Universidad de las Ciencias Informáticas

Facultad 4

Sistema basado en ontologías para la evaluación de productos de software en la UCI

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor**: Sergio Alberto Bustamante Ramírez**

Tutores:

Ing. Aliuska Castañeda Martínez

MSc. Yoan Antonio López Rodríguez

MSc. Yamilis Fernández Pérez

La Habana, 2018

Declaración de Autoría

Declaro por este medio que yo Sergio Alberto Bustamante Ramírez con carnet de identidad 94121427648 soy el único autor del trabajo diploma “Sistema basado ontologías para la evaluación de productos de *software* en la UCI.” Autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo a los ___ días del mes de _____ del año _____.

_____

Sergio Alberto Bustamante Ramírez

_____ _____

Yamilis Fernández Pérez Aliuska Castañeda Martínez

_____

Yoan Antonio López Rodríguez

Agradecimientos

Debo agradecer en primer lugar a mi mamá porque juntos logramos hacer realidad nuestro sueño. Por ser una madre tan especial.

A mi papá por siempre estar a mi lado, dándome tanto apoyo, por saber guiarme y por inspirarme a ser como tú desde niño.

A mi novia, mi princesa porque desde que entraste en mi vida has estado en cada momento a mi lado, por tanto apoyo que me has dado.

A mis tutoras Yamilis y Aliuska por asignarme este tema de tesis y saber guiarme.

A Yoan por asumir con tanta responsabilidad este tema y siempre estar dispuesto a ayudar sin importar la hora.

A todas mis amistades que de una forma u otra formaron parte de este momento.

A mi amigo Leandro por convertirse en el hermano que no tuve.

A la universidad por darme la posibilidad de superarme.

Dedicatoria

A mi familia, a ella me debo.

Resumen

La calidad de un *software* es una preocupación a la que se dedican muchos esfuerzos. Todo proyecto tiene como objetivo producir *software* con la mejor calidad posible y que cumplan con las expectativas del cliente. La presente investigación tiene como precedente la necesidad de la Dirección de Calidad de la UCI de velar por la calidad de los productos de *software*. Se han propuesto ontologías para guiar el proceso de evaluación y el subproceso de prueba de productos de *software*, pero se hace necesario una herramienta para manipularlas y extraer conocimiento implícito. Por esta razón, la investigación que se propone tiene como objetivo crear un Sistema de Información Basado en Ontologías que sea capaz de manipular dos ontologías previamente creadas: la ontología del proceso de pruebas de *software* y la del proceso de evaluación de productos de *software.*

El sistema se desarrolló a partir de tecnologías libres, con XP como metodología de desarrollo, se creó un método que tiene las actividades de importar el modelo ontológico, inferir conocimiento, brindar salidas del sistema y validar el sistema. En la implementación del sistema se utilizó como lenguaje de programación a Java, como entorno integrado de desarrollo el NetBeans y como principales bibliotecas para transformar el modelo ontológico a la programación orientada a objetos Jena y OWL-API.

Esta aplicación permitió la gestión del conocimiento generado en estos procesos facilitando la institucionalización de los mismos.

Palabras claves: Sistema de Información Basado en Ontologías, ontología, evaluación de *software*.

Índice de Contenido

Índice de Contenido

Índice de Tablas

Índice de Figuras

INTRODUCCIÓN

Cada día, aumenta la presencia del *software* [1]en todos los aspectos de la sociedad, convirtiéndose en un producto vital y estratégico para las empresas, organismos y servicios. No puede dejar de mencionarse su impacto en las actividades cotidianas de los ciudadanos, como la toma de decisiones, la gestión de información y del conocimiento.

En la actualidad, con el desarrollo de las tecnologías, el uso y manejo de la información ha cambiado vertiginosamente. Su tratamiento mediante el uso de la computación, permite la creación de sistemas informáticos que ofrecen al ser humano la posibilidad de resolver disímiles problemas, que de forma manual resultarían demasiado complicados y prolongados en el tiempo.

Ha surgido un aumento exponencial de la información producida por las organizaciones, lo que ha impulsado el desarrollo de soluciones cada vez más eficientes para su almacenamiento, actualización y recuperación. Es posible ahora acceder a grandes cantidades de datos, situados en distintos lugares, con múltiples formatos, contextos y plataformas. Y es que la sociedad de la información y el conocimiento, demanda el acceso completo a los mismos. Muchos problemas técnicos han sido solucionados compartiendo eficientemente la información; primero localizando la fuente adecuada, contentiva de los datos necesarios para la tarea dada, para después proceder a su recuperación y filtrado.

Sin embargo, el conocimiento utilizado en los procesos de toma de decisiones no ha experimentado un incremento proporcional al de la información disponible. El descubrimiento de nuevo conocimiento impulsa, por tanto, la investigación en métodos avanzados de búsqueda, integración y procesamiento de información, así como de gestión del conocimiento (GC) (Fernández Pérez, 2018).

La GC es un conjunto de procesos sistemáticos que transita desde una etapa de identificación y captación del capital intelectual, a otra de tratamiento, desarrollo, compartimiento y utilización del conocimiento. Este proceso está orientado al desarrollo

[1] Conjunto de programas y rutinas que permiten a la computadora realizar determinadas tareas.

organizacional y personal, lo que trae como consecuencia la generación de una ventaja competitiva para la organización y el individuo (Fernández Pérez, 2018).

Existe un creciente interés sobre la gestión del conocimiento en innumerables actividades del ser humano, entre ellas se encuentran todas aquellas relacionadas con el sector industrial, el comercio, la salud, la educación, el transporte o el medio ambiente. Dentro del sector industrial, resulta vital gestionar adecuadamente el conocimiento en el proceso de desarrollo de *software* (Canals, 2003), tanto el que se da de manera explícita como el tácito o implícito.

El conocimiento explícito es la información mostrada de manera asequible, es fácil de encontrar, generalmente reside en las bases de datos. El tácito o implícito, más difícil de capturar, se basa más bien en la experiencia, está guiado por el contexto, y por lo general, reside en los individuos (Barchini, y otros, 2007).

Dentro del proceso de desarrollo del *software*, las actividades de aseguramiento de la calidad del producto y del proceso han recobrado notable importancia. Esto se debe al incremento de la complejidad del *software*, lo que provoca entre las entidades productoras y comercializadoras del mismo un especial interés por la calidad, convirtiéndose así en uno de los elementos diferenciadores entre las diversas compañías a nivel mundial por las ventajas competitivas que puede aportar. La búsqueda de la calidad de los sistemas, ha propiciado la creación de modelos, estándares, marcos de trabajo, metodologías y sistemas para evaluarla, asegurarla y controlarla. Este trabajo se orienta, específicamente, hacia la evaluación de la calidad de productos de *software*.

La evaluación es un proceso costoso y complejo por la cantidad de recursos involucrados, resultando de importancia capital para la toma de decisiones, ya que una evaluación equívoca puede provocar cuantiosos gastos monetarios innecesarios a la entidad donde se implanta. De ahí la relevancia de lograr una adecuada correlación entre los resultados obtenidos al evaluar un producto *software* con la calidad que muestra en la realidad.

Otro factor esencial es la correcta determinación de los indicadores de problemas importantes, algo que puede implicar un refinamiento del proceso de desarrollo, reducción de costos y mejora de la competitividad de las empresas. Es primordial también, lograr que la evaluación de la calidad de un *software* se traduzca en valores que puedan ser

comparados e integrados con criterios de costo y beneficio (Fernández Pérez, 2018). En este proceso es necesario hacer una correcta gestión del conocimiento para garantizar su buen desempeño.

La evaluación de la calidad de *software* ha sido durante muchos años un área de conocimiento reservada a expertos y profesionales. Sin embargo, dicha experiencia está siendo recopilada en multitud de artículos, estándares y libros, e implementada en varios sistemas a los que cualquiera puede acceder (IEEE Computer Society, 2009.), (ISO/IEC, 2008). Actualmente se dispone de suficientes técnicas, herramientas y habilidades para poder desarrollar una representación del conocimiento generado en dicho proceso. Es por esta razón, que en la evaluación de la calidad de productos de *software* se necesita contar con métodos y herramientas que la faciliten.

De aquí se deriva la necesidad de capturar y gestionar el conocimiento disponible en un formato y representación adecuados. Esto puede realizarse para la mayoría de áreas de conocimiento por los expertos, donde pueden definirse un conjunto de conceptos, aserciones, reglas e inferencias. La información resultante puede guardarse utilizando alguna de las representaciones de conocimiento existentes (Fernández Pérez, 2018).

Una de las formas de representación del conocimiento más empleadas actualmente son las ontologías, las cuales ofrecen disímiles ventajas para la modelación, generación, distribución y uso del conocimiento producido y acumulado en las organizaciones (Ferreira de Souza, 2014.).

Las ontologías son herramientas útiles para la comunicación entre expertos en un área determinada. Pueden ayudar en la generación de consenso entre especialistas sobre el vocabulario técnico usado en las interacciones. De igual forma, pueden emplearse para la formalización del conocimiento eliminando las contradicciones e inconsistencias en un dominio para que sean automáticamente verificadas y validadas por computadores. Finalmente, como representación del conocimiento, las ontologías conforman un vocabulario de dominios que puede ser reutilizado en distintas aplicaciones y para propósitos específicos (Echeverría Pérez, y otros, 2012).

Para la recuperación y comunicación del conocimiento de una manera genérica y fácil, a través de las ontologías, es necesario el desarrollo de un sistema de información (SI).

Un SI puede definirse técnicamente como el conjunto de componentes interrelacionados que permiten capturar, procesar, almacenar y distribuir información para apoyar la toma de decisiones y el control en una organización (Barchini, y otros, 2007).

Esto ha llevado a la noción de Sistemas de Información Basados en Ontología (SIBO), un concepto que, aunque en una fase preliminar de desarrollo, cubre las dimensiones conceptuales y técnicas de los SI. Las ontologías y SIBO están desarrollándose y empleándose en una variedad de áreas de aplicación emergentes, tales como el modelado de empresas, diagnósticos, toma de decisión, planeamiento y adaptación, modelado de procesos y sistemas (Barchini, y otros, 2007).

Debido a los beneficios que reporta el uso de las ontologías y SIBO, en nuestro país se han realizado diversos trabajos orientados a su utilización.

La industria cubana de *software*, con la Universidad de las Ciencias Informáticas (UCI) a la vanguardia, trata de mantenerse al corriente de las políticas y estándares establecidos para el desarrollo de aplicaciones informáticas. En este centro de altos estudios, existen varios centros productivos dedicados a la satisfacción de las necesidades de los clientes con los que mantienen contratos. Además, se encuentra una dirección de calidad, encargada de velar por la calidad del desarrollo de *software*, involucrando la calidad del producto y del proceso.

En la UCI, el proceso productivo se encuentra certificado por CMMI [2]nivel 2. Dentro de este nivel, se definen las actividades de aseguramiento de la calidad del producto y del proceso (PPQA), las cuales deben estar institucionalizadas.

Tanto en los centros de desarrollo como en la dirección de calidad, hay fluctuación del personal, elemento que se determina en entrevistas realizadas, por lo que se hace difícil conservar el conocimiento tácito. Además, alarga el tiempo dedicado a la preparación del personal.

[2] es un modelo para la mejora y evaluación de procesos para el desarrollo, mantenimiento y operación de sistemas de software.

En cuanto a la evaluación de la calidad del producto, este proceso se encuentra en definición, lo que resulta bastante complejo, debido al volumen de información y al conjunto de conceptos ambiguos y solapados que se generan (Castañeda Martinez, y otros, 2017). Esto trae consigo una extensión de la duración de los procesos y errores que se repiten de un producto a otro. Además, existe dificultad en la comunicación entre los actores implicados en el proceso de evaluación del producto, en el reúso del conocimiento generado de una evaluación a otra, debido a que no se integra la información obtenida de fuentes de conocimiento heterogéneas. También, se entorpece el proceso de toma de decisiones en cuanto a equipos y directivos involucrados en el proceso de evaluación.

Se han desarrollado ontologías para la evaluación de productos y el subproceso de pruebas, las cuales representan el conocimiento generado en dichos procesos. Pero el uso y manejo de estas se obstaculiza debido a la necesidad de dominar SPARQL [3]y de aprender a utilizar los razonadores para obtener el conocimiento implícito. Además, no se dispone de un SI que permita la integración de las diferentes ontologías.

Al realizar el estudio de los sistemas de información (SI) existentes para la manipulación de las ontologías, es decir SIBO, se concluyó que estos están centrados en dominios específicos y su programación es dependiente del dominio. Es preciso aclarar que los SIBO seleccionados fueron aquellos que tienen disponible su documentación.

Luego del análisis realizado, puede concluirse que no existe una plataforma capaz de manipular la información generada en los procesos de pruebas y evaluación de productos en la UCI. Por ende, se hace necesario el desarrollo de un sistema que permita la representación, recuperación e inferencia del conocimiento generado en dichos procesos de la Dirección de Calidad.

Por lo expuesto anteriormente se define como **problema de la investigación** ¿Cómo contribuir a la gestión del conocimiento generado en el proceso de evaluación de productos de *software* en la Dirección de Calidad de la UCI con la ayuda de las Tecnologías de la Información y las Comunicaciones (TIC)?

[3] es un lenguaje de consulta para datos RDF.

Se determina como **objeto de estudio:** Los Sistemas de Información Basados en Ontologías (SIBOs) y enmarcando como **campo de acción:** SIBO para proceso de evaluación de productos de *software*.

Para dar solución al problema de investigación se define como **objetivo general**: Desarrollar un SIBO para la gestión del conocimiento generado en el proceso de evaluación de productos de *software* en la Dirección de Calidad de la UCI.

Para dar cumplimiento al objetivo general se plantean los siguientes **objetivos específicos:**

- Elaborar el marco teórico de la investigación a través del estudio del estado del arte sobre las tendencias actuales de los SIBO.
- Seleccionar las metodologías, herramientas y tecnologías para el desarrollo del SIBO.
- Implementar un SIBO para la gestión del conocimiento generado en el proceso de evaluación de productos de *software* en la Dirección de Calidad de la UCI.
- Validar el SIBO, haciendo uso de los métodos definidos en la investigación.

Entre los **métodos científicos** utilizados en la investigación se destacan los mencionados a continuación. Además, se brinda una breve explicación de los fines para los que fueron utilizados.

Entre los **métodos teóricos** se tienen**:**

- El **histórico-lógico** para la revisión de la bibliografía existente y poder fundamentar la evolución de las ontologías y los SIBO, así como las técnicas, metodologías, y herramientas para su implementación.
- El **analítico-sintético** al descomponer los conceptos relacionados con las ontologías y los SIBO en elementos por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución propuesta.
- El **método de modelación** para el análisis, diseño e implementación del SIBO.

Entre los **métodos empíricos** se tienen:

- El **método experimental** para comprobar la utilidad de los resultados obtenidos a partir de la aplicación del sistema de información en un entorno real.

- La **observación**, la cual permite observar cómo se comporta el sistema propuesto y para la realización del diagnóstico.

Estructura del trabajo diploma:

El presente trabajo de diploma consta de introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas, además de los anexos que complementan el cuerpo del trabajo.

Para una mejor comprensión se presenta una breve descripción de cada capítulo.

Capítulo I: Se define el marco teórico de la investigación; así como las tecnologías, herramientas y metodologías para llevar a cabo un SIBO para la evaluación de productos de *software*.

Capítulo II: Diseño y análisis, en el cual se definen las Historias de Usuarios (HU) como las principales funcionalidades que debe cumplir el sistema, además de los requerimientos mínimos para su desarrollo. También se elaboran las tarjetas CRC, se explica la arquitectura utilizada y los principales patrones de diseño utilizados.

Capítulo III: Implementación y prueba, se aplican métodos, técnicas y los casos de prueba relacionados con la implementación; se realizan las pruebas unitarias y de aceptación para lograr el desarrollo de un sistema con la calidad requerida.

CAPÍTULO I: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

El objetivo fundamental de este capítulo es definir el marco teórico de la investigación, centrándose en sus principales conceptos y definiciones. Se enfatiza en las ontologías y SIBO. Además se realiza un estudio de SIBOs existentes y se expone la metodología de desarrollo de *software*, las técnicas y herramientas que sirven para la implementación de la propuesta de solución.

1.2 Proceso de evaluación de la calidad de productos de software

Es importante comenzar con el análisis de los conceptos de calidad, calidad de *software* y evaluación.

Según el diccionario de la Real Academia Española de la Lengua (Real Academia Española, 2018), la palabra calidad proviene del latín *qualitas* (cualidad), definida como: "Propiedad o conjunto de propiedades inherentes a algo, que permiten juzgar su valor" (Real Academia Española n.d.). Visto así, la calidad puede aplicarse a cualquier contexto en que se desee hacer un juicio o establecer una comparación entre dos o más cosas. Como es posible observar, el concepto tiene un alto grado de subjetividad porque depende del punto de vista de quien la valore y del patrón con que se juzgue.

De tal modo, puede plantearse que, en la concepción de la calidad, influyen diversos factores o dimensiones. Es un concepto multifactorial, en el cual la dimensión técnica abarca los detalles científicos y tecnológicos relacionados con el producto, y la dimensión humana incluye la satisfacción del cliente, influida por sus necesidades, expectativas y cultura. Por último, está la dimensión económica, que comprende la reducción de los costos, tanto para el productor, como para el cliente.

Al particularizar a la industria del *software*, lo primero que debe quedar claro es el propio concepto de *software*. Muchas personas asocian *software* solo con programas de computadoras. El *software* es más amplio, e incluye la documentación asociada y la configuración de datos necesarios para hacer que estos programas operen de manera correcta.

Un *software* de calidad es aquel que cumple su función correctamente, de manera eficiente, flexible, confiable, fácil de mantener y usar, portable y seguro. Su calidad la determina un conjunto de cualidades que formaliza su utilidad y existencia, reforzando, además, su carácter multidimensional. No obstante, existen elementos subjetivos que dificultan la medición; por ejemplo, si es atractivo, fácil de usar y aprender y satisface las necesidades del cliente. Si se analizan los diferentes conceptos expuestos por Pressman (Preesman, 2010), y los estándares internacionales IEEE 610 (IEEE Computer Society, 2009.) e ISO 25000 (ISO/IEC/ONN-NC, 2011) se aprecia coincidencia con esta definición. Las anteriores insisten en el cumplimiento de tres puntos importantes para lograr la calidad. El primero, es el cumplimiento de los requisitos del *software*; la falta de concordancia con los requisitos es una falta de calidad; los requisitos son la base de las medidas de la calidad. El segundo, es el cumplimiento de un conjunto de requisitos implícitos o subjetivos (referente a la interfaz, satisfacción); si el *software* se ajusta a sus requisitos explícitos, pero falla en alcanzar los requisitos implícitos, la calidad del *software* se debilita. El tercero, se refiere al cumplimiento de los estándares especificados, que definen un conjunto de criterios y buenas prácticas que guían la forma de aplicación de la ingeniería de *software* en el desarrollo del producto; cuando se irrespeten los mismos, habrá falta de calidad (Fernández Pérez, 2018).

