe em disco. Caso negativo, o agente clona o repositório, que é aberto e atualizado. Após, são obtidos os últimos envios do repositório e verificados de acordo com a sua mensagem.

Figura 27 – Capacidade de verificar correções das anomalias.



Como demonstrado na Figura 27, caso a mensagem possua o identificador específico do estado da anomalia, o agente *Gerente das anomalias* extrai o identificador da anomalia e a obtém do banco de dados. Os identificadores específicos de cada estado da anomalia são: #novo-id, #em-andamento-id, #resolvido-id e #fechado-id, no qual o "id" representa o número identificador da anomalia a ser alterada. Conseguinte, ele altera o estado da anomalia de acordo com o estado informado na mensagem de envio, e a salva. Em seguida o agente obtém o inspetor líder da inspeção e envia um *e-mail* para que ele fique informado. Por fim, o agente *Gerente das anomalias* busca o último envio do repositório, para que quando seja realizado novamente esse comportamento, obtenha apenas as novas alterações.

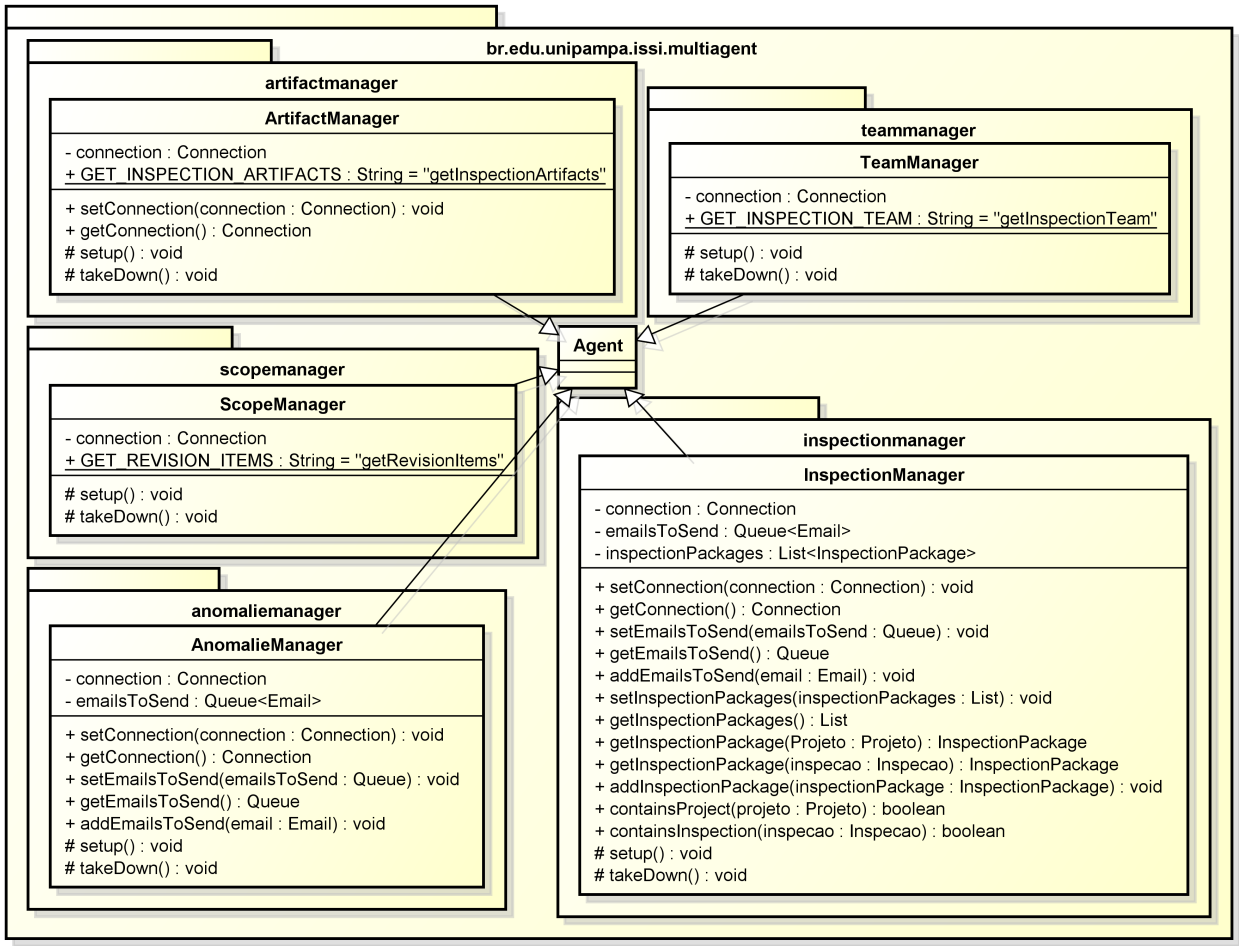
4.3 Implementação

Após a realização das quatro etapas da metodologia Tropos, se fez necessária a implementação do módulo multiagente. Para isso, foi utilizada a plataforma JADE,

que provê uma camada de funcionalidades básicas para a implementação de sistemas multiagentes.

Cada agente foi implementado seguindo a arquitetura proposta pela JADE, como é apresentado na Figura 28. Os agentes foram divididos por responsabilidades no qual cada um estende a classe *Agent*, pertencente à JADE. O agente *InspectionManager* é equivalente ao agente *Gerente da Inspeção*, no qual é responsável pela manipulação dos dados da inspeção, requisitando aos outros agentes artefatos, material de revisão e o time de inspeção quando necessários. Já o agente *ArtifactManager*, que é o mesmo agente *Gerente dos Artefatos*, é responsável pela manutenção dos artefatos, tanto na obtenção quanto no envio para a inspeção.

Figura 28 – Diagrama de classes dos agentes.

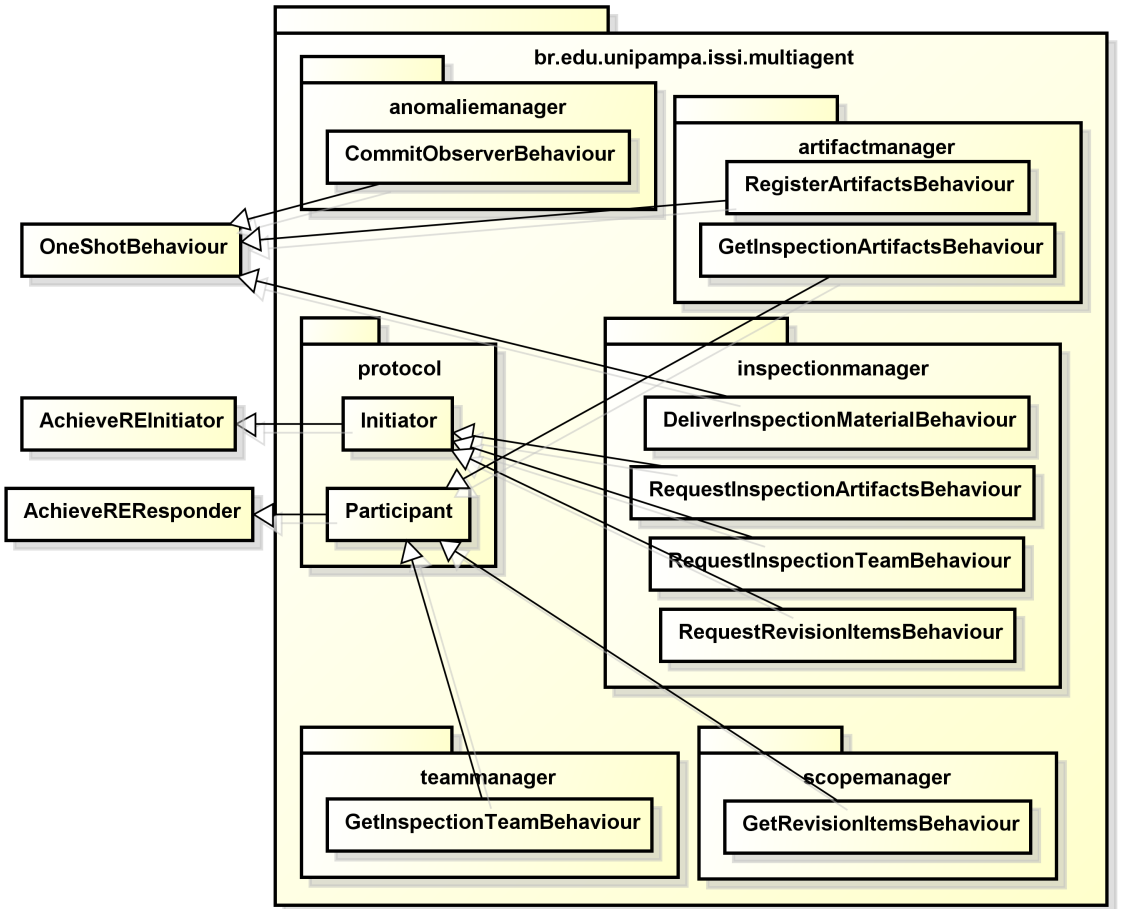


Ainda na Figura 28, há o agente *TeamManager*, o qual representa o agente *Gerente do time*, que é responsável pela obtenção dos membros do time de inspeção. O agente *ScopeManager* é responsável pela consulta e envio dos itens de revisão do artefato, representando o agente *Gerente do escopo*. Por fim, o agente *AnomalieManager*,

que representa o agente *Gerente das anomalias*, realiza o gerenciamento das anomalias, notificando o inspetor líder nas mudanças de estado.

