UNIVERSIDADE FEDERAL DE SANTA CATARINA

OntoScrum:
Uma proposta de uso de ontologias e inferências na gestão de projetos Scrum

Anna Clara Oriques

Florianópolis – SC
2013/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA

DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

CURSO DE SISTEMAS DE INFORMAÇÃO

OntoScrum

Uma proposta de uso de ontologias e inferências na gestão de projetos Scrum

Anna Clara Oriques

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Florianópolis – SC

2013/1

Anna Clara Oriques

OntoScrum

Uma proposta de uso de ontologias e inferências na gestão de projetos Scrum

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Prof. Dr. João Cândido Dovicchi

Co-orientador: Marcos Henrique Santos

Banca examinadora

_____	_____
Profa. Dra. Patricia Della Méa Plentz	Prof. Dr. Luiz Fernando Jacintho Maia

Dedicatória

Dedico este trabalho à Vó Laia.

Agradecimentos

Ao Marcos, pelo inestimável apoio e amizade.

Ao Professor Dovicchi, pelas orientações, e principalmente pelas desorientações.

Aos Nerds da turma 041 por todos os anos de convivência, amizade e colaboração.

À Aquarela, por ser o ambiente inspirador que faltava.

A todos os que sempre que me encontravam perguntavam "E o TCC, como anda?". Vocês me irritavam, mas agora eu agradeço.

À minha família.

Ao Berners-Lee, ao Aristóteles, ao Porfírio. E ao Turing, claro.

Sumário

Lista de Figuras

Lista de Quadros

Resumo

Neste trabalho busca-se demonstrar o uso de ontologias e inferências na gestão de projetos que utilizam a metodologia Scrum. Para tal, foi especificada uma ontologia de aplicação e instanciados indivíduos que simulam seu uso real. Por fim, foram realizadas inferências com o intuito de descobrir as atividades similares, bem como o tempo médio destas, e assim facilitar a estimação do tempo necessário para cumprir tarefas em projetos seguintes, assim demonstrando a utilidade da abordagem proposta.

Palavras-chave: Ontologia, Scrum, Web semântica, Inferência.

Abstract

This project aims to demonstrate the use of ontologies and inferences in managing projects that follow the Scrum method. In order to achieve this, an application ontology was specified, and individuals were created to simulate its use in real projects. Then, inferences were made to discover similar activities and their average durations, thus making it easier to estimate the time needed to execute tasks in future projects, therefore demonstrating the usefulness of the proposed aproach.

Keywords: Ontology, Scrum, Semantic Web, Inference.

1 Introdução

1.1 O problema

Projetos de desenvolvimento de software, com frequência, não atingem o objetivo desejado ou mesmo são cancelados porque ultrapassam o tempo ou orçamento inicialmente estimados (DIPPELREITER, 2008). Este problema ocorre tanto em projetos de escopo fechado quanto naqueles de escopo aberto.

A modadelidade de projetos denominda de escopo fechado ou fixo, pode-se entender como aquela cujas funcionalidades e requisitos do projeto são previamente acordados, e então é determinado um prazo e orçamento. Por outro lado, projetos de escopo aberto (hora técnica) são remunerados pelas horas executadas pelos profissionais envolvidos no projeto.

Em um estudo do Standish Group repetido ao longo dos últimos vinte anos, projetos de desenvolvimento de software foram classificados em três tipos de resolução:

- Tipo de resolução 1, ou projeto bem sucedido: O projeto é concluído dentro do prazo e orçamento, com todas as características e funcionalidades especificadas inicialmente.

- Tipo de resolução 2, ou projeto em dificuldades[1]: O projeto é concluído e está funcional em ambiente de produção, porém ultrapassa o orçamento, o cronograma, e não apresenta todas as características e funcionaldades da especificação original.

- Tipo de resolução 3, ou projeto fracassado[2]: O projeto é cancelado em algum ponto durante o ciclo de desenvolvimento.

1 Neste trabalho o termo projeto fracassado foi decorrente de tradução nossa do termo original em inglês *challenged*
2 Neste trabalho o termo projeto fracassado foi decorrente de tradução nossa do termo original em inglês *impaired*

Table I

Standish project benchmarks over the years

Year	Successful (%)	Challenged (%)	Failed (%)
1994	16	53	31
1996	27	33	40
1998	26	46	28
2000	28	49	23
2004	29	53	18
2006	35	46	19
2009	32	44	24

Figura 1: Tipos de resolução em projetos de TI. Fonte: (EVELEENS; VERHOEF, 2010)

Segundo estudos de Keaveney e Conboy (2006) cerca de 24% dos projetos de TI são cancelados ou simplesmente abandonados e 44% custam quase o dobro da estimativa inicial.

Ainda na visão dos mesmos autores, erros em estimativas podem tanto ocorrer tanto por falta de formalidade no processo de estimação, quanto por má qualidade dos dados de projetos passados (KEAVENEY; CONBOY, 2006).

Ao seu tempo, a imprecisão em estimativas pode ser causada pela falta de procedimentos e políticas sobre como lidar com falhas e evitar repetir erros aprendendo com experiências passadas (EWUSI-MENSAH; PRZASNYSKI, 1995).

1.2 Motivação

Pela observação do grande número de projetos que falham em suas estimativas, bem como em seus valores orçados, a comunidade de desenvolvimento vêm cada vez mais usando alternativas para os métodos de gestão de projetos. Dentre estas alternativas se destaca-se o *Scrum*.

Nesta metodologia de gestão, a duração das tarefas dos projetos é estimada durante as reuniões de especificação e planejamento. Então após o delineamento dos atividades, um *Sprint* é definido e iniciado.

Um *Sprint*, trata-se de conjunto de atividades levadas a cabo pela equipe de desenvolvimento com o objetivo de alcançar um determinado conjunto de funcionalidades de uma dada aplicação (programa de computador). Ao fim de cada *Sprint*, em alguns casos, estas estimativas podem ser confirmadas, em outros, podem se mostrar equivocadas.

Na busca da implementação das funcionalidades de um determinado *Sprint*, um projeto pode ter tarefas que são muito próximas, ou até mesmo idênticas à outros *Sprints* ou projetos, tanto dentro da equipe quanto em outras equipes ou organizações.

Assim, a principal motivação deste trabalho, é buscar formas de prover um melhor reuso do conhecimento legado de projetos e *Sprints* já executados para melhor estimar novos *Sprints* e projetos por meio do emprego de técnicas, processos e tecnologias advindas da Web Semântica.

Assim, considerando que estimativas e resultados anteriores podem ser entendidos como métricas, ou seja dados dentro de um determinado contexto, e por sua vez a Web semântica busca atribuir significado aos dados, podemos utilizar o arcabouço definicional da Web Semântica, e assim abrir uma nova fronteira de possibilidades na busca de melhores estimativas em nossos projetos por meio de seus recursos.

A proposta deste trabalho tem como centro o uso de ontologias no mapeamento dos elementos, processsos e demais artefatos do *Scrum*, e ao seu tempo, o uso de inferência como ferramenta de estimatição assistida.

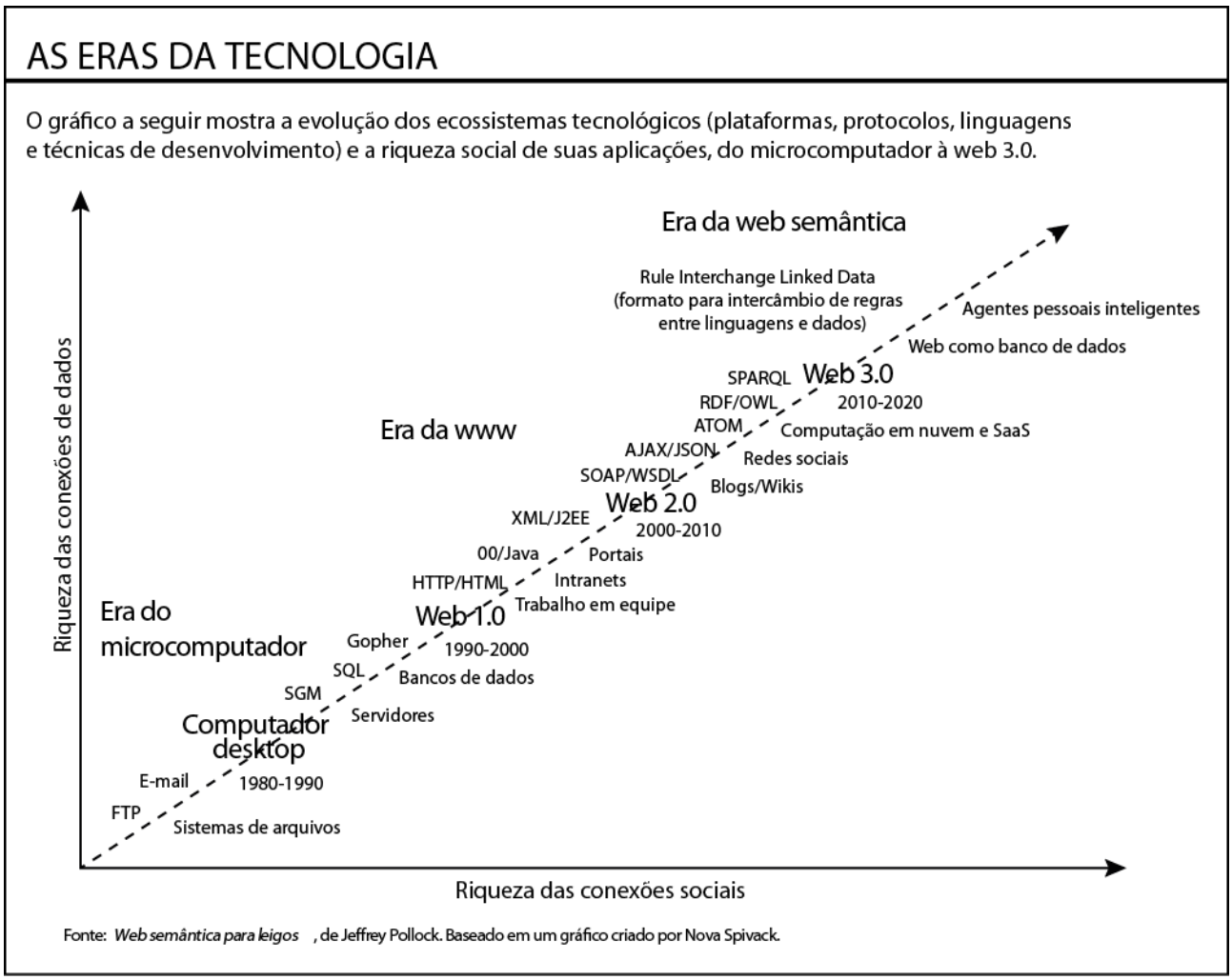


Figura 2: Evolução da Web. Fonte: (LAFUENTE, 2011)

Após serem alimentadas com dados de sucessivos projetos, estas ontologias podem ser úteis para melhorar o processo de estimativas, através de mecanismos de inferência computacional sobre os dados armazenados.

1.3 Objetivos

1.3.1 Objetivo Geral

Criar uma ontologia que apoie a formalização, execução e estimativas de projetos gerenciados pela metodologia Scrum.

1.3.2 Objetivos Específicos

Para a consecução deste trabalho, são estabelecidos os seguintes objetivos específicos:

i) Modelar uma ontologia de aplicação para apoio à execução de projetos gerenciados utilizando-se a metodologia *Scrum*;

ii) Instanciar a ontologia de aplicação com dados de projetos fictícios; e

iii) Executar inferências na base de conhecimento gerada pelas instâncias na geração de indicadores do projeto.

1.4 Delimitação de escopo

Não faz parte do escopo deste trabalho:

i) Esgotar os conceitos acerca do *Scrum* na ontologia de aplicação;

ii) Instanciar um projeto real na ontologia criada;

iii) Criar interfaces de usuário para a população da ontologia de aplicação criada; e

iv) Detalhar exaustivamente as classificações de *Backlog Items* e *Activities*.

Este trabalho se divide em:

i) Fundamentação teórica, abordando os conceitos do *Scrum*, Web Semântica, Ontologias e Inferência;

ii) A OntoScrum, explicando a ontologia criada; e

iii) Prova de conceito, demonstrando a factibilidade da abordagem proposta.

2 Fundamentação Teórica

2.1 Scrum

O *Scrum* pertence à família de métodos ágeis de desenvolvimento de software, assim como *Extreme Programming*. Entretanto, enquanto *Extreme Programming* foca na programação, com práticas como *pair programming*, desenvolvimento orientado a testes e integração contínua, Scrum se concentra no gerenciamento de projetos de software.

O Scrum advém da premissa que o desenvolvimento de software é complexo e imprevisível demais para ser planejado com exatidão com antecedência (MAHNIC; DRNOVSCEK, 2005). Ao invés disso, um controle empírico de processos deve ser aplicado para assegurar visibilidade, inspeção e adaptação. Por sua vez, as diferentes variáveis técnicas e de ambiente, como tempo, qualidade, recursos e tecnologia devem ser controladas constantemente para tornar possível uma adaptação flexível à mudanças. Isto é alcançado através de um processo de desenvolvimento iterativo e incremental.

Os conceitos explicados a seguir em sua maioria advém dos *Scrum Guides* de 2009 e 2011. Em 2011, Ken Schwaber e Jeff Sutherland resolveram simplificar o *Scrum Guide* para conter apenas o que eles consideram as regras do *Scrum*. Ou seja, foram retiradas do documento todas as coisas que não são obrigatórias à prática do *Scrum*, como dicas, práticas e técnicas opcionais. Essa mudança foi justificada como uma tentativa de diminuir os erros de interpretação acerca do que é ou não Scrum.(AGUAS, 2011). Trechos que não foram retirados dos *Scrum Guides* estão com as devidas citações.

2.1.1 Os pilares do Scrum

Esta metodologia é fundamentada na teoria empírica de controle de processos, e emprega uma abordagem iterativa e incremental para maximizar a previsibilidade e o controle de riscos

(SCHWABER; SUTHERLAND, 2011) . Três pilares sustentam toda implementação de controle de processos empíricos: a transparência, a inspeção e a adaptação (SCHWABER, 2004). Cada um dos pilares é explicado em detalhes a seguir.

2.1.1.1 Transparência

A transparência assegura que aspectos do processo que afetam o resultados devem ser visíveis à quem controla o processo. Estes aspectos devem não apenas ser visíveis, mas também o que é visto deve ser verdadeiro. Não há espaço para ilusões no controle empírico de processos. Quando alguém, por exemplo, diz que certa funcionalidade está pronta, o que isso significa? No universo do desenvolvimento de software, dizer que uma funcionalidade está pronta pode levar alguém a assumir que o código está limpo, bem estruturado, seguindo *Design Patterns*, refatorado, com testes unitários, implantado, e bem sucedido nos testes de aceitação. Porém é possível que para alguém isto signifique simplesmente que o código foi compilado. O simples fato de ser visível que a funcionalidade está pronta não tem importância se os envolvidos não compartilham o mesmo significado da palavra “pronto” (SCHWABER, 2004).

Portanto, quando o responsável por inspecionar um processo acredita que algo está pronto, isto deve ser equivalente à definição de pronto estabelecida.

