SOPORTE RECUPERATORIO DE ARQUITECTURAS DE SOFTWARE A TRAVÉS DE ONTOLOGÍAS

DAVID PULIDO

GABRIEL MEJÍA COELLO

TRABAJO DE GRADO

UNIVERSIDAD PILOTO DE COLOMBIA

FACULTAD DE INGENIERÍA DE SISTEMAS

BOGOTÁ D.C

11-2014

LISTA DE TABLAS

GLOSARIO

ONTOLOGÍA es un constructo ingenieril que consiste en un esquema de conceptos y las relaciones entre los mismos para describir arduamente el dominio de un problema.

WEB SEMÁNTICA es una actualización de la “web tradicional” en donde los contenidos de la misma se encuentran categorizados otorgándoles significado, siendo posible de esta manera realizar consultas a un nivel de especificación más flexible y detallado.

STAKEHOLDERS son las personas que poseen el deseo de otorgar una solución informática a uno de sus problemas y contratan a especialistas o casas creadoras de software para ello.

ARQUITECTURA DE SOFTWARE es un conjunto de decisiones que los arquitectos do software deben tomar para satisfacer las necesidades de los stakeholders.

DECISIONES DE ARQUITECTURA DE SOFTWARE son una explicación detallada acerca de lo que se le debe añadir o sustraer a una arquitectura de software, junto con su correspondiente justificación, reglas y restricciones que rigen su desarrollo.

REQUERIMIENTO FUNCIONAL es una descripción de una necesidad de un stakeholder.

REQUERIMIENTO NO FUNCIONAL es una descripción de una condición o restricción que debe tomarse en cuenta en el proceso de desarrollo, inherente a los intereses del stakeholder.

WIKI es un sitio web donde el contenido de la misma puede ser añadido, modificado e incluso eliminado por los usuarios utilizando su navegador.

WIKI SEMÁNTICA es una wiki en donde se implementa un enfoque de administración de conocimiento (usualmente una ontología). Wiki dotada de significado, en donde cada una de sus páginas (conceptos o entidades) se relacionan de una manera más descriptiva entre una y otra.

ADMINISTRACIÓN DEL CONOCIMIENTO es un modelo que describe lo que se debe hacer para comprender los conceptos o información inherente a un proceso u objeto de conocimiento, usualmente se consideran la captura, la socialización y recuperación de información como las actividades dentro de un modelo básico.

1. DESCRIPCIÓN DEL PROBLEMA

La arquitectura de software es el resultado de un conjunto de decisiones que un arquitecto toma para balancear los intereses de los stakeholders (interesados). Son un conjunto de vistas que contienen modelos, sin embargo resulta que estos, por regla general, no suelen documentarse y justificarse adecuadamente lo que causaría conflictos comunicativos entre los colegas y clientes al no poder identificar las razones del porque el modelo se desarrolló, como se desarrolló. Por lo tanto es menester recuperar las decisiones y justificaciones realizadas.

1.1. JUSTIFICACIÓN

Las empresas que ofrecen servicios orientados a tecnologías (específicamente las empresas generadoras de software), tienen que lidiar con necesidades varias del cliente, como son las aplicaciones móviles que representan amplias oportunidades para el desarrollo de soluciones corporativas (PROEXPORT, 2009). Por lo tanto la agilización en la producción es un detalle del cual no se debe obviar.

Las organizaciones son sistemas de actividades conscientes coordinadas, es decir son sistemas cooperativos, no son productos mecánicos del trabajo de ingenieros de eficiencia (Barnard, 1938). En estas circunstancias la comunicación juega un papel determinante. En el caso de las empresas diseñadoras de software la comunicación no solo se da verbalmente, los modelos que se generan dentro de todo el proceso de desarrollo que se sigue, desde que se conoce el cliente hasta que se entrega el producto terminado y posteriormente se mantiene, son un canal comunicativo importante, ya que estos representan gráficamente los requisitos del usuario y son el producto de una actividad de toma de decisiones.

