SOFIA: Um Sistema de Suporte para as Inspeções de Software

Thiago C. Krug[1], Juliano P. Pires[2], Vitor Hugo M. dos Santos[2], João Pablo S. da Silva[2]

[1]Unversidade Federal de Santa Maria (UFSM)
Av. Roraima, 1000, Santa Maria, RS, Brasil

[2]Universidade Federal do Pampa (UNIPAMPA)
Av.Tiarajú, 810, Alegrete, RS, Brasil

{thiagockrug, jjulianopires, vitao375, jpabloss}@gmail.com

Abstract. *The inspection of software is a important practice of engineering software to detect preliminary of defects, however, its adoption still faces resistance, because it is difficult to relate the effort with the gain in the quality of the product. To support the process of inspection, it presents in this work the SOFIA, a support system for inspection of software. The inspection of software is a important practice of engineering software to detect preliminary of defects, however, its adoption still faces resistance, because it is difficult to relate the effort with the gain in the quality of the product. To support the process of inspection, it presents in this work the SOFIA, a support system for inspection of software.*

Resumo. *A inspeção de software é uma prática importante de engenharia de software para detecção preliminar de defeitos, porém, a sua adoção ainda enfrenta resistência, pois é difícil relacionar o esforço despendido com o ganho na qualidade do produto. Para dar apoio ao processo de inspeção, apresenta-se neste trabalho o SOFIA, um sistema de suporte para inspeções de software. O SOFIA provê suporte para execução de todo o ciclo de vida de uma inspeção. Além disso, ele possui agentes de software que automatizam as tarefas durante o processo de inspeção, o SOFIA é capaz de apoiar a execução de inspeções de software, indicando potencial para otimizar a realização de inspeções e, por consequência, aumentar a produtividade das equipes.*

1. Introdução

A descoberta precoce de defeitos é uma ótima estratégia para se obter softwares economicamente viáveis [Pressman 2009]. As inspeções de software, introduzidas por *Michael E. Fagan* em 1976 [Fagan 1986], têm se mostrado uma prática muito efetiva para detectar anomalias em projetos de software [Sommerville 2010]. Uma inspeção de software pode ser definida como um processo sistemático para verificar se o software satisfaz suas especificações, exibe os atributos de qualidade especificados, segue as regulamentações, padrões, orientações, planos, especificações e procedimentos, entre outros [IEEE 2008].

Apesar de seus benefícios, a adoção das inspeções de software ainda enfrenta resistência, pois é difícil relacionar o esforço despendido com o ganho na qualidade do produto [da Silva et al. 2013]. O processo de inspeção não é suficientemente adaptado para o contexto do projeto e dos membros das equipes de desenvolvimento atuais. Outro

fator que dificulta a adoção é que o investimento inicial parece ser alto, uma vez que não há ligação entre as atividades da inspeção e as atividades usuais de desenvolvimento [Denger and Shull 2007].

Ferramentas de apoio ao processo de inspeção podem contribuir para a adoção da prática. Tais ferramentas podem minimizar a sobrecarga de controle imposta pela sistematização e integrar as inspeções ao processo de desenvolvimento das equipes. Motivado por isso, apresenta-se neste trabalho o SOFIA, um sistema de suporte para inspeções de software baseado no processo da norma IEEE Std 1028-2008. O SOFIA provê suporte para execução do ciclo de vida de uma inspeção, além de ter agentes de software que automatizam as tarefas de: i) seleção e envio de artefatos para inspeção, ii) organização e distribuição de material para inspeção e iii) monitoramento das correções de anomalias.

O SOFIA é formado por um ambiente *web* desenvolvido em *JavaServer Pages* (JSP) para interação com os usuários, um módulo multiagentes desenvolvido em *Java Agent Development Framework* (JADE) para suporte automatizado na execução das inspeções, uma base de conhecimento desenvolvida em *Web Ontology Language* (OWL) para definir o processo de software e uma base de dados para persistir os resultados das inspeções. Os testes preliminares em ambiente controlado mostraram que o SOFIA é capaz de apoiar a execução de inspeções de software, indicando potencial para otimizar a realização de inspeções e, por consequência, aumentar a produtividade das equipes.

