INSTITUTO TECNOLÓGICO DE SONORA
Educar para Trascender

“Soporte para la Localización del Expertise en el Desarrollo de Software mediante una plataforma basada en Agentes Inteligentes”

Tesis
que para obtener el título de
Maestro en Ciencias de la Ingeniería

Presenta

José Ramón Martínez García

Cd. Obregón, Sonora; mes de año

CAPÍTULO I

INTRODUCCIÓN

1.1 Antecedentes

En la actualidad las organizaciones tienen un especial interés por tratar el conocimiento como un recurso organizacional. Este tipo de recurso ha provocado dentro de las organizaciones un cambio en la manera de administrar la información que les generan sus diferentes procesos y con el uso de dicha información, las organizaciones fomentan el intercambio de conocimiento entre sus miembros, para que a partir del conocimiento base con el que cuentan, éste se incremente y perfeccione sus practicas de trabajo para lograr la creación del conocimiento organizacional. Para esto, las organizaciones tienen que fomentar el apoyo interno entre los miembros para generar y transferir conocimiento que posteriormente reditué en mejores tomas de decisiones. Lo anterior es posible mediante el apoyo de sus sistemas de información, pues permiten que el capital intelectual aumente de manera significativa con una buena gestión de sus capacidades para la resolución de problemas (P&MS, 2000).

Un nuevo conocimiento siempre tiene su origen en un individuo de la organización, de manera que ese conocimiento se transforma de individual a organizacional. Es decir la organización no puede crear conocimiento sin la iniciativa del individuo y la interacción que se da dentro de los equipos de trabajo. El conocimiento puede aumentar o consolidarse en el grupo a través del dialogo, la discusión, el intercambio de experiencias y la observación. Por lo que los miembros de una organización generan nuevos puntos de vista a través del diálogo y la discusión. Este diálogo puede abarcar conflictos y desacuerdos considerables, pero es precisamente tal conflicto lo que presiona a los empleados a cuestionar las premisas existentes y a dar un nuevo sentido a sus experiencias (Nonaka & Takeuchi, 1995).

Cabe resaltar que este trabajo se va enfocar particularmente en organizaciones de desarrollo de software, considerando que el desarrollo de software es un proceso meramente intelectual y complejo donde todos los productos son representaciones del conocimiento de todos los involucrados para generar una aplicación computacional que debe solucionar una necesidad planteada o identificada por un cliente. Para esto se requiere la interacción constante entre los stakeholders, pues se caracteriza por ser un proceso de constante cambio, donde muchas personas trabajan en diferentes fases, actividades y proyectos (Pressman, 2009; Schwaber, 2004).

En el desarrollo de software, una gran cantidad de conocimiento puede ser compartido entre los miembros de desarrollo ya sea entre analistas y programadores o entre programadores e ingenieros de pruebas, por lo que mucho de este conocimiento permanece tácito, ya que depende mucho de la interacción entre los miembros del equipo de trabajo para poder llegar a obtener o compartir dicho conocimiento. También, el desarrollo de software es llevado a cabo mediante equipos de trabajo virtuales que es conocido como el Desarrollo Global de Software (Barceló, Rubio, & Velthuis, 2014; Prikladnicki, Damian, & Audy, 2008), donde los miembros de un equipo están distribuidos en diferentes ubicaciones geográficas del mundo, lo cual acentúa de manera constante

malentendidos (Herbsleb & Moitra, 2001; R. E. Jensen, 2012), errores (Basili & Perricone, 1984; Westland, 2002) y el desperdicio de recursos (Basili & Caldiera, 1995) (económicos y/o humanos) al no reutilizar el conocimiento con el que ya se cuenta (Van Vliet, 2010). Dicho conocimiento según (Becerra-Fernandez & Sabherwal, 2010) puede encontrarse en:

- *Artefactos:* éste se refiere al conocimiento que se encuentra en la documentación por ejemplo el libro de requerimientos, documento de visión, base de código fuente y manual de usuario (Jacobson, Booch, Rumbaugh, Rumbaugh, & Booch, 1999) obtenido a partir de las prácticas de la empresa, los repositorios donde los desarrolladores consultan documentos físicos o digitales para completar sus actividades diarias por ejemplo en los blogs (Vasilescu, Filkov, & Serebrenik, 2013; Vasilescu, Serebrenik, Devanbu, & Filkov, 2014), manuales (Dagenais & Robillard, 2010; Dekel & Herbsleb, 2009), marcadores (Cockburn & McKenzie, 2001; W. Jones, Dumais, & Bruce, 2002), control de versiones (Loeliger & McCullough, 2012; Pilato, Collins-Sussman, & Fitzpatrick, 2008) y en las herramientas para administrar conocimiento de los procesos, etapas y actividades de los proyectos (Colmenar Santos et al., 2007; Sarkan, Ahmad, & Bakar, 2011).
- *Personas:* se refiere al conocimiento que se encuentra en los desarrolladores, obtenido de la experiencia de trabajar en los proyectos de software (asesoría, consultas con otros desarrolladores, cursos de capacitación, así como también de la búsqueda individual en sitios web, libros y manuales) (Basili, Caldiera, & Rombach, 1994; Johansson, Hall, & Coquard, 2000).

A partir de estas fuentes se puede producir riqueza, multiplicar la producción de bienes físicos, y crear ventajas competitivas utilizando este conocimiento generado por todos los miembros de la organización (Bontis, 2001; Fragouli, 2015).

En muchas de las situaciones cuando se tiene alguna dificultad para resolver una actividad se suele buscar el conocimiento, pues la meta es encontrar e*l expertise* (conocimiendo de mejor calidad) demandando un grado mayor de conocimiento, de tal manera que quien posee experiencia es capaz de realizar una tarea mucho mejor que el que no la posee (Ericsson, Prietula, & Cokely, 2007). Esto es un problema debido a que en la industria del software los productos generados durante el proceso de producción se generan diferentes artefactos (ej. requerimientos del sistema, módulos, componentes del software, manuales, etc.) y dichos artefactos están conectados o relacionados con su/s creador/es (programadores, arquitectos de software, analistas, etc.) (Pressman, 2009). Para esto, ellos requieren del *expertise* existente en la organización de tal forma que recurren a sus colegas (expertos que son las personas que poseen altos niveles de habilidades o conocimientos) para intercambiar conocimientos con la finalidad de resolver obstáculos que impiden que se logre alcanzar en primera instancia el artefacto individual y en segunda el artefacto grupal. Por lo que el reto para este tipo de organizaciones es la obtención de *expertise* adecuada de manera oportuna con la finalidad de mantener el nivel de competencia de la empresa para de ganar más contratos y cumplir sus compromisos a tiempo con sus clientes, y para esto la organización requiere que sus miembros sean efectivos con la generación de sus artefactos (Rus & Lindvall, 2002).

Generalmente las organizaciones buscan constantemente reducir el tiempo y los costos de desarrollo de proyectos de software, para lograr esto se necesita aplicar los conocimiento adquiridos en proyectos anteriores y/o en los recursos utilizados por los miembros de la organización. Esto es un problema debido a que por desagracia los equipos de desarrollo no se benefician de este conocimiento existente y se repiten errores de proyectos anteriores a pesar de que algunos personas en la organización saben como evitarlos (Kautz & Thaysen, 2001; Nielsen & Kautz, 2008; Rus & Lindvall, 2002). Constantemente los desarrolladores deben resolver problemas todos los días para poder realizar su trabajo, para esto, ellos necesitan encontrar los recursos o las personas con experiencia especifica para resolver algún problema, además cada persona involucrada debe tomar

decisiones técnicas o de gestión la mayor parte del tiempo basadas en el conocimiento personal y/o experiencia (generalmente conocimiento adquirido a través de los proyectos). Esto es factible en las organizaciones pequeñas, pero a medida que las organizaciones crecen y manejan un volumen mayor de información, este proceso se convierte en ineficiente. Las grandes organizaciones no pueden confiar en el intercambio informal de conocimiento personal de los empleados.

La localización del *expertise* en el desarrollo de software es un tema complejo y poco abordado pues existe conocimiento que se encuentra tácito en las personas y para acceder a dicho conocimiento generalmente los trabajadores del desarrollo de software interactúan entre ellos para enterarse quién es la persona más adecuada para resolver alguna duda o tarea del proyecto. Para esto se han realizado esfuerzos para gestionar la localización del expertise mediante el paradigma orientado a agentes, como en (Rodríguez, Vizcaíno, Martínez, Piattini, & Favela, 2004) donde se presenta arquitectura multi-agente diseñada para gestionar la información y el conocimiento que se genera durante el proceso de mantenimiento de software, utilizando tecnologías web para apoyar la interacción entre los agentes utilizando diferentes técnicas de razonamiento para generar nuevo conocimiento a partir de información previa y de aprendizaje de su propia experiencia. Dicho razonamiento esta basado en los distintos casos que sucedieron en algún proyecto anterior o previamente en el mismo, asi como tambien en la experiencia documentada por los stakeholders. Otra propuesta que trabaja con el paradigma orientado a agentes es (Vivacqua, 1999) que presenta una implementación inicial de un sistema que trabaja con agentes que ayudan a la localización del *expertise* (expertos) teniendo como tema de dominio el lenguaje de programación en java, donde los agentes se encargan de programar citas para el intercambio de conocimiento, detectar proactivamente cuando se necesita ayuda, proporcionar información adicional durante la interacción y ajustar sus propios mecanismos de perfiles de acuerdo a los comentarios de los usuarios.

También se han desarrollado sistemas que se enfocan a un tipo de conocimiento en particular (artefactos) como BluePrint (Brandt, Dontcheva, Weskamp, & Klemmer, 2010), el cual describe el diseño, implementación y evaluación de una interfaz de búsqueda web integrada al entorno de desarrollo Adobe Flex Builder que ayuda a los usuarios a localizar ejemplos de código de proyectos anteriores utilizando palabras clave (e.g., lenguaje de programación, framework, nombre de la clase y/o método). SNIFF (Chatterjee, Juvekar, & Sen, 2009), trabaja con la localización de artefactos, el cual facilita la búsqueda de librerías existentes por medio de un plugin para el entorno de desarrollo eclipse para el lenguaje de programación java, se basa en la premisa de que las librerías están muy bien documentadas lo que facilita emparejar la búsqueda de la librería con las disponibles del dominio de programación en java. En ese mismo sentido, Exemplar (Grechanik et al., 2010) es una herramienta para la búsqueda de proyectos de software de gran relevancia para reutilizar el código fuente, para lo cual utiliza las palabras clave del proyecto y la descripción bien fundamentada, de manera que se busca hacer una coincidencia entre las palabras y que la descripción refleje las necesidades del usuario.

Finalmente, también se han encontrado trabajos que localizan conocimiento con respecto al nivel de conocimiento de las personas (expertos) como QuME (Zhang, Ackerman, Adamic, & Nam, 2007), que es un prototipo de interfaz web personalizada para los usuarios de comunidades de ayuda online de java, el cual presenta un mecanismo para inferir el nivel de conocimiento en java de los usuarios, que se calcula utilizando parámetros como las preguntas que coloca en el foro, la frecuencia de respuesta, las palabras clave de su perfil y otros aspectos más que proporcionan el grado de *expertise* de la persona. Expertise Recommender (McDonald & Ackerman, 2000) es un sistema de recomendación de expertos que utiliza una arquitectura de recomendación general que se basa en un estudio de campo de localización experiencia, el cual se ajusta de acuerdo a las necesidades del usuario y al entorno en el cual es solicitado dicho experto. Estos trabajos han permitido ubicar el nivel de conocimiento de las personas en

base a parámetros específicos y el historial de conocimientos que se han compartido en un foro especializado.