2.1.1.2 Inspeção

Os vários aspectos do processo devem ser inspecionados com uma frequência suficiente para que variações inaceitáveis do processo possam ser detectadas. A frequência da inspeção deve levar em consideração que qualquer processo é modificado pelo próprio ato da inspeção. Curiosamente, a frequência de inspeção necessária frequentemente excede a tolerância do processo à inspeção. Felizmente, isto não é comum em desenvolvimento de software. O outro fator em inspeção é o inspetor, que deve possuir a habilidade de avaliar o que inspeciona.(SCHWABER, 2004)

2.1.1.3 Adaptação

Se o inspetor determinar, a partir da inspeção, que um ou mais aspectos do processo estão fora dos limites aceitáveis e que o produto resultante será inaceitável, ele deverá ajustar o processo ou o material sendo processado. Esse ajuste deve ser feito o mais rápido possível para minimizar desvios posteriores.

Existem três pontos para inspeção e adaptação no *Scrum*.

1. A *Daily Scrum* é utilizada para inspecionar o progresso em direção à *Sprint Goal* e para realizar adaptações que otimizem o valor do próximo dia de trabalho.

2. Além disso, as reuniões *Sprint Review Meeting* e *Sprint Planning Meeting* são utilizadas para inspecionar o progresso em direção à *Sprint Goal* para entrega e para fazer as adaptações que otimizem o valor da próxima *Sprint*.

3. Finalmente, a *Sprint Retrospective* é utilizada para revisar a *Sprint* passada e definir que adaptações tornarão a próxima *Sprint* mais produtiva, recompensadora e gratificante.

2.1.2 Papéis do Scrum

A metodologia *Scrum* trata os papéis a serem desempenhados no decorrer do projeto como sendo apenas três: *Scrum Master, Development Team* e *Product Owner.*

Figura 3: Galinhas e porcos. Fonte: implementingscrum.com

De acordo com o *Scrum Guide* de Maio de 2009, Os membros do *Scrum Team* são chamados “porcos”. Qualquer outra pessoa é chamada de “galinha”. “Galinhas” não podem dizer aos “porcos” como eles devem fazer seu trabalho. Esta nomenclatura vêm da seguinte história:

Logo, podemos considerar porcos o *Product Owner, Scrum Master* e o *Development Team*, que estão comprometidos com o projeto e processo *Scrum*, construindo software regularmente e frequentemente.

E temos como galinhas os usuários, *stakeholders*, consultores e qualquer um que esteja envolvido, engajado e interessado no projeto. Eles não são parte do processo *Scrum*. Suas ideias, desejos e necessidades são levados em consideração, mas de maneira alguma devem afetar ou distorcer o projeto *Scrum*.

2.1.2.1 Scrum Master

O *Scrum Master* é responsável por garantir que toda a equipe envolvida no projeto esteja aderindo aos valores do *Scrum*, às práticas e às regras, além de orientar oferecendo treinamento que leve a uma maior produtividade, e ao desenvolvimento de produtos de maior qualidade.

O *Scrum Master* garante a funcionalidade e produtividade da equipe, removendo impedimentos e protegendo seus membros de interferências externas.

É responsabilidade do *Scrum Master* ajudar a equipe envolvida no projeto a entender e usar auto-gerenciamento e interdisciplinaridade. No entanto, o *Scrum Master* não gerencia a equipe. A equipe é auto-organizável.

Também fica sob sua responsabilidade assegurar que os processos são seguidos. É ele que convida a equipe para as reuniões.

Uma mesma pessoa pode representar os papéis de *Scrum Master* e membro do *Development*

Team. No entanto, isso frequentemente leva a conflitos quando o *Scrum Master* deve escolher entre remover impedimentos e realizar tarefas. No entanto, o *Scrum Master* e o *Product Owner* nunca podem ser a mesma pessoa.

2.1.2.2 Development Team

Development Teams transformam o *Product Backlog* em incrementos de funcionalidades potencialmente entregáveis em cada *Sprint*.

Os porcos são responsáveis por selecionar o *Sprint Goal* e especificar os resultados.

Os membros do *Development Team* frequentemente possuem conhecimentos especializados, como programação, controle de qualidade, análise de negócios, arquitetura, projeto de interface de usuário ou projeto de banco de dados. No entanto, os conhecimentos que os membros do *Development Team* devem compartilhar - isto é, a habilidade de pegar um requisito e transformá-lo em um produto utilizável - tendem a ser mais importantes do que aqueles que eles não dividem. As pessoas que se recusam a programar porque são arquitetas ou designers não se adaptam bem a *Development Teams*. Todos contribuem, mesmo que isso exija aprender novas habilidades ou lembrar-se de antigas. Não há títulos em *Development Teams*, e não há exceções a essa regra.

Não existem sub-times dedicados a áreas particulares, como testes ou análise de negócios.

Times são auto-organizáveis, e tem autonomia para fazer qualquer coisa dentro dos limites do projeto para que o *Sprint Goal* seja alcançado. Nem mesmo o *Scrum Master* pode dizer ao *Development Team* como transformar o *ProductBacklog* em incrementos entregáveis de funcionalidades. É responsabilidade do time organizar seu trabalho.

O tamanho ideal de um *Development Team* é entre cinco e nove integrantes. Equipes muito grandes tem pouca interação, o que gera menor produtividade. Alem disso, uma quantidade pequena de colaboradores pode ocasionar em momentos em que o time encontra limitações de conhecimento

durante um *Sprint* que prejudica a entrega de todas as funcionalidades prometidas para o final do *Sprint*. Em *Development Teams* muito grandes, há simplesmente a necessidade de muita coordenação. Equipes com mais do que nove membros geram muita complexidade para que um processo empírico consiga gerenciar. No entanto, é possível encontrar *Development Teams* bem-sucedidos que excederam os limites superior e inferior dessa faixa de tamanhos.

O *Product Owner* e o *Scrum Master* não estão incluídos nessa conta, a menos que também sejam porcos.

A composição do *Development Team* pode mudar a cada *Sprint*. Porém, isto deve ser evitado sempre que possível, pois estas mudanças costumam reduzir a produtividade que é ganha através da auto-organização.

2.1.2.3 Product Owner

O *Product Owner* é uma pessoa responsável por garantir o valor do trabalho realizado pelo *Development Team*.

Ele gerencia o *Product Backlog*, o que lhe dá a responsabilidade de definir as funcionalidades do produto. É ele quem decide as datas de lançamento. e garante que todos tenham acesso à ele. Todos sabem quais itens têm a maior prioridade, de forma que todos sabem em que se irá trabalhar.

Uma responsabilidade muito importante do *Product Owner* é a de garantir que o produto desenvolvido trará benefícios à empresa. Ele precisa garantir a rentabilidade do produto, avaliando a prioridade de cada funcionalidade de acordo com valores de mercado.

A cada *Sprint* ele pode modificar características das funcionalidades ainda não implementadas e suas prioridades.

O *Product Owner* é quem aceita ou rejeita os resultados entregues pelo *Development Team* no final de cada *Sprint*, de forma a garantir que as funcionalidades desenvolvidas atendem às necessidades do cliente.

O *Product Owner* deve ser apenas uma pessoa, e nunca uma equipe. Ele pode representar os desejos de uma equipe no *Product Backlog*, mas quem quiser mudar a prioridade de um item deve convencer o *Product Owner.*

A pessoa representado o papel de *Product Owner* está na verdade, representando o cliente, a pessoa que encomendou o produto. Ele pode ser o gerente de produto para desenvolvimento comercial, ou, para sistemas que serão utilizados dentro da própria empresa, o gerente da função de negócios em que se está trabalhando.

Para que o *Product Owner* obtenha sucesso, todos na organização precisam respeitar suas decisões (SCHWABER; SUTHERLAND, 2011). Ninguém tem a permissão de dizer ao *Development Team* para trabalhar em um outro conjunto de prioridades, e os *Development Teams* não podem dar ouvidos a ninguém que diga o contrário. As decisões do *Product Owner* são visíveis no conteúdo e na priorização do *Product Backlog*. Essa visibilidade requer que o *Product Owner* faça seu melhor, o que faz o papel de *Product Owner* exigente e recompensador ao mesmo tempo.

Uma mesma pessoa pode representar os papéis de *Product Owner* e de membro do *Development Team*, porém essa responsabilidade adicional pode reduzir a sua habilidade de lidar com as partes interessadas.

No entanto, como já dito anteriormente, o *Product Owner* e o *Scrum Master* nunca podem ser a mesma pessoa, de acordo com o *Scrum Guide* de Maio de 2009.

2.1.3 Eventos do Scrum

Eventos prescritos são usados no *Scrum* para criar uma rotina e minimizar a necessidade de

reuniões não definidas no *Scrum*. (SCHWABER; SUTHERLAND, 2011)

Todos os eventos do *Scrum* tem duração máxima pré-estabelecida, constituindo o conceito de *time-box*, o que garante que uma quantidade adequada de tempo seja gasta no planejamento sem permitir perdas no processo de planejamento.

Além da *Sprint*, que é um container para outros eventos, cada evento no *Scrum* é uma oportunidade de inspecionar e adaptar alguma coisa. Estes eventos são especificamente projetados para permitir uma transparência e inspeção criteriosa. A não inclusão de qualquer um dos eventos resultará na redução da transparência e da perda de oportunidade para inspecionar e adaptar. (SCHWABER; SUTHERLAND, 2011)

2.1.3.1 Release Planning Meeting

Esta reunião opcional visa estabelecer um plano e metas que todos envolvidos no projeto possam entender e comunicar.

Durante a *Release Planning Meeting* são definidos a meta da versão, os itens do *Product Backlog* com maior prioridade, os principais riscos, as características gerais e funcionalidades que estarão contidas na versão, a data de entrega e os prováveis custos.

Produtos desenvolvidos utilizando a metodologia *Scrum* são construídos de maneira iterativa. Em cada *Sprint* o produto é incrementado em alguma(s) funcionalidade(s), dando se preferência aos itens do *Product Backlog* com maior prioridade. Cada incremento deve ser um pedaço completo e potencialmente entregável do produto final.

Muitas empresas realizam reuniões deste tipo, porém na maioria delas este planejamento é feito no início do projeto e não é modificado com o passar do tempo. Porém, não é possível se ter noção de todos os requisitos enquanto o desenvolvimento do produto não começa. (MAHNIC; DRNOVSCEK, 2005) Na *Release Planning Meeting* do *Scrum*, são definidos uma meta geral e

resultados prováveis.

Segundo o *Scrum Guide* de maio de 2009, esse planejamento geralmente não requer mais do que 15-20% do tempo que uma organização costumava utilizar para produzir um plano de versão para entrega tradicional. Porém, em um projeto gerenciado com *Scrum*, é feito planejamento a cada reunião, seja ela a *Sprint Review*, a *Sprint Planning Meeting*, da mesma forma que é feito planejamento diário no momento da execução de cada *Daily Scrum Meeting*. Então, de forma geral, os esforços de planejamento de um projeto desenvolvido sob a metodologia *Scrum* consomem ligeiramente mais tempo do que para um projeto com planejamento tradicional.

2.1.3.2 Sprint

O *Sprint* é a unidade básica de desenvolvimento em um projeto *Scrum*. Ele é uma caixa de tempo (do inglês *time-box*) de duração específica e constante, de normalmente um mês ou menos, durante a qual uma versão incremental potencialmente utilizável do produto é criada. *Sprints* tem durações coerentes em todo o esforço de desenvolvimento. Uma nova *Sprint* inicia imediatamente após a conclusão da *Sprint* anterior.

As *Sprints* são compostas por uma reunião de planejamento da *Sprint*, reuniões diárias, o trabalho de desenvolvimento, uma revisão da *Sprint* e a retrospectiva da *Sprint*.

Durante a *Sprint* a composição do *Development Team* não deve ser modificada.

Não devem ser feitas mudanças que possam afetar o *Sprint Goal*.

Cada *Sprint* pode ser considerada um projeto com horizonte não maior que um mês. Como os projetos, as *Sprints* são utilizadas para realizar algo. Cada *Sprint* tem a definição do que é para ser construído, um plano projetado e flexível que irá guiar a construção, o trabalho e o resultado do produto.

Sprints são limitadas a um mês corrido. Quando o horizonte da *Sprint* é muito longo, a definição do que será construído pode mudar, a complexidade pode aumentar e o risco pode crescer. *Sprints* permitem previsibilidade que garante a inspeção e adaptação do progresso em direção a meta pelo menos a cada mês corrido. *Sprints* também limitam o risco ao custo de um mês corrido.

A *Sprint* poderá ser cancelada se o *Sprint Goal* se tornar obsoleto. Isto pode ocorrer se a organização mudar sua direção ou se as condições do mercado ou das tecnologias mudarem. Geralmente a *Sprint* deve ser cancelada se ela não faz mais sentido às dadas circunstâncias. No entanto, devido a curta duração da *Sprint*, raramente cancelamentos fazem sentido. Somente o *Product Owner* tem a autoridade para cancelar a *Sprint*, embora ele (ou ela) possa fazer isso sob influência das partes interessadas, do *Development Team* ou do *Scrum Master.*

Quando a *Sprint* é cancelada, qualquer item do *Product Backlog* completado e "Pronto[3]" é revisado. Se uma parte do trabalho estiver potencialmente utilizável, tipicamente o *Product Owner* o aceita. Todos os itens do *Product Backlog* incompletos são re-estimados e colocados de volta no *Product Backlog*. O trabalho feito se deprecia rapidamente e deve ser frequentemente re-estimado.

O cancelamentos de *Sprints* consome recursos, já que todos tem que se reagrupar em outra reunião de planejamento da *Sprint* para iniciar outra *Sprint*. Cancelamentos de *Sprints* são frequentemente traumáticos para o *Scrum Team*, e são muito incomuns.

2.1.3.3 Sprint Planning Meeting

O que será feito durante a *Sprint* é planejado colaborativamente entre todo o *Scrum Team* nesta reunião. Em projetos cujos *Sprints* tenham um mês de duração, ela é um *time-box* de oito horas. Para projetos em que os *Sprints* sejam mais curtos, este evento é proporcionalmente mais

3 Quando um item do Product Backlog ou um Incremento é descrito como "Pronto", todos devem compreender o que "Pronto" significa. Todos os membros do Scrum Team devem ter um entendimento compartilhado do que significa o trabalho estar completo, assegurando a transparência. A mesma definição orienta a equipe de desenvolvimento no conhecimento de quantos itens do Product Backlog podem ser selecionados durante a Sprint Planning Meeting. O propósito de cada Sprint é entregar incrementos de funcionalidades potencialmente utilizáveis que aderem à definição atual de "Pronto" do Scrum Team. Com um Time Scrum maduro, é esperado que a sua definição de "Pronto" seja expandida para incluir critérios mais rigorosos de qualidade.

curto.

Esta reunião é formada por duas partes de igual duração. Na primeira parte decide-se o que será entregue no Incremento resultante da próxima *Sprint*, e na segunda decide-se como alcançar este objetivo.

2.1.3.3.1 Primeira parte: O que será feito neste Sprint?

Na primeira parte da *Sprint Planning Meeting*, o time de desenvolvimento tenta estimar as funcionalidades que serão desenvolvidas durante a próxima *Sprint* com base nos itens do *Product Backlog* ordenados, que são apresentados pelo *Product Owner.*

Os insumos desta reunião são o *Product Backlog*, o último Incremento, a capacidade projetada do time de desenvolvimento durante a *Sprint*, além de sua performance passada.

Somente o time de desenvolvimento pode escolher o número de itens do *Product Backlog* que serão selecionados para o *Sprint*, pois somente os desenvolvedores tem condições de avaliar o que pode ser realizado.

Após decidir quais itens do *Product Backlog* serão realizados no próximo *Sprint*, todo o *Scrum Team* define o *Sprint Goal*, a meta do *Sprint.* O *Sprint Goal* é um objetivo que será atingido durante a Sprint através da implementação do *Product Backlog*, e provê direção ao time de desenvolvimento a respeito dos motivos da construção do Incremento.