O restante deste trabalho está organizado como segue. Na section 2 é explicado como se deu o desenvolvimento do SOFIA em uma perspectiva de engenharia de software. Por fim, na section 3 são apresentadas as considerações finais e os trabalhos futuros.

2. Desenvolvimento do Sistema

O presente trabalho apresenta o SOFIA, um sistema de suporte para inspeções de software baseado no processo da norma IEEE Std 1028-2008. O SOFIA 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.

2.1. Arquitetura da Solução

Conforme apresentado na Figura 1, o SOFIA é composto por uma base de conhecimento, um ambiente *web* e um banco de dados. O módulo multiagente interage com a base de conhecimento, na qual o conhecimento necessário para auxiliar a tomada de decisões durante o processo de inspeção é computacionalmente representado. Os agentes também interagem com uma base de dados onde são registrados os dados relevantes ao processo. Os dados podem ser apresentados posteriormente pelo ambiente *web* que interage com o usuário, relatando o andamento do processo de inspeção. No momento que o ambiente *web* é inicializado, o módulo multiagente também é iniciado para que ambos possam ser utilizados pelo processo de inspeção em andamento.

O SOFIA foi construído partindo de algumas premissas. Considera-se que o projeto em execução possui um repositório sob o sistema de controle de versão Git[1], contendo

[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/.

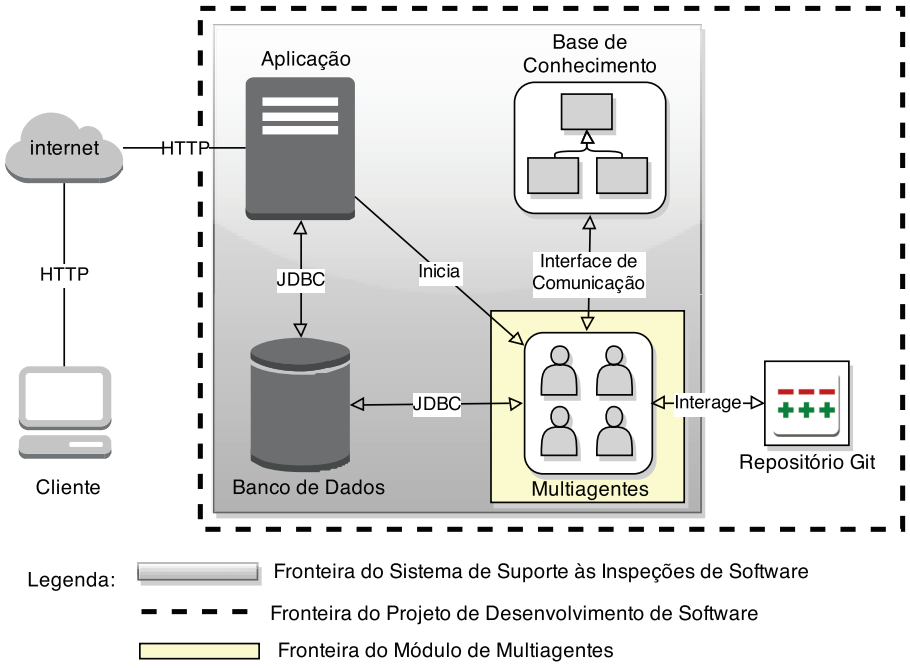

Figura 1. Fronteiras da aplicação.

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 ser populada com o conhecimento do processo de desenvolvimento em andamento e seguir as restrições do projeto.

O SOFIA tem por finalidade prover suporte para as atividades do processo de inspeção de software normatizadas pela IEEE 1028-2008, na forma de automatizar fases, etapas ou procedimentos de processos de garantia da qualidade. Na mesma perspectiva, é possível verificar uma tendência na utilização de agentes de software em conjunto com bases de conhecimento ontológicas para o suporte a tais atividades [da Silva et al. 2013] [Lee and Wang 2009] [Wang and Lee 2008]. Entretanto, mesmo em processos novos ou modificados, a necessidade de dispor de uma ferramenta que auxilie no processo é aparente [Mishra and Mishra 2009] [Lucia et al. 2011] [Liu et al. 2012] [Mishra and Mishra 2012]. A utilização de agentes nesse tipo de situação se torna adequada, pois um agente é um sistema computacional que está situado em algum ambiente onde é capaz de realizar ações autônomas neste ambiente para atingir seus objetivos [Wooldridge and Jennings 1995]. Uma definição mais simples é que um agente é qualquer coisa que consegue perceber o ambiente através de sensores e atuar neste ambiente através de atuadores [Russell and Norvig 2009].