Na Figura 29 são apresentados os comportamentos dos agentes, os quais se encontram no mesmo pacote onde ficam localizados seus agentes. A classe *OneShotBehaviour* é provida pela plataforma JADE, que estabelece o comportamento de atomicidade para as demais classes. Dela são estendidas as classes *CommitObserverBehaviour*, *RegisterArtifactsBehaviour* e *GetInspectionArtifactsBehaviour*.

Figura 29 – Diagrama de classes dos comportamentos, sem a visualização dos atributos e métodos.



A classe *CommitObserverBehaviour* implementa o comportamento do agente *Gerente das anomalias* de verificar as correções das anomalias. Já a classe *RegisterArtifactsBehaviour* implementa o comportamento do agente *Gerente dos artefatos* de enviar os artefatos para a inspeção. Por último, a classe *DeliverInspectionMaterialBehaviour* provê o comportamento de distribuir os materiais da inspeção.

Como apresentado na Figura 29, a classe *AchieveREInitiator* representa o comportamento de requisição, especificado pela FIPA. Dessa classe é estendida a classe *Initiator*, que apenas apresenta a troca de mensagens entre os agentes. Da classe *Initiator* são estendidas as classes *RequestInspectionArtifactsBehaviour*, *RequestInspectionTeamBehaviour* e

RequestRevisionItemsBehaviour.

O comportamento do agente *Gerente da inspeção* de requisitar os artefatos da inspeção para o agente *Gerente dos artefatos* é implementado pela classe *RequestInspectionArtifactsBehaviour*. Já a classe *RequestInspectionTeamBehaviour* provê o comportamento de realizar a requisição do time de inspeção para o agente *Gerente do time*. Em seguida, a classe *RequestRevisionItemsBehaviour* implementa o comportamento de requisitar os itens de revisão para o agente *Gerente do escopo*.

Por último, a classe *AchieveREResponder*, fornecido pela plataforma JADE, define o comportamento de responder a uma requisição da especificação FIPA. Como pode ser visualizado na Figura 29, a classe *Participant* é estendida da *AchieveREResponder*, no qual apenas retorna mensagens para o requisitante. Dessa classe são estendidas as classes *GetInspectionTeamBehaviour*, *GetRevisionItemsBehaviour* e *GetInspectionArtifactsBehaviour*.

O comportamento pertencente ao agente *Gerente do escopo* de obter o time de inspeção é realizado pela classe *GetRevisionItemsBehaviour*. A classe *GetInspectionTeamBehaviour* implementa o comportamento de obter o time de inspeção do agente *Gerente do time*. E o agente *Gerente dos artefatos* tem o classe *GetInspectionArtifactsBehaviour* que implementa o comportamento de obter os artefatos da inspeção.

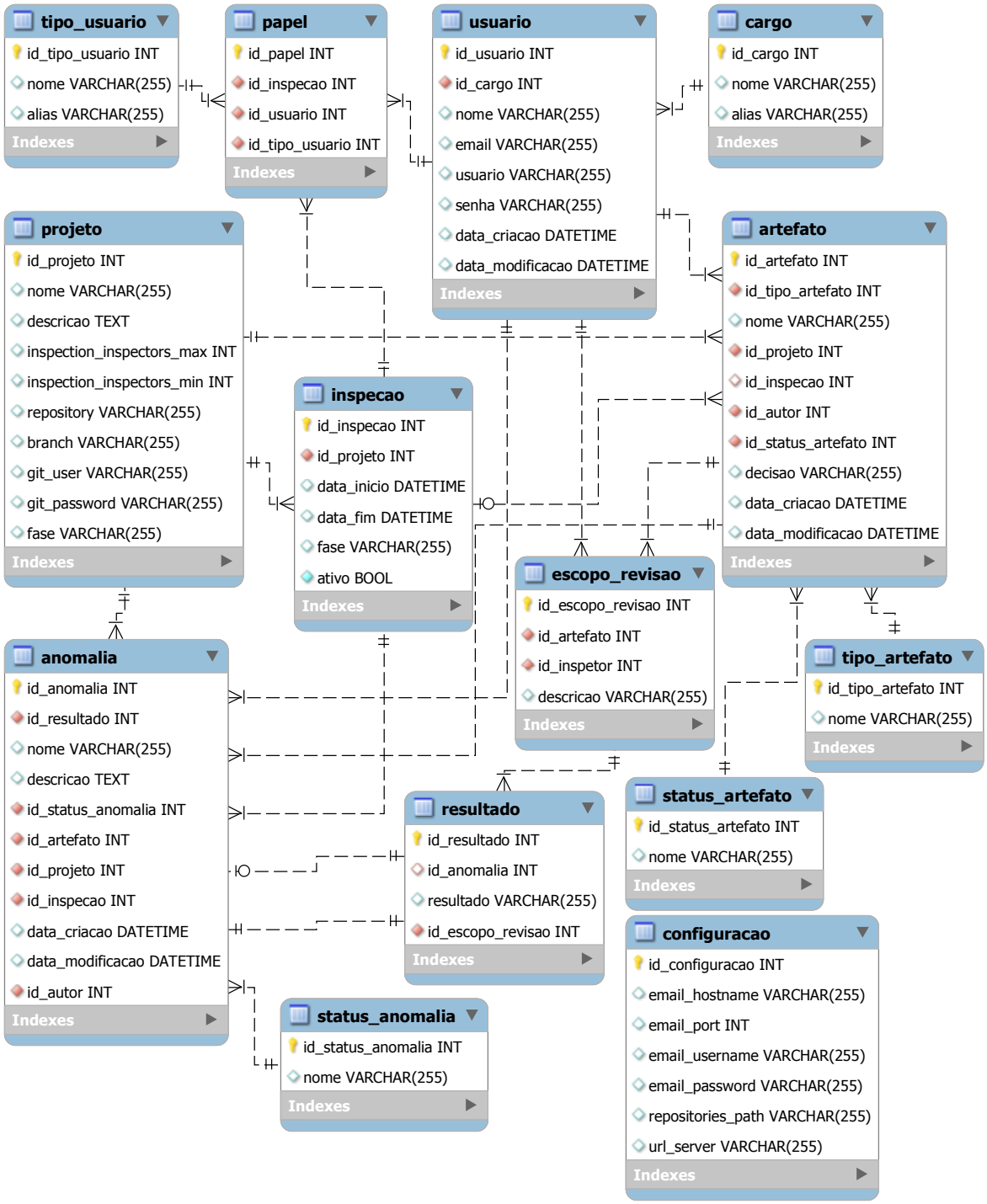
4.4 Contextualização do Ambiente Externo

Para que os sensores dos agente possam ser ativados, eles necessitam estar em um ambiente. O ambiente em que os sensores do sistema multiagente observam são o banco de dados e o repositório Git.

A cada envio no ramo das inspeções do repositório Git, os agentes *Gerente dos artefatos* e *Gerente das anomalias* são disparados em busca de novos artefatos e em busca da mensagem de alteração do estado da anomalia respectivamente. Porém, nenhum agente atua no ambiente Git, pois é apenas uma forma de visualização que os agentes possuem do ambiente de desenvolvimento. Outra forma de visualização do ambiente de desenvolvimento é através do banco de dados, como é apresentado na Figura 30. Entretanto, o banco de dados sofre modificações tanto do módulo multiagente quanto pelo time de inspeção por intermédio da aplicação *web*.

Na Figura 30 é apresentada a tabela *Projeto*, que mantém os dados referentes ao projeto de desenvolvimento que está sendo realizado no momento. Um projeto possui várias inspeções, onde cada inspeção possui um conjunto de artefatos, papéis e anomalias. A tabela *Usuario* contém os dados pertencentes à um usuário do sistema *web*. Esse usuário tem um *Cargo* que é definido pelo processo de desenvolvimento de software em execução.

Figura 30 – Diagrama Entidade-Relacionamento.



