Um Sistema para Inspeções de Garantia da Qualidade Baseado em Ontologias e Agentes

João Pablo Silva da Silva[1]
Pablo Dall'Oglio[2]
Sérgio Crespo Coelho da Silva Pinto[3]
Ig Ibert Bittencourt[4]

Resumo: A implementação de práticas de garantia da qualidade possui uma relação custo/benefício difícil de balancear. Isso ocorre porque o custo de execução das práticas é direto, enquanto que, o benefício obtido com estas é indireto. No intuito de melhorar essa relação, este trabalho apresenta um sistema de suporte para as inspeções de garantia da qualidade capaz de automatizar a definição de escopo e o endereçamento de não conformidades, além de gerenciar cadastros e calcular indicadores. O sistema é formado por uma ontologia, a qual mapeia a semântica envolvida nas inspeções de garantia da qualidade; e por agentes de software, os quais implementam as regras necessárias para automação das atividades mencionadas. Os experimentos realizados em uma Fábrica de Software mostraram melhora na produtividade nas inspeções, maximizando a cobertura, sem impactar no esforço demandado.

Abstract: The implementation of quality assurance practices has a cost-benefit hard to balance. This is because the cost of implementing the practices is direct, whereas, the benefit obtained is indirect. In order to improve the cost-benefit of this practices, this work present a support system for quality assurance inspection able to automate the scope definition and non-compliance assignment, besides to manage records and to generate indicators. The system is composed by an ontology, which maps the semantic of quality assurance inspections; and by software agents, which implements the automation rules for aforementioned activities. The experiments carried out in a Software Factory showed that the system improved the productivity of inspections, without impacting the effort required.

[1]Universidade Federal do Pampa, Av. Tiarajú, 810, Alegrete/RS, Brasil
{joaosilva@unipampa.edu.br}
[2]Universidade do Vale do Rio dos Sinos, Av. Unisinos, 950, São Leopoldo/RS, Brasil
{pablo.dalloglio@gmail.com}
[3]Universidade Federal de Minas Gerais, Av. Antônio Carlos, 6627, Belo Horizonte/MG, Brasil
{crespo.sergio@gmail.com}
[4]Universidade Federal de Alagoas, Av. Lourival Melo Mota, s/n, Maceió/AL, Brasil
{ig.ibert@gmail.com}

1 Introdução

A garantia da qualidade é responsável por avaliar e reportar a efetividade e a completude das atividades dos processos de software [1]. Nesse sentido, o *Capability Maturity Model Integration* (CMMI) propõe um conjunto de práticas essenciais para a implementação da disciplina em equipes de desenvolvimento [2]. Em [3] é apresentada uma alternativa de solução para a implementação dessas práticas, a qual aborda a garantia da qualidade a partir de três frentes: i) *definição*, onde é feito o mapeamento de atributos de qualidade desejados para os produtos de trabalho; ii) *inspeção*, onde a execução dos projetos é avaliada contra a definição dos processos recorrentemente; iii) *medição*, onde indicadores são gerados como resultados das inspeções.

Os resultados apresentados em [3] mostram que a institucionalização das práticas de garantia da qualidade gera um problema gerencial, uma vez que o custo da garantia da qualidade é direto, pois é necessário alocar horas-homem para realização das inspeções; entretanto o seu benefício é indireto, pois a garantia da qualidade atua sobre o processo de software e não sobre o produto de software. Esta relação custo/benefício faz com que gerentes de projeto, em seus planejamentos, minimizem o esforço despendido em inspeções. Tal estratégia de planejamento pode impactar na efetividade das inspeções, o que expõe o projeto a riscos relacionados a tempo, custo e qualidade.

Uma forma de endereçar esse problema é aumentar a produtividade das inspeções de garantia da qualidade. Para tanto, faz-se necessário otimizar o trabalho do analista de qualidade de forma que a integralidade do esforço reservado pelo gerente de projetos seja efetivamente empreendido na execução das inspeções e não na sua inicialização e finalização. Motivado por essa questão, este trabalho apresenta um sistema de suporte para a abordagem apresentada em [3], o qual automatiza a definição de escopo e endereçamento de não conformidades em inspeções de garantia da qualidade. A arquitetura do sistema conta com um modelo ontológico, responsável por descrever a semântica envolvida; e uma estrutura de agentes, responsável por implementar a automação das atividades. Os resultados obtidos com experimentos em uma Fábrica de Software foram positivos, pois o sistema viabilizou uma melhor cobertura das inspeções, sem impactar no esforço despendido.

O restante deste trabalho está organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica necessária para o pleno entendimento da solução apresentada. A Seção 3 apresenta o estado da arte sobre o uso de ontologias e agentes em soluções orientadas a qualidade de software. A Seção 4 descreve detalhadamente a arquitetura do sistema de suporte para inspeções de garantia da qualidade. A Seção 5 relata os experimentos realizados para validar o sistema, além dos resultados obtidos. Por fim, a Seção 6 apresenta as considerações finais e os trabalhos futuros.

2 Fundamentação Teórica

Este trabalho propõe o uso de ontologias e agentes em uma solução computacional, a qual tem a garantia da qualidade como área de aplicação. No intuito de fundamentar os principais conceitos envolvidos, a Subseção 2.1 apresenta uma visão geral sobre garantia da qualidade, a Subseção 2.2 define e caracteriza as ontologias e a Subseção 2.3 estabelece conceitos sobre agentes de software.

2.1 Garantia da Qualidade

A garantia da qualidade aborda a qualidade de software a partir de uma perspectiva de processo, avaliando e reportando a efetividade e a completude das atividades de um processo de software [4]. A disciplina é definida como um meio sistemático e planejado para assegurar que o processo e seus padrões, práticas, procedimentos e métodos definidos sejam devidamente aplicados [2]. O papel da garantia da qualidade é certificar que um processo é executado como planejado, provendo para a organização um conjunto de indicadores de processo [5]. Assim, práticas de garantia da qualidade são definidas com processos organizacionais e requerem o envolvimento de terceiros para que não haja conflitos de interesse [1].