Si se analiza el concepto de evaluación, según la Real Academia (Real Academia Española, 2018) Es la acción de estimar, apreciar, calcular o señalar el valor de algo. Al evaluar se emite un juicio o valoración sobre algo, teniendo como base la información del objeto a valorar (evidencias, mediciones) y un conjunto de criterios como patrón. Con el resultado de la evaluación, se toman decisiones y se conduce hacia la mejora continua. Evaluar conlleva una valoración subjetiva (cualificación) y objetiva (cuantificación). En el caso de la industria de *software*, suele asociarse de manera errónea la evaluación de la calidad con las pruebas de *software*. Evaluar es más que probar, es realizar la valoración de un producto a partir de los resultados obtenidos en el proceso de Verificación y Validación (V&V); se analizan los resultados y se continúa con la toma de decisión.

La evaluación de la calidad del producto de *software* tiene como propósito emitir un juicio de cuán bueno puede ser este para una función determinada. Permite seleccionar entre dos o más alternativas; predecir los valores de las características de calidad y conceptualizar un nuevo producto.

Como resultado del proceso de evaluación, se obtiene un valor global de la calidad (cuantificación) y una valoración (cualificación), que permite aislar los problemas que pudiera presentar el producto. Dicho proceso es esencial para la toma de decisiones por las empresas productoras, evaluadoras y las que adquieren el *software*.

La evaluación, como cualquier proceso, posee un conjunto de actividades interrelacionadas, que transforman elementos de entrada en salidas. En este caso, tiene como entrada los productos, que se conducen por un flujo de actividades, entre las que se incluyen la medición y el análisis, y obtiene como salida el resultado de la valoración de la calidad. Para evaluar y juzgar la calidad, se necesita determinar los criterios que caracterizan la calidad del producto. En el *software*, dichos criterios se encuentran estructurados y organizados en los modelos de calidad. Además, están los estándares ISO 14598 y 25040 (ISO/IEC/ONN-NC, 2011), que ofrecen un *framework* para la evaluación, que detalla el proceso a través de un flujo de actividades y tareas.

El modelo de calidad, representa los criterios a tener en cuenta en el proceso de evaluación, organizados de forma jerárquica. Asociado a estos, se encuentran las medidas que permiten valorarlos. Las medidas provienen de diferentes fuentes: pruebas automáticas, encuestas a expertos, listado de No Conformidades (NC), entre otras.

El proceso lo ejecutan los evaluadores, involucrando a los clientes, y se realiza sobre la base de la ISO 25040 (ISO/IEC/ONN-NC, 2011). Tiene como entrada los productos a evaluar y como salida el índice de calidad de cada uno, ordenado de manera descendente. Para la evaluación, se necesitan recursos como el modelo de calidad, y el módulo de evaluación, consistente en las herramientas, técnicas y medidas de los atributos de calidad.

Este es un proceso complejo y costoso, que implica el uso de gran cantidad de recursos, tanto materiales como humanos. Por esa razón, es preciso lograr una relación entre el resultado de la evaluación y la calidad mostrada por el producto en explotación. De ahí la importancia de la fiabilidad de los resultados, sobre todo cuando conlleva a la certificación de un producto de *software*.

Su complejidad viene aparejada a la variedad de información que maneja, procedente de diversas fuentes, con distintos puntos de vista y suposiciones acerca del dominio de estudio. También, la existencia de información vaga, incompleta, conceptos imprecisos y presencia

definitoria del juicio humano en la decisión. Esto provoca problemas de comunicación por falta de entendimiento compartido. A partir del análisis documental realizado a los estándares e investigaciones del área de la evaluación de la calidad, se ha determinado que frecuentemente se presentan inconsistencias entre diferentes términos (Castañeda Martinez, y otros, 2017). En particular, no existe consenso sobre los conceptos y terminologías utilizadas en esos campos. Hay principios y métodos que aún están siendo definidos y consolidados.

Todo lo anterior, conlleva a la búsqueda de técnicas y herramientas que proporcionen un vocabulario común para resolver el problema de integridad e inconsistencia, identificado. Para brindar apoyo a dicho proceso, se decide desarrollar ontologías a los subprocesos involucrados en él. Así estas ontologías sirven de guía a dichos subprocesos, logrando crear en esta mayor confiabilidad. Además de ahorrar tiempo y esfuerzo a los especialistas involucrados en los procesos.

1.3 Ontologías

Las ontologías son tan antiguas como la disciplina de la filosofía. Se definió en la Inteligencia Artificial (IA) como: “Una ontología define los términos básicos y relaciones que conforman el vocabulario de un área específica, así como las reglas para combinar dichos términos y las relaciones para definir extensiones de vocabularios”. (Guarino, 1998)

Una de las definiciones más extendidas es la dada por Tom Gruber (Gruber, 1993): "Una ontología es una especificación explícita de una conceptualización”. El término proviene de la filosofía, donde una ontología es un recuento sistemático de la existencia. En sistemas de IA, lo que existe es lo que puede ser representado. Cuando el conocimiento de un dominio se representa mediante un formalismo declarativo, el conjunto de objetos que puede ser representado se llama universo del discurso. Esos conjuntos de objetos, y las relaciones que se establecen entre ellos, son reflejados en un vocabulario con el cual representamos el conocimiento en un sistema basado en conocimiento. Así, en el contexto de IA, podemos describir la ontología de un programa como un conjunto de términos. En tal ontología, las definiciones asocian nombres de entidades del universo del discurso con textos comprensibles por los humanos que describen el significado de los nombres, y

axiomas formales que limitan la interpretación y buen uso de dichos términos. Formalmente, una ontología es una teoría lógica.

En la disciplina de los SI se la considera como: "un artefacto del *software* (o lenguaje formal) diseñado para un conjunto específico de usos y ambientes computacionales" (Barchini, y otros, 2007).

En este trabajo la definición que se usa es una unión de la Gruber y la dada para los SI. Ontología es un artefacto diseñado para un conjunto específico de usos y ambientes computacionales, la cual especifica explícitamente una conceptualización, en este caso el proceso de evaluación de productos de *software*.

Según varios autores (Barchini, y otros, 2007) las ontologías presentan varios componentes, estos son:

- Conceptos: Son las ideas básicas que se intentan formalizar. Los conceptos pueden ser clases de objetos, métodos, planes, estrategias, procesos de razonamiento.
- Relaciones: Representan la interacción y enlace entre los conceptos, formando la taxonomía del dominio. Las relaciones básicas son: sub-clase-de, parte-de, conectada-a.
- Instancias: Se utilizan para representar objetos determinados de un concepto.
- Axiomas: Son teoremas que se declaran sobre relaciones que deben cumplir los elementos de la ontología. Especifican las definiciones de los términos en la ontología y las restricciones de sus interpretaciones. Los axiomas deben proveerse para definir la semántica o el significado de los términos.

En la literatura se realizan diferentes clasificaciones a las ontologías (Barchini, y otros, 2009). Entre estas podemos encontrar:

- Ontologías para la representación de conocimiento: Permiten explicar las conceptualizaciones que subyacen de los formalismos de representación de conocimiento.
- Ontologías genéricas: Definen conceptos considerados genéricos en diferentes áreas, describe la categoría más ato del nivel, describiendo conceptos generales (como tiempo, espacio u objeto). Estas ontologías son reutilizables en diferentes dominios. Se

llaman también ontologías abstractas o superteorías porque permiten definir conceptos abstractos, y dichas ontologías pueden ser usadas para definir conceptos de forma más específica en diferentes dominios.

- Ontologías del dominio: Definen conceptualizaciones específicas del dominio.
- Ontología de la aplicación: Están ligadas al desarrollo de una aplicación concreta. Tales ontologías cubren los aspectos relacionados con aplicaciones particulares. Típicamente, estas ontologías toman conceptos de ontologías del dominio y genéricas, así como métodos específicos para realizar la tarea, por lo que no son muy adecuadas para ser reutilizadas.

Las ontologías desarrolladas en la dirección de calidad y que son utilizadas en este trabajo de diploma se clasifican como de aplicación. El sistema que se desarrolló en esta investigación se basa en el conocimiento representado en esas ontologías.

1.4 Sistemas de Información Basados en Ontologías

Según Graciela Barchini y su equipo, los SI son esencialmente artefactos del conocimiento que capturan y representan el conocimiento sobre ciertos dominios (Álvarez, y otros).

Los SI ofertan, regulan y gestionan todo tipo de recursos de información. Con este objetivo se producen los procesos de almacenamiento, identificación, transformación, organización, tratamiento y recuperación de la información. En estos pasos o fases interviene la tecnología, que facilita el cumplimiento de los usos y funciones de la información. Como resultado se generan cambios en el estado del conocimiento que poseen las personas, la solución de problemas informativos, o la toma de decisiones operacionales (Díaz, y otros, 2009).

Los SI son esencialmente artefactos de conocimiento que capturan y representan el conocimiento sobre ciertos dominios. Los profesionales e investigadores de los SI han tratado tradicionalmente con los problemas de identificar, capturar, y representar el conocimiento del dominio dentro de los SI (Barchini, y otros, 2006).

Según Graciela Barchini y su equipo (Barchini, y otros, 2006) las cuatro principales funciones del SI son:

- Recogida de la información: Es la actividad de registrar o captar información para que pueda utilizarse con posterioridad. El problema principal radica en la creación de un soporte físico adecuado y la elección de un código eficiente para su representación.
- Acopio o acumulación: consiste en la agrupación de la información recogida en lugares y momentos diferentes.
- Tratamiento de la información: En él se pueden distinguir tres operaciones fundamentales: de ordenamiento, de cálculo aritmético-lógico y de transferencia de información. Una vez transformada la información, ella debe cumplir con una serie de requisitos de los cuales los más relevantes son: claridad, precisión, ser oportuna, directamente utilizable, coordinada, completa, jerarquizada, sintética y necesaria. Aunque, en la mayoría de los casos, la información adolece de defectos, de los cuales los más comunes son: proliferación excesiva, anarquía, lentitud de avance y tendencia a la aproximación.
- Difusión de la información: El problema de la difusión consiste en dar respuesta a tres preguntas fundamentales: cómo, cuándo y a quién.

La nueva generación de los SI deberá ser capaz de trabajar con la semántica de la información, en la cual un hecho puede ser más que una descripción, para poder hacer un buen uso de las informaciones disponibles como la llegada de Internet y la computación distribuida (Barchini, y otros, 2007).

Las ontologías generalmente se usan para especificar y comunicar el conocimiento del dominio de una manera genérica y son muy útiles para estructurar y definir el significado de los términos (Barchini, y otros, 2006).

Según Barchini (Barchini, y otros, 2009) las ontologías permiten crear un entendimiento compartido al unificar los diferentes puntos de vista y sirven para:

- Facilitar la comunicación entre los actores implicados en la construcción de los SI.
- Permitir el reúso del conocimiento de un dominio.
- Facilitar la recuperación, integración e interoperabilidad entre fuentes de conocimiento heterogéneas.
- Proveer una base para la representación del conocimiento del dominio y ayudar a identificar las categorías semánticas del dominio.

Usar ontologías en el desarrollo de un SI va a traer consigo mejor calidad a la hora de presentar el producto final.

Las ontologías y los SIBO están desarrollándose y aplicándose en una variedad de áreas de aplicación emergentes tales como modelización de empresas, diagnósticos, toma de decisión, planeamiento y adaptación, modelado de procesos y sistemas. Mención especial merece la Web semántica que aborda los problemas semánticos que plantea cualquier tipo de comunicación mediante la interacción, principalmente de agentes y ontologías (Barchini, y otros, 2009).

1.5 Estudio de aplicaciones similares

Se estudiaron algunos SIBOs existentes, donde se analizó e investigó cuales fueron las principales herramientas utilizadas para su creación, además de representar las funciones que realizan estos sistemas.

Se realizó un estudio tanto a los sistemas extranjeros como nacionales. A continuación se exponen los principales.

1.5.1 Sistemas foráneos

Se realizó un análisis a SIBOs foráneos donde se obtuvo información de sus aplicaciones hoy en día.

- **Multilingual Knowledge Based European e-Market (MKBEEM)**

Se trata de un portal multilingüe de comercio electrónico que combina procesamiento basado en Ontologías y procesamiento de lenguaje natural con el fin de proporcionar servicios multilingües de mediación para el comercio electrónico (Leger, y otros, 2001). La combinación del procesamiento del lenguaje natural con ontologías desempeña un papel decisivo en:

- La creación de una correspondencia entre una consulta en lenguaje humano y su representación en términos ontológicos.
- La producción de servicios combinados a partir de productos o servicios procedentes de los catálogos de diferentes proveedores de contenido.

- La creación de una correspondencia entre consultas ontológicas, que proporcionan una visión abstracta de la información existente, y sus representaciones ontológicas de los catálogos. Los catálogos recogen los servicios ofrecidos, que deben tener implementada una consulta ontológica asociada.

Concretamente MKBEEM introduce las ventajas de una concepción multilingüe en todos los estados del ciclo de la información. Este es un ejemplo típico del uso de ontologías en el ámbito del comercio electrónico basado en lenguaje natural (Leger, y otros, 2001).

- **Presentation modeling Environment for Generic Adaptive hypermedia Support Systems (PEGASUS)**

Es un sistema de presentación dinámica en entornos web para representaciones personalizadas del conocimiento. Se trata de un sistema genérico de presentación para sistemas hipermedia de enseñanza adaptativa altamente independiente de la representación del conocimiento del dominio y del mantenimiento del estado de la aplicación (PEGASUS).

En PEGASUS, las ontologías se utilizan para proporcionar la máxima flexibilidad en la representación del conocimiento pedagógico. Son, además, un elemento esencial para conseguir la separación entre presentación y contenidos. La ontología del dominio en PEGASUS consiste en un conjunto de las clases que mejor se adecuan a un campo de aplicación determinado, o que reflejen la visión particular de un autor sobre el dominio. Las ontologías incluyen elementos para representar información sobre la materia, información pedagógica, e información sobre el estado del usuario y del entorno en tiempo ejecución. Todo este conocimiento se recoge mediante la definición de atributos para las clases, y relaciones entre clases (PEGASUS).

En estos dos sistemas analizados no se consiguió acceder a la documentación de las herramientas que fueron usadas para su desarrollo.

1.5.2 Sistemas nacionales

Luego de un análisis a estos sistemas, se realiza un estudio a mayor profundidad de los sistemas existentes dentro de nuestro país.

Aplicación informática para gestionar ontologías representativas del conocimiento en la web.

Esta aplicación fue desarrollada por los ingenieros Lieny Díaz Cardoso y Raul García Pérez (Diaz, y otros, 2013).

La aplicación permite gestionar las ontologías representativas del conocimiento en la web. Cuenta con restricciones de acceso, de acuerdo al rol que posee el usuario que interactúa con él. Existen dos roles, el rol administrador y el rol registrado. El usuario que cuente con el rol administrador, es el encargado el correcto funcionamiento del sistema y tiene acceso a la gestión de usuarios y a la asignación de roles. El sistema admite que exista más de un usuario con este rol. El usuario que posea el rol registrado tiene privilegios sobre el sistema que le permite el uso pleno de las funcionalidades de la aplicación. La aplicación cuenta con un repositorio común, donde todos los usuarios pueden acceder a las ontologías. El repositorio está compuesto por ontologías de extensión *.owl (Diaz, y otros, 2013).

La aplicación brinda la opción de crear, cargar, modificar, almacenar y descargar ontologías.

Para el desarrollo de la misma se utilizaron las siguientes tecnologías.

- Java como lenguaje de programación.
- NetBeans como entorno de desarrollo.
- OWL como lenguaje de representación de ontología.
- OWL-API como biblioteca para manipular la ontología.
- Apache Tomcat como servidor de aplicaciones.
- Dojo Toolkit como *framework* para el desarrollo de la aplicación.
- Para el modelado y diseño de la aplicación se utilizó la herramienta CASE Visual Paradigm.
- PostgreSQL como gestor de base de datos.
- XP como metodología para guiar el desarrollo de la aplicación.

Sistema de información basado en ontología para la gestión del conocimiento y la toma de decisiones en el sistema de gestión de proyectos XEDRO GESPRO

Esta aplicación fue desarrollada por la ingeniera Yanet Riquelme Santiago (Riquelme Santiago, 2015).

Sistema web que cuenta con un servicio de entrada-salida, en el cual cada entrada es una consulta en lenguaje natural que es pre-procesada permitiendo obtener las palabras claves de la misma, estos datos se envían a la ontología y con un motor de inferencia se responde según el tipo de pregunta siendo esta la salida del sistema (Riquelme Santiago, 2015).

El sistema abarca el desarrollo de varios módulos, los cuales son:

- Interfaces de usuario
- Pre-procesador de Lenguaje
- Motor de Inferencia
- Explicación
- Ontología
- Editor de Ontología
- Gestión de Datos

Para el desarrollo de la misma se utilizaron las siguientes tecnologías.

- PostgreSQL como gestor de base de datos.
- Servidor HTTP Apache como servidor web.
- Para el modelado y diseño de la aplicación se utilizó la herramienta CASE Visual Paradigm.
- HTML 5 como lenguaje de etiquetado para la creación de la aplicación web.
- JavaScript como lenguaje *scripting* multiplataforma y orientado a objetos.
- PHP como lenguaje de programación de uso general de código del lado del servidor para el desarrollo web de contenido dinámico.
- Entre los *framework* utilizados se encuentra JQuery y Twitter Boostrap.
- XP como metodología para guiar el desarrollo de la aplicación.

Sistema para la evaluación de créditos en el Banco Nacional de Cuba

Este sistema fue desarrollado por el Máster en Informática aplicada Yoan Antonio López Rodríguez (López Rodriguez, 2017).

En este sistema se propone un método para la integración de ontologías con bases de datos relacionales. El método está basado en siete actividades que incluyen: desarrollar el modelo relacional, desarrollar el modelo ontológico, cargar el modelo ontológico, poblar el modelo ontológico, inferir conocimiento, brindar las salidas y validar la solución. Para evaluar la aplicabilidad del método propuesto, se utilizó en la implementación de un sistema para la evaluación de créditos basado en ontologías para el Banco Nacional de Cuba. Con la aplicación del método propuesto se obtuvo un sistema reutilizable, flexible ante los cambios y de fácil mantenimiento (López Rodriguez, 2017).

Para el desarrollo del sistema para la evaluación de créditos del Banco Nacional de Cuba, fueron utilizadas las siguientes herramientas:

- Protégé para la creación de ontologías.
- Java como lenguaje de programación.
- NetBeans como entorno de desarrollo integrado.
- Para manipular la ontología se utilizan los marcos de trabajo Jena y OWL-API.
- Pellets como razonador para realizar la inferencia del conocimiento ontológico.
- Lenguaje SPARQL para consultar las relaciones dentro del modelo ontológico.

Mediante la investigación desarrollada a estos sistemas se logró un mejor entendimiento de la aplicación que se iba a implementar así como de las herramientas a utilizar para el desarrollo de la misma. Los sistemas mencionados responden a los modelos ontológicos que le fueron asignados, aunque el sistema desarrollado por Díaz y García (Diaz, y otros, 2013) es genérico, este no posee la capacidad de recuperación de información que se necesita, está enfocado en la edición de ontologías.

Se analizaron algunas herramientas que comúnmente se utilizaron en el desarrollo de estos sistemas estudiados como el lenguaje de programación Java, los marcos de trabajo Jena y OWL-API, la herramienta CASE Visual Paradigm y el lenguaje OWL para las ontologías que fueron importadas.