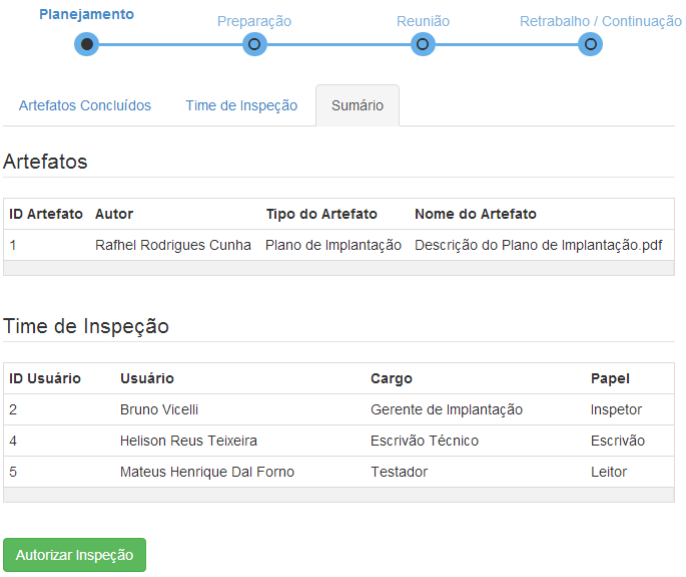
A tabela *Tipo_Usuario* contém os tipos de usuário do processo de inspeção, sendo eles autor, inspetor líder, inspetor, leitor e escrivão. No momento que existir uma associação entre um usuário, um tipo de usuário de uma inspeção, esse usuário terá um papel na inspeção, sendo armazenado na tabela *Papel*.

A tabela *Artefato* armazena os dados referentes aos artefatos das inspeções. Já a

tabela *Status_Artefato* diz em quais tipos de estado que o artefato pode estar, sendo eles "Novo", "Pronto para a Inspeção", "Em Inspeção" e "Inspecionado". Cada artefato da inspeção possui um tipo de artefato, que define quais itens de revisão serão adicionados para o artefato. A tabela *Escopo_Revisao* mantém os registros referentes a cada item de revisão de um inspetor e de um artefato, formando assim um *checklist* de revisão. Cada item do escopo de revisão pode ter vários resultados, que podem futuramente se tornar anomalias. Em seguida, a tabela *Anomalia* contém todas as anomalias do projeto. Uma anomalia tem um *Status_Anomalia* que define o estado da correção da mesma. Os estados podem ser "Novo", "Em Andamento", "Resolvido" e "Fechado". Por fim, há a tabela *Configuracao* que contém os dados globais para o envio de *e-mail*, o caminho das pastas dos projetos e o *link* para a aplicação *web* como pode-se visualizar na Figura 30.

Uma forma de inserir e recuperar dados dessa base é através da utilização da aplicação *web*. Na Figura 31 é apresentada a tela de planejamento da inspeção. No qual o inspetor líder escolhe os artefatos, seleciona o time de inspeção, designando para cada usuário um papel, e autoriza a inspeção. A partir desse momento o módulo multiagente obtém os artefatos, o time de inspeção, consulta os itens de revisão da ontologia e distribui esse material para os membros do time de inspeção.

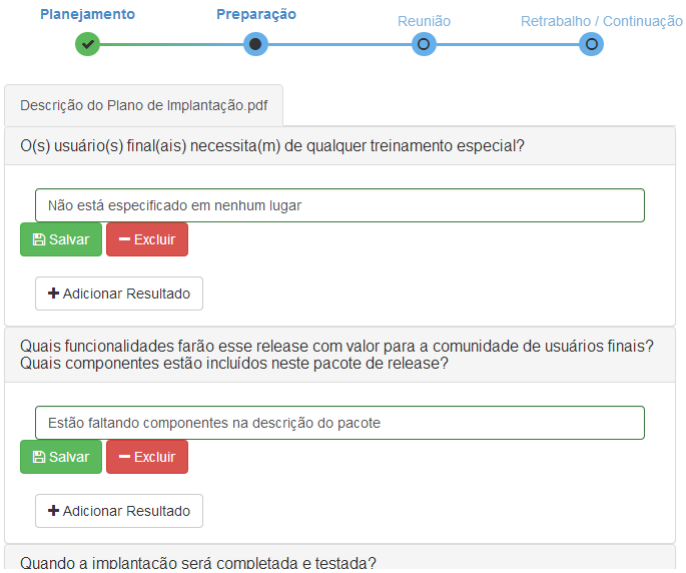
Figura 31 – Tela de Planejamento da Inspeção da Aplicação *Web*.



Após, a Figura 32 demonstra como um inspetor informa os resultados de cada item de revisão do artefato. Cada inspetor possui sua tela de resultados informados por ele. O inspetor seleciona o item de revisão no qual ele queira adicionar um resultado, insere o texto do resultado que ele identificou e clica em salvar. Nesse momento, na tela de preparação do inspetor líder, ele pode visualizar os resultados informados de todos os inspetores, para verificar o andamento e iniciar a reunião. Caso o inspetor desejar, ele

pode excluir um resultado informado, apenas clicando em excluir.

Figura 32 – Tela de Preparação da Inspeção da Aplicação *Web*.



Quando o inspetor líder inicia a reunião, ele é levado até a tela de reunião, como é apresentado na Figura 33. Nessa tela, somente o escrivão registra as anomalias, de acordo com o que é decidido na reunião de inspeção. Todo item de revisão que possui algum resultado é mostrado nessa tela, assim, o escrivão pode tornar aquele resultado uma anomalia, inserindo o nome e uma descrição para a mesma, e por fim salvando-a. Ao final da reunião, é decidida a saída de cada artefato da inspeção, que pode ser selecionado entre "Aceito sem correções", "Aceito com correções" e "Reinspeção". Após o escrivão salvar a decisão, ele finaliza a reunião.

Na tela de retrabalho / continuação, são apresentadas as anomalias encontradas durante a reunião, como pode ser visualizado na Figura 34. Nesse momento, o autor corrige as anomalias que foram passadas para ele, realizando o envio das alterações no repositório Git. Na mensagem de envio, ele insere o identificador específico para o estado da anomalia. Dessa forma, o sistema multiagente busca a anomalia informada e a altera de acordo com o estado informado na mensagem.

4.5 Fechamento do Capítulo

O presente capítulo apresentou a visão da solução desenvolvida durante este trabalho para o problema da produtividade em relação ao processo de inspeção de software. O uso da metodologia Tropos foi de grande ajuda, pois contribuiu na identificação do domínio do processo desde a fase de elicitação de requisitos até a implementação do agente de software. A modelagem realizada através da Tropos possibilitou a escolha das ati-

Figura 33 – Tela de Reunião da Inspeção da Aplicação *Web*.

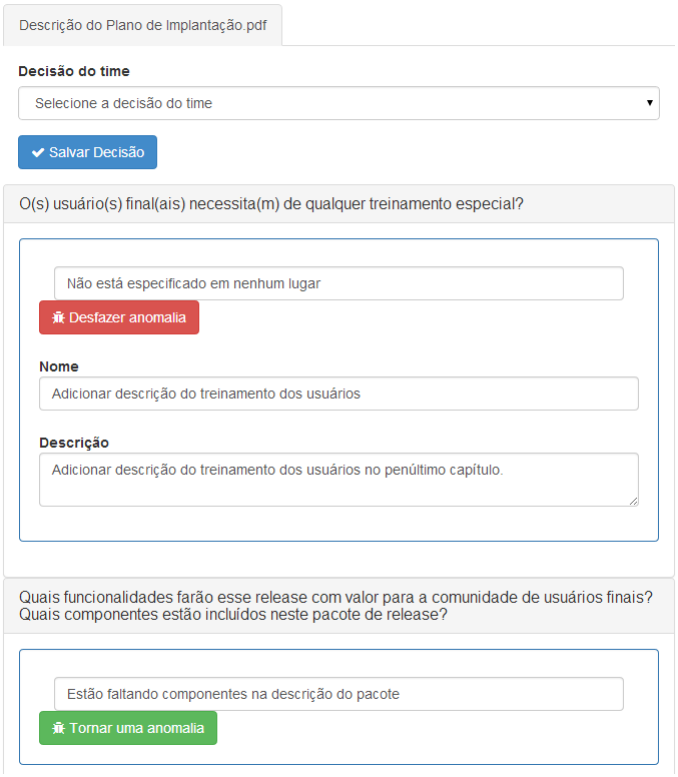
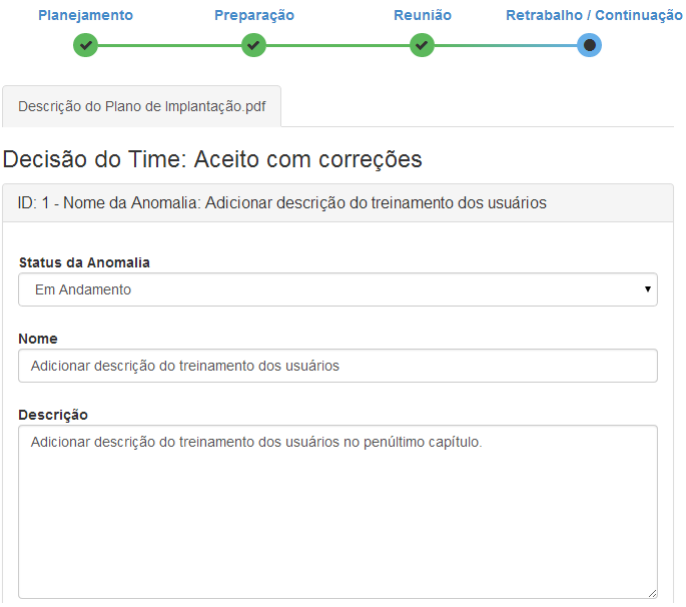


Figura 34 – Tela de Retrabalho / Continuação da Inspeção da Aplicação *Web*.



vidades que mais agregam valor quando automatizadas. Uma contribuição do trabalho para a metodologia tropos é a modelagem dos agentes e seus comportamentos através do diagrama de classes.

