ISSN 2007-9737

Modelo de condensación de conocimiento para soporte a la localización del expertise en el desarrollo de software

Jose Ramón Martínez-Garcia[1], Ramón René Palacio[1],
Francisco Edgar Castillo-Barrera[2], Juan Carlos Cuevas-Tello[2], Gilberto Borrego[1]

[1] Instituto Tecnológico de Sonora,
México

[2] Universidad Autónoma de San Luis Potosí,
México

joseramonmg26@gmail.com, {ramon.palacio,
gilberto.borrego}@itson.edu.mx, {ecastillo, cuevas}@uaslp.mx

Resumen. La localización del expertise una necesidad crítica en las organizaciones de software y es utilizado para resolver necesidades de conocimiento. Los desarrolladores acumulan expertise de las tecnologías utlizadas y los problemas resueltos durante los proyectos. Actualmente las organizaciones no se benefician de este, saber como modelar el expertise permitir a las organizaciones acceder a mejores prácticas y soluciones a problemas. Este trabajo presenta un modelo que implementa el concepto de condensación de conocimiento; el cual tiene como objetivo clasificar, recuperar y compartir conocimientos valiosos entre los interesados de una manera que facilite su recuperación. El modelo consta de tres módulos principales: gramática formal, conocimiento semántico y herramientas de experiencia. En el módulo de gramática formal, se presenta un formalismo para describir cómo los desarrolladores almacenan y comparten su conocimiento. En el modulo de conocimiento semántico, se propone un modelo de conocimiento arquitectónico. Finalmente, en el módulo de herramientas de expertise se presentan dos prototipos que implementan los elementos del modulo de conocimiento semántico.

Palabras clave. Desarrollo de software, representación de conocimiento, conocimiento arquitectónico.

Knowledge Condensation Model for Support of Expertise Location in Software Development

Abstract. To support their knowledge needs, finding relevant expertise is a critical need in software organizations. Developers use it and accumulate expertise from the technologies they use and the problems solved within projects. Currently, members of the organizations have yet to benefit from this expertise. Learning how to model it might enable access to the best practices and problem-solved information. This work presents a model that implements the knowledge condensation concept to capture expertise during the software development process. The aim is to classify, retrieve, and share valuable knowledge among stakeholders in an unsuitable form for its recovery. The model consists of three modules: formal grammar, semantic knowledge, and expertise tools. The formal grammar module presents an approach to formalize how developers store and share their knowledge. The semantic knowledge module presents an architectural knowledge model. Finally, we presented two prototypes in the expertise tools module that use the semantic knowledge module elements.

Keywords. Knowledge representation, software development, architectural knowledge.

1. Introducción

El conocimiento en las organizaciones de software juega un rol crucial en la mejora y éxito del proceso de desarrollo de software, pues este consiste en una serie de actividades que demandan la aplicación de representaciones de conocimiento para su comprensión y ejecución [23, 36]. Una representación de conocimiento puede involucrar distintas situaciones: (1) en algunas ocasiones durante un proyecto un desarrollador puede llegar a utilizar tecnologías nuevas con las que no tiene tanta experiencia, de modo que necesitan prepararse para utilizarlas [32]; (2) durante los proyectos, regularmente los desarrolladores enfrentarán situaciones donde se necesita tomar decisiones: técnicas y administrativas [1, 47]; (3) frecuentemente los desarrolladores tendrán que resolver distintos tipos de problemáticas (e.g., código, documentación, requerimientos, errores de diseño y arquitectura) [14].

Eventualmente debido a estas situaciones, el conocimiento del desarrollador puede no ser suficiente para salir adelante; lo cual los lleva efectuar el proceso de localización de expertise; el cual es el proceso de encontrar conocimiento de alto nivel (persona o recurso) adecuado para resolver alguna de las situaciones antes mencionadas [41, 16].

En las organizaciones de software el expertise puede encontrarse en el conocimiento arquitectónico (AK, por sus siglas en inglés); este consiste en los elementos empleados para construir un diseño arquitectónico (requerimientos, diseños, diagramas y componentes), decisiones de diseño y el razonamiento utilizado para obtener una solución arquitectónica [10].

AK incorpora la experiencia y el razonamiento de problemas de los recursos humanos a través de dos fuentes primarias: artefactos y expertos. Los artefactos denotan documentos físicos y digitales (e.g., documento de requisitos o visión) y código fuente [20, 37], mientras que los expertos son desarrolladores especializados en resolver algún tipo de problema durante el proceso desarrollo de software [16, 35].

Cada ocasión en que un desarrollador lleva a cabo el proceso de localización del expertise genera una acumulación de conocimiento de alto nivel (tecnologías nuevas, decisiones tomadas o problemas resueltos).

Es por esto que este se ha vuelto un recurso valioso en las organizaciones de software, Mohagheghi & Conradi [31] observaron que la reutilización de conocimiento incrementa la productividad y la calidad del proceso de desarrollo de software.

Sin embargo, las características de los paradigmas nuevos como el ágil dificultan la reutilización del conocimiento; este paradigma es muy utilizado hoy en día y esta enfocado en el conocimiento tácito [33].

En su manifiesto[1], el paradigma ágil establece la comunicación directa entre desarrolladores como la manera más efectiva de compartir conocimiento; también establece que solo se documenta la información que el equipo de desarrollo considere importante o suficiente para el producto a desarrollar.

Como resultado, una gran cantidad de conocimiento tácito permanece sin explotar; este conocimiento representa la experiencia acumulada de los desarrolladores y es propenso a evaporarse con el tiempo.

La vaporización del conocimiento se define como la perdida de AK debido a su poca documentación durante los proyectos de software; adicionalmente involucra la falta de expertise cuando un desarrollador renuncia, abandona el proyecto o no se encuentra disponible [6].

Lo anterior resalta a la localización del expertise como una tarea importante que se realiza durante los proyectos de desarrollo de software; por lo tanto, saber cómo modelar la experiencia de forma explícita permitirá aprovechar las mejores prácticas, el conocimiento de capacitación y los datos en múltiples aplicaciones en la organización.

Como resultado, podría ayudar a evitar tres problemas recurrentes en el desarrollo de software [18, 45]: a) pérdida de tiempo respondiendo y encontrando soluciones a problemas ya resueltos; b) falta de visibilidad de las soluciones técnicas

[1] https://agilemanifesto.org/

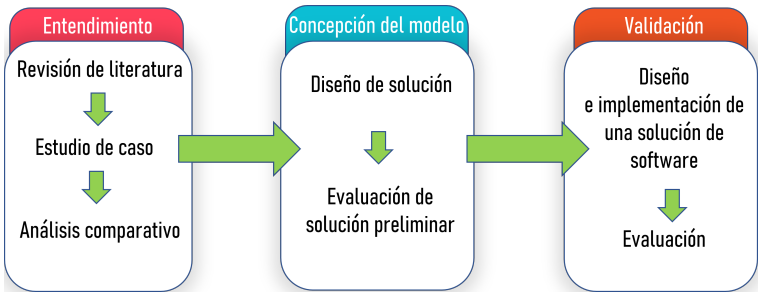

Fig. 1. Metodología seguida durante el proceso de investigación

Fig. 2. Método para investigar la localización del expertise en el desarrollo de software

y c) pérdida de conocimiento debido a la falta de disponibilidad de los desarrolladores.

Las organizaciones deben desarrollar enfoques formales para condensar el conocimiento de diferentes fuentes de manera sistemática. Borrego et al. [6] establece la condensación del conocimiento como el proceso de capturar y clasificar el expertise antes de que se pierda; el objetivo principal es facilitar la recuperación del expertise acumulado durante los proyectos de software [35].

1.1. Planteamiento del problema

Mucha investigación en los últimos años se ha centrado en la gestión del expertise; particularmente en el proceso de externalización, el cual consiste en una transición del conocimiento tácito al explícito [11]. Las propuestas externalizan el expertise de alguna de las dos fuentes principales: artefacto y expertos.

La propuestas enfocadas en artefactos externalizan el expertise generado por los desarrolladores durante un proyecto de desarrollo de software; actualmente estas se centran en el código fuente, el objetivo es apoyar a los programadores con fragmentos de código de fuentes como StackOverflow [34, 29, 8].

Los medios textuales y electrónico no estructurados (UTEM, por sus siglas en inglés) es otro tipo de artefacto explotado en la literatura; estos son utilizados para complementar y contextualizar documentos informales [7].

Por último, los investigadores utilizan los diagramas del lenguaje de modelado unificado (UML, por sus siglas en inglés), como los casos de uso, para almacenar datos de diseño arquitectónico [4].

Además de eso, los investigadores proponen utilizar expertos como fuente de experiencia en el desarrollo de software; estas propuestas apoyan tareas como el manejo de un informe de error, el diseño del sistema y la resolución de problemas [1, 28, 22, 30].

Aunque varios investigadores han presentado propuestas para gestionar el expertise, aún existen aspectos a mejorar.

Las propuestas de artefactos explotan la experiencia de los desarrolladores durante el proceso de desarrollo de software con dos limitaciones características: (1) un enfoque estrecho en el código fuente; (2) acumulación

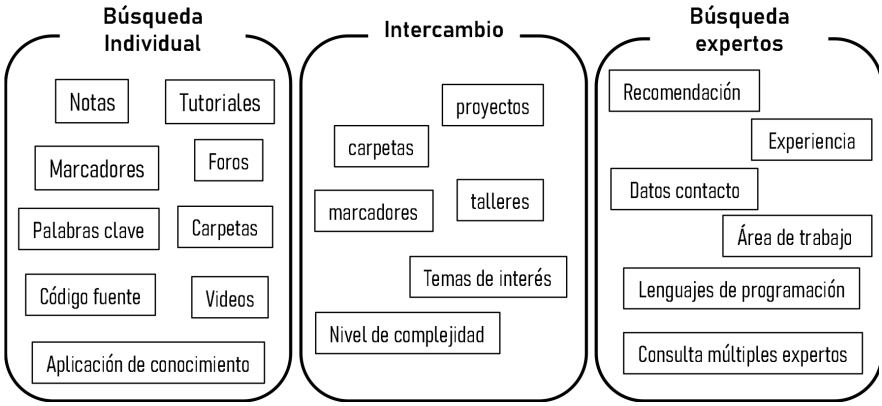

Fig. 3. Diagrama de afinidad con los resultados del grupo focal

informal de conocimientos inadvertidos por las organizaciones.

Por otro lado, las propuestas que utilizan a expertos como fuente de expertise tiene inconvenientes debido a que estas propuestas no otorgan acceso a los artefactos producidos por los desarrolladores; por lo tanto, la disponibilidad de los desarrolladores limita el acceso al expertise.

Las propuestas actuales tienen algunos desafíos relevantes que abordar: (1) otorgar acceso tanto a los individuos con experiencia particular como a sus artefactos, este acceso asegurará la disponibilidad de los recursos necesarios incluso si el proveedor no está disponible, y reducirá la erosión de las relaciones interpersonales; (2) incorporar más fuentes de expertise además del código fuente, ya que existen otras fuentes digitales (e.g., marcadores, libros, manuales y tutoriales).