2.1.3.3.2 Segunda parte: Como serão executados os itens selecionados?

Na segunda parte da reunião é definido como será criado um novo Incremento "Pronto" do produto durante o próximo *Sprint*.

O *Sprint Backlog* é basicamente o conjunto formado pelos itens do *Product Backlog* selecionados para o *Sprint*, e o plano para a entrega destes itens executados. Cada item do *Product Backlog* é decomposto em atividades de um dia ou menos. O *Development Team* se auto-organiza

na atribuição das atividades do *Sprint Backlog*, tanto durante esta reunião quanto no decorrer do *Sprint* quando necessário.

Nesta fase pode-se convidar outras pessoas para a reunião com a finalidade de prover conhecimento técnico e aconselhamento sobre o domínio.

2.1.3.3.3 Sprint Goal

O *Sprint Goal* é a meta da *Sprint*, e dá ao *Development Team* alguma flexibilidade em relação as funcionalidades a serem implementadas dentro da *Sprint*.

O *Development Team* deve manter a *Sprint Goal* em mente durante todo o *Sprint*. Caso o resultado das atividades seja diferente do esperado pelo *Development Team*, o ideal é negociar com o *Product Owner* o escopo do *Sprint Backlog* dentro da *Sprint*.

2.1.3.4 Daily Scrum

A *Daily Scrum* é uma *time-box* de 15 minutos que acontece diariamente entre os membros do *Development Team* para sincronizar as atividades e criar um plano para as próximas 24 horas. Isto é feito através da inspeção do trabalho feito desde a última *Daily Scrum* e da previsão do trabalho que pode ser feito até a próxima.

Estas reuniões devem acontecer sempre no mesmo horário e local com o objetivo de reduzir a complexidade.

Durante a reunião cada integrante do *Development Team* deve informar:

- O que foi realizado desde a última reunião
- O que será feito até a próxima reunião
- Quais os obstáculos estão no caminho

Estas reuniões diárias são utilizadas pela equipe de desenvolvimento para avaliar o progresso em direção à *Sprint Goal*. Elas aumentam a probabilidade de sucesso do time de desenvolvimento com relação ao mesmo.

O *Scrum Master* deve assegurar que o *Development Team* realize a reunião, mas o *Development Team* é responsável por conduzi-la, e seus membros são os únicos que devem participar da mesma. Esta não é uma reunião de status, ela é voltada para as pessoas que transformam os itens do *Product Backlog* em um Incremento.

A realização das *Daily Scrums* melhora a comunicação, elimina a necessidade de outras reuniões, auxilia a identificação e remoção de impedimentos ao desenvolvimento do produto, agiliza a tomada de decisão e melhora o nível de conhecimento dos desenvolvedores a respeito do projeto. Esta é uma reunião chave para inspeção e adaptação.

2.1.3.5 Sprint Review

Uma reunião de revisão acontece ao final da *Sprint* para inspecionar o Incremento e adaptar o *Product Backlog* se necessário. Nela se expõe o que foi feito durante o *Sprint*. Baseado nisso e e nas mudanças que podem ter ocorrido no *Product Backlog* durante o *Sprint*, é decidido o que pode ser feito a seguir. Esta reunião é informal, e uma apresentação do Incremento destina-se a motivar e obter comentários e promover a colaboração.

O *time-box* desta reunião é de quatro horas para *Sprints* de um mês, sendo proporcionalmente menor em caso de *Sprints* menores.

A *Sprint Review* inclui os seguintes elementos:

- O *Product Owner* identifica o que está e o que não está "Pronto"
- O *Development Team* discute o que foi bem durante a *Sprint*, quais problemas ocorreram, e

como estes problemas foram resolvidos

- O *Development Team* demonstra o trabalho que está "Pronto" e responde as questões sobre o incremento

- O *Product Owner* discute o *Product Backlog* tal como está e projeta as prováveis datas de conclusão baseado no progresso até então

- O grupo todo colabora sobre o que fazer a seguir, e é assim que a *Sprint Review* fornece valiosas entradas para a próxima *Sprint Planning Meeting*.

O resultado da *Sprint Review* é um *Product Backlog* revisado que define os itens do *Product Backlog* que provavelmente farão parte do próximo *Sprint Backlog*.

2.1.3.6 Sprint Retrospective

A *Sprint Retrospective* é uma oportunidade para o *Scrum Team* inspecionar a si próprio e criar um plano para melhorias a serem aplicadas na próxima *Sprint*. Ela ocorre após a *Sprint Review* e antes da próxima *Sprint Planning Meeting*. É um *time-box* de três horas para *Sprints* de um mês e proporcionalmente mais curta em casos de *Sprints* menores.

O propósito da *Sprint Retrospective* é:

- Inspecionar como a última *Sprint* foi em relação as pessoas, relações, processos e ferramentas

- Identificar e ordenar os principais itens que foram bem e as potenciais melhorias

- Criar um plano para implementar melhorias no modo que o *Scrum Team* faz seu trabalho

O *Scrum Master* encoraja o *Scrum Team* a melhorar, dentro do processo do *framework* do *Scrum*, o processo de desenvolvimento e as práticas para fazê-lo mais efetivo e agradável para a

próxima *Sprint*. Durante cada *Sprint Retrospective*, o time planeja formas de aumentar a qualidade do produto, adaptando a definição de "Pronto" quando apropriado.

Ao final da *Sprint Retrospective*, o *Scrum Team* deverá ter identificado melhorias que serão implementadas na próxima *Sprint*. A implementação destas melhorias na próxima *Sprint* é a forma de adaptação à inspeção que o *Scrum Team* faz a si próprio. A *Sprint Retrospective* fornece um evento dedicado e focado na inspeção e adaptação, no entanto, as melhorias podem ser adotadas a qualquer momento.

2.1.4 Artefatos do Scrum

O *Scrum* é um *framework* de natureza leve, e isso fica evidente no número pequeno de artefatos nele definidos.

Os artefatos são documentos, normalmente listas ou gráficos, que guiam e acompanham o desenvolvimento do produto.

Os artefatos do *Scrum* são oportunidades para inspeção e adaptação. Eles são especialmente projetados para maximizar a transparência de informações chave necessárias para garantir que o time obtenha sucesso na entrega do incremento pronto.

2.1.4.1 Product Backlog

Quando se trabalha com *Scrum*, não é necessário começar um projeto com uma extensa documentação dos requisitos. As funcionalidades principais devem simplesmente ser descritas em tópicos curtos, em uma lista que será melhor detalhada quando, e se necessário.

Backlog, em português, pode ser traduzido como demanda não atendida, uma fila de espera. Esta lista é atualizada constantemente, para atender à mudanças de negócio, condições de mercado, tecnologia... É por isso que o *Scrum* torna possível o desenvolvimento com mudanças constantes de requisitos. Porque não se dedica tempo descrevendo detalhadamente funções que só serão

implementadas muito adiante, e que podem se mostrar desnecessárias ou inviáveis.

O *Product Backlog* é esta lista com tudo que ainda pode ser feito no produto final, ordenada por prioridade. Este artefato deve ser a única fonte de requisitos para qualquer mudança no produto.

Nele estão todas as funcionalidades, requisitos, melhoras e correções que constituem as mudanças nas próximas versões do produto.

Por ser quem representa o cliente, e quem melhor compreende o que ele espera e precisa, o responsável pelo *Product Backlog* é o *Product Owner*. É ele quem determina as funcionalidades que o produto deve apresentar, e ele quem deve definir quais itens tem maior prioridade e importância. O ideal é que os itens que adicionam mais valor ao produto sejam desenvolvidos primeiro.

Este artefato é dinâmico, e sofre mudanças frequentemente. Funcionalidades que foram adicionadas à lista no começo podem ser excluídas por diversos motivos, outras podem ser adicionadas, seja porque no início não se percebeu sua necessidade, ou porque a necessidade veio depois. Funcionalidades podem ser modificadas, e prioridades também.

Um item do *Product Backlog* é composto por sua descrição, prioridade e estimativa. Quem estima o tempo para que o item seja dado como feito é o *Development Team*. O *Product Owner* pode auxiliar o *Development Team* ajudando a compreender o que deve ser feito, mas a estimativa final é dada pelo *Development Team*.

2.1.4.2 Sprint Backlog

O *Sprint Backlog* é um subconjunto de itens do *Product Backlog* selecionados para o *Sprint*, além de um plano para a entrega de um Incremento do produto e a realização do *Sprint Goal*. O *Sprint Backlog* é a previsão da equipe de desenvolvimento sobre qual funcionalidade estará no próximo incremento e do trabalho necessário para entregar a funcionalidade.

O *Sprint Backlog* define o trabalho que deve ser feito para transformar itens do *Product Backlog* em um Incremento “Pronto”.

O *Sprint Backlog* é um plano detalhado o suficiente para que mudanças durante o progresso possam ser entendidas durante as *Daily Scrums*. O *Development Team* modifica o *Sprint Backlog* ao longo de toda a *Sprint*, e o *Sprint Backlog* emerge durante a *Sprint*. Esta emergência ocorre quando o *Development Team* trabalha segundo o plano e aprende mais sobre o trabalho necessário para alcançar o *Sprint Goal*.

Quando um novo trabalho é necessário, o *Development Team* adiciona este ao *Sprint Backlog*. Conforme o trabalho é realizado ou completado, a estimativa do trabalho restante deve ser atualizada. Quando elementos do plano são considerados desnecessários, eles devem ser removidos. Somente o *Development Team* pode alterar o *Sprint Backlog* durante a *Sprint*. O *Sprint Backlog* deve ser altamente visível, uma imagem em tempo real do trabalho que o *Development Team* planeja completar durante a *Sprint*.

Para monitorar o progresso durante a *Sprint*, a qualquer momento deve ser possível saber o total de trabalho que ainda falta ser feito. O *Development Team* acompanha estes dados diariamente e projeta a probabilidade de alcançar o *Sprint Goal*.

2.1.4.3 Incremento

O Incremento é a soma do todos os itens do *Product Backlog* que estão prontos ao final de um *Sprint*. Estes itens são tanto do último *Sprint* quanto de todos os anteriores. Ao final do *Sprint*, o novo Incremento deve estar “Pronto”, o que significa que deve estar um condição usável e atender a definição de “Pronto” definida anteriormente. Deve estar em condição usável independente de o *Product Owner* colocá-la em produção ou não.

O *Development Team* entrega um incremento de funcionalidade do produto a cada *Sprint*.

Este incremento é utilizável, assim o *Product Owner* pode escolher por liberá-lo imediatamente. Cada incremento é adicionado a todos os incrementos anteriores e completamente testado, garantindo que todos os incrementos funcionam juntos.

2.2 Web Semântica

"Sonho com uma web na qual os computadores seráo capazes de analisar o conteúdo e entender as transações entre as pessoas e as máquinas. Chama-se 'web semântica', mas ainda tem de ser desenvolvida. Quando se tornar realidade, nossa vida será manejada por computadores. O agente inteligente, que a humanidade buscou tanto, por fim se materializará." (BERNERS-LEE, TIM; FISCHETTI, [S.d.])

Embora a web semântica ainda esteja em uma fase preliminar, a ideia não é nova. Na verdade, é tema de discussão desde 1999, quando Timothy Berners-Lee, diretor do World Wide Web Consortium (W3C) organização que promove os standards da web, alcunhou o termo. Segundo Yihong Ding, pesquisador da web semântica e colaborador no site de notícias sobre tecnologia ZDNet, estamos em um momento de transição, passando de uma web definida pelo editor a outra determinada pelo usuário (LAFUENTE, 2011). Ele explica as etapas didaticamente:

- Em sua versão 1.0 (1990-1999), os espaços da web era páginas que representavam a visão da empresa ou da organização proprietária do domínio.

- Na 2.0 (2000-2009), os espaços passaram a ser as contas pessoais dos usuários, embora alojadas em sites pensados e desenhados por um editor

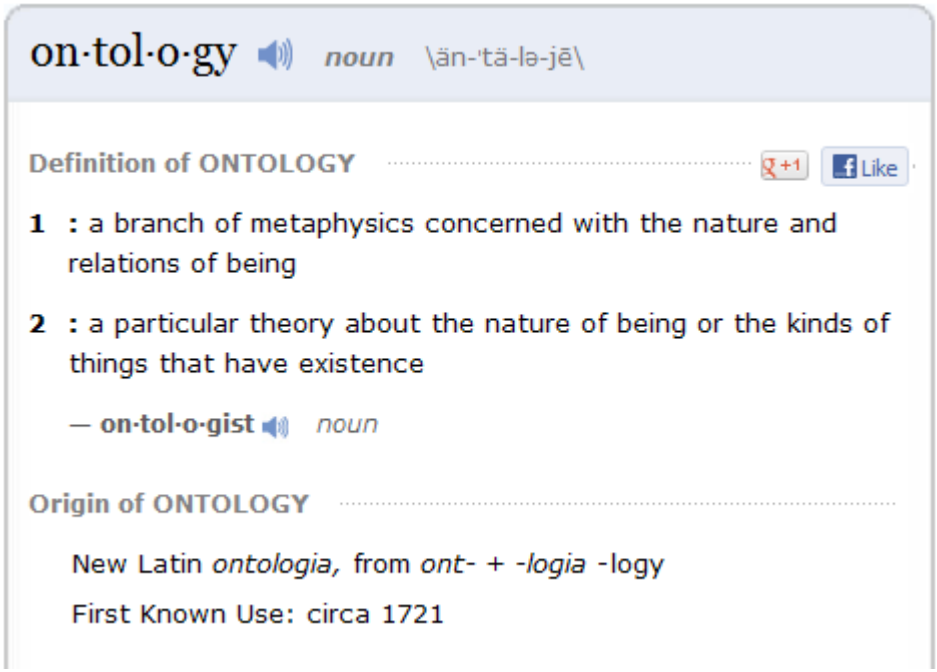
- na 3.0 (2010-2020, segundo previsões), os espaços da web serão uma coleção de recursos de diferentes sites que se organizarão em tempo real, de acordo com a visão do usuário (LAFUENTE, 2011)

Com as novas tecnologias de NLP (processamento da linguagem natural, sigla em inglês), a noção de busca na internet mudará radicalmente: em vez de fazer consultas por palavras-chave, poderão ser pesquisados conceitos ou ideias, porque as máquinas entenderão o significado da palavra escrita.

Atualmente, os sistemas de NLP são usados em atividades que demandam processamento de grandes quantidades de dados, como detecção de fraudes, prevenção de lavagem de dinheiro e terrorismo, análise de inteligência de negócios, marketing e publicidade, pesquisa científica e medicina. Mediante a aplicação de complexos algoritmos ao texto desordenado, os motores de NLP são capazes de gerar dados estruturados ou modelos de dados. Os especialistas garantem que esses sistemas são a única tecnologia viável para automatizar a extração de informação valiosa do atual caos da web. Segundo a organização worldwidewebsize.com, a anarquia é grande: 12,830 bilhões de sites registrados em janeiro de 2011.

2.3 Ontologias

A palavra ontologia vem do grego *ontós* (ser, existência, indivíduo) e *logos* (palavra, estudo, discurso).


on·tol·o·gy noun \än-'tä-lə-jē\

Definition of ONTOLOGY

1 : a branch of metaphysics concerned with the nature and relations of being

2 : a particular theory about the nature of being or the kinds of things that have existence

— on·tol·o·gist noun

Origin of ONTOLOGY

New Latin *ontologia,* from *ont-* + *-logia* -logy

First Known Use: circa 1721

Figura 4: Definição de Ontologia Fonte: http://www.merriam-webster.com

O dicionário online Merriam Webster, como pode-se observar na Figura 4, define ontologia como:

- 1: um ramo da metafísica voltado à natureza e relações entre seres
- 2: uma teoria específica sobre a natureza dos seres e tipos de coisas que existem.