Teniendo una arquitectura de software bien documentada y justificada las empresas desarrolladoras de software reducirían sus tiempos estimados de acción, ya que no existiría el inconveniente de recapitular lo decidido cuando se pensó o realizó la arquitectura.

1.2. ALCANCES

Desarrollar un prototipo funcional el cual se validará con un equipo de arquitectos de un proyecto.

1.3. LÍMITES

El proceso de aplicar perspectiva de seguridad en arquitectura propuesto por Rozanski.

1.4. OBJETIVOS

1.4.1. Generales

- Desarrollar una estrategia para capturar, recuperar y reutilizar las decisiones que toman los equipos de arquitectura en el diseño de dicha arquitectura.

1.4.2. Específicos

- Diseñar una ontología para representar decisiones de arquitectura.
- Construir una wiki semántica para recuperar y reutilizar las decisiones de arquitectura.
- Aplicar la wiki con el caso de estudio: Sistema de alerta y prevención de tsunamis.
- Validar la wiki con un conjunto de arquitectos.

2. MARCO TEÓRICO

2.1. ARQUITECTURA DE SOFTWARE

Según el estándar IEEE 1471[1], una arquitectura de software es una organización fundamental de un sistema de software formado por sus componentes, relaciones entre ellos y su entorno y unos principios que rigen su diseño y evolución. Sin embargo para desarrollar una arquitectura de software es necesario tomar en cuenta las decisiones tomadas y el proceso que se llevó a cabo para llegar a ellas. Es en este contexto entonces que se considerará la definición dada por RUP (Rational Unified Process)[2] que dice que una arquitectura de software es un conjunto de decisiones de diseño acerca de la organización de un sistema de software. Una arquitectura de software también incluye la funcionabilidad, el rendimiento, la reutilización, la inteligibilidad, las condiciones económicas y tecnológicas e intereses estéticos[3].

Existen dos formas básicas de representar arquitecturas de software:

1. Usando vistas[4], representaciones gráficas bajo un estándar dado. Uno de los estándares difundidos y utilizados en el desarrollo de las vistas el IEEE 1471[5].
2. Usando un lenguaje de descripción de arquitecturas.

Las arquitecturas no son creadas en un proceso solitario, es un trabajo colaborativo y dinámico entre los stakeholders y los arquitectos, en donde los requerimientos asumen un papel no trivial en la construcción del conocimiento de la arquitectura[6], siendo esto último lo documentable, lo que se necesita conocer y entender.

---

[1] IEEE. 1471-2000 - IEEE Recommended Practice for Architectural Description for Software-Intensive Systems [en línea].<http://standards.ieee.org/findstds/standard/1471-2000.html> (citado el 8 de octubre del 2014). Citado por: KRUCHTEN Philippe. An Ontology of Architectual Desig Decisions in software-Intensive Systems. p.1.

[2] IBM. IBM.Rational Unified Process: Best Practices for Software Development Teams [en línea]. <https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf>. (citado el 8 de octubre del 2014).
Citado por: KRUCHTEN Philippe. An Ontology of Architectual Desig Decisions in software-Intensive Systems. p.1.

[3] KRUCHTEN Philippe. An Ontology of Architectual Desig Decisions in software-Intensive Systems. 1 p.

[4] Ibid 3, p.1.

[5] Ibid 1, p.1.

[6] VLIET Hans, BOER Remco. Experiences with Semantic Wikis for Architectural Knowlege Management. 1 p.

Como se considerar una arquitectura de software como un conjunto de decisiones acerca de la organización de un sistema de software, estas son las constituyentes primordiales del conocimiento de dicha arquitectura, y por tanto es requerido saber que es una decisión de diseño de arquitectura, que tipos de decisiones podemos encontrar y que consideraciones debemos tener en cuenta al momento de tomar una decisión.

2.2. DECISIONES DE DISEÑO DE ARQUITECTURA

Una decisión de diseño de arquitectura es una descripción de adiciones, sustracciones y modificaciones a una arquitectura de software, lo que lleva a un proceso de racionalización e identificación de reglas y restricciones de diseño y los requerimientos adicionales[7].