1.2. Objetivos de la investigación

Desarrollar y validar un modelo de condensación de conocimiento para capturar el expertise de los desarrolladores durante el proceso de desarrollo de software. Como objetivos particulares:

1. Conocer cómo llevan a cabo los desarrolladores el proceso de localización del expertise y qué fuentes utilizan.
2. Construir una ontología para capturar y recuperar expertise en la fase de codificación durante el proceso de desarrollo de software.
3. Diseñar e implementar la aplicación ExCap y la extensión B4U para capturar y clasificar el expertise.
4. Validar el modelo de condensación de conocimiento.

1.3. Metodología

El presente trabajo de investigación siguió una metodología de investigación que consta de tres fases: comprensión, concepción del modelo y validación del modelo.

Durante la fase de comprensión, se realizaron actividades para comprender el proceso de localización del expertise y las fuentes que utilizan los desarrolladores.

En la etapa de concepción se propuso un diseño inicial para un modelo de condensación de conocimiento.

Finalmente, en la fase de validación del modelo, se desarrolló y evaluó un mecanismo para el modulo de herramientas de expertise (ver Figura 1).

1. **Revisión de la literatura**: el propósito de esta actividad fue redactar el marco teórico e identificar los trabajos anteriores que abordan el mismo problema.

Tabla 1. Fuentes de conocimiento utilizadas por los desarrolladores durante la localización del expertise

Fuentes de conocimiento	Viana et al. [47]	Viana et al. [46]	Josyula et al. [21]	Grupo Focal	Li et al. [25]	Borrego et al. [5]	Ali [3]	LaToza et al.[24]
Colegas			✓	✓				✓
Herramientas de gestión	✓							
Blogs			✓		✓			
Foros			✓	✓	✓			
Tutoriales				✓	✓			
Cursos			✓					
Libros		✓	✓					
Proyectos previos		✓	✓	✓			✓	
Redes sociales			✓					
Documentación Oficial				✓	✓			
Sitios de código		✓	✓		✓			
Marcadores				✓				
UTEMS						✓		

Además, de acuerdo con la literatura revisada, encontramos ideas para respaldar la localización del expertise en función de lo que han hecho los investigadores.

2. **Estudio de caso**: la actividad de estudio de caso tenía el propósito de complementar la literatura y comprender la localización del expertise directamente de la experiencia de los desarrolladores de software.
3. **Análisis comparativo**: La siguiente actividad analizó estudios similares para identificar una amplia gama de problemas y fuentes de desarrolladores durante la localización del expertise.
4. **Diseño de solución**: La actividad consistió en diseñar un modelo de condensación de conocimiento para apoyar la localización del expertise; el cual utiliza los resultados del estudio de caso y la revisión de la literatura.
5. **Evaluación preliminar de la solución tecnológica**: se desarrolló y evaluó una ontología durante la implementación del módulo de conocimiento semántico en el modelo de condensación de conocimiento.
6. **Diseñar e implementar una solución de software**: esta actividad consistió en diseñar e implementar una solución de software para capturar y clasificar la experiencia.

   La solución de software integra elementos de la ontología desarrollada para el módulo de conocimiento semántico).
7. **Evaluación**: Esta actividad comprendió los procesos de evaluación del prototipo de la solución de software y evaluación del modelo de condensación de conocimiento.

2. Trabajos relacionados

El concepto de condensación de conocimiento propuesto por Borrego et al. [6], describe el proceso de captura y clasificación de la experiencia antes de que se pierda.

Durante la presenta investigación, se analizaron trabajos realizan externalización, el cual es el proceso de convertir conocimiento tácito a explicito.

Existe una amplia variedad de enfoques que buscan externalizar el expertise acumulado en

Tabla 2. Necesidades de conocimiento que llevan a los desarrolladores a realizar búsquedas de expertise

Necesidad de conocimiento	Josyula et al.	Grupo Focal	Borrego et al.	LaToza et al.	Viana et al.
Aprender a utilizar una herramienta	✓				
Mejorar habilidades de programación	✓	✓			
Resolver algún error/bug	✓	✓	✓	✓	✓
Codificar un modulo	✓				
Realizar una prueba de software	✓				
Diseño de producto o arquitectura	✓	✓			
Información Técnica			✓		

las organizaciones de software, investigadores de diferentes tecnologías y modelos para apoyar el proceso de localización de la experiencia.

Algunos de los principales temas explorados por los investigadores para condensar el conocimiento son ontologías y sistemas (herramientas y otras aplicaciones). A continuación se describen algunas de las propuestas identificadas en la literatura.

2.1. Ontologías

En la literatura actual, podemos encontrar una variedad de enfoques en ingeniería de software, los cuales emplean ontologías [2]: ingeniería de requisitos, reutilización de componentes, documentación de código y soporte en la codificación.

El objetivo principal de las ontologías propuestas es aumentar el reúso de conocimiento arquitectónico (AK, por sus siglas en inglés) a través de una descripción y organización de este.

Los requisitos especificados mediante ontologías son adecuados para herencia, extensibilidad, uso compartido y reutilización [26, 12]. Además, podemos utilizarlo como herramienta de comunicación entre diferentes stakeholders.

Los investigadores también han propuesto ontologías para la reutilización de componentes [17]. Los componentes son paquetes de software, servicios web o un módulo que encapsula conjunto de funciones relacionadas (o de datos) [9, 40].

Los desarrolladores suelen reutilizar componentes de software al implementar una funcionalidad.

Las ontologías proporcionan una forma de describir la funcionalidad de los componentes que permitirá consultas más amplias puesto que utilizarán la descripción semántica de cada uno de los componentes (e.g., lenguaje de programación, proyecto, librerías, dominio).

La documentación de código y el soporte de codificación son otras áreas del proceso de desarrollo de software en el dominio abordado mediante ontologías [13, 39, 44].

Los desarrolladores manejan muchos frameworks y bibliotecas, a las que deben acceder a través de interfaces de programación de aplicaciones (API, por sus siglas en inglés) [15].

La ventaja de utilizar ontologías dentro del soporte de codificación es proporcionar una gran variedad de identificadores para los recursos relacionados a la codificación (palabras clave).

Además, el uso de ontologías en la documentación del código proporciona una representación unificada, es decir describir de una manera consistente los recursos (e.g., lenguaje, tema, funcionalidad, versión, autor) facilitará su recuperación.

2.2. Herramientas y otras aplicaciones

Existe una amplia variedad de enfoques para externalizar el conocimiento en la organización de software, el propósito de los investigadores es apoyar la localización del expertise a través de aplicaciones o sistemas que se centran en fuentes de AK.

Las propuestas actuales se centran únicamente en una fuente de AK en particular; con respecto a las propuestas de artefactos, el enfoque principal es el código fuente.

Por ejemplo, Ponzanelli et al. [34] presenta SeaHawk, un complemento de eclipse que ayuda a los programadores con fragmentos de código de Stack Overflow; además, SeaHawk utiliza anotaciones independientes del idioma para vincular documentos con fragmentos de código.

Del mismo modo, Brandt et al. [8] presenta BluePrint, una interfaz de búsqueda web integrada en el entorno de desarrollo de Adobe Flex Builder; Blueprint aumenta las consultas con el contexto del código a través de una vista centrada en el código de los resultados de búsqueda incrustados en el editor.

Además, McMillan et al. [29] presenta Portfolio, un sistema de búsqueda de código para ayudar a los desarrolladores; el sistema recupera funciones y describe su uso; además, utilizando una combinación de modelos, el sistema aborda el comportamiento de compartir funciones de los desarrolladores.

Aparte de eso, solo unas pocas obras utilizan otros tipos de artefactos; por ejemplo, Borrego et al. [7] presenta una herramienta de complementación a slack, que consiste en un mecanismo de clasificación basado en el etiquetado social; este mecanismo aprovecha el conocimiento arquitectónico de los medios electrónicos textuales y no estructurados (UTEM).

En paralelo, Bonilla-Morales [4] propone una herramienta para reutilizar diagramas de casos de uso; la herramienta permite almacenar información de diagramas de casos de uso.

Por otro lado, algunas propuestas utilizan expertos como fuente de AK, intentan concienciar a las organizaciones de su experiencia.

Por ejemplo, Bhat et al. [1] proponen un sistema de recomendación que identifica a los desarrolladores que podrían participar en el diseño de un sistema de software; el enfoque cuantifica las habilidades de los desarrolladores para hacer coincidir y recomendar un individuo adecuado a las necesidades del sistema a diseñar.

Alternativamente, Matter et al. [28] propone un modelo basado en vocabulario, que sugiere desarrolladores con la experiencia adecuada para manejar un informe de error; el modelo utiliza las contribuciones de los desarrolladores para sugerir a alguien.

Además, Kagdi et al. [22] presenta xFinder, una herramienta que recomienda expertos en función de su experiencia, que se mide utilizando las contribuciones de compromiso del desarrollador.

Además, Minto y Murphy et al. [30] proponen EEL, un Localizador de experiencia emergente, el cual se enfoca en facilitar el proceso de encontrar al experto adecuado para preguntar sobre un problema durante una tarea de desarrollo de software.

Las propuestas en la literatura afirman la importancia de la gestión del expertise en las organizaciones de desarrollo de software.

Estas propuestas distinguen entre artefactos y expertos como la principal fuente de experiencia para las organizaciones.

Las propuestas de artefactos aprovechan la experiencia generada por los desarrolladores durante el proceso de desarrollo de software con dos limitaciones características: un enfoque estrecho en el código fuente y una acumulación informal de conocimiento realizado por los desarrolladores pero desapercibido por las organizaciones.

Las propuestas de expertos, por otro lado, emplean a los desarrolladores como una fuente relevante de experiencia en el proceso de desarrollo de software; para realizar tareas como el manejo de un informe de error, el diseño del sistema y la resolución de problemas.