Para el desarrollo de este nuevo sistema es analizado detalladamente el método propuesto en el sistema para la evaluación de créditos en el Banco Nacional de Cuba (López Rodriguez, 2017), donde se puede arribar a la conclusión que realizando algunas

modificaciones a dicho método este servirá de base para la creación del sistema para la evaluación de productos de *software*.

1.6 Herramientas y tecnologías para la creación de un SIBO

En este epígrafe se describe cuáles son las herramientas que pueden ser utilizadas para la creación de un SIBO. Se argumenta en las que se utilizaron para el desarrollo de la investigación propuesta.

1.6.1 Lenguajes de representación de ontologías

Existen diferentes formas para la representación del conocimiento ontológico, pero su principal problema es encontrar un sistema de razonamiento que lo soporte, que pueda hacer las inferencias que necesite la aplicación dentro de los límites de recursos del problema a tratar. Se ha incorporado el desarrollo de lenguajes y estándares de representación del conocimiento ontológico (C. Blanco, 2008).

A continuación, se muestran los principales lenguajes de representación de ontologías.

SHOE

SHOE (*Simple HTML2 Ontology Extensions*). Fue el primer lenguaje de etiquetado para diseñar ontologías en la web. Este lenguaje nació antes de que se ideara la Web Semántica. Las ontologías y las etiquetas se incrustaban en archivos HTML. Este lenguaje permite definir clases y reglas de inferencia, pero no negaciones o disyunciones (C. Blanco, 2008).

RDF

Marco de Descripción de Recursos del inglés *Resource Description Framework*, es un *framework* [4]para metadatos en la *World Wide Web* (WWW), desarrollado por el *World Wide Web Consortium* (W3C). Es un lenguaje de objetivo general para representar la información en la web. Es una descripción conceptual, permite describir recursos mediante propiedades y valores de propiedades (Diaz, y otros, 2013).

[4] es un conjunto estandarizado de conceptos, prácticas y criterios para enfocar un tipo de problemática particular que sirve como referencia, para enfrentar y resolver nuevos problemas de índole similar.

OWL

OWL (*Ontology Web Language*) o Lenguaje de Ontologías para la *web* es un lenguaje de etiquetado semántico para publicar y compartir ontologías en la web. Se trata de una recomendación del W3C, y puede usarse para representar ontologías de forma explícita, es decir, permite definir el significado de términos en vocabularios y las relaciones entre aquellos términos. En realidad, OWL es una extensión del lenguaje RDF y emplea las tripletas de RDF, aunque es un lenguaje con más poder expresivo que éste. Se trata de un lenguaje diseñado para usarse cuando la información contenida en los documentos necesita ser procesada por programas o aplicaciones, en oposición a situaciones donde el contenido solamente necesita ser presentado a los seres humanos. OWL surge como una revisión al lenguaje DAML-OIL y es mucho más potente que éste. Al igual que OIL, OWL se estructura en capas que difieren en la complejidad y puede ser adaptado a las necesidades de cada usuario, al nivel de expresividad que se precise y a los distintos tipos de aplicaciones existentes (Manchester, T. U. O., 2016).

Consta de tres sublenguajes con un nivel de expresividad creciente: *OWL Lite, OWL DL* y *OWL Full* (Bañon Peñuelas, 2013).

- *OWL Lite:* Está diseñado para los usuarios que necesiten una clasificación jerárquica y restricciones simples. Por ejemplo, admite restricciones de cardinalidad, pero solo permite establecer valores cardinales de 0 o 1. Está diseñado de esta manera para que sea más sencillo proporcionar herramientas de soporte a *OWL Lite* que a sus *OWL DL* y *OWL Full* que contarán con mayor nivel de expresividad (Bañon Peñuelas, 2013).
- *OWL DL:* Está diseñado para aquellos usuarios que quieren la máxima expresividad conservando completitud computacional (se garantiza que todas las conclusiones sean computables), y resolubilidad (limitación de los cálculos en tiempo). *OWL DL* incluye todas las construcciones del lenguaje de OWL, pero sólo pueden ser usados bajo ciertas restricciones (por ejemplo, mientras una clase puede ser una subclase de otras muchas clases, una clase no puede ser una instancia de otra). *OWL DL* es denominado de esta forma debido a su correspondencia con la lógica de descripción, un campo de investigación que estudia la lógica que compone la base formal de OWL (Bañon Peñuelas, 2013).

- *OWL Full:* Está dirigido a usuarios que quieren máxima expresividad y libertad sintáctica de RDF sin garantías computacionales. Por ejemplo, en *OWL Full* una clase puede ser considerada simultáneamente como una colección de clases individuales y como una clase individual propiamente dicha. *OWL Full* permite una ontología para aumentar el significado del vocabulario preestablecido (RDF u OWL). Es poco probable que cualquier *software* de razonamiento sea capaz de obtener un razonamiento completo para cada característica de *OWL Full* aunque es cierto que motores como Hermit han mostrado un buen comportamiento (Bañon Peñuelas, 2013).

Justificación de los lenguajes seleccionados

Las ontologías que utiliza el sistema han sido definidas en los lenguajes OWL y RDF. Se utilizó el lenguaje OWL a la hora de realizar un razonamiento ontológico, ya que provee un vocabulario más rico y estándar, con una semántica formal para expresar conceptos y relaciones entre ellos, de los sublenguajes de OWL, se utilizó *OWL DL* por poseer toda la semántica posible y además ser computable. Se realizarán consultas SPARQL, estas consultas son realizadas mediante la biblioteca Jena, la cual trabaja con el lenguaje RDF.

1.6.2 Herramientas para la manipulación de ontologías

Para proveer el acceso, la manipulación y la consulta de las ontologías en sistemas informáticos se han desarrollado marcos de trabajo que permiten transformar los archivos de las ontologías a la Programación Orientada a Objetos (POO). En comparaciones realizadas a los marcos de trabajo Jena y OWL API, Jena ofrece un mejor rendimiento que OWL API tanto para la inserción como para la consulta, sin embargo para el proceso de inferencia OWL API es más potente (López Rodriguez, 2017).

Jena

Fue originalmente desarrollado por investigadores de los laboratorios de *Hewlett Packard* (HP), en Bristol, Reino Unido, en el año 2000. Jena ha sido siempre un proyecto de código abierto, ampliamente utilizado en una gran variedad de aplicaciones de web semántica y demostraciones. En 2009, HP decidió una reorientación de las actividades de desarrollo fuera del mantenimiento directo de Jena, aunque sigue siendo de apoyo de los objetos del

proyecto. Desde noviembre del 2010, Jena es propiedad de *Apache Software Foundation* (Jena, 2011).

El marco de trabajo Jena incluye (Jena, 2011):

- Una API para leer, procesar y escribir datos en XML.
- Una API de ontologías para manejar ontologías en OWL y RDFS.
- Un motor de inferencia basado en reglas para el razonamiento con fuentes de datos RDF y OWL.
- Permite que se puedan almacenar en disco un gran número de triples RDF de forma eficiente.
- Un motor de consultas compatible con la especificación de SPARQL.
- Servidores para permitir la publicación de datos RDF para que pueda ser usado por otras aplicaciones diferentes.

OWL-API

OWL-API es un marco de trabajo diseñado en la Universidad de Manchester para satisfacer las necesidades de los que desarrollan aplicaciones basadas en OWL, así como editores y razonadores de OWL. Constituye una API de Java de alto nivel (López Rodriguez, 2017).

Es una referencia para crear, manipular y serializar ontologías en OWL. Es código abierto y disponible bajo licencias de Apache y LGPL (International Hellenic University, 2013).

Incluye, entre otros, los siguientes componentes:

- Analizador y escritor para RDF/XML
- Analizador y escritor para OWL/XML
- Analizador y escritor para OWL Functional Syntax
- Interfaces de razonamiento para diferentes razonadores

Para el desarrollo de la aplicación se decide utilizar el marco de trabajo OWL-API para inferir el conocimiento en la ontología, ya que este permite trabajar directamente con los axiomas por ser más potente, además de traer consigo los razonadores internos. También se utiliza Jena para realizar las consultas a la ontología, ya que accede a estas consultas mediante SPARQL.

1.6.3 Lenguaje para el acceso a datos de un modelo otológico

El lenguaje SPARQL se utiliza en los modelos ontológicos de un modo similar al lenguaje SQL en las bases de datos relacionales.

SPARQL es un acrónimo recursivo del inglés (SPARQL *Protocol and RDF Query Language*). Es un lenguaje de recuperación para RDF. Surgió como una recomendación de la W3C para un lenguaje de consulta dentro de la Web Semántica. SPARQL está implementado en muchos lenguajes de programación tales como Java donde el API de SPARQL se encuentra integrado con el marco de trabajo Jena (López Rodriguez, 2017).

Permite centrarse en la información que se desea tener recopilada sin tener en cuenta la tecnología de la base de datos o el formato utilizado para almacenar esos datos. Ha sido diseñado para un uso a escala de la web, por ello permite hacer consultas sobre orígenes de datos distribuidos, independientemente del formato. En la recuperación de información la creación de una sola consulta a través de diferentes almacenes es mejor que múltiples consultas (Manchester, T. U. O., 2016).

La sintaxis de la consulta SPARQL es similar a la conocida SQL, añadiendo algunas modificaciones para facilitar el análisis sintáctico del lenguaje:

PREFIX prefix

SELECT? subject?object

WHERE {conditions}

En el primer bloque (PREFIX) es donde se definen los prefijos que serán empleados en las consultas. Para trabajar con la ontología creada hay que copiar la dirección URI generada dentro de un nuevo prefijo que se define. En el segundo bloque (SELECT) se incluyen los valores que se desean retornar. Mientras que en el tercero (WHERE) se incluyen las condiciones.

1.6.4 Razonadores

Un razonador es una pieza de *software* capaz de deducir las consecuencias lógicas de un conjunto de hechos afirmados o axiomas. La noción de un razonador semántico generaliza

la de un motor de inferencia, al ofrecer un conjunto más amplio de mecanismos para trabajar con ellos. Las reglas de inferencia comúnmente se especifican mediante un lenguaje de ontologías y con frecuencia un lenguaje de descripción. Algunos de los razonadores que son capaces de trabajar sobre OWL y que se pueden utilizar tanto en Protégé como en OWL API son FaCT++, Hermit y Pellet (International Hellenic University, 2013). A continuación se explican.

FaCT++

Es un clasificador de lógica descriptiva. El Sistema de Información incluye dos razonadores, los cuales utilizan algoritmos de robustos y completos. FaCT++ es la nueva generación de FaCT que usa los algoritmos FaCT pero con diferente arquitectura interna y razona sobre OWL DL. Está implementado en C++ para crear herramientas *software* más eficiente y facilitar la portabilidad. El razonador implementa un procedimiento de decisión de tableaux de SHOIQ, con apoyo adicional para los tipos de datos, incluyendo cadenas y enteros. FaCT++ se distribuye bajo una licencia pública de GNU y está disponible para su descarga, tanto como un archivo binario y como el código fuente (Espín Martín, 2016).

Hermit

Es un razonador de ontologías en el lenguaje OWL. Dado un fichero OWL, Hermit puede determinar al menos si la ontología es consistente e identificar las relaciones entre las clases de subsunción. Las ontologías, que antes requerían horas o minutos para clasificarse, pueden a menudo clasificarse en segundos gracias a Hermit. Además, es el primer razonador capaz de clasificar ontologías que se han tomado como demasiado complejas de manejar anteriormente por cualquier sistema disponible. Hermit utiliza la semántica directa y pasa todas las pruebas de conformidad OWL2 [5]para razonadores de semántica directa. Es de código abierto (Espín Martín, 2016).

Pellet

[5] es un lenguaje para la descripción del conocimiento, con un enfoque específico en definir ontologías para la Web

Es un razonador de código abierto para OWL en Java e incluye soporte para OWL2. Proporciona servicios estándar de última generación de razonamiento para ontologías OWL. Pellet proporciona funcionalidades para ver la validación de especies, chequear consistencia de ontologías, clasificar la taxonomía, comprobar implicaciones lógicas y contestar consultas. Las consultas pueden ser formuladas mediante SPARQL (Espín Martín, 2016).

Justificación del razonador seleccionado

Se utilizará para el desarrollo de la aplicación el razonador Pellet, ya que este está implementado en Java, permite trabajar directo con los axiomas de las ontologías en formato OWL y su librería es de fácil acceso.

1.6.5 Lenguajes de programación para el desarrollo de la aplicación

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Una característica relevante de los lenguajes de programación, es precisamente que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos, para así realizar la construcción de un programa de forma colaborativa (Diaz, y otros, 2013).

Lenguaje de programación C++

El lenguaje de programación C++ fue creado en los años 80 por Bjarne Stroustrup basando en el lenguaje C. Es un lenguaje orientado a objetos al que se le añadieron características y cualidades de las que carecía el lenguaje C (Lenguaje de programación C++).

El lenguaje de programación C++ depende mucho del hardware, tiene una gran potencia en la programación a bajo nivel, y se le agregaron herramientas para permitir programar a alto nivel. Es uno de los lenguajes más potentes porque nos deja programar a alto y a bajo nivel, pero a su vez es difícil de aprender porque es necesario hacerlo casi todo manualmente (Lenguaje de programación C++).

El nombre fue propuesto por Rick Masciatti, al utilizarse C++ fuera de los laboratorios donde se creó. Con el nombre de C++ se quiso dar a entender que es una extensión del lenguaje C (Lenguaje de programación C++).

Es un lenguaje de programación híbrido, al que se le puede compilar.

Lenguaje de programación Java

Es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria. Con respecto a la memoria, su gestión no es un problema, ya que esta es gestionada por el propio lenguaje y no por el programador (Java).

Algunas características que describen este lenguaje:

- Orientado a Objeto: Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.
- Distribuido: Java se ha construido con extensas capacidades de interconexión TCP/IP. La característica de ser distribuido es debido a que proporciona las librerías y herramientas para que los programas puedan ejecutarse en varias máquinas.
- Robusto: Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores lo antes posible en el ciclo de desarrollo.
- Interpretado: Se traduce el código fuente a un código intermedio denominado bytecode, que es interpretado por la máquina virtual de Java, lo cual permite que se pueda ejecutar en cualquier sistema operativo.
- Seguro: No se permite el acceso ilegal a memoria ya que no se trabaja con punteros. El código Java pasa muchas pruebas antes de ejecutarse en una máquina. El código se pasa a través de un verificador de bytecode que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de

código ilegal, que falsean punteros, violan derechos de acceso sobre objetos o intentan cambiar el tipo o clase de un objeto.

- Dinámico: No conecta todos los módulos que comprende una aplicación hasta el tiempo de ejecución, ya que las librerías nuevas o actualizadas no paralizarán las aplicaciones actuales siempre que mantengan el API anterior.

Justificación del lenguaje seleccionado

Para la implementación de la aplicación se utilizó Java como lenguaje de programación por las ventajas que brinda de ser un lenguaje orientado a objetos, robusto, seguro, de arquitectura neutra, portable, multitarea y dinámico. Además, es un lenguaje independiente de la plataforma, dado que un programa escrito en Java puede ser corrido sobre cualquier sistema operativo. Los marcos de trabajo utilizados, Jena y OWL-API son basados en este lenguaje.

1.6.6 Entorno Integrado de Desarrollo

Un entorno de desarrollo integrado o (IDE por sus siglas en inglés), es un programa informático compuesto por un conjunto de herramientas de programación. Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (Diaz, y otros, 2013).

NetBeans

Es una herramienta para programadores, pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java y soporta el desarrollo de todos los tipos de aplicación Java. NetBeans proporciona herramientas para la programación en el lenguaje Java e incluye resaltado de sintaxis y auto-completado de código. Provee soporte para las bibliotecas de Jena y OWL-API, además cuenta con un módulo de subversión para facilitar la manipulación de las versiones del código durante la implementación (NetBeans, 2012).

Eclipse

Es una plataforma universal para integrar herramientas de desarrollo, con una arquitectura abierta y basada en *plugins*. Da soporte a todo tipo de proyectos que abarcan desde el ciclo de vida del desarrollo de aplicaciones, incluyendo soporte para modelado (Java).

Justificación del entorno de desarrollo utilizado

Para el trabajo con el modelo persistente se utilizó el IDE NetBeans, por ser multiplataforma, multilenguaje y de código abierto. Dicho IDE permite realizar consultas y validar que la propuesta ontológica a desarrollar funcione correctamente, de manera fiable y segura. Además, permite la integración con Jena para extraer de la base de datos la información de la ontología.

1.6.7 Herramientas CASE

Las herramientas CASE (*Computer Aided Software Engineering*) son diversas aplicaciones informáticas o programas informáticos destinadas a aumentar la productividad en el desarrollo de *software* reduciendo el costo de las mismas en términos de tiempo y de dinero. Algunos objetivos a los que están dirigidas son (Vyeira, 2016):

- Permitir la aplicación práctica de metodologías estructuradas, que al ser realizadas con una herramienta consiguen agilizar el trabajo.
- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento de los programas.
- Mejorar y estandarizar la documentación.
- Aumentar la portabilidad de las aplicaciones.
- Facilitar la reutilización de componentes *software*.
- Permitir un desarrollo y un refinamiento visual de las aplicaciones, mediante la utilización de gráficos.

Visual Paradigm

Visual Paradigm es una herramienta que sirve para realizar modelado de *software* siguiendo el estándar UML. Posee características gráficas muy cómodas, que facilitan la realización de diferentes diagramas de modelado. Además, puede ser utilizada para la modelación de procesos de desarrollo de *software* que sigan la filosofía de *software* libre. Es una herramienta que permite (Visual Paradigm, 2014):

- Dibujar todos los tipos de diagramas de clases.
- Generar código directo, inverso y desde diagramas.
- Generar informes para generación de documentación (expediente de proyecto).

- Realizar ingeniería directa e inversa. Esta última se refiere al proceso ingenieril en el que se obtienen modelos conceptuales a partir de los artefactos *software* como código fuente, ejecutables, binarios y ficheros intermedios.
- Importación y exportación de ficheros XML.
- Generación de objetos Java desde la base de datos, entre otras posibilidades.

Justificación de la herramienta CASE seleccionada

Para el análisis, el diseño y el desarrollo del sistema se decidió utilizar la herramienta CASE Visual Paradigm en su versión 8.0, permitiendo modelar todos los diagramas necesarios para representar gráficamente el ciclo de vida del desarrollo de *software*. Esta herramienta propicia un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación.

1.6.8 Metodologías para el desarrollo de software

El proceso de desarrollar *software* no es una tarea fácil, se debe contar con un proceso bien detallado y para esto se necesita aplicar una metodología que sea capaz de llevar a cabo el control total del producto (Maida, y otros, 2015).

Las metodologías de desarrollo de *software* surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto de *software*. Dichas metodologías pretenden guiar a los desarrolladores al crear un nuevo *software*, pero los requisitos de un *software* a otro son tan variados y cambiantes, que ha dado lugar a que exista una gran variedad de metodologías para la creación del *software* (Maida, y otros, 2015).