La racionalización hace referencia a las razones que están detrás de una decisión de diseño de arquitectura, estas explican porque se ha hecho un cambio a una arquitectura de software. Las reglas de diseño describen que está permitido realizarse cuando se esté desarrollando el diseño. Las restricciones son las contrarias a las reglas, estas nos dicen que no podemos realizar en un futuro, es decir prohíben ciertos comportamientos. Por último cuando se esté en proceso de toma de decisiones, usualmente se presentan requerimientos adicionales que deben ser satisfechos y añadidos[8].

Se supondrá que un arquitecto entra en la nómina de una empresa, con la labor de continuar el proceso de diseño de una arquitectura de software para un sistema contable. En estas circunstancias deberá familiarizarse con el trabajo que su par dejó. Es muy común que suceda que el nuevo arquitecto vea, con cierta dificultad e incluso incredulidad, las razones de porqué se decidieron en construir un API intermedio para el control del proceso de transacciones entre el cliente y la empresa (siendo esto una suposición) cuando podría llegar a ser más económico y con mayor soporte la compra a una casa generadora. Resulta que las decisiones tomadas son circunstanciales, es decir, únicamente tienen sentido en las condiciones en que se toman[9].

Sin embargo no todos los diseñadores ven a la arquitectura de software como un conjunto de decisiones que se deben tomar para satisfacer las necesidades de los stakeholders, como se puede notar en el estándar IEEE 1471[10]. En este caso la arquitectura es vista, más bien como un producto y cuando esto ocurre, y no se le

---

[7] BOSCH Jan, JANSEN Anton. Software Architecture as a Set of Architectural Design Decisions. p.2.
[8] Ibid 6, p.2
[9] TYREE Jeff, AKERMAN Art. Architecture Decisions: Demystifying Architecture. 20 p
[10] KRUCHTEN, Op.Cit., p.1

otorga la suficiente importancia a la fundamentación del conjunto de decisiones tomadas en el proceso de desarrollo de dicho producto, se pueden presentar problemas como los que Bosch identificó:

1. Violación de las decisiones tomadas en etapas previas en el proceso de desarrollo. En la etapa de evolución del software (especialmente en esta), suelen violarse reglas y restricciones de diseño que se consideraron cuando se seguía la etapa de desarrollo. La violación de estas reglas y restricciones acarrea problemas (costo de mantenimiento), asumiendo el hecho de que tanto las reglas como las restricciones de diseño especifican lineamientos, tanto como para el presente como para el futuro del proceso de diseño.
2. Algunas decisiones obsoletas pueden no ser removidas. Esto puede causar que el sistema se "corroa" más rápidamente.
3. Toma de diversas decisiones "excesivamente relacionadas". En el proceso de diseño de decisiones pueden existir algunas que presenten una relación íntima. Pero cuando la relación se lleva fuera de "proporciones normales" se presentarán situaciones en donde fuere difícil encontrar cambios en el diseño[11].

Los tipos de decisiones de diseño pueden variar de autor en autor, no existe una aceptación generalizada para cada una de ellas. Pero en orden de entender porque es necesario conocerlas se considerará las identificadas por Bosh:

1. Decisiones de existencia (ontocríticas). Son decisiones de estructura y de comportamiento. La primera lidera la creación de subsistemas, capas, particiones y componentes en una vista de la arquitectura. Las decisiones de comportamiento son las relacionadas a la forma en como los elementos de la arquitectura interactúan entre sí, para proveer una funcionabilidad que satisfaga un requerimiento funcional y se limiten según los requerimientos no funcionales. Por ejemplo:
   - La vista lógica está organizada en tres capas: Capa de Datos, Capa de inteligencia de Negocios y la Capa de Interfaz de Usuario.
   - La comunicación entre las clases usa RMI (Remote Method Invocation).

   Este tipo de decisiones no son tan importantes de capturar ya que ellas están usualmente visibles en el diseño o implementación del sistema, y las razones por las cuales se tomaron estas decisiones se encontrarán descritas en la documentación desarrollada.