A estrutura do módulo multiagente é composta por cinco agentes com responsabilidades distintas, estruturalmente definida como reativa simples. Apesar disso, o módulo multiagente atende a 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.

Para a realização da funcionalidade de enviar o artefato para a inspeção, o autor deve realizar o envio do artefato no ramo das inspeções. Após, o módulo 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.

Outra funcionalidade se dá quando o inspetor líder acessa o ambiente *web*, seleciona os artefatos disponíveis para a inspeção e os membros do time de inspeção, e autoriza a inspeção. Em seguida, o módulo 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 e obtém os itens de revisão pertencentes àquele artefato, realizando dessa forma a funcionalidade de reunir os materiais de inspeção. Por fim, o módulo envia um *e-mail* para cada membro do time informando a nova inspeção e fornecendo um *link* para a página em que se encontra o *checklist*. Assim, distribuindo o material de inspeção e possibilitando os inspetores analisarem o artefato de software.

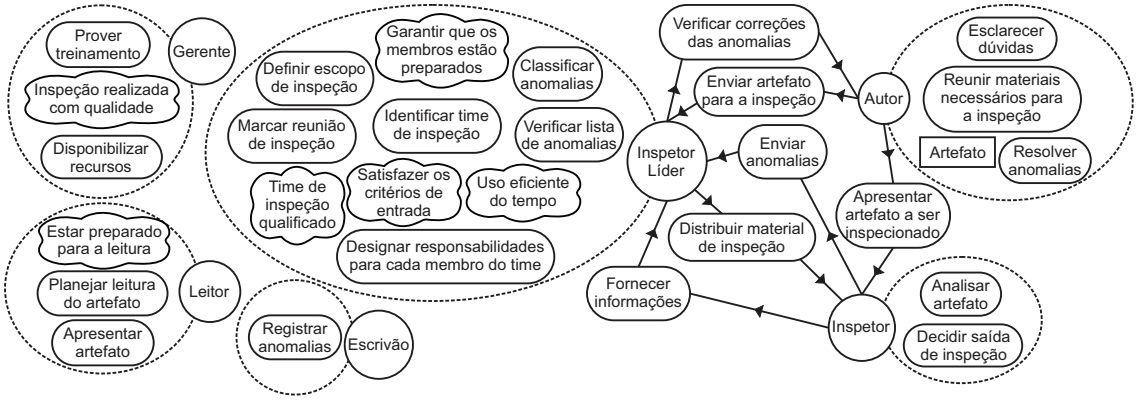

Figura 2. Modelo de dependências entre atores e seus objetivos.

A última funcionalidade trata da verificação das correções das anomalias. Por meio dela, 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. 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.

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, a qual possibilitou a obtenção do modelo de dependências entre atores e seus objetivos, como é apresentado na Figura 2.

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 refinada e incrementalmente estendida do sistema e do seu ambiente. A metodologia Tropos possui quatro fases de desenvolvimento: (i) *requisitos preliminares*, (ii) *requisitos finais*, (iii) *arquitetura do projeto*, e (iv) *projeto detalhado* [Bresciani et al. 2004]. Elas direcionaram o desenvolvimento desde sua concepção até o projeto dos agentes de software.

2.2. Implementação

A implementação de agentes de software, diferentemente de outros sistemas, requer a utilização de estruturas que possibilitem seu monitoramento e seu controle. Nesse sentido, destaca-se o *framework Java Agent Development* (JADE), que provê uma camada *middleware* de funcionalidades básicas que são independentes das especificação da aplicação e que simplificam o desenvolvimento de aplicações distribuídas que utilizam agentes de software. Além disso, a plataforma JADE provê aos programadores um sistema completamente distribuído habitado por agentes e possui compatibilidade total com as especificações da *Foundation for Intelligent Physical Agents* (FIPA) [Bellifemine et al. 2007].