RUP

Es una metodología tradicional que provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de *software* de alta calidad que satisfaga los requerimientos de los usuarios finales (respetando cronograma y presupuesto). Puede ser adaptado y extendido para satisfacer las necesidades de la organización que lo adopte (Cabrera, 2014). Esta metodología presenta como características que es guiada por casos de uso, está centrada

en arquitectura y su desarrollo está basado en componentes. Define un conjunto de roles, actividades y artefactos. Utiliza procesos integrados, además de modelar a través del Lenguaje Unificado de Modelado (UML) (Maida, y otros, 2015).

RUP presenta una serie de ventajas para el desarrollo de proyectos ya que en cada fase se hacen evaluaciones que permite cambios de objetivos, funciona bien en proyectos de innovación, ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el *software*.

Esta metodología ha demostrado ser efectiva y necesaria en proyectos de gran tamaño, respecto a tiempo y recursos, sin embargo, este enfoque no resulta ser el más adecuado para entornos donde el sistema es muy cambiante, y en donde se exige reducir los tiempos de desarrollo sin afectar la alta calidad del sistema.

XP

La metodología XP (Extreming Programing) o Programación Extrema es la más destacada de los procesos ágiles de desarrollo de *software*. Se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad. Está centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de *software*, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo (Maida, y otros, 2015). Se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Es adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico. Permite realizar con frecuencia pruebas unitarias y de regresión para mantener el *software* sin errores y actualizado cada vez que se realice un cambio en los requerimientos (Riquelme Santiago, 2015). XP consta de cuatro fases fundamentales: Planeación, Diseño, Desarrollo y Pruebas.

- Planeación: La planeación es la etapa inicial de todo proyecto en XP. En este punto se comienza a interactuar con el cliente y el resto del grupo de desarrollo para identificar los requerimientos del sistema. En este punto se identifica el número y tamaño de las iteraciones al igual que se plantean ajustes necesarios a la metodología según las características del proyecto.

- Diseño: En XP solo se diseñan aquellas HU que el cliente ha seleccionado para la iteración actual por dos motivos. Por un lado, se considera que no es posible tener un diseño completo del sistema y sin errores desde el principio. El segundo motivo es que, dada la naturaleza cambiante del proyecto, el hacer un diseño muy extenso en las fases iniciales el proyecto para luego modificarlo, se considera un desperdicio de tiempo.
- Desarrollo: El desarrollo es un proceso que realiza en forma paralela con el diseño y la cual está sujeta a varias observaciones por parte de XP consideradas controversiales por algunos expertos tales como la rotación de los programadores o la programación en parejas.
- Pruebas: Cuando se tienen bien implementadas las pruebas no habrá temor de modificar el código del otro programador en el sentido que, si se daña alguna sección, las pruebas mostrarán el error y permitirán encontrarlo. Uno de los elementos que podría obstaculizar que un programador cambie una sección de código funcional es precisamente hacer que este deje de funcionar. Si se tiene un grupo de pruebas que garantice su buen funcionamiento, este temor se mitiga en gran medida.

Los principales artefactos en la metodología XP son los siguientes:

- Historias de Usuario
- Tareas de Ingeniería
- Pruebas de Aceptación
- Prueba Unitarias y de Integración
- Plan de Entrega
- Código

Justificación de la metodología seleccionada

Para guiar el desarrollo de esta aplicación se siguió la metodología ágil XP por las características que posee de centrarse básicamente en la implementación de las soluciones y no en el soporte documental, además de dar respuesta y posible solución a cambios repentinos en los requisitos funcionales del sistema. Logra que cada miembro del equipo incluyendo el cliente de desarrollo esté listo para enfrentar cualquier cambio en el *software*, promoviendo el trabajo en equipo y preocupándose por el aprendizaje de los desarrolladores. Esta metodología se basa en realimentación continua entre el cliente y el

equipo de desarrollo y tiene como objetivo lograr la satisfacción del cliente (Diaz, y otros, 2013).

1.7 Conclusiones del capítulo

A partir del estudio realizado referente a los principales conceptos de calidad y evaluación de software, las ontologías y SIBO se arribó a las consideraciones siguientes:

- La organización de la información basada en ontología proporciona un vocabulario común para resolver el problema de integridad e inconsistencia que se identificó en el proceso de evaluación de software, facilitando la gestión del conocimiento.
- El estudio de los temas relacionados con las ontologías y SIBO, proporciona un mejor entendimiento de los principales conceptos para llegar a una propuesta de solución.
- El uso de herramientas y tecnologías tales como el lenguaje de programación Java, el entorno de desarrollo NetBeans, el lenguaje OWL y el uso de la librería OWL-API facilitará el desarrollo del SIBO.

CAPÍTULO II: DISEÑO Y ANÁLISIS DEL SIBO PARA LA EVALUACIÓN DE PRODUCTOS DE SOFTWARE.

2.1 Introducción:

En el presente capítulo se expone la estructura de la solución para lograr comprender mejor el sistema. Se realiza una descripción de la propuesta de solución y demás elementos que comprenden la fase de análisis y diseño, como son los patrones y la arquitectura a realizar. Se crean las HU, las cuales describen las funcionalidades definidas. Además, se crean los artefactos correspondientes a la etapa de diseño que propone la metodología XP.

2.2 Propuesta de solución

Haciendo uso de las distintas herramientas, tecnologías y marcos de trabajo actuales, se propone la implementación de un sistema de información que sea capaz de manipular las ontologías involucradas en los subprocesos de la dirección de calidad de la UCI.
La solución propuesta gestiona el conocimiento generado en el proceso de evaluación de productos de *software* (ver anexo 1) en la dirección de calidad de la UCI, incluyendo el subproceso de pruebas de *software* (ver anexo 2). La información involucrada en este proceso se almacena en dos ontologías, una que abarca los detalles del proceso de evaluación y otra que almacena las características de las pruebas de *software*. La aplicación cumple las siguientes funcionalidades para la identificación y recuperación del conocimiento:

- Importar las ontologías involucradas en el proceso.
- Mostrar los componentes de una ontología seleccionada, dígase individuos, clases y sus relaciones.
- Realizar consultas SPARQL, de manera visual, sencilla y sin necesidad de tener dominio de este lenguaje con el objetivo de recuperar el conocimiento.
- Clasificar los individuos a partir de un razonamiento ontológico, permitiendo obtener el conocimiento implícito.

En la figura 1 se muestra el funcionamiento del SIBO, donde se expone como el usuario interactúa con el sistema para la evaluación de productos. Utiliza las bibliotecas Jena y

OWL-API para acceder a la otología, permitiendo al sistema importarlas y recuperar la información.

A través de la interfaz gráfica del sistema, el usuario define las consultas SPARQL. La aplicación utiliza el API Jena para realizar las consultas y mostrar los resultados.
Se muestra al usuario las principales clasificaciones de las instancias descritas en las ontologías desarrolladas, mediante la biblioteca OWL-API se realiza un razonamiento ontológico que es capaz de mostrar el conocimiento implícito.

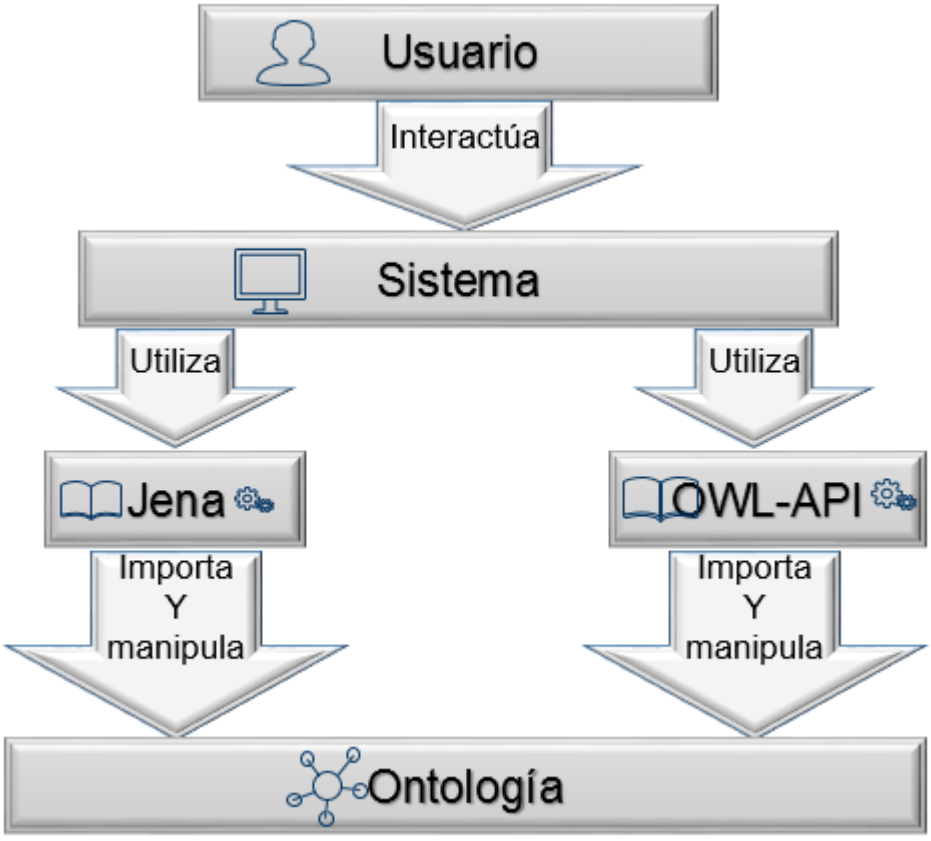


Figura 1. Propuesta de solución. Fuente. Elaboración propia

Mediante este sistema se garantiza una mejor manipulación de la información, ayudando así a la gestión del conocimiento y facilitando la toma de decisiones.
La interfaz gráfica es sencilla permitiendo una fácil interacción usuario/sistema, por poseer la característica de ser intuitiva.

Para facilitar la recuperación de información que las ontologías brindan es necesario el uso de un método para la integración de las mismas. A continuación, se detalla dicho método adaptado al SIBO para la evaluación de productos de *software* en la dirección de calidad de la UCI.

2.3 Método para la integración de ontologías en el sistema para la evaluación de productos de software.

En la presente investigación se optó por la adaptación de un método definido en la tesis de maestría titulada “Método para la integración de ontologías en sistemas relacionales para la evaluación de créditos” (López Rodriguez, 2017). Este método establece un conjunto de actividades ordenadas, que son realizadas por el equipo de desarrollo para apoyar el diseño del sistema. En él se definen resultados/entregas de actividades.

El método adaptado consta de cuatro subprocesos como se muestra en la figura 2, los cuales son: importar el modelo ontológico, inferir conocimiento, brindar salidas del sistema y por último validar la solución.

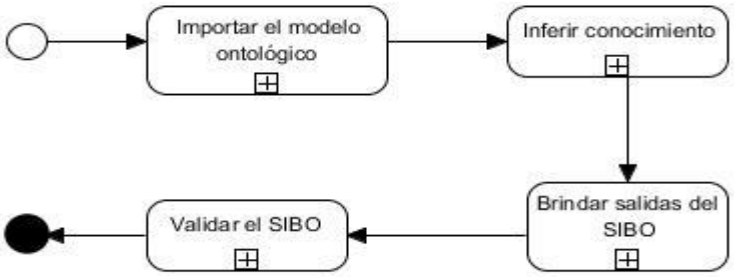


Figura 2. Método para la integración de ontología en sistema para la evaluación del producto de software.
Fuente. Elaboración propia

2.3.1 Importar el modelo ontológico

En este subproceso se tienen en cuenta aspectos tales como: el entorno de desarrollo integrado que se utiliza para la construcción del sistema y el marco de trabajo que permite la manipulación de los ficheros de las ontologías en la POO.

Las ontologías desarrolladas en la herramienta de edición, son exportadas hacia un directorio seleccionado para utilizarlas en el entorno de desarrollo del sistema. Como se fundamenta en el capítulo anterior, se utilizó el entorno de desarrollo NetBeans para

desarrollar el sistema y para el acceso y manipulación de las ontologías se trabaja con los marcos de trabajo Jena y OWL-API.

Mediante los marcos de trabajo Jena y OWL-API se transforman los ficheros de las ontologías a la POO. Para importar el modelo ontológico se crea una ruta con la ubicación del fichero de la ontología y luego se importará el fichero a las clases de la POO según el marco de trabajo utilizado.

En las figuras 3 y 4 se muestra un ejemplo de la importación del modelo ontológico a través de las bibliotecas de OWL-API y Jena (International Hellenic University, 2013).

```
//cargar Ontologia del proceso de Evaluacion
OWLOntologyManager manager = OWLManager.createOWLOntologyManager();
File file;
file = new File("C:/Users/leidyss/Documents/OntologiaEvaluacion.owl");
OWLOntology ontologyEva = manager.loadOntologyFromOntologyDocument(file);
```

Figura 3. Importar el modelo ontológico con OWL-API. Fuente. Elaboración propia

```
// Cargar la Ontolgia del proceso de Evaluacion
 OntModel model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
 model.read("file:///Users/Documents/NetBeansProjects/Proceso_Evaluacion.owl","RDF/XML");
```

Figura 4. Importar el modelo ontológico con Jena. Fuente. Elaboración propia

En la figura 5 se muestra el subproceso importar el modelo ontológico. En este se importan los ficheros de la ontología en el sistema los cuales son manipulados mediante los marcos de trabajo Jena y OWL-API. La entrada al subproceso es el fichero de la ontología. La salida del subproceso es el sistema con los ficheros de las ontologías importadas. Las herramientas y tecnologías utilizadas son el NetBeans y los marcos de trabajo Jena y OWL-API.

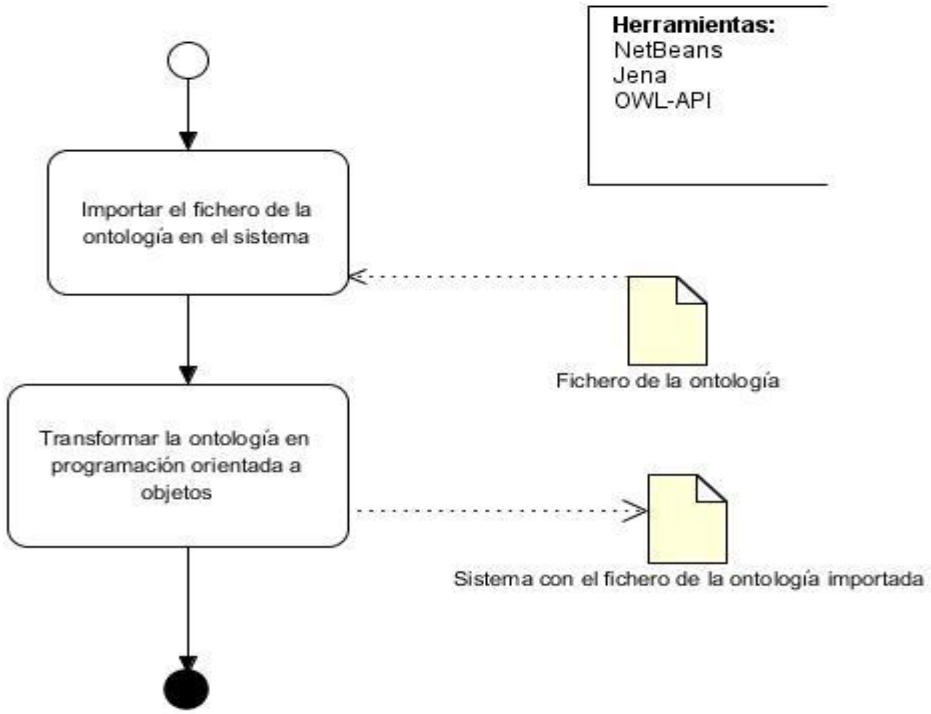


Figura 5. Importar modelo ontológico. Fuente. Elaboración propia

2.3.2 Inferir conocimiento

Con el subproceso Inferir conocimiento se persigue transformar el conocimiento implícito en conocimiento explícito para lograr obtener un nuevo modelo ontológico con las clasificaciones principales de la ontología.

El proceso de inferencia se define como el proceso abstracto de derivar información adicional y es llevado a cabo por un motor de inferencia o razonador que se corresponde con el objeto de código que realiza esa tarea.

En cada proceso de inferencia se tienen en cuenta aspectos tales como: el razonador que realiza la inferencia, el marco de trabajo para la manipulación de ontologías que se utiliza y donde se lleva a cabo el proceso de inferencia.

En este subproceso (ver figura 6) se configura el marco de trabajo OWL-API, se selecciona un razonador para la inferencia, en este caso Pellets (ver figura 7) y se obtiene un nuevo modelo ontológico con la inferencia realizada.

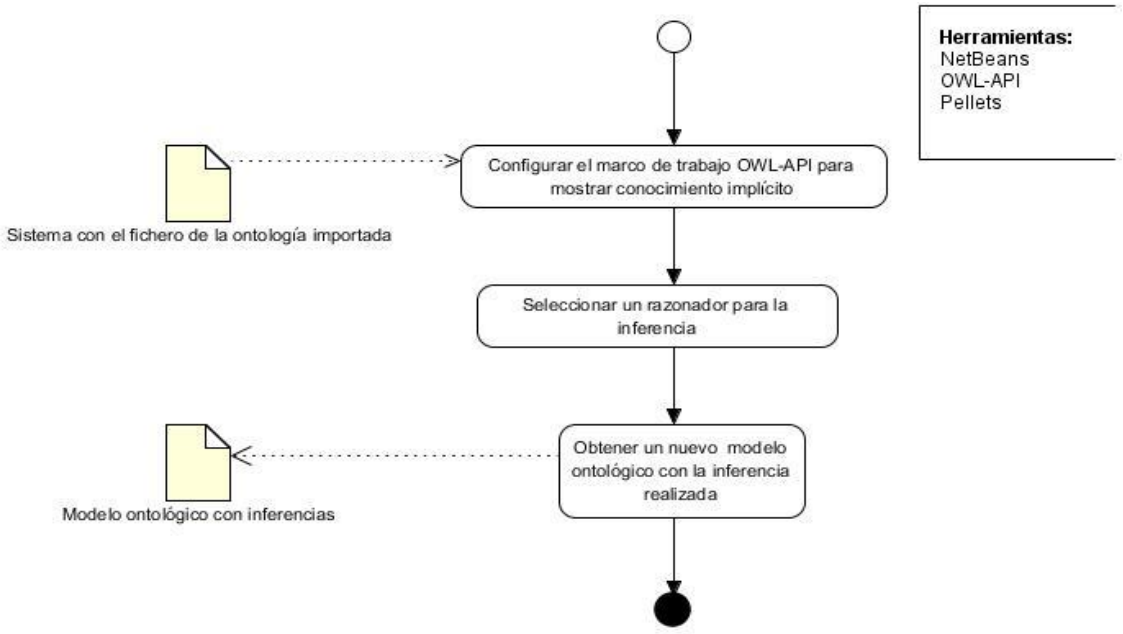


Figura 6.Inferir conocimiento Fuente. Elaboración propia

La entrada al subproceso es el sistema con el fichero de la ontología importada. La salida es el nuevo modelo ontológico con la inferencia realizada. Las herramientas y tecnologías utilizadas son: NetBeans, el marco de trabajo OWL-API y el razonador Pellets.

```
//Agregar el razonador Pellet
PelletReasoner reasoner = com.clarkparsia.pellet.owlapiv3.
PelletReasonerFactory.getInstance().createReasoner(ontologyEva);
```

Figura 7. Razonador Pellet. Fuente. Elaboración propia

2.3.3 Brindar salidas al sistema

Las salidas del sistema (interfaces de usuario) son imprescindibles en los sistemas informáticos que se desarrollan para interactuar con las personas. En los sistemas basados en ontologías se utilizan para la consulta de las mismas el lenguaje SARQL.