Si bien la evidencia empírica de las obras descritas respalda el valor de los expertos en ayudar, las propuestas dan poca importancia a los artefactos que generan o utilizan; además, las preguntas constantes a los

Tabla 3. Problemas comunes durante la localización del expertise

Problemas	Viana et al. [47]	Grupo Focal	Ali [3]	Borrego et al. [5]	LaToza et al.[24]	Viana et al.[46]
Transferencia de conocimiento		✓				✓
Interrupciones				✓	✓	
Captura de conocimiento	✓		✓	✓	✓	
Recuperación de conocimiento		✓		✓		

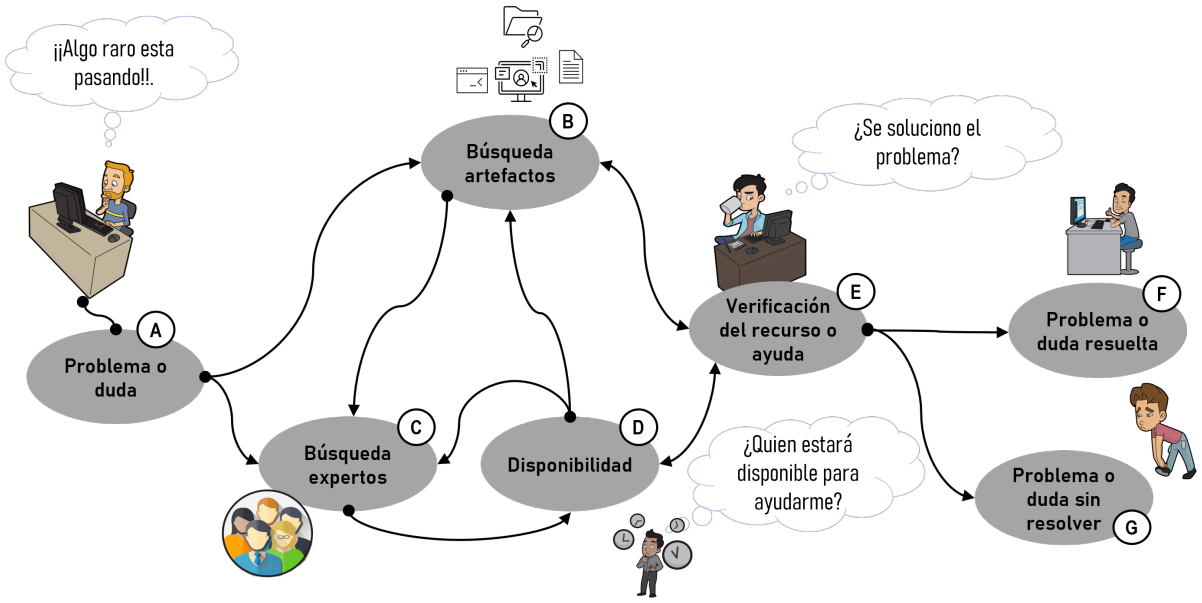

Fig. 4. Modelado del proceso de localización del expertise en el desarrollo de software

expertos podrían conducir a la erosión de las relaciones interpersonales.

3. Localización del expertise

La condensación de conocimiento en el desarrollo de software es un desafío ya que se requiere identificar las fuentes de conocimiento consultadas por los desarrolladores, además del proceso de captura e intercambio de estas.

Para abordar estos desafíos mencionados anteriormente, se realizó un estudio para comprender el proceso de localización del expertise en las organizaciones de software. El estudio consistió en tres etapas: grupo focal, modelado de la localización de expertos y un análisis comparativo (ver Figura 2).

3.1. Estudio grupo focal

En esta etapa se realizó un grupo focal, el cual tuvo una duración de una hora y media. El protocolo del grupo focal incluyó los siguientes temas:

1. **Búsqueda individual**: esto representa el proceso que siguen los desarrolladores para realizar la localización del expertise y una vez que la encuentran, se guarda la experiencia.

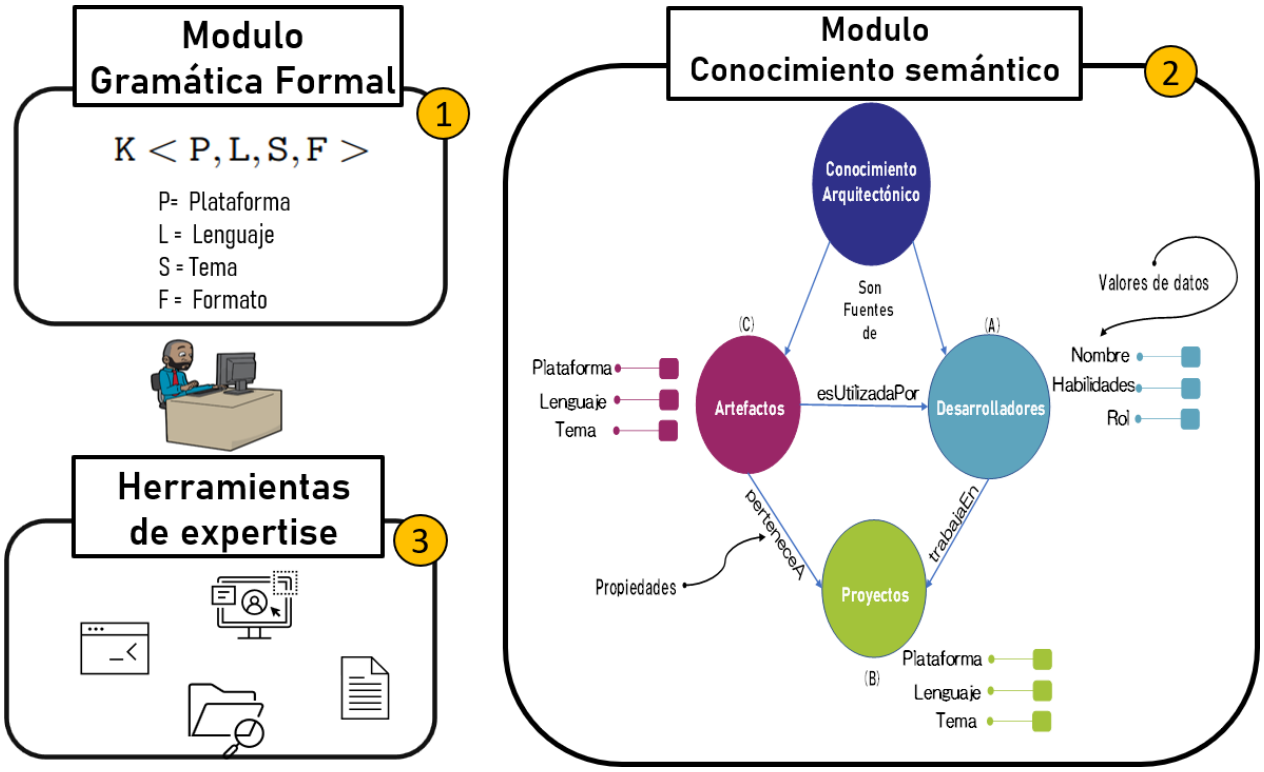

Fig. 5. Modelo de condensación de conocimiento

- ¿Cómo se busca o se encuentra el conocimiento?
- ¿Que se hace una vez que ya se tiene el conocimiento?

2. **Intercambio de conocimiento**: Esta categoría se centró en el proceso de transferencia de conocimientos cuando un desarrollador solicita ayuda.
   - ¿Cómo se transfieren conocimientos?
   - ¿Que medios utilizan para compartir conocimiento?
   - ¿Que tipo de conocimiento acostumbran intercambiar?
3. **Búsqueda de expertos**: Esta categoría se centró en el proceso de selección de un experto para pedir ayuda.
   - ¿Cómo identificas a un experto?
   - ¿Que características tiene un experto?
   - ¿Que separa a un programador normal de un experto?

3.1.1. Participantes

Para el grupo focal se invitaron a 5 desarrolladores de software, de una organización de software enfocada en proyectos de desarrollo web en México. La edad de los participantes osciló entre 26 y 36 años (edad media 29.45 años), con una experiencia laboral media de 4.72 años (min = 2; max = 9).

3.1.2. Análisis de resultados

Durante el análisis de datos, se utilizó un enfoque de teoría fundamentada [43], la cual es utilizada para identificar coincidencias en varios individuos mediante codificación abierta, axial y selectiva.

Tabla 4. Elementos para construir el perfil de un usuario

Elemento	Propiedades
Datos Personales	Nombre
	Rol
	Telefono
	Email
Historial de proyectos	Nombre del proyecto
	Plataforma
	Lenguaje
	Role
Contribuciones (tuplas)	Manuales
	Tutoriales
	Código fuente
	Documentación

En la codificación abierta, el grupo focal fue analizado y codificado frase por frase. A continuación, en la codificación axial, ideamos relaciones entre los códigos y las categorías establecidas.

Finalmente, mediante codificación selectiva, clasificamos las relaciones identificadas en las categorías mencionadas anteriormente.

Para la codificación selectiva, se utilizaron diagramas de afinidad (ver Figura 3), que es una herramienta que sintetiza un conjunto de datos verbales (por ejemplo, ideas, opiniones, expresiones), agrupándolos de acuerdo con la relación que tienen con las categorías.

En la Figura 3, se muestra el diagrama de actividad que clasifica los datos de las respuestas que aparecieron de manera más recurrente.

También se seleccionaron citas relevantes que crean una relación más fuerte entre los conceptos para enriquecer los resultados.

3.2. Análisis comparativo

Con el objetivo de comprender de una mejor manera el proceso de localización del expertise: fuentes, necesidades, problemas y flujo; se analizaron distintos trabajos encontrados literatura los cuales abordan objetivos similares.

Los investigadores han empleado diferentes métodos para obtener datos sobre la localización del expertise: estudios cualitativos y revisión de la literatura (encuestas).

Viana et al. realizó un estudio cualitativo basado en entrevistas y etnografía; su propósito fue comprender cómo se crean y mantienen las lecciones aprendidas en los proyectos de software; las lecciones aprendidas se utilizan en entornos ágiles para describir discusiones y situaciones para resolver problemas [47].

Además, Viana et al. llevó a cabo un estudio cualitativo de la transferencia de conocimiento en organizaciones pequeñas desde el punto de vista de un profesional de desarrolladores de software [46].

Ali [3] realizó una revisión de la literatura sobre evidencia de reutilización de software en ingeniería de software; el objetivo era identificar cómo los desarrolladores encuentran información confiable para tomar decisiones correctas e informadas.

Josyula et al. [21] tuvo como objetivo identificar las necesidades de información y los recursos utilizados por los desarrolladores en el desarrollo de software. En su trabajo, realizaron 17 entrevistas semiestructuradas (6 de ellas fueron entrevistas cara a cara y 11 a través de Skype).

Borrego et al. [5] presenta un estudio empírico para comprender la articulación AK en UTEM en equipos AGSD. El estudio consistió en una observación y exploración de sitio de una de cuatro empresas. Li et al. [25] llevo a cabo informe empírico con el objetivo de comprender cómo los desarrolladores buscan información y cuáles prefieren fuentes.

El estudio incluye dos experimentos humanos: 24 desarrolladores que realizan dos tareas típicas de desarrollo de software.

LaToza et al. [24] realizó dos encuestas y once entrevistas para comprender las herramientas, actividades y prácticas de los desarrolladores para administrar el conocimiento.

A continuación, se describen los resultados del análisis comparativo de los resultados de nuestro grupo focal con el trabajo relacionado en la literatura sobre el conocimiento: fuentes, necesidades y problemas.