2. Decisiones de Propiedad (diacríticas). Son decisiones relacionadas a las reglas de diseño y restricciones de diseño. Las decisiones de propiedad son difíciles de encontrar, ya que estas suelen afectar a muchos elementos del sistema y otras pueden ser implícitas siendo estas olvidadas fácilmente.
3. Decisiones ejecutivas (pericríticas). Son decisiones relacionadas con el ambiente de negocio y afecta el proceso de desarrollo (metodologías), las

[11] BOSCH, JANSEN, Op.Cit, p.2.

personas (educación y entrenamiento), la organización y a una larga extensión de elecciones de tecnologías y herramientas. Por ejemplo:

- Todos los cambios en subsistemas deben ser aprobados por el equipo de arquitectura.
- El sistema es desarrollado usando J2EE.
- El sistema es desarrollado en Java.
- El sistema es desarrollado usando Eclipse.

Este tipo de decisiones están íntimamente relacionadas con todos los aspectos políticos, personales, culturales, financieros y tecnológicos que generan grandes cantidades de restricciones y, todas las decisiones asociadas difícilmente (por regla general) se capturan o documentan[12].

Es razonable o lógico pensar que cuanto más grande y complejo sea el sistema informático a construir, un mayor número de decisiones se deberán tomar, y cuando mayor sea el número de restricciones y condiciones que se tengan, más *críticas* se vuelven estas decisiones. Es aquí en donde radica la importancia de la buena captura y comprensión de estas para evitar problemas futuros de mantenimiento y escala. Estas decisiones deben ser socializadas no solo con el equipo de desarrollo o entre los arquitectos, sino que también entre los stakeholders. Estos últimos son los más importantes en vista de que son los inversionistas del trabajo desarrollado. Entre los métodos más utilizados para la exposición o socialización de las decisiones tomadas, pero no implementadas aun, se puede nombrar la clásica documentación y las presentaciones en Microsoft Power Point, sin embargo trabajos como los de Kruchten o los de Tyree y Akerman, exponen, entre otras cosas, las importantes ventajas que las ontologías pueden llegar a tener en la administración del conocimiento de las arquitecturas, aunque las webs semánticas, por su naturaleza, puede proveer de un entorno apropiado para la captura, socialización, recuperación y por tanto comprensión de los conocimientos que podamos obtener de una arquitectura de software determinada.

[12] KRUCHTEN, Op.Cit., p.2.

2.3. WEB SEMÁNTICA

Según la W3C la web semántica es una web extendida, dotada de mayor significado en la que cualquier usuario en el internet podría encontrar respuestas a sus preguntas de forma más sencilla gracias a una información mejor definida. Al dotar la web de mayor significado se pueden obtener soluciones a inconvenientes habituales en la búsqueda de información gracias a la utilización de una infraestructura común, resolviendo problemas ocasionados por una web carente de semántica que, en ocasiones, el acceso a la información se convierte en una tarea difícil y frustrante[13].

La web semántica busca dotar de estructura y anotar los recursos con semántica explícita procesable por las máquinas, es decir, darle mayor autonomía de decisión a las máquinas intermediarias. Por ejemplo un software puede aplicar ciertos algoritmos de decisión basados en modelos estadísticos para definir la página destino en un direccionamiento, pero los equipos son "ignorantes" acerca de lo que pasa, las aplicaciones son las que se encargan de ello y estas pueden diferir entre sí.

Al extenderse más allá de las webs semánticas, otorgando más libertad en edición y refinamiento de contenido, estas webs extendidas se conocerán como wiki semánticas[14]. Esta wiki se diferencia de una wiki tradicional en el sentido que la primera posee todas las bondades descritas asociadas a una web semántica, como lo son la descripción del contenido web y la existencia de significado en este para generar una relación explícita entre sus componentes hipervinculados (páginas webs - vistas- conectadas por hipervínculos).