A continuación se muestra en la figura 8 un ejemplo de una consulta realizada en el lenguaje SPARQL.

```
PREFIX eva: <http://www.semanticweb.org/parker/ontologies/2017/10/untitled-ontology-16#>

SELECT ?subject ?object
        WHERE { ?subject eva:descripcion_de_artefacto ?object }
```

Figura 8. Consulta SPARQL. Fuente. Elaboración propia

En la figura 9 se muestra este subproceso. Se prepara la información a mostrar, dígase la perteneciente al modelo ontológico inferido como las relaciones de objetos y datos. A continuación se crean las vistas para la interacción entre el usuario y el sistema. La entrada al subproceso es el modelo ontológico resultante de la inferencia. Las salidas son el conjunto de interfaces de usuario creadas para mostrar la información requerida por el usuario. Las herramientas y tecnologías que se utilizarán son: NetBeans y el lenguaje de consultas SPARQL.

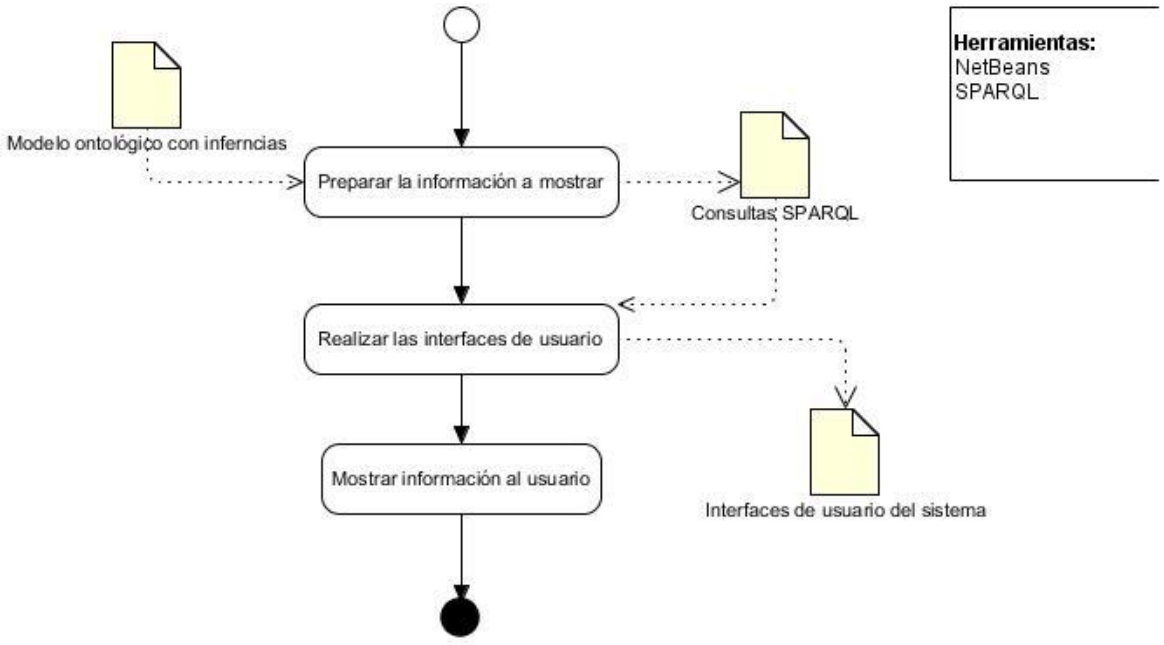


Figura 9. Brindar salidas del sistema. Fuente. Elaboración propia

2.3.4 Validar el Sistema

Para la validación del sistema se realizarán pruebas que permitan verificar que el sistema realiza correctamente las funcionalidades para las cuales fue creado y que cumple con las expectativas del cliente. Como la metodología propuesta es XP, las pruebas a realizar son unitarias y de aceptación, las mismas quedarán explicadas con mayor detalle en el próximo capítulo.

Este método adaptado facilita la implementación del SIBO para la evaluación del producto de *software* en la Dirección de Calidad de la UCI.
A continuación, se describe el artefacto principal correspondiente a la actividad de planeación definido por la metodología XP.

2.4 Historias de Usuarios

Las HU son utilizadas para especificar los requisitos de un *software*. Encierran en ellas las funcionalidades que se van a implementar en el sistema. Permiten administrar los requisitos sin tener que elaborar gran cantidad de documentos formales y sin requerir de mucho tiempo (Beck, 2009).

Las HU son tarjetas en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. Cada historia es lo suficientemente comprensible y delimitada para que los programadores puedan implementarla en unas semanas. Permiten responder rápidamente a los requisitos cambiantes (Beck, 2009).

En las siguientes tablas, se presentan las principales HU del sistema, respondiendo a las funcionalidades del SIBO.

Tabla 1. HU. Seleccionar la ontología. Fuente. Elaboración propia

Historia de usuario	
Número 1	**Nombre** Seleccionar la ontología
Actor Usuario	**Iteración asignada** 1
Prioridad de negocio Alta	**Puntos estimados** 1 semana
Nivel de complejidad Baja	**Puntos reales** 1 semana
Descripción Es la vista de inicio de la aplicación, se encuentran los logotipos que la identifican. Se muestra un menú con las ontologías involucradas en el proceso de evaluación de productos de *software* y los subprocesos involucrados en este. El usuario podrá seleccionar la ontología	

con que desea trabajar. Después de seleccionado el modelo ontológico, se mostrará otra ventana con toda información referente a este.
Observaciones La ontología debe estar escrita en idioma español.
Prototipo de interfaz 

Tabla 2. HU. Mostrar instancias. Fuente. Elaboración propia

Historia de usuario	
Número 2	**Nombre** Mostrar instancias
Actor Usuario	**Iteración asignada** 1
Prioridad de negocio Alta	**Puntos estimados** 1 semana
Nivel de complejidad Media	**Puntos reales** 1 semana
Descripción Como prerrequisito el usuario debe haber seleccionado el modelo ontológico. Luego se muestra al usuario todas las instancias existentes en dicho modelo. Estas instancias serán mostradas en una tabla, donde se describe en forma de leyenda su significado conceptual.	
Observaciones Son mostradas a partir de esta HU los prototipos de interfaz de un único modelo ontológico, para los demás sería muy parecido.	
Prototipo de interfaz	

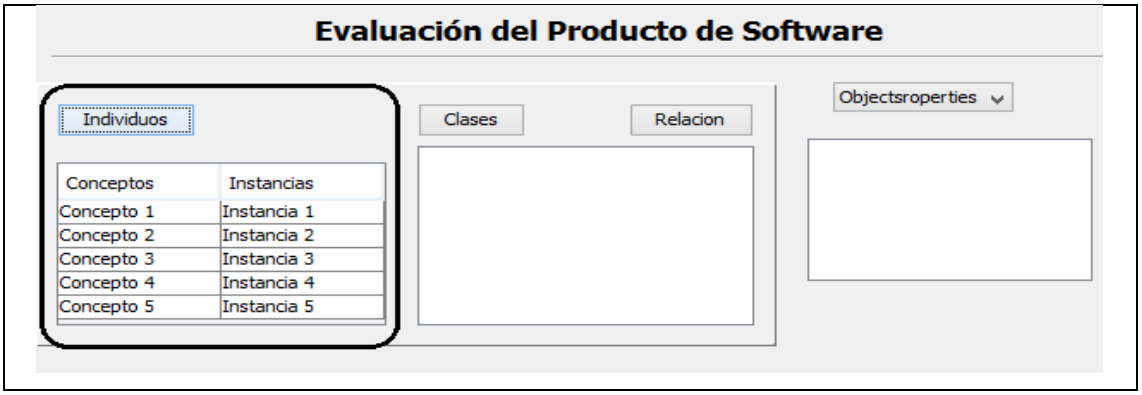


Tabla 3. HU. Mostrar clases. Fuente. Elaboración propia

Historia de usuario	
Número 3	**Nombre** Mostrar las clases
Actor Usuario	**Iteración asignada** 1
Prioridad de negocio Alta	**Puntos estimados** 1 semana
Nivel de complejidad Media	**Puntos reales** 1 semana
Descripción Se muestran todas las clases existentes en la ontología. El usuario selecciona un botón que hace referencia a las clases, estas serán mostradas en forma de árbol para lograr una mejor organización del conocimiento y obtener mejor información sobre el dominio en que se está trabajando.	
Observaciones	
Prototipo de interfaz	

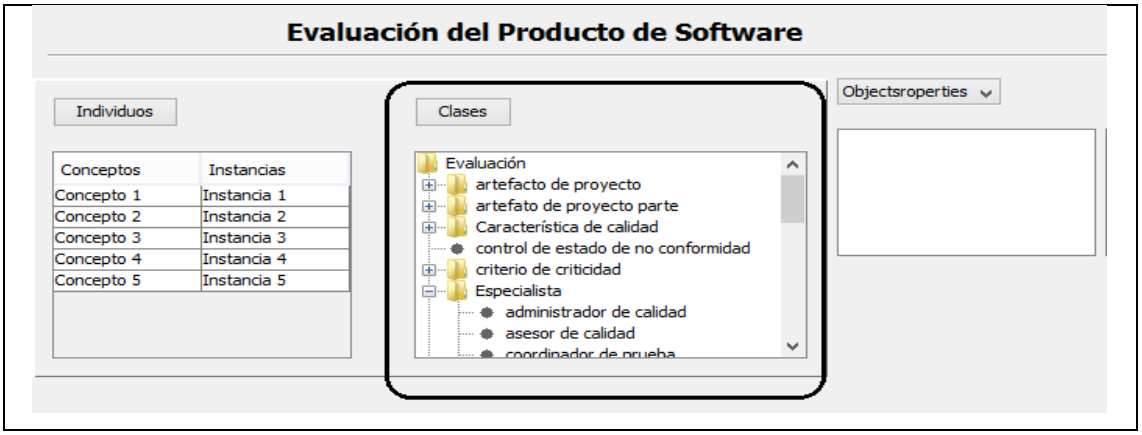


Tabla 4. HU. Relación clases e individuos. Fuente. Elaboración propia

Historia de usuario	
Número 4	**Nombre** Relación entre las instancias y las clases
Actor Usuario	**Iteración asignada** 1
Prioridad de negocio Alta	**Puntos estimados** 1 semana
Nivel de complejidad Alta	**Puntos reales** 1 semana
Descripción Al seleccionar el botón “Relación”, se muestran todos los individuos que contiene cada una de las clases del modelo ontológico. Esta información va a ser expuesta en un área de texto que se muestra debajo de las dos primeras observaciones. El principal objetivo de esta vista es relacionar al usuario con el modelo ontológico seleccionado.	
Observaciones	
Prototipo de interfaz	

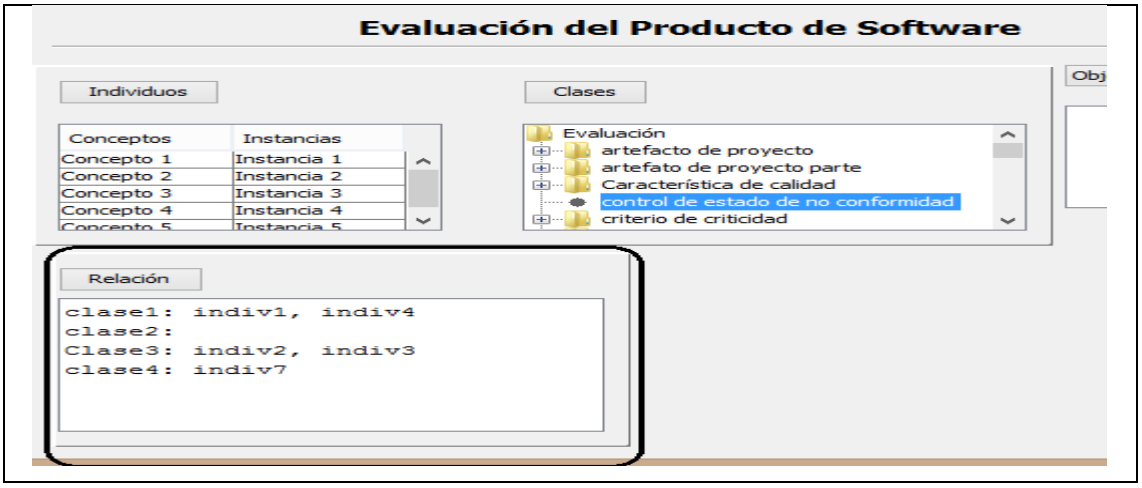


Tabla 5. HU. Mostrar propiedades de objetos. Fuente. Elaboración propia

Historia de usuario	
Número 5	**Nombre** Mostrar propiedades de objetos
Actor Usuario	**Iteración asignada** 1
Prioridad de negocio Alta	**Puntos estimados** 1 semana
Nivel de complejidad Media	**Puntos reales** 1 semana
Descripción Se muestra al usuario todas las relaciones entre objetos existentes en el modelo ontológico seleccionado. Para ello se utiliza un combo box. El principal objetivo de mostrar esta relación es para que el usuario se vincule con el modelo ontológico seleccionado.	
Observaciones	
Prototipo de interfaz	

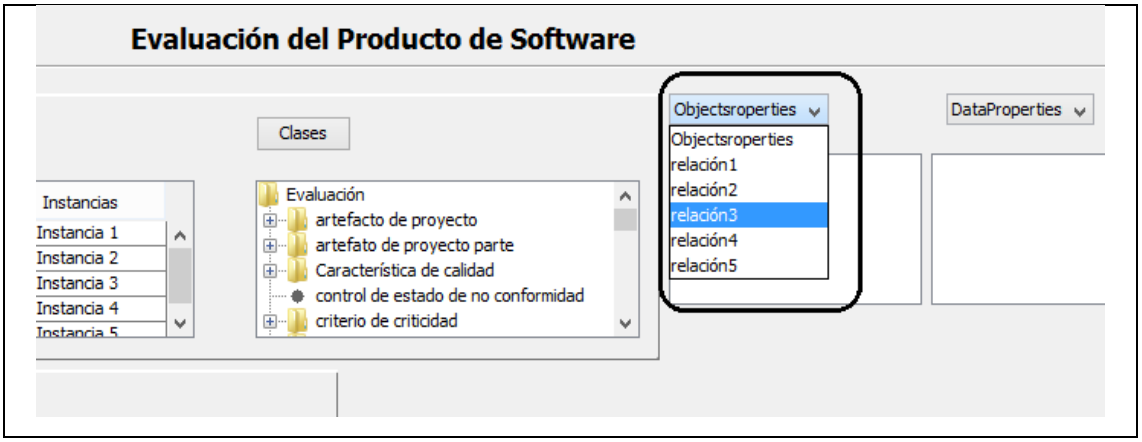


Tabla 6. HU. Mostrar propiedades de datos. Fuente. Elaboración propia

Historia de usuario	
Número 6	**Nombre** Mostrar propiedades de datos
Actor Usuario	**Iteración asignada** 1
Prioridad de negocio Alta	**Puntos estimados** 1 semana
Nivel de complejidad Media	**Puntos reales** 1 semana
Descripción Se muestra al usuario todas las relaciones entre datos existentes en el modelo ontológico seleccionado. Para ello se utiliza un combo box. El principal objetivo de mostrar esta relación es para que el usuario se vincule con el modelo ontológico seleccionado.	
Observaciones	
Prototipo de interfaz	

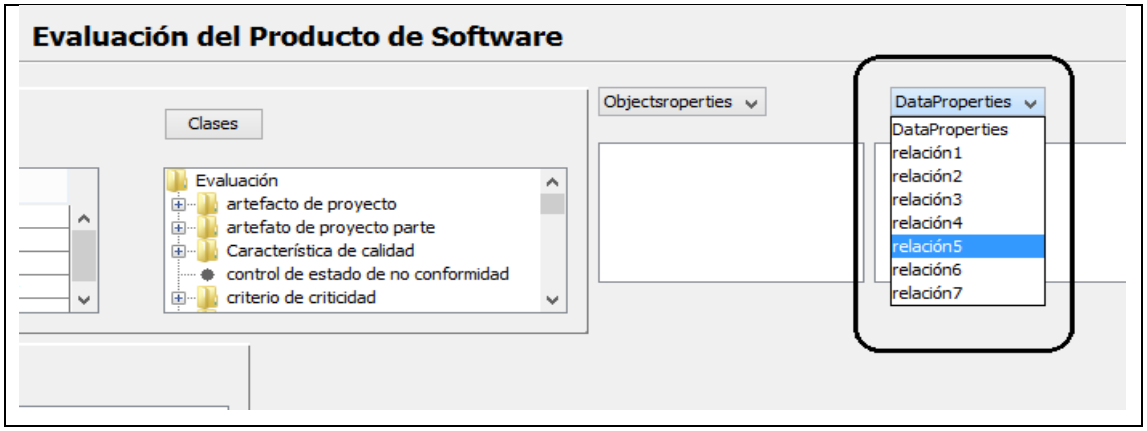


Tabla 7. HU. Consultas a las relaciones. Fuente. Elaboración propia

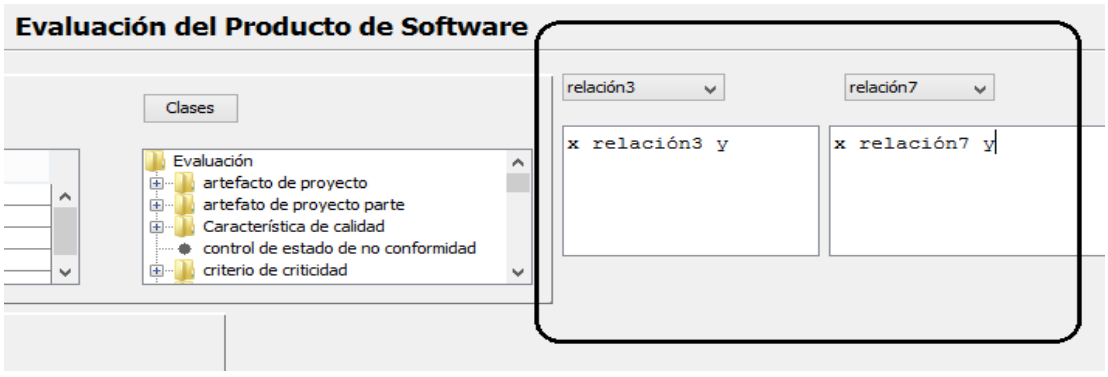
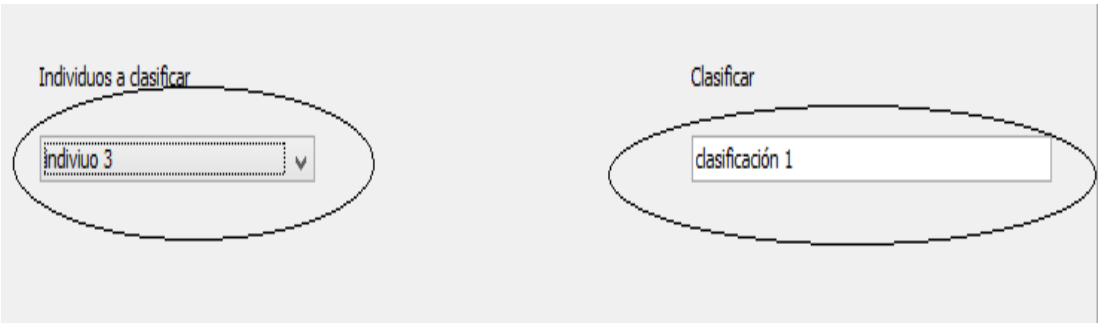
Historia de usuario	
Número 7	**Nombre** Consultas
Actor Usuario	**Iteración asignada** 2
Prioridad de negocio Alta	**Puntos estimados** 1 semana
Nivel de complejidad Alta	**Puntos reales** 1 semana
Descripción De manera sencilla y sin necesidad de tener los conocimientos básicos sobre el lenguaje SAPRQL, el usuario podrá realizar todas las consultas posibles al modelo ontológico seleccionado. Las respuestas a estas consultas serán mostradas en un área de texto debajo.	
Observaciones	
Prototipo de interfaz 	

Tabla 8. HU. Clasificación de las instancias. Fuente. Elaboración propia

Historia de usuario	
Número 8	**Nombre** Mostrar clasificación
Actor Usuario	**Iteración asignada** 3
Prioridad de negocio Alta	**Puntos estimados** 1 semana
Nivel de complejidad Alta	**Puntos reales** 1 semana
Descripción Mediante el uso de un razonador, se muestran todas las clases equivalentes definidas en el modelo ontológico, estas clases van a corresponder con las principales clasificaciones según las definiciones de los axiomas. La información se muestra de la siguiente manera: el usuario selecciona desde un combo box las instancias a clasificar, luego en un campo de texto será mostrada la clasificación correspondiente a dicha instancia.	
Observaciones Esto es logrado con el razonamiento ontológico, ya que el conocimiento representado está implícito en la ontología.	
Prototipo de interfaz 	

La computadora que hará el uso de la aplicación debe constar al menos con 2 Gigabytes de RAM, una capacidad libre en el disco duro de 20 Gigabytes y un microprocesador Dual Core a 3.0 GH. El sistema podrá ser ejecutado desde cualquier el sistema operativo.