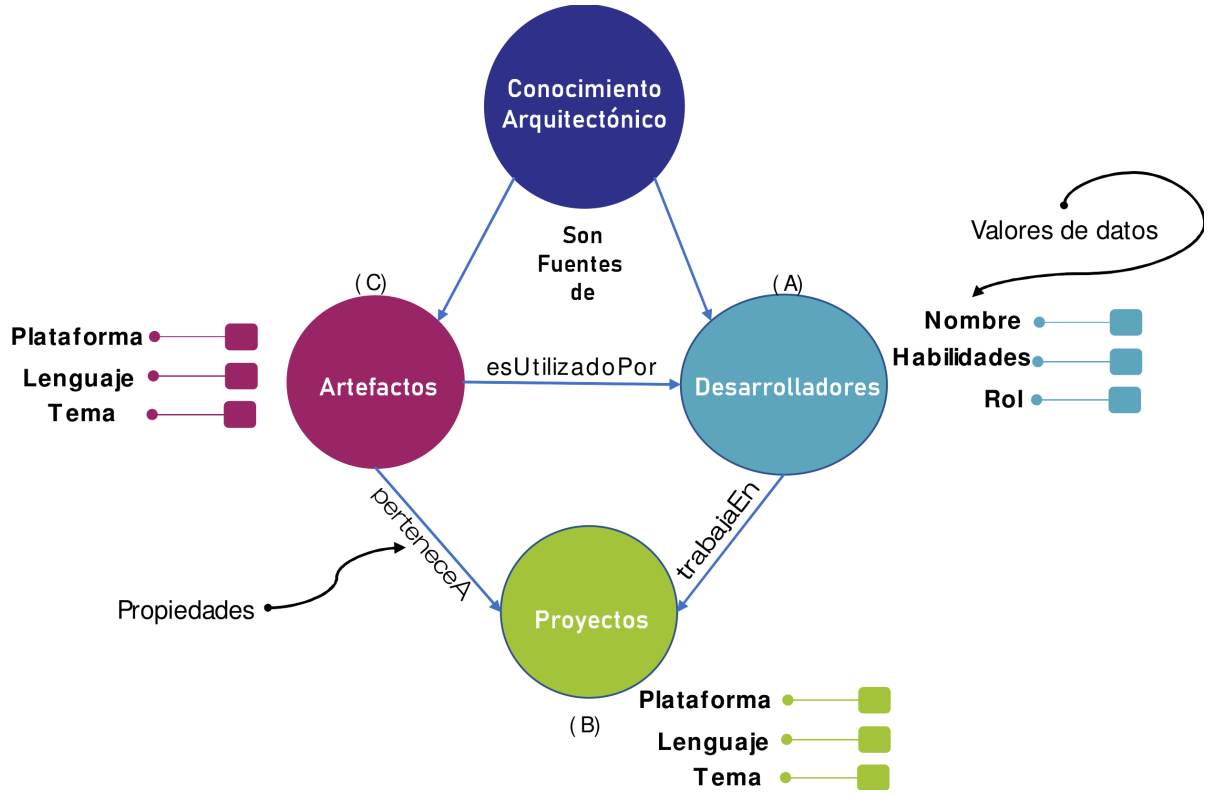

Fig. 6. Modelo semántico del conocimiento arquitectónico (AK)

3.2.1. Fuentes de conocimiento

En nuestro enfoque se identificó que los desarrolladores utilizan una fuente de conocimiento de acuerdo con la situación.

En la situación de entender cómo solucionar un error/bug los foros, tutoriales online y marcadores son las fuentes de conocimiento más comunes, "[Participante 4]: bueno, mi forma de investigar ... bueno, es similar... bueno, busco en foros, reviso proyectos y los que podrían ahorrarme tiempo los registro, y además guardo marcadores ", otra opción es la discusión con compañeros de trabajo," [Participante 5]: normalmente le pregunto a una persona (desarrollador), porque sé que tiene algunos conocimientos de acuerdo a lo que necesito".

Además, cuando se necesita mejorar las habilidades de programación en un lenguaje, la fuente más común es la documentación oficial "[Participante 1]: cuando empiezo a trabajar en un proyecto con nueva tecnología, primero busco todo en la documentación oficial, por ejemplo si es una tecnología que fue creada por una empresa como Google".

La Tabla 1 muestra las diferentes fuentes identificadas por los investigadores durante sus estudios. Las fuentes más comunes identificadas son compañeros de trabajo, sitios de códigos y proyectos anteriores.

3.2.2. Necesidades de conocimiento

En el desarrollo de software existe una gran variedad de necesidades de conocimiento, en los resultados del estudio se identificaron tres necesidades principales.

Los desarrolladores a veces necesitan comprender y resolver errores / errores "[Participante 1]: cuando tengo un error en mi

ISSN 2007-9737

código, busco ejemplos (por ejemplo, foros)". Otra necesidad surge cuando un desarrollador trabaja en un proyecto con nuevas tecnologías.

Finalmente, los desarrolladores necesitan el conocimiento para mejorar su experiencia "[Participante 4]: recientemente, nos ocupamos aprendiendo nuevos lenguajes de programación que nunca habíamos visto, y ni siquiera sabíamos que existían".

En la Tabla 2, se muestra la comparación de los resultados del grupo focal con los trabajos anteriormente mencionados.

La necesidad de conocimiento más común que presentan los desarrolladores es comprender y resolver errores; estos dedican una cantidad considerable de tiempo a este tipo de necesidades; esto significa que se genera una gran cantidad de conocimiento con soluciones que podrían evitar la repetición de errores durante un proyecto de desarrollo de software.

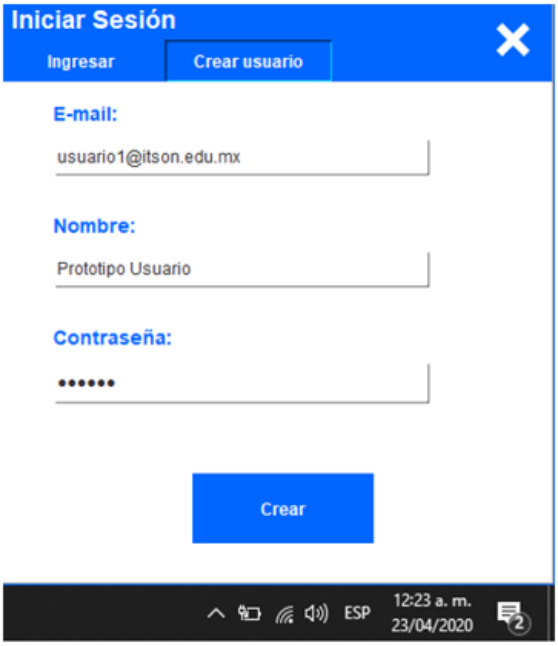

Fig. 7. Inicio de sesión ExCap

3.2.3. Problemas relacionados con el conocimiento

Durante el estudio se identificaron diferentes problemas asociados con el proceso de localización del expertise. En la Tabla 3, se muestran los problemas que surgen debido a las características ágiles de desarrollo de software.

Por ejemplo, una característica crucial es una fuerte comunicación sobre una extensa documentación. Durante el proceso de localización del expertise, los desarrolladores sufren la falta de documentación de los entornos de desarrollo ágiles.

Algunos de los problemas causados por esta falta de documentación son la captura y recuperación de conocimientos, las interrupciones de los compañeros de trabajo y la transferencia de conocimientos.

La captura de conocimiento en software es una tarea desafiante, ya que el conocimiento permanece tácito en su mayoría en entornos ágiles.

Borrego et al. [5] sustenta que en las herramientas de comunicación (por ejemplo, skype, Trello), podemos encontrar información importante para resolver problemas; sin embargo, tiende a perderse en las conversaciones con el tiempo ya que los desarrolladores no lo capturan.

Además, Viana et al. [47] informan que algunas lecciones no se aprenden durante el ciclo de vida de las lecciones aprendidas.

A veces, las lecciones no se aprenden durante la reunión porque los desarrolladores solo resumen el problema y la solución; una cantidad considerable de conocimiento permanece tácito ya que los desarrolladores solo discutieron informalmente durante las reuniones de SCRUM.

La recuperación de conocimientos implica diferentes problemas durante el proceso de localización del expertise.

Identificamos que los desarrolladores, a veces, los desarrolladores no encuentran el conocimiento previamente guardado "[Participante 1]: en el caso de los marcadores, a veces puede tener problemas para recordar las palabras clave o el tema del recurso".

El conocimiento de los desarrolladores no se puede recuperar a menos que alguien conoce las habilidades de los desarrolladores.

LaToza et al. [24] identificó que la comprensión de la lógica detrás del código es un desafío para los desarrolladores debido a la falta de documentación; esto los lleva a provocar interrupciones de los compañeros de trabajo para pedir consejo; con el tiempo, las interrupciones constantes pueden producir la erosión de las relaciones interpersonales.

3.3. Modelado de la localización del expertise

A partir de la información obtenida en la sección 3.1, mapeamos el proceso de localización del expertise, la Figura 4 presenta el flujo de conocimiento de la localización del expertise, usamos un escenario para ilustrar la localización del expertise en el desarrollo de software.

El escenario comienza con un desarrollador el cual su conocimiento no es suficiente para resolver una duda o problema durante alguna de sus actividades; esto lo lleva necesitar expertise (A); el desarrollador puede elegir entre una búsqueda expertos (C) o una búsqueda de artefactos (B); en el caso de elegir una búsqueda de artefactos (B), el desarrollador realiza una búsqueda inicialmente entre sus recursos de proyectos anteriores y después en caso de no tener éxito se procede a buscar en internet; posteriormente realiza una verificación del o los artefactos seleccionados para ver si se cumple el objetivo (E); si el desarrollador satisfizo su necesidad de expertise el problema se considera como resuelto (F).

Por el contrario, si no se satisface la necesidad de expertise, el desarrollador deberá verificar más artefactos para resolver el problema (A).

Por otro lado, en el caso de optar por una búsqueda experta (C), el desarrollador realiza una búsqueda buscando a alguien con los conocimientos para resolver el problema o duda, una vez que se encuentra la persona adecuada se debe verificar su disponibilidad (D).

Finalmente, si el experto cumplió con el objetivo, el problema está resuelto (E). De lo contrario, se necesita otra búsqueda experta (C). A veces, un experto puede llevar al solicitante a un artefacto (B) y viceversa (C). A veces, los desarrolladores no pueden resolver el problema, por lo que continúan al día siguiente tratando de resolverlo (G).

Fig. 8. Captura de conocimiento

4. Modelo de condensación de conocimiento

Basados en el proceso de localización del expertise descrito en la sección anterior, en esta sección, se presenta el modelo de condensación de conocimiento, el cual tiene como objetivo dar soporte a la localización del expertise; este modelo comprende tres módulos que ayudan a condensar el conocimiento dentro de las organizaciones de software: gramática formal, conocimiento semántico y herramientas de expertise.

La Figura 5 ilustra estos módulos, donde para apoyar a la representación del conocimiento del dominio de la ingeniería de software, se propuso una gramática formal (1), la cual describe la estructura de las frases y palabras de un idioma; es decir, se aplica de manera similar a los lenguajes de programación o los lenguajes humanos.

También se presenta, un modelo de conocimiento semántico para centralizar y compartir conocimiento de un dominio (2).