O CMMI trata a garantia da qualidade como uma área de processo de suporte. A Garantia da Qualidade de Processo e de Produto, *Product and Process Quality Assurance* (PPQA), tem como propósito prover às gerências da organização uma visão objetiva sobre a execução dos processos e produtos de trabalho relacionados. PPQA envolve a avaliação objetiva da execução dos processos e produtos de trabalho contra descrições de processo, procedimentos e padrões. As avaliações viabilizam a identificação de não conformidades, desvios na execução do processo, que possam expor o projeto a riscos relacionados a custo, tempo ou qualidade. Complementarmente, PPQA envolve a explicitação de informações sobre a aderência dos projetos aos processos definidos, provendo o embasamento necessário para a tomada de decisão por parte das gerências competentes, avaliação e melhoria contínua dos processos de software [2].

A Figura 1 apresenta uma visão geral da abordagem proposta em [3] para a implementação das práticas de PPQA. A *Definição* remete ao mapeamento de pontos críticos de um processo através do levantamento dos principais produtos de trabalho e a geração de uma coleção de itens de revisão que garantam a verificação dos fatores de sucesso para o projeto. A *Inspeção* remete à execução das inspeções em projetos através da definição de escopo, seleção e análise de amostras, e corroboração e publicação dos resultados. A *Medição* estabelece dois indicadores fundamentais para o monitoramento, tanto dos projetos de desenvolvimento, quanto das próprias atividades de garantia da qualidade. Cabe salientar que a abordagem foi validada em avaliações formais CMMI.

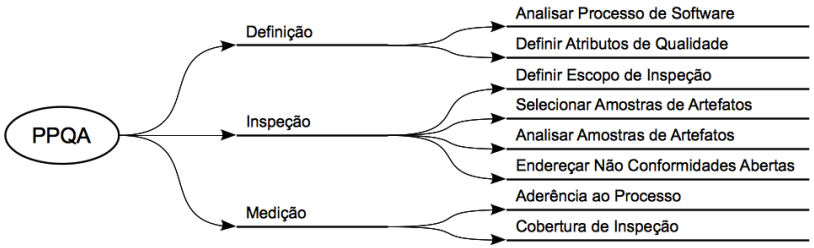

Figura 1. Uma abordagem para implementação de PPQA.

2.2 Ontologias

O termo ontologia vem de um ramo da filosofia que estuda o ser e sua existência. O termo foi inicialmente adotado na computação por pesquisadores de inteligência artificial, os quais identificaram sua aplicabilidade e construíram modelos computacionais com algum tipo de raciocínio automatizado. A partir da década de 90, as ontologias começaram a ser tratadas como parte integrante de sistemas baseados em conhecimento [6], sendo definida como uma especificação explícita de conceitualizações. Conceitualizações são abstrações, ou seja, uma visão simplificada daquilo que se quer representar por algum motivo [7].

Mais tarde, a definição de ontologias foi refinada para uma especificação explicita e formal de conceitualizações compartilhadas [8]. No contexto da ciência da computação e da ciência da informação, uma ontologia define um conjunto de primitivas representacionais de um determinado domínio de conhecimento. As primitivas representacionais são fundamentalmente classes, atributos e relacionamentos, incluindo seus significados e restrições que garantam logicamente aplicações consistentes. Atualmente, as ontologias[5] fazem parte da pilha de padrões da *Word Wide Web Consortium* (W3C) para Web Semântica [9].

A Figura 2 apresenta um método de engenharia de ontologias chamado *Ontology Development 101*. Na etapa *Determinar Domínio e Escopo* é definido o domínio de conhecimento que se quer representar e o escopo da aplicação que se deseja construir. Na etapa *Considerar Reuso de Ontologias* é feita uma busca por ontologias já definidas para reuso. Na etapa *Enumerar Termos Importantes* é feito um levantamento de termos relacionados junto aos especialistas do domínio de conhecimento. Na *Definir Classes e Hierarquias* é criada uma hierarquia de classes a partir da listagem de termos do domínio. Na etapa *Definir Propriedades de Classes* a descrição das classes é complementada com propriedades . Na etapa *Definir Restrições de Propriedades* as propriedades são refinadas com restrições para aumentar sua expressividade. Na etapa *Criar Instâncias* a ontologia é povoada e validada [10].

[5]Exemplos de ontologias podem ser obtidos em: `http://protegewiki.stanford.edu/wiki/Protege_Ontology_Library`

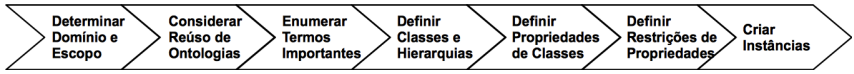

Figura 2. Um método de engenharia de ontologias.

Em uma perspectiva prática, a implementação de ontologias requer meios para descrever suas estruturas e realizar consultas em sua base de conhecimento. Nesse sentido, cabe destacar a *Web Ontology Language* (OWL) e a *SPARQL Query Language* (SPARQL), ambas, recomendações da W3C. A OWL é utilizada na formalização de um domínio através da definição de classes e de propriedades para essas classes, na definição de indivíduos e afirmações sobre as propriedades desses indivíduos, no raciocínio sobre essas classes e esses indivíduos de acordo com a semântica formal definida pela linguagem [11]. A SPARQL é uma linguagem de consulta em grafos *Resource Description Framework* (RDF) que também pode ser usada para recuperar indivíduos e suas propriedades em modelos OWL [12].