Cada agente foi implementado seguindo a arquitetura proposta pela JADE, como é apresentado na Figura 3. Os agentes foram divididos por responsabilidades e cada um estende a classe *Agent* pertencente à JADE. O agente *InspectionManager* é equivalente ao agente *Gerente da Inspeção*. Já o agente *ArtifactManager* é o mesmo agente *Gerente dos Artefatos*. Ainda na Figura 3, há o agente *TeamManager*, o qual representa o agente *Gerente do time*. O agente *ScopeManager* representa o agente *Gerente do escopo*. Por fim o agente *AnomalieManager* equivale ao agente *Gerente das anomalias*.

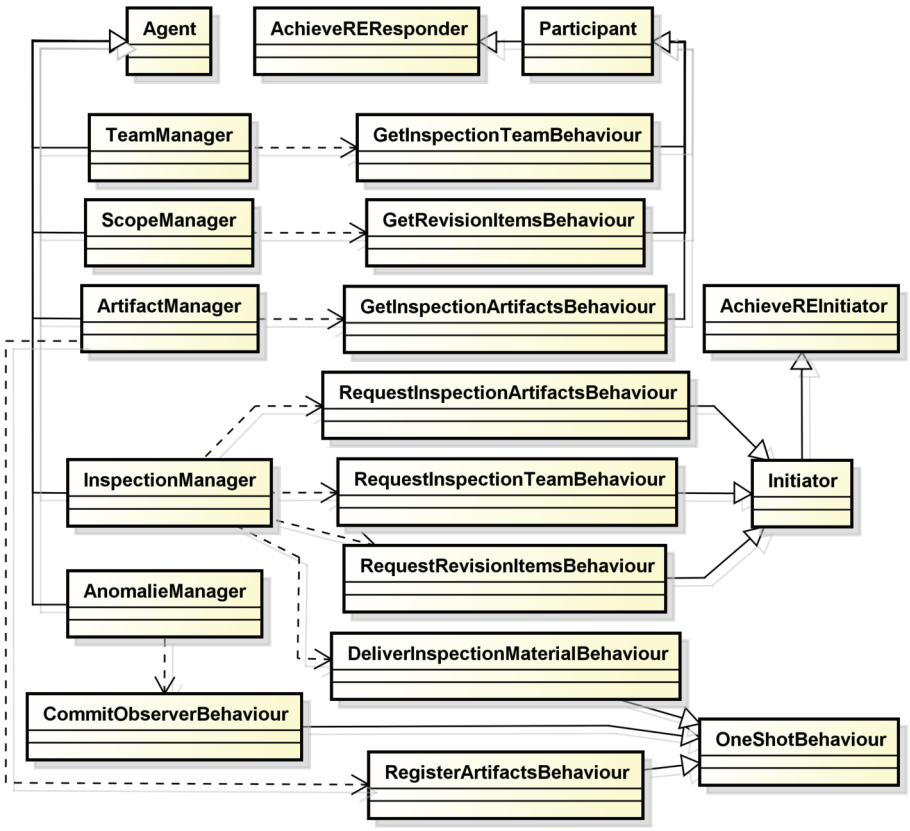

Figura 3. Diagrama de classes dos agentes.

Na Figura 3 são apresentados os comportamentos dos agentes. A classe *OneShotBehaviour*, provida pela JADE, estabelece o comportamento de atomicidade para as demais classes. A classe *CommitObserverBehaviour* implementa o comportamento do agente *Gerente das anomalias* de verificar as correções das anomalias. Já a classe *Re-*

gisterArtifactsBehaviour implementa o comportamento do agente *Gerente dos artefatos* de enviar os artefatos para a inspeção. Por último, a classe *DeliverInspectionMaterial-Behaviour* provê o comportamento de distribuir os materiais da inspeção.

Para que os sensores dos agente possam ser ativados, eles necessitam estar em um ambiente. Os ambientes 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. Porém, nenhum agente atua no ambiente Git, pois é apenas uma forma de visualização que os agentes possuem do ambiente de desenvolvimento. O banco de dados, por sua vez, sofre modificações tanto do módulo multiagente quanto pelo time de inspeção por intermédio do ambiente *web*.

2.3. Testes

Foram realizadas cinco baterias de testes para a verificação do SOFIA. Em cada bateria foram verificadas as unidades de cada classe e a integração entre os componentes dos módulos, além dos testes de comportamento e comunicação entre os agentes.