Estos trabajos muestran como los investigadores han realizado distintos esfuerzos para localizar el expertise en ambientes de desarrollo de software, estos trabajos han propuesto razonamiento para generar nuevo conocimiento a partir de un conocimiento base, tambien se han realizado plataformas de software para ubicar artefactos de los proyectos de desarrollo y también se han propuesto sistemas que permiten la ubicación de personas expertas en tópicos especificos. Sin embargo ninguno de los trabajos encontrados han logrado integrar artefactos y expertos, pues dichos trabajos se limitan a la recolección del conocimiento para utilizarlo en ese momento sin compartirlo, de manera que nadie más puede tener acceso a el o conocer quien es el proveedor, la mayoría de los trabajos citados recolectan el expertise individual de los usuarios pero no trabajan en la manera de integrarlo y compartirlo para hacerlo accesible para todos los desarrolladores en una organización, esto es importante debido a que en el desarrollo de software la experiencia es un factor importante para las entregas a tiempo como lo resaltan en (Becerra-Fernandez & Sabherwal, 2010), pues se estaria apoyando de manera importante a la reutilización de conocimiento organizacional (Ericsson et al., 2007).

Por lo anterior se proponen las siguientes preguntas de investigación:

¿Qué características de las personas o artefactos son adecuadas para identificar el expertise en el desarrollo de software?

¿Cómo puede emularse proceso para localizar el expertise en el desarrollo de software mediante una arquitectura basada en agentes?

¿Es posible que los agentes puedan emular el proceso del expertise?

1.2 Objetivo

Modelar y validar una plataforma basada en agentes, utilizando escenarios de la localización del *expertise* en el desarrollo de software con la finalidad de identificar las fuentes más apropiadas para apoyar a las actividades de los desarrolladores de software.

1. Identificar el flujo de conocimiento dentro de las organizaciones de desarrollo de software
2. Definir un formalismo para la representación del conocimiento de las organizaciones de desarrollo de software
3. Diseñar una plataforma basada en agentes para dar soporte a la localización del *expertise*.
4. Validar el proceso que proporcione el soporte de la localización del *expertise*.

1.3 Justificación

Uno de los principales retos de la localización del *expertise* son los elementos clave que podrían servir para identificar un artefacto o un experto y clasificación. Por ello, este trabajo servirá para encontrar dichos elementos o características para facilitar su identificación. Por lo que existe en un vacío notable en la literatura con respecto a las definiciones de expertos y/o artefactos dentro del desarrollo de software.

Además este trabajo proporcionara una base, para la generación de nuevos proyectos o la refinación de este modelo para facilitar la localización del *expertise*, no solamente en el desarrollo de software si no en otras empresas que tengan las características similares en cuanto a las fuentes de conocimiento. Por lo que la principal contribución de esta tesis radica en probar que el razonamiento de la plataforma propuesta es adecuado, compresible y claro para localizar expertise, el

cual podrá facilita a diseñadores de sistemas multi-agentes enfocarse al diseño conceptual y de especificaciones de sus propios sistemas.

1.4 Limitaciones

Debido a la falta de información formal de los elementos de la localización del *expertise* en la literatura, este trabajo esta basado en la información obtenida de las empresas de software de la región a través de grupos focales y entrevistas. Por otro lado la validación se basara en escenarios apegados a las actividades de los desarrolladores de software tomadas de sus experiencias.

1.5 Delimitaciones

En este trabajo no habrá una implementación del sistema por lo que solo se harán pruebas apegadas a los escenarios de la localización del expertise en el desarrollo de software.

CAPÍTULO II

MARCO TEORICO

2.1 Desarrollo de Software

El desarrollo de software es el proceso de programación de computadoras, documentación, pruebas y corrección de errores involucrados en la creación y mantenimiento de las aplicaciones y los marcos que participan en el ciclo de vida de liberación de software y que resulta en un producto de software. Donde este

es considerado un proceso meramente intelectual donde todos los productos son representaciones del conocimiento de todos los involucrados para generar una aplicación computacional que debe para solucionar una necesidad planteada o identificada por un cliente. Para esto se requiere la interacción constante entre los stakeholders, pues se caracteriza por ser un proceso de constante cambio, donde muchas personas trabajan en diferentes fases, actividades y proyectos (Pressman, 2009; Schwaber, 2004).

El resultado de un proyecto de software es un producto que se forma durante el desarrollo a partir de la intervención de los desarrolladores de un equipo de trabajo. Un proceso de desarrollo de software guía los esfuerzos de los desarrolladores implicados en el proyectos, de manera que se explican los pasos necesarios para terminar el proyecto. Este proceso esta automatizado por medio de una herramienta o de un conjunto de ellas. En el desarrollo de software intervienen variables como:

- Personas: Los principales autores de los proyectos de software son los arquitectos, desarrolladores, ingenieros de prueba, y el personal de gestión que les da soporte, también los usuarios y clientes.

- Proyecto: Elemento organizativo a través del cual se gestiona el desarrollo de software. El resultado de un proyecto es una versión del producto.

- Producto: Artefactos que se crean durante la vida del proyecto, como los modelos, código fuente, ejecutables y documentación.

- Proceso: Un proceso de ingeniería de software es una definición del conjunto completo de actividades que se necesitan para transformar los requisitos del usuario en un producto.

- Herramientas: Software que se utiliza para automatizar las actividades definidas en el proceso.

Muchas personas están involucradas en el desarrollo de un producto de software durante todo su ciclo de vida. Dichas personas lo financian el producto, lo planifican, desarrollan, prueban y lo utilizan. Por lo que este proceso debe orientarse a las personas, es decir de funcionar correctamente para las personas que lo utilizaran (Jacobson, Booch, & Rumbaugh, 2000).

2.1.1 Proceso de Desarrollo

El proceso de software es el modo para producir software, donde se incorpora una metodología con su modelo de ciclo de vida y las técnicas de las herramientas que se utilizan, así como también de las personas que están involucradas en el proceso.

Cada empresa tiene diferentes procesos de programación. Por ejemplo en el aspecto de la documentación. Algunas empresas determinan que el software que se produce es por si mismo, el documento, es decir ellos consideran que puede entenderse el producto simplemente leyendo el código fuente. Por otro lado otras empresas producen mucha documentación, redactan con mucho cuidado todo el proceso, después realizan varias actividades de diseño, revisan y vuelven a revisar los diseños antes de comenzar con la codificación, de manera que a los programadores se les proporcionan descripciones detalladas de cada artefacto de código, se plantean con anticipación los casos de prueba, se documenta el resultado de cada corrida de pruebas y se archiva. Independientemente del procedimiento exacto, el proceso de desarrollo de software se estructura en torno a los cinco flujos de trabajo: requerimientos, análisis, diseño, implementación y pruebas (Schach, Fernández, Guerrero, Ramírez, & Betancourt, 2006).

El proceso de software cambia drásticamente debido a la falta habilidades en los desarrolladores, pocos se mantienen actualizados, también se debe a que hay muchas diferencias en el proceso de software en algunos casos los gerentes de proyectos son excelentes pero saben muy poco acerca del desarrollo de software o de su mantenimiento, esa falta de conocimiento técnico puede ocasionar retrasos importantes en los calendarios de los proyectos hasta el punto de que no haya modo de continuar. Con frecuencia este motivo es por el cual muchos proyectos de software nunca se terminan.

2.1.2 Roles Desarrollo de Software

El desarrollo de software es una actividad social que comprende muchas actividades, donde se necesita un esfuerzo de equipos, se requiere la cooperación e interacción social de todos los involucrados. Las relaciones entre los equipos y los recursos son generalmente asignados y controlados por las funciones que se definen en una metodología de desarrollo de software (Yilmaz, O'Connor, & Clarke, 2015).

Basados en (Yilmaz, O'Connor, & Clarke, 2012) a continuación se describirán en las siguientes subsecciones los diferentes roles que existen según su metodología de desarrollo (desarrollo tradicional de software, ISO/IEC 12207, programación extrema, Desarrollo basado en funcionalidades (FDD), SCRUM)

2.1.2.1 Roles Desarrollo tradicional de Software

Gerente de proyecto: es el responsable de la asignación de los recursos, los gastos del proyecto y responsable de los objetivos generales de un proyecto de software.

Desarrollador de Software: Un desarrollador de software se encarga de diseñar y mantener los programas de software.

Ingeniero de pruebas: un ingeniero de pruebas de software es responsable de crear planes de prueba y prueba de los programas desarrollados.

Diseñador de interfaces: es el diseñador de interfaces de usuario para máquinas y software, tales como computadoras, electrodomésticos, dispositivos móviles y otros dispositivos electrónicos.

Diseñador de bases de datos: es el encargado de producir modelos detalladas de las bases de datos que se utilizan en los proyectos de software.

Arquitecto de Software: es un experto en software que hace que las opciones de diseño de alto nivel y dicta normas técnicas, incluyendo la codificación de software estándares, herramientas y plataformas.

Analista de negocios: un analista de negocios es responsable de resolver los problemas mediante la regulación de las relaciones entre la empresas y el cliente, así como también de documentar varias partes de un proyecto de software (p.e. documentación de requerimientos).

Ingeniero de requisitos: encargado de definir, documentar y mantener los requisitos a partir de los cuales el proyecto de software se llevara a cabo.

Analista de sistemas: es un profesional de TI que se especializa en el análisis, diseño e implementación de sistemas de información. Los analistas de sistemas se encargan de evaluar la idoneidad de los sistemas de información en términos de sus resultados previstos y servir de enlace con los usuarios finales, proveedores de software y programadores para lograr estos resultados.

2.1.2.2 Roles ISO/IEC 12207

Adquisidor: es una persona o un grupos de interés que obtiene productos o servicios de proveedores de software.

Proveedor: es en el encargado de una organización de proporcionar los productos o servicios de software.

Implementador: Encargado de ejecutar las tareas de desarrollo de un proyecto de software.

Mantenimiento: puede ser tanto un implementador o el encargado de llevar a cabo el mantenimiento del software.

Operador: es responsable de la ejecución del sistema que se esta desarrollando.

Configurador: es responsable de la creación y transformación de la información necesaria por un individuo o un grupo.

Evaluador: es el encargado de hacer pruebas y medir un proceso de software o un producto mediante el uso de los datos recogidos durante las tareas reales que se realizan.

Auditor: es el encargado de investigar los productos y procesos compatibles con los acuerdos del proyecto.

Especialista en usabilidad: se ocupa de las demandas y necesidades de las partes interesadas, como las actividades de diseño basados en factores y habilidades humanas y su cumplimiento.

Gerente: identifica y gestiona el estado del proyecto (es decir la condición y la progresión del proyecto) con respecto a las limitaciones del proyecto (e.g. objetivos, presupuesto, horarios).

Gestión de activos: es un tipo de gerente que se encarga de la gestión y optimización de los activos en relación con el plan de que él o ella prepara.

Administrador del conocimiento: trabaja en la colección de especial conocimiento y habilidades en toda la organización y utiliza esto para la mejora de los productos y servicios.

Administrador de reutilización: busca encontrar circunstancias favorables o ventajosas para las partes reutilizables de un producto o un servicio.

2.1.2.3 Roles en Programación Extrema

Programadores: son las personas que necesitan tener buenas habilidades de comunicación y colaboración, tanto para el equipo e individual. Ellos son responsables de desarrollar, mantener y probar el software. Los programadores toman las decisiones técnicas.