Una wiki es un sistema que se basa en el contenido web (puede llegar a ser visto como un sitio web) que basa su atractivo en la actualización y modificación de su contenido (conocimiento explícito), en texto y multimedia, de una manera fácil y flexible. En este contexto no difiere mucho, en cuanto a propiedades se trata, la web tradicional, más que es la característica clave ya expuesta. La wiki semántica ofrece una forma más sencilla de incluir anotaciones semánticas a conceptos ontológicos reflejados como cada uno de los artículos que se encuentran en ella, así en un artículo de la wiki semántica estas anotaciones, que toman la forma de hipervínculos, nos permiten navegar entre artículos (conceptos ontológicos) relacionados con el que actualmente se está observando.

En este contexto la idea de la web semántica es que exista una red de nodos tipificados e interconectados mediante clases, y relaciones definidas por una

---

[13] W3C. Guía Breve de Web Semántica [en línea].
<http://www.w3c.es/Divulgacion/GuiasBreves/WebSemantica> (citado el 10 de octubre del 2014).

[14] REUTELSHOFER Jochen, BAUMEISTER Joachim, PUPPE Frank. KnowWE: A Semantic Wiki for Knowlede Engineering. 3 p.

ontología compartida por sus distintos autores. Por ejemplo una vez creada una ontología sobre cuadros y pinturas, un museo virtual puede organizar sus contenidos definiendo instancias de pintores, cuadros, etcétera, interrelacionándolas y publicándolas en la web semántica.

2.4. ONTOLOGÍAS

En el sentido filosófico una ontología es un sistema particular de categorías para una cierta visión del mundo. Por otro lado, en el uso más prevaleciente en el campo de la inteligencia artificial, una ontología hace referencia a un "constructo ingenieril", constituido por un vocabulario específico usado para describir cierta realidad, además de un conjunto de especulaciones independientes del significado de las palabras que componen dicho vocabulario[15]. No obstante dentro del área de las ciencias computacionales una ontología representa el esfuerzo por formular un esquema conceptual formal y muy riguroso de un dominio dado, en general puede ser una estructura de datos jerarquizada que contenga todos los elementos relevantes, junto con las relaciones y las reglas que los rigen[16] En este orden de ideas una ontología puede proveer de un lenguaje común para favorecer el la comprensión de las ideas generadas en procesos de análisis, procesos que se ven bastante arraigados a las etapas de un ciclo de vida un software (levantamiento, diseño, implementación, pruebas y mantenimiento).

Aunque las ontologías ven sus inicios en la filosofía, ciertamente uno de sus principales beneficios es la interrelación de conceptos o ideas que diversos autores han sabido explotar. Autores como Bench-Capon proponen la utilización de ontologías para corroborar la coherencia de los conocimientos, proveer un medio para estructurar pruebas y para sugerir propuestas de acción cuando se presente una falla[17]. Por regla general (e inclusive sentido común) para mantener un software es necesario tener una comprensión profunda del mismo, no solo a nivel funcional sino también a un nivel interactivo, como cada uno de los componentes que

[15] GUARINO N. Formal Ontology in Information Systems. 4p.

[16] Wongthongtham Pornpit, KHADEER Farookh, CHANG Elizabeth, DILLON Tharam. Multi-site Software Engineering Ontology Instantiations Management Using Reputation Based Decision Making. 2 p.

[17] T. J. M. Bench-Capon, "The Role of Ontologies in the Verification and Validation of Knowledge Based Systems," en 9th International Workshop on Database and Expert Systems Applications (DEXA), 1998. Citado por: ZAPATA Carlos, GIRALDO Gloria, URREGO Germán. Las ontologías en la ingeniería de *software*: un acercamiento de dos grandes áreas del conocimiento. En: Rev. ing. univ. Medellín. No.16 (Jun,, 2010). ISSN 1692-3324.

componen el software se integran para satisfacer los requerimientos establecidos. Inclusivamente en la etapa de levantamiento de requerimientos se pueden presentar vacíos comunicativos entre los interesados (stakeholders) y los analistas, Benaroch en este contexto plantea un método para capturar los requerimientos e incluye en él una ontología local que luego se traducirá en una base de datos relacional[18].