O termo ontologia foi introduzido na filosofia, no século 19, pelo filósofo alemão Rudolf Gockel, em seu *Lexicon Philosophicum*, para distinguir o estudo do "ser" do estudo dos vários tipos de seres nas ciências naturais.

Na visão filosófica, a construção de ontologias é encarregada de prover sistemas de categorias que representem determinada visão de mundo.

O primeiro sistema de categorias foi proposto por Aristóteles. Neste sistema, uma categoria é utilizada para classificar qualquer coisa que possa ser dita sobre algo.

No século III A.C. , Porfírio, um filósofo grego, deu acabamento à classificação das categorias propostas por Aristóteles, comentando sua estrutura, e organizando as categorias propostas em um diagrama em formato de árvore. Esta estrutura ficou conhecida como Árvore Porfiriana.

Em web semântica, a definição de ontologia citada mais frequentemente é a de (GRUBER, 1993): "Ontologia é uma especificação formal e explícita de uma conceitualização compartilhada".

Significando:

- explícito: os elementos devem ser claramente definidos
- formal: a especificação deve ser passível de processamento por máquina

- conceitualização: um modelo abstrato

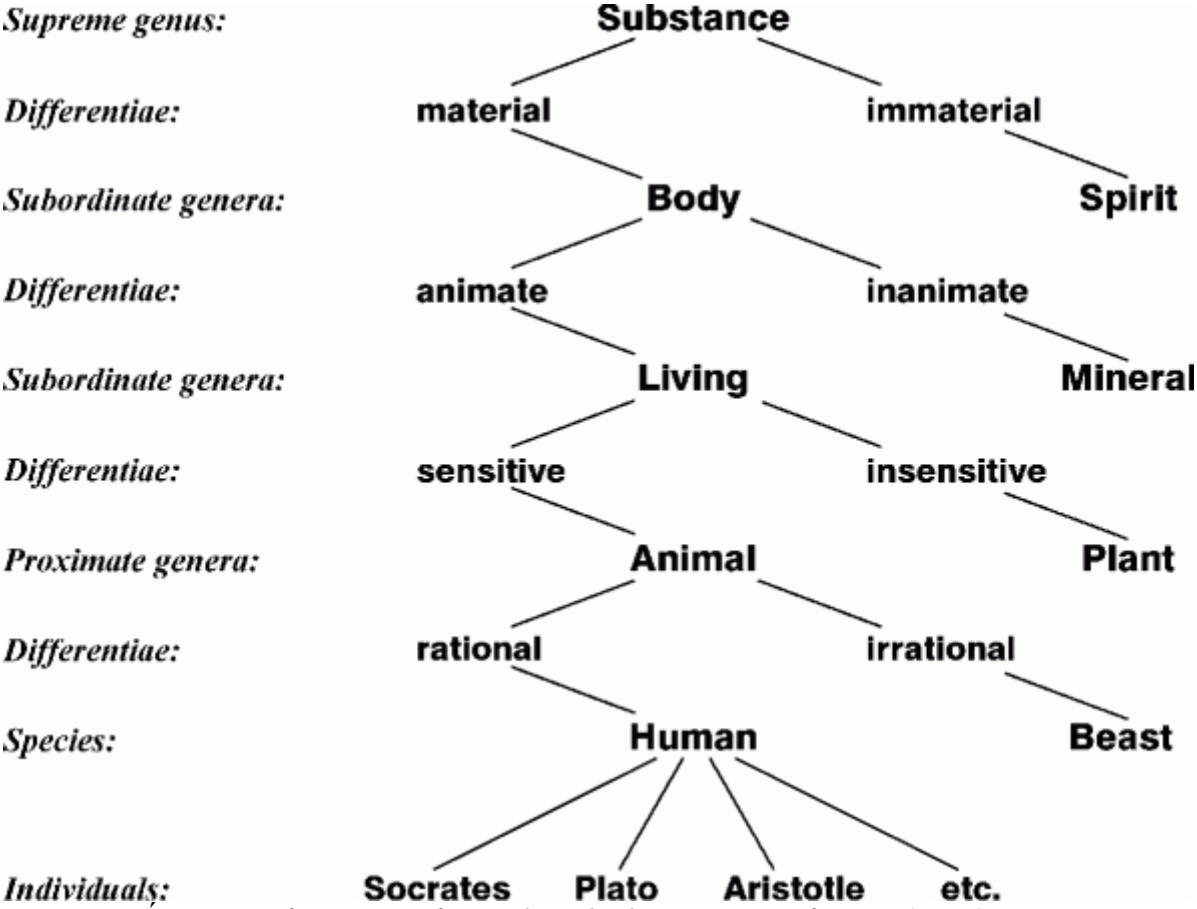


Figura 5: Árvore Porfiriana, conforme desenhada por Peter of Spain (1329)

Assim sendo, pela visão de Gruber, uma ontologia é a representação do conhecimento sobre um certo domínio, onde um conjunto de objetos e as relações entre eles é descrita por um vocabulário.

Sowa, em Principles of Ontology (SOWA, 1997), define ontologia como o estudo das categorias das coisas que existem ou podem existir em algum domínio. O resultado deste estudo, chamado ontologia, é um catálogo de tipos de coisas que se assume que existam em um domínio de interesse D pela perspectiva de uma pessoa que utiliza a linguagem L com o intuito de falar sobre D.

Portanto, não há uma definição universal para ontologia. Isto se deve em parte ao grande

leque de possíveis utilizações de ontologias.

Ontologias são utilizadas para a comunicação entre humanos e sistemas computacionais, inferência computacional, e para reuso e organização de conhecimento.(GRUNNINGER; LEE, 2002)

Ontologias vem sendo usadas em diferentes áreas da ciência da computação como inteligência artificial, representação do conhecimento, processamento de linguagens naturais, web semântica, engenharia de software, entre outras. Assim sendo, é natural que existam divergências entre suas múltiplas definições. Apesar de que uma ontologia possa assumir diferentes formatos, ela tipicamente inclui um vocabulário de termos, uma especificação de seu significado, e uma indicação de como os termos estão ligados (USCHOLD; JASPER, 1999).

Ontologia é um termo emprestado da filosofia que se refere à ciência de descrever os tipos de entidades no mundo e como elas se relacionam (W3C ® (MIT, ERCIM, KEIO), 2013a).

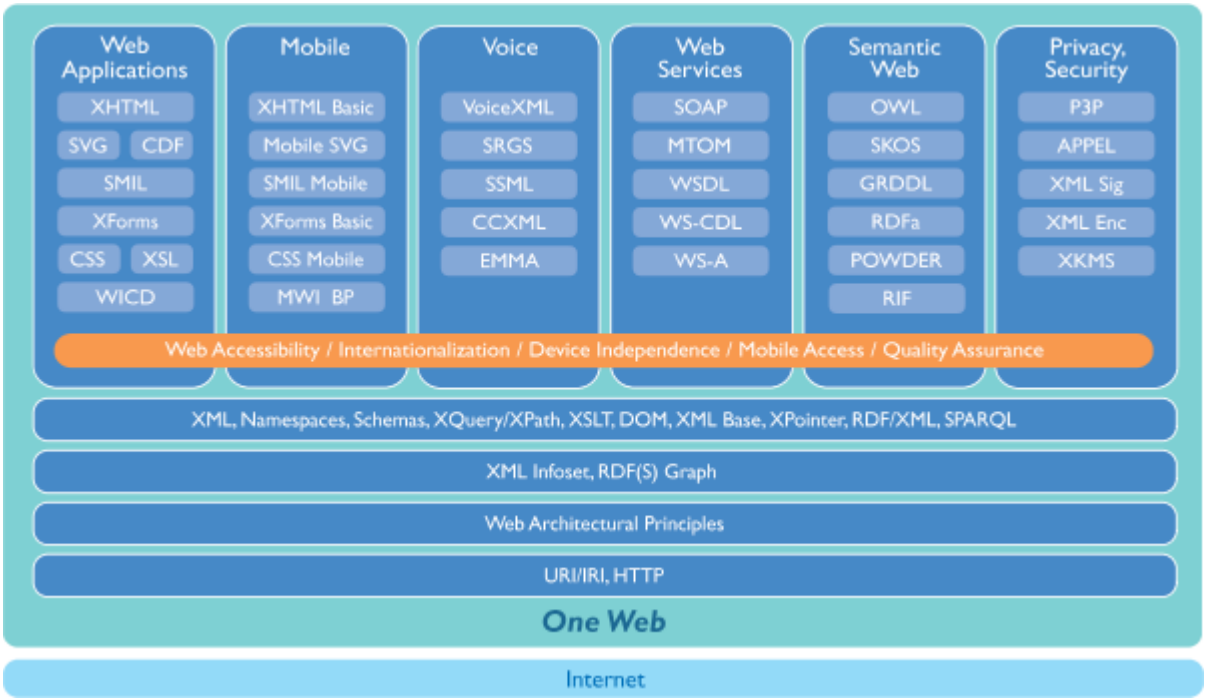


Figura 6: Arquitetura da Web Semântica em paralelo com as demais iniciativas da W3C. Fonte: w3.org

Ontologias são estruturas em grafo formadas por triplas compostas pelas partículas *Domain*,

Property e *Range*. Tais triplas são a base do RDF (RDF WORKING GROUP, 2004), uma das linguagens da pilha de tecnologias que formam a Web Semântica (BERNERS-LEE, T. *et al.*, 2001)

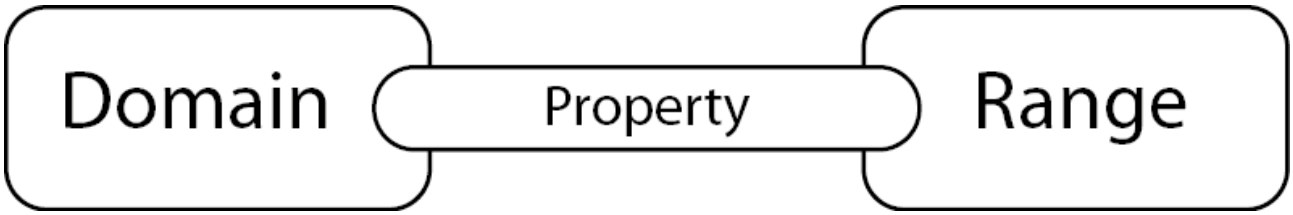


Figura 7: Representação gráfica de uma tripla RDF. Fonte: (SANTOS, 2012)

Um exemplo didático de como é possível formalizar assertivas sobre uma determinada realidade por meio de triplas RDF é apresentado na Figura 8, de (SANTOS, 2012), em que "João" e "Fusca" são indivíduos das classes "Pessoa" e "Carro", enquanto "é proprietário" é uma propriedade que tem como *domain* "Pessoa", e *range* "Carro".

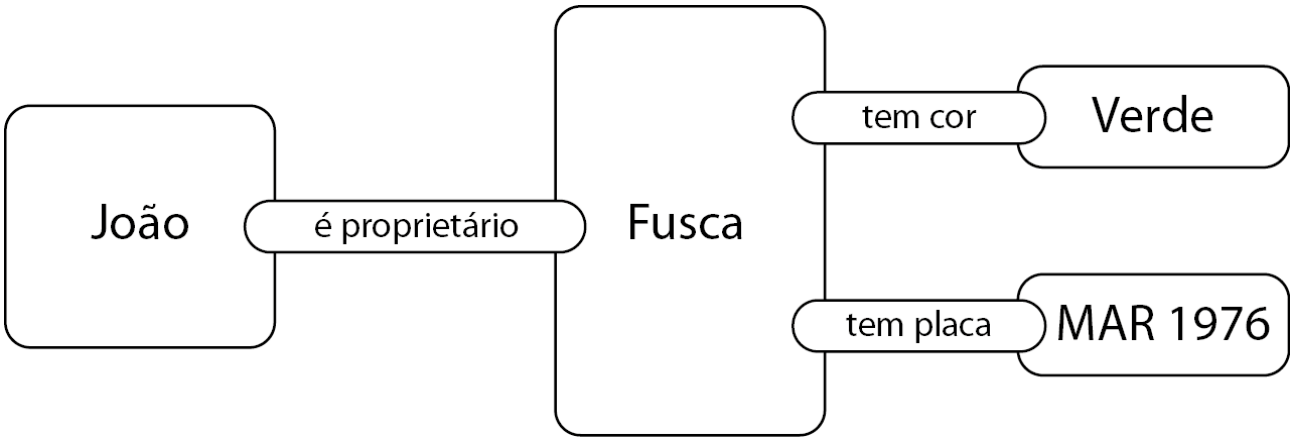


Figura 8: Ilustração de triplas RDF representando assertivas. Fonte:(SANTOS, 2012)

Cada elemento desse exemplo é identificado, em uma ontologia, por uma IRI (*Internationalized Resource Identifier*), identificador muito semelhante à conhecida URL (*Uniform Resource Locator*), com uma diferença marcante: a URI permite o uso de caracteres internacionalizados, como, por exemplo, o cedilha (ç), e de caracteres de línguas orientais, entre elas o árabe, o hebreu e o japonês (DUERST; SUIGNARD, 2005).

2.4 Inferência

De maneira geral, a inferência na Web Semântica pode ser caracterizada pela descoberta de novos relacionamentos. Na Web Semântica os dados são modelados como um conjunto de relacionamentos rotulados entre recursos. Inferência significa que procedimentos automatizados podem gerar novos relacionamentos baseados em informações adicionais na forma de um conjunto de regras (W3C ® (MIT, ERCIM, KEIO), 2013b).

A fonte dessas informações adicionais pode ser definida via vocabulários (ontologias) ou conjuntos de regras. Ambas as abordagens utilizam técnicas de representação do conhecimento. Em geral, ontologias se concentram em métodos de classificação, com ênfase na definição de classes, subclasses, em como recursos individuais podem ser associados à estas classes, e em caracterizar os relacionamentos entre as classes e suas instâncias. Por sua vez, a utilização de regras se concentra em definir um mecanismo geral para descobrir e gerar novos relacionamentos com base nos relacionamentos existentes. A W3C recomenda a utilização de RDFS, OWL, ou SKOS para definir ontologias, enquanto o RIF foi desenvolvido para cobrir a abordagem baseada em regras (W3C ® (MIT, ERCIM, KEIO), 2013b).

Em uma definição bem genérica, ontologias são melhores para problemas de raciocício taxonômico, enquanto sistemas baseados em regras são melhores para problemas de raciocício relacionados aos dados. Entretanto, o uso de ontologias e regras frequentemente se sobrepõe (W3C ® (MIT, ERCIM, KEIO), 2013b).

Na web semântica, utiliza-se inferência para melhorar a qualidade da integração de dados da web, descobrir novos relacionamentos, analisar automaticamente o conteúdo de dados, ou gerenciar o conhecimento na web em geral. Técnicas baseadas em inferência são também importantes para descobrir possíveis inconsistências em dados já integrados (W3C ® (MIT, ERCIM, KEIO), 2013b).

2.5 SPARQL

RDF é um formato de dados para a representação de informações na web. Ele é frequentemente utilizado para representar entre outras coisas, informações pessoais, redes sociais, metadados sobre artefatos digitais, assim como prover meios de integração entre diferentes fontes de informações. Esta especificação define a sintaxe e semântica do SPARQL (W3C®(MIT, ERCIM, KEIO, BEIHANG), 2013).