O sistema multiagente se mostrou completamente capaz de automatizar as atividades de agregação e distribuição dos materiais da inspeção, além do auxílio nas atividades de notificação dos membros. Porém, entende-se que se os agentes fossem dotados de capacidade cognitiva, poderia ser aumentada a contribuição em relação ao processo de inspeção. Essas características reforçariam os atributos de autonomia, pró-atividade e habilidade social dos agentes.

Pelo fato de não existir um alto acoplamento entre os módulos *web* e multiagente, entende-se que é possível a substituição da aplicação *web* sem impactos para o módulo multiagente, afinal o ambiente onde os agentes possuem sensores e atuam são o banco de dados e o repositório Git. Além disso, é possível a ampliação das funcionalidades de ambas as ferramentas sem impacto direto entre as mesmas.

5 Avaliação do Sistema Multiagente

Neste capítulo é apresentada a estratégia de testes realizada com o objetivo de verificar o sistema multiagente desenvolvido no Capítulo 4. Na seção 5.1 são descritas as estratégias de teste utilizadas para a verificação do módulo. A seção 5.2 apresenta os resultados obtidos dos casos de teste realizados. Já na seção 5.3 é feita uma análise dos resultados para a verificação do atendimento dos objetivos. Por fim, a seção 5.4 faz o fechamento do capítulo.

5.1 Estratégia de Testes

Foram realizadas cinco baterias de testes para a verificação do sistema multiagente desenvolvido. Em cada bateria foram verificadas as unidades de cada classe e a integração entre os componentes do módulo, além dos testes de comportamento e comunicação entre os agentes. O sistema multiagente foi testado inicialmente através da realização dos casos de teste unitários, em que cada unidade foi verificada em termos de entradas e saídas. Ou seja, cada método foi invocado, e testado o seu retorno de acordo com cada parâmetro enviado. Já para a realização dos casos de teste de integração, foram criados um conjunto de cenários aleatórios e experimentados, verificando se a saída do ambiente era condizente com o estado esperado do cenário.

Para a realização dos casos de teste dos comportamentos dos agentes, foram criados dados no ambiente em que há a sensibilização dos sensores. A partir dos dados no ambiente, os sensores eram disparados, no qual o agente realizava seu comportamento e atuava no ambiente, modificando-o. Foi verificado se o estado do ambiente após a atuação do agente condizia com o estado esperado.

A comunicação entre os agentes foi testada de maneira semelhante. O ambiente era preparado com dados que disparavam o comportamento sob teste. Após, era inicializada a plataforma JADE juntamente com os agentes que fossem se comunicar. Ao final do teste, foram comparadas as saídas resultantes dos comportamentos dos agentes com o as saídas esperadas. Ao total foram gerados 183 casos de teste unitários e de integração.

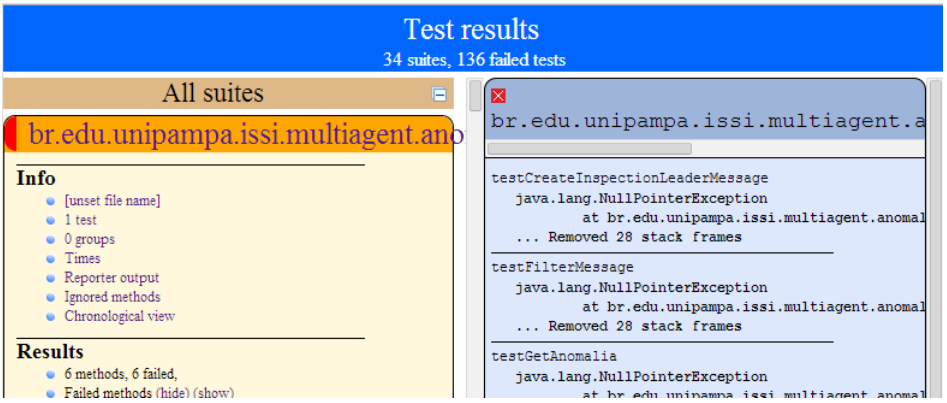
Por fim, através da simulação do processo de inspeção de software apresentada na subseção 2.1.2, foi realizada verificação do sistema multiagente em integração com a aplicação *web*, o banco de dados e a base de conhecimento. Para isso, foram criados casos de teste das fases do processo de inspeção que a ferramenta abrange.

5.2 Resultados dos Testes

Os resultados dos casos de teste unitários, integração, comportamento e comunicação entre os agentes foram separados pelas bateria de testes realizadas. Em cada bateria foram verificadas a quantidade de falhas encontradas, a quantidade de linhas cobertas e a quantidade de caminhos ou ramos percorridos.

A Figura 35 apresenta o relatório de falhas encontradas da primeira bateria de testes. Foram criados nessa bateria 181 casos de teste. Primeiro foi criado o esqueleto de cada teste, e após iniciou-se a codificação de cada *script* para a verificação. Pode-se visualizar que dos 181 casos de teste criados, 136 falharam. Isso se justifica pois nem todos os casos de teste criados possuíam um *script* para verificar as entradas e saídas dos métodos. Além disso, nem todos os casos de teste que verificavam o comportamento dos métodos estavam prontos.

Figura 35 – Relatório de falhas encontradas na bateria 1.



A cobertura do código-fonte na primeira bateria foi de 68%, e 64% a cobertura dos ramos, como pode ser visualizado na Figura 36. Pode-se verificar que o pacote *protocol* não possui dados para a cobertura dos ramos, pois as três classes desse pacote não possuem nenhuma clausula condicional ou de iteração.

Em seguida, foram refatorados os *scripts* de teste e o código-fonte do sistema multiagente, aumentando o número de testes de 181 para 195. Após, foi executada a segunda bateria de testes que foram evidenciadas 50 falhas, como é demonstrado na Figura 37. Houve um esforço para a correção dos casos de teste da bateria anterior, reduzindo a quantidade de falhas e aumentando a cobertura.

Na segunda bateria de testes, a cobertura do código-fonte aumentou para 80% e a cobertura dos ramos para 74%. Os casos de teste desenvolvidos antes da execução dessa bateria focaram na cobertura dos ramos, aumentando em 10% da bateria anterior, como pode ser visualizado na Figura 38;

Figura 36 – Relatório de cobertura do código-fonte da bateria 1.

Package	# Classes	Line Coverage	Branch Coverage
All Packages	40	68% 781/1138	64% 181/282
br.edu.unipampa.issi.multiagent	6	71% 208/289	64% 53/82
br.edu.unipampa.issi.multiagent.anomaliemanager	10	37% 79/209	30% 12/40
br.edu.unipampa.issi.multiagent.artifactmanager	5	75% 121/160	88% 23/26
br.edu.unipampa.issi.multiagent.git	2	88% 88/100	91% 22/24
br.edu.unipampa.issi.multiagent.inspectionmanager	10	85% 174/204	73% 59/80
br.edu.unipampa.issi.multiagent.protocol	3	55% 21/38	N/A N/A
br.edu.unipampa.issi.multiagent.scopemanager	2	74% 37/50	50% 1/2
br.edu.unipampa.issi.multiagent.teammanager	2	60% 53/88	39% 11/28

Figura 37 – Relatório de falhas encontradas na bateria 2.