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, e o agente realizava seu comportamento e atuava modificando o ambiente. 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.

Os resultados dos casos de teste unitários, integração, comportamento e comunicação entre os agentes foram separados pelas bateria de testes. Em cada bateria foram verificadas a quantidade de falhas encontradas, a quantidade de linhas cobertas e a quantidade de caminhos ou ramos percorridos. Para isso, foi utilizado o Cobertura[2] em conjunto com o JUnit.

O gráfico apresentado na Figura 4 é referente a quantidade total de falhas encontradas durante cada bateria de testes unitários, integração, comportamento e comunicação entre os agentes. 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.

Na Figura 5 é apresentada a percentagem da cobertura das linhas e da cobertura dos ramos por bateria de testes. Houve um aumento da cobertura durante as três primeiras baterias de teste, uma redução na quarta bateria, e um novo aumentando 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 4 e a Figura 5, pode-se concluir que apesar dos casos de teste cobrirem 86% do código-fonte, esses 86% estão corretos de acordo com os resultados

[2]Ferramenta para teste de cobertura de código-fonte em Java. http://cobertura.github.io/cobertura/.

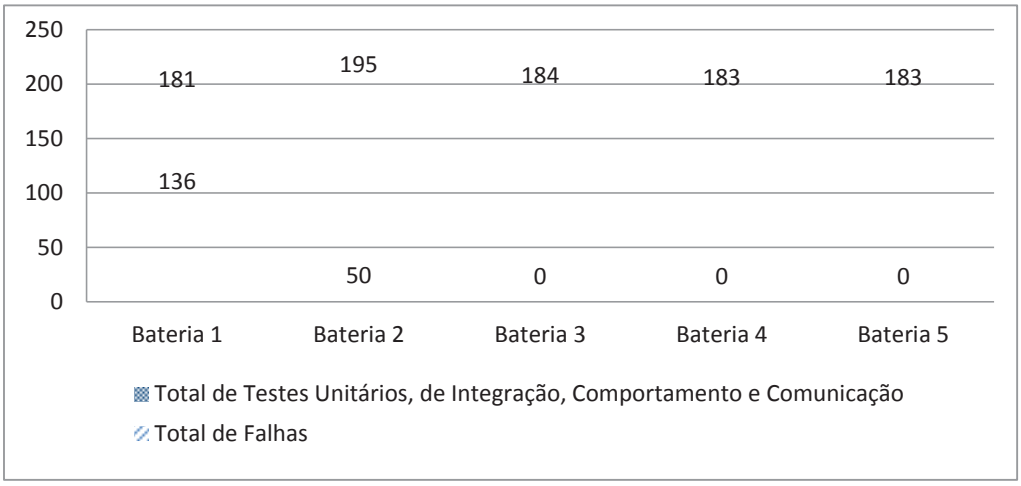

Figura 4. Gráfico de Quantidade de Falhas por Bateria de Teste.

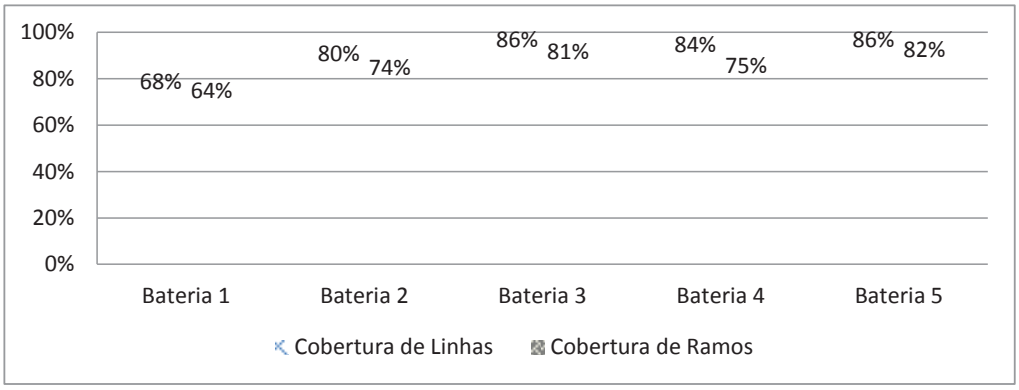

Figura 5. Gráfico de Cobertura do Código por Bateria de Teste.

esperados. Fazem parte dos 86% todos os fluxos principais e secundários do SOFIA.