2.3 Agentes de Software

O termo agente é usado na literatura computacional para designar vários tipos de programas, os quais podem ou não possuir algum tipo de comportamento inteligente [13]. Essencialmente, um agente é algo capaz de perceber e agir sobre o ambiente em que está inserido [14]. Em uma definição mais completa, pode-se dizer que agentes são sistemas de computador situados em algum ambiente, capazes de tomar ações de forma autônoma dentro deste ambiente, a fim de atingir seus objetivos específicos [15]. Conforme visto na Figura 3 os agentes percebem o ambiente através de sensores e atuam sobre o ambiente através de atuadores.

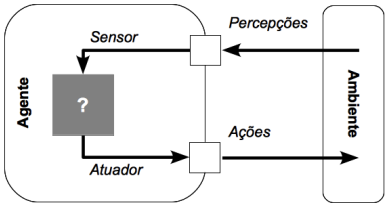

Figura 3. A interação de um agente com o seu ambiente.

Os agentes podem ser vistos como sistemas: (i) *autônomos*, agindo de forma independente, sem intervenção humana ou de outros agentes; (ii) *proativos*, exibindo comportamento dirigido a objetivos; e (iii) *sociáveis*, interagindo com outros agentes ao mesmo tempo ou em tempos diferentes [15]. Entretanto, cabe observar que nem todos agentes de software

apresentam integralmente tais propriedades, embora suas capacidades estejam diretamente ligadas à presença delas [13].

Existem quatro tipos básicos de agentes que incorporam princípios inerentes a quase todos os sistemas, sendo esses: *reativo simples*, os quais selecionam ações com base na percepção atual, ignorando o restante do histórico de percepções; *reativos baseados em modelos*, mantém estados internos dependentes do histórico de percepções, refletindo pelo menos algum dos estados observáveis a partir do estado atual; *baseados em objetivos*; o agente, além de conhecer o universo de estados, também tem informações que descrevam as situações desejáveis em função de seus objetivos; *baseados em utilidade*, uma função de utilidade mapeia um estado, ou uma sequência de estados, em um valor que descreve o grau de utilidade para o agente, assim este pode tomar suas decisões com base neste grau, buscando sempre os estados mais funcionais [14].

Diferentemente de outros sistemas computacionais, a implementação de agentes requer, em menor ou maior grau, estruturas de gerenciamento que possibilitem o seu efetivo monitoramento e controle. Nesse sentido, destaca-se a *Java Agent Development* (JADE), a qual provê uma camada *middleware* de funcionalidades básicas e independentes de aplicação que podem simplificar o desenvolvimento e a distribuição de soluções orientadas a agentes. Em JADE, os agentes são criados em contêineres específicos, os quais dispõem de todos os serviços necessários para sua execução. A plataforma também provê suporte à *Foundation for Intelligent Physical Agents* (FIPA), possibilitando a interação entre os agentes locais ou remotos [16].

3 Trabalhos Relacionados

No intuito de estabelecer o estado da arte, fez-se uma revisão sistemática da literatura que objetiva investigar como agentes e ontologias podem ser aplicados em soluções de engenharia orientadas à qualidade de software ao longo do ciclo de vida de desenvolvimento. Para tanto, realizou-se uma busca em bases[6] de publicações reconhecidas pela comunidade científica da computação. Foram selecionados artigos completos com soluções orientadas a qualidade de software que tenham como resultado o incremento da qualidade, sem decrementar a produtividade. Assim, chegou-se a uma relação de doze trabalhos relacionados apresentados subsequentemente.

O primeiro grupo de trabalhos relacionados se refere ao uso de ontologias. Em [17] é apresentada uma ontologia de teste de software baseada no SWEBOK e na ISO/IEC 9126 para dar suporte ao reuso de casos de teste. Em [18] é proposto um método de verificação

[6]ACM Digital Library (`http://dl.acm.org/`), IEEE Xplore (`http://ieeexplore.ieee.org/Xplore/guesthome.jsp`), ScienceDirect (`http://www.sciencedirect.com/`), e SpringerLink (`http://www.springerlink.com/`).

baseado em uma ontologia de requisitos para obter requisitos de software suficientes e de alta qualidade. Em [19] é apresentada uma ontologia para mapear as características de qualidade de um modelo conceitual para servir de ferramenta de apoio ao processo de modelagem. Em [20] é proposta uma abordagem de levantamento de requisitos para prover aos analistas uma base de conhecimento que apoie no processo de captura de requisitos não funcionais.

O segundo grupo de trabalhos relacionados se refere ao uso de agentes. Em [21] é proposto um agente inteligente de automação de teste em aplicações web com o objetivo de melhorar a efetividade dos processos de controle de qualidade. Em [22] é apresentado um agente de busca inteligente capaz de estabelecer uma sequência ótima de execução de testes para garantir uma cobertura técnica e economicamente viável. Em [23] é proposto agentes *Beliefs-Desires-Intentions* (BDI) para facilitar o planejamento, monitoramento, controle e execução de testes de desempenho adaptativos em serviços web. Em [24] é apresentado uma arquitetura multiagente para realizar uma análise dos atributos de qualidade de processos em ambientes distribuídos.

O terceiro grupo de trabalhos relacionados se refere ao uso de ontologias e de agentes. Em [25] é proposto um agente inteligente baseado na lógica *fuzzy* que, através de uma ontologia de PPQA, provê suporte à sumarização de relatórios de avaliações CMMI. Em [26] é apresentado o desenvolvimento de um serviço web inteligente baseado em ontologias e agentes para apoiar o processo de avaliações CMMI. Em [27] é proposto um sistema multiagente baseado em ontologias para prover suporte à sumarização de avaliações CMMI. Em [28] é apresentado um sistema multiagente para prover adaptabilidade, dinamismo e colaboração na realização de testes em sistemas baseados em serviços.