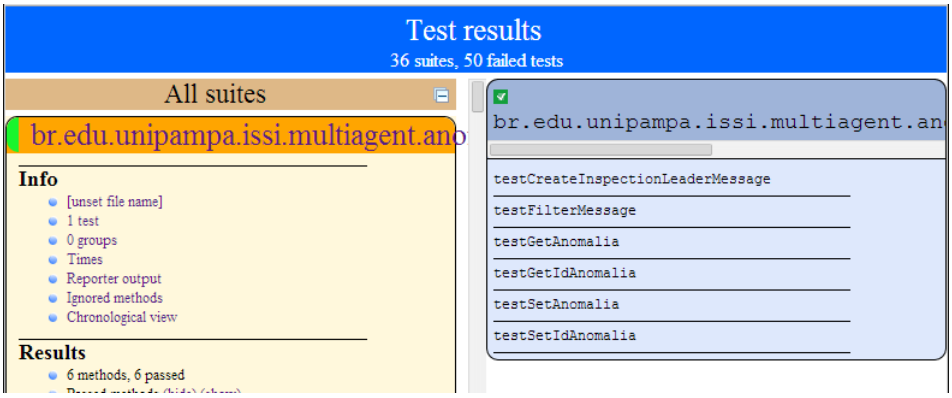


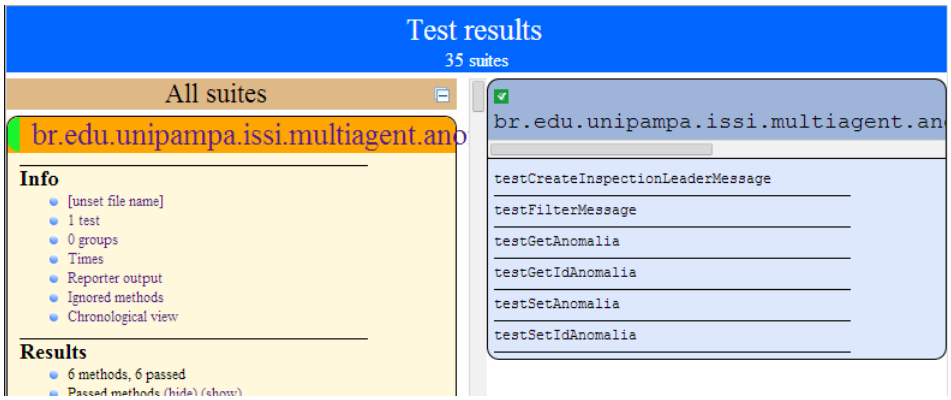
Figura 38 – Relatório de cobertura do código-fonte da bateria 2.

Package	# Classes	Line Coverage	Branch Coverage
All Packages	42	80% 960/1187	74% 241/324
br.edu.unipampa.issi.multiagent	6	90% 258/286	89% 86/96
br.edu.unipampa.issi.multiagent.anomaliemanager	9	77% 158/203	51% 27/52
br.edu.unipampa.issi.multiagent.artifactmanager	5	74% 117/158	89% 25/28
br.edu.unipampa.issi.multiagent.git	2	83% 96/115	92% 24/26
br.edu.unipampa.issi.multiagent.inspectionmanager	10	85% 176/206	71% 60/84
br.edu.unipampa.issi.multiagent.ontology	3	76% 33/43	75% 6/8
br.edu.unipampa.issi.multiagent.protocol	3	76% 29/38	N/A N/A
br.edu.unipampa.issi.multiagent.scopemanager	2	80% 40/50	100% 2/2
br.edu.unipampa.issi.multiagent.teammanager	2	60% 53/88	39% 11/28

Após a segunda bateria de testes, foi aumentado o esforço na correção das falhas encontradas. Foi verificada a existência de alguns métodos não utilizados no código-fonte e nos *scripts* de teste, os quais foram retirados. Ao final, foi executada a terceira bateria de testes como é apresentada na Figura 39. Nenhuma falha foi encontrada dos 184 casos de teste realizados.

Como consequência da correção das falhas encontradas, a cobertura do código-

Figura 39 – Relatório de falhas encontradas na bateria 3.



fonte aumentou em 6%, e a cobertura dos ramos em 7%, como demostrado na Figura 40.

Figura 40 – Relatório de cobertura do código-fonte da bateria 3.

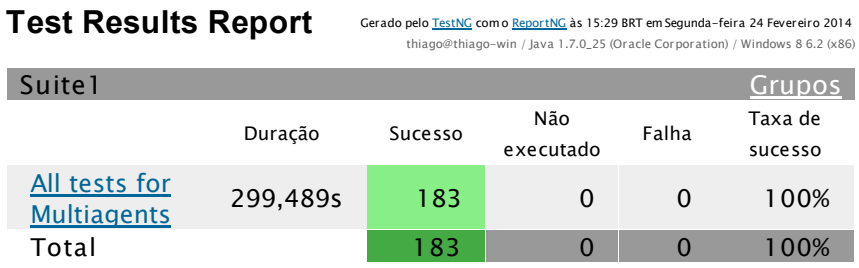
Package	# Classes	Line Coverage		Branch Coverage	
All Packages	41	86%	941/1087	81%	232/286
br.edu.unipampa.issi.multiagent	6	89%	253/282	86%	83/96
br.edu.unipampa.issi.multiagent.anomaliemanager	9	80%	164/205	53%	29/54
br.edu.unipampa.issi.multiagent.artifactmanager	4	88%	104/117	95%	23/24
br.edu.unipampa.issi.multiagent.git	2	83%	96/115	92%	24/26
br.edu.unipampa.issi.multiagent.inspectionmanager	10	90%	172/190	82%	56/68
br.edu.unipampa.issi.multiagent.ontology	3	86%	31/36	100%	6/6
br.edu.unipampa.issi.multiagent.protocol	3	91%	33/36	N/A	N/A
br.edu.unipampa.issi.multiagent.scopemanager	2	80%	38/47	100%	2/2
br.edu.unipampa.issi.multiagent.teammanager	2	84%	50/59	90%	9/10

O gerador automático de relatórios foi trocado, assim possibilitando uma melhor visualização das falhas, como pode ser visto na Figura 41. Com nenhuma falha sendo encontrada, o foco passou a ser a refatoração do código-fonte e dos casos de teste. Foram retirados outros métodos e casos de teste que não estavam sendo utilizados, o que fez a cobertura de linhas de código diminuir em 2% e a de ramos em 6%. Ocorreu essa redução, pois os métodos que foram retirados contavam para o número de linhas e de ramos cobertos, diminuindo assim a percentagem de código coberto, como pode ser visualizado na Figura 42. Além disso, a quantidade de casos de teste diminuiu de 184 para 183.

Por fim, houve o esforço para o aumento da cobertura das linhas e dos ramos do código-fonte. Os casos de teste continuaram não encontrando falhas durante a quinta bateria de testes, como a é apresentado na Figura 43. Porém, houve o aumento da cobertura das linhas e ramos, chegando respectivamente em 86% e 82%, como a pode ser visualizado na Figura 44.

O restante das linhas e ramos não cobertos corresponde principalmente aos não tratamento de exceções que são disparadas pelas APIs utilizadas. Entretanto, o fluxo

Figura 41 – Relatório de falhas encontradas na bateria 4.



Test Results Report

Gerado pelo TestNG como o ReportNG às 15:29 BRT em Segunda-feira 24 Fevereiro 2014
thiago@thiago-win / Java 1.7.0_25 (Oracle Corporation) / Windows 8 6.2 (x86)

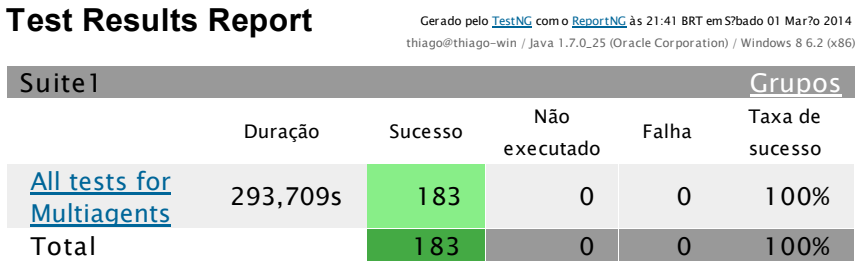
Suite1 — Grupos

	Duração	Sucesso	Não executado	Falha	Taxa de sucesso
All tests for Multiagents	299,489s	183	0	0	100%
Total		183	0	0	100%

Figura 42 – Relatório de cobertura do código-fonte da bateria 4.

Package	# Classes	Line Coverage		Branch Coverage	
All Packages	41	84%	900/1061	75%	205/272
br.edu.unipampa.issi.multiagent	6	89%	253/282	86%	83/96
br.edu.unipampa.issi.multiagent.anomaliemanager	9	80%	164/205	53%	29/54
br.edu.unipampa.issi.multiagent.artifactmanager	4	88%	104/117	95%	23/24
br.edu.unipampa.issi.multiagent.git	2	80%	72/89	91%	11/12
br.edu.unipampa.issi.multiagent.inspectionmanager	10	81%	155/190	61%	42/68
br.edu.unipampa.issi.multiagent.ontology	3	86%	31/36	100%	6/6
br.edu.unipampa.issi.multiagent.protocol	3	91%	33/36	N/A	N/A
br.edu.unipampa.issi.multiagent.scopemanager	2	80%	38/47	100%	2/2
br.edu.unipampa.issi.multiagent.teammanager	2	84%	50/59	90%	9/10

Figura 43 – Relatório de falhas encontradas na bateria 5.