O nome SPARQL é um acrônimo recursivo para SPARQL *Protocol and RDF Query Language*. Ele é uma linguagem de consulta para bancos de dados, capaz de retornar e manipular dados armazenados em RDF. É considerada a linguagem padrão para isto pelo *RDF Data Access Working Group* (DAWG) da *World Wide Web Consortium*, e é considerada uma das tecnologias chave da web semântica(W3C®(MIT, ERCIM, KEIO, BEIHANG), 2013).

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name (COUNT(?friend) AS ?count)
WHERE {
    ?person foaf:name ?name .
    ?person foaf:knows ?friend .
} GROUP BY ?person ?name
```

Quadro 1: Exemplo de código SPARQL. Fonte: (W3C®(MIT, ERCIM, KEIO, BEIHANG), 2013)

O Quadro 1 mostra o código SPARQL que faz uma consulta que retorna os nomes das pessoas e o número de amigos que elas possuem. O Quadro 2 mostra um possível resultado para esta consulta.

?name	?count
"Alice"	3
"Bob"	1
"Charlie"	1

Quadro 2: Resultado da consulta do Quadro 1. Fonte: (W3C®(MIT, ERCIM, KEIO, BEIHANG), 2013)

3 A OntoScrum

OntoScrum é a ontologia criada para modelar o processo *Scrum*. Os conceitos básicos do *framework* são nela representados, assim como outros conceitos que se mostram necessários. Suas classes são apresentadas na Figura 9 em forma de diagrama, e na Figura 10, em uma estrutura de árvore.

3.1 Principais classes e propriedades

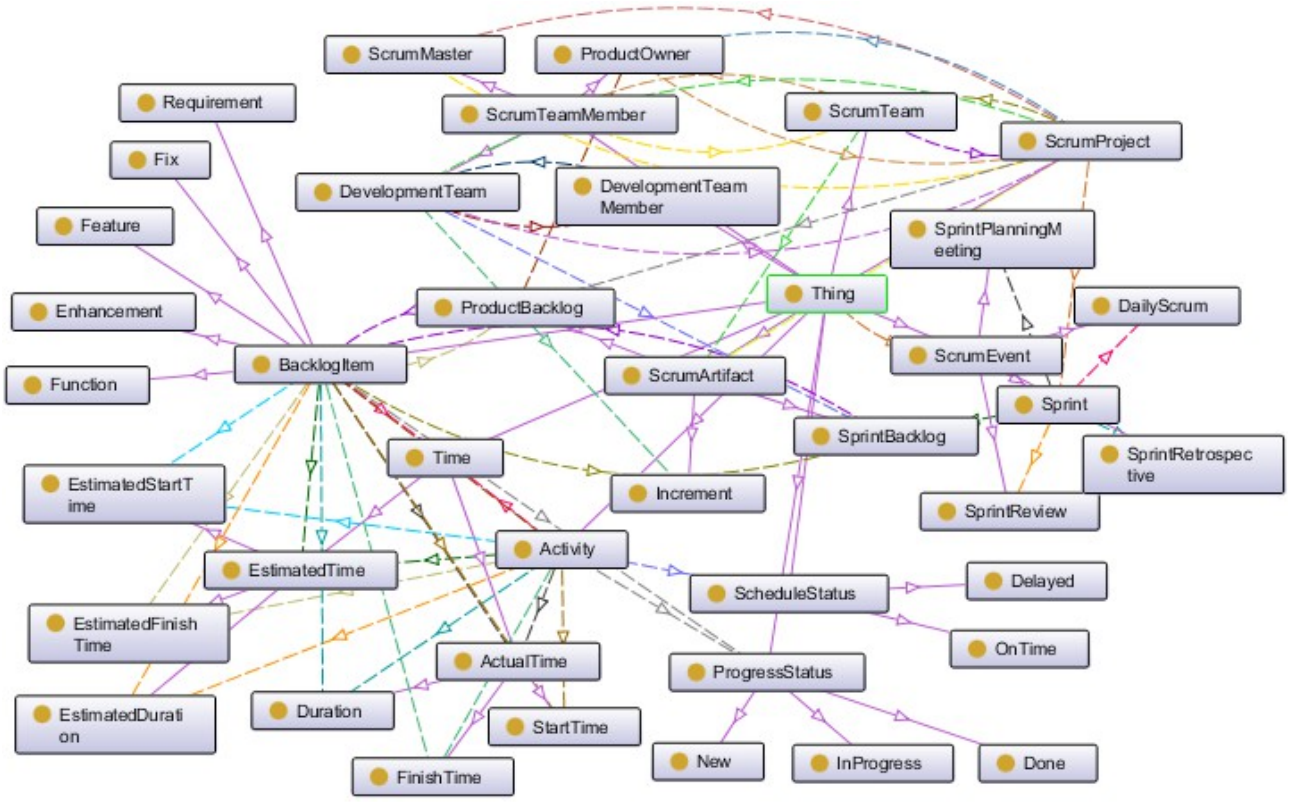


Figura 9: OntoScrum - Diagrama com todas as classes. Fonte: elaborado pelo autor

Há uma classe *ScrumTeam*, sem filhos. Em princípio, *ScrumMaster*, *ProductOwner* e *DevelopmentTeam* seriam filhas desta, porém, como por exemplo, um *ScrumMaster* não é um *ScrumTeam*, e sim membro dele, foi criada uma classe *ScrumTeamMember*, que tem essas classes como filhas, e uma propriedade *hasScrumTeamMember* que tem como *domain ScrumTeam*, e *range ScrumTeamMember*, havendo também uma propriedade inversa à essa chamada *isMemberOfScrumTeam*. Esta estrutura de classes pode ser observada na Figura 11.

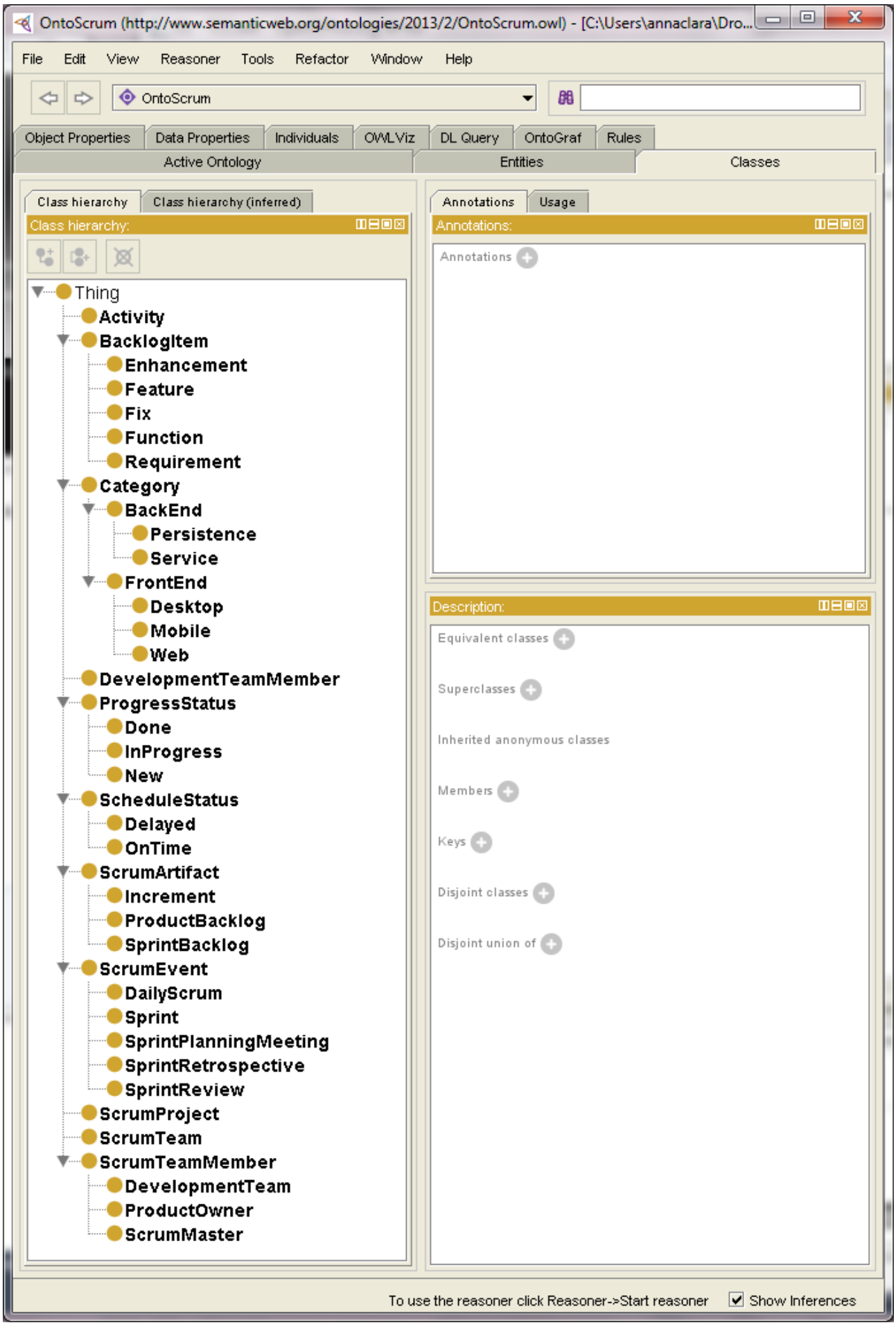


Figura 10: OntoScrum - Todas as classes. Fonte: elaborado pelo autor

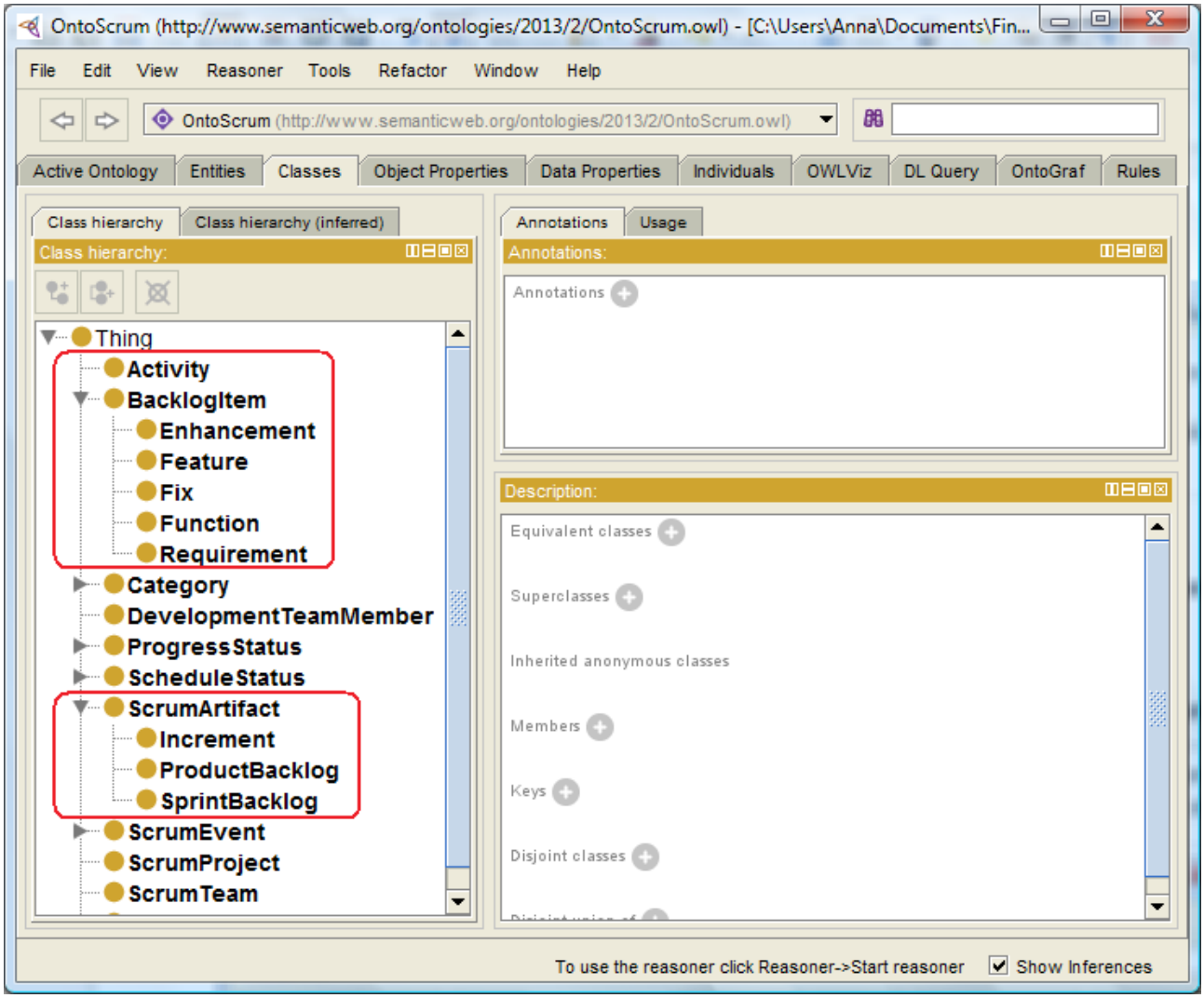


Figura 12: OntoScrum - Artefatos, Atividades e BacklogItems. Fonte: elaborado pelo autor

A classe *ScrumArtifacts* e suas filhas *Increment*, *ProductBacklog*, e *SprintBacklog* representam os artefatos do projeto. Há uma classe chamada *BacklogItem* que representa os itens tanto do *ProductBacklog* quanto do *SprintBacklog*. Um *BacklogItem* pode ser uma funcionalidade, melhoria, correção ou requisito, e é formado por uma descrição, uma estimativa de tempo e um número inteiro que indica o valor de negócio deste item. Quanto maior este valor, mais alta a prioridade, e antes este item deve ser implementado. Diversas atividades (*Activities)* podem ser realizadas para completar um *BacklogItem*. As *Activities* por sua vez também tem estimativa de tempo. Estas classes podem ser visualizadas na Figura 12.