O resultado desta revisão sistemática mostra que o uso de agentes e ontologias em soluções orientadas a qualidade de software é um tema de interesse da comunidade científica relacionada à engenharia de software. Ao reorganizar os trabalhos relacionados a partir de uma perspectiva de aplicação, percebem-se dois grupos distintos de soluções. Em [22], [23], [24], [17], [18], [19], [20] e [25] são apresentadas soluções para o controle da qualidade, ou seja, certificar-se que o produto atende às suas especificações. Já em [21], [26], [27] e [28] são apresentadas soluções para a garantia da qualidade, ou seja, certificar-se que o processo foi executado corretamente.

Os trabalhos orientados à garantia da qualidade representam 33% da amostra selecionada, o que indica uma carência de alternativas de soluções para a área. Ao analisar exclusivamente os trabalhos orientados à garantia da qualidade se percebe que nenhuma das soluções tem como foco a produtividade das inspeções. No intuito de preencher essa lacuna, este trabalho apresenta uma solução computacional que, ao automatizar a definição de escopo e endereçamento de não conformidades, maximiza a cobertura das inspeções de garantia da qualidade sem impactar no esforço necessário para a execução destas.

4 Sistema de Inspeção de Garantia da Qualidade

O *System for Quality Assurance Inspection* (SystemQAI) é um sistema de suporte para as inspeções de garantia da qualidade, baseado na abordagem de implementação de PPQA apresentada em [3]. O SystemQAI objetiva aumentar a produtividade nas inspeções através da automatização das atividades de definição de escopo e endereçamento de não conformidades. Complementarmente, o sistema gerencia cadastros auxiliares e calcula os indicadores de Cobertura de Inspeção e Aderência ao Processo, também apresentados em [3]. Acredita-se que com o SystemQAI o analista de qualidade pode concentrar seus esforços na execução das inspeções, ou seja, na avaliação dos produtos de trabalho selecionados contra seus respectivos itens de revisão.

A Figura 4 apresenta uma visão geral do SystemQAI. A *Aplicação* provê uma interface para o analista de qualidade interagir com o sistema. O *AgentQAI* provê agentes de software responsáveis pela manipulação da ontologia e automação das atividades de inspeção anteriormente citadas. O *OntoQAI* estabelece um modelo conceitual responsável por representar a semântica envolvida em uma inspeção de garantia da qualidade. A seguir, a Subseção 4.1 e a Subseção 4.2 detalham a implementação da ontologia e dos agentes, respectivamente.

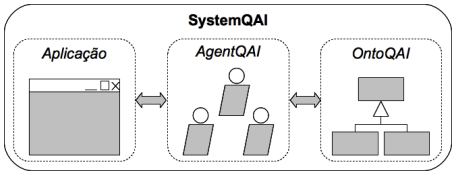

Figura 4. Uma visão geral do SystemQAI.

4.1 Ontologia de Inspeção de Garantia da Qualidade

A *Ontology for Quality Assurance Inspection* (OntoQAI) estabelece as primitivas representacionais mínimas e necessárias para descrever a semântica envolvida nas inspeções de garantia da qualidade. O modelo foi desenvolvido através da metodologia *Ontology Development 101*, sendo usadas as linguagens OWL para especificação e SPARQL para consultas, conforme já visto na Subseção 2.2. Cabe observar que a OntoQAI se difere das ontologias apresentadas na Seção 3 por mapear classes, propriedades e restrições relacionadas à execução de inspeções de garantia da qualidade.

O conhecimento necessário para modelar a OntoQAI foi obtido junto a um grupo de especialistas em engenharia de software, composto por: quatro acadêmicos do Programa

Interdisciplinar de Pós-Graduação em Computação Aplicada (PIPCA)[7]; quatro analistas de qualidade de uma Fábrica de Software sediada no TecnoSinos[8]. Primeiramente, cada integrante enumerou termos relacionados à garantia da qualidade; após, as listagens foram consolidadas para eliminar redundâncias e termos fora de escopo; por fim, o grupo validou a lista consolidada de termos. O modelo conceitual da OntoQAI foi concebido a partir dessa lista.

Em termos práticos, a garantia da qualidade avalia a execução de um projeto contra o processo definido para este. Assim, pode-se organizar o conhecimento envolvido em uma inspeção de garantia da qualidade em três grupos distintos: i) *processo de software*, o qual estabelece como o software deve ser desenvolvido, servindo de linha de base para as inspeções; ii) *projeto de software*, o qual executa o processo de software e é alvo de avaliação durante as inspeções; iii) *inspeção de software*, a qual é um instrumento usado para avaliar os produtos de trabalho do projeto contra os itens de revisão definidos para este.

Na Figura 5 é apresentado o modelo conceitual da OntoQAI. O pacote OWL contém a superclasse *Thing*[9], a qual é acrescida das propriedades de tipo dado *hasName* e *hasDescription*, já que todas as coisas podem ser nominadas e descritas. O pacote Processo de Software mapeia classes intrinsecamente ligadas à definição de processo. A classe *Process* representa o conjunto de processos que uma organização pode ter. A classe *Discipline* representa um conjunto de tarefas agrupadas por suas características essenciais. Já a classe *Phase* representa a organização sistemática das tarefas de um processo. A classe *Task* representa os procedimentos necessários para o desenvolvimento de um software, relacionando-se com papéis e produtos de trabalho. A classe *Role* representa os atores responsáveis pela execução das tarefas. A classe *WorkProduct* representa a generalização das saídas de uma tarefa, podendo ser especializada em: *Artifact*, caracterizada pela existência de um documento físico; e *Outcome*, caracterizada pela não existência de um documento físico.