Test Results Report

Gerado pelo TestNG como o ReportNG às 21:41 BRT em S?bado 01 Mar?o 2014
thiago@thiago-win / Java 1.7.0_25 (Oracle Corporation) / Windows 8 6.2 (x86)

Suite1 — Grupos

	Duração	Sucesso	Não executado	Falha	Taxa de sucesso
All tests for Multiagents	293,709s	183	0	0	100%
Total		183	0	0	100%

Figura 44 – Relatório de cobertura do código-fonte da bateria 5.

Package	# Classes	Line Coverage		Branch Coverage	
All Packages	40	86%	909/1049	82%	219/266
br.edu.unipampa.issi.multiagent	6	90%	257/283	89%	86/96
br.edu.unipampa.issi.multiagent.anomaliemanager	9	80%	167/207	64%	35/54
br.edu.unipampa.issi.multiagent.artifactmanager	4	87%	102/117	87%	21/24
br.edu.unipampa.issi.multiagent.git	2	80%	72/89	91%	11/12
br.edu.unipampa.issi.multiagent.inspectionmanager	10	88%	169/190	79%	54/68
br.edu.unipampa.issi.multiagent.ontology	2	100%	25/25	100%	4/4
br.edu.unipampa.issi.multiagent.protocol	3	91%	33/36	N/A	N/A
br.edu.unipampa.issi.multiagent.scopemanager	2	80%	38/47	100%	2/2
br.edu.unipampa.issi.multiagent.teammanager	2	83%	46/55	100%	6/6

principal e vários caminhos alternativos que o sistema multiagente utiliza estão sendo verificados e cobertos.

Os casos de teste do sistema foram realizados a partir da simulação do processo de inspeção apresentado na subseção 2.1.2. Os cenários foram separados de acordo com cada fase do processo de inspeção de software que envolve o sistema de suporte. Adicionalmente foi criado o caso de teste para o cadastro das inspeções. Os resultados da realização dos casos de testes do sistema podem ser vistos através da Tabela 1.

Tabela 1 – Resultados dos casos de teste do sistema

Casos de Teste do Sistema	**Resultados**
Cenário - Criar Inspeção	
CT-01.01 - Inspeção não existente	✓Passou
CT-01.02 - Data inválida	✓Passou
Cenário - Planejamento	
CT-02.01 - Autorização com sucesso	✓Passou
CT-02.02 - Autorização sem artefatos	✓Passou
CT-02.03 - Autorização sem inspetores	✓Passou
Cenário - Preparação	
CT-03.01 - Informar um resultado	✓Passou
CT-03.02 - Excluir um resultado	✓Passou
CT-03.03 - Excluir sem salvar	✓Passou
CT-03.04 - Iniciar reunião	✓Passou
CT-03.05 - Iniciar reunião sem resultados	✓Passou
Cenário - Reunião	
CT-04.01 - Tornar anomalia	✓Passou
CT-04.02 - Desfazer anomalia	✓Passou
CT-04.03 - Salvar decisão	✓Passou
CT-04.04 - Retirar decisão do time	✓Passou
CT-04.05 - Finalizar Reunião	✓Passou
Cenário - Retrabalho / Continuação	
CT-05.01 - Alterar o status da anomalia para em andamento	✓Passou
CT-05.02 - Alterar o status da anomalia para resolvido	✓Passou
CT-05.03 - Alterar o status da anomalia para fechado	✓Passou

No primeiro cenário, é verificada a criação da inspeção. Foram criados dois casos de teste para esse cenário: *cadastrar uma inspeção não existente* e *cadastrar uma inspeção com uma data inválida*. O primeiro caso de teste foi realizado utilizando a simulação apresentada anteriormente, e a inspeção foi cadastrada com sucesso, realizando assim o comportamento planejado. O segundo caso de teste também foi executado a partir da simulação, e como resultado uma mensagem de data inválida foi exibida, realizando assim o comportamento planejado.

O segundo cenário trata sobre os casos de testes do sistema para a fase de planejamento do processo de inspeção. Desse cenário foram criados os casos de teste: *autorização*

com sucesso, *autorização sem artefatos* e *autorização sem inspetores*. O caso de teste de *autorização com sucesso* verifica se uma inspeção com artefatos e time de inspeção selecionados é autorizada. Já o caso de teste de *autorização sem artefatos* verifica se uma inspeção sem artefatos escolhidos é autorizada. Por fim, o caso de teste de *autorização sem inspetores* verifica se uma inspeção sem um time de inspeção selecionado é autorizada. Todos os casos de teste do cenário de planejamento executaram a simulação do processo e passaram conforme pode ser visualizado na Tabela 1.

O cenário de preparação aborda os casos de testes relacionados com a fase de preparação da inspeção de software. Os casos de teste *informar um resultado*, *excluir um resultado*, *excluir sem salvar*, *iniciar reunião* e *iniciar reunião sem resultados* foram derivados desse cenário. O caso de teste *informar um resultado* verifica se o sistema de suporte insere corretamente um resultado informado por um inspetor. Já o caso de teste *excluir um resultado* põe à prova se o sistema de suporte exclui corretamente um resultado informado anteriormente. Em seguida, o caso de teste *excluir sem salvar* inspeciona se é possível excluir um resultado não salvo. Conseguinte, o caso de teste *inciar reunião* verifica se é possível iniciar a reunião de inspeção corretamente. Por fim, o caso de teste *iniciar reunião sem resultados* testa se é possível iniciar a reunião sem resultados informados. O comportamento do sistema de suporte na execução dos casos de teste do sistema através da simulação do processo de inspeção também foi o correto, como é demonstrado na Tabela 1.

O quarto cenário é relacionado com a fase de reunião do processo de inspeção de software. Desse cenário foram criados cinco casos de teste: *tornar anomalia*, *desfazer anomalia*, *salvar decisão*, *retirar decisão do time* e *finalizar reunião*. O primeiro caso de teste verifica se é possível tornar um resultado de um inspetor em uma anomalia. O caso de teste *desfazer anomalia* testa se, após um resultado se tornar uma anomalia, é possível desfazer tal anomalia. Em seguida, o caso de teste *salvar decisão* põe à prova se o sistema de suporte consegue salvar uma decisão selecionada pelo escrivão. Entretanto, o caso de teste *retirar decisão* do time verifica se é possível desfazer a decisão selecionada anteriormente. Por último, o caso de teste *finalizar reunião* testa a capacidade do sistema de suporte de finalizar a reunião em andamento. A Tabela 1 mostra que a execução da simulação para os casos de teste da fase de reunião foram positivos.

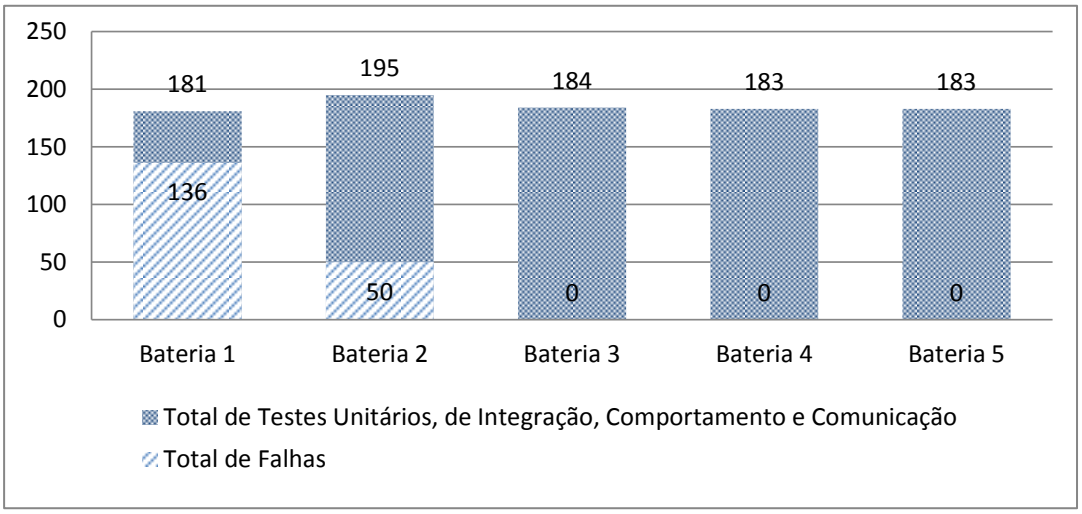
Enfim, o último cenário ilustra os casos de testes do sistema relacionados com a fase de retrabalho / continuação do processo de inspeção de software. Três casos de testes foram extraídos desse cenário: *alterar o status da anomalia para em andamento*, *alterar o status da anomalia para resolvido*, e *alterar o status da anomalia para fechado*. Ambos os três testes verificam se o sistema multiagente consegue alterar o estado da anomalia para o estado que foi informado na mensagem de envio. Como pode ser verificado na Tabela 1, a execução de todos os casos de teste utilizando a simulação do processo de

inspeção passaram.