Durante a modelagem dos artefatos foi encontrada uma inconsistência na definição dos *Backlogs*. O *Product Backlog* é descrito no *Scrum Guide* como uma lista ordenada de tudo que deve existir no produto, no qual somente o *Product Owner* pode mexer. O *Sprint Backlog* é um conjunto de itens do *Product Backlog* selecionados para a *Sprint*, juntamente com o plano de entrega do incremento do produto, e nele só os membros do *Development Team* podem mexer. Se um item está em uma lista na qual só o *Product Owner* pode mexer, e também está em uma lista na qual só o *Development Team* pode mexer, a modelagem fica inconsistente. Outra inconsistência é que ao longo do *Sprint* novos itens devem surgir no *Sprint Backlog* e ao mesmo tempo, tudo que diz respeito ao projeto deve estar no *Product Backlog*. E o *Development Team* não pode mexer no *Product Backlog*.

A abordagem escolhida neste caso foi manter o *Sprint Backlog* como um subconjunto do *Product Backlog*, o primeiro apresentando apenas itens que já existem no segundo. O que surgir além destes, serão atividades (*Activities*), portanto foi criada uma propriedade *hasActivity* que tem como *domain BacklogItem*, e como *range Activity*.

O que faz com que os itens do *Product Backlog* e do *Sprint Backlog* sejam os mesmos são as propriedades *isProductBacklogItemOf* e *isSprintBacklogItemOf*. A *isProductBacklogItemOf*, que pode ser vista na Figura 13, tem como *domain BacklogItem* e como *range Product Backlog*. Já a *isSprintBacklogItemOf*, que aparece na Figura 14, tem como *domain BacklogItem* e como *range Sprint Backlog*, além de ter como super propriedade *isProductBacklogItemOf*.

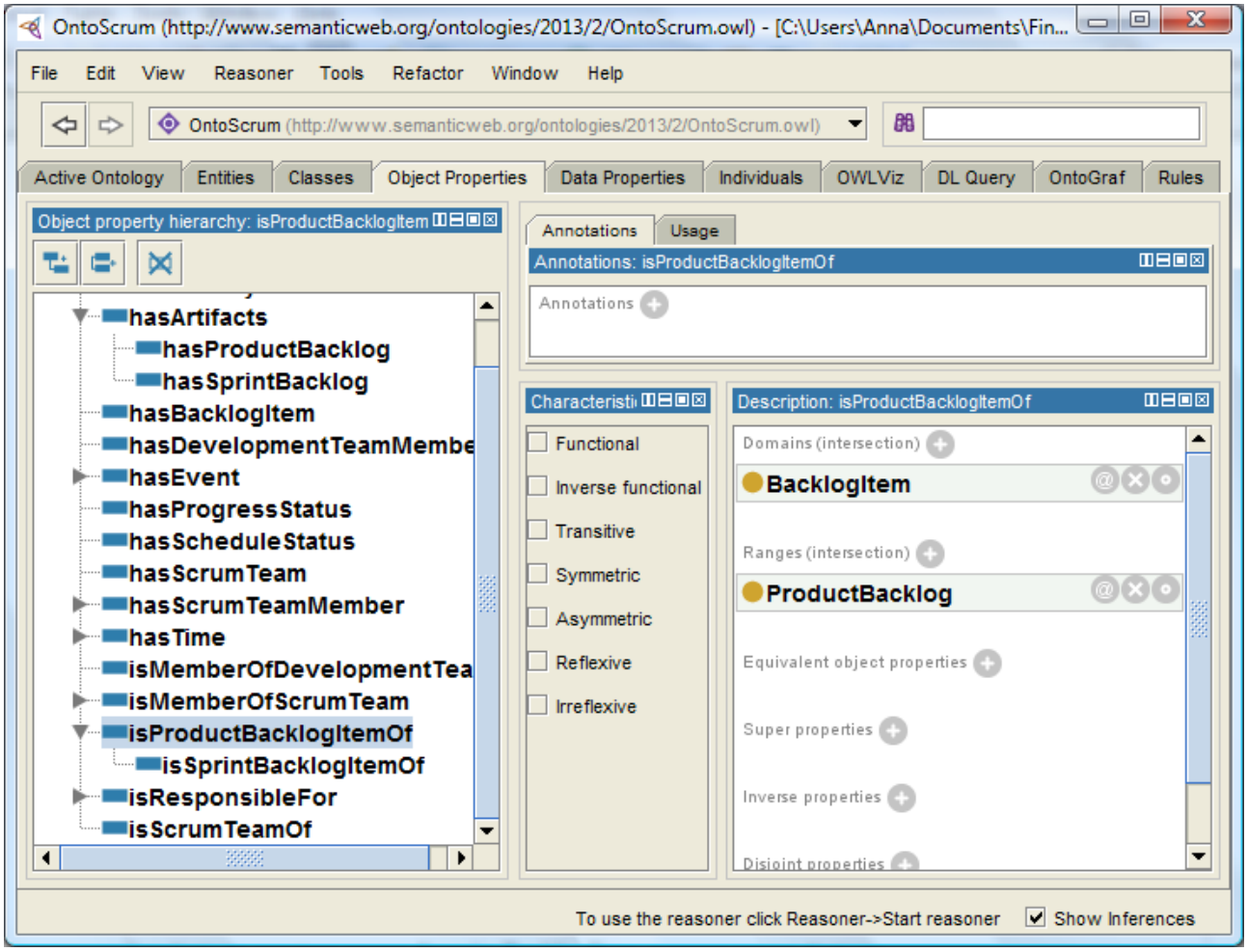


Figura 13: OntoScrum – Propriedade isProductBacklogItemOf. Fonte: elaborado pelo autor

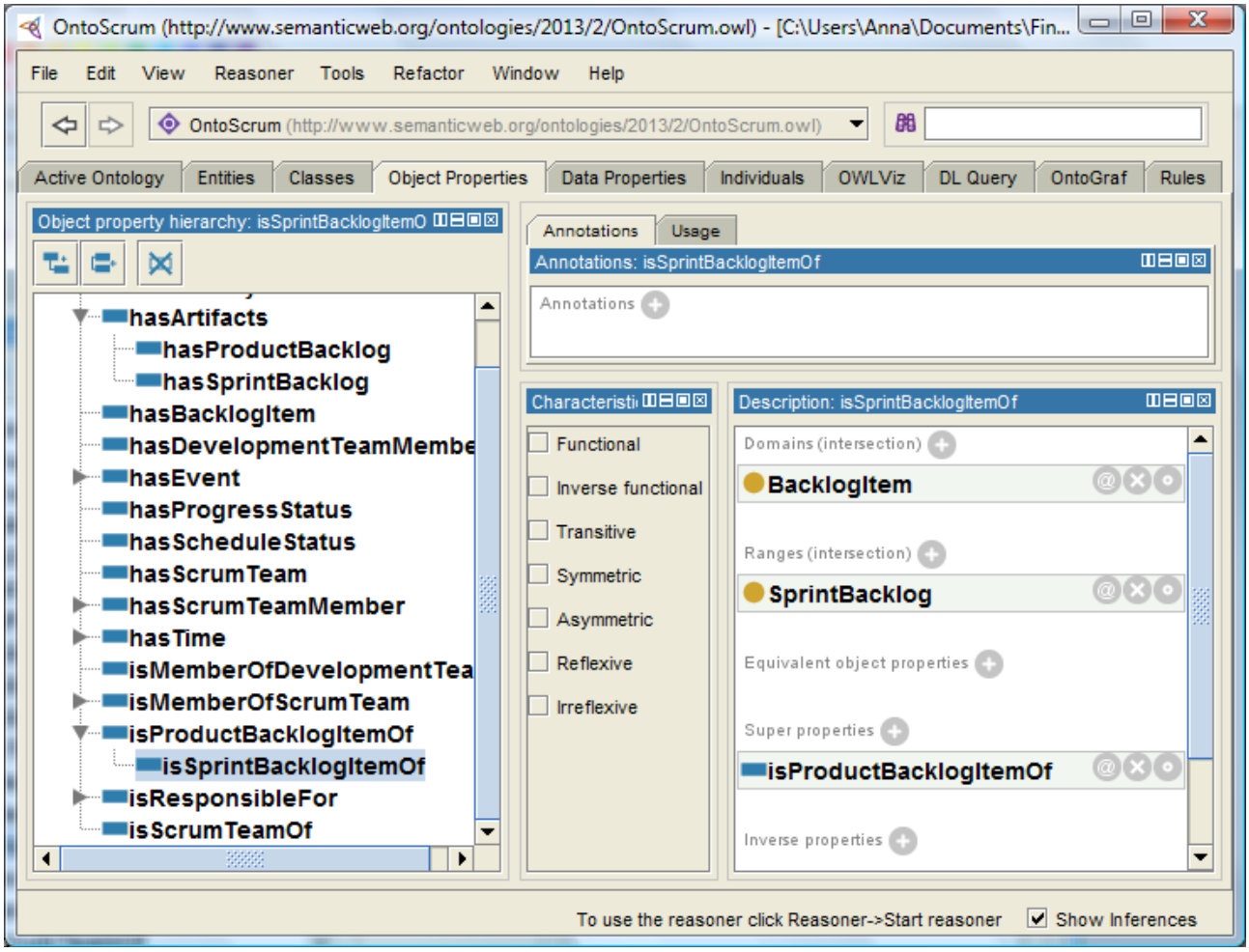


Figura 14: OntoScrum – Propriedade isSprintBacklogItemOf. Fonte: elaborado pelo autor

Na OntoScrum, a modelagem do tempo se dá através de *data properties*, como pode ser visto na Figura 15. A nomenclatura, baseada no PMI (PROJECT MANAGEMENT INSTITUTE, 2008, cap. 6) especifica *ActualTime* e *EstimatedTime*. *EstimatedTime* representa o tempo estimado, antes de uma tarefa ser concluída, e *ActualTime*, o tempo realmente utilizado para aquela tarefa. Tanto *ActualTime* quanto *EstimatedTime* possuem momento de início, o momento de término e a duração, cada um sendo uma propriedade.

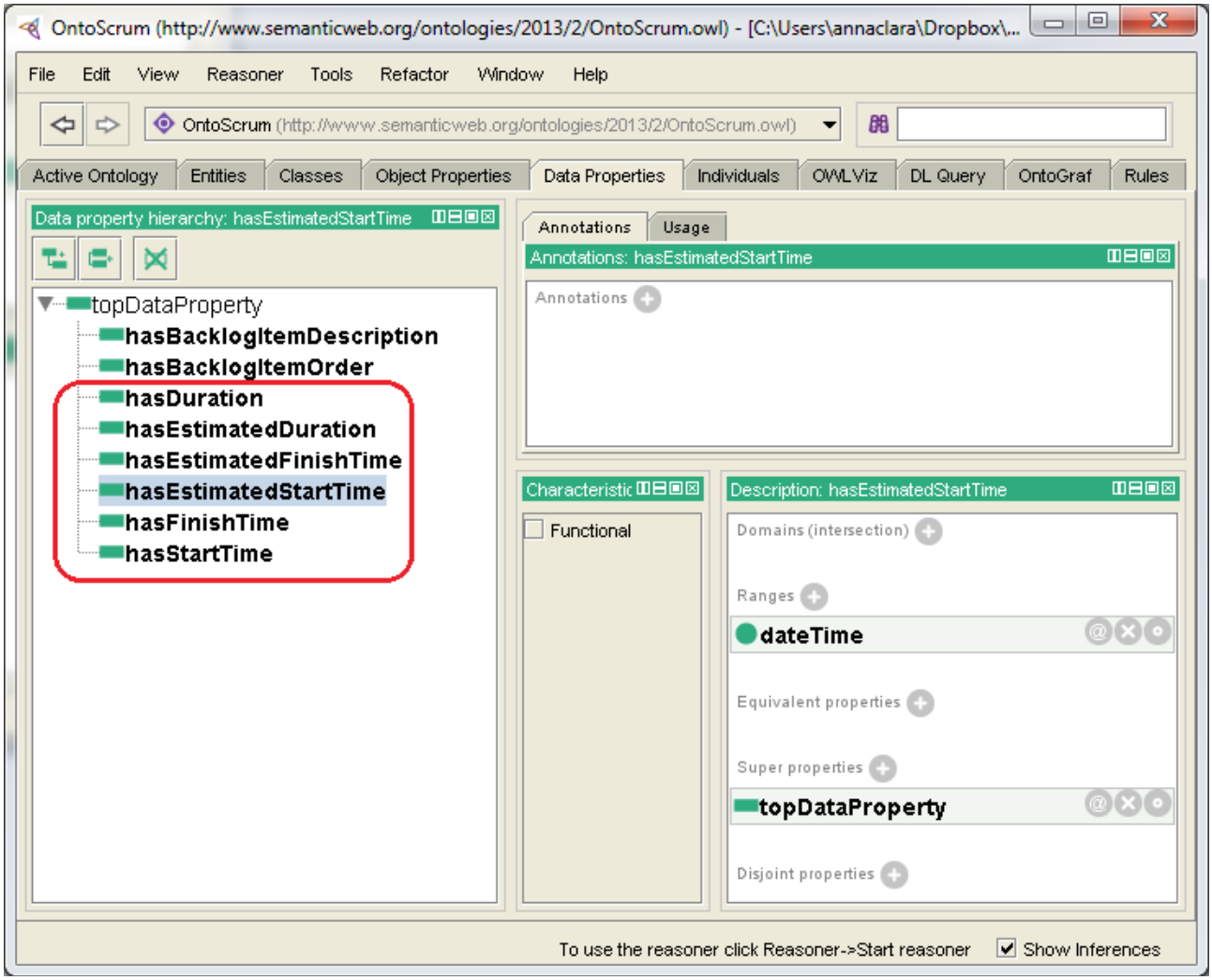


Figura 15: OntoScrum – Tempo estimado e executado. Fonte: elaborado pelo autor

A classe *ProgressStatus* representa o status atual de uma *Activity* ou *BacklogItem*, podendo ser *Done*, para tarefas concluídas, *InProgress*, para tarefas já iniciadas porém não concluídas, e *New*, para tarefas ainda não iniciadas. A classe *ScheduleStatus* indica o status de uma *Activity* ou *BacklogItem*, podendo ser *Delayed*, para tarefas atrasadas ou *OnTime*, para tarefas não atrasadas, como pode-se observar na Figura 16.

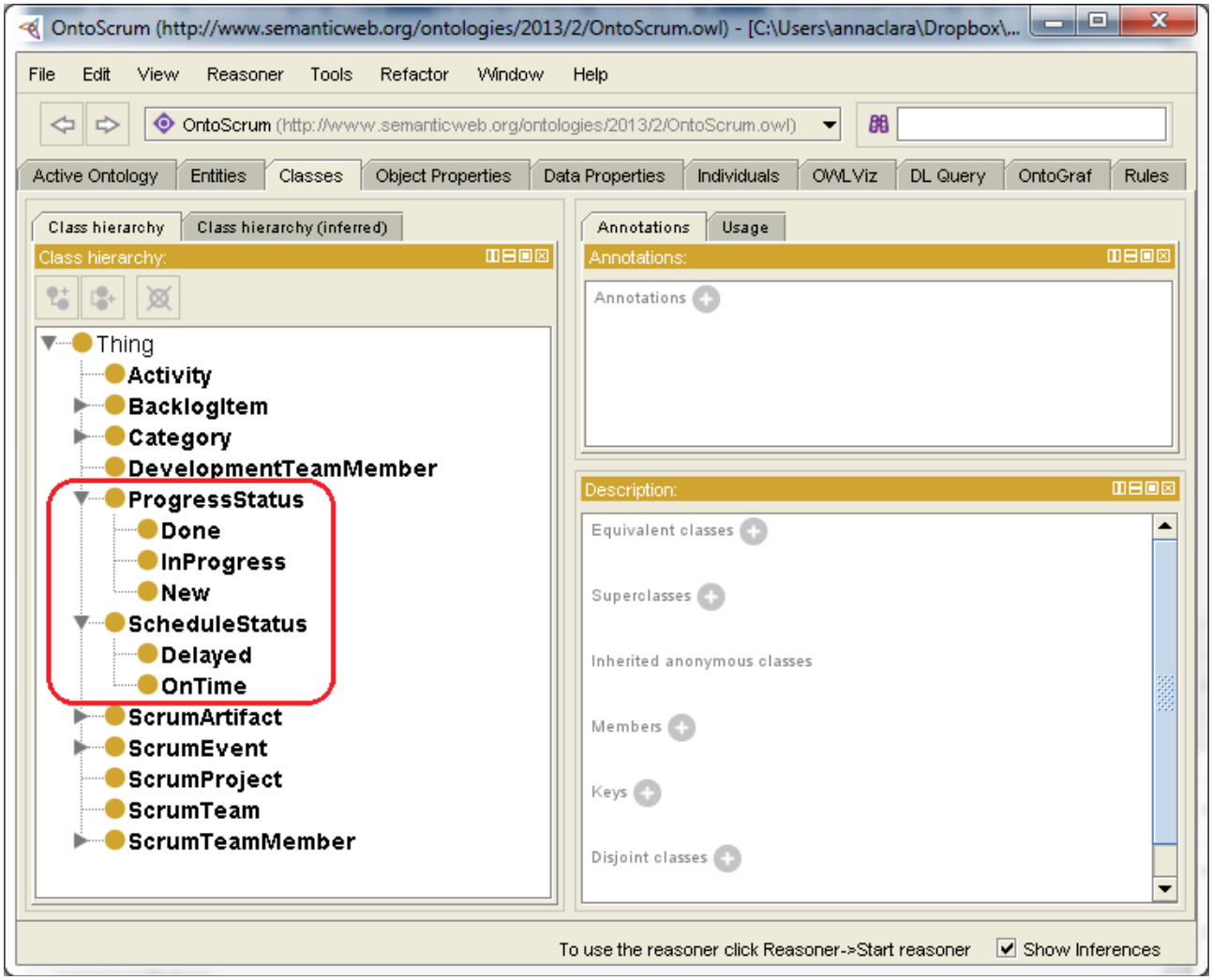


Figura 16: OntoScrum - Progresso das tarefas. Fonte: elaborado pelo autor

Os itens de B*acklog* do *Scrum* podem ser ordenados por valor, prioridade, risco e necessidade. Neste trabalho há apenas uma ordenação. Fica como um indicação de trabalhos futuros implementar as diferentes formas de ordenação. Cada item recebe um número inteiro que representa sua ordem. Itens com ordem mais baixa são executados primeiro. Não podem haver dois itens com o mesmo valor. Quando um item termina de ser executado, seu *ProgressStatus* torna-se *Done*. Ele não é apagado ou modificado, e um outro item não pode assumir seu valor de ordenação. Fica também como indicação de trabalho futuro a implementação de uma melhor forma para ordenar os *Backlog Items*, como uma estrutura de lista encadeada, por exemplo.