O pacote Projeto de Software mapeia minimamente os conceitos necessários para executar as inspeções de garantia da qualidade. A classe *Project* representa os projetos alvo de inspeção. A classe *Resource* representa os recursos demandados pelo projeto, podendo ser especializada em: *Person*, definida como recursos humanos; e *Material*, definida como recursos materiais. O pacote Inspeção de Software mapeia conceitos diretamente ligados à execução das inspeções de garantia da qualidade. A classe *Inspection* representa o conjunto de inspeções de um projeto. A classe *QualityCriterion* representa todos os atributos de qualidade de um terminado produto de trabalho. A classe *RevisionItem* representa o escopo de uma inspeção, relacionando a inspeção com atributos de qualidade e produtos de trabalho. A classe *Finding* representa os possíveis resultados de uma inspeção, sendo estes: *NoApply*, quando o item de revisão não se aplica à inspeção em questão; *Positive*, quando o produto

[7] `http://www.unisinos.br/mestrado-e-doutorado/computacao-aplicada/apresentacao`
[8] `http://www.tecnosinos.com.br/index.php`
[9] Em OWL toda classe é uma subclasse de *Thing*.

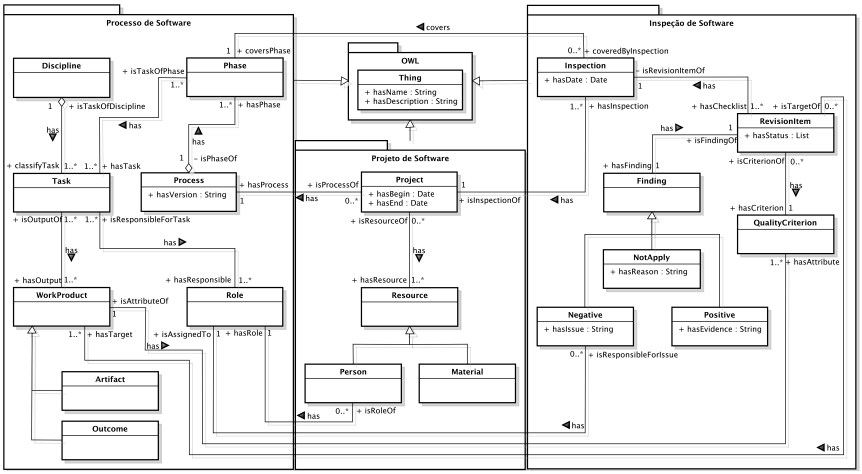

Figura 5. O modelo conceitual da OntoQAI.

de trabalho está em conformidade com o item de revisão; e *Negative*, quando o produto de trabalho não está em conformidade com o item de revisão.

Ao término do processo de modelagem conceitual, a OntoQAI foi escrita em OWL[10] e povoada com indivíduos extraídos de um processo de software real de uma Fábrica de Software com CMMI Nível de Maturidade 3. Após, a ontologia foi submetida a uma verificação de consistência através do motor de inferência *Pellet*, disponível no Protégé-OWL[11]. A Tabela 1 apresenta o resultado obtido com esse procedimento, onde podem ser observados alguns indicadores e a expressividade obtida com o modelo em questão.

Tabela 1. Indicadores e expressividade da OntoQAI.

Class Count	20
Object Property Count	32
Data Property Count	10
Individual Count	6149
DL Expressivity	ALIQ(D)

[10]Disponível para download em: `http://porteiras.s.unipampa.edu.br/lesa/files/2013/05/OntoQAI.owl_.zip`.
[11]`http://protege.stanford.edu/overview/protege-owl.html`

4.2 Agentes de Inspeção de Garantia da Qualidade

O *Agent for Quality Assurance Inspection* (AgentQAI) constitui um grupo de três agentes de software, do tipo reativo simples, que respondem a estímulos gerados na interface do SystemQAI. Os agentes são inicializados juntamente com o SystemQAI e ficam em estado de espera até que algum estímulo seja percebido. A partir desse momento, o agente atua sobre a ontologia, criando, modificando ou eliminando indivíduos, de acordo com as regras definidas para cada um. Após, os agentes voltam a um estado de espera até que um novo estimulo seja gerado. Cabe observar que não há interação entre os agentes, tendo suas existências justificadas na necessidade de segmentar a manipulação do modelo ontológico.

O modelo de casos de uso apresentado na Figura 6 sintetiza as responsabilidades de cada um dos agentes. Os agentes *Mantenedor de Processo* e *Mantenedor de Projeto* são agentes de suporte responsáveis por manter os conceitos da ontologia referentes a processo de software e projeto de software, respectivamente. O agente *Mantenedor de Inspeção* implementa as principais funcionalidades do SystemQAI, sendo também o responsável pela automação das atividades de definição de escopo e endereçamento de não conformidades.

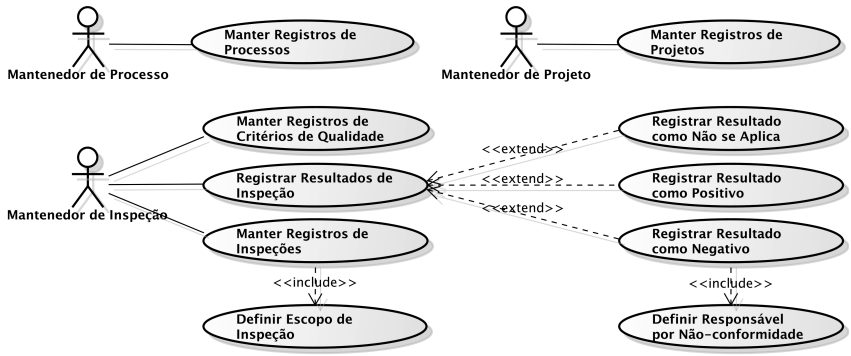

Figura 6. Definição de responsabilidades para cada agente.