Clientes: encargados de formar a los equipos de dirección en términos de negocio y, en particular, en las decisiones de satisfacción del requisito.

Testers: encargados de ayudar a los clientes a crear casos de prueba funcionales.

Tracker: El papel de seguimiento compone un mecanismo de traza y la retroalimentación en la programación extrema, las estimaciones, las metas y las iteraciones realizadas por equipos son controlados por un rastreador, que proporciona retroalimentación. También responsable de la medición de las

limitaciones como la escasez de recursos y los plazos de entrega en comparación con la evaluación de los objetivos.

Entrenador: es responsable del proyecto, necesita entender los problemas que se producen durante el proceso para instruir a los miembros del equipo y transferir la información o, a veces experiencias entre los equipos y las personas.

Gerente: es responsable de las decisiones finales, y también un objetivo de esta función es la de reconocer los problemas que probablemente ocurren durante el ciclo de vida de desarrollo, también se encarga de la toma de decisiones y de finalizar las actividades en diferentes etapas del proceso de desarrollo, tales como la evaluación de los objetivos, la recopilación de requisitos, etc.

2.1.2.4 Roles en FDD

Administrador de proyectos: que administra la totalidad del proyecto y mantiene la configuración de trabajo del equipo de software.

Arquitecto de Software: se encarga de tomar las decisiones apropiadas para el desarrollo de software.

Gerente de Desarrollo de Software: es un papel que se centra en las actividades y negociaciones equipo diariamente durante el desarrollo de software actividades.

El programador jefe, el dueño de clases y el experto del dominio son los tres papeles utilizados en FDD. Los papeles secundarios incluyen; gerente (liberación), experto en el conocimiento, ingeniero de procesos, toolsmith y administrador del sistema. Por otra parte, los probadores, expertos de documentos y de implementación de software personal técnico son los otros papeles utilizados en estas prácticas.

2.1.2.5 Roles en SCRUM

Scrum Master: encargado de funciones de gestión específicas de Scrum, es responsable de la alineación de las prácticas y normas, ya que se han organizado. También interactúa con el equipo del proyecto, cliente y realiza la gestión entre ellos. Su objetivo es maximizar la productividad mediante la práctica de los valores ágiles y Scrum y seguimiento del equipo para evitar cualquier tipo de complicaciones.

Dueño del producto: es responsable de ejercer las actividades de gestión y control de proyectos. Además, también se encarga de transformar la acumulación del producto en las características del producto.

Cliente: El cliente evaluará continuamente los elementos del backlog, y ayuda a la selección para un sprint.

Equipo Scrum: es considerado como un equipo auto-organizado para producir una pieza de trabajo de un producto, donde el principal objetivo del equipo es lograr tiempo objetivos específicos de cada sprint.

Gerente: es responsable de la aplicación de las normas adecuadas para el proceso de desarrollo de software.

2.1.2 Conocimiento Desarrollo de Software

En el desarrollo de software, una gran cantidad de conocimiento puede ser compartido entre los miembros de desarrollo ya sea entre analistas y programadores o entre programadores e ingenieros de pruebas, por lo que mucho

de este conocimiento permanece tácito, ya que depende mucho de la interacción entre los miembros del equipo de trabajo para poder llegar a obtener o compartir dicho conocimiento. Muchos conceptos se han desarrollado para aliviar o guiar la transformación del conocimiento en el desarrollo de software, incluyendo la clandestinidad información, modularidad, objetos, funciones y procedimientos, los patrones, y más. Estos conceptos son apoyados por varios métodos, enfoques y herramientas que utilizan símbolos, gráficos, e idiomas.

También, el desarrollo de software es llevado a cabo mediante equipos de trabajo virtuales (Desarrollo Global de Software (Barceló et al., 2014; Prikladnicki et al., 2008)), donde los miembros de un equipo estan distribuidos en diferentes ubicaciones geográficas en el mundo, lo cual a conllevado a malentendidos (Herbsleb & Moitra, 2001; R. E. Jensen, 2012), errores (Basili & Perricone, 1984; Westland, 2002) y al desperdicio de recursos (Basili & Caldiera, 1995) (económicos y/o humanos) al no reutilizar el conocimiento con el que ya se cuenta (Van Vliet, 2010).

2.1.3 Fuente de Conocimiento Desarrollo de Software

En este tipo de ambientes de trabajo existen diferentes fuentes de conocimiento, las cuales se identifican en las personas y los artefactos como propone (Becerra-Fernandez & Sabherwal, 2010). Una descripción de estas fuentes se presenta a continuación:

Personas: El conocimiento que se encuentra en los desarrolladores, obtenido de la experiencia de trabajar en los proyectos de software (asesoría, consultas con otros desarrolladores, cursos de capacitación, así como también de la búsqueda individual en sitios web, libros y manuales) (Basili et al., 1994; Johansson et al., 2000).

Artefactos: El conocimiento que se encuentra en la documentación (e.g. libro de requerimientos, documento de visión, base de código fuente y manual de usuario (Jacobson et al., 1999)) obtenido a partir de las practicas de la empresa, los repositorios donde los desarrolladores consultan documentos físicos o digitales para completar sus actividades diarias (e.g. blogs (Vasilescu et al., 2013; Vasilescu et al., 2014), manuales (Dagenais & Robillard, 2010; Dekel & Herbsleb, 2009), marcadores (Cockburn & McKenzie, 2001; W. Jones et al., 2002), control de versiones (Loeliger & McCullough, 2012; Pilato et al., 2008)) y en las herramientas para administrar conocimiento de los procesos, etapas y actividades de los proyectos (Colmenar Santos et al., 2007; Sarkan et al., 2011).

2.2 Gestión del Conocimiento

El entorno actual de las organizaciones se caracteriza por el cambio continuo y la evolución en cuanto a las prácticas y tecnologías utilizadas. Las acciones deben ser anticipadas y adaptativas, basadas en un ciclo rápido de creación de conocimiento. De ahí la necesidad de un proceso de negocio que formalice la gestión y aseguramiento de los activos intelectuales (Ammar-Khodja & Bernard, 2008; Serban & Luan, 2002).

Es por ello que se han hecho esfuerzos por tener una adecuada gestión del conocimiento, la cual es ampliamente conocida y practicada por grandes organizaciones, como una herramienta útil para que cada uno de los miembros de la empresa sepa lo que otros conocen para mejorar los resultados de sus actividades. De manera que si las organzaciones pueden manejar el proceso de aprendizaje, pudieran ser más eficientes (Prusak, 2001).

Este tipo de gestión se compone de una serie de prácticas que permiten identificar crear, representar y distribuir el conocimiento para su reutilización, distribución y aprendizaje. Por ello existe un gran interés en el tratamiento de conocimiento

como un recurso significativo en las organizaciones, enfocándose en los conocimientos humanos, y cómo explotarlos para tener el máximo rendimiento en la organización (Ponelis & Fairer-Wessels, 2014).

El creciente interés por el conocimiento y la gestión del conocimiento organizacional se deriva de la transición a la economía del conocimiento, donde el conocimiento es visto como la principal fuente de creación de valor y ventaja competitiva sostenible (Alavi & Leidner, 2001; Wang, Noe, & Wang, 2014).

Por tal motivo, la gestión del conocimiento tiene como meta mitigar la pérdida y el riesgo en los procesos de producción de la organización, mejorar la eficiencia de la organización e innovar. Esto añade un gran valor a las organizaciones, de manera que se pueden tomar mejores decisiones, tener un flujo libre de ideas que llevan al conocimiento e innovación, eliminar los procesos redundantes, mejorar el servicio al cliente y la eficiencia conduciendo a una gran productividad.

La gestión del conocimiento ha beneficiado a diferentes áreas, tales como de educación (G. Jones & Sallis, 2013; Petrides & Nodine, 2003), cuidado de la salud (Abidi, 2001; Nicolini, Powell, Conville, & Martinez-Solano, 2008), desarrollo de software (Jain, 2011; Rodríguez et al., 2004), entre otras.

Particularmente en la industria del software se han abordado los retos de adaptación del conocimiento a las tecnologías emergentes (p. ej. cuando los desarrolladores utilizan una tecnología que no es familiar para los miembros del equipo, ellos utilizan la técnica de “aprender sobre la marcha”, lo que muchas veces resulta en retrasos), acceso al dominio del conocimiento (p. ej. cuando una organización debe adquirir dominio de conocimientos nuevos, ya sea por formación o mediante la contratación de empleados con conocimientos y difundirla por todo el equipo), intercambio del conocimiento sobre las políticas y prácticas locales (p. ej. cuando los desarrolladores suelen difundir el conocimiento a través de reuniones informales, de manera que, no todo el mundo tiene acceso al

conocimiento que necesita), capturar el conocimiento y saber qué es lo que sabe cada uno (p. ej. cuando algún desarrollador con conocimiento crítico deja la organización, crea huecos de conocimiento que ninguno se da cuenta hasta que es necesario dicho conocimiento) y colaboración e intercambio de conocimiento (p. ej. cuando los miembros de un equipo a menudo están distribuidos geográficamente con diferentes zonas horarias y deben interactuar para intercambiar o transferir información) (Rus & Lindvall, 2002).

2.2.1 La naturaleza del conocimiento

El conocimiento es muy distinto de los datos e información, aunque los tres términos se utilizan a veces indistintamente. Sin embargo, son bastante distintos en la naturaleza.

Los datos comprenden hechos, observaciones o percepciones (que pueden o no ser correcta). Por sí mismo, los datos representan números crudos o afirmaciones y por lo tanto pueden estar fuera de contexto, significado o intención. Veamos tres ejemplos de lo que se considera como datos. A continuación, se basará en estos ejemplos para examinar el significado de información y conocimientos.

La información es un subconjunto de datos, sólo incluyendo aquellos datos que poseen contexto, relevancia y propósito. Información general implica la manipulación de datos en bruto para obtener una indicación más significativa de las tendencias o patrones en los datos.

El conocimiento se ha distinguido de los datos y la información de dos maneras diferentes. Una visión más simplista considera el conocimiento como al más alto nivel en una jerarquía con la información en el nivel medio y los datos en el nivel más bajo. Según esta visión, el conocimiento se refiere a la información que permita la acción y las decisiones o información con la dirección. Por lo tanto, el conocimiento es intrínsecamente similar a la información y datos, aunque es el

más rico y el más profundo de los tres, y, en consecuencia, también el más valioso. Sobre la base de este punto de vista, los datos se refieren al descubierto hechos vacío de sentido, por ejemplo, un número de teléfono. La información es de datos en el contexto, por ejemplo, una guía telefónica. El conocimiento es la información que facilita la acción, por ejemplo, las personas que son los expertos en el dominio dentro de una organización.

Aunque esta visión simplista de conocimiento puede no ser completamente inexacta, sentimos que no explica en su totalidad las características del conocimiento. En lugar de ello, utilizamos una perspectiva más completa, según el cual el conocimiento es intrínsecamente diferente de la información. En lugar de considerar el conocimiento como un conjunto más rico o más detallada de los hechos, se define el conocimiento en un área tan creencias justificadas acerca de las relaciones entre los conceptos relevantes para esa área en particular (Nonaka & Takeuchi, 1995).