Finalmente, el modulo de herramientas de expertise describe mecanismos para condensar

ISSN 2007-9737

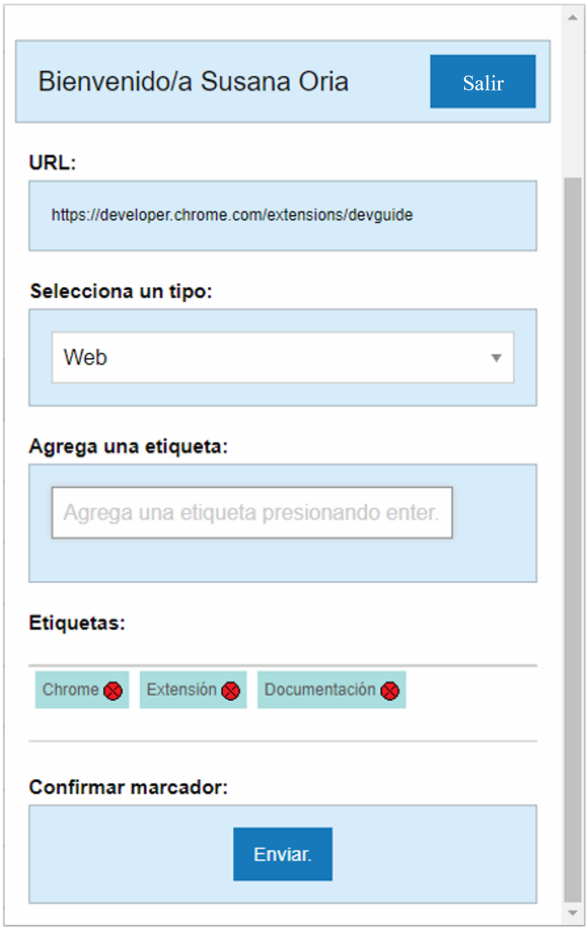

Fig. 9. Extensión B4U

el conocimiento basado en el Módulo de conocimiento semántico (3).

A continuación, se describen cada uno de los módulos y las implementaciones desarrolladas durante este trabajo.

4.1. Módulo de gramática formal

El conocimiento en los entornos de desarrollo de software cambia continuamente; el módulo de gramática formal describe el flujo de conocimiento en las organizaciones de software a través de un formalismo, el cual describe cómo los desarrolladores crean, comparten y almacenan conocimiento.

Además, dicho formalismo incluye la actualización de conocimiento, puesto que el conocimiento está en constante actualización a través de los proyectos y la experiencia de los desarrolladores.

Para entender de mejor manera la localización del expertise, el formalismo define el conocimiento como un conjunto de secuencias de datos ordenadas (tuplas) de la forma:

$$\mathtt{K} < \mathtt{P}, \mathtt{L}, \mathtt{S}, \mathtt{F} >. \tag{1}$$

Cada elemento de las tuplas ayuda a describir el conocimiento durante el proceso de desarrollo de software. $\mathtt{P}$ representa la plataforma de desarrollo asociada con una instancia de software.

Los proyectos de software generalmente se desarrollan para una plataforma en particular (por ejemplo, Web, Móvil, Escritorio, Híbrido). $\mathtt{L}$ representa un lenguaje de programación relacionado con la plataforma utilizada en un proyecto.

Dado un proyecto donde la plataforma es $\mathtt{P} = \mathtt{Web}$, $\mathtt{L}$ puede ser cualquier lenguaje para desarrollo web (por ejemplo, python, PHP, HTML, CSS).

$\mathtt{S}$ representa el tema, donde el objetivo es proporcionar una descripción detallada del conocimiento utilizando palabras clave (por ejemplo, GUI, Scripts, Eventos).

Finalmente, $\mathtt{F}$ representa el formato en el cual el conocimiento (por ejemplo, manuales, código fuente, marcadores).

Como podemos notar, cada tupla $\mathtt{K}$ representa conocimiento de los desarrolladores de software, en los siguientes apartados se describe como estas, ayudan al proceso de localización del expertise.

Tabla 5. Estadísticas descriptivas de la evaluación de los resultado del TAM

	Resultados del TAM para ExCap						
Constructo	**Mediana**	**Moda**	**Mínimo**	**Máximo**	**Percentil 25**	**Percentil 50**	**Percentil 75**
Utilidad Percibida	6	6	1	7	5	6	7
Facilidad de uso percibida	6	7	2	7	6	6	7
	Resultados del TAM para las herramientas tradicionales						
Constructo	**Mediana**	**Moda**	**Mínimo**	**Máximo**	**Percentil 25**	**Percentil 50**	**Percentil 75**
Utilidad Percibida	5	5	2	7	5	5	6
Facilidad de uso percibida	5	5	1	7	5	5	6
	Resultados del TAM para la extension B4U						
Constructo	**Mediana**	**Moda**	**Mínimo**	**Máximo**	**Percentil 25**	**Percentil 50**	**Percentil 75**
Utilidad percibida	6	6	3	7	6	6	7
Facilidad de uso percibida	7	7	2	7	6	7	7

4.1.1. Creación de perfil de conocimiento

Con la finalidad de asociar cada tupla con su proveedor o creador es necesario construir un perfil. Para construir un perfil, se necesita obtener información personal y los recursos utilizados por el perfil para resolver problemas o dudas durante un proyecto de desarrollo de software (contribuciones).

Además, la construcción del perfil implica la capacidad de un desarrollador registrado para ayudar a un colega y el valor que los recursos compartidos podrían tener dentro de la organización.

Por lo tanto, la construcción de un perfil permitirá el acceso a potenciales expertos y sus recursos para resolver problemas o dudas particulares.

La Tabla 4 se muestran los elementos principales que son necesarios para construir un perfil: datos personales, historial del proyecto, contribuciones (tuplas).

Estas contribuciones, hacen referencia a los artefactos, los cuales son recursos que los desarrolladores generan o encuentran para resolver un problema o duda en particular durante un proyecto de software.

Con respecto a los Datos personales representan información básica que permite a los usuarios ponerse en contacto con expertos y asociarlos con sus contribuciones.

Estas contribuciones se basan en el historial de proyectos del desarrollador, debido a que los desarrolladores obtienen experiencia y recursos para resolver problemas durante la ejecución de cada proyecto.

Cada perfil registrado almacena sus contribuciones (expertise acumulado) en tuplas, tal como se muestra a continuación:

$$\mathrm{K_i} \begin{bmatrix} \mathrm{k_1} < \mathrm{P, L, S, F} > \\ \mathrm{k_2} < \mathrm{P, L, S, F} > \\ \mathrm{k_3} < \mathrm{P, L, S, F} > \\ \vdots \\ \mathrm{k_n} < \mathrm{P, L, S, F} > \end{bmatrix}. \tag{2}$$

4.1.2. Búsqueda de expertise

Una vez creados los perfiles de conocimiento, los desarrolladores pueden realizar búsquedas de expertise para resolver una duda o un problema durante un proyecto. Esta búsqueda

se puede modelar como una tupla, como se muestra a continuación:

$$\mathtt{Sr} < \mathtt{P}, \mathtt{L}, \mathtt{S} >. \tag{3}$$

Todo el conocimiento se divide en subcadenas de información que forman el conocimiento de la búsqueda solicitada por un desarrollador.

La solicitud de búsqueda se representa por $\mathtt{Sr}$ (Search request), el elemento $\mathtt{F}$ se omite debido a que cuando se busca un recurso, el desarrollador no requiere algún formato (e.g., pdfs, hojas de calculo, marcadores, entre otros) en particular.

La búsqueda de conocimiento se describe de la siguiente manera. Dada $y = f(\mathtt{Sr}, \mathtt{K_{in}})$, donde $\mathtt{Sr}$ representa el conocimiento que necesita un desarrollador para resolver un problema o duda, y $\mathtt{K_{in}}$ un artefacto de un desarrollador particular de la organización.

$f(\mathtt{Sr}, \mathtt{K_{in}})$ realiza una comparación entre la solicitud y el conocimiento actual dentro de la organización. Por lo tanto, y es la intersección de las tuplas que cumplen con la solicitud de búsqueda del desarrollador, que se denota como $\mathtt{Sr} \cap \mathtt{K_{in}}$ cuando $\mathtt{K_{in}} \in \mathtt{S}$.

En este sentido, $\mathtt{S} = \{\mathtt{E1}, \mathtt{E2}\}$ representan el conjunto de escenarios en los que un recurso podría ser útil dado un $\mathtt{Sr} < \mathtt{P}, \mathtt{L}, \mathtt{S} >$ particular.

$\mathtt{E1}$ es el escenario que ocurre cuando la solicitud $\mathtt{Sr}$ coincide exactamente con $\mathtt{K_{in}}$. Por ejemplo, $\mathtt{Sr} < \mathtt{Web}, \mathtt{Python}, \mathtt{GUI} >$ y $\mathtt{K_{in}} < \mathtt{Web}, \mathtt{Python}, \mathtt{GUI} >$.

$\mathtt{E2}$ es el escenario que ocurre cuando la solicitud $\mathtt{Sr}$ coincide solo con la plataforma y el idioma de $\mathtt{K_{in}}$. Los desarrolladores pueden malinterpretar la causa raíz de sus problemas, utilizando las palabras clave incorrectas cuando buscan recursos útiles.

Por tanto, será útil sugerir recursos relacionados con el mismo idioma. Por ejemplo, $\mathtt{Sr} < \mathtt{Web}, \mathtt{Python}, \mathtt{GUI} >$ y $\mathtt{K_{in}} < \mathtt{Web}, \mathtt{Python}, \mathtt{Eventos} >$.

4.1.3. Actualización de conocimiento

El perfil del desarrollador y sus contribuciones deben actualizarse continuamente. La actualización del conocimiento puede realizarse a través del historial del proyecto; los desarrolladores adquieren experiencia y generan diferentes artefactos de cada proyecto.

Estos artefactos (por ejemplo, manuales, marcadores, código fuente) generados durante un software pueden verse como contribuciones, ya que podrían ser útiles en proyectos futuros. Cada vez que un desarrollador realiza una experiencia, encuentra o crea un recurso.

Por lo tanto, sus contribuciones aumentaron durante un proyecto de desarrollo de software. Lo anterior se denota de la siguiente manera:

$$\mathtt{K_i} \begin{bmatrix} \mathtt{k_1} < \mathtt{P}, \mathtt{L}, \mathtt{S}, \mathtt{F} > \\ \mathtt{k_2} < \mathtt{P}, \mathtt{L}, \mathtt{S}, \mathtt{F} > \\ \mathtt{k_3} < \mathtt{P}, \mathtt{L}, \mathtt{S}, \mathtt{F} > \\ \vdots \\ \mathtt{k_n} < \mathtt{P}, \mathtt{L}, \mathtt{S}, \mathtt{F} > \end{bmatrix} \therefore \mathtt{K'_i} \begin{bmatrix} \mathtt{k_1} < \mathtt{P}, \mathtt{L}, \mathtt{S}, \mathtt{F} > \\ \mathtt{k_2} < \mathtt{P}, \mathtt{L}, \mathtt{S}, \mathtt{F} > \\ \mathtt{k_3} < \mathtt{P}, \mathtt{L}, \mathtt{S}, \mathtt{F} > \\ \vdots \\ \mathtt{k_{n+j}} < \mathtt{P}, \mathtt{L}, \mathtt{S}, \mathtt{F} > \end{bmatrix}. \tag{4}$$

4.1.4. Construcción de conocimiento

Usar la gramática propuesta facilita construcción de nuevo conocimiento utilizando las tuplas, esto a través del análisis del comportamiento de los desarrolladores para resolver dudas o problemas.