3. Conclusão

O uso de ferramentas de suporte pode melhorar a relação custo/benefício nas inspeções de software. Neste sentido, apresentou-se neste trabalho o SOFIA, um sistema de suporte para inspeções de software baseado no processo da norma IEEE Std 1028-2008. O SOFIA oferece suporte a inspeções, utilizando agentes para selecionar, organizar e distribuir materiais e para monitorar correções de anomalias.

Nos testes foi possivel verificar que o sistema atende a todas a funcionalidades especificadas, mostrando assim que o sistema está apto para experimentos em um contexto real, para que o mesmo demonstre o quão útil é no contexto de inspeção de software.

A ferramenta foi aplicada somente em ambiente controlado demonstrando resultados positivos. Entretanto, em contextos reais pode ser possível encontrar adversidades não exploradas. Assim, dentre os trabalhos futuros está a aplicação da ferramenta em um contexto real e evolução das funcionalidades que apresentarem menor eficiência.

Referências

[Bellifemine et al. 2007] Bellifemine, F. L., Caire, G., and Greenwood, D. (2007). *Developing multi-agent systems with JADE*, volume 7. John Wiley & Sons.

[Bresciani et al. 2004] Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., and Mylopoulos, J. (2004). Tropos: An agent-oriented software development methodology. *Autonomous Agents and Multi-Agent Systems*, 8(3):203–236.

[da Silva et al. 2013] da Silva, J. P. S., Dall'Oglio, P., da Silva Pinto, S. C. C., and Bittencourt, I. I. (2013). Um sistema para inspeções de garantia da qualidade baseado em ontologias e agentes. *Revista de Informática Teórica e Aplicada*, VIII(1):1–18.

[Denger and Shull 2007] Denger, C. and Shull, F. (2007). A practical approach for quality-driven inspections. *IEEE Software*, 24(2):79–86.

[Fagan 1986] Fagan, M. E. (1986). Advances in software inspections. *IEEE Transactions on Software Engineering*, 12(7):744–751.

[IEEE 2008] IEEE, I. C. S. (2008). *IEEE 1028: Standard for Software Reviews and Audits*.

[Lee and Wang 2009] Lee, C.-S. and Wang, M.-H. (2009). Ontology-based computational intelligent multi-agent and its application to cmmi assessment. *Applied Intelligence*, 30(3):203–219.

[Liu et al. 2012] Liu, S., Chen, Y., Nagoya, F., and McDermid, J. A. (2012). Formal specification-based inspection for verification of programs. *IEEE Transactions on Software Engineering*, 38(5):1100–1122.

[Lucia et al. 2011] Lucia, A. D., Fasano, F., Scanniello, G., and Tortora, G. (2011). Improving artefact quality management in advanced artefact management system with distributed inspection. *IET Software*, 5(6):510–527.

[Mishra and Mishra 2009] Mishra, D. and Mishra, A. (2009). Simplified software inspection process in compliance with international standards. *Computer Standards & Interfaces*, 31(4):763–771.

[Mishra and Mishra 2012] Mishra, D. and Mishra, A. (2012). A global software inspection process for distributed software development. *Journal UCS*, 18(19):2731–2746.

[Pressman 2009] Pressman, R. S. (2009). *Software Engineering: A Proactitioner's Approach*. McGraw-Hill, New York, NY, USA, 7 edition.

[Russell and Norvig 2009] Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

[Sommerville 2010] Sommerville, I. (2010). *Softwware Engineering*. Addison-Wesley, Boston, MA, USA, 9 edition.

[Wang and Lee 2008] Wang, M.-H. and Lee, C.-S. (2008). An intelligent ppqa web services for cmmi assessment. In *International Conference on Intelligent Systems Design and Applications*, volume 1, pages 229–234.

[Wooldridge and Jennings 1995] Wooldridge, M. and Jennings, N. R. (1995). Agent theories, architectures, and languages: a survey. In *Intelligent agents*, pages 1–39. Springer Berlin Heidelberg, Berlin.