O SystemQAI foi projetado segundo a arquitetura estabelecida pela plataforma JADE, conforme apresentado na Figura 7. No lado esquerdo, pode ser visto as classes que implementam os agentes, sendo todas subclasses de *Maintainer*, uma classe abstrata que sobrescreve métodos da classe *Agent*, customizando-os de acordo com as necessidades do sistema. No lado direito, pode ser visto uma classe para cada comportamento demandado pelos agentes. As classes de comportamento estendem a classe *OneShotBehaviour*, a qual permite a implementação de ações dos agentes. Basicamente, os agentes podem criar e excluir indivíduos na ontologia, *IndividualCriation* e *IndividualDeletion*, podem adicionar e configurar propriedades do tipo dado e objetos, *ObjectPropertyAddition*, *ObjectPropertySetting*, *Dat-*

aPropertySetting, e podem realizar consultas SPARQL na ontologia, *OntologyQuering*.

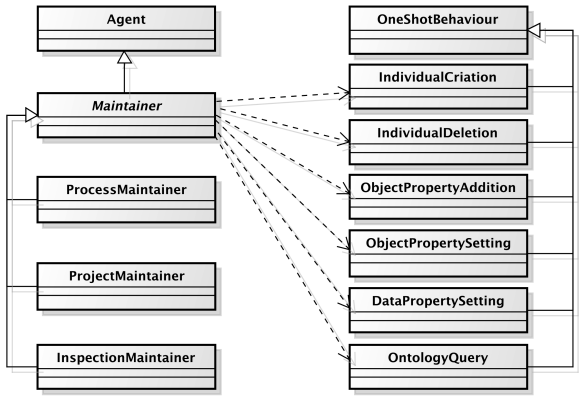

Figura 7. Modelo de classes do SystemQAI.

As classes que implementam os comportamentos dos agentes fazem uso do *framework* Jena[29] para manipular o modelo ontológico. Cada classe de comportamento foi desenvolvida de forma customizada, ou seja, fez-se uso somente das estruturas da Jena necessárias para implementar o comportamento desejado. Para ilustrar isso, a Figura 8 apresenta um diagrama de sequência que mostra a troca de mensagens necessárias para que um agente possa executar uma consulta SPARQL na OntoQAI.

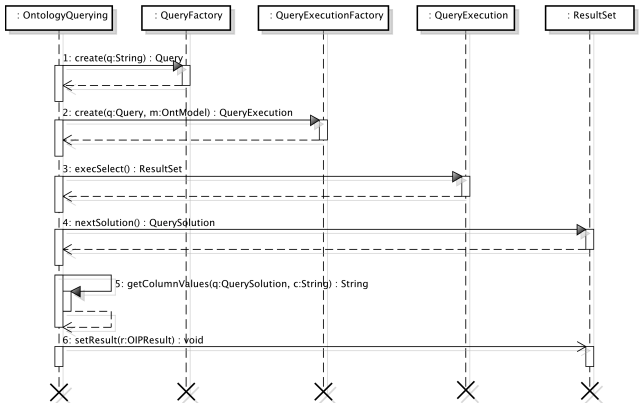

Figura 8. Diagrama de sequência para realização de consultas SPARQL.

O Algoritmo 1 descreve o procedimento realizado pelo agente *Mantenedor de In-*

speção para a definição do escopo de uma inspeção. Primeiramente, pode-se observar a criação na OntoQAI do indivíduo que representa uma determinada inspeção (linha 2). Esse indivíduo é devidamente associado aos indivíduos fase e projeto, ambos, previamente criados na OntoQAI (linhas 3 e 4). A seguir é executada uma consulta SPARQL para recuperar todos os produtos de trabalho relacionados à fase em questão (linha 5). Para cada produto de trabalho é feita uma nova consulta SPARQL para recuperar todos os atributos de qualidade definidos para este (linha 8). Para cada atributo de qualidade é criado um indivíduo que representa um item de revisão da inspeção em questão (linha 11). Esse indivíduo é associado aos indivíduos que representam a inspeção, o atributo de qualidade e o produto de trabalho (linha 12, 13 e 14). Por fim, todos os itens de revisão definidos para a inspeção em questão são apresentados para o Analista de Qualidade (linha 19).

Algoritmo 1. Pseudocódigo para definição de escopo.

```
1  INÍCIO
2    criaIndivíduo(inspeção);
3    associaIndivíduo(inspeção, fase);
4    associaIndivíduo(inspeção, projeto);
5    consultaSPARQL(produtoTrabalho, fase);
6    PARA CADA produtoTrabalho FAÇA
7      INÍCIO
8        consultaSPARQL(atributoQualidade, produtoTrabalho);
9        PARA CADA atributoQualidade FAÇA
10         INÍCIO
11           criaIndivíduo(itemRevisão)
12           associaIndivíduo(itemRevisão, inspeção);
13           associaIndivíduo(itemRevisão, atributoQualidade);
14           associaIndivíduo(itemRevisão, produtoTrabalho);
15         FIM
16       FIM PARA CADA
17     FIM
18   FIM PARA CADA
19   mostraTodos(itemRevisão, inspeção);
20 FIM
```

5 Experimentos e Resultados

O SystemQAI é um sistema de suporte para as inspeções de garantia da qualidade baseado na abordagem apresentada em [3]. Logo, entende-se que os experimentos de validação devem ser feitos em uma equipe que use a abordagem em questão, pois assim, torna-se possível verificar se há ganhos com o uso do sistema. Por esse motivo, os experimentos foram realizados em uma Fábrica de Software que usa a abordagem apresentada em [3] para atender

as práticas de PPQA[12]. A Fábrica de Software está sediada no TecnoSinos e provê serviços de desenvolvimento para o mercado nacional e internacional.