5.3 Análise dos Resultados

O primeiro gráfico a ser analisado é referente a quantidade total de falhas encontradas durante cada bateria de testes unitários, integração, comportamento e comunicação entre os agentes, como é apresentado na Figura 45. Pode-se verificar que houve uma redução das falhas durante as duas primeiras baterias de testes, não havendo nenhuma falha a partir da terceira bateria.

Figura 45 – Gráfico de Quantidade de Falhas por Bateria de Teste.



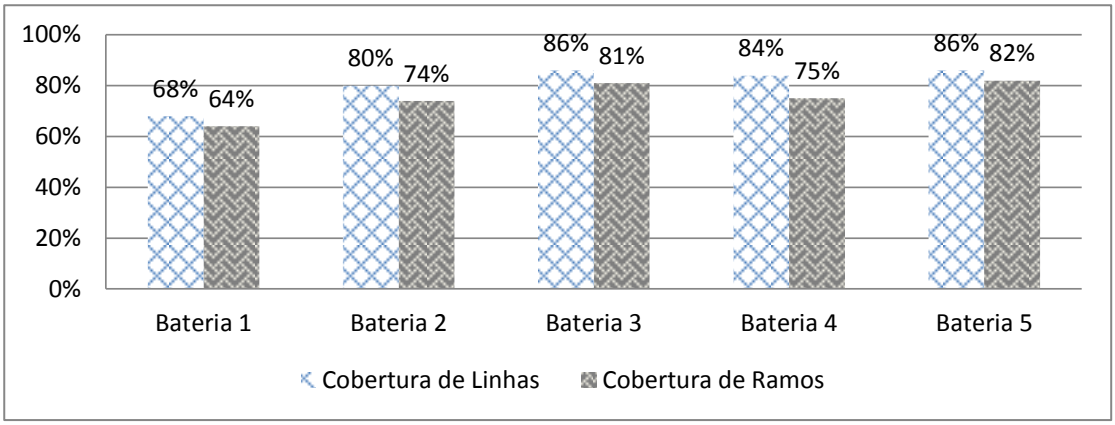
Na Figura 46 é apresentado a percentagem da cobertura das linhas e a cobertura dos ramos por bateria de testes. Houve um aumento da cobertura durante as três primeiras baterias de teste, no qual foi reduzida na quarta bateria, e após aumentando novamente na quinta bateria. Ao final, se chegou a um total de 86% de cobertura de linhas de código-fonte e a 82% de cobertura de ramos.

Ao analisar a Figura 45 e a Figura 46, pode-se concluir que apesar dos casos de teste cobrirem 86% do código-fonte, esses 86% estão corretos de acordo com os resultados esperados. Fazem parte dos 86% todos os fluxos principais e secundários do sistema multiagente.

Através da realização dos casos de testes do sistema, foi possível verificar a integração do sistema multiagente com a aplicação *web*, o banco de dados e a base de conhecimento. Analisando-se a Tabela 1 é possível afirmar que o sistema multiagente é funcional, e atende a todas as atividades que fazem parte do seu escopo.

A Tabela 2 apresenta as fases do processo de inspeção em que cada módulo do sistema atua. Pode ser verificado que as fases de *Preparação Gerencial*, *Apresentação dos*

Figura 46 – Gráfico de Percentagem de Cobertura do Código-Fonte por Bateria de Teste.



Procedimentos de Inspeção e *Apresentação do Produto de Inspeção* não possuem interação com o sistema de suporte. Isso é justificável pois essas fases não fazem parte do escopo definido pelo sistema. A fase de *Preparação Gerencial*, possui apenas atividades de suporte à inspeção. Já as fases de *Apresentação dos Procedimentos de Inspeção* e *Apresentação do Produto de Inspeção* possuem atividades intrinsecamente humanas. Dessa forma, o sistema de suporte não agregaria tanto quanto a automatização de atividades das outras fases.

Tabela 2 – Atuação dos módulos do sistema de suporte no processo de inspeção de software.

Módulos do Sistema de Suporte à Inspeção de Software	**Preparação Gerencial**	**Planejamento da Inspeção**	**Apresentação dos Procedimentos de Inspeção**	**Apresentação do Produto de Inspeção**	**Preparação**	**Reunião**	**Retrabalho / Continuação**
Aplicação *Web*	×	✓	×	×	✓	✓	✓
Módulo Multiagente	×	✓	×	×	×	×	✓
Banco de Dados	×	✓	×	×	✓	✓	✓
Base de Conhecimento	×	✓	×	×	×	×	✓

O módulo multiagente está presente nas fases de *Planejamento da Inspeção* e *Retrabalho / Continuação* do processo de inspeção de software, como pode ser visualizado na Tabela 2. Para a fase de *Planejamento da Inspeção*, o sistema multiagente realiza as atividades de enviar o artefato para a inspeção, reunir e distribuir os materiais da inspeção e possibilitar o inspetor analisar o artefato. Na fase de *Retrabalho / Continuação* o módulo multiagente atua na verificação das correções das anomalias para o inspetor líder.

Dessa forma, é possível observar indícios de que o módulo multiagente aumenta a produtividade em uma inspeção de software, pois realiza atividades que eram feitas manualmente. As atividades de reunir e distribuir o material da inspeção impactam positivamente pois são onerosas se feitas manualmente. Além disso, as atividade de enviar o artefato para a inspeção e verificar as correções das anomalias necessitam que a pessoa pare a sua atividade atual para que realize essas tarefas. Com a utilização do módulo multiagente, é possível a realização dessas tarefas de forma mais rápida e sem grandes impactos no processo de desenvolvimento.

5.4 Fechamento do Capítulo

O presente capítulo trouxe como proposta a avaliação do sistema multiagente para suporte à inspeção de software apresentado no Capítulo 4. No decorrer do capítulo é possível verificar que o aplicativo e seus casos de teste ainda podem ser aprimorados. O sistema de suporte pode ser melhorado quanto a sua integração com outros ambientes, assim como pode ser aumentada a cobertura dos testes. Entretanto, a estratégia adotada se mostrou válida pois podem ser verificadas todas as funcionalidades do sistema multiagente, tanto isoladamente quanto integrado ao sistema de suporte. Os resultados obtidos são satisfatórios, pois verificam que o sistema multiagente foi capaz de automatizar algumas atividades das fases de planejamento da inspeção e retrabalho / continuação do processo de inspeção de software. Além disso, o módulo multiagente se comportou de forma esperada, realizando as atividades de forma correta, o que atende aos objetivos do trabalho.

Uma forma de provar que o sistema multiagente aumenta a produtividade pode ser através de estudos de caso de várias aplicações de inspeções de software sem utilizar o módulo multiagente, e utilizando-o. Dessa maneira, pode ser criado um comparativo verificando qual abordagem proporciona maior suporte e produtividade ao processo de inspeção de software.

6 Considerações Finais

Foi verificado durante o presente trabalho que há dificuldades ao se tentar implantar um processo de inspeção de software. Essas dificuldades variam da complexidade do processo à adoção de forma equivocada. Dessa forma, a produtividade da inspeção é impactada, pois há um maior esforço em sua execução do que na busca por anomalias. Sendo assim, foi criado um projeto de pesquisa vinculado ao LESA, da Universidade Federal do Pampa, para auxiliar no processo de inspeção de software. Nesse projeto de pesquisa está incluso o presente trabalho, o qual tem por objetivo desenvolver uma solução para automatizar atividades específicas do processo de inspeção de software visando uma maior produtividade.

O sistema multiagente para suporte à inspeção de software apresentado durante o presente trabalho se mostrou em conformidade com os objetivos inicialmente traçados. Essa conclusão é fundamentada da análise e modelagem do processo de inspeção, da elicitação e escolha das atividades a serem automatizadas, e da integração e testes do módulo multiagente com o sistema de suporte às inspeções de software.

A realização da análise e modelagem do ambiente do processo de inspeção de software realizada através da metodologia Tropos possibilitou a conclusão do primeiro objetivo específico. Por intermédio dessa análise, foram identificadas e escolhidas as atividades que possibilitavam uma maior produtividade para o processo de inspeção, completando-se o segundo objetivo específico. Após foi realizado o projeto arquitetural e detalhado do módulo multiagente. Em seguida, foi implementado o sistema multiagente e integrado ao sistema de suporte. Por fim, o sistema multiagente foi testado e foi realizada a coleta de dados referente aos testes, onde foi possível verificar que o módulo multiagente atende a todas as funcionalidades especificadas, realizando o terceiro objetivo específico.