2.5 Planificación

En esta fase el cliente establece la prioridad de cada HU, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. La planificación se puede realizar basándose en el tiempo o el alcance (Diaz, y otros, 2013).

2.5.1 Plan de Iteraciones

El plan de iteraciones es uno de los artefactos propuestos por la metodología XP para organizar el proceso de desarrollo del sistema, mostrando la duración y el orden en que serán implementadas las HU de la aplicación en cada iteración (Pino, y otros, 2016).

Tabla 9. Plan de iteraciones. Fuente. Elaboración propia

Iteración	Historia de Usuario	Duración estimada (semanas)
1	Seleccionar la ontología del proceso de Evaluación	4
	Seleccionar la ontología del subproceso de Prueba	
	Mostrar las instancias del proceso de Evaluación	
	Mostrar las instancias del subproceso de Prueba	
	Mostrar las clases del proceso de Evaluación	
	Mostrar las clases del subproceso de Prueba	
	Mostrar las relaciones de individuo clase del proceso de Evaluación	
	Mostrar las relaciones de individuo clase del subproceso de Prueba	
	Mostrar todos las relaciones de datos del proceso de Evaluación	
	Mostrar todos las relaciones de datos del subproceso de Prueba	
	Mostrar todos las relaciones de objetos del proceso de Evaluación	
	Mostrar todos las relaciones de objetos del subproceso de Prueba	
	Consultas SPARQL a las relaciones de objetos del proceso de Evaluación	

2	Consultas SPARQL a las relaciones de objetos del subproceso de Prueba	4
	Consultas SPARQL a las relaciones de datos del proceso de Evaluación	
	Consultas SPARQL a las relaciones de datos del subproceso de Prueba	
3	Principal clasificación en el proceso de Evaluación	3
	Principal clasificación en el subproceso de Prueba	

2.6 Diseño. Tarjetas CRC

La tarjeta CRC (Clase-Responsabilidad-Colaborador) es un artefacto generado por la metodología XP para diseñar una solución informática con el paradigma de la POO. Las clases se refieren a las clases persistentes del sistema a desarrollar, las responsabilidades representan las funciones que realizan dichas clase dentro del sistema y por último se detallan las relaciones que tiene la clase con otras clases persistentes del sistema para cumplimentar la responsabilidad que representa el colaborador. En resumen, una tarjeta CRC representa una entidad del sistema, a la cual se le asigna responsabilidades y colaboraciones, permitiendo el trabajo independiente basado en procedimientos con una metodología basada en objetos (Beck, 2009).

Durante el proceso de diseño de la aplicación se elaboraron 8 tarjetas CRC, a continuación se describen 4, las restantes se encuentran en el anexo 3.

Tabla 10. Tarjeta CRC. Clasificacion_Evaluacion.java. Fuente. Elaboración propia

Clase: Clasificacion_Evaluacion.java	
Responsabilidad:	Colaborador:
➢ Devolver la clasificación correspondiente de la instancia seleccionada.	

Tabla 11. Tarjeta CRC. Controloadora_Evaluacion.java. Fuente. Elaboración propia

Clase: Controladora_Evaluacion.java	
Responsabilidad:	Colaborador:

➢ Es la clase controladora del proceso de Evaluación, es la encargada de relacionar el modelo con la vista.	Manipulando_Ontologia_Evaluacion.java

Tabla 12. Tarjeta CRC. Manipulando_Ontologia_Evaluacion.java. Fuente. Elaboración propia

Clase: Manipulando_Ontologia_Evauacion.java	
Responsabilidad:	Colaborador:
➢ Cargar la ontología del proceso de Evaluación con la biblioteca OWL-API. ➢ Cargar el razonador Pellet para su utilización. ➢ Realizar las principales clasificaciones a la ontología del proceso mediante un razonamiento ontológico.	Clasificacion_Evaluacion.java

Tabla 13. Tarjeta CRC. Proceso_Evaluacion_Jena.java. Fuente. Elaboración propia

Clase: Proceso_Evaluacion_Jena.java	
Responsabilidad:	Colaborador:
➢ Cargar la ontología del proceso de Evaluación utilizando la biblioteca Jena. ➢ Mostrar un listado de las instancias. ➢ Mostrar un listado de las clases. ➢ Mostrar un listado de relaciones. ➢ Realizar las consultas SPARQL correspondientes en cada relación.	

2.7 Arquitectura de software

La arquitectura de *software* representa soluciones a problemas que surgen cuando se desarrolla un *software* en un contexto particular. Una arquitectura de *software* diseñada en

capas consiste en la definición de niveles de abstracción, los cuales tienen una función específica permitiendo un diseño modular. La programación por capas es un estilo de programación en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño. Una variante de este patrón muy utilizada es la de tres capas. Permite dividir la aplicación informática en tres capas, capa de presentación, capa de lógica de negocio y la capa de acceso a datos (Tedeschi, 2015). A continuación se detalla cada capa.

- Capa de presentación: Es la interfaz de comunicación de la aplicación con un usuario determinado, a través de ella se exponen las funcionalidades que se presentan al mismo. Esta capa tiene interacción directa con la de Negocio.
- Capa de negocio: Esta capa es la intermediaria entre la capa de presentación mediante la cual el usuario hace la solicitud y la capa de modelo de datos para obtener la información. Es donde se establecen todas las reglas que deben cumplirse para dar respuesta a las peticiones del usuario.
- Capa de acceso a datos: Es la que gestiona el almacenamiento de los datos, ya sea en una base de datos o en un fichero, así como la consulta a los mismos.

La figura 10 muestra el estilo de la arquitectura seleccionada para el desarrollo del sistema.

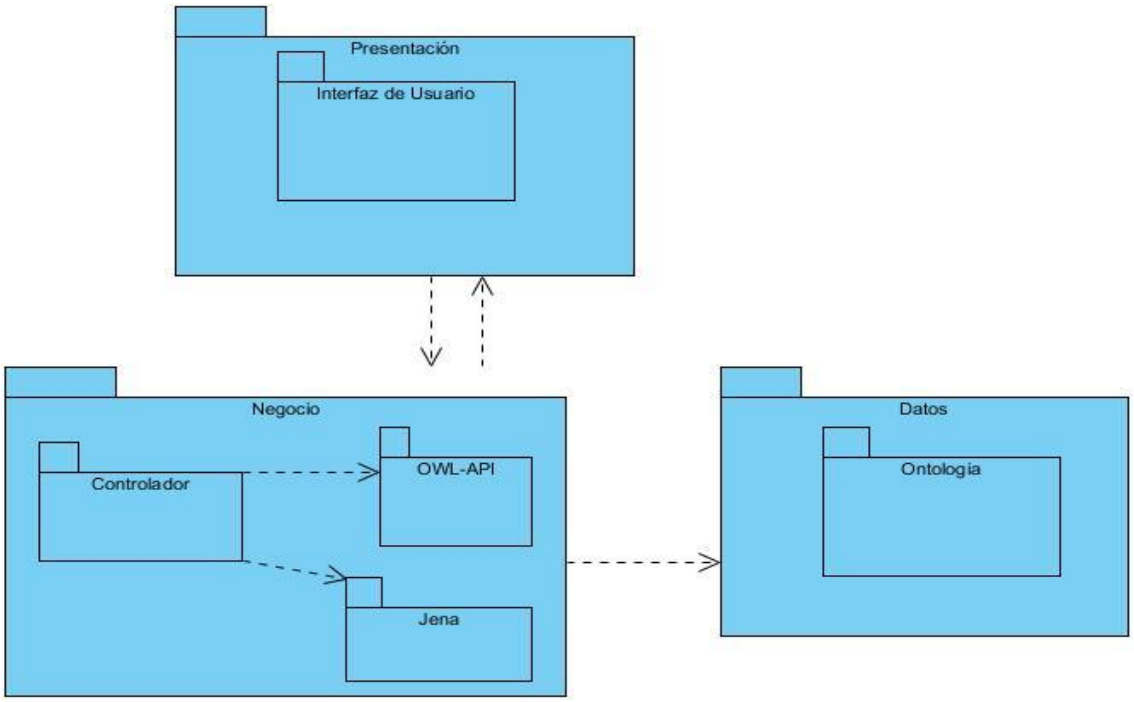


Figura 10. Arquitectura de software. Fuete. Elaboración propia

2.8 Patrones de diseño

Para la realización del diseño de la aplicación se tomaron en cuenta un conjunto de patrones que sirven como guía para resolver problemas comunes que surgen con frecuencia en la programación.

Un patrón de diseño es una solución estándar para un problema común de programación, siendo más práctico a la hora de describir ciertos aspectos de la organización de un programa y conexiones entre componentes de programas (Tedeschi, 2015).

2.8.1 Patrones GRASP

Los patrones GRASP (*General Responsibility Assignment Software Patterns*, en español Patrones Generales de *Software* para la Asignación de Responsabilidades) describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable (Larman, 1999).

Durante el desarrollo del sistema, se aplicaron los siguientes patrones GRASP:

- Experto: Soluciona el problema de asignar una responsabilidad de forma general, tomando decisiones sobre la asignación de responsabilidades a las clases. Una clase tiene toda la información necesaria para realizar la labor que le fue encomendada. Este patrón se puede observar en la clase Manipulando_Ontologia_Evaluacion.java que es la encargada en el sistema de llamar al razonador para lograr mostrar el conocimiento implícito.
- Controlador: Es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma es la clase que recibe los datos del usuario y los envía a las distintas clases según el método llamado. Este patrón es implementado en la clase Controladora_Evaluacion.java, desde la clase Evaluacion.java contenida en el paquete de vistas, es llamada la clase controladora y esta accede a la clase Manipulando_Ontologia_Evaluacion.java que es donde están implementados los métodos principales del modelo ontológico transformado a la POO.

- Bajo acoplamiento: Las clases del sistema se comunican solo con las clases necesarias para desarrollar cada flujo del evento. Este patrón se ve reflejado en la clase Proceso_Evaluacion_Jena.java en la que se realizan todos los métodos vinculados con el proceso de negocio, esta clase solo va a ser llamada por la clase Evaluacion.java contenida dentro de paquete vistas.
- Alta cohesión: la información que almacena una clase debe ser coherente y debe estar (en la medida de lo posible) relacionada con la clase. Se evidencia este patrón en a clase Manipulando_Ontologia_Prueba.java donde solo se trabaja con la información referente al modelo ontológico desarrollado para el proceso de prueba.

2.8.2 Patrón GOF

Los patrones GOF describen soluciones simples y elegantes a problemas específicos en el diseño de *software* orientado a objetos.

- *Singlenton.* (Solitario): Garantiza que cada clase sólo tenga una única instancia, proporcionando un punto de acceso global a la misma. La clase Controlador_Prueba.java es un ejemplo de la utilización del patrón, el acceso a ella está restringido, posee una única instancia.
- *Facade* (Fachada): Provee de una interfaz unificada simple para acceder a otra interfaz o grupo de interfaces de un subsistema. Se ha creado este patrón para reducir la dependencia entre clases. En el sistema esta implementada la clase Servicio.java que consta con todas los métodos. Con esta clase se logra que los cambios realizados en el paquete de negocio, no afecten a la vistas.

2.9 Conclusiones del Capítulo:

Con el desarrollo de este capítulo, se arriban a las siguientes conclusiones.

- La selección de la metodología XP permite que el sistema tenga un desarrollo ágil, iterativo, con poca documentación y que los cambios a los requisitos no se tomen como un problema.
- La definición de las 8 HU, brinda una guía descriptiva de cómo se comportará la implementación del sistema.

- El método para la integración de ontologías adaptado facilitó la definición de un SIBO capaz de realizar una clasificación de la información mediante el razonamiento ontológico.
- La arquitectura de 3 capas y el uso de los patrones de diseño GRASP y GOF permitieron obtener un diseño robusto de la aplicación.

CAPÍTULO III: IMPLEMENTACIÓN Y PRUEBA

3.1 Introducción

En la etapa de implementación de un *software* se desarrolla cada funcionalidad del sistema y se comprueba que se realizó lo propuesto. El resultado final, es un sistema que en la etapa de prueba, se evalúa su desempeño como producto de *software*, además se detectan y corrigen errores para su posterior aceptación. El presente capítulo tiene como objetivo esencial presentar los resultados de la implementación y pruebas realizadas al SIBO. Son descritas las tareas ingenieriles, así como las pruebas unitarias y de aceptación realizadas. Se describe además los estilos de programación.

3.2 Implementación

En esta fase de implementación esas funcionalidades definidas en HU se dividen en tareas más pequeñas, denominadas tareas de ingeniería, con el objetivo de realizar un análisis con mayor detalle y realizar una estimación real de su tiempo de desarrollo. Estas pueden estar descritas por un lenguaje técnico y no ser necesariamente entendible por el cliente y se encargan de guiar la implementación.

Tabla 14. Tarea de Ingeniería 1. Fuente. Elaboración propia

Tarea de Ingeniería	
Número 1	**Nombre de la Historia de Usuario** Seleccionar la Ontología
Número de la Historia de Usuario 1	**Puntos estimados** 0.1
Tipo de tarea Desarrollo	**Programador Responsable** Sergio Alberto Bustamante
Fecha Inicio 12/02/2018	**Fecha Fin** 15/02/2018
Descripción En la ventana principal del sistema, aparece un menú de barra, en este menú aparecen las ontologías involucradas en velar por la calidad de un producto de *software*, ya sea la del proceso de Evaluación o el subproceso de Prueba. El usuario puede seleccionar donde desea trabajar al desplegar el menú.	

Observaciones

Tabla 15. Tarea de Ingeniería 2. Fuente. Elaboración propia

Tarea de Ingeniería	
Número 2	**Nombre de la Historia de Usuario** Mostrar instancias
Número de la Historia de Usuario 2	**Puntos estimados** 0.5
Tipo de tarea Desarrollo	**Programador Responsable** Sergio Alberto Bustamante
Fecha Inicio 16/02/2018	**Fecha Fin** 25/02/2018
Descripción Mediante la POO son recorridas todas las instancias de la ontología seleccionada. Luego son mostradas estas instancias en un panel de salida en lenguaje natural entendibles por los especialistas involucrados.	
Observaciones	

Tabla 16. Tarea de Ingeniería 3. Fuente. Elaboración propia

Tarea de Ingeniería	
Número 3	**Nombre de la Historia de Usuario** Mostrar clases
Número de la Historia de Usuario 3	**Puntos estimados** 0.5
Tipo de tarea Desarrollo	**Programador Responsable** Sergio Alberto Bustamante
Fecha Inicio 26/02/2018	**Fecha Fin** 3/03/2018
Descripción	

Mediante la POO son recorridas todas las clases de la ontología seleccionada. Luego son mostrados los nombres de estas clases en un panel de salida en lenguaje natural entendibles por los especialistas involucrados.
Observaciones Serán mostrados en forma de árbol para lograr una mayor organización del conocimiento.