Na OntoScrum, como mostra a Figura 16, cada *BacklogItem* possui uma *data property* chamada *hasBacklogItemOrder*, que tem como *range* um número inteiro.

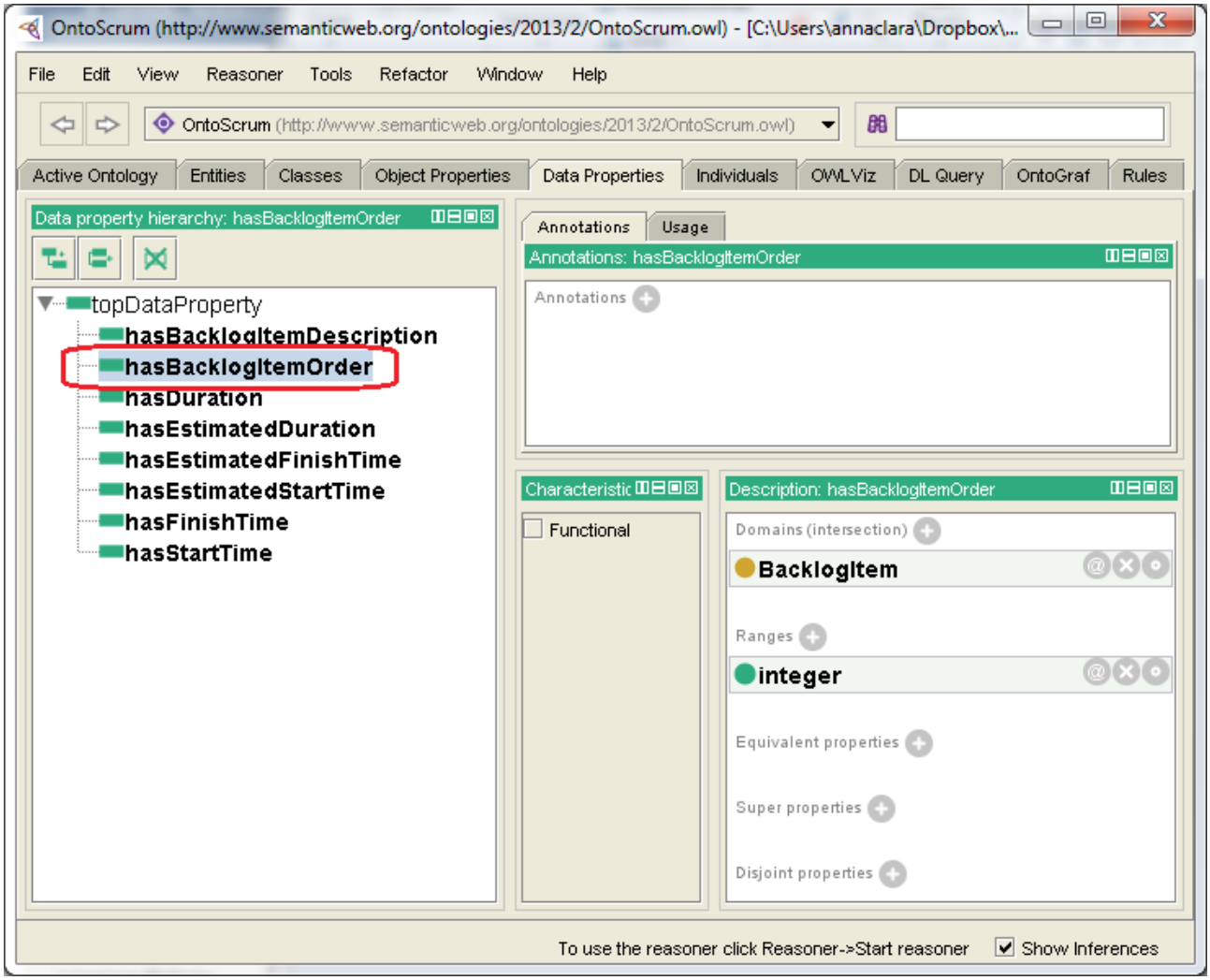


Figura 17: OntoScrum - Ordem de desenvolvimento das tarefas. Fonte: elaborado pelo autor

Para alcançar o objetivo de utilizar inferências sobre a OntoScrum para encontrar atividades similares, a abordagem escolhida foi classificar as atividades por meio de um hierarquia de categorias.

Para realizar certos tipos de inferências foi utilizado o *design pattern* de punning que (SANTOS, 2012) apresenta em seu trabalho. A partir deste *design pattern* é possível realizar inferências tanto ABOX quanto TBOX (SANTOS, 2012) de forma que as possibilidades de

obtenção de resultados são potencialmente maximizadas.

As *object properties* destacadas na Figura 19 fazem com que seja possível organizar as categorias de maneira hierárquica, o que permite que sejam feitas inferências em árvore com facilidade.

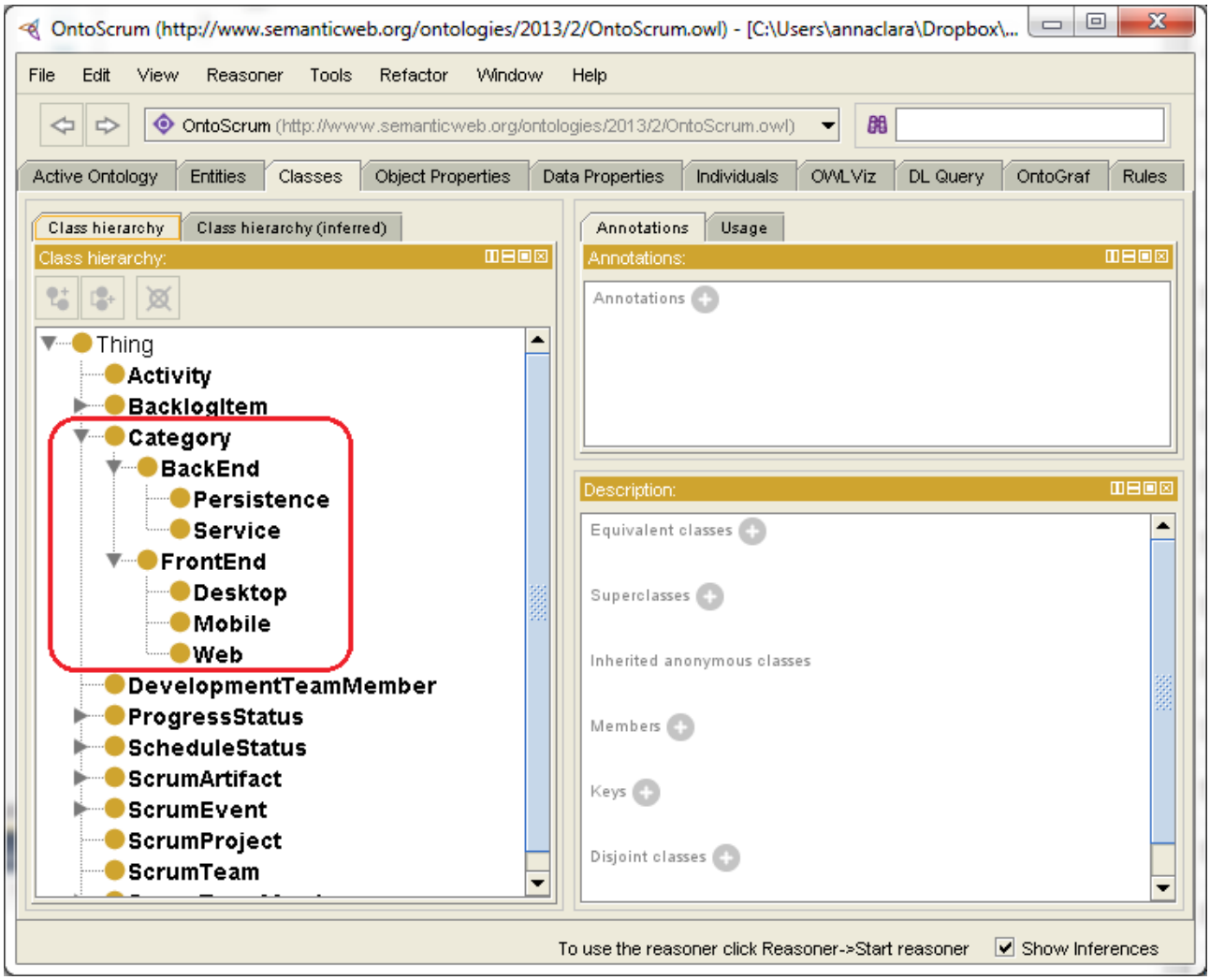


Figura 18: OntoScrum - Classes de categorias. Fonte: elaborado pelo autor

Na Figura 18 é apresentada uma estrutura de categorias bastante sucinta que foi criada para a prova conceito. Entretanto, entende-se que para uma aplicação de uso real, uma categorização detalhada seria necessária para fazer com que inferências interessantes fossem feitas.

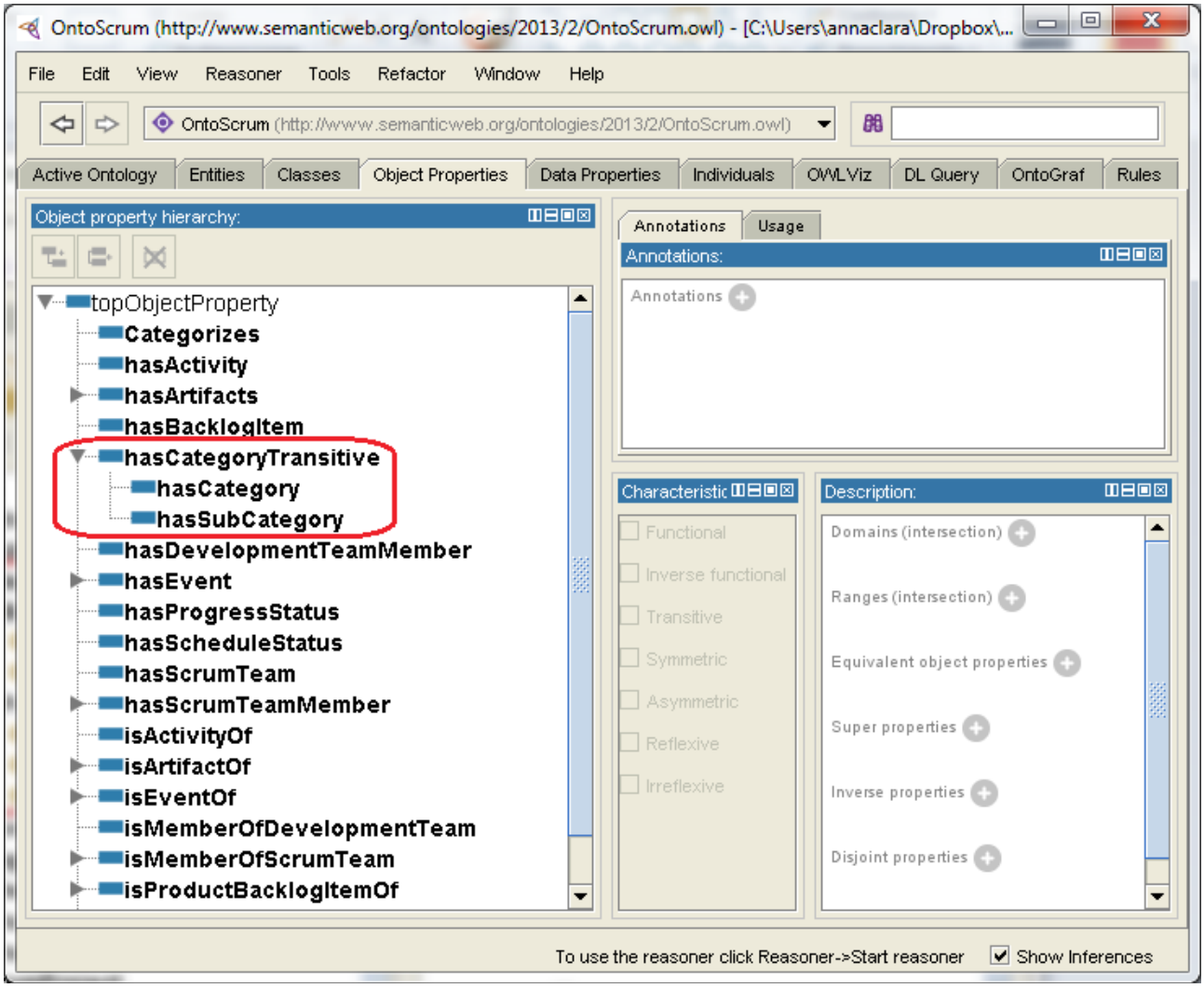


Figura 19: OntoScrum - Propriedades de categorias. Fonte: elaborado pelo autor

4 Prova de conceito

Após a conclusão da modelagem, a OntoScrum foi populada com dados de dois projetos fictícios. Além disto, foram cadastradas algumas categorias para as *Activities*. As categorias cadastradas foram:

- *FrontEnd*
    - *Desktop*
    - *Web*
    - *Mobile*
- *BackEnd*
    - *Service*
    - *Persistence*

Todas as atividades dos dois projetos foram classificadas de acordo com as categorias disponibilizadas.

Os projetos fictícios cadastrados foram um sistema para farmácia e um para livraria. Eles apresentam *BacklogItems* como a criação do cadastro de clientes, criação do cadastro de medicamentos, etc.. Cada um destes *BacklogItems* depende de *Activities,* que podem ser visualizadas na Figura 20, como a criação da camada de persistência de clientes, a criação da camada de serviços de medicamentos, e a criação da interface gráfica da listagem de livros, por exemplo.

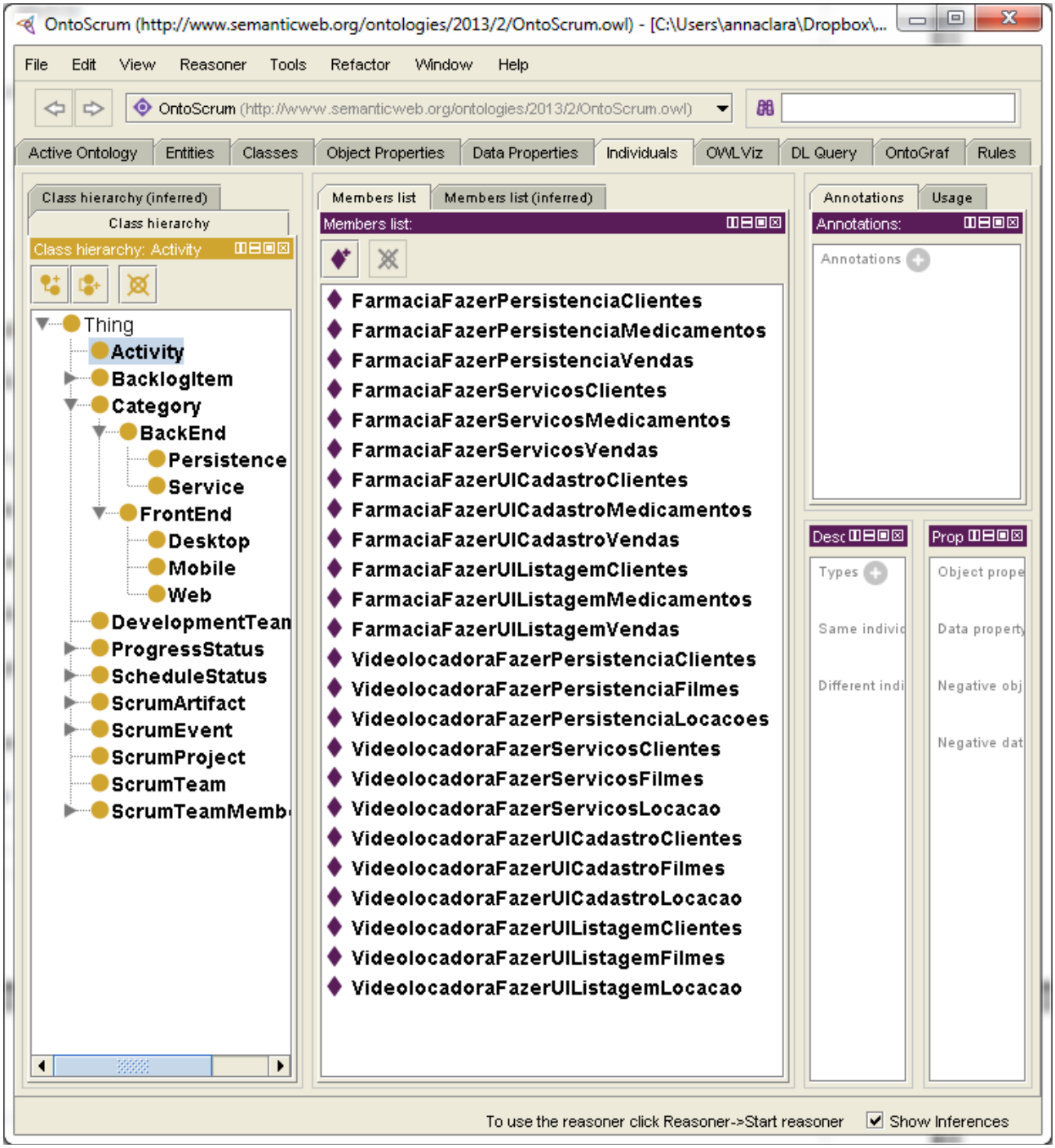


Figura 20: OntoScrum - Activities para a prova de conceito. Fonte: elaborado pelo autor

Para provar a utilidade da abordagem proposta, foram realizadas inferências sobre os dados para dada uma atividade pré-cadastrada, buscar todas as atividades similares de experiências anteriores, suas durações, e a duração média. Estas informações podem então ser usadas para auxiliar a estimativa de tempo necessário para completar uma atividade similar à estas.