Otros trabajos como AutoBayes, una aplicación Open Source (código abierto) de la NASA que permite la "síntesis" de algoritmos personalizados a partir de unas especificaciones declarativas y compactas en el dominio del análisis, utilizan algoritmos basados en modelos estadísticos y de predicción de estados que resultaron ser difíciles de mantener por su alta complejidad. Sin embargo los desarrolladores de esta aplicación están explorando la posibilidad de usar ontologías, ya que estos creen que facilitará la extensión del dominio de análisis, la escritura de los esquemas, la validación en salida de estos y la generación de artefactos adicionales, entre otras ventajas.

2.5. SISTEMA DE HIPÓTESIS

2.5.1. Hipótesis de trabajo

La utilización de una wiki semántica con enfoque en administración de conocimiento para la captura, representación, divulgación y recuperación de arquitecturas de software ayudará, significativamente, a la comprensión las decisiones tomadas para el diseño de dicha arquitectura.

[18] M. Benaroch, "Specifying Local Ontologies in Support of Semantic Interoperability of Distributed Inter-organizational Applications," en Proc. of the 5th Intl. Workshop on Next Generation Inf. Techn. and Systems, Caesarea, Israel, 2002, pp. 90-106.
Citado por: ZAPATA Carlos, GIRALDO Gloria, URREGO Germán. Las ontologías en la ingeniería de *software*: un acercamiento de dos grandes áreas del conocimiento. En: Rev. ing. univ. Medellín. No.16 (Jun, 2010). ISSN 1692-3324.

2.5.2. Hipótesis nula

La utilización de una wiki semántica con enfoque en administración de conocimiento para la captura, representación, divulgación y recuperación de arquitecturas de software no ayudará, significativamente, a la comprensión las decisiones tomadas para el diseño de dicha arquitectura.

2.6. SISTEMA DE VARIABLES

2.6.1. Variables independientes

- Utilización de una wiki semántica (tecnología).
- Modelo de decisiones.

2.6.2. Variables dependientes

- La compresión de las decisiones.

2.6.3. Variables intervinientes

- Enfoque de administración de conocimiento (base metodológica).
- Enfoque de diseño de arquitectura.

3. DISEÑO METODOLÓGICO DE LA INVESTIGACIÓN

3.1. TÉCNICAS DE LA RECOLECCIÓN DE LA INFORMACIÓN

Se realizará una entrevista explicativa en donde se evaluará el nivel de comprensión de los conocimientos de la arquitectura software (eficacia), la productividad en la comprensión de la arquitectura y facilidad de uso del prototipo (eficiencia), la cual se aplicará a diferentes grupos de arquitectos de software, quienes desde su experiencia previa con otros enfoques de administración de conocimiento calificarán el prototipo según las categorías de interés expuestas anteriormente. Por lo tanto es necesario que el prototipo se haya utilizado previamente para poder aplicarla.

3.2. METODOLOGÍA DE DESARROLLO

Para la construcción del prototipo de wiki semántica se seguirá el modelo de desarrollo en espiral definido por Barry Boehm[19], con ciertas modificaciones en las actividades establecidas en cada uno de los ciclos del modelo original. No se levantará, ni se validarán requerimientos con los stakeholders porque ya están definidos en nuestro problema. Las posteriores actividades propuestas en el modelo original se seguirán normalmente, aunque se pueden omitir algunas tareas.

Se optó por seguir este modelo por su enfoque en el control de riesgos a lo largo del ciclo de vida del software al igual que su metodología de desarrollo evolutivo, donde podemos reducir o ampliar el número de ciclos de trabajo según se requiera. Lo anterior unido a una flexibilidad inherente en la determinación de cada una de las fases a desarrollar a lo largo de cada uno de las iteraciones, convierte la metodología de desarrollo en espiral en una decisión adecuada.

El desarrollo de la wiki semántica se basará en las tareas y actividades expuestas en la tabla uno.

[19] BOEHM Barry. A Spiral Model of Software Development and Enhancement.