Tabla 17. Tarea de Ingeniería 4. Fuente. Elaboración propia

Tarea de Ingeniería	
Número 4	**Nombre de la Historia de Usuario** Relación instancias clases
Número de la Historia de Usuario 4	**Puntos estimados** 0.6
Tipo de tarea Desarrollo	**Programador Responsable** Sergio Alberto Bustamante
Fecha Inicio 5/03/2018	**Fecha Fin** 15/03/2018
Descripción Mediante la POO son recorridas todas las clases de la ontología seleccionada, luego se realiza otro ciclo incluido dentro del primero para mostrar todos los individuos con el que este instancia. Son mostradas estas relaciones en un panel de salida en lenguaje natural entendible por los especialistas involucrados.	
Observaciones	

Tabla 18. Tarea de Ingeniería 5. Fuente. Elaboración propia

Tarea de Ingeniería	
Número 5	**Nombre de la Historia de Usuario** Mostrar objects properties
Número de la Historia de Usuario 5	**Puntos estimados** 0.5
Tipo de tarea Desarrollo	**Programador Responsable** Sergio Alberto Bustamante
Fecha Inicio	**Fecha Fin**

16/03/2018	23/03/2018
Descripción Mediante la POO son recorridas todas los data objects de la ontología seleccionada. Luego son mostrados estos datos en un combo box en lenguaje natural entendibles por los especialistas involucrados.	
Observaciones	

Tabla 19. Tarea de Ingeniería 6. Fuente. Elaboración propia

Tarea de Ingeniería	
Número 6	**Nombre de la Historia de Usuario** Mostrar objects properties
Número de la Historia de Usuario 6	**Puntos estimados** 0.5
Tipo de tarea Desarrollo	**Programador Responsable** Sergio Alberto Bustamante
Fecha Inicio 26/03/2018	**Fecha Fin** 29/03/2018
Descripción Mediante la POO son recorridas todas los objects properties de la ontología seleccionada. Luego son mostrados estos datos en un combo box en lenguaje natural entendibles por los especialistas involucrados.	
Observaciones	

Tabla 20. Tarea de Ingeniería 7. Fuente. Elaboración propia

Tarea de Ingeniería	
Número 7	**Nombre de la Historia de Usuario** Consultas
Número de la Historia de Usuario 7	**Puntos estimados** 0.7
Tipo de tarea Desarrollo	**Programador Responsable** Sergio Alberto Bustamante

Fecha Inicio	**Fecha Fin**
30/03/2018	17/04/2018
Descripción En la herramienta donde fueron creadas y exportadas las ontologías, en este caso fue el Protégé, el cual presenta un plugin donde se realizan consultas SPARQL, los especialistas definen varias consultas. Luego estas consultas SPARQL definidas son copiadas al IDE utilizado para transformarlas a POOy así poder mostrar a los usuarios involucrados en los procesos de velar por la calidad de un producto de *software* el resultado de las mismas de una manera asequible para estos.	
Observaciones	

Tabla 21. Tarea de Ingeniería 8. Fuente. Elaboración propia

Tarea de Ingeniería	
Número 8	**Nombre de la Historia de Usuario** Mostrar clasificación
Número de la Historia de Usuario 8	**Puntos estimados** 0.8
Tipo de tarea Desarrollo	**Programador Responsable** Sergio Alberto Bustamante
Fecha Inicio 19/04/2018	**Fecha Fin** 19/05/2018
Descripción Mediante la POO y utilizando la biblioteca OWL-API, es importado un razonador, en este caso fue Pellet. A La ontología seleccionada se le realiza un razonamiento ontológico donde se logra mostrar conocimiento implícito dentro de la ontología. Esta información va a ser equivalente con la clasificación principal de las instancias dado un serie de axiomas.	
Observaciones	

3.3 Diagrama de Componentes

Los diagramas de componentes permiten describir los elementos físicos que integran el sistema y las relaciones que existen entre ellos. Los componentes representan todos los

tipos de elementos *software* que entran en la producción de aplicaciones informáticas. Pueden ser simples archivos, paquetes, y/o bibliotecas cargadas dinámicamente (Larman, 1999).

A continuación, se expone el Diagrama de Componentes asociado a varios de los subsistemas de implementación identificados.

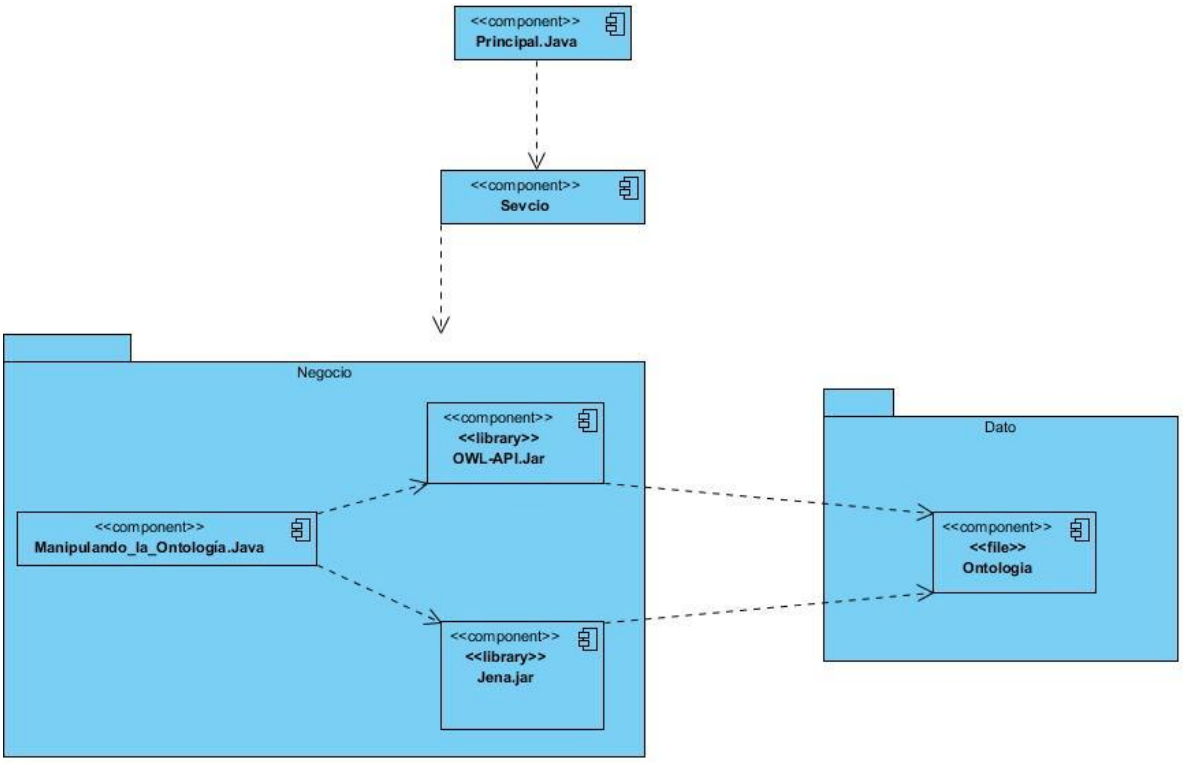


Figura 11. Diagrama de componentes. Fuete. Elaboración propia

3.4 Estándares de codificación

Un completo estándar de codificación comprende todos los aspectos de la generación de código. Un código fuente completo debe reflejar un estilo armonioso, como si se hubiese escrito el código una única vez por el mismo programador. Es recomendado establecer un estándar de codificación al comenzar un proyecto de *software*, para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. XP propone una serie de convenciones o estándares de códigos enfocados a la estructura, apariencia física de la aplicación, comprensión y mantenimiento del código. A continuación se muestran algunas de estas convenciones tenidas en cuenta para la implementación del sistema.

- Identación: Se debe emplear cuatro espacios como unidad de identación (ver figura 12).

```
public void setClasificacion(String Clasificacion) {
    this.Clasificacion = Clasificacion;
}
```

Figura 12. Identación. Fuente. Elaboración propia

- Longitud de línea: Evitar las líneas con más de 120 caracteres, ya que estas no van a ser bien manejadas por deferentes terminales y herramientas.
- Comentarios: Existen dos tipos de comentario, los de implementación y los de documentación. Este último se encuentra limitado por /*…*/.
- Declaración: Se realiza una declaración por línea, ya que facilita los comentarios.

```
String iri = owlNamedIndividual.getIRI().toString();
String[] arregloAuxiliar = iri.split(NS);
```

Figura 13. Declaración. Fuente. Elaboración propia

- Sentencia for-each: La sentencia for-each debe usar siempre llaves {}, en la figura 14 se muestra un ejemplo de esta sentencia.

```
for (OWLClass clases : ontology.getClassesInSignature()){
    System.out.println(clases);
    }
```

Figura 14. Sentencia for-each. Fuente. Elaboración propia

Los nombres de las clases deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas (Ej. Principal, Clasificacion_Evaluacion.java). Mantener los nombres de las clases simples y descriptivas. Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo). Los nombres de las interfaces siguen la misma regla que las clases.

Los métodos deben ser verbos, cuando son compuestos tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma en mayúscula (Ejemplo. getOWLDataFactory()).

3.5 Pruebas de Software

La prueba es un proceso de ejecución de un programa con la intención de descubrir errores, las cuales incluyen un conjunto de tareas que garantizan que el *software* implemente correctamente funciones específicas y aseguran que se construye cumpliendo con los requerimientos del cliente.

En este proceso se ejecutan pruebas dirigidas a componentes del *software* o al sistema de *software* en su totalidad, con el objetivo de medir el grado en que el *software* cumple con los requerimientos. En las pruebas se usan casos de prueba, especificados de forma estructurada mediante técnicas y estrategias de prueba.

Existen diferentes estrategias de pruebas, pero el estudio de este trabajo se ha centrado en las pruebas de la metodología de desarrollo de *software* empleada en el presente trabajo de diploma, XP, la cual divide las pruebas de sistema en dos grupos: pruebas unitarias y pruebas de aceptación.

- Pruebas unitarias

Enfoca los esfuerzos de verificación en una unidad pequeña del *software*, se prueba a descubrir errores dentro de la frontera de la unidad enfocándose en la lógica de procesamiento interno y de las estructuras de datos. Generalmente son realizadas por el mismo programador, debido a que al conocer con mayor detalle el código, se les simplifica la tarea de elaborar conjuntos de datos de prueba para testearlo. Las pruebas unitarias siempre están centradas en la técnica de caja blanca.

En las pruebas de caja blanca se comprueban los caminos lógicos del *software* proponiendo casos de prueba que examinen que están correctas todas las condiciones y/o bucles para determinar si el estado real coincide con el esperado. Se genera gran variedad de posibles caminos, por lo que hay que dedicar esfuerzos a las condiciones de prueba que serán analizadas.

Se utiliza la técnica de camino básico para guiar las pruebas de caja blanca. Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico.

La esencia es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática.

- Pruebas de Aceptación

Las pruebas de aceptación, se realizan sobre el producto terminado, listo para implantarse en el entorno del cliente, están concebidas para que sea un usuario final quien detecte los posibles errores. En esta prueba se evalúa el grado de calidad del *software* con relación a todos los aspectos relevantes para que el uso del producto se justifique. Este tipo de pruebas son comúnmente realizadas por el usuario final, quien debe informar de todas las deficiencias o errores que encuentre antes de dar por aprobado el sistema definitivamente.

Para asegurarse que la aplicación desarrollada cumple sus requisitos, se definió realizarle pruebas de aceptación, puesto que representan la satisfacción del cliente con el producto desarrollado. Estas pruebas conllevan al cliente a precisar lo que la aplicación debe hacer en determinadas circunstancias. Por ello es que el cliente es la persona ideal para diseñar las dichas pruebas.

3.6 Técnicas de prueba

Cualquier proyecto que trace una estrategia de prueba debe contar con métodos y técnicas para la aplicación de cada una de estas pruebas. A continuación, se realiza una breve caracterización de dos técnicas importantes.

Pruebas de Caja Blanca

Las pruebas de caja blanca en ocasiones llamadas pruebas de cajas de vidrio se basan en el examen cercano de los detalles de procedimiento del *software*.

Estas pruebas de caja blanca fueron aplicadas con la técnica del camino básico, con el objetivo de evaluar la complejidad lógica de un diseño procedimental y usar esta medida como guía para la definición de un conjunto básico de caminos de ejecución. Esta prueba permite garantizar que en los casos de prueba obtenidos a través del camino básico se ejecute cada sentencia del programa por lo menos una vez.

A continuación se muestra un ejemplo de aplicación de esta técnica.

En la clase Proceso_Evaluacion_Jena.java se encuentra contenido el método consultas_data (String s). Donde se analizó y enumeró las sentencias del código que será mostrado en la figura 15. Este método permite realizar consultas SPARQL a todas las relaciones de datos existentes en la ontología.

```
98
99      public String consultas_data(String s) {
100     String temp = "PREFIX evaluacion: <http://www.semanticweb.org/parker/ontologies/2017/10/untitled-ontology-16#>\n"
101             + "SELECT ?subject ?object\n"
102             + "     WHERE { ?subject evaluacion:" + s + " ?object }";
103
104     Query query_temp = QueryFactory.create(temp);
105
106     QueryExecution qa_temp = QueryExecutionFactory.create(query_temp, model);
107     ResultSet out = qa_temp.execSelect();
108
109     String salida = "";
110
111     List<QuerySolution> arreglo = ResultSetFormatter.toList(out);
        for (QuerySolution querySolution : arreglo) {
113         String subjectlargo = querySolution.get("?subject").toString();
114         String objectlargo = querySolution.get("?object").toString();
115         String[] arregloAuxiliarsubject = subjectlargo.split(NS);
116         String[] arregloAuxiliarobject = objectlargo.split(NS);
117         // System.out.println("La salida es:   "+arregloAuxiliarsubject[1]);
118         // System.out.println("La salida es:   "+arregloAuxiliarobject[1]);
119         salida += arregloAuxiliarsubject[1] + "   " + s + "   " + arregloAuxiliarobject[0] + "\n";
120     }
121     return salida;
122 }
123
```

Figura 15. Método para consultas SPARQL. Fuente. Elaboración propia

Luego, es necesario representar el grafo de flujo asociado al código antes presentado a través de nodos, aristas y regiones como se muestra en la figura 16.

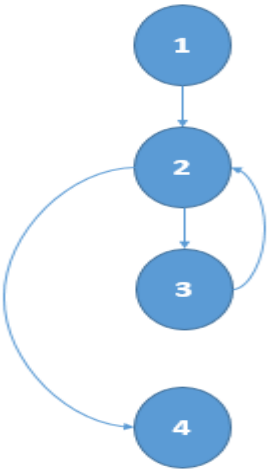


Figura 16. Flujo generado por el método para consultas SPARQL. Fuente. Elaboración propia

Construido el grafo de flujo asociado al procedimiento anterior, se determina la complejidad ciclomática, esta es una métrica de *software* muy útil y proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar.

Para el cálculo de la complejidad ciclomática, se emplean las siguientes ecuaciones:

$$V(g) = (e - n) + 2 * p \qquad (1)$$

$$V(g) = p + 1 \qquad (2)$$

$$V(g) = r \qquad (3)$$

Donde:

- $V(g)$ valor de la complejidad ciclomática
- e número de aristas del grafo
- n cantidad de nodos total en el grafo
- p cantidad de nodos predicados (nodos que tienen más de una salida) existentes
- r la cantidad total de regiones

Aplicando las ecuaciones 1, 2 y 3 al grafo mostrado en la figura 17, se obtiene que la complejidad ciclomática es igual a 2, lo cual se interpreta que existen dos caminos posibles por donde el flujo puede circular

$$V(g) = (4 - 4) + 2 * 1$$

$$V(g) = 2$$

$$V(g) = 1 + 1 = 2$$

$$V(g) = 2$$

Estos caminos son:

Camino básico # 1: 1 – 2 – 3 – 2 - 4

Camino básico # 2: 1 – 2 – 4

Lo anterior conlleva a realizar dos casos de prueba para este ejemplo.

Pruebas de Caja Negra

Las pruebas de caja negra también llamadas pruebas de comportamiento se refieren a las pruebas que se llevan a cabo en la interfaz del *software*. Examina algunos aspectos fundamentales de un sistema con poca preocupación por la estructura lógica interna del *software*. Se enfocan en los requerimientos funcionales del *software*; es decir, las técnicas de prueba de caja negra permiten derivar conjuntos de condiciones de entrada que revisarán por completo todos los requerimientos funcionales para un programa.

- Casos de prueba

Los casos de prueba contienen un conjunto de valores de entrada, precondiciones de ejecución, resultados esperados y pos condiciones de ejecución, desarrollados para un objetivo particular de condición de prueba, tal como para ejercer una ruta de un programa en particular o para verificar el cumplimiento de un requisito específico.

A continuación, se muestra el Diseño de caso de prueba correspondiente a la funcionalidad de “Clasificación”.

Tabla 22. Caso de prueba. Fuente. Elaboración propia

Descripción general: Consultar la clasificación de la instancia.			
Condiciones de ejecución: para consultar una clasificación, debe de estar predefinida la instancia dentro de una clase equivalente de la ontología.			
Escenario	**Descripción**	**Respuesta del sistema**	**Flujo central**
EC 1.1 Consultar individuo	El usuario selecciona la opción (Seleccionar un individuo)	El sistema muestra la clase equivalente de dicho individuo	Principal/Clasificación

- Resultados de las pruebas

Con la aplicación de las pruebas de *software* se logra corregir los problemas presentes en las funcionalidades.

En las pruebas de aceptación realizadas se comprobó el cumplimiento de los requisitos propuestos, arrojando los siguientes resultados mostrados en la figura 17:

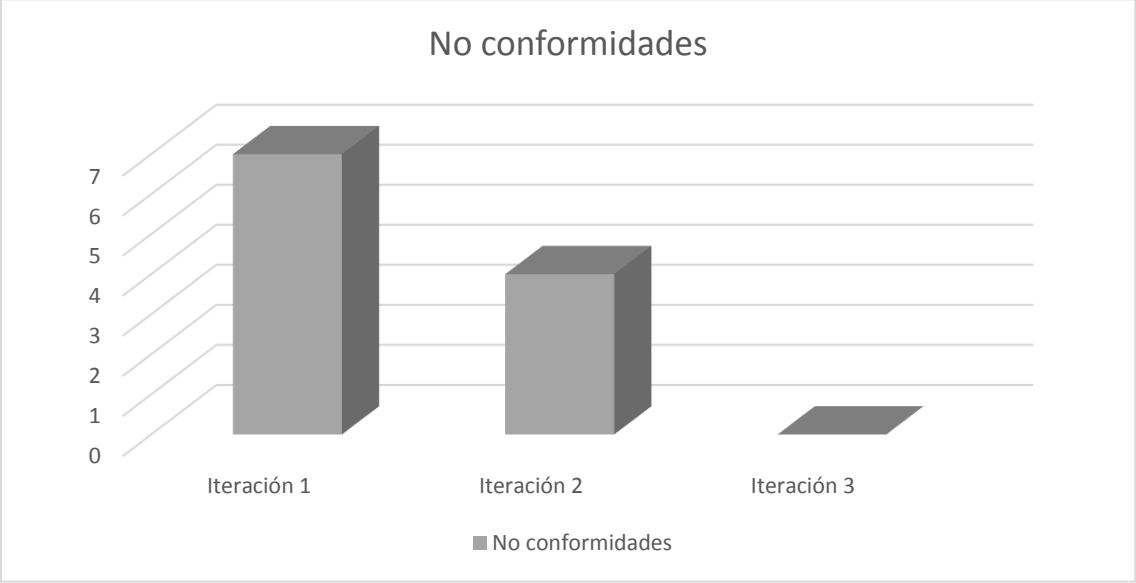


Figura 17. Resultado de las pruebas. Fuente. Elaboración propia

Las pruebas se aplicaron a 8 HU en 3 iteraciones.

3.7 Conclusiones del Capítulo

Al concluir este capítulo se arribaron a las siguientes consideraciones:

- Mediante las tareas de ingeniería se evitó la sobrecarga de trabajo en el equipo de desarrollo, esto logró mejor organización a la hora de la implementación del sistema y agilizó el proceso.
- El diagrama de componentes, permitió reflejar la distribución física de los elementos del diseño.
- El estándar de codificación seleccionado para el lenguaje de programación Java permitió organizar el código de la solución.
- La aplicación de las pruebas unitarias permitieron validar las funcionalidades críticas del sistema, mientras que las pruebas de aceptación posibilitaron comprobar la correcta implementación de las HU definidas con anterioridad.

CONCLUSIONES

Luego de realizar el desarrollo del SIBO para gestionar el conocimiento generado en los procesos que velan por la calidad de un producto de *software* en la dirección de calidad de la UCI, se ha arribado a las siguientes conclusiones.