```
SELECT  x FROM TABLE(SEM_MATCH(
    '
      ( ?x rdf:type ns:Activity)
      ( ns:FarmaciaFazerPersistenciaClientes ns:hasCategoryTransitive ?y )
      ( ?x ns:hasCategoryTransitive ?y )

    ',
    SEM_Models('ontoscrum'),
    SEM_Rulebases('OWLPRIME'),
    SEM_ALIASES(

SEM_ALIAS('ns','http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#'),
                SEM_ALIAS('owl','http://www.w3.org/2002/07/owl#'),
                SEM_ALIAS('rdfs','http://www.w3.org/2000/01/rdf-schema#')
                )
    ,null)
)
```

Quadro 3: Código SPARQL - Atividades similares

O Quadro 3 mostra o código SPARQL utilizado para buscar todas as atividades de projetos anteriores que são similares à atividade de fazer a camada de persistência de clientes de um sistema de farmácia. O Quadro 4 mostra o resultado desta consulta.

x
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#VideolocadoraFazerPersistenciaFilmes
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#VideolocadoraFazerPersistenciaClientes
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#FarmaciaFazerPersistenciaMedicamentos
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#FarmaciaFazerPersistenciaVendas
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#FarmaciaFazerPersistenciaClientes
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#VideolocadoraFazerPersistenciaLocacoes

Quadro 4: Resultado da consulta do Quadro 3

```
SELECT  x, duracao FROM TABLE(SEM_MATCH(
    '
      ( ?x rdf:type ns:Activity)
      ( ns:FarmaciaFazerPersistenciaClientes ns:hasCategoryTransitive ?y )
      ( ?x ns:hasCategoryTransitive ?y )
      ( ?x ns:hasDuration ?duracao)


    ',
    SEM_Models('ontoscrum'),
    SEM_Rulebases('OWLPRIME'),
    SEM_ALIASES(

SEM_ALIAS('ns','http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#'),
                SEM_ALIAS('owl','http://www.w3.org/2002/07/owl#'),
                SEM_ALIAS('rdfs','http://www.w3.org/2000/01/rdf-schema#')
                )
    ,null)
)
```

Quadro 5: Código SPARQL - Atividades similares e suas durações

O Quadro 5 apresenta um código SPARQL semelhante ao do Quadro 3, com uma linha a mais, `( ?x ns:hasDuration ?duracao)` que faz com que seja buscada também a duração de cada atividade. O Quadro 6 apresenta o resultado desta consulta.

X	DURACAO
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#VideolocadoraFazerPersistenciaClientes	4
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#VideolocadoraFazerPersistenciaFilmes	5
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#VideolocadoraFazerPersistenciaLocacoes	10
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#FarmaciaFazerPersistenciaClientes	6
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#FarmaciaFazerPersistenciaMedicamentos	6
http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#FarmaciaFazerPersistenciaVendas	7

Quadro 6: Resultado da consulta do Quadro 5

```
SELECT avg(duracao) FROM TABLE(SEM_MATCH(
    '
      ( ?x rdf:type ns:Activity)
      ( ns:FarmaciaFazerPersistenciaClientes ns:hasCategoryTransitive ?y )
      ( ?x ns:hasCategoryTransitive ?y )
      ( ?x ns:hasDuration ?duracao)
    ',
    SEM_Models('ontoscrum'),
    --null,
    SEM_Rulebases('OWLPRIME'),
    SEM_ALIASES(

SEM_ALIAS('ns','http://www.semanticweb.org/ontologies/2013/2/OntoScrum.owl#'),
                SEM_ALIAS('owl','http://www.w3.org/2002/07/owl#'),
                SEM_ALIAS('rdfs','http://www.w3.org/2000/01/rdf-schema#')
                )
    ,null)
)
```

Quadro 7: Código SPARQL - Duração média das atividades similares

O Quadro 7 apresenta o código SPARQL que busca a duração média das atividades de projetos anteriores que são similares à atividade de fazer a camada de persistência de clientes de um sistema de farmácia. Esta é uma consulta que pode ser bastante útil em casos em que são encontradas muitas atividades similares, e analisar uma a uma se tornaria inviável. O resultado é apresentado no Quadro 8.

```
AVG(DURACAO)

6.333333333333333333333333333333333333333
```

Quadro 8: Resultado da consulta do Quadro 7

5 Conclusões

No âmbito do presente trabalho buscou-se a formalização do *Scrum* em uma ontologia de aplicação. Depois dos trabalhos de formalização, bem como testes e prova de conceito, chegou-se a uma ontologia que pode facilmente ser usada como um atalho para o conhecimento para aqueles que objetivam enteder o *Scrum*.

Por meio desta ontologia é possível facilmente entender os conceitos, suas relações, propriedades e implicações. Assim, a utilidade de uma ontologia para a definição de um arcabouço de gestão de projetos vais além da simples representação do mesmo, podendo ser considerado como um instrumento de disseminação e evolução do conhecimento ali formalizado.

Ao seu tempo, no capítulo 4 foram criadas instâncias da ontologia de aplicação, as quais atuam como tuplas de bancos de dados. Tal abordagem se mostrou bastante eficiente, uma vez observada a possível realização de inferências na extração de informações, e por meio destas, se ter uma melhor estimativa de tempo necessário em projetos ou *Sprints* futuros.

Assim, com o uso do conhecimento previamente obtido de experiências anteriores, o analista pode lançar mão do uso desta base de conhecimento no refinamento de suas projeções, com isso, obter melhores resultados em projetos de escopo fechado, os quais são em grande parte a realidade das empresas de software de nosso país.

Entretanto, a abordagem apresentada não só auxilia aquelas equipes/empresas que trabalham com projetos de escopo fechado, também podendo servir como ferramental útil para aqueles que trabalham com projetos de escopo aberto, servindo no auxilio do batimento de realidade das estimativas da equipe com as experiências anteriores, assim, servindo como instrumento de mitigação de riscos.

6 Trabalhos Futuros

No presente trabalho foram identificadas diversas oportunidades para trabalhos futuros. Para diminuir a complexidade do desenvolvimento da OntoScrum, Os *Backlog Items* foram modelados de forma a receber números inteiros que representam a ordem em que serão executados. Isto implica que o sistema que for alimentar a ontologia defina a ordem por números inteiros, e não permita que dois itens recebam o mesmo número. Além disso, se um novo item for adicionado, que tenha uma prioridade maior que um item pré-cadastrado, seria necessário alterar o número de diversos itens. Um trabalho futuro poderia modificar a OntoScrum para que utilize uma abordagem mais eficiente, possivelmente uma estrutura de lista encadeada.

Neste trabalho a única forma de ordenação dos *Backlog Items* é por valor. O *Scrum Guide* refere-se também à ordenação por prioridade, risco e necessidade. A OntoScrum poderia ser modificada para aceitar todas estas formas de ordenação.

Quanto à ambiguidade na definição do *SprintBacklog* e do *ProductBacklog*, a escolha feita neste trabalho foi a de criar *Activities* para que um item do *SprintBacklog* não acabasse alterando o *ProductBacklog*. Porém, um trabalho futuro pode resolver este problema de outra maneira.

Um outro trabalho futuro seria expandir as *Categories*, que servem para classificar as *Activities*. Neste trabalho foram criadas poucas *Categories*, apenas para ilustrar o funcionamento da OntoScrum em um ambiente real. Porém, um trabalho futuro bastante interessante seria criar classes na ontologia para os diversos tipos de atividades que existem no desenvolvimento de *software*.

Uma última indicação para trabalhos futuros seria fazer com que algumas datas e durações fossem calculadas automaticamente. Por exemplo, a propriedade *hasDuration* de uma *Activity* poderia ser preenchida pelo sistema, caso estejam disponíveis os valores das propriedades *hasStartTime* e *hasFinishTime*.

7 Referências

AGUAS, R. **Scrum Guide 2011, o que mudou? | Blog ScrumHalf - Gerência de Projetos Ágeis e Scrum - Brasil**. Disponível em: <http://blog.myscrumhalf.com/2011/07/scrum-guide-2011-o-que-mudou/>. Acesso em: 12 nov. 2012.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O.; OTHERS. The semantic web. **Scientific american**, v. 284, n. 5, p. 28–37, 2001.

BERNERS-LEE, TIM; FISCHETTI, M. **Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor**. [S.l: s.n.], [S.d.].

DIPPELREITER, B. Semantic based Project Management. 2008.

DUERST, M.; SUIGNARD, M. **Internationalized Resource Identifiers (IRIs)**. . [S.l: s.n.]. Disponível em: <http://www.ietf.org/rfc/rfc3987.txt>. Acesso em: 9 jul. 2013. , 2005

EVELEENS, J. L.; VERHOEF, C. The Rise and Fall of the Chaos Report Figures. **IEEE Software**, v. 27, n. 1, p. 30-36, 2010.

EWUSI-MENSAH, K.; PRZASNYSKI, Z. H. Learning from abandoned information systems development projects. **Journal of Information Technology**, v. 10, n. 1, p. 3-14, mar 1995.

GRUBER, T. R. A translation approach to portable ontology specifications. **Knowledge acquisition**, v. 5, n. 2, p. 199–220, 1993.

GRUNNINGER, M.; LEE, J. Introduction to the ontology application and design. **Communications of the ACM**, v. 45, n. 2, p. 39-41, 2002.

KEAVENEY, S.; CONBOY, K. **Cost estimation in agile development projects**. Proc. 14th European Conf. Information Systems (ECIS). **Anais**... [S.l: s.n.]. Disponível em: <http://is2.lse.ac.uk/asp/aspecis/20060016.pdf>. Acesso em: 7 ago. 2012. , 2006

LAFUENTE, F. A era do raciocínio artificial. **HSM Management**, n. 86, maio 2011.

MAHNIC, V.; DRNOVSCEK, S. **Agile Software Project Management with Scrum**. EUNIS 2005 Conference-Session papers and tutorial abstracts. **Anais**... [S.l: s.n.]. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.126.901&rep=rep1&type=pdf>. Acesso em: 7 ago. 2012. , 2005

RDF WORKING GROUP. **Resource Description Framework (RDF)**. . [S.l: s.n.]. Disponível em: <http://www.w3.org/RDF>. Acesso em: 9 jul. 2013. , 2004

SANTOS, M. **UM MODELO PARA A GESTÃO COLEGIADA ORIENTADA AO SIGNIFICADO POR MEIO DA REALIZAÇÃO DE PDCAs**. [S.l.]: Universidade Federal de Santa Catarina, 2012.

SCHWABER, K. **Agile project management with Scrum**. Redmond, Wash.: Microsoft Press, 2004.

SCHWABER, K.; SUTHERLAND, J. **Scrum Guide**. . [S.l: s.n.]. Disponível em: <http://www.scrum.org/storage/scrumguides/Scrum_Guide.pdf>. Acesso em: 7 ago. 2012. , 2011

SOWA, J. F. **Principles of ontology**. . [S.l: s.n.]. , 1997

USCHOLD, M.; JASPER, R. **A framework for understanding and classifying ontology applications**. Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden. **Anais**... [S.l: s.n.]. Disponível em: <http://ftp.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-18/11-uschold.pdf>. Acesso em: 21 jun. 2013. , 1999

W3C ® (MIT, ERCIM, KEIO). **Vocabularies**. . [S.l: s.n.]. Disponível em: <http://www.w3.org/standards/semanticweb/ontology>. Acesso em: 9 jul. 2013a. , 2013

W3C ® (MIT, ERCIM, KEIO). **Inference**. . [S.l: s.n.]. Disponível em: <http://www.w3.org/standards/semanticweb/inference>. Acesso em: 21 jun. 2013b. , 2013

W3C®(MIT, ERCIM, KEIO, BEIHANG). **SPARQL 1.1 Overview**. . [S.l: s.n.]. Disponível em: <http://www.w3.org/TR/sparql11-overview/>. Acesso em: 26 jun. 2013. , 2013