2.2.2. Tipos de Conocimiento

El conocimiento procedimental o declarativa: la primera distinción que examinamos es la que existe entre el conocimiento declarativo (hechos) y el conocimiento procedimental (cómo montar una bicicleta). El conocimiento declarativo (o conocimiento sustantivo, como también se le llama) se centra en las creencias acerca de las relaciones entre las variables. Por ejemplo, todos los demás en igualdad de condiciones, un precio mayor cobrado por un producto podría causar una reducción en su número de ventas. El conocimiento declarativo puede afirmar en forma de proposiciones, correlaciones esperadas o fórmulas relativas conceptos representados como variables. Por ejemplo, indica que la suma del cuadrado de la seno de un ángulo y el cuadrado del coseno del mismo ángulo sería igual es un ejemplo de conocimiento declarativo. Del mismo modo, la identificación del producto específico cuenta con una talla específicas de los

clientes es también un ejemplo de conocimiento declarativo. El conocimiento procedimental, en cambio, se centra en las creencias relacionadas secuencias de pasos o acciones a los resultados deseados (o no deseados). Un ejemplo de tal conocimiento procedimental es el conjunto de creencias justificadas sobre el procedimiento que deben seguirse en una organización del gobierno en la decisión sobre a quién adjudicar el contrato para un área en particular como por ejemplo, el desarrollo de sistemas de información (Kogut & Zander, 1992).

Conocimiento Tácito o Explicito: Otra clasificación importante de conocimiento ve como tácito o explícito. El conocimiento explícito típicamente se refiere al conocimiento que se ha expresado en palabras y números. Tal conocimiento puede ser compartida formal y sistemática en forma de datos, especificaciones, manuales, dibujos, audio y cintas de vídeo, programas informáticos, patentes, y similares. También hay que señalar que si bien el conocimiento explícito podría asemejarse a los datos o información en forma, se conserva la distinción mencionada anteriormente en este capítulo; si bien explicado, los principios de análisis de mercado de valores son creencias justificadas acerca de las relaciones en lugar de hechos u observaciones simples. También las reglas acerca de cómo procesar un reembolso de viajes, lo que se convierte en incrustado en un sistema de planificación de recursos empresariales, se considera el conocimiento explícito (Nonaka & Takeuchi, 1995).

Por el contrario, el conocimiento tácito incluye ideas, intuiciones y corazonadas. Es difícil expresar y formalizar, y por lo tanto difícil de compartir. Es más probable que sea personal y basado en experiencias y actividades individuales conocimiento tácito. Por ejemplo, a través de años de observación de una industria en particular, un analista de mercado de valores podría adquirir conocimientos que le ayuda a hacer recomendaciones a los inversores en el mercado de valores con respecto a las tendencias del mercado probables a corto plazo ya largo plazo de las poblaciones de las empresas dentro de esa industria. Tal conocimiento se considerará tácita, a menos que el analista pueda verbalizar en la forma de un documento que otros puedan usar y aprender. El conocimiento tácito puede incluir

también la experiencia que es tan específico que puede ser demasiado caro para hacer explícita; por lo tanto, la organización opte por dejar que residen con el experto (Polanyi, 1967).

Como se mencionó anteriormente, las formas explícitas y tácitas de conocimiento son muy diferentes. Sin embargo, es posible convertir el conocimiento explícito en tácito, como ocurre, por ejemplo, cuando una persona lee un libro y aprende de él, convirtiendo así el conocimiento explícito contenido en el libro en el conocimiento tácito en la mente del individuo. Del mismo modo, el conocimiento tácito a veces se puede convertir en conocimiento explícito, como sucede cuando un individuo con un considerable conocimiento tácito sobre un tema, escribe un libro o manual de formalizar ese conocimiento.

2.2.3. Proceso de Gestión del Conocimiento

Anteriormente definimos la gestión del conocimiento como la realización de actividades involucradas en el descubrimiento, captura, intercambio y la aplicación de conocimiento con el fin de mejorar, de manera rentable el impacto del conocimiento en el logro de los objetivos de la empresa.

Descubrimiento del Conocimiento: se puede definir como el desarrollo de nuevos conocimientos tácito o explícito de los datos y la información o de la síntesis de los conocimientos previos. El descubrimiento de nuevo conocimiento explícito se basa más directamente en la combinación, mientras que el descubrimiento de nuevo conocimiento tácito se basa más directamente en la socialización. En cualquiera de los casos, los nuevos conocimientos se descubre mediante la síntesis de conocimiento a partir de dos o más zonas diferenciadas con el conocimiento explícito de dos áreas que están siendo sintetizados a través de la combinación, y el conocimiento tácito de dos áreas que se sintetiza a través de la socialización.

Captura del Conocimiento: el conocimiento puede existir dentro de las personas (individuos o grupos) y artefactos (prácticas, tecnologías, o repositorios). Puede ser que a veces residir dentro de la mente de un individuo sin que el individuo sea capaz de reconocer y compartirlo con los demás. De manera similar, el conocimiento puede residir en una forma explícita en un manual pero poca gente puede ser consciente de ello. Es importante obtener el conocimiento tácito de las mentes de los individuos, así como el conocimiento explícito del manual, de manera que el conocimiento puede entonces ser compartida con otros. Este es el objetivo de la captura del conocimiento que puede ser definido como el proceso de recuperar el conocimiento, ya sea explícita o tácita que reside dentro de las personas, los objetos, o entidades de organización. Además, el conocimiento de ser capturado podría residir fuera de los límites de la organización, incluidos los consultores, competidores, clientes, proveedores y empleadores anteriores de los nuevos empleados de la organización.

Intercambio de Conocimiento: El intercambio de conocimientos es el proceso a través del cual el conocimiento explícito o tácito se comunica a otras personas. Tres aclaraciones importantes están en orden. En primer lugar, el intercambio de conocimientos medios de transferencia efectiva, para que el destinatario del conocimiento puede entender lo suficientemente bien como para actuar sobre ella (M. C. Jensen & Meckling, 1992). En segundo lugar, lo que se comparte es el conocimiento en lugar de recomendaciones basadas en el conocimiento; el primero consiste en el receptor de adquirir el conocimiento compartido, así como ser capaz de actuar en base a ella, mientras que el segundo (que es la dirección, se discute en la siguiente sección), simplemente implica la utilización de los conocimientos sin destinatario internalizar el conocimiento compartido. En tercer lugar, el intercambio de conocimientos puede tener lugar a través de las personas, así como a través de grupos, departamentos u organizaciones (Alavi & Leidner, 2001).

Aplicación del Conocimiento: El conocimiento contribuye directamente al rendimiento de la organización cuando se utiliza para tomar decisiones y realizar

tareas. Por supuesto, el proceso de aplicación del conocimiento depende del conocimiento disponible y el conocimiento en sí depende de los procesos de descubrimiento de conocimiento, capturar y compartir.

2.3 Representación del Conocimiento

La inteligencia exhibida por las personas es ciertamente uno de los fenómenos más complejos. Un aspecto sorprendente del comportamiento de la inteligencia es que esta claramente condicionada por el conocimiento: por una gran variedad de actividades que realizamos, por las decisiones que tomamos de que hacer a partir de lo que ya sabemos o creemos sobre el mundo.

El conocimiento es la relación que existe entre el sujeto que tiene información y la proposición, que es una idea expresada por una oración declarativa simple.

La representación del conocimiento es la disciplina encargada de utilizar símbolos formales para representar una colección de proposiciones, que son generadas por alguna persona con conocimiento sobre algún tema en particular. Se centra en representar y razonar en un ambiente con diferentes tipos de propiedades, usualmente con la meta de hacer toma de decisiones, por ejemplo como actuar de la mejor manera en dicho ambiente. Un termino nuevo que ha surgido es el de agentes que son entidades, que son programas computacionales (en este caso se les llama agentes de software) o podrían ser personas ordinarias que toman decisiones en base a cierto conocimiento representado de una manera formal en proposiciones.

2.4 Paradigma de Agentes en la Ingeniería de Software

Los ingenieros de software han derivado progresivamente una mejor comprensión de las características de complejidad en el software. Es ampliamente reconocido que la interacción es probablemente la característica más importante de software complejo. Las arquitecturas de software que contienen muchos componentes que interactúan de forma dinámica, cada uno con su propio hilo de control, y participan en protocolos complejos y coordinados, son típicamente órdenes de magnitud más compleja para diseñar correctamente y eficientemente que los que simplemente calcular una función de alguna entrada a través de un único hilo de control. Actualmente muchas de las aplicaciones del mundo real tienen este tipo de características

Como consecuencia un tema importante en la investigación ciencias de la computación durante al menos las últimas tres décadas ha sido el desarrollo de herramientas y técnicas para modelar, entender e implementar sistemas en los que la interacción es la norma. Muchos investigadores creen que, en el futuro, el cálculo en sí se comprenderá principalmente como un proceso de interacción. Del mismo modo que podemos comprender muchos sistemas como compuesta de objetos esencialmente pasivos, que tienen un estado, y sobre la cual podemos realizar operaciones, por lo que podemos entender muchos otros como siendo compuesto de interactuar, agentes semiautónomos. Este reconocimiento ha llevado al crecimiento del interés por los agentes como un nuevo paradigma de la ingeniería de software.

Donde podemos definir a un agente como un sistema informático que se encuentra en algún ambiente y que es capaz de una acción autónoma en este entorno con el fin de cumplir con sus objetivos de diseño.

2.4.1. Agentes Inteligentes

Podemos definir un agente inteligente mediante las siguientes propiedades (Wooldridge & Jennings, 1995):

- Reactividad: Los agentes inteligentes son capaces de percibir su entorno y responder de manera oportuna a los cambios que se producen en el.
- proactividad: Los agentes inteligentes son capaces de mostrar un comportamiento dirigido a un objetivo al tomar la iniciativa.

Estas propiedades son más exigentes de lo que podrían parecer a primera vista. Para ver por qué, consideremos ellos a su vez. En primer lugar, considere proactividad: comportamiento dirigido a un objetivo. No es difícil construir un sistema que muestra un comportamiento dirigido a un objetivo lo hacemos cada vez que escribimos un procedimiento en Pascal, una función en C, o un método en Java. Cuando escribimos un procedimiento de este tipo, se describe en términos de los supuestos sobre los que se basa (formalmente, su precondición) y el efecto que tiene de las hipótesis son válidas (su condición posterior).

Los efectos del procedimiento son su objetivo: lo que el autor del software se propone el procedimiento de lograr. Si la condición se cumple cuando se invoca el procedimiento, entonces esperamos que el procedimiento se ejecutará correctamente: que va a terminar, y que a la terminación, la postcondición sea verdad, es decir, se logrará el objetivo.

Este es un comportamiento dirigido a un objetivo: el procedimiento es simplemente un plan o una receta para alcanzar la meta. Este modelo de programación está bien para muchos entornos. Por ejemplo, funciona bien cuando consideramos sistemas funcionales, como se discutió anteriormente. Pero para los sistemas no funcionales, este sencillo modelo de programación dirigida a objetivos no es aceptable, ya que hace algunos supuestos limitantes importantes.

En particular, se supone que el medio ambiente no cambia, mientras que el procedimiento se está ejecutando. Si el entorno cambia, y, en particular, si las suposiciones subyacentes (condición previa) el procedimiento es falsa, mientras que el procedimiento se está ejecutando, entonces el comportamiento del procedimiento no pueden ser definidos, a menudo pueden colapsar estrellarse. Además, se supone que el objetivo, es decir, la razón de la ejecución del procedimiento, sigue siendo válida al menos hasta que el procedimiento termina. Si la meta, no permanece válida, simplemente no hay razón para continuar la ejecución del procedimiento.

2.4.2. Agentes y Sistemas Expertos

Los sistemas expertos fueron la tecnología más importante en la Inteligencia Artificial en 1980 (Hayes-Roth, Waterman, & Lenat, 1984). Un sistema experto es un sistema capaz de resolver problemas o dar consejos en algún dominio de conocimiento especifico.