É possível concluir que o módulo multiagente desenvolvido, responsável por automatizar algumas atividades específicas, é plenamente funcional. Além disso, as possibilidades de autonomia e pró-atividade dos agentes não foram esgotadas. Através da análise dos resultados é possível verificar que há fases do processo de inspeção que ainda podem ser automatizados. Finalmente, os casos de teste podem ser aprimorados para uma maior detecção de erros e para uma maior cobertura.

A análise e modelagem do domínio do processo de inspeção deixou clara a possibilidade da automatização de atividades do cotidiano das inspeções. Essa modelagem se tornou a base da construção do módulo multiagente, e a partir dela que foi possível o projeto e implementação das funcionalidades desenvolvidas.

A utilização da metodologia Tropos foi fundamental para o desenvolvimento do

sistema multiagente. Porém, sua curva de aprendizagem não é rápida, com poucos modelos práticos e sendo necessário a visita constante na documentação que a descreve. Porém, os diagramas de dependência e de perspectiva são de grande valia para a observação e entendimento do domínio descrito. A metodologia em si é consistente, possibilitando o desenvolvimento de sistemas multiagentes desde a concepção até a implementação. Quanto a plataforma JADE, sua curva de aprendizado é mais otimizada. Ela provê as estrutura necessária para a construção de sistemas multiagentes, através de uma documentação abrangente e bem exemplificada.

6.1 Trabalhos Futuros

Os testes realizados durante o presente trabalho conseguem verificar a eficácia do sistema multiagente. Porém, ainda não é possível provar se o módulo multiagente desenvolvido aumenta a produtividade em ambientes reais de inspeções de software. Dessa forma, é proposto uma validação do sistema multiagente em um ambiente real de inspeções de software. Assim, serão coletados dados que podem trazer melhorias para tornar o sistema com maior eficiência.

A modelagem de domínio feita durante a fase de requisitos preliminares configura uma coleção de objetivos em que se há a possibilidade de automatização, seja em parte ou completamente. Assim, essa modelagem pode servir de novas entradas para a adição de funcionalidades para o aprimoramento do sistema multiagente.

Além disso, o sistema multiagente pode ser considerado um primeiro passo para a construção de um sistema de suporte às inspeções de software de forma autônoma e pró-ativa. Através do desenvolvimento das habilidades dos agentes e na melhoria da capacidade cognitiva, é possível que o sistema possa identificar problemas na execução do processo. Dessa maneira, alinhando o sistema multiagente com um conjunto de ontologias específicas, seria possível a criação de um sistema que consegue se adiantar às inspeções, já selecionando artefatos, inspetores e materiais de inspeção.

Referências

BARTIÉ, A. *Garantia da qualidade de software: adquirindo maturidade organizacional*. Rio de Janeiro: Elsevier, 2002. 295 p. ISBN 978-85-352-1124-5. Citado na página 23.

BELLIFEMINE, F. L.; CAIRE, G.; GREENWOOD, D. *Developing multi-agent systems with JADE*. [S.l.]: John Wiley & Sons, 2007. Citado 2 vezes nas páginas 43 e 44.

BORDINI, R. H. et al. A survey of programming languages and platforms for multi-agent systems. *Informatica (Slovenia)*, v. 30, n. 1, p. 33–44, 2006. Citado na página 45.

BRESCIANI, P. et al. Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, Kluwer Academic Publishers, v. 8, n. 3, p. 203–236, 2004. ISSN 1387-2532. Disponível em: <http://dx.doi.org/10-.1023/B%3AAGNT.0000018806.20944.ef>. Citado 4 vezes nas páginas 39, 40, 41 e 42.

CIOLKOWSKI, M.; LAITENBERGER, O.; BIFFL, S. Software reviews, the state of the practice. *Software, IEEE*, IEEE, v. 20, n. 6, p. 46–51, 2003. Nenhuma citação no texto.

DENGER, C.; SHULL, F. A practical approach for quality-driven inspections. *Software, IEEE*, v. 24, n. 2, p. 79–86, 2007. ISSN 0740-7459. Citado na página 23.

FAGAN, M. E. Advances in software inspections. In: *IEEE Transactions on Software Engineering*. [S.l.: s.n.], 1986. v. 12, n. 7, p. 744–751. Citado na página 23.

FAGAN, M. E. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, v. 38, p. 258–287, 1999. Citado na página 23.

FIPA, F. *The Foundation for Intelligent Physical Agents*. 2013. Disponível em: <http://www.fipa.org/>. Citado 2 vezes nas páginas 38 e 39.

IEEE Standard for Software Reviews and Audits. *IEEE STD 1028-2008*, p. 1–52, 2008. Citado 7 vezes nas páginas 23, 27, 28, 30, 31, 32 e 33.

LEE, C.-S.; WANG, M.-H. Ontology-based computational intelligent multi-agent and its application to cmmi assessment. *Applied Intelligence*, Springer US, v. 30, n. 3, p. 203–219, 2009. ISSN 0924-669X. Disponível em: <http://dx.doi.org/10.1007/s10489-007-0071-1>. Citado 2 vezes nas páginas 48 e 50.

LIU, S. et al. Formal specification-based inspection for verification of programs. *Software Engineering, IEEE Transactions on*, v. 38, n. 5, p. 1100–1122, 2012. ISSN 0098-5589. Citado 2 vezes nas páginas 49 e 50.

LUCIA, A. D. et al. Improving artefact quality management in advanced artefact management system with distributed inspection. *Software, IET*, v. 5, n. 6, p. 510–527, 2011. ISSN 1751-8806. Citado 2 vezes nas páginas 49 e 50.

MISHRA, D.; MISHRA, A. Simplified software inspection process in compliance with international standards. *Computer Standards & Interfaces*, v. 31, n. 4, p. 763 – 771, 2009. ISSN 0920-5489. Disponível em: <http://www.sciencedirect.com/science/article-/pii/S0920548908001177>. Citado 2 vezes nas páginas 49 e 50.

MISHRA, D.; MISHRA, A. A global software inspection process for distributed software development. *J. UCS*, v. 18, n. 19, p. 2731–2746, 2012. Citado 2 vezes nas páginas 49 e 50.

NöDLER, J.; NEUKIRCHEN, H.; GRABOWSKI, J. A flexible framework for quality assurance of software artefacts with applications to java, uml, and ttcn-3 test specifications. In: *Proceedings of the 2009 International Conference on Software Testing Verification and Validation*. Washington, DC, USA: IEEE Computer Society, 2009. (ICST '09), p. 101–110. ISBN 978-0-7695-3601-9. Disponível em: <http://dx.doi.org/10.1109/ICST.2009.34>. Citado 2 vezes nas páginas 48 e 50.

PAULA FILHO, W. d. P. *Engenharia de software: fundamentos, métodos, e padrões*. 3. ed. Rio de Janeiro: LTC, 2009. 1248 p. ISBN 978-85-216-1650-4. Citado na página 28.

PRESSMAN, R. S. *Engenharia de Software*. 6. ed. São Paulo: MCGRAW-Hill, 2006. ISBN 85-86804-57-6. Citado na página 23.

RUSSELL, S.; NORVIG, P. *Artificial Intelligence: A Modern Approach*. 3rd. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009. ISBN 0136042597, 9780136042594. Citado 4 vezes nas páginas 24, 35, 36 e 37.

SAMPAIO, R. Estudos de revisão sistemática: Um guia para síntese criteriosa da evidência científica. SciELO Brasil, v. 11, n. 1, p. 83–89, 2007. Citado na página 47.

SILVA, J. P. S. da et al. Um sistema para inspeções de garantia da qualidade baseado em ontologias e agentes. *Revista de Informática Teórica e Aplicada*, VIII, n. 1, p. 1–18, 2013. Citado 2 vezes nas páginas 48 e 50.

SOMMERVILLE, I. *Engenharia de software*. 8. ed. São Paulo: Pearson Addison-Wesley, 2007. ISBN 978-85-88639-28-7. Citado 2 vezes nas páginas 23 e 27.

WANG, M.-H.; LEE, C.-S. An intelligent ppqa web services for cmmi assessment. *Intelligent Systems Design and Applications, International Conference on*, IEEE Computer Society, Los Alamitos, CA, USA, v. 1, p. 229–234, 2008. Citado 2 vezes nas páginas 48 e 50.

WOOLDRIDGE, M. *An introduction to multiagent systems*. Hoboken, NJ, USA: John Wiley & Sons, 2002. Citado 3 vezes nas páginas 37, 38 e 39.

WOOLDRIDGE, M.; JENNINGS, N. R. Agent theories, architectures, and languages: a survey. In: *Intelligent agents*. Berlin: Springer Berlin Heidelberg, 1995. p. 1–39. Citado na página 35.