- A partir del estudio realizado referente a los SIBOs existentes se determinaron las principales tecnologías, herramientas y metodología para su creación, permitiendo obtener la información necesaria que ha servido de base para ejecutar la solución que se propone.
- El uso de un enfoque de desarrollo ágil con XP, así como de herramientas y tecnologías tales como Visual Paradigm, el lenguaje de programación Java y NetBeans facilitó el correcto desarrollo del sistema de información.
- La implementación del sistema para la evaluación de productos de *software* demostró la validez del método propuesto para la integración de ontologías y conllevó a la utilización de dos ontologías que permitieron formalizar el conocimiento sobre los indicadores de prueba y las evaluaciones de productos.
- Con la realización de las pruebas de software se validó el correcto funcionamiento del sistema y se demostró que la implementación satisface los requisitos definidos por el cliente.
- El SIBO para la evaluación de productos de software facilitó la gestión del conocimiento generado en dicho proceso en la Dirección de Calidad de la UCI.

RECOMENDACIONES

- Se propone crear una versión 2.0 de este SIBO para la evaluación de productos de *software* en la dirección de calidad de la UCI, donde sea capaz de agregar y eliminar instancias, conceptos y relaciones de sus modelos ontológicos poblados para así lograr una independencia total del Protégé.

REFERENCIAS

Álvarez, Margarita, y otros. Sistemas de Información basados en ontologías. Un área emergente. Santiago del Estero : s.n.

Bañon Peñuelas, José María. 2013. Estudio del manejo de ontologías para la monitorización de pacientes. Madrid, España : Universidad de Carlos III de Madrid, Octubre de 2013. págs. 31-32.

Barchini, Graciela, Álvarez, Margarita y Herrera, Susana. 2006. Sistemas de Información: Nuevos escenarios basados en Ontologias. Santiago de Estero : s.n., 2006. 1807-1775.

Barchini, Graciela, y otros. 2007. El rol de las Ontologías en los Sistemas de Información. Santiago del Estero : s.n., 2007.

Barchini, Graciela, y otros. 2009. Evaluación de la calidad delos sistemas de Información basados en Ontologías. Valencia, España : s.n., Marzo de 2009.

Beck. 2009. *Extreme Programming Explained.* 2009.

C. Blanco, J. Lasheras, R. Valencia-garcía, E. Fernández-medina, and A. Toval. 2008. *"Ontologías de Seguridad : Revisión sistemática y comparativa".* Departamento de Tecnología y Sistemas de Información., Universidad Castilla de la Mancha. Ciudad Real. Castilla de la Mancha. España : Ministerio de Educación y Ciencia., 2008. Informe Técnico UCLM-TSI-003.

Canals, Agustín. 2003. La Gestión del conocimiento. Julio de 2003.

Castañeda Martinez, Aliuska, y otros. 2017. *Diseño de una ontología para la evaluación de productos de software en la UCI.* CALIPROT, UCI. Habana : s.n., 2017.

Diaz, Leiny y Garcia, Raul Alejandro. 2013. Aplicación Informática para gestionar Ontlogías representativas del conocimiento en la web. La Habana, Cuba : s.n., junio de 2013.

Díaz, Midelyn, Contreras, Timian y Rivero, Soleidys. 2009. Características de los sistemas de Información que permiten la gestión oportuna de la Información y el conocimiento institucional. octubre de 2009.

Echeverría Pérez, Delvis, Fernández Pérez, Yamilis y Pozo Zulueta , Delmys. 2012. "Ontología de apoyo al procedimiento de pruebas del Departamento de Pruebas de Software de Calisoft". Panamá : LACCEI'2012, 2012.

Espín Martín, Vanesa. 2016. Sistemas de recomendación semánticos para la comparticion de conocimiento y la explotación de tesauros: un enfoque práctico en los sistemas nutrcionales. . [ed.] Univrsidad de Granada Tesis Doctorales. Granada, España : s.n., mayo de 2016. 978-84-9125-959-6.

Fernández Pérez, Yamilis. 2018. Modelo computacional para la evaluación y selección de productos de software. Granada, España : s.n., 2018.

Ferreira de Souza, Érica. 2014.. *"Doctorate Thesis: Knowledge management applied to software testing : an ontology based framework".* Sao José dos Campos, SP, Brasil : INPE, 2014.

Gruber, Thomas. 1993. Toward Principles for the Design of Ontologies Use for Knowledge Sharing. Stanfort : s.n., 1993.

Guarino. 1998. *Formal Ontology and Information Systems.* Amsterdam. : s.n., 1998.

IEEE Computer Society, S. E. StandardsCommittee. 2009.. *IEEE Std 1061-1998 IEEE Standard for a Software Quality Metrics Methodology ".* New York. Estados Unidos : IEEE Computer Society, 2009.

International Hellenic University. 2013. Knowledge Management (in the Web) . Manchester : s.n., 2013.

ISO/IEC. 2008. *"ISO / IEC 15939: 2007 Systems and software engineering — Measurement process".* Switzerland : ISO/IEC, 2008.

ISO/IEC/ONN-NC. 2011. *NC ISO/IEC 25000:2011 Ingeniería de software— Requisitos de calidad de productos de software(SQuaRE).* La Habana, Cuba : Cuban National Bureau of Standards, 2011.

Java. Java. [En línea] https://www.java.com/es/.

Jena, Apache. 2011. [En línea] 2011. [Citado el: 13 de Noviembre de 2017.] https://jena.apache.org/index.html.

Larman, Craig. 1999. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Mexico : s.n., 1999. 970-17-0261-1.

Leger, Alain, Aarno, Lehtola y Vilagra, Victor. 2001. Developing Mutilingual Knowledge-Besed Marketplace. 2001.

Lenguaje de programación C++. La Revista Informática.com.

López Rodriguez, Yoan Antonio. 2017. Metodo para la Integración de ontologías en sistemas relacionales para la evaluacion de creditos. La Habana : s.n., 2017.

Maida, Esteban Grabiel y Pacienzia, Julián. 2015. *Metodologías de desarrollo de software.* Argentina : Universidad Católica Argentina, 2015.

Manchester, T. U. O. 2016. *Updated List of OWL reasoners. Information Management Group at the School of Computer Science, The University of Manchester, United Kingdom.* 2016.

NetBeans. 2012. NetBeans. *Oracle Corporation and/or its affiliates, 2012.* [En línea] 2012. [Citado el: 12 de noviembre de 2014.] http://netbeans.org/..

Palmisano, Ignazio. 2011. The Rough Guide to the OWL API: a tutorial Version 3.2.3 for OWL 2. Manchester : University of Manchester, 2011.

Paradigm, Visual. 2012. Visual Paradigm. *Visual Paradigm.* [En línea] 2012. [Citado el: 11 de noviembre de 2014.] http://www.visual-paradigm.com./..

PEGASUS. PEGASUS. [En línea] https://web.colpegasus.com/.

Pino, David y Castillo, Tony. 2016. Sistema informático de ayuda a la decisión para la evaluación de la calidad de productos de software. La Habana, La Habana, Cuba : s.n., Junio de 2016.

Preesman, Roger. 2010. Ingeniería del software. Un enfoque práctico. 7 [ed.] Pablo Roig Vazquez. Mexico : s.n., 2010. 978-607-15-0314-5.

Real Academia Española. 2018. Diccionario de la lengua española. [En línea] 2018. http://dle.rae.es/..

Riquelme Santiago, Yanet. 2015. Sistema de información basado en ontología para la gestión del conocimiento y la toma de deciciones en en el sistema de gestión de proyectos XEDRO GESPRO. La Habana : s.n., Junio de 2015.

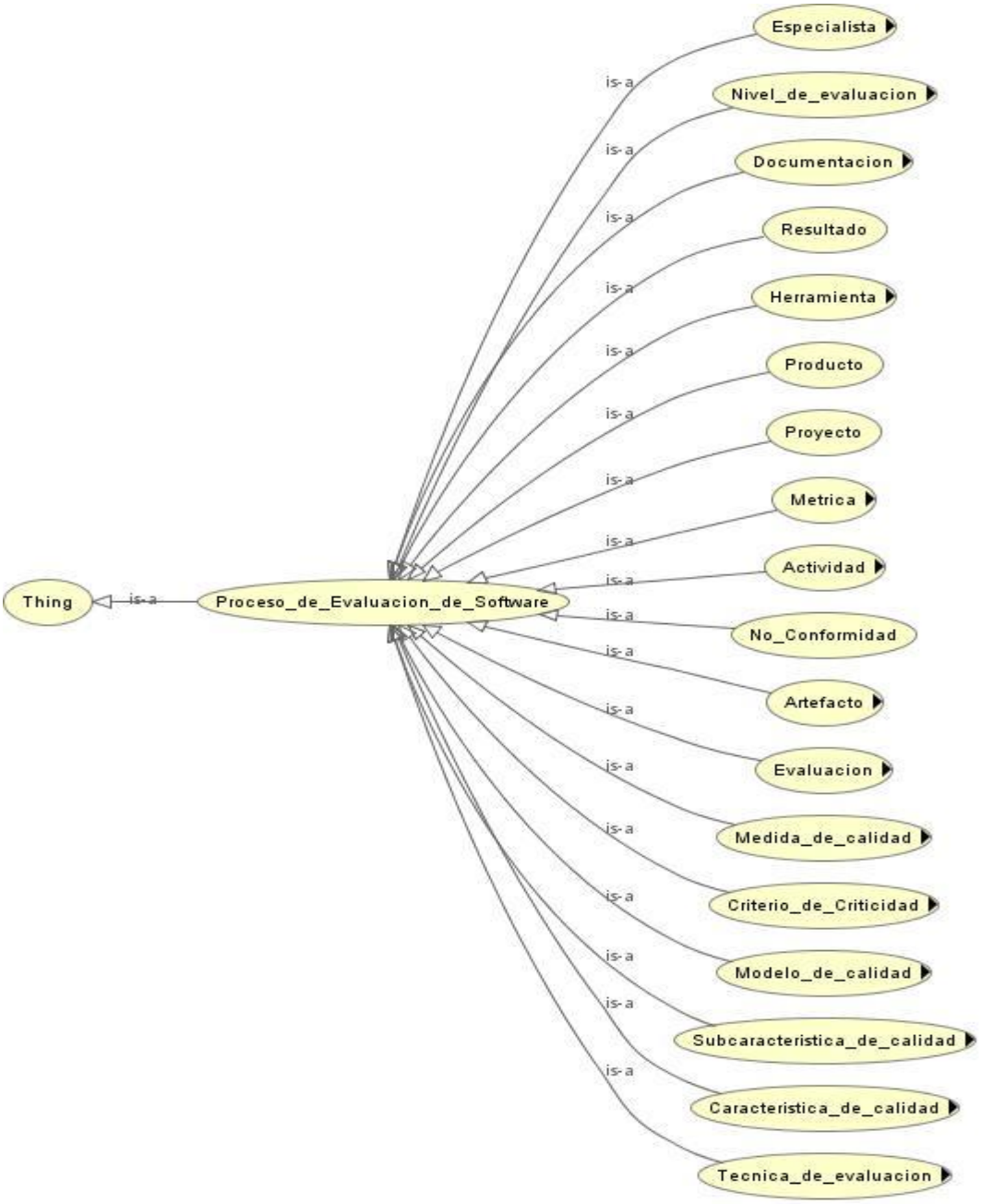
Tedeschi, Nicolás. 2015. ¿Qué es un Patrón de Diseño? [En línea] Microsoft, 2015. https://msdn.microsoft.com/es-es/library/bb972240.aspx.

Visual Paradigm. 2014. Visual Paradigm. [En línea] 2014. [Citado el: 12 de 12 de 2017.] http://www.ie.inf.uc3m.es/grupo/docencia/reglada/Is1y2/PracticaVP.pdf.

Vyeira, Victor. 2016. mindmeister. *Herramientas CASE.* [En línea] 9 de 7 de 2016. [Citado el: 8 de 12 de 2017.] https://www.mindmeister.com/es/752475322/herramientas-case.

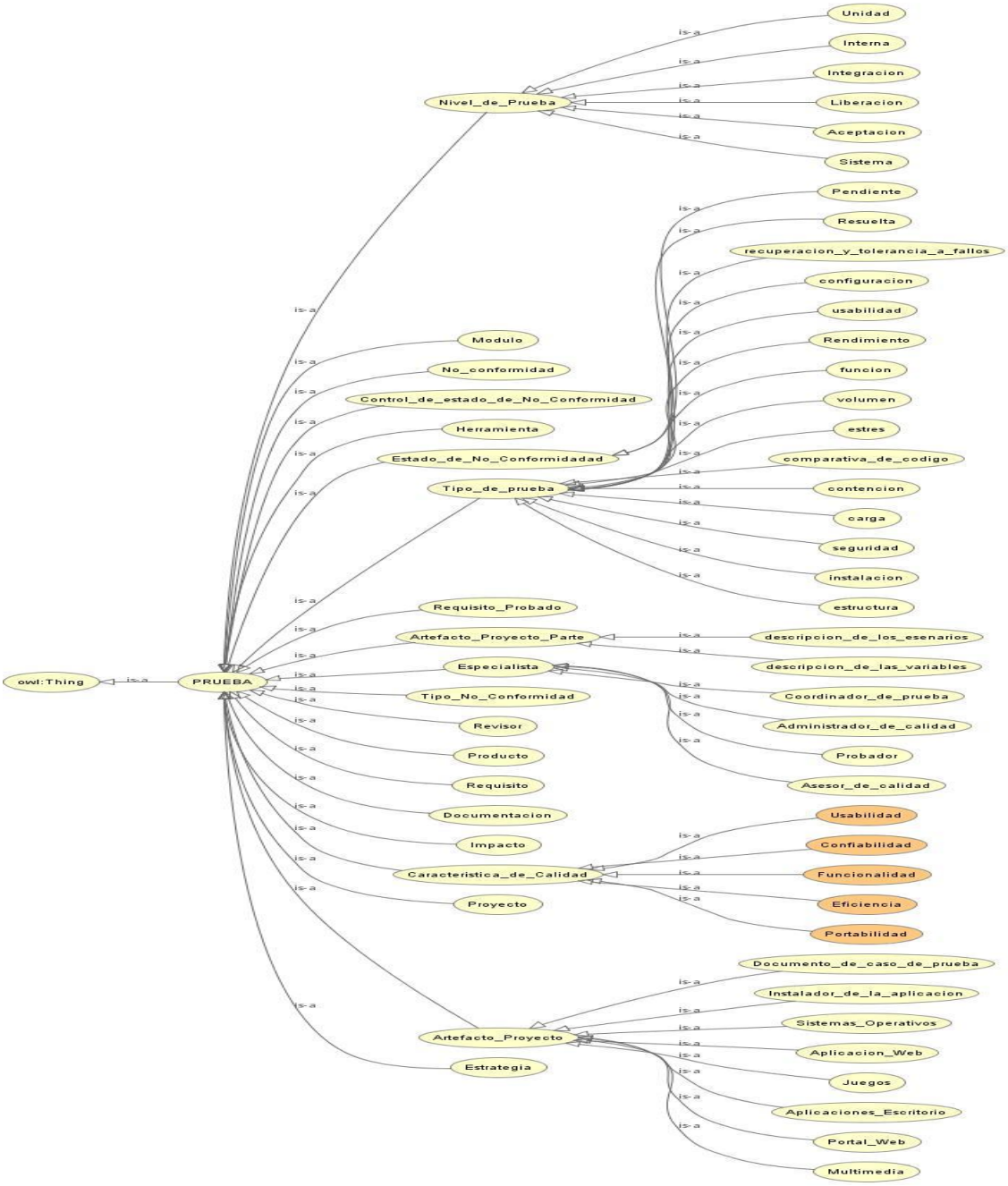
ANEXOS

Anexo 1. Ontología del proceso de Evaluación



No.	**Instancias**	**Descripción**
1	act1	Instancia perteneciente a la clase Actividad
2	artefacto1	Instancia perteneciente a la clase Artefacto
3	cc1	Instancia perteneciente a la clase Características de calidad
4	criterioc1	Instancia perteneciente a la clase Criterio de criticidad
5	documentación1	Instancia perteneciente a la clase Documentación
6	especialista1	Instancia perteneciente a la clase Especialista
7	evaluación1	Instancia perteneciente a la clase Evaluación
8	herramienta1	Instancia perteneciente a la clase Herramienta
9	medida de calidad1	Instancia perteneciente a la clase Medida de calidad
10	mc1	Instancia perteneciente a la clase Modelo de calidad
11	métrica1	Instancia perteneciente a la clase Métrica
12	nivel1	Instancia perteneciente a la clase Nivel de evaluación
13	nc1	Instancia perteneciente a la clase No Conformidad
14	producto1	Instancia perteneciente a la clase Producto
15	proyecto1	Instancia perteneciente a la clase Proyecto
16	resultado1	Instancia perteneciente a la clase Resultado
17	subcaract1	Instancia perteneciente a la clase Sub-característica de calidad
18	técnica1	Instancia perteneciente a la clase Técnica de calidad

Anexo 2. Ontología del subproceso de Prueba



No.	**Instancias**	**Descripción**
1	ap1	Instancia perteneciente a la clase Artefacto de Proyecto
2	apparte1	Instancia perteneciente a la clase Artefacto Proyecto parte
3	cc1	Instancia perteneciente a la clase Característica de Calidad
4	controlestado1	Instancia perteneciente a la clase Control de Estado de No Conformidad
5	documentacion1	Instancia perteneciente a la clase Documentación
6	especialista1	Instancia perteneciente a la clase Especialista
7	estado1	Instancia perteneciente a la clase Estado de No Conformidad
8	estrategia1	Instancia perteneciente a la clase Estrategia
9	herramienta1	Instancia perteneciente a la clase Herramienta
10	imp1	Instancia perteneciente a la clase Impacto
11	modulo1	Instancia perteneciente a la clase Módulo
12	nivelp1	Instancia perteneciente a la clase Nivel de Prueba
13	nc1	Instancia perteneciente a la clase No Conformidad
14	producto1	Instancia perteneciente a la clase Producto
15	proyecto1	Instancia perteneciente a la clase Proyecto
16	r1	Instancia perteneciente a la clase Requisito
17	rp1	Instancia perteneciente a la clase Requisito Probado
18	revisor1	Instancia perteneciente a la clase Revisor
19	tipop1	Instancia perteneciente a la clase Tipo de prueba
20	tnc1	Instancia perteneciente a la clase Tipo de No Conformidad

Anexo 3. Tarjetas CRC del modelo ontológico importado

Clase: Clasificacion_Prueba.java	
Responsabilidad:	Colaborador:
➢ Devolver la clasificación correspondiente de la instancia seleccionada.	

Clase: Controladora_Prueba.java	
Responsabilidad:	Colaborador:
➢ Es la clase controladora del proceso de Evaluación, es la encargada de relacionar el modelo con la vista.	Manipulando_Ontologia_Prueba.java

Clase: Manipulando_Ontologia_Prueba.java	
Responsabilidad:	Colaborador:
➢ Cargar la ontología del proceso de Evaluación con la biblioteca OWL-API. ➢ Cargar el razonador Pellet para su utilización. ➢ Realizar las principales clasificaciones a la ontología del proceso mediante un razonamiento ontológico.	Clasificacion_Prueba.java

Clase: Proceso_Prueba_Jena.java	
Responsabilidad:	Colaborador:
➢ Cargar la ontología del proceso de Evaluación utilizando la biblioteca Jena. ➢ Mostrar un listado de las instancias.	

➢ Mostrar un listado de las clases. ➢ Mostrar un listado de relaciones. ➢ Realizar las consultas SPARQL correspondientes en cada relación.	