La distinción más importante entre los agentes y los sistemas expertos es que los sistemas expertos son inherentemente sin cuerpo. Esto quiere decir que no interactúan directamente con cualquier entorno: obtienen su información a través de sensores, pero a través de un usuario que actúa como intermediario. De la misma manera, no actúan en cualquier entorno, dan información o asesoramiento a un tercero. Además, los sistemas expertos no son generalmente capaces de cooperar con otros agentes.

A pesar de estas diferencias, algunos sistemas expertos (en particular aquellos que realizan tareas de control en tiempo real) se parecen mucho a los agentes.

2.5 Localización del expertise en el Desarrollo de Software

La localización del expertise es un tema complejo que aborda la búsqueda del conocimiento de mejor nivel, donde para esto se necesita tener una buena gestión de todo este conocimiento (2.2) que circula dentro de la empresa, se necesitan llevar a cabo ciertos procesos para identificar las fuentes de conocimientos, los proveedores del conocimiento. Para esto es necesario conocer el proceso de desarrollo de software (2.1) de las empresas, quienes son los involucrados y donde obtienen el conocimiento dichos proveedores. Utilizando un paradigma orientado a agentes (2.4) se propone una arquitectura basada en agentes para poder ayudar a automatizar este proceso de gestión del conocimiento para dar soporte a la búsqueda de expertise en el desarrollo de software. Es necesario establecer un lenguaje mediante el cual los agentes dentro de la arquitectura se comunicaran para esto se necesita una representación adecuada del conocimiento (2.3) que circula dentro de la empresa para crear una semántica.

CAPÍTULO III
MÉTODO

En esta sección se detallará el proceso mediante el cual se desarrollo este proyecto tesis. Se mostrarán cada una de las etapas definidas para llegar al prototipo de la arquitectura propuesta para dar soporte a la localización del *expertise* en el desarrollo de software mediante en enfoque basado en agentes.

3.1 Participantes

Se realizaron entrevistas y un grupo focal a desarrolladores de software de empresas de la región.

Grupo focal:

4 Desarrolladores

2 Diseñadores

Entrevistas:

5 Desarrolladores

3 Líder de proyecto

3.2 Materiales

- Guía entrevista
- Guía grupo focal
- Lista de asistencia
- Grabadora de audio
- Cámara digital

3.3 Procedimiento

Se definió una metodología general con varias etapas que contemplan los pasos necesarios para llegar al prototipo antes mencionado.

Figura 3.1 Metodología empleada en el proyecto de investigación.

3.3.1 Análisis de literatura

Dentro de esta primera etapa se estableció el tema de investigación que se aborda dentro de este trabajo de tesis, posteriormente se hizo una selección de trabajos relacionados al tema de investigación, a partir de estos se hizo una clasificación para estudiar el estado del arte.

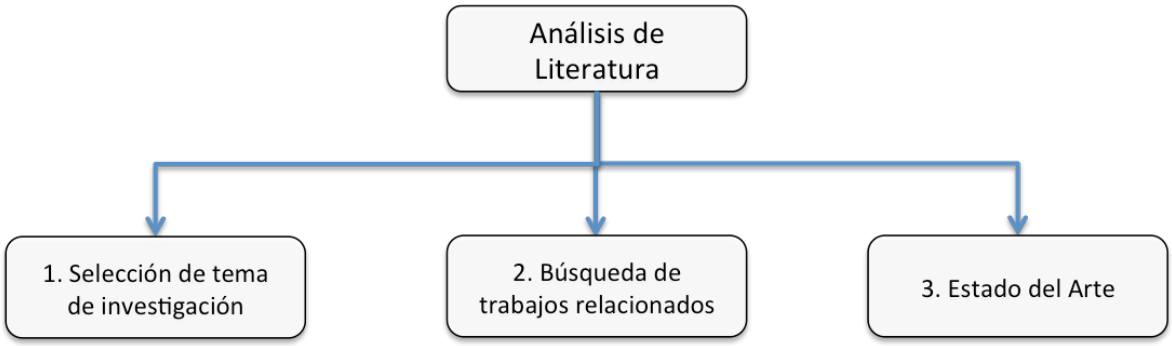

Figura 3.2 Fase Análisis de Literatura

1. *Selección del Tema de Investigación*: se establecen los objetivos, el área que se desea estudiar, formulación del problema. Como resultado se obtiene el tema de investigación sobre el cual se buscaran trabajos relacionados para saber de que manera lo han abordado otros investigadores.
2. *Búsqueda de trabajos relacionados*: una vez definido el tema de investigación se procede a buscar trabajos que aborden la misma problemática en el contexto de investigación que se definió.
3. *Estado del arte*: A partir de todos los trabajos relacionados que se seleccionaron, se hace una revisión del estado del arte, ordenando cronológicamente los trabajos describiendo su aportación, metodología utilizada y los autores.

Como resultado de esta fase se obtiene el tema de investigación y un resumen de todos los trabajos que han abortado este tema, a partir de esto se puede

trabajar en alguna problemática que no se haya abordado o en mejorar alguna existente por lo que esto sirve de base para la siguiente fase.

3.3.2 Identificación del problema

Como parte de esta fase se uso la metodología (KoFI) Identificación del Flujo de Conocimiento (Rodríguez-Elias, Vizcaíno, Martínez-García, Favela, & Piattini, 2009). Esta metodología se basa en la identificación de los flujos de conocimiento a través del estudio y modelización de los procesos de una organización. KoFI se centra en la identificación del conocimiento generado en las actividades de los procesos principales de una organización. KoFI también identifica la ubicación de las fuentes de almacenamiento y cómo la información puede ser recuperada de ellos. Esta metodología tiene dos objetivos principales: 1) para obtener información que le ayudará a construir un mapa de conocimiento; y 2) para obtener los requisitos que le ayudarán en el diseño de sistemas basados en el conocimiento. La metodología Kofi consta de 4 fases:

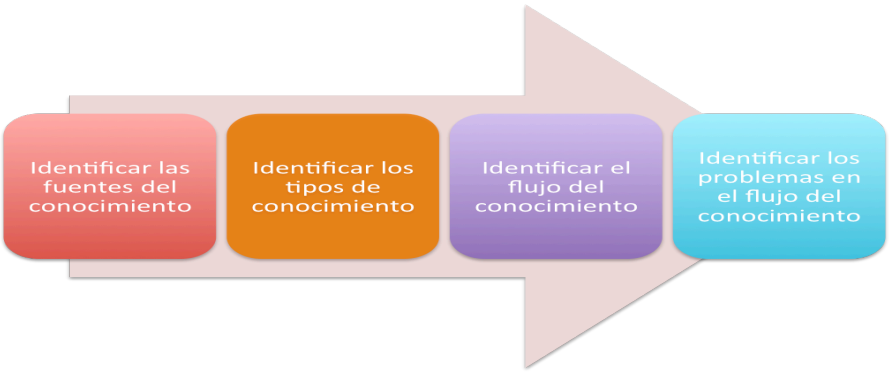

Figura 3.3 Fases metodología KoFI

Para poder obtener la información de cada una de estas fases de la metodología se realizara un grupo focal con desarrolladores de software de empresa de la

región para consultarlos sobre el tema de investigación, para determinar el flujo actual del conocimiento y los obstáculos que se presentan en el.

Fase 1: consiste en la identificación de las diferentes fuentes en las que se genera o almacena el conocimiento. En esta fase fue necesario tener en cuenta las fuentes de conocimiento que podrían ser utilizadas para localizar a un experto para dar solución a un problema dentro de una actividad, para obtener esta información se realizaran grupos focales con desarrolladores de software de las empresas de la región.

Fase 2: Consiste en la identificación de los tipos de conocimiento utilizados y generados en los procesos principales de la organización. En esta fase fue necesario identificar los temas de conocimiento implicados en el proceso de búsqueda de *expertise* en el desarrollo de software, teniendo en cuenta los diferentes tipos de conocimientos generados por la organización. Los tipos de conocimiento relevantes para este trabajo están relacionados con las características de las actividades de los desarrolladores de software. La actividad en esta fase no trata de describir los temas en detalle, pero si identificarlos como parte de los requisitos de conocimientos.

Fase 3: Identifica cómo fluye el conocimiento dentro de la organización. Esta fase implicó la creación de un modelo de flujo de conocimiento del proceso de búsqueda de expertise en los equipos de desarrollo de software.

Fase 4: Consiste en la identificación de los principales problemas que obstaculizan el flujo de este conocimiento. Esta fase consiste en la identificación de los obstáculos que se presentan en el proceso de búsqueda de *expertise* los cuales serán obtenidos a partir de las entrevistas. Esto permitirá el proceso de clasificación de los obstáculos y la búsqueda de una posible solución.

Como resultado de esta fase se obtienen los obstáculos identificados así como también las implicaciones de diseño para abordar dichos obstáculos. Que serán la base de los requerimientos para la fase siguiente (diseño)

3.3.3 Diseño

En esta etapa se trabaja a partir de los resultados de la fase de identificación del problema donde se trabaja en posibles mecanismos para solucionar la problemática identificada.

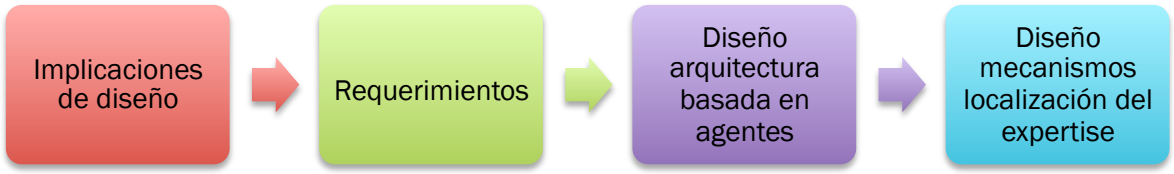

Figura 3.4 Etapas Fase de Diseño

Requerimientos: A partir de las implicaciones de diseño propuestas en la fase anterior, en esta se convierten en una lista de requerimientos de una propuesta de sistema para dar solución a la problemática identificada.

Diseño de Casos de uso: Con la lista de requerimientos se plantean entradas y salidas y el comportamiento que tendrán estas dentro del sistema propuesto.

Diseño de diagramas de secuencia: Se diseña la comunicación de los módulos del sistema.

Diseño de casos de escritorio: se diseñan de manera enriquecida los escenarios donde se presentan estos obstáculos de la manera mas apegada a la realidad para identificar todas los comportamientos que debe tener el sistema y la manera en que debe reaccionar ante dichas situaciones.

Como resultado de esta fase se obtienen la propuesta descrita del sistema que dará solución a la problemática identificada en fases anteriores, así como también los escenarios bajo los cuales se le harán pruebas al sistema una vez que este desarrollado.

3.3.4 Modelado

La descripción del sistema es importante para la creación de uno de ellos así como para el análisis de los ya existentes. Esta fase se generara un modelado para describir el sistema propuesto como solución para las problemática identificada. En lo que se refiere al diseño o construcción de sistemas, la especificación formal o modelado constituye una etapa importante. A partir de un diseño se construirá un modelo a partir de las especificaciones del sistema y de este modelo se pasa a la etapa de desarrollo o implementación.

En esta fase se trabaja en base a los requerimientos de la fase anterior generando un modelo formal para describir la comunicación de los procesos del sistema propuesto para dar solución a la problema identificada, este proceso incluye una validación de dicho modelo utilizando los casos de escritorio obtenidos en la fase anterior para, ver el comportamiento del sistema con los escenarios apegados a la vida real, este proceso se repite varias veces hasta que el modelo este congruente

y funcione en los casos de escritorio. A partir de esto ya se puede pasar a la siguiente fase que es el desarrollo.