Los desarrolladores suelen crear soluciones a problemas de programación, utilizando ejemplos en lenguajes de programación distintos al del problema.

4.2. Módulo de conocimiento semántico

En la sección 2, se identificó que las propuestas acumulan experiencia sin que la organización sea consciente de su existencia.

Cada propuesta utiliza sus propias entradas y salidas, lo cual dificulta la centralización de conocimiento.

En este sentido, se propone un modelado semántico del conocimiento para vincular a los expertos y los artefactos que los desarrolladores producen y consumen durante el desarrollo del software.

El objetivo es destacar cómo el modelado semántico podría mejorar la condensación del conocimiento.

El modelo semántico se desarrolló utilizando técnicas de modelado del conocimiento; estas técnicas consisten en realizar tareas como la adquisición de conocimientos, la especificación de requisitos y la conceptualización del conocimiento.

La tarea de adquisición de conocimiento consiste en adquirir el conocimiento de los expertos en el dominio (por ejemplo, entrevistas, grupos focales o encuestas) o de la literatura.

La tarea de especificación de requisitos consiste en seguir las pautas para capturar el conocimiento de los usuarios y producir un vocabulario de conceptos principales.

Finalmente, la tarea de conceptualización consiste en organizar y modelar el conocimiento adquirido utilizando representaciones externas (por ejemplo, UML, IDEF5).

Las organizaciones de software producen y consumen conocimiento, el cual se conoce formalmente como AK. La Figura 6 muestra los elementos principales del modelo semántico: proyectos, artefactos y desarrolladores.

El Desarrollador representa la descripción de un miembro del equipo en un proyecto de desarrollo de software; esta descripción podría incluir propiedades como un nombre, correo electrónico o teléfono celular.

Los proyectos representan una descripción de los trabajos actuales o pasados de un desarrollador; el objetivo es crear un perfil de habilidades de desarrollador basado en el registro del proyecto; las propiedades que se utilizan para describir un proyecto son el idioma, la plataforma, el rol y el nombre del proyecto.

Los artefactos representan una descripción de los diferentes recursos de los desarrolladores para resolver problemas o dudas mientras desarrollan software (por ejemplo, marcadores, fragmentos de código, documentación y tutoriales); las descripciones de artefactos incluyen propiedades como una plataforma y un tema.

El modelo presentado permite un vínculo entre sus elementos a través de las siguientes características. La primera característica del modelo son los valores de los datos.

La característica de valores de datos ayuda a describir un elemento; por ejemplo, describe información personal en el elemento desarrollador.

La segunda característica son las propiedades, las cuales son relaciones binarias en instancias de elementos del modelo; su propósito es vincular dos instancias.

Por ejemplo, la propiedad esUtilizadoPor vincula una instancia de Artefactos con Desarrolladores; por lo tanto, conocer al proveedor de un artefacto otorga acceso al experto y su experiencia.

La propiedad perteneceA vincula una instancia de Artefactos con un Proyecto; el objetivo aquí es localizar en qué proyectos los desarrolladores crean o generan un artefacto.

Finalmente, la última propiedad es trabajaEn; la propiedad vincula a los desarrolladores y proyectos para crear un trasfondo de las habilidades de los desarrolladores basándose en el historial del proyecto.

Como se puede notar por la descripción de las propiedades, el objetivo principal de este módulo es generar una representación de conocimiento del proceso de desarrollo de software para capturar y clasificar el expertise.

4.3. Módulo de herramientas de expertise

La finalidad principal del módulo de herramientas de expertise es proporcionar mecanismos para condensar el conocimiento.

En la Sección 3.1, se identificaron las fuentes más populares entre los desarrolladores (por ejemplo, sitios web, documentación oficial, foros y tutoriales en video).

Además de esto, se identificó que comúnmente, independientemente de la fuente utilizada, los desarrolladores utilizan dos formas principales de capturar este conocimiento: marcadores o archivar en la computadora.

En el caso de la documentación oficial o los foros de programación, los desarrolladores suelen guardar páginas mediante marcadores.

Por otro lado, en muchos casos, los desarrolladores externalizan el conocimiento utilizando documentos como Word, Bloc de notas, PDF o descargando videos.

Durante este trabajo se propusieron dos herramientas para capturar la experiencia en la fase de codificación durante el proceso de desarrollo de software.

A continuación, se describen ExCap y B4U, las herramientas propuestas en este trabajo para para implementar algunos de los elementos del módulo de conocimiento semántico; estas propuestas tienen como propósito proporcionar a los desarrolladores una interfaz de usuario para capturar artefactos de dos fuentes (marcadores y documentos digitales).

4.3.1. ExCap

ExCap es una herramienta para condensar el expertise, particularmente el que se encuentra en los artefactos digitales que los desarrolladores crean y consumen durante un proyecto de desarrollo de software.

Los usuarios se registran e ingresan su información personal para asociar esta a sus contribuciones (ver Figura 7).

Los usuarios en ExCap pueden registrar sus artefactos utilizando una función de arrastrado y agregando la clasificación correspondiente para facilitar la recuperación del mismo tal y como se muestra en la Figura 8.

4.3.2. B4U Extensión de marcadores

B4U es una extensión para Google Chrome capaz de capturar un sitio web y clasificarlo agregando categorías y etiquetas. B4U conecta y guarda marcadores a través de una API REST que almacena los usuarios y los datos guardados por ellos.

Los valores del usuario y el campo de correo se guardan en localStorage, por lo tanto, cuando un usuario inicia sesión, los datos del usuario se extraen de localStorage.

Cuando se inicia la extensión, muestra una pantalla de inicio de sesión, así como una función de creación de usuario; Ambas pantallas solicitan dos datos principales: nombre de usuario y correo electrónico.

La Figura 9 muestra la vista principal de B4U y los elementos utilizados para capturar la descripción de los marcadores, que son URL, Tipo y Etiquetas. URL: muestra al usuario la URL que se guardará en el marcador, este se guarda automáticamente.

Tipo: el marcador debe formar parte de una de las tres categorías predefinidas: Web, Móvil o Escritorio. Etiquetas: las etiquetas crean una meta descripción del conocimiento que los desarrolladores quieren almacenar.

Los usuarios pueden almacenar todos los marcadores que necesiten; además un aspecto importante es que entre más (palabras clave) se utilicen, será más fácil clasificar y describir el recurso para su recuperación.

5. Evaluación del modelo de condensación de conocimiento

En la sección 4, se presentaron dos mecanismos de captura de conocimiento para la fase de codificación durante el desarrollo de software; con la finalidad de obtener valoración de los desarrolladores con respecto a la captura y clasificación de conocimiento a través de las herramientas se realizó una evaluación.

En esta sección se presenta el método utilizado para realizar la evaluación, los resultados y la discusión de los hallazgos más relevantes.

5.1. Método

El método utilizado para evaluar los mecanismos de extensión ExCap y B4U, se estructuró de la siguiente manera: objetivo, participantes, materiales, procedimiento, variables e hipótesis.

5.1.1. Objetivo

El propósito del experimento fue evaluar la percepción de los mecanismos en términos de utilidad y facilidad de uso percibida al capturar y clasificar conocimiento.

5.1.2. Participantes

Para confirmar nuestras hipótesis, se formaron 3 grupos de estudios empíricos. Dos de los grupos evaluaron los mecanismos propuestos, mientras que el tercer grupo evaluó las herramientas tradicionales utilizadas en el desarrollo de software.

Los participantes evaluaron estas, acorde a su herramienta predilecta; es decir las herramientas tradicionales son aquellas que usualmente utilizan los desarrolladores para capturar y consultar conocimiento (e.g., marcadores, bloc de notas, carpetas de archivos y repositorios).

Cada grupo formado estuvo compuesto por participantes de dos contextos: académico e industrial.

Los participantes de contexto académico fueron estudiantes de la carrera de Ingeniería de Software del Instituto Tecnológico de Sonora (ITSON), los cuales fueron seleccionados del último semestre de la carrera ya que, en este punto, ya estaban en contacto con metodologías ágiles y con el desarrollo de software.

En cuanto a los participantes del contexto industrial, fueron desarrolladores de software con experiencia en desarrollo ágil y distribuido; todas las empresas en el contexto de la industria están completamente dedicadas al desarrollo de software o tienen un departamento dentro de su organización dedicado al desarrollo de software para la misma organización.

Se contó con un total de 110 participantes distribuidos entre los tres grupos de estudio empírico.

20 participantes para el grupo ExCap (4 académicos y 16 desarrolladores), 44 para la extensión de marcadores B4U (23 académicos y 21 desarrolladores) y 46 para el grupo de herramientas tradicionales (25 académicos y 21 desarrolladores).

La edad promedio fue de 21.41 años para los participantes del contexto académico y 27.37 años para el contexto industrial.

5.1.3. Instrumentos

Para llevar a cabo la evaluación se utilizaron los siguientes materiales:

– Manual con instrucciones de uso de Excap: Los participantes recibieron un documento que explica la funcionalidad de la herramienta ExCap y las instrucciones para realizar algunas pruebas de captura y clasificación de conocimientos.

– Manual con instrucciones de uso de la extensión B4U: Los participantes recibieron un documento que explica la funcionalidad del complemento de extensión B4U. Además, el documento incluye la instrucción para probar la captura y clasificación de conocimientos.

– Presentación de herramientas tradicionales: para evaluar las herramientas tradicionales, se utilizó una presentación de PowerPoint para describir el proceso de localización del expertise. El objetivo fue discutir y evaluar las herramientas tradicionales utilizadas por los participantes para capturar conocimientos.

– Cuestionario TAM (Modelo de Aceptación de Tecnología): se preparó un cuestionario que los participantes respondieron acorde a la herramienta que les tocó evaluar [27].

5.1.4. Procedimiento

Como se mencionó anteriormente, la evaluación contó con tres grupos de estudio empírico que duraron aproximadamente 1 hora y media, para cada uno de los mecanismos propuestos (aplicación ExCap y extensión B4U) y para las herramientas tradicionales.

Grupo ExCap

– Preparación: Se envió por correo a todos los participantes una carpeta comprimida con los siguientes elementos: aplicación ExCap, pautas para la interacción y material de apoyo.

– Ejecución: Primero, se dio una introducción (dentro del documento) a los participantes explicando el objetivo del experimento.

  Luego, siguiendo el documento de la aplicación de ExCap, los participantes interactuaron con la extensión como se indica en la guía.

  Finalmente, los participantes respondieron un cuestionario para evaluar la aplicación ExCap.

Grupo B4U

– Preparación: Primero, se verificó que los participantes estuvieran conectados a la sesión y pudieran escuchar.

  Posteriormente, se compartió un archivo comprimido con los materiales necesarios para el experimento.

  Finalmente, se verificó que los participantes pudieron visualizar la presentación para contexto del problema abordado.
– Ejecución: Primero, se dio una introducción a los participantes explicando el objetivo del experimento.