O experimento iniciou com a seleção de inspeções realizadas pela Fábrica de Software no intuito de estabelecer uma linha de base para as análises posteriores. Para tanto, foram selecionadas inspeções realizadas em 10 projetos de desenvolvimento, dos quais 5 tiveram duração inferior a seis meses e 5 tiveram duração superior a seis meses. Todos os projetos executaram o mesmo processo de desenvolvimento, o qual é organizado em quatro fases distintas: planejamento, especificação, desenvolvimento e entrega. A amostra de inspeções usada no experimento foi constituída selecionando uma inspeção de cada fase de cada projeto, o que gerou um conjunto de 40 inspeções de garantia da qualidade.

A Fábrica de Software tem definido em seu processo de desenvolvimento os indicadores de Cobertura de Inspeção, Aderência ao Processo e Variação de Esforço para o gerenciamento da garantia da qualidade. Por esse motivo, o experimento também considerou os mesmos indicadores para estabelecer as linhas de base, o que viabilizou as análises comparativas posteriores. Então, o próximo passo do experimento envolveu a coleta e a tabulação dos valores de cada um dos indicadores para cada uma das 40 inspeções selecionadas.

O SystemQAI foi instalado em um ambiente simulado. Após, o processo de desenvolvimento da Fábrica de Software foi cadastrado no sistema. A seguir, os atributos de qualidade foram cadastrados e vinculados com os produtos de trabalho definidos juntamente com o processo de desenvolvimento. Os atributos de qualidade foram obtidos junto aos *ckecklists* de inspeção usados pelos analistas de qualidade da Fábrica de Software. Por fim, os 10 projetos foram também cadastrados no sistema, já que esses são alvo das inspeções realizadas subsequentemente. Após as configurações iniciais, as 40 inspeções selecionadas foram realizadas através do SystemQAI, sendo obtidos novos valores para os indicadores.

O gráfico da Figura 9 mostra um aumento de cobertura em todas as fases do processo de desenvolvimento, tendo a menor variação na fase de Planejamento, 39%, e a maior variação na fase de Especificação, 81%. A análise do comportamento do indicador de Cobertura de Inspeção mostra que, para o cenário deste experimento, o SistemQAI foi capaz de estabelecer escopos de inspeções mais abrangentes, o que aumenta a confiabilidade do resultado das inspeções de garantia da qualidade.

O gráfico da Figura 10 mostra valores próximos de aderência em todas as fases do processo de desenvolvimento, tendo a menor variação na fase de Planejamento, -0,52%, e a maior na fase de Especificação, 1,69%. A análise do comportamento do indicador de Aderência ao Processo mostra que, para o cenário deste experimento, SystemQAI é capaz de gerar resultados similares, onde as diferenças entre percentuais se caracterizam como um ajuste

[12]Em 2007 a Fábrica de Software obteve CMMI Nível de Maturidade 2 e em 2009 obteve o CMMI Nível de Maturidade 3.

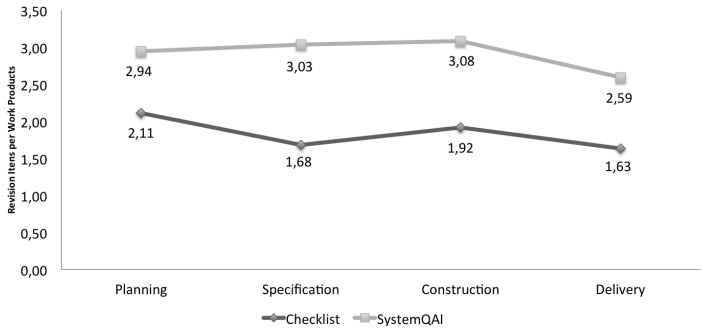

Figura 9. Gráfico comparativo entre valores de Cobertura de Inspeção.

fino e não como um reenquadramento de aderência.

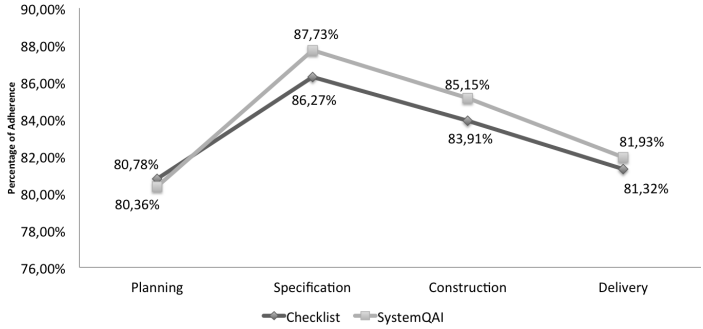

Figura 10. Gráfico comparativo entre valores de Aderência ao Processo.

O último indicador analisado para o contexto deste experimento foi o de Variação de Esforço, obtido a partir da comparação do esforço despendido em inspeções baseadas em Checklist com o esforço despendido em inspeções baseadas no SystemQAI. O indicador apresentou uma variação de -1,89%, o que indica uma pequena redução de esforço despendido para realizar cada inspeção. A análise do comportamento do indicador de Variação de Esforço mostra que, para o cenário deste experimento, o SystemQAI é capaz de melhorar a produtividade das inspeções de garantia da qualidade.

6 Conclusões

Este trabalho tem por objetivo maximizar a produtividade das inspeções de garantia da qualidade. Para tanto, foi desenvolvido um sistema de suporte para a abordagem de implementação de PPQA apresentada em [3]. O SystemQAI conta com um modelo ontológico, responsável por descrever a semântica envolvida; e uma estrutura de agentes, responsável por implementar a automação das atividades. A análise dos resultados obtidos em experimentos realizados em uma Fábrica de Software mostrou que, no cenário de teste apresentado, o sistema foi capaz de maximizar a cobertura das inspeções sem impactar expressivamente no esforço despendido na execução das inspeções.