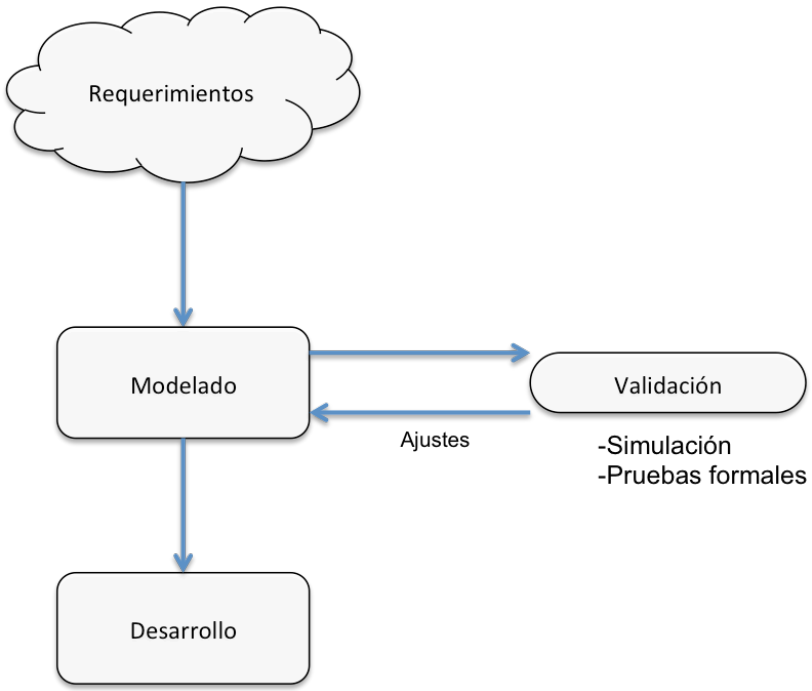

Figura 3.5 Etapas Fase de Modelado.

3.3.5 Desarrollo

En esta fase una vez que se tiene el modelado del sistema lo siguiente será desarrollar un prototipo del sistema, para hacer pruebas de validación al sistema, haciendo pruebas de escritorio de escenarios ideales de la manera en que debe comportarse el sistema.

CAPÍTULO IV
RESULTADOS

En esta sección se presentan los resultados obtenidos a partir del procedimiento descrito en el capitulo anterior.

4.1 Resultados de la Metodología KoFI

Para conocer el contexto del flujo de conocimiento y los obstáculos que existen para la distribución del mismo se presentan los entregables de la metodología KoFI (Rodríguez-Elias et al., 2009), los cuales son presentados por fase:

4.1.1 Fuentes de Conocimiento

A partir del grupo focal realizado con esta metodología se identificaron las fuentes de conocimiento en el desarrollo de software, las cuales resultaron ser muy cercanas a las que la literatura revisada menciona.

Tabla 4.1 Fuentes de Conocimiento en el Desarrollo de Software

Fuentes	Descripción
Artefactos	*Marcadores*: cuando los desarrolladores encuentran un sitio web que les fue de ayuda, ellos usualmente guardan el sitio en sus navegadores.
	Proyectos: los desarrolladores suelen guardar el código fuente de proyectos anteriores para poder reutilizarlos.
	Documentación Oficial: En algunos casos los desarrolladores suelen visitar el sitio oficial de alguna tecnología para obtener información acerca de ella.
	Manuales: En algunos casos el conocimiento adquirido es guardado como un manual de usuario.
Personas	Los desarrolladores suelen buscar a expertos que son las personas con cierto grado de conocimiento especifico de algún tópico o área en particular que pudiera ser de utilidad para resolver alguna problemática presentada en las actividades de desarrollo, por lo que esto involucra que todas las personas dentro de la organización representan una fuente de conocimiento.

En la tabla se muestran las fuentes de conocimiento en el desarrollo de software las cuales coinciden con la literatura, por lo cual se tomo en cuenta la manera en que (Becerra-Fernandez & Sabherwal, 2010) organiza las fuentes de conocimiento. Dichas fuentes de conocimiento identificadas sirven para confirmar lo que literatura menciona, así como también para profundizar particularmente en las fuentes de una organización de desarrollo de software y en los formatos en los que se encuentran dichas fuentes de conocimiento.

4.1.2 Tópicos del Conocimiento

Con el fin de identificar los escenarios donde se involucra el conocimiento, se hizó clasificación y organización del conocimiento identificado en las organizaciones de desarrollo de software a partir del grupo focal realizado, muestran los tres escenarios en los que el conocimiento esta involucrado (ver Tabla 4.2), los cuales son los escenarios conocimiento individual, intercambio de conocimiento y el de búsqueda de expertos.

El escenario del *conocimiento individual* se caracteriza por una necesidad de conocimiento, se trata de un desarrollador en busca de información en foros, documentos oficiales y en sus propios marcadores. Una vez que el desarrollador

encuentra información útil, la transición (tácito a explícito) comienza. Esto sucede cuando el desarrollador almacena esta información.

El escenario de *intercambio de conocimiento* se caracteriza por la forma en que el conocimiento se transfiere, que se puede obtener mediante la formación directamente con un experto (por ejemplo, asesoramiento y / o de formación) o indirectamente a cabo mediante el uso de algún tipo de acción explícita recomendado por los expertos (por ejemplo, manuales, foros especializados, etc.).

Por último, el escenario de *búsqueda de expertos* se refiere a la identificación de la experiencia y el conocimiento que tienen los colegas acerca de una consulta en particular para adquirir nuevos conocimientos o resolver un obstáculo en relación con cualquier actividad laboral. Una vez que un desarrollador se encuentra un experto, intercambian Intercambio de conocimientos.

Tabla 4.2 Representación del Conocimiento

Escenario	Característica	Representación
Conocimiento Individual	Necesidad de Conocimiento	Foros
		Documentación oficial
		Video Tutoriales
		Marcadores
		Proyectos
	Transición(tácito a explicito)	Formato
		Autor
		Tema
		Nombre Carpeta
		Notas
		Lenguaje
		Fuente
Intercambio de conocimiento	Directo	Cursos de capacitación
		Asesoría
	Indirecto	Palabras clave
		Marcadores
		Sitios Web
		Manuales
Búsqueda de Expertos	Nivel de Expertise	Experiencia
		Área de trabajo
		Disponibilidad
		Lenguajes de programación
		Recomendaciones
		Aportaciones

4.1.3 Flujo del Conocimiento en el Desarrollo de Software

Para comprender el flujo de conocimiento identificado se construyo un modelo referente al flujo de conocimiento en el ambiente del desarrollo de software el cual esta representado mediante un autómata (ver Figura 4.1).

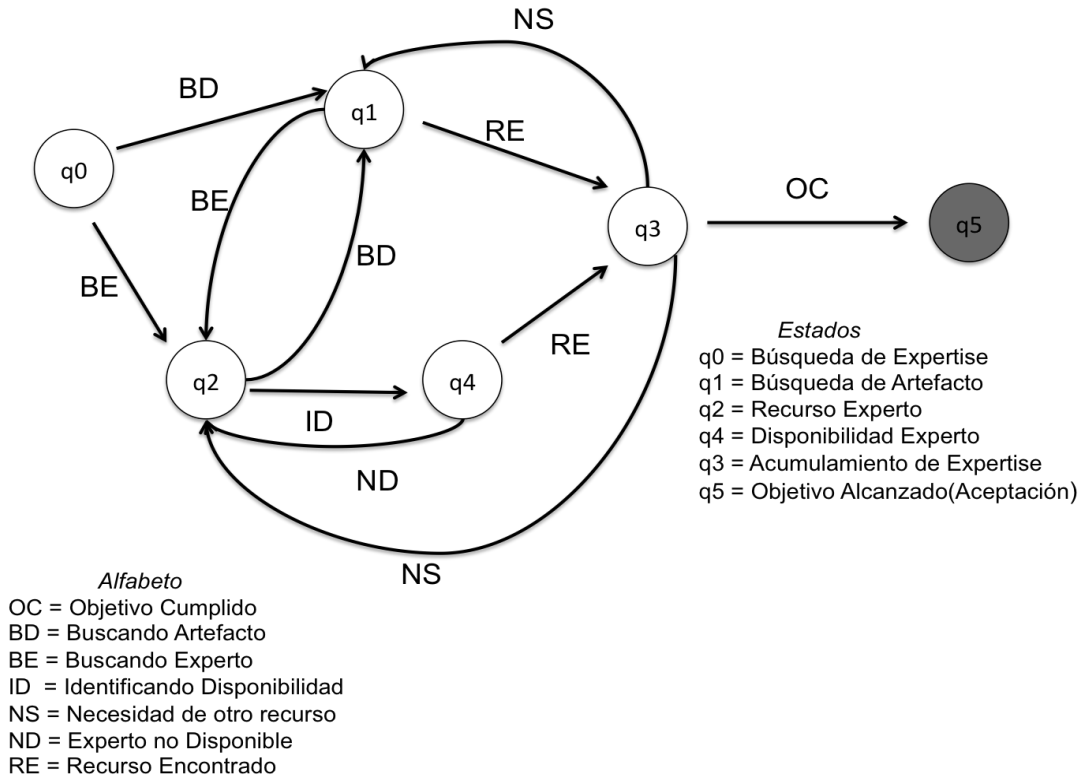

Figura 4.1 Implementación móvil de las esferas de trabajo colaborativas.

El autómata representa el flujo del conocimiento del proceso búsqueda de *expertise*, donde se inicia teniendo la necesidad de conocimiento para resolver alguna dificultad en una actividad (q0). Posteriormente se puede elegir entre hacer

una búsqueda de artefactos (q1) o una búsqueda de un experto (q2). En el caso de elegir una búsqueda de artefactos (q1) se busca entre todos los artefactos disponibles (páginas, manuales, videos, código reutilizado), al encontrar un artefacto se verifica si este ayudó a cumplir el objetivo o aún es necesario buscar más artefactos (q3). Si no es necesario buscar más artefactos entonces el objetivo se cumplió (q5), de lo contrario se puede buscar más artefactos (q1). Un artefacto puede sugerir que se busque a un experto (q2). En el caso de elegir una búsqueda de experto (q2) se inicia buscando expertos con el grado de conocimiento para resolver la dificultad que se tienen en la actividad, una vez que se encuentra un experto se necesita comprobar su disponibilidad (q4) para iniciar una interacción con él, posteriormente se comprueba si la consulta del experto fue suficiente (q3) o se necesita consultar al experto o a otro (q2) , también en el caso de estar buscando un experto puede ocurrir que un experto sugiera algún artefacto (q1), si el objetivo se cumplió el proceso termina (q5).

4.1.4 Problemas con el flujo de conocimiento en el Desarrollo de Software

Como resultado del modelo del flujo del conocimiento se identificaron los problemas que impiden la distribución del conocimiento, los cuales se muestran en la Tabla 4.3.

Tabla 4.3 Problemas Flujo del Conocimiento

Problemas	**Situaciones**
1. Administración de los artefactos (individual o grupal)	En algunos casos se conoce al proveedor del conocimiento pero no se tiene acceso a sus artefactos (blogs, manuales, código reutilizado).
2. Administración de los expertos	Muchas veces es difícil encontrar a la persona con el nivel adecuado de *expertise*.
3. Disponibilidad de los expertos	En algunos casos no se sabe si el *expertise* o el experto está disponible para la persona que lo está buscando.
4. Tiempo de resolución de dificultades	En algunos casos se pierde mucho tiempo en la búsqueda de *expertise* por qué no se cuenta con el conocimiento de donde se encuentra o quienes son los proveedores.