  Luego, siguiendo el documento de la extensión B4U, los participantes interactuaron con la extensión como se indica en la guía.

  Finalmente, los participantes respondieron un cuestionario para evaluar la extensión B4U.

Grupo herramientas tradicionales

– Preparación: Primero, se verificó que los participantes estuvieran conectados a la sesión y pudieran escuchar y que pudieran visualizar la presentación para dar contexto en el problema abordado.

  Finalmente, les damos a los participantes una presentación de PowerPoint para dar contexto a los participantes con el proceso de localización del expertise.
– Ejecución: Finalmente, los participantes respondieron un cuestionario para evaluar las herramientas tradicionales utilizadas por cada participante.

5.1.5. Variables e hipótesis

Las variables independientes fueron los mecanismos de captura y clasificación del conocimiento: ExCap, extensión B4U y las herramientas tradicionales. Las variables dependientes fueron las siguientes:

– Utilidad: el grado en el que una persona cree que el uso de un sistema en particular mejoraría su desempeño laboral.
– Facilidad de uso: el grado en que una persona cree que usando un sistema en particular no supondría ningún esfuerzo.

Ambas variables dependientes se midieron mediante un cuestionario con 12 preguntas (seis de utilidad y seis de facilidad de uso). Finalmente, en base a las variables dependientes, se establecieron las siguientes hipótesis.

Utilidad

– $H1_a$: Existe una diferencia entre los mecanismos propuestos sobre las herramientas tradicionales en cuanto a la utilidad para capturar y clasificar el conocimiento.
– $H1_0$: No existe diferencia entre los mecanismos propuestos sobre las herramientas tradicionales en cuanto a la utilidad para capturar y clasificar el conocimiento.

Facilidad de uso

– $H1_a$: Existe una diferencia entre los mecanismos propuestos sobre las herramientas tradicionales en cuanto a la facilidad de uso para capturar y clasificar el conocimiento.
– $H1_0$: No existe diferencia entre los mecanismos propuestos sobre las herramientas tradicionales en cuanto a la facilidad de uso para capturar y clasificar el conocimiento.

5.2. Resultados

En esta sección, se describen los resultados obtenidos durante los tres estudios grupales empíricos. Como se mencionó anteriormente, la evaluación de cada herramienta se realizó mediante un cuestionario (escala Likert 7) basado en el TAM.

Los valores y la interpretación de las posibles respuestas son los siguientes: extremadamente probable (7), bastante probable (6), levemente probable (5), ninguna (4), levemente improbable (3), bastante improbable (2) y extremadamente improbable (1).

La Tabla 5 reporta estadísticas descriptivas generales de los resultados de TAM para cada herramienta. Primero, en la aplicación ExCap, los resultados de TAM muestran una mediana y una moda de 6 (bastante probable) para el constructo de utilidad.

Para el constructo de facilidad de uso, obtenemos una mediana de 6 y una moda de 7 (muy probable). Por lo tanto, en general, los participantes percibieron la aplicación ExCap como (muy probablemente) útil y (muy probablemente) utilizable.

Por otro lado, en el caso de la extensión B4U, los resultados de TAM muestran una mediana y una moda de 6 (bastante probable) para la construcción de utilidad.

Para el constructo de facilidad de uso, obtenemos una mediana de 6 y una moda de 7 (muy probable).

Por lo tanto, en general, los participantes percibieron la aplicación del complemento de extensión B4U como (muy probablemente) útil y (muy probablemente) utilizable.

A diferencia del mecanismo propuesto, el resultado de TAM de las herramientas tradicionales muestran una mediana y moda de 5 (ninguno) para ambos constructos (utilidad y facilidad de uso).

Nuestros resultados mostraron una diferencia significativa de mejora en cuanto a la utilidad y facilidad de uso para la captura y clasificación del conocimiento.

Para verificar la validez de nuestros hallazgos, se realizó U de Mann-Whitney; la cual fue seleccionada porque los datos no siguen una distribución normal.

Se probó cada mecanismo en comparación con las herramientas tradicionales para ambos constructos: utilidad y facilidad de uso; La Tabla 5 muestra los resultados generales de la prueba U de Mann-Whitney.

A partir de los mecanismos comparados con las herramientas tradicionales sobre el constructo de utilidad percibida, obtenemos los siguientes resultados (valor U = 14842, puntaje Z = -1,94879, valor p = 0,022559, alfa = 0,05) para la aplicación ExCap y (U- value = 31665, Z-score = -3.01737, p-value = 0.00126, alpha = 0.05) para el complemento de extensión B4U.

Las pruebas sobre el constructo de utilidad percibida indican una diferencia significativa entre los resultados de ambos mecanismos en comparación con las herramientas tradicionales; los participantes percibieron los mecanismos como útiles. Por lo tanto, rechazamos la utilidad $H1_o$.

A partir de los mecanismos comparados con las herramientas tradicionales sobre el constructo de facilidad de uso percibida, obtenemos los siguientes resultados (valor U = 10911.5, puntaje Z = -5.63576, valor p = 0.00001, alfa = 0.05) para la aplicación ExCap y (Valor U = 24161, puntaje Z = -7.09082, valor p = .00001, alfa = 0.05) para el complemento de extensión B4U.

Las pruebas U de Mann-Whitney sobre el constructo de facilidad de uso percibida indican una diferencia significativa entre los resultados de ambos mecanismos en comparación con las herramientas tradicionales; los participantes percibieron los mecanismos como utilizables. Por lo tanto, rechazamos la facilidad de uso de $H2_o$.

5.3. Discusión

Como parte de la evaluación se obtuvieron resultados significativos de nuestros mecanismos de captura de conocimiento propuestos. Sin embargo, a pesar de los importantes resultados obtenidos, hay algunos aspectos cruciales que discutir.

Los valores bajos son un aspecto fundamental a discutir a partir de nuestros resultados. La razón

principal que puede asociarse con los valores bajos es la experiencia de los participantes en la industria del desarrollo de software.

Como mencionamos, la evaluación estuvo compuesta por grupos de dos contextos (académico e industrial).

En el contexto académico, los estudiantes aún no son conscientes de las demandas del entorno de desarrollo de software profesional donde constantemente necesitan producir más rápido y mejor.

Por lo tanto, es posible que los estudiantes no perciban la importancia de reutilizar el conocimiento de proyectos anteriores y con el proceso de localización del expertise.

La evaluación realizada se enfocó en la captura y clasificación del conocimiento. Por tanto, es necesario discutir la relación de nuestros resultados con el modelo de condensación de conocimiento.

Dado que la captura y clasificación se basan en nuestro modelo, lo siguiente implica que los participantes aceptan capturar y clasificar el conocimiento utilizando un modelo semántico.

En el futuro, la implementación de la búsqueda de los mecanismos facilitará la recuperación de la experiencia.

En el caso de las herramientas tradicionales, en este sentido algunos aspectos importantes a discutir son el tipo de herramientas utilizadas y los bajos valores obtenidos en la evaluación.

Los desarrolladores utilizan varias formas de capturar conocimiento (proceso de externalización) [21].

Como se mencionó anteriormente, existen dos tipos de conocimiento explícito: documentado y formalizado.

El conocimiento documentado tiende a predominar debido a las prácticas ágiles. Los desarrolladores usan o adaptan diferentes herramientas para capturar conocimiento [38, 19].

Algunas herramientas, como las notas del bloc de notas, los repositorios o los marcadores convencionales, ayudan a capturar el conocimiento [42].

Con estas herramientas, los desarrolladores externalizan el conocimiento para su uso particular: resolución de problemas durante una tarea de desarrollo de software.

Algunas herramientas, como las notas del bloc de notas, los repositorios o los marcadores convencionales, ayudan a capturar el conocimiento.

Sin embargo, todavía existen limitaciones para estos o dificultades en términos de uso (e.g., en el caso del uso de marcadores el desarrollador puede olvidar las etiquetas o el propósito del mismo) [48].

6. Conclusiones

En este trabajo se describió el proceso de construcción de un modelo de condensación de conocimiento, el cual tiene la finalidad de apoyar el en el proceso de localización del expertise para reducir la vaporización del conocimiento causada por la preferencia de los métodos ágiles por el conocimiento tácito.

Se siguió el concepto de condensación como base para la construcción del presente modelo. Como resultado del proceso de construcción, se analizaron diferentes aspectos del proceso de localización del expertise, dando lugar a las siguientes contribuciones principales:

– El diseño de un mapa de proceso de localización del expertise.
– El modelo de conocimiento semántico para el proceso de desarrollo de software.
– La gramática formal para la descripción del conocimiento.
– Los mecanismos para capturar el conocimiento que implementa los elementos del modelo semántico.

El modelo propuesto en este trabajo consta de tres módulos: gramática formal, conocimiento semántico y herramientas de expertise.

6.1. Gramática formal

En el Módulo de gramática formal se propone un lenguaje basado en tuplas el cual se centra en las contribuciones de los desarrolladores, las cuales representan los artefactos generados por un desarrollador durante un proyecto de desarrollo de software.

Las tuplas ayudan a describir la captura, búsqueda y actualización de conocimientos de los desarrolladores.

Nuestro objetivo es proporcionar información sobre una forma de modelar la manera en que los desarrolladores gestionan el AK, lo cual involucra establecer un formalismo para describir el proceso de externalización y combinación de conocimientos.

6.2. Conocimiento semántico

Este módulo presentó un modelo de conocimiento semántico para datos estructurados y no estructurados; el objetivo es buscar y centralizar aplicaciones, bases de datos y archivos.

Las propuestas acumulan experiencia acumulada sin que la organización sea consciente de su existencia, y dado que cada propuesta utiliza sus entradas y salidas, no se pueden centralizar.

En este sentido, el modelado del conocimiento semántico podría ser una forma de vincular a los expertos y los artefactos que los desarrolladores producen y consumen durante el desarrollo del software.

6.3. Herramientas de captura de expertise

En el módulo de herramientas de expertise, se presentaron los primeros esfuerzos para implementar el concepto de condensación de conocimiento, el cual se basa en una ontología obtenida y validada formalmente.

En consecuencia, el uso de una ontología permite el razonamiento automatizado sobre el conocimiento arquitectónico (artefactos y expertos), razonamiento con conceptos y relaciones similares a cómo los humanos perciben conceptos interconectados y un modelo que evoluciona con el crecimiento de datos sin afectar los procesos.

Los mecanismos propuestos en este módulo fueron ExCap, una herramienta para capturar artefactos digitales (por ejemplo, manuales, archivos de código y vídeos), y la extensión B4U, un complemento para capturar marcadores desde el navegador web.

Los resultados de la evaluación de estas herramientas mostraron que los participantes perciben como útil y fácil de usar las herramientas para capturar y clasificar el conocimiento; esto implica que el uso de mecanismos de captura basados en la condensación de conocimiento permitirá capturar y clasificar los diferentes tipos de conocimiento (documentados y formalizados), aún cuando estos no hayan sido capturados formalmente por preferencias en el paradigma ágil.