Complementarmente, pode-se destacar como contribuição deste trabalho a definição de uma ontologia para explicitar o conhecimento relacionado às inspeções de garantia da qualidade. Como visto na Seção 3 há uma carência em modelos semânticos orientados a implementação de práticas de PPQA. Por questões de escopo de aplicação, a OntoQAI mapeou os conceitos considerados mínimos e necessários para o domínio de inspeções de garantia da qualidade. Assim, propõe-se como extensão deste trabalho o refinamento da ontologia para aumentar a sua expressividade.

Os agentes de software desenvolvidos neste trabalho têm propósitos bem definidos, ou seja, cada um encapsula uma parte do conhecimento envolvido e implementa as regras necessárias para automatizar a definição de escopo e atribuição de não conformidades. Nesse ponto, pode-se observar outra oportunidade de extensão, pois a exploração das capacidades sociais dos agentes de software pode aumentar o valor agregado do sistema. Por fim, também se destaca como trabalho futuro a realização de experimentos em outros ambientes de desenvolvimento, para assim, certificar os resultados obtidos até o momento.

Referências

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

[2] Mary Beth Chrissis, Michael D. Konrad, and Sandra Shrum. *CMMI for Development: Guidelines for Process Integration and Product Improvement*. SEI Series in Software Engineering. Addison-Wesley Professional, Boston, MA, USA, 3 edition, 2011.

[3] João Pablo S. da Silva, Pablo Dall'Oglio, Sergio Crespo, and João Carlos Gluz. Uma abordagem prática para implementação da garantia da qualidade de processo e de produto. In *VI Simpósio Brasileiro de Sistemas de Informação*. Sociedade Brasileira de Computação, 2010.

[4] Ian Sommerville. *Software Engineering*. Addison Wesley, 9 edition, 2010.

[5] IEEE. *Guide to the Software Engineering Body of Knowledge*. IEEE Press, Piscataway, NJ, USA, 2004.

[6] Giancarlo Guizzardi. *Ontological foundations for structural conceptual models*. PhD thesis, University of Twente, Enschede, 2005.

[7] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[8] Willem Nico Borst. *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*. PhD thesis, University of Twente, Enschede, 1997.

[9] Tom Gruber. Ontologies. In Ling Liu and M. Tamer Özsu, editors, *Encyclopedia of Database Systems*, pages 1959–1959. Springer US, 2009.

[10] Natalya F. Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Knowledge Systems Laboratory, 2001.

[11] Deborah L. McGuinness and Frank Van Harmelen. Owl web ontology language overview. Available at: http://www.w3.org/TR/owl-features/, 2004.

[12] Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. Available at: http://www.w3.org/TR/rdf-sparql-query/, 2008.

[13] Anita Maria da Rocha Fernandes. *Inteligência Artificial: Noções Gerais*. Visual Books, Florianópoles, SC, 2005.

[14] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Moder Approach*. Prentice Hall, Upper Saddle River, NJ, USA, 2004.

[15] Michael Wooldridge. *An Introdution to MultiAgent Systems*. Jonh Wiley and Sons, Hoboken, NJ, USA, 2002.

[16] Fabio Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, Hoboken, NJ, USA, 2007.

[17] Lizhi Cai, Weiqin Tong, Zhenyu Liu, and Juan Zhang. Test case reuse based on ontology. In *15th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 103–108, 2009.

[18] Dang Viet Dzung and A. Ohnishi. Improvement of quality of software requirements with requirements ontology. In *9th International Conference on Quality Software*, pages 284–289, 2009.

[19] Qi Yu-dong, Qu Ning, and Xie Xiao-fang. Towards a preliminary ontology for conceptual model quality evaluating. In *International Conference on Web Information Systems and Mining*, volume 1, pages 329–334, 2010.

[20] Ting Wang, Yuanjie Si, Xiao Xuan, Xinyu Wang, Xiaohu Yang, Shanping Li, and Aleksander J. Kavs. A qos ontology cooperated with feature models for non-functional requirements elicitation. In *Proceedings of the Second Asia-Pacific Symposium on Internetware*, pages 17:1–17:4, New York, NY, USA, 2010.

[21] Lei Xu and Baowen Xu. Applying agent into intelligent web application testing. In *International Conference on Cyberworlds*, pages 61–65, 2007.

[22] D. J. Mala and V. Mohan. Intelligentester - software test sequence optimization using graph based intelligent search agent. In *International Conference on Conference on Computational Intelligence and Multimedia Applications*, volume 1, pages 22–27, 2007.

[23] Bo Ma, Bin Chen, Xiaoying Bai, and Junfei Huang. Design of bdi agent for adaptive performance testing of web services. In *10th International Conference on Quality Software*, pages 435–440, 2010.

[24] S. Bukhari and F. Arif. Analyzing effects of multi agent's technology towards software quality assurance and quality engineering. In *International Conference on Information and Emerging Technologies*, pages 1–8, 2010.

[25] Mei-Hui Wang and Chang-Shing Lee. An intelligent fuzzy agent based on ppqa ontology for supporting cmmi assessment. In *IEEE International Fuzzy Systems Conference*, pages 1–6, 2007.

[26] Mei-Hui Wang and Chang-Shing Lee. An intelligent ppqa web services for cmmi assessment. In *Eighth International Conference on Intelligent Systems Design and Applications*, volume 1, pages 229–234, 2008.

[27] Chang-Shing Lee and Mei-Hui Wang. Ontology-based computational intelligent multiagent and its application to cmmi assessment. *Applied Intelligence*, 30:203–219, 2009.

[28] Xiaoying Bai, Bin Chen, Bo Ma, and Yunzhan Gong. Design of intelligent agents for collaborative testing of service-based systems. In *Proceedings of the 6th International Workshop on Automation of Software Test*, AST '11, pages 22–28, New York, NY, USA, 2011. ACM.

[29] Apache Software Foundation. Jena: A semantic web framework. Available at: http://jena.apache.org/, 2012.