Actividades	Tareas	Resultados
Diseño de la ontología.	Definición del dominio de la ontología	Modelo conceptual del dominio
	Definición de las clases o entidades.	Documento de diseño de la ontología.
	Definición de los individuos.	
	Definición de las propiedades.	
	Definición de las relaciones.	
Aprendisaje de las herramientas tecnológicas.	Aprendisaje de la herramienta Protégé.	Ninguno.
	Aprendisaje de SPARQL.	
	Aprendisaje de la herramienta Proty.	
Implementación de la ontologia en Protégé	Creación del proyecto.	Archivo OWL que representa la ontología (prototipo de la ontología en Protégé).
	Creación de las clases y subclases.	
	Creación y asociación de las propiedades.	
	Creación de los objetos o relaciones.	
	Creación de las instancias o individuos.	
Pruebas a la ontología desarrollada	Creación de consultas SPARQL a la ontología	Permiso para continuar el proceso.
Construir la wiki semántica	Definir la estructura de la wiki semántica.	Documento de diseño de la wiki semántica (plantillas, formularios, categorías).
	Integrar los wireframes con la ontología.	Una wiki semántica funcional (prototipo).
Pruebas a la wiki desarrollada	Diseño del plan de pruebas.	Documento del plan de pruebas.
	Verificación de la navegación entre páginas	Aprobación para continuar.
	Verificación de modificación de contenido textual	
	Verificación de carga apropiada de imágenes	
	Verificación del correcto funcionamiento de los formularios.	
	Verificación de correcta carga de anotaciones	
Diseño del caso de estudio.	Realización del documento del caso de estudio.	Documento del caso de estudio.
Implementación del caso de estudio	Modificar el contenido de la wiki semántica con lo requerido por el caso de estudio en cuestión.	Una wiki semántica adaptada al caso de uso específico.
Pruebas a la wiki semántica adaptada	Verificación de la navegación entre páginas.	Aprobación para continuar.
	Verificación de modificación de contenido textual.	
	Verificación de carga apropiada de imágenes.	
	Verificación de correcta carga de anotaciones.	
Diseño del experimento de validación.	Diseño del experimento.	Guía paso a paso del experimento.
	Diseño de la encuesta.	Encuesta de percepción.
Validación con los arquitectos	Encontrar arquitectos que deseen evaluar.	Hipótesis probada o no, posible replantamiento de la estrategia y modificación de la wiki.
	Explicación del caso de estudio.	
	Aplicar la encuesta.	
	Analizar los resultados.	

Tabla 1. Modelo de desarrollo pensado para el desarrollo de la wiki semántica.

4. PLAN ADMINISTRATIVO DE LA INVESTIGACIÓN

4.1. RECURSOS HUMANOS

Los precios mostrados en la tabla dos se basan en las actividades necesarias para el desarrollo de un prototipo de wiki semántica general, adaptable a cualquier propósito de administración del conocimiento de arquitectura de software, es decir, sin la aplicación y la validación del caso de un caso de estudio específico.

Actividad	Precio	Precio acumulado
Diseño de la ontología	700000	500000
Implementación de la ontología	850000	1350000
Pruebas a la ontología en Protégé	400000	1750000
Construcción de la wiki semántica	900000	2650000
Pruebas a la wiki desarrollada	600000	3250000
	Costo total	$3250000

Tabla 2. Tabla de cotización por actividad de los involucrados.

4.2. RECURSOS TECNOLÓGICOS

Los precios mostrados en la tabla tres representan los costos necesarios para suplir necesidades tecnológicas, herramientas necesarias para el desarrollo de la wiki semántica.

Herramientas a usar	Valor	Valor acumulado
Computadores X 2	192000	192000
Internet	90000	282000
	Costo total	$282000

Tabla 3. Tabla de cotización por recursos tecnológicos.