Además, los mecanismos basados en nuestro modelo de conocimiento semántico centralizarán datos valiosos para la organización.

Referencias

1. **Bhat, M., Shumaiev, K., Koch, K., Hohenstein, U., Biesdorf, A., Matthes, F. (2018).** An expert recommendation system for design decision making: Who should be involved in making a design decision? Proceedings of the 15th International Conference on Software Architecture, pp. 85–94. DOI: 10.1109/ICSA.2018.00018.
2. **Bhatia, M. P., Kumar, A., Beniwal, R. (2016).** Ontologies for software engineering: Past, present and future. Indian Journal of Science and Technology, Vol. 9, No. 9. DOI: 10.17485/ijst/2016/v9i9/71384.
3. **bin Ali, N. (2016).** Is effectiveness sufficient to choose an intervention?. Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement - ESEM '16. DOI: 10.1145/2961111.2962631.
4. **Bonilla-Morales, B., Crespo, S., Clunie, C. (2012).** Reuse of use cases diagrams: An approach based on ontologies and semantic web technologies. .

5. **Borrego, G., Moran, A. L., Palacio, R., Rodriguez, O. M. (2016).** Understanding architectural knowledge sharing in AGSD teams: An empirical study. Proceedings of the 11th International Conference on Global Software Engineering (ICGSE). DOI: 10.1109/icgse.2016.29.
6. **Borrego, G., Morán, A. L., Palacio, R. R., Vizcaíno, A., García, F. O. (2019).** Towards a reduction in architectural knowledge vaporization during agile global software development. Information and Software Technology, Vol. 112, pp. 68–82. DOI: 10.1016/J.INFSOF.2019.04.008.
7. **Borrego, G., Salazar-Lugo, G., Parra, M., Palacio, R. (2019).** Slack's knowledge classification mechanism for architectural knowledge condensation. International Conference on Computational Science and Computational Intelligence, pp. 1121–1126. DOI: 10.1109/CSCI49370.2019.00212.
8. **Brandt, J., Dontcheva, M., Weskamp, M., Klemmer, S. R. (2010).** Example-centric programming. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 513. DOI: 10.1145/1753326.1753402.
9. **Cheesman, J., Daniels, J. (2000).** UML components: a simple process for specifying component-based software.
10. **Clerc, V., Lago, P., Van Vliet, H. (2011).** Architectural knowledge management practices in agile global software development. Proceedings of the 6th IEEE International Conference on Global Software Engineering Workshops, ICGSE Workshops 2011, pp. 1–8. DOI: 10.1109/ICGSE-W.2011.17.
11. **Dalkir, K. (2013).** Knowledge Management in Theory and Practice. DOI: 10.4324/9780080547367.
12. **Decker, B., Decker, B., Ras, E., Rech, J., Klein, B., Hoecht, C. (2005).** Self-organized reuse of software engineering knowledge supported by semantic wikis. Proceedings of the 5th International Workshop on Semantic Web Enabled Software Engineering.
13. **Devanbu, P., Brachman, R. J., Selfridge, P. G., Ballard, B. W. (1990).** LaSSIE: A knowledge-based software information system. Proceedings - International Conference on Software Engineering, pp. 249–261. DOI: 10.1109/ICSE.1990.63631.
14. **Dorairaj, S., Noble, J., Malik, P. (2012).** Knowledge management in distributed agile software development. Agile Conference. DOI: 10.1109/agile.2012.17.
15. **Eberhart, A., Agarwal, S. (2004).** SmartAPI-associating ontologies and apis for rapid application development. Ontologien in der und für die Softwaretechnik, pp. 43–48.
16. **Ericsson, K. A., Prietula, M. J., Cokely, E. T. (2007).** The making of an expert. Harvard business review, Vol. 85, No. 7/8, pp. 114.
17. **Happel, H. J., Korthaus, A., Seedorf, S., Tomczyk, P. (2006).** KOntoR: An ontology-enabled approach to software reuse. Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering.
18. **Holz, H., Melnik, G., Schaaf, M. (2003).** Knowledge management for distributed agile processes: models, techniques, and infrastructure. Proceedings of the 12th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, pp. 291–294. DOI: 10.1109/ENABL.2003.1231423.
19. **Hummel, O., Atkinson, C. (2006).** Using the web as a reuse repository. Lecture Notes in Computer Science, Springer Berlin Heidelberg, pp. 298–311. DOI: 10.1007/11763864_22.
20. **Jedlitschka, A., Ciolkowski, M., Denger, C., Freimut, B., Schlichting, A. (2007).** Relevant information sources for successful technology transfer: A survey using inspections as an example. Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement, pp. 31–40. DOI: 10.1109/ESEM.2007.60.
21. **Josyula, J., Panamgipalli, S., Usman, M., Britto, R., Ali, N. B. (2018).** Software practitioners' information needs and sources: A survey study. Proceedings of the 9th International Workshop on Empirical Software Engineering in Practice (IWESEP), pp. 1–6.
22. **Kagdi, H., Hammad, M., Maletic, J. I. (2008).** Who can help me with this source code change? IEEE International Conference on Software Maintenance, ICSM, pp. 157–166. DOI: 10.1109/ICSM.2008.4658064.
23. **Larrucea, X., O'Connor, R. V., Colomo-Palacios, R., Laporte, C. Y. (2016).** Software process improvement in very small organizations. IEEE Software, Vol. 33, No. 2, pp. 85–89. DOI: 10.1109/MS.2016.42.

24. **LaToza, T. D., Venolia, G., DeLine, R. (2006).** Maintaining mental models. Proceeding of the 28th international conference on Software engineering - ICSE '06. DOI: 10.1145/1134285.1134355.

25. **Li, Z., Liang, P., Avgeriou, P. (2013).** Application of knowledge-based approaches in software architecture: A systematic mapping study. Information and Software Technology, Vol. 55, No. 5, pp. 777–794. DOI: 10.1016/j.infsof.2012.11.005.

26. **Lin, J., Fox, M. S., Bilgic, T. (1996).** A requirement ontology for engineering design. Concurrent Engineering, Vol. 4, No. 3, pp. 279–291. DOI: 10.1177/1063293x9600400307.

27. **Marangunić, N., Granić, A. (2014).** Technology acceptance model: a literature review from 1986 to 2013. Universal Access in the Information Society, Vol. 14, No. 1, pp. 81–95. DOI: 10.1007/s10209-014-0348-1.

28. **Matter, D., Kuhn, A., Nierstrasz, O. (2009).** Assigning bug reports using a vocabulary-based expertise model of developers. Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories, MSR 2009, pp. 131–140. DOI: 10.1109/MSR.2009.5069491.

29. **McMillan, C., Poshyvanyk, D., Grechanik, M., Xie, Q., Fu, C. (2013).** Portfolio: Searching for relevant functions and their usages in millions of lines of code. ACM Transactions on Software Engineering and Methodology, Vol. 22, No. 4, pp. 1–30. DOI: 10.1145/2522920.2522930.

30. **Minto, S., Murphy, G. C. (2007).** Recommending emergent teams. Proceedings of the 4th International Workshop on Mining Software Repositories, MSR 2007. DOI: 10.1109/MSR.2007.27.

31. **Mohagheghi, P., Conradi, R. (2007).** Quality, productivity and economic benefits of software reuse: A review of industrial studies. Empirical Software Engineering, Vol. 12, No. 5, pp. 471–516. DOI: 10.1007/s10664-007-9040-x.

32. **Moreno, L., Bavota, G., Di Penta, M., Oliveto, R., Marcus, A. (2015).** How can i use this method? Proceedings of the 37th International Conference on Software Engineering, pp. 880–890. DOI: 10.1109/ICSE.2015.98.

33. **Nerur, S., Balijepally, V. (2007).** Theoretical reflections on agile development methodologies. Communications of the ACM, Vol. 50, No. 3, pp. 79–83. DOI: 10.1145/1226736.1226739.

34. **Ponzanelli, L., Bavota, G., Di Penta, M., Oliveto, R., Lanza, M. (2014).** Mining stackoverflow to turn the ide into a self-confident programming prompter. Proceedings of the 11th Working Conference on Mining Software Repositories, pp. 102–111. DOI: 10.1145/2597073.2597077.

35. **Rupakheti, C. R., Hou, D. (2011).** Satisfying programmers' information needs in api-based programming. Proceedings of the 19th International Conference on Program Comprehension, pp. 250–253. DOI: 10.1109/ICPC.2011.16.

36. **Schneider, K. (2009).** Experience and knowledge management at work. In Experience and Knowledge Management in Software Engineering. pp. 165–202. DOI: 10.1007/978-3-540-95880-2_6.

37. **Sharif, K. Y., Buckley, J. (2009).** Observation of open source programmers' information seeking. Proceedings of the 17th International Conference on Program Comprehension, pp. 307–308. DOI: 10.1109/ICPC.2009.5090071.

38. **Shiva, S. G., Shala, L. A. (2008).** Using semantic wikis to support software reuse. Journal of Software, Vol. 3, No. 4. DOI: 10.4304/jsw.3.4.1-8.

39. **Siricharoen, W. V. (2007).** Ontologies and software engineering. Handbook on Ontologies, Vol. 4488 LNCS, pp. 1155–1161. DOI: 10.1007/978-3-540-92673-3_27.

40. **Sommerville, I. (2016).** Software Engineering. Pearson, 10 edition.

41. **Sonnentag, S., Niessen, C., Volmer, J. (2006).** Expertise in software design. pp. 373–387.

42. **Spoelstra, W., Iacob, M., van Sinderen, M. (2011).** Software reuse in agile development organizations. Proceedings of the 2011 ACM Symposium on Applied Computing - SAC '11. DOI: 10.1145/1982185.1982255.

43. **Strauss, A., Corbin, J. M. (1997).** Grounded theory in practice. Sage Publications, Inc.

44. **Sydorov, N. A., Sydorova, N. N., Mendzebryovsky, I. B. (2018).** Software engineering ontologies categorization. Problems in Programming, , No. 1, pp. 55–64. DOI: 10.15407/pp2018.01.055.

45. **Uikey, N., Suman, U., Ramani, A. K. (2011).** A documented approach in agile software development. Technical Report 2.

46. **Viana, D., Conte, T., de Souza, C. R. (2014).** Knowledge transfer between senior and novice

software engineers: A qualitative analysis. International Conference on Software Engineering and Knowledge Engineering, pp. 235–240.

47. **Viana, D., Rabelo, J., Conte, T., Vieira, A., Barroso, E., Dib, M. (2013).** A qualitative study about the life cycle of lessons learned. Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering, pp. 73–76. DOI: 10.1109/CHASE.2013.6614734.
48. **Zagalsky, A., German, D. M., Storey, M.-A., Teshima, C. G., Poo-Caamaño, G. (2017).** How the R community creates and curates knowledge: an extended study of stack overflow and mailing lists. Empirical Software Engineering, Vol. 23, No. 2, pp. 953–986. DOI: 10.1007/s10664-017-9536-y.

Article received on 03/12/2021; accepted on 04/01/2023.
Corresponding author is Jose Ramón Martínez-Garcia.