En la tabla se presentan los problemas más comunes que presentan los desarrolladores que participaron en el grupo focal, los problemas identificados fueron: la administración de los artefactos (individual o grupal), administración de los expertos, disponibilidad de los expertos y el tiempo de resolución de dificultades. Estas problemas sirvieron para poder diseñar posibles soluciones para tratar de abordar una o varias de estas situaciones que se presentan en el desarrollo de software.

4.2 Diseño

Como resultado de la metodología utilizada se obtuvieron las bases para el diseño de una arquitectura para dar soporte a la localización del *expertise* en el Desarrollo de Software.

4.2.1 Implicaciones de Diseño

Con base en el conjunto de problemas de flujo de conocimiento identificados (ver Tabla 4.3), estos fueron transformados en características de la búsqueda de expertise para ayudar a la definición de las implicaciones de diseño para una posible solución a la problema identificada. Por lo que en la Tabla 4.4 se presentan las implicaciones del diseño de un sistema que podría proporcionar un apoyo para la búsqueda de expertise en el desarrollo de software. La primera columna define la característica que sería deseables en el proceso de búsqueda de *expertise*. La segunda columna describe la implicación de diseño que se debe tomar en cuenta para poder dar soporte a la búsqueda de *expertise* durante las actividades del desarrollo de software.

Dichas implicaciones consisten en la gestión del conocimiento, la cual es necesario para tener la relación de los expertos y sus productos (artefactos) dentro de la organización. Otras implicaciones a que considerar son la búsqueda de artefactos y expertos esto es importante ya que es un proceso muy común que los desarrolladores busquen artefactos para solucionar problemas y así como también

que consulten a sus colegas para asesoría. Por ultimo es deseable que haya un acceso al conocimiento por parte de cualquier miembro dentro de una organización de desarrollo de software, lo cual implica que cada usuario debe estar dispuesto a compartir su conocimiento, para que a través de algún mecanismo esto este se convierta en accesible para cualquier usuario y en cualquier momento.

Tabla 4.4 Implicaciones de diseño

Características	**Implicaciones de diseño**
1. Gestión del conocimiento	En algunos casos se conoce al proveedor del conocimiento pero no se tiene acceso a sus artefactos (blogs, manuales, código reutilizado).
2. Búsqueda de artefactos	Muchas veces es difícil encontrar a la persona con el nivel adecuado de *expertise*.
3. Búsqueda de expertos	En algunos casos no se sabe si el *expertise* o el experto está disponible para la persona que lo está buscando.
4. Acceso al conocimiento	En algunos casos se pierde mucho tiempo en la búsqueda de *expertise* por qué no se cuenta con el conocimiento de donde se encuentra o quienes son los proveedores.

4.2.2 Diseño de Requerimientos

Con la información obtenida a partir de la metodología de KoFI, fue posible para obtener los requisitos del sistema para la localización de expertos (ver la Tabla 4.5). Estos requisitos, serán usados para diseñar un modelo de una arquitectura basada en agentes.

Tabla 4.5 Requerimientos arquitectura basada en agentes

Requerimientos
R1. El sistema tendrá una interfaz donde podrá hacer las consultas de información y dar de alta nuevo conocimiento.
R2. El sistema se encargara de capturar el conocimiento nuevo en los repositorios y realizara las búsquedas de los usuarios.
R3. El sistema se encargara de tomar decisiones para encontrar los mejores recursos para el usuario de acuerdo a sus necesidades.
R4. El sistema se encargara de hacer el cálculo de los posibles expertos que tengan el conocimiento y la disponibilidad para proveer de conocimiento al usuario.

4.2.3 Diseño arquitectura basada en agentes para la localización del expertise

Con el propósito de dar apoyo a la localización del *expertise* en el desarrollo de software, se diseñó una arquitectura basada en agentes para dar soporte a la localización del *expertise*, proporcionando el mejor nivel de conocimiento que existe en la organización dada una petición del usuario. En esta sección, se presenta una arquitectura basada en agentes (ver Figura 4.2), que consta de dos tipos de agentes: Agente de Usuario (UA) y el agente de búsqueda central (CSA).

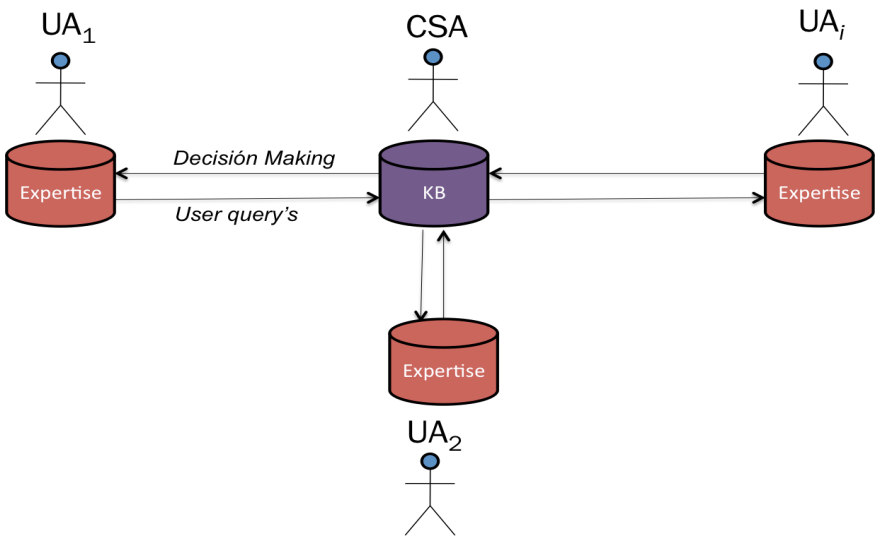

Figura 4.2 Arquitectura de agentes para la localización del *expertise*.

El agente de usuario (UA) es responsable de la gestión del conocimiento individual de los usuarios registrados; cada usuario almacenar firmado sus conocimientos. El agente UA crea un perfil basado en el conocimiento registrado por el usuario, permitiendo la búsqueda de conocimiento en los repositorios de los otros usuarios registrados. Hay i agentes registrados en el sistema, donde i es el número de usuarios. Se supone que es un ambiente cooperativo; Por lo tanto, todos los

usuarios están dispuestos a compartir sus conocimientos con el resto de los usuarios registrados.

El agente CSA se comunica con cada Agente UA. Este agente actúa como un experto en estrategias de localización experiencia en el desarrollo de software. Cuando un usuario realiza una consulta, recibe esa consulta y busca coincidencias entre la consulta y el conocimiento de los usuarios de la UA_i.

Es importante definir una clasificación del conocimiento en un sistema de colaboración para el intercambio de conocimiento (artefactos y expertos) en el desarrollo de software, para esto se diseño la siguiente clasificación de conocimiento para apoyar a la arquitectura presentada, en la cual el conocimiento esta organizado de la siguiente manera (ver Figura 4.3). El conocimiento se clasifica en cuatro niveles, específicamente en el dominio de la programación en una organización de desarrollo software.

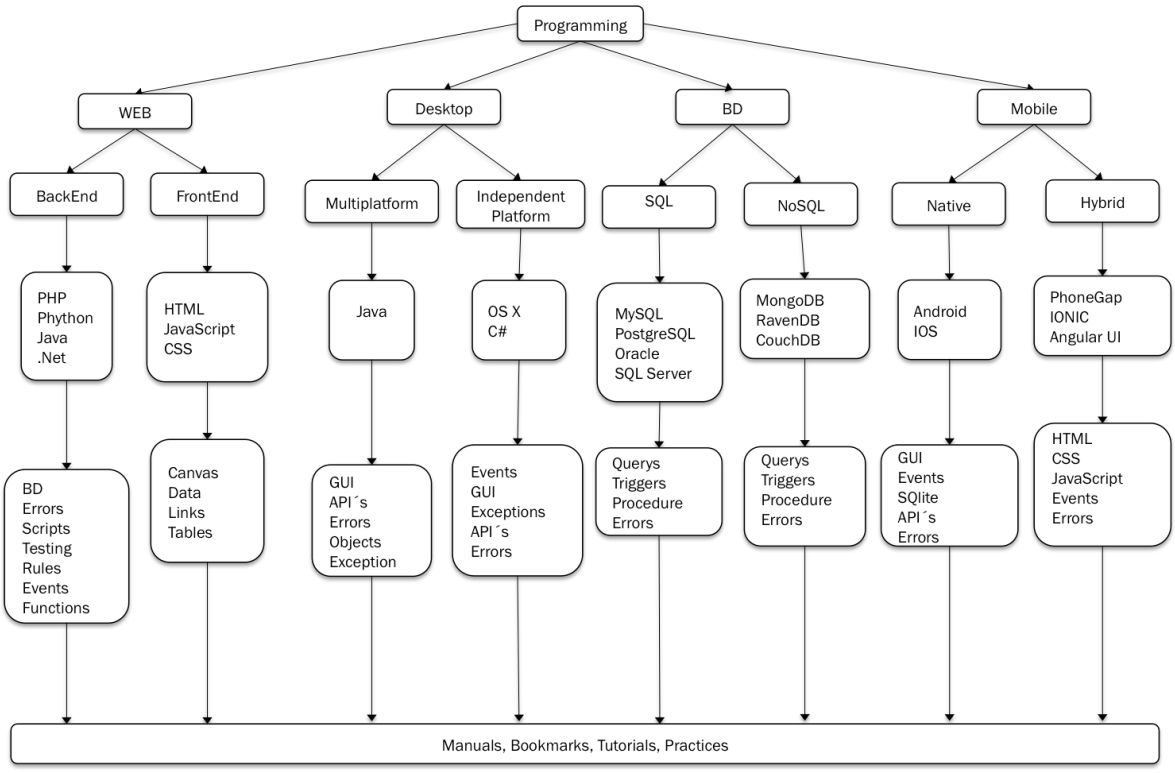

Figura 4.3 Clasificación del conocimiento en el dominio de programación

El primer nivel de la clasificación se relaciona con las plataformas que se pueden utilizar para desarrollar un proyecto (por ejemplo, Web, móvil, de escritorio, de base de datos), y luego el segundo nivel es el tipo de programación de acuerdo a la plataforma seleccionada (por ejemplo, backend, frontend), el tercer nivel es el lenguaje que se utiliza en la plataforma seleccionada (por ejemplo, Java, Python, PHP, etc.), el cuarto nivel es el tema relacionado con los conocimientos (palabras clave) y el último nivel es el formato en el que se representa el conocimiento (por ejemplo, manuales, tutoriales en vídeo, código fuente).

4.3 Modelado

Con el propósito de ayudar a los agentes de la arquitectura presentada a cumplir sus objetivos fue necesario modelar un formalismo para describir los mecanismos que utilizan los agentes. Este formalismo consiste en un conjunto de tuplas de la forma:

$$\mathcal{K} <\mathcal{P}, \mathcal{T}, \mathcal{L}, S, \mathcal{F}>$$

$\mathcal{P}$ = Platform where $\mathcal{P} \in$ {web, escritorio, basde de datos, movil}
$\mathcal{T}$ =Type related to the platform (e.g., $\mathcal{P}$=web) $\rightarrow$ $\mathcal{T} \in$ {backend, frontend}
$\mathcal{L}$ = Language (e.g., $\mathcal{P}$=web, $\mathcal{T}$=web) $\rightarrow$ $\mathcal{L} \in$ {Java,PHP,Python,..}
S = Subject (e.g., $\mathcal{P}$=web, $\mathcal{T}$=web, $\mathcal{L}$=PHP) $\rightarrow$ $S \in$ {GUI, Scripts, Events,..}
$\mathcal{F}$ = Format in wich knowledge is represented F $\in$ {manuals, source code, bookmarks}

Los mecanismos contribuyen a construir un perfil de usuario y con la búsqueda del conocimiento, a la vez por el uso de estos mecanismos es posible construir nuevo conocimiento basado en el perfil de usuario.