4.3. ANÁLISIS COSTO-BENEFICIO

La mayor parte del costo invertido en la creación de un software se ve en el mantenimiento del mismo. Mantener un software conlleva un trabajo arduo de comprensión, en especial si las personas que lo necesitan no estuvieron involucradas en el proceso inicial de diseño y desarrollo. En estas circunstancias es claro que el tiempo invertido necesario para empezar un trabajo de mantenimiento puede ser bastante amplio, aún más si se trata de proyectos de gran envergadura. Por eso el uso de la wiki semántica representaría una inversión a largo plazo que reduciría el tiempo necesario para comenzar trabajos de mantenimiento, posiblemente en un factor de 2 a 3 (reduce el tiempo a la mitad hasta un tercio del mismo). Lo anterior representaría una reducción proporcional de los costos necesarios posteriores a la salida del software.

4.4. CRONOGRAMA DE TRABAJO

Semana	Actividad	Tiempo a invertir (horas)
01.12.14 - 07.12.14	Diseño de la ontología	12
	Apredizaje de las herramientas tecnológicas	8
08.12.14 - 14.12.14	Aprendizaje de las herramietas tecnológicas	12
	Implementación de la ontología en Protégé	8
15.12.14 - 21.12.14	Pruebas a la ontología desarrollada	8
	Construir la wiki semántica	12
22.12.14 - 28.12.14	Construir la wiki semántica	20
29.12.14 - 04.01.15	Construir la wiki semántica	8
	Pruebas a la wiki desarrollada	8
	Diseño del caso de estudio	4
05.01.15 - 11.01.15	Diseño del caso de estudio	6
	Implementación del caso de estudio	14
12.01.15 - 18.01.15	Implementación del caso de estudio	8
	Pruebas a la wiki semántica adaptada	8
	Diseño del experimento de validación	4
19.01.15 - 25.01.15	Diseño del experimento de validación	4
	Validación con los arquitectos	16
26.01.14 - 01.02.14	Validación con los arquitectos	20
02.01.14 - 08.02.14	Validación con los arquitectos	20

Tabla 4, Cronograma de trabajo por actividades.

El cronograma de trabajo mostrado en la tabla cuatro se basa en las actividades identificadas y expuestas en la tabla 1, sobre el cual se estima lo siguiente:

- Un total de 200 horas para trabajar.
- Un promedio de 20 horas laborales por semana distribuidas en 4 horas diarias durante 5 días.
- Un estimado de 10 semanas de trabajo que se planean iniciar el primero de diciembre y terminar la segunda semana de febrero.

BIBLIOGRAFÍA

BOEHM Barry. A Spiral Model of Software Development and Enhancement.

BOSCH Jan, JANSEN Anton. Software Architecture as a Set of Architectural Design Decisions.

GUARINO N. Formal Ontology in Information Systems.

IBM. IBM.Rational Unified Process: Best Practices for Software Development Teams [en línea]. <https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf>

IEEE. 1471-2000 - IEEE Recommended Practice for Architectural Description for Software-Intensive Systems [en línea].<http://standards.ieee.org/findstds/standard/1471-2000.html>.

KRUCHTEN Philippe. An Ontology of Architectual Desig Decisions in software-Intensive Systems.

REUTELSHOFER Jochen, BAUMEISTER Joachim, PUPPE Frank. KnowWE: A Semantic Wiki for Knowlede Engineering.

TYREE Jeff, AKERMAN Art. Architecture Decisions: Demystifying Architecture.

VLIET Hans, BOER Remco. Experiences with Semantic Wikis for Architectural Knowlege Management.

W3C. Guía Breve de Web Semántica [en línea] <http://www.w3c.es/Divulgacion/GuiasBreves/WebSemantica>.

Wongthongtham Pornpit, KHADEER Farookh, CHANG Elizabeth, DILLON Tharam. Multi-site Software Engineering Ontology Instantiations Management Using Reputation Based Decision Making.

ZAPATA Carlos, GIRALDO Gloria, URREGO Germán. Las ontologías en la ingeniería de *software*: un acercamiento de dos grandes áreas del conocimiento. En: Rev. ing. univ. Medellín. No.16 (Jun, 2010). ISSN 1692-3324.