REFERENCIAS

Abidi, Syed Sibte Raza. (2001). Knowledge management in healthcare: towards 'knowledge-driven'decision-support services. *International Journal of Medical Informatics, 63*(1), 5-18.

Alavi, Maryam, & Leidner, Dorothy E. (2001). Review: Knowledge management and knowledge management systems: Conceptual foundations and research issues. *MIS quarterly*, 107-136.

Ammar-Khodja, S, & Bernard, Alain. (2008). An overview on knowledge management *Methods and tools for effective knowledge life-cycle-management* (pp. 3-21): Springer.

Barceló, Aurora Vizcaíno, Rubio, Félix O. García, & Velthuis, Mario Piattini. (2014). *Desarrollo Global de Software* (RA-MA Editorial ed.). España.

Basili, Victor R, & Caldiera, Gianluigi. (1995). Improve Soft-ware Quality by Reusing Knowledge and Experience. *Sloan management review, 37*, 55-55.

Basili, Victor R, Caldiera, Gianluigi, & Rombach, H Dieter. (1994). Experience factory. *Encyclopedia of software engineering*.

Basili, Victor R, & Perricone, Barry T. (1984). Software errors and complexity: an empirical investigation. *Communications of the ACM, 27*(1), 42-52.

Becerra-Fernandez, Irma, & Sabherwal, Rajiv. (2010). *Knowledge management: systems and processes*: ME Sharpe.

Bontis, Nick. (2001). Assessing knowledge assets: a review of the models used to measure intellectual capital. *International Journal of Management Reviews, 3*(1), 41-60.

Brandt, Joel, Dontcheva, Mira, Weskamp, Marcos, & Klemmer, Scott R. (2010). *Example-centric programming: integrating web search into the development environment.* Paper presented at the SIGCHI Conference on Human Factors in Computing Systems.

Chatterjee, Shaunak, Juvekar, Sudeep, & Sen, Koushik. (2009). Sniff: A search engine for java using free-form queries *Fundamental Approaches to Software Engineering* (pp. 385-400): Springer.

Cockburn, Andy, & McKenzie, Bruce. (2001). What do Web users do? An empirical analysis of Web use. *International Journal of human-computer studies, 54*(6), 903-922.

Colmenar Santos, Antonio, Gil, Castro, Manuel, A, Martínez, Elio Pérez, de LLano, Julio Vara, Castallón, Alfonso Cruz, & Javier, Francisco. (2007). Gestión de proyectos con Microsoft Project 2007. *España, Alfaomega*.

Dagenais, Barthélémy, & Robillard, Martin P. (2010). *Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors.* Paper presented at the International Symposium on Foundations of Software Engineering.

Dekel, Uri, & Herbsleb, James D. (2009). *Improving API Documentation Usability with Knowledge Pushing.* Paper presented at the 31st International Conference on Software Engineering

Ericsson, K Anders, Prietula, Michael J, & Cokely, Edward T. (2007). The making of an expert. *Harvard business review, 85*(7/8), 114.

Fragouli, Evangelia. (2015). Intellectual Capital & Organizational Advantage: an economic approach to its valuation and measurement. *Business and Management, 7*(1), 36-57.

Grechanik, Mark, Fu, Chen, Xie, Qing, McMillan, Collin, Poshyvanyk, Denys, & Cumby, Chad. (2010). *A search engine for finding highly relevant applications.* Paper presented at the Software Engineering, 2010 ACM/IEEE 32nd International Conference on.

Hayes-Roth, Frederick, Waterman, Donald, & Lenat, Douglas. (1984). Building expert systems.

Herbsleb, James D, & Moitra, Deependra. (2001). Global software development. *Software, IEEE, 18*(2), 16-20.

Jacobson, Ivar, Booch, Grady, & Rumbaugh, James. (2000). *El proceso unificado de desarrollo de software* (Vol. 7): Addison Wesley Reading.

Jacobson, Ivar, Booch, Grady, Rumbaugh, James, Rumbaugh, James, & Booch, Grady. (1999). *The unified software development process* (Vol. 1): Addison-wesley Reading.

Jain, Rituraj. (2011). Improvement in Software Development Process and Software Product through Knowledge Management. *International Journal of Computer Technology and Applications, 2*(05), 1557-1562.

Jensen, Michael C, & Meckling, William H. (1992). Specific and general knowledge and organizational structure.

Jensen, Rasmus Eskild. (2012). *Communication breakdowns in global software development teams: is knowledge creation the answer?* Paper presented at the 17th ACM international conference on Supporting group work.

Johansson, Conny, Hall, Patrik, & Coquard, Michael. (2000). “Talk to paula and peter—They are Experienced” The Experience Engine in a Nutshell *Learning Software Organizations* (pp. 171-185): Springer.

Jones, Gary, & Sallis, Edward. (2013). *Knowledge management in education: Enhancing learning & education*: Routledge.

Jones, William, Dumais, Susan, & Bruce, Harry. (2002). Once found, what then? a study of “keeping” behaviors in the personal use of web information. *Proceedings of the American Society for Information Science and Technology, 39*(1), 391-402.

Kautz, Karlheinz, & Thaysen, Kim. (2001). Knowledge, learning and IT support in a small software company. *Journal of knowledge management, 5*(4), 349-357.

Kogut, Bruce, & Zander, Udo. (1992). Knowledge of the firm, combinative capabilities, and the replication of technology. *Organization science, 3*(3), 383-397.

Loeliger, Jon, & McCullough, Matthew. (2012). *Version Control with Git: Powerful tools and techniques for collaborative software development*: " O'Reilly Media, Inc.".

McDonald, David W, & Ackerman, Mark S. (2000). *Expertise Recommender: A Flexible Recommendation System and Architecture.* Paper presented at the ACM Conference on Computer Supported Cooperative Work.

Nicolini, Davide, Powell, John, Conville, Paul, & Martinez−Solano, Laura. (2008). Managing knowledge in the healthcare sector. A review. *International Journal of Management Reviews, 10*(3), 245-263.

Nielsen, Peter Axel, & Kautz, Karlheinz. (2008). *Software Processes & Knowledge*: Software Innovation.
Nonaka, Ikujiro, & Takeuchi, Hirotaka. (1995). *The knowledge-creating company: How Japanese companies create the dynamics of innovation*: Oxford University Press.
P&MS, U.N.I. (2000). Intellectual Capital, People First in the Information Age Economy *World Conference* (pp. 21-23). Singapore.
Petrides, Lisa A, & Nodine, Thad R. (2003). Knowledge Management in Education: Defining the Landscape.
Pilato, C Michael, Collins-Sussman, Ben, & Fitzpatrick, Brian W. (2008). *Version control with subversion*: " O'Reilly Media, Inc.".
Polanyi, Michael. (1967). The tacit dimension.
Ponelis, Shana, & Fairer-Wessels, Felicite A. (2014). Knowledge management: A literature overview. *South African Journal of Libraries and Information Science, 66*(1).
Pressman, Roger S. (2009). *Software Engineering: A Practitioner's Approach, 7/e*: Mc Graw-Hill.
Prikladnicki, Rafael, Damian, Daniela, & Audy, Jorge Lis Nicolas. (2008). *Patterns of evolution in the practice of distributed software development: quantitative results from a systematic review.* Paper presented at the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE).
Prusak, Laurence. (2001). Where did knowledge management come from? *IBM systems journal, 40*(4), 1002-1007.
Rodríguez, Oscar M, Vizcaíno, Aurora, Martínez, Ana I, Piattini, Mario, & Favela, Jesús. (2004). *Using a multi-agent architecture to manage knowledge in the software maintenance process*. Paper presented at the Knowledge-Based Intelligent Information and Engineering Systems.
Rodríguez-Elias, Oscar M, Vizcaíno, Aurora, Martínez-García, Ana I, Favela, Jesús, & Piattini, Mario. (2009). Knowledge Flow Identification. *Encyclopedia of Information Science and Technology*, 2337-2342.
Rus, Ioana, & Lindvall, Mikael. (2002). Guest editors' introduction: Knowledge management in software engineering. *IEEE software, 19*(3), 26-38.
Sarkan, Hasliza Md, Ahmad, Tengku Puteri Suhilah, & Bakar, Afarulrazi Abu. (2011). *Using JIRA and Redmine in requirement development for agile methodology.* Paper presented at the 5th Malaysian Conference in Software Engineering (MySEC), Johor Bahru, Malaysia.
Schach, Stephen R, Fernández, Esther, Guerrero, Ekaterina, Ramírez, Raúl Antonio Trejo, & Betancourt, Saturnina Teodora Juárez. (2006). *Ingeniería de software clásica y orientada a objetos*: McGraw-Hill.
Schwaber, Ken. (2004). *Agile project management with Scrum*: Microsoft Press.
Serban, Andreea M, & Luan, Jing. (2002). Overview of knowledge management. *New Directions for Institutional Research, 2002*(113), 5-16.
Van Vliet, Hans. (2010). *Knowledge sharing in software development.* Paper presented at the 10th International Conference on Quality Software (QSIC).
Vasilescu, Bogdan, Filkov, Vladimir, & Serebrenik, Alexander. (2013). *StackOverflow and GitHub: Associations Between Software Development*

and Crowdsourced Knowledge. Paper presented at the International Conference on Social Computing (SocialCom).
Vasilescu, Bogdan, Serebrenik, Alexander, Devanbu, Prem, & Filkov, Vladimir. (2014). *How social Q&A sites are changing knowledge sharing in open source software communities.* Paper presented at the 17th ACM conference on Computer supported cooperative work & social computing.
Vivacqua, A. (1999). *Agents for expertise location.* Paper presented at the AAAI Spring Symposium Workshop on Intelligent Agents in Cyberspace.
Wang, Sheng, Noe, Raymond A, & Wang, Zhong-Ming. (2014). Motivating Knowledge Sharing in Knowledge Management Systems A Quasi–Field Experiment. *Journal of Management, 40*(4), 978-1009.
Westland, J Christopher. (2002). The cost of errors in software development: evidence from industry. *Journal of Systems and Software, 62*(1), 1-9.
Wooldridge, Michael, & Jennings, Nicholas R. (1995). Intelligent agents: Theory and practice. *The knowledge engineering review, 10*(02), 115-152.
Yilmaz, Murat, O'Connor, Rory V, & Clarke, Paul. (2015). Software development roles: a multi-project empirical investigation. *ACM SIGSOFT Software Engineering Notes, 40*(1), 1-5.
Yilmaz, Murat, O'Connor, Rory V, & Clarke, Paul. (2012). A systematic approach to the comparison of roles in the software development processes *Software Process Improvement and Capability Determination* (pp. 198-209): Springer.
Zhang, Jun, Ackerman, Mark S, Adamic, Lada, & Nam, Kevin Kyung. (2007). *QuME: a mechanism to support expertise finding in online help-seeking communities.* Paper presented at the 20th annual ACM symposium on User